УНИВЕРЗИТЕТ У БЕОГРАДУ
МАТЕМАТИЧКИ ФАКУЛТЕТ

Александра Дивљаковић

ПРЕБРОЈАВАЊЕ МОНОТОНИХ БУЛОВИХ ФУНКЦИЈА

мастер рад

Београд, 2023.

Ментор:

др Миодраг Живковић,
Универзитет у Београду, Математички факултет

Чланови комисије:

др Филип Марић, редовни професор
Универзитет у Београду, Математички факултет

др Иван Чукић, доцент
Универзитет у Београду, Математички факултет

Датум одбране: 2023.

Мами, тати и брату

Наслов мастер рада: Пребројавање монотоних Булових функција

Резиме: У овом раду проучавају се монотоне Булове функције. Анализирају се и имплементирају неки од познатих алгоритама за пребројавање монотоних Булових функција, а самим тим и за решавање Дедекиндовог проблема.

Кључне речи: алгоритми, Булове функиције, монотоност, Дедекиндов проблем, Дедекиндови бројеви

Садржај

Глава 1

Увод

Нека је скуп $B = \{0, 1\}$. Булове функције од n променљивих су функције $B^n \mapsto B$. Булове функције су откривене у 19. веку и добиле су назив по творцу Џорџу Булу (George Boole). Користе се у рачунарству, електроници и представљају основу математичке логике, а нашле су примену и у различитим другим областима. Ево неких од најзначајнијих примена Булових функција:

- Рачунарство - Булове функције су основа за рад са логичким променљивим у програмирању. Користе се за доношење одлука, контролу тока програма и проверу различитих услова. Условни изрази, логички оператори (И, ИЛИ, НЕ) и све компаративне операције (нпр. $<$, $>$, $\leq$, $\geq$) конструисане су на основима Булове алгебре.

- Електроника - У дигиталној електроници, Булове функције представљају основне градивне блокове комбинационих логичких кола. Користе се за обраду и манипулацију података, као што је усмеравање сигнала, селекција података и аритметичке операције.

- *CPU* архитектура - Процесори и меморије у рачунарима користе Булове функције за извршавање логичких операција. Примена Булових функција у централним процесорима доприноси обради података и доношењу одлука у складу са програмом.

- Криптографија - Булове функције користе се у криптографији за шифровање и дешифровање података. Користе се у AEC алгоритму.

- Вештачка интелигенција - У области вештачке интелигенције, Булове функције користе за моделирање логичких веза и различитих аспеката логике.

- Математика и логика - Булова алгебра представља област математике која се бави анализом Булових функција и њиховим својствима.

Укратко, Булове функције су саставни део модерних технологија и научних истраживања. Оне омогућавају да се логички изрази примењују на обраду информација и доношење одлука у рачунарима.

Монотоне Булове функције су подскуп Булових функција које имају својство монотоности. Монотона функција је функција код које када се било која вредност улазног аргумента промени са 0 на 1, онда вредност функције остаје непромењена или се промени са 0 на 1.

Одређивање броја монотонох Булових функција је познат као Дедекиндов проблем.

1.1 Дедекиндов проблем

Дедекиндов проблем, који је назван по математичару Р. Дедекинду (Richard Dedekind), је математички проблем који се односи на одређивање броја $d(n)$ за дати природни број n. Број $d(n)$ представља број различитих монотоних Булових функција од n променљивих. Првих девет до сад израчунатих бројева приказано је у табели 1.1.

n	$d(n)$	математичари
0	2	
1	3	
2	6	Р. Дедекинд 1897.
3	20	
4	168	
5	7581	Р. Черч 1940.
6	7828354	М. Вард 1946.
7	2414682040998	Р. Черч 1956.
8	5613043722868755790788	Д. Виједеман 1991.
9	286386577668298411128469151667598498812366	К. Јакел 2023.

Табела 1.1: До сада познати Дедекиндови бројеви

Одређивање тачних вредности броја $d(n)$ за веће вредности n представља изазов и не постоји једноставна аналитичка формула за израчунавање сваког од њих. Запажа се да Дедекиндови бројеви представљају брзо растући низ бројева. Потребно је пронаћи алгоритам за њихово брзо и ефикасно рачунање. Познато је неколико алгоритама који се користе у сврху рачунања овог проблема.

Историјски преглед израчунавања првих девет Дедекиндових бројева:

1. Прва четири броја ($d(1)$, $d(2)$, $d(3)$, $d(4)$) открио је сам Дедекинд. Представио их је у свом раду 1897. године. У овом раду је и постављена основа за проблем Дедекиндових бројева [1].

2. Пети број $d(5)$ открио је Р. Черч (*Randolph Church*) 1940. године [2].

3. Шести број $d(6)$ открио је М. Вард ($Morgan\ Ward$) 1946. године [8].

4. Након петог броја, Черч је успео да открије и седми број $d(7)$ 1956. године [3].

5. Осми број $d(8)$ је израчунао Д. Вијдеман (*Doug Wiedemann*) 1991. године [10].

6. Девети број $d(9)$ израчунао је К. Јакел (*Christian Jakel*) 2023. године [5].

Овај хронолошки приказ показује да је истраживање Дедекиндових бројева веома дуготрајан процес. Поред тога приказује и да је у истраживање Дедекиндових бројева био укључен велики број математичара кроз временски период од преко сто година.

У овом раду приказује се и реализује неколико алгоритама за израчунавање Дедекиндових бројева.

Глава 2

Основни појмови и дефиниције

Булове вредности се обично представљају преко битова, где се вредност 0 користи за представљање нетачне вредности, а вредност 1 за представљање вредности тачно. Овај начин представљања се користи у рачунарству и логици, где се операције обично изводе уз помоћ Булових функција и логичких оператора. Бинарни систем са две вредности 0 и 1 је основа савремених рачунарских система и представља основу за обраду информација у рачунарским програмима.

Скуп Булових променљивих обележава се са $B = \{0, 1\}$.

2.1 Булове функције

Булове функције су функције које врше израчунавања на Буловим вредностима. Другим речима, улазни аргументи Булових функција, као и сама функција, узимају вредности из скупа $B = \{0, 1\}$. Булове функције од n променљивих су пресликавање $f : B^n \mapsto B$.

Таблице истинитости приказују све комбинације улазних вредности и добијене вредности Булових функција.

Пример 2.1.1. *Пример Булове функције је дусјункција $f(x, y) = x \vee y$ (операција ИЛИ). Њена таблица истинитости је приказана у табели 2.1.*

За сваку функцију $f : B^n \mapsto B$ постоји 2^n комбинација улаза, јер свака компонента улаза може узети вредност 0 или 1. Поред тога, постоје две могућности за излаз ових функција, 0 или 1. Према томе постоји 2^{2^n} функција $f : B^n \mapsto B$.

x	y	$f(x,y)$
0	0	0
0	1	1
1	0	1
1	1	1

Табела 2.1: Табела истинитости дисјункције

2.2 Монотоне Булове функције

Релација поретка у скупу B представља однос „МАЊЕ ИЛИ ЈЕДНАКО" између елемената скупа. Релација поретка обично се обележава са $\leq$. Пошто скуп Булових вредности садржи само два елемента 0 и 1, релација поретка између њих је следећа:

- $0 \leq 0$
- $0 \leq 1$
- $1 \leq 1$.

Дефиниција 2.2.1. *Булова функција $f : \{0,1\}^n \mapsto \{0,1\}$ је монотона ако за свака два улазна вектора $x = (x_1, x_2, ..., x_n)$ и $y = (y_1, y_2, ..., y_n)$ таква да је $x_i \leq y_i$ за свако i, важи $f(x_i) \leq f(y_i)$.*

Пример 2.2.1. *Један пример монотоне Булове функције је дисјункција $f(x) = x_1 \vee x_2$. Ова функција је монотона, јер ако је $(x_1, x_2) \leq (y_1, y_2)$ онда и $f(x_1, x_2) \leq f(y_1, y_2)$. Наиме, дисјункција $f(y)$ је нетачна, односно 0, само уколико су све вредности улазних аргумената нетачне (табела 2.1). Тада је $x = (0,0) \leq y = (0,0)$ и $f(x) = 0 \leq f(y) = 0$.*

Пример 2.2.2. *Пример монотоне Булове функције је конјукција $f(x) = x_1 \wedge x_2$. Ова функција је монотона, јер ако је $(x_1, x_2) \leq (y_1, y_2)$ онда и $f(x_1, x_2) \leq f(y_1, y_2)$. Наиме, конјукција $f(y)$ је тачна, односно 1, само уколико су све вредности улазних аргумената тачне (табела 2.2). Тада је $x = (1,1) \leq y = (1,1)$ и $f(x) = 1 \leq f(y) = 1$.*

Пример 2.2.3. *Булова функција која није монотона је негација, односно $f(x) = \neg x$ (табела 2.3). Ова функција није монотона, јер вредност 1 применом негације постаје 0, а то доводи до нарушавања поретка.*

x	y	$f(x,y)$
0	0	0
0	1	0
1	0	0
1	1	1

Табела 2.2: Табела истинитости конјукција

x	$f(x)$
0	1
1	0

Табела 2.3: Табела истинитости негације

Број монотоних Булових функција од n променљивих обележава се са $d(n)$. Израчунавање овог броја је познато као Дедекиндов проблем, а сам број $d(n)$ као Дедекиндов број. Скуп монотонох Булових функција од n променљивих обележава се са M_n.

