Универзитет у Београду

Математички факултет, Београд

МАСТЕР РАД ИЗ РАЧУНАРСТВА И ИНФОРМАТИКЕ

Алгоритам AKS за испитивање да ли је задати број прост

Аутор:

Миша Алимпић 1166/2011

Ментор:

др Миодраг Живковић

Београд, септембар 2012. године

Садржај

1 Увод

Прости бројеви чине окосницу данашњих рачунарских техника сигурности. Они се највише користе у асиметричним шифарским алгоритмима чији је најпознатији представник RSA. У већини асиметричних алгоритама за генерисање кључева се користе велики прости бројеви са стотину и више декадних цифара. Предмет овог рада су тестови за испитивање да ли је дати број прост. Поред неколико пробабилистичких алгоритама за испитивање да ли је задати број прост, у раду се излаже детерминистички алгоритам AKS и његова програмска реализација.

2 Прости бројеви

„Математичари су до данашњих дана узалудно покушавали да открију неки ред у низу простих бројева и имамо разлога да верујемо да је то тајна у коју ум никада неће продрети.“- Ојлер, (Хавил 2003, стр. 163).

„Због тога што и приватност система и сигурност дигиталног новца зависи од шифровања, напредак било математике или информатике који би оповргнуо криптографски модел, био би права пропаст. Очигледан математички пробој био би напредак неког лаког начина факторисања великих простих бројева.“- Бил Гејтс, (Хавил 2003, стр. 171).

2.1 Особине простих бројева

Прост број је позитиван цео број $p > 1$ који нема позитивне целе делитеље осим 1 и самог p. Позитивни цели бројеви, осим броја 1[1], који нису прости се зову сложеним бројевима. Скуп свих простих бројева обично обележавамо са $\mathbb{P}$.

Одувек је међу математичарима постојала жеља за проналажењем „формуле за добијање простих бројева“, чија би сва решења припадала скупу $\mathbb{P}$, али она још увек није откривена. За разлику од ње постоје формуле које обично као резултат враћају прост број. Можемо их поделити на полиномне, експоненцијалне и приморијалне формуле.

Логично прва идеја би била наћи полиномну формулу која као резултат увек враћа прост број. Али за полиномну формулу са једном променљивом и целобројним коефи-

[1] Број 1 је посебан случај за који се узима да није ни прост, ни сложен. Разлог због кога се 1 не узима као прост број је зато што онда не би важила јединствена факторизација броја у *Основној теореми аритметике* [4]

цијентима је и доказано да она не постоји. Теорема која то потврђује каже да за сваки полином $f(x)$ са целобројним коефицијентима, постоји бесконачно много позитивних целих бројева m за које је $f(m)$ сложен број [2]. Насупрот њима, постоје полиноми више променљивих чији су скупови позитивних вредности прости бројеви, али је њихово коришћење за налажење простих бројева веома непрактично.

Постоје два најкоришћенија облика експоненцијалне формуле који дају две значајне класе простих бројева:

1. $M(n) = 2^n - 1$ - *Мерсенови бројеви* и
2. $F(n) = 2^{2^n} + 1$ - *Фермаови бројеви*,

где је $n > 0$. Експонент n мора бити прост број да би и $M(n)$ био Мерсенов прост број. Заиста, из претпоставке да је $n = rs$, где је $1 < r \leq s < n$, следи

$$M(n) = 2^n - 1 = 2^{rs} - 1 = (2^r - 1)(2^{r(s-1)} + 2^{r(s-2)} + \cdots + 2^r + 1).$$

Значи, ако $r \mid n$, онда и $M(r) \mid M(n)$. Првих неколико простих бројева n за које је $M(n)$ прост су: 2, 3, 5, 7, 13, 17, 19, 31, ... Интересантна чињеница је да је највећи до сада откривен прост број са 12978189 декадних цифара $M(47)$[2] Мерсенов број и облика је: $2^{43112609} - 1$ [10]. *Лукас - Лемеров тест* [2] се користи за испитивање да ли је дати Мерсенов број прост и на тај начин су и откривени највећи прости бројеви. Што се тиче Фермаових бројева, сам Ферма је открио да ако је $2^m + 1$ прост, онда m мора бити степен двојке. До данас је откривено свега пет простих Фермаових бројева[3] [11].

Приморијална функција за задати број p дефинише се као производ свих простих бројева мањих или једнаких са p и обележавамо је са $p^\sharp$. Она има облик:

$$f(p) = p^\sharp + 1.$$

Тврђење 2.1. *Ако $p^\sharp + 1$ није прост број, најмањи фактор добијеног броја је увек већи од p.*

Доказ. Тврђење се може доказати контрадикцијом. Претпоставка је да $p^\sharp + 1$ има прост фактор $q \leq p$. Пошто $p^\sharp$ представља производ свих простих бројева до p, следи да и q такође дели $p^\sharp$. Онда би q делило и $(p^\sharp + 1) - p^\sharp = 1$. То је могуће само за $q = 1$. Али то је у супротности са почетном претпоставком да је q прост. Тиме је тврђење доказано. □

[2] Претпоставља се, али још увек није доказано да је то сигурно 47 Мерсенов број

[3] То су бројеви $F(0), \ldots, F(4)$, док је факторизација извршена за $F(5), \ldots, F(11)$

Претходно тврђење сугерише наредни поступак за налажење великих простих бројева на следећи начин: претпоставимо да знамо све просте бројеве мање или једнаке од p. Одредимо сада вредност приморијалне функције за p тј. израчунајмо $p^\sharp + 1$. Ако је добијени број прост потрага се завршава. У супротном, прост број се добија тако што се одреди најмањи фактор од $p^\sharp + 1$ (за који је доказано у Тврђењу 2.1 да је већи од p). У оба случаја нађен је нови прост број већи од p. Главни разлог због кога се овај алгоритам не користи је тај што је неопходно извршити факторизацију броја $p^\sharp + 1$, а то је веома сложен процес.

Следи теорема која илуструје једну битну особину простих бројева.

Теорема 2.1. *Постоји бесконачно много простих бројева.*

Доказ. Претпоставимо супротно да има коначно много простих бројева. Онда постоји и највећи прост број међу њима, нека то буде m. То повлачи са собом да су сви бројеви већи од m сложени. Али на основу Тврђења 2.1 број $m^\sharp + 1$ не може да има просте факторе мање или једнаке са m. Одатле следи да $m^\sharp + 1$ нема уопште простих фактора. Али то би онда противречило Основној теореми аритметике. Дошло се до контрадикције тј. доказано је да постоји бесконачно много простих бројева. □

2.2 Псеудопрости бројеви

Сада ће бити речи о специјалној класи сложених бројева тзв. *псеудопростим бројевима*. Њихово дефинисање је неопходно за разумевање пробабилистичких тестова. Они представљају сложене бројеве који пролазе тест или секвенце тестова за испитивање да ли је задати број прост, а који иначе падају за већину сложених бројева.

Дефинишимо најпре *малу Фермаову теорему* која је неопходна за разумевање одређених класа псеудопростих бројева.

Теорема 2.2 (мала Фермаова). *Нека је p прост број и $a \in \mathbb{Z}$ такав да $p \nmid a$. Тада је*

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

Доказ. Из дефиниције теореме имамо да $p \nmid a$. Докажимо најпре да скуп целих бројева: $0a, 1a, \ldots, (p-1)a$ представља комплетан скуп остатака по модулу p. Претпоставимо супротно. У том случају би два елемента, рецимо ia и ja, били у истом скупу остатака по модулу p тј. важило би да је $ia \equiv ja \pmod{p}$. То би онда повлачило да $p \mid (i-j)a$, али

како a није дељиво са p према почетној претпоставци, имали бисмо да $p \mid (i-j)$. Пошто су и i и j мањи од p, једини случај када је то могуће је да је $i = j$. На основу претходног закључка следи да скуп бројева: $0a, 1a, \ldots, (p-1)a$ уствари представља пермутацију скупа елемената: $1, 2, \ldots, (p-1)$, ако ове скупове посматрмо у простору остатака по модулу p. Одатле следи да је производ елемената првог скупа конгурентан производу елемената другог скупа по модулу p тј. $a^{p-1}(p-1)! \equiv (p-1)! \pmod{p}$. Одатле, $p \mid (p-1)!(a^{p-1}-1)$. Пошто $(p-1)!$ сигурно не дели p, имамо да $p \mid (a^{p-1}-1)$, што је и требало доказати. □

Ако користимо појам псеудопростих бројева без било каквих додатних спецификација, онда обично мислимо на*Фермаове апсеудопросте бројеве*.*Кармајклови бројеви* су непарни сложени Фермаови псеудопрости бројеви за сваку од база. Често се називају и апсолутним псеудопростим бројевима.

Дефиниција 2.1. *Сложен цео број n називамо Фермаовим апсеудопростим бројем, ако за цео број $a > 1$ узајамно прост са њим, важи да: $n \mid a^{n-1} - 1$.*

Другим речима, то су бројеви који задовољавају малу Фермаову теорему за одређену базу a.

Дефиниција 2.2. *Сложен цео број n називамо Кармајкловим бројем, ако $n \mid a^{n-1} - 1$ за свако $a \in \mathbb{Z}$.*

Закључујемо да ако p није прост, могуће је да једначина (1) ипак важи. Ако (1) важи за појединачну вредност за a онда говоримо о Фермаовим a псеудопростим бројевима, а ако је (1) задовољено за свако a онда су у питању Кармајклови бројеви.

Позната класа псеудопростих бројева су *јаки псеудопрости бројеви*.

Дефиниција 2.3. *Јак псеудопрост број за базу a је непаран сложен број $n = d \cdot 2^s + 1$, где је d непарно и за који важи једна од следећих конгруенција:*

$$a^d \equiv 1 \pmod{n} \tag{2}$$

$$a^{d \cdot 2^r} \equiv -1 \pmod{n} \tag{3}$$

за неко $r = 0, 1, \ldots, s-1$.

Јак псеудопрост број за базу a је увек Фермаов a псеудопрост број, али не важи и супротно. Кармајклови бројеви могу бити јаки псеудопрости бројеви за неке вредности

база. Сложен број n је јак псеудопрост број за највише $1/4$ свих база мањих од n [4]. Ова чињеница се користи код *Милер-Рабиновог теста*.

