SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Projekt iz predmeta Bioinformatika

Cuckoo filter

Dominik Kotarski, Mateo Stakor, Anita Vencl

Voditelj: doc. dr. sc. Mirjana Domazet Lošo

Zagreb, Siječanj, 2020

**Sadržaj**

[1. Seminarski rad 4](#_Toc30002143)

[1.1 Opis algoritma/strukture 4](#_Toc30002144)

[1.2 Rezultati 9](#_Toc30002145)

[2. Zaključak 13](#_Toc30002146)

[3. Sažetak 14](#_Toc30002147)

[4. Literatura 15](#_Toc30002148)

# Seminarski rad

## Uvod

Cuckoo filter je probabilistička struktura podataka koja omogućuje brzo i efikasno testiranje pripadanje elemenata skupovima. Ovaj filter je zapravo poboljšana verzija 'bloom' filtra.*.*

Cuckoo filter vraća *'false positive'* vrijednosti pripadnosti skupu odnosno može se sa sigurnošću zaključiti da neki element ne pripada pojedinom skupu. Cuckoo filter je minimizirana varijanta *hash* tablica koja koristi *cuckoo hashing* za razrješavanje sukoba tokom popunjavanja tablice. Prostorna kompleksnost se minimizira tako što se zadržava samo *fingerprint* (eng. otisak prsta)od vrijednosti koju želimo dodati setu. *Fingerprint* je broj dobiven od podatka koristeći *hash* funkciju, te je *f-*bitne veličine, gdje se *f* odabere s obzirom koja se vjerojatnost *false positive* rješenja želi. Filter je gusto popunjen sa *fingerprint-*ovima cca 95% što omogućava dobru iskoristivost memorije. Kada se određeni podatak traži, pretražuje se *hash* tablica za njegov *fingerprint* i vraća vrijednost *true* ako je identični *fingerprint* pronađen. Kod konstruiranja filtera, veličina *fingerprint-*a određena je sa željenim *false positive* postotkom koji se označava sa . Što manji želimo, potrebniji je veći *fingerprint.*

Prednosti *cuckoo* filtera u odnosu na *bloom* filter :

* Podržava umetanje i brisanje podataka dinamički
* Ima bolje performanse pretraživanja elemenata, čak i kad je filter gotovo pun npr. Oko 95% popunjenosti
* Jednostavniji je za implementirati od drugih filtera sličnih performansi
* Koristi manje prostora od *bloom* filtera, ako je ciljani *false positive* manji od 3%

Postoje i limitacije *cuckoo* filtera: moguće je izbrisati jedino elemente za koje se zna da su umetnuti prije i umetanje nije uvijek moguće te je onda potrebno ponovno napraviti *hashing.*

## Opis strukture podataka

A close up of a logo

Description automatically generated

Slika 1 Struktura podataka

Gdje je:

m - broj *bucketa*

b - broj ulaza odnosno *fingerprinta* za svaki *bucket*

f - broj bitova korištenih za reprezentaciju *fingerprinta*

## Cuckoo hashing

*Cuckoo hash* tablica se sastoji od polja *buckets-*a (eng. pretinci) gdje svaki podatak

ima dva moguća pretinca u koja može ući, a oni se određuju pomoću *hash* funkcija

. Kada se podatak pretražuje provjere se obje pretinci. Kada se

podatak umeće može doći do dva scenarija :

1. Barem jedan pretinac od dva je slobodan i podatak se umeće u njega
2. Nijedan pretinac nije slobodan, podatak se umeće u jedan od dva nasumično odabrana kandidata, izbacuje podatak koji se nalazi u njemu i taj se podatak onda naziva *victim* (eng. žrtva) i to se sprema na neku alternativnu lokaciju

-ako i sljedeća lokacija nije slobodna, *victim*  se ubacuje u tu lokaciju, a podatak koji je tamo bio postaje novi *victim* i izbacuje se te ide na novu lokaciju

-ovakva procedura se može ponavljati određeni broj puta koji je zadan i to se zove *number of kicks*(engl. broj izbačaja) i ako se i nakon određenog *number of kicks* nemože naći slobodan pretinac filter se smatra punim i nije moguće umetanje

