Универзитет у Београду
Математички факултет

Мастер рад

Програмска реализација неких тестова да ли је задати број прост

СТУДЕНТ:
Марија Борисов 1169/12

МЕНТОР:
проф. др Миодраг Живковић

Београд
2013.

Садржај

Предговор

Прости бројеви и особине простих бројева проучавани су још у античкој Грчкој. Од тада па до данас њихова практична примена је у сталном порасту. Након појаве рачунара развијене су многобројне корисне апликације у чијој основи леже прости бројеви.

У овом раду изложена су својства простих бројева и одређени алгебарски алгоритми. Централна тема су тестови за испитивање да ли је задати број прост. Детаљно су описани тестови простости и наведени су примери њихове употребе. Такође су дате основе паралелног програмирања са применом на Ератостеново сито и основи криптографије, који су неопходни да би била приказана општа представа о значају и могућностима у тестирању простости бројева.

На крају је објашњена реализација одређених тестова простости, и алгоритма за генерисање *велике* базе простих бројева.

***

Желим да изразим своју дубоку захвалност људима који су ми помогли у бољем разумевању теорије бројева, криптографије и паралелног програмирања, а самим тим и омогућили писање овог рада.

Највећу захвалност дугујем проф. др Миодрагу Живковићу и проф. др Жарку Мијајловићу. Проф. др Миодраг Живковић ми је дао смернице у организовању рада и помогао у реализацији тестова за испитивање да ли је задати број прост. Предавања и савети проф. др Жарка Мијајловића највише су допринели да научим неопходне основе теорије бројева и криптографије.

Велику захвалност дугујем Математичком институту САНУ, нарочито проф. др Миодрагу Михаљевићу, чија су предавања на семинарима била велики подстицај за даљи рад и напредовање.

Такође се захваљујем колегама Александру Пејовићу и Славку Моцоњи на програмерској и теоријској помоћи.

***

Комисија:

проф. др Миодраг Живковић

проф. др Предраг Јаничић

доц. др Саша Малков

1 Увод

Цео број $n > 1$ је **прост број** (енгл. *prime number*) ако су једини његови позитивни делитељи број 1 и сам број n.[1] Цео број $n > 1$ је **сложен број** (енгл. *composite number*) ако није прост.

Иако прости бројеви имају веома једноставну дефиницију, вековима су инспирисали математичаре. Поред занимљиве улоге у теоријским истраживањима, прости бројеви имају и значајну примену у модерном информационо-технолошком свету. Развојем апликација за електронско пословање питање сигурности постаје све значајније, а самим тим и проблем својства бројева који се користе у тим апликацијама, да ли су прости и да ли се *лако* могу факторисати.

У **Поглављу 2** је представљена теорија која је потребна за разумевање тестова да ли је задати број прост. Неке од основних особина простих бројева се лако показују. На пример, сви прости бројеви већи од 3 су облика $6k + 1$ или $6k + 5$, $k \geq 0$ цео број.[2] Интуитивно је јасно да што је број већи, теже је да буде прост јер постоји све више његових могућих делитеља. Ако се зна да прости бројеви постају све ређи у прстену целих бројева, постављају се питања: како су они распоређени, да ли постоји нека *шема* која дозвољава да се предвиди где се следећи прост број појављује, као и колико има простих бројева у одређеном интервалу? Нарочито је интересантно питање да ли се у полиномијалном времену може доказати да ли је произвољан цео број прост или сложен.

Примењена теорија бројева добија све више на значају. Да би реализација тестова простости била могућа потребно је детаљније обрадити одређене алгоритме из теорије бројева. У **Поглављу 3** су приказани алгоритми за проналажење највећег заједничког делитеља, инверза елемента, степеновање елемената, израчунавање Јакобијевог симбола, који чине основне делове тестова простости. Узастопно пробно дељење и Ератостеново сито представљају једноставне методе за проналажење простих бројева.

У **Поглављу 4** су описани тестови да ли је задати број прост. **Тест простости**, тј. **тест прималности**, је тест којим се одређује да ли је задати број прост.[3] Тестови простости се деле у две групе: вероватносни тестови простости и тестови за доказивање простости.

Вероватносни тест простости (енгл. *probabilistic primality test, pseudo-primality test*), тј. **пробабилистички тест простости**, у зависности од избора одређеног параметра који помаже у откривању да ли је задати број сложен, може дати нетачан резултат, односно ређи за сложен број да је прост. Вероватноћа грешке може бити смањена понављањем теста за неколико различитих вредности тог параметра. Ако се на основу вероватносног теста простости верује да је цео број n прост, каже се да је број n **вероватно прост** (енгл. *probable prime*).

Вероватносни тестови прималности дају *делимичне* информације о простости неког броја. Често се ови тестови називају и **тестови сложености**.

Прави тест простости (енгл. *true primality test, primality proving test*) је **тест за доказивање простости** задатог целог броја. Ако је простост целог броја n утврђена на основу правог теста простости, каже се да је цео број n **доказив прост број** (енгл. *provable prime*).

Постоје и **технике за конструисање** доказивих простих бројева, тј. алгоритми за генерисање (случајних) доказивих простих бројева.

Током последњих неколико деценија представљен је велики број алгоритама који су омогућили лакше налажење простих бројева. Посебно су интересантна истраживања на пољу елиптичких кривих, Јакобијевих и Гаусових сума, као и проналажење полиномијалног теста за доказивање простости.

У **Поглављу 5** су дате основе паралелног програмирања и стандарда MPI. Како је време извршавања тестова прималности, алгоритама за факторизацију, као и алгоритама за шифровање и дешифровање веома битно, **паралелно програмирање** може помоћи у постизању потребних захтева.

Једноставан пример паралелизације код вероватносних тестова прималности може бити паралелна провера прималности броја за различите параметре, где сваки процес независно проверава

[1] Број 1 по дефиницији није прост број, иако су до 19-ог века многи математичари сматрали супротно.

[2] Следи из чињенице да су сви позитивни цели бројеви неког од облика: $6k$, $6k + 1$, $6k + 2$, $6k + 3$, $6k + 4$, $6k + 5$, за цео број $k \geq 0$.

[3] У овом раду се потпуно равноправно употребљавају термини "тест простости", "тест прималности", "тест да ли је задати број прост", јер су то устаљени термини у литератури на српском језику.

да ли је број сложен, и у случају проналажења сведока сложености датог броја шаље поруку другим процесима, или у супротном, ако сваки процес врати резултат да је број прошао тест, онда се може рећи да је дати број са одређеном вероватноћом прост.

Због великог значаја који тестови простости имају у криптографији са јавним кључем, нарочито код криптосистема DSA и RSA, као и у Дифи-Хелман размени кључева за системе са тајним кључем, у **Поглављу 6** дати су основни појмови и битни криптографски алгоритми. Разноврсне идеје и резултати из теорије бројева веома утичу на развој **криптографије**. Заузврат, свеприсутност криптографије у данашњем технолошком свету мотивисала је обимна истраживања многих поља теорије бројева, укључујући и тестове прималности. Постављају се нови рекорди у проналажењу највећег простог броја, највећих простих бројева близанаца, повећавају се лимити до којих су проверене многе хипотезе, као на пример Риманова и Голдбахова хипотеза. Унапређују се алгоритми и софтвер за рад са великим (простим) бројевима, доказују нове теореме везане за расподелу простих бројева и просте бројеве у аритметичким прогресијама, креирају бољи тестови простости и методе за факторизацију.

У **Поглављу 7** је дат приказ програмских реализација одабраних тестова простости, садржаних у прилогу који иде уз овај рад. У додатку су дате и верзије програма за просејавање простих бројева, пробног дељења и факторизације, функција за израчунавање највећег заједничког делиоца, Јакобијевог симбола.

Посебна пажња је посвећена **програмској реализацији одређених тестова простости**. Имплементирани су Фермаов тест, Соловеј-Штрасенов тест, Милер-Рабинов тест, n - 1 тест, Пепинов тест, Лукас-Лемеров тест за Мерсенове бројеве, и AKS тест. Реализације вероватносних тестова су урађене у C-у, а употребом библиотеке BigInteger у програмском језику Јава реализовани су сви наведени тестови. Времена извршавања програма су упоређена за тестирање бројева различите величине, у просеку до 16,384 бита, док су специјализовани тестови за Фермаове и Мерсенове бројеве тестирани за бројеве величине и до 131,000 битова. Одговарајући графици приказују понашање ових тестова у зависности од величине тестираног броја.

У прилогу су дати примери употребе функција из програмског пакета Mathematica које илуструју значајне концепте из теорије бројева, а могу се користити и за практичну проверу многих резултата и тврђења изложених у овом раду.

2 Теорија бројева

Теорија бројева[4] је велико и фасцинантно поље математике које проучава особине како целих, тако и рационалних бројева. То је једна од најстаријих грана математике. Први проблеми из теорије бројева записани су још у старом Вавилону и Египту 2000–3000 година п.н.е. Основе теорије бројева поставили су старогрчки математичари. Још су Питагорејци били заинтересовани за мистична и нумеричка својства простих бројева, и разумели идеју савршених бројева.

2.1 Елементарни појмови алгебре и теорије бројева

Дефиниција 2.1 (Група)**. Група** $(\mathbb{G}, \circledcirc)$ је скуп $\mathbb{G}$ са бинарном операцијом $\circledcirc$ која сваком уређеном пару (a, b) елемената из $\mathbb{G}$ додељује нови елемент $a \circledcirc b$ из $\mathbb{G}$, (тј. $\forall a, b \in \mathbb{G}$, $a \circledcirc b \in \mathbb{G}$) и која задовољава следеће особине:

1. **Асоцијативност**: За свака три елемента $a, b, c \in \mathbb{G}$, важи: $(a \circledcirc b) \circledcirc c = a \circledcirc (b \circledcirc c)$.
2. **Неутрал**: Постоји елемент e из $\mathbb{G}$ такав да за свако a из G, важи: $e \circledcirc a = a \circledcirc e = a$. Елемент e је једнозначно одређен.
3. **Инверз**: За сваки елемент a из $\mathbb{G}$, постоји елемент b, такође из G, такав да је $a \circledcirc b = b \circledcirc a = e$, где је e неутрал. За свако изабрано a, елемент b је једнозначно одређен.

Дефиниција 2.2 (Абелова група)**. Абелова група** (**комутативна група**) је група $(\mathbb{G}, \circledcirc)$ у којој важи закон комутативности:

$$(\forall a, b \in \mathbb{G})\ a \circledcirc b = b \circledcirc a$$

Дефиниција 2.3 (Подгрупа)**.** Ако је $\mathbb{H}$ подскуп скупа $\mathbb{G}$, $\mathbb{H} \subset \mathbb{G}$, онда је $(\mathbb{H}, \circledcirc)$ **подгрупа** групе $(\mathbb{G}, \circledcirc)$ ако је $\circledcirc$ бинарна операција на $\mathbb{H}$ и $(\mathbb{H}, \circledcirc)$ је група.

Дефиниција 2.4. Број елемената коначне групе $\mathbb{G}$ је **ред групе**, у ознаци $|\mathbb{G}|$.

Пример 2.1 (Примери група)**.**

- $(\mathbb{Z}, +)$ је најмања *адитивна* група која садржи природне бројеве.
- Скуп рационалних бројева $\mathbb{Q} = \{\frac{p}{q} | p \in \mathbb{Z}, q \in \mathbb{N}\}$ је група у односу на операцију сабирања, у ознаци $(\mathbb{Q}, +)$.
- $(\mathbb{R}, +)$, $(\mathbb{C}, +)$ су *адитивне* групе.
- $(\mathbb{Q}\backslash\{0\}, \cdot)$, $(\mathbb{R}\backslash\{0\}, \cdot)$, $(\mathbb{C}\backslash\{0\}, \cdot)$ су *мултипликативне* групе.
- $(\mathbb{Q}[\sqrt{2}], \cdot)$ је мултипликативна група свих реалних бројева облика $a = p + q\sqrt{2}$, $p, q \in \mathbb{Q}$.
- $a \in (\mathbb{G}, \cdot)$ је **инвертибилан** елемент (енгл. *invertible element*) ако постоји елемент $b \in (\mathbb{G}, \cdot)$ такав да је $a \cdot b = 1$. Елемент b је **мултипликативни инверз** елемента a.
- **Група пермутација**. Пермутације се могу посматрати као бијективна пресликавања π, таква да је $\pi : \mathbb{G} \to \mathbb{G}$, $\mathbb{G} = \{1, 2, ..., n\}$, $n > 1$ природан број. Ако посматрамо две пермутације π, ρ над истим скупом $\mathbb{G}$, њихова композиција $\rho \circ \pi$ такође је пермутација од n елемената скупа $\mathbb{G}$. Одавде лако следи да је $(\mathbb{S}_n, \circ)$, $\mathbb{S}_n = \{\pi : \mathbb{G} \to \mathbb{G} | \pi$ је бијекција$\}$, група пермутација [9]. Комутативност не важи. Пермутације и инверзи пермутација се често користе у различитим криптографским апликацијама.

Дефиниција 2.5 (Прстен)**.** Нека је $\mathbb{G}$ скуп са две бинарне операције које се обележавају адитивно и мултипликативно. Ако је $(\mathbb{G}, +)$ Абелова група, док множење задовољава аксиому асоцијативности и у $(\mathbb{G}, \cdot)$ постоји јединични елемент, и важи дистрибутивност (за свака три елемента $a, b, c \in \mathbb{G}$ важи $(a + b) \cdot c = a \cdot c + b \cdot c$ и $a \cdot (b + c) = a \cdot b + a \cdot c$), каже се да је скуп $\mathbb{G}$ са ове две операције **прстен** (енгл. *ring*).

[4] Гаус (Carl Friedrich Gauss), 1777–1855, је рекао: "Математика је краљица наука, а теорија бројева краљица математике."

Дефиниција 2.6 (Поље)**.** Скуп $\mathbb{G}$ са бинарним операцијама сабирања и множења, у ознаци $(\mathbb{G}, +, \cdot)$, је **поље** (енгл. *field*) ако су $(\mathbb{G}, +)$ и $(\mathbb{G} \setminus \{0\}, \cdot)$ Абелове групе и важи дистрибутивни закон.

Напомена 2.1. Елемент $a \in (\mathbb{G}, +, \cdot)$ је **делитељ нуле** ако је $a \cdot b = 0$ за бар једно $b \neq 0$. Ако је и $a \neq 0$, онда је a прави делитељ нуле. **Домен** је комутативни прстен без правих делитеља нуле. Сваки *коначни* домен је поље. Прстен $(\mathbb{Z}, +, \cdot)$ је пример који показује да комутативни прстен без правих делитеља нуле не мора бити и поље.

Пример 2.2.

- У сваком прстену се може дефинисати и операција одузимања, као операција супротна операцији сабирања, а у пољу и операција дељења — инверзна операција операцији множења. На пример, операције сабирања, одузимања и множења су затворене у $\mathbb{Z}$, док операција дељења није.
- $\mathbb{Q}, \mathbb{R}$, и $\mathbb{C}$ су поља.

Дефиниција 2.7. Карактеристика поља $(\mathbb{G}, +, \cdot)$, где је 0 неутрал и $a \neq 0$ елемент поља, је најмањи позитиван цео број k такав да важи: $\underbrace{a + ... + a}_{k\ puta} = 0$. Ако такав број не постоји, каже се да је карактеристика поља 0.

Другачије речено, карактеристика поља $(\mathbb{G}, +, \cdot)$ је коначан адитивни ред јединице.

Посматрајмо *мултипликативну* групу $(\mathbb{G}, \cdot)$ и елемент $a \in \mathbb{G}$. Свака подгрупа $(\mathbb{H}, \cdot)$ групе $(\mathbb{G}, \cdot)$ која садржи елемент a, мора садржати и све степене a^n. Ово се лако показује. Ако су $a, b \in \mathbb{H}$, узима се да је $a = b = c$. По дефиницији, ако је $(\mathbb{H}, \cdot)$ подгрупа, важи да је $a \cdot b = c \cdot c = c^2 \in \mathbb{H}$. Нека је сада $a = c, b = c^2$. Онда важи да је $a \cdot b = c \cdot c^2 = c^3 \in \mathbb{H}$, итд.
Скуп свих степена елемента $a \in (\mathbb{G}, \cdot)$, у ознаци $< a >= \{a^n | n \in \mathbb{Z}\}$, је комутативна подгрупа у $(\mathbb{G}, \cdot)$. Аналогно, у адитивној групи $(\mathbb{G}, +)$ је $< a >= \{n \cdot a | n \in \mathbb{Z}\}$.

Дефиниција 2.8. Група $< a >$ назива се **циклична група** генерисана елементом a. **Ред елемента** a, у ознаци $r(a)$, је најмањи број $k \in \mathbb{N}$, такав да у мултипликативној групи важи $a^k = e$, e је неутрални елемент, односно $k \cdot a = e$ у адитивној групи. Ако такав број k постоји, a је коначног реда, док је $< a >$ **коначна циклична група**. У супротном, a је бесконачног реда, и $< a >$ је **бесконачна циклична група**.
У општем случају, ред елемента $a \in (\mathbb{G}, \circledcirc)$ је најмањи број $k \in \mathbb{N}$, такав да је $\underbrace{a \circledcirc ... \circledcirc a}_{k\ puta} = e$.

Ако постоји елемент $a \in \mathbb{G}$ такав да је $\mathbb{G} =< a >$, каже се да је $\mathbb{G}$ циклична група генерисана елементом a.

Наредна дефиниција помаже да се уоче и боље разумеју сличности између различитих група.

Дефиниција 2.9. Пресликавање $h : \mathbb{G} \to \mathbb{H}$ групе $(\mathbb{G}, *)$ у групу $(\mathbb{H}, \star)$ је **хомоморфизам** ако за свака два елемента $a,\ b \in \mathbb{G}$ важи $h(x * y) = h(x) \star h(y)$. Ако је ово пресликавање и бијекција, каже се да је то **изоморфизам** група. Групе $(\mathbb{G}, *)$ и $(\mathbb{H}, \star)$ су **изоморфне**, у ознаци $\mathbb{G} \cong \mathbb{H}$, ако постоји изоморфизам $h : \mathbb{G} \to \mathbb{H}$.

Ако је $h : \mathbb{G} \to \mathbb{H}$ хомоморфизам групе $(\mathbb{G}, *)$ у групу $(\mathbb{H}, \star)$, онда важи: $h(e_1) = e_2$, e_1 је неутрал у $\mathbb{G}$, e_2 је неутрал у $\mathbb{H}$; $h(a^{-1}) = h(a)^{-1}$; $h(a^k) = h(a)^k$, $k \in \mathbb{Z}$.
Релација изоморфности међу групама је релација еквиваленције.

Пример 2.3.

- $(\mathbb{Z}, +)$ је бесконачна циклична група генерисана елементом 1, $\mathbb{Z} =< 1 >$.
- Произвољна група $(\mathbb{Z}_n, +)$, $\mathbb{Z}_n = \{0, 1, ..., n-1\}$, је коначна циклична група генерисана елементом 1, односно $\mathbb{Z}_n =< 1 >$. Ово тврђење се једноставно показује. Сваки $k \in (\mathbb{Z}_n, +)$, $0 < k < n$, произвољни елемент који није неутрал, може се записати као $\underbrace{1 + ... + 1}_{k\ puta} = k \cdot 1 \neq 0$.
  Како важи $\underbrace{1 + ... + 1}_{n\ puta} = n \cdot 1 = 0$ у $\mathbb{Z}_n$, ред елемента 1 у групи $(\mathbb{Z}_n, +)$ је једнак $r(1) = n$.

- Све подгрупе групе $(\mathbb{Z}, +)$ су облика $< d >= d\mathbb{Z} = \{d \cdot n | n \in \mathbb{Z}\}$, $d \in \mathbb{Z}$. На пример, $(2\mathbb{Z}, +)$ је група свих парних бројева, и подгрупа групе $(\mathbb{Z}, +)$.

- Ред сваке подгрупе коначне групе мора делити ред групе (**Лагранжева теорема**). Одавде следи да ако је ред групе прост број, онда она нема правих подгрупа.

- Ред сваког елемента групе мора делити ред групе.

- Свака група простог реда мора бити циклична.

- Свака бесконачна циклична група је изоморфна $\mathbb{Z}$.

- Свака коначна циклична група је изоморфна $\mathbb{Z}_n$.

- **Остаци по модулу** n. Нека је $n > 1$ фиксиран цео број и нека је $\mathbb{Z}_n = \{0, 1, 2, ..., n-1\}$. У овај скуп се може увести бинарна операција $+_n$ на следећи начин:

  Ако су $a, b \in \mathbb{Z}_n$, нека је $a +_n b = r$ остатак дељења броја $a + b$ са n. Тај остатак је једнозначно одређен и $r \in \mathbb{Z}_n$. Ова бинарна операција се назива **сабирање по модулу** n, а $\mathbb{Z}_n$ је **скуп остатака по модулу** n.

- $(\mathbb{Z}_n, +_n)$ је Абелова група, тзв. **група остатака по модулу** n. За асоцијативност треба показати да за $\forall a, b, c \in \mathbb{Z}_n$ важи $(a +_n b) +_n c = a +_n (b +_n c)$. Нека је $a +_n b = p$, $p +_n c = q$, $b +_n c = r$, $a +_n r = s$, где $p, q, r, s \in \mathbb{Z}_n$. Одавде следи да је $a + b = t \cdot n + p$, $p + c = u \cdot n + q$, $b + c = v \cdot n + r$, $a + r = w \cdot n + s$. То значи да је $a + b + c = t \cdot n + u \cdot n + q = (t + u) \cdot n + q$. С друге стране, $a + b + c = a + v \cdot n + r = w \cdot n + v \cdot n + r = (w + v) \cdot n + r$. Сада због јединствености остатка следи да је $q = r$, а одатле тражено тврђење.
  Неутрал је 0. Инверз броја $a \in \mathbb{Z}_n$ је број $n - a \in \mathbb{Z}_n$. Доказ: $a + (n - a) = n = n \cdot 1 + 0 \Rightarrow a +_n (n - a) = 0$. Комутативност очигледно важи.

- $(\mathbb{Z}_n, +_n, \cdot_n)$ је комутативни прстен са јединицом, **прстен остатака по модулу** n.

- **Када је** p **прост број,** $\mathbb{Z}_p$ **представља поље** и његова карактеристика је p. Ово се једноставно доказује ако се претпостави супротно и искористи чињеница да у пољу производ два елемента различита од нуле не може бити нула. На сличан начин може се доказати супротно тврђење: **ако је** $\mathbb{Z}_p$ **поље,** p **је прост број**. Треба приметити да ако је карактеристика поља број p, онда је p прост број.

- $(\mathbb{Z}_5 \setminus \{0\}, \cdot)$ је мултипликативна Абелова група елемената $\{1, 2, 3, 4\}$. Ред ове групе је 4. Ред елемената групе: $r(1) = 1$, $r(2) = 4$, $r(3) = 4$, $r(4) = 2$, ред елемента дели ред групе. При томе у $(\mathbb{Z}_5 \setminus \{0\}, \cdot)$ важи $2 = 2$, $2^2 = 4$, $2^3 = 3$, $2^4 = 1$, односно $3 = 3$, $3^2 = 4$, $3^3 = 2$, $3^4 = 1$, одакле следи да су 2 и 3 генератори групе $(\mathbb{Z}_5 \setminus \{0\}, \cdot)$. Наравно, $(\mathbb{Z}_5, +)$ је комутативна група елемената $\{0, 1, 2, 3, 4\}$, реда $|\mathbb{Z}_5| = 5$, док је $\mathbb{Z}_5$ поље карактеристике $char\ \mathbb{Z}_5 = 5$.
  Такође, види се да је $(\{1, 4\}, \cdot)$ циклична подгрупа групе $(\mathbb{Z}_5 \setminus \{0\}, \cdot)$ генерисана елементом 4. Ред ове подгрупе, и ред броја 4, је 2. Ред подгрупе дели ред групе, тј. $2|4$.

Више о овим тврђењима и осталим основним појмовима алгебре може се наћи у [8, 9].

Елементарни појмови теорије бројева

Дефиниција 2.10. За целе бројеве a и $b \neq 0$ каже се да b **дели** a, у ознаци $b|a$, или да је b **фактор** тј. **делитељ** броја a, ако постоји цео број n такав да је $a = b \cdot n$.

Напомена 2.2. У скупу $\mathbb{N}$ релација дељивости је **релација поретка**. Рефлексивност је задовољена јер за $\forall n \in \mathbb{N}$ важи $n \mid n$. Релација дељивости је антисиметрична јер за $\forall n, m \in \mathbb{N}$, $n \mid m \wedge m \mid n \Rightarrow n = m$. Транзитивност следи из: $\forall n, m, k \in \mathbb{N}$, $n \mid m \wedge m \mid k \Rightarrow n \mid k$.
Битно је напоменути да у подскупу $\mathbb{N} \setminus \{1\}$ нема најмањег елемента у односу на релацију дељивости, али има бесконачно много минималних — то су сви прости бројеви[5] [9].

[5] Ово запажање може послужити као један од разлога зашто број 1 није прост број.

Дефиниција 2.11. Највећи заједнички делитељ (енгл. *greatest common divisor, gcd*) целих бројева a и b, у ознаци $\text{nzd}(a,b) = d$, је највећи цео број d такав да важи: $d|a$ и $d|b$.

Дефиниција 2.12. Два цела броја a и b су **узајамно прости** (енгл. *coprime*) ако је $\text{nzd}(a,b) = 1$.

Другачије речено, a и b су узајамно прости ако немају заједничких делитеља.

Дефиниција 2.13. Каже се да су бројеви $a_1, a_2, ..., a_m$ **узајамно прости у паровима** ако је $\text{nzd}(a_i, a_j) = 1$, за $i = 1, 2, ..., m$, $j = 1, 2, ..., m, i \neq j$.

Пример 2.4.

- Бројеви 176 и 105 су узајамно прости јер је $\text{nzd}(105, 176) = 1$, али ни 105 нити 176 нису прости.
- Ако $c|a$ и $c|b$, онда број $c \neq 0$ дели и сваку линеарну комбинацију бројева a, b, односно: $c|a \wedge c|b \Leftrightarrow (\forall x, y \in \mathbb{Z})\ c|(ax + by)$. На пример, $2|6$, $2|10$ и $2|(6x + 10y)$ за све целе бројеве x, y.

Теорема 2.1 (Алгоритам дељења). *Ако су a и b цели бројеви и $b > 0$, онда постоје јединствени цели бројеви q и r такви да је $a = q \cdot b + r$ и $0 \leq r < b$.*

Доказ. Дат је доказ за природан број a. Случај када је a негативан цео број доказује се аналогно. Добра уређеност скупа $\mathbb{N}$ је од кључне важности. Ако је $q \cdot b$ највећи садржалац од b такав да је мањи или једнак броју a, онда је цео број $r = a - qb$ сигурно ненегативан и, како је $b(q + 1) > a$, важи да је $r = a - qb < b(q + 1) - qb = b$. Јединственост: ако постоје q', r' са истим својствима, такви да важи $a = q'b + r'$, тада је $r - r' = (q - q')n$, односно дељиво са n, и $-n < r - r' < n$. Једини број између $-n$ и n дељив са n је 0, одакле следи јединственост. □

Напомена 2.3. Претходна теорема се назива и "**Лема о остатку**". Она даје основе за дефинисање операција сабирања и множења по модулу.

Дефиниција 2.14. Број q из претходне теореме називамо **количником**, а број r **остатком** при дељењу броја a бројем b. Наравно, ако је a дељиво са b, остатак је нула.

Напомена 2.4. Како теорема остаје тачна за свако $b \neq 0$, каже се да постоје јединствени цели бројеви q и r такви да је $a = q \cdot b + r$ и $0 \leq r < |b|$.

Овде је реч о **еуклидском дељењу**[6]. Ако се, на пример, у програмском језику C целобројно подели број -7 са -3, добија се да је количник једнак $q = 2$ а остатак $r = -1$, док се користећи претходно тврђење добија да је $q = 3$, $r = 2$. Такође, према претходној теореми, количник при дељењу -7 са 3 је $q = -3$, а остатак $r = 2$.

Теорема 2.2. *Ако је $d = \text{nzd}(a,b)$, онда постоје цели бројеви $x, y \in \mathbb{Z}$ такви да је $x \cdot a + y \cdot b = d$.*

Доказ. Посматрају се цели бројеви облика $xa + yb$, $x, y \in \mathbb{Z}$. Изабира се најмањи природан број таквог облика, означимо га са $n = xa + yb$. Треба доказати да $n|a$, $n|b$. Претпоставимо супротно, да n не дели a. На основу леме о остатку, постоје јединствени $q, r \in \mathbb{Z}$, $0 < r < n$, такви да је $a = nq + r$. Али, тада је $r = a - n \cdot q = a - q(xa + yb) = (1 - xq)a - yqb$. Одавде следи да је r природан број мањи од n и истог облика као n, што је контрадикција са претпоставком. Дакле, $n|a$. Аналогно се доказује да $n|b$.

Сада треба показати да је n највећи такав број, тј. да је $n = d$, $d = \text{nzd}(a,b)$. Пошто је $d = \text{nzd}(a,b)$, важи $a = q_1 \cdot d$, $b = q_2 \cdot d$, за неке целе бројеве q_1, q_1. Тада је $n = xq_1 \cdot d + yq_2 \cdot d = d(xq_1 + yq_2)$. Одавде следи да $d|n$, односно $d \leq n$. Како је d *највећи* заједнички делитељ бројева a и b, следи да је $d = n$, а одатле тражено тврђење [25]. □

Напомена 2.5. На основу доказа претходне теореме, може се закључити да је највећи заједнички делитељ бројева a и b најмањи позитиван цео број облика $xa + yb$, $x, y \in \mathbb{Z}$. Специјално, ако су a, $b \in \mathbb{Z}$ узајамно прости, односно $d = \text{nzd}(a,b) = 1$, онда постоје цели бројеви $x, y \in \mathbb{Z}$ такви да је $xa + yb = 1$.

[6] У наредном поглављу детаљно је описан Еуклидов алгоритам.

Напомена 2.6. Цели бројеви a и b су узајамно прости ако и само ако постоје $x, y \in \mathbb{Z}$ такви да је $xa + yb = 1$.

Може се лако показати да важе следећа тврђења:

Тврђење 2.1. *За целе бројеве $a, b \in \mathbb{Z}$ важи:* $\text{nzd}(a, b) = nzd(a, a + b)$, $\text{nzd}(a, b) = \text{nzd}(a, a - b)$.

Тврђење 2.2. *Ако је $d = \text{nzd}(a, b)$, и са $s = [a, b]$ означимо* ***најмањи заједнички садржалац*** *целих бројева $a, b \in \mathbb{Z}$, онда важи: $d \cdot s = |a \cdot b|$. Специјално, ако је $d = 1$, $s = |a \cdot b|$.*

Теорема 2.3 (**Основна теорема аритметике**). *Сваки позитиван цео број n, такав да је $n > 1$, има јединствену факторизацију $n = p_1^{a_1} p_2^{a_2} \cdot ... \cdot p_k^{a_k}$, где су $a_i > 0$ цели бројеви, а p_i прости бројеви и важи $p_i < p_{i+1}$.*

Другим речима, сваки позитиван цео број n се може на јединствен начин представити као производ својих чинилаца, до на њихов редослед. Ова репрезентација се назива **канонски облик или канонска факторизација** позитивног целог броја n.
Одавде следи да је сваки цео број n, такав да је $n > 1$, или прост број, или производ простих бројева.

> Проблем раздвајања простих бројева од сложених бројева и проблем факторизације сложених бројева у просте чиниоце су међу најзначајнијим и најкориснијим проблемима у аритметици.[7]

Дефиниција 2.15. Нека је $n = p_1^{a_1} p_2^{a_2} \cdot ... \cdot p_k^{a_k}$ канонска факторизација броја n, и означимо са $\tau(n)$ **укупан број позитивних делилаца** броја n, укључујући број n и 1. Тада је $\tau(n) = (a_1 + 1) \cdot (a_2 + 1) \cdot ... \cdot (a_k + 1)$.

Основна теорема аритметике показује значај простих бројева: прости фактори целих бројева одредjују особине целих бројева. Такође, ако се зна канонска факторизација целих позитивних бројева a и b, лако се може одредити њихов највећи заједнички делилац и најмањи заједнички садржалац.

Пример 2.5.

- Нека је a позитиван цео број и p прост број. За канонску факторизацију $a = p$ је $\tau(a) = 2$. Ако је $a = p^n$ канонски облик броја a, тада је $\tau(a) = (n+1)$. Ако је $a = p_1 \cdot p_2 \cdot ... \cdot p_n$ канонска факторизација броја a, тада је $\tau(a) = 2^n$.
- Ако је $\tau(n)$ непаран број, n је **потпун квадрат** (енгл. *perfect square*). Из чињенице да ако би неки степен у канонској факторизацији броја n био непаран, онда би $\tau(n)$ био паран, следи да су сви степени у канонском облику броја n парни, односно да је n потпун квадрат.
- Лако се може показати да је $\tau(n) < 2 \cdot \sqrt{n}$. На пример, за $n = 16 = 2^4$ је $\tau(16) = 4 + 1 = 5$ и $5 < 2 \cdot \sqrt{16} = 8$.
- Нека је m произвољан цео број и $m = nq + r$. Онда је са $h : \mathbb{Z} \to \mathbb{Z}_n$, $h(m) = r$ дефинисан један хомоморфизам. Треба показати да важи $h(a+b) = h(a) +_n h(b)$. Ако су $a = nq_1 + r_1$, $b = nq_2 + r_2$ произвољни цели бројеви, тада је $h(a) = r_1$, $h(b) = r_2$, и $h(a + b) = h(nq_1 + r_1 + nq_2 + r_2) = h(r_1 + r_2) = r$, где је $r_1 + r_2 = nq_3 + r$. С друге стране, на основу дефиниције операције $+_n$ важи да је $h(a) +_n h(b) = r_1 +_n r_2 = r$, одакле следи тврђење. Приметимо да је ово пресликавање "на" односно епиморфизам, али није "1 − 1" јер различити бројеви могу имати исти остатак по модулу n.
- Нека је $d > 0$ произвољан цео број. Онда је са $h : \mathbb{Z} \to d\mathbb{Z}$, $h(a) = d \cdot a$, $a \in \mathbb{Z}$, дефинисан један изоморфизам.

Дефиниција 2.16. Савршен број[8] је цео број већи од 1, једнак суми свих својих делитеља (осим њега самог), тј. важи $\sigma(n) = 2n$, $\sigma(n)$ — сума свих делилаца броја n.

[7] Гаус, *Disquisitiones Arithmeticae*, 1801.
[8] Декарт (René Descartes), 1596–1650, је рекао: "Савршени бројеви су, баш као и савршени људи, веома ретки."

На пример, савршени бројеви су: $6 = 1 + 2 + 3$, $28 = 1 + 2 + 4 + 7 + 14$, 496, 8128, итд.

Тврђење 2.3. *Паран број је савршен ако и само ако је облика $2^{n-1}(2^n - 1)$, где су n и $2^n - 1$ прости бројеви.*

Доказ. $\boxed{\Leftarrow}$ Још је Еуклид доказао да ако је број $2^n - 1$ прост[9], онда је број $2^{n-1}(2^n - 1)$ савршен број. Ово се лако показује. Нека је $N = 2^{n-1}(2^n - 1)$. Треба показати да је $\sigma(N) = 2 \cdot 2^{n-1} \cdot (2^n - 1) = 2^n \cdot (2^n - 1)$. Ако је $2^n - 1$ прост број, онда су једини његови делитељи 1 и $2^n - 1$. Делитељи броја 2^{n-1} су $2^0, 2^1, ..., 2^{n-1}$. Сада важи да је $\sigma(N) = (1 + 2^n - 1)(2^0 + 2^1 + ... + 2^{n-1}) = 2^n \cdot \sum_{i=0}^{n-1} 2^i$, одакле следи тврђење.
$\boxed{\Rightarrow}$ Ојлер[10] је показао да су сви парни савршени бројеви облика $2^{n-1}(2^n - 1)$. Сваки паран број $N > 0$ може се записати у облику $N = 2^n \cdot m$, где је $n > 0$ цео број, и $m > 0$ непаран цео број. Како је N савршен, онда је $\sigma(N) = 2 \cdot N = 2^{n+1} \cdot m$. Као што је већ показано, збир делитеља броја 2^n је $2^{n+1} - 1$. Одавде следи да је $\sigma(N) = 2^{n+1} \cdot m = (2^{n+1} - 1) \cdot \sigma(m)$, односно да је $m = (2^{n+1} - 1)k$ и $\sigma(m) = 2^{n+1} \cdot k$, за неки позитиван цео број k. Нека је $k > 1$. Тада број m има делитеље 1, m, k и $\sigma(m) \geq 1 + m + k$. Али, важи да је $k + m = k + (2^{n+1} - 1)k = 2^{n+1} \cdot m = \sigma(m)$, што је контрадикција са претпоставком. Према томе је $k = 1$ и $\sigma(m) = 2^{n+1} = m + 1$. Одавде следи да је $m = 2^{n+1} - 1$ прост број. Доказ да ако је $m = 2^{n+1} - 1$ прост број, онда је $n + 1$ прост број дат је у поглављу о Мерсеновим бројевима [3]. □

Није познато да ли постоји непаран савршен број. До 2012. год. су проверени бројеви до 10^{1500} и није нађен ниједан непаран савршен број.

2.2 Конгруенције. Мала Фермаова теорема. Ојлерова теорема

Дефиниција 2.17. Два цела броја a и b су **конгруентна по модулу** n ако је њихова разлика дељива са n или, еквивалентно, ако имају исти остатак при дељењу са n:

$$a \equiv b \pmod{n} \text{ акко } n|(a - b).$$

Пример 2.6.

- Бројеви 7 и 12 су еквивалентни по модулу 5: $7 = 2 \pmod 5$, $12 = 2 \pmod 5$, $5|(12 - 7)$.
- Ако је $a \cdot b \equiv 1 \pmod n$, онда каже се да је b **мултипликативни инверз од a по модулу** n. Мултипликативни инверз: $2 \cdot 3 = 1 \pmod 5$, $4 \cdot 4 = 1 \pmod 5$. Приметимо да $4 = 5 - 1$ и $4 \equiv -1 \pmod 5$.
- Квадрат произвољног целог броја има остатак 0 или 1 по модулу 4, тј. **квадрат непарног броја има остатак 1 по модулу 4**. Сваки цео број може се записати у облику $2k$ или $2k + 1$, k цео број. Онда је $(2k)^2 = 4k^2$, и $4k \equiv 0 \pmod 4$, односно $(2k + 1)^2 = 4k^2 + 4k + 1$, и $4k^2 + 4k + 1 \equiv 1 \pmod 4$, па важи тврђење.
- **Квадрат произвољног целог броја има остатак** 0 **или** 1 **по модулу** 3. Сваки цео број може се записати у облику $3k$, $3k + 1$, $3k - 1$, k цео број. Онда је $(3k)^2 = 9k^2$, $(3k + 1)^2 = 9k^2 + 6k + 1 = 3(3k^2 + 2k) + 1$, $(3k + 2)^2 = 9k^2 + 12k + 4 = 3(3k^2 + 4k + 1) + 1$, одакле следи тврђење.
- Претходна два примера, заједно са чињеницом да последња цифра квадрата произвољног целог броја може бити само једна од цифара $0, 1, 4, 5, 6, 9$, дају једноставнију проверу да ли је дати број може бити квадрат неког целог броја.

Тврђење 2.4. *За све $a, b \in \mathbb{Z}$ важи:*

1. $a \equiv b \pmod n \Leftrightarrow (\exists q \in \mathbb{Z})\ a = n \cdot q + b.$
2. $a \equiv b \pmod n \Leftrightarrow (\exists q_1, q_2 \in \mathbb{Z})\ a = n \cdot q_1 + r,\ b = n \cdot q_2 + r,\ 0 \leq r < n.$

[9] Бројеви овог облика, тзв. Мерсенови *прости* бројеви, дефинисани су касније у тексту.
[10] Ојлер (Leonhard Euler), 1707–1783, швајцарски математичар и физичар.

3. $a \equiv b \pmod{n},\ c \equiv d \pmod{n} \Rightarrow (\forall x, y \in \mathbb{Z})\ a \cdot x + c \cdot y \equiv b \cdot x + d \cdot y \pmod{n}$.

4. $a \equiv b \pmod{n},\ c \equiv d \pmod{n} \Rightarrow a \cdot c \equiv b \cdot d \pmod{n}$.

5. $a \equiv b \pmod{n},\ n = q \cdot d,\ d > 0 \Rightarrow a \equiv b \pmod{d}$

6. $a \equiv b \pmod{n},\ a \equiv b \pmod{m} \Leftrightarrow a \equiv b \pmod{[n, m]}$, *где је* $[n, m]$ *најмањи заједнички садржалац бројева* n *и* m.

7. *Ако је* $P(x)$ *полином по* x *са целим коефицијентима, онда ако је* $a \equiv b \pmod{n}$ *следи да је* $P(a) \equiv P(b) \pmod{n}$.

Последица 2.1. *Ако је* $a \equiv b \pmod{n}$, $a \equiv b \pmod{m}$, *и* $\mathrm{nzd}(n, m) = 1$, *онда је* $a \equiv b \pmod{n \cdot m}$.

Доказ. Из претпоставки теореме следи да је $a - b = c \cdot n$ и $a - b = d \cdot m$, за неке $c, d \in \mathbb{Z}$. Одатле важи да $c \cdot n = d \cdot m$, тј. да $n | d \cdot m$, а како је $\mathrm{nzd}(n, m) = 1$, следи да $n | d$. Зато постоји $q \in \mathbb{Z}$ такав да је $d = n \cdot q$, па је $a - b = q \cdot n \cdot m$, тј. $a \equiv b \pmod{n \cdot m}$. □

Бити **конгруентан по датом модулу је релација еквиваленције** у скупу целих бројева — задовољава особине рефлексивности, симетричности и транзитивности. Сви цели бројеви који су конгруентни по датом модулу образују једну **класу бројева**. Тако по модулу 2 постоје две дисјунктне класе бројева — парни и непарни бројеви, а по модулу 3 постоје три класе бројева: класа бројева облика $3k$, класа бројева облика $3k + 1$, и класа бројева облика $3k + 2$.

У општем случају, за цео број $n > 1$ посматрајмо класе целих бројева конгруентне бројевима $0, 1, ..., n - 1$ по модулу n. Ове класе су међусобно дисјунктне класе бројева којима су обухваћени сви цели бројеви — извршено је разлагање скупа $\mathbb{Z}$ на n класа. Сваки од бројева $0, 1, ..., n - 1$ *репрезентује* своју класу. Ови бројеви се могу изабрати и на други начин, једино је битно да из сваке класе постоји тачно један број.

Другачије речено, нека је $a = k \cdot q_1 + r_1$, $b = k \cdot q_2 + r_2$, где бројеви $a,\ b \in \mathbb{Z}$. Онда је са $a \sim b \Leftrightarrow a \equiv b \pmod{k}$ дефинисана једна еквиваленција у прстену $(\mathbb{Z}, +, \cdot)$ која се назива конгруенција (по модулу k). Ова конгруенција је сагласна са сабирањем и множењем. Класа конгруенције којој припада број k (и сви бројеви који су дељиви са k) је у ствари циклична група $(k\mathbb{Z}, +)$. Заправо, све класе конгруенције су облика $r + k\mathbb{Z}$, $0 \leq r < k$. Зато се количнички скуп $\mathbb{Z}/\sim$ означава и са $\mathbb{Z}/k\mathbb{Z} = \{r + k\mathbb{Z} : 0 \leq r < k\}$. Како је са $h(r) = r + k\mathbb{Z}$ дефинисан један изоморфизам прстена $(\mathbb{Z}_k, +, \cdot)$ на тај количнички прстен $\mathbb{Z}/k\mathbb{Z}$, следи да важи $\mathbb{Z}_k \cong \mathbb{Z}/k\mathbb{Z}$.

Дефиниција 2.18. Скуп целих бројева $a_1, a_2, ..., a_n$ је **потпун систем остатака** по модулу n уколико не постоје два елемента овог скупа која дају исти остатак по модулу n (ниједна два нису конгруентна по модулу n).

Дефиниција 2.19. Сведени систем остатака по модулу n се добија ако се из потпуног система остатака одстране бројеви који нису узајамно прости са n. У сведеном систему остатака има $\phi(n)$ елемената, $\phi(n)$ је број позитивних целих бројева мањих од n и узајамно простих са n.

Теорема 2.4 (Велика Фермаова теорема). *Фермаова*[11] *највећа и такође његова последња теорема гласи:* $x^n + y^n = z^n$ *нема нетривијалних решења међу целим бројевима* x, y, z *за* $n > 2$.[12]

Ферма је један од оснивача савремене теорије бројева. Решавање проблема које је он поставио, међу којима је и ова теорема, довело је до корисних резултата и значајних достигнућа. Велику Фермаову теорему је коначно доказао Вајлс 1995. године.[13]

Приметимо да за $n = 2$ постоји једначина $x^2 + y^2 = z^2$ чија решења представљају тзв. Питагорине тројке (x, y, z), јер ако те бројеве посматрамо као дужине страница неког троугла, онда је према Питагориној теореми тај троугао правоугли. На пример, $(x = 3,\ y = 4,\ z = 5)$, као и $(x = 5,\ y = 12,\ z = 13)$, и $(x = 7,\ y = 24,\ z = 25)$ су Питагорине тројке.

Примитивно решење једначине $x^2 + y^2 = z^2$ је решење у коме x, y, z немају заједничких делилаца. Налажењем свих примитивних решења $(x,\ y,\ z)$, налазе се и сва остала решења $(kx,\ ky,\ kz)$, $k \in \mathbb{Z}$.

[11] Ферма (Pierre de Fermat), 1607/8–1665, француски адвокат и математичар.

[12] Поред овог тврђења, Ферма је написао: "Имам заиста величаствен доказ ове теореме, али је маргина исувише уска да би он на њу стао."

[13] Вајлс (Andrew John Wiles), 1953– , британски математичар.

Лако се показује да примитивно решење (x, y, z) једначине $x^2 + y^2 = z^2$ у скупу природних бројева задовољава следеће једнакости: $x = n^2 - m^2$, $y = 2nm$, $z = n^2 + m^2$, где су n, m позитивни цели бројеви такви да је $\text{nzd}(n, m) = 1$, $n > m$.

Теорема 2.5 (Мала Фермаова теорема). *Ако је p прост број и ако је a цео број, онда важи:* $a^p \equiv a \pmod{p}$. *Специјално, ако p не дели a (тј.* $\text{nzd}(a, p) = 1$*), онда важи:*

$$a^{p-1} \equiv 1 \pmod{p}.$$

Доказ. Ако се цели број a подели са p, добија се да је $a = p \cdot q + r$, $r \in \mathbb{Z}_p$. Тада је $a^{p-1} - 1 = (p \cdot q + r)^{p-1} - 1 = p \cdot x + r^{p-1} - 1$, за неки цео број x. Ако p не дели a, тада је $r \in \mathbb{Z}_p^*$ и $(\mathbb{Z}_p^*, \cdot) = \{1, 2, ..., p-1\}$ је мултипликативна група реда $p - 1$. Онда је $r^{p-1} = 1$, тј. $r^{p-1} - 1 = 0$, одакле следи да је $a^{p-1} = 1 \pmod{p}$. Ако се обе стране једначине помноже са a, добија се да важи тврђење у општем случају $a^p \equiv a \pmod{p}$ за $\text{nzd}(a, p) = 1$. Ако p дели a, онда важи $a^p \equiv a \pmod{p}$ јер су обе стране једнаке 0 по модулу p. □

Мала Фермаова теорема је основа многих резултата у теорији бројева и основа многих метода које проверавају да ли су бројеви прости. Треба приметити да ова теорема не говори ништа о томе да ли је a прост или сложен број. Такође, као што је у поглављу 4 показано, постоје сложени бројеви $n \in \mathbb{Z}$ за које важи $a^{n-1} \equiv 1 \pmod{n}$, $\text{nzd}(a, n) = 1$.

Пример 2.7.

- Треба показати да ако је p непаран прост број и p не дели цео број a, тада је један и само један од бројева $a^{1+2+...+(p-1)} - 1$ и $a^{1+2+...+(p-1)} + 1$ дељив са p. Ово тврђење је еквивалентно са или је $a^{1+2+...+(p-1)} \equiv 1 \pmod{p}$ или $a^{1+2+...+(p-1)} \equiv -1 \pmod{p}$. Како је p непаран прост број, тада је $1 + 2 + ... + (p-1) = \frac{p \cdot (p-1)}{2}$ цео број. С друге стране, из Мале Фермаове теореме важи да је $a^{p-1} \equiv 1 \pmod{p}$, односно $a^{(p-1) \cdot p} \equiv 1^p \equiv 1 \pmod{p}$. Одавде следи да је или $a^{\frac{(p-1) \cdot p}{2}} \equiv 1 \pmod{p}$ или $a^{\frac{(p-1) \cdot p}{2}} \equiv -1 \pmod{p}$.

- Нека је p прост број, a, b цели бројеви. Тада је $(a+b)^p = a^p + \sum_{k=1}^{p-1} \frac{p!}{k!(p-k)!} \cdot a^{p-k} \cdot b^k + b^p$, па следи да је $(a+b)^p \equiv a^p + b^p \pmod{p}$. С друге стране, ако се узме да је $c = a + b$ и примени Малу Фермаову теорему, добија се да је $c^p \equiv c \pmod{p}$, односно $(a+b)^p \equiv a + b \pmod{p}$. На пример, за $p = 3$, $a = 2$, $b = 3$ важи да је $(2+3)^3 \equiv 2 + 3 \pmod{3} \equiv 2 \pmod{3}$. Такође, $(2+3)^3 \equiv 2^3 + 3^3 \pmod{3} \equiv 35 \pmod{3} \equiv 2 \pmod{3}$.

- Ако је $\mathbb{G}$ комутативни прстен карактеристике p, где је p прост број, онда за сваки цео број $n > 0$ важи $(a_1 + a_2 + \ldots + a_n)^p = {a_1}^p + {a_2}^p + \ldots + {a_n}^p$. За $\mathbb{G} = \mathbb{Z}_p$ и $a_1 = a_2 = \ldots = a_n = 1$, се добија тврђење Мале Фермаове теореме [8].

Најмањи број k за који важи $a^k \equiv 1 \pmod{n}$ је **ред или поредак броја a по модулу n**. Обележимо га са $r_n(a)$.

Ојлер је 1760. године дао општије тврђење од мале Фермаове теореме:

Теорема 2.6 (Ојлерова теорема). *Ако су a и n узајамно прости бројеви,* $\text{nzd}(a, n) = 1$*, тада је* $a^{\phi(n)} \equiv 1 \pmod{n}$*, где је са $\phi(n)$ означен број природних бројева, не већих од n, узајамно простих са n.*

Доказ. За доказ ове теореме користи се тврђење да је у комутативном прстену $\mathbb{G}$ скуп свих инвертибилних елемената једна мултипликативна група. У $\mathbb{Z}_n$ елементи узајамно прости са n су инвертибилни, и образују мултипликативну групу $(\mathbb{Z}_n^*, \cdot)$, која има $\phi(n)$ елемената. Број a је узајамно прост са n, па важи $a \equiv a_1 \pmod{n}$, $a_1 \in \mathbb{Z}_n^*$. Како се зна да ред елемента дели ред групе и $|(\mathbb{Z}_n^*, \cdot)| = \phi(n)$, следи тражено тврђење. □

Функција $\phi(n)$ се назива **Ојлерова функција**. У Ојлеровој теореми бројеви a и n не морају бити прости, већ само узајамно прости бројеви.

Пример 2.8.

