SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Dokumentacija

Logaritamski dinamični *cuckoo* filter

Petra Habjanec i Andrea Milanović

Zagreb, svibanj, 2024.

Sadržaj

1. Uvod

U današnje vrijeme, s pojavom aplikacije koje obrađuju velike količine podataka, javlja se problem reprezentacije skupova. Organiziranjem elemenata skupa pomoću određene strukture podataka, reprezentacija skupa omogućuje pristup informacijama o elemenetima, podržavajući operacije kao što su umetanje, provjera članstva i brisnaje podataka. Vremenski i prostorno učinkovita struktura podataka za reprezentaciju skupa važna je u raznim primjenama aplikacija, kao što su pohrana na *cloud*-u, zaštita privatnosti, brojanje k-mera u sekvenciranju DNK i pretražuvanju mreže.

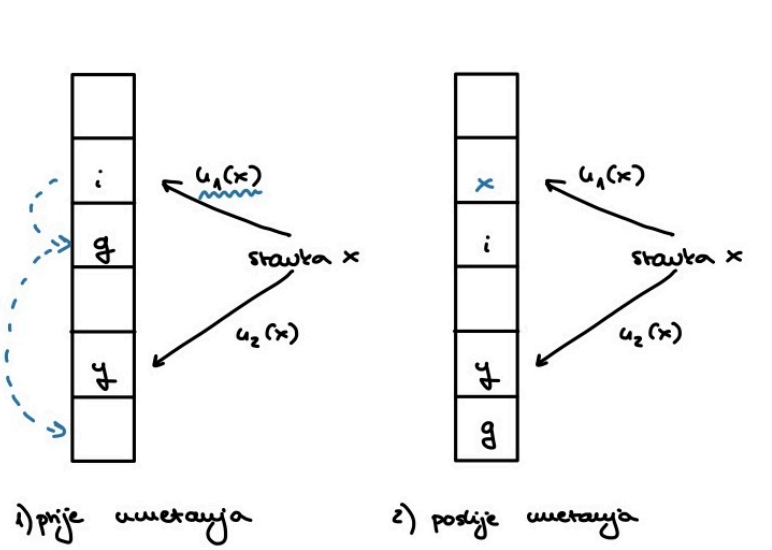
Logaritamski dinamični c*uckoo* filter predstavlja inovativno rješenje za ove izazove, nudeći poboljšane perfomanse kod vremena pristupa i utroška memorije. Ovaj algoritam koristi prednosti c*uckoo hash*-iranja s naprednin skupom podataka - binarnim stablom. Cilje je ove dokumentacije objasniti principe rada Logaritamskog dinamčnog c*uckoo* filtera, te našu implementaciju istog.

2. Logaritamski dinamički *cuckoo* filter

2.1. *Cuckoo hash* tablica

Cuckoo hash tablica[1] se sastoji od liste *l* kanti, svaka od kanti je jedinica u koju se pohranjuje stavka. Za svaku se stavku računaju dva neovisna *hash*-a koji označavaju dvije opcije kanti u koje možemo staviti stavku. Proces umetanja stavke izgleda ovako, gdje x označava stavku:

1) Tablica *cuckoo hash*-eva računa kandidate kanti adresa $h_1(x)$ i $h_2(x)$.
2) Ukoliko je bilo koja od $h_1(x)$ i $h_2(x)$ kanti prazna, spremamo stavku u praznu kantu, no ako ni jedna od njih nije prazna slučajno biramo jednu od njih, te umećemo stavku u nju.
3) Sada imamo jednu stavku koja je izbačena iz svog mjesta. Tu stavku zovemo žrtva te nju stavljamo u sebi alternativnu kantu.
4) Ukoliko je alternativna kanta prazna, stavka se stavlja u nju i proces je gotov, no ako nije vraćamo se na korak 3) te ga ponavljamo.[2]



Slika 1: Cuckoo hash tablica

Na slici 1 možemo vidjeti primjer umetanja stavke x u *cuckoo hash* tablicu. Vidimo da su obje moguće kante $h_1(x)$ i $h_2(x)$ pune te tako moramo slučajno jednu iz koje ćemo izbaciti stavku. U ovom slučaju biramo kantu $h_1(x)$ i tako izabacujemo stavku i, i tako postaje žrtva. Ona mora ići na svoje druge moguće mjesto u tablici, no na tom se mjesto nalazi stavka g. Na ovaj način g postaje sljedeća žrtva te se onda mora *preseliti* na svoju drugu opciju, koja je u ovom slučaju na sreću prazna. Tako dobivamo krajnje stanje tablice nakon umetanja stavke x.