A close up of a device

Description automatically generated A close up of a device

Description automatically generated

Slika 2- a) scenarijj Slika 3 b) scenarijj

ALGORITMI

Algoritmi rade na principu *partial-key cuckoo hashing* što je varijanta standardnog *cuckoo hashinga* koja omogućava dinamički *insert.*

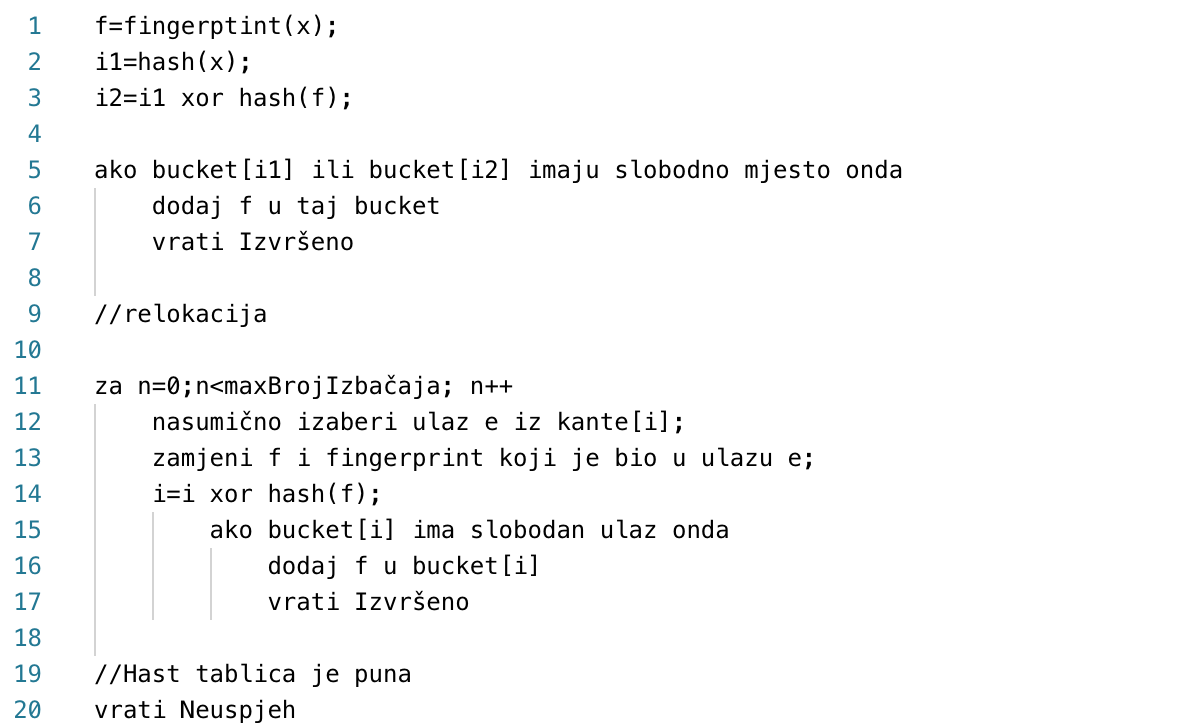
INSERT

Za svaki podatak *x* algoritam računa indekse od dva *bucket* kandidata :

h1(x) = hash(x)

h2(x) = h1(x) ⊕ hash(x’s fingerprint)

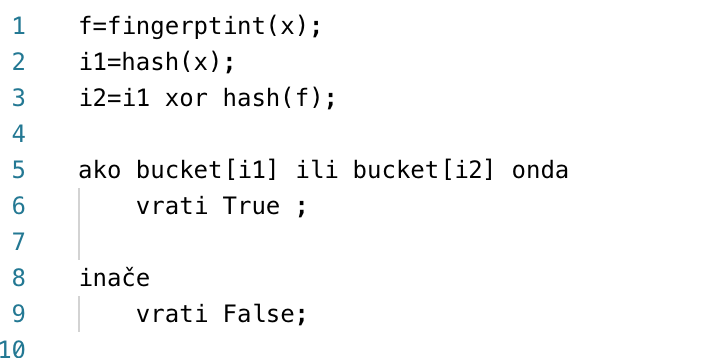
Xor operacija omogućava da se h1(x) može također izračunati od h2(x) i *fingerprint-*a koristeći istu formulu. Odnosno, da stavimo nešto u *bucket* **i**istovremeno računamo i alternativni *bucket* **j**koristeći *bucket* indeksa **i***.* Koristi se *hash*-irani *fingerprint* jer bi inače podaci kada se izbace iz *bucket-*a bili smješteni svi blizu jedni drugih (ako je *fingerprint* mali s obzirom na veličinu tablice). Ovim postupkom se osigurava da se podatci premjeste na potpuno drugi dio tablice kako bi došlo do manje kolizija. Moguće je umetnuti dva ili više podatka s istim *fingerprintom*, ali do *2b* puta, gdje je *b* veličina *bucketa.*