Пример 2.2.4. *У скупу од* 16 *Булових функција, односно функција две променљиве, само* 6 *су монотоне (табела* 2.3*).*

$x1$	$x2$	$f1$	$f2$	$f3$	$f4$	$f5$	$f6$
0	0	0	0	0	0	0	1
0	1	0	0	0	1	1	1
1	0	0	0	1	0	1	1
1	1	0	1	1	1	1	1

Табела 2.4: Скуп M_2 монотоних Булових функција 2 променљиве

За две монотоне Булове функције f_1 и f_2 важи:

$$x \leq y \rightarrow (f_1(x) \wedge f_2(x) \leq f_1(y) \wedge f_2(y))$$

$$x \leq y \rightarrow (f_1(x) \vee f_2(x) \leq f_1(y) \vee f_2(y).$$

Ово повлачи да је конјукција, односно дисјункција две монотоне Булове функције такође монотона Булова функција.

Глава 3

Алгоритми за рачунање Дедекиндових бројева

За израчунавање досада израчунатих Дедекиндових бројева коришћено је неколико алгоритама. Као што је речено, због брзине раста Дедекиндових бројева, може се закључити да израчунавање може бити изазовно у погледу временске сложености или меморијских захтева, посебно за велике вредности параметра n. Алгоритми који се разматрају у овом раду заснивају се на теоремама које описују својства монотоних Булових функција [4].

3.1 Неопходне дефиниције и теореме

Булове функције n променљивих се могу представити низовима нула и јединица дужине 2^n. Овај низ представља резултат Булове функције од n променљивих. Из табеле 2.1, може се запазити да би запис дисјункције био $(0, 1, 1, 1)$.

Приликом имплементације алгоритама, Булове функције n променљивих могу да се представе као стрингови (нпр. *Python*) дужине 2^n, или као битсетови уколико то програмски језик подржава (нпр. $C++$, $std :: bitset$). Конкатенација (спајање) два стринга представља надовезивање битова.

Пример 3.1.1. *У скупу M_0 елементи су $\{0, 1\}$. Као стрингови, елементи су „0" и „1", а као битсетови се представљају као $0b0$ и $0b1$. Битсетови представљају низове битова фиксне дужине над којима могу да се извршавају логичке операције. Конкатенацију код стрингова добијамо као $0+1 = "0"+"1" = "01"$.*

Уколико се користе битсетови, потребно је користити конверзију између битсетова и стрингова.

Релација поретка у скупу M_n монотоних Булових функција дефинише се неједнакошћу $g \leq h$ која важи ако и само ако $g(x) \leq h(x)$ за свако $x \in B^n$.

Лема 3.1.1. *[4] Свака монотона функција $q \in M_n$ може бити представљена јединственом формом*

$$q(x_1, x_2, ..., x_n) = q_0(x_2, ..., x_n) + q_1(x_2, ..., x_n) \cdot x_1,$$

где је $q_0 \leq q_1$, $q_0, q_1 \in M_{n-1}$. При томе је $q_0(x_2, \ldots, x_n) = q(0, x_2, \ldots, x_n)$ и $q_1(x_2, \ldots, x_n) = q(1, x_2, \ldots, x_n)$.

Доказ. Заменом x_1 најпре са 0, па са 1, непосредно се доказује да је $q(x_1, x_2, \ldots, x_n) = q(0, x_2, \ldots, x_n) + x_1 \cdot q(1, x_2, \ldots, x_n)$. Претпоставимо да постоји представа функције q у овом облику, тј. да је $q_0, q_1 \in M_{n-1}$, $q_0 \leq q_1$ и

$$q(x_1, x_2, .., x_n) = q_0(x_2, .., x_n) + q_1(x_2, .., x_n) \cdot x_1.$$

Функција q је монотона као композиција монотоних функција и важи $q_0(x_2, \ldots, x_n) = q(0, x_2, \ldots, x_n)$ и $q_1(x_2, \ldots, x_n) = q(1, x_2, \ldots, x_n)$. Заиста,

$$\begin{aligned} q_0(x_2, \ldots, x_n) &= q_0(x_2, \ldots, x_n) + 0 = \\ &= q_0(x_2, \ldots, x_n) + 0 \cdot q1(x_2, \ldots, x_n) = \\ &= q(0, \ldots, x_n) \end{aligned}$$

На сличан начин изводи се и $q_1(x_2, \ldots, x_n) = q(1, x_2, \ldots, x_n)$. У формули $q(x_1, x_2, \ldots, x_n)$ заменимо $x_1 = 1$:

$$\begin{aligned} q(x_1, x_2, \ldots, x_n) &= q_0(x_2, \ldots, x_n) + q_1(x_2, \ldots, x_n) \cdot 1 = \\ &= q(0, x_2, \ldots, x_n) + q_1(x_2, \ldots, x_n) = \\ &= q_1(x_2, \ldots, x_n), \end{aligned}$$

где последња неједнакост следи из $q_0 \leq q_1$.
Користећи добијене вредности $q_0(x_2, \ldots, x_n)$ и $q_1(x_2, \ldots, x_n)$ добијамо:
$q(x_1, x_2, \ldots, x_n) = q(0, x_2, \ldots, x_n) + x_1 \cdot q(1, x_2, \ldots, x_n) =$
$= q_0(x_2, \ldots, x_n) + x_1 \cdot q_1(x_2, \ldots, x_n).$
Ово потврђује јединственост. □

Из ове леме може се извести неколико последица које омогућавају извођење алгоритама за израчунавање Дедекиндових бројева. Најважнија последица је да се за конструисање нових Булових функција може користити конкатенација.

Последица 3.1.1. *[4] Постоји бијекција између елемената из M_n и скупа монотоних функција које пресликавају B у M_{n-1}.*

Доказ. Да би се показало да постоји бијекција између елемената из скупа M_n и скупа функција које пресликавају B у M_{n-1}, треба се доказати да може да се установи један-на-један кореспонденција између тих скупова, што значи да сваком елементу из M_n одговара једна монотона функција која пресликава B у M_{n-1}, и обрнуто.

Нека је $f \in M_n$ монотона Булова функција од n променљивих. Потребно је дефинисати функцију $g_f : B \mapsto M_{n-1}$ која одговара функцији f.

Нека је функција $g_f : B \mapsto M_{n-1}$ дефинисана једнакостима:

$g_f(0) = f(0, x_2 \dots x_n)$

$g_f(1) = f(1, x_2 \dots x_n)$.

Придруживање $f \mapsto g_f$ је једнозначно јер свака функција из $f \in M_n$ једнозначно одређује монотони пар функција $f_o, f_1 \in M_{n-1}$ за које важи да је $f_o \leq f_1$, и обрнуто сваки пар дефинише једну функцију из M_n. Пар ових функција одговара монотоној функцији $B \mapsto M_{n-1}$. Стога постоје функције f_0 и f_1 такве да важе следеће неједнакости:

$g_f(0) = f_0(x_2 \dots x_n) = f(0, x_2 \dots x_n)$

$g_f(1) = f_1(x_2 \dots x_n) = f(1, x_2 \dots x_n)$.

Према томе, доказано је да постоји бијекција између скупа M_n и скупа функција које пресликавају скуп B у скуп монотоних функција M_{n-1}. □

Важи и општија верзија ове последице.

Последица 3.1.2. *[4] За произвољно $k, 1 \leq k < n$, постоји бијекција између елемената из скупа M_n и скупа функција које пресликавају B^k у M_{n-k}.*

Доказ. Знамо да постоји функција $g_f : B \mapsto M_{n-1}$ која одговара функцији $f \in M_n$ дефинисана једнакостима:

$g_f(0) = f(0, x_2 \dots x_n)$

$g_f(1) = f(1, x_2 \dots x_n)$.

