

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

**Vlastita implementacija filtera:
The Logarithmic Dynamic Cuckoo
Filter**

Fran Ostroški, Elena Wachtler

Voditelj: *Mirjana Domazet-Lošo*

Zagreb, svibanj 2023.

SADRŽAJ

1. Uvod	1
2. Opis algoritma	2
3. Analiza	7
4. Zaključak	10
5. Literatura	11
6. Sažetak	12

1. Uvod

Ovaj seminar opis je projekta u sklopu kolegija Bioinformatika 1 Fakulteta elektrotehnike i računarstva Sveučilišta u Zagrebu, a predstavlja vlastitu implementaciju takozvanog cuckoo filtera, točnije njegovu dinamičku logaritamsku varijantu.

Cuckoo filter probabilistička je struktura podataka koja se koristi za određivanje pripadnosti zadanog elementa nekome skupu. Sam filter nastao je kao proširenje već postojećih Bloom filtera, a naziv je dobio prema ptici kukavici koja izbacuje jaja iz tuđih gnijezda kako bi ubacila svoja. Iako je riječ o općenitim podacima, cuckoo filter često se primjenjuje u bioinformatici jer je pogodan za određivanje pripadnosti podnizova nukleotidnih slijedova nekom većem nizu nukleotida ili provjeru nalaze li se podnizovi jednog slijeda u nekom drugom slijedu.

Pripadnost elementa nekom skupu ovom strukturom ne možemo garantirati, jer zbog implementacije algoritma postoji malena vjerojatnost za lažno pozitivne (eng. *false positive*) i lažno negativne (eng. *false negative*) rezultate. Upravo zbog toga što postoji određena vjerojatnost pogreške, cuckoo filter je probabilistički. Međutim, nepripadnost nekom skupu može se pouzdano odrediti.

2. Opis algoritma

Nastanak i evolucija Cuckoo filtera

Cuckoo filtere kao ideju za rješavanje problema koji zahtjevaju nisku stopu lažno pozitivnih primjera predstavili su Fan, Kaminsky i Andersen [2], kao strukturu čija je glavna prednost u odnosu na dotad korištene Bloom filtere sposobnost brisanja. Čak štoviše, uvode cuckoo filtere kao strukturu koja ne samo da omogućuje dinamičko dodavanje i brisanje objekata, već postiže bolje *lookup* performanse, pri tome koristeći manje prostora od dotad optimalnih Bloom filtera. Teoriju koju su predložili kasnije su i dokazali [3] - cuckoo filteri uistinu su nadmoćna struktura za pohranu velikog broja objekata, kada je nužan uvjet niska stopa lažno pozitivnih primjera. Uz to, dodana su i poboljšanja kako bi se smanjila prostorna složenost, i to djelomičnim sortiranjem bucketa te optimizacijom veličine bucketa - najboljim se pokazao (2, 4)-cuckoo filter, što znači da svaki objekt ima dva moguća kandidata za bucket, a svaki bucket u sebi sadrži podatke o četiri otiska (eng. *fingerprint*). O bucketima je više rečeno u nastavku poglavlja gdje je dan detaljan opis rada algoritma uz primjer.

Korak dalje učinili su Chen, Liao, Jin i Wu [1] uvođenjem dinamičkog cuckoo filtera, koji bi istovremeno zadovoljavao i uvjet da se veličina strukture može fleksibilno mijenjati, ali i da operacija brisanja bude pouzdana. Autori tvrde da ovakva implementacija cuckoo filtera koja u sebi sadrži povezanu listu filtera ostvaruje 75% smanjeno zauzeće memorije, poboljšanje u konstrukciji strukture od 50% te ubrzanje u operaciji *query* od čak 80%. Također, kao dodatnu optimizaciju uvode gornje ograničenje na broj relokacija. Relociranje je vizualizirano u nastavku poglavlja gdje je algoritam primijenjen na jednostavan primjer.

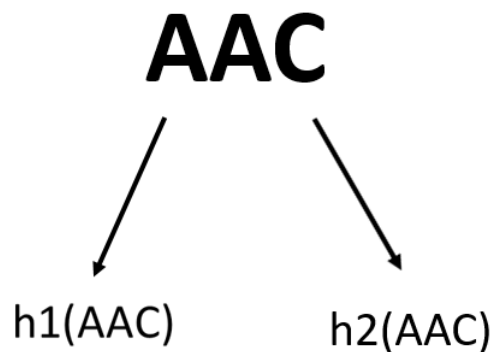
Posljednju nadogradnju ostvarili su Zhang, Chen, Jin i Reviriego [4] ističući linearno rastuću računalnu i prostornu složenost s rastom veličine prostora podataka kao glavni nedostatak dinamičkih cuckoo filtera, te uvodeći višerazinsko binarno stablo kao rješenje tog nedostatka, zbog čega se njihova struktura naziva logaritamskim dinamičkim cuckoo filterom, LDCF. S ovim poboljšanjem, operacije dodavanja novih