- $\phi(1) = \phi(2) = 1$ и $\phi(3) = \phi(4) = 2$. Слично, $\phi(5) = 4$, јер су 1, 2, 3, и 4 узајамно прости са 5.
- Нека је $a = 7$, $n = 6$, $\text{nzd}(7, 6) = 1$. Једини природни бројеви мањи од 6 и узајамно прости са 6 су $\{1, 5\}$. Одатле следи да је $\phi(6) = 2$, $a^{\phi(6)} = 7^2 = 1 \pmod 6$.
- Пример сведеног система остатака: За $n = 12$ добија се сведени систем остатака: $1, 5, 7, 11$.
- Ако је $\{a_1, a_2, ..., a_k\}$ потпун систем остатака по модулу n и $\text{nzd}(b, n) = 1$, онда је и $\{b \cdot a_1$, $b \cdot a_2$, ..., $b \cdot a_k\}$ потпун систем остатака по модулу n. Аналогно тврђење важи за сведен систем остатака [25].
- Пример еквивалентности по модулу: $1 \equiv 5 \equiv -3 \pmod 4$. Заправо, за сваки цео број n, $1 + 4n \equiv 1 \pmod 4$.
- Нека је $p > 2$ прост број. Ред броја a по модулу p је $r_p(a) = 2$ ако и само ако је $a \equiv -1 \pmod p$. Ако је ред броја a по модулу p једнак 2, тј. $a^2 \equiv 1 \pmod p$, онда је или $a \equiv -1 \pmod p$ или $a \equiv 1 \pmod p$. Ако би било $a \equiv 1 \pmod p$, онда би ред броја a био 1, па следи да важи тврђење. С друге стране, ако је $a \equiv -1 \pmod p$, онда је $a^2 \equiv 1 \pmod p$, па следи и други део тврђења.

Теорема 2.7. *Цео број p је прост ако и само ако је $\phi(p) = p - 1$.*

Доказ. $\boxed{\Rightarrow}$ Како су за прост број p сви елементи скупа $1, 2, ..., p$ узајамно прости са p, (осим самог броја p), важи $\phi(p) = p - 1$. $\boxed{\Leftarrow}$ Ако је $\phi(p) = p - 1$, одатле следи да је $\text{nzd}(p, p) = p \neq 1$ и $\text{nzd}(a, p) = 1$ за све $a < p$, па одатле по дефиницији следи да је p прост број. □

Напомена 2.7. Приметимо да ако су m и n *узајамно прости бројеви*, онда се на основу основне теореме аритметике лако показује да је $\phi(mn) = \phi(m)\phi(n)$. На пример, $\phi(15) = \phi(3)\phi(5) = 2 \cdot 4 = 8$, $\phi(42) = \phi(6)\phi(7) = 2 \cdot 6 = 12$.

Последица 2.2 (Последица Ојлерове теореме)**.** *Ако је n производ простих бројева p и q, $n = p \cdot q$, и $\text{nzd}(a, n) = 1$, онда је $a^{(p-1)(q-1)} \equiv 1 \pmod n$, $n = p \cdot q$.*

Напомена 2.8. За било који степен p^k простог броја p, сваки мањи цео број је узајамно прост са p^k изузев умножака од p. Зато је $\phi(p^k) = p^k - p^{k-1} = (p - 1)p^{k-1}$.

Напомена 2.9. Ако је $p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k}$ факторизација целог броја n, тада је сваки степен простог броја узајамно прост са сваким другим, па важи:

$$\phi(n) = \phi(p_1^{a_1})\phi(p_2^{a_2}) \cdot ... \cdot \phi(p_k^{a_k}) = (p_1 - 1)p_1^{a_1 - 1}(p_2 - 1)p_2^{a_2 - 1} \cdot ... \cdot (p_k - 1)p_k^{a_k - 1}.$$

На основу овог тврђења, лако се може показати да важи $\phi(n \cdot m) = \phi(n)\phi(m)\frac{d}{\phi(d)}$, за $\text{nzd}(n, m) = d$.

За неколико наредних тврђења смо већ рекли да важе у општем случају. Сада су дати докази помоћу конгруенција.

Теорема 2.8. *Ред $r_n(a)$ броја a по модулу n постоји ако и само ако су a и n узајамно прости бројеви.*

Доказ. $\boxed{\Rightarrow}$ Ако је $a^k \equiv 1 \pmod n$ за неко k, онда је $a^k - 1 = n \cdot u$ за неко u, тј. важи: $a \cdot a^{k-1} - n \cdot u = 1$. Следи да је $\text{nzd}(a, n) = 1$. Дакле, ако ред $r_n(a)$ постоји, онда су a и n узајамно прости бројеви. $\boxed{\Leftarrow}$ Супротно, ако је $\text{nzd}(a, n) = 1$, онда је по Ојлеровој теореми $a^{\phi(n)} \equiv 1 \pmod n$ па ред броја постоји. □

Може се закључити да бројеви који имају ред по модулу n припадају сведеном систему остатака по модулу n.

Теорема 2.9. *Ако је $\text{nzd}(a, n) = 1$ и $a^k \equiv 1 \pmod n$, онда ред броја a по модулу n дели k.*

Доказ. Нека је m ред броја a по модулу n и нека је k најмањи цео број такав да важи: $a^k \equiv 1 \pmod{n}$. На основу алгоритма дељења, m се може представити у облику $m = q \cdot k + r,\ 0 \leq r < k$. Онда је $1 \equiv a^k \equiv a^{q \cdot k + r} \equiv (a^k)^q \cdot a^r \equiv 1^q \cdot a^r \equiv a^r \pmod{n}$. Ако је r позитиван цео број, то нарушава минималност броја k. Зато мора да важи да је $r = 0$, па је $m = q \cdot k$ и k дели m. □

Напомена 2.10. Не мора да важи: $r_n(a) = \phi(n)$, али увек важи: $r_n(a) | \phi(n)$ (ако је ред дефинисан то јест ако постоји).

Дефиниција 2.20. Ако је ред броја a по модулу n једнак $\phi(n)$, каже се да је a **примитивни корен по модулу** n, (енгл. *primitive root modulo n*).

Теорема 2.10. *Цео број n има примитивне корене ако и само ако је $n = 1,\ 2,\ 4,\ p^k$, или $2 \cdot p^k$, где је p непаран прост број. Даље, ако n има примитивне корене, он има тачно $\phi(\phi(n))$ примитивних корена.*

Другачије речено, примитивни корен по модулу n је **циклични генератор** мултипликативне групе $\mathbb{Z}_n^*$ сведеног система остатака по модулу n. Ова група је циклична ако и само ако $n > 4$ није дељив са 4 и није дељив са два различита непарна проста броја. Доказ овог тврђења може се наћи у [3].

Теорема 2.11. *Ако је a примитивни корен по модулу n, бројеви $a^0 = 1, a, a^2, \ldots, a^{\phi(n)-1}$ образују сведени систем остатака по модулу n.*

Последица 2.3. *Ако је k ред броја a по модулу n, онда важи $a^x \equiv a^y \pmod{n} \Leftrightarrow x \equiv y \pmod{k}$. Општије, ако је $\mathrm{nzd}(a, n) = 1$, онда важи $a^x \equiv a^y \pmod{n} \Leftrightarrow x \equiv y \pmod{\phi(n)}$.*

Пример 2.9.

- Сви бројеви који су конгруентни по модулу n, односно бројеви који припадају истој класи еквиваленције, имају исти поредак по модулу n. Наравно, бројеви из различитих класа могу имати исти поредак.
- За $n = 4$, примитивни корен је 3 јер важи $3^{\phi(4)} = 3^2 = 9,\ 9 = 1 \pmod{4}$.
- За $n = 3^2 = 9$ важи да је $\phi(9) = 6$, и $1^6 = 1 \pmod{9}$, $2^6 = 1 \pmod{9}$, $4^6 = 1 \pmod{9}$, $5^6 = 1 \pmod{9}$, $7^6 = 1 \pmod{9}$, $8^6 = 1 \pmod{9}$. Такође, $\phi(6) = \phi(3) \cdot \phi(2) = 2$, и 2 и 5 су примитивни корени, јер је $1 = 1 \pmod{9}$, $4^3 = 1 \pmod{9}$, $7^3 = 1 \pmod{9}$, $8^2 = 1 \pmod{9}$.
- Такође, како су 2 и 5 примитивни корени за $n = 9$, важи $2^0 = 1 \pmod{9}$, $2^1 = 2 \pmod{9}$, $2^2 = 4 \pmod{9}$, $2^3 = 8 \pmod{9}$, $2^4 = 7 \pmod{9}$, $2^5 = 5 \pmod{9}$ — сведени систем остатака по модулу 9. Аналогно за примитивни корен 5.

Теорема 2.12 (Вилсонова теорема[14]). *Ако је p прост број, онда важи:* $(p-1)! \equiv -1 \pmod{p}$.

Доказ. За $p = 2$ треба проверити да ли је $1! \equiv -1 \pmod{2}$. Како је $2 \equiv 0 \pmod{2}$, следи да тврђење теореме важи за $p = 2$.

Ако је p непаран прост број, у производу $(n-1)! = 1 \cdot 2 \cdot \ldots \cdot (p-1)$ има паран број чинилаца. Важи да је $1 \equiv 1 \pmod{p}$ и $p - 1 \equiv -1 \pmod{p}$. За сваки цео број n, $2 \leq n \leq p-2$, постоји јединствен цео број m, $2 \leq m \leq p-2$, такав да је $n \cdot m \equiv 1 \pmod{p}$ и важи $m \neq n$. Да би се показало да је ово тачно, треба приметити да бројеви $1 \cdot n,\ 2 \cdot n, \ldots,\ (p-1) \cdot n,\ p \cdot n$ образују сведени систем остатака по модулу p. Важи да је $n \equiv n \pmod{p}$, $(p-1) \cdot n \equiv (pn - n) \equiv p - n \neq 1 \pmod{p}$, $p \cdot n \equiv 0 \pmod{p}$, и само један од бројева $n \cdot m$, $m = 2, \ldots,\ p-2$ је еквивалентан 1 по модулу p. Треба показати да је $m \neq n$. Претпоставимо супротно, да је $n \cdot n \equiv 1 \pmod{p}$. Како конгруенција $n^2 \equiv 1 \pmod{p}$ има решења $n = 1 \pmod{p}$ и $n = -1 \pmod{p}$, и $2 \leq n \leq p-2$, следи да је претпоставка погрешна. Одавде следи тврђење теореме [25]. □

Вилсонова теорема се може једноставно доказати ако се зна да је за p прост број $\mathbb{Z}_p$ поље, да само нула нема инверз, да је за сваки елемент инверз јединствено одређен, као и да ред елемента дели ред групе, и да само за $n = 1$ и $n = p - 1$ важи да је $n^2 = 1$, па за прост број $p > 2$ тврђење директно следи. Такође, ако је n сложен број, онда $\mathbb{Z}_n$ није поље, и постоје делитељи нуле. Одавде следи да је $(n-1)! \equiv 0 \pmod{n}$, $n > 4$ сложен цео број. Треба приметити да за $n = 4$ важи $3! \equiv 1 \cdot 2 \cdot 3 \not\equiv 0 \pmod{4}$, јер је $2 \cdot 3 \equiv 2 \pmod{4}$, $2 \cdot 2 \equiv 0 \pmod{4}$.

[14] Вилсон (John Wilson), 1741-1793, енглески математичар.

Напомена 2.11. Важи и обрнуто тврђење : ако је $(p-1)! \equiv -1 \pmod{p}$, онда је p прост број. Ово следи на основу запажања да ако је $p > 4$ сложен број, онда сваки његов прави делитељ дели $(p-1)!$, па је $(p-1)! \equiv 0 \pmod{p}$.

Претходна два тврђења нам дају једноставан (и врло неефикасан) критеријум за налажење простих бројева.

Дефиниција 2.21. Позитиван цео број n је **безквадратан** или **квадратно слободан** (енгл. *square free*) ако n није дељив квадратом неког простог броја.

Дефиниција 2.22. Мебијусова функција (енгл. *Möbius function*) је дефинисана за све позитивне целе бројеве n. У зависности од факторизације броја n на просте чиниоце, ова функција може имати вредности из скупа $\{-1, 0, 1\}$. Нека је n позитиван цео број. Тада:

- $\mu(n) = 1$, ако је n безквадратан и има паран број различитих простих фактора.
- $\mu(n) = -1$, ако је n безквадратан и има непаран број различитих простих фактора.
- $\mu(n) = 0$, ако је n дељив квадратом неког броја већег од 1.

Другачије речено, нека је $n = p_1^{k_1} \cdot p_2^{k_2} \cdot ... \cdot p_m^{k_m}$ факторизација броја n на просте чиниоце. Онда је:

$$\mu(n) := \begin{cases} 0, & (\exists i \in [1,m]) \ : \ k_i > 1 \\ (-1)^m, & (\forall i \in [1,m]) \ : \ k_i = 1 \end{cases}$$

Специјално, узима се да је $\mu(1) = 1$.

Пример 2.10.

- Први бројеви за које је $\mu(n) = 0$ су: 4, 8, 9, 12, 16, 18, 20, 24, 25, 27, 28, 32, 36, 40, 44, 45, 48, 49, 50, 52, итд.
- Ако је n прост број, онда важи $\mu(n) = -1$. Обрнуто тврђење није тачно. Први број који није прост и за кога важи $\mu(n) = -1$ је $30 = 2 \cdot 3 \cdot 5$.
- Бројеви који имају три различита проста чиниоца: 30, 42, 66, 70, 78, 102, 105, итд. Бројеви који имају пет различитих простих чинилаца: 2310, 2730, 3570, 3990, 4290, 4830, 5610, итд.

Дефиниција 2.23. Функција $f : \mathbb{N} \to \mathbb{Z}$ је **мултипликативна** ако важи:

1. Постоји позитиван цео број n_0 такав да је $f(n_0) \neq 0$,
2. Ако је $\text{nzd}(m, n) = 1$, следи да важи $f(mn) = f(m) \cdot f(n)$.

Пример 2.11.

- Ојлерова функција је мултипликативна функција.
- Мебијусова функција је мултипликативна.
- τ функција (укупан број позитивних делитеља неког броја) је мултипликативна.

Једна од основних функција у теорији бројева је и функција $\pi(n)$.

Дефиниција 2.24. Функција $\pi(n)$ се дефинише као број простих бројева мањих или једнаких броју n.

Скуп свих простих бројева се означава са $\mathbb{P}$ или са primes.

2.3 Линеарне конгруенције. Диофантове једначине. Квадратне конгруенције

Доказ наредне теореме се изводи математичком индукцијом. Теорема не важи у случају када је модуо p сложен број.

Теорема 2.13 (Лагранжова теорема). *Нека је $f(x) = a_n \cdot x^n + ... + a_1 \cdot x + a_0$ полином степена n са целобројним коефицијентима. Нека је p прост број и нека важи да p не дели a_n. Тада конгруенција $f(x) \equiv 0 \pmod{p}$ има највише n решења по модулу p.*

Сада су дата нека од основних тврђења у вези са конгруенцијама.

Тврђење 2.5. *Нека је $d = \text{nzd}(a, n)$. Онда важи: $ax \equiv ay \pmod{n}$ ако и само ако је $x \equiv y \pmod{\frac{n}{d}}$.*

Доказ. $\boxed{\Rightarrow}$ $ax \equiv ay \pmod{n}$ је еквивалентно са $a(x-y) \equiv b \cdot n$, за неки цео број b. Одатле се добија да $\frac{a}{d}(x-y) \equiv b \cdot \frac{n}{d}$. Како је $\text{nzd}(\frac{a}{d}, \frac{n}{d}) = 1$, следи да $\frac{n}{d} | (x-y)$, тј. $x \equiv y \pmod{\frac{n}{d}}$.
$\boxed{\Leftarrow}$ Ако важи $x \equiv y \pmod{\frac{n}{d}}$, онда $\frac{n}{d} | (x-y)$, па за неки цео број c слеи да је $n \cdot c = d(x-y)$. Множењем и леве и десне стране са $\frac{a}{d}$, добија се да је $n \cdot c \cdot \frac{a}{d} = a(x-y)$. Следи да $n | a(x-y)$, тј. $ax \equiv ay \pmod{m}$. □

Тврђење 2.6. *Нека је $\text{nzd}(a, n) = 1$. Онда важи: $ax \equiv ay \pmod{n}$ ако и само ако је $x \equiv y \pmod{n}$.*

Доказ. $\boxed{\Rightarrow}$ $ax \equiv ay \pmod{n}$ је еквивалентно са $a(x-y) \equiv b \cdot n$, за неки цео број b. Како су a и n узајамно прости бројеви, следи да $n | (x-y)$ тј. $x \equiv y \pmod{n}$.
$\boxed{\Leftarrow}$ Ако важи $x \equiv y \pmod{n}$, онда $n | (x-y)$, па за неки цео број c следи да је $n \cdot c = (x-y)$. Множењем и леве и десне стране са a, добија се $n \cdot c \cdot a = a(x-y)$. Одатле следи да $n | a(x-y)$, тј. $ax \equiv ay \pmod{n}$. □

Тврђење 2.7. *Ако је $\text{nzd}(a, n) = 1$, конгруенција $ax \equiv b \pmod{n}$ има јединствено решење по модулу n.*

Доказ. Једно очигледно решење горње конгруенције је $x = a^{\phi(n)-1} \cdot b$, јер из Ојлерове теореме и услова тврђења следи да $a \cdot a^{\phi(n)-1} \cdot b \equiv a^{\phi(n)} \cdot b \equiv b \pmod{n}$.

Претпоставимо супротно, да постоји још једно решење, означимо га са y. Тада важи да је $ax \equiv ay \pmod{n}$, па на основу претходног тврђења следи да је $x \equiv y \pmod{n}$. □

Свака **линеарна конгруенција** може се представити у облику: $ax \equiv b \pmod{n}$, односно $ax - b = c \cdot n$, за неки цео број c. Видели смо да ако је $\text{nzd}(a, n) = 1$, ова конгруенција има **јединствено решење**. Претпоставимо сада да је $d = \text{nzd}(a, n)$. Ако d **не дели** b онда конгруенција **нема решења**. Ако d **дели** b, онда $\frac{a}{d}x - \frac{b}{d} \equiv c\frac{n}{d}$ и $\text{nzd}(\frac{a}{d}, \frac{n}{d}) = 1$, па конгруенција $\frac{a}{d}x \equiv \frac{b}{d} \pmod{\frac{n}{d}}$ има јединствено решење x_0. Одавде следи да је потпун скуп решења по модулу n конгруенције $ax \equiv b \pmod{n}$ дат са $x = x_0 + \frac{n}{d}m$, $m = 0, 1, 2, ..., d-1$, односно конгруенција $ax \equiv b \pmod{n}$ има **тачно** d **решења по модулу** n.

Напомена 2.12. Овде под јединственим решењем подразумевамо **јединствено решење у потпуном систему остатака по датом модулу**, на пример по модулу n у скупу $\{0, 1, ..., n-1\}$. Наравно, ако је x решење дате конгруенције, тада је и сваки број из његове класе конгруенције по модулу n такође решење исте конгруенције. Ово важи и у наредним разматрањима.

Више линеарних конгруенција не мора да има заједничко решење, иако свака конгруенција појединачно има решење. Следећа теорема даје услове под којима систем линеарних конгруенција има решење.

Теорема 2.14 (Кинеска теорема о остацима). *Нека су $n_1, ..., n_k$ позитивни цели бројеви, има их $k > 0$, и претпоставимо да су они узајамно прости у паровима, $\text{nzd}(n_i, n_j) = 1$ за $i \neq j$. Тада за сваки цео број $c_1, ..., c_k$ конгруенције $x \equiv c_i \pmod{n_i}$, $0 \leq i \leq k$, су истовремено решиве за неки цео број x, тј. постоји јединствено решење по модулу $n = n_1 \cdots n_k$.*

Доказ. Нека је $m_i = \frac{n}{n_i}$, $1 \leq i \leq k$. Тада је $\text{nzd}(n_i, m_i) = 1$ и зато постоји цео број x_i такав да је $m_i \cdot x_i \equiv c_i \pmod{n_i}$. Сада следи да $x = m_1 \cdot x_1 + ... + m_k \cdot x_k$ задовољава $x \equiv c_i \pmod{n_i}$, као што је требало доказати. Јединственост следи из чињенице да ако би постојала два решења x, y, онда би било $x \equiv y \pmod{n_i}$, $1 \leq i \leq k$, а како су n_i узајамно прости у паровима, онда је $x \equiv y \pmod{n}$ [3]. □

Кинеска теорема о остацима (енгл. *Chinese remainder theorem, CRT*), заједно са условима за решавање произвољне линеарне конгруенције, каже да ако су $n_1, ..., n_k$ узајамно прости бројеви онда су конгруенције $a_i \cdot x \equiv b_i \pmod{n_i}$, $1 \leq i \leq k$, истовремено решиве ако и само ако $d_i = \text{nzd}(a_i, n_i)$ дели b_i за све i.

Пример 2.12. Нека је дата конгруенција $6x \equiv 15 \pmod{21}$. Следи да је $d = \text{nzd}(6, 21) = 3$ и $d = 3$ дели 15, тако да има решења. Сада се разматра конгруенција $2x \equiv 5 \pmod{7}$, $\text{nzd}(2, 7) = 1$ која има решење $x_0 = 6$ по модулу 7. Одавде следи да полазна једначина има решења $x = 6 + \frac{21}{3}m = 6 + 7m$, $m = 0, 1, 2$, односно постоје три решења полазне конгруенције у потпуном систему остатака по модулу 21 — то су $x_1 = 6$, $x_2 = 13$, $x_3 = 20$. Наравно, решења су и сви бројеви који су конгруентни датим решењима по модулу 21, на пример бројеви $27 \equiv 6 \pmod{21}$, $34 \equiv 13 \pmod{21}$, $41 \equiv 20 \pmod{21}$ су такође решења конгруенције $6x \equiv 15 \pmod{21}$.

Диофантове једначине[15] (енгл. *Diophantine equations*) су једначине са целобројним коефицијентима у којима су дозвољена само целобројна решења. Матијашевич[16] је 1970. године доказао да не постоји опште решење тј. алгоритам за одређивање да ли произвољна Диофантова једначина има решење. Овакав алгоритам постоји за решавање Диофантових једначина првог реда.

Дефиниција 2.25. Линеарне Диофантове једначине су једначине облика $ax + by = c$, где важи да $a, b, c \in \mathbb{Z}$.

Свака линеарна једначина са две променљиве и целобројним коефицијентима може се свести на Диофантову једначину $ax + by = c$. С друге стране, решавање линеарне Диофантове једначине $ax + by = c$ своди се на решавање линеарне конгруенције $ax \equiv c \pmod{b}$.

Може се дати следећа теорема:

Теорема 2.15. *Линеарна Диофантова једначина $ax + by = c$, $d = \text{nzd}(a, b)$, има решење ако и само ако d дели c. У овом случају опште решење једначине је облика $x = \frac{c}{d}x_0 + \frac{b}{d}m$, $y = \frac{c}{d}y_0 - \frac{a}{d}m$, $m \in \mathbb{Z}$, где се решење (x_0, y_0) једначине $ax_0 + by_0 = d$ добија проширеним Еуклидовим алгоритмом.*[17] *Специјално, ако је $\text{nzd}(a, b) = 1$, онда једначина сигурно има решење јер $d = 1$ дели сваки број c.*

Квадратне конгруенције

Може се показати да се **решавање опште квадратне конгруенције** (енгл. *quadratic congruence*), $ax^2 + bx + c \equiv 0 \pmod{p}$, $\text{nzd}(a, p) = 1$ и p прост број, своди на решавање квадратне конгруенције $x^2 \equiv a \pmod{p}$, $\text{nzd}(a, p) = 1$, p прост број, где је случај $p = 2$ тривијалан. Ако је $p > 2$ прост број, и ова конгруенција има решење x_1, онда је и $-x_1$ решење исте конгруенције, и бројеви x_1 и $-x_1$ нису конгруентни по модулу p, јер би у супротном из $x_1 \equiv -x_1 \pmod{p}$ следило да $2x_1 \equiv 0 \pmod{p}$, што је немогуће јер је $\text{nzd}(2, p) = \text{nzd}(a, p) = 1$.

Ако је $\text{nzd}(a, p) = 1$ и p непарно, на основу Мале Фермаове теореме важи да је $0 \equiv a^{p-1} - 1 \equiv (a^{\frac{p-1}{2}} - 1)(a^{\frac{p-1}{2}} + 1) \pmod{p}$. Одатле следи да је или $a^{\frac{p-1}{2}} \equiv 1 \pmod{p}$ или $a^{\frac{p-1}{2}} \equiv -1 \pmod{p}$.

С друге стране, ако конгруенција $x^2 \equiv a \pmod{p}$, $\text{nzd}(a, p) = 1$ има решење, онда је $x^{p-1} \equiv a^{\frac{p-1}{2}} \pmod{p}$. Како из Мале Фермаове теореме следи да је $x^{p-1} \equiv 1 \pmod{p}$, мора бити $a^{\frac{p-1}{2}} \equiv 1 \pmod{p}$ [4].

На основу претходних разматрања важи тврђење:

Теорема 2.16 (Ојлеров критеријум). *Ако је $p > 2$ прост број и $\text{nzd}(a, p) = 1$, тада једначина $x^2 \equiv a \pmod{p}$ има два решења по модулу p ако је $a^{\frac{p-1}{2}} \equiv 1 \pmod{p}$, односно нема решење ако је $a^{\frac{p-1}{2}} \equiv -1 \pmod{p}$.*

[15] Диофант (Diophantus), 3. век н.е., старогрчки математичар, понекад се назива и *оцем алгебре*. Најпознатији по свом делу *Arithmetica*.

[16] Матијашевич (Yuri Matiyasevich), 1947. - , најпознатији по негативном решењу десетог Хилбертовог проблема.

[17] Проширен Еуклидов алгоритам је детаљно описан у наредном поглављу.

Теорема 2.17. *Нека је p прост број. У сведеном систему остатака $\{0, 1, ..., p-1\}$ по модулу p има тачно $\frac{p-1}{2}$ бројева који су конгруентни квадратима целих бројева по модулу p.*

Доказ. Квадратима целих бројева конгруентни су квадрати бројева $-\frac{p-1}{2}, ..., -2, -1, 1, 2, ..., \frac{p-1}{2}$, тј. бројеви $1^2, 2^2, ..., (\frac{p-1}{2})^2$. Међу овим бројевима не може бити конгруентних, јер би $k^2 \equiv 1^2 \pmod{p}$, $0 < k < 1 \leq \frac{p-1}{2}$, имало за последицу да једначина $x^2 \equiv 1 \pmod{p}$ има четири решења у скупу $-\frac{p-1}{2}, ..., -2, -1, 1, 2, ..., \frac{p-1}{2}$, што је доказано да је немогуће за просте бројеве [25]. □

Дефиниција 2.26. Нека је n позитиван цео број. Ако је $\text{nzd}(a, n) = 1$ и конгруенција $x^2 \equiv a \pmod{n}$ има решење, онда се каже да је a **квадратни остатак по модулу** n (енгл. *quadratic residue modulo n*). Ако је $\text{nzd}(a, n) = 1$ и конгруенција $x^2 \equiv a \pmod{n}$ нема решење, онда се каже да је a **квадратни неостатак по модулу** n (енгл. *quadratic nonresidue modulo n*).

Број n у претходној дефиницији не мора да буде прост, већ је битан услов да је $\text{nzd}(a, n) = 1$. Зато за сложен број n постоје бројеви који нису ни квадратни остаци ни квадратни неостаци, а то су бројеви за које је $\text{nzd}(a, n) \neq 1$. Такође, за прост број n, број $0 \pmod{n}$ је увек квадрат али није ни квадратни остатак ни квадратни неостатак.

Проблем налажења квадратног корена по модулу n је важан за криптографију. Он гласи: за дати број n, број a такав да је $\text{nzd}(a, n) = 1$, и a који је квадратни остатак по модулу n, наћи квадратни корен од a по модулу n, тј. решити једначину $x^2 \equiv a \pmod{n}$.

Више о основним појмовима теорије бројева може се наћи у [3, 4, 25].

2.4 Лежандров симбол. Јакобијев симбол

Ово поглавље, са малим изменама, представља материјал изложен у [4].

Дефиниција 2.27. Лежандров симбол $(\frac{a}{p})$ за $\text{nzd}(a, p) = 1$ и *p непаран прост број* дефинише се на следећи начин: $(\frac{a}{p}) = 1$ ако конгруенција $x^2 \equiv a \pmod{p}$ има решења, односно $(\frac{a}{p}) = -1$ ако $x^2 \equiv a \pmod{p}$ нема решење. За $\text{nzd}(a, p) > 1$ се дефинише да је $(\frac{a}{p}) = 0$.

Дефиниција 2.28. Јакобијев симбол (енгл. *Jacobi symbol*) је уопштење Лежандровог симбола. Нека је *n позитиван непаран број*. Онда је $n = p_1 \cdot p_2 \cdot ... \cdot p_k$ производ простих бројева, не неопходно различитих. Тада за сваки цео број a за који важи да је $\text{nzd}(a, n) = 1$, Јакобијев симбол је дефинисан са:

$$(\tfrac{a}{n}) = (\tfrac{a}{p_1}) \cdot ... \cdot (\tfrac{a}{p_k}),$$

где су са десне стране Лежандрови симболи. За $n = 1$ Јакобијев симбол је дефинисан као 1, а када је $\text{nzd}(a, n) > 1$ дефинисан је као 0.

Теорема 2.18 (Особине Лежандровог симбола)**.** *Нека је p непаран прост број и* $\text{nzd}(a, p) = 1$, $\text{nzd}(b, p) = 1$. *Тада је:*

1. $(\frac{a}{p}) \equiv a^{\frac{p-1}{2}} \pmod{p}$. *Ово је **Ојлеров критеријум**.*
2. $(\frac{a}{p})(\frac{b}{p}) \equiv (\frac{ab}{p})$.
3. $a \equiv b \pmod{p} \Rightarrow (\frac{a}{p}) \equiv (\frac{b}{p})$.
4. $(\frac{a^2}{p}) \equiv 1$, $(\frac{1}{p}) \equiv 1$, $(\frac{-1}{p}) \equiv (-1)^{\frac{p-1}{2}}$.

На основу ових особина за Лежандров симбол могу се извести аналогна тврђења за Јакобијев симбол. Исто важи и за наредне две теореме.

Теорема 2.19. *Ако је p непаран прост број, онда је* $(\frac{2}{p}) = (-1)^{(p^2-1)/8}$.

Доказ. Нека је $\text{nzd}(a, p) = 1$, и $p' = \frac{p-1}{2}$ (из услова теореме следи да је p непаран прост број). Посматрајмо конгруенције:

$$\begin{aligned} a \cdot 1 &\equiv \epsilon_1 r_1 \pmod{p} \\ a \cdot 2 &\equiv \epsilon_2 r_2 \pmod{p} \\ &\dots \\ a \cdot p' &\equiv \epsilon_{p'} r_{p'} \pmod{p} \end{aligned}$$

где је $\epsilon_x r_x$ најмањи остатак по апсолутној вредности броја $a \cdot x \pmod{p}$, и $r_x = |\epsilon_x \cdot r_x|$, а $\epsilon_x = 1$ или $\epsilon_x = -1$.
Како важи $\text{nzd}(a, p) = 1$, и како бројева $a \cdot 1,\ -a \cdot 1,\ a \cdot 2,\ -a \cdot 2,\ ...,\ a \cdot p',\ -a \cdot p'$ има укупно $p-1$, они образују сведени систем остатака по модулу p. Њихови најмањи остаци по апсолутној вредности су бројеви $\epsilon_1 r_1,\ -\epsilon_1 r_1,\ \epsilon_2 r_2,\ -\epsilon_2 r_2, ...,\ \epsilon_{p'} r_{p'},\ -\epsilon_{p'} r_{p'}$. Позитивни од ових бројева $(r_1, r_2, ..., r_{p'})$ се поклапају са бројевима $1, 2, ..., p'$. Зато из горњих конгруенција помоћу множења и скраћивањем са $1 \cdot 2 \cdot ... \cdot p' = r_1 r_2 \cdot ... \cdot r_{p'}$ добија се да је $a^{\frac{p-1}{2}} \equiv \epsilon_1 \epsilon_2 \cdot ... \cdot \epsilon_{p'} \pmod{p}$.

Одавде, на основу особина Лежандровог симбола, следи да важи $(\frac{a}{p}) = \epsilon_1 \epsilon_2 \cdot ... \cdot \epsilon_{p'}$. Користећи чињеницу да се сваки реалан број x може записати у облику $x = [x] + \{x\}$, где је $[x]$ цели део, а $\{x\}$ разломљени део од x, из релације

$$[\tfrac{2ax}{p}] = [2[\tfrac{ax}{p}] + 2\left\{\tfrac{ax}{p}\right\}] = 2[\tfrac{ax}{p}] + [2\left\{\tfrac{ax}{p}\right\}]$$

следи да је $[\frac{2ax}{p}]$ парно или непарно, у зависности да ли је најмањи ненегативни остатак ax по модулу p мањи или већи од $\frac{p}{2}$, тј. да ли је $\epsilon_x = 1$ или $\epsilon_x = -1$. Зато је $\epsilon_x = (-1)^{[\frac{2ax}{p}]}$, и важи:

$$(\tfrac{a}{p}) = \epsilon_1 \epsilon_2 \cdot ... \cdot \epsilon_{p'} = (-1)^{[\frac{2a\cdot 1}{p}]}(-1)^{[\frac{2a\cdot 2}{p}]} \cdot ... \cdot (-1)^{[\frac{2a\cdot p'}{p}]} = (-1)^S, \text{ где је } S = \sum_{x=1}^{p'} [\tfrac{2ax}{p}].$$

Ако је a непарно, онда је $a + p$ парно (јер је и p непаран број), и између осталог, користећи чињеницу да је $(\frac{2}{p})^2 \equiv 1$, добија се:

$$(\tfrac{2a}{p}) = (\tfrac{2a+2p}{p}) = (\tfrac{4\cdot\frac{(a+p)}{2}}{p}) = (\tfrac{\frac{(a+p)}{2}}{p}) = (-1)^{\sum_{x=1}^{p'}[\frac{(a+p)x}{p}]} = (-1)^{\sum_{x=1}^{p'}[\frac{ax}{p}]+\sum_{x=1}^{p'} x} = (-1)^{\sum_{x=1}^{p'}[\frac{ax}{p}]+\frac{p^2-1}{8}}$$

Ако се стави да је $a = 1$, онда је $\sum_{x=1}^{p'}[\frac{1\cdot x}{p}] = 0$, одакле следи почетно тврђење. $\square$

Одавде следи да је $(\frac{2}{p}) \equiv 1$ ако је $p \equiv 1$ или 7 по модулу 8, тј. да је $(\frac{2}{p}) \equiv -1$ ако је $p \equiv 3$ или 5 по модулу 8. Следећа теорема је један од основних резултата теорије квадратних остатака.

Теорема 2.20 (Гаусов закон квадратног реципроцитета). *Ако су p, q различити непарни прости бројеви, онда је $(\frac{p}{q}) \cdot (\frac{q}{p}) = (-1)^{\frac{p-1}{2}\cdot\frac{q-1}{2}}$.*

Другим речима, $(\frac{p}{q}) \equiv (\frac{q}{p})$, осим ако су p и q оба конгруентни 3 по модулу 4, и у том случају је $(\frac{p}{q}) \equiv -(\frac{q}{p})$.

Доказ. Полази се од тврђења да важи $(\frac{2}{p}) = (-1)^{(p^2-1)/8}$. Број p је облика $p = 8m + s$, где је $m \geq 0$ а s је један од бројева $1, 3, 5, 7$ (p је непаран број). Из овог тврђења такође следи да је $(\frac{2}{p}) = 1$ (2 квадратни остатак по модулу p) ако је $p = 8m + 1$ и $p = 8m + 7$, а да је $(\frac{2}{p}) = -1$ (2 квадратни неостатак) ако је $p = 8m + 3$ и $p = 8m + 5$.

Посматрајмо сада бројеве $q \cdot x$, $p \cdot y$, где је $1 \leq x \leq p'$, $1 \leq y \leq q'$, $p' = \frac{p-1}{2}$, $q' = \frac{q-1}{2}$, где су p и q различити непарни прости бројеви. Укупно је $p' \cdot q'$ таквих бројева, а како је $\text{nzd}(p, q) = 1$, једнакост $p \cdot y = q \cdot x$ није могућа. Зато је $p' \cdot q' = S_1 + S_2$, где је S_1 број парова (x, y) за које важи $q \cdot x \leq p \cdot y$, а S_2 је број парова (x, y) за које је $p \cdot y < q \cdot x$.

Ако се са $[x]$ означи цео део од x, онда је услов $q \cdot x < p \cdot y$ еквивалентан са $q \cdot x \leq p \cdot y$ (тј. $x \leq \left[\frac{p\cdot y}{q}\right]$) па важи: $S_1 = \sum_{y=1}^{q'}[\frac{p\cdot y}{q}]$, $S_2 = \sum_{x=1}^{p'}[\frac{q\cdot x}{p}]$.

Из претходног доказа смо видели да важи $(\frac{2a}{p}) = (-1)^{\sum_{x=1}^{p'}[\frac{ax}{p}]+\frac{p^2-1}{8}}$, а како је $(\frac{2a}{p}) = (\frac{2}{p})(\frac{a}{p})$ и $(\frac{2}{p}) = (-1)^{(p^2-1)/8}$, следи да је $(\frac{p}{q}) = (-1)^{S_1}$, $(\frac{q}{p}) = (-1)^{S_2}$, и коначно $(\frac{p}{q}) \cdot (\frac{q}{p}) = (-1)^{S_1+S_2} = (-1)^{p'\cdot q'} = (-1)^{\frac{p-1}{2}\cdot\frac{q-1}{2}}$. $\square$

Закон квадратног реципроцитета омогућава решавање различитих проблема у вези са квадратним остацима. Може се формулисати и на следећи начин:

Нека су p и q различити прости бројеви. Ако су и p и q облика $4k+3$, једна од једначина $x^2 \equiv p \pmod{q}$, $x^2 \equiv q \pmod{p}$ је решива, а друга није. Ако p и q оба нису облика $4k+3$, тада су или обе једначине $x^2 \equiv p \pmod{q}$, $x^2 \equiv q \pmod{p}$ решиве или ниједна [25].

Напомена 2.13. Постоји велики број доказа ове теореме. У [27] се може наћи доказ заснован на Гаусовим сумама (чија дефиниција је дата у поглављу 4).

2.5 Проблем квадратних остатака

Скуп свих квадратних остатака по модулу n означавамо са Q_n, а **скуп свих квадратних неостатака по модулу** n са $\overline{Q}_n$.

Пример 2.13.

- Да би се показало да неки велики цео број a није квадрат неког броја довољно је наћи мали прости модуо p за који је тај број квадратни неостатак, што се може урадити рачунајући Лежандров симбол $(\frac{a}{p})$.

- Треба наћи све целе бројеве a који задовољавају једначину $x^2 \equiv a \pmod{21}$, x припада потпуном систему остатака по модулу 21. Једноставним израчунавањем добија се да је $1^2 = 1 \pmod{21}$, $2^2 = 4 \pmod{21}$, $3^2 = 9 \pmod{21}$, $4^2 = 16 \pmod{21}$, $5^2 = 4 \pmod{21}$, $6^2 = 15 \pmod{21}$, $7^2 = 6 \pmod{21}$, $8^2 = 1 \pmod{21}$, $9^2 = 18 \pmod{21}$, $10^2 = 16 \pmod{21}$, $11^2 = 16 \pmod{21}$,..., $20^2 = 1 \pmod{21}$. Види се да бројеви a који задовољавају једначину $x^2 \equiv a \pmod{21}$ за неки цео број x су $\{1, 4, 7, 9, 15, 16, 18\}$. Како су квадратни остаци и квадратни неостаци дефинисани само за бројеве a такве да је $\text{nzd}(a, 21) = 1$, уводи се услов да x припада сведеном систему остатака, и добија за a скуп $\{1, 4, 16\}$. Одатле следи да су квадратни остаци само бројеви $\{1, 4, 16\}$.

  С друге стране, рачунајући Јакобијеве симболе у сведеном систему остатака по модулу 21, добија се да је $(\frac{1}{21}) = (\frac{1}{3})(\frac{1}{7}) = 1 \cdot 1 = 1$, $(\frac{2}{21}) = (\frac{2}{3})(\frac{2}{7}) = -1 \cdot 1 = -1$, $(\frac{4}{21}) = (\frac{2^2}{3})(\frac{2^2}{7}) = 1 \cdot 1 = 1$, $(\frac{5}{21}) = (\frac{5}{3})(\frac{5}{7}) = (\frac{2}{3})(\frac{2}{5}) = (-1)(-1) = 1$, $(\frac{8}{21}) = (\frac{2^3}{3})(\frac{2^3}{7}) = (-1)^3(1)^3 = -1$, итд. Треба приметити да је Јакобијев симбол једнак 1 за бројеве $\{1, 4, 5, 16, 17, 20\}$.

Дефиниција 2.29. Нека је $n \geq 3$ *непаран* цео број, не нужно прост, и нека је $J_n = \left\{a \in \mathbb{Z}_n^* | (\frac{a}{n}) \equiv 1\right\}$. Скуп **псеудоквадрата по модулу** n се дефинише као $\widetilde{Q_n} = J_n - Q_n$.

Лако се показује да су половина елемената у J_n квадратни остаци, а половина елемената су псеудоквадрати.

Дефиниција 2.30. Проблем квадратних остатака (енгл. *quadratic residuosity problem, QRP*) се дефинише на следећи начин: за дати непаран сложени цео број n и $a \in J_n$, тј. Јакобијев симбол $(\frac{a}{n}) = 1$, одредити да ли је или не a квадратни остатак по модулу n.

Не постоји детерминистички полиномијални алгоритам за проналажење квадратног неостатка по модулу n, иако је половина елемената из $\mathbb{Z}_n^*$ квадратни неостаци.

Напомена 2.14. Ако је n прост број, лако је одредити да ли је $a \in Z_n^*$ квадратни остатак модуо n пошто је, по дефиницији, $a \in Q_n$ ако и само ако $(\frac{a}{n}) = 1$. Лежандров симбол $(\frac{a}{n})$ може се ефикасно израчунати помоћу алгоритма 3.6 датог у поглављу 3.

Претпоставимо сада да је n производ два различита непарна проста броја p и q, $n = p \cdot q$ и да $a \in J_n$. Онда $a \in Q_n$ ако и само ако је $(\frac{a}{p}) = 1$. Зато, ако је позната факторизација броја n, *QRP* може бити решен једноставним израчунавањем Лежандрових симбола за све просте факторе од n.

Ова чињеница се може генерализовати на све целе бројеве n:

Тврђење 2.8. *Проблем квадратних остатака се у полиномијалном времену своди на проблем факторизације.*

Са друге стране, ако није позната факторизација броја n, не постоји ефикасан поступак за решавање *QRP*. Ако је $n = q \cdot p$, тада је вероватноћа тачног одговора 1/2.

2.6 Специјални облици простих бројева

Специјалне класе простих бројева, као што су Мерсенови бројеви, значајне су јер постоје ефикасни алгоритми за доказивање њихове прималности. Дате су неке битне особине ових бројева.

Фермаови бројеви

Дефиниција 2.31. Фермаови бројеви су бројеви облика $F_n = 2^{2^n} + 1$. **Фермаови прости бројеви** су *прости* бројеви облика $F_n = 2^{2^n} + 1$.

Ферма је претпоставио да су бројеви облика $F_n = 2^{2^n} + 1$, $(n = 1, 2, ...)$ прости. Ово је тачно за $n = 1, 2, 3, 4$, јер је $F_0 = 3$, $F_1 = 5$, $F_2 = 17$, $F_3 = 257$ и $F_4 = 65537$, али је нетачно већ за $n = 5$, као што је Ојлер показао. Он је, наиме, уочио да 641 дели $2^{32} + 1 = 4,294,967,297$. Није познато да ли постоји коначно или бесконачно много Фермаових простих бројева.

Напомена 2.15. Бројеви облика $2^n + 1$, $n > 0$ непаран цео број, су дељиви са 3. Ово се лако показује. Остатак при дељењу броја 2^n са 3 је 1 или 2, при чему је $2^{2\cdot k+1} = 2 \pmod 3$, $2^{2\cdot k} = 1 \pmod 3$, односно $2^{2\cdot k+1} + 1 = 0 \pmod 3$, тј. када је n непаран, број $2^{2\cdot k+1} + 1$ је дељив са 3, па самим тим и сложен, док за паран број n важи $2^{2\cdot k} + 1 = 2 \pmod 3$.

Из претходне напомене следи да ако је $2^n + 1$ прост број, n мора бити паран. Важи и јаче тврђење:

Теорема 2.21. *Ако је $p = 2^n + 1$ непаран прост број, онда је n степен броја 2.*

Доказ. Претпоставимо супротно, да је $n = 2^k \cdot m$, $m > 0$ непаран цео број. Тада је $2^n+1 = (2^{2^k})^m+1$, па како важи да је $(x+y)|(x^m+y^m)$ за $m > 0$ непаран цео број, следи да $(2^{2^k}+1)|(2^n+1)$, односно да је $2^n + 1$ сложен број, што је контрадикција са претпоставком. Одавде следи тражено тврђење [25]. □

Ова теорема заправо говори да је сваки прост број који је за 1 већи од неког степена броја 2 у ствари Фермаов број.

Лако се математичком индукцијом показује да се сви Фермаови бројеви $2^{2^n} + 1$, $n > 1$ цео број, завршавају цифром 7, односно да се бројеви 2^{2^n} завршавају цифром 6.

Следећа теорема смањује број простих фактора које треба проверити при тестирању F_n:

Теорема 2.22. *За $n \geq 2$, сваки прост фактор r од F_n мора задовољавати $r \equiv 1 \pmod{2^{n+2}}$. Другим речима, сваки прости фактор r Фермаовог броја F_n је облика $k \cdot 2^{n+2} + 1$.*

Доказ. Нека је r прост фактор од F_n и нека је h најмањи позитивни цео број такав да је $2^h \equiv 1 \pmod r$. Како је $2^{2^n} \equiv -1 \pmod r$, онда је $h = 2^{n+1}$. Такође, у мултипликативној групи $\mathbb{Z}_r$ број 2 има ред 2^{n+1}, а како ред елемента дели ред групе, следи да $2^{n+1}|(r-1)$, односно $r \equiv 1 \pmod{2^{n+1}}$. Пошто је $n \geq 2$, добија се да је $r \equiv 1 \pmod 8$. Сада је $(\frac{2}{r}) = (-1)^{(r^2-1)/8} = 1$, одакле следи да је 2 квадратни остатак по модулу r, односно да је $2^{\frac{r-1}{2}} \equiv 1 \pmod r$. Како је h најмањи позитивни цео број такав да је $2^h \equiv 1 \pmod r$, види се да $h = 2^{n+1}$ дели $\frac{r-1}{2}$, одакле следи тврђење теореме. □

Ефикасна метода за проверу простости броја F_n је Пепинов тест, који је детаљно описан у поглављу 4. Занимљиво је поменути да је до 2008. године само првих 12 Фермаових бројева *комплетно* факторисано.

Напомена 2.16. Једна од значајних примена Фермаових бројева је у генерисању псеудослучајних низова бројева у интервалу од 1 до N, где је N степен броја 2.

Мерсенови бројеви

Дефиниција 2.32. Нека је p, такав да важи $p \geq 2$, цео број. **Мерсенов**[18] **број** је цео број облика $2^p - 1$. **Мерсенови прости бројеви** су *прости* бројеви облика $2^p - 1$.

Теорема 2.23. *Ако је $M_p = 2^p - 1$ прост број, онда је p прост број.*

[18] Мерсен (Marin Mersenne), 1588–1648, француски теолог, математичар, филозоф, музички теоретичар.

Доказ. Претпоставимо супротно, да је $2^n - 1$ прост број и $n = q_1 \cdot q_2$ сложен број. Али тада би број $2^n - 1 = (2^{q_1})^{q_2} - 1$ био дељив са $2^{q_1} - 1$, одакле следи тражено тврђење.[19] □

Треба приметити да обрат претходног тврђења не мора да важи.

Теорема 2.24. *Ако је* $\forall p > 2$*, онда за сваки прост фактор* q *од* M_p *важи:* $q \equiv 1 \pmod{p}$ *и* $q \equiv \pm 1 \pmod{8}$.

Доказ. Као што је показано, ако је $2^p - 1$ прост број, онда је и $p > 2$ прост број. Ако је q прост фактор броја M_p, онда је $2^p \equiv 1 \pmod{q}$, и како је p прост број, онда је баш p једнак најмањем позитивном експоненту k таквом да је $2^k \equiv 1 \pmod{q}$. Зато у мултипликативној групи $\mathbb{Z}_q$ број 2 има ред p. Како ред елемента дели ред групе, следи да је $q \equiv 1 \pmod{p}$. Пошто је p непаран прост број, следи да $p | \frac{q-1}{2}$, односно $2^{\frac{q-1}{2}} \equiv 1 \pmod{q}$. Сада из Ојлеровог критеријума следи да је 2 квадратни остатак по модулу q, одакле на основу $(\frac{2}{p}) = (-1)^{(p^2-1)/8}$ важи да је $q \equiv \pm 1 \pmod{8}$. □

Ове две теореме омогућују да се много ефикасније налазе Мерсенови прости бројеви него просте бројеве уопште. Метод за тестирање Мерсенових бројева може бити следећи: Почиње се са тестирањем Мерсенових бројева M_p само за просте вредности p. Онда се проверава да ли је $M_p \equiv 1 \pmod{q}$ за неколико малих простих бројева q који задовољавају $q \equiv 1 \pmod{p}$ и $q \equiv \pm 1 \pmod{8}$. Бројеви који прођу овај тест се онда подвргавају Лукас-Лемеровом тесту (енгл. *Lucas-Lehmer test*), који је тест за доказивање прималности за Мерсенове бројеве јер захтева познавање факторизације броја $n + 1$ да би се одредило да ли је n прост. Како је $M_p + 1 = 2^p$, овај тест се лако промењује на Мерсенове просте бројеве [1].

Један од нерешених математичких проблема је да ли има бесконачно много Мерсенових простих бројева. До 2012. год. је откривено 46 Мерсенових простих бројева. Такође је занимљиво да није познато да ли постоји неки *неоткривени* Мерсенов прост број између 41.ог ($M_{24,036,583}$) и 46.ог ($M_{43,112,609}$) Мерсеновог простог броја. У децембру 2011. је *доказано* да је 41. Мерсенов прост број $M_{24,036,583}$.[20]

Тврђење 2.9. *Бројеви* $2^n - 1$ *и* $2^n + 1$ *не могу истовремено бити прости за цео број* $n > 2$.

Доказ. Ово следи из тврђења које смо претходно навели. Наиме, ако је Мерсенов број $m = 2^n - 1$ прост, онда је n прост број. С друге стране, ако је $2^n + 1$ прост број, онда је $n = 2^k$, односно $f = 2^n + 1 = 2^{2^k} + 1$ је Фермаов број. Ова два захтева за n су у супротности један са другим за $n > 2$. За $n = 1$ следи да је $m = 1$, $f = 3$, и треба приметити да 1 није прост број, док за $n = 2$ важи да је $m = 3$, $f = 5$, па су $2^n - 1$ и $2^n + 1$ истовремено прости једино за $n = 2$.

Други облици простих бројева

Прот[21] је 1878. године објавио следећу теорему:

Теорема 2.25 (Протова теорема). *Нека је* $n = h \cdot 2^k + 1$*, где је* $2^k > h$*,* h *непаран цео број. Ако постоји цео број* a *такав да је* $a^{\frac{n-1}{2}} = -1 \pmod{n}$*, онда је* n *прост број.*

Другим речима, ако је $n > 1$, $2^k | (n-1)$, $2^k > \sqrt{n}$, и $a^{\frac{n-1}{2}} = -1 \pmod{n}$ за неки цели број a, онда је n прост број.