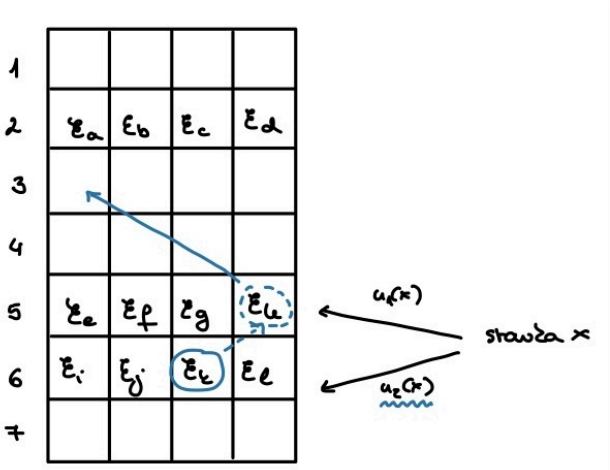
2.2. *Cuckoo* filter

Zamijenimo li originalnu stavku x s otiskom stavke(kasnije zvano ξ_x) dobivamo c*uckoo* filter. Tako preoblikovana *hash* tablica zauzima manje mjesta nego pohranjivanje samih stavaka. *Cuckoo* filter se sastoji od lista kanti veličine dužine *l*, svaka od kanti ima određen broj mjesta, *b*.

Kako ovdje je pohranjujemo same podatke već samo otiske podataka, javlja se problem pronalaženja alternativnih *hash*-eva za relociranje podataka. Kako bismo rješili ovaj problem, c*uckoo* filter se temelji na c*uckoo* *hash*-iranju djelomičnih ključeva. Takvim se *hash*-iranjem alternativna adresa računa XOR operacijom temeljenom na adresi trenutne kante i otiska.[3]

$$h_1(x) = hash(x)$$
$$h_2(x) = h_1(x) \oplus hash(_x)$$

Na slici 2 možemo vidjeti primjer umetanja u *cuckoo* filter. Na ovome gledamo u filter koji ima 7 kanti te svaka kanta ima zapremninu 4 stavke. Gledamo kako umećemo stavku x. Gledamo moguće kante $h_1(x)$ i $h_2(x)$, no obje kante su pune. Slučajnim odabirom biramo da ćemo stavku x umetnuti na $h_2(x)$, ali kako je ta kanta puna biramo jednu stavku koju ćemo izbaciti iz te kante, u ovom slučaju to je ξ_k, ta stavka sada postaje žrtva. ξ_k sada mora ići na svoje alternativno mjesto, ali je ta kanta već popunjena tako da i iz te kante moramo izbaciti slučajno odabranu stavku iz te kante. Sada bez mjesta ostaje ξ_h koji odlazi onda na svoje alternativno mjesto koje ima slobodnih mjesti u kanti. Tako da je sada proces umetanja u filter dovršen.