objekata i provjera nalazi li se objekt u strukturi (eng. *membership testing*) od složenosti $O(N)$ (gdje je N veličina skupa podataka) padaju na složenost $O(1)$. Dodatno, ponudili su i varijantu kompaktnog LDCF-a u kojoj je dodatno smanjeno prostorno zauzeće, na način da se zadržava samo najviša razina cuckoo filtera u višerazinskom stablu. U ovome radu u poglavlju 3 analizirane su performanse vlastite implementacije upravo ove, najviše unaprijeđene varijante cuckoo filtera - LDCF.

Vizualizacija algoritma na jednostavnom primjeru

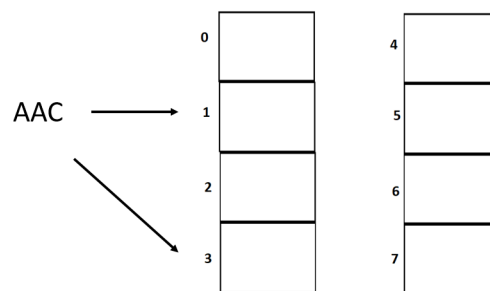
U općenitom slučaju, cuckoo filter sastoji se od nizova memorijskih lokacija koje se nazivaju *bucketima*. Bucketi mogu imati mjesta za više unosa, ali u ovom jednostavnom primjeru svaki bucket moći će primiti po jedan uneseni podatak.

Neka je dan jedan kratak nukleotidni slijed, npr. AACTGAT, te se kao ulazi u filter trebaju unijeti svi njegovi k -meri, za $k = 3$. To su : AAC, ACT, CTG, TGA i GAT . U prvom koraku algoritma za svaki unos računaju se dva sažetka (eng. *hash*) pomoću odabranih funkcija sažimanja (tzv. *hash* funkcija).



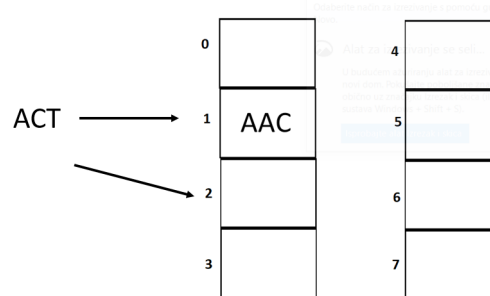
Slika 2.1: Generiranje hasha - Fran Ostroški

U drugom se koraku iz generiranih sažetaka određuje indeks bucketa u koji će biti smješten ulazni niz. To se najčešće određuje uporabom operacije dijeljenja *modulo* s ukupnim brojem bucketa. Kada se generiraju oba indeksa, element će se spremiti na jedno od dvije lokacije s tim indeksima. Svrha drugog sažetka je smanjenje vjerojatnosti kolizije i potrebe za premještanjem.



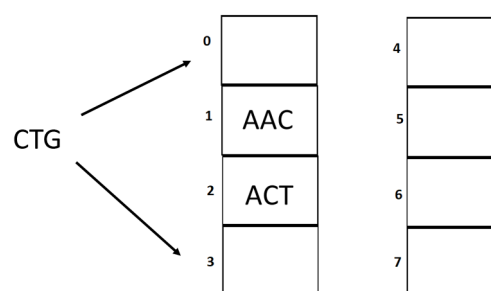
Slika 2.2: Odabir lokacije za niz AAC - prikaz po uzoru na [2]

Idući je na redu niz ACT. Nakon generiranja sažetaka smješta se u prazni bucket. Budući da je bucket na jednom od dobivenih indeksa već popunjen, sprema se na drugo, prazno mjesto.



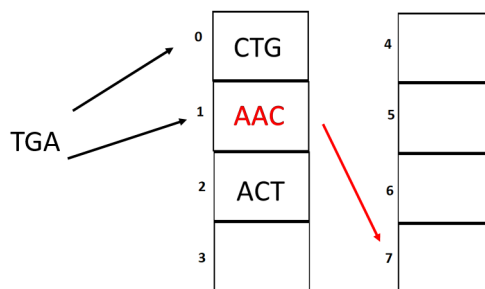
Slika 2.3: Odabir lokacije za niz ACT - prikaz po uzoru na [2]

Isti korak ponavlja se za niz CTG.