као и парови функција $f_o, f_1 \in M_{n-1}$ где важи да је $f_o \leq f_1$, тако да важи

$g_f(0) = f_0(x_2 \dots x_n) = f(0, x_2 \dots x_n)$

$g_f(1) = f_1(x_2 \dots x_n) = f(1, x_2 \dots x_n)$.

Уколико се на ове функције примени лема 3.1.1. добијају се следеће једнакости:

$g_f(0) = f_0(x_2 \dots x_n) = f_{00}(x_3, \dots, x_n) + f_{01}(x_3, \dots, x_n) \cdot x_2 = f(0, x_2 \dots x_n)$
$g_f(1) = f_1(x_2 \dots x_n) = f_{10}(x_3, \dots, x_n) + f_{11}(x_3, \dots, x_n) \cdot x_2 = f(1, x_2 \dots x_n)$.
Постојање функција $f_{00}, f_{01}, f_{10}, f_{11} \in M_{n-2}$ показује да постоји бијекција између скупа M_n и скупа функција које пресликавају B у M_{n-2}. Уколико се k пута примени лема 3.1.1, добија се да постоји бијекција између скупа M_n и скупа функција које пресликавају B у M_{n-k}. □

На основу ове последице могу се извести алгоритми за пребројавање монотоних Булових функција, сваки алгоритам за другу вредност параметра k.

Дефиниција 3.1.1. *Дисјункција* $+$ *у скупу* M_n *се дефинише једнакошћу*

$$(a_1, a_2, ..., a_{2^n}) + (b_1, b_2, ..., b_{2^n}) = (a_1 + b_1, a_2 + b_2, ..., a_{2^n} + b_{2^n}).$$

Конјукција $\cdot$ *у скупу* M_n *се дефинише једнакошћу*

$$(a_1, a_2, ..., a_{2^n}) \cdot (b_1, b_2, ..., b_{2^n}) = (a_1 \cdot b_1, a_2 \cdot b_2, ..., a_{2^n} \cdot b_{2^n}).$$

Пример 3.1.2. *Пример дисјункције у скупу* M_2 *монотоних Булових функција:*

$$(0, 0, 0, 1) + (0, 1, 1, 1) = (0, 1, 1, 1).$$

Пример конјукције у скупу M_2 *монотоних Булових функција:*

$$(0, 0, 0, 1) \cdot (0, 1, 1, 1) = (0, 0, 0, 1).$$

Низови $(0, 0, 0, 1)$ *и* $(0, 1, 1, 1)$ *представљају монотоне Булове функције, па су и њихова дисјункција* $(0, 1, 1, 1)$*, односно конјукција* $(0, 0, 0, 1)$ *монотоне Булове функције.*

Пример 3.1.3. *Пример рачунања скупа* M_2 *из скупа* M_0*. При чему се сваки наредни скуп добија конкатенацијом свих монотоних парова елемената његовог претходника. Полазећи од скупа Булових функција, за које је дужина сваког елемента* $2^0 = 1$*, односно од скупа* $M_0 = \{0, 1\} = B$*, треба да се добије скуп* M_1*, чија дужина елемената треба да буде* $2^1 = 2$*. Пореде се сви елементи почетног скупа. У скупу* M_0 *пореде се елементи, јер се ниске састоје од једног бита, док за дуже Булове функције, пореди се бит по бит, односно утврђује се да су битови на истим местима у поретку. Уколико су елементи у поретку, вршимо конкатенацију. У скупу* M_0 *имамо три монотона пара*

$0 \leq 0$, $0 \leq 1$, $1 \leq 1$, *тако да се добија* $M_1 = \{0+0, 0+1, 1+1\} = \{00, 01, 11\}$. *Број Булових функција у овом скупу је 3, што представља први Дедекиндов број* $d(1)$.

Када је добијен скуп M_1, *може се добити његов следбеник* M_2, *чији елементи су дужине* $2^2 = 4$. *Поново се понавља поступак. Проверава се који парови елемената скупа* M_1 *су монотони. Пошто је елемент скупа дужине* M_1 2, *вршимо проверу првог бита једног елемента са првим битом другог и другог бита првог елемента са другим битом другог. У скупу* M_1, *овом провером се може закључити да постоје следећи монотони парови:* $00 \leq 00$, $00 \leq 01$, $00 \leq 11$, $01 \leq 01$, $01 \leq 11$, $11 \leq 11$. *Конкатенацијом ових парова добија се* $M_2 = \{00+00, 00+01, 00+10, 00+11, 01+011+1\} = \{0000, 0001, 0011, 0101, 0111, 1111\}$.

$$\{0, 1\} \mapsto \{00, 01, 11\} \mapsto \{0000, 0001, 0011, 0101, 0111, 1111\}$$

Дужина скупа M_2 *је* $d(2) = 6$.

Сваки следећи скуп се може одредити на исти начин. Због природе ових бројева и скупова, за сваки следећи елемент потребно је много више времена за креирање скупа.

3.2 Први алгоритам

За извођење првог алгоритма [4] користи се последица 3.1.1.

У првом алгоритму на основу претходног скупа M_{n-1}, формира се скуп M_n као у примеру 3.1.3. Процес се обавља применом конкатенације на све монотоне парове чланова скупа M_{n-1}. Скупу M_n припада и елемент који се добија конкатенацијом елемента из M_{n-1} са самим собом.

Имплементација овог алгоритма у $C++$ приказана је у одељку 4.1.

У овом алгоритму постоје две угнежђене петље исте дужине. Спољна петља се извршава $m = d(n-1)$ пута. Унутар угнежђене петље извршава се провера монотоности код парова елемената скупа M_{n-1}, која има временску сложеност $O(2^{n-1})$. Временска сложеност овог алгоритма може се апроксимирати са $O(2^n \cdot m^2)$. Другим речима, временска сложеност расте квадратно у односу на величину улазних података $m = d(n-1)$.

Алгоритам 1: Први алгоритам

Улаз : $m = |M_n|$ величина скупа M_{n-1},
$M = M_n$ скуп M_{n-1}
Излаз: $m_{new} = |M_n|$ величина скупа M_n

```
1  k = 0 // бројач елемената новог скупа
2  for i = 0 to m do
3      for j = 0 to m do
4          if M[i] ≤ M[j] /* проверава се да ли елементи одржавају
               поредак                                          */
5          then
6              k = k + 1 // повећава се бројач
7              M_new[k] = M[i] + M[j] /* креира се елемент новог скупа
                   M_new                                        */
8          end if
9      end for
10 end for
11 m_new = k /* величина следећег низа се поставља на вредност
       бројача                                                  */
```

Меморијска сложеност овог алгоритма је $O(1)$, јер није потребно да се чувају елементи новог низа - могу се исписати у фајл. Уколико чувамо те елементе меморијска сложеност алгоритма је $O(len(M_n))$.

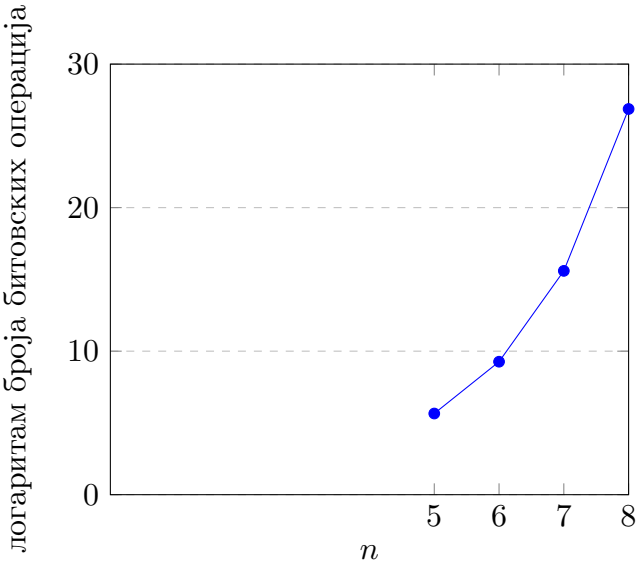
У табели 3.1 је представљен број битовских операција потребних за рачунање $d(5)$, $d(6)$, $d(7)$ и $d(8)$. Дијаграм 3.1 представља зависност логаритма броја битовских операција првог алгоритма од n.

n	број битовски операција
5	451584
6	1.83908995×10^9
7	$3.92212009 \times 10^{15}$
8	$7.46328238 \times 10^{26}$

Табела 3.1: Број битовски операција првог алгоритма

3.3 Други алгоритам

За извођење другог алгоритма [4] користи се последица 3.1.2 за вредност параметра k узима $k = 2$.