Осим набројаних, постоје и друге класе псеудопростих бројева: *Ојлерови, Лукасови, Фибоначијеви, ...* Њихова карактеристика је да „опонашају“просте бројеве за различите тестове.

2.3 Примене простих бројева у RSA алгоритму

Систем RSA је један од најважнијих подстицаја за интерес који се придаје проналажењу великих простих бројева. Систем RSA је најпознатији и најкоришћенији асиметрични шифарски систем. Уобичајна претпоставка је да потенцијални нападач зна алгоритам шифровања; међутим не зна се полиномијални алгоритам за одређивање кључа за дешифровање полазећи од кључа за шифровање. Најважније примене алгоритма RSA су за усаглашавање кључа за симетрични шифарски систем и дигиталног потписа.

Систем RSA је добио име по тројици својих проналазача Ривесту (Rivest), Шамиру (Shamir) и Адлиману $(\mathrm{Adleman})$. За почетак је неопходно одабрати два различита велика проста броја (са преко 100 декадних цифара) p и q и израчунати њихов производ $n = pq$. Бројеви p и q морају се чувати у тајности, али не и број n, који је део јавног кључа. n мора бити довољно велико да се одговарајући тајни кључ не би могао одредити потпуном претрагом. За комерцијалну употребу се обично користе бројеви n од 1024 бита, односно $n \approx 10^{308}$. За важне потребе се обично користи 2048-битно n, односно $n \approx 10^{617}$ [6]. До пре неку годину је било уобичајно да се користи 768-битно n ($n \approx 10^{232}$), али је оно успешно факторизовано 12.12.2009. године од стране тима научника које је предводио Клеињун $(\mathrm{T.Kleinjung})$ [9]. RSA-768 је тренутно највећи факторисан број, али се са развојем рачунара успешно факторишу нови, што утиче на промену границе за величину кључа.

Најпре се порука која се жели шифровати подели у блокове величине мање од n и сваки од тих блокова се посебно шифрује. Након израчунавања броја n, одреди се *Ојлерова ϕ функција*[4] тог броја. С обзиром да n представља производ два проста броја, лако се показује да је

$$\phi(n) = (p-1)(q-1).$$

Затим се изабере број e такав да је $(e, \phi(n)) = 1$[5] и израчуна његов мултипликативни инверз d у $\mathbb{Z}/\phi(n)\mathbb{Z} : d \equiv e^{-1} \pmod{\phi(n)}$. Када су познати e и $\phi(n)$ лако

[4] Ојлерова ϕ функција враћа број позитивних целих бројева мањих од n који су са њим узајамно прости

[5] У раду (a, b) представља скраћеницу за нзд(a, b)

се *проширеним Еуклидовим алгоритмом* израчунава d [6]. Приметимо да је $ed \equiv 1 \pmod{\phi(n)}$ и $1 < e, d < \phi(n)$. Пар (n, e) представља јавни кључ криптосистема RSA који се имплементира, док се у тајности чува d. Да би се алгоритам RSA могао провалити, неопходно је знати тајни кључ, односно одредити d на основу e и n. Може се показати да је комплексност проблема проналажења броја d еквивалентана са самом факторизацијом броја n .

Нека је $0 < a < n$ блок поруке која се шифрује. Сваки од блокова се посебно шифрује, тако да шифрована порука представља низ шифрованих блокова. Са b означавамо шифрат отвореног текста a. Једначина за његово израчунавање је:

$$b \equiv a^e \pmod{n}.$$

Ако нам је познат тајни кључ, декрипција шифрата b се извршава на следећи начин:

$$a \equiv b^d \pmod{n}.$$

Следи доказ да ово заиста функционише тј. да се декрипцијом шифрата b стварно добија почетна порука a. Имамо да је:

$$b^d \equiv (a^e)^d \equiv a^{ed} \equiv a^1 = a \pmod{n}.$$

Докажимо последњи корак у низу еквиваленција тј. да је $a^{ed} \equiv a^1 \pmod{n}$. Пошто важи да је $ed \equiv 1 \pmod{\phi(n)}$, значи да постоји број k такав да важи: $ed = 1 + k\phi(n)$. Како су e и d бројеви већи од 1 и $\phi(n) > 0$, следи да је и $k > 0$. Мењајући ed са $1 + k\phi(n)$ имамо:

$$a^{ed} \equiv a^{1+k\phi(n)} \equiv (a^{\phi(n)})^k a \pmod{n}$$

Сада на основу *Ојлерове теореме* која гласи:

$$a^{\phi(n)} \equiv 1 \pmod{n} \text{ ако је } gcd(a, n) = 1,$$

доказан је тражени корак. Да би Ојлерова теорема важила бројеви a и n треба да буду узајамно прости тј. треба водити рачуна да дужина блокова који се шифрују буде узајамно проста са n. Ипак то није неопходно, што се доказује на основу *Корселтове теореме* [2], тако да је једини услов за њихову дужину да буду мањи од n.

За крај износимо препоруке о којима треба водити рачуна приликом избора великих простих бројева p и q:

1. p и q се обично бирају тако да $(p - 1, q - 1)$ буде мали број.

2. Бројеви $p - 1, q - 1, p + 1$ и $q + 1$ треба да имају велики прост чинилац.

3. Бројеви p и q не треба да буду превише близу један другом; с друге стране, однос већег и мањег од њих је обично мањи од 4. Постоје специјални алгоритми за факторизацију који могу да се искористе ако није испоштована било која од ове три препоруке.

4. Обично је e релативно мало, да би се смањило време шифровања. Често се узима $e = 3$.

3 Тестови за проверу да ли је задати број прост

У овом поглављу разматрају се тестови, односно алгоритми за проверу да ли је задати број прост. За разлику од факторизације, поменути тестови генерално не враћају просте факторе, већ само одговарају на питање да ли је унети број прост. Проблем факторизације броја је значајно комплекснији од провере да ли је прост користећи се поменутим тестовима. Већина тестова показују да је број прост, али постоје и они као што је Милер-Рабинов (видети тачку 3.1.3), који показују да је унети број сложен, тако да њих можемо звати и тестовима сложености.

Тестове за проверу да ли је задати број прост делимо на две велике групе:

1. *Пробабилистички тестови*

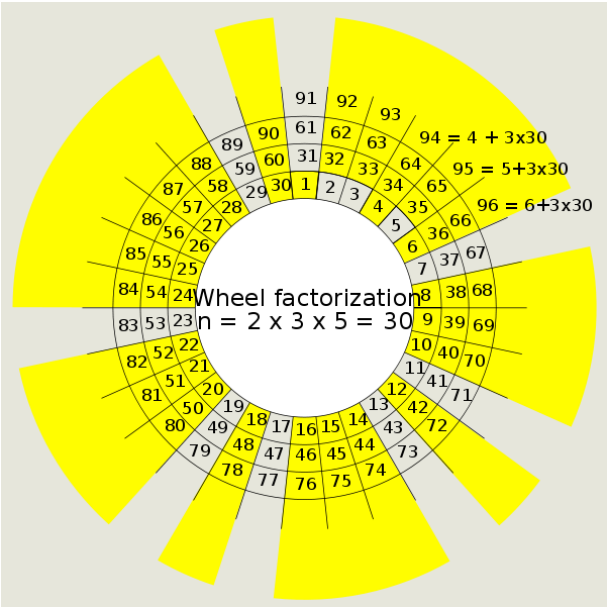
2. *Детерминистички тестови*.

Детерминистички тестови са потпуном сигурношћу утврђују да ли је тражени број прост. За разлику од њих, пробабилистички тестови могу потенцијално (мада са веома малом вероватноћом) погрешно идентификовати сложени број као прост. Пробабилистички тестови су знатно бржи, док је значај детерминистичких тестова у њиховој сигурности. Зато се у пракси најчешће користи њихова комбинација. Најпре се примени неки од пробабилистичких тестова и ако задати број прође дати тест означи се као вероватно прост број. Затим помоћу неког од детерминистичких тестова се може са сигурношћу утврдити да ли је дати број прост тј. одбацити псеудопрости бројеви који могу да прођу пробабилистичке тестове, а иначе су сложени.

Генерисање великих простих бројева има примену у криптографији. Генерални метод за њихово генерисање је да се изабере непаран број n одговарајуће дужине, а затим на сцену ступају поменути тестови, којима се утврђује да ли је изабран прост број или се мора покушати са генерисањем новог.

Постоји велики број тестова, зато кренимо од најлакших. Сигурно најједноставнији начин за утврђивање да ли је унети број n прост, је ако се провери да ли било који од

целих бројева из интервала од 2 до $n-1$ дели број n. Ако постоји такав број, n је сложен и уједно је пронађен и један од фактора. У супротном, ако такав број не постоји, n је прост. Прва модификација поменутог алгоритма подразумева да није неопходно тестирати све бројеве до $n-1$, већ да горња граница интервала испитивања треба да буде $\sqrt{n}$, јер ако је n сложен он се може представити као производ барем два броја од којих један мора бити мањи или једнак са $\sqrt{n}$. Следеће унапређење алгоритма подразумева прескакање свих парних бројева, осим 2, јер ако паран број дели n, онда га дели и 2. Даље се може видети да су сви прости бројеви облика $6k \pm 1$, осим 2 и 3 који представљају једине изузетке. То је због тога што се сви цели бројеви могу представити у облику $6k+i$, за целе бројеве k и $i = -1, 0, 1, 2, 3, 4$. 2 дели бројеве облика $6k, 6k+2$ и $6k+4$, док 3 дели бројеве облика $6k+3$. На описан начин довољно је тестирати бројеве 2 и 3, и све бројеве облика $6k \pm 1 \leq \sqrt{n}$, што је око 3 пута брже од почетног поступка. Генерализујући даље, може се приметити да су сви прости бројеви облика $c^\sharp k + i$, где су c и k цели бројеви, а за i важи да је узајамно просто са $c^\sharp$ и $i < c^\sharp$. Приметимо да ако i и c нису узајамно прости, онда је $c^\sharp k + i$ дељив са (c, i). Што је c веће, то се број корака за испитивање сложености броја n смањује. За овај метод неопходно је такође испитати и дељивост са свим простим бројевима мањим или једнаким са c. Овај поступак је приказан у следећем примеру.