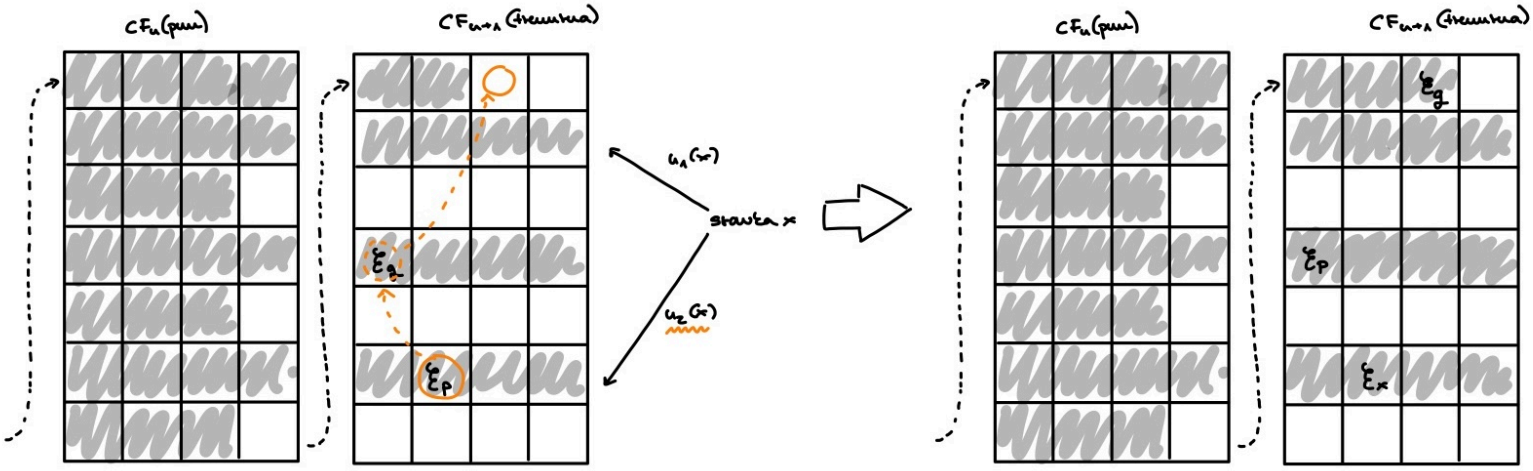


Slika 2: *Cuckoo* filter

2.3. Dinamični c*uckoo* filter

Kako bi se poduprijele velike količine podataka, smišljena je struktura dinamičnog c*uckoo* filtera, koji se sastoji od homogenih *cuckoo* filtera koji su povezani[4]. Svaki blok *cuckoo* filtera koristi brojač elemenata koji su pohranjeni u njemu, ukoliko je ta broj manji od predefiniranog kapaciteta taj se blok gleda kao aktivan. Kod umetanja u dinamični *cuckoo* filter prvo se pokušava umetnuti u prvi

aktivan blok, te ukoliko to ne uspije, dodaje se novi blok u kojeg se onda umeće nova stavka. Provjera pripadnosti dinamičnom *cuckoo* filteru se radi tako da prolazimo kroz listu te provjeravamo pripadnost otiska stavke svakog od blokova *cuckoo* filtera. Ukoliko prođemo sve blokove i ne pronađemo otisak koji odgovara otisku tražene stavke, funkcija vraća negativan odgovor. Ukoliko u bilom kojem od blokova u nizu pronađemo otisak koji odgovara otisku tražene stavke, funkcija vraća pozitivan odgovor. Brisanje se provodi prolaženjem kroz blokove te pokušajem brisanja u bloku, sve dok za jedan od blokova ne dobijemo pozivan odgovor da je izbrisan otisak koji odgovara otisku stavke koju pokušavamo obrisati.



Slika 3: Umetanje stavke u DCF

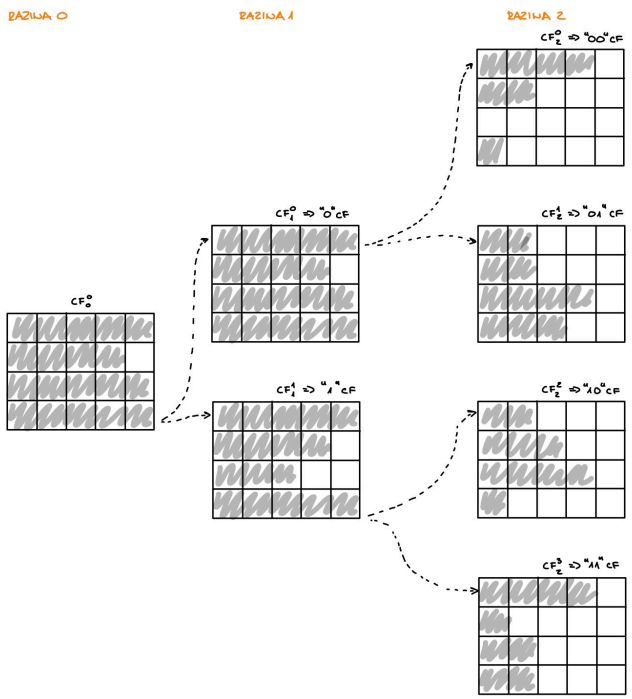
Na slici 3 možemo vidjeti primjer umetanja u dinamični *cuckoo* filter. Gdje umećemo stavku x, te ima izbor kanti $h_1(x)$ i $h_2(x)$, no obje kante na izbor su pune, tako da standardnim postupkom za *cuckoo* filter slučajno biramo između $h_1(x)$ i $h_2(x)$ - u ovome primjeru biramo $h_2(x)$. Tako da jednoga izbacujemo iz kante $h_2(x)$, u ovome je slučaju to ξ_p. ξ_p sada postaje žrtva te mora ići na svoje alternativno mjesto. Nažalost kanta s njegovim alternativnih mjestom je puna te se tako slučajnim odabirom ξ_q izbacuje iz kante, te on postaje žrtva i traži svoju alternativnu kantu. U njegovoj alternativnoj kanti ima mjesta te se smješta tamo i tako završava postupak.