Слика 3.1: Зависност логаритма броја битовских операција првог алгоритма од n

Последица 3.3.1. *Постоји бијекција између елемената из M_n и скупа функција које пресликавају $B^2 \mapsto M_{n-2}$.*

Идеја је да се за сваки пар елемената $u, v \in M_{n-2}$ преброје монотоне функције $q : B^2 \mapsto M_{n-2}$ за које важи $q(00) = u$ и $q(11) = v$.

Нека r_e означава матрицу броја елемената који се у поретку налазе између два елемента у скупу M_{n-2}. Врсте и колоне матрице r_e „нумерисане" су елементима скупа M_{n-2}:

$$r_e[u, v] = |\{w | u \leq w \leq v; u, v, w \in M_{n-2}\}|.$$

Матрица r представља матрицу која показује да ли су два елемената из скупа M_{n-2} у одговарајућем поретку: $r[u, v] = 1$ ако је $u \leq v$, иначе $r[u, v] = 0$.

Лема 3.3.1. *[4] Матрица r_e се може изразити преко матрице r:*

$$r_e = rr$$

Доказ. Матрично множење матрице r са самом собом може се записати следећим изразом:

$$rr[u, v] = \sum_{k \in M_{n-2}} r[u, k] \cdot r[k, v].$$

Вредност суме кад испитамо вредности k:

- Ако је $k < u$ или $v < k$, тада је $r[u,k] = 0$ или $r[k,v] = 0$, ови сабирци не доприносе суми.
- Ако је $u \leq k \leq v$, тада је $r[u,k] = 1$ и $r[k,v] = 1$, што значи да је сума једнака броју елемената w који задовољавају услов $u \leq w \leq v$, односно $r_e[u,v]$

Може се видети да сума у изразу за $(rr)[u,v]$ одговара изразу за $r_e[u,v]$, што значи да су матрице r_e и rr једнаке. □

Алгоритам функционише на следећи начин: рачуна број функција q које задовољавају услове $q(0,0) = u$ и $q(1,1) = v$, где је $u \leq v$ и $u, v \in M_{n-2}$. Може се приметити да за сваки пар $w, x \in M_{n-2}$ који задовољава $u \leq w, v \leq x$, постоји монотона функција q за коју важи $q(00) = u, q(01) = w, q(10) = x, q(11) = v$. С друге стране, важи $u = q(00) \leq q(01), q(10) \leq q(11) = v$. Број функција које задовољавају овај услов једнак је квадрату броја елемената који су по поретку између u и v, односно $(r_e[u,v])^2$. На крају, дужина низа M_n добија се сумирањем квадрата ових вредности из матрице r_e.

Овај алгоритам омогућава ефикасно израчунавање броја елемената скупа M_n на основу претходно израчунатог скупа M_{n-2}. Алгоритам не рачуна конкретне елементе скупа M_n, већ само одређује колико скуп има елемената на основу поретка у претходном скупу. Ово може допринети смањењу временске сложености алгоритма у односу на први алгоритам, под претпоставком да није потребно одредити скуп M_n.

Алгоритам 2: Други алгоритам

Улаз : $m - |M_{n-2}|$ величина скуп M_{n-2}
$M - [1, ..., m_{n-2}]$ скуп M_{n-2}
Излаз: $m_n - |M_n|$ величина скупа M_n

1 израчунавање матрице r
2 израчунавање матрице $re = rr$
3 $m_n = \sum_{i,j \in M_{n-1}} r_e(i,j)^2$

Имплементација овог алгоритма у $C++$ налази се у одељку 4.2.

Временска сложеност рачунања матрице r је $O(2^{n-2} \cdot m^2)$ јер се састоји од две угнежђене петље дужине од $m = d(n-2)$ и провере монотоности парова елемената скупа M_{n-2}. Рачунање матрице r_e се састоји од три угнежђене петље чија је временска сложеност $O(m^3)$, а провера монотоности парова

елемената скупа M_{n-1} има временску сложеност $O(2^{n-2})$, тако да је сложеност овог дела $O(2^{n-2} \cdot m \cdot m \cdot m) = O(2^n \cdot m^3)$.

Поред тога, алгоритам се састоји и од петље у којој се рачуна $m_n = d(n)$. Због тога што је петља двострука, временска сложеност за ову радњу је $O(m^2)$.

Укупна временска сложеност алгоритма је једнака збиру временских сложености операција $O(2^n \cdot m^2) + O(2^n \cdot m^3) + O(m^2) = O(2^n \cdot m^3)$, што представља сложеност рачунања матрице r_e.

Овај алгоритам садржи рачунање две матрице, тако да његова меморијска сложеност зависи од величина тих матрица. Матрица r има $m \cdot m$ елемената, тако да је за чување ове матрице потребно алоцирати m^2 места у меморији. Као и матрица r, матрица r_e има исти број елеманата, тако да је и за чување ове матрице потребно исто алоцирати m^2 елемената. С обзиром на то, меморијска сложеност овог алгоритма је $O(m^2)$.

У табели 3.2 је представљен број битовских операција потребних за рачунање $d(5)$, $d(6)$, $d(7)$ и $d(8)$. Дијаграм 3.2 представља зависност логаритма броја битовских операција другог алгоритма од n.

n	број битовски операција
5	64000
6	75866112
7	$1.39421409 \times 10^{13}$
8	$3.07037445 \times 10^{22}$

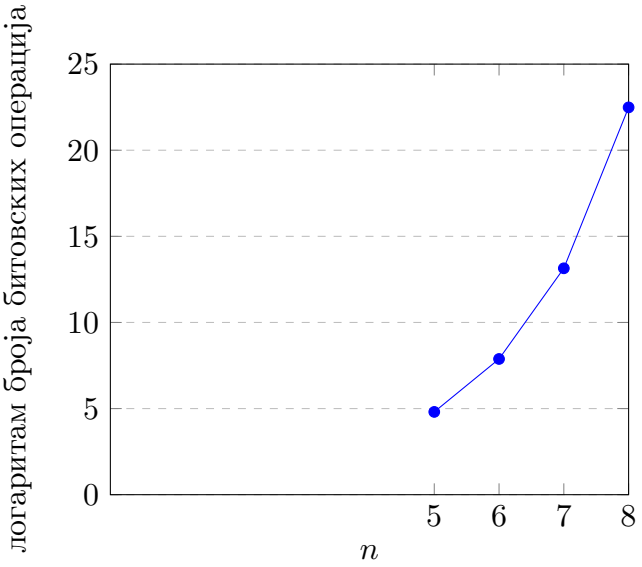
Табела 3.2: Број битовски операција другог алгоритма

3.4 Трећи алгоритам

За извођење трећег алгоритма [4] користи се, као и код другог алгоритма, Последица 3.1.2, али за вредност параметра $k = 4$.

Последица 3.4.1. *Постоји бијекција између елемената из M_n и скупа функција које пресликавају B^4 у M_{n-4}.*

За сваких шест елемената a, b, c, d, e, f из скупа M_{n-4}, треба да се преброји колико функција g задовољава: $g(0011) = a$, $g(0101) = b$, $g(1001) = c$, $g(0110) = d$, $g(1010) = e$, $g(1100) = f$. Другим речима, броји се на ко-



Слика 3.2: Зависност логаритма броја битовских операција другог алгоритма од n

лико начина могу да се изаберу вредности g за остале елементе из скупа B^4. Структура B^4 скупа представљена је на слици 3.3.

Потребно је најпре израчунати на колико начина могу да се изаберу вредности за горње елементе (1111), (0111), (1011), (1101), (1110), слика 3.3.

Вредност $g(1111)$ треба да буде већа или једнака од сваког од шест фиксираних елемената. Другим речима, овај елемент може бити изабран између вредности већих или једнаких од $a+b+c+d+e+f$. Затим, вредности $g(0111)$, $g(1011)$, $g(1101)$, $g(1110)$ могу се изабрати независно једна од друге.

