SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

BIOINFORMATIKA 1

Seminar

Smanjenje premještanja u Cuckoo filtru

Iva Bokšić
Karolina Mirković
Voditelj: *Mirjana Domazet-Lošo*

Zagreb, svibanj 2022.

SADRŽAJ

1. Uvod

Pohrana velikog broja podataka te pretraživanje podskupova u podatcima predstavljali su problem dugi niz godina. S napretkom tehnologije kapacitet memorije za pohranjivanje narastao je i do nekoliko TB. Time je prividno riješen problem pohrane podataka, ali je ostao neriješen problem pretraživanja. Da bi se pronašao određeni podskup u podatcima moralo bi se redom proći po cijeloj dužini pohranjenoga podatka što predstavlja vremenski vrlo zahtjevan posao. U nadi da se riješe problemi i pohrane i pretraživanje podataka javljaju se brojni algoritmi među kojima je i Cuckoo filter, koji je nastao iz prethodnog Bloom filtra kao njegova poboljšana verzija. Cuckoo filter pokazao je značajne prednosti u pretraživanju podskupova te brisanju podataka u usporedbi s Bloom filtrom iz kojeg je nastao. Iako je poboljšana verzija i sam Cuckoo filter pati od nekih problema. Kako bi pohranio podatke Cuckoo filter koristi takozvane pretince ili košare (*engl.* bucket) koje odabire metodom slučajnog odabira što može dovesti do njihove prepunjenosti te velikog broja premještanja čime mu se performanse drastično smanjuju.

Tema ovog seminara se temelji na implementaciji Cuckoo filtera i pokušaju poboljšanja njegovih performansi pri premještanju podataka. U drugom poglavlju objašnjen je princip rada orginalnog Cuckoo filtra te njegove poboljšane verzije koja smanjuje broj premještanja. Treće poglavlje razmatra rezultate koji su dobiveni pri analizi performansa izgrađenih Cuckoo filtera. Četvrto poglavlje donosi zaključak je li moguće izgraditi uspješan Cuckoo filter s manjim brojem premještanja i jesu li se performanse drastično promijenile. Popis literature korištene pri izradi rada se nalazi u petom poglavlju. Šesto poglavlje predstavlja sažetak cijelog seminara.

2. Cuckoo filter

Cuckoo filter (**CF**) predstavlja vjerojatnosnu strukturu podataka koja se koristi pri pohrani podataka te ispitivanju postoji li se u podatku određeni podskup. Prilikom ispitivanja moguće je postojanje lažno pozitivnih podudaranja odnosno moguće je da upit vrati da se podskup nalazi u podatku iako se on ne nalazi. S druge strane nije moguće postojanje lažno negativnih rezultata upita što znači da je nemoguće da podskup postoji u podatkovnoj strukturi, a da upit vrati da ga nije pronašao. Cuckoo filter odmogućava i brisanje podataka iz svoje stukture što ga razlikuje od Bloom filtra iz kojeg je sam i nastao. Iako može vratiti lažno pozitivne rezultate prostorna učinkovitost pohranjivanja podataka te brzina pronalaska podskupova mu omogućuju dobre performanse. Glavne četiri prednosti Cuckoo filtra su:

• podržano dinamičko dodavanje i uklanjanje stavki
• pružanje veće izvedbe pretraživanja
• laka implementacija za razliku od alternativa
• korištenje manje prostora od Bloom filtra ako je stopa lažno pozitivnih manja od 3%.

2.1. Podatkovna stuktura Cuckoo filtra

Cuckoo filter je dvodimenzionalna podatkovna stuktura koja se sastoji od košara ili pretinaca u koje se može unjeti unaprijed definiran broj podataka. Stuktura se može predočiti tablicom u kojoj redci predstavljaju košare, a broj stupaca odgovara zadanom broju podataka koje se u njega mogu pohraniti. CF zapravo ne pohranjuje izvorne podatke već koristi hash funkciju kako bi izračunao "otisak prsta" (*engl.* fingerprint) te također koristi hash funkciju kao bi odabrao broj pretinca odnosno poziciju unutar tablice u koju će podatak pohraniti. Prilikom konstruiranja CF veličina otiska određena je ciljanom stopom lažno pozitivnih rezultata. Manje stope lažno pozitivnih zahtijevaju dulje otiske jer je time moguće postići da više podataka ima različiti otisak.