Ова теорема може послужити као тест прималности за бројеве облика $h \cdot 2^k + 1$, тзв. **Протове бројеве**. Први **прости Протови бројеви** су 3, 5, 13, 17, 41, 97, 113, 193. Уз Мерсенове просте бројеве, Протови прости бројеви су међу највећим познатим простим бројевима. Протови прости бројеви су такође битни јер представљају могуће факторе Фермаових бројева. Приметимо и да су Фермаови бројеви специјалан случај Протових бројева.[22]

Важи и општије тврђење:

[19] Број $n^k - 1$, $n \geq 2$, $k > 0$ је дељив са $n - 1$.

[20] GIMPS, велико интернет истраживање простих бројева, је извршило проверу свих мањих Мерсенових бројева. Како нису пронађени прости бројеви, "41. ***познати*** Мерсенов прост број" сада је једноставно "41. Мерсенов прост број".

[21] Прот (François Proth), 1852 – 1879, француски математичар.

[22] За више информација о Протовим бројевима, факторизацији Фермаових бројева, и другим интересантним темема и програмима, видети `http://www.prothsearch.net/` и `http://primes.utm.edu/programs/gallot/index.html`.

Теорема 2.26. *Нека је $n = h \cdot q^k + 1$, где је q прост број и $q^k > h$, $h \neq q$. Ако постоји цео број a такав да је $a^{n-1} = 1 \pmod n$, и $\mathrm{nzd}(a^{\frac{n-1}{q}} - 1, n) = 1$, онда је n прост број.*

Дефиниција 2.33. Рамануџанов[23] n-ти прост број је најмањи цео број R_n такав да важи $\pi(x) - \pi(x/2) \geq n$ за свако $x \geq R_n$.

Најмањи Рамануџанови прости бројеви су 2, 11, 17, 29, 41, итд.

2.7 Риманова зета-функција. Расподела простих бројева.

Риманова зета-функција $\zeta(x)$ је због своје везе са расподелом простих бројева једна од најважнијих функција у теорији бројева.

Дефиниција 2.34. Риманова функција, у ознаци $\zeta(s)$, је функција комплексне променљиве s, $s = \sigma + i \cdot t$, где су σ, t реални бројеви, и дефинисана је следећом бесконачном сумом:

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s}, (Re(s) > 1),$$

при чему ред на десној страни апсолутно и униформно конвергира за $Re(s) > 1$ (тј. за $\sigma > 1 + \delta$ за свако $\delta > 0$), и у тој равни представља регуларну функцију од s. За остале вредности s, зета-функција се дефинише аналитичким продужењем кроз комплексну раван и регуларна је осим за $s = 1$.

Функција $\zeta(s)$ даје аналитички приступ теорији простих бројева. Ојлер је открио везу зета-функције и расподеле простих бројева:

$$\sum_{n\geq 1} \frac{1}{n^s} = \prod_{p\in\mathbb{P}} \frac{1}{1-p^{-s}} = (1 + \frac{1}{2^s} + \frac{1}{4^s} + ...)(1 + \frac{1}{3^s} + \frac{1}{9^s} + ...) \cdot ... \cdot (1 + \frac{1}{p^s} + \frac{1}{p^{2s}} + ...) \cdots \quad (\text{за } \sigma > 1)$$

где је по дефиницији лева страна $\zeta(s)$, а бесконачни производ на десној страни је по свим простим бројевима p.

Ојлеров производ показује да Риманова зета-функција нема нуле за $\sigma > 1$. Такође, $\zeta(s)$ нема нуле за $\sigma < 0$ осим у тачкама $s = -2, -4, -6, ...$ које се називају *тривијалне нуле*. Све друге нуле Риманове зета-функције морају лежати између $0 \leq \sigma \leq 1$. Чувена **Риманова хипотеза** каже да ове нуле у ствари леже на линији $\sigma = \frac{1}{2}$. Иако Риманова хипотеза није доказана, постоје многа тврђења и рачунарска израчунавања која сведоче у њену корист.

Зета-функција има следеће вредности за неке одабране бројеве: $\zeta(0) = -\frac{1}{2}$, $\zeta(\frac{1}{2}) \approx -1.46035$, $\zeta(1) = 1 + \frac{1}{2} + \frac{1}{3} + ... = \infty$, (хармонијски низ).

Теорема 2.27. *Постоји бесконачно много простих бројева.*

Доказ. Овај доказ дао је Еуклид. Претпоставимо супротно, да постоји коначни скуп који садржи све просте бројеве, $\{p_1, p_2, ..., p_k\}$. Размотримо број $n = p_1 \cdot p_2 \cdot ... \cdot p_k + 1$. Према основној теореми аритметике, број n се може представити као производ простих бројева, што значи да постоји прост број p такав да $p|n$. Према дефиницији, $n \equiv 1 \pmod{p_i}$ за све p_i, тј. $p_i|n - 1$ и $p_i \nmid n$. Међутим, важи да $p|n$, па се може закључити да је $p \neq p_i$ за свако p_i. Одатле следи да је p прост број који не припада скупу свих простих бројева, што је контрадикција. Одавде следи тражена теорема. □

Напомена 2.17. Посматрајмо бројеве облика $n = p_1 \cdot p_2 \cdot ... \cdot p_k + 1$, где је $\{p_1, p_2, ..., p_k\}$ скуп свих простих бројева до простог броја p_k заједно са p_k. Примери простих бројева овог облика: $2 \cdot 3 + 1 = 7$, $2 \cdot 3 \cdot 5 + 1 = 31$, $2 \cdot 3 \cdot 5 \cdot 7 + 1 = 211$, итд. Прости бројеви овог облика се зову и **Еуклидови прости бројеви** (енгл. *Euclid primes*). У општем случају, бројеви овог облика, тј. Еуклидови бројеви не морају да буду прости. Пример је број $510511 = 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 + 1$ који је дељив бројевима 19, 91, и 277.

[23] Рамануџан (Srinivasa Ramanujan), 1887–1920, један од највећих индијских математичара.

Већина тестова прималности се примењује са жељом да се покаже простост веома велики прости бројеви, и да се то уради што брже. Иако се прости бројеви *проређују* у низу природних бројева и нису правилно распоређени, показали смо да их има бесконачно много.
Из претходне теореме следи да је $\lim_{n\to\infty}\pi(n) = +\infty$. За мале вредности броја n лако је наћи вредност функције $\pi(n)$, али када су у питању произвољно велики бројеви, било би лепо имати једноставан израз који даје $\pi(n)$. Нажалост, не постоји тачан израз за $\pi(n)$, али постоје апроксимације за које је доказано да су приближно тачне.

Теорема 2.28 (**Основна теорема о простим бројевима**). *Асимптотски закон расподеле простих бројева гласи:* $\pi(n) \sim \frac{n}{\ln n}$, *тј.* $\pi(n)$ *је асимптотски једнако* $\frac{n}{\ln n}$, $\lim_{n\to\infty}\frac{\pi(n)}{\frac{n}{\ln n}} = 1$.

Основну теорему о простим бројевима (енгл. *prime number theorem, PNT*) доказали су, независно један од другог, Адамар[24] и Вале-Пуасен[25] 1896. године. Оно што су они заправо доказали је да за неки број $C > 0$ важи: $\pi(n) = li(n) + O(n \cdot e^{-C\sqrt{\ln n}})$, где је $li(n)$ дефинисано на следећи начин: $li(n) = \int_2^n \frac{dx}{\ln x}$. Следи да је $li(n) \sim \frac{n}{\ln n}$.

Пример 2.14. У следећој табели је дата тачна вредност $\pi(n)$ и њена апроксимација:

n	2	3	4	5	6	7	8	9	10	100	1000	10000
$\pi(n)$	1	2	2	3	3	4	4	4	4	25	168	1229
$\frac{n}{\ln n}$	2.9	2.7	2.9	3.1	3.3	3.6	3.8	4.1	4.3	22	145	1086

Поређење тачних вредности за $\pi(n)$ и апроксимације

Апроксимације у горњој табели не изгледају блиско тачним вредностима, али како n расте, апроксимација се побољшава [1].

Последица 2.4. *Апроксимација* n*-тог простог броја је последица Основне теореме о простим бројевима. Ако се са* p_n *означи* n*-ти прост број, онда је* $p_n \sim n \cdot \ln n$.

Основна теорема о простим бројевима се често користи да би се извршила процена колико има простих бројева у одређеном интервалу, тако што се једноставно одузимају две вредности за $\pi(n)$.

Пример 2.15. Претпоставимо да је потребно проценити колико има простих бројева између 10^{10} и 10^{11}: $\pi(10^{11}) - \pi(10^{10}) \approx 3.9481 \cdot 10^9 - 4.3429 \cdot 10^8 \approx 3.512 \cdot 10^9$, тј. у овом интервалу има око $3.512 \cdot 10^9$ простих бројева. Ако се овај број подели са величином интервала, добија се да је око 3.9% целих бројева у овом интервалу просто, или у просеку један од сваких 26 броја.

На пример, ако је у криптографској апликацији потребан случајан прост број између 10^{10} и 10^{11}, онда је у просеку потребно да се тестира 26 случајна цела броја пре него што се нађе прост број. Такође, ако неко жели да погоди који прост број из овог интервала је коришћен у криптографском алгоритму, он треба више пута да врши погађање, што представља заштиту.

Уопштено говорећи, Основна теорема о простим бројевима каже да ако је потребно наћи прост број око броја n, треба тестирати око $\ln n$ случајно изабраних целих бројева. Ово може бити упола смањено ако се за случајне узорке узимају само непарни бројеви у близини броја n [1].

Напомена 2.18. Може се показати да је $\pi(n) \sim \frac{n}{\ln n - 1}$ боља оцена за $\pi(n)$. За више информација видети [17].

Тврђење 2.10. *За дати произвољан цео број* $k > 0$, *може се наћи* k *узастопних сложених бројева.*

Претходно тврђење, које још једном потврђује да су прости бројеви ретки у скупу природних бројева, може се лако доказати конструкцијом тражених бројева.

Теорема 2.29 (**Бертранов постулат**). *Прости бројеви су добро распоређени у смислу да, за свако* $n > 1$, *увек постоји прост број између* n *и* $2n$.

[24] Адамар (Jacques Salomon Hadamard), 1865–1963, француски математичар.
[25] Вале-Пуасен (Charles Jean de la Vallée-Poussin), 1866–1962, белгијски математичар.

Чебишев[26] је дао потпун доказ овог тврђења, тако да се он још назива и Бертран-Чебишевљева теорема или Чебишевљева теорема.

Занимљиво је питање које класе остатака a по модулу d садрже просте бројеве, и у класама које их садрже, колико су честа појављивања простих бројева. Ако a и d имају заједнички прост фактор, онда тај прост број дели сваки елемент те класе остатака, одакле следи да та класа остатака не може садржати више од тог једног простог броја. Један од најбитнијих резултата оваквог разматрања је да је то једина препрека да класа остатака садржи бесконачно много простих бројева.

Теорема 2.30 (Дирихлеова теорема). [27] *Свака аритметичка прогресија $a + k \cdot n$, $k \geq 0$ цео број, и бројеви a и n узајамно прости,* $\mathrm{nzd}(a, n) = 1$*, садржи бесконачно много простих бројева.*

Напомена 2.19. Јака форма Дирихлеове теореме каже да за било коју овакву аритметичку прогресију, сума реципрочних вредности простих бројева у прогресији дивергира.

Ако се са $\pi(x; n, a)$ означи број простих бројева у класи остатака a по модулу n који су мањи или једнаки од x и важи $\mathrm{nzd}(a, n) = 1$, онда за фиксиране целе бројеве a, $n > 0$ важи да је $\pi(x; n, a) \sim \frac{1}{\phi(n)} \cdot \pi(x) \sim \frac{1}{\phi(n)} \cdot \frac{x}{\ln x} \sim \frac{1}{\phi(n)} \cdot li(x)$. Како класе остатака по модулу n које нису узајамно просте са n могу да садрже највише један прост број, следи да су сви прости бројеви, осим њих коначно много, у осталих $\phi(n)$ класа остатака по модулу n, и свака од тих класа остатака по модулу n садржи асимптотски исти део простих бројева. Овај резултат се још назива и **теорема о простим бројевима за класе остатака**. Другачије речено, различите аритметичке прогресије са истим модулом n имају приближно исте пропорције простих бројева.

Пример 2.16.

- За $a = 5$, $k = 6$, важи да је $\mathrm{nzd}(5, 6) = 1$ и простих бројева облика $6n + 5$ има бесконачно много, тј. аритметичка прогресија $5, 11, 17, 23, 29, 35, 41, 47, 53, 59, 65, ...$ садржи бесконачно много простих бројева.
- Постоји бесконачно много простих бројева облика $2n+1$, $4n+1$, $4n+3$, $6n+5$. Међу бројевима облика $4n + 3$, прости бројеви су $3, 7, 11, 19, 23, 31, 43, 47$, итд.

  На основу јаке Дирихлеове теореме важи да $\frac{1}{3} + \frac{1}{7} + \frac{1}{11} + \frac{1}{19} + \frac{1}{23} + \frac{1}{31} + \frac{1}{43} + \frac{1}{47} + ...$ дивергира.

2.8 Поље Галоа

Галоаова поља[28] су поља коначног реда. Коначна поља имају важну улогу у теорији бројева, алгебарској геометрији, криптографији, теорији кодирања. Користи се нотација: $\mathbb{GF}(n)$ — поље Галоа реда n, (енгл. *Galois field*).

Дефиниција 2.35. Нека је $\mathbb{L}$ поље које садржи подскуп $\mathbb{K}$, $\mathbb{K} \subset \mathbb{L}$, и $\mathbb{K}$ је са операцијама наслеђеним из $\mathbb{L}$ сам за себе поље. Тада се $\mathbb{L}$ назива **проширењем** поља $\mathbb{K}$ (енгл. *extension field*), а $\mathbb{K}$ се назива **подпољем** од $\mathbb{L}$ (енгл. *subfield*). Свако поље има најмање подпоље, које се назива **просто подпоље** (енгл. *prime subfield*), и које је изоморфно или пољу рационалних бројева $\mathbb{Q}$ (у том случају се каже да је поље карактеристике нула), или је изоморфно са $\mathbb{GF}(p)$ за неки прост број p (у овом случају се каже да је поље карактеристике p).

Дефиниција 2.36. Нека је $\mathbb{L}$ проширење поља $\mathbb{K}$. Тада се за $\mathbb{L}$ може рећи и да је **векторски простор** над подпољем $\mathbb{K}$, где су векторско сабирање и скаларно множење једноставно операције сабирања и множења у $\mathbb{L}$. Димензија овог векторског простора се зове **степен** од $\mathbb{L}$ над $\mathbb{K}$ (енгл. *degree*), и означава се са $[\mathbb{L} : \mathbb{K}]$.

Тврђење 2.11 (Особине генератора)**.**

1. *Ред n поља $\mathbb{GF}(n)$ мора бити* ***степен простог*** *броја. Нека је p карактеристика поља $\mathbb{GF}(n)$. Онда $\mathbb{GF}(n)$ садржи просто поље $\mathbb{GF}(p)$. Лако се може показати да ако је $\mathbb{GF}(n)$ поље са n елемената, онда је $n = p^k$, односно n је степен карактеристике p, p је прост број. Такође,*

[26] Чебишев (Pafnuty Chebyshev или Tchebysheff), 1821-1894, руски математичар.
[27] Дирихле (Gustav Lejeune Dirichlet), 1805-1859, немачки математичар.
[28] Галоа (Evariste Galois), 1811–1832, француски математичар.

за сваки степен $k > 0$ простог броја p постоји поље од n елемената $n = p^k$ и ово поље је јединствено (до на изоморфизам). Два поља су изоморфна ако имају исту структуру, иако репрезентација њихових елемената може бити различита.

2. *Свако подпоље $\mathbb{GF}(q)$ коначног поља $\mathbb{GF}(n)$, $n = p^m$, има ред $q = p^k$, где k дели m.*

3. *Елемент $a \in \mathbb{Z}_n^*$, односно $a \in \mathbb{GF}(n)$, који има ред $\phi(n)$ се зове **генератор** или **примитивни корен**. Свако поље $\mathbb{GF}(n)$ садржи бар један генератор a.*

4. *Нека $a \in \mathbb{Z}_n$. Тада a има инверз ако и само ако је $\text{nzd}(a, n) = 1$. Мултипликативна група од $\mathbb{Z}_n$ се означава са $\mathbb{Z}_n^*$: $\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n | \text{ nzd}(a, n) = 1\}$. Ако је a генератор у $\mathbb{Z}_n^*$, онда је $\mathbb{Z}_n^* = \left\{a^i \pmod n | 0 \leq i \leq \phi(n) - 1\right\}$.*
Другачије речено, ако је $\mathbb{Z}_n$ поље сви не-нула елементи у $\mathbb{Z}_n$ могу бити репрезентовани као $\phi(n) = n - 1$ узастопних степена генератора a: $1,\ a,\ a^2,\ a^3,...,\ a^{(n-2)},\ a^{(n-1)} = 1,\ a^n = a, ...$

5. *$\mathbb{Z}_n^*$ има генератор ако и само ако је $n = 2, 4, p^k$ или $2p^k$, где је p непаран прост број и $k \geq 1$. Ако је p прост број, онда је $\mathbb{Z}_p$ поље и има генератор. Ако је $n = p \cdot q$, p, q различити прости бројеви већи од 2, онда n нема примитивни корен. Ово следи из чињенице да је $\phi(n) = \phi(p)\phi(q)$, $\phi(p) = p - 1$ и $\phi(q) = q - 1$ су парни бројеви, и за сваки број a, такав да је $\text{nzd}(a, n) = 1$, на основу Ојлерове теореме важи да је $a^{\frac{\phi(n)}{2}} = (a^{\phi(p)})^{\frac{\phi(q)}{2}} = 1 \pmod p$, и слично $a^{\frac{\phi(n)}{2}} = 1 \pmod q$, одакле следи да је $a^{\frac{\phi(n)}{2}} = 1 \pmod n$, па $n = p \cdot q$ не може имати примитивни корен.*

6. *Претпоставимо да је a генератор у $\mathbb{Z}_n^*$. Тада је $b = a^k \pmod n$ такође генератор у $\mathbb{Z}_n^*$ ако и само ако је $\text{nzd}(k, \phi(n)) = 1$. Одатле следи да ако је $\mathbb{Z}_n^*$ циклична, тада је број генератора $\phi(\phi(n))$.*

7. *Број $a \in \mathbb{Z}_n^*$ је генератор у $\mathbb{Z}_n^*$ ако и само ако је $a^{\phi(n)/p} \neq 1 \pmod n$ за сваки прост делитељ p од $\phi(n)$.*

Пример 2.17. На основу мале Фермаове теореме следи да у $\mathbb{Z}_5$ важи да је $x^4 \equiv 1 \pmod 5$. Такође, једноставним израчунавањем се добија да је $(x-1)(x-2)(x-3)(x-4) = x^4 - 10x^3 + 35x^2 - 50x + 24 \equiv x^4 - 1 \pmod 5$. Заправо, на основу мале Фермаове теореме, у општем случају важи да је $x^{p-1} - 1 \equiv (x-1)(x-2) \cdot ... \cdot (x-p+1) \pmod p$, $x \in \mathbb{Z}_p$, p прост број.

Дефиниција 2.37. Нека је p *велики* прост број и $g, a \in \mathbb{Z}_p^*$ (у општем случају $g, a \in \mathbb{G}_p$, где је $(\mathbb{G}_p, \cdot)$ коначна циклична група), и нека је g генератор те групе. Тада је *веома тешко* наћи цео број x који задовољава једначину $g^x = a \mod p$. Овај проблем се назива **проблем дискретног логаритма** (енгл. *discrete logarithm problem, DLP*). Број x се зове **индекс** (енгл. *index*) броја a у односу на g (по модулу p) и означава се са $ind\ a$.

Полиноми над пољем Галоа

Дефиниција 2.38. Прстен полинома $\mathbb{GF}(n)[x]$ је скуп свих полинома $a_0 + a_1x + a_2x^2 + ... + a_mx^m$ по променљивој x са коефицијентима $a_i \in \mathbb{GF}(n)$. Највећи цео број m за који је $a_m \neq 0$ је степен полинома $f(x)$, у ознаци $deg\ f(x)$, а коефицијент a_m је водећи коефицијент. Полином је **моничан** (енгл. *monic*) ако је водећи коефицијент једнак 1.

За свако поље $\mathbb{GF}(n)$, прстен полинома $\mathbb{GF}(n)[x]$ је комутативан, нема правих делитеља нуле, и његови инвертибилни елементи су заправо ненула елементи поља $\mathbb{GF}(n)$.

Дефиниција 2.39. Полином $p(x)$ је **нерастављив** (енгл. *irreducible*) у пољу $\mathbb{GF}(n)$ ако $p(x)$ не може бити факторисан у производ два полинома нижег степена у $\mathbb{GF}(n)[x]$, искључујући константе.

Прстен полинома има многе заједничке особине са прстеном целих бројева. Нерастављиви полиноми у прстену полинома имају исту улогу као прости бројеви у прстену целих бројева. Прстен полинома има **јединствену факторизацију** (енгл. *unique factorization*) што значи да сваки моничны полином може бити записан на само један начин, до на распоред фактора, као производ моничних нерастављивих полинома. Полином који није моничан може бити записан на јединствен начин као константа пута такав производ. Важно је поменути да за полиноме над коначним пољем

важе тврђења еквивалентна Еуклидовом алгоритму, алгоритму дељења, као и особине релације конгруенције. За више информација о проширеном Еуклидовом алгоритму за полиноме, који се користи и за израчунавање инверза полинома, видети [17, 28].

Дефиниција 2.40. Ако $g(x), h(x) \in \mathbb{GF}(n)[x]$, онда е каже да је $g(x)$ **конгруентан** $h(x)$ **по модулу** $f(x)$ ако $f(x)$ дели $g(x) - h(x)$. Ово се означава са $g(x) \equiv h(x) \pmod{f(x)}$.

Дефиниција 2.41. Полином $p \neq 0$ је **прост** у $\mathbb{GF}(n)$ ако није инветибилан и ако за произвољне полиноме $a,\ b$ важи: $p = a \cdot b \Rightarrow p|a \vee p|b$

Теорема 2.31. *Полином p у $\mathbb{GF}(n)$ је прост ако и само ако је и нерастављив. Посебно, сваки полином степена* 1 *је прост.*

Дефиниција 2.42. Нерастављив полином $f(x)$ степена m са коефицијентима из $\mathbb{Z}_p$ је **примитиван** (енгл. *primitive polynomial*) ако је x генератор у $\mathbb{GF}^*(p^m)$, мултипликативне групе свих ненула елемената из $\mathbb{GF}(p^m) = \mathbb{Z}_p[x]/(f(x))$.

Дефиниција 2.43. Нерастављив полином $f(x) \in \mathbb{GF}(n)[x]$ степена m је примитиван ако и само ако је најмањи позитиван цео број k за који $f(x)$ дели $x^k - 1$, једнак $k = n^m - 1$.

Другачије речено, полином $f(x)$ степена m са коефицијентима у $\mathbb{GF}(p) = \mathbb{Z}_p$ је примитиван полином ако има корен $a \in \mathbb{GF}(p^m)$ такав да скуп $\{0, 1, a, a^2, a^3, ..., a^{p^m-2}\}$ чини цело поље $\mathbb{GF}(p^m)$, и $f(x)$ је полином са најмањим степеном који има a као корен.

Теорема 2.32. *Корени a_j примитивног полинома m-ог степена, $f(x) \in \mathbb{GF}(n)[x]$, имају ред $n^m - 1$.*

Треба приметити да су корени a_j примитивног полинома m-ог степена заправо примитивни елементи у $\mathbb{GF}(n^m)$.

Теорема 2.33. *За сваки $m \geq 1$ постоји тачно $\frac{\phi(p^m-1)}{m}$ моничних примитивних полинома степена m над $\mathbb{Z}_p$.*

Дефиниција 2.44. $\mathbb{GF}(n)[x]/(f(x))$ означава скуп (тј. класу еквиваленције) полинома у $\mathbb{GF}(n)[x]$ степена мањег од степена полинома $f(x)$. Сабирање и множење се извршавају по модулу $f(x)$.

Теорема 2.34. *$\mathbb{GF}(n)[x]/(f(x))$ је комутативни прстен. Ако је $f(x)$ нерастављив полином степена m над $\mathbb{GF}(n)$, онда је $\mathbb{GF}(n)[x]/(f(x))$ поље.*

Конструкција поља Галоа $\mathbb{GF}(2^m)$

Пример 2.18. Конструкција $\mathbb{GF}(2)[x]$ Полином степена n над коначним пољем $\mathbb{GF}(2)$, тј. са коефицијентима 0 или 1, је примитиван ако његов корен a има ред $2^n - 1$. Следи да је $x^2 + x + 1$ нерастављив над $\mathbb{GF}(2)$ јер није дељив ниједним полиномом првог степена. Полином $x^2 + x + 1$ је и примитиван, и има ред $2^2 - 1 = 3$.

Такође, $p(x) = x^3 + x + 1$ је примитиван у $\mathbb{GF}(2)[x]$ и има ред $2^3 - 1 = 7$. Посматрајмо сада полиноме из $\mathbb{GF}(2)[x]$ и њихове остатке по модулу $p(x) = x^3 + x + 1$. Нека је a корен од $p(x) \Rightarrow a^3 + a + 1 = 0 \Rightarrow a^3 = a + 1$. Приметимо да се коефицијенти рачунају по модулу 2, а полиноми по модулу $x^3 + x + 1$. Такође, важи да је $-1 = 1$.

Експоненцијални запис	Полиномијални запис	Векторски запис
0	0	(0, 0, 0)
a^0	1	(0, 0, 1)
a^1	a	(0, 1, 0)
a^2	a^2	(1, 0, 0)
a^3	$a + 1$	(0, 1, 1)
a^4	$a^2 + a$	(1, 1, 0)
a^5	$a^2 + a + 1$	(1, 1, 1)
a^6	$a^2 + 1$	(1, 0, 1)
a^7	1	(0, 0, 1)

Конструкција поља $\mathbb{GF}(2^3)$

Сабирање: $a^2 + a^5 = (a^2) + (a^2 + a + 1) = (a + 1) = a^3$;
Множење: $a^4 \cdot a^5 = (a^2 + a)(a^2 + a + 1) = a^4 + a = a^2 + a + a = a^2$, тј. $a^4 \cdot a^5 = a^{9 \pmod{2^3-1}} = a^2$.
Одавде следи да је $\mathbb{GF}(2)[x]/(x^3 + x + 1) = \{0,\ 1,\ x,\ x + 1,\ x^2,\ x^2 + x,\ x^2 + 1,\ x^2 + x + 1\}$ поље од 8 елемената тј. поље $\mathbb{GF}(2^3)$ које је репрезентовано скупом свих полинома над $\mathbb{GF}(2)$ степена мањег од 3. Другачије записано, $\mathbb{GF}(2^3) = \{a_2x^2 + a_1x + a_0|\ a_i \in \{0, 1\}\}$. Може се користити и векторски запис $\mathbb{GF}(2^3) = \{(a_2, a_1, a_0)|\ a_i \in \{0, 1\}\}$.

Примитивни полиноми степена n над $\mathbb{GF}(2)$ се употребљавају за генерисање псеудо-случајних низова n-торки нула и јединица, што може бити корисно за криптографске апликације.

2.9 Елиптичке криве

Дефиниција 2.45. Нека је $\mathbb{K}$ поље карактеристике $char\ \mathbb{K} \neq 2, 3$ и нека је $f(x) = x^3 + ax + b$, где $a, b \in \mathbb{K}$, полином трећег степена без вишеструких корена, тј. важи $4a^3 + 27b^2 \neq 0$. **Елиптичка крива** над пољем $\mathbb{K}$ (енгл. *elliptic curve*), у ознаци $E(\mathbb{K})$, је скуп тачака (x, y), таквих да $x, y \in \mathbb{K}$ задовољавају једначину $y^2 = x^3 + ax + b$, укључујући елемент O кога називамо бесконачна тачка (енгл. *point at infinity*).

Општа форма једначине за елиптичку криву која се може применити за свако поље $\mathbb{K}$ је $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$, и која се за $char\ \mathbb{K} \neq 2$ може трансформисати у $y^2 = x^3 + ax^2 + bx + c$, а за $char\ \mathbb{K} > 3$ у једначину $y^2 = x^3 + ax + b$.

Ако је $F(x, y) = 0$ имплицитна једначина за y као функције од x, тј. $F(x, y) = y^2 - x^3 - ax - b$, онда је тачка (x, y) на кривој несингуларна (енгл. *non-singular*) ако је бар један од парцијалних извода $\partial F/\partial x$, $\partial F/\partial y$ различит од нуле у тој тачки. Није тешко показати да је услов да кубна једначина на десној страни нема вишеструке корене еквивалентан услову да све тачке на кривој буду несингуларне.

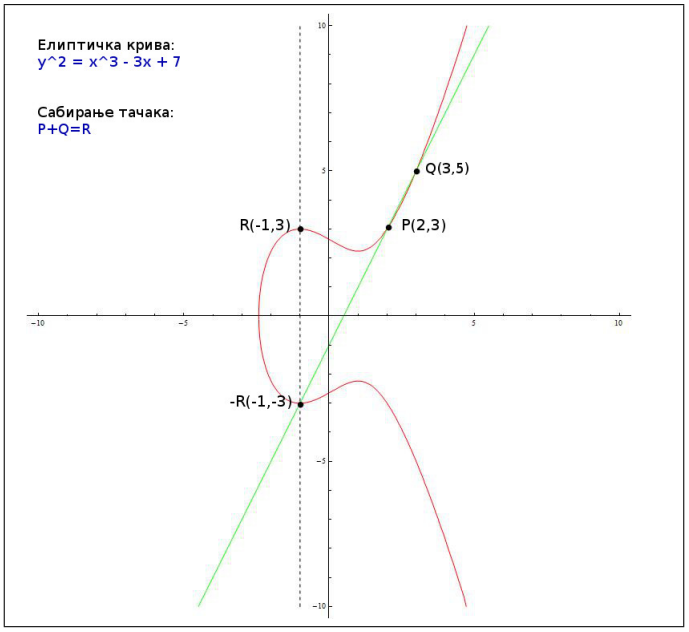
Интересантно је истакнути да скуп тачака на елиптичкој кривој образује комутативну групу. Обично се разматрају елиптичке криве над коначним пољем $\mathbb{Z}_p$, где је p прост број већи од 3. Такође, поља $\mathbb{GF}(2^n)$ карактеристике 2 су посебно интересантна због ефикаснијих израчунавања над елиптичким кривама.

Дефиниција 2.46. Нека је E елиптичка крива над скупом реалних бројева, дефинисана једначином $y^2 = x^3 + a \cdot x + b$, и нека су P и Q две тачке на E. Негативни број од P и збир $P + Q$ се дефинишу на следећи начин:

- Нека је O бесконачна тачка. Ако је $P = O$, онда је $-P$ једнако O и $P + Q = Q$, тј. O је адитивни неутрални елемент групе тачака на елиптичкој кривој.
- Ако су кординате тачке P једнаке (x, y), онда су координате тачке $-P$ једнаке $-(x, y) = (x, -y)$. Наравно, важи да $(x, -y) \in E$.
- Ако је $Q = -P$, онда је $P + Q = O$.
- Ако P и Q имају различите x координате, онда линија $l = \overline{PQ}$ пресеца криву у тачно још једној тачки (осим ако та линија није тангента и у том случају се узима да је $R = Q$). Ако се дефинише $P + Q = R$, чије су координате (x_3, y_3), онда је тачка пресека праве са кривом тачка $-R$ са координатама $(x_3, -y_3)$. Лако се може извести (пресек праве $l = \overline{PQ}$ и криве) да је $x_3 = (\frac{y_2 - y_1}{x_2 - x_1})^2 - x_1 - x_2$, $y_3 = -y_1 + (\frac{y_2 - y_1}{x_2 - x_1})(x_1 - x_3)$.
- Ако је $P = Q$, l је тангента на криву у тачки P, онда је са $-R$ означена једина друга тачка пресека тангенте l и криве, и дефинише се да је $P + Q = R$, чије су координате (x_3, y_3). Лако се може показати (имплицитним диференцирањем) да је $x_3 = (\frac{3x_1^2 + a}{2y_1})^2 - 2x_1$, $y_3 = -y_1 + (\frac{3x_1^2 + a}{2y_1})(x_1 - x_3)$.

Приметимо да је операција групе означена као сабирање тачака које припадају елиптичкој кривој над пољем $\mathbb{K}$, за разлику од операције у $\mathbb{K}$ која је означена мултипликативно.

Следи график који илуструје сабирање тачака $P(2, 3)$ и $Q(3, 5)$ које припадају елиптичкој кривој $y^2 = x^3 - 3x + 7$ над $\mathbb{R}$. График је урађен у програму Mathematica.



Пример сабирања тачака на елиптичкој кривој

Ленстра је 1985. године увео употребу елиптичких кривих у методе за факторизацију, а убрзо потом је кренула примена елиптичких кривих у тестовима прималности. Данас се све више користе криптографски алгоритми засновани на елиптичким кривама.

Више о елиптичким кривама се може наћи у [27].

2.10 Неки нерешени проблеми теорије бројева

Одређени тестови простости ослањају се на хипотезе које још нису доказане, а за које се верује да су тачне. Нарочито су познати тестови простости који претпостављају тачност Риманове хипотезе.

Следе неки од најбитнијих нерешених проблема из теорије бројева:

1. **Хипотеза о простим бројевима близанцима** (енгл. *Twin prime conjecture*): има бесконачно много парова простих бројева који се разликују само за 2.
2. **Голдбахова хипотеза**: [29] сваки паран број већи од 2 може се написати као сума два проста броја.
3. Да ли важи **Риманова хипотеза**?
4. Да ли увек постоји прост број између n^2 и $(n+1)^2$, тј. да ли увек **постоји бар један прост број између свака два квадрата целих бројева**?
5. Да ли постоји бесконачно много **Фермаових простих бројева**? Заправо, да ли постоји било који Фермаов прост број после F_4?
6. Да ли постоји бесконачно много **Мерсенових простих бројева**?
7. Да ли има бесконачно много простих бројева облика $n^2 + 1$?
8. Да ли постоји **непаран савршен број**?
9. Ако је p прост, да ли је $2^p - 1$ увек безквадратан? Другачије речено, да ли су Мерсенови бројеви простог индекса увек безквадратни?
10. Да ли **Фибоначијев низ**[30] садржи бесконачно много простих бројева, тј. да ли постоји бесконачно много Фибоначијевих простих бројева?

[29] Поменута у Голдбаховом писму Ојлеру, 1742. године.
[30] Низ бројева у коме збир претходна два броја у низу даје вредност наредног члана низа.

3 Основни алгебарски алгоритми

Једноставно речено, **алгоритам** је скуп добро дефинисаних корака потребних да се изврши неки задатак. Под појмом алгоритма сматрамо методу (процедуру) за решавање неке класе проблема, која за улазне податке одређеног типа даје одговор (тј. излазне податке) у коначном времену. Другим речима, алгоритам је ефективни метод изражен помоћу коначне листе добро дефинисаних инструкција за израчунавање функције.

Обично се под појмом алгоритма подразумевају детерминистички алгоритми. **Детерминистички алгоритам** је алгоритам који за дати улаз увек даје исти излаз, пролазећи кроз исти скуп добро дефинисаних корака.

Недетерминистички алгоритам се разликује од детерминистичког по томе што може да дође до излаза користећи различите *путеве*, што се види ако се за њихов опис користи недетерминистичка Тјурингова машина.[31]

Пробабилистички алгоритми (енгл. *randomized algorithms*), тј. **алгоритми са случајношћу**, се у зависности од улаза и одређеног случајног параметра извршавају на различите начине. Постоје алгоритми који користе случајност да би ефикасније вратили тачан одговор.[32]**Монте Карло алгоритам** је пробабилистички алгоритам који може вратити нетачан резултат са одређеном (малом) вероватноћом. Други тип пробабилистичког алгоритма је **Лас Вегас тип алгоритма** који никад не враћа погрешан резултат, али његово време извршавања није гарантовано [10].[33]

У овом раду тестови простости су подељени на оне који враћају резултат да је број прост са одређеном вероватноћом, и оне који доказују простост датог броја. Тестови се могу поделити и на детерминистичке и пробабилистичке, али треба имати у виду да, на пример, Лас Вегас тип алгоритма (ако врати резултат) тачно одређује да ли је задати број прост или сложен, као и да постоје тестови који користе детерминистички поступак, а чији је резултат вероватно прост број [28].

3.1 Перформансе програма. Сложеност израчунавања

Количина меморије, спољашње меморије и процесорског времена који су потребни да би се програм извршио од пресудног су значаја за његову *употребну вредност*. За почетак, уводе се јединице за чување дигиталних информација и брзину преноса података.

Код свих програма, а нарочито код имплементације тестова прималности, алгоритама за факторизацију и сита, поред брзине којом се извршавају, веома је битна и количина меморије и спољашње меморије коју они користе. Јединица mega-bit означава 1,000,000 бита, mega-byte означава 1,000,000 бајта, док јединица mebi-bit означава 1,048,576 бита, а mebi-byte означава 1,048,576 бајта. Ако суфикс b означава битове, а суфикс B бајтове, претходно се може записати и као Mb $= 1000^2 = 10^6$ бита, MB $= 8 \cdot 1000^2$ бита, Mib $= 1024^2 = 2^{20}$ бита, MiB $= 8 \cdot 1024^2$ бита. Другим речима, префикси kibi-, mebi-, gibi-, tebi-, итд, означавају јединице у односу на степене броја 2, док су kilo-, mega-, giga-, tera-, итд, SI префикси и *увек* означавају степене броја 10. Међутим, у пракси често долази до погрешне употребе ових јединица. На пример, када оперативни систем MS Windows показује величину датотеке на диску, он користи степене броја 2 да би израчунао величину, међутим добијену величину приказује уз SI префиксе.

Како је мрежни пренос података веома важан, нарочито код паралелних програма са великим бројем улазно-излазних операција, битно је знати да 10 Base-T Ethernet подржава брзину од 10 Mbps, 100 Base-TX, тј. Fast Ethernet, брзину од 100 Mbps, док 1000 Base-T, тј. Gigabit Ethernet, подржава брзину од 1000 Mbit/s = 1000 Mbps = 1 Gbps = 1,000,000,000 бита у секунди, односно $125,000,000$ бајта у секунди. Постоји и 10 Gigabit Ethernet. Што се тиче могуће брзине приступа диску, за SATA дискове друге генерације, односно SATA 3 Gbit/s та брзина је 375 MB/s, тј. у пракси максимално 300 MB/s, а за SATA треће генерације, односно SATA 6 Gbit/s, брзина је 750 MB/s, тј. у пракси максимално 600 MB/s.[34]

[31] За више информација о Тјуринговим машинама, видети [10, 16, 26].

[32] За више информација, видети "Увод у теоријско рачунарство", Зоран Огњановић, Ненад Крџавац, 2004.

[33] За више информација, видети *"Computational complexity"*, C. H. Papadimitriou, Addison-Wesley, 1994.

[34] За више информација, видети `http://en.wikipedia.org/wiki/Data_rate_units`, `http://en.wikipedia.org/wiki/Serial_ATA`.

Ефикасност алгоритма служи да бисмо описали количину одређеног ресурса који тај алгоритам користи.

Честа грешка која се прави при анализирању ефикасности алгоритма је погрешно изједначавање појма **перформансе програма** — количина процесора/меморије/диска коју тај програм користи, са појмом сложености (енгл. *complexity*) алгоритма.[35] Сложеност алгоритма је мера одређеног ресурса потребног да се изврши тај алгоритам (формално се појмови временске и просторне сложености израчунавања дефинишу у односу на Тјурингову машину), па се **сложеност алгоритма** дефинише као мера броја израчунавања, или операција, потребних да се изврши одређени алгоритам. При томе, није неопходно мерити тачан број операција, већ одредити како број извршених операција зависи од величине проблема. Мерећи сложеност алгоритма, жели се предвидети како ће се програм заснован на том алгоритму извршавати — сложеност утиче на перформансе програма, док обрнуто није тачно.

Дефиниција 3.1. Величина улаза је укупан број битова потребних да се репрезентује улаз помоћу одговарајућег бинарног записа. У неким случајевима, величина улаза може бити број објеката (енгл. *item*) на улазу.

Сложеност свих алгоритама, па и тестова прималности се мери користећи нотацију "великог О" величине улаза - записује се као велико O иза кога следи израз који представља неки раст у зависности од величине проблема, означеног словом n. На пример, $O(1)$ — "ред 1", означава да се алгоритам извршава у **константном времену**, $O(\log n)$ — алгоритам се извршава у **логаритамском времену**, обично логаритма са основом 2, $O(n)$ — означава да се алгоритам извршава у **линеарном времену**, тј. да је број операција директно пропорционалан величини проблема, док, например, $O(n \log n)$ — означава да се алгоритам извршава у времену пропорционалном величини проблема и логаритамског времена (енгл. *linearithmic time*).

При томе, например, са $O(n)$ се означава алгоритам чије је очекивано време извршавања пропорционално n, $2n$, или $1000n$. Наравно, обично је много лакше преполовити време извршавања алгоритма $O(n)$, него променити алгоритам који се извршава у $O(n^2)$ да се извршава у $O(n)$.

Алгоритам се извршава у **полиномијалном времену** ако је његово време извршавања ограничено одозго полиномијалним изразом од величине улаза за алгоритам, тј. $O(n^k)$, за неку константу k, док се каже да се алгоритам извршава у **полилогаритамском времену** (енгл. *polylogarithmic time*) ако је $O((\log n)^k)$, за неку константу k. Битно је рећи и да произвољан степен логаритамска функција расте спорије од линеарне функције. На пример, са $O(n^2)$ обележавамо алгоритме који се израчунавају у квадратном времену (енгл. *quadratic time*). Ако је временска сложеност алгоритма полином већег степена, у пракси је такав алгоритам скоро бескорисан за решавање већих проблема. Алгоритам је **експоненцијалан** ако је ограничен са, на пример, $O(2^{n^k})$, за неку константу k, или другачије речено, ако је ограничен са $2^{poly(n)}$, где је $poly(n)$ полином од n. У општем случају, посматрамо експоненцијалне функције са различитом основом, не само са основом 2.

Као што је речено, величина улаза n се обично мери као број битова у бинарној репрезентацији од n, што је $s = \log_2 n$. Ово значи да алгоритам који узима n као улаз и ради n операција, има време извршења $O(n) = O(2^s)$. Види се да је алгоритам који је полиномијалан за n експоненцијалан за s. Каже се да се алгоритам извршава у полиномијалном времену ако је он полиномијалан за s, $s = \log_2 n$.

Анализа најгорег случаја даје процену која је најбоља општа метода за поређење сложености алгоритама. У криптографији је сложеност алгоритма у општем случају (енгл. *average-case complexity*) важнија од сложености најгорег случаја (енгл. *worst-case complexity*) јер је шифровање сигурно ако је одговарајући криптоаналитички проблем тежак у општем случају (или скоро увек тежак), а не само за изоловане класе проблема [28]. С друге стране, ако је дати криптоаналитички проблем лако решити за одређене класе проблема, онда криптографски алгоритам који се заснива на таквом проблему мора предвидети те случајеве, иначе се не може сматрати сигурним.

Како тестови прималности раде са веома великим бројевима, ефикасност често коришћених операција је веома битна. Програми за факторизацију, као и тестови који проверавају прималност великих бројева се могу извршавати сатима, па и данима. У тим случајевима је и оптимизација која убрзава извршавање за само неколико процената веома корисна. Интересантна су питања

[35] За више информација, видети *"Beginning Algorithms"*, S. Harris, J. Ross, Wrox, Indiana, 2006.

ефикасности алгоритама за израчунавање највећег заједничког делитеља и инверза, као и ефикасност алгоритама за извршавање модуларних операција, која се могу побољшати користећи *паметне* алгоритме.

Класе сложености

Проблем одлучивања је проблем за који су могући само одговори "да" и "не". Велики број проблема се може свести на проблеме одлучивања.

Класа сложености P је скуп свих проблема одлучивања који се могу **решити** (енгл. *solve*) у полиномијалном времену помоћу детерминистичке Тјурингове машине. Другим речима, са P означавамо класу детерминистичких полиномијалних алгоритама (енгл. *deterministic polynomial time algorithm*). Ако за неки проблем одлучивања постоји алгоритам полиномијалне сложености, онда каже се да је он **ефикасно решив**.

На пример, израчунавање највећег заједничког делиоца, Јакобијевог симбола, проблем одређивања да ли је број прост, су у P.

Класа сложености NP, скраћено од недетерминистички полиномијални проблем (енгл. *nondeterministic polynomial time problem*), је скуп свих проблема одлучивања који су **проверљиви** (енгл. *verifiable*) у полиномијалном времену помоћу недетерминистичке Тјурингове машине. Проверљивост у полиномијалном времену подразумева да се за неки конкретан улаз за одређени проблем, у полиномијалном времену добија одговор "да". Проверљивост у полиномијалном времену не повлачи и решавање у полиномијалном времену.

Проблем одлучивања за **целобројну факторизацију** (да ли за дате $n, k \in \mathbb{Z}$ постоји $d \in (1, k)$ такав да $d|n$) је у NP.
Очигледно важи да је $P \subseteq NP$, међутим, не зна се да ли је $P = NP$ — то је највећи **нерешен проблем теоријског рачунарства**.

Проблем је **NP-тежак** ако је сваки NP проблем у полиномијалном времену сводив на тај проблем. Проблем је **NP-комплетан** ако припада класи NP и ако је NP-тежак. NP-комплетни проблеми су најтежи проблеми у класи NP.

Како важи да ефикасан алгоритам за неки NP-комплетан проблем постоји ако и само ако за сваки NP-комплетан проблем постоји ефикасан алгоритам, проналажење ефикасног алгоритма за било који NP-комплетан проблем би значило доказ да је $P = NP$.

Поред класа P и NP, постоје и друге класе сложености.

Више о сложености алгоритама може се наћи у [10, 16, 26].

3.2 Еуклидов алгоритам

Уопштено, сви алгоритми за тражење највећег заједничког делиоца користе исту идеју што ефикаснијег редуковања улазних бројева a и b на a' и b', тако да важи $\text{nzd}(a, b) = \text{nzd}(a', b')$. Овај поступак се примењује неколико пута, све док не постане могуће израчунати $\text{nzd}(a', b')$ директно из a' и b'.

Тврђење 3.1. *За целе бројеве a, b важи да је* $\text{nzd}(a, b) = \text{nzd}(a, a - b)$.

Доказ. Ако је $d = \text{nzd}(a, b)$, онда постоје $x, y \in \mathbb{Z}$ такви да је $a = dx$, $b = dy$. Сада је $a - b = dx - dy = d(x - y)$, односно да $d|(a - b)$. Треба показати да је d *највећи* такав делилац. Претпоставимо супротно, да $\exists d_1$ такав да је $d_1 = \text{nzd}(a, a - b)$ и $d_1 > d$. Сада постоје x_1, z такви да је $a = d_1 \cdot x_1$, $a - b = d_1 \cdot z$. Одавде следи да је $d_1 \cdot x_1 - b = d_1 \cdot z$, тј. $b = d_1 \cdot z + d_1 \cdot x_1 = d_1(z + x_1)$, односно да $d_1|b$. Међутим, $d_1|b$, $d_1|a$ и $d_1 > d$ је контрадикција са $d = \text{nzd}(a, b)$. Одавде следи тражено тврђење. □

На основу претходног тврђења написан је следећи програм:

```
unsigned long long nzd (long long a, long long b) {
 if (a < 0) a = -a;
 if (b < 0) b = -b;
 if (a == 0) return b;
 if (b == 0) return a;
 while (a != b) {
  if(a > b) a = a - b;
  else b = b - a;
 }
```

```
 return a;
}
```

Теорема 3.1 (Еуклидов алгоритам). *Ако је* $a = b \cdot q + r$, $0 \leq r < b$, *за неке целе бројеве* $a, b \in \mathbb{Z}$, *онда је* $\mathrm{nzd}(a, b) = \mathrm{nzd}(b, r)$.

Доказ. Ако је $d = \mathrm{nzd}(a, b)$, онда је $a = d \cdot q_1$, $b = d \cdot q_2$ за неке целе бројеве q_1, q_2, односно $r = a - b \cdot q = d(q_1 - q_2 \cdot q)$. Одатле следи да $d|r$, тј. $d = \mathrm{nzd}(b, r)$. Дакле, $d|d_1$, $d_1 = \mathrm{nzd}(b, r)$. Сада важи да је $b = q_3 \cdot d_1$, $r = q_4 \cdot d_1$, $a = d_1(q_3 \cdot q + q_4)$, односно да је $d_1|a$ и $d_1|d$, одакле се добија тражено тврђење. □

Сада треба наћи $d = \mathrm{nzd}(a, b)$, $a, b \in \mathbb{Z}$. Ради једноставности, претпоставимо да су $a, b > 0$. На основу претходне теореме следи:

$$a = b \cdot q_1 + r_1,\ 0 \leq r_1 < b,$$

$$b = r_1 \cdot q_2 + r_2,\ 0 \leq r_2 < r_1,$$

$$r_1 = r_2 \cdot q_3 + r_3,\ 0 \leq r_3 < r_2,$$

$$\ldots$$

$$r_{n-2} = r_{n-1} \cdot q_n + r_n,\ 0 \leq r_n < r_{n-1},$$

$$r_{n-1} = r_n \cdot q_{n+1}.$$

Како је r_n опадајући низ природних бројева, након коначно много корака долази се до једнакости $r_{n-1} = r_n \cdot q_{n+1}$. При томе, $\mathrm{nzd}(a, b) = \mathrm{nzd}(b, r) = \ldots = \mathrm{nzd}(r_{n-1}, r_n)$. Пошто из последње једначине следи да $r_n|r_{n-1}$, односно $r_n = \mathrm{nzd}(r_n, r_{n-1})$, важи да је $r_n = \mathrm{nzd}(a, b)$, $r_n \neq 0$.

Једна од употреба Еуклидовог алгоритма је у решавању линеарних Диофантових једначина $ax + by = c$. Ова једначина је решива за x и y ако $d = \mathrm{nzd}(a, b)$ дели c. Ако се запишу остаци док се Еуклидовим алгоритмом налази $\mathrm{nzd}(a, b)$, може се обрнути поступак и наћи x и y.

Претходна теорема се формулише и на следећи начин:

Теорема 3.2. *Ако је* $a \geq 0$ *и* $b \geq 1$, *онда важи:* $\mathrm{nzd}(a, b) = nzd(b, a \pmod b)$.

```
// rekurzivna funkcija
unsigned long long euklidrec (long long a, long long b) {
 if (a < 0) return euklidrec (-a, b);
 if (b < 0) return euklidrec (a, -b);
 if (b == 0) return a;
 else return euklidrec (b, a%b);
}

// nerekurzivna funkcija
unsigned long long euklid (long long a, long long b) {
 unsigned long long ost;

 if (a < 0) a = -a;
 if (b < 0) b = -b;
 while (b != 0 ) {
   ost = a % b;
   a = b;
   b = ost;
 }
 return a;
}
```

Време извршавања овог алгоритма је полиномијално у односу на величину бинарних репрезентација улазних величина. Сложеност алгоритма се може побољшати употребом брже операције израчунавања остатка по модулу [17, 28].

3.3 Проширени Еуклидов алгоритам

Као што је речено, проширени Еуклидов алгоритам (енгл. *Extended Euclidean algorithm, EEA*), поред израчунавања $\text{nzd}(a, b)$, налази и бројеве x и y такве да важи: $\text{nzd}(a, b) = ax + by$.

