УНИВЕРЗИТЕТ У БЕОГРАДУ
МАТЕМАТИЧКИ ФАКУЛТЕТ

Милица Михаиловић

ПРОБАБИЛИСТИЧКЕ СТРУКТУРЕ ПОДАТАКА ЗА ПРИБЛИЖНУ ПРОВЕРУ ПРИПАДНОСТИ

мастер рад

Београд, 2026.

Ментор:

др Весна Маринковић, ванредни професор
Универзитет у Београду, Математички факултет

Чланови комисије:

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

др Мирко Спасић, доцент
Универзитет у Београду, Математички факултет

Датум одбране:

Захваљујем се својој менторки др Весни Маринковић на корисним саветима и издвојеном времену током израде рада. Рад посвећујем породици и пријатељима који су пружали велику подршку током студирања.

Наслов мастер рада: Пробабилистичке структуре података за приближну проверу припадности

Резиме: Пробабилистичке структуре података представљају класу структура података које се одричу потпуне прецизности ради постизања боље временске и просторне ефикасности. Посебну класу пробабилистичких структура података представљају структуре података за ефикасну проверу припадности елемената скупу података уз постизање високе стопе тачности. При том је могуће добити лажно позитиван, али не и лажно негативан резултат. Ове врсте структура података постижу добре перформансе техником хеширања (једном или већим бројем хеш функција), тако што уместо самих података чувају њихове дигиталне отиске. Најпознатији представници ових структура су Блумов филтер, филтер количника и филтер кукавице. С обзиром на то да су меморијски веома ефикасне и да омогућавају брзу обраду података у реалном времену, ове структуре података су погодне за коришћење у системима са великим скуповима података, за анализу мрежног саобраћаја, у великим базама података.

У овом раду биће описане и имплементиране наредне структуре података: Блумов филтер, филтер количника и филтер кукавице. Биће извршена њихова упоредна експериментална евалуација као и поређење са класичним структурама података којима се проблем провере припадности елемената скупу података стандардно решава.

Кључне речи: алгоритми и структуре података, пробабилистичке структуре података, Блумов филтер, филтер кукавице, филтер количника

Садржај

Глава 1

Увод

Решавање *проблема провере припадности* је један од основних задатака структура података који подразумева утврђивање да ли елемент припада скупу података. Један од традиционалних приступа решавању овог проблема је претрага скупа података. Елементи скупа могу бити сачувани у погодној структури података, као што је хеш табела. *Хеш табеле* (енг. hash tables) су структуре података које чувају парове (кључ, вредност) у низу слотова. *Хеш функције* (енг. hash functions) на основу датог кључа рачунају вредност позиције слота унутар хеш табеле у који ће бити смештен пар (кључ, вредност). Ова вредност се другачије назива *хеш код* (енг. hash code). Када желимо да пронађемо елемент са датом вредношћу кључа у хеш табели, најпре рачунамо вредност хеш функције за дати кључ, а затим проверавамо да ли се елемент налази на израчунатој позицији у хеш табели. Постоји могућност да вредност хеш функције два различита кључа буде једнака. Ова појава се назива *хеш колизија* (енг. hash collision). Постоји неколико начина за превазилажење овог проблема. Једно од могућих решења је коришћење секундарних структура података. Наиме, елементи хеш табеле могу бити показивачи на листе парова (кључ, вредност). Алтернативно решење је коришћење *отвореног адресирања* (енг. open addressing). Отворено адресирање представља поступак којим се приликом уметања елемената због појаве колизије тражи алтернативна слободна позиција у хеш табели. Постоји неколико имплементација отвореног адресирања. Једна од њих је *хеширање кукавице* (енг. cuckoo hashing) о ком ће бити више речи у наставку. Претраживањем хеш табеле можемо добити одговор на питање да ли елемент припада скупу. Временска сложеност претраге елемента у хеш табели у најгорем случају износи $O(n)$, где је n број

елемената који се налази у хеш табели. Такође, представљање великих скупова података у оквиру структура података може бити меморијски захтевно, па чак и неизводљиво у неким случајевима. Алтернативни приступ у решавању проблема припадности је коришћење *пробабилистичких структура података* (енг. probabilistic data structures).

1.1 Мотивација

Поједини веб прегледачи (енг. web browsers) користе пробабилистичке структуре података за проверу злонамерних УРЛ-ова (енг. malicious URLs). Када корисник унесе УРЛ адресу потребно је брзо проверити да ли та адреса садржи злонамеран софтвер и обавестити корисника о евентуалној опасности. Очигледно решење овог проблема би било чување свих злонамерних адреса у једној датотеци коју претражујемо када год корисник унесе УРЛ адресу. Међутим, ово је практично немогуће због великог броја адреса. Потребна нам је структура података која неће расти линеарно за свако додавање нове УРЛ адресе, као и структура података која добро ради у реалном времену, како корисници не би дуго чекали на одговор.

Пробабилистичке структуре података представљају класу структура података које су изузетно меморијски ефикасне, али не дају увек тачан одговор на питање да ли елемент припада скупу података. Ове структуре података за разлику од класичних структура података не чувају податке у целости, већ само њихове *отиске* (енг. fingerprints). Отисци представљају кратке секвенце битова добијене на основу оригиналних података, што омогућава велику уштеду меморије код коришћења пробабилистичких структура података.

Провера припадности елемената скупу података код пробабилистичких структура података се врши коришћењем различитих техника хеширања. Хеширање је по правилу временски ефикасно, што омогућава брз одговор на питање да ли елемент припада скупу података и чини пробабилистичке структуре података погодним за коришћење у реалном времену.

Међутим, оно што је специфично за ову класу структура података је мо-

гућност добијања нетачног одговора при провери припадности елемента скупу података. Наиме, ове структуре података имају две битне особине:

- могући су *лажно позитивни* (енг. false positive) резултати приликом проверавања да ли неки елемент припада скупу података. Ово би значило да се погрешно тврди да неки елемент припада скупу, иако му заиста не припада.

- нису могући *лажно негативни* (енг. false negative) резултати приликом проверавања да ли неки елемент припада скупу података. Ово би значило да се не може погрешно тврдити да елемент не припада скупу, иако му заиста припада.

Ово на први поглед може деловати лоше, али је веома корисна особина у појединим ситуацијама. Рецимо, у случају злонамерних УРЛ-ова, ако добијемо одговор да се УРЛ налази на листи злонамерних, морамо извршити претрагу листе како бисмо са сигурношћу утврдили да ли је дати УРЛ злонамеран или је добијен лажно позитиван резултат. У супротном, ако добијемо одговор да се УРЛ не налази на листи злонамерних, можемо бити сигурни да је дати УРЛ безбедан за коришћење, због особине пробабилистичких структура података која гарантује одсуство лажно негативних резултата. Ово значајно смањује број непотребних претраживања листе злонамерних УРЛ-ова јер се испоставља да је стопа лажно позитивних резултата при провери припадности елемената неком скупу у већини случајева јако мала. С друге стране, веома непожељно би било да тврдимо да се УРЛ не налази на листи злонамерних УРЛ-ова у случају да се заиста налази на листи.

Због свега наведеног, пробабилистичке структуре података имају широку примену у различитим рачунарским и мрежним системима. Користе се у областима рачунарства као што су безбедност и приватност, статичка анализа програма, базе података, веб претрага, системи за препоручивање и блокчејн протоколи [1]. Њихова временска и меморијска ефикасност их чини веома популарним за коришћење у апликацијама које раде са великим скуповима података који се не могу складиштити нити обрађивати на традиционалним рачунарским системима.

Постоје различите имплементације ових структура података. У овом раду детаљније ће бити приказана три представника ових структура података: Блумов филтер (енг. Bloom filter), филтер количника (енг. Quotient filter) и филтер кукавице (енг. Cuckoo filter).

1.2 Примене

Пробабилистичке структуре података су широко распрострањене захваљујући својим карактеристикама које нарочито долазе до изражаја у системима са великим протоком података.

Претраживачи их користе за безбедну претрагу како би брзо проверили да ли је нека адреса штетна на начин који смо описали у претходном одељку. Конкретно, Гугл Хром (енг. Google Chrome) користи Блумов филтер у ову сврху [2]. Такође, неке платформе користе ове структуре података у оквиру функционалности предлагања садржаја за избегавање приказивања већ виђених садржаја од стране истог корисника.

Ове структуре података значајно смањују број упита упућен базама података. Њиховим коришћењем избегавају се претраге базе података за вредности кључева који се не налазе у табели. Један од познатих система за управљање базама података који користи функционалности пробабилистичких структура података је Апачи Касандра (енг. Apache Cassandra).

Пробабилистичке структуре података у великој мери побољшавају перформансе мрежних и фајл система јер могу да провере да ли се неки садржај већ налази у кеш меморији и на тај начин омогуће избегавање непотребних приступа диску.