Слика 1. Графички приказ методе коришћене у примеру 3.1.

Пример 3.1. *Нека је* $c = 6$. *Онда је* $c^\sharp = 2 \cdot 3 \cdot 5 = 30$. *Сваки цео број је облика* $30k + i$, *где је* $i = 0, 1, \ldots, 29$ *и* $k \in \mathbb{Z}$. *Али имамо да 2 дели* $0, 2, \ldots, 28$ *и 3 дели*

$0, 3, \ldots, 27$ *и* 5 *дели* $0, 5, \ldots, 25$. *Из овога се може закључити да су сви прости бројеви облика* $30k + i$, *где је* $i = 1, 7, 11, 13, 17, 19, 23, 29$ *(односно за оне* $i < 30$ *такве да је* $(i, 30) = 1$*) и* $k \in \mathbb{Z}$. *Ако не би важило да је* $(i, 30) = 1$, *онда би* $30k + i$ *било дељиво са неким простим фактором од* 30, *односно* $30k + i$ *не би био прост број. Дати пример илустрован је на слици 1. Прости бројеви се не могу наћи у жутим областима.*

Следи опис једноставног, прастарог алгоритма за проналажење простих бројева до задате горње границе -*методи Ератостеновог сита* (слика 2). Овај алгоритам одређује све просте бројеве мање или једнаке задатом броју n, тако да ће се у том скупу налазити и сам број n, ако је прост. Поступак се састоји у следећем: најпре се испишу сви бројеви мањи или једнаки са n, почевши од 2. Сада се почиње са „сејањем“ бројева кроз сито. Узме се први непрецртан број (у првом кораку је то 2, пошто на почетку нема прецртаних бројева) и затим се прецртавају сви бројеви који су дељиви са њим. Поступак се понавља све док следећи изабрани непрецртани број са којим се почиње следећа итерација, не буде већи од $\sqrt{n}$. Ако је у току поменутог поступка у неком кораку број n прецртан, следи да је он сложен, у супротном је прост. Добар начин да се убрзају алгоритми за проверу да ли је задати број прост је да се на почетку одреде и сачувају сви прости бројеви до неке границе, рецимо 200. Таква листа се може оформити Ератостеновим ситом. Затим, пре извршавања неког конкретног алгоритма се испита дељивост задатог броја са бројевима из поменутог скупа и ако се установи да је дељив са неким од тих бројева, даљи рад се прекида и констатује да је број сложен.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40	41	42	43	44	45
46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70	71	72	73	74	75
76	77	78	79	80	81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100	101	102	103	104	105
106	107	108	109	110	111	112	113	114	115	116	117	118	119	120
121	122	123	124	125	126	127	128	129	130	131	132	133	134	135
136	137	138	139	140	141	142	143	144	145	146	147	148	149	150
151	152	153	154	155	156	157	158	159	160	161	162	163	164	165
166	167	168	169	170	171	172	173	174	175	176	177	178	179	180
181	182	183	184	185	186	187	188	189	190	191	192	193	194	195
196	197	198	199	200	201	202	203	204	205	206	207	208	209	210
211	212	213	214	215	216	217	218	219	220	221	222	223	224	225

Слика 2. Ератостеново сито

Пре детаљнијег разматрања појединих тестова, потребно је нешто више рећи о две битне карактеристике било ког теста који проверава да ли је задати број прост: о његовој временској сложености и одредити којој класи сложености припада. Те две карактеристике одређују ефикасност и значај самог теста.

Временска сложеност и класе сложености

У рачунарству, *временска сложеност алгоритма* представља математичку функцију која величину уноса претвара у количину времена потребну да се алгоритам изврши. Она се најчешће изражава користећи се O нотацијом која занемарује константне факторе и сабирке нижег реда величине. Када се користи на претходно описан начин, каже се да се временска сложеност означава асимптотски када величина улаза тежи бесконачности. Временска сложеност се обично процењује на тај начин што се одреди број елементарних операција које се извршавају у алгоритму, где се претпоставља да је за извршење сваке од тих операција потребна фиксно одређена количина времена. Стога се количина времена и број елементарних операција алгоритма разликују за највише константан фактор.

У рачунској теорији сложености, *класа сложености* је скуп проблема повезане сложености. Типична класа сложености се дефинише као скуп проблема који се може решити уз помоћ апстрактне машине M користећи $O(f(n))$ ресурса R, где n представља величину улаза. За алгоритам се каже да је полиномијалне временске сложености, ако је његово време извршавања ограничено одозго са полиномијалним изразом у зависности од величине улаза. На пример, ако је број n улазна величина неког алгоритма полиномијалне временске сложености, онда се његова временска сложеност може представити са $O(n^k)$, за неки константан фактор k. Проблеми за које алгоритми са полиномном временском сложеношћу постоје, припадају класи сложености P. Класа сложености P је централна у рачунској теорији сложености, а такође и у нашој причи о простим бројевима. Она се дефинише као класа сложености проблема одлучивања која се могу решити помоћу детерминистичке Тјурингове машине у полиномијалном времену.

За проблем се каже да је у одређеној класи сложености ако најбољи постојећи алгоритам за решавање тог проблема задовољава критеријуме те класе. Стога током времена, ефикаснији алгоритми за решавање проблема се могу пронаћи и на тај начин и проблем класификовати у ужу класу сложености. Тако је било и за проблем да ли је задати број прост који се означава са $PRIMES$. Дуго се веровало да $PRIMES \in P$, али

то је тек доказано са открићем алгоритма AKS[6]. У томе је уједно и главни значај AKS теста. Ипак, још увек није доказано да ли се $PRIMES$ налази у класи *P-комплетних* проблема и такође се не зна да ли припада некој од класа сложености које се налазе унутар класе P, као што су NC или L.

3.1 Пробабилистички тестови

Због своје једноставности, најпопуларнији тестови за проверу да ли је задати број прост су управо пробабилистички. Ови тестови поред броја n чија се сложеност тестира, користе и случајно изабран број a из одређеног скупа могућих вредности. Они никада не прогласе прост број сложеним, али је могуће да сложен број буде означен као прост (псеудопрости бројеви). Најпознатији пробабилистички тестови су: *Фермаов тест, Соловеј-Штрасенов тест, Леманов тест, Милер-Рабинов тест, тест помоћу елиптичких кривих,* ... Вероватноћа грешке се може смањити понављањем теста за више независно изабраних вредности за a. На пример, за Соловеј-Штрасенов тест или за Фермаов тест, за свако сложено n најмање половина a детектује његову сложеност [3]. Тако да понављајући тест k пута смањујемо вероватноћу грешке на највише 2^{-k}, што се може учинити довољно малим повећавајући k. Кораци у извршавању произвољног пробабилистичког теста су следећи:

1. Произвољно се изабере број a.
2. Испита се нека једнакост (која је специфична за сваки од тестова) која укључује изабрано a и број n који се тестира. Ако се добије негативан одговор на постављену једнакост, онда је n сложен број, a се назива *сведоком сложености* броја n и тест се прекида.
3. Вратити се поново на корак 1. све док се не постигне жељена сигурност.

Ако је након одређеног броја итерација постигнута жељена тачност и за n није установљено да је сложен, онда се он може декларисати као *вероватно прост број.* Термин веоватног простог броја подразумева да ће у већини случајева дати број бити стварно прост, али ће у неким бити псеудопрост број.

[6] AKS тест представља централни део рада и описан је у параграфу 3.2.1

3.1.1 Фермаов тест

Најпростији пробабилистички тест је *Фермаов тест* за испитивање да ли је задати број прост (тачније у овом случају говоримо о тесту сложености). Тест функционише на следећи начин: за задати цео број n, изабере се произвољно a и онда ако важи да је:

$$a^{n-1} \not\equiv 1 \pmod{n}, \tag{4}$$

n је сложен. У супротном је n вероватно прост. Сваки број a за који важи (4) када је n сложен, се назива*Фермаов лажни сведок*. Да би се повећала прецизност, тест се понавља више пута за различите вредности за a из интервала $[1, n-1]$.

Алгоритам за дати тест може се представити следећим псеудокодом:

```
Algoritam Ferma(n,k)
Ulaz: n > 3, neparan ceo broj cija se slozenost proverava
Ulaz: k, parametar koji odredjuje broj iteracija testa
Izlaz: slozen ako je n slozen, u suprotnom verovatno prost
ponoviti k puta:
    izabrati proizvoljan ceo broj a iz intervala [1, n-1]
    if a^{n-1} ≢ 1 (mod n) then return slozen
return verovatno prost
```

Фермаов тест је хеуристичке природе - неки сложени бројеви као што су Кармајклови ће увек бити декларисани као вероватно прости, без обзира колико различитих a буде искоришћено. И поред свега реченог, овај метод се користи приликом грубог и брзог тестирања, нпр. у фази генерисања кључа алгоритма RSA. Време извршавања овог алгоритма је $O(k \cdot \log^2 n \cdot \log\log n \cdot \log\log\log n)$, где k представља број различитих a-ова за које се тестирање обавља.

3.1.2 Соловеј-Штрасенов тест

У Соловеј-Штрасеновом тесту костисти се *Јакобиев симбол* који представља генерализацију *Лажандровог симбола*.

Дефиниција 3.1. *Нека је p непаран прост број и $a \in \mathbb{Z}$. Лажандров симбол $\left(\frac{a}{p}\right)$ се дефинише на следећи начин:*

$$\left(\frac{a}{p}\right) = \begin{cases} 0 \text{ ако је } a \equiv 0 \pmod{p} \\ +1 \text{ ако је } a \not\equiv 0 \pmod{p} \text{ и за неке } x \in \mathbb{Z},\ a \equiv x^2 \pmod{p} \\ -1 \text{ ако не постоји такво } x \end{cases}$$

Дефиниција 3.2. *Нека је n произвољан позитиван непаран број и $a \in \mathbb{Z}$. Јакобиев симбол $\left(\frac{a}{n}\right)$ се дефинише као производ Лажандрових симбола на следећи начин:*

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{\alpha_1}\left(\frac{a}{p_2}\right)^{\alpha_2}\cdots\left(\frac{a}{p_k}\right)^{\alpha_k}$$

где је $n = p_1^{\alpha_1}p_2^{\alpha_2}\cdots p_k^{\alpha_k}$, а $\left(\frac{a}{p_i}\right)$ представљају Лажандрове симболе за $i = 1, \ldots, k$.