АЛГОРИТАМ: Проширени Еуклидов алгоритам

EEA(a,b)
УЛАЗ: два ненегативна цела броја a и b, са условом да је $a \geq b$ (ови услови не умањују општост алгоритма).
ИЗЛАЗ: $\text{nzd}(a, b) = d$ и цели бројеви x и y такви да важи $ax + by = d$.
if $(b = 0)$ **then return** $(a, 1, 0)$;
$x_2 = 1$, $x_1 = 0$, $y_2 = 0$, $y_1 = 1$;
while $(b > 0)$ **do**
  $q = \lfloor a/b \rfloor$, $r = a - q \cdot b$, $x = x_2 - q \cdot x_1$, $y = y_2 - q \cdot y_1$;
  $a = b, b = r, x_2 = x_1$, $x_1 = x$, $y_2 = y_1$, $y_1 = y$;
return (a, x_2, y_2);

Проширени Еуклидов алгоритам асимптотски извршава исти број операција као и Еуклидов алгоритам, осим што враћа додатне информације у сваком позиву.

Ако је $d = \text{nzd}(a, b) = 1$, онда се овај алгоритам може користити за израчунавање инверза броја a по модулу b. Да би се то извело, рачуна се $1 \equiv a \cdot x + b \cdot y \pmod{b}$. Како је $b \cdot y \equiv 0 \pmod{b}$, зато је $a \cdot x \equiv 1 \pmod{b}$ и x је инверз од a по модулу b. Слично се врши израчунавање инверза b по модулу a.

```
// iterativna verzija prosirenog Euklidovog algoritma iz [28]
void eea (long long a, long long b, long long *x,
                          long long *y, unsigned long long *d) {

 long long q, r, x1, x2, y1, y2;

 // uslov a, b >= 0
 if ((a < 0) || (b < 0)) {
   printf ("Unesite nenegativne brojeve a i b za EEA.\n");
   *x = 0, *y = 0, *d = 0;
   return;
 }

 // d = nzd(a , b), ax + by = d
 // kada je d = 1 i b > 0, ostaci y (mod a), x (mod b) su
 // inverzi od b (mod a), odnosno a (mod b)
 if (b == 0) {
   *d = a, *x = 1, *y = 0;
   return;
}

 // pocetne vrednosti promenljivih
 x2 = 1, x1 = 0;
 y2 = 0, y1 = 1;

 while (b > 0) {
   // a, b >= 0, a = b * q + r, 0 <= r < b
   q = a / b, r = a - q * b;
   *x = x2 - q * x1, *y = y2 - q * y1;
   a = b, b = r;
   x2 = x1, x1 = *x, y2 = y1, y1 = *y;
 }

 *d = a, *x = x2, *y = y2;
 return;
}
```

Напомена 3.1. Постоје две опште методе за налажење мултипликативног инверза у коначним пољима: проширени Еуклидов алгоритам и Лагранжева теорема. Такође, на основу Ојлерове теореме следи да се једноставан метод за налажење инверза заснива на чињеници да када $a^{-1} \pmod{m}$ постоји, увек важи једнакост $a^{-1} \pmod{m} = a^{\phi(m)-1} \pmod{m}$.

Постоји аналогни Еуклидов алгоритам за полиноме. Коначна поља, нарочито $\mathbb{GF}(2^n)$, интензивно се користе у криптографији и кодовима за корекцију грешака (енгл. *error correction codes*). На пример, криптографски алгоритам AES израчунава инверзе елемената у $\mathbb{GF}(2^8) = \mathbb{GF}(2)[x]/(x^8 + x^4 + x^3 + x + 1)$. Налажење инверза користи се и у криптографији заснованој на елиптичким кривама. Reed-Solomon и други BCH кôдови за корекцију грешака који се користе у комуникацији и преносу информација, на пример код CD, DVD, дискова, такође користе инверзе елемената у $\mathbb{GF}(2^n)$. Зато је корисно знати ефикасно израчунавање инверза елемената ових поља.

3.4 Бинарни нзд алгоритам

Постоји ефикаснији алгоритам за израчунавање највећег заједничког делитеља. Лемер је 30.их година прошлог века пронашао начин да побољша Еуклидов алгоритам, користећи чињеницу да не захтевају све операције дељења потпуну прецизност (енгл. *full precision*). Silver и Tersian су 1962, и независно од њих Stein 1967. године, предложили бинарни алгоритам за израчунавање $\text{nzd}(a, b)$. Алгоритам је заснован на бинарној репрезентацији бројева, и не извршава дељење (осим дељења са 2).

Концепт: Алгоритам редукује проблем налажења НЗД узастопно примењујући следеће идентитете:

---

1. Ако је $u = 0$, онда је $\text{nzd}(u, v) = v$, јер је нула дељива сваким бројем, и v је највећи број који дели v.
2. Ако је u паран и v паран, онда $\text{nzd}(u, v) = 2 \cdot \text{nzd}(\frac{u}{2}, \frac{v}{2})$, јер је 2 заједнички делитељ.
3. Ако је u паран и v непаран, онда $\text{nzd}(u, v) = \text{nzd}(\frac{u}{2}, v)$, јер 2 није заједнички делитељ.
4. Слично, ако је u непаран и v паран, онда $\text{nzd}(u, v) = \text{nzd}(u, \frac{v}{2})$
5. Ако су и u и v непарни, онда је $|u - v|$ парно и мање од $max\{m, n\}$. Такође, ако су и u и v непарни, из претходног следи да је $\text{nzd}(u, v) = \text{nzd}(\frac{|u-v|}{2}, v) = \text{nzd}(u, \frac{|u-v|}{2})$.

На основу претходног разматрања једноставно је написати ефикасну имплементацију на рачунару, јер се дељење са 2 може извршити једноставним померањем (енгл. *shift*).

```
unsigned long long binarygcd (unsigned long long u, unsigned long long v) {
  unsigned long long k = 0;

  if (u == 0 || v == 0) return u|v;
  if (u == v) return u;

  // sve dok su u i v oba parni
  while ((u & 1) == 0 && (v & 1) == 0) {
   u >>= 1U;              // pomeranje udesno, tj.  deljenje sa 2
   v >>= 1U;
   k ++;                  // dodaje stepen dvojke
  }

  // sada je bar jedan od brojeva  u ili v (ili oba) neparan
  while (u > 0) {
    while ((u & 1) == 0) // u je paran
      u >>= 1U;          // deli u sa 2, sve dok je u paran
    while ((v & 1) == 0) // ako je v paran
      v >>= 1U;          // deli v sa 2, dok je v paran

    if (u >= v)          // oba u i v su neparni,
      u = (u - v) >> 1U; // njihova razlika je paran broj
    else                 // ako je v>u
      v = (v - u) >> 1U;
  }

  return v << k;         // rezultat je v*2^k
}
```

Као што се види, битске операције, нарочито операције битског померања (тј. шифтовања) операнада су основа реализације програма. При **померању у лево** $x << y$ **операнда** x за y места, крајњи десни битови се попуњавају нулама. Померање у лево је еквивалентно множењу

степенима двојке, тј. множењу броја x са 2^y. Код **померања у десно** $x >> y$ **операнда** x за y места, попуњавање крајњих левих битова зависи од типа података и врсте рачунара. Ако су операнди типа unsigned, као што је овде случај, битови се попуњавају нулама. Када су операнди типа signed, код неких рачунара се врши попуњавање јединицама ако је крајњи леви бит јединица, а нулама ако је у крајњем левом биту нула (аритметичко померање), док се на неким рачунарима врши попуњавање нулама (логичко померање). Следи да декларисање аргумента x као unsigned осигурава да се при померању у десно битови попуњавати нулама.

Рекурзивна верзија претходног алгоритма:

```
unsigned long long recursivebinarygcd (unsigned long long u, unsigned long long v) {
 if (u == 0 || v == 0) return u | v;
 if (u == v) return u;

 if((u & 1) == 0)    // u je paran
  if((v & 1) == 0)   // oba u i v su parni
   return (recursivebinarygcd (u >> 1U, v >> 1U) << 1U);
  else               // v je neparan, u je paran
   return recursivebinarygcd (u >> 1U, v);
 if((v & 1) == 0)    // u je neparan, v je paran
  return recursivebinarygcd (u, v >> 1U);

 // oba u i v su neparni
 if(u > v)
  return recursivebinarygcd ((u - v) >> 1U, v);
 return recursivebinarygcd ((v - u) >> 1U, u);
}
```

Иако је алгоритам за бинарни нзд бољи од класичног Еуклидовог алгоритма и његове имплементације ефикасније, асимптотско време извршавања ова два алгоритма је исто. Постоје ефикасније верзије.

3.5 Степеновање поновљеним квадрирањем

Многи тестови прималности и њихове примене укључују израчунавање $a^k \pmod n$, за неке целе бројеве a, k и n. Један начин да се ово постигне је да се a помножи самим собом k пута и онда редукује са n. Међутим, са великим бројевима ово непотребно заузима пуно меморије. Други приступ је множити a самим собом k пута, редукујући са n после сваког множења. Овим избегавамо чување великих бројева, али је сада потребно k множења и k редуковања по модулу.

Боље решење је **бинарно степеновање** или **степеновање поновљеним квадрирањем** (енгл. *repeated squares algorithm, binary ladder exponentiation*) које смањује и број операција множења и операција редукције (тј. операције о модулу) и минимизује количину потребне меморије.

Користећи тврђење 2.3 следи да ако важи $\text{nzd}(a,n) = 1$ онда $x \equiv y \pmod{\phi(n)} \Rightarrow a^x \equiv a^y \pmod n$. Другачије речено, када се извршава операција по модулу n, експоненти се могу свести по модулу $\phi(n)$. Зато, у случају када је $\text{nzd}(a,n) = 1$ и $k \geq \phi(n)$ може се прво извести свођење $k \pmod{\phi(n)}$.

Најважнији корак је да **помоћу бинарне репрезентације броја** k**, посматрамо** k **као збир степена броја 2**: $k = k_0 \cdot 2^0 + k_1 \cdot 2^1 + ... + k_m \cdot 2^m$ тј. $k = k_0 + 2 \cdot (k_1 + 2 \cdot (k_3 + 2 \cdot (\cdots + 2 \cdot k_m)))$, где су коефицијенти k_i једнаки 0 или 1. Користећи овај нови израз за k, следи да је a^k производ само $O(\log k)$ вредности, уместо k вредности. Свака вредност је a на неки степен двојке, па те вредности израчунавамо узастопним квадрирањем броја a. На овај начин је број потребних множења бити знатно смањен. Ова техника се примењује за ефикасно израчунавање $a^k \pmod n$ где се **вредности редукују по модулу** n **у сваком кораку**, да би остале релативно мале.

Следи да бинарно степеновање захтева $O(\log_2 k)$ операција квадрирања и $O(\log_2 k)$ операција множења, и при сваком квадрирању и множењу извршава редукцију по модулу n.

```
unsigned long long binary_exp(unsigned long long x,
         unsigned long long k, unsigned long long n){
  unsigned long long d, r=1;

  d=x % n;              // za slucaj da je x > n
  // k = k_0*2^0 + k_1*2^1 + ... + k_m*2^m
  while (k) {           // sve dok je k!=0
```

```
    if (k & 1) {       // ako je k neparno, tj. k_i == 1
     r = (r * d) % n; // tekuci rezultat je x^(k_0 + .. + k_i*2^i) (mod n)
     // k --;
    }
   // if (k > 1) {
   // u svakom koraku izracunava x^2^i = x^2^(i-1) * x^2^(i-1) (mod n)
     d = (d * d) % n;
     k >>= 1U;          // deli k sa 2
   // }
  }
  return r;
}
```

3.6 Јакобијев симбол

Ефикасно израчунавање Јакобијевог симбола $(\frac{a}{n})$ је важна компонента Соловеј-Штрасеновог теста прималности, као и многих других алгебарских алгоритама, на пример за израчунавање квадратних корена по модулу [17]. Алгоритми за израчунавање Јакобијевог симбола имплементирани су и у системима симболичке алгебре као што су Mathematica и Maple.

За Јакобијев симбол важи $(\frac{m}{n})(\frac{n}{m}) = (-1)^{\frac{m-1}{2}\cdot\frac{n-1}{2}}$. Као и код Лежандровог симбола, $(\frac{m}{n}) = (\frac{n}{m})$, осим ако су и m и n конгруентни 3 по модулу 4, и у том случају је $(\frac{m}{n}) = -(\frac{n}{m})$.

На основу особина Јакобијевог симбола следи да ако је n непаран и $a = 2^k \cdot a_1$, где је a_1 непаран, онда је:

$$(\tfrac{a}{n}) \equiv (\tfrac{2^k}{n})(\tfrac{a_1}{n}) = (\tfrac{2}{n})^k(\tfrac{n \pmod{a_1}}{a_1})(-1)^{(a_1-1)(n-1)/4}$$

Ово запажање води до следећег рекурзивног алгоритма за израчунавање $(\frac{a}{n})$, који не захтева факторизацију броја n.

АЛГОРИТАМ: Израчунавање Јакобијевог (и Лежандровог) симбола

---

jacobi(a,n)

УЛАЗ: непаран цео број $n \geq 3$, и цео број a, $0 \leq a \leq n$.

ИЗЛАЗ: Јакобијев симбол $(\frac{a}{n})$ (и самим тим Лежандров симбол када је n прост).

if $(a = 0)$ **return** (0);

if $(a = 1)$ **return** (1);

Записати $a = 2^k \cdot a_1$, где је a_1 непаран број.

if (k паран) $s = 1$;

else // к је непарно

  if $(n \equiv 1 \pmod 8)$ **or** $n \equiv 7 \pmod 8)$ $s = 1$;

  if $(n \equiv 3 \pmod 8)$ **or** $n \equiv 5 \pmod 8)$ $s = -1$;

if $(n \equiv 3 \pmod 4)$ **and** $a_1 \equiv 3 \pmod 4)$ $s = -1$; // Гаусов закон квадратног реципроцитета

$n_1 = n \pmod{a_1}$;

if $(a_1 = 1)$ **return** (s);

else return $(s \cdot jacobi(n_1, a_1))$;

Време извршавања овог алгоритма је полиномијално у односу на величину бинарних репрезентација улазних величина.

Напомена 3.2. Коректност алгоритма следи на основу наведених особина и теорема. На пример, користи се чињеница да је $(\frac{a^2}{p}) = 1$, као и теорема 2.19, и Гаусов закон квадратног реципроцитета, док се у последњем делу користи особина да је $(\frac{1}{n}) = 1$. За више информација видети [17, 28].

Приметимо да ако се на почетку стави да је $s = 1$, s се мења само ако је k непарно и $n \equiv 3 \pmod 8$ или $n \equiv 5 \pmod 8$, и ако важи одређени услов Гаусовог закона квадратног реципреоцитета. Следи кôд функције у C-у која израчунава Јакобијев симбол, засноване на датом алгоритму и алгоритму за израчунавање Јакобијевог симбола у библиотеци BigInteger за програмски језик Јава.[36]

[36] Видети библиотеку за програмски језик C чији је аутор Colin Plumb, као и [17].

```
long long jacobi (long long a, long long n) {
    long long nmod, k, s = 1;

    if (a == 0) return 0;
    if (a == 1 || n == 1) return 1;

   /**
    * a = 2^k * a_1, a_1 neparan ceo broj
    * nema potrebe da se pamti k, jer se s menja (s = -s) samo ako je
    * k neparan i nmod = 3 ili 5
    * s = ((k % 2 == 0) || nmod == 1 || nmod == 7) ? 1 : -1;
    */

    // sve dok je a deljivo sa 4, deli a sa 4, k je paran
    while ((a & 3) == 0)
       a >>= 2;

    // ako je a deljivo sa 2, podeli a sa 2, k je neparan
    if((a & 1) == 0) {
       a >>= 1;
       // ostatak po modulu 8
       nmod = n & 7;
       if(nmod == 3 || nmod == 5)
         s = -s;
    }

    /**
     * Gausov zakon kvadratnog reciprociteta
     * if ((n % 4 == 3) && (a % 4 == 3) ) s = -s;
     * kako su a i n neaparni, vazi: 1 & 1 & 2 = 0, 1 & 3 & 2 = 0, 3 & 3 & 2 = 2
     */

    if ((n & a & 2) != 0) // n = a = 3 (mod 4)
      s = -s;

    return ((a == 1) ? s : s * jacobi (n % a, a));
}
```

Могуће је адаптирати алгоритам бинарни нзд да израчунава Јакобијев симбол.

3.7 Поновљено дељење

Ради провере да ли је цео број d делитељ броја n може се користити операција свођења по модулу, тј. проверити да ли је $n \equiv 0 \pmod{d}$. Следећа теорема је основно тврђење које се користи код поновљеног дељења и Ератостеновог сита:

Теорема 3.3. *Ако не постоји цео број d такав да је $1 < d \leq \sqrt{n}$ и d је делитељ броја n, онда је n прост број.*

Доказ. Претпоставимо супротно, да n нема делитељ d такав да је $1 < d \leq \sqrt{n}$ и да је n сложен број. Тада према претпоставци, n има делитељ a такав да је $\sqrt{n} < a < n$, па се цео број n може записати као $n = a \cdot b$, за неки цео број b такав да је $\sqrt{n} < b < n$. Међутим, одавде следи да је $a \cdot b > \sqrt{n} \cdot \sqrt{n} = n$, што је у супротности са чињеницом да је $n = a \cdot b$. Зато следи да је претпоставка нетачна, па n мора бити прост број. □

Поновљено дељење или **узастопно пробно дељење** (енгл. *trial division*) је метод узастопне провере да ли су бројеви до $\sqrt{n}$ делитељи броја n, са циљем да се провери прималност броја n или да се потпуно или делимично факторише број n.

Када се поновљеним дељењем проверава да ли је број n прост, један начин може бити провера да ли је n дељив бројем 2 и свим непарним бројевима до $\sqrt{n}$. Даље, уочава се да је довољно проверити да ли је цео број n дељив са 2, са 3, а онда само проверавати бројеве који су еквивалентни 1 или 5 по модулу 6. Ова оптимизација може се наставити елиминисањем бројева чији су делитељи 5, 7, итд.

```
unsigned prost(unsigned long long n) {
 unsigned long long i, d, sqrtn;
```

```
 sqrtn = sqrt(n);
 d = 4; // d naizmenicno dobija vrednosti 2 i 4

 if (n == 2 || n == 3) return 1;
 if (n % 2 == 0 || n % 3 == 0) {
  printf ("Dati broj je deljiv brojem 2 ili 3.\n");
  return 0;
 }

 for (i = 5; i <= sqrtn; i += d) {
  if (n % i == 0) {
   printf("Broj je deljiv sa %llu.\n", i);
   return 0;
  }

  d = 6 - d;
 }

 return 1;
}
```

Ипак, време извршавања овог алгоритама је експоненцијално. Одређено побољшање се постиже ако се проверава да ли су само прости бројеви до $\sqrt{n}$ делитељи броја n.

Уводи се појам **точка** (енгл. ***wheel***) који се користи и код просејавања простих бројева. Како су сви прости бројеви већи од броја 3 једнаки 1 или 3 по модулу 6, наизменичним додавањем 2 и 4 пролази се кроз све **кандидате за просте бројеве**. Како је $\phi(2 \cdot 3 \cdot 5) = 8$, следи да је 8 бројева мањих од 30 узајамно просто са 30. У овом сведеном систему остатака су бројеви $1, 7, 11, 13, 17, 19, 23, 29$, одакле следи да се сви прости бројеви могу наћи у некој од 8 класа бројева који су по модулу 30 једнаки неком од наведених бројевима. Јасно је да је точак који пролази кроз бројеве овог облика веће од 7 једнак: $4, 2, 4, 2, 4, 6, 2, 6$, па се овим точком пролази кроз бројеве $7, 11, 13, 17, 19, 23, 29, 31, 37$, итд. Одавде следи и да су у 8 бита, односно у 1 бајту, записане информације о сложености 30 целих бројева. Слично, како је $\phi(2310) = \phi(2) \cdot \phi(3) \cdot \phi(5) \cdot \phi(7) \cdot \phi(11) = 480$, закључује се да 60 бајта могу чувати информације о 2310 целих бројева, односно да 1 бит носи информације за 4.8 броја. Може се приметити и да до 2310 има 175 простих бројева.

Међутим, *точкови* брзо постају веома компликовани, и самим тим непрактични. Занимљиво је поменути да само нешто више од 50% свих бројева узајамно простих са 2 и 3 има прост фактор мањи од 30, одакле следи да није ефикасно правити точак који пролази кроз бројеве који имају ово својство, јер је овај точак величине $\phi(2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23 \cdot 29) = 1,021,870,080$.

Потребно је прво **направити листу простих бројева до корена броја кога треба тестирати да ли је прост** (или кога треба факторисати).

Поновљено дељење може бити корисно за доказивање да је неки број који има до 19 цифара прост, али за веће бројеве убрзо постаје превише споро.

Најгори случај је када је n прост број, јер тада треба делити са свим простим бројевима до $\sqrt{n}$. Из Основне теореме о простим бројевима следи да је број операција дељења у овом случају око $2 \cdot \frac{\sqrt{n}}{\ln n}$. Треба имати на уму да је проблем препознавања простих бројева у општем случају доста лакши од проблема факторизације.

Када се поновљено дељење користи као метод факторизације, почиње са са првим простим бројем, бројем 2, и дели се са 2 све док је n дељив са 2, а онда се прелази на следећи прост број, број 3, и **понавља поступак за нефакторисани део** броја n. Поступак се завршава када се прође кроз све просте бројеве до корена из нефакторисаног дела јер је тада нефакторисани део прост број.

3.8 Ератостеново сито

У многим случајевима може бити веома корисно познавање свих простих бројева из одређеног интервала. Као што је наглашено, листа простих бројева који треба да се провере као делитељи чини метод доказивања простости помоћу поновљеног дељења ефикаснијим.
Ератостеново сито[37] је једноставан алгоритам за генерисање простих бројева до задатог броја n.

[37] Ератостен (Eratosthenes), око 276.-194. п.н.е., старогрчки математичар, географ, путописац и астроном.

Концепт: Ератостеново сито

eratosten (n)

УЛАЗ: ненегативан цео број n.

ИЗЛАЗ: сви прости бројеви мањи од n.

1. Запишу се сви бројеви од 2 до n.

2. Почевши од првог броја на списку (двојка) прецртају се са списка сви бројеви дељиви са два и упише се да је двојка прост број.

3. Понавља се поступак са следећим непрецртаним бројем m. Дакле, прецртају се сви бројеви дељиви са m, **полазећи од m^2 до** n, а сам број m се обележи да је прост.

4. Поступак се завршава када је **прецртан сваки сложен број — умножак неког простог броја до** $\sqrt{n}$.

Следи једноставан C програм који илуструје идеју Ератостеновог сита:

```c
#include <stdio.h>
#include <math.h>
#include <malloc.h>

#define MAX 100000000

int main (int argc, char *argv[]) {
  FILE *fajl;
  char *name = "baza.txt";
  unsigned long max;
  unsigned long i, j;
  unsigned long sqrtn, count = 0;
  unsigned char *brojevi;
  clock_t begin;

  if (argc > 1)
    max = strtoul(argv[1], NULL, 0);
  else
    max = MAX;

  brojevi = (unsigned char*) calloc (max + 1, sizeof(unsigned char));
  if (!brojevi) {
    printf ("Ne moze da alocira dovoljno memorije.\n");
    return 1;
  }

  sqrtn = sqrt(max);

  for (i = 0; i <= max; i++)
    brojevi[i] = 1;

  begin = clock();
  for (i = 2; i <= sqrtn; i++)
     if (brojevi[i] != 0)                    // ako i nije markiran kao slozen
        for (j = (i * i); j <= max; j+=i)    // krece od i^2 do max
          if (brojevi[j] != 0)               // ako broj nije vec oznacen kao slozen
            if ((j % i) == 0)
              brojevi[j] = 0;

  if ((fajl = fopen (name, "w")) == NULL) {
   printf ("Ne moze da otvori fajl.\n");
   return 1;
  }

  for (i = 2; i <= max; i++) {              // preskace 0 i 1
    if (brojevi[i] != 0) {
     count++;
     fprintf (fajl, "%lu,", i);
    }
  }

  free(brojevi);
```

```
  fclose (fajl);
  printf ("Prostih brojeva do %lu ima: %lu. Prosti brojevi su nadjeni za
         %.6f sekundi.\n", max, count, (double)(clock() - begin)/CLOCKS_PER_SEC);
  return 0;
}
```

Сложеност Ератостеновог сита је $O(n \cdot \ln \ln n)$, и n је експоненцијално по броју цифара. Највеће практично ограничење свих сита је огромна количина меморије и простора на диску коју захтевају, па треба смањити ове захтеве. Често је потребно поделити на сегменте бројеве од 2 до n. Али да би Ератостеново сито било ефикасно, дужина сегмента не треба да буде мања од $\sqrt{n}$. На овај начин се креира тзв. **сегментисано сито** (енгл. *segmented sieve*).

Ако се са 1 у датом низу означи да је број сложен, онда се сито веома једноставно може променити тако да уместо 1 чува најмањи прости фактор сваког сложеног броја. Овакво сито заузима више меморијског простора. На сличан начин се прави сито које за сложене бројеве даје потпуну факторизацију.

3.9 Модификовано Ератостеново сито

Класично Ератостеново сито почиње од бројева $2, 3, ..., n$ и сукцесивно одбацује оне који су производ простих бројева до $\sqrt{n}$. Овде је дат опис модификованог Ератостеновог сита које је брже од класичног, користи мање меморије и може да ради са много већим бројевима. Ово модификовано Ератостеново сито је један од резултата семинара и консултација које је 2009. године одржао проф. Ж. Мијајловић.

Неколико битних особина овог сита и побољшања које оно доноси:

- **Својство целог броја да ли је прост или сложен, на рачунару може бити репрезентовано једним битом** — 0 или 1. На пример, 0 означава прост број, тј. број који није прецртан током просејавања. Сваки рачунар има ограничену количину меморије, па је потребно да пронађи начине за њено *рационално* коришћење.

- **За просејавање се користе само прости бројеви**. Прости бројеви до $\sqrt{n}$, n број до кога се извршава просејавање, су генерисани (нпр. помоћу претходог сита) и сачувани као низ.

- Како је број 2 једини паран прост број, не само да **се парни бројеви не проверавају, него се ни не записују**. На пример, на овај начин се постиже да се у сваком бајту чувају информације који бројеви из интервала дужине 16 су прости.

- У општем случају при означавању бројева дељивих са m, **полазећи од m^2 обележава се као сложен сваки $(2m)$-ти број**, тј. m^2, $m^2 + 2m$, $m^2 + 4m$, ..., до n. Како је m непарно, сви ови бројеви су непарни. Остали бројеви дељиви са m су парни, па се зато не разматрају.

- $\phi(2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19) = \phi(9699690) = 1 \cdot 2 \cdot 4 \cdot 6 \cdot 10 \cdot 12 \cdot 16 \cdot 18 = 1,658,880$ - оволико је бројева који су узајамно прости са $9,699,690$, тј. **број елемената сведеног система остатака**. Међу овим бројевима су сви прости бројеви из овог интервала, а остали бројеви су сигурно сложени. Следи да 1,658,880 бита, односно 207,360 бајта може носити информације о свим бројевима из интервала од $9,699,690 \approx 10^7$ бројева (тј. каже се да 1 бит чува информације за приближно 5.85 бројева, што преставља побољшање). Приметимо да простих бројева до $9,699,690$ има 646,029.

- Конструише се низ mbiti који има $9,699,690$ елемената и за индексе који не припадају сведеном систему елемената има вредност 0, док за елементе сведеног система даје који је то остатак по реду, односно чува позицију сведеног остатка који је означен индексом низа. Помоћу низа mbiti проверава се да ли је број кандидат или не, тј. **да ли је елемент сведеног система елемената** — ако је вредност *mbiti* неког остатка 0, он је сигурно сложен број. На овај начин се омогућава и чување информација само о бројевима из сведеног система остатака.

- Примењује се модификација која се односи на облик бројева који се проверавају. Као што је на почетку речено, а код алгоритма поновљеног дељења примењено, **сви прости бројеви**

су облика $6k+1$ **или** $6k+5$. Треба приметити да су квадрати бројева облика $6k+1$ и $6k+5$ оба облика $6k+1$.

- Ово сито је **сегментисано**: интервал $[1, n]$, који означава бројеве на које се примењује сито, се дели на сегменте дужине $PBLOK = 9,699,690$. У сваком сегменту, све операције се извршавају по модулу. После сваког сегмента записује се на диск низ, назовимо га marker, који чува информације о простим бројевима из тог сегмента. Сито је најефикасније када је $n \leq 9699690^2$.
- **Елиминише се операција по модулу**, тј. остатак при дељењењу, колико год је могуће.
- **Припрема** од ког броја у сваком сегменту почиње просејавање заузима битан део програма. У овом делу сито се припрема тако што се примењују сви претходни захтеви. Ова израчунавања и многе провере могу се редуковати ако се узме да је број N до кога се просејава умножак сегмента $PBLOK$.
- За приступ битовима који чувају информације о простим бројевима, ради бројања или генерисања простих бројева, користи се низ **који садржи размаке** (енгл. *gap*) између кандидата за просте бројеве.

На основу претходних запажања написан је програм чији се кôд може наћи у прилогу уз рад.

Ератостеново сито на интервалу

Претпоставимо да је потребно наћи све просте бројеве из интервала $[n, m]$, где су n и m велики прости бројеви, али је њихова разлика релативно мала. Да би се смањила количина потребне меморије, примењује се следећи поступак: прво се уз помоћ Ератостеновог сита генерише листа свих простих бројева од 2 до $\sqrt{m}$. Ти прости бројеви се ставе у низ, тако да тај низ садржи само просте бројеве. Тада сваки број између n и m који је сложен мора имати фактор из ове листе, па се једноставно може просејати дати интервал простим бројевима из ове листе.

Овај алгоритам има неколико предности над стандардним ситом за велики цео број m. Меморија коју заузима је смањена јер када се једном добију сви прости бројеви до $\sqrt{m}$, они се сабију у низ дужине $\pi(\sqrt{m})$. Такође, други део алгоритма захтева низ величине $m-n$ уместо дужине m. Време извршавања је такође краће јер се проверава мање бројева, односно мање умножака сваког простог броја. Међутим, и овде постоји проблем са оптималном величином интервала за просејавање [17].

4 Тестови простости

Тестови простости, тј. тестови да ли је задати број прост, су веома битни за криптографију. У последње време дошло је до значајних открића у овој области и побољшања постојећих алгоритама.

Вероватносни тест простости је тест којим се произвољни позитивни цели бројеви тестирају да би се обезбедиле *делимичне* информације у погледу њихове простости. Они не доказују да је дати број прост. Уместо тога, они класификују цео број на улазу као сложен или **вероватно прост**.

За сваки непаран позитиван цео број n, скуп $W(n) \subset \mathbb{Z}_n$ је дефинисан тако да задовољава следеће особине:

1. За дато $a \in \mathbb{Z}_n$ може се у детерминистичком полиномијалном времену одредити да ли $a \in W(n)$;
2. Ако је n прост број, тада је $W(n) = \emptyset$, $W(n)$ је празан скуп;
3. Ако је n сложен број, онда је $|W(n)| \geq \frac{n}{2}$.

Дефиниција 4.1. Ако је n сложен број, елементе скупа $W(n)$ називамо **сведоцима сложености** броја n, а елементе комплементарног скупа $L(n) = \mathbb{Z}_n - W(n)$ називамо **лажовима** [28].

Вероватносни тестови покушавају да нађу доказ, тј. сведоке да је број сложен, и враћају да је број вероватно прост ако сведоци нису нађени.

Вероватноћа грешке да ако се тест изведе k пута независно за сложен број n, и буде проглашен као "прост" свих k пута је највише $(\frac{1}{2})^k$.

Основна идеја свих вероватносних тестова простости је провера неке особине коју имају сви прости бројеви, а већина сложених не задовољава овај услов. Вероватносни тестови прималности могу се комбиновати ради креирања веома брзог алгоритма за показивање прималности бројева. Заправо, уобичајни начин провере прималности неког броја заснива на провери да ли је број дељив малим простим бројевима до одређене границе, па се тек онда примењује тест прималности, или комбинација два или више (међусобно независних) вероватносних тестова прималности, ради креирања бржег и сигурнијег теста.

Прави тест простости је тест за доказивање да ли задати број прост.

Мотивација за проналазење доказивих простих бројева. Ако је потребно користити просте бројеве за *индустријску* примену често не треба доказати да су они прости. Може бити довољно да је вероватноћа да они могу да се расставе веома мала (нпр. мања од $10^{-24}\%$). У овом случају могу се користити (јаки) вероватносни тестови прималности.

Ипак, када се нађе *вероватно прост* број, уводи се мала вероватноћа да је тај број сложен. У зависности од даље примене таквог простог броја, могу се десити озбиљни проблеми. Као у претходном примеру, одређени криптографски алгоритми се ослањају на специјалне особине простих бројева, па ако се изабере сложен број, криптосистем може бити озбиљно ослабљен. Иако је могућност оваквог неуспеха екстремно мала, она ипак постоји, и зато је интересантно размотрити алгоритме за проналажење доказивих простих бројева.

Тестове простости треба разликовати од алгоритама за факторизацију бројева на просте чиниоце, јер они у општем случају не факторишу бројеве, већ (само) дају одговор на питање да ли је број прост или не.

Како се у одређеним тестовима прималности захтева делимична или потпуна факторизација неког броја, тј. прималност траженог броја се заснива на познавању факторизације неког *повезаног* броја, чија факторизација је очигледна или позната, или је лако факторисати тај број због његових специфичних особина, прво је дат преглед метода за факторизацију.

4.1 Факторизација

Проблем одлучивања да ли је број прост или не је у општем случају лакши од проблема факторизације. Зато се прво може тестирати да ли је број заиста сложен, па тек онда покушати са извршавањем његове факторизације.

За проналажење *малих простих бројева* користити се Ератостеново сито или поновљено дељење. Ове методе су сигурне, и најбоље методе за мале бројеве. Овде се под малим простим бројевима обично сматрају они мањи од 2^{64}. Али заправо, овај термин зависи од контекста. Ако смо фокусирни на проналажење простих бројева са више од милион цифара, сматрамо да су прости бројеви са пар стотина цифара мали, а на ове бројеве је неефикасно примењивати поменуте алгоритме.

Поновљеним дељењем се проверава да ли је број n дељив неким простим бројем од 2 до $\sqrt{n}$. Овај алгоритам даје и делимичну или потпуну факторизацију броја n ако је n сложен — када се пронађе делитељ i, онда се једноставно може рестартовати алгоритам за $\frac{n}{i}$ док се у потпуности не факторише број n.

Методе факторизације се могу поделити на опште и специјалне. Код општих метода очекивани број операција зависи само од величине броја n, док код специјалних зависи и од особина фактора броја n. Код оцене сигурности криптосистема заснованих на проблему факторизације битне су опште методе.

Специјалне методе:

1. Полардов ро алгоритам за факторизацију (енгл. *Pollard's rho factoring algorithm*).
2. Полардова $p-1$ метода за факторизацију (енгл. *Pollard's $p-1$ factoring algorithm*).
3. Факторисање помоћу елиптичких кривих (енгл. *elliptic curve factoring, ECM*).
4. Специјално сито у пољу бројева (енгл. *special number field sieve, SNFS*).

Опште методе:

1. Квадратно сито (енгл. *quadratic sieve factoring, QS*).
2. Сито у пољу бројева (енгл. *number field sieve factoring, NFS*), тј. његова варијанта — опште сито у пољу бројева (енгл. *general number field sieve, GNFS*) је тренутно најбољи алгоритам за факторизацију.

У време писања овог рада највећи факторисани RSA број је RSA-768 са 232 цифара (768 битова) факторисан помоћу GNFS.[38]
Сигурност многих криптографских техника зависи од тежине проблема факторизације. Проблем RSA и проблем тражења квадратног остатка повезани су са проблемом факторизације. Неки алгоритми за факторизацију који су експоненцијални и субекспоненциални чине основу за аналогне алгоритме за израчунавање дискретног логаритма.

Fancy Ератостеново сито

Као илустрацију једне једноставне методе за факторизацију, где су примењена основна знања из теорије бројева, дат је алгоритам *Fancy* Ератостеновог сита из [17]:

АЛГОРИТАМ: *Fancy* Ератостеново сито

---

fancy-eratosthenes-sieve(N)

УЛАЗ: Цео број $N \geq 4$.

ИЗЛАЗ: Скуп простих бројева у $[1, N]$.

Са p_l се означава l-ти прост број. Нека је $M_l = p_1 \cdot p_2 \cdot ... \cdot p_l$, и нека је са S_l означен скуп бројева у $[1,\ N]$ који су узајамно прости са M_l. Приметимо да ако је $p_{m+1} > \sqrt{N}$, онда је скуп простих бројева у $[1,\ N]$ једнак $(S_m)\backslash\{1\} \cup \{p_1,\ p_2, ...,\ p_m\}$. Алгоритам рекурзивно проналази $S_k, S_{k+1}, ..., S_m$, тако што почиње од вредности k која је умерене величине и завршава са $m = \pi(\sqrt{N})$.

[Setup]

Поставити k на цео број такав да је $M_k \leq \frac{N}{\ln N} < M_{k+1}$;

[38] RSA Factoring Challenge је било такмичење за факторисање великих целих бројева и разбијање RSA кључева, завршено 2007. године.

$m = \pi(\sqrt{N})$;

Користи се обично Ератостеново сито да се нађу прости бројеви $p_1, p_2, ..., p_k$ и да се нађе скуп целих бројева у $[1,\ M_k]$ узајамно простих са M_k (тј. сведени скуп остатака по модулу M_k);

[Покренути точак]

Покренути M_k "точак" да се нађе скуп S_k;

$S = S_k$.

[Пронађи размаке (енгл. gaps)]

while $(l \in [k+1,\ m])$ **do**

$p = p_l$ = најмањи члан скупа S који је већи од 1; У овом тренутку, $S = S_{l-1}$.

Пронаћи скуп G размака између узастопних чланова скупа $S \cap [1, N/p]$;

Сваки број који је размак се броји само једном у G.

Наћи скуп $pG = \{p \cdot g : g \in G\}$;

[Пронаћи специјални скуп]

Пронаћи скуп $pS \cap [1, N] = \{p \cdot s : p \cdot s \leq N, s \in S\}$ на следећи начин: Ако су s и s' узастопни чланови скупа S такви да је $s' \cdot p \leq N$ и $s \cdot p$ је већ израчунат, онда је $p \cdot s' = p \cdot s + p \cdot (s' - s)$;

Треба приметити да је $s' - s$ члан скупа G и да је број $p \cdot (s' - s)$ већ израчунат у кораку [Пронаћи размаке]. Дакле, $p \cdot s'$ се може израчунати помоћу одузимања (да се нађе $s' - s$), налажењем $(p \cdot (s' - s))$ и онда сабирањем. (Како је најмањи члан скупа S број 1, прва вредност за $p \cdot s$ је сам број s и није потребно никакво израчунавање.)

[Пронаћи следећи скуп S]

$S = S \backslash (pS \cap [1, N])$;

$l = l + 1$;

[Вратити скуп простих бројева у [1,N]]

return $(S \backslash \{1\}) \cup \{p_1, p_2, ..., p_m\}$;

Сваки скуп S_l се састоји од бројева у $[1, N]$ који су узајамно прости са $p_1, p_2, ..., p_l$. Одавде следи да је први члан после 1 заправо $(l+1)$-ти прост број, p_{l+1}.

И ово сито, као и претходно поменута сита, може послужити за **генерисање простих бројева** из одређеног интервала.

Дефиниција 4.2. Позитиван цео број x је **N-гладак** (енгл. *N-smooth*) ако су сви његови прости фактори мањи од N.

Глатки бројеви или бројеви без великих простих фактора су веома битни. Поменимо да у просеку има више од 30% бројева из интервала $[1, N^2]$ који су N-глатки [17].

Описана сита и поновљено дељење се могу модификовати да проналазе глатке бројеве из неког интервала, или да обезбеде делимичну факторизацију неког броја, односно прављење табеле најмањих простих фактора бројева из неког интервала, што се онда користи у одређеним тестовима прималности или другим алгоритмима из теорије бројева.

4.2 Фермаов тест простости

Фермаов тест је једноставан вероватносни тест прималности заснован на малој Фермаовој теореми.

Као што смо већ показали, ако је p прост број, онда важи $2^{p-1} \equiv 1 \pmod{p}$. Нажалост, постоје и сложени бројеви који задовољавају ово тврђење. На пример, $n = 341 = 11 \cdot 31$ је сложен број и важи $2^{341-1} \equiv 1 \pmod{341}$.

Дефиниција 4.3. Нека је n непаран *сложен* број. Цео број a, $a \in [1, n-1]$, такав да важи $a^{n-1} \neq 1 \pmod{n}$ назива се **Фермаовим сведоком сложености** броја n.

Дефиниција 4.4. Фермаов псеудопрост број у бази a је *сложен* цео број n који задовољава конгруенцију $a^{n-1} \equiv 1 \pmod{n}$. Цео број a се назива **Фермаовим лажовом** за n.

Када не би постојали псеудопрости бројеви за базу a, онда би мала Фермаова теорема била једноставан тест прималности. Ови псеудопрости бројеви су прилично ретки, на пример, 341 је најмањи псеудопрост у бази 2, и постоји само 22 псеудопроста броја у бази 2 у првих 10^5 целих бројева. Међутим, највећа мана Фермаовог теста је то што постоји већа вероватноћа да специјални сложени бројеви, тзв. Кармајклови бројеви (енгл. *Carmichael numbers*) буду проглашени као вероватно прости, него што је то вероватноћа за остале сложене бројеве.

Дефиниција 4.5. Кармајклов број је сложен цео број n који задовољава $a^{n-1} \equiv 1 \pmod{n}$ за сваки број a такав да је $\text{nzd}(a, n) = 1$.

Најмањи Кармајклов број је $561 = 3 \cdot 11 \cdot 17$. Сваки Кармајклов број је производ најмање три различита проста броја и безквадратан је број. Кармајклови бројеви су још ређи од псеудопростих у бази 2: има их само 255 у првих 10^8 целих бројева [17].

Фермаов тест проверава да ли је n псеудопрост у k случајно изабраних база, и класификује га као вероватно прост ако прође свих k тестова. Намеће се једноставан начин смањивања грешке у класификацији тако што се задати број провери за више различитих база, тј. за већи број итерација k. На пример, $341 = 11 \cdot 31$ је псеудопрост у бази 2, али није у бази 3.

АЛГОРИТАМ: Фермаов вероватносни тест прималности

fermat(n, k)
УЛАЗ: Непаран цео број $n \geq 3$ и параметар $k \geq 1$.
ИЗЛАЗ: Одговор "прост" или "сложен" на питање: "Да ли је n прост број?"

for (i = 1 **to** k) **do**
    Изабрати случајан цео број a, $a \in [2, n-2]$;
    Израчунати $r = a^{n-1} \pmod{n}$; // може се користити алгоритам binary_exp дат у поглављу 3
    if $(r \neq 1)$ **return** ("сложен");
return ("прост");

Напомена 4.1. Као што смо рекли, постоје бројеви који су псеудопрости за скоро сваку базу a. Ако је n Кармајклов број, онда су једини Фермаови сведоци за n цели бројеви a, $a \in [1, n-1]$ за које је $\text{nzd}(a, n) > 1$. На пример, за базу 3, Кармајклов број 561 је на основу датог Фермаовог теста сложен број.

Фермаов тест је и веома ефикасан алгоритам. Али, нас занимају вероватносни тестови који су подједнако поуздани за све бројеве на улазу.

Напомена 4.2. Другим пробабилистичким тестовима простости одговарају други типови псеудопростих бројева.

4.3 Соловеј-Штрасенов тест простости

Соловеј и Штрасен су 1977. године развили вероватносни тест који нема слабости Фермаовог теста. Он је био фундаменталан за стварање криптосистема RSA. Иако га је Милер-Рабинов тест заменио на месту најбољег вероватносног теста прималности, наводи се због значаја који је имао.

Као и Фермаов тест, Соловеј-Штрасенов тест (енгл. *Solovay-Strassen test*) је заснован на једноставном тесту сложености из теорије бројева — на Ојлеровом критеријуму.

Теорема 4.1. *Ако је n непаран прост број, онда је $(\frac{a}{n}) \equiv a^{\frac{n-1}{2}} \pmod{n}$ за свако a такво да је $1 \leq a \leq n-1$. Општије, $(\frac{a}{n}) \equiv a^{\frac{n-1}{2}} \pmod{n}$ за све целе бројеве a такве да је $\text{nzd}(a, n) = 1$.*

Дефиниција 4.6. Нека је n непаран сложен цео број и $1 \leq a \leq n-1$. Ако је $\text{nzd}(a, n) > 1$ или $a^{\frac{n-1}{2}} \neq (\frac{a}{n}) \pmod{n}$, онда је a **Ојлеров сведок сложености** броја n. У супротном, ако је $\text{nzd}(a, n) = 1$ и $a^{\frac{n-1}{2}} \equiv (\frac{a}{n}) \pmod{n}$, онда је a **Ојлеров лажов** за n.

АЛГОРИТАМ: Соловеј-Штрасенов вероватносни тест прималности

solovay-strassen(n, k)
УЛАЗ: Два цела броја n и k, $n \geq 3$, $k \geq 1$.
ИЗЛАЗ: Одговор "тачно" или "нетачно" на питање: "Да ли је n прост број?"

for (i = 1 **to** k) **do**
    Изабрати случајан цео број a, $a \in [2, 2-n]$;
    Израчунати $r = a^{\frac{n-1}{2}} \pmod{n}$; // нпр. алгоритмом датим у поглављу 3

if $(r \neq 1$ **and** $r \neq n-1)$ **return** ("сложен"); // тј. $r \neq \pm 1 \pmod{n}$
Израчунати Јакобијев симбол $s = (\frac{a}{n})$; // нпр. алгоритмом датим у поглављу 3
if $(r \neq s \pmod{n})$ **return** ("сложен");
return ("прост");

Први корак овог алгоритма је да се израчуна $r = a^{\frac{n-1}{2}} \pmod{n}$, што је скоро исто као код Фермаовог теста. Следећи корак је рачунање Јакобијевог симбола $(\frac{a}{n})$ и поређење са r. Ако они нису једнаки, n је сложен по Ојлеровом критеријуму. У супротном, a није Ојлеров сведок.

Соловеј-Штрасенов тест је спорији од Фермаовог теста. Овај алгоритам захтева рачунање Јакобијевог симбола, па његова ефикасност зависи од тражења једноставнијег израчунавања Јакобијевог симбола од n.

Соловеј-Штрасенов алгоритам тестира да ли је цео број n прост тако што извршава горњи алгоритам тражења Ојлеровог сведока, за k случајно изабраних вредности броја a. Ако је нека од k вредности Ојлеров сведок, онда је n сложен. Ако ниједна вредност за a није сведок, онда је или n прост, или су свих k вредности Ојлерови лажови.

Неефикасно је тестирати све могуће вредности за a, зато је потребно знати колика је вероватноћа да ако је n сложен, случајно изабран цео број a буде Ојлеров сведок сложености броја n. Следећа теорема гарантује да се, за разлику од Фермаовог теста где постоје Кармајклови бројеви, овај тест равноправно понаша за било који цео број n на улазу.

Теорема 4.2. *Ако је n непаран сложен број, онда постоји највише $\frac{\phi(n)}{2}$ Ојлерових лажова за n у интервалу $[2, n-2]$ [17, 28].*

Како је $\phi(n) < n$, онда је за непаран сложен број n мање од $\frac{n}{2}$ избора за a који су Ојлерови лажови, па следи да су најмање $\frac{n}{2}$ избора Ојлерови сведоци. Одавде следи да ако је n сложен, постоји бар 50% шанси да је било која случајно изабрана вредност за a сведок. Ако тест класификује број као сложен, не постоји вероватноћа грешке. Ако тест класификује број као прост, може доћи до грешке. Вероватноћа грешке је дата следећом лемом:

Последица 4.1. *Ако Соловеј-Штрасенов тест са параметрима n и k класификује n као прост број, вероватноћа да је одговор нетачан је највише $(\frac{1}{2})^k$.*

На основу овог тврђења следи да се може достићи произвољан степен *тачности* једноставним повећањем вредности броја k.

4.4 Милер-Рабинов тест простости

Милер-Рабинов тест простости је познат и као **јак тест псеудопрималности** (енгл. *strong pseudoprime test*). Овај вероватносни тест је највише коришћен у пракси.

Постоји релативно једноставно проширење мале Фермаове теореме које омогућава тестирање да ли је број прост са много већом вероватноћом коректности него код Фермаовог теста.

Следеће тврђење је основа Милер-Рабиновог теста:

Теорема 4.3. *Нека је n непаран прост број и нека је $n-1 = 2^r \cdot s$, где је s непаран.*[39] *Нека је a било који цео број такав да је $\mathrm{nzd}(a, n) = 1$. Тада је или $a^s \equiv 1 \pmod{n}$ или је $a^{2^i \cdot s} \equiv -1 \pmod{n}$ за неко i, $0 \leq i \leq r-1$.*

Доказ. Нека је n непаран број који је потребно тестирати. Треба раставити $n-1$ у форму $2^r \cdot s$, где је s непаран број. Како је $n-1$ паран број, важи да је $r \geq 1$. Изабере се цео број a такав да је $\mathrm{nzd}(a, n) = 1$, који је онда база и рачуна се следећи низ бројева:

$$a^s \pmod{n},\ (a^s)^2 \pmod{n},\ \ldots,\ (a^s)^{2^r} \pmod{n}.$$

Важи $(a^s)^{2^r} \equiv a^{n-1} \equiv 1 \pmod{n}$ на основу мале Фермаове теореме, јер је $\mathrm{nzd}(a, n) = 1$. Приметимо да ако је $\mathrm{nzd}(a, n) > 1$, n није прост број и заправо постоји фактор броја n. Према томе, дати низ бројева мора да се заврши са 1 или је у супротном n сложен број.

[39] Приметимо да је $n = 2^r \cdot s + 1$, $s < 2^r$, заправо Протов број.

Ако једначина $x^2 \equiv 1 \pmod{n}$ има нетривијална решења, онда је n сложен број. Другим речима, над пољем квадратна једначина $x^2 = 1$ има само тривијална решења, тј. тачно два решења: 1 и -1. Ово значи да ако се може наћи неко x такво да је $x \neq \pm 1$, а чији је квадрат једнак 1 по модулу n, онда n мора бити сложен, и x је сведок сложености броја n. На основу овог разматрања важи да број $(a^s)^{2^{r-1}}$ који претходи $(a^s)^{2^r}$ мора бити једнак ± 1. Ако је $a^s \cdot 2^{r-1} \pmod{n} = -1$, нађен је $j = r-1$ за које важи тврђење. Ако је $a^s \cdot 2^{r-1} \pmod{n} = 1$, онда следи да је $a^s \cdot 2^{r-2} \pmod{n} = \pm 1$, на који се примењује исто разматрање, итд. Ако је $a^s \cdot 2 \pmod{n} = 1$, следи да је $a^s \pmod{n} = \pm 1$, одакле следи тврђење. □

Ова теорема даје потребан, али не и довољан услов да је неки број прост.

Дефиниција 4.7. Број n за који је Милер-Рабинов тест дао одговор "прост" у некој бази a се назива **јак вероватно прост број** у бази a (енгл. *strong probable prime*). Ако је n сложен број за који Милер-Рабинов тест да одговор да је "прост", онда се каже да је n **јак псеудопрост број** у бази a. У пракси, овакви бројеви су изузетно ретки [17].