Глава 2

Блумов филтер

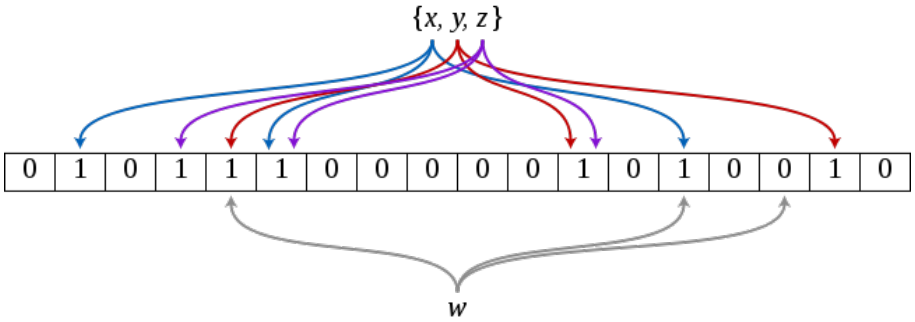
Најједноставнија и најпознатија пробабилистичка структура података која решава проблем припадности је *Блумов филтер*, који је предложио Бартон Блум (енг. Burton Bloom) 1970. године [3]. Блумов филтер је просторно ефикасна пробабилистичка структура података која подржава две основне операције над скупом $D = \{x_1, x_2, \ldots, x_n\}$ од n елемената:

- додавање елемента у скуп;
- проверу да ли се елемент налази у скупу [4].

Ова структура података не чува оригиналне податке, одакле и потиче њена просторна ефикасност.

На слици 2.1 приказан је пример Блумовог филтера. Блумов филтер се састоји од низа битова и одређеног броја хеш функција које се користе за трансформацију и представљање података. Провера припадности елемента скупу који је представљен Блумовим филтером може произвести лажно позитивне, али не и лажно негативне резултате. Тачније, Блумов филтер нам може дати два одговора на питање да ли елемент припада скупу: „дати елемент можда припада скупу" или „елемент дефинитивно не припада скупу" [5]. Стопа лажно позитивних резултата је обично мала, али расте при додавању нових елемената у скуп података. Пораст стопе лажно позитивних резултата се може контролисати одабиром оптималног броја хеш функција или проширивањем низа битова који се користе. Битно је нагласити да временска ефикасност Блумовог филтера не зависи од величине скупа података (у наставку n), нити од величине низа битова (у наставку m), већ од броја

хеш функција (у наставку k) које пресликавају скуп елемената у целобројне вредности од 0 до $m-1$ (индекси низа битова). Уколико претпоставимо да се вредност хеш функција рачуна у константном времену, што је обично случај, временска сложеност додавања нових елемената и провере припадности елемента скупу износи $O(k)$. Истакнимо да Блумов филтер не подржава операцију брисања елемената из скупа података и операцију излиставања свих елемената који припадају скупу података.



Слика 2.1: Блумов филтер

2.1 Иницијализација и додавање елемената

Претпоставимо да је Блумов филтер дужине $m = 10$. Празан Блумов филтер у том случају представља низ од 10 битова чије су вредности постављене на нулу:

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0

Нов елемент x додајемо у Блумов филтер тако што најпре рачунамо вредност свих k хеш функција $\{h_i\}_{i=1,k}$ за дати податак, а затим постављамо вредност битова на позицијама $\{h_i(x)\}_{i=1,k}$ у низу на 1.

Пример 2.1.1. Претпоставимо да имамо две хеш функције h_1 и h_2. Уколико желимо да додамо ниску „banana" скупу података, израчунаћемо вредности $h_1(banana)$ и $h_2(banana)$. Нека је $h_1(banana) = 7$ и $h_2(banana) = 3$. Тада ћемо „укључити" битове у низу на позицијама 3 и 7, тј. поставићемо њихову вредност на 1:

0	1	2	3	4	5	6	7	8	9
0	0	0	1	0	0	0	1	0	0

Након додавања ниске „jabuka" за коју су вредности хеш функција $h_1(jabuka) = 3$ и $h_2(jabuka) = 1$ добијамо следећи низ битова:

0	1	2	3	4	5	6	7	8	9
0	1	0	1	0	0	0	1	0	0

Уочавамо на претходним примерима да су могуће колизије хеш функција тј. могуће је да два различита податка произведу укључивање бита на истој позицији у низу ($h_2(banana) = h_1(apple) = 3$). Управо је могућност доласка до колизије приликом хеширања података узрок пробабилистичке природе Блумовог филтера. Из тог разлога није дозвољено брисање елемената из скупа јер не можемо бити сигурни да ли је неки бит у низу постављен на вредност 1 од стране једног или већег броја елемената. Такође, увек постоји вероватноћа да елемент не припада скупу иако су све вредности битова на одговарајућим позицијама постављене на 1.

2.2 Провера припадности елемента скупу података

Проверу припадности елемента скупу података започињемо на исти начин као и додавање нових елемената. Рачунамо вредности свих хеш функција $\{h_i\}_{i=1,k}$ за елемент x, а затим проверавамо вредности битова у низу на позицијама $\{h_i(x)\}_{i=1,k}$. Уколико је вредност бар једног бита једнака нули, са сигурношћу можемо тврдити да елемент не припада скупу. У супротном, ако сви одговарајући битови имају вредност 1, тада можемо рећи да елемент можда припада скупу. Могући су лажно позитивни резултати, тј. може се десити да сви срачунати битови за неки елемент имају вредност 1, али тај елемент не припада скупу. Појава лажно позитивних резултата приликом провере припадности настаје услед колизија хеш функција.

Пример 2.2.1. Након додавања ниски „banana" и „jabuka", имамо следећи низ битова:

0	1	2	3	4	5	6	7	8	9
0	1	0	1	0	0	0	1	0	0

Желимо да проверимо да ли ниска „malina" припада скупу података. Рачунамо вредност хеш функција h_1 и h_2. Нека је $h_1(malina) = 5$ и $h_2(malina) = 6$. Пошто бит на позицији 5 у низу има вредност једнаку нули, одмах можемо рећи да ниска „malina" не припада скупу, без провере вредности бита на позицији 6.
Проверимо за ниску „jagoda" чије су вредности хеш функција $h_1(jagoda) = 1$ и $h_2(jagoda) = 7$ да ли припада скупу података. На основу низа битова можемо да закључимо да елемент припада скупу пошто одговарајући битови имају вредност 1. Међутим, знамо да ниска „jagoda" не припада скупу јер је нисмо додали у скуп, па ово представља лажно позитиван резултат. Дакле, не можемо са сигурношћу тврдити да ли је неки податак елемент скупа, али када добијемо одговор да елемент не припада скупу у њега можемо бити у потпуности сигурни.

2.3 Одабир оптималних вредности параметара

Испоставља се да је вероватноћа лажно позитивних резултата мала. Она расте с порастом додатих елемената, али се може ублажити одабиром правог броја хеш функција, као и повећањем величине низа битова. Гранични случај представља ситуација када сви битови у низу имају вредност 1 и тада имамо велики број лажно позитивних резултата. Погодним одабиром вредности наведених параметара можемо минимизовати вероватноћу лажно позитивних резултата и искористити пун потенцијал Блумовог филтера.

Претпоставимо да имамо низ битова дужине m, k независних хеш функција и да је свака вредност хеш функције подједнако вероватна. Тада вероватноћа да вредност неког бита у низу није постављена на 1 на основу израчунате вредности једне хеш функције износи

$$1 - \frac{1}{m}.$$

Вероватноћа да вредност неког бита није постављена на 1 на основу израчунатих вредности свих k хеш функција износи

$$(1 - \frac{1}{m})^k.$$

Како бисмо проценили вероватноћу овог догађаја за велике вредности m можемо искористити следећу једнакост:

$$\lim_{m\to\infty}(1 - \frac{1}{m})^m = \frac{1}{e}.$$

Дакле, вероватноћа да вредност неког бита није постављена на 1 на основу вредности свих хеш функција за довољно велико m је:

$$(1 - \frac{1}{m})^k = ((1 - \frac{1}{m})^m)^{\frac{k}{m}} \approx e^{-\frac{k}{m}}.$$

Уколико смо додали n елемената у скуп, вероватноћа да вредност неког бита није постављена на 1 је

$$(1 - \frac{1}{m})^{kn} \approx e^{-\frac{kn}{m}},$$

стога је вероватноћа да је вредност неког бита постављена на 1 након n додавања елемената у скуп:

$$1 - (1 - \frac{1}{m})^{kn} \approx 1 - e^{-\frac{kn}{m}}.$$

Како би алгоритам за проверу припадности неког елемента скупу произвео лажно позитиван резултат, потребно је да за елемент који не припада скупу вредности свих одговарајућих битова у низу буду 1, а вероватноћа таквог догађаја износи

$$P_{lp} = (1 - (1 - \frac{1}{m})^{kn})^k \approx (1 - e^{-\frac{kn}{m}})^k. \tag{2.1}$$

P_{lp} представља добру апроксимацију вероватноће лажно позитивних резултата. На основу вредности P_{lp} можемо закључити да та вероватноћа расте са бројем додатих елемената скупу (n), а смањује се с повећањем дужине низа битова (m). Такође, можемо одредити оптималне вредности за k и m тако да P_{lp} буде минимална.