Ојлер је доказао да за непаран прост број p и произвољан цео број a важи да је:

$$a^{(p-1)/2} \equiv \left(\frac{a}{p}\right) \pmod{p},$$

где $\left(\frac{a}{p}\right)$ представља *Лажандров симбол*. Пошто тест обавља проверу да ли је унети број прост, уместо Лажандровог симбола користи се Јакобијев симбол $\left(\frac{a}{n}\right)$, где n може бити било који позитиван непаран број.

Соловеј-Штрасенов тест се састоји у следећем: ако се испитује сложеност непарног броја n, изабере се произвољан цео број a и ако је:

$$a^{(n-1)/2} \not\equiv \left(\frac{a}{n}\right) \pmod{n}, \tag{5}$$

онда је n сложен и a је сведок његове сложености. У супротном, n је вероватно прост. У једнакости (5) $\left(\frac{a}{n}\right)$ представља поменути Јакобијев симбол. Видимо да се опет ради о тесту сложености.

Наредним псеудокодом приказан је Соловеј-Штрасенов алгоритам:

```
Algoritam Solovej-Strasen(n,k)
Ulaz: n > 3, neparan ceo broj cija se slozenost testira
Ulaz: k, parametar koji odredjuje preciznost testa
Izlaz: slozen ako je n slozen, u suprotnom verovatno prost
ponoviti k puta:
    izabrati proizvoljan ceo broj a iz intervala [2, n-1]
    x ← (a/n)
    if x = 0 or a^((n-1)/2) ≢ x (mod n) then return slozen
return verovatno prost
```

За сваки сложен непаран број n, најмање половина од свих база $a \in (\mathbb{Z}/n\mathbb{Z})^*$ су Ојлерови сведоци [3]. Због довољно велике учесталости Кармајклових бројева, у пракси се Соловеј-Штрасенов тест обично користи уместо Фермаовог теста. Време извршавања овог алгоритма је $O(k \cdot \log^3 n)$.

3.1.3 Милер-Рабинов тест

Најпознатији и свакако најкоришћенији пробабилистички тест је Милер-Рабинов, који се још назива и „јаки псеудопрости тест“, јер се заснива на особинама јаких псеудопростих бројева. Аутор његове оригиналне верзије, која је иначе била детерминистичка, је Милер (G.L.Miller). Детерминизам те прве верзије је заснован на недоказаној*уопштеној Римановој хипотези*. Рабин (M.O.Rabin) га је модификовао тако да буде безусловни[7] пробабилистички алгоритам. Баш као и претходно описани пробабилистички тестови, и Милер-Рабинов се заснива на једнакостима које важе за просте бројеве, а помоћу којих се испитује сложеност траженог броја.

За разумевање теста неопходна је наредна лема о квадратним коренима из јединице у коначном пољу $\mathbb{Z}/p\mathbb{Z}$, где је p прост број и $p > 2$. Очигледно је да тривијални квадратни корени -1 и 1 припадају траженом скупу бројева који при квадрирању дају остатак 1 по модулу p. Следећа лема нам говори да су они и једини корени.

Лема 3.1. *Не постоје нетривијални квадратни корени из јединице у коначном пољу* $\mathbb{Z}/p\mathbb{Z}$.

Доказ. За почетак, претпоставимо да је x квадратни корен од 1 по модулу p. То уствари значи да је:

$$x^2 \equiv 1 \pmod{p} \text{ тј.}$$

$$(x-1)(x+1) \equiv 0 \pmod{p}.$$

Другим речима, p дели производ $(x-1)(x+1)$. Пошто је p прост број следи да он дели један од фактора, тј. $x = 1$ или $x = -1$. □

Сада, нека је n прост број и $n > 2$. Онда је $n-1$ паран број и може се представити као $2^s \cdot d$, где су s и d позитивни цели бројеви и d је непарно. За свако $a \in (\mathbb{Z}/n\mathbb{Z})^*$ важи једна од следећих двеју једнакости:

$$a^d \equiv 1 \pmod{n} \text{ или}$$

$$a^{2^r \cdot d} \equiv -1 \pmod{n}, \text{ за неко } 0 \leq r \leq s-1.$$

За доказ претходно реченог послужићемо се малом Фермаовом теоремом од које крећемо. На основу Леме 3.1, ако се r пута узастопно рачунају квадратни корени из

[7] За алгоритам кажемо да је безуслован, ако његова тачност не зависи ни од једне недоказане хипотезе

a^{n-1} по модулу p, стално ће се као резултат добијати -1 или 1. Ако се добије као резултат -1, друга једначина је задовољена и ту је крај доказа. У супротном, ако се никада као резултат не добије -1, онда када се потроше сви степени броја 2, остаје да важи прва једнакост.

Милер-Рабинов тест се заснива на контрапозитивном тврђењу претходном које каже да ако се може наћи такво a тако да важе следеће две једнакости:

$$a^d \not\equiv 1 \pmod{n} \tag{6}$$

$$a^{2^r \cdot d} \not\equiv -1 \pmod{n}, \tag{7}$$

за све $0 \leq r \leq s-1$, онда је n сложен број. Број a у том случају називамо сведоком сложености броја n. У супротном, a називамо јаким лажним сведоком и n представља јак вероватно прост број за базу a.

За сваки непаран сложен број n постоји много сведока његове сложености a. Тачније, више од $3/4$ произвољно изабраних база $0 < a < n$, ће указати на његову сложеност [3]. Али и поред тога, не постоји лак начин генерисања таквих бројева a. Решење је да се тест учини пробабилистичким: бира се произвољно $a \in (\mathbb{Z}/n\mathbb{Z})$ различито од 0 и провери да ли је изабрано a сведок сложености траженог броја. Ако је n сложено, тест ће са вероватноћом од преко $3/4$ показати сложеност броја n. Али постоји могућност да смо изабрали баш оно a које је јак лажни сведок траженог броја. Ту вероватноћу можемо драстично смањити понављајући дати поступак за више независно изабраних вредности за a.

Следи псеудокод Милер-Рабиновог теста:

Algoritam Miler-Rabin(n,k)
Ulaz: $n > 3$, neparan ceo broj cija se slozenost testira
Ulaz: k, parametar koji odredjuje tacnost testa
Izlaz: *slozen* ako je n slozen, u suprotnom *verovatno prost*
odrediti s i d na opisan nacin tako da vazi $n - 1 = 2^s \cdot d$
ITERACIJA: ponoviti k puta:
    izabrati proizvoljan ceo broj a iz intervala $[2, n-2]$
    $x \leftarrow a^d \pmod{n}$
    if $x = 1$ or $x = n - 1$ then sledeca ITERACIJA
    for $r = 1 \ldots s - 1$
        $x \leftarrow x^2 \pmod{n}$
        if $x = 1$ then return *slozen*

```
        if x = n − 1 then sledeca ITERACIJA
    return slozen
return verovatno prost
```

Користећи се методом *модуларног степеновања поновљеним квадрирањем* [6] у конструкцији самог алгоритма, може се показати да је сложеност датог алгоритма $O(k \cdot \log^3 n)$, где број k представља број итерација тј. за колико различитих вредности a је тестиран дати алгоритам. На основу претходно реченог видимо да је ово је ефикасан полиномијалан алгоритам. Време извршавања се може редуковати на $O(k \cdot \log^2 n)$, користећи се множењем заснованим на брзој Фоуриеровој трансформацији.

Сада да кажемо неколико речи о тачности резултата. Како су за сваки непаран сложен број n најмање $3/4$ база a сведоци сложености броја, алгоритам проглашава n вероватно простим са вероватноћом од највише $1/4$. То значи да k итерација датог теста одређују границу прецизности од 4^{-k}. Седам најмањих сложених бројева који пролазе Милер-Рабинов тест су: 2047, 1373653, 25326001, 3215031751, 2152302898747, 3474749660383, 341550071728321. Према томе овај тест (знајући ових првих седам бројева) је валидан за све бројеве до $3.4 \cdot 10^{14}$. Нешто ефикаснији тест је *Фробенијусов тест псеудопростих бројева* [7]. Један циклус овог алгоритма је три пута дужи од циклуса Милер-Рабиновог теста, али достиже границу вероватноће која се постиже са седам циклуса Милер-Рабина.

Јасно је да се овај алгоритам може учинити детерминистичким понављајући поступак за све вредности a до одређене границе. Проналажење те границе представља главни проблем. Детерминистичка верзија Милер-Рабиновог алгоритама се не користи у пракси. Знатно је спорија од пробабилистичког алгоритма, а пошто се њен детерминизам заснива на недоказаној хипотези, за детерминистичке потребе може се користи детерминистички алгоритам AKS.

У погледу коректности резултата Милер-Рабинов тест никада није лошији од Соловеј-Штрасеновог, а Соловеј-Штрасенов никада није лошији од Фермаовог теста за било које изабрано n. Међу наведеним тестовима најбољи је Милер-Рабинов, јер захтева најмањи број операција и има једноставнију имплементацију и најмању вероватноћу грешке у односу на све остале тестове.

За крај, напоменимо још да је Милер-Рабинов алгоритам саставни део стандардне java библиотеке **java.math.BigInteger**. Имплементиран је функцијом: **boolean isProbablePrime(** **int** preciznost), где параметар preciznost показује да ако је повратна вредност функције true, вероватноћа да је тестирани број стварно прост износи: $1 - 1/2^{preciznost}$.

3.2 Детерминистички тестови

Детерминистички тестови одређују са апсолутном сигурношћу да ли је тражени број прост. Они нам дају сертификат, односно доказ да је унети број прост. Први детерминистички тест који је био значајно бржи од најпростијих линеарних метода (поменутих при уопштеној причи о тестовима) је *Адлиман* (Adleman)-*Померанс* (Pomerance)-*Ромали* (Rumely)-*ов тест* [7]. Његово време извршавања је $O((\log n)^{c \log\log\log n})$, где је n број који се тестира, а c константа независна од тог броја. За *тест помоћу елиптичких кривих* се може показати да је његово време извршавања $O((\log n)^6)$, али само ако се за нека недоказана тврђења из аналитичке теорије бројева (која се користе у његовој имплементацији) претпостави да су тачна. Слично, под претпоставком да важи уопштена Риманова хипотеза, Милер-Рабинов тест се претвара у детерминистички тест и назива Милеров тест. Његово време извршавања је $O((\log n)^4)$. При избору методе коју желимо да испрограмирамо, обично се пре одлучујемо за мало спорију али једноставнију, него за ону чија имплементација може довести до бројних програмских грешака.