АЛГОРИТАМ: Милер-Рабинов сведок

mr-witness(a, n)
УЛАЗ: Непаран цео број n и цео број a такав да је $\text{nzd}(a, n) = 1$ и $a \in [2, n-2]$.
ИЗЛАЗ: Одговор "тачно" или "нетачно" на питање: "Да ли је a Милер-Рабинов сведок?"
Нека је $n - 1 = s \cdot 2^r$, где је $r > 0$ и s је непаран цео број.
Нека је $x_1 = a^s \pmod{n}$; // израчунати нпр. алгоритмом датим у поглављу 3
if $(x_1 \neq 1$ **and** $x_1 \neq n-1)$
  $i = 1$;
  while $(i \leq r-1$ **and** $x_1 \neq n-1)$ **do**
    $x_2 = x_1 \cdot x_1 \pmod{n}$;
    if $(x_2 = 1)$ **return** ("тачно");
    $x_1 = x_2$;
    $i = i + 1$;
  if $(x_2 \neq n-1)$ **return** ("тачно");
else return ("нетачно");

Напомена 4.3. Алгоритам прво израчунава a^s, онда га квадрира r пута, памтећи последњу (x_1) и тренутну (x_2) вредност у сваком кораку. Ако је $x_2 = 1$, онда је x_1 квадратни корен јединице по модулу n. Ако је $x_1 \neq \pm 1$, тада је он нетривијални квадратни корен јединице по модулу n, и зато је n сложен. Ако mr-witness(a,n) врати "тачно", a је **јак сведок** сложености броја n.

Милер-Рабинов тест једноставно извршава mr–witness за k случајно изабраних вредности броја a. Ако је било која од ових вредности сведок сложености броја n, онда је n сложен. У супротном, n је вероватно прост број.

АЛГОРИТАМ: Милер-Рабинов вероватносни тест прималности

miller-rabin(n, k)
УЛАЗ: Непаран цео број n и цео број k.
ИЗЛАЗ: Одговор "прост" или "сложен" на питање: "Да ли је n прост?"
for $(i = 1$ **to** $k)$ **do**
  $a = random\ (2,\ n-2)$;
  if (mr-witness $(a,\ n)$) **return** ("сложен"); // a је сведок
return ("прост"); // није нађен сведока, па је n вероватно прост

Теорема 4.4. *Ако је n непаран сложен број, онда је број сведока сложености броја n најмање $\frac{3}{4}(n-1)$ [17].*

Ова теорема гарантује да ако је n сложен, постоји најмање 75% шанси да је било који случајно изабрани број сведок његове сложености. Милер-Рабинов тест прави грешку само ако су све изабране вредности за k из скупа бројева који нису сведоци.

Последица 4.2. *Ако Милер-Рабинов тест са параметрима n и k класификује n као прост број, вероватноћа грешке је највише* $(\frac{1}{4})^k$.

Ова тривијална горња граница вероватноће грешке која је $(\frac{1}{4})^k$ може бити значајно побољшана [17].[40]

Иако обезбеђује веома високу вероватноћу да је излаз тачан, Милер-Рабинов тест није тест за доказивање простости, и као сваки пробабилистички тест, може дати нетачан одговор. У већини апликација тако мала вероватноћа грешке је потпуно прихватљива, али излаз вероватносног теста никада не може бити сматран доказом прималности.[41]

Пример 4.1. Размотримо сада одређене примере:[42]

- **Варијанта Милер-Рабиновог теста.** За $\frac{1}{3}$ целих бројева $n > 3$, бројева облика $3k + 1$, могуће је извршити следећу факторизацију: $n-1 \equiv 3^s \cdot m$, где је $\mathrm{nzd}(m, 3) = 1$. Даље, одабира се број a који је база и рачуна се следећи низ бројева: $a^m \pmod n$, $(a^m)^3 \pmod n, ..., (a^m)^{3^s} \pmod n$. Као и претходно, на основу тога како су конструисани бројеви, важи да је $(a^m)^{3^s} \equiv a^{3^s \cdot m} \equiv a^{n-1}$. На основу мале Фермаове теореме следи да ако је n прост број и $\mathrm{nzd}(a, n) = 1$, онда се овај низ мора завршити са 1, тј. $a^{n-1} \equiv 1 \pmod n$.

  Сада се разматрају особине претпоследњег члана низа, означимо га са x. Ако је n прост, $\mathbb{Z}_n$ је поље, и важи да је $x^3 - 1 = (x - 1)(x^2 + x + 1) = 0$, одакле следи да је $x = 1$ или је x решење једначине $x^2 + x + 1 = 0$ у $\mathbb{Z}_n$. Иако није познато који су бројеви осим 1 кубни корени јединице у $\mathbb{Z}_n$, једноставно се може израчунати $x^2 + x + 1 \pmod n$ да би се обавила провера да ли је једнако 0.

  Заправо, при покретању теста се извршава следеће:
  - Провера да ли је сваки од бројева који претходи члану који је једнак 1, једнак 1 или је решење једначине $x^2 + x + 1 = 0 \pmod n$.
  - Провера да ли је $(a^m)^{3^s} \equiv 1 \pmod n$.

- **Друга варијанта.** Ако је $\mathbb{F}$ коначно поље, онда је $\mathbb{F}^* = \mathbb{F}\backslash\{0\}$ циклична група за операцију множења. Претпоставимо да је $|\mathbb{F}^*| = m - 1$, онда је $a^{m-1} = 1$.

  Ово тврђење се може искористити да би развили тест прималности. Претпоставимо да је n број који треба тестирати. Ако је n прост, онда је $\mathbb{Z}_n$ поље и $\mathbb{Z}_n[x]$ прстен полинома са коефицијентима у $\mathbb{Z}_n$. Прво се *гради* поље. Налази се полином који је нерастављив по модулу n. Нека је то полином $p(x)$ степена k. Ако је n прост, онда је $\mathbb{Z}_n[x]/(p(x))$ поље које има n^k елемената. То значи да ако је $m = n^k$ и ако је $f(x)$ елемент у $\mathbb{Z}_n[x]/(p(x))$, следи да је $f(x)^{m-1} = 1 \pmod{p(x)}$. Наредни поступак даје тест да ли је n прост број:

  - Бира се нерастављив квадратни полином $p(x)$ над $\mathbb{Z}_n[x]$.
  - Бира се базни полином у $\mathbb{Z}_n[x]/(p(x))$. Нека је $x + 1$ базни полином.
  - Раставља се $n-1 = 2^s \cdot q$, где је q непаран број. Рачуна се низ $(x+1)^q \pmod{p(x)}, ((x+1)^q)^2 \pmod{p(x)}, ..., ((x + 1)^q)^{2^s} \equiv (x + 1)^{m-1} \pmod{p(x)}$.

  Овај низ полинома треба да се заврши са 1. Користи се и чињеница да $x^2 = 1$ има тачно два решења у пољу $\mathbb{Z}_n$.

[40] За више информација, видети *"Average case error estimates for the strong probable prime test"*, Ivan Damgård, Peter Landrock, Carl Pomerance, Mathematics of computation.

[41] За више информација, видети *"A study of Maurer's algorithm for finding provable primes in relation to the Miller-Rabin algorithm"*, Schwarz, Andersen, 2007.

[42] *"Finding prime numbers: Miller Rabin and beyond"*, Christina Mcintosh, Furman University, Eletronic Journal of Undergraduate Mathematics, Volume 12, 1 - 4, 2007.

4.5 n - 1 тест простости

За n - 1 тест је потребна (делимична) факторизација броја $n-1$ да би одредио да ли је n прост број.

Теорема 4.5 (Лукас). *Нека су a и n цели бројеви и $n > 1$. Ако је $a^{n-1} \equiv 1 \pmod{n}$, и $a^{\frac{n-1}{q}} \neq 1 \pmod{n}$ за сваки прост број q, $q|n-1$, онда је n прост број.*

Доказ. Нека је k ред броја a по модулу n. Из $a^{n-1} \equiv 1 \pmod{n}$ следи да $k|n-1$. Али, k није једнако било ком делитељу броја $n-1$ зато што је $a^{\frac{n-1}{q}} \neq 1 \pmod{n}$ за сваки прост број $q|n-1$. Зато k мора бити једнако $k = n-1$. Такође, $k = n-1$ дели $\phi(n)$, па је $\phi(n) \geq n-1$. На основу дефиниције Ојлерове функције ϕ важи да је $\phi(n) \leq n-1$, одакле следи да је $\phi(n) = n-1$. Сада, на основу Теореме 2.7 следи да је n прост број. □

Тест који се заснива на овој теореми је **тест за доказивање прималности**. Када је n прост број, вредности броја a које задовољавају услове теореме су они чији је ред баш једнак $n-1 = \phi(n)$. Тачније, то су примитивни корени броја n. Сваки прост број има примитивни корен. На основу Теореме 2.10 следи да прост број n има $\phi(\phi(n)) = \phi(n-1)$ примитивних корена. Ако не важе услови теореме, онда морамо тражити други број a. На пример, за $n = 7$ и $a = 2$, важи да је $2^6 = 1 \pmod{7}$, али је $2^3 = 1 \pmod{7}$, и важи да је $\phi(7) = 6$, $\phi(6) = 2$, и $3|6$, где су примитивни корени 3 и 5.

Није познат ефикасан алгоритам за проналажење примитивних корена. Али, основна препрека у имплемантацији Лукасове теореме као теста прималности није тражења примитивног корена a, већ проналажење потпуне факторизације броја $n-1$. Зато је овај тест посебно интересантан за употребу са Фермаовим бројевима, као и у случајевима када се специфичним методама конструише број n који је кандидат за проверу.

Делимична факторизација

Најтежи део n - 1 теста је факторисање броја $n-1$. Проблем факторизације у општем случају је тежак, али није тежак за све бројеве, па $n-1$ тест има различиту делотворност. На пример, прости бројеви који су за један већи од неког степена броја 2 су на основу теореме 2.21 заправо Фермаови бројеви. За Фермаове бројеве факторизација $n-1$ је већ позната.[43]

Такође, може се искористити неколико резултата који омогућују тестирање n без потпуне факторизације броја $n-1$.

Напомена 4.4. Нека су $\mathbb{Z}_n^*$ и $\mathbb{Z}_m^*$ мултипликативне групе такве да $n|m$. Нека је $r_n(x)$ ред елемента x у групи $\mathbb{Z}_n^*$ и $r_m(x)$ ред елемента x у групи $\mathbb{Z}_m^*$. Тада важи да је $x^{r_n(x)} \equiv 1 \pmod{n}$ и $x^{r_m(x)} \equiv 1 \pmod{m}$. Како $n|m$ следи да је m најмањи заједнички садржалац за n и m, па на основу теореме 2.4 под 6. следи да је $x^{r_m(x)} \equiv 1 \pmod{n}$. Како је $r_n(x)$ најмањи цели број који има ово својство, следи да $r_n(x)|r_m(x)$. Другачије речено, за сваки $x \in \mathbb{Z}_m^*$ важи $n|m \Rightarrow r_n(x)|r_m(x)$.

Ради теорема које следе, претпоставимо да је $n-1 = F \cdot R$, где је позната комплетна факторизација за F, али не и за R.

Теорема 4.6 (Поклингтонова теорема). *Ако постоји цео број a такав да је $a^{n-1} \equiv 1 \pmod{n}$, $n \geq 3$ и $n = RF + 1$, и важи $\mathrm{nzd}(a^{\frac{n-1}{q}} - 1, n) = 1$ за сваки прост број q такав да $q|F$, онда је сваки прост фактор p броја n конгруентан 1 по модулу F, тј. $p \equiv 1 \pmod{F}$. Другачије речено, $p = m \cdot F + 1$ за неки цео број $m \geq 1$.*

Доказ. Из услова теореме важи да је $a^{n-1} \equiv 1 \pmod{n} = (a^R)^F \equiv 1 \pmod{n}$, одакле следи да за сваки прост број $p|n$, ред $r_p(a^R)$ елемента a^R у пољу $\mathbb{Z}_p^*$ дели F, и такође дели ред поља који је једнак $p-1$. Како је $\mathrm{nzd}(a^{\frac{n-1}{q}} - 1, n) = 1$ за сваки прост број $q|F$, следи да је $r_p(a^R) = F$. Одатле се добија да F дели $p-1$, тј. тврђење теореме [17]. □

Поклингтонова теорема је у основи **Маурерового алгоритма** (енгл. *Maurer algorithm*), који представља технику за конструисање доказивих простих бројева [28, 30].

Из претходне теореме следи да је $F+1$ најмањи могући прост фактор од n. Одавде следи да важи следеће тврђење:

[43] Приметимо да се основу овог запажања и претходне теореме може извести Пепинова теорема.

Последица 4.3. *Ако претпоставка претходне теореме важи и $F \geq \sqrt{n}$, онда је n прост број.*

Теорема 4.7. *Нека важе претпоставке из претходне теореме и нека је $n^{\frac{1}{3}} \leq F < n^{\frac{1}{2}}$. Нека је $n = c_2 \cdot F^2 + c_1 \cdot F + 1$ репрезентација (енгл. expansion) броја n у бази F и $c_1, c_2 \in [0, F-1]$. Тада је n прост број ако и само ако $c_1^2 - 4c_2$ није квадрат неког броја [17].*

Са датом делимичном факторизацијом $n-1 = F \cdot R$, где је $F \geq n^{\frac{1}{3}}$, горње теореме дају следећи тест прималности.

АЛГОРИТАМ: n-1 тест

---

n-1 test (n, F)

УЛАЗ: Ненегативан цео број n и F, где је $n-1 = F \cdot R$, за F је позната потпуна факторизација.

ИЗЛАЗ: "тачно", "нетачно", "потребан је већи број F" — одговори на питање да ли је n прост број.

[**Поклингтонов тест**]: // наћи одговарајуће a:

тражи = true;

while (тражи) **do**

Изабрати случајан цео број a, $a = random[2, n-2]$;

if $(a^{n-1} \neq 1 \pmod{n})$ **return** ("нетачно");

for (прости фактори q од F, $q|F$) **do**

$d = \text{nzd}(a^{\frac{n-1}{q}} - 1, n)$;

if $(d \neq 1$ **and** $d \neq n)$ **return** ("нетачно"); // нађен је делитељ броја n

тражи = false;

if $(d = n)$

тражи = true; // $a^{\frac{n-1}{q}} \equiv 1 \pmod{n}$, a није примитивни корен, тражи друго a

break;

// петља је прошла кроз све просте факторе q од F и услови теореме су задовољени

// провера да ли је F довољно велико да би се показала прималност броја n:

[**Први тест магнитуде**]:

if $(F \geq \sqrt{n})$ **return** ("тачно");

[**Други тест магнитуде**]:

if $(F \geq n^{\frac{1}{3}})$

Нека је $n = c_2 \cdot F^2 + c_1 \cdot F + 1$;

if ($c_1^2 - 4 \cdot c_2$ није квадрат) **return** ("тачно");

else return ("нетачно");

else return ("потребан је већи број F");

Позната су побољшања датог теста која омогућавају да се утврди да ли је број n прост или сложен и за мање вредности F, тј. постоји и трећи тест магнитуде (енгл. *third magnitude test*) [17].

Иако n - 1 тест не може да у свим случајевима одреди да ли је или не број прост, ако је добијен **резултат да је број прост или да је број сложен, то је стриктно тврђење** (енгл. *rigorous declaration*).

Овај тест користи елементе Фермаовог теста: ако је $a^{n-1} \neq 1 \pmod{n}$, онда је n прост број. Слично, ако се нађе делитељ броја n, јасно је да је n сложен. Како је Фермаов тест прилично ефикасан за већину сложених бројева, овај тест елиминише многе сложене бројеве пре него што дођу до другог дела теста.

Неколико фактора утиче на практичну корист овог теста. Прво, мора постојати могућност проналаска довољно велике делимичне факторизације броја $n-1$. Многи цели бројеви имају релативно мале делитеље и могу лако бити факторисани, али за друге је то теже. Да би овај тест прималности био бржи, треба употребити и ефикасан метод за проверу да ли је $c_1^2 - 4 \cdot c_2$ квадрат. Такође, морамо бити сигурни да $F|(n-1)$, као и да су бројеви из факторизације броја F заиста прости бројеви. Тражење одговарајућег a је насумична потрага (енгл. *random search*), и време тог тражења је зато променљиво. Ако је n прост број, онда било који од његових примитивних корена задовољава услове за a. Насумично тражење примитивних корена би требало да у просеку захтева мање од $2 \ln \ln n$ погађања [17, 28].

4.6 Пепинов тест простости

Фермаови бројеви представљају бројеве облика $F_n = 2^{2^n} + 1$. За ове бројеве је већ позната факторизација броја $n-1$. Следећа теорема даје једноставан тест за доказивање прималности Фермаових бројева, познат као Пепинов тест, објављен 1877.

Теорема 4.8 (Пепинов тест). *Нека је са F_n означен n-ти Фермаов број, $F_n = 2^{2^n} + 1$ и $n > 1$. F_n је прост број ако и само ако је $3^{\frac{F_n-1}{2}} = 3^{2^{2^n-1}} \equiv -1 \pmod{F_n}$.*

Доказ. $\boxed{\Leftarrow}$ За доказ у овом смеру, претпоставимо да важи конгруенција $3^{\frac{F_n-1}{2}} = -1 \pmod{F_n}$. Онда важи $3^{F_n-1} = 1 \pmod{F_n}$, па према томе мултипликативни ред броја 3 модуло F_n дели $F_n - 1 = 2^{2^n}$. Са друге стране, ред не дели $(F_n-1)/2 = 2^{2^n-1}$, па зато ред мора бити једнак $F_n - 1 = 2^{2^n}$. Посебно, онда постоји бар $F_n - 1$ бројева мањих од F_n који су узајамно прости са F_n, а ово може бити могуће само ако је F_n прост број.
$\boxed{\Rightarrow}$ За доказ у другом смеру, претпоставимо да је F_n прост. Према Ојлеровом критеријуму важи:

$$3^{\frac{F_n-1}{2}} \equiv \left(\frac{3}{F_n}\right) \pmod{F_n},$$

где је $(\frac{3}{F_n})$ Лежандров симбол. Понављајући квадрирање, следи да је $2^{2^n} \equiv 1 \pmod 3$, тј. $F_n \equiv 2^{2^n} + 1 \equiv 2 \pmod 3$ и $(\frac{F_n}{3}) = -1$, тј. не постоји цео број x такав да је $x^2 \equiv F_n \pmod 3$. Како је $F_n \equiv 1 \pmod 4$, на основу закона о квадратним остацима се закључује да је $(\frac{3}{F_n}) = -1$. □

Иако није познат ниједан прост Фермаов број F_n за $n > 4$, питање колико има простих Фермаових бројева је и даље отворено. Ако је F_n прост број, његова прималност може бити показана Пепиновим тестом, али ако је F_n сложен, овај тест не налази његове факторе, већ само даје одговор да број није прост. На пример, Selfridge и Hurwitz су још 1963. године доказали да је број F_{14} сложен, али и даље се не знају његови чиниоци. Приметимо да је Протова теорема аналогна Пепиновој теореми, за $h = 1$, $n = F_n$.

Овај алгоритам има веома ефикасно време извршавања, али број битова код Фермаових бројева расте експоненцијално, што значајно умањује његову практичност. Такође, основна операција Пепиновог теста је квадрирање великих бројева, односно множење великих бројева по модулу, одакле следи да ефикасност Пепиновог теста зависи од ефикасности алгоритма за ове операције. У [17] је дат Шенхагеов алгоритам за брзо множење по модулу $2^n + 1$, док је у [11] дата детаљна анализа, имплементација и паралелизација Пепиновог теста заснованог на Шенхаге - Штрасеновом алгоритму (енгл. *Schönhage - Strassen*, користећи брзу Фуријеову трансформацију, *FFT*).

4.7 Лукас-Лемеров тест простости за Мерсенове бројеве

Ради њиховог значаја за развијање Лукас-Лемеровог теста за бројеве у општем облику, одакле се добија и овај тест за Мерсенове бројеве, прво је дата дефиниција и битни примери Лукасових низова.

Лукасови низови (енгл. *Lucas sequences*) $U_n(P,Q)$ и $V_n(P,Q)$ су целобројни низови који задовољавају рекурентну релацију $X_n = P \cdot X_{n-1} - Q \cdot X_{n-2}$, где су P и Q фиксирани цели бројеви.

За дате целе бројеве P и Q, Лукасов низ првог реда $U_n(P,Q)$ и другог реда $V_n(P,Q)$ су дефинисани рекурентним релацијама: $U_0(P,Q) = 0, U_1(P,Q) = 1, U_n(P,Q) = P \cdot U_{n-1}(P,Q) - Q \cdot U_{n-2}(P,Q)$ за $n > 1$, односно $V_0(P,Q) = 2$, $V_1(P,Q) = P$, $V_n(P,Q) = P \cdot V_{n-1}(P,Q) - Q \cdot V_{n-2}(P,Q)$ за $n > 1$.

Карактеристична једначина рекурентне релације за Лукасове низове $U_n(P,Q)$ и $V_n(P,Q)$ је $x^2 - Px + Q = 0$, чија је дискриминанта $\Delta = P^2 - 4Q$. Корени једначине су $a = \frac{P+\Delta}{2}$ и $b = \frac{P-\sqrt{\Delta}}{2}$, одакле следи да је $a+b = P$ и $a \cdot b = \frac{P^2-\Delta}{4} = Q$. Када је $\Delta \neq 0$, лако се показује да је $a^n = \frac{V_n+U_n\sqrt{\Delta}}{2}$ и $b^n = \frac{V_n-U_n\sqrt{\Delta}}{2}$.

На основу овога могу се дефинисати елементи Лукасовог низа као $U_n = \frac{a^n-b^n}{a-b} = \frac{a^n-b^n}{\sqrt{\Delta}}$, $V_n = a^n + b^n$.

Мерсенови бројеви $M_n = 2^n - 1$ су специјалан случај Лукасових низова за $U_n(3,2)$, тј. $P = 3$, $Q = 2$ и ако се означи да је $U_n(3,2) = U_n$, онда је $U_0 = 0$, $U_1 = 1$, $U_n = 3 \cdot U_{n-1} - 2 \cdot U_{n-2}$. Одатле је $U_2 = 3$, $U_3 = 9 - 2 = 7$, $U_4 = 21 - 6 = 15$, ..., што одговара Мерсеновим бројевима.

Лукасови низови имају многе интересантне особине и примене. Поменимо још да су Фибоначијеви бројеви, Лукасови бројеви, и бројеви облика $2^n + 1$, у које спадају и Фермаови бројеви, такође примери Лукасових низова.

Лукас-Лемеров тест (енгл. *Lucas-Lehmer*), или **n+1 тест**, је пример још једног теста прималности који се заснива на познавању делимичне факторизације броја неког броја повезаног са бројем n, у овом случају броја $n + 1$.

Како је за Мерсенове бројеве позната потпуна факторизација за $n+1$, овај тест је прави тест за доказивање прималности за Мерсенове бројеве, као што је то Пепинов тест за Фермаове бројеве.

Лукас-Лемеров тест за Мерсенове бројеве се заснива на следећој теореми, чији је доказ дат у [17].

Теорема 4.9. *Посматрајмо низ v_k за $k = 0, 1, ...,$ дефинисан рекурзивно са $v_0 = 4$ и $v_{k+1} = v_k^2 - 2$. Нека је p непаран прост број. Онда је $M_p = 2^p - 1$ прост ако и само ако је $v_{p-2} \equiv 0 \pmod{M_p}$.*

Сада је описан добро познати тест за Мерсенове бројеве:

АЛГОРИТАМ: Лукас-Лемеров тест простости за Мерсенове бројеве

mersenne-test (n)
УЛАЗ: Непаран прост број $n \geq 3$, $M_n = 2^n - 1$
ИЗЛАЗ: "тачно", "нетачно" одговори на питање да ли је M_n Мерсенов прост број.
[**Провера да ли је n прост број**]
    Користи се поновљено дељење због провере да ли n има неки фактор између 2 и $\lfloor \sqrt{n} \rfloor$.
    Ако има, **return** ("нетачно");
[**Иницијализација**]
    $u = 4$;
[**Израчунати Лукас-Лемеров низ**]
    for $(k = 1$ **to** $n - 2)$ **do**
        $u = u^2 - 2 \pmod{M_n}$;
[**Проверити остатак**]
    if $(u = 0)$ **return** ("тачно");
    else return ("нетачно");

Ефикасност овог алгоритма зависи од ефикасности $n-2$ поновљена квадрирања по модулу M_n код израчунавања Лукас-Лемеровог низа.

4.8 Тестови простости са Гаусовим и Јакобијевим сумама

Дирихлеов карактер и особине ове функције у основи су тестова прималности заснованих на Јакобијевим и Гаусовим сумама.

Дефиниција 4.8. Нека је n позитиван цео број и χ функција која пресликава скуп целих бројева у скуп комплексних бројева таква да је:

- $\chi(a \cdot b) = \chi(a) \cdot \chi(b)$ за $\text{nzd}(a, n) = \text{nzd}(b, n) = 1$.
- χ је периодична по модулу n, тј. $\chi(a) = \chi(b)$ за $a \equiv b \pmod{n}$.
- $\chi(a) = 0$ ако и само ако је $\text{nzd}(a, n) > 1$.
- $\chi(1) \neq 0$

Онда се каже да је χ **Дирихлеов карактер по модулу** n. Како на основу претходне дефиниције следи да једначина $\chi(a \cdot b) = \chi(a) \cdot \chi(b)$ важи за све $a, b \in \mathbb{Z}$, следи да је сваки карактер χ по неком модулу k тотално мултипликативна функција.

Пример Дирихлеовог карактера по модулу n, за $n > 1$ непаран цео број, је Јакобијев симбол $(\frac{a}{n})$. Приметимо да ако је $\text{nzd}(a, n) = 1$, онда је на основу Ојлерове теореме $\chi(a)^{\phi(n)} = \chi(a^{\phi(n)}) = \chi(1) = 1$, односно $\chi(a)$ је **корен јединице** (енгл. *root of unity*), тј. $\phi(n)$-ти корен јединице, $\text{nzd}(a, n) = 1$.

Може се показати да не само да постоји највише $\phi(n)^{\phi(n)}$ карактера по модулу n, већ да их има тачно $\phi(n)$.

Још једна особина Дирихлеовог карактера је да је **затворен за множење**. Ако је χ_1 Дирихлеов карактер по модулу n_1, а χ_2 по модулу n_2, онда је $\chi_1\chi_2$ Дирихлеов карактер по модулу $[n_1, n_2]$, где је са $[n_1, n_2]$ означен најмањи заједнички садржалац. Важи и да је $(\chi_1\chi_2)(a) = \chi_1(a)\chi_2(b)$. Лако се показује да **Дирихлеови карактери по модулу** n **образују мултипликативну групу** $\mathbb{Z}_n^*$, где је χ_0 **идентитет**, са особинама да је $\chi_0(a) = 1$ када је $\text{nzd}(a, n) = 1$, а једнако 0 у супротном, тј. са $\chi_0(a) = \begin{cases} 1, & \text{nzd}(a,n) = 1, \\ 0, & \text{nzd}(a,n) > 1 \end{cases}$ је дефинисан карактер по модулу n. Мултипликативни инверз од χ по модулу n је његов **комплексни конјугат** $\bar{\chi}$. Такође важи и да је $|\chi(a)| = 1$.

Пример 4.2. За $n = 5$, за све карактере по модулу 5, за $\text{nzd}(a, 5) = 1$ важи да је $(\chi(a))^{\phi(5)} = (\chi(a))^4 = 1$, односно да карактера по модулу 5 има 4, то су χ_0, χ_1, χ_2, χ_3, χ_4, и да су једине могуће вредности за $\chi(a)$ бројеви $1, -1, i, -1$. Одатле, и на основу дефиниције карактера, следи да је $1 = \chi(1) = \chi(1)\chi(1) = \chi(6) = \chi(2)\chi(3)$ и $\chi(4) = \chi^2(2)$. Све вредности за $\chi_0(a)$ су једнаке 1, док на пример за $\chi_1(a)$ важи да је по дефиницији $\chi_1(1) = 1$, док је $\chi_1(2) = i$ (могући избори су још -1 и i), па је онда $\chi_1(3) = -i$ и $\chi_1(4) = -1$.

Карактери по модулу степена простог броја $q = p^k$ су веома битни. Нека је g примитивни корен по модулу q. Онда степени броја g по модулу q генеришу све елементе редукованог система остатака по модулу q (елементи који су узајамно прости са q). Ако се изабере $\phi(q)$-ти корен јединице r, онда је јединствени карактер $\chi \pmod q$ такав да је $\chi(g) = r$. Следи да постоји $\phi(q) = \phi(p^k)$ различитих карактера $\chi \pmod q$.

Ако је p прост број са примитивним кореном g и ξ је комплексни број такав да је $\xi^{p-1} = 1$, онда се може узети да је карактер χ по модулу n једнак $\chi(g^k) = \xi^k$ за сваки цео број k.

Такође, ако χ пролази скупом свих $\phi(n)$ карактера по модулу n, онда важи да је:

$$\sum_{\chi \pmod n} \chi(a) = \begin{cases} \phi(n), & a \equiv 1 \pmod n, \\ 0, & a \not\equiv 1 \pmod n \end{cases}$$

Доказ овог тврђења се може наћи у [4].

Каже се да је n-ти корен јединице **n-ти примитивни корен јединице** ако није k-ти корен јединице за неки мањи број k. Примитивни n-ти корен јединице се обележава са $\xi_n = e^{\frac{2\pi i}{n}}$.

Дефиниција 4.9. Циклотомично поље $\mathbb{Q}[\xi_n]$ је циклотомично раширење реда n поља $\mathbb{Q}$, где је $n > 2$ и ξ_n је n-ти примитивни корен јединице. Ово циклотомично поље садржи све нуле полинома $x^n - 1$, па је тако и Галоаово раширење поља $\mathbb{Q}$.

На сличан начин се може дефинисати коренско поље $\mathbb{F}[\xi_n]$ полинома $X^n - 1$ над пољем $\mathbb{F}$ карактеристике p, где важи да $p \nmid n$. Више о циклотомичним пољима видети у [8].

Дефиниција 4.10. Нека је χ карактер по модулу q, q прост број са примитивним кореном g. **Гаусова сума** (енгл. *Gauss sum*) повезана са овим карактером се дефинише на следећи начин:

$$G(\chi) = \sum_{j=1}^{q-1} \chi(j)\xi_q^j = \sum_{k=1}^{q-1} \chi(g^k)\xi_q^{g^k} = \sum_{k=1}^{q-1} \xi^k \xi_q^{g^k}.$$

Ако се као карактер узме Лежандров симбол, Гаусова сума се дефинише као $G = \sum_{j=1}^{q-1} (\frac{j}{q})\xi_q^j$. За доказ закона квадратног реципроцитета могу се користити ове Гаусове суме.

Дефиниција 4.11. Нека је p фиксиран прост број и a цео број. Гаусова сума је:

$$G(a, p) = G_a = \textstyle\sum_{j=0}^{p-1} (\frac{j}{p})\xi^{aj}, \text{ где је } \xi = \xi_p = e^{\frac{2\pi i}{p}} = \cos(\frac{2\pi}{p}) + i\sin(\frac{2\pi}{p}).$$

Дефиниција 4.12. Нека су $a, n > 0 \in \mathbb{Z}$. Онда је **квадратна Гаусова сума** дефинисана са:

$$G(a, n) = \sum_{j=0}^{n-1} e^{2\pi i a j^2/n}$$

Дефиниција 4.13. Нека је g најмањи позитивни примитивни корен за q, и нека је $\chi_{p,q}(g^k) = \xi_p^k$ за сваки цео број k, ξ_p је p-ти примитивни корен јединице. Како је $\xi_p^{q-1} = 1$, на овај начин смо дефинисали карактер по модулу q. Такође $\chi_{p,q}$ је реда p.

Дефиниција 4.14. На основу претходне дефиниције, Гаусова сума $G(p,q)$ је дефинисана као:

$$G(p,q) = \sum_{n=1}^{q-1} \chi_{p,q}(n)\xi_q^n = \sum_{k=1}^{q-1} \xi_p^k \xi_q^{g_q^k} = \sum_{k=1}^{q-1} \xi_p^{k \pmod p} \xi_q^{g_q^k \pmod q}$$

Неке особине Гаусових сума:

- За било који цео број a, $\sum_{n=0}^{p-1} \xi^{an} = \begin{cases} p, & ako\ a \equiv 0 \pmod p, \\ 0, & ako\ a \not\equiv 0 \pmod p. \end{cases}$
- Ако су x и y цели бројеви, онда $\sum_{n=0}^{p-1} \xi^{(x-y)n} = \begin{cases} p, & ako\ x \equiv y \pmod p, \\ 0, & ako\ x \not\equiv y \pmod p. \end{cases}$
- $G_0 = 0$. Такође, за сваки цео број a, важи $G_a = (\frac{a}{p})G_1$.
- За сваки цео број a, такав да a није дељиво са p, $G_a^2 = (-1)^{\frac{p-1}{2}} p$.
- Гаусова сума $G(p,q)$ је елемент прстена $\mathbb{Z}[\xi_p, \xi_q]$. Елементи овог прстена могу се изразити на јединствен начин као суме $\sum_{j=0}^{p-2}\sum_{k=0}^{q-2} a_{j,k}\xi_p^j\xi_q^k$, где сви $a_{j,k} \in \mathbb{Z}$.

Дефиниција 4.15 (Јакобијева сума)**.** Нека су a и b два цела броја, p, q два проста броја таква да $p|(q-1)$, и p је непаран прост број. Нека је $b = b(p)$ најмањи позитивни цео број такав да је $(b+1)^p \neq b^p + 1 \pmod{p^2}$. Јакобијева сума је $J(p,q) = \sum_{n=1}^{q-2} \chi_{p,q}(n^b(n-1))$.

Тест простости са Гаусовим сумама

Битна тврђења и дефиниције које су у основи теста простости са Гаусовим сумама:

Теорема 4.10. *Нека је са $I(x)$ означен најмањи позитивни безквадратни цео број I за који важи да је производ простих бројева p, таквих да $p-1|I$, већи од x. Тада постоји број c такав да је $I(x) < (\log x)^{c \log\log\log x}$ за све $x > 16$.*

Тврђење 4.1. *Ако су p, q прости бројеви и $p|q-1$, онда је $G(p,q)\overline{G(p,q)} = q$.*

Тврђење 4.2. *Ако су p, q, n прости бројеви и $p|q-1$ и $\mathrm{nzd}(pq,n) = 1$, онда је $G(p,q)^{n^{p-1}-1} \equiv \chi_{p,q}(n) \pmod n$.*

Претходно тврђење се може посматрати као **аналогно малој Фермаовој теореми**.

Тврђење 4.3. *Ако су n, m позитивни цели бројеви, и n није дељив са m, и нека је $\xi_n^j \equiv \xi_n^k \pmod m$, тада је $\xi_n^j = \xi_n^k$.*

Дефиниција 4.16. Нека су p, q различити прости бројеви. Ако је $a \in \mathbb{Z}[\xi_p, \xi_q]/\{0\}$, где је $a = \sum_{i=0}^{p-2}\sum_{k=0}^{q-2} a_{ik}\xi_p^i\xi_q^k$, онда се са $c(a)$ означава највећи заједнички делитељ коефицијената a_{ik}. Такође, нека је $c(0) = 0$.

Скица теста са Гаусовим сумама:

Основни кораци алгоритма су иницијализација и припрема, у којима се конструише позитиван цео број I који је безквадратан и за који важи да је број F, који је једнак производу простих бројева q таквих да $q-1|I$, безквадратан и $F > \sqrt{n}$. Проверава се да ли је n прост фактор броја $I \cdot F$, ако јесте враћа резултат да је n прост. Извршава се и провера да ли је $\mathrm{nzd}(n, IF) \neq 1$. Затим је потребно за сваки прости фактор q броја F наћи најмањи позитивни примитивни корен g_q. У наредном кораку се за све просте бројеве p и q такве да $p|I$, $q|F$ и $p|q-1$, врше израчунавања заснована на провери Тврђења 4.2. Ако се не пронађу одговарајуће вредности, онда је број n сложен. Ако је пронађена тражена вредност, обележимо је са $w(p,q)$, прелази се на тражење максимума $w(p)$ броја $w(p,q)$ за све вредности простих бројева p и q, и обележи са $q_0(p)$ најмањи такав прост број q да важи $w(p) = w(p,q)$. Затим се проналази степен $l(p,q)$ од ξ_p, такав да је $\xi_p^{l(p,q)}$ по модулу n једнак одговарајућем (максималном) степену Гаусове суме $G(p,q)$. Наредни кораци подразумевају израчунавања одређених Гаусових сума, и nzd-а броја n и добијених бројева, где су израчунавања

заснована на дефиницији 4.16. На крају следи провера да ли број n има делитељ. Проверавају се само бројеви одређеног облика и користи Кинеска теорема о остацима за њихово конструисање.

Тест простости заснован на Гаусовим сумама је прави тест простости. Постоје многе варијанте и побољшања теста са Гаусовим сумама. Али, најзначајније побољшање доноси замена Гаусових сума Јакобијевим сумама, и самим тим израчунавања у мањем прстену.

Тест простости са Јакобијевим сумама

Тест прималности са Јакобијевим сумама (енгл. *Jacobi sum test*) је тест за доказивање простости. Основна идеја је тестирати скуп конгруенција које су аналогне малој Фермаовој теореми у одређеним циклотомичним прстенима. Јакобијеве суме $J(p,q)$ леже у много мањем прстену $\mathbb{Z}[\xi_p]$ у односу на Гаусове суме $G(p,q)$, па су зато израчунавања једноставнија.

Скица теста са Јакобијевим сумама:[44]

Изабрати целе бројеве t и s тако да је $s = \prod q$ производ простих бројева q са особином да $q-1|t$ и $s > \sqrt{n}$. За сваки пар (p^k, q), такав да је $q|s$ прост делитељ од s и p^k највећи степен простог броја p који дели $q-1$, извршити тест са Јакобијевим сумама који се, грубо говорећи, састоји од дизања елемената из $\mathbb{Z}[\xi_{p^k}]/n\mathbb{Z}[\xi_{p^k}]$ на степен n. На крају проверити да цели бројеви $1 \leq r_i \leq \sqrt{n}$ не деле n, где је $r_i \equiv n^i \pmod{s}$ за $i = 1, 2, ..., t$.

Босма и ван дер Хулст су направили практична побољшања алгоритма на неколико начина. На првом месту, тест са Јакобијевим сумама може се комбиновати са Лукас-Лемеровим типом тестова. Пошто су модификовани Лукас-Лемерови тестови обично доста ефикаснији од тестова са Јакобијевим сумама, ово може бити велика добит.

Са друге стране, могуће је смањити количину израчунавања у оквиру теста. Уместо да се ради n-то степеновање у $\mathbb{Z}[\xi_{p^k}]/n\mathbb{Z}[\xi_{p^k}]$ степена $\phi(p^k)$ над $\mathbb{Z}/n\mathbb{Z}$, могуће је радити у екстензијама прстена степена једнаког реду ($n \in \mathbb{Z}/p^k\mathbb{Z}$), који је делитељ од $\phi(p^k)$ и може бити знатно мањи.

Треће, могуће је комбиновати неколико тестова за парове (p^k, q) у један већи тест, под претпоставком да су прости бројеви p различити.

Тест прималности заснован на Јакобијевим сумама је практичан у смислу да се прималност бројева од неколико стотина децималних цифара може обавити на рачунару за само неколико минута. Али, овај тест није лако програмски имплементирати као што је то, на пример, могуће за пробабилистички Милер-Рабинов тест.

Оригинални тест са Јакобијевим сумама су дали Адлеман, Померанс, и Румли 1983. године — то је познати **APR тест**. Време извршавања овог теста је ограничено са $O((\log n)^{c \log\log\log n})$, за неку позитивну константу c. Ово је скоро полиномијалан алгоритам, пошто се експонент $\log\log\log n$ понаша као константа за све употребљене вредности од n [17]. Аутори су дали пробабилистичку и детерминистичку верзију овог алгоритма, и времена извршавања оба теста су ограничена датом границом. Такође, ако пробабилистички алгоритам врати резултат, онда он тачно одређује да ли је задати број прост или сложен [13]. Cohen и Lenstra су поједноставили алгоритам, теоријски и алгоритамски. Они су такође дали и програмску имплементацију овог теста. Даља побољшања теста заснованог на Јакобијевим сумама дали су Босма и ван дер Хулст, који су 1998. године имплементирали алгоритам који веома брзо доказује прималности бројева од неколико стотина или хиљада цифара [7, 28].[45]

4.9 AKS тест простости

AKS тест представља теоријску прекретницу у тестирању простости бројева. У 2002. години је одговорено на питање које је дуго било отворено: да ли се у полиномијалном времену може доказати или оповргнути прималност произвољног целог броја. Неки од претходних алгоритама су стигли близу (ECPP је скоро полиномијалан). Агравал, Кајал и Саксена су позитивно одговорили на питање дајући једноставан полиномијални алгоритам, тзв. AKS тест прималности. Њихов рад, у коме су изложили овај алгоритам, имао је једноставан наслов: "PRIMES is in P".

[44] За више информација, видети *"Fast Primality testing"*, Bosma, van der Hulst, Springer-Verlag, 1998.

[45] За више информација, видети и *"Primality testing and Jacobi sums"*, H. Cohen, H. W. Lenstra, Jr., *Mathematics of computation*, Vol. 42, 1984, pp. 297–330.

Кључна важност AKS-а је да је то први објављени алгоритам за доказивање прималности који је истовремено:

- **Општи** — може се применити на било коју *класу* простих бројева.
- **Полиномијалан** — асимптотско време извршавања алгоритма је полиномијално.
- **Детерминистички** — детерминистички алгоритам који увек даје тачан одговор.
- **Безуслован** — не ослања се на било коју недоказану хипотезу као што је генерализована Риманова хипотеза.

Претходни алгоритми су задовољавали нека три од ових својстава, али не сва четири. Ипак, у пракси овај алгоритам је и даље прилично спор.

Следе тврђења која су основа за AKS тест.

Биномни тест простости

Дефиниција 4.17. Биномни коефицијент са параметрима n и $k \leq n$ је цео број дефинисан са: $\binom{n}{k} = \frac{n!}{k!(n-k)!}$.

Биномни коефицијенти се често представљају помоћу Паскаловог троугла, где је k-ти елемент n-ог реда је једнак $\binom{n}{k}$. За биномне коефицијенте важи и $\binom{n}{k} = \binom{n}{n-k}$ и $\binom{n+1}{k+1} = \binom{n}{k} + \binom{n}{k+1}$.

Теорема 4.11. *Цео број n је прост ако и само ако је $\binom{n}{k} \equiv 0 \pmod{n}$ за све $1 \leq k \leq n-1$.*

Доказ. $\boxed{\Rightarrow}$ Нека је n прост број и нека је $1 \leq k \leq n-1$. По дефиницији је $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ цео број. Бројилац овог разломка садржи делитељ броја n, али именилац не садржи јер је сваки прости фактор у $k!$ и $(n-k)!$ мањи од n. Делитељ броја n у бројиоцу не може се скратити са имениоцем зато што је n прост, и зато је $\binom{n}{k}$ умножак од n, одакле следи да је $\binom{n}{k} \equiv 0 \pmod{n}$.
$\boxed{\Leftarrow}$ Претпоставимо сада да је $\binom{n}{k} \equiv 0 \pmod{n}$ за све $1 \leq k \leq n-1$ и претпоставимо да је n сложен. Нека је p прост фактор од n, и нека је p^c највећи степен броја p који дели n. По дефиницији је $\binom{n}{p} = \frac{n!}{p!(n-p)!} = \frac{n(n-1)(n-2)\cdots(n-p+1)}{p!}$. У бројиоцу има тачно c фактора броја p, а у имениоцу има само један. Зато $\binom{n}{p}$ има тачно $c-1$ делитеља броја p, и $p^c \not| \binom{n}{p}$. Ово значи да $\binom{n}{p}$ није садржитељ броја p^c, па зато ни броја n. Ово је у контрадикцији са претпоставком да је $\binom{n}{k} \equiv 0 \pmod{n}$ за све $1 \leq k \leq n-1$, па је претпоставка погрешна. Одатле следи да је n прост број. □

Ова теорема даје једноставан Биномни тест простости (енгл. *Binomial test*) заснован на биномним коефицијентима: за дати цео број n, израчунати $\binom{n}{k} \pmod{n}$ за све $2 \leq k \leq n-1$. Ако су све вредности нула, број је прост. Чим вредност постане различита од нуле, број мора бити сложен. Заправо, ова теорема даје и метод факторизације: ако је $\binom{n}{k} \neq 0 \pmod{n}$, онда је k делитељ броја n. Итеративно тражећи најмањи делитељ могу се пронаћи сви прости фактори броја n.

Да би се на овај начин тестирала прималност целог броја n, мора се израчунати $n-2$ коефицијента, тако да је време извршавања $O(2^s)$, $s = \log_2 n$. Он није бржи од поновљеног дељења, па је непрактичан као тест прималности. Ипак, овај тест даје теоријску основу за AKS тест.

Тест простости са полиномима

Полином са целобројним коефицијентима је функција $f(x) = a_n x^n + a_{n-1}x^{n-1} + ... + a_1 x + a_0$, $a_i \in \mathbb{Z}$ за свако i.

Дефиниција 4.18. Два полинома $f(x) = \sum_{i=1}^{n} a_1 x^i$ и $g(x) = \sum_{i=1}^{n} b_1 x^i$ су **конгруентна по модулу** n ако је сваки пар одговарајућих коефицијената конгруентан по модулу n, то јест $f(x) \equiv g(x) \pmod{n}$ значи да је $a_i \equiv b_i \pmod{n}$ за свако i.

Теорема 4.12 (Биномна теорема)**.** *Ако је n позитиван цео број, онда је $(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^k y^{n-k}$.*

Теорема 4.13. *Ако је $\mathrm{nzd}(a,n) = 1$, онда је $a^k \neq 0 \pmod{n}$ за свако $k \geq 0$.*

Доказ. Претпоставимо да је $a^k \equiv 0 \pmod{n}$ за неко k, тј. да $n|a^k$. Одавде следи да n и a^k имају заједничке факторе, што је контрадикција са условом теореме да је $\mathrm{nzd}(a,n) = 1$. Зато је претпоставка нетачна, па важи тврђење теореме. □

AKS тест је у великој мери заснован на следећој чињеници:

Теорема 4.14. *Нека је* $\text{nzd}(a,n)=1$. *Тада је* n *прост број ако и само ако је* $(x+a)^n \equiv x^n + a \pmod{n}$.

Доказ. $\boxed{\Rightarrow}$ Претпоставимо да је n прост. На основу биномне теореме, $(x+a)^n = \sum_{k=0}^{n} \binom{n}{k} a^k x^{n-k}$. Израз са x^n је $\binom{n}{0}x^n a^0 = x^n$, а израз са x_0 је $\binom{n}{n}x^0 a^n = a^n$. Као што смо већ показали, $\binom{n}{k} \equiv 0 \pmod{n}$ за све $1 \leq k \leq n-1$, па су сви остали изрази у суми конгруентни 0 по модулу n. Зато је $(x+a)^n \equiv x^n + a^n \pmod{n}$, као што смо већ показали и у уводном поглављу. Како је n прост број, на основу мале Фермаове теореме је $a^n \equiv a \pmod{n}$. Одатле важи тврђење теореме.
$\boxed{\Leftarrow}$ Претпоставимо сада да је $(x+a)^n \equiv x^n + a \pmod{n}$, и $\text{nzd}(a,n)=1$. Користећи биномну теорему записује се $(x+a)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k}a^k \equiv x^n + a \pmod{n}$, па следи да је $\binom{n}{k}a^k \equiv 0 \pmod{n}$ за све $1 \leq k \leq n-1$. Како важи услов $\text{nzd}(a,n)=1$, онда је $a^k \neq 0 \pmod{n}$ за свако k, па одатле следи да је $\binom{n}{k} \equiv 0 \pmod{n}$ за све $1 \leq k \leq n-1$. На основу Теореме 4.11 ово значи да је n прост број. □

Претходна теорема даје тест прималности са полиномима (енгл. *polynomial test*) који је компликованији од биномног теста, али се есенцијално своди на исти алгоритам. За цео број n, једноставно се одабира неки број a који је узајамно прост са n, онда рачунају сви коефицијенти у развоју $(x+a)^n \pmod{n}$ осим првог и последњег. Ако неки од коефицијената није нула, онда n није прост. Ако су сви коефицијенти нула, n је прост број. Овај тест је неефикаснији од Биномног теста простости због додатних израчунавања.

AKS тест простости

AKS тест је много ефикаснија модификација теста заснованог на Теореми 4.14. Концепт полиномијалне конгруенције је најважнији концепт који доноси побољшање AKS теста у односу на Полиномијални тест.

Дефиниција 4.19. Ако су $f(x)$ и $g(x)$ полиноми, $f(x) \pmod{g(x)}$ је остатак полиномијалног дељења $f(x)/g(x)$.

Дефиниција 4.20. За дате полиноме $f(x)$, $g(x)$ и $h(x)$, се каже да су $f(x)$ и $g(x)$ конгруентни по модулу $h(x)$ и n ако су њихови остаци при дељењу са $h(x)$ конгруентни по модулу n:

$$f(x) \equiv g(x) \pmod{(h(x),n)} \text{ ако } [f(x) \pmod{h(x)}] \equiv [g(x) \pmod{h(x)}] \pmod{n}.$$

Потребно је редуковати број коефицијената који се израчунавају са $O(2^s)$ на $O(s^k) = O(\log_2^k n)$.

Ако је $f(x) \in \mathbb{Z}[x]$ произвољан моничан полином, онда из Теореме 4.14 следи да је $(x+a)^n \equiv x^n + a \pmod{f(x), n}$ за сваки цео број a и n прост број. Ова конгруенција даје неопходан, али не и довољан услов за прималност броја n.

Приметимо да не се не мора узети полином првог степена као $f(x)$. На пример, може се узети да је $f(x) = x^r - 1$ за неки мали број r, тако да се сада имплицитно ради са r-тим коренима јединице. Такође, може се изабрати одговарајуће r и проверити дата конгруенција за сваки број a до неке границе.

Теорема 4.15. *Ако је* n *прост број и* a, n *су цели бројеви, онда је* $(x+a)^n \equiv (x^n+a) \pmod{(x^r-1, n)}$.

Доказ. Ово директно следи из Теореме 4.14 и дефиниције полиномијалне конгруенције. □

Нажалост, ова теорема не обезбеђује тест за доказивање прималности зато што обрат не мора да важи. Ипак, Агравал, Кајал, и Саксена су показали да се детерминистички резултат може постићи са одређеним вредностима броја r и скупом вредности за a. Треба изабрати одговарајуће r и проверити конгруенцију за сваки цео број a до одређене границе (вредности за r и a су ограничене полилогаритамским изразом за n). Следе две теореме на којима се заснива AKS алгоритам [6]:

Теорема 4.16 (Agrawal, Kayal, Saxena)**.** *Нека су* $n \geq 2$ *и* $r > 0$ *узајамно прости цели бројеви такви да је ред броја* n *у* $\mathbb{Z}_r^*$ *већи од* $\log_2^2 n$, *и* $(x+a)^n \equiv (x^n + a) \pmod{(x^r-1, n)}$ *важи за сваки цели број* a *такав да је* $0 \leq a \leq \sqrt{\phi(r)}\log_2 n$. *Ако* n *има прост фактор* $p > \sqrt{\phi(r)}\log_2 n$, *онда је* $n = p^m$ *за неки цео број* m. *Специјално, ако* n *нема прост делилац* a *такав да је* $1 \leq a \leq \sqrt{\phi(r)}\log_2 n$ *и* n *није прави степен (енгл. proper power), онда је* n *прост [6, 17].*

Теорема 4.17. *За дати цео број $n \geq 3$, нека је r најмањи цео број реда n у $\mathbb{Z}_r^*$ већи од $\log_2^2 n$. Тада је $r \leq \log_2^5 n$ [6, 17].*

Претходна теорема омогућава да се AKS алгоритам извршава у полиномијалном времену.