Slika 4 - Pseudokod za insert

LOOKUP

Za podatak x, algoritam prvo računa *fingerprint* i sukladno tome dva *bucketa* koji su kanditati da sadrže x. Ako x postoji u ijednom vraća se pozitivna vrijednost. Zbog ovakvog načina pretraživanja nikad ne dolazi do *false negative* rezultata odnosno ako x stvarno postoji algoritam će ga uvijek pronaći (osim ako ne dođe do *bucket overflow-a*).



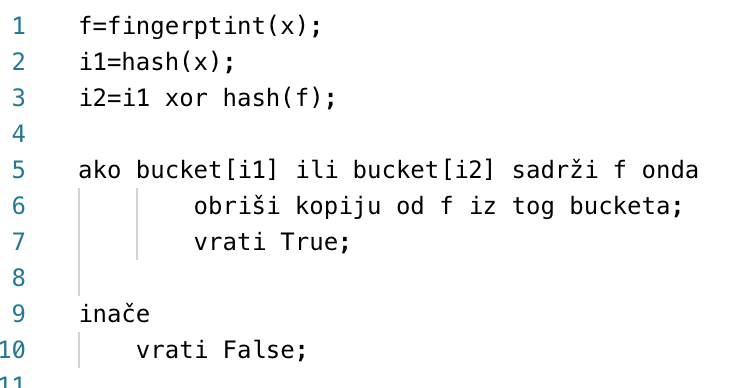
Slika 5 - Pseudokod za lookup

DELETE

Algoritam provjerava oba kandidata gdje bi se podatak mogao nalaziti bazirano na njegovom *fingerprintu* za podatak x. Ako se nađe *fingerprint* u tom *bucketu* tada se ta kopija *fingerprinta* u *bucketu*briše.

U *cuckoo* filteru je izbjegnut problem lažnog brisanja kada dva podatka djele isti *candidate bucket* i imaju isti *fingerprint*. Ako su x i y u *bucket i1* i imaju *fingerprint* *f*  , algoritam *hashi-*inga osigurava da oboje također mogu biti u *bucket i2* zbog toga što je :

Kada se briše x, nije bitno briše li algoritam kopiju *f* od x ili y.



Slika - Pseudokod za delete

## Rezultati

U sklopu ovog projekta implementirali smo vlastiti statički i dinamički *Cuckoo* filter. U nastavku su dani rezultati usporedbe statičkog, dinamičkog, te referentnog *Cuckoo* filtra. Za statički filter korišteni su sljedeći parametri:

* Veličina pretinca = 4;
* Broj pretinca = broj ulaznih k-mera / 4
* Veličina fingerprinta = 16bit;
* Najveći broj izbacivanja = 500;

Za referentni statički filter, ostavljene su zadane vrijednosti. Jedina iznimka je veličina otiska prsta koju smo zbog bolje usporedbe postavili na 16 bitova.

Za dinamički filter, korišteni su sljedeći parametri

* Veličina pretinca = 4;
* Broj pretinca u jednom filtru = 500
* Veličina otiska prsta = 16bit;
* Najveći broj izbacivanja = 500;

Kako bi usporedili performanse filtera, potrebno je pronaći vrijeme izgradnje, memorijsko zauzeće izgrađenog filtera, te *false positive* *rate* nad izgrađenim filterom.

Testovi su izvršeni na sljedeći način. Prvo se filter puni s M k-mera duljine N. Mjerenjem dobivamo vrijeme potrebno za izgradnju filtra. Zatim je vlastitim algoritmom napravljeno 1000 k-mera duljine N koji **sigurno** nisu u skupu originalnih M k-mera (ali su i dalje iste duljine i formata) nad kojima je izgrađen filter. Zatim se nad tih 1000 k-mera zove funkcija lookup() filtra. Broj pozitivno vraćenih vrijednosti dijeli se s 1000, te tako dobivamo *false* *positive rate*. U svakom filteru je također implementirana funkcija koja vraća zauzeće memorije tog filtra. Tako dobivamo zauzeće memorije.