За дати однос $\frac{m}{n}$, који представља број додељених битова по елементу, вероватноћа лажно позитивног резултата може се подесити избором броја хеш

функција k. Оптимална вредност за k одређује се минимизацијом вероватноће лажно позитивних резултата [4]:

$$k = \frac{m}{n} \ln 2. \tag{2.2}$$

Претходну формулу можемо извести на основу формуле (2.1). Применом функције ln на обе стране једначине у формули (2.1) добија се:

$$\ln P_{lp} = k \ln(1 - e^{-\frac{kn}{m}}). \tag{2.3}$$

Уводи се смена $x = \frac{n}{m}$ у једнакост (2.3):

$$\ln P_{lp} = k \ln(1 - e^{-kx}). \tag{2.4}$$

Диференцирањем једнакости (2.4) по промењливој k добија се:

$$\frac{\partial}{\partial k} \ln P_{lp} = \ln(1 - e^{-kx}) + \frac{kxe^{-kx}}{1 - e^{-kx}} \tag{2.5}$$

У једнакост (2.5) уводи се смена $t = e^{-kx}$:

$$\frac{\partial}{\partial k} \ln P_{lp} = \ln(1 - t) - \frac{t \ln t}{1 - t}. \tag{2.6}$$

Како би израз на левој страни једнакости (2.4) имао минумалну вредност потребно је одредити вредност промењливе t за коју је вредност израза са леве стране једнакости (2.6) једнака нули. То важи за $t = \frac{1}{2}$. Када се замени вредност $\frac{1}{2}$ на место промењливе t у смени $t = e^{-kx}$, а затим примени функција ln на обе стране једнакости, добија се $-xk = -\ln 2$. С обзиром да је уведена смена $x = \frac{n}{m}$, а на основу претходно добијене једнакости $-xk = -\ln 2$, закључује се да израз са леве стране једнакости (2.3) има минималну вредност за $k = \frac{m}{n} \ln 2$. Како је функција ln растућа, следи да је и вредност израза са леве стране једнакости (2.1) минимална за $k = \frac{m}{n} \ln 2$.

У пракси, дужина филтера m, при задатој вероватноћи лажно позитивних резултата P_{lp} и очекиваном броју елемената n, може се одредити помоћу формуле:

$$m = -\frac{n \ln P_{lp}}{(\ln 2)^2}. \tag{2.7}$$

Ову формулу можемо извести када у формулу (2.1) заменимо формулу за рачунање оптималне вредности k коју смо претходно извели.

Дакле, дужина филтера мора линеарно расти са повећањем бројем елемената да би се одржала фиксна вероватноћа лажно позитивних резултата [4]. Из тог разлога за m се узимају доста веће вредности у односу на вредност n. Притом, морамо водити рачуна да низ битова не постане превелики што би резултовало његовим пребацивањем на диск јер чести приступи диску могу значајно деградирати перформансе Блумовог филтера.

2.4 Предности и мане Блумовог филтера

Највећа предност Блумовог филтера је његова просторна ефикасност. Наиме, у односу на хеш табеле он заузима значајно мање простора. Ово је последица чињенице да Блумов филтер не чува оригиналне податке, већ представља један елемент скупа постављањем вредности k битова у одговарајућем низу битова. Због тога може веома лако обрадити велику количину информација. Операције додавања елемената и провере припадности елемента скупу су временски изузетно ефикасне јер временска сложеност не зависи од количине података или дужине низа, већ од броја хеш функција за које можемо да претпоставимо да се извршавају у константном времену. С обзиром на то да приликом провере припадности елемената структури нису могући лажно негативни резултати, Блумов филтер је идеалан за примене у којима је пожељно да избегнемо скупе провере, нпр. у базама података. Приметимо и да је ова структура података веома једноставна за имплементацију захваљујући једноставности хеш функција.

Блумов филтер има и одређена ограничења: то су могућност добијања лажно позитивних резултата, немогућност брисања и ишчитавања елемената, фиксна дужина, повећање стопе лажно позитивних резултата услед лошег одабира вредности k и m. Ипак, у одређеном контексту и приликом правилног избора вредности параметара наведени недостаци су занемарљиви у односу на корист коју остварујемо коришћењем Блумових филтера.

2.5 Варијације и примена Блумових филтера

Постоје различите варијанте Блумовог филтера чијом имплементацијом је могуће превазићи неке од поменутих мана.

Бројачки Блумови филтери (енг. Counting Bloom filters) подржавају операцију брисања елемената. Наиме, они уместо низа битова поседују низ бројача. Инкрементирање бројача на позицији i аналогно је постављању вредности бита на позицији i на вредност 1 код класичног Блумовог филтера. Покушај брисања елемената изводимо тако што декрементирамо бројаче на одговарајућим позицијама. Ако притом вредност неког бројача постане једнака нули, може се обрисати одговарајући елемент. Проблем може настати приликом прекорачења максималне вредности за бројач, што се може превазићи алокацијом довољног броја битова за бројаче. С обзиром на то да се у овом случају алоцира додатна меморија за бројаче, ову структуру података карактерише већа просторна сложеност.

Скалабилни Блумови филтери (енг. Scalable Bloom filters) су дизајнирани за рад са подацима чији број није унапред познат и који се може повећавати. Функционише тако што систем надовезује нов, већи Блумов филтер на скоро попуњен постојећи Блумов филтер. Вероватноћа лажно позитивних резултата у овом случају не расте с порастом количине података.

Постоји још неколико имплементација Блумових филтера које нуде додатне функционалности и превазилазе недостатке основне имплементације [6].

Блумов филтер је пронашао примену у различитим областима рачунарства. Може се користити за проверу да ли је УЛР злонамеран у веб претраживачима, избегавање скупих провера на диску у базама података, брже прослеђивање пакета у мрежним рутерима, убрзавање синхронизације кеш меморије у дистрибуираним системима, брзу проверу да ли нека реч постоји у речнику код програма за проверу правописа ...

Глава 3

Филтер количника

Филтер количника је просторно ефикасна пробабилистичка структура података која је представљена 2011. године [7]. Ова структура података има сличне просторне и временске перформансе као Блумов филтер, уз неке додатне погодности. За почетак, код филтера количника је могуће брисање елемената. Као што смо већ поменули, Блумов филтер не пружа оптималне перформансе уколико се чува на диску. Узрок томе је потреба за великим бројем приступа диску. С друге стране, филтер количника је структура података која је погодна за кеширање (енг. caching) и не захтева велики број узастопних приступа диску. Такође, може се по потреби једноставно проширивати или смањивати. Постоји и могућност спајања два филтера количника.

Основне операције које подржава филтер количника су:

- додавање елемента у скуп података
- провера да ли елемент припада скупу података
- брисање елемента из скупа података.

Приликом провере припадности елемента скупу можемо добити лажно позитивне резултате, али не и лажно негативне резултате.

Филтер количника можемо посматрати као хеш табелу засновану на отвореном адресирању која користи тачно једну хеш функцију и уместо оригиналних података чува њихове отиске. Отисак f неког елемента x добијамо тако што рачунамо вредност хеш функције h за дати елемент x. Затим се

добијени отисак f, који је дужине p битова, партиционише на два дела поступком предложеним од стране Доналда Кнута (енг. Donald Knuth) који се назива *формирање количничког скупа* (енг. quotienting). Овим поступком добијамо *остатак* (енг. remainder), у ознаци f_r, који представља r битова најмање тежине отиска f и *количник* (енг. quotient), у ознаци f_q, који представља $q = p - r$ битова највеће тежине отиска f. Управо одатле и потиче назив ове структуре података. Вредности f, f_r и f_q можемо израчунати према следећим формулама:

$$f = h(x)$$
$$f_r = f \bmod 2^r$$
$$f_q = \left\lfloor \frac{f}{2^r} \right\rfloor$$

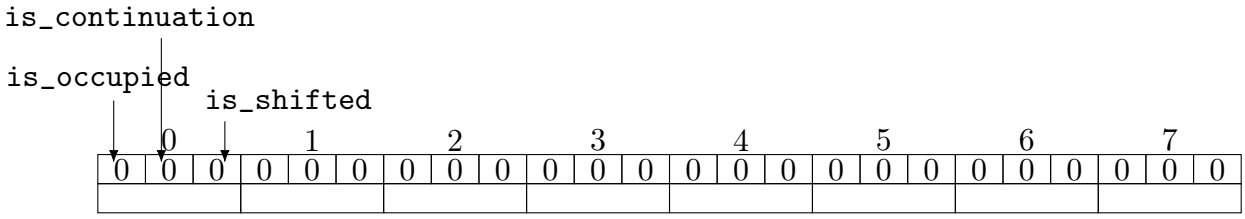
Рачунањем количника f_q добијамо позицију слота у хеш табели у који уписујемо израчунату вредност остатка f_r. Слот на позицији f_q назива се *канонски слот* (енг. canonical slot) елемента x. Може се догодити да два различита елемента имају исте количнике f_q, што би значило да имају исти канонски слот. Ова појава се назива *блага колизија* (енг. soft collision). Блага колизија код филтера количника се решава техником отвореног адресирања која се назива *линеарно попуњавање* (енг. linear probing). Ова техника се састоји у тражењу првог следећег слободног слота у хеш табели у случају да је слот на израчунатој позицији f_q заузет. Уколико не нађемо ни један слободан слот до краја хеш табеле, претрагу настављамо од позиције 0. Битно је нагласити да се код филтера количника остаци који имају исти канонски слот морају налазити на узастопним позицијама у хеш табели, и то у растућем поретку. Ова варијанта линеарног попуњавања се зове *Робин Худ хеширање* (енг. Robin-Hood hashing) и може довести до померања остатака дуж хеш табеле приликом додавања или брисања елемената [8]. Низ остатака којима одговара иста вредност количника (тј. имају исти канонски слот), а који се налазе на узастопним позицијама у хеш табели називају се *сегменти* (енг. runs). Неколико узастопних сегмената између којих нема празних слотова формира *кластер* (енг. cluster). Кластеру претходи непопуњен слот и на његовом почетку се налази елемент који је у свом канонском слоту.