АЛГОРИТАМ: AKS

УЛАЗ: цео број $n \geq 2$.

ИЗЛАЗ: "да" или "не" одговор на питање: "Да ли је n прост број?"

[**Тест степена**]

if $(n = a^b$ за $n \in \mathbb{N}$ и $b \geq 2)$ **return** ("не");

// тестирамо да ли је n облика a^b само за $b \leq \log_2 n$. [17]

[**Припрема**]

Наћи најмањи прост број r такав да је ред броја n у $\mathbb{Z}_r^*$ већи од $\log_2^2 n$;

if $(\exists a$: $\mathrm{nzd}(a, n) \neq 1, n$ за неко $a \leq r)$ **return** ("не"); // доказали смо да је n сложен

if $(n \leq r)$ **return** ("да");

[**Биномна конгруенција**]

while $(1 \leq a \leq \sqrt{\phi(r)} \log_2 n)$ **do**

if $((x + a)^n \neq x^n + a \pmod{x^r - 1, n})$ **return** ("не");

return ("да");

Овај алгоритам је дат на основу [6] и објашњења из [17]. После објављивања AKS алгоритма, појавиле су се многе анализе и варијанте овог алгоритма, па је настао и термин AKS класа алгоритама. Циљ је смањење сложености експоната k у изразу за сложеност. Сложеност оригиналног Аграваловог алгоритма је била $O(\log^{12+\epsilon} p)$, али је од тада значајно смањена на $O(\log^{6+\epsilon} p)$ за опште целе бројеве[46], или $O(\log^{4+\epsilon} p)$ за одређене целе бројеве, и за опште целе бројеве али са скоро занемарљивом могућношћу да алгоритам може да врати резултат "можда".

У варијанти AKS теста коју је дао Ленстра[47], прво се проверава да n није прави степен неког позитивног простог броја, онда се бира прост број r такав да је мултипликативни ред броја n по модулу r већи од $v = round(\log_2^2 n)$. Затим се проверава полиномијална релација $(x - a)^n = x^n - a \pmod{x^r - 1, n}$ за $a \in [1, \phi(r) - 1]$. Тада је n прост број ако и само ако је ова полиномијална релација тачна за све такве бројеве a. Овај алгоритам су у програмском језику C имлементирали Крандал и Померанс, а кôд програма се може наћи на интернет страници `http://www.perfsci.com/free-software.asp#primekit`.

4.10 Други тестови простости

Многе теореме које смо представили у поглављу о теорији бројева могу чинити основу за неке тестове прималности. То је случај са Протовом теоремом, која јесте тест којим се веома ефикасно проналазе прости Протови бројеви, као и Вилсонова теорема, која се због своје неефикасности не користи.

Поред n - 1 теста и n + 1 теста, постоји и **комбиновани тест простости** n^2 - 1 тест. Сва три теста захтевају потпуну или делимичну факторизацију одређеног броја који је повезан са задатим бројем n чију прималност тестирамо. Ови тестови се понекад називају и **класичним тестовима простости**. Због зависности од факторизације, ови тестови нису ефикасни у општем случају, али за одређене класе бројева су најбржи познати тестови за доказивање простости, као што је то за бројеве облика $k \cdot 2^n - 1$, $k \cdot 2^n + 1$, $k < 2^n$, итд.

Такође постоје тестови простости који доказују простост бројева и који се извршавају у полиномијалном времену, али се њихова коректност ослања на тачност Риманове хипотезе, односно проширене Риманове хипотезе (енгл. *Extended Riemann Hypothesis, ERH*).

Дефиниција 4.21. Ако је χ Дирихлеов карактер по модулу m, онда је са $L(s, \chi) = \sum_{n=1}^{\infty} \frac{\chi(n)}{n^s}$ дефинисана **Дирихлеова L-функција**.

Дефиниција 4.22 (Проширена Риманова хипотеза). Нека је χ произвољни Дирихлеов карактер. Онда нуле Дирихлеове функције $L(s, \chi)$ за $Re(s) > 0$ леже на линији $Re(s) = \frac{1}{2}$.

[46] Ленстра и Померанс су 2005. године модификовали и убрзали AKS алгоритам.

[47] *"On the implementation of AKS-class primality tests"*, Crandall, Papadopoulos.

Милеров тест (енгл. *Miller primality test*) је тест за доказивање простости под условом да важи *ERH*. Овај тест **систематском провером** сведока сложености a до $\min\{\lfloor 2\ln^2 n \rfloor, n-1\}$ доказује да је n прост или сложен [17]. Прво је настао Милеров тест, па је онда због практичних потреба креиран вероватносни Милер-Рабинов тест, који је бржи јер се очекује да се случајним избором брзо нађе сведок сложености броја који се тестира. Соловеј-Штрасенов тест може се такође модификовати да буде тест за доказивање простости, под условом да важи *ERH*.

Тестови простости засновани на елиптичким кривама, у којима теорема за елиптичке криве аналогна Поклингтоновој теореми чини основу теста, су веома значајни. Ови тестови спадају у најчешће употребљаване и најбрже методе за доказивање прималности. **ECPP** тест (енгл. *Elliptic curve primality proving algorithm*), и његова веома ефикасна варијанта **Еткинов тест**[48], је општи тест прималности који се често користи у пракси и не захтева да број који се испитује буде у некој специјалној форми као што је то случај са, на пример, Пепиновим тестом. ECPP тест је бољи од вероватносних тестова јер као резултат даје кратак сертификат прималности који се онда може користити да ефективно потврди (енгл. *verify*) прималност неког броја. Хеуристичка оцена времена извршавања овог алгоритма за показивање прималности целог броја n је $O((\log n)^{6+\epsilon})$ битских операција за неко $\epsilon > 0$, и експонент се може за неке хеуристичке аргументе смањити. Може се рећи да је ECPP *скоро* полиномијалан тест за доказивање прималности. ECPP тест је употребљен за доказивање прималности бројева са више од 25000 цифара [17].

Сертификат простости (енгл. *primality certificate*) се креира да би било могуће брзо потврдити доказ простости без понављања целог поступка доказивања, тако што се генерише листа свих битних корака (енгл. *stepping stones*) у доказивању, заједно са одређеним објашњењима.

Постоје и многи други тестови који овде нису поменути, а који се могу ефикасно искористити за тестирање да ли је задати број прост. Комбиновање различитих тестова да би се добили бржи или сигурнији алгоритми, специјални случајеви општих тестова простости који за бројеве одређеног облика дају доказе о простости, или су значајно бржи од своје опште верзије, су уобичајне методе које се користе да би се постигли жељени циљеви у тестирању простости.

4.11 Преглед новијих резултата у откривању великих простих бројева

Сада наводимо неке рекордно велике просте бројеве. Први прост број са више од 10 милиона цифара био је $2^{43112609}-1$, пронађен 23. августа 2008. То је Мерсенов прост број са 12,978,189 цифара.

И пре појаве електронских рачунара Мерсенови бројеви су били у центру многих истраживања везаних за велике просте бројеве. [49]

Број	Број цифара	Година	Доказао	Метод
$2^{17}-1$	6	1588	*Cataldi*	пробно дељење
$2^{19}-1$	6	1588	*Cataldi*	пробно дељење
$2^{31}-1$	10	1772	*Euler*	пробно дељење++
$(2^{59}-1)/179951$	13	1867	*Landry*	пробно дељење++
$2^{127}-1$	9	1876	*Lucas*	Лукасови низови
$(2^{148}+1)/17$	44	1951	*Ferrier*	Протова теорема

Рекорди постављени пре електронских рачунара

Године 1951, Милер и Вилер су започели доба електронског проналажења простих бројева, доказавши да су бројеви $k \cdot M_{127}+1$, прости за $k = 114, 124, 388, 408, 498, 696, 738, 774, 780, 934, 978$, као и број $180(M_{127})^2+1$, $M_{127} = 2^{127}-1$, тадашњи рекорд од 79 цифара. Овај рекорд је оборен већ следеће године, када је Робинсон открио пет нових Мерсенових простих бројева, користећи *SWAC* (енгл. *Standards Western Automatic Computer*).

У Априлу 2012. године пронађен је 47. познат Мерсенов прост број $2^{42,643,801}-1$, са 12,837,064 цифара, а у јануару 2013. је доказано да је број $2^{57,885,161}-1$ прост број са $17,425,170$ цифара.

[48] Еткин и Мореин су дали веома ефикасну имплементацију овог теста (енгл. *Atkin-Morain primality proving test*).

[49] Потпуније табеле и више информација може се наћи на страницама `http://primes.utm.edu/notes/by_year.html`, `http://mersennewiki.org/`.

Интересантни пројекти у вези са откривањем простих бројева:

- GIMPS, *Great Internet Mersenne Prime Search* — велико интернет истраживање Мерсенових простих бројева. PrimeNet је грид систем за GIMPS, који се састоји од десетина хиљада PC рачунара, лаптопова и сервера.[50]
- $100MDPP$ — *"100 Million Digit Prefactor Project"*, пројекат који факторише Мерсенове експоненте како би их припремио за евентуално
  проналажење првог простог броја са 100 милиона цифара.
- 12121 Search — пројекат који тражи просте бројеве у облику: $121 \cdot 2^n - 1$. Тренутно је њихов највећи број $121 \cdot 2^{2033941} - 1$, са 510,399 цифара.
- 27121 Search — пројакат који тражи просте бројеве у облику $27 \cdot 2^n - 1$, $27 \cdot 2^n + 1$, и $121 \cdot 2^n + 1$.

Наравно, нови рекорди не би требало да буду сами себи циљ, већ да, посредно или непосредно, омогуће или мотивишу решавање отворених проблема теорије бројева, или креирање сигурнијих криптографских алгоритама.

У наредној табели је дат кратак преглед проналажења простих бројева употребом рачунара.

Број	Број цифара	Година	Машина	Доказао
$180(M_{127})^2 + 1$	79	1951	*EDSAC*1	*Miller* и *Wheeler*
M_{1279}	386	1952	*SWAC*	*Robinson*
M_{3217}	969	1957	*BESK*	*Riesel*
M_{4423}	1,332	1961	*IBM*7090	*Hurwitz*
M_{11213}	3,376	1963	*ILLIAC* 2	*Gillies*
M_{19937}	6,002	1971	*IBM*36091	*Tuckerman*
M_{21701}	6,533	1978	*CDC Cyber* 174	*Noll* и *Nickel*
M_{44497}	13,395	1979	*Cray* 1	*Nelson* и *Slowinski*
M_{132049}	39,751	1983	*Cray* $X - MP$	*Slowinski*
$391581 \cdot 2^{216193}$-1	65,087	1989	*Amdahl* 1200	*Amdahl Six*
M_{756839}	227,832	1992	*Cray*-2	*Slowinski* и *Gage*
$M_{1257787}$	378,632	1996	*Cray T*94	*Slowinski* и *Gage*
$M_{1398269}$	420,921	1996	*Pentium* (90 *Mhz*)	*Armengaud*, *Woltman*, [*GIMPS*]
$M_{24036583}$	7,235,733	2004	*Pentium* 4 (2.4*GHz*)	*Findley*, *Woltman*, *Kurowski*, [*GIMPS*, *PrimeNet*]
$M_{32582657}$	9,808,358	2006	*Pentium* 4 (3*GHz*)	*Cooper*, *Boone*, *Woltman*, *Kurowski*, [*GIMPS*, *PrimeNet*]
$M_{43112609}$	12,978,189	2008	*Intel Core*2*Duo* E6600 (2.4*GHz*)	*E. Smith*, *Woltman*, *Kurowski*, [*GIMPS*, *PrimeNet*]
$M_{42643801}$	12,837,064	2012	*Intel Core*2 (3.0*GHz*)	*Odd Magnar Strindmo*, [*GIMPS*]

Неки од рекорда постављени помоћу електронских рачунара

Амдалову шесторку (енгл. *Amdahl six*) чине J. Brown, C. Noll, B. Parady, G. Smith, J. Smith и S. Zarantonello.

[50] За више информација, видети `http://en.wikipedia.org/wiki/Great_Internet_Mersenne_Prime_Search`.

5 Паралелизација алгоритама

Ово поглавље је у основи превод материјала из [23].

5.1 О паралелном програмирању

Може се рећи да је паралелно израчунавање употреба паралелних рачунара ради решавања проблема који на секвенцијалном рачунару дуго трају, на пример због обраде огромне количине података или зато што је за њихово решавање потребна велика процесорска моћ. Како су основни проблеми везани за показивање простости великих бројева брзина и количина меморије који су потребни, а за сита за просејавање простих бројева и велика количина података која се обрађује, паралелно програмирање помаже у успешној реализацији ових алгоритама и постизању бољих резултата.

Разликује се **паралелизам ниског нивоа** — систем преклапања извршавања инструкција, коришћење више водова инструкција и уградња више функционалних јединица у један процесор, и **паралелизам високог нивоа** — коришћење више процесора у једном рачунарском систему.

Модели паралелних рачунара

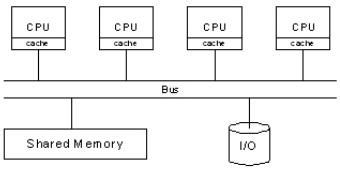
Појавом VLSI (енгл. *Very Large Scale Integration*) крајем 70-тих година прошлог века паралелни рачунари су постали занимљиви ширем кругу програмера. Стварање веома *снажних* паралелних рачунара почиње 1980-тих година, када настају комерцијалне компаније које лансирају своје паралелне рачунаре, и паралелни рачунари постају незаменљиви за многа научна истраживања и симулације, али и за многе комерцијалне апликације.

Паралелни рачунар је мултипроцесорски рачунарски систем који подржава паралелно програмирање.

Две *најважније* категорије паралелних рачунара су мултипроцесори (енгл. *multiprocessors*) и мултирачунари (енгл. *multicomputers*).

Мултипроцесор је рачунар са више процесора који имају **заједничку дељену меморију** (енгл. *shared memory*). Једна од њихових битних особина је и да иста адреса на два различита процесора означава исту меморијску локацију. Постоје два основна типа мултипроцесора: **централизовани** и **дистрибуирани мултипроцесори**.

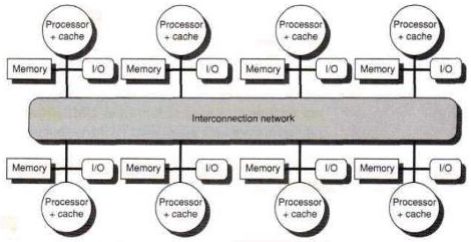
Централизовани мултипроцесори, или **симетрични мултипроцесор SMP**, је интегрисани систем у коме сви процесори деле **приступ једној глобалној меморији** која подржава комуникацију и синхронизацију међу процесорима, па се при конструкцији ових мултипроцесора мора водити рачуна о проблему усаглашености кеш меморије (енгл. *cache coherence problem*) и проблему синхронизације процесора.



Симетрични мултипроцесор [23]

Проблем усаглашености кеш меморије — како сваки процесор види меморију кроз свој кеш, мора се обезбедити да различити процесори немају различите вредности за исту меморијску локацију. **Синхронизација процесора** — при извршавању кооперативних процеса могу бити потребне различите врсте синхронизације као што су међусобно искључивање (енгл. *mutual exclusion*) и синхронизација помоћу баријера (енгл. *barrier synchronization*).

Дистрибуирани мултипроцесори имају дистрибуирану меморију међу процесорима, али и даље један глобални адресни простор, где сваки процесор може директно да приступи целокупној меморији.



Мултипроцесор са дистрибуираном меморијом [23]

И код ових система иста адреса на различитим процесорима означава исту меморијску локацију. Овај тип система се још назива **мултипроцесор са неравномерним приступом меморији** (енгл. *nonuniform memory access (NUMA) multiprocessor*) зато што приступ меморији значајно варира у зависности да ли је референцирана адреса у локалној меморији процесора или је у локалној меморији неког другог процесора. Овде се проблем усаглашености кеш меморије најчешће решава протоколом заснованом на директоријуму (енгл. *directory-based protocol*) — један директоријум садржи заједничке информације о сваком меморијском блоку који може бити кеширан.

Мултирачунар је паралелни рачунар састављен од више рачунара повезаних мрежом[51]. Процесори на различитим рачунарима комуницирају један са другим слањем порука.

Код мултирачунара, за разлику од нпр. NUMA мултипроцесора, постоје раздвојени адресни локални простори — сваки процесор има директан приступ само својој локалној меморији. Иста адреса на различитим процесорима означава две различите меморијске локације.

Први мултирачунари су били **асиметрични** (Intel iPSC и nCUBE/ten) и имали су битне недостатаке. Један контролни рачунар (енгл. *front-end manager*) је контролисао мрежу помоћних, тј. позадинских (енгл. *back-end*) рачунара, који су употребљавани само за извршавање паралелних програма. Неки од недостатака ових мултирачунара били су ограничена скалабилност, квар на контролном рачунару значио би да је цео систем ван употребе, отежано проналажење грешака у програмима (енгл. *debugging*), различити програми за контролни рачунар и помоћне рачунаре. **Симетрични мултирачунари** решавају многе од проблема који су се појавили код асиметричних мултирачунара. Код ових система сваки рачунар има исти оперативни систем и исту функционалност, бољу подршку при откривању грешака и омогућује лакше програмирање, али такође имају своје мане.

Код једноставних, неспецијализованих кластера (енгл. *commodity clusters*) за разлику од комерцијалних мултирачунарских система, да би се направила локална мрежа (енгл. LAN) користе се рачунари, свичеви (енгл. *switch*), и друга опрема која је широко доступна. Као мрежа се обично користи *Fast Ethernet*, *Gigabit Ethernet*, или *Myrinet*.

Може се рећи да је најбоље решење за једноставан кластер *хибридни* модел који има особине и асиметричног и симетричног дизајна. Треба разликовати једноставне (енгл. *commodity*) кластере од мреже радних станица (енгл. *network of workstations*).

Поред мултипроцесора и мултирачунара, интересантни су и **векторски** рачунари. Векторски рачунари су рачунари који имају могућност истовременог обављања операција над скупом података.

Напомена 5.1. Сви системи са више процесора морају омогућити паралелизам, комуникацију међу процесорима, и синхронизацију. Комуникација међу процесорима се може одвијати помоћу **заједничког канала** (енгл. *shared medium*), где је дозвољено слање само једне поруке у једном тренутку. Етернет је пример ове топологије. Колизија (тј. судар) порука може значајно умањити перформансе овог система. Насупрот овом приступу, постоји мрежа помоћу **свичева**, где је могућа директна комуникација између два процесора и истовремено слање више порука између различитих парова процесора. Мрежни свич је телекомуникациони уређај који прима поруку од било ког уређаја повезаног са њим и онда шаље ту поруку само уређају коме је порука послата.

Мрежа са свичевима може бити приказана помоћу графа где сваки чвор (енгл. *node*) представља процесор или свич, док гране (енгл. *edge*) представљају комуникационе канале. Разликују се

[51] У општем случају обрада података у рачунарској мрежи може се вршити на три начина — централизована обрада, обрада на мрежи равноправних рачунара (енгл. *peer-to-peer*) или у клијент – сервер окружењу.

директна топологија, код које је однос броја свичева и процесора 1 : 1, и **индиректна** топологија, код које је већи број свичева. Примери директних топологија су дводимензионални процесорска мрежа (тј. *2D Mesh*), хиперкоцка, мрежа са укрштаљкама (енгл. *shuffle-exchange* или *perfect shuffle*), док су хипердрво, лептир мрежа (енгл. *butterfly*), мрежа са структуром бинарног дрвета — индиректне топологије.

Критеријуми за оцењивање мрежних топологија са свичевима су: пречник (енг. *diameter*), који представља највећу удаљеност између два свича, ширина пресека (енг. *bisection width*), који представља најмањи број *грана/жица* (енгл. *edge*) између свича које треба уклонити да би се мрежа поделила на два *одвојена* дела, број грана по свичу, и дужина гране. Ширина пресека се може дефинисати и као број порука које се истовремено могу послати, не користећи исте гране, тј. не користећи исте процесоре за комуникацију. Добра мрежна топологија са свичевима има мали пречник, велику ширину пресека, константан број грана по свичу (тј. број грана не зависи од величине мреже), и константну дужину гране.

Флинова класификација

Иако је по мишљењу неких стручњака застарела, Флинова класификација је још увек битна. Према броју токова инструкција и броју токова података рачунарски системи се могу поделити на следећи начин:

	Једна инструкција	Више инструкција
Један податак	**SISD**	**MISD**
Више података	**SIMD**	**MIMD**

Флинова класификација (енгл. *Flynn's taxonomy*)

Рачунари SISD су једнопроцесорски рачунари, док остале класе представљају различите моделе паралелних система. Једнопроцесорски рачунари такође могу подржавати одређене облике конкурентног извршавања. Преклапање фаза инструкција (енгл. *pipelining*) и предчитање инструкција (енгл. *instruction prefetching*) су само неки од примера конкурентности на SISD рачунарима.

Рачунари SIMD су рачунарски системи који истовремено извршавају једну инструкцију над различитим подацима (енгл. *single instruction, multiple data*). Низ процесора (енгл. *processor array*) је рачунарски систем код кога је један процесор (тзв. контролни процесор) повезан са скупом индентичних, синхронизованих процесорских елемената који истовремено обављају исту операцију над различитим подацима. Примери ових SIMD рачунара су ILLIAC IV, Connection Machine, модели 1 и 2 (CM-1 и CM-2, који су имали један процесор и хиљаде аритметичко-логичких јединица). Због неколико битних недостатака, низови процесора се више не користе као паралелни рачунари опште намене. Пример *pipelined* векторског рачунара (векторског рачунара са преклапањем извршавања инструкција) је Cray-1. У многим класификацијама, *pipelined* векторски рачунари нису паралелни рачунари [23, 24, 26].[52]
Данас се SIMD *инструкције* веома често користе у обради 3D графике. Више о тзв. проточном процесирању (енгл. *stream processing*) видети на [26].

Рачунари MISD су имали мало комерцијалног успеха, и веома мали број паралелних система је заснован на овом моделу.

Класи рачунара MIMD припадају скоро сви комерцијални паралелни рачунари. Рачунаре SM MIMD (енгл. *shared memory MIMD*), као што је речено називамо мултипроцесорима (енгл. *tightly-coupled multiprocessor systems*), док мрежне MIMD рачунаре називамо мултирачунарима (енгл. *loosely-coupled machines*).

Пример мрежног MIMD је кластер Beowulf. **Кластер рачунар** представља групу повезаних аутономних рачунара који раде заједно као један паралелни рачунар. Најчешћи тип кластера је Beowulf кластер — кластер имплементиран на више идентичних комерцијалних рачунара, спојених преко TCP/IP Етернет локалне мреже. Beowulf технологију развили су Стерлинг и Бекер. Обично се користи оперативни систем Linux, софтверски пакет PVM (енгл. *parallel virtual machine*) и стандард MPI, и организован је тако да постоји један мастер, тј. сервер чвор (енгл. *management node*) и више клијент чворова (енгл. *computational nodes*).

[52] Постоје и друге класификације за векторске процесоре. За више информација, видети `http://www.phy.ornl.gov/csep/CSEP/CA/CA.html`.

Грид обрада представља најдистрибуиранији облик паралелне обраде. За рад на проблему грид обрада користи рачунаре удаљене километрима, који су повезани помоћу Интернета.

Занимљиво је поменути да паметни телефони (енгл. *smartphones*), таблети, *iPod* уређаји имају вишејезгарне процесоре (енгл. *multi-core processors*).

Основни модели паралелних програма. Паралелизација алгоритма

Дефиниција 5.1. Паралелно програмирање је програмирање на језику који омогућава експлицитно задавање како различити процеси треба да истовремено извршавају различите делове израчунавања.

Дефиниција 5.2. Граф зависности података (енгл. *data dependence graph*) је директни граф у коме сваки чвор (енгл. *vertex*) представља задатак који треба да се изврши. Грана од чвора a до чвора b значи да задатак a мора бити извршен пре него што задатак b почне (каже се још да задатак b зависи од задатка a).

Дефиниција 5.3. Паралелизам података (енгл. *data parallelism*) је техника чија је суштина дељење великог скупа података на мање и паралелна примена исте операције на те скупове (независни задаци примењује исту операцију на различите елементе *великог* скупа података).

Дефиниција 5.4. Функционални паралелизам (енгл. *functional parallelism*) је техника када независни задаци примењују различите операције на различите скупове података.

Дефиниција 5.5. Преклапање фаза инструкција (енгл. *pipelining*) је техника поделе инструкције у фазе и паралелно извршавање фаза и иначе зависних задатака.

Различити начини за развијање софтвера за паралелне рачунаре:

- Проширивање постојећег компајлера тако да преводи секвенцијалне програме у паралелне програме. Најчешће се разматра паралелизација преводиоца за Фортран.
- Проширивање постојећег језика новим операцијама које дозвољавају програмерима да обаве (тј. опишу) паралелизацију. Сматра се најлакшим и најпопуларнијим начином паралелног програмирања. Разматрају се стандарди MPI и OpenMP. Мана је недостатак подршке од стране преводиоца при развијању програма и при проналажењу грешака.
- Додавање новог слоја (енгл. *layer*) за паралелни језик изнад постојећег секвенцијалног језика.
- Креирање паралелног језика и преводиоца. Erlag, Occam су примери развијања потпуно новог паралелног језика. Concurrent Pascal, C*, C++ AMP, High Performance Fortran су настали додавањем паралелних конструкција постојећим језицима. Битно је поменути паралелно израчунавање на графичким процесорима, посебно CUDA (енгл. *Compute Unified Device Architecture*), као и окружење OpenCL (енгл. *Open Computing Language*).

Паралелизација алгоритама

Дефиниција 5.6. Рачунари који садрже само један процесор обављају инструкције једну по једну, то јест серијски. Алгоритми који су конструисани са намером да се извршавају у таквом окружењу називају се **секвенцијални алгоритми**.

Дефиниција 5.7. Паралелни алгоритми користе могућност вишепроцесорског система тако што проблем деле на више мањих потпроблема које сваки процесор решава засебно, а онда се ти резултати спајају. Паралелни алгоритми уз ресурсе потребне за обраду података такође имају и (малу) потрошњу ресурса на комункацију између више процесора.

Позитивна својства паралелних алгоритама:

- Истовременост или конкурентност (енгл. *concurrency*) — извршавање више операција/задатака истовремено.

- Скалабилност (енгл. *scalability*) — једноставно прилагођавање алгоритма произвољном броју процесора и произвољном моделу паралелног рачунара.

- Локалност (енгл. *locality*) — већи приступ локалној меморији у односу на *удаљену* меморију, смањење комуникације са другим процесорима, односно комуникација само са малим бројем суседних процесора.

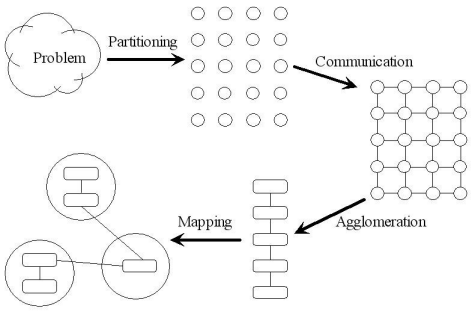
- Модуларност (енгл. *modularity*) — могућност употребе делова алгоритма у различитим програмима.

Једна од методологија за развијање паралелних алгоритама заснива се на **моделу задатак/канал** (енгл. *task/channel model*) који је предложио Фостер. Овај модел се посебно бави описом дизајна ефикасних паралелних алгоритама за дистрибуиране паралелне рачунаре.

Задатак/канал модел представља паралелно израчунавање као скуп задатака који међусобно комуницирају слањем порука кроз канале. **Задатак** се састоји од програма, локалне меморије и скупа улазно-излазних портова. **Канал** је ред порука који повезује излаз једног задатка са улазом другог задатка. Примање порука је синхрона операција, док је слање порука ансихрона операција. Ово значи да задатак који прима поруку може бити **блокиран** док чека да се на улазу појави вредност.

Паралелно израчунавање се може посматрати као усмерени граф у коме тачке представљају задатке, а усмерене гране представљају комуникационе канале.

Треба још напоменути да када се говори о времену извршавања паралелног алгоритма, мисли се на период времена у коме је било који од задатака активан — почетни тренутак је када сви задаци крену да се симултано извршавају, а крајњи тренутак је када последњи задатак заврши извршавање.
На основу овог модела, четири корака у развоју паралелног алгоритма су: **подела** (енгл. *partitioning*), **комуникација** (енгл. *communication*), **груписање/агломерација** (енгл. *agglomeration*), и **мапирање/придруживање** (енгл. *mapping*).



Фостерова методологија дизајна паралелних алгоритама [23]

Подела је процес партиционисања (разлагања) израчунавања и података у мање делове — **податкована и функционална декомпозиција** (енгл. *domain decomposition* и *functional decomposition*). Код обе декомпозиције, ове мање делове називамо **примитивним задацима**. Циљ је идентификовати што више примитивних задатака, јер што је већи број примитивних задатака, то је већа могућност извршавања ефикасније паралелизације. Код податковане декомпозиције прво се врши подела података у мање делове, а онда одређује како да се повежу израчунавања са тим подацима, док је код функционалне декомпозиције обрнут процес. Функционална декомпозиција често има за резултат паралелизам путем преклапања израчунавања, односно паралелно извршавање примитивних задатака насталих партиционисањем израчунавања.

Добра конструкција треба да има следећа својства: постоји бар за један ред величине више примитивних задатака него процесора, редудантна израчунавања и редудантне структуре података су минимализоване, примитивни задаци су скоро исте величине, и број задатака зависи од величине проблема.

Основни начини комуникације међу задацима су: **комуникација тачка-тачка** (енгл. *point-to-point*) где се порука шаље из једног задатка у други тако што се праве канали од сваког од потребних задатака до посматраног задатка, и комуникација помоћу глобалних операција које

омогућују задатку да истовремено комуницира са великим бројем или са свим осталим задацима — **глобална комуникациона шема**.

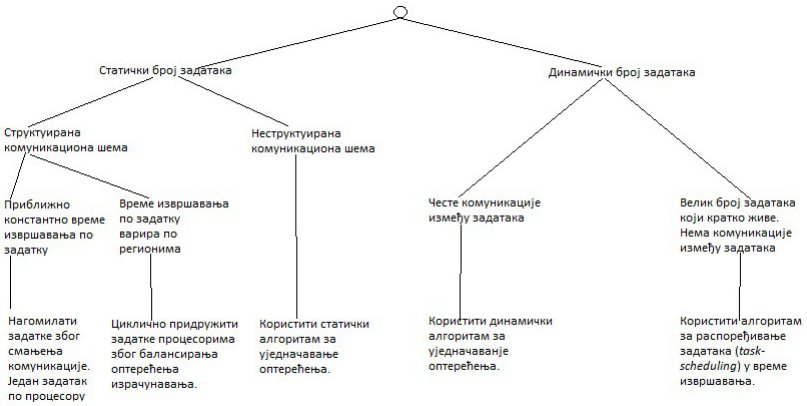
Каже се да је комуникација између задатака део ***overhead*-а** тј. додатног времена потребног код паралелног алгоритма, јер тај део не постоји код секвенцијалних алгоритама. Да би комуникациона структура била добра, операције комуникације треба да су балансиране између задатака, сваки задатак треба да комуницира само са малим бројем својих суседа, задаци треба да конкурентно извршавају и комуникације и израчунавања.

Груписање или агломерација је процес обједињавања задатака у веће задатке како би се побољшале перформансе или поједноставило програмирање. Један од циљева агломерације је смањење додатног времена потребног за комуникацију (тј. комуникационог *overhead*-а) **повећањем локалности** паралелног алгоритма или редуковањем броја послатих порука — **груписање порука** у веће поруке за чије слање је потребно мање времена — због **латентности поруке** (енгл. *message latency, message startup cost*). Ако се са λ означи латентност, а са β брзина преноса *bandwidth*, тј. број података (пакета) који се у јединици времена могу послати кроз канал, онда слање поруке која садржи n података захтева време од $\lambda + \frac{n}{\beta}$. Ако задаци не могу да се извршавају конкурентно јер зависе један од другог, обично је потребно извршити агломерацију. Агломерацијом, поред општих позитивних својстава паралелног алгоритма, треба да се постигне и снижење цене софтвера већим искоришћењем постојећег секвенцијалног кода. Код агломерације треба водити рачуна и о архитектури паралелног рачунара за који са алгоритам развија.

Мапирање или придруживање је процес доделе задатака процесорима. Такође, и код мапирања треба водити рачуна о архитектури рачунара. Ако се програм извршава на централизованом мултипроцесору, оперативни систем аутоматски додељује процесе процесорима. Зато се раматрају паралелни рачунари са дистрибуираном меморијом.

Циљ мапирања је максимизација искоришћености процесора (енгл. *processor utilization*) и минимизација међупроцесорске комуникације, што су често супростављени задаци. Различите карактеристике паралелних алгоритама воде различитим стратегијама придруживања. Нажалост, налажење оптималног решења за проблем мапирања је NP-тежак проблем.

На следећем дијаграму су приказане неке од стратегија које помажу при придруживању задатака процесорима у различитим ситуацијама:



Дрво одлучивања коју стратегију мапирања/придруживања изабрати [23]

Сада се може рећи да **процес развијања паралелног програма** укључује:

1. Анализа проблема. Ако постоји секвенцијални алгоритам, проналажење делова алгоритма који се могу паралелизовати. У неким случајевима је потребан нови алгоритам;

2. Добар паралелни алгоритам није само једноставно проширење одговарајућег секвенцијалног алгоритма. Развој паралелног алгоритма обухвата:

   - Партиционисања проблема и података — податковна и функционална декомпозиција;
   - Проналажење најбољег комуникационог модела — начин, синхронизација, учесталост комуникације;
   - Груписање задатака;

- Мапирање задатака по процесорима.

Потребно је извршити и анализу ефикасности добијеног паралелног алгоритма. Нарочито је битно добро проценити очекивано време извшавања, при чему треба имати у виду и модел паралелног рачунара за који се алгоритам креира.

3. Писање програма: избор програмског језика и окружења. И овде треба водити рачуна о моделу паралелног рачунара на коме се програм извршава — архитектура рачунара, хомогена или хетерогена платформа. Најчешћи избори су или одређени паралелни програмски језик, или MPI/OpenMP, заједно са неким од програмских језика C/C++/Fortran.

4. Исправљање грешака;

5. Тестирање (енгл. *benchmark*) и оптимизација.

5.2 Стандард MPI

MPI (енгл. *Message Passing Interface*) је један у низу стандарда који је створен да би се одговорило на потребу за бржим и јачим рачунарима, као и за побољшањем искоришћености постојећих ресурса. Да би се омогућило знатно брже решавање актуелних проблема, појавила се идеја о коришћењу више једнопроцесорских (или вишепроцесорских) рачунара. MPI се посебно користи код система са дистрибуираном меморијом.

MPI је стандардна спецификација библиотека за комуникацију порукама. Циљ MPI-а је да обезбеди широко коришћени стандард за писање програма заснованих на моделу комуникације порукама. Он би требало да буде практичан, преносив, ефикасан и флексибилан. Већина MPI имплементација комбинује C, C++ и асемблерски језик.

MPI даје спецификацију скупа функција којима се симулира рад модела комуникације порукама, тј. модела слања порука (енгл. *message-passing*). Овај скуп функција омогућава симулирање рада паралелних програма тако што при покретању програма створи задати број процеса. Сваки процес изводи секвенцијални програм написан, на пример у C-у, а комуницира са другим процесима помоћу функција за примање, односно слање порука.

MPICH2 је имплементација MPI-1 и MPI-2 стандарда. MPICH2 је имплементација MPI која ефикасно подржава различита израчунавања и платформе — Unix, Linux, Windows, 32-битне, 64-битне архитектуре рачунара, *commodity* кластере, веома брзе мреже (енгл. *high-speed networks — 10 Gigabit Ethernet, InfiniBand,* итд.), и специјализоване врхунске (енгл. *proprietary high-end*) системе за израчунавање. Како MPICH2 не подржава хетерогене платформе, за такве платформе треба користити MPICH1 имплементацију — имплементацију *само* MPI-1.1 стандарда.[53]

Веома популарна имплементација стандарда MPI је и **OpenMPI**. OpenMPI је настао спајањем више добро познатих MPI имплементација.

Интерфејс **mpiJava** је објектно-оријентисани Јава интерфејс према MPI-у. mpiJava 1.2 обезбеђује потпуну функционалност MPI 1.1.

Стандард **OpenMP** је API стандард за писање паралелних апликација са заједничком (дељеном) меморијом (енгл. *shared memory parallel applications*) у програмским језицима C, C++, Fortran. Идеално је прилагођен вишејезгарним архиртектурама.[54]

Иако су MPI и OpenMP дизајнирани за употребу са програмским језицима Фортран и C/C++, постоје повезивања за многе друге језике, укључујући програмске језике Pascal, Perl, Python, Java. Једна од мана коришћења MPI и OpenMP са Јавом је то што они нису објектно оријентисани, односно одређене Јавине карактеристике не одговарају оваквом начину програмирања.

Док је MPI одличан начин комуникације међу процесорима у мултирачунару (тј. међу процесорима у различитим SMP-овима), OpenMP је бољи начин за комуникацију међу процесорима у једном SMP-у. Такође, показало се да је хибридни MPI/OpenMP програм решење које постиже најбоље резултате. Обично је могуће трансформисати MPI програм у хибридни MPI/OpenMP за рад на кластеру мултипроцесора.

[53] За више информација и преузимање одговарајућег пакета, видети: `http://www.mcs.anl.gov/research/projects/mpich2/`.

[54] За више информација, видети: `http://openmp.org`, OpenMP API спецификацију за паралелно програмирање.

Модел комуникације путем прослеђивања порука

Модел задатак/канал који је описан може се имплементирати уз помоћ модела комуникације путем слања порука. Задатак из задатак/канал модела постаје **процес** у моделу прослеђивања порука. Претпоставља да се програм извршава на мултирачунару или мултипроцесору са дистрибуираном меморијом.

Корисник обично одређује број конкурентних процеса када програм почне. Сваки процес извршава исти програм и има своју копију свих променљивих декларисаних у програму, али како сваки од процеса има јединствени **идентификациони број** (тј. *ID*), различити процеси могу да изводе различите операције у току извршавања програма.

Битно је нагласити да процеси шаљу поруке и због међусобне комуникације и због синхронизације. Како само примање поруке другог процеса говори нешто о стању тог процеса, може се рећи да чак и порука без садржаја има значење.

Стандард MPI и програмски језик C

Када је потребно у програмском језику C користити MPI функције, потребно је укључити заглавље (енгл. *header file*) за MPI помоћу следеће претпроцесорске директиве:

```
#include <mpi.h>
```

Имена свих MPI функција и константи почињу са `MPI_`. Прва функција која се позива од стране сваког MPI процеса је:

```
MPI_Init(int *argc, char **argv);
```

Изузетак је функција `MPI_Initialized`, која проверава да ли је MPI иницијализован и може бити позвана пре `MPI_Init`.

Када је MPI иницијализован, сваки активни процес постаје члан подразумеваног (енгл. *default*) комуникатора `MPI_COMM_WORLD`. **Комуникатор** (енгл. *communicator*) је објекат који обезбеђује окружење за прослеђивање порука између процеса. Могу се креирати и сопствени комуникатори ако је потребно поделити процесе у независне комуникационе групе.

Ранг процеса (енгл. *rank*) је његова позиција у општем поретку. У комуникатору са p процеса сваки процес има јединствени ранг (ID број) између 0 и $p-1$. Процес може користити свој ранг да одреди за који део израчунавања или за који део података је одговоран.

```
MPI_Comm_rank(MPI_Comm comm, int *rank);
MPI_Comm_size(MPI_Comm comm, int *size);
```

Прву функцију процес позива да би одредио свој ранг, а другу функцију да би одредио укупан број процеса у комуникатору.

Последња MPI функција коју процес позива је `MPI_Finalize()`. Ова функција омогућава систему да ослободи све ресорсе који су били заузети од стране MPI.

Редослед по коме се резултати из различитих процесора појављују на стандардном излазу не осликава у потпуности стваран редослед како су се резултати појављивали. Ово је битно знати да не бисмо доносили погрешне закључке о извршавању паралелног алгоритма.

Колективна комуникација (енгл. *collective communication*) је операција комуникације у којој група процесора ради заједно у циљу дистрибуирања или скупљања скупа од једне или више вредности.

Редукција[55] је пример операције која захтева колективну комуникацију у окружењу где се комуникација обавља слањем порука. У општем случају, број комуникационих корака који су потребни да n задатака изврши редукцију је $\lceil \log_2 n \rceil$. Такође, може се извршити агломерацију више задатака у један на сваком процесору, тако да сваки од добијених задатака користи секвенцијални алгоритам да пронађе локални резултат редукције, пре него што комуницира са другим задацима.

```
MPI_Reduce(void *operand, void *result, int count, MPI_Datatype type, MPI_Op operator,
           int root, MPI_Comm comm)
```

Помоћу функције `MPI_Reduce` израчунава се један резултат на основу података из свих процеса. Пети параметар ове функције означава која врста операције се изводи над тим подацима (минимум, максимум, сабирање, множење, битско и, битско или, ...).

[55] Редукција је примена асоцијативне бинарне операције на скуп података. Пример редукције је тражење суме $a_0 + a_1 + \cdots + a_n$.

Веома је битно запамтити да док само један процес добија глобални резултат, сваки процес који учествује у израчунавању мора позвати функцију `MPI_Reduce`.

Дефиниција 5.8. Тривијално паралелни програм (енгл. *embarrassingly parallel*) је програм који захтева мало труда да би се поделио у одређени број паралелних задатака. Обично између тих паралелних задатака нема комуникације, односно нема зависности. Ако неки (или сви) од ових задатака производи излаз, постоји канал између сваког таквог задатка и излазног уређаја.

Дефиниција 5.9. Циклична (или испреплетана) алокација (енгл. *cyclic/interleaved allocation*) је циклично додељивање сваком од p процеса одређеног дела посла, тако да ако је посао подељен у n делова, и $0 \leq k < n$, тада се k-ти део посла додељује ($k \mod p$)-ом процесу.

Мерење паралелних перформанси

Може се рећи да перформанса представља меру обављене количине рада у јединици времена. На пример, за одређивање перформанси процесора користи се број операција у секунди.

Benchmark је стандард по коме се нешто мери или процењује. Сврха мерења (енгл. *benchmarking*) је предвиђање перформанси паралелног програма на основу познавања перформанси секвенцијалног програма, времена кашњења порука (енгл. *latency*), протока комуникације међу процесорима и мрежног протока (енгл. *bandwidth*).

После развијања паралелног програма, поставља се питање колико побољшање он постиже у односу на секенцијални. Потребно је измерити колико је дати паралелни програм бржи у односу на секвенцијални и то у израчунавањима, без читања базе података и уписа резултата.

```
double MPI_Wtime(void);
double MPI_Wtick(void);
```

Прва функцуја враћа број секунди које су протекле од неке временске тачке у прошлости. Друга функција враћа прецизност резултата који смо добили помоћу `MPI_Wtime`. Мерење времена извршавања дела кода се изводи позивајући `MPI_Wtime` пре и после те секције, и налазећи разлику враћених вредности.

У случајевима када је у питању операција редукције, мора се обезбедити да ни један процес не настави извршавање пре него што сви процеси обаве редукцију.

```
int MPI_Barrier(MPI_Comm comm);
```

Синхронизација уз помоћ **баријере** гарантује да ниједан процес не може да настави после баријере док сви процеси не дођу до ње. Обично се баријера поставља између две фазе у извршавању програма.

Са растом броја процесора, расте и додатно време утрошено на комуникацију.

Декомпозиција података

Дефиниција 5.10. Декомпозиција података (енгл. *data decomposition*), или једноставно декомпозиција, је финално груписање података, које је резултат поделе, агломерације и мапирања.

Разматрају се испреплетана и блок декомпозиција података. Претпоставимо да постоји низ елемената и да је први индекс 0.

Испреплетана декомпозиција података (енгл. *interleaved data decomposition*): процес 0 је одговоран за индексе 0, $0+p$, $0+2p, ...$, процес 1 је одговоран за индексе 1, $1+p$, $1+2p, ...$, и тако даље.

Предност оваквог приступа је што за дати индекс низа i лако одредити који процес контролише овај индекс — то је процес ($i \mod p$).

Блок декомпозиција података подразумева дељење низа података у p узастопних блокова скоро исте величине. Ако n није дељив са p, тада је потребна шема алокације блокова која сваком процесу додељује или $\lceil \frac{n}{p} \rceil$ или $\lfloor \frac{n}{p} \rfloor$ елемената (ако је n дељиво са p, сваком процесу је додељено $\frac{n}{p}$ елемената). Постоје два начина да се ово постигне.

Код **прве шеме блок декомпозиције** израчунава се $r = n \pmod p$. Ако је $r > 0$, првих r процеса добија блокове величине $\lceil \frac{n}{p} \rceil$, а осталих $p - r$ процеса добија блокове елемената величине $\lfloor \frac{n}{p} \rfloor$.

Сада се постављају питања који ранг елемената контролише одређени процес и који процес контролише одређени елемент?

Први елемент који контролише i-ти процес је: $i \cdot \lfloor \frac{n}{p} \rfloor + \min(i, r)$. Последњи елемент који контролише i-ти процес је тачно један елемент пре првог елемента контролисаног од стране $i+1$-ог процеса: $(i+1) \cdot \lfloor \frac{n}{p} \rfloor + \min(i+1, r) - 1$. Процес који контролише j-ти елемент низа је дат изразом: $\min(\lfloor \frac{j}{\lfloor \frac{n}{p} \rfloor + 1} \rfloor, \lfloor \frac{j-r}{\lfloor \frac{n}{p} \rfloor} \rfloor)$.

Пример 5.1. Нека је $n = 20$, $p = 6$. Тада је $r = 20 \pmod 6 = 2$, $20 = 3 \cdot 6 + 2$. Ово значи да прва 2 процеса имају по 4 елемената ($\lceil \frac{20}{6} \rceil = 4$), а остала 4 процеса имају по 3 елемената ($\lfloor \frac{20}{6} \rfloor = 3$), тј. блокове дужине 3. Како постоји 6 процеса и 20 елемената низа података, $p = 0, \ldots, 5$ и $i = 0, \ldots, 19$. Следећа табела показује распоред елемената по процесорима:

процес	Индекс елемента			
0	0	1	2	3
1	4	5	6	7
2	8	9	10	
3	11	12	13	
4	14	15	16	
5	17	18	19	

- Први елемент контролисан од стране 0.ог процеса: $0 \cdot \lfloor \frac{20}{6} \rfloor + \min(0, 2) = 0 + 0 = 0$.
- Први елемент контролисан од стране 3. процеса: $3 \cdot \lfloor \frac{20}{6} \rfloor + \min(3, 2) = 3 \cdot 3 + 2 = 11$.
- Последњи елемент који контролише 0. процес: $(0+1) \cdot \lfloor \frac{20}{6} \rfloor + \min(0+1, 2) - 1 = 3 + 1 - 1 = 3$
- Процес који контролише 10. елемент низа података: $\min(\lfloor \frac{10}{\lfloor \frac{20}{6} \rfloor + 1} \rfloor, \lfloor \frac{10-2}{\lfloor \frac{20}{6} \rfloor} \rfloor) = \min(2, 2) = 2$
- Процес који контролише 17. елемент низа података: $\min(\lfloor \frac{17}{\lfloor \frac{20}{6} \rfloor + 1} \rfloor, \lfloor \frac{17-2}{\lfloor \frac{20}{6} \rfloor} \rfloor) = \min(4, 5) = 4$

Приметимо да последњи пример није тачан. Заправо, може се показати да ако је процес $i < r$, важи $i = \frac{j}{\frac{n}{p}+1}$, док за процесе $i > r$ важи да је $i = \frac{j-r}{\frac{n}{p}}$.

Друга шема блок алокације не групише све велике блокове. Претпоставимо да је n број елемената и да је p број процеса.

Први елемент који контролише i-ти процес је $\lfloor \frac{i \cdot n}{p} \rfloor$. Последњи елемент који контролише i-ти процес је тачно један елемент пре првог елемента контролисаног од стране $i+1$-ог процеса: $\lfloor \frac{(i+1) \cdot n}{p} \rfloor - 1$. Процес који контролише j-ти елемент низа је дат изразом $\lfloor \frac{(j+1) \cdot p - 1}{n} \rfloor$.

Други приступ је бољи јер захтева мање операција при извршавању три најчешћа израчунавања, нарочито зато што целобројно дељење у C-у аутоматски заокружује резултат на нижу вредност (енгл. *rounds down*). Овај начин блоковске декомпозиције се најчешће користи.

Пример 5.2. Нека су све вредности параметара једнаке вредностима из претходног примера. Следећа табела приказује распоред елемената по процесорима у овој шеми:

процес	Индекс елемента			
0	0	1	2	
1	3	4	5	
2	6	7	8	9
3	10	11	12	
4	13	14	15	
5	16	17	18	19

Макрои за блок декомпозицију. MPI_Bcast функција

Наредна четири *C* макроа могу бити искоришћена у било ком паралелном програму где је група података дистрибуирана међу скупом процесора користећи блок декомпозицију, а која су написана на основу претходних разматрања:

```
#define BLOCK_LOW(id,p,n) ((id)*(n)/(p))
#define BLOCK_HIGH(id,p,n) (BLOCK_LOW((id)+1,p,n)-1)
#define BLOCK_SIZE(id,p,n) (BLOCK_LOW((id)+1,p,n)-BLOCK_LOW(id,p,n))
#define BLOCK_OWNER(index,p,n) (((p)*((index)+1)-1)/(n))
```

За дистрибуцију вредности елемената се користи функција:

```
int MPI_Bcast(void *buffer, int count, MPI_Datatype, int root, MPI_Comm com);
```

Након извршења ове функције, сваки процес има нове вредности елемената које им је требало послати (адреса првог елемента је одређена првим параметром функције, а број елемената другим параметром).

Локални и глобални индекси

Када се обавља разлагање низа у делове дистрибуиране међу процесима, треба направити разлику између локалног индекса и глобалног идекса неког елемента низа. Секвенцијални код увек користи глобалне индексе да реферише на елементе низа, па се при трансформисању програма у паралелни програм мора водити рачуна о овоме.

5.3 Паралелизација Ератостеновог сита

За почетак се разматра основна верзија Ератостеновог сита која је дата у поглављу 3. Претпоставимо да постоји p процеса и да треба пронаћи све просте бројеве до n, тако што се за сваки неозначени број k налазе и означавају сви бројеви дељиви са k, у интервалу од k^2 до n.

Циљ је развијање што ефикасније паралелне верзије Ератостеновог сита, чија секвенцијална верзија је већ дата. Иако ово сито није практично за проналажење великих простих бројева са неколико стотина цифара, модификована верзија сита је и даље веома важно оруђе у истраживањима везаним за теорију бројева.

Развијање паралелног алгоритма

Испреплетана декомпозиција, када се примењује на бројеве k којима се просејава, има битних мана јер може водити ка значајном дисбалансу међу процесима (ако два процеса маркирају бројеве дељиве са 2, први процес маркира $\lceil \frac{n-1}{2} \rceil$ елемената, док други процес не маркира ниједан — неуравнотеженост рада процесора). Такође мана ове имплементације је што је и даље потребна нека врста операција редукција/дистрибуција (да би се одредила нова вредност за k и да би се онда та нова вредност послала свим процесима), па стога и већа комуникација.

Из наведених разлога разматра се како блок декомпозиција утиче на имплементацију паралелног алгоритма. Процеси деле интервал бројева који се просејава на p делова.