2.2. Umetanje podataka u Cuckoo filter

Kako bi se pohranio podatak u CF najprije je potrebno izračunati njegov otisak i broj pretinca pomoću hash funkcije. Svakom podatku dodijeljena su dva moguća pretinca u koje ih je moguće pohraniti. Kako bi se dobili brojevi pretinaca potrebno izračunati hash vrijednost od samog podatka *x* te tu istu vrijednost provući kroz XOR operator zajedno s hash vrijednosti otiska podatka:

$$h_1(x) = hash(x) \tag{2.1}$$

$$h_2(x) = h_1(x) \oplus hash(otisak\ od\ x) \tag{2.2}$$

Hashiranje otiska podatka osigurava da pozicije pretinaca se nalaze na različitim mjestima unutar tablice čime se smanjuje broj kolizija prilikom umetanja te se poboljšava iskorištenost tablice.

Kako brojevi pretinaca ne bi prelazili unaprijed definiran ukupan broj pretinaca m, potrebno je za vrijednosti h_1 i h_2 izračunati modulo m.

$$index_1(x) = h_1(x) \mod m \tag{2.3}$$

$$index_2(x) = h_2(x) \mod m \tag{2.4}$$

S obzirom na popunjenost pretinaca na pozicijama i_1 i i_2 razlikuju se tri varijante umetanja podataka.

2.2.1. Oba pretinca su slobodna

Najjednostavniju varijantu predstavlja slučaj kada su oba pretinca slobodna odnosno imaju praznih mjesta u koje se može pohraniti otisak podatka. U tom slučaju slučajnim odabirom se bira koji će to pretinac biti. U pretinac se pohranjuje novi podatak te funkcija umetanja vraća informaciju da je novi podatak uspješno pohranjen.

2.2.2. Jedan pretinac je slobodan

U slučaju da samo jedan pretinac ima slobodnih mjesta za pohranu on se odabire i u njega se smješta otisak podatka. Funkcija umetanja također vraća informaciju da je novi podatak uspješno pohranjen.

2.2.3. Oba pretinca su popunjena

Najsloženiji slučaj je kada su oba pretinca popunjena. Tada se slučajnim odabirom izabire jedan od dva puna pretinca te iz njega izbacuje jedan otisak te se na njegovo mjesto pohranjuje novi otisak. Za izbačeni otisak računa se alternativna pozicija na koju se on može spremiti prema izrazu:

$$h_i(r) = index(x) \oplus hash(otisak\ od\ r) \tag{2.5}$$

gdje *x* predstavlja novi podatak koji je potrebno pohraniti u podatkovnu strukturu, a *r* označava podatak koji je izbačen iz pretinca kako bi se *x* u njega spremio. Pozicija na koju se *r* treba pokušati spremiti računa se prema izrazu **(2.6)**.

$$index(r) = h_i(r) \mod m \tag{2.6}$$

Ako pretinac na izračunatoj poziciji *index(r)* nije pun podatak *r* se pohranjuje u njega i algoritam završava. U slučaju da je pun algoritam nastavlja petlju u kojoj izbacuje jedan podatak iz pretinca računa njegovu alternativnu poziciju i pokušava ga smjestiti u pretinac novoizračunate pozicije. Ovisno je li pretinac novoizračunate pozicije slobodan ili ne algoritam se prekida ili nastavlja pretragu sve dok ne pronađe slobodni pretinac. Ovakav sistem nažalost ima i negativnu stranu, a to je da može zapeti u beskonačnoj petlji gdje se podaci redom izbaciju iz pretinaca, a sve alternativne pozicije pretinaca su pune. Kako bi se to spriječilo potrebno je definirati unaprijed zadani broj koji se označava kao **MNK** te predstavlja najveći broj dozvoljenih premještanja podataka unutar stukture da bi se novi podatak pohranio. Ako se dođe do MNK vrijednosti, zadnje izbačeni podatak iz pretinca se izbacuje iz cijele podatkovne strukture CF-a te funkcija umetanja vraća informaciju da podatak nije uspješno umetnut.

2.3. Smanjenje premještanja podataka

3. Rezultati

4. Zaključak

Zaključak.

5. Literatura

6. Sažetak

Sažetak.