У оквиру хеш табеле код филтера количника уз сваки слот чувају се и три додатна *бита метаподатака* (енг. metadata bits). Ови додатни битови имају

улогу у проналажењу елемената у хеш табели, с обзиром да се они не морају увек налазити у свом канонском слоту. Битови метаподатака имају следеће називе и улоге:

- `je_zauzet` (енг. `is_occupied`), чија се вредност поставља на 1 када слот представља канонски слот неког остатка који се налази у хеш табели. На основу броја ових битова чија је вредност постављена на 1, можемо одредити број сегмената унутар кластера;
- `je_nastavak` (енг. `is_continuation`), чија се вредност поставља на 1 ако се у слоту налази остатак који није први унутар сегмента. На основу овог бита можемо одредити почетак сегмента, тако што ћемо наћи слот чија је вредност овог бита једнака 0;
- `je_pomeren` (енг. `is_shifted`), чија се вредност поставља на 1 уколико остатак није у свом канонском слоту, а иначе има вредност 0.

Код празног филтера количника вредности свих битова метаподатака су постављене на 0. Број слотова рачунамо по формули $m = 2^q$, јер помоћу q битова количника добијамо 2^q могућих вредност количника, тј. можемо индексирати 2^q слотова.



Слика 3.1: Филтер количника

3.1 Додавање, провера припадности и брисање елемената

Када желимо да додамо елемент x у филтер количника, најпре рачунамо његов отисак $f = h(x)$, а на основу њега количник f_q и остатак f_r. Најпре проверавамо да ли је слот на позицији f_q празан. Уколико јесте, додајемо остатак f_r у слот и постављамо вредност бита `je_zauzet` на 1. У супротном,

морамо да пронађемо одговарајући сегмент ком је потребно додати елемент. Свакако, постављамо вредност бита `je_zauzet` на 1. Крећемо се улево кроз хеш табелу, тражећи почетак кластера. Почетак кластера препознајемо по вредности бита `je_pomeren` која је једнака 0. Успут, пребројавамо колико битова `je_zauzet` има вредност 1, на основу чега ћемо знати колико има сегмената у кластеру до позиције f_q. Нека смо рецимо пребројали да постоје два бита `je_zauzet` чија је вредност једнака 1 с леве стране. То значи да се наш елемент налази у трећем сегменту од почетка кластера. Почетак сегмента сигнализира вредност бита `je_nastavak` која је једнака нули. Дакле, тражићемо трећи по реду бит `je_nastavak` који има вредност 0 од почетка кластера, тачније трећи сегмент. Преостаје да пронађемо позицију уметања елемента унутар сегмента, јер вредности остатака у сегменту треба да буду сортиране растуће. Уколико је нађена позиција заузета, потребно је померити све елементе удесно почев од те позиције до краја кластера уз ажурирање вредности битова `je_nastavak` и `je_pomeren`. Уколико је потребно, поступак се наставља циклично од почетка хеш табеле.

Пример 3.1.1. Претпоставимо да имамо 16-битну хеш функцију h и нека је $q = 3$. Тада је величина хеш табеле $m = 2^3 = 8$ и $r = 16 - 3 = 13$. Ради бољег разумевања свих корака које желимо да представимо у овом примеру, користићемо графичку репрезентацију филтера количника као на слици 3.1.

Претпоставимо да желимо да додамо елемент a у хеш табелу. Најпре рачунамо отисак $f(a) = h(a)$, затим и одговарајући количник и остатак $f_q(a) = \left\lfloor \frac{f(a)}{2^{13}} \right\rfloor = 7, f_r(a) = f(a) \bmod 2^{13}$. С обзиром на то да је хеш табела празна, можемо додати остатак у његов канонски слот уз постављање вредности бита `je_zauzet` на 1. Аналогно се поступа приликом додавања елемената b и v за које важи $f_q(b) = 1$ и $f_q(v) = 4$, пошто су њихови канонски слотови празни.

0			1			2			3			4			5			6			7		
0	0	0	**1**	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	**1**	0	0
			$f_r(b)$									$f_r(v)$									$f_r(a)$		

Покушајмо да додамо елемент s. Вредност количника износи $f_q(s) = 1$.

Видимо да је слот на позицији 1 заузет, те је потребно наћи почетак кластера. Пошто је вредност бита `je_pomeren` на позицији 1 једнака нули, управо од позиције 1 почиње кластер, самим тим и сегмент. Пошто важи $f_s > f_b$,[1] прелазимо на следећи слот сегмента на позицији 2 који је празан. Пронашли смо место уметања. Постављамо вредност битова `je_nastavak` и `je_pomeren` на 1, али не и вредност `je_zauzet`, јер слот на позицији 2 није канонски слот за вредност $f_r(s)$.

0			1			2			3			4			5			6			7		
0	0	0	1	0	0	0	**1**	**1**	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0
			$f_r(b)$			$f_r(s)$						$f_r(v)$									$f_r(a)$		

Додајемо елемент z за који важи $f_q(z) = 2$. Пошто је слот на позицији 2 заузет, морамо пронаћи место уметања елемента z. Најпре поставимо вредност бита `je_zauzet` на 1 за слот на позицији 2. Затим се померамо улево до почетка кластера, који се налази на позицији 1. Тражимо други сегмент одатле, јер можемо пребројати да два бита `je_zauzet` имају вредност 1. Дакле тражимо други по реду бит `je_nastavak` чија је вредност једнака 0. Испоставља се да је тражени почетак сегмента на позицији 3, која је празна. Уписујемо $f_r(z)$ и постављамо бит `je_pomeren` на 1.

0			1			2			3			4			5			6			7		
0	0	0	1	0	0	**1**	1	1	0	0	**1**	1	0	0	0	0	0	0	0	0	1	0	0
			$f_r(b)$			$f_r(s)$			$f_r(z)$			$f_r(v)$									$f_r(a)$		

Када покушамо да додамо елемент d за који важи $f_q(d) = 1$, можемо да приметимо да је слот на тој позицији заузет. Такође, то је почетак кластера, као и сегмента. Дакле, позицију за уметање елемента d тражимо у првом сегменту. То ће бити позиција 2 која је заузета, стога треба померити све елементе почев од позиције 2 удесно, док не наиђемо на празан слот. Након тога ажурирамо одговарајуће битове метаподатака.

0			1			2			3			4			5			6			7		
0	0	0	1	0	0	1	1	1	0	**1**	**1**	1	0	**1**	0	0	**1**	0	0	0	1	0	0
			$f_r(b)$			$f_r(d)$			$f_r(s)$			$f_r(z)$			$f_r(v)$						$f_r(a)$		

[1] За потребе овог примера претпостављамо да је функција f_r растућа, што не мора бити случај.

Провера да ли елемент припада структури се изводи на сличан начин. За почетак рачунамо вредности $f(x), f_q(x)$ и $f_r(x)$ за елемент x за који вршимо проверу. Уколико је слот на позицији $f_q(x)$ празан или је вредност бита `je_zauzet` једнака 0, то значи да у структури не постоји остатак $f_r(x)$. Ако је ипак вредност тог бита једнака 1, тражимо одговарајући сегмент. Прво идемо улево и бројимо сегменте док не дођемо до почетка кластера, а затим се враћамо удесно док не наиђемо на почетак одговарајућег сегмента. Претражујемо жељени сегмент и ако пронађемо остатак чија је вредност једнака вредности $f_r(x)$, тада можемо да кажемо да елемент x са одговарајућим количником и остатком можда припада структури.

Брисање елемената из структуре се изводи слично као и додавање елемената у структуру. Можемо поступком којим проверавамо припадност наћи позицију одговарајућег остатка који треба обрисати. За разлику од поступка додавања елемената, елемент који се налазе са десне стране обрисаног је потребно померати улево уз адекватно подешавање вредности битова метаподатака. Поступак се завршава када наиђемо на празан слот.