Приметимо да је највећи прост број који се користи за *просејавање* целих бројева до n број $\lfloor \sqrt{n} \rfloor$. Ако је први процес задужен за целе бројеве до $\sqrt{n}$, тада тражење следеће вредности за k не захтева комуникацију. Први процес има око $\frac{n}{p}$ елемената. Ако је $\frac{n}{p} > \sqrt{n}$ (тј. $p < \sqrt{n}$), тада он контролише све просте бројеве до $\sqrt{n}$.

Следећа предност блок декомпозиције је та да може убрзати означавање бројева дељивих са k — тако што проналази први број j дељив са k, а онда означава све бројеве $j+k$, $j+2k$, итд., све до краја блока — користи петљу (енгл. *loop*) сличну као код секвенцијаног алгоритма. Оваква блок декомпозиција резултује мањим бројем корака израчунавања и комуникације.

Разматра се секвенцијални алгоритам Ератостеновог сита да би се његовом анализом утврдило како је могуће сваки корак превести у паралелни алгоритам. У прилогу су изложене паралелизације побољшаних верзија Ератостеновог сита, чије секвенцијалне верзије су дате у претходним поглављима.

1. Прво треба направити листу целих бројева $2, 3, 4 \ldots, n$. Овај корак се може једноставно паралелизовати, тако што сваки процес у паралелном програму креира свој део листе, који садржи $\lfloor \frac{n}{p} \rfloor$ или $\lceil \frac{n}{p} \rceil$ битова.

2. Сада треба да се *проспеје* листа бројева која тако што се налазе сви бројеви који су дељиви са бројем k и маркирају се. Поставља се k на 2, $k = 2$, то је први немаркирани број у листи. Сваки процес треба да зна вредност броја k да би маркирао све бројеве дељиве са k у делу који он контролише. Зато сваки процес у паралелном програму извршава овај корак.

3. Следи главни део програма. Следећа петља је *језгро* програма:

   while $(k^2 <= n)$ **do**

Маркирати све бројеве дељиве са k у интервалу од k^2 до n. И овај корак је лако трансформисати. Сваки процес је одговоран за маркирање свих бројева дељивих са k у блоку који он контролише, јер је интервал (k^2, n) подељен на блокове поступком који је у претходном поглављу описан. Потребна су само мања додатна израчунавања да би била одређена локацију првог броја у блоку који је дељив са k, али после тога је поступак веома једноставан — маркира се само сваки k-ти елемент у блоку.

Пронађи се најмањи број већи од k који није маркиран. Поставља се k на ту нову вредност. Ако је, као што смо већ рекли, 0. процес одговоран за одређивање следеће вредности за k, јер је $p < \sqrt{n}$, онда сви други процеси морају да добију нову вредност за k да би могли да израчунају услов у while петљи и можда је искористе и за следећу итерацију петље. Следи да је у овом кораку потребна функција глобалне комуникације — због преноса, тј. *broadcasting*-а.

4. Немаркирани бројеви су прости бројеви које је било потребно наћи.

У случају овог паралелног алгоритма, позив функције за дистрибуцију има облик:

```
MPI_Bcast(&k, 1, MPI_LONG_LONG, 0, MPI_COMM_WORLD);
```

Како сваки процес рачуна број простих бројева у свом блоку, на крају треба сабрати све те подзбирове како би се добио укупан број простих бројева до n — као што смо видели, за то је потребна функција `MPI_Reduce`.

Побољшања представљају **брисање парних бројева**, тј. не тестирају се и не записују парни бројеви, што смањује захтеве за меморијом и утиче на брзину програма, и **елиминација дистрибуције** броја k која се постиже тако што уместо да у сваком кораку један процес идентификује наредну вредност броја k и проследи је осталим процесима, дозвољава се да сваки процес идентификује нову вредност за k. У оригиналној декомпозицији података ово је немогуће, јер је овај део задатка под контролом 0.ог процеса. Треће побољшање односи се на **повећање *cache hit rate***, односно максимално искоришћење кеш меморије ради бржег извршавања инструкција програма над операндима који су још у брзој кеш меморији.

Анализа паралелног алгоритма. Оцењивање перформанси датог програма

Нека је χ **време потребно да се маркира одређени бит**. Ово време укључује сетовање бита на 1, време потребно да се увећа индекс петље и тестира завршни услов. Секвенцијални алгоритам има сложеност $\Theta(n \ln \ln n)$ [17, 23], тј. приближно $\chi \cdot n \ln \ln n$. Константа χ се може експериментално одредити покретањем секвенцијалног алгоритма.

Пошто се преноси само један податак у свакој итерацији, цена сваког преноса је око $\lambda \lceil \log p \rceil$, где је λ кашњење поруке, а p број процесора.

Како простих бројева до n на основу теореме о простим бројевима има прилижно $\frac{n}{\ln n}$, постоји око $\frac{\sqrt{n}}{\ln \sqrt{n}}$ итерација у петљи. Промена алгоритма тако да се узимају у обзир само непарни бројеви смањује заузету меморију за пола и скоро дуплира брзину којом се проналазе бројеви дељиви одређеним простим бројем. Са овом променом процењено време извршавања секвенцијалног алгоритма постаје приближно: $\frac{\chi(n \ln \ln n)}{2}$, док је процењено време извршења паралелног алгоритма око $\frac{\chi \cdot (n \ln \ln n)}{2p} + \frac{\sqrt{n}}{\ln \sqrt{n}} \cdot \lambda \lceil \log p \rceil$. За интервале бројева који су интересантни за проверу други сабирак се чини занемарљивим. Међутим, како број процеса расте, релативна важност (учешће) комуникационе компоненте у целокупном времену извршавања такође расте [23]. Зато је у општем случају добро смањити количину комуникацију између процеса, тј. смањити број операција дистрибуције, редукције, синхронизације.

Као што се лако показује, елиминација дистрибуције може побољшати перформансе овог паралелног алгоритма. Сваки задатак може користити секвенцијални алгоритам да нађе просте бројеве од 3 до $\lfloor \sqrt{n} \rfloor$, или користи већ генерисани низ простих бројева из тог интервала. Лако се показује да је у првом случају очекивано време извршавања програма приближно $\chi(\frac{n \ln \ln n}{2p} + \sqrt{n} \ln \ln \sqrt{n}) + \lambda \lceil \log p \rceil$. Ако се користи унапред генерисана листа простих бројева до $\sqrt{n}$, онда је време извршавања око $\frac{\chi(n \ln \ln n)}{2p} + \lambda \lceil \log p \rceil$.

У наставку је дат једноставан паралелни програм који реализује Ератостеново сито. Парни бројеви су елиминисани. Дистрибуција је задржана, да би била илустрована ова техника

паралелног програмирања. У прилогу уз рад је дата паралелизација модификованог сита где су примењена наведена побољшања.

```
// Modifikovano Eratostenovo sito na osnovu Eratostenovog sita iz [23]

#define BLOCK_LOW(id, p, n) ((id) * (n) / (p))
#define BLOCK_HIGH(id, p, n) (BLOCK_LOW((id) + 1, p, n) - 1)
#define BLOCK_SIZE(id, p, n) (BLOCK_LOW((id) + 1, p, n) - BLOCK_LOW(id, p, n))

#include <stdio.h>
#include <mpi.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

int main (int argc, char **argv) {
  FILE *baza, *rez;

  // ako se program pokrece sa najvise 8 procesa
  char* ime[10] = {"0.txt", "1.txt", "2.txt", "3.txt", "4.txt",
                   "5.txt", "6.txt", "7.txt", "8.txt", "9.txt"};

  long long count;              // lokalni brojac prostih brojeva
  double elapsed_time;          // vreme izvrsavanja
  long long first;              // indeks prvog broja deljivog sa k
  long long global_count;       // globalni brojac prostih brojeva
  long long high_value;         // najveca vrednost u datom procesu
  int id;                       // ID procesa
  long long index;              // indeks trenutnog prostog broja
  long long low_value;          // najniza vrednost u datom procesu
  char *marked;                 // deo niza od 2 do n
  long long n;                  // prosejavanje brojeva od 2 do n
  int p;                        // ukupan broj procesa
  long long proc0_size;         // velicina podniza 0. procesa
  long long prime;              // trenutni prost broj
  long long size;               // broj elemenata u marked
  long long global_b;
  long long i, j, k, b=0;

  MPI_Init (&argc, &argv);
  MPI_Barrier (MPI_COMM_WORLD);
  MPI_Comm_rank (MPI_COMM_WORLD, &id);
  MPI_Comm_size (MPI_COMM_WORLD, &p);

  if (argc < 2) {
    if (!id) printf ("Komandna linija: %s\n", argv[0]);
    n = 10000000000LL;
  } else n = strtoll (argv[1], NULL, 0);

  if(!id) printf ("Program trazi proste brojeve do: %llu.\n", n);

  // deljenje niza po procesorima
  low_value = 1 + BLOCK_LOW(id, p, n);
  high_value = 1 + BLOCK_HIGH(id, p, n);
  size = BLOCK_SIZE (id, p, (n >> 1U));

  // svi prosti brojevi koji se koriste za prosejavanje su pod kontrolom 0. procesa
  proc0_size = (n) / (p << 1);
  if ((2 + proc0_size) < (long long) sqrt(n)) {
    if (!id) printf ("Previshe procesa.\n");
      MPI_Finalize ();
      exit (1);
  }

  // alocira za svaki proces njegov deo niza
  marked = (char*) calloc (size, sizeof (char));
  if (marked == NULL) {
    printf ("Ne mozhe alocirati dovoljno memorije.\n");
    MPI_Finalize ();
    exit(1);
  }
```

```
  // za svaki proces; primer: za 1,3,5,7,9,... indeksi: 0,1,2,3,4,...
  memset (marked, 0x00, size);

  if (!id) index = 1;            // samo 0. proces
  prime = 3;

  MPI_Barrier (MPI_COMM_WORLD);
  elapsed_time = MPI_Wtime();  // pokrece tajmer

  while (prime * prime <= n) {
      k = prime * prime;

      if ((k > low_value) && (k < high_value)) {
        // postavlja index prvog broja deljivog sa prime:
        first = k - low_value;
        first >>= 1U;            // first = (first - 1)/2;
      } else if (k < high_value) {
          // ako je low_value deljiv sa prime, indeks je jednak 0 (1. element u nizu)
          if (!(low_value % prime)) {
            first = 0;
        } else { // u suprotnom, nalazi indeks broja koji je deljiv sa prime
          first = prime - (low_value % prime);
          if (!((first +low_value)&1)) first += prime;
          first >>= 1U;
        }
      } else first = size;

      // setuje 1 tamo gde su slozeni brojevi
      for (i = first; i < size; i += prime) marked[i] = 1;

      if (!id) {
        // trazi prvi nemarkirani broj
        while ((index<size) && marked[++index]);
        prime = (index << 1)+1;
      }
      // broadcast-ovanje pronadjene sledece vrednosti
      MPI_Bcast (&prime, 1, MPI_LONG_LONG, 0, MPI_COMM_WORLD);
      MPI_Barrier (MPI_COMM_WORLD);
  }

  /*
  if ((baza = fopen(ime[id+1], "wb")) == NULL) {
    return 1;
    printf ("Ne moze da otvori bazu.\n");
  }
  */

  count = 0;
  for (i = 0; i < size; i++)
    if (!marked[i]) {
          count++;
          // fprintf (baza, "%lld,", (low_value + (i << 1U)));
        }

  MPI_Barrier (MPI_COMM_WORLD);
  // sabira sve podzbirove:
  MPI_Reduce (&count, &global_count, 1, MPI_LONG_LONG, MPI_SUM, 0, MPI_COMM_WORLD);

  elapsed_time += MPI_Wtime();  // Zaustavlja tajmer

  if (!id) {
    printf ("Nadjeno je %lld prostih brojeva.\n", global_count);
    printf ("Vreme za koje su pronadjeni: %.6f sekundi.\n", elapsed_time);

    if ((rez = fopen(ime[id], "a")) == NULL) {
      printf ("Ne moze otvoriti bazu.\n");
      exit(2);
    }
    fprintf (rez, "Prostih brojeva do %lld ima %lld. Prosti brojevi su nadjeni za
                %.6f sec.\r\n",n, global_count, elapsed_time);
    fclose (rez);
  }

  free (marked);
```

```
  //fclose (baza);
  MPI_Finalize ();

  return 0;
}
```

Претходни програм је веома једноставан и постоји много начина да се побољша.

Оцењивање перформанси паралелног алгоритма

При оцењивању паралелних алгоритама критеријуми који се најчешће користе су време извршавања, број процесора на којима се програм извршава и цена.

Време извршавања параленог алгоритма означавамо са $T(n,p)$, где је n величина улаза, а p број процеса.

Цена (енгл. *cost*) паралелног алгоритма је дефинисана као $C(n,p) = T(n,p) \times p$, тј. једнака је (потенцијалном) укупном броју корака извршених у свим процесима при решавању датог проблема.

Убрзање (енгл. *speedup*) је $S(n,p) = \frac{T(n,1)}{T(n,p)}$, тј. представља однос између времена извршавања секвенцијалног алгоритма и времена извршавања паралелног алгоритма.

Операције које извршава паралелни алгоритам се деле на:

1. Израчунавања која се морају секвенцијално извршити. Означимо их са $\sigma(n)$.
2. Израчунавања која се могу паралелно извршити, у ознаци $\phi(n)$.
3. Додатна израчунавања (енгл. *parallel overhead*) настала због паралелизације, $\kappa(n,p)$.

Комплетан израз за убрзање је дат са: $S(n,p) \leq \frac{\sigma(n)+\phi(n)}{\sigma(n)+\frac{\phi(n)}{p}+\kappa(n,p)}$.

Сада следи познати **Амдалов закон**:

Дефиниција 5.11. Означимо са f удео секвенцијалних операција, $f = \frac{\sigma(n)}{\sigma(n)+\phi(n)}$, $0 \leq f \leq 1$. Онда је $1-f$ део операција које могу бити извршене паралелно. Максимално убрзање S које може бити постигнуто паралелним израчунавањем на p процесора дато је изразом: $S \leq \frac{1}{f+\frac{1-f}{p}}$.

Овај закон даје горњу границу убрзања које може бити постигнуто паралелизацијом проблема на одређеном броју процесора. Такође може бити коришћен да се одреди асимптотско убрзање које се постиже како се број процесора повећава.

Код извршавања тривијално паралелних програма на вишејезгарном процесору, на основу Амдаловог закона следи да убрзање које они остварују може бити једнако броју језгара, нарочито ако се проблем подели тако да одговара кеш меморији сваког језгра. Ако је у језгрима имплементирана *hyper-threading* технологија (или нека напреднија технологија), убрзање може бити и веће.[56]

У последњем поглављу се упоређују времена извршавања датог паралелног сита и модификованог сита и описује понашање програма за различити број процеса.

5.4 О паралелизацији других значајних алгоритама

Многи алгоритми за факторизацију могу се паралелизовати. Квадратно сито је веома погодно за факторизацију. На пример, сваки процес једноставно просејава различите скупове полинома, и шаље резултате једном *главном* процесу. Познате су имплементације алгоритама за факторисање помоћу елиптичких кривих на SIMD машини, коју су урадили Диксон и Ленстра. Паралелизације општег сита у пољу бројева су довеле до факторисања бројева од неколико стотина цифара [28].

Слично као код факторизације, алгоритми за тражење дискретног логаритма захтевају доста времена, а могу се тривијално паралелизовати, тако да процеси скоро независно врше израчунавања. И други алгебарски алгоритми се могу паралелизовати. Код Кинеске теореме о остацима

[56] За више информација, видети `https://en.wikipedia.org/wiki/Hyper-threading`.

сваки процес може радити израчунавања у односу на мањи модуо n_i, и онда послати резултате једном процесу да израчуна коначну вредност.

Многи криптографски алгоритми, као што је то AES, су погодни за паралелна израчунавања.

У уводу смо поменули најједноставнији начин паралелизације тестова простости. Ови тестови често користе (ефикасне) алгоритме за степеновање и рачунање по модулу, као што су то алгоритми FFT, DFT, који се лако могу паралелизовати.[57] На пример, AKS тест је тривијално паралелизовати. За више информација о паралелизацији ових алгоритама, видети [11, 17, 23].

[57] Видети "Нумеричке методе", Десанка Радуновић, Академска мисао, 2004.

6 Криптографија

Криптологија је наука која проучава математичке технике за обезбеђивање тајности, аутентичности, интегритета *дигиталних* информација, укључујући и безбедну имплементацију ових техника. Криптологија представља научну дисциплину која се бави сигурним (тајним) комуникацијама. Две основне и уско повезане гране криптологије су криптографија и криптоанализа.

Криптографија (енгл. *cryptography*) је наука и вештина очувања безбедности порука. Предмет криптографије је синтеза поступака за обезбеђивање тајности информација, тзв. крипто-заштиту. Применом криптографије реализују се четири основна безбедносна захтева (тј. циља):

1. **Тајност** — очување **поверљивости** поруке (енгл. *confidentiality*),
2. **Интегритет** — очување садржаја поруке (енгл. *integrity*),
3. **Аутентикација** — потврђивање порекла поруке, **провера идентитета**, (енгл. *authentication*),
4. **Непорецивост** — немогућност пошиљаоца да негира да је послао своју поруку, (енгл. *non-repudiation*).

Постоје многобројне корисне примене криптографије: безбедна комуникација приликом приступа удаљеним подацима, обезбеђивање идентификације и аутентичности на Интернету, сертификација, дистрибуција кључа, електронска трговина (енгл. *e-commerce*), мобилна телефонија, кабловска телевизија.

Порука је отворен текст (енгл. *plaintext*).

Шифровање или **енкрипција** (енгл. *encryption*) је трансформација поруке у форму коју је практично *немогуће* прочитати без одговарајућег знања (тј. знања кључа). Њена сврха је очување приватности информација тако што их сакрива од свакога коме оне нису намењене, чак и од оних који имају приступ шифрованим (енкриптованим) подацима.

Другим речима, шифровање је процес маскирања поруке који за резултат има сакривање њене садржине.

Шифрована порука је **шифрат** (енгл. *ciphertext*).

Дешифровање или **декрипција** (енгл. *decrypt, decipher*) је обрнут поступак од шифровања. То је трансформација шифрованих (енкриптованих) података у разумљиву форму (тј. процес претварања шифрата у отворени текст).

Предмет криптоанализе је разматрање метода којима се компромитују тј. "разбијају" од стране неовлашћених корисника поступци крипто-заштите информација. Може се рећи и да је криптоанализа *вештина* проваљивања шифроване поруке.

Покушај криптоанализе назива се **напад** (енгл. *attack*). Под овим претпоставкама, напади су класификовани на основу тога којим информацијама криптоаналитичар има приступ.

Типови криптоаналитичког напада [5]:

1. Напад "са познавањем само шифрата" (енгл. *ciphertext-only attack*).
2. Напад "са познатим отвореним текстом" (енгл. *known-plaintext attack*). Криптоаналитичар има приступ не само неким шифратима, него и отвореном тексту тих порука. Примери овог типа су напад потпуна претрага и напад сусрет на пола пута (енгл. *meet-in-the-middle*).
3. Напад "са изабраним отвореним текстом" (енгл. *chosen-plaintext attack*). Овај тип напада је бољи од претходног, јер криптоаналитичар може да изабере специфичне парове отворених текстова и шифрата, који дају више информација о кључу.

Осим ових најбитнијих типова, додатни типови напада су:

1. Напад "са прилагодљивим одабраним отвореним текстом" (енгл. *adaptive-chosen-plaintext attack*). Овај адаптивни напад је посебан случај напада "са изабраним отвореним текстом".

2. Напад "са изабраним шифратом" (енгл. *chosen-ciphertext attack*).

3. Напад "са изабраним кључем" (енгл. *chosen-key attack*). Криптоаналитичар поседује одређено знање о вези између различитих кључева.

Откривање кључа некриптоаналитичким методама назива се компромитовање (енгл. *compromise*). Понекад се криптоанализа претњом или крађом (енгл. *rubber-hose cryptanalysis*) сматра типом криптоаналитичког напада.

6.1 Основни појмови у криптографији

Шифра, тј. шифарник (енгл. *cipher*) или **криптографски алгоритам** је математичка функција која се користи за шифровање (енгл. *encrypt, crypt, encipher*) података, као и за дешифровање података. У општем случају, ради се о две функције: једној за шифровање и једној за дешифровање.

Оба поступка (шифровање и дешифровање) обично захтевају коришћење неке тајне информације коју називамо **кључ**. У неким случајевима, користи се исти кључ и за шифровање и за дешифровање, док се у другим случајевима користе различити кључеви.

У општем случају мора се претпоставити да је јавно познат тип примењеног шифарског система. У супротном, ако се сигурност алгоритма заснива на чувању у тајности који алгоритам се користи, такав алгоритам се назива **тајни** (поверљив) алгоритам (енгл. *restricted*). Али, и поред великих недостатака тајни алгоритми су веома популарни.

Криптосистем (енгл. *cryptosystem, cipher system*) је криптографски алгоритам са свим отвореним текстовима, шифратима и кључевима, као и свим потребним алгоритмима за његово функционисање. Може се рећи да је криптосистем скуп свих алгоритама потребних да се имплементира одређени начин шифровања и дешифровања.

Једнократна бележница тј. **шифра са једнократним кључем** (енгл. *one-time pad*) - шифровање са тајном случајном шифром (енгл. *secret random key, pad*), савршена шема за шифровање, немогућа за криптоанализу, тј. за дешифровање без познавања тајног кључа који се користи само једном и исте је дужине као и отворени текст.
Појам савршене сигурности је увео Клод Шенон 1949. године. Односи се на захтев да криптосистем у којем шифрат не даје никакву информацију о отвореном тексту.
Простор кључева (енгл. *keyspace*) је распон могућих вредности кључа.

Дигитални потпис (енгл. *digital signature*) је низ битова који се придружују документу при потписивању. Дигитални потпис може да идентификује аутора неке поруке, али и да докаже да порука при комуникацији није измењена (провера аутентичности и интегритета поруке). У креирању и верификацији дигиталног потписа користе се асиметрични криптосистеми, као што су DSA и RSA. Потписи су могући и применом симетричних криптосистема уз помоћ посредника.[58]

Да бисмо били сигурни да је заиста коришћен јавни кључ одговарајуће особе, користе се **дигитални сертификати** (енгл. *digital certificate*) који се шаљу заједно са кључем. Дигитални сертификати доводе у једнозначну везу идентитет пошиљаоца са његовим јавним кључем, и могу бити самопотписани или квалификовани. Компаније које издају квалификоване сертификате су сертификациона тела тј. аутентикациони центри (енгл. *Certificate Authority, CA*) и имају улогу *треће стране*.

Дигиталне временски печат (енгл. *digital timestamp*), тј. временско означавање, је додатна заштита уз дигиталне потписе.

Ови механизми се могу користити у свакодневном животу, за контролу приступа различитим подацима у рачунару, високо-безбедним инсталацијама, или ТВ каналима који се плаћају (сателитским или кабловским).

Док се модерна криптографија убрзано развија, криптографија се фундаментално заснива на проблемима које је **тешко решити** већ дуги низ година. Проблем може бити тежак зато што његово решење захтева одређено скривено знање, или зато што је сам по себи тежак за решавање.

Протокол је низ корака између два или више учесника, дефинисан да би се решио неки задатак.

Криптографски протокол је протокол који користећи криптографске алгоритме обезбеђује не само тајност, потврду идентитета и потпис, већ учествује у спречавању и откривању напада.

[58] Асиметрични и симетрични алгоритми су детаљније описани у наредном поглављу.

Између осталог, користе се за успостављање безбедне комуникације преко непоузданих глобалних мрежа и дистрибуираних система.

Криптографски стандарди су услови и протоколи који омогућавају једнозначност у комуникацији, трансакцијама и свим рачунарским активностима. Један од основних циљева које стандарди постижу је да технологије различитих произвођача "говоре истим језиком". Такође, успостављањем добро испитаних стандарда, могу се направити поузданији производи. Нажалост, и криптографски алгоритми који се користе у протоколима могу бити предмет напада.

Иако саме по себи нису протоколи, **једносмерне функције** (енгл. *one-way function*) су основни елементи за дефинисање протокола. Налажење инверзне вредности једносмерне функције треба да буде тешко (енгл. *computationally infeasible*).

Једносмерна функција са замком, тј. **привидно једносмерна функција** (енгл. *trapdoore one-way function*) је посебан тип једносмерне функције који има скривену *з*амку, чиме је израчунавање у супротном смеру отежано ако се не зна *тајна информација*, али је са том информацијом израчунавање једноставно.

Хеш функција је функција која на улазу узима скуп вредности променљиве дужине и враћа излазни скуп вредности фиксне дужине, који се назива **хеш вредност**. Пример је XOR функција свих улазних бајтова. Хеш функције обично мапирају више различитих стрингова у један. XOR није *једносмерна* хеш функција.

Једносмерна хеш функција (енгл. *one-way hash function, message digest*) је функција која ради у једном смеру — лако је израчунати хеш вредност, али је тешко из ње извести улазну вредност. Пример су MD5 и SHA, које се веома често користе, нарочито у различитим серверским и интернет апликацијама, посебно за обезбеђивање лозинки, иако то није препоручљиво. Постоје имплементације у програмским језицима PHP, MySQL. Хеш функцијама се обезбеђује интегритет поруке. Код добре једносмерне хеш функције је тешко генерисати исту хеш вредност за две различите улазне вредности (енгл. *collision-free*). Код једносмерних хеш функција не постоји нека скривена информација, већ се сигурност заснива на њиховој једносмерности.

Аутентикациони кôд поруке (енгл. *message authentication code*, **MAC**) је хеш функција која користи тајни кључ. MAC обезбеђује интегритет поруке и аутентикацију.

Центар за дистрибуцију кључева (енгл. *key distribution center, KDC*) је део криптосистема намењен смањењу ризика везаног за размену кључева.

Неки од стандарда у криптографији су: ISO, ANSI, IEEE, NIST i IETF.

Битну улогу у криптографији има влада САД-а. Њена заинтересованост за криптологију се повећава како расте употреба криптографије не само у војни сврхе, већ и у економске и рачунарске.

Влада САД-а је 1952. године основала NSA (енгл. *National Security Agency*), чији је посао било управљање владиним и војним тајним подацима као и прикупљање информација о свим врстама комуникација. Такође је основан и NIST (енгл. *National Institute of Standards and Technology*), који игра главну улогу у развоју криптографских стандарда.

Током 70-тих година прошлог века, IBM и NIST су (заједно са NSA) развили DES стандард. Овај алгоритам је био стандард од 1977. године, међутим постао је недовољно отпоран на нападе за данашње потребе шифровања. Као прелазно решење, док није уведен нови стандард AES, користио се троструки DES.

6.2 Криптографски алгоритми

Криптографски алгоритми се могу поделити у две групе:

- **Проточни алгоритми** (алгоритми тока, енгл. *stream algorithms*) — раде са поруком бит по бит, или бајт по бајт.
- **Блоковски алгоритми** (енгл. *block algorithms* или *block ciphers*) — раде са порукама у групама битова. На пример, користе се блокови величине 64 бита.

Криптографске алгоритме могу се поделити и на шифре **замене**, шифре **транспозиције** (премештања), и **комбиноване** шифре (енгл. *product ciphers*).

Најопштија подела криптосистема:

1. **Системи са тајним кључем** или **симетрични криптосистеми** (енгл. *symmetric key cryptosystem, single-key cryptosystem*). Најпознатији пример оваквог система је DES. Поред DES-а, симетрични алгоритми су и Triple DES, RC4, AES, IDEA (енгл. *International Data Encryption Algorithm*), основни алгоритам за програм PGP. Симетрични алгоритми називају се још и **конвенционални алгоритми** (енгл. *conventional algorithms*).

2. **Системи са јавним кључем**, или **асиметрични криптосистеми** (енгл. *public key cryptosystem*). Код ових система сваки корисник има јавни и свој приватни кључ. Шифровање се изводи са јавним кључем, а дешифровање са приватним. Најпознатији пример оваквог система је RSA. Такође популарани криптосистеми са јавним кључем су криптосистем ЕлГамал, ЕлГамалова шема за потпис, и DSA који се користи за дигиталне потписе. ECC (енгл. *elliptic curve cryptosystems*) су криптосистеми са јавним кључем засновани на алгебарским структурама елиптичких кривих над коначним пољима.

6.3 Криптосистеми са тајним кључем

Криптосистеми са тајним кључем су традиционалнији облик криптографије, где се један кључ користи и за шифровање и за дешифровање. Ови криптосистеми се користе за шифровања и код аутентикације. Поменимо да се техника са тајним кључем користи и код MAC који се употребљава за аутентикацију и проверу интегритета поруке, и где се обично користе криптографске хеш функције. Такође, аутентикациони кôд поруке се разликује од дигиталних потписа јер користи тајни кључ за генерисање и проверу MAC вредности.

У симетричној криптографији пошиљалац и прималац поруке знају и користе исти тајни кључ. Пошиљалац користи тајни кључ да шифрује поруку, а прималац користи исти тајни кључ да је дешифрује. Најпознатији симетрични криптосистеми су блоковски. Један од главних проблема ових система је пронаћи ефикасан унапред договорен метод за сигурну размену кључева (енгл. *key distribution problem*).

Одређене мане криптографије са тајним кључем:

1. Аутентикација путем система са тајним кључем захтева дељење неких тајни и понекад захтева поверење треће стране (енгл. *trusted third party*). Као резултат, пошиљалац може одбити претходно аутентиковану поруку тврдећи да је компромитована. Криптосистеми са јавним кључем могу обезбедити дигитални потпис који се не може одбити.

2. Проблем дистрибуције кључа. Главна предност асиметричних криптосистема у односу на симетричне је то што се приватни кључ никада не преноси нити открива, па нема проблема везаних за дистрибуцију кључа.

Одређене предности криптографије са тајним кључем:

1. Брзина. Симетрична криптографија је обично бржа од криптографије са јавним кључем.

2. Да би постигли исти степен сигурности као симетрични криптосистеми, одређени асиметрични криптосистеми морају користити кључеве *знатно* веће дужине. У неким ситуацијама, криптографија са тајним кључем може бити довољна, и нема потребе за криптографијом са јавним кључем.

Пример система за аутентикацију са тајним кључем који нема проблем одбијања претходно аутентиковане поруке је симетрични систем Керберос, који захтева поверење треће стране, а опционо у одређеним фазама аутентикације може користити и криптографију са јавним кључем [26]. Протокол аутентикације Керберос, заједно са изворним кôдом се може преузети са странице `http://web.mit.edu/kerberos/`.

DES

DES (енгл. *Data Encryption Standard*) је симетрични шифарски систем који користи 56-битни кључ (заједно са 8 додатних битова за проверу парности) за шифровање 64-битног отвореног текста

у 64-битни шифрат. Како је DES *разбијен* већ током 90-тих година прошлог века, данас се употребљава троструки DES, систем који примењује DES три пута са два или три различита кључа, такође за шифровање 64-битних порука (тј. 64-битних блокова поруке).

AES

Године 1997. National Institute of Standards and Technology (NIST) објавио је такмичење за криптосистем који треба да као опште прихваћени стандард замени DES. Победник је добио име Advanced Encryption Standard (AES).

NIST је поставио следеће захтеве за криптосистем:

- Мора бити симетричан.
- Мора бити блоковски.
- Треба да ради са 128-битним блоковима и кључевима с три дужине: 128, 192 и 256 битова.

Године 2000. је објављено да је алгоритам Рајндол (енгл. *RIJNDAEL*) победник такмичења за нови стандард за шифровање AES. Рајндол су развили белгијски криптографи Daemen и Rijmen. Алгоритам AES је и комбинована шифра, која примењује замене и премештања.

Дифи-Хелман размена кључева

Код симетричних шема за шифровање, и особа A и особа B имају исти кључ за шифровање и дешифровање порука. Како и прималац и пошаљилац морају знати исти кључ, он мора бити на неки начин прослеђен између њих, обично преко несигурних канала. Једно решење овог проблема је Дифи-Хелман размена кључа (енгл. *Diffie-Hellman key exchange*). То је **протокол усаглашавања кључа** тј. метод да две стране заједно генеришу приватни кључ преко јавног канала.

Дифи-Хелман размена кључева (основна верзија)

Особе A и B се **јавно договоре око великог простог броја p и примитивног корена g** у $\mathbb{Z}_p^*$ и објаве их.

Акције протокола: извршавају се сваког пута када је потребан **заједнички кључ** (енгл. *shared key*):

Особа A случајно бира тајни $x \in [1, p-2]$ и шаље $a = g^x \pmod p$ особи B.

Особа B случајно бира тајни $y \in [1, p-2]$ и шаље $b = g^y \pmod p$ особи A.

Особа B добија поруку a и рачуна заједнички кључ као $K \equiv a^y \equiv (g^x)^y \equiv g^{x \cdot y} \pmod p$.

Особа A добија поруку b и рачуна заједнички кључ као $K \equiv b^x \equiv (g^y)^x \equiv g^{x \cdot y} \pmod p$.

На крају и особа A и особа B имају исти кључ K, који онда могу употребити за шифровање и дешифровање својих порука.

Једине информације које су јавне су p, g, a и b. Неовлашћена особа може израчунати k на три начина: $g^{x \cdot y} \pmod p$, $a^y \pmod p$, $b^x \pmod p$. Само особа A зна x и само особа B зна y. Упркос познавању g, p и $b = g^x \pmod p$, веома је тешко наћи x. Овај проблем је познат као **проблем дискретног логаритма**, и ослања се на чињеницу да је p веома велики број, тако да x може бити нађено само применом грубе силе (енгл. *brute force search*). Заправо **Дифи-Хелман проблем** каже да за дати прост број p, генератор g од $\mathbb{Z}_p^*$, и елементе $g^x \pmod p$ и $g^y \pmod p$ је тешко наћи $g^{x \cdot y} \pmod p$. Зато је Дифи-Хелман проблем веома важан за криптографију са јавним кључем. Он представља основу Дифи-Хелман размене кључева и криптосистема ЕлГамал са јавним кључем. Приметимо да се за проверу простости генерисаних бројева могу користити тестови простости.

6.4 Криптосистеми са јавним кључем

Као што је објашњено, у симетричној криптографији сви кључеви морају да остану тајни, па често постоје потешкоће са управљањем кључевима, нарочито код отворених система са великим бројем корисника. Да би решили овај проблем, 1976. године Дифи и Хелман су увели концепт криптографије са јавним кључем тј. асиметричне криптографије, из које је настао концепт **инфраструктуре система са јавним кључем** (енгл. *Public Key Infrastructure, PKI*). Због ове чињенице, 1976. година се може сматрати прекретницом у дугој и интересантној историји криптографије.

Криптосистеми са јавним кључем, поред улоге у дистрибуцији кључа за симетричне алгоритме, имају примене у шифровању (за обезбеђивање приватности) и код дигиталних потписа (за аутентикацију). У овом систему, свака особа добија пар кључева, један јавни и један приватни кључ. Јавни кључ се објави, док приватни остаје тајан. Потреба да пошиљалац и прималац деле тајне информације је елиминисана; сва комуникација укључује само јавни кључ, а приватни кључ се не преноси нити дели. Свако може послати поверљиву поруку користећи само јавне информације, али порука може бити дешифрована само са приватним кључем, кога поседује само жељени прималац. Овде постоји **проблем аутентичности јавног кључа**, тј. учесници у комуникацији морају бити сигурни да јавни кључеви заиста припадају одговарајућим особама.

У криптосистемима са јавним кључем, **приватни кључ је увек математички повезан са јавним кључем**. Зато је увек могућ напад на систем са јавним кључем, тако што се приватни кључ изведе из јавног кључа. Обично се одбрана заснива у отежавању овог извођења. На пример, извођење приватног из јавног кључа захтева од нападача да факторише велике бројеве. Ово је идеја која стоји иза криптосистема RSA.

Заправо, услов да би асиметрични криптосистем функционисао како треба је да кључ за шифровање и кључ за дешифровање буду повезани, али да је практично немогуће извести један кључ ако је познат други. Тешко извођење једног кључа из другог значи да се кључ за шифровање може јавно објавити без компромитовања кључа за дешифровање. Прости бројеви, а самим тим и тестови простости, чине битан део криптосистема са јавним кључем.

Као што смо видели, основну идеју криптосистема са јавним кључем увели су Дифи и Хелман 1976. године, а током наредних година предложено је неколико различитих метода за имплементацију њихове идеје, међу којима и алгоритам RSA.

Шифровање

Када особа A жели да пошаље поруку особи B, она погледа јавни кључ особе B, користи га да шифрује поруку и пошаље је. Особа B онда користи свој приватни кључ да дешифрује поруку и прочита је. Нико ко *слуша* не може да дешифрује поруку. Свако може да пошаље шифровану поруку особи B, али само особа B може да је прочита (зато што само она зна свој приватни кључ).

Дигитални потпис

Да би потписала поруку, особа A при израчунавању мора да узме у обзир свој приватни кључ (приватни кључ особе A) и саму поруку. Излаз се назива дигитални потпис и он се додаје на поруку. Да би потврдила потпис, особа B врши израчунавања која укључују поруку, садржај потписа и јавни кључ особе A. Ако је резултат одређен математичким релацијама тачан, потврђује се да је потпис оригиналан. У супротном, потпис је лажан.

Криптосистем RSA

Ривест, Шамир и Адлеман су 1983. године патентирали шифарски систем са јавним кључем познат под именом RSA. Алгоритам RSA је један од најпознатијих и најпримењиваних асиметричних алгоритама. RSA је ефикасан у шифровању кратких порука и за креирање дигиталних потписа. Сигурност алгоритма се заснива на чињеници да је изузетно тешко факторисати велике бројеве на просте факторе. Основни кораци тог алгоритма су:

- Сваки корисник A изабере два различита проста броја p и q и рачуна $n = p \cdot q$. То значи да корисник A користи групу $\mathbb{Z}_n^*$. Ред те групе је $\phi(n) = \phi(p \cdot q) = (p-1) \cdot (q-1)$. Кориснику A је лако да израчуна овај ред, пошто зна p и q.

- Затим A изабере позитиван број e, $1 \leq e < \phi(n)$, такав да је узајамно прост са редом групе, тј. такав да је $\text{nzd}(\phi(n), e) = 1$ (другачије речено: e и производ $(p-1) \cdot (q-1)$ су узајамно прости).

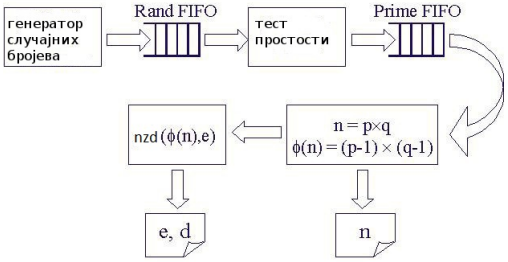
- На пример, помоћу проширеног Еуклидовог алгоритма корисник A израчуна инверзни елемент d од e у $\mathbb{Z}_{\phi(n)}$, број d је јединствен мултипликативни инверз од e по модулу $\phi(n)$. Значи $e \cdot d \equiv 1 \pmod{\phi(n)}$, где је $1 \leq d < \phi(n)$, тј. $d \equiv e^{-1} \pmod{(p-1) \cdot (q-1)}$.

- Јавни кључ корисника A је пар (n, e), док је његов приватни кључ број d.

- Бројеви p, q и $\phi(n)$ такође се морају држати у тајности.

- Ако корисник A жели да пошаље поруку m другом кориснику B, користи јавни кључ корисника B, (n_b, e_b) да би израчунао вредност c, $m^{e_b} \equiv c \pmod{n_b}$, коју шаље кориснику B.
- Да би реконструисао поруку, B користи свој приватни кључ, и рачуна $c^{d_b} \equiv (m^{e_b})^{d_b} \equiv m^{e_b \cdot d_b} \equiv m \pmod{n_b}$, јер за произвољан број a важи: $a^{e \cdot d} \equiv a \pmod{n}$.

Напомена 6.1. Да би корисник A шифровао поруку m, он прво треба да је подели на нумеричке блокове мање од n. Као и код свих блоковских шифри, начин како се алгоритам шифровања узастопно примењује на ове блокове зависи од тога који метод заштите, односно који начин употребе блоковске шифре се користи [10, 12]. Шифрована порука c је састављена од блокова поруке отприлике исте дужине.



RSA: системска архитектура за генерисање кључа

Исправно функционисање алгоритма RSA зависи од тога да ли важи $m^{d \cdot e} \equiv m \pmod{n}$. Важи да је $d \cdot e = 1 + k \cdot \phi(n)$. Ако је $\text{nzd}(m, n) = 1$, на основу Ојлерове теореме следи да је $m^{d \cdot e} = m \cdot (m^{\phi(n)})^k \equiv m \pmod{n}$. Претпоставимо сада да је m дељив са неким од простих бројева p или q. Нека је m дељив са p. Онда је $m^{1+k \cdot (p-1)(q-1)} \equiv 0 \equiv m \pmod{p}$, а на основу Ојлерове теореме је $m^{1+k \cdot (p-1)(q-1)} = m \cdot (m^{(q-1)})^{k \cdot (p-1)} \equiv m \pmod{q}$. Како је $\text{nzd}(p, q) = 1$, на основу последице 2.1 важи и $m^{d \cdot e} = m^{1+k \cdot (p-1)(q-1)} = m \pmod{p \cdot q}$.

Пример 6.1 (Примери употребе RSA)**.**

- Оперативни системи. Microsoft, Apple, Google, Oracle (Sun), Novell. Многе велике фирме уградиле су RSA у своје програме или оперативне системе.
- Хардвер. Мобилни телефони, *ATM* машине, опрема за бежичне мреже (енгл. *wireless network device*), паметне картице (енгл. *smart card*).
- Безбедна интернет комуникација. Претраживачи, *SSL, S/WAN, S/MIME, PGP, e-mail*.

Параметри алгоритма RSA су: p, q, два проста броја (приватни, генерисани), $n = p \cdot q$ (јавни, израчунава се), e такав да $\text{nzd}(\phi(n), e) = 1$ (јавни, одабира се), $d \equiv e^{-1} \pmod{\phi(n)}$ (приватни, израчунава се). Јавни кључ је (e, n), а приватни кључ је број d, тј. пар (d, n).

Пример 6.2. Дат је једноставан пример шифровања броја помоћу алгоритма RSA[59]. Корисник A формира јавни и тајни кључ:

- Прво бира два проста броја $p = 2399$ и $q = 3001$.
- Затим израчунава број n, $n = p \cdot q = 7199399$.
- Онда се израчунава број $\phi(n) = (p-1) \cdot (q-1) = 7194000$.
- Случајно се бира број e такав да је $\text{nzd}(\phi(n), e) = 1$, $e = 3874979$ (део јавног кључа).
- Одговарајућим алгоритмом се рачуна d, $d = e^{-1} \pmod{\phi(n)}$, $d = 460619$, и то је приватни (тајни) кључ. (овај инверз постоји, јер су $\phi(n)$ и e узајамно прости).
- Дакле, јавни кључ је пар $(e = 3874979, n = 7199399)$, а приватни кључ је пар $(d = 460619, n = 7199399)$.

[59] Списак простих бројева са мање од 10 цифара, као и могућност провере да ли је одређени број прост, дати су на страници: `www.prime-numbers.org`.

- Да би особа B шифровала поруку $m = 987654$ (то је број који треба послати) особи A мора да користи јавни кључ особе A и да рачуна $c = m^{e_a} \pmod{n_a} = 3398479$.
- Тај број $c = 3398479$ (шифрат оригиналне поруке m) особа B шаље особи A. Особа A приступа дешифровању тако што користи свој тајни кључ и рачуна $c^d \pmod{n_a} = 987654$, а то је баш оригинална порука m.

Ради једноставности у примеру смо користили мале бројеве. У реалним имплементацијама криптосистема RSA обично се користе 1024- или 2048-битни бројеви, који имају око 300 односно око 600 цифара, и користе Милер-Рабинов тест за проверу простости бројева p, q који се генеришу. Такође, у израчунавањима се уместо Еуклидовог алгоритма користи Кинеска теорема о остацима. Могуће су и друге модификације и побољшања [12].

Битно је истакнути да компанија RSA Security тврди да су 1024-битни RSA кључеви еквивалентни по сигурности 80-битним симетричним кључевима, 2048-битни кључеви еквивалентни 112-битним симетричним кључевима, а 3072-битни RSA кључеви еквивалентни 128-битним симетричним кључевима. Такође, 1024-битни кључеви су подложни разбијању, док се тврди да су 2048-битни кључеви довољне сигурности до 2030. године, а након тога треба користити 3072-битне кључеве.[60]

Напомена 6.2. Веома је битан и избор бројева p и q јер неке вредности омогућавају лакше разбијање шифре [12].

Једине информације које се *деле* преко јавних канала су n, e и c, тако да свако може да шифрује поруку и пошаље је особи A, али само особа A може да је дешифрује са d. Процес шифровања и дешифровања су брзи зато што m^e и c^d могу бити израчунати неким од ефикасних алгоритама степеновања.

Да би се демонстрирало да овај систем омогућује безбедну комуникацију, показује се да је порука безбедна од напада неке треће особе. Неовлашћена особа може да зна само c, n и e. Постоји неколико начина на које она може да употреби ове информације да би нашла m. На пример, $c = m^e \pmod{n}$, тако да је потребно израчунати e-ти корен од c по модулу n. Међутим, тражење корена по модулу n је, као што је речено, тежак проблем. Како је n велики број, није могуће у пракси погодити m и проверити да ли је m^e једнако c. Ово је познати **RSA проблем**: за дати позитиван број n који је производ два различита проста броја p и q, позитиван цео број e такав да је $\text{nzd}(e, (p-1)(q-1)) = 1$, и цео број c, наћи цео број m такав да је $m^e \equiv c \pmod{n}$.

Неовлашћена особа може и да покуша да открије тајни кључ d. Кључ d је повезан са јавним кључем једначином: $d \cdot e \equiv 1 \pmod{\phi(n)}$. Међутим, да би се израчунало d из ове једначине, мора се знати $\phi(n)$, а једини начин да се $\phi(n) = (p-1) \cdot (q-1)$ израчуна је да се знају прости фактори од n, p и q. Када се зна $\phi(n)$, лако је наћи d на основу e користећи проширени Еуклидов алгоритам. Зато се верује да је откривање d еквивалентно факторизацији броја n, што непријатељу треба да онемогући да открије m.

Пример 6.3. Налажење бројева p и q ако је познато $\phi(n)$:
$a = n - \phi(n) + 1 = p \cdot q - (p-1) \cdot (q-1) + 1 = p \cdot q - (p \cdot q - p - q + 1) + 1 = p + q$
$b = \sqrt{a^2 - 4 \cdot n} = \sqrt{(p+q)^2 - 4 \cdot n} = \sqrt{p^2 + 2 \cdot p \cdot q + q^2 - 4 \cdot p \cdot q} = \sqrt{(p-q)^2} = p - q$
Када смо израчунали a и b онда су p и q једнаки: $p = \frac{a+b}{2}$, $q = \frac{a-b}{2}$.

Поред свега што је речено, потпуно је оправдано **постојање озбиљних резерви и критика у погледу сигурности критпосистема RSA**, нарочито за дужине кључева које су у широкој употреби, а које су често и мање величине од препоручене.

[60] За више информација, видети `http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf`.

Криптосистеми ЕлГамал и DSA

Криптографски алгоритам ЕлГамал је асиметрични криптографски алгоритам заснован на проблему дискретног логаритма, и разликује се од **ЕлГамалове шеме за потпис** (енгл. *ElGamal signature scheme*). ЕлГамалова шема за потпис се ретко користи. **DSA** (енгл. *Digital Signature Algorithm*) је варијанта ЕлГамалове шеме за потпис. DSA је део стандарда дигиталног потписа DSS (енгл. *Digital Signature Standard*), и често се користи.

ЕлГамал криптографски алгоритам се обично користи за шифровање кључева који се онда користе у симетричним криптосистемима, иако се може користити и за шифровање самих порука. Проналазач је ЕлГамал[61].

Исте године када се појавио алгоритам ЕлГамал, предложен је криптосистем заснован на тежини израчунавања дискретног логаритма над елиптичким кривама.

Криптосистеми засновани на елиптичким кривама

Употребу елиптичких кривих у криптографији су 1985. године, независно један од другог, предложили Коблиц[62] и Милер[63].

Криптосистеми ECC се могу поделити у две категорије, на основу тога да ли су аналогни систему RSA или системима заснованим на дискретним логаритмима. Испоставило се да су системи ECC аналогни RSA углавном само од академског значаја и немају практичних побољшања у односу на RSA, јер се њихова безбедност заснива на истом проблему, факторизацији великих бројева. Ситуација је прилично другачија када су у питању ECC аналогни системима заснованим на дискретним логаритмима. Безбедност ових система зависи од следећег тешког проблема: За дате две тачке P и Q на елиптичкој кривој, такве да је $P = k \cdot Q$, наћи цео број k. Овај проблем се обично назива **проблемом дискретног логаритма елиптичке криве** (енгл. *elliptic curve discrete logarithm problem*).

За сада је ефикасност израчунавања дискретног логаритма елиптичке криве много мања од ефикасности оних за факторисање или израчунавање обичних дискретних алгоритама. Као резултат ове чињенице, кључеви мање величине могу се користити да би се постигла иста безбедност као код конвенционалних криптосистема са јавним кључем, што може водити бољој искоришћености меморије и побољшању перформанси. Могу се лако конструисати шеме за шифровање, потпис, или размену кључева засноване на елиптичким кривама, које су аналогне ЕлГамал, DSA, или Дифи-Хелман. На пример, алгоритам ECDSA (енгл. *Elliptic Curve Digital Signature Algorithm*) је варијанта DSA заснована на елиптичким кривама. Ове варијанте често показују одређене предности у имплементацији у односу на оригиналне шеме, и привлаче све више пажње. Криптографија заснована на елиптичким кривама добија све више на популарности.
За више информација о криптографији, видети [5, 12, 28, 30].

Програми PGP и GnuPG

Програм PGP (енгл. *Pretty Good Privacy*) је написао Zimmerman. Служи за безбедну комуникацију путем електронске поште (енгл. *e-mail*). Користи се и за потписивање, шифровање и дешифровање текстова, датотека, електронске поште. Написан је у C-у (критични делови на машинском језику) и пренесен на велик број платформи и оперативних система. PGP користи комбинацију криптографије са тајним кључем и криптографије са јавним кључем. Он користи између осталих алгоритама, алгоритам IDEA за шифровање порука, RSA за управљање кључевима и дигиталне потписе, и једносмерну хеш функцију MD5. Програм је бесплатан и врло популаран. PGP нуди шифровање/дешифровање порука, аутентикацију путем дигиталних потписа, компресију података, компатибилност са SMTP, POP3 и MIME стандардима комуникације путем електронске поште.

Компатибилност са електронском поштом је постигнута путем *Radix-64* конверзије података пре слања [26]. Има разрађен систем чувања јавних и тајних кључева у сопствене мале базе података. Даје могућност накнадног потписивања и одређивања "степена поверења" самих кључева. Уз сваки

[61] ЕлГамал (Taher Elgamal), 1955– , египатски криптограф.
[62] Коблиц (Neal Koblitz), 1948 – , амерички математичар.
[63] Милер (Victor S. Miller), 1947– , амерички математичар.

појединачни кључ сачувана је и MD5 информација о контролном збиру (енгл. *checksum*) кључа и времену креирања.

Могу се наћи две верзије програма, Америчка и интернационална, због извозних клаузула САД-а. Интернационалну верзију је забрањено употребљавати у САД-у, а Америчку ван САД-а. По функционалности и компатибилности обе верзије су идентичне. На Интернету се могу пронаћи и PGP KeyServer сервиси који су специјализовани за чување и дистрибуцију PGP јавних кључева, а сами су међусобно повезани у један ланац.