Први детерминистички тест који се доказано извршава у полиномијалном времену је AKS. Откривен је 06.08.2002. године од стране тројице аутора са Индијског Института Технологије Канпур: Агравал (M.Agrawal), Кајал (N.Kayal) и Саксена (N.Saxena). Нове верзије овог теста су све ефикасније. Време извршавања прве верзије алгоритма је износило $O((\log n)^{12})$ [1], а затим користећи резултате из теорије сита[8] редуковано на $O((\log n)^{7.5})$[8]. Даље се његова сложеност може редуковати на $O((\log n)^6)$, ако узмемо у обзир да важи недоказана *Софи Жерменова хипотеза*. Ленстра (Lenstra) и Померанс (Pomerance) су унапредили претходну верзију и добили безусловни алгоритам чије је време извршавања $O((\log n)^6)$ [8].

3.2.1 AKS тест

Као што је већ речено, алгоритам одређује да ли је дати број прост или сложен у полиномијалном времену.

Кључна предност AKS теста у односу на друге тестове је у томе што је први објављен[9] алгоритам који је задовољавао свако од следећих својстава:

[8] Теорија сита је скуп техника у теорији бројева дизајнираних да изброје или прецизније процене број елемената „просејаних“скупова целих бројева

[9] Објављен је у документу под називом „ Primes is in P“ [1], који је осим самог алгоритма важан због чињенице да је доказан вековни проблем да прости бројеви припадају класи сложености P

1. **Општост**: AKS алгоритам се користи за испитивање сложености произвољно изабраног броја. За многе друге тестове се зна да раде само са бројевима који поседују нека специјална својства. На пример, Лукас-Лемеров тест [4] за Мерсенове бројеве, док Пепинов тест [4] се може употребити само за Фермаове бројеве.

2. **Полиномијално време извршавања**: Максимално време извршавања AKS теста је полиномијално у зависности од броја цифара тестираног броја. Тест помоћу елиптичких кривих и Адлеман-Померанс-Ромалиов тест такође доказују да ли је дати број прост или сложен, али за њих се не зна да ли им је горња граница времена извршавања полиномијална за све могуће улазе.

3. **Детерминизам**: Дати алгоритам је детерминистички. За разлику од њега Милер-Рабинов тест може испитати да ли је број прост у полиномијалном времену, али је зато његово решење пробабилистичке природе.

4. **Безусловност**: Тачност алгоритма AKS не зависи ни од једне недоказане хипотезе. Насупрот њему, Милеров тест је детерминистички и има полиномијално време извршавања, али се зато његова тачност заснива на недоказаној Римановој хипотези.

Као што се може видети, претходни алгоритми су задовољавали три од поменутих услова, али не и сва четри, као што је случај са AKS тестом.

Тест се у основи заснива на малој Фермаовој теореми. Теорема сама по себи намеће очигледан тест: да би тестирали сложеност броја n, треба изабрати неколико вредности a и проверити да ли једнакост (1) важи (ово у ствари представља поменути пробабилистички Фермаов тест). Ако тест падне за неку од изабраних вредности за a, онда је n сложен. Нажалост, не важи и обрнуто тврђење: постоје псеудопрости бројеви n, знани као Кармајклови бројеви, за које једнакост (1) важи за свако изабрано a. Стога мала Фермаова теорема није сама по себи довољна да би се од ње добио детерминистички тест. Због тога постоји јача верзија поменуте теореме, која елиминише све сложене бројеве и дата је у Леми 3.3. У доказу Леме 3.3 користи се следећа лема:

Лема 3.2. *Нека је p прост број. Онда је,*

$$\binom{p}{r} = \frac{p(p-1)(p-2)\cdots(p-r+1)}{r!}$$

где је $0 < r < p$, дељиво са p.

Доказ. $p(p-1)\cdots(p-r+1)$ је производ од r узастопних целих бројева, сходно томе дељиво је са $r!$. Пошто је $r!$ узајамно просто са p, следи да $r!$ дели производ $(p-1)(p-2)\cdots(p-r+1)$. То доказује да је дати биномни коефицијент $\binom{p}{r}$ дељив са p. □

Лема 3.3. *Нека $a \in \mathbb{Z}$, $n \in \mathbb{N}$, $n \geq 2$ и $(a,n) = 1$. Број n је прост ако и само ако је:*

$$(X+a)^n = X^n + a \pmod{n}. \tag{8}$$

Доказ. За $0 < i < n$, коефицијент уз X^i у развоју полинома $((X+a)^n - (X^n+a))$ је $\binom{n}{i}a^{n-i}$.

Претпоставимо најпре да је n прост број. Тада су на основу Леме 3.2 сви коефицијенти $\binom{n}{i}a^{n-i}$ конгурентни са 0.

Претпоставимо сада да је n сложен број. Нека је q прост фактор броја n такав да $q^k \mid n$ и $q^{k+1} \nmid n$. Тада q^k не дели $\binom{n}{q}$ и узајамно је просто са a^{n-q} и одатле следи да је коефицијент уз X^q различит од 0 по $\pmod{n}$. Одатле следи да $((X+a)^n - (X^n+a))$ није идентички једнако 0 у $\mathbb{Z}_n$. □

Овај алгоритам иако детерминистички, није полиномијални, зато што је за израчунавање $n+1$ коефицијената у развоју полинома $(X+a)^n$ потребно најмање $O(n)$ времена. Један од начина за редукцију броја коефицијената је израчунавање обе стране једнакости (8) по модулу полинома $X^r - 1$ за пригодно изабрано мало r. То значи да се тест сада заснива на следећој једнакости:

$$(X+a)^n = X^n + a \pmod{X^r - 1, n}. \tag{9}$$

Из Леме 3.3 очигледно је да сви прости бројеви задовољавају једнакост (9) за све вредности a и n. Проблем је у томе што неки сложени бројеви n такође задовољавају дату једнакост за одређене вредности a и n. Али, може се показати да за погодно изабрано r, ако једначина (9) важи за одређни скуп a-ова, онда је број n прост или степен простог броја. На овај начин се добија детерминистички полиномијални алгоритам за тестирање да ли је задати број прост.

За разумевање теста AKS неопходни су неки појмови и тврђења. Са $\mathbb{Z}_n$ означимо прстен бројева по модулу n. $\mathbb{F}_p$ представља коначно поље са p елемената, где је p прост број. Ако је p прост број и $h(X)$ полином степена d који је несводљив у коначном пољу $\mathbb{F}_p$, онда $\mathbb{F}_p[X]/(h(X))$ представља коначно поље реда p^d. Једнакост $f(X) = g(X) \pmod{h(X), n}$ означава једнакост $f(X) = g(X)$ у прстену $\mathbb{Z}_n[X]/(h(X))$.

Користићемо ознаку $\log$ за логаритам за основу 2 и ln за природни логаритам.

Ако $r \in \mathbb{N}$, $a \in \mathbb{Z}$ и $(a, r) = 1$, *ред броја* a *по модулу* r представља најмањи број k такав да је $a^k \equiv 1 \pmod{r}$ и обележавамо га са $o_r(a)$. За $r \in \mathbb{N}$, са $\phi(r)$ означавамо*Ојлерову* ϕ *функцију* која је дефинисана као број позитивних целих бројева мањих од r који су са њим узајамно прости.

За доказивање алгоритма AKS неопходна је проста чињеница исказана у следећој леми [1].

Лема 3.4. *Са* $NZD(m)$ *ћемо означавати најмањи заједнички садржалац првих* m *бројева. За* $m \geq 7$ *важи да је:*

$$NZD(m) \geq 2^m.$$

Следи приказ псеудокода за алгоритам AKS и након тога је изнет његов доказ у целости.

Algoritam AKS(n)
Ulaz: ceo broj $n > 1$
1. **If** ($n = a^b$ za neko $a \in \mathbb{N}$ i $b > 1$) return *SLOZEN*
2. Odrediti najmanje r tako da vazi: $o_r(n) > log^2 n$
3. **If** $1 < (a, n) < n$ za neko $a \leq r$ return *SLOZEN*
4. **If** $n \leq r$ return *PROST*
5. **For** $a = 1$ to $\lfloor \sqrt{\phi(r)} logn \rfloor$ do
   if $((X + a)^n \neq X^n + a \pmod{X^r - 1, n})$ return *SLOZEN*
6. return *PROST*

Теорема 3.1. *Алгоритам враћа* $PROST$ *ако и само ако је* n *прост број.*

Еквиваленција из тврђења теореме доказује се помоћу неколико наредних лема. Најпре је у наредној леми доказана лакша од двеју импликација.

Лема 3.5. *Ако је* n *прост број, онда алгоритам враћа* $PROST$.

Доказ. Ако је n прост број, онда кораци 1 и 3 никада не могу имати повратну вредност $SLOZEN$. На основу Леме 3.3, **for** петља такође не може вратити $SLOZEN$. На основу реченог, закључујемо да ће алгоритам идентификовати n као $PROST$, било у кораку 4 или 6. □

Импликација у супротном смеру је знатно сложенија и представља остатак доказа. Треба уствари показати да ако алгоритам враћа $PROST$, да је онда n прост број. Ако алгоритам избаци као повратну вредност $PROST$ у кораку 4, онда n мора бити прост,

јер би иначе корак 3 идентификовао n као сложен и нашао његов нетривијални фактор. Значи да је једини преостали случај када алгоритам врати $PROST$ у кораку 6. У наставку претпостављамо да је то случај који посматрамо.

Практично, кораци који захтевају мало више објашњења и пажње су 2 и 5. Корак 2 проналази одговарајућу вредност за r, док се у кораку 5 проверава да ли једнакост (9) важи за изабрани скуп вредности a. У наредној леми показује се да се на основу корака 3 може ограничити скуп могућих вредности за r.