2.4. Logaritamski dinamični *cuckoo* filter

Logaritamski dinamični *cuckoo* filter[2], u usporedbi s dinamičnim *cuckoo* filterom, koristi strukturu binarnog stabla. Takva struktura omogućava značajno nižu vremensku kompleksnost umetanja i provjere pripadnosti stavaka. Na slici 4 možemo vidjeti primjer izgleda LDCF-a.

Na početku LCDF započinje s jednim *cuckoo* filterom, u kojeg normalno umećemo otiske stavaka, sve dok jedno od umetanja ne uspije. Kada jedno umetanje ne uspije, zaključujemo da u taj filter više ne možemo umetati otiske stavaka. Ovisno o tome započinje li otisak stavke bitom nule ili jedinice, biramo hoće li taj otisak ići u novu lijevu ili desnu granu stabla. Nova je grana stabla isti cuckoo

filter kao i prijašnji no otisak koji se pohranjuje u njega je za jedan bit kreći. Kako određujemo poziciju u otiska u stablu ovisno o bitovima ostisaka, bitove kojima smo određivali poziciju u stablu ne moramo pohranjivati. Na taj način smanjujemo i memorijsko opterećenje našeg algoritma, pogotovo kod podataka velikih razmjera. Isto tako ovaj nam algoritam daje mogućnost korištenja dužih otisaka, kojima se smanjuju razmjeri lažnih negatnivih kod provjere pripadnosti.



Slika 4: Primjer izgleda LDCF-a

Kod provjere pripadnosti, na svakoj će razini samo jedan blok *cuckoo* filtera biti provjeren, te se tako jako smanjuje vremenska kompleksnost samog algoritma. U usporedbi s već prije spomenutim DCF algoritmom kojim ima vremensku kompleksnost $O(2^l - 1)$, LDCF $O(l)$. Time je vremenska složenost LDCF-a logaritamskom skalom. Iako LDCF ima veće vrijeme računanja, to je vrijeme za računanje mali trošak za platiti s tim da nemamo veliki broj selidba kod umetanja u blok *cuckoo* filtera[5].

2.4.1. Analiza točnosti, vremena izvođenja i utroška memorije

Svi su testovi izvršeni na LDCF-u s *cuckoo* filterima kapaciteta 20, kanti veličine 10 i maksimalnim brojem premještaja od 5. Duljine se otisaka stavaka mijenja kroz test primjere, što se može vidjeti i u tablicama.

U ovome smo radu sagledali dinamični *cuckoo* filter kao inovativno rješenje za efikasnu reprezentaciju velikih skupova podataka. U uvodnom dijelu, postavljamo temelje razumijevanja problematike velikih podataka i potrebe za vremenski i prostorno učinkovitim algoritmima i strukturama podataka.