U nastavku slijede 3 slike. Na svakoj slici se prikazuje jedna metrika (vrijeme, memorija, fp-rate). Također, na jednoj slici se nalaze 3 grafa: Za M=10000, M=10000, te M=1000000. Na svakom grafu je prikazana metrika te slike za 3 filtra – Referentni statički, vlastito implementirani statički, te vlastito implementirani dinamički filter.

A screenshot of a cell phone

Description automatically generated

Slika - Vrijeme izvođenja insert-a za različit broj ulaznih k-mera u filtru. X – os duljina k-mera, Y – os, vrijeme izvođenja (u ms).

Komentar: Rezultati su očekivani. Za manji broj ulaznih k-mera vrijeme izgradnje filtra je zanemarivo. Kako broj k-mera raste, vidi se nedostatak statičkog filtra u odnosu na dinamički. Za velik broj k-mera dolazi do velikog broja izbacivanja(kicks) prilikom ubacivanja novog člana zbog nemogućnosti internog hash algoritma da se nosi s tolikom veličinom filtra.

A picture containing stationary, writing implement

Description automatically generated

Slika - Iskorištena memorija svakog filtra. X – os duljina k-mera, Y – os, Zauzeće memorije (u Bajtovima).

Komentar: Zauzeće memorije je konstantno i ovisi isključivo o veličini filtra, koja je kod nas u svakom slučaju ista, jer smo te parametre fiksirali. Može se primijetiti da je referentna implementacija lošija po pitanju memorije, no do toga dolazi jer se memorija alocira pri prevođenju, na najmanju veću potenciju broja 2, dok se u našoj vlastitoj implementaciji podatci alociraju dinamički. Također, dinamički filter koristi neznatno više memorije zbog dodatnih struktura koje omogućuju dinamičku alokaciju novih filtera.

A screenshot of a cell phone

Description automatically generated

Slika - Količina False positive primjera. X – os duljina k-mera, Y – os, Postotak False positive (u Bajtovima).

Komentar: Vidljivo je da je *false positive rate* kod statičkih filtera gotovo uvijek jednak 0 ili vrlo malen. To je za očekivati, jer je prilikom biranja parametara odabran dovoljno velik fingerprint (16bit). Međutim, kod dinamičkog filtera, pošto smo odabrali izuzetno malen, fiksan broj pretinaca po pojedinom filtru. Nad 1 000 000 različitih k-mera, unutarnja hash funkcija redovito vraća slične pretince, što uzrokuje velik *false pozitive* rate. Naime, k-mer koji se **sigurno** ne nalazi u originalnom setu nad kojim je izgrađen filter, vjerojatnije ima isti hash (kako raste broj ulaznih k-mera) kao i k-mer koji se nalazi.

# Zaključak

Implementirani filteri ponašaju se shodno očekivanjima. Referentna implementacija je malo efikasnija u vidu trajanja insert operacije pa samim time i izgradnje filtera. Također, za razliku od referentne implementacije, naša verzija ne podržava parametrizirano mijenjanje veličine *fingerprinta* u radnoj memoriji računala. Implementacija toga, zahtijevala bi korištenje struktura koje rade direktno s bitovima. U vidu potrošnje memorije, referentna implementacija koristi *compile-time* fiksnu alokaciju, te time zauzeće odgovara najmanjoj potenciji broja 2 većoj od veličine filtra. Naša implementacija memoriju alocira dinamički. *False* *pozitive* rate je za statičke filtre s dobro odabranim parametrima vrlo blizu nule. Isto vrijedi i za dinamički filter, no radi ilustracije, pokazali smo kako se *false pozitive* *rate* povećava kako raste broj unutarnjih filtera u dinamičkom filteru.

# Literatura

<https://brilliant.org/wiki/cuckoo-filter/>

<https://bdupras.github.io/filter-tutorial/>

<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8117563&tag=1>

<http://www.cs.cmu.edu/~binfan/papers/conext14_cuckoofilter.pdf>

<https://ieeexplore.ieee.org/abstract/document/8117563>