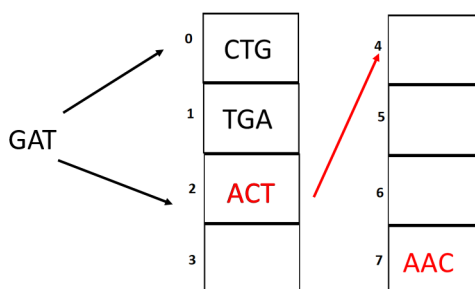
Slika 2.4: Odabir lokacije za niz CTG - prikaz po uzoru na [2]

Za niz TGA dogodila se situacija da oba pokazivača pokazuju na indekse s već popunjenim bucketima. Tada se u algoritmu događa "izbacivanje". Obabrani se element izbacuje, umeće se TGA, a za izbačeni element traži se nova lokacija. Odabir lokacije razlikuje se od algoritma do algoritma.

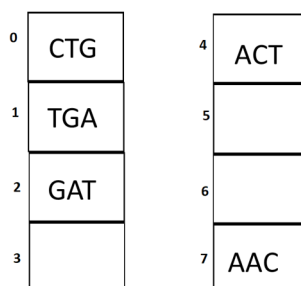


Slika 2.5: Izbacivanje AAC i ubacivanje TGA - prikaz po uzoru na [2]

Za niz GAC također se mora izvršiti izbacivanje starog elementa.



Slika 2.6: Izbacivanje ACT i ubacivanje GAC - prikaz po uzoru na [2]



Slika 2.7: Konačni razmještaj elemenata - prikaz po uzoru na [2]

Može se primijetiti da što je popunjenost veća, raste i vjerojatnost da će doći do izbacivanja i relokacije starijih elemenata u filteru. Dogodit će se situacije da će element koji se premješta biti premješten na već popunjeni bucket. U tom slučaju prijašnji element će se premjestiti, te će postupak nastaviti sve dok se ne pronađe slobodno mjesto ili se dosegne maksimalan broj operacija premještanja. U slučaju da se dosegne maksimalan broj, to je obično indikator score popunjenosti filtera. U tom slučaju mogu se poduzeti razne akcije, koje su ponovno različite za svaku implementaciju filtera.

TODO NAPOMENA: Eventualno doradit opise, dodat usporedbe tih algoritama iz dokumentacije po čemu se razlikuju, ali neznam dal smijem njihove slike (npr. tablice s vremenskim složenostima i to) jer piše u uputama kao da treba dopuštenje autora.

Dodatak opisa rada LDCF

Algoritam LDCF započinje samo jednim CF-om, koji smatramo da je nulta razina budućeg višerazinskog stabla. Pri tome se element dodaje u CF tako da se njegov ranije spomenuti otisak (eng. *fingerprint*) dodaje u bucket. Jednom kada se dogodi da operacija insert ne uspije, smatra se da je CF pun, te se dodaju dva nova CF-a. Kada se dodaje novi objekt u jedan od dva nova CF-a, promatra se prvi bit njegova fingerprinta, te prema tome određuje hoće li biti smješten u lijevi novododani CF, za vrijednost prvog bita fingerprinta jednaku nula, ili desni novododani CF, za slučaj kada je prvi bit njegova fingerprinta jednak jedan. Budući da je time vrijednost prvog bita implicitno zapisana u poziciji elementa u stablu, taj se bit uklanja iz fingerprinta i fingerprint se pohranjuje bez njega. Taj se postupak dalje propagira na niže razine.

Jedna od glavnih prednosti algoritma LDCF nad DCF jest velika mogućnost paralelizacije. Naime, u DCF-u sve operacije *insert* izvode se na istom CF-u, dok su u LDCF-u grane s CF-ima međusobno neovisne zbog opisanog načina kako se konstruiraju, te se zbog toga operacije *insert* mogu izvoditi istovremeno.

3. Analiza

U tekstu koji slijedi dana je analiza točnosti, vremena izvođenja i utroška memorije za različite testne slučajeve. Testovi su izvođeni u nekoliko inačica, koristeći:

- sintetske podatke - duljina niza od 10^3 do 10^7 znakova
- stvarne podatke - genom bakterije *Escherichia coli*

Analiza na sintetskim podacima

Ručno generirani nukleotidni sljedovi				
r. br. testa	vrijeme izvođenja operacije insert (<i>s</i>)	vrijeme izvođenja operacije query (<i>s</i>)	zauzeće memorije (<i>MB</i>)	
1	AF	AFG	004	
2	AX	ALA	248	
3	AL	ALB	008	
4	DZ	DZA	012	
5	AS	ASM	016	
6	AD	AND	020	
7	AO	AGO	024	

TODO NAPOMENA: Ako budemo koristili više tablica, treba voditi računa o tome gdje će se tablica nalaziti nakon compileiranja i onda ako je potrebno, premjestiti ju tako da ne bude zbunjujuće o kojim se podacima radi! Također, treba referencirati sve tablice, tako je vjerojatno najsigurnije.