Временска сложеност наведених операција ограничена је величином кластера, због тога што се операције додавања и брисања елемената, као и провера припадности елемента структури изводе унутар једног кластера. У већини случајева, дужина кластера је ограничена са $O(1)$, иако постоји вероватноћа да је дужина свих кластера ограничена са $O(\log m)$, где m представља број слотова хеш табеле.

3.2 Вероватноћа лажно позитивног резултата

Лажно позитивни резултати приликом провере припадности елемената структури настају када два различита елемента имају једнаке отиске f. Ова појава се назива *снажна колизија* (енг. hard collision).

Отисци су секвенце дужине p битова, тако да вероватноћа да два различита елемента имају исти отисак износи:

$$\frac{1}{2^p}.$$

Стога, вероватноћа да два елемента немају исти отисак износи:

$$1 - \frac{1}{2^p}.$$

Вероватноћа да елемент нема отисак једнак било ком од n отисака елемената из структуре износи:

$$(1 - \frac{1}{2^p})^n.$$

Дакле, вероватноћа да елемент има исти отисак као неки од n елемената у структури, тј. вероватноћа лажно позитивног резултата при провери припадности елемента структури износи:

$$P_{lp} = 1 - (1 - \frac{1}{2^p})^n.$$

Можемо користити следећу једнакост за велике вредности променљиве x:

$$\lim_{x \to -\infty} (1 + \frac{1}{x})^x = e.$$

Применом претходне једнакости на формулу за рачунање вероватноће лажно позитивних резултата при провери припадности елемента структури P_{lp} добијамо следећу процену:

$$P_{lp} = 1 - (1 - \frac{1}{2^p})^n = 1 - ((1 + \frac{1}{-2^p})^{-2^p})^{-\frac{n}{2^p}} \approx 1 - e^{-\frac{n}{2^p}} \tag{3.1}$$

На основу неједнакости $e^{-x} \geq 1 - x$, вредност P_{lp} можемо ограничити одозго:

$$P_{lp} \approx 1 - e^{-\frac{n}{2^p}} \leq \frac{n}{2^p}.$$

Можемо да закључимо да вероватноћа лажно позитивног резултата при провери припадности елемента структури зависи од броја елемената који се налазе у структури и дужине отиска елемената. Очекујемо да P_{lp} расте пропорционално повећању броја додатих елемената структури.

Можемо израчунати оптималну дужину остатка за задату вероватноћу лажно позитивног резултата провере припадности елемента структури P_{lp} на основу формуле (3.1):

$$r = \left\lceil \log(\frac{n}{2^q} \cdot \frac{1}{\ln(\frac{1}{1-P_{lp}})}) \right\rceil. \tag{3.2}$$

На почетку овог поглавља набројали смо предности филтера количника у односу на Блумов филтер. У наставку ћемо размотрити пробабилистичку структуру података која има сличне перформансе као две поменуте структуре, а имплементацију значајно једноставнију од филтера количника.

Глава 4

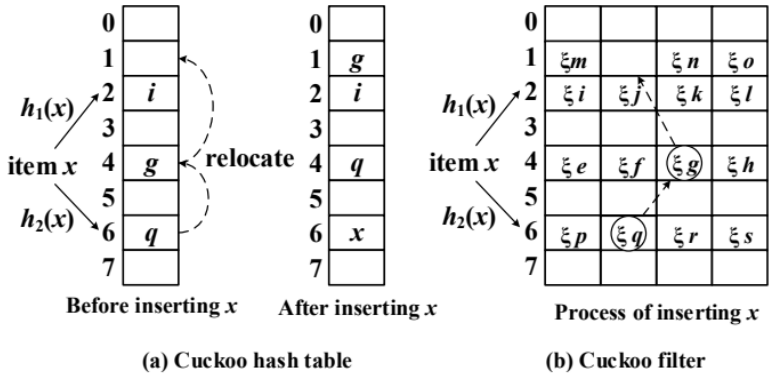
Филтер кукавице

Многе модификације Блумовог филтера које подржавају операцију брисања елемената из скупа деградирају просторну ефикасност и перформансе структуре. У секцији 2.5 смо видели да бројачки Блумов филтер подржава операцију брисања елемената, међутим заузима чак три до четири пута више меморије од обичног Блумовог филтера [4]. *Филтер кукавице* је просторно ефикасна пробабилистичка структура података која је представљена 2014. године [6]. Ова структура подржава три основне операције:

- додавање елемената у скуп података;
- проверу да ли елемент припада скупу података;
- брисање елемента из скупа података.

Филтер кукавице користи хеш табелу која садржи m *слотова* (енг. buckets), где сваки слот може чувати до b елемената. Ова хеш табела се користи за складиштење *отисака* података код филтера кукавице. Због чињенице да су отисци података мањи од оригиналних података (дужине свега неколико битова) и да се операције над хеш табелама у већини случајева изводе у константном времену, филтер кукавице је погодан за рад са великим скуповима података. Филтер кукавице користи *хеширање кукавице са парцијалним кључем* (енг. partial-key cuckoo hashing) за разрешавање колизија приликом додавања нових елемената у структуру. Приликом провере припадности елемента структури постоји могућност добијања лажно позитивних резултата, док лажно негативни резултати нису могући. Једна од највећих предности

филтера кукавице у односу на остале представнике пробабилистичких структура података је ефикасна имплементација операције брисања елемената.



Слика 4.1: Филтер кукавице (преузето са [9])

4.1 Хеширање кукавице

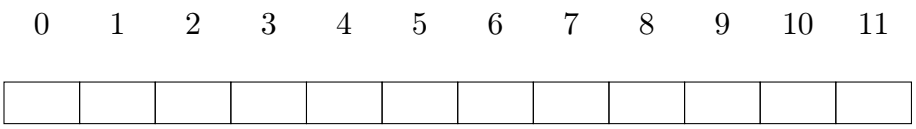
Хеширање кукавице је алгоритам отвореног адресирања који је представљен 2001. године [10]. Алгоритам је добио назив по истоименој птици[1]. Основна идеја код овог начина решавања колизија је хеширање помоћу две хеш функције које пресликавају скуп података у скуп вредности $0, 1, \ldots, m-1$ за хеш табелу која има m слотова. На тај начин постоје тачно два слота у хеш табели у којима неки елемент може бити сачуван.

Када додајемо нов елемент x у хеш табелу која решава колизије хеширањем кукавице, рачунамо вредност хеш функција $i = h_1(x)$ и $j = h_2(x)$. Израчунате вредности i и j представљају позиције слотова унутар хеш табеле у које треба сместити елемент x. Када ни један слот на израчунатим позицијама није слободан, елемент x записујемо у било који од та два слота. Елемент x' који је до тада био у одабраном слоту пребацујемо у алтернативни слот рачунањем његових вредности хеш функција. Овај поступак се понавља све док се не пронађе слободан слот или док се не прекорачи унапред задат максималан број итерација. Битно је нагласити да је код хеширања кукавице неопходно чувати оригиналне елементе у хеш табели у случају поновног ра-

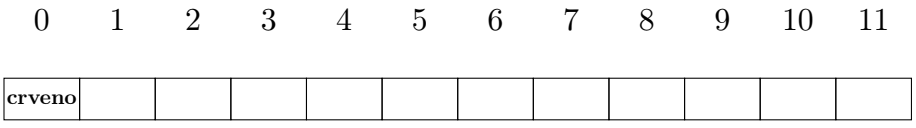
[1]Кукавице не праве гнезда, већ подмећу своја јаја у гнезда других птица. Младунци кукавице када се излегу, избаце друге птиће и јаја из гнезда. Решавање колизија хеширањем кукавице функционише по сличном принципу.

чунања вредности хеш функција.

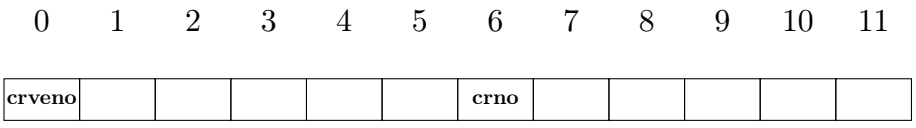
Пример 4.1.1. Дата је празна хеш табела величине $m = 12$ и две хеш функције h_1 и h_2 чије вредности су из скупа $0, 1, \ldots, m-1$.



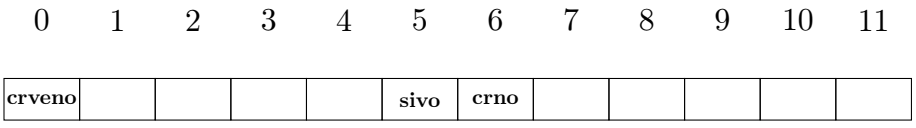
Покушаћемо да додамо ниску „crveno” у хеш табелу. Поступак додавања новог елемента започињемо рачунањем вредности хеш функција $h_1(crveno)$ и $h_2(crveno)$. Нека је $h_1(crveno) = 0$ и $h_2(crveno) = 8$. Хеш табела је празна, па можемо одабрати било коју од две добијене позиције, нпр. сачуваћемо ниску у слоту на позицији 0.