Програм GNU Privacy Guard (GnuPG или GPG) је алтернатива за PGP криптографски софтверски пакет. То је такође хибридни криптографски софтвер, јер користи и симетричну и криптографију са јавним кључем. GnuPG је компатибилан са OpenPGP стандардом. Иако је првобитно креиран као програм са интерфејсом преко командне линије, развијени су и графички интерфејси, и GnuPG је интегрисан у многе e-mail клијенте, нарочито за Linux оперативни систем.

Корпорација Symantec је 2010. године купила PGP. Старије верзије програма, као и GnuPG са изворним кодом, могу се пронаћи на страници `www.pgpi.org`.

6.5 Протоколи

Криптографски протокол је алгоритам дефинисан низом корака који прецизно одређују акције које је потребно да два или више ентитета изврше да би постигли одређени сигурносни циљ. Сигурносни протоколи примењују криптографске методе да би омогућили сигурну размену података.

Комуникациони протокол је систем формата за дигиталне протоколе и правила за размену тих порука између рачунарских система.

Пример једноставног протокола је протокол за усаглашавање кључа (енгл. *key agreement protocol*) преко несигурног канала. Такође, TLS је криптографски протокол који се користи за сигурне HTTP конекције. Сада је дат преглед неких протокола који се често користе.

Протоколи TLS и SSL

Протокол TLS (енгл. *Transport Layer Security*) и његов претходник SSL (енгл. *Secure Socket Layer*) су криптографски протоколи који обезбеђују сигурнију комуникацију преко Интернета, користећи асиметричну криптографију за размену кључева, симетричну криптографију за поверљивост (енгл. *confidentiality*), и MAC за интегритет поруке. Протоколи TLS и SSL су протоколи руковања (енгл. *handshake protocol*). Протокол SSL је развијен од стране компаније Netscape. Протокол подржава идентификовање и осигуравање аутентичности и послужиоца и корисника.

Сам протокол руковања се састоји од две фазе: фазе идентификације послужиоца и фазе идентификације корисника.

Фаза идентификације корисника је опциона, док приликом фазе идентификације послужиоца корисник добија информацију о сертификату (потписаном јавном кључу) заједно са подржаним начинима шифровања. Тада се са корисничке стране генерише главни кључ (енгл. *master key*), потписује се јавним кључем послужиоца и шаље назад. Послужитељ тада дешифрује садржај корисниковог главног кључа и сигнализира кориснику да је све у реду шаљући поруку шифровану корисниковим главним кључем. Остатак комуникације тече шифрованим кључевима изведеним из главног кључа.

Уколико се захтева идентификација корисника, тада послужитељ то сигнализира кориснику који пошаље назад свој потписани јавни кључ.

Новије верзије ових протокола су SSL 3.0 и TLS 1.2.

Протокол HTTPS

Протокол HTTPS (енгл. *Hypertext Transfer Protocol Secure*) је протокол замишљен као допуна стандардном HTTP-у (енгл. *Hypertext Transfer Protocol*) у правцу додавања сигурних сервиса путем криптографије. У широкој је употреби за обезбеђивање сигурније комуникације на Интернету.

Технички, HTTPS није одвојени протокол, већ представља HTTP преко шифроване SSL/TLS конекције.

Протокол HTTPS омогућује аутентикацију веб странице и повезаног веб сервера, као и шифровање комуникације између клијента и сервера. Основна идеја је креирање сигурног канала преко

несигурне мреже да би се обезбедила разумна заштита. Администратори веб сервера који прихватају HTTPS конекције треба да креирају сертификат [26]. Интернет претраживачи знају којој HTTPS интернет страници да верују на основу сертификационих ауторитета који су уграђени у претраживаче. HTTPS *url* почиње са "https://" и подразумевано користи порт 443, док HTTP користи порт 80 или порт 8080.

Како протокол SSL ради испод протокола HTTP[64], SSL сервери могу да дају само један сертификат за одређену IP/порт комбинацију.

Протокол SSH

Протокол SSH (енгл. *Secure Shell Protocol*) је криптографски протокол за сигурно повезивање рачунара и дељење информација. Он се најчешће употребљава за логовање на машине у локалној мрежи и извршавање команди, мада се може користити и за сигуран трансфер датотека, тунеловање (енгл. *tunneling*), а данас постаје посебно значајан у *cloud computing*-у. За успостављање протокола SSH је потребно повезати SSH сервер и SSH клијент. SSH сервери обично користе TCP порт 22.

Данас је у употреби верзија SSH 2. Протокол SSH 2 користи, између осталог, алгоритме RSA и DSA за аутентикацију помоћу јавног кључа, и Керберос као спољашњи механизам за аутентикацију. Најпопуларнија имплементација протокола SSH је софтвер OpenSSH. Софтверски пакет OpenSSH је *open source* програм који је креиран као алтернатива софтверском пакету Secure Shell.

Протокол S/MIME

Протокол S/MIME (енгл. *Secure/Multipurpose Internet Mail Extensions*) је протокол који се директно надограђује на постојећи Internet MIME протокол додајући могућност аутентикације, интегритета поруке, непорецивости, приватности и сигурности података путем шифровања електронске поште. Стандард S/MIME је усвојио велики број произвођача програма (ConnectSoft, Microsoft, Lotus, VeriSign, Netscape, Novell), али одређени програми и webmail клијенти не подржавају S/MIME потписе.
За више информација, видети `http://datatracker.ietf.org/wg/smime/`.

Програмски језик Obol

Програмски језик Obol[65] развијен на Универзитету у Тромсу је специјализовани језик високог нивоа. Намењен је искључиво имплементацији сигурносних протокола. Идеја целог пројекта је да омогући фокусирање на опис и анализу самих сигурносних протокола тако да остатак непотребних рутинских послова (као што су комуникација, примитивне криптографске трансформације и презентација порука) обавља програмско окружење, чиме се избегавају типичне сметње које настају због грешака у тим деловима програмске имплементације сигурносних протокола.

О Obol-у се може размишљати као о програмском језику који жели да постигне за сигурносне протоколе оно што је SQL урадио за базе података.

Имплементиран је у Јави, користећи ANTLR за парсирање. Старији прототипови су имплементирани у Common Lisp-у и Python-у.

[64] Протокол HTTPS ради на највишем нивоу TCP/IP модела, на апликационом, а сигурносни протоколи TLS/SSL раде испод овог нивоа. За више информација, видети и `http://en.wikipedia.org/wiki/OSI_model`.

[65] Постоји и објектно-оријентисани програмски језик под истим називом, као и Obol — "A Language for Open Bio-Ontologies".

7 Програмска реализација одређених алгоритама

Програмску реализацију алгоритама олакшава примена неких постојећих програма. Следи кратак преглед софтвера који омогућава да се развију бољи програми, лакше провере добијени резултати и изврше потребне оптимизације.

7.1 Помоћни алати

Испитивање датотека који чувају податке у бинарном облику и провера вредности одређеног бита је важна, јер се обично информације о прималности бројева из неког интервала чувају у појединачним битовима за сваки број, у одређеном облику.

Hex едитори омогућују приступ појединачним битовима података у датотеци. Подаци се могу приказати у хексадецималном облику, октално, или децималном. Hex едитори олакшавају руковање великим датотекама (датотекама које заузимају неколико GB простора на диску), упоређивање и проналажење разлика између две датотеке, једноставно мењање појединачних битова.

Софтверски пакет **Wolfram Mathematica** обезбеђује хиљаде алгоритама, укључујући алгоритме за рад са простим бројевима, једноставно генерисање графика и потпуно интерактивних апликација, као и манипулацију подацима.

Алати за **профилисање програма** (енгл. *program profiling*) врше анализу програма и његових перформанси, и веома су битни у обезбеђивању информација потребних за оптимизацију програма. Најзначајнији је алат **gprof** који се користи у различитим верзијама оперативног система Linux, али и у окружењу **cygwin** за Windows. Корисна је и time команда у Linux-у.[66]

7.2 Софтвер за рад са великим бројевима

У већини програмских језика, целобројни тип података је 32-битни или 64-битни. Са појавом 64-битних процесора је омогућено адресирање довољно меморије, што је често битан фактор при раду са великим бројевима. Стандард ANSI/ISO C је ажуриран 1999. године и додати су нови целобројни типови у програмски језик C, између осталих и signed long long и unsigned long long. Ова имена потичу од gcc и још неколико других компајлера који су први подржали овај тип. На Windows преводиоцима, као што су MS Visual и C++Builder, ово исто проширење је имало име *__int*64.[67]

Тип података unsigned long long може садржати све вредности између 0 и ULLONG_MAX, где је ULLONG_MAX најмање $2^{64} - 1$. Тип signed long long може садржати све вредности од $-(2^{63})$ до $2^{63} - 1$.

Стандард C++11 (раније познат као C++0x) је најновији стандард програмског језика C++. У C++11 је додата подршка за наведене типове података. Организација ISO је одобрила овај стандард 2011. године.

Рад са *великим* целим бројевима подржан је и у другим програмским језицима. На пример, систем за управљање базама података MySQL има тип података BIGINT који обухвата исти скуп вредности као и long long у C-у. У програмском језику SQL постоје аналогни типови bigint, unsigned bigint, а у Pascal-у тип података int64.[68]

Већина апликација у којима се користе прости бројеви захтева много веће вредности, као што су 1024 или 2048-битне вредности, па је зато потребан нови тип података који ради са овако великим бројевима, са тзв. *big integer* типом података.

У програмској језику Јава је обезбеђена класа **BigInteger**. Она омогућава **чување целих бројева произвољне величине**, и обезбеђује многобројне алгебарске функције и битске операције, као што су метод isProbablePrime за тестирање простих бројева, функција nextProbablePrime, генератор великих случајних простих бројева probablePrime, функција modPow за модуларно степеновање, функција gcd за тражење нзд, функција modInverse за тражење инверза по модулу n.[69]

[66] За више информација о овим алатима видети `http://es.elfak.ni.ac.rs/es/Materijal/Pog.55\-Tajming$\%$20i$\%$20profilisanje.doc`, `http://poincare.matf.bg.ac.rs/~milan/download/nar/gnu.ps`.

[67] За више информација, видети `http://msdn.microsoft.com/en-US/library/s3f49ktz`.

[68] За додатне информације, видети `http://en.wikipedia.org/wiki/C%2B%2B11`.

[69] Документација о овој библиотеци: `http://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html`.

Међутим, не обезбеђује налажење n-ог корена нити логаритама.

Библиотека **BigDigits** је библиотека аритметичких рутина написана на ANSI C-у, која омогућава извршавање рачунских операција над великим природним бројевима.[70] Осим класичних аритметичких алгоритама, укључује и модуларно множење, степеновање и налажење инверза, као и израчунавање Јакобијевог симбола. Имплементиран је и Милер-Рабинов вероватносни тест прималности. Корисно је видети и кôд за ову библиотеку, јер су одређене рутине урађене на асемблерском језику, многе функције су имплементиране на основу алгоритама датих у књизи Knuth, Vol 1, 2. При тестирању да ли је дати број прост прво се проверава да ли је број дељив неким простим бројем до 1000, где се користи унапред генерисана листа простих бројева, а тек онда се примењује Милер-Рабинов тест.

Библиотека **GMP** (енгл. *Gnu multiple-precision Library*) је напреднија и већа библиотека за рад са великим целим бројевима, написана у *C*-у. Она обезбеђује све што и Јавина класа BigInteger, али има и неке новине: израчунавање најмањег заједничког садржаоца, Јакобијевих и Лежандрових симбола, факторијела, FFT множење, рачунање биномних коефицијената, Фибоначијевих бројева, Лукасових бројева, као и функције за налажење n-ог корена. Библиотека GMP омогућава и проверу да ли је неки број савршен, као и рад са рационалним и реалним бројевима. Како је написана на *C*-у, не само да је комплетнија од BigInteger, него је и бржа.[71]

Библиотека **Intel IPP** (енгл. *Intel Integrated Performance Primitives*) је Интелова библиотека направљена са циљем да унесе заштиту у софтверске апликације и да користећи криптографске функције побољша приватност, контролу приступа, поверење, електронско плаћање. Криптографске функције ове библиотеке укључују имплементацију алгоритама DES, TripleDES, Rijndael, SHA1, RSA. Програмски језици који имају подршку за ову библиотеку су C, C++, Fortran 90, Microsoft .NET (*C#*, Visual Basic), Delphi, Java.

Библиотека IPP подржава Интел и компатибилне процесоре, а постоје верзије за Windows, Linux, Android, OS X. За разлику од горенаведених библиотека, ова библиотека није доступна бесплатно.

7.3 Програмске реализације сита за просејавање простих бројева

Модификовано Ератостеново сито

У прилогу који иде уз овај рад дат је програм који представља модификацију програма са интернет странице `http://www.fpx.de/fp/Software/sieve.c`. Овај програм може за неколико минута генерисати базу свих простих бројева до 10^{10}, то јест наћи првих 455,052,511 простих бројева и записати их у датотеку. База која се у овом случају генерише заузима око 4.60 GiB простора на диску, али је величина променљиве niz која у току програма чува информације о прималности бројева до 10^{10} *само* око 596 MiB, тј. 625,000,000 бајтова.

У другој верзији овог сита сви бројеви са којима се ради су облика $6k + 1$ или $6k + 5$. При дељењу са 24, остатака који су овог облика има 8, па се у једном бајту чувају информације о простим бројевима из интервала величине 24. Количина меморије која се на овај начин заузима је знатно мања него код претходног програма — овај програм заузима 397 MiB за генерисање свих простих бројева до 10^{10}, док претходни програм заузима 1.5 пута више.

Како је $\phi(2 \cdot 3 \cdot 5) = 8$, ово сито се лако може изменити да заузима мање меморије јер један бајт *покрива* 30 уместо 24 броја.

Сегментисано модификовано Ератостеново сито

У додатку је дата имплементација у програмском језику C модификованог Ератостеновог сита чија је идеја изложена у поглављу 3. Такође је реализована још једна верзија, где се низ gap користи за маркирање следећег сложеног броја у процесу просејавања. Код овог модификованог сита треба искористити чињеницу да је највећи размак између два елемента у сведеном систему остатака (који је описан у поглављу 3) једнак 34, а занимљиво је поменути и да је највећи размак између два проста броја до 9,699,690 једнак 222. Иако то овде није урађено, лако се може применити оптимизација

[70] Више о овој библиотеци видети на интернет страници `http://www.di-mgt.com.au/bigdigits.html`.
[71] Библиотека GMP може се преузати са веб презентације `http://gmplib.org/`.

да се у програму за бројеве до 2^{32} користи тип података unsigned long, а онда за бројеве до 2^{64} се користи тип unsigned long long.
Времена извршавања претходно описана четири сита приказана су наредним графиком.

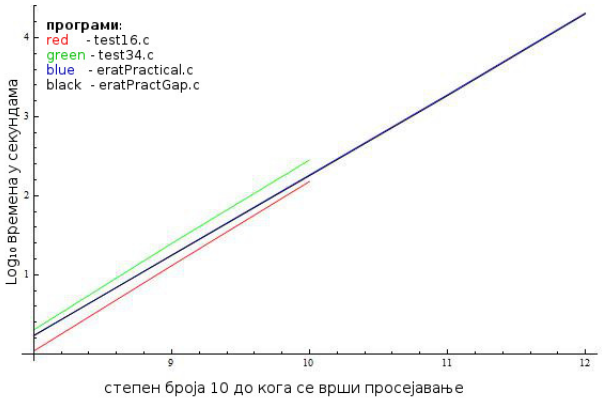


График времена извршавања датих сита

Графиком су приказана времена извршавања програма за проналажење простих бројева до одређеног броја, без њиховог записа.

Модификована Ератостенова сита доносе одређена побољшања у брзини извршавања програма, али и у величини интервала за просејавање, и разлику у начину записа информација о простости бројева. У наредној табели приказано је време за које програм проналази и записује информације о простим бројевима. Као што се види из претходног графика, прво сито test16.c је веома ефикасно за проналажење малих простих бројева, али је доста времена утрошено на њихов запис.

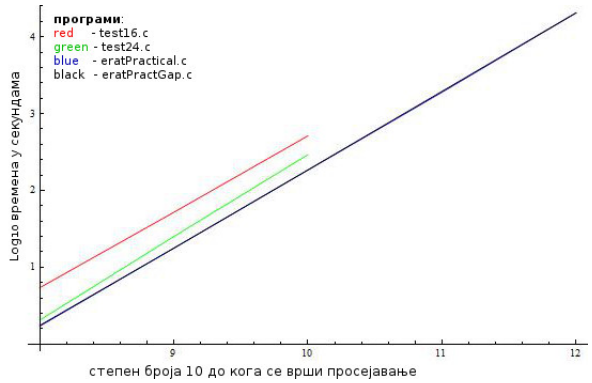


График времена извршавања сита за проналажење и запис простих бројева

N	test16	test24	eratPactical	eratPractGap
10^8	5.42	2.05	1.71	1.75
10^9	52.12	25.19	17.81	17.75
10^{10}	515.46	292.85	184.59	184.49
10^{11}	–	–	1909.45	1958.50
10^{12}	–	–	20584.30	20646.50

Времена извршавања програма (у секундама) за налажење и запис података о простим бројевима

Времена извршавања ових програма на рачунару Intel Core i3 CPU 2.2 GHz, 6GB RAM и оперативном систему Windows 7, уз употребу програма cygwin, су упоређена и на основу графика следи да су асимптотска времена извршавања ових програма слична. Програми су употребљени за генерисање база простих бројева до 10^{12}. Треба приметити да прва два сита не успевају да алоцирају довољно меморије за интервале веће од 10^{10}, док су наредна два сита сегментисана, па је омогућено просејавање већих интервала.[72]

Ради поређења, на рачунару Intel Core2Duo, 2.67GHz, 2GB RAM меморије, потребно је 25,168 секунди за налажење и запис простих бројева из интервала 10^{12} уз помоћ другог модификованог сита. На истом рачунару, на оперативном систему Windows 7, програм cygwin и компајлер gcc, за проналажење и запис информација о простим бројевима до 10^{13} је потребно 296,276 секунди (око 82h). Овим програмом је пронађено и записано свих $346,065,536,839$ простих бројева до 10^{13}, чиме је направљена релативно велика база простих бројева која заузима око 200 GiB на диску, тачније $215,363,629,056$ бајтова.

[72] За генерисање графика је употребљен софтвер Mathematica и програм Gimp за рад са сликама.

На основу претходних графика и запажања, следи да иако се претходна сита могу оптимизовати, то не доноси значајнија побољшања. Из овога се може закључити да је **потребан другачији приступ и ефикаснији начин за просејавање** простих бројева, како бисмо постигли боље резултате. Најбоља сита за рад са великим интервалима бројева су дата у поглављу 4.

За више информација о сличним програмима видети интернет презентацију [2], као и `http://primes.utm.edu/links/programs/sieves/Eratosthenes/C_source_code/`, где су илустроване и другачије технике програмског кодирања које могу бити веома корисне у различитим програмима за тестирање прималности бројева, генерисање простих бројева или факторизацију.

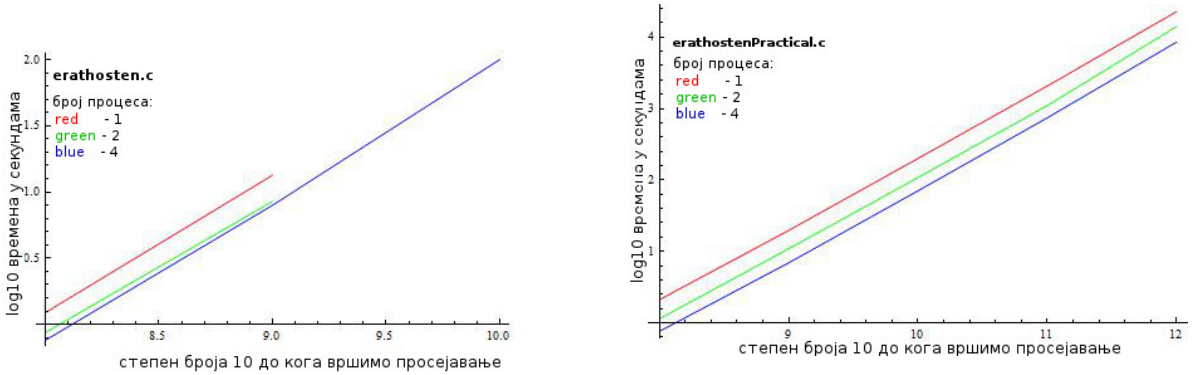
Као пример једноставног сита реализованог у програмском језику Јава, видети стандардну библиотеку BitSieve.

7.4 Паралелизација модификованог Ератостеновог сита

Програм **сегментисаног модификованог Ератостеновог сита** који је изложен у поглављу 3 може се једноставно паралелизовати у сврху генерисања *велике* базе простих бројева. Паралелна имплементација је дата у прилогу. Реализација је дата у програмском језику C, уз употребу функција из MPI. Ради добијања информација о временима извршавања битних делова програма и њихове оптимизације, коришћен је алат gprof.

Анализа програма подразумева како анализу времена извршавања (временску сложеност), тако и анализу меморијског простора који програм захтева (просторну сложеност). Дати програм показује добре карактеристике на основу оба критеријума.

Времена извршавања овог програма су упоређена са временима извршавања једноставног Ератостеновог сита чија је паралелизација дата у поглављу 5. Иако је стандард MPI прилагођен за системе са дистрибуираном меморијом, видимо да библиотека MPICH2 (као и OpenMPI) добро функционише и за рачунаре са вишејезгарним процесорима. Програми су извршавани на процесору Intel Core i3 CPU 2.2 GHz, 6GB RAM.



Графици који упоређују времена извршавања програма са 1, 2, и 4 процеса

Графиком су упоређена времена извршавања модификованих Ератостенових сита за налажење и бројање колико има простих бројева у одређеном интервалу. Ако се узме у обзир процесор на коме се програми извршавају, који има два језгра, а на коме је омогућено извршавање две процесорске нити по језгру[73], следи да модификовано сито показује добре карактеристике када се изврши паралелизација, јер су паралелизације тривијалне и није утрошено пуно времена на комуникацију међу процесорима, а различити сегменти (тј. блокови) се једноставно додељују процесима за просејавање. Што се тиче основног Ератостеновог сита, може се видети да је убрзање које постиже када се програм извршава за 2 и 4 процеса приближно, што значи да је потребно извршити одређена побољшања да би процеси били равномерније оптерећени, као и да је доста времена утрошено на комуникацију између процеса.

Ради поређења, на рачунару Intel Core2Duo, 2.67GHz, 2GB RAM меморије, потребно је 275,131 секунди за налажење и запис простих бројева из интервала $1.5 \cdot 10^{13}$ помоћу паралелизованог

[73] За више информација о овим Intel процесорима, видети `http://www.intel.com/content/www/us/en/processors/core/3rd-gen-core-family-mobile-brief.html`.

модификованог сита покренутог са два процеса (који раде на 2 језгра датог рачунара), уз употребу функција из MPI.

Интересантне су информације колико има простих бројева у одређеном интервалу и колико меморије на диску заузима запис информација о њиховој простости, што је битан фактор када просејавамо бројеве из великих интервала.

	erathosten.c	erathostenPractical.c	број процеса:	нађено простих бројева:
10^8	1.23	2.11	-np 1	5761455
	0.87	1.14	-np 2	
	0.76	0.76	-np 4	
10^9	13.59	19.70	-np 1	50847534
	8.56	10.89	-np 2	
	7.95	6.93	-np 4	
10^10	--	200.44	-np 1	455052511
	--	106.24	-np 2	
	100.00	70.57	-np 4	
10^11	--	2067.70	-np 1	4118054813
	--	1098.90	-np 2	
	--	740.77	-np 4	
10^12	--	22734.50	-np 1	37607912018
	--	14031.70	-np 2	
	--	8510.20	-np 4	

Време извршавања програма у секундама (налажење и бројање)

	erathosten.c	erathostenPractical.c
10^8	(око) 51,105,792 bytes	2,297,856 bytes
10^9	(око) 501,968,896 bytes	21,725,184 bytes
10^10	(око) 4,948,221,952 bytes	215,371,776 bytes
10^11	--	2,153,717,760 bytes
10^12	--	21,536,550,912 bytes

Потребна количина простора на диску

Иако је друго модификовано сито спорије за мање вредности броја n до кога се просејавање извршава, како се број n повећава, ово сито показује значајно боље карактеристике у односу на основну верзију, не узимајући у обзир да прва верзија коју смо дали у поглављу 5 не успева да алоцира довољно меморије за веће интервале. Модификовано сито erathostenPractical.c показује још боље карактеристике у односу на прво сито када се обавља и запис информација о простости бројева. Одговарајући графици и додатне информације дате су у прилогу који иде уз овај рад.

Листа простих бројева из неког интервала се може чувати и тако што се записују размаци (енгл. *gaps*) између тих простих бројева.

Овај програм се *лако* проширује да пронађе факторизацију на просте чиниоце свих целих бројева до n. Осим у криптографији, листа свих простих бројева из интервала $[2, n]$ може омогућити и проверу одређених хипотеза из теорије бројева.

7.5 Програмске реализације одабраних тестова простости

Наредни тестови простости су имплементирани у Јави и C-у. Како програми који су написани у програмском језику C раде са бројевима до 2^{64}, у овом поглављу се разматрају само реализације у Јави уз употребу библиотеке BigInteger.

Фермаов, Соловеј-Штрасенов и Милер-Рабинов тест су општи вероватносни тестови простости, односно могу се применити на све бројеве. Ови тестови су упоређени са временом извршавања функције isProbablePrime из класе BigInteger, која врши тестирање броја користећи два независна вероватносна теста, Милер-Рабинов и један од Лукас-Лемерових тестова, тако да је веома мала вероватноћа грешке када ова функција врати излаз да је неки број прост. Врше се тестирања бројева величине до 16,384 бита.

Узета је у обзир зависност величине задатог броја и броја итерација које су потребне да би са *довољном* сигурношћу Милер-Рабин тест вратио одговор да ли је неки број прост. За утврђивање колико је рунди/итерација потребно да се изврши тест употребљен је поступак описан у функцији primeToCertainty из BigInteger. Милер-Рабинов тест је сигурнији од Фермаовог и Соловеј-Штрасеновог теста простости ако се извршавају исти број пута (тј. за исти број различитих база проверавају простост броја), али овде смо усредсређени на време које је потребно да се ови тестови изврше исти број пута, а не да постигну исти степен сигурности.

Фермаов тест простости

Фермаов тест је једноставно реализован употребом функције modPow из BigInteger. Треба приметити да функција modPow користи и Кинеску теорему о остацима за брже израчунавање модуо операције раздвајајући паран и непаран део модула по коме се рачуна, и на тај начин рачуна по два мања модула, на крају спајајући резултате, што је брже него рачунање по модулу великог броја. Иако је веома једноставан, Фермаов тест представља основу и део је многих тестова простости. Такође је и веома *брз* вероватносни тест простости.

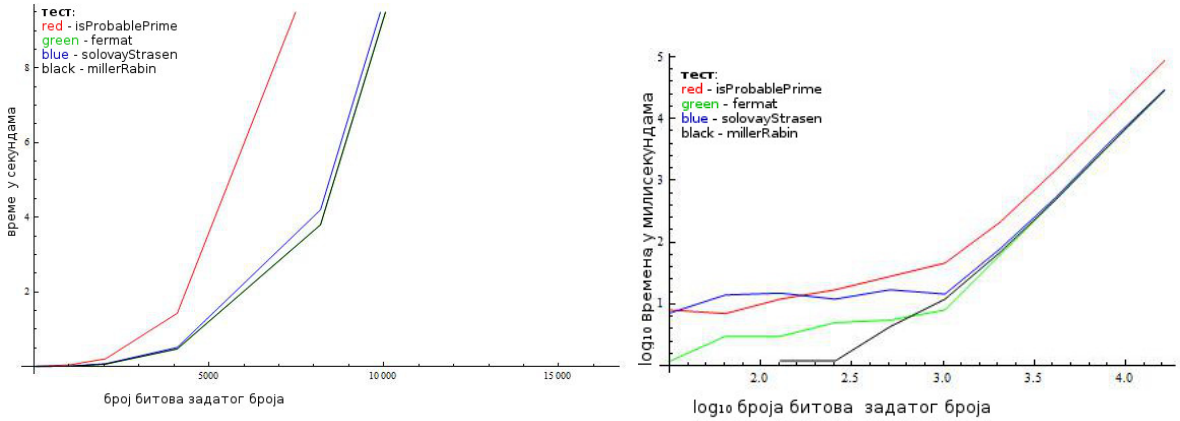
Соловеј-Штрасенов тест простости

За ефикасно извршавање Соловеј-Штрасеновог теста је битна добра имплементација израчунавања Јакобијевог симбола. Функција за израчунавање Јакобијевог симбола је имплементирана помоћу алгоритама датих у [17, 28], као и имплементације у библиотеци BigInteger, где је функција jacobi приватна функција са два аргумента, од којих је први аргумент типа int, а не BigInteger, јер је такав формат потребан за функцију isProbablePrime која користи функцију jacobi. Рад са (малим) целим бројевима је бржи од рада са типом података BigInteger, па је потребно имати ово у виду када се имплементирају функције. Поред израчунавања Јакобијевог симбола, Соловеј-Штрасенов тест такође користи функцију modPow, али са мањим експонентом у односу на Фермаов тест.

Милер-Рабинов тест простости

Како је Милер-Рабинов тест део функције isProbablePrime, адаптирана је функција passesMiller-Rabin из класе BigInteger. Највећи степен броја 2 који дели $n-1$, где је n задати број, је утврђен испитивањем најмањег постављеног бита у запису задатог броја као BigInteger типа помоћу јавне функције getLowestSetBit. Множење са два је реализовано као померање у десно, односно употребом јавне функције shiftRight из BigInteger. Такође се користи функција modPow.

Подаци који илуструју времена извршавања ових програма су дати у следећим графицима и табелама.



Графици који упоређују времена извршавања датих вероватносних тестова простости

Тестови за бројеви величине до 8,921 бита су проверавани за 10 различитих простих бројева, док је за бројеве од 8,921 бита узето 5 простих бројева, а за 16,384 бита су тестови проверени за 2 различите вредности. Сви тестови враћају тачан резултат за све бројеве, иако са различитом сигурношћу.

број битова	isProbablePrime	fermat	solovayStrassen	millerRabin
32	7.6	1.6	7.8	6.4
64	6.2	3.2	14.0	0.0
128	9.4	3.2	15.4	1.6
256	15.8	9.3	10.9	1.5
512	29.8	6.3	15.5	4.6
1024	47.0	6.2	15.5	14.0
2048	213.7	61.0	76.5	66.9
4096	1428.9	474.2	516.3	474.4
8192	11051.4	3731.2	3872.0	3734.6
16384	87152.5	28782.5	29538.5	28907.0

Просечно време извршавања вероватносних тестова у милисекундама

Занимљиве информације се могу добити када се ови тестови "пусте" да за исту базу, на пример за $a = 2$, $a = 3$, итд, у одређеном интервалу пронађу вероватно просте бројеве, и онда се упореди који бројеви су у којим тестовима за које базе псеудопрости.

n - 1 тест простости

Овај тест се разликује у односу на претходне јер захтева факторизацију броја F, таквог да је $n-1 = F \cdot R$, где је n задати број чија се прималност испитује. Факторисани део F мора бити довољно велики, иначе тест није у могућности да врати одговор да ли је или не број n прост. Приликом реализације овог теста морају се проверити улазне вредности, тј. да ли $F|n-1$ и да ли је низ бројева који је дат као факторизација броја F заиста низ простих бројева чији производ даје број F. Факторизација тј. делимична факторизација броја $n-1$ је једноставна за неке случајеве, док у неким случајевима може бити подједнако тешка као и факторизација броја n. Зато је *облик* броја n, тј. броја $n-1$ веома битан за овај тест. Пример када је факторизација броја $n-1$ позната су поред Фермаових бројева, и Еуклидови бројеви.

Један од начина имплементације овог теста може бити да програм покушава да пронађе довољну делимичну факторизацију тако што након сваког пронађеног фактора проверава да ли је тако добијени F довољан да $n-1$ тест да одговор да ли је или не број n прост. Треба приметити да је на овај начин проблем факторизације постао део теста прималности, што у општем случају није добро. Када буду објављени ефикасни алгоритми за факторизацију, и овакав приступ провере простости датог броја биће ефикаснији. Али, у овом случају, n - 1 тест није потребан јер је лако наћи факторизацију самог броја n, и на тај начин доказати његову сложеност, односно простост?

Факторизација броја $n-1$ се може добити, на пример, помоћу једноставне функције faktorizacija чија имплементација је дата у C-у, модификујући неко од датих сита или користећи просте бројеве нађене помоћу сита ради провере да ли $n-1$ има фактор у том интервалу, или употребом функције FactorInteger из програма Mathematica која даје потпуну или делимичну факторизацију датог броја.

На пример, да би се помоћу n - 1 теста проверила прималност броја 2,147,483,659 (који јесте прост), испоставља се да је потребна потпуна факторизација броја $n-1 = 2 \cdot 3 \cdot 149 \cdot 2402107$, јер познавање прва три фактора не даје довољно велики број F. Како број n - 1 у овом случају није неког специјалног облика, следи да тест n - 1 није практично примењивати на овакве бројеве. Функција FactorInteger за неколико милисекунди факторише овај број, који је дат само као пример броја који није специјалног облика. Претходно запажање се лако проверава ако се покуша делимична или потпуна факторизација нпр. бројева величине 512 или 1024 бита. Ако се израчуна факторизација неколико различитих бројева исте величине, тј. бројева за чији је запис потребан исти број бита, показује се да је за неке лако наћи делимичну факторизацију броја $n-1$ која омогућава да n - 1 тест да одговор да ли је n прост или не, док је за друге тешко наћи довољно велику делимичну факторизацију, па за такве бројеве није практично користити овај тест. У прилогу су дати примери бројева за које је Mathematica успешно дала довољну делимичну факторизацију, и време за које је извршена та факторизација и време извршавања теста, као и примере када то није успешно урађено у *разумном* времену. Нарочито су занимљиви примери за 512 битне и 1024 битне бројеве.

Са друге стране, ако постоји већ позната довољна делимична факторизација броја $n-1$ на просте бројеве, тј. ако се претпостави да делимична факторизација броја $n-1$ није део теста већ да је она позната, односно дата као улазне вредности за тест, као што је то претпостављено у [17, 28], овај тест може бити веома ефикасан.

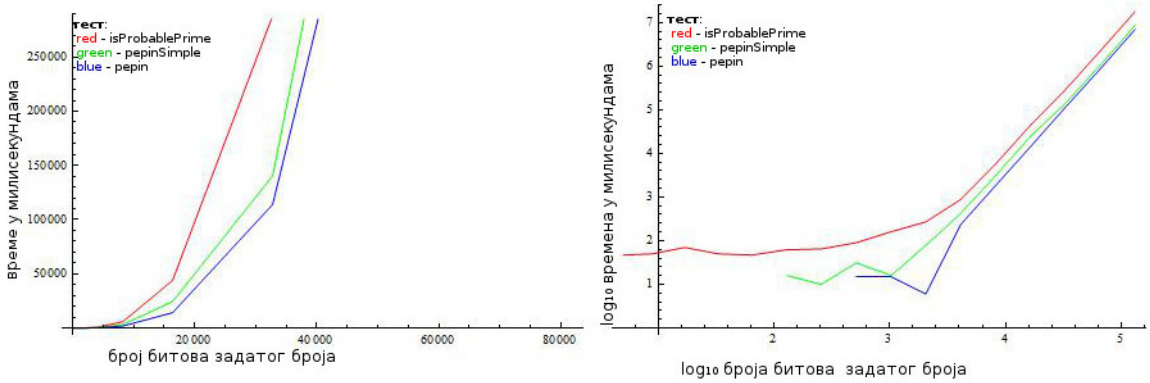
Ако се узму у обзир сва претходна разматрања, у општем случају постоји проблем у великој варијабилности резултата, што добар график времена извршавања овог теста чини *немогућим*.

У имплементацији овог теста у програмском језику Јава, прости фактори са улаза су стављени у тип ArrayList и онда је, када је потребно, вршена итерација над овим низом. Део $n-1$ теста је и Фермаов тест. Коришћене су функције из библиотеке BigInteger, између осталих и modPow, gcd, divide, subtract. Где год је било могуће употребљена је функција степеновања која као аргумент узима цео број. Провера да ли је $c_1^2 - 4 \cdot c_2$ квадрат је имплементирана једноставно, користећи чињеницу да тип BigInteger не чува разломљени део. Могуће су ефикасније реализације.

Пепинов тест простости

Пепинов тест је варијанта n - 1 теста за Фермаове бројеве. Уз Пепинов тест је дата и функција која за задати број n израчунава Фермаов број F_n. Једноставна имплементација Пепиновог теста користи квадрирање броја 3 и одмах врши израчунавање по модулу датог Фермаовог броја, понављајући поступак $2^n - 1$ пута. Ако се у овој for петљи квадрирање извршава помоћу функције modPow, примећује се да је тест *знатно* спорији. Са друге стране, ако се примени функција modPow на број 3 да се одмах израчуна крајњи резултат, са првим аргументом тј. експонентом $(F_n-1)/2$

и другим аргументом F_n који је модуо по коме се рачуна, ова *верзија* имплементације Пепиновог теста је ефикаснија. То је због начина на који је степеновање по модулу реализовано у BigInteger, где функција modPow користи приватне функције које ефикасно врше узастопно квадрирање и израчунавање по модулу степена броја два.



Графици који упоређују времена извршавања Пепиновог теста и функције isProbablePrime

Времена ових тестова су дата и у табели:

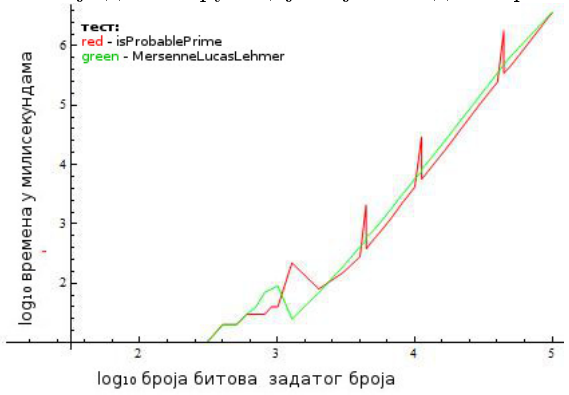
n, $F_n = 2^{2^n} + 1$	број битова	isProbablePrime	pepinSimple	pepin
8	257	65	10	0
9	513	90	31	15
10	1025	160	16	15
11	2049	270	78	60
12	4097	880	422	234
13	8193	5697	3088	1825
14	16385	44000	24477	14200
15	32769	288000	140369	114000
16	65537	2236045	1063157	869014
17	131073	17863437 (5h)	8560344(2h 23min)	6964389 (2h)

Просечно време извршавања наведених тестова простости у милисекундама

Може се приметити да ова реализација Пепиновог теста омогућава да се на персоналном рачунару или лаптопу за мање од једне секунде провери прималност првих 12 Фермаових бројева.

Лукас-Лемеров тест простости за Мерсенове бројеве

Овај тест је прави тест за доказивање простости који представља специјалан случај $n+1$ теста, где је позната потпуна факторизација броја $n+1$. Лукас-Лемеров тест је веома ефикасан за проверу простости Мерсенових бројева, и употребљен је за доказивање прималности неких од највећих познатих простих бројева. Реализација самог теста је, као и Пепиновог теста, веома једноставна. Уз тест је дата и функција која за задати прост број n израчунава Мерсенов број M_n.



n	isProbablePrime	MersenneLucasLehmer
1279	250	20
4423	2,000	500
11213	28,950	7,600
44497	1,800,360	470,200

Времена у милисекундама за просте Мерсенове бројеве

Мерсенови бројеви: Лукас-Лемер наспрам isProbablePrime

Ова имплементација Лукас-Лемеровог теста за Мерсенове бројеве омогућава да се за мање од

једног минута провере Мерсенови бројеви $M_n = 2^n - 1$ до $n = 23000$, док је за $n = 100,003$ потребно око 1 сат.

На основу графика и табеле времена извршавања програма за неке од познатих Мерсенових простих бројева, изводи се закључак да је Лукас-Лемеров тест за Мерсенове бројеве ефикаснији од функције isProbablePrime када су у питању прости Мерсенови бројеви, и да смо овим тестом доказали прималност датог простог броја.

За опште Мерсенове бројеве, ова имплементација Лукас-Лемеровог теста је спорија од isProbablePrime, осим када наиђе на прост Мерсенов број. Тако се намеће једноставан начин проналажења простих Мерсенових бројева: прво се *пусти* неки брзи вероватносни тест да за неколико малих база, које се бирају на основу разматрања у поглављу 2, провери прималност Мерсенових бројева M_n за n прост број, па тамо где M_n прође тест, онда се за те вредности примењује оптимизован Лукас-Лемеров тест за Мерсенове бројеве. Оптимизације се односе на ефикасне алгоритме за квадрирање и операције тражења остатка по модулу. За више информација, видети [17, 19].

AKS тест простости

Тест AKS је нешто теже реализовати у односу на претходно изложене тестове. У овом тесту се извршавају операције са полиномима, па је од суштинског значаја ефикасно израчунавање полинома $(x + a)^n$, односно ефикасније множење полинома, и рачунање по модулу полинома $x^r - 1$ и по модулу n. Реализација алгоритма AKS дата је једноставно, пратећи кораке алгоритма датог у поглављу 4.

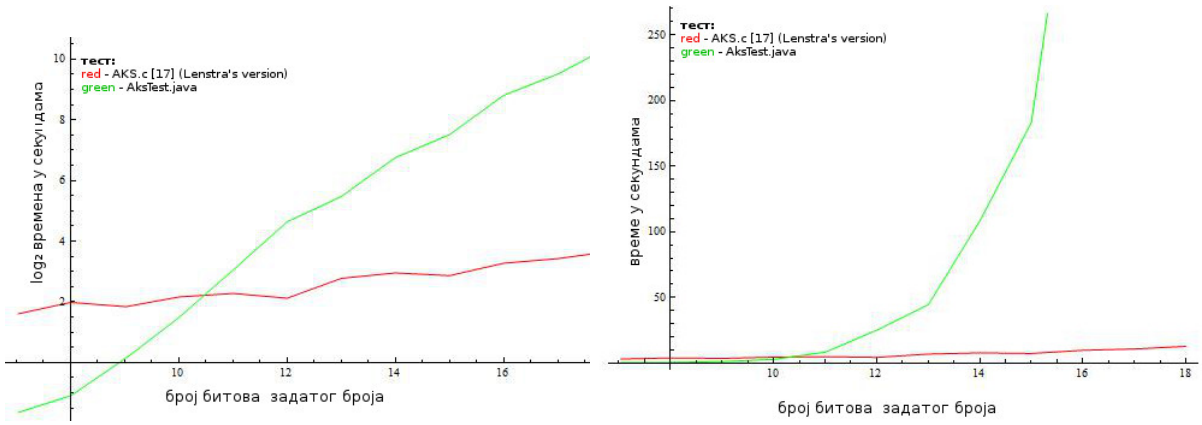
Функција која налази k-ти корен реализована је на основу алгоритма датог у [17]. Функција log2 која враћа логаритам за основу 2 за бројеве типа BigInteger реализована је на основу јавне функције bitlength ове библиотеке, класе BigDecimal и основних идентитета који важе за логаритме. Реализована је функција која дати полином множи полиномом $x+a$, али се она због неефикасности оваквог приступа решавања проблема множења полинома не користи у завршној верзији реализације AKS теста.

Множење и квадрирање полинома реализовани су на основу Алгоритма 9.6.1 из [17], који омогућава брзо множење полинома путем бинарне сегментације (енгл. *fast polynomial multiplication via binary segmentation*). Реализација функције која обавља редукцију по модулу $x^r - 1$ и n урађена је на основу једноставних идентитета који важе за израчунавања по овим модулима. Ове фунције су онда употребљене у модификованом алгоритму бинарног степеновања датог у поглављу 3, како би биле проверене неопходне биномне конгруенције.

За реализацију у Јави употребљен је тип података ArrayList за чување коефицијената одређеног полинома (могу се користити и HashMap или LinkedHashMap). Како су коефицијенти датих полинома типа BigInteger, коришћене су многе од претходно наведених функција ове библиотеке.

Занимљиво је поменути да ако се уместо корака [биномна конгруенција] прво систематски примењује Фермаов тест за n, за базе a до дате границе, па онда провери да ли су сви одговарајући биномни коефицијенти тј. коефицијенти полинома $(x+1)^n$ по модулу $x^r - 1$ и по модулу n једнаки нули, а први и последњи коефицијент једнаки 1, и да ли је степен полинома одговарајући, добија се прилично сигуран, и бржи тест у односу на алгоритам AKS (али и даље знатно спорији у односу на изложене вероватносне тестове). Овај тест простости врло брзо елиминише сложене бројеве, једино је питање са коликом вероватноћом враћа да је дати број n прост. И овде је, као и код алгоритма AKS, највише времена утрошено на израчунавања са полиномима. Ако се код биномних коефицијената искористи својство симетрије, израчунавање је нешто ефикасније.

Следи график времена извршавања основне верзије теста AKS реализоване у програмском језику Јава, и реализације Ленстрине верзије теста AKS у програмском језику C коју су имплементирали Крандал и Померанс, а која је овде изложена ради поређења перформанси програма.



Времена извршавања основне верзије теста AKS и верзије из [17]

Анализом графика се може закључити да је потребно употребити ефикасније алгоритме како би се достигло време реализације из [17].

7.6 Закључак

У овом раду приказани су неки од тестова да ли је задати број прост и описане су њихове једноставне реализације. Даљи рад подразумева модификације и оптимизације датих програма, употребом ефикаснијих алгоритама и бољих техника програмирања, како би сви расположиви ресурси на најбољи начин били искоришћени.

Поред употребе већ постојећих библиотека за рад са великим бројевима, ако је потребно да имплементација тестова простости буде ефикаснија, често се користи програмски језик C и креира сопствена библиотека за рад са великим бројевима.

У овом раду су наведене многобројне референце на анализе и реализације различитих алгоритама. Нажалост, постоје примери када разматрани алгоритми нису коректно описани или имплементирани. Према свим изворима информација, дакле и према овом раду, мора постојати критички однос, а имплементација одређеног теста простости се не сме заснивати само на једном извору [12].

Као примери употребе тестова простости дати су криптосистеми са јавним кључем. У одређеним криптографским алгоритмима са тајним кључем као што је то алгоритам AES, или у неким генераторима псеудослучајних низова битова (енгл. *pseudorandom bit generator*) који се могу користити за генерисање низа кључа код симетричних криптосистема, ради се у коначним прстенима и пољима полинома или бројева, а самим тим имплицитно и са простим бројевима. Теорија изложена у овом раду важна је за разумевање ових алгоритама.

Сведоци смо сталног усавршавања многих алгебарских алгоритама и тестова да ли је задати број прост. Теоријска истраживања су веома битна како би одређени вероватносни тестови простости постали сигурнији, а прави тестови доказали простост провером мањег броја вредности, или једноставније и брже одговорили на питање да ли је задати број прост.

Многи класични алгоритми могу се модификовати да раде са полиномима или великим бројевима. Како перформансе рачунара стално напредују, а примена паралелног програмирања постаје распрострањенија и лакша, чак и мање ефикасни алгоритми могу постићи жељене циљеве. Као што је речено, постоје многи пројекти чији је циљ проналажење великих простих бројева, и може се приметити да се нови рекорди непрекидно проналазе.

Због свега наведеног, проблеми за које се верује да су тешки временом постају једноставнији за решавање, или практично једноставнији за решавање.

Литература

[1] Jeff Wehrwein, *Primality testing*, Senior Thesis in Computer Science, Middlebury College, 2008.

[2] *C Source Code for a Sieve Program*, online: http://www.rsok.com/~jrm/source/sieve2310.c.html. Напомена: Овај линк може послужити као пример тзв. *spoof website* — лажног сајта или веб сајта направљеног као пародија на презентације са научним или *e-commerce* садржајем.

[3] Alan Baker, *A concise intoruction to the theory of numbers*, Cambrige University Press, 1984.

[4] Александар Ивић, *Увод у аналитичку теорију бројева*, Издавачка књижарница Зорана Стојановића, Сремски Карловци, Нови Сад, 1996.

[5] Bruce Schneier, *Примењена криптографија, протоколи, алгоритми и изворни кôд на језику C*, Микро књига, Београд, 2007.

[6] Manindra Agrawal, Neeraj Kayal, Nitin Saxena, *PRIMES is in P*, *Annals of Mathematics*, Vol. 160, No. 2, 781–793, September 2004.

[7] Robert Rumely, *Recent Advances in Primality Testing*, *Notices of the AMS*, 475–477, 1983.

[8] Гојко В. Калајџић, *Алгебра*, Веста – Математички факултет, Београд, 1998.

[9] Александар Липковски, *Линеарна алгебра и аналитичка геометрија*, Научна књига, Београд, 1992.

[10] Миодраг Живковић, *Алгоритми*, Математички факултет, Београд, 2000.

[11] Миленко Р. Мосуровић, *Парелелизација алгоритама за множење великих бројева и њихова примена на испитивање прималности Фермаових бројева*, Магистарски рад, Београд, 1996.

[12] Жарко Мијајловић, *Асиметрични криптографски поступци/системи*, online: http://poincare.matf.bg.ac.rs/nastavno/zmijaj.html.

[13] R. Rumely, L. Adleman, C. Pomerance, *On distinguishing prime numbers from composites*, *Annals of Mathematics 117*, 173–206, 1983.

[14] Brian W. Kernighan, Dennis M. Ritchie, *C Programming Language*, 2nd ed., Prentice Hall Software Series, 1988.

[15] Bruce Eckel, *Thinking in Java*, 4th ed., Prentice Hall, 2006.

[16] Предраг Јаничић, *Математичка логика у рачунарству*, Математички факултет, Београд, 2009.

[17] R. Crandall, C. Pomerance, *Prime numbers. A computational perspective*, 2nd ed., Springer, 2005.

[18] *The Prime Pages, prime number research, records, and resources*, online: http://primes.utm.edu.

[19] *GIMPS, Great Internet Mersenne Prime Search*, online: http://www.mersenne.org.

[20] *The Mathematical Atlas*, online: http://www.math.niu.edu/~rusin/known-math.

[21] *RSA. Security, compliance, and risk-management solutions, Frequently Asked Questions about Today's Cryptography*, online: http://www.emc.com/domains/rsa/, http://www.rsa.com/rsalabs/faq/files/rsalabs_faq41.pdf.

[22] *The International PGP Home Page*, online: http://www.pgpi.org.

[23] Michael J. Quinn, *Parallel programming in C with MPI and OpenMP*, McGraw-Hill, 2004.

[24] Ненад Митић, *Основи рачунарских система*, Математички факултет, 2002.

[25] В. Мићић, З. Каделбург, *Материјали за младе математичаре, св. 15*, Друштво математичара СР Србије, 1986.

[26] *Wikipedia, the free encyclopedia that anyone can edit*, online: http://en.wikipedia.org/, http://sr.wikipedia.org.

[27] Neal Koblitz, *A Course in Number Theory and Cryptography*, 2nd ed., Springer-Verlag, New York, 1994.

[28] A. Menezes, P. van Oorschot, S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Boca Raton, 1997.

[29] *A free resource from Wolfram Research built with Mathematica technology*, online: http://mathworld.wolfram.com/.

[30] Ueli M. Maurer, *Fast Generation of Prime Numbers and Secure Public-Key Cryptographic Parameters*, *Journal of Cryptology*, Vol. 8, Issue 3, 123–155, 1995.

[31] Миодраг Живковић, *Криптографија*, online: http://poincare.matf.bg.ac.rs/~ezivkovm/nastava/kripto.pdf.

[32] Hans Riesel, *Prime Numbers and Computer Methods for Factorization*, 2nd ed., Birkhäuser, 1994.

[33] Robert G. Salembier, Paul Southerington, *An Implementation of the AKS Primality Test*, ECE 746 Project Report, Spring 2005.