Лема 3.6. *Постоји $r \leq max\{3, \lceil log^5 n \rceil\}$ тако да важи корак 2 алгоритма, односно услов да је $o_r(n) > log^2 n$.*

Доказ. За $n = 2$, тривијално одређујемо да $r = 3$ задовољава дату неједнакост. Зато претпоставимо да је $n > 2$. Нека је $r_1, r_2, \ldots, r_t$ скуп бројева мањих од $\lceil \log^5 n \rceil$ таквих да сваки од њих задовољава бар један од следећа два услова:

1. $o_{r_i}(n) \leq \log^2 n$ или
2. $r_i \mid n$.

Сваки од бројева r_i из поменутог скупа дели следећи производ:

$$n \cdot \prod_{i=1}^{\lfloor \log^2 n \rfloor} (n^i - 1).$$

Користећи се елементарним неједнакостима претходни производ се може ограничити одозго са:

$$n \cdot \prod_{i=1}^{\lfloor \log^2 n \rfloor} (n^i - 1) < n^{\log^4 n} \leq 2^{\log^5 n}.$$

Пошто је $n > 2$, имамо да је $\lceil \log^5 n \rceil > 10$ и на основу Леме 3.4 следи да је $NZD(\lceil \log^5 n \rceil) \geq 2^{\lceil \log^5 n \rceil} > 2^{\log^5 n}$. Стога, мора постојати број $s \leq \lceil \log^5 n \rceil$ тако да $s \notin \{r_1, r_2, \ldots, r_t\}$. Могућа су два случаја:

1. Ако је $(s, n) = 1$ онда је: $o_s(n) > \log^2 n$ и тражено $r = s$.
2. Ако је $(s, n) > 1$, пошто је s изабрано тако да не дели n и $(s, n) \in \{r_1, r_2, \ldots, r_t\}$, следи да је $r = \frac{s}{(s,n)}$, јер $r \notin \{r_1, r_2, \ldots, r_t\}$ и стога је и $o_r(n) > \log^2 n$. □

На основу претходне леме видимо да је корак 4 алгоритма релевантан само за $n \leq r_{max} = \lceil \log^5 n \rceil$, односно за $n \leq 5690034$, што се може непосредно проверити.

Пошто је $o_r(n) > 1$, мора постојати прост делилац p броја n, такав да је $o_r(p) > 1$, јер би у супротном (што значи да је p конгурентно $1 \pmod r$) производ свих простих чинилаца броја n био конгурентан са $1 \pmod r$, односно било би $o_r(n) = 1$, што је у супротности са претпоставком. Остатак доказа ће уствари показати да ако алгоритам у кораку 6 врати $PROST$, онда је $n = p$. Мора бити $p > r$, иначе би трећи корак алгоритма указао на сложеност броја n. Такође, имамо да је $(n, r) = 1$, јер би у супротном корак 3 идентификовао n као сложен. Закључујемо да $p, n \in \mathbb{Z}_r^*$. Бројеви p и r ће бити фиксирани у наставку доказа. Уведимо сада ознаку $l = \lfloor \sqrt{\phi(r)} \log n \rfloor$, која представља горњу границу вредности за a.

Корак 5 алгоритма проверава тачност једнакости (9) за l различитих вредности за a. На основу претпоставке да алгоритам враћа $PROST$ у кораку 6, следи да је једнакост у кораку 5 тачна за све $0 \leq a \leq l$. Пошто $p \mid n$, следи да је

$$(X + a)^n = X^n + a \pmod{X^r - 1, p} \tag{10}$$

за све $0 \leq a \leq l$. На основу Леме 3.3, пошто је p прост, имамо да је

$$(X + a)^p = X^p + a \pmod{X^r - 1, p} \tag{11}$$

Сада ће на основу претходних двеју једнакости бити доказано да важи и

$$(X + a)^{\frac{n}{p}} = X^{\frac{n}{p}} + a \pmod{X^r - 1, p} \tag{12}$$

за све $0 \leq a \leq l$. Нека је за неко a, $0 \leq a \leq l$,

$$(X + a)^{n/p} = \sum_{i=0}^{r-1} c_i X^i \pmod{X^r - 1, p}.$$

После степеновања обе стране једнакости са p добија се

$$(X + a)^n = \sum_{i=0}^{r-1} c_i^p X^{ip} \pmod{X^r - 1, p}.$$

У овом изразу се степени рачунају по модулу r; сви ти степени су различити, јер је $(r, p) = 1$ и $r < p$. Претходно и једнакост (10) су идентитети по X, па се изједначавањем коефицијената добија да је $c_i = 0$ за све i такве да је $ip \neq 0 \pmod r$ или $ip \neq p \cdot (n/p) \pmod r$, тј. сем за $i = 0$ или $i = n/p$. За ове i је $c_0^p = a \pmod p$, односно $c_{n/p}^p = 1 \pmod p$, одакле је на основу мале Фермаове теореме: $c_0 = a$, $c_{n/p} = 1$. Тиме је једнакост (12) доказана.

Из једнакости (10) и (12) може се приметити да се бројеви n и $\frac{n}{p}$ понашају као прост број p. Овом својству ће бити дато посебно име објашњено у следећој дефиницији:

Дефиниција 3.3. *Каже се да је број $m \in \mathbb{N}$ интроспективан за полином $f(X)$ ако важи да је:*

$$[f(X)]^m = (f(X^m)) \pmod{X^r - 1, p}. \tag{13}$$

Бројеви p и $\frac{n}{p}$ су на основу (11) и (12) интроспективни за полином $X + a$, када је $0 \leq a \leq l$. У наредне две леме износимо и доказујемо две битне особине интроспективних бројева. Прва од њих говори да су интроспективни бројеви затворени у односу на производ.

Лема 3.7. *Ако су m и m' интроспективни бројеви за полином $f(X)$, онда је то и њихов производ тј. $m \cdot m'$.*

Доказ. Пошто је m интроспективно за $f(X)$ имамо да је:

$$[f(X)]^{m \cdot m'} = (f(X^m))^{m'} \pmod{X^r - 1, p}.$$

Такође, пошто је и m' интроспективно за $f(X)$, имамо након замене X са X^m у интроспективној једначини (13) за m' да је:

$$[f(X^m)]^{m'} = f(X^{m \cdot m'}) \pmod{X^{m \cdot r} - 1, p} = f(X^{m \cdot m'}) \pmod{X^r - 1, p}$$

на основу чињенице да полином $X^r - 1$ дели полином $X^{m \cdot r} - 1$. Из претходних двеју једнакости коначно добијамо да је:

$$[f(X)]^{m \cdot m'} = f(X^{m \cdot m'}) \pmod{X^r - 1, p}. \qquad \square$$

За фиксирани број m, скуп полинома у односу на које је он интроспективан је такође затворен у односу на множење и то је исказано у следећој леми:

Лема 3.8. *Ако је број m интроспективан за полиноме $f(X)$ и $g(X)$, онда је он интроспективан и за њихов производ $f(X) \cdot g(X)$.*

Доказ. Доказ следи из једнакости:

$$[f(X) \cdot g(X)]^m = [f(X)]^m \cdot [g(X)]^m = f(X^m) \cdot g(X^m) \pmod{X^r - 1, p}. \qquad \square$$

Претходне две леме показују да сваки од бројева из скупа $I = \{\binom{n}{p}^i \cdot p^j \mid i, j \geq 0\}$ је интроспективан за сваки полином из скупа $P = \{\prod_{a=0}^{l}(X + a)^{e_a} \mid e_a \geq 0\}$. Полазећи од ових скупова уводе се две групе које имају битну улогу у наставку доказа.

Прву групу чини скуп свих остатака бројева из скупа I по модулу r. Она је подгрупа од $\mathbb{Z}_r^*$, пошто је на основу реченог $(n, r) = (p, r) = 1$. Обележимо ову групу са G и

нека је њена кардиналност t тј. $\mid G \mid = t$. G је генерисана са n и p по модулу r и пошто је из другог корака алгоритма AKS $o_r(n) > \log^2 n$, следи да је сигурно и $t > \log^2 n$.

За дефинисање друге групе, неопходни су основни појмови о*циклотомичним полиномима*. У кораку 5 алгоритма тестирамо да ли једнакост (9) важи за изабрани скуп вредности за a. Али најпре морамо да сагледамо основне особине и модуларну аритметику везану за полином $X^r - 1$. Корени овог полинома су r-ти корени из јединице. Може се приметити да ови корени формирају групу у односу на множење. Тачније, формирају цикличну групу, па тако можемо говорити о генератору ове групе.

Дефиниција 3.4. *r-ти корен из јединице ζ називамо примитивним r-тим кореном из јединице ако је ζ генератор цикличке групе r-тих корена из јединице.*

Другим речима, ζ је број такав да је $\zeta^r = 1$ и $\zeta^t \neq 1$ за свако $t < r$. Нека је ζ примитивни r-ти корен из јединице. Остали примитивни корени из јединице су облика ζ^t, где је t узајамно просто са r. Стога их има укупно $\phi(r)$. Посматрајмо сада следећи полином:

$$\Phi_r(X) = \prod_{\zeta primitivan} (X - \zeta) \tag{14}$$

Полином (14) називамо r-тим циклотомичним полиномом. Коефицијенти циклотомичних полинома су цели бројеви. Пошто полином $X^r - 1$ има као корене све r-те корене из јединице, па и примитивне, следи да $\Phi_r(X) \mid X^r - 1$. У нашем доказу користимо циклотомичне полиномиме над коначним пољем F_p. Нека је $Q_r(X)$ r-ти циклотомични полином над F_p. Из већ реченог $Q_r(X)$ дели полином $X^r - 1$. Иако су циклотомични полиноми нерастављиви над $\mathbb{Q}$, може се показати да над коначним пољем F_p полином $Q_r(X)$ може раставити на нерастављиве факторе степена $o_r(p)$ [5]. Нека је $h(X)$ један од тих нерастављивих фактора. Степен $h(X)$ је већи од један, јер је $o_r(p) > 1$. Споменуту групу чини скуп свих остатака полинома из P по модулу $h(X)$. Означимо ту групу са $\mathcal{G}$. Она је генерисана са елементима $X, X+1, X+2, \ldots, X+l$ у пољу $F = F_p[X]/(h(X))$ и представља подгрупу групе F.