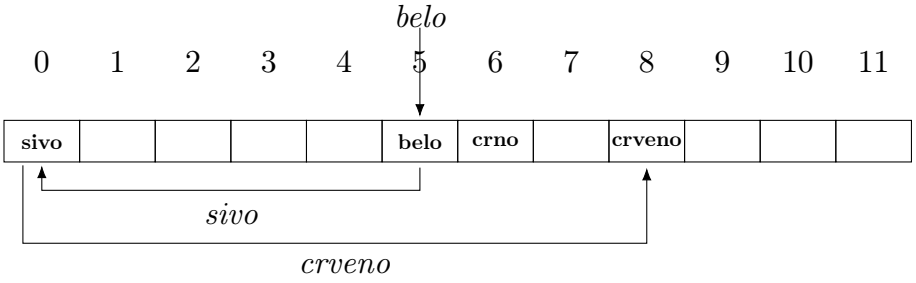
Следећа ниска коју желимо да додамо је „crno”. Рачунамо вредности хеш функција $h_1(crno) = 6$ и $h_2(crno) = 0$ и додајемо ниску у слот на позицији 6, јер тај слот није заузет.



Поступак је сличан уколико желимо да додамо ниску „sivo” чије су вредности хеш функција $h_1(sivo) = 5$ и $h_2(sivo) = 0$.



Када покушамо да додамо ниску „belo" чије су вредности хеш функција $h_1(belo) = 5$ и $h_2(belo) = 6$, долази до појаве колизије јер су слотови на позицијама 5 и 6 заузети. Колизију решавамо тако што ниску „belo" уписујемо на позицију 5, ниску „sivo" премештамо на њену алтернативну позицију, тј. позицију 0, а ниску „crveno" на њену алтернативну позицију, тј. позицију 8. Како је слот на позицији 8 празан, ту се поступак додавања ниске „belo" завршава.



Хеш табеле које користе хеширање кукавице за решавање колизија називају се другачије и *табеле кукавице* (енг. cuckoo tables).

Временска сложеност операције додавања елемента у табелу кукавице у најгорем случају износи $O(m)$, где m представља број слотова. Ово се дешава код евентуалног великог броја премештања елемената услед колизије. Амортизована сложеност додавања елемената у табеле кукавице износи $O(1)$. Што је већи број додатих елемената у табелу, очекује се већи број премештања елемената услед колизије. Како би се избегао велики број премештања потребно је да величина табеле m буде већа од броја елемената које треба додати. Време провере припадности елемента табели кукавице, као и време брисања елемента је константно, јер за сваки елемент постоје само две одговарајуће позиције у табели кукавице на којима се тај елемент може наћи.

4.2 Основне операције филтера кукавице

Ради постизања просторне оптимизације, филтер кукавице не чува оригиналне елементе скупа у хеш табели, већ њихове отиске који представљају ниске битова дужине f. На овај начин се значајно смањује меморијски про-

стор који филтер заузима, али се елиминише могућност примене хеширања кукавице. Наиме, могуће је иницијално израчунати вредности хеш функција приликом додавања елемената, међутим не бисмо могли поново израчунати вредност хеш функција у случају колизије, с обзиром да су у хеш табели сачувани отисци елемената. Овај проблем се превазилази применом хеширања кукавице са парцијалним кључевима.

Хеширање кукавице са парцијалним кључевима представља адаптацију алгоритма хеширања кукавице за примену код филтера кукавице. Код ове имплементације хеширања могуће је израчунати вредност хеш функције, тј. позицију елемента у хеш табели на основу његовог отиска. Вредности одговарајућих позиција елемента x у филтеру кукавице се рачунају према следећим формулама:

$$\begin{aligned} i &= h(x) \\ j &= i \oplus h(fingerprint(x)), \end{aligned} \tag{4.1}$$

где је h хеш функција, $fingerprint$ отисак елемента x, а $\oplus$ операција ексклузивне дисјункције (XOR). Операција ексклузивне дисјункције у формули (4.1) обезбеђује могућност рачунања алтернативне позиције елемента у хеш табели на основу тренутне позиције и отиска елемента без познавања оригиналног елемента. Вредност хеш функције отиска се рачуна у циљу боље расподеле елемената у структури података. Када не бисмо рачунали вредност хеш функције отиска, операција ексклузивне дисјункције би утицала само на неколико битова вредности позиције i, што би значило да се позиција елемента не би значајно изменила приликом поновног хеширања.

Додавање новог елемента филтеру кукавице се изводи на сличан начин као у примеру 4.1.1. Основна разлика је у томе што смо у поменутом примеру додавали оригиналне елементе у структуру, док код филтера кукавице додајемо отиске елемената у структуру. Најпре рачунамо отисак елемента x који желимо да додамо у филтер кукавице, а затим кандидате за његову позицију i и j на основу формула (4.1). Уколико су обе позиције заузете, отисак уписујемо нпр. у слот на позицији i. Отисак који је до тада био на позицији i премештамо на позицију j' коју рачунамо ексклузивном дисјункцијом тог отиска и вредности i. Поступак се понавља док се не наиђе на непопуњену позицију или док се не прекорачи максималан број итерација. Уколико не

пронађемо слободну позицију, додавање елемента се сматра неуспешним, а структура попуњеном.

Провера припадности елемента структури започиње рачунањем отиска елемента и одговарајућих позиција i и j. Упоређују се отисци који се налазе на позицијама i и j са израчунатим отиском и ако постоји поклапање бар на једној позицији, тада можемо да кажемо да елемент можда припада скупу. Иначе, уколико не постоји поклапање, можемо са сигурношћу тврдити да елемент не припада скупу, јер постоје тачно два кандидата за позиције на којима се може пронаћи неки елемент. Лажно позитивни резултати су могући јер два различита елемента могу имати исти отисак и одговарајућу позицију i у структури, а самим тим и исте позиције j. Паметним одабиром одговарајућих параметара вероватноћа лажно позитивних резултата при провери припадности елемента се може свести на минимум.

Брисање елемента из филтера кукавице се изводи на исти начин као провера припадности елемента структури уз брисање елемента у случају поклапања отисака.

4.3 Вероватноћа лажно позитивних резултата

Вероватноћа лажно позитивног резултата при провери припадности елемента структури у највећој мери зависи од дужине отиска f и броја отисака елемената b који се може чувати у сваком слоту. Лажно позитиван резултат приликом провере припадности елемента структури настаје када постоји поклапање отисака. Вероватноћа поклапања израчунатог отиска елемента са неким отиском у структури износи:

$$\frac{1}{2^f}.$$

Стога, вероватноћа непоклапања отисака износи:

$$1 - \frac{1}{2^f}.$$

Међутим, да бисмо били сигурни да нисмо добили лажно позитиван резултат, морамо размотрити свих b отисака у слотовима на одговарајућим позицијама. Дакле, морамо извршити највише $2b$ поређења. Вероватноћа да се неки

отисак не налази у структури података износи:

$$(1 - \frac{1}{2^f})^{2b}.$$

Вероватноћа добијања лажно позитивног резултата, тј. вероватноћа да се отисци поклопе у неком од $2b$ поређења износи:

$$1 - (1 - \frac{1}{2^f})^{2b}.$$

За довољно мале вредности променљиве x и природне бројеве n, можемо искористити следећу формулу:

$$(1 - x)^n \approx 1 - nx.$$

Коначно, вероватноћа лажно позитивног резултата приликом провере припадности износи:

$$P_{lp} = 1 - (1 - \frac{1}{2^f})^{2b} \approx 1 - (1 - \frac{2b}{2^f}) \approx \frac{2b}{2^f}.$$

Како бисмо имали жељену стопу лажно позитивних резултата при провери припадности P_{lp} неопходно је да дужина отиска f износи:

$$f \geq \left\lceil \ln \frac{2b}{P_{lp}} \right\rceil.$$

Претходна формула се изводи применом функције ln на формулу за рачунање вредности P_{lp}. Интуитивно, што је већа дужина отисака, то је погоднија вредност P_{lp}, али то захтева додатну меморију.

Коефицијент оптерећења структуре α (енг. load factor) дефинишемо као количник броја додатих елемената у структуру и капацитета структуре. Другим речима, коефицијент оптерећења нам говори колико је нека структура попуњена. За додавање n елемената структури а како бисмо обезбедили дати коефицијент оптерећења структуре α, неопходно је да број слотова m задовољава наредну неједнакост:

$$m \geq \left\lceil \frac{n}{b\alpha} \right\rceil. \tag{4.2}$$

Занимљиво је да филтер кукавице одржава добре перформансе чак и када је скоро попуњен (нпр. искоришћеност простора од 95%) [6]. Када је $b = 1$, тада

можемо искористити само 50% простора структуре, а ако је $b = 2$ или $b = 4$, тај проценат расте на 84%, односно 95% [4]. Када у потпуности попунимо капацитете структуре, потребно је проширити је новом хеш табелом. Имајући у виду све представљене чињенице о филтеру кукавице, можемо закључити да он представља одличну алтернативу Блумовом филтеру за приближну проверу припадности елемента скупу.