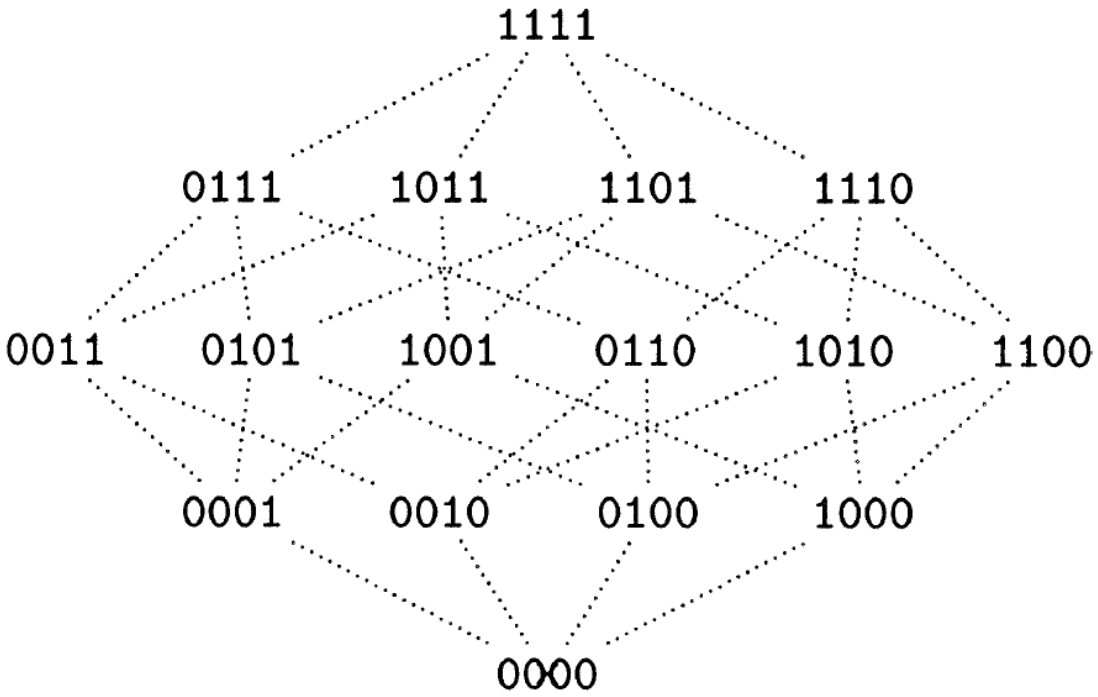
Елемент $g(0111)$ мора бити већи или једнак $a+b+d$, односно треба да важи неједнакост $a+b+d \leq g(0111) \leq g(1111)$. Из овога следи да вредност $g(0111)$ може бити одабрана на $r_e[a+b+d, 1111]$ начина.

Сличне неједнакости треба да важе и за остале елементе $g(1011)$, $g(1101)$, $g(1110)$, па се они могу изабрати на $r_e[x, 1111]$, где x представља дисјункцију одговарајућа три од елемената a, b, c, d, e, f, тако да њихова дусјункција буде мања или једнака од елемента x.

Број начина на који g може да се прошири на горње елементе једнак је:

$$H = \sum_{u \geq a+b+c+d+e+f} r_e[a+b+d, u] +$$

$$r_e[a+c+e, u] + r_e[b+c+f, u] + r_e[d+e+f, u]$$



Слика 3.3: Структура B^4.

Сличан поступак се понавља и за доње елемете скупа B^4, $g(0000)$, $g(0001)$, $g(0010)$, $g(0100)$, $g(1000)$. Број начина на који g може да се прошири на доње елементе се може израчунати на основу израза:

$$h = \sum_{v \leq a \cdot b \cdot c \cdot d \cdot e \cdot f} r_e[v, a \cdot b \cdot c] +$$

$$r_e[v, a \cdot d \cdot e] + r_e[v, b \cdot d \cdot f] + r_e[v, c \cdot e \cdot f]$$

Имплементација овог алгоритма у $C++$ налази се у одељку 4.3.

Временска сложеност која се односи на рачунање матрица r и r_e је идентична као код другог алгоритма. Разлика између другог и трећег алгоритма је у томе што се у другом користи M_{n-2}, док у трећем M_{n-4}, при чему је скуп M_{n-2} доста већи од скупа M_{n-4}. Сложеност рачунања матрице r је $O(2^n \cdot m^2)$, а за рачунање матрице r_e је сложеност $O(2^n \cdot m^3)$.

У делу где се рачунају вредности за горње елементе H и доње елементе h, постоји по седам угнежђених петљи, као и провера монотоности елемената, тако да је сложеност овог дела $O(2^{n-4} \cdot m \cdot m \cdot m \cdot m \cdot m \cdot m \cdot m) = O(2^n \cdot m^7)$. Дакле израчунавање m_n овим алгоритмом је временске сложености $O(2^n \cdot m^7)$.

Алгоритам 3: Трећи алгоритам

Улаз : $M = [1, ..., m_{n-4}]$ скуп M_{n-4}
Излаз: $m_{new} = |M_n|$ величина скупа M_n

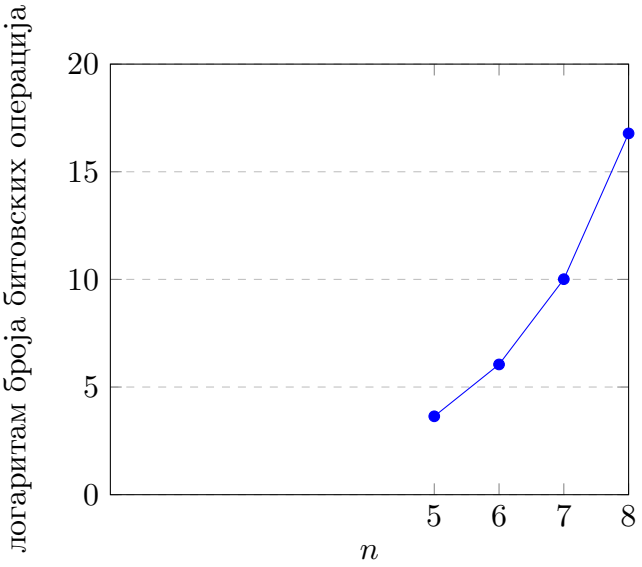
израчунати матрицу r
израчунати матрицу r_e
$m_{new} = 0$ `// величина новог низа`
for $a, b, c, d, e, f \in M_{n-4}$ **do**
  $H = 0$ `// вредност за горње елементе`
  $h = 0$ `// вредност за доње елементе`
  for $u \in M_{n-4}, u \geq a + b + c + d + e + f$ **do**
    $H = H + re[a+b+d, u] * re[a+c+e, u] * re[b+c+f, u] * re[d+e+f, u]$
  end for
  for $v \in M_{n-4}, v \leq a * b * c * d * e * f$ **do**
    $h = h + re[v, a * b * c] * re[v, a * d * e] * re[v, b * d * f] * re[v, c * e * f]$
  end for
  $m_n = m_n + H * h$
end for

Овај алгоритам се састоји од рачунања две матрице и броја m_n, тако да меморијска сложеност зависи од величина те две матрице. Матрица r има m^2 елемената тако да је рачунање ове матрице меморијске сложености $O(m^2)$. Као и матрица r, матрица r_e има исти број елеманата, тако да је и рачунање ове матрице исто меморијске сложености $O(m^2)$. Из тога следи да је и укупна меморијска сложеност алгоритма $O(m^2)$.

У табели 3.3 је представљен број битовских операција потребних за рачунање $d(5)$, $d(6)$, $d(7)$ и $d(8)$. Дијаграм 3.4 представља зависност логаритма броја битовских операција трећег алгоритма од n.

n	број битовски операција
5	4374
6	1119744
7	1.024×10^{10}
8	6.0434503×10^{16}

Табела 3.3: Број битовски операција трећег алгоритма



Слика 3.4: Зависност логаритма броја битовских операција трећег алгоритма од n

3.5 Поређење алгоритама

У табели 3.4 је приказан број битовских операција приликом извршавања сва три алгоритма.

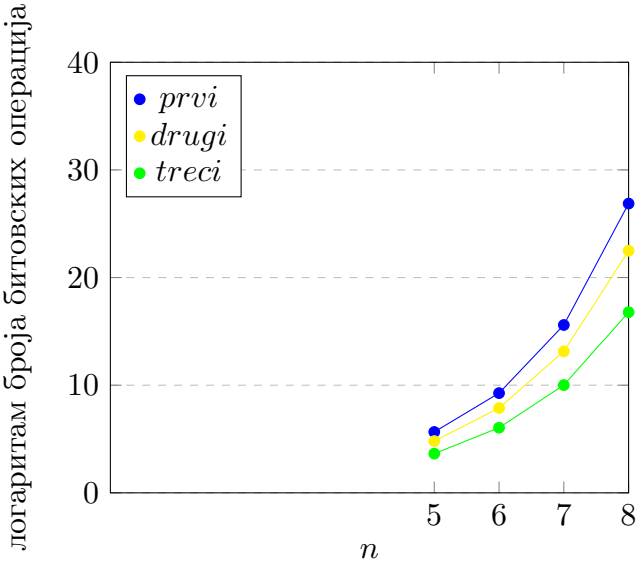
Посматрајући табелу 3.4 и дијаграм 3.5 долази се до закључка да најлошију временску сложеност има први алгоритам, иако његова асимптоматска временска сложеност делује најбоље. Ово следи из тога што је улазни скуп M_{n-1} код овог алгоритма највећи. После првог алгоритма, иде други, па трећи алгоритам. Асимптоматска сложеност другог алгоритма делује боље од асимптоматске сложености трећег алгоритма, али скуп M_{n-2} је већи од скупа M_{n-4}.