test_ecoli

k	duljina otiska	duljina genoma	broj uspješno umetnutih	broj netočno negativnih kod pronalaska	postotak netočno pozitivnih	vrijeme umetanja[ms]	vrijeme pronalaženja[ms]	vrijeme pronalaženja netočno pozitivnih(100)[ms]	vrijeme brisanja[ms]	memorija[kb]
10	8	4641652	464165	0	100%	1902421	306066	1021	307798	13912
10	16	4641652	464165	0	100%	1738187	1591029	531	1632264	20820
20	8	4641652	232082	0	100%	996570	196150	479	166256	13900
20	16	4641652	232082	0	99%	714848	849130	610	873111	16916
50	8	4641652	92833	0	100%	435725	105314	1220	107655	13920
50	16	4641652	92833	0	78%	320324	464520	905	467423	14896
100	8	4641652	46416	0	100%	235436	72358	1536	71395	13892
100	16	4641652	46416	0	45%	180747	303710	1487	303507	14208
200	8	4641652	23208	0	100%	128478	50969	2269	51033	13912
200	16	4641652	23208	0	32%	105942	195047	2094	196202	13788

Tablica 1: Rezultati testa za genom E. Coli

test3

k	duljina otiska	duljina genoma	broj uspješno umetnutih	broj netočno negativnih kod pronalaska	postotak netočno pozitivnih	vrijeme umetanja[ms]	vrijeme pronalaženja[ms]	vrijeme pronalaženja netočno pozitivnih(100)[ms]	vrijeme brisanja[ms]	memorija[kb]
10	8	1000	100	0	29%	45	37	299	34	2720
10	16	1000	100	0	0%	43	37	156	35	2704
20	8	1000	50	0	18%	20	20	392	19	2704
20	16	1000	50	0	0%	22	21	197	19	2704
50	8	1000	20	0	11%	14	13	516	12	2704
50	16	1000	20	0	0%	13	13	337	11	2704
100	8	1000	10	0	2%	10	10	650	8	2704
100	16	1000	10	0	0%	10	10	592	9	2704
200	8	1000	5	0	0%	7	7	1380	6	2724
200	16	1000	5	0	0%	7	7	1048	5	2704

Tablica 2: Rezultati testa za simulirane podatke veličine 10^3

test4

k	duljina otiska	duljina genoma	broj uspješno umetnutih	broj netočno negativnih kod pronalaska	postotak netočno pozitivnih	vrijeme umetanja[ms]	vrijeme pronalaženja[ms]	vrijeme pronalaženja netočno pozitivnih(100)[ms]	vrijeme brisanja[ms]	memorija[kb]
10	8	10000	1000	0	97%	895	560	323	571	2748
10	16	10000	1000	0	2%	857	916	267	930	2740
20	8	10000	500	0	88%	376	278	380	280	2740
20	16	10000	500	0	0%	376	370	293	370	2740
50	8	10000	200	0	41%	167	143	569	139	2736
50	16	10000	200	0	1%	163	153	403	158	2740
100	8	10000	100	0	39%	85	83	696	87	2736
100	16	10000	100	0	1%	84	83	578	82	2736
200	8	10000	50	0	12%	64	59	1205	60	2736
200	16	10000	50	0	0%	58	58	1042	56	2724

Tablica 3: Rezultati testa za simulirane podatke veličine 10^4

test5

k	duljina otiska	duljina genoma	broj uspješno umetnutih	broj netočno negativnih kod pronalaska	postotak netočno pozitivnih	vrijeme umetanja[ms]	vrijeme pronalaženja[ms]	vrijeme pronalaženja netočno pozitivnih(100)[ms]	vrijeme brisanja[ms]	memorija[kb]
10	8	100000	10000	0	100%	19652	6324	342	6418	3288
10	16	100000	10000	0	9%	16643	20436	403	20605	3252
20	8	100000	5000	0	100%	7197	3636	535	3667	3168
20	16	100000	5000	0	5%	7292	9118	482	9065	3208
50	8	100000	2000	0	100%	2986	2066	636	2039	3128
50	16	100000	2000	0	1%	3108	3996	583	3984	3140
100	8	100000	1000	0	99%	1497	1334	850	1328	3120
100	16	100000	1000	0	0%	1558	2172	879	2140	3128
200	8	100000	500	0	80%	843	827	1282	835	3116
200	16	100000	500	0	0%	839	1122	1355	1121	3116