Глава 5

Имплементација, евалуација и упоредна анализа

У раду су представљене три различите пробабилистичке структуре података које се могу користити за приближну проверу припадности елемената скупу. Структуре података и њихове основне операције имплементиране су у програмском језику *C++* [11], при чему су коришћене постојеће структуре података стандардне библиотеке шаблона (енг. Standard Template Library) као што су динамички низ и хеш табела. За обраду и графичко представљање резултата коришћен је програмски језик *Python* [12] и библиотеке *Pandas* [13] и *Matplotlib* [14]. Комплетна имплементација доступна је на јавном *GitHub* репозиторијуму: *https://github.com/milicagajic/Probabilisticke-strukture-podataka.git*.

Тестирање имплементираних структура је извршено на рачунару са следећом конфигурацијом:

- Процесор: Intel® Core™ 5 120U × 12
- RAM меморија: 16.0 GiB
- Оперативни систем: Ubuntu 26.04 LTS

Програм за тестирање је покренут 100 пута за различит број елемената, почев од 100000 до 10000000 елемената. За генерисање скупа података за додавање и претраживање коришћен је генератор псеудослучајних бројева. При сваком покретању генерисана су два различита скупа бројева, један за

Слика 5.1: Стандардни излаз након покретања програма

додавање елемената у одговарајуће структуре података, а други за проверу припадности елемената структурама. То је урађено како бисмо омогућили добијање лажно позитивних резултата приликом провере припадности. Мерене су следеће вредности:

- време потребно за додавање елемената у структуру
- време потребно за проверу припадности елемента структури
- меморија коју структура заузима

- стопа лажно позитивних резултата при провери припадности елемената структури.

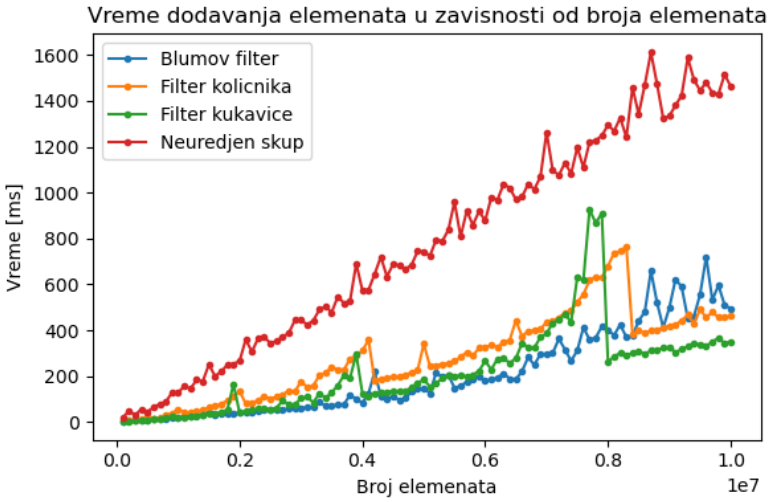
Иста мерења су извршена и за `std::unordered_set<T>`, структуру података у оквиру стандардне библиотеке шаблона која у основи представља хеш табелу. Резултати мерења свих извршавања програма уписани су у засебну датотеку. Резултати су затим обрађени помоћу `Python` програма који користи библиотеке `Pandas` и `Matplotlib`. Пример изгледа стандардног излаза након једног покретања програма дат је на слици 5.1.

Потребно је нагласити неке појединости имплементације структура.

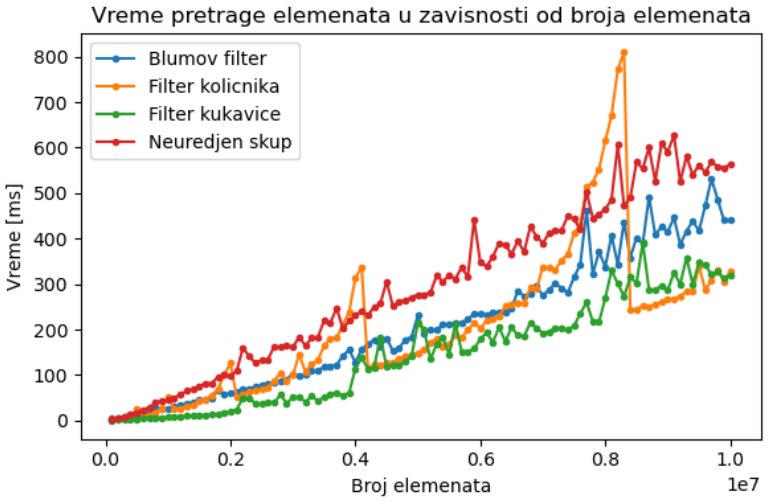
Низ битова код Блумовог филтера имплементиран је као низ 64-битних неозначених целих бројева и примењивано је двоструко хеширање за сваку од k хеш функција. Број потребних битова m и хеш функција k рачуна се на основу броја очекиваних елемената које треба додати структури и циљане стопе лажно позитивних резултата при тестирању припадности елемената структури на основу формула (2.2) и (2.7).

Код филтера количника слот је представљен као структура која садржи 32-битни неозначен цео број за чување остатка r и три податка типа `bool` који представљају метаподтке. Оваква структура података заузима нешто више меморије од представљене структуре у поглављу 3. Број слотова m једнак је 2^q за број битова количника q који се рачуна тако да m буде довољно за очекиван број елемената. Број битова остатка r се рачуна на основу задатих вредности броја битова количника и циљане стопе лажно позитивних резултата при провери припадности према формули (3.2).

Код филтера кукавице број елемената у сваком слоту је постављен на четири, а број слотова m рачуна се на основу очекиваног броја елемената за додавање у структуру n и жељеног коефицијента оптерећења структуре α по формули (4.2). За случајан одабир позиције за уписивање отиска користи се генератор псеудослучајних бројева.



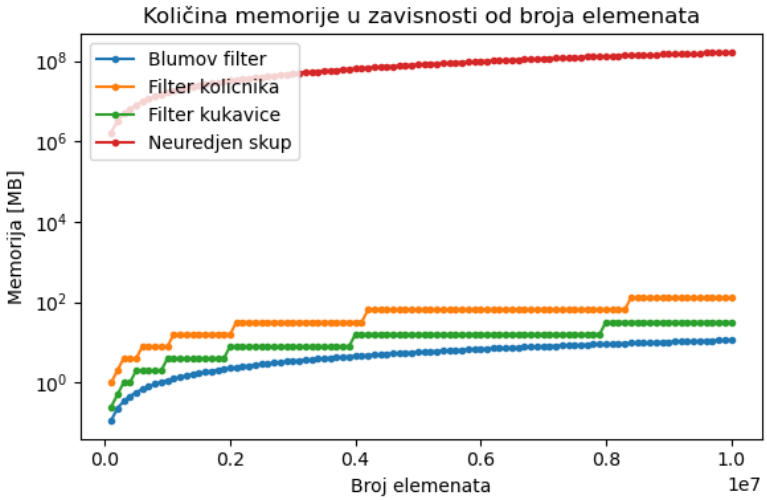
Слика 5.2: Зависност времена додавања елемената структури од броја елемената



Слика 5.3: Зависност времена провере припадности елемената структури од броја елемената

На сликама 5.2 и 5.3 можемо видети како број елемената утиче на време потребно за додавање елемената у скуп података и време потребно за проверу припадности елемената скупу података. Резултати потврђују оно о чему је било речи у претходним поглављима. Представљене структуре података имају приближне перформансе при извршавању ових операција и значајно су временски ефикасније од хеш табеле. На обе слике можемо уочити изражене скокове у одређеним тачкама, нарочито код филтера количника и

филтера кукавице. То је последица високог коефицијента оптерећења α ових структура података. Када се повећа капацитет структуре, долази до наглог смањења времена потребног за додавање и претраживање елемената у структури. Дакле, закључујемо да на временске перформансе ових структура података више утиче коефицијент оптерећења него повећање броја елемената. Уколико ограничимо коефицијент оптерећења структуре података, можемо рећи да време линеарно расте у односу на број елемената. Ово је и очекивано, с обзиром да операција додавања и провере припадности елемента за представљене структуре података има константну амортизовану временску сложеност.



Слика 5.4: Зависност потребне количине меморије од броја елемената који се додаје у структуру

На слици 5.4 можемо видети како број елемената утиче на меморијско заузеће структуре података. Потребна количина меморије код представљених структура података веома споро расте с повећањем броја елемената и значајно је мања од меморије која је потребна за чување истог броја елемената у хеш табели. Треба имати у виду и следеће: ово је груба процена потребне меморије код хеш табеле, јер је рачунат само простор који заузимају елементи унутар слотова. Сваки слот у хеш табели садржи и додатне податке који заузимају додатни простор, па би меморија која је потребна код хеш табеле требало да буде већа. Такође, имплементација филтера количника заузима више простора од структуре приказане у овом раду, што је раније поменуто. Код филтера количника и филтера кукавице можемо да приметимо каскадне

скокове у потрошњи меморије с повећањем броја елемената. Ово је последица повећања капацитета поменутих структура података услед великог коефицијента оптерећења структуре података. Можемо видети и да је најмања количина меморије потребна код Блумовог филтера, што је и очекивано.