У односу на временску сложеност, меморијска сложеност нам показује мало другачији резултат. Уколико није потребно чувати елементе новог скупа, најбољу меморијску сложеност има први алгоритам, константну меморијску сложеност. Потом следи трећи алгоритам јер његове матрице r и r_e заузимају мање меморијског простора од матрица r и r_e другог алгоритма, због величине улазног низа.

Општи закључак је да промена параметра k из последице 3.1.2 доводи до побољшања алгоритама.

	први	други	трећи
5	451584	64000	4374
6	1.83908995×10^{9}	75866112	1119744
7	$3.92212009 \times 10^{15}$	$1.39421409 \times 10^{13}$	1.024×10^{10}
8	$7.46328238 \times 10^{26}$	$3.07037445 \times 10^{22}$	6.0434503×10^{16}

Табела 3.4: Број битовски операција алгоритама



Слика 3.5: Зависност логаритма броја битовских операција сва три алгоритма од n

Глава 4

Реализација алгоритама и резултати

Описани алгоритми из поглавља 3, су реализовани у програмском језику $C++$ и извршавани на рачунару са $Intel\ Core\ i9-10850K$ процесором. За потребе временског мерења извршавања алгоритама коришћена је $C++$ библиотека *chrono*.

4.1 Први алгоритам

Имплементација

```
#include <iostream>
#include <vector>
#include <bitset>
#include <ctime>
#include <iomanip>
#include <string>
#include <chrono>

using namespace std;
/**
 * pomocna funkcija za proveru poretka bitova
 * @param a - element niza Mn-1
 * @param b - element niza Mn-1
```

```
 * @return oznaka da li su elementi u poretku
 */
bool check_bits(string a, string b) {
    for (int i = 0; i < a.length(); i++) {
        if (int(a[i]) > int(b[i])) {
            return false;
        }
    }
    return true;
}

/**
 * funkcija prvog algoritma
 * @param M - niz Mn-1
 * @return duzinu niza Mn
 */
template<size_t N>
int first_algorithm(const
vector<bitset<N>>& M) {
    typedef std::chrono::high_resolution_clock Clock;
    typedef std::chrono::milliseconds milliseconds;
    Clock::time_point t0 = Clock::now();
    unsigned long long int mn = 0;
    size_t m = M.size();

    // prolazak parovima kroz Mn-1
    for (size_t i = 0; i < m; ++i) {
        for (size_t j = 0; j < m; ++j) {
        // provera da li elementi odrzavaju poredak
        if(check_bits(M[i].to_string(),M[j].to_string()))
            {
                mn++;
            }
        }
    }
```

```
    // ispisivanje vremena izvrsavanja
    Clock::time_point t1 = Clock::now();
    milliseconds duration_time =
    chrono::duration_cast<milliseconds>(t1 - t0);
    cout << duration_time.count() << endl;
    return mn;
}


int main() {
    // primer poziva funkcije za n=1
    vector<bitset<1>> bitsetVector1 = {0b0, 0b1};
    unsigned long long int result1 =
    first_algorithm(bitsetVector1);
    cout << result1 << endl;

    return 0;
}
```

Пример извршавања

Први улаз за овај алгоритам је $M = \{0, 1\} = M_0$. Другим речима, први позив функције имплементиране у $C++$ је first_algorithm($\{0b0, 0b1\}$). За рачунање сваког следећег Дедекиндовог броја може се користи претходно добијен резултат, односно добијени низ и дужина тог низа.

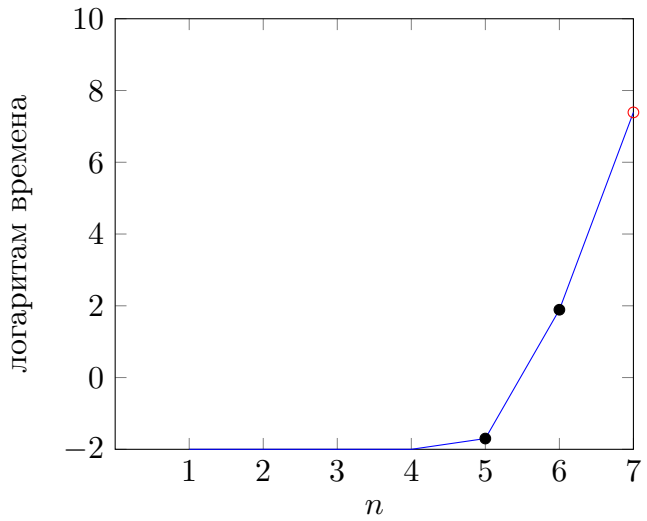
Резултати мерења

Примењена је методологија мерења времена за извршавање овог алгоритма у $C++$ уз помоћ библиотеке *chrono*. Резултати добијени овим експериментом су наведени у табели 4.1. Дијаграм односа зависности логаритма потребног времена за израчунавање од n приказан је на слици 4.1.

Ово време извршавања у складу је са предвиђеном временском сложеношћу алгоритма, која је дискутована у секцији 3.2, $O(2^{2^{n-1}} \cdot m^2)$, где је $m = d(n-1)$ број елемената улазног низа. Када се број елемената улазног

n	$d(n)$	време
1	3	0.000s
2	6	0.000s
3	20	0.000s
4	168	0.000s
5	7581	0.020s
6	7828354	77.364s
7	2414682040998	$\approx$ 24367836s

Табела 4.1: Време извршавања првог алгоритма



Слика 4.1: Зависност логаритма времена извршавања првог алгоритма од n

скупа M_{n-1} повећа, време извршавања знатно расте, што се може запазити из дијаграма 4.1. Време извршавања знатно је порасло за $n = 6$, за које је време извршавања било 77.364 секунди. Ово показује да је алгоритам ефикасан само за мале вредност n.

На основу описаног алгоритма и времена извршавања за првих шест елемената, може се проценити потребно времена за израчунавање седмог Дедекиндовог броја.

На основу тога, преко пропорције односа времена и броја елемената, може се израчунати колико елемената може да се добије у једној секунди. Затим, тај податак се може употребити да се апроксимира време потребно за израчунавање седмог броја. Ако алгоритам израчунава приближно исти број елемената у секунди као у претходном тесту за $n = 6$, онда је тај број око

101188 елемената у секунди. Применом овог односа на величину седмог Дедекиндовог броја, може се апроксимирати време израчунавања на приближно 23863324 секунде, што је око 276 дана, односно око 9 месеци.

4.2 Други алгоритам

Имплементација

```
#include <iostream>
#include <vector>
#include <bitset>
#include <chrono>
#include <algorithm>
#include <typeinfo>
#include <string>

using namespace std;

/**
 * pomocna funkcija za proveru poretka bitova
 * @param a - element niza Mn-2
 * @param b - element niza Mn-2
 * @return oznaka da li su elementi u poretku
 */
bool check_bits(string a,string b) {
    for (int i = 0; i < a.length(); i++) {
        if (int(a[i]) > int(b[i])) {
            return false;
        }
    }
    return true;
}

/**
 * pomocna funkcija za racunanje matrice re
```

```
 * @param M - niz Mn-2
 * @return matrica re
 */
template<size_t N>
vector<vector<int>> re_help(vector<bitset<N>> M) {
    vector<vector<int>> r;
    size_t m = M.size();
    // racunanje matrice r
    for (int u = 0; u < m; u++) {
        vector<int> temp;
        for (int v = 0; v < m; v++) {
            if(
            check_bits(M[u].to_string(),M[v].to_string()))
            {
                temp.push_back(1);
            }
            else {
                temp.push_back(0);
            }
        }
        r.push_back(temp);
    }

    // racunanje matrice re
    vector<vector<int>> re(M.size(), vector<int>(M.size(), 0));
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < m; j++) {
            for (int k = 0; k < m; k++) {
                re[i][j] += r[i][k] * r[k][j];
            }
        }
    }

    return re;
}
```

```
/**
 * funkcija drugog algoritma
 * @param M - niz Mn-2
 * @return duzina niza Mn
 */
template<size_t N>
int second_algorithm(
const std::vector<std::bitset<N>>& M) {
    typedef std::chrono::high_resolution_clock Clock;
    typedef std::chrono::milliseconds milliseconds;
    Clock::time_point t0 = Clock::now();
    unsigned long long int int mn = 0;
    size_t m = M.size();

    vector<vector<int>> re = re_help(M);

    for (size_t i = 0; i < m; i++) {
        for (size_t j = 0; j < m; j++) {
            mn += re[i][j] * re[i][j];
        }
    }

    Clock::time_point t1 = Clock::now();
    milliseconds duration_time =
    std::chrono::duration_cast<milliseconds>(t1 - t0);
    std::cout << duration_time.count() << std::endl;
    return mn;
}

int main() {
    // primer poziva funkcije za n=2
    std::vector<std::bitset<1>> bitsetVector1 = {0b0,0b1};
    unsigned long long int result1 =
    second_algorithm(bitsetVector1);
```

```
    std::cout << result1 << std::endl;

    return 0;
}
```

Пример извршавања

За рачунање другог Дедекиндовог броја $d(2)$, улаз за други алгоритам је $m = 2, M = \{0, 1\} = M_0$, односно први позив функције имплементиране у $C++$ је second_algorithm($\{0b0, 0b1\}$). Затим, за рачунање трећег елемента улаз је $m = 3 = d(1)$, $M = \{00, 01, 11\} = M_1$, односно позив функције је second_algorithm($\{0b0, 0b1, 0b11\}$).