Следећа лема дефинише доњу границу за величину групе $\mathcal{G}$. Ленстра(H.Lenstra Jr.) је унапредио ту границу у односу на оне у ранијим верзијама алгоритма AKS и она се доказује у следећој леми:

Лема 3.9 (Ленстра)**.** $\mid \mathcal{G} \mid \geq \binom{t+l}{t-1}$.

Доказ. Најпре приметимо на основу изнесених чињеница, да из тога што је $h(X)$ фактор циклотомичног полинома $Q_r(X)$, повлачи са собом да је X r-ти примитивни корен из јединице у F.

Сада ће бити показано да се било која два различита полинома степена мањег од t из скупа P по модулу $h(X)$ пресликавају у два различита елемента из $\mathcal{G}$. Нека су $f(X)$ и $g(X)$ два полинома из P који задовољавају поменуте услове (различити и степена мањег од t). Претпоставимо супротно да је $f(X) = g(X)$ у пољу F. Узмимо произвољно $m \in I$. Логично је и $[f(X)]^m = [g(X)]^m$ у F. Пошто је m интроспективно за оба полинома f и g и $h(X)$ дели $X^r - 1$ ($h(X) \mid Q_r(X) \mid X_r - 1$), добијамо:

$$f(X^m) = g(X^m)$$

у F. То нам уствари говори да је X^m корен полинома $Q(Y) = f(Y) - g(Y)$ за свако $m \in G$. На основу тога да је $(m, r) = 1$ (за G је већ објашњено да је подгрупа од $\mathbb{Z}_r^*$), свако X^m је примитиван r-ти корен из јединице. Сходно томе, постоје $\mid G \mid = t$ различитих корена полинома $Q(Y)$ у F. Али на основу избора полинома f и g, степен полинома $Q(Y)$ мора бити мањи од t. Стога долазимо до контрадикције и закључујемо да је $f(X) \neq g(X)$ у F.

Приметимо да је $i \neq j$ у коначном пољу F_p за $1 \leq i \neq j \leq l$ на основу следећег низа неједнакости:

$$l = \lfloor \sqrt{\phi(r)} \log n \rfloor < \sqrt{r} \log n < \sqrt{r} \cdot \sqrt{r} = r < p$$

На основу тога елементи $X, X+1, X+2, \ldots, X+l$ су сви различити у F. Такође, пошто је степен полинома h већи од један, $X + a \neq 0$ у F за свако a такво да је $0 \leq a \leq l$. Стога, постоји најмање $l + 1$ различитих полинома степена један у $\mathcal{G}$, на основу чега се лако закључује да постоји најмање $\binom{t+l}{t-1}$ различитих полинома степена мањег од t у $\mathcal{G}$. Тиме смо скуп $\mathcal{G}$ ограничили одоздо са $\binom{t+l}{t-1}$. □

Сада је циљ да се група $\mathcal{G}$ ограничи и одозго и да се затим искористи однос између тих граница. У случају да тестирани број n није једнак степену простог броја p, горња граница се може одредити на основу следеће леме:

Лема 3.10. *Ако је $n \neq p^k$, онда је $\mid \mathcal{G} \mid \leq n^{\sqrt{t}}$.*

Доказ. Дефинишимо скуп $\widehat{I}$, који је подскуп скупа I, на следећи начин:

$$\widehat{I} = \{(\frac{n}{p})^i \cdot p^j \mid 0 \leq i, j \leq \lfloor \sqrt{t} \rfloor\}.$$

Уколико n није степен броја p, онда скуп $\widehat{I}$ има $(\lfloor \sqrt{t} \rfloor + 1)^2 > t$ различитих бројева. Пошто је $\mid G \mid = t$ следи да бар два броја у скупу $\widehat{I}$ морају бити једнака по модулу r. Нека су то бројеви m_1 и m_2, такви да је $m_1 > m_2$. Значи, имамо да је

$$X^{m_1} = X^{m_2} \pmod{X^r - 1}.$$

Нека је $f(X)$ произвољан полином из P. Тада је

$$[f(X)]^{m_1} = f(X^{m_1}) \pmod{X^r - 1, p} = f(X^{m_2}) \pmod{X^r - 1, p}$$

$$= [f(X)]^{m_2} \pmod{X^r - 1, p}.$$

То показује да је

$$[f(X)]^{m_1} = [f(X)]^{m_2}$$

у пољу F. Следи да је $f(X) \in \mathcal{G}$ корен полинома $Q'(Y) = Y^{m_1} - Y^{m_2}$ у пољу F. Како је $f(X)$ произвољно изабран елемент из $\mathcal{G}$, следи да $Q'(Y)$ има најмање $\mid \mathcal{G} \mid$ различитих корена у F. Степен полинома $Q'(Y)$ је $m_1 \leq (\frac{n}{p} \cdot p)^{\lfloor\sqrt{t}\rfloor} \leq n^{\sqrt{t}}$. Пошто је на основу реченог $\mid \mathcal{G} \mid \leq m_1$, добијамо коначно да је $\mid \mathcal{G} \mid \leq n^{\sqrt{t}}$. □

Након што је група $\mathcal{G}$ ограничена, користећи се елементарним неједнакостима између граница, коначно се доказује тачност алгоритма AKS.

Лема 3.11. *Ако алгоритам врати $PROST$, онда је n прост број.*

Доказ. Претпоставимо да је повратна вредност алгоритма $PROST$. На основу Леме 3.9 имамо да је:

$$\mid \mathcal{G} \mid \geq \binom{t+l}{t-1}$$

$$\geq \binom{l+1+\lfloor\sqrt{t}\log n\rfloor}{\lfloor\sqrt{t}\log n\rfloor} (jerje : t > \sqrt{t}\log n)$$

$$\geq \binom{2\lfloor\sqrt{t}\log n\rfloor + 1}{\lfloor\sqrt{t}\log n\rfloor} (jerje : l = \lfloor\sqrt{\phi(r)}\log n\rfloor \geq \lfloor\sqrt{t}\log n\rfloor)$$

$$> 2^{\lfloor\sqrt{t}\log_n n\rfloor + 1} (jerje : \lfloor\sqrt{t}\log n\rfloor > \lfloor\log^2 n\rfloor \geq 1)$$

$$\geq 2^{\log_{n^{\sqrt{t}}}} = n^{\sqrt{t}}.$$

На основу Леме 3.10 важи да је $\mid \mathcal{G} \mid \leq n^{\sqrt{t}}$, ако n није степен броја p. Стога, мора бити $n = p^k$ за неко $k > 0$. Ако је $k > 1$, алгоритам ће онда у првом кораку идентификовати n као сложен. Значи да је $n = p$ и тиме је доказ завршен. □

Тиме је комплетиран доказ Теореме 3.1 и показана коректност алгоритма AKS. У наставку следи имплементација алгоритма. Претходно ће бити речи о временској сложености алгоритма AKS.

Временска сложеност AKS теста

У првој верзији документа, аутори су доказали да је асимптотска временска сложеност алгоритма $O(\log^{12}(n))$. Након што је објављен оригиналан документ, појавило се више усавршених варијанти алгоритма AKS. То унапређење може бити постигнуто на два начина:

1. Унапређујући процену за r у Леми 3.6. Најбољи могући сценарио био би када је $r = O(\log^2(n))$ и у том случају сложеност алгоритма би износила $O(\log^{6+\varepsilon}(n))$ за било које $\varepsilon > 0$.

2. Унапређујући процену за l које представља горњу границу скупа вредности за a за које проверавамо истинитост једнакости (9). У алгоритму AKS (приказаном псеудокодом на страни 21), **for** петља у кораку 5 мора да се изврши $\lfloor \sqrt{\phi(r)} \log n \rfloor$ пута да би се осигурало да величина групе $\mathcal{G}$ буде довољно велика. Број итерација у петљи се може редуковати, ако се може показати да мањи скуп елемената $(X + a)$ такође генерише поменуту групу.

Наредне верзије алгоритма AKS су смањивале његов степен сложености. Померанс и Ленстра су 2005. године предложили верзију сложености $O(\log^6(n))$, чиме је направљен значајан напредак и то је тренутно најбржа могућа верзија. Постоје и бројне друге, које у основи имају мању временску сложеност, али које се заснивају на још увек недоказаним претпоставкама.

Временска сложеност претходно описане и доказане верзије алгортима AKS је $O(\log^{21/2}(n))$.

4 Имплементација алгоритма AKS

Најпре ће бити говора о мотивима избора програмског језика **Java** у имплементацији алгоритма AKS. Како је тежиште програма засновано на бројним математичким израчунавањима као логичан избор се наметнуо програмски језик Java, који је последњих година знатно унапређен да подржава многе математичке функције и који се нашироко користи за израду нумеричких програма. Пакет java.math.*, дефинисан у Java библиотеци садржи у себи бројне корисне класе и функције. Пошто је основна намена алгоритма AKS рад са великим бројевима, типови података као што су: int, double или long нису довољно велики да би се у њему користили. У израчунавањима су нам потребни велики бројеви са 50 и више декадних цифара, и у ту сврху користимо

класу **java.math.BigInteger**, која је дефинисана у Java библиотеци. Такође, пошто је овај алгоритам најчешће коришћен у криптографији, а криптографски елементи су обично имплементирани у Javi (због бројних добрих заштитних својстава које Java поседује), закључујемо да се програмски језик Java намеће као прави избор.

Централни део овог рада представља доказ и имплементација алгоритма AKS. Али пре него што се крене у програмску реализацију, неопходно је најпре извршити анализу корака у његовој имплементацији. Главни задатак алгоритама је да се сви кораци извршавају у полиномијалном времену. Да би био поједностављен сам процес анализе, алгоритам је подељен у три основна дела.

Програм се састоји од две класе: **AKS.class** и **Polinom.class**. Главни део класе AKS.class чини функција **boolean prost()** која представља програмску имплементацију самог алгоритама AKS и има повратну вредност false ако је задати број сложен, односно у супротном true.