Слика 5.5: Зависност стопе лажно позитивних резултата при провери припадности елемента структури од броја елемената

На слици 5.5 можемо видети како број елемената утиче на стопу лажно позитивних резултата при провери припадности структури.

Код хеш табеле стопа лажно позитивних резултата је увек једнака нули јер хеш табела неће произвести лажно позитиван резултат при провери припадности елемента скупу. Насупрот овоме, код структура података које су разматране у овом раду лажно позитивни резултати приликом провере припадности елемента структури су могући. Стопа лажно позитивних резултата код Блумовог филтера је прилично константна за сва извршавања и износи око 1%. Ово је очекивано због имплементације Блумовог филтера која на основу вредности жељене стопе лажно позитивних резултата рачуна оптимално k и m. Дакле, оно што можемо да закључимо је да код Блумовог филтера можемо одржавати вредност стопе лажно позитивних резултата, али то за последицу има повећање потрошње меморије, што се могло уочити на претходном графику. Код филтера количника можемо видети да је стопа лажно позитивних резултата мања од 1%. Поново, ово је очекивано јер је стопа лажно позитивних резултата код филтера количника ограничена бројем еле-

мената који се налазе у структури и дужином отиска, а наша имплементација рачуна дужину остатка на основу задате дужине количника, броја елемената и циљане стопе лажних резултата. Такође, можемо приметити скокове у вредностима стопе лажно позитивних резултата који одговарају високим вредностима коефицијента оптерећења ове структуре података. Коначно, код филтера кукавице можемо да приметимо да је стопа лажних резултата константна и веома мала. С обзиром да стопа лажно позитивних резултата код филтера кукавице зависи од броја елемената у слоту и дужине отиска, а предложена имплементација користи четири 16-битна отиска у сваком слоту, овакав резултат је оправдан. Као последицу имамо већу потрошњу меморије.

Оно што можемо видети на основу изведених тестова јесте да пробабилистичке структуре података представљене у овом раду заиста имају боље временске и просторне перформансе од класичних структура података, али увек постоји одређена стопа лажно позитивних резултата при провери припадности елемената скупу података. Такође, висок коефицијент оптерећења ових структура података може лоше утицати на њихове перформансе. Оно што је јако добра особина ових структура података јесте да можемо подешавањем одређених параметара поправити одговарајуће понашање структуре.

Глава 6

Закључак

Пробабилистичке структуре података представљају класу структура података које се одричу потпуне прецизност ради постизања боље меморијске ефикасности и перформанси у раду са подацима у реалном времену. Посебно су значајне пробабилистичке структуре података које се користе за ефикасну проверу припадности елемента скупу података уз високу стопу тачности добијених резултата. Приликом тестирања припадности елемента скупу података могуће је добити лажно позитивне, али не и лажно негативне резултате. Ове структуре података постижу добре перформансе коришћењем хеширања и чувањем отисака елемената. Неки од најзначајнијих представника пробабилистичких структура података су Блумов филтер, филтер количника и филтер кукавице.

У овом раду детаљно су описане све три поменуте пробабилистичке структуре података. Описане и имплементиране су основне операције које подржавају ове структуре, са акцентом на провери припадности елемента скупу података и анализирана је њихова временска сложеност. Анализиране су вероватноће лажно позитивних резултата за проверу припадности елемената скупу података и изведене су формуле којима се оне могу проценити. Евалуиране су операције додавања елемената структури и тестирање припадности елемената скупу података при чему је мерено време потребно за извршавање ових операција, меморија коју структуре заузимају и стопа лажно позитивних резултата при провери припадности елемената скупу података. Такође, исти скуп тестова спроведен је за неуређен скуп који у основи представља хеш табелу ради поређења пробабилистичких структура података са класич-

ном структуром података која се стандардно користи за решавање проблема припадности.

Након анализе добијених резултата можемо закључити да све три структуре података имају приближне перформансе, а значајно су ефикасније у односу на стандарну хеш табелу. Време извршавања основних операција додавања елемената и тестирања припадности елемената скупу података код ових структура података је за већину извршавања краће у односу на хеш табелу. Можемо уочити скокове у времену потребном за додавања и претраживања елемената с повећањем коефицијента оптерећења структуре података, посебно у случају филтера количника и филтера кукавице. Још већа предност ових структура података у односу на хеш табелу види се у погледу потрошње меморије. Блумов филтер користи најмање меморије. Код филтера количника и филтера кукавице потрошња меморије се може подесити помоћу вредности одговарајућих параметара. Стопа лажно позитивних резултата је релативно мала код свих структура података. Код филтера кукавице смо видели најмању стопу лажно позитивних резултата при провери припадности елемента скупу података.

Одабир структуре података која ће бити коришћена зависи од проблема који се решава. Рецимо, Блумов филтер има већу меморијску ефикасност у односу на филтер количника и филтер кукавице, али за разлику од њих не подржава операцију брисања елемената и не може се ефикасно проширити. С друге стране, иако филтер количника захтева више меморије од Блумовог филтера, може имати веома добре перформансе у системима који користе кеширање. Свака од три представљене структуре података успешно и ефикасно решава проблем приближног тестирања припадности. Подешавањем њихових параметара можемо поправити њихову временску и просторну ефикасност или стопу лажно позитивних резултата. Битно је приметити да вредност коефицијента оптерећења α разматраних структура података значајно утиче на њихове перформансе, о чему треба водити рачуна приликом подешавања вредности одговарајућих параметара.

Због постизања добре меморијске ефикасности и перформанси у раду са великим скуповима података разматране структуре података имају широку

примену у рачунарским системима који обрађују велике количине података, анализи мрежног саобраћаја и у базама података.

Неке од идеја за наставак истраживања биле би побољшање приложених имплементација разматраних структура података, детаљније упознавање са различитим модификацијама сваке од представљених структура података, као и истраживање других представника, рецимо структура података Count-min sketch или HyperLogLog.

Напомена: У изради мастер рада коришћен је ChatGPT за генерисање слика у 3. и 4. поглављу, као помоћ при извођењу математичких формула за процену вероватноће лажно позитивних резултата, као помоћ у писању кода, за имплементацију разматраних структура података и генерисање скрипте за обраду и графички приказ добијених резултата. Сугестије и резултати добијени од стране ChatGPT-а нису преузети у потпуности, већ су проверени и прилагођени контексту од стране аутора. Одговорност за исправност текста, кода и слика у раду преузима аутор.

Литература

[1] K. Gopinathan and I. Sergey, "Certifying certainty and uncertainty in approximate membership query structures," in *Lahiri, S., Wang, C. (eds) Computer Aided Verification. CAV 2020. Lecture Notes in Computer Science(), vol 12225. Springer*, 2020.

[2] Medium, "Bloom filter: Application and implementation," 2024. [Online]. Available: https://harish-bhattbhatt.medium.com/bloom-filter-application-and-implementation-52c6d4512c21

[3] B. H. BLOOM, "Space/time trade-offs in hash coding with allowable errors," in *Communications of the ACM,Volume 13, Number 7*, 1970.

[4] A. Gakhov, *Probabilistic Data Structures and Algorithms for Big Data applications*, 2019.

[5] A. Tsun, *Probability & Statistics with Applications to Computing*, 2020.

[6] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, "Cuckoo filter: Practically better than bloom," in *CoNEXT '14: Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, 2014, pp. 75–88.

[7] M. A. Bender, M. Farach-Colton, R. Johnson, R. Kraner, B. C. Kuszmaul, D. Medjedovic, P. Montes, P. Shetty, R. P. Spillane, and E. Zadok, "Don't thrash: How to cache your hash on flash," in *Proceedings of the VLDB Endowment, Vol. 5, No. 11*, 2012, pp. 1627–1637.

[8] R. Wen, H. McCoy, D. Tench, G. Tagliavini, M. A. Bender, A. Conway, M. Forach-Colton, R. Johnson, and P. Pandey, "Adaptive quotient filters," in *Proc. ACM Manag. Data, Vol. 2, No. 4 (SIGMOD), Article 192*, 2024.

[9] C. Miao, "Cuckoo filter." [Online]. Available: https://miaochenlu.github.io/2022/04/30/cuckoo-filter/

[10] R. Pagh and F. F. Rodler, "Cuckoo hashing," 2001.

[11] "C++ reference." [Online]. Available: https://www.cppreference.com/

[12] "Python." [Online]. Available: https://www.python.org/

[13] "Pandas." [Online]. Available: https://pandas.pydata.org/

[14] "Matplotlib." [Online]. Available: https://matplotlib.org/