Резултати мерења

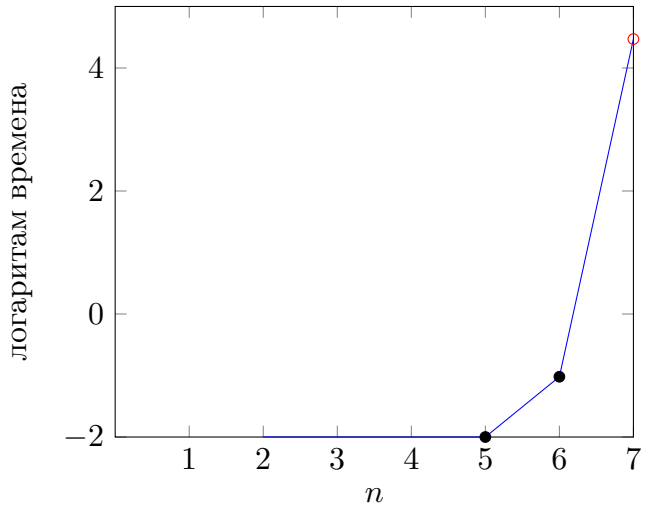
Резултати тестова показали су да је време извршавања другог алгоритма изузетно ефикасно за првих шест Дедекиндових бројева. Најдуже време извршавања било је потребно за израчунавање шестог Дедекиндовог броја, али и то је било у прихватљивом опсегу (0.095 секунди). Резултати су наведени у табели 4.2.

n	$d(n)$	$vreme$
2	6	0.000s
3	20	0.000s
4	168	0.000s
5	7581	0.000s
6	7828354	0.095s
7	2414682040998	$\approx$ 29303s

Табела 4.2: Време извршавања другог алгоритма

Дијаграм односа зависности логаритма потребног времена за израчунавање од n приказан је на слици 4.2.

За израчунавање седмог Дедекиндовог број применом другог алгоритма, процењује се да је потребно више времена. По резултатима наведеним у табели 4.2, може се израчунати однос времена и броја елемената, односно може се апроксимирати број елемената, које овај алгоритам рачуна у једној секунди $7828354 \div 0.095 \approx 82403726$. Затим, на основу тога и вредности



Слика 4.2: Зависност логаритма времена извршавања другог алгоритма од n

седмог Дедекиндов број процењује се потребно време за његово рачунање, а то је $2414682040998 \div 82403726 \approx 29303s \approx 8$сати.

4.3 Трећи алгоритам

Имплементација

```
#include <iostream>
#include <vector>
#include <bitset>
#include <chrono>
#include <algorithm>
#include <typeinfo>
#include <string>

using namespace std;

/**
 * pomocna funkcija za proveru poretka bitova
 * @param a - element niza Mn-4
 * @param b - element niza Mn-4
```

```
 * @return oznaka da li su elementi u poretku
 */
bool check_bits(string a,string b) {
    for (int i = 0; i < a.length(); i++) {
        if (int(a[i]) > int(b[i])) {
            return false;
        }
    }
    return true;
}

/**
 * pomocna funkcija za racunanje re matrice
 * @param M - niz Mn-4
 * @return matrica re
 */
template<size_t N>
vector<vector<unsigned long long int>>
re_help(vector<bitset<N>> M) {
    vector<vector<int>> r;
    size_t m = M.size();
    // racunanje matrice r
    for (int u = 0; u < m; u++) {
        vector<int> temp;
        for (int v = 0; v < m; v++) {
            if (check_bits(M[u].to_string(),
                M[v].to_string())) {
                temp.push_back(1);
            }
            else {
                temp.push_back(0);
            }
        }
        r.push_back(temp);
    }
```

```
    // racunanje matrice re
    vector<vector<int>> re(M.size(),
                           vector<int>(M.size(), 0));
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < m; j++) {
            for (int k = 0; k < m; k++) {
                re[i][j] += r[i][k] * r[k][j];
            }
        }
    }
    return re;
}

/**
 * funkcija treceg algoritma
 * @param M - niz Mn-4
 * @return matrica re
 */
template<size_t N>
int third_algorithm(const std::vector<std::bitset<N>>& M) {
    typedef std::chrono::high_resolution_clock Clock;
    typedef std::chrono::milliseconds milliseconds;
    Clock::time_point t0 = Clock::now();
    unsigned long long int mn = 0;
    size_t m = M.size();

    vector<vector<int>> re = re_help(M);

    for (bitset<N> a : M) {
    for (bitset<N> b : M) {
    for (bitset<N> c : M) {
    for (bitset<N> d : M) {
    for (bitset<N> e : M) {
    for (bitset<N> f : M) {
        int H = 0;
```

```
int h = 0;
\\ racunanje za gornje elemente
for (bitset<N> u : M) {
    if (
    check_bits((a | b | c | d | e | f).to_string(),
    u.to_string())) {
        int index_u = distance(M.begin(),
        find(M.begin(), M.end(), u));
        int index_a_b_d = distance(M.begin(),
        find(M.begin(), M.end(), (a | b | d)));
        int index_a_c_e = distance(M.begin(),
        find(M.begin(), M.end(), (a | c | e)));
        int index_b_c_f =distance(M.begin(),
        find(M.begin(), M.end(), (b | c | f)));
        int index_d_e_f = distance(M.begin(),
        find(M.begin(), M.end(), (d | e | f)));
        H += re[index_a_b_d][index_u]
        * re[index_a_c_e][index_u]
        * re[index_b_c_f][index_u]
        * re[index_d_e_f][index_u];
    }
}
\\ racunanje za donje elemente
for (bitset<N> v : M) {
    if (check_bits(v.to_string(),
    (a & b & c & d & e & f).to_string())) {
        int index_v = distance(M.begin(),
        find(M.begin(), M.end(), v));
        int index_a_b_c = distance(M.begin(),
        find(M.begin(), M.end(), (a & b & c)));
        int index_a_d_e = distance(M.begin(),
        find(M.begin(), M.end(), (a & d & e)));
        int index_b_d_f = distance(M.begin(),
        find(M.begin(), M.end(), (b & d & f)));
        int index_c_e_f = distance(M.begin(),
```

```
                find(M.begin(), M.end(), (c & e & f)));
                h += re[index_v][index_a_b_c]
                * re[index_v][index_a_d_e]
                * re[index_v][index_b_d_f]
                * re[index_v][index_c_e_f];
            }
        }
     mn += H * h;
 }}}}}}

    Clock::time_point t1 = Clock::now();
    milliseconds duration_time =
    std::chrono::duration_cast<milliseconds>(t1 - t0);
    std::cout << duration_time.count() << std::endl;
    return mn;
}

int main() {
    // primer poziva funkcije za n=4
    std::vector<std::bitset<1>> bitsetVector1 = {0b0,0b1};
    unsigned long long int result1 = third_algorithm(bitsetVector1);
    std::cout  << result1 << std::endl;

    return 0;
}
```

Пример извршавања

За израчунавање четвртог Дедекиндовог броја $d(4)$ уз помоћ трећег алгоритма улаз за овај алгоритам је $m = 2$, $M = \{0, 1\} = M_0$, односно први позив функције имплементиране у $C++$ је third_algorithm($\{0b0, 0b1\}$). За рачунање петог елемента позив је third_algorithm($\{0b0, 0b1, 0b11\}$).