Tablica 4: Rezultati testa za simulirane podatke veličine 10^5

test6

k	duljina otiska	duljina genoma	broj uspješno umetnutih	broj netočno negativnih kod pronalaska	postotak netočno pozitivnih	vrijeme umetanja[ms]	vrijeme pronalaženja[ms]	vrijeme pronalaženja netočno pozitivnih(100)[ms]	vrijeme brisanja[ms]	memorija[kb]
10	8	1000000	100000	0	100%	399673	67321	1023	67294	6748
10	16	1000000	100000	0	81%	268195	305545	494	309717	7844
20	8	1000000	50000	0	100%	194062	35265	1031	35163	6704
20	16	1000000	50000	0	63%	125395	158414	597	155988	7020
50	8	1000000	20000	0	100%	71378	22612	1202	22226	6692
50	16	1000000	20000	0	30%	50483	77831	838	76727	6568
100	8	1000000	10000	0	100%	32163	14891	1429	15365	6408
100	16	1000000	10000	0	11%	28663	48788	1258	51703	6416
200	8	1000000	5000	0	100%	15807	10957	2108	10743	6320
200	16	1000000	5000	0	7%	16236	29619	1923	32548	6324

Tablica 5: Rezultati testa za simulirane podatke veličine 10^6

test7

k	duljina otiska	duljina genoma	broj uspješno umetnutih	broj netočno negativnih kod pronalaska	postotak netočno pozitivnih	vrijeme umetanja[ms]	vrijeme pronalaženja[ms]	vrijeme pronalaženja netočno pozitivnih[100][ms]	vrijeme brisanja[ms]	memorija[kb]
10	8	10000000	1000000	0	100%	4236839	663262	488	674296	36052
10	16	10000000	1000000	0	100%	4039263	3830902	1286	3631466	48288
20	8	10000000	500000	0	100%	2121752	371563	512	367344	34988
20	16	10000000	500000	0	99%	1643173	1870392	574	1882340	41828
50	8	10000000	200000	0	100%	948209	222644	1345	225583	35088
50	16	10000000	200000	0	95%	782771	1051208	902	1047218	37776
100	8	10000000	100000	0	100%	538848	157864	966	160930	35064
100	16	10000000	100000	0	77%	446413	722295	1381	723963	36220
200	8	10000000	50000	0	100%	310949	111471	1632	112083	35060
200	16	10000000	50000	0	52%	265791	484264	2188	485085	35400

Tablica 6: Rezultati testa za simulirane podatke veličine 10^7

3. Zaključak

Prolaskom kroz razvoj od osnovne cuckoo hash tablice, preko cuckoo filtera i dinamičnog cuckoo filtera, došli smo do LDCF-a, koji predstavlja značajan napredak u smislu performansi. Korištenjem binarnog stabla u strukturi LDCF-a, postignuta je smanjena vremenska složenost operacija umetanja i provjere pripadnosti.

Kod analize točnosti, vremenske i prostorne učinkovitosti možemo jasno vidjeti da se očitavaju prednosti LDCF-a, s povećanjem duljine otiska, smanjuje se postotak lažnih pozitivnih, no zauzeta memorija se previše ne povećava.

4. Literatura

[1] R. Pagh and F. F. Rodler, “Cuckoo hashing,” Journal of Algo- rithms, vol. 51, no. 2, pp. 122–144, 2004.

[2] F. Zhang, H. Chen, H. Jin, and P. Reviriego, "The Logarithmic Dynamic Cuckoo Filter," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, Wuhan, China, 2021, pp. 948-959. Huazhong University of Science and Technology, National Engineering Research Center for Big Data Technology and System, Cluster and Grid Computing Lab, Services Computing Technology and System Lab, School of Computer Science and Technology; Universidad Carlos III de Madrid, Departamento de Ingeniería Telemática.

[3] B. Fan, D. G. Andersen, M. Kaminsky, and M. Mitzenmacher, “Cuckoo filter: Practically better than bloom,” in Proceedings of CoNEXT, 2014.

[4] H. Chen, L. Liao, H. Jin, and J. Wu, “The dynamic cuckoo filter,” in Proceedings of ICNP, 2017.

[5] F. Wang, H. Chen, L. Liao, F. Zhang, and H. Jin, “The power of better choice: Reducing relocations in cuckoo filter,” in Proceedings of ICDCS, 2019.