Tablica 3.1: Rezultati testiranja na sintetskim podacima

r. br. testa	duljina nuk. slijeda (10^x)	duljina k-mera	vrijeme izvođenja operacije insert (<i>s</i>)	vrijeme izvođenja operacije query (<i>s</i>)	zauzeće memorije (<i>MB</i>)
1	3	10	tbd	tbd	tbd
2	3	20	tbd	tbd	tbd
3	3	50	tbd	tbd	tbd
4	3	100	tbd	tbd	tbd
5	3	200	tbd	tbd	tbd
6	5	10	tbd	tbd	tbd
7	5	20	tbd	tbd	tbd
8	5	50	tbd	tbd	tbd
9	5	100	tbd	tbd	tbd
10	5	200	tbd	tbd	tbd
11	6	10	tbd	tbd	tbd
12	6	20	tbd	tbd	tbd
13	6	50	tbd	tbd	tbd
14	6	100	tbd	tbd	tbd
15	6	200	tbd	tbd	tbd

Napomena: Podatci su odvojeni prema duljini nukleotidnog slijeda, a potom prema duljini k-mera. Kategorije vremena izvođenja i zauzeća memorije prikazuju prosječne vrijednosti.

Analiza na stvarnim podacima

Za analizu na stvarnim podacima korišten je genom bakterije *Escherichia coli*. Ova je bakterija modelni organizam te se njezin genom u bioinformatički koristi vrlo često i do danas je u potpunosti sekvenciran za velik dio njezinih sojeva. Konkretno, u nastavku su prikazani rezultati primjene algoritma na genom soja GCA 000798515.1, preuzetog iz baze podataka Ensembl Bacteria [TODO NAĆI REFERENCU ZA WEB STRANICU](#).

Tablica 3.2: Rezultati testiranja na sintetskim podacima

r. br. testa	duljina nuk. slijeda (10^x)	duljina k-mera	vrijeme izvođenja operacije insert (s)	vrijeme izvođenja operacije query (s)	zauzeće memorije (MB)
1	335	10	tbd	tbd	tbd
2	335	20	tbd	tbd	tbd
3	335	50	tbd	tbd	tbd
4	335	100	tbd	tbd	tbd
5	335	200	tbd	tbd	tbd

Napomena: Podatci su odvojeni prema duljini nukleotidnog slijeda, a potom prema duljini k-mera. Kategorije vremena izvođenja i zauzeća memorije prikazuju prosječne vrijednosti.

4. Zaključak

Zaključak.

TODO Komentirati kako naša implementacija radi u odnosu na predloženu, komentirati analizu.

5. Literatura

- [1] Hanhua Chen, Liangyi Liao, Hai Jin, i Jie Wu. The dynamic cuckoo filter. U *2017 IEEE 25th International Conference on Network Protocols (ICNP)*, stranice 1–10, 2017. doi: 10.1109/ICNP.2017.8117563.
- [2] Bin Fan, David G. Andersen, i Michael Kaminsky. Cuckoo filter: Better than bloom. *login Usenix Mag.*, 38(4), 2013.
- [3] Bin Fan, Dave G. Andersen, Michael Kaminsky, i Michael D. Mitzenmacher. Cuckoo filter: Practically better than bloom. U *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies, CoNEXT '14*, stranica 75–88, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450332798. doi: 10.1145/2674005.2674994.
- [4] Fan Zhang, Hanhua Chen, Hai Jin, i Pedro Reviriego. The logarithmic dynamic cuckoo filter. U *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, stranice 948–959, 2021. doi: 10.1109/ICDE51399.2021.00087.

6. Sažetak

U ovome radu dan je pregled vlastite implementacije rješenja problema opisanog u znanstvenim radovima koja je napravljena u sklopu projekta iz kolegija Bioinformatika 1 na Fakultetu elektrotehnike i računarstva Sveučilišta u Zagrebu. Dan je opis algoritma, motivacija za njegovo korištenje, kao i primjer primjene na odabrani nukleotidni slijed. Nadalje, analiziran je rad algoritma u ovoj implementaciji u odnosu na dosadašnje implementacije, njegove prednosti i nedostatci.