Након уношења броја n који се тестира, први део кода проверава да ли је $n = a^b$ за $b > 1$. Ако се установи да јесте, функција prost() добија повратну вредност false и даље извршавање алгоритма се прекида. Прва идеја за реализацију овог корака би била да се испита да ли је $(\lfloor n^{\frac{1}{b}} \rfloor)^b = n$, за $2 \leq b \leq \log n$. Проблем ове идеје је у погледу структуре података које користимо, јер класа java.math.BigInteger не садржи уграђену функцију за степеновање чији је експонент рационалан број, тако да се мора користити неки други приступ проблему. Решење се састоји у рачунању вредности a^m за различите вредности a и за $2 \leq m \leq \log n$. Ако је била која од тих вредности једнака са n, значи да је $n = a^m$ једнак степену броја a и функција $prost()$ добија повратну вредност false. У овом случају, корак се извршава без употребе децималних бројева. Пошто се потрага за бројем a ради бинарном претрагом, временска сложеност првог корака је $\log^2 n$ (сложеност бинарне претраге је $\log n$). Стога се целокупан корак извршава у полиномијалном времену. Како се у овом кораку први пут јавља потреба за израчунавањем $\log n$, објаснићемо како се он рачуна. У питању је логаритам великог броја за који не постоји уграђена функција за израчунавање. Зато се користи чињеница да је:

$$\log_{10} n = Math.\log10(d) + l,$$

где је $d = n/10^{length(n)}$, а $l = length(n)$. Наравно, у програму се користи логаритам за основу 2 који добијамо формулом за промену основе логаритма

$$\log_2 n = \frac{\log_{10} n}{\log_{10} 2} = \frac{Math.\log10(d) + l}{Math.\log10(2)}.$$

На основу овога лако се може израчунати логаритам било ког великог броја.

Други део анализе обухвата други, трећи и четврти корак алгоритма. Потребно је најпре наћи најмање r тако да $o_r(n) > \log^2 n$. За рачунање $o_r(n)$ написана је функција $\mathrm{BigInteger\ multiplicativeOrder(BigInteger\ r)}$. Класа $\mathrm{java.math.BigInteger}$ садржи бројне функције за рад са великим бројевима међу којима и $\mathrm{BigInteger\ gcd(Biginteger\ a)}$, а коју користимо у реализацији трећег корака алгоритма. Четврти корак алгоритма је једноставна провера.

n	r	t(s)
83	59	1
167	101	3
353	79	4
761	97	11
1669	131	36
3719	167	103
8221	179	129
17923	227	439
38953	239	622
84121	269	1193
180569	359	3688
386153	349	4253
821753	389	6513
1742647	457	14644

Слика 3. Табеларни приказ тестираних насумично изабраних простих бројева и њима одговарајућих добијених вредности за r и време извршавања

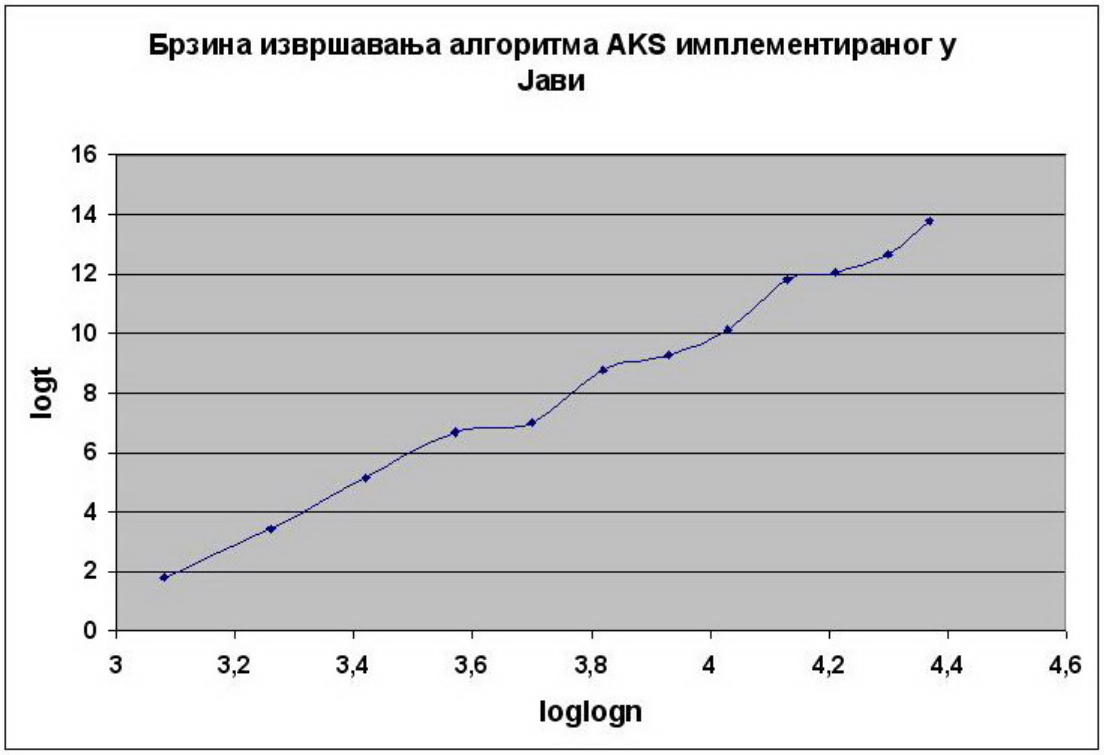
Трећи део анализе представља свакако његов главни део који узима највише процесорског времена и од кога зависи временска сложеност самог програма. Модуларно степеновање поновљеним квадрирањем и полиномна аритметика примењена у овом кораку је урађена у посебној класи $\mathrm{Polinom}$. Треба посебно издвојити функцију $\mathrm{Polinom\ modPow(BigInteger\ eksponent,\ Polinom\ mPolinom,\ BigInteger\ m)}$ која као повратну вредност има полином $this^{eksponent}(mod\ m, mPolinom)$ и игра кључну улогу у имплементацији корака 5 алгоритма AKS. Сва израчунавања у класи $\mathrm{Polinom}$ се извршавају у полиномијалном времену. Овде се први пут користи Ојлерова ϕ функција за одређивање горње границе **for** петље и њен аргумент је r. Из резултата рада програма приказаних на слици 3 може се закључити да се удвостручавањем броја бита тестираног броја n, одговарајућа вредност за r повећа тек за неку цифру. Ојлерова ϕ функција је имплеметирана функцијом $\mathrm{BigInteger\ totient(BigInteger\ n)}$. Приликом њене програмске реализације коришћена је чињеница да се за задати број $r = p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k}$,

његова Ојлерова функција може израчунати помоћу следеће формуле:

$$\phi(r) = r \cdot (1 - 1/p_1) \cdot (1 - 1/p_2) \cdots (1 - 1/p_k).$$

Аргументи ове функције нису јако велики бројеви. Пошто је њен аргумент у програму r, на основу реченог, закључујемо да она задовољава тражене услове.

Овим је укратко објашњена имплементација комплетног алгоритма. Класа AKS садржи све неопходне функције и конструкторе, који су убачени у једноставан GUI. Простим притиском на дугме алгоритам врши проверу да ли је задати број прост.



Слика 4. Графички приказ трајања времена извршавања имплементираног алгоритма AKS за просте бројеве различите величине

На слици 4. дат је графички приказ зависности времена извршавања имплементираног алгоритма AKS од величине тестираног простог броја. Тестирани прости бројеви су насумично изабрани и могу се видети у табели на слици 3. Из нагиба криве у приказаном $\log\log$ дијаграму може се оценити да је време извршавања имплементираног алгоритма за тестиране просте бројеве приближно пропорционално са $O((logn)^9)$, односно мање је од предвиђене теоретске временске сложености алгоритма која износи $O((logn)^{21/2})$. Осим алгоритма AKS, тестирање је урађено на истоветним бројева и на уграђеној библиотечкој функцији $\mathrm{isProbablePrime(\ int\ preciznost)}$, која представља

имплеметацију Милер-Рабиновог теста у Javi. Претходна функција је за све бројеве открила да су вероватно прости за мање од једне секунде, док се рецимо за последњи тест пример имплементирани алгоритам AKS извршавао више од четири сата. Ово илуструје претходно објашњење да је главни значај AKS теста теоретски и да се у пракси користи само у ретким ситуацијама.

5 Закључак

Прости бројеви имају веома битну улогу у математици, а нарочито у криптографији. Развојем рачунара, али пре свега откривањем нових метода и алгоритама или унапређивањем пређашњих, постаје неопходно да се тестирање да ли је задати број прост обавља брже. Многи алгоритми, укључујући и најзаступљенији шифарски асиметрични систем RSA, морају да користе јавни кључ толико велики да се одговарајући тајни кључ не може открити потпуном претрагом. Генерисање кључева повезано је са проналажењем великих простих бројева, а тај проблем своди се на проверу да ли су насумично изабрани природни бројеви одговарајуће величине прости. Сложеност броја се може утврдити помоћу пробабилистичких и детерминистичких тестова. Као најпознатији представник пробабилистичких тестова подробније је обрађен Милер-Рабинов, а највећи део рада посвећен је детерминистичком AKS тесту.

AKS тест је веома важан алгоритам у теорији сложености - на основу њега је доказано да проблем да ли је број прост, припада класи сложености P. С друге стране, алгоритам AKS нема велики практичан значај и потиснут је у области криптографије од других, значајно бржих алгоритама. У пракси се често користи Милер-Рабинов алгоритам, а затим се за добијање сертификата да је задати број заиста прост користи неки од детерминистичких тестова. Уз комплетан доказ коректности алгоритма AKS, у раду је описана и његова имплементација у програмском језику Java.

Литература

[1] M.Agrawal, N.Kayal, N.Saxena, *Primes is in P*, Annals of mathematics 160(2): 781-793, 2004.

[2] S.C.Coutinho, *The mathematics of ciphers: number theory and RSA cryptography*, A K Peters, Natick, 1999.

[3] N.Koblitz, *A course in number theory and cryptography*, Springer-Verlag, New York, 1998.

[4] R.Crandall, C.Pomerance, *Prime numbers: a computional perspective*, Springer, New York, 2005.

[5] R.Lidl, H.Niederreiter, *Introduction to finite fields and their applications*, Cambrige Univ. Press, Cambrige, 1986.

[6] poincare.matf.bg.ac.rs/∼ezivkom/nastava/kripto.pdf

[7] en.wikipedia.org/wiki/Primality_test

[8] en.wikipedia.org/wiki/AKS_primality_test

[9] www.rsa.com/rsalabs/node.asp?id=3723

[10] primes.utm.edu/largest.html

[11] www.prothsearch.net/fermat.html