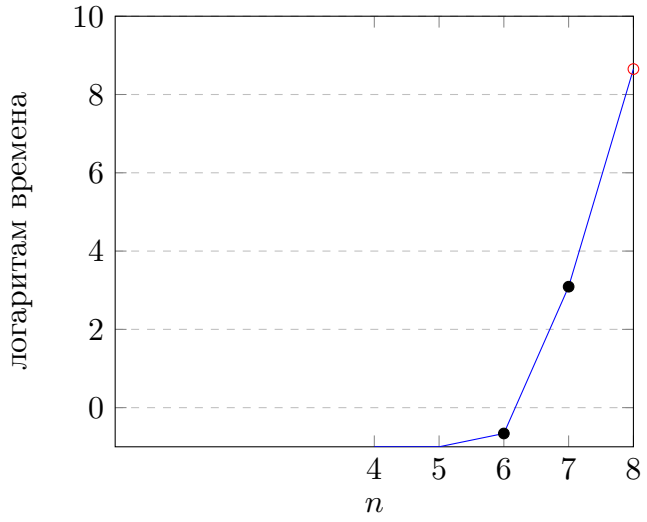
Резултати мерења

Као и за претходне алгоритме извршено је тестирање за колико времена се извршава овој алгоритам у $C++$ приликом израчунавања првих седам елемената. Добијени резултати су приказани у табели 4.3.

n	$d(n)$	$vreme$
4	168	0.000s
5	7581	0.000s
6	7828354	0.220s
7	2414682040998	$1234.941s$
8	56130437228687557907788	$\approx 449661480s$

Табела 4.3: Време извршавања трећег алгоритма

Дијаграм зависности логаритма потребног времена за израчунавање од n приказан је на слици 4.3.



Слика 4.3: Зависност логаритма времена извршавања трећег алгоритма од n

Да се израчуна осми елемент потребно је много више времена, тако да се само може проценити потребно времена за рачунање овог елемета. По резултатима наведеним у табели 4.3 може се одредити однос величина броја m_7 и m_8, пошто брзина израчунавања зависи од потребног времена за рачунање матрице r_e и каснијих провера као и рачунања вредности H и h.

По конструкцији алгоритма имамо пролаз кроз шесторке елемената скупа M_{n-4}. За рачунање седмог Дедикиндовог броја имамо $20^6 = 64000000$ комби-

нација кроз које треба да се прође. На основу израчунатог времена за седми елемент, може се проценити колико времена је потребно по једној комбинацији:

$$1234.941 \div 64000000 = 0.00002.$$

Затим, на основу тога колико износи осми Дедекиндов број, процењује се потребно време за његово рачунање а то је

$$168^6 \cdot 0.00002s \approx 449661480s.$$

4.4 Поређење алгоритама

Времена извршавања за сва три алгоритма се могу видети у табели 4.4. Може се запазити да сва три алгоритма имају занемарљиво мало време извршавања за првих пет бројева.

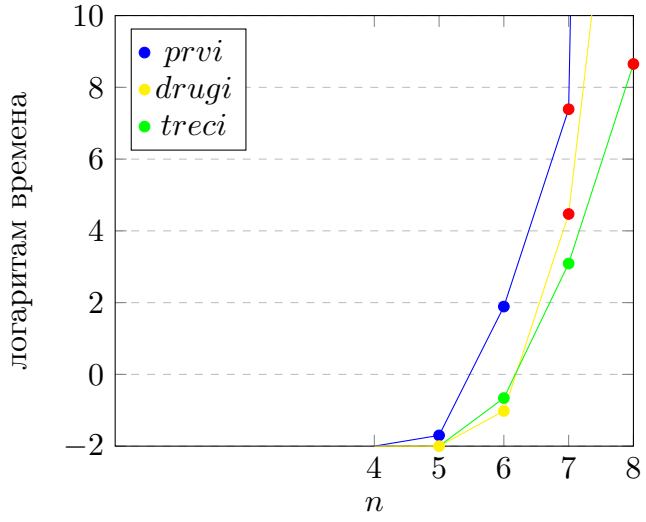
Други алгоритам не може израчунати први број, јер рачуна M_n од M_{n-2}. Трећи алгоритам не може израчунати прва три броја, јер рачуна M_n од M_{n-4}. Првом алгоритму за шести број је потребно знатно више времена у односу на друга два алгоритма. Други и трећи имају слично време за шести елемент. За седми елемент други алгоритам извршавао би се око осам сати. Трећи алгоритам успева да израчуна седми број за око двадесет минута.

На основу дијаграма 4.4, може се донети сличан закључак као и из табеле 4.4. Може се приметити да дијаграм првог алгоритма нагло расте после петог елемента. Други и трећи имају сличну трајекторију све до седмог елемента, када за други алгоритам имамо само апроксимацију потребног времена (црвени круг на дијаграму представља апроксимацију).

Када упоредимо дијаграм 4.4 са дијаграмом 3.5, закључујемо да је број битовских операција у алгоритмима и њихов међусобни однос у складу са временом извршавања алгоритма. Као и код временске сложености, и за времена извршавања трећи алгоритам показује најбоље резултате, затим следи други, а најлошији је први алгоритам.

n	$d(n)$	први	други	трећи
1	3	$0.000s$	/	/
2	6	$0.000s$	$0.000s$	/
3	20	$0.000s$	$0.000s$	/
4	168	$0.000s$	$0.000s$	$0.000s$
5	7581	$0.020s$	$0.000s$	$0.000s$
6	7828354	$79.400s$	$0.095s$	$0.220s$
7	2414682040998	$\approx 24367836s$	$\approx 29303s$	$1234.941s$
8	5613043722868755790788	/	/	$\approx 449661480s$

Табела 4.4: Време извршавања сва три алгоритма



Слика 4.4: Зависност логаритма времена извршавања сва три алгоритма од n

Глава 5

Закључак

У овом раду, разматрају се монотоне Булове функције и представљени су алгоритми за њихово пребројавање, односно за решавање Дедекиндовог проблема. Имплементирана су и анализирана три алгоритма за пребројавање монотоних Булових функција из рада [4].

На основу извршених имплементација и мерења времена извршавања, уочено је да се са прва два алгоритма успешно израчунава шест Дедекиндових бројева. Додатно, трећи алгоритам је показао могућност израчунавања седмог Дедекиндовог броја у релативно кратком временском периоду. Такође, рад садржи и приближну процену времена потребног за израчунавање седмог броја помоћу прва два алгоритма, односно осмог броја уз помоћ трећег алгоритма.

Интересантно би било реализовати усавршен алгоритам којим је у раду [5] израчунат девети Дедекиндов број.

Још увек се не зна довољно ефикасан алгоритам који може да израчуна Дедекиндове бројеве за $n > 9$. Ова изазовна област оставља отворене могућности за будућа истраживања и развој нових техника које би се суочиле са овим проблемом.

Библиографија

[1] H. Beckurts and R. Dedekind. *Über Zerlegungen von Zahlen durch ihre grössten gemeinsamen Theiler*. Springer, 1897.

[2] R. Church. Numerical analysis of certain free distributive structures. 1940.

[3] R. Church. Enumeration by rank of the elements of the free distributive lattice with seven generators. *Not. Amer. Math. Soc 12, 724*, 1965.

[4] R. Fidytek, A. W Mostowski, R. Somla, and A. Szepietowski. Algorithms counting monotone boolean functions. *Information Processing Letters, 79(5), 203–209*, 2001. Lemma 1, Corollary 2, Corollary 6.

[5] C. Jäkel. A computation of the ninth dedekind number. *arXiv preprint arXiv:2304.00895*, 2023.

[6] S. Jukna et al. *Boolean function complexity: advances and frontiers*, volume 5. Springer, 2012.

[7] D. Kleitman. On dedekind's problem: The number of monotone boolean functions. *Proceedings of the American Mathematical Society, 21(3), 677–682*, 1969.

[8] M. Ward. Note on the order of the free distributive lattice. In *Bulletin of the American Mathematical Society, 52(5), 423–423*. AMER MATHEMATICAL SOC 201 CHARLES ST, PROVIDENCE, RI 02940-2213, 1946.

[9] I. Wegener. *The complexity of Boolean functions*. John Wiley & Sons, Inc., 1987.

[10] D. Wiedemann. A computation of the eighth dedekind number. *Order*, 8:5–6, 1991.

Биографија аутора

Александра Дивљаковић рођена је 26.7.1996. у Лозници. Основну школу завршила је 2011. у Крупњу. Исте године уписује Гимназију у Крупњу, општи смер. Гимназију завршава 2015. године и уписује Математички факултет Универзитита у Београду, смер Рачунарство и информатика. Основне академске студије завршава 2020. и исте године уписује и мастер студије на истом смеру.