SVEUČILIŠTE U ZAGREBU
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

Seminar

# PRIMJENA BAMBOO FILTERA PRI TRAŽENJU SLUČAJNIH PODNIZOVA U E. COLI GENOMU

Filip Belina,
Jakov Lovaković

Voditeljica:
izv. prof. dr. sc. Mirjana Domazet-Lošo

Zagreb, lipanj, 2025

# Sadržaj

# 1. Uvod

Podatkovna struktura upita o približnom članstvu (*engl.* approximate membership query (AMQ)) ima efikasnu i kompaktnu organizaciju podataka te može odrediti postoji li element u setu približno. Popularni primjeri ovakve strukture koji se koriste za reprezentaciju velikih podataka su Cuckoo Filteri [1] (*engl.* Cuckoo Filters), Bloom Filteri [2] (*engl.* Bloom Filters) i sl. Broj elemenata se često mijenja u AMQ podatkovnim strukturama radi čestih umetanja i brisanja podataka. Zbog navedenog AMQ podatkovne strukture trebaju mijenjati svoju veličinu tokom rada. Bamboo Filter uvodi linearno raspršivanje parcijalnog ključa (*engl.* partial-key linear hashing), tj. segmenti primjenjuju promjenu veličine u logičnom poretku umjesto u poretku preljeva. Time serijski broj idućeg segmenta za promjenu veličine dijeli tablicu raspršivanja na dva dijela, dio čiji su segmenti proširili, i dio čiji će se segmenti tek proširiti. [3]

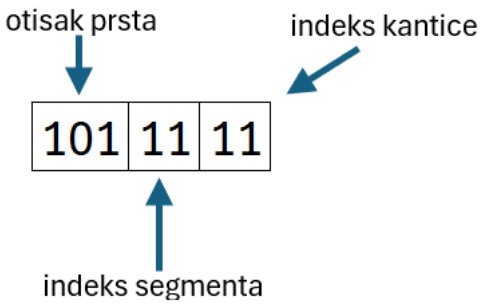
Jedan od problema na koji se može primijeniti Bamboo Filter je traženje slučajnih pod nizova u genomu bakterije *E. coli*, a sam genom bakterije *E. coli* možemo smatrati velikom podatkovnom strukturom. Bamboo Filter ima $O(1)$ vremensku kompleksnost za operacija umetanja, pretraživanja i brisanja elemenata. Ta vremenska kompleksnost omogućuje iznimno brzo pretraživanje pod nizova unutar samog genoma bakterije *E. coli* te zbog navedenih razloga koristimo Bamboo Filter u ovome radu kao alat za pretraživanje pod nizova. Za potrebe ovoga rada implementirane su funkcije umetanja, pretraživanja i ekspanzije. Funkcija brisanja za ovaj problem u ovome radu nije implementirana.

# 2. Bamboo Filter za pretraživanje pod nizova u genomu bakterije *E. coli*

U ovom poglavlju dan je opis algoritma u pod poglavlju 1.1. U pod poglavlju 1.2. dan je opis korištenja Bamboo Filtera za problem pretraživanja pod nizova unutar genoma bakterije *E. coli*.

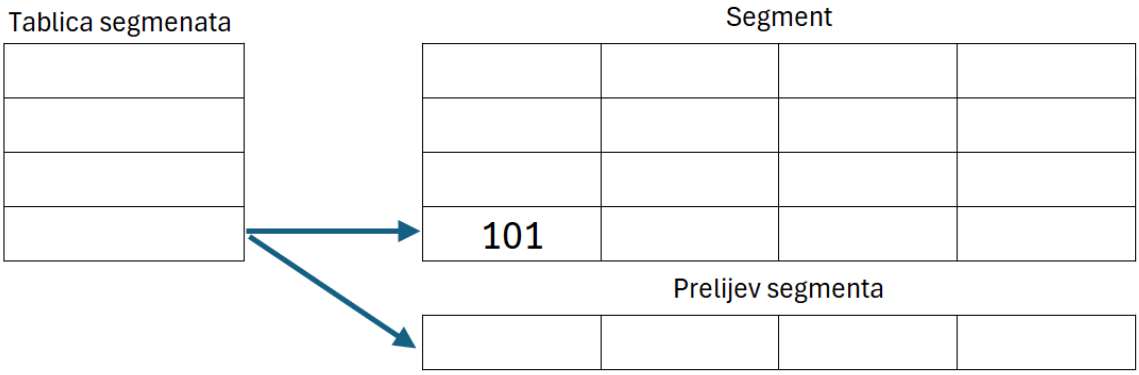
## 2.1. Opis rada Bamboo Filtera

Kao što je navedeno u prethodnom poglavlju, Bamboo filter je AMQ podatkovna struktura. Sastoji se od tablice čiji su gradivni elementi segmenti. Svaki segment u sebi ima fiksiran broj kantica (*engl*. bucket). Svaka kantica u sebi ima fiksiran broj elemenata (*engl*. entry). Element u kantici jest zapravo otisak prsta (*engl*. fingerprint) originalnog elementa koji se umeće u filter. Otisak prsta se dobiva primjenom jednog od algoritama raspršivanja (*engl*. hashing algorithm). Na temelju vrijednosti raspršivanja (*engl*. hash) određuje se otisak prsta, indeks segmenta i indeks kantice. Pri implementaciji samog filtera odredi se s koliko se bitova mogu indeksirati segmenti i kantice te kolika će biti veličina otiska prsta. Na temelju određenog, iz vrijednosti raspršivanja za element se izdvajaju tražene vrijednosti. Pri tome se bitovi indeksa segmenta i otiska prsta mogu preklapati. Na temelju indeksa segmenta se određuje segment u koji se postavlja element, a na temelju indeksa kantice određuje se kantica u koju se postavlja element. Unutar same kantice element se postavlja na prvo slobodno mjesto. Pretraživanje kantica nije vremenski skupa operacija jer same kantice ne sadržavaju puno elemenata.




Slika 1: Vrijednost raspršivanja elementa.

Na slici 1 može se vidjeti primjer dobivanja otiska prsta te indeksa segmenta i kantice iz vrijednosti raspršivanja. Gledajući s desna na lijevo, prvih *lb* bitova predstavljaju indeks kantice, zatim idućih *ls* bitova predstavljaju indeks segmenta. Gledajući s desna na lijevo, prvih *lf* bitova predstavljaju otisak prsta. U početku rada, filter se sastoji od 2^*ls* segmenata, a svaki segment se sastoji od 2^*lb* kantica. Pošto je Bamboo filter AMQ podatkovna struktura, broj tih segmenata će se povećavati tokom njegovog rada. Preciznije, pri implementaciji se definira kriterij proširivanja, koji se provjerava nakon svakog inserta. Ukoliko je kriterij zadovoljen, proširuje se, tj. ekspandira se, sljedeći po redu segment označen za ekspanziju. Ekspanzija se radi tako da se kreira novi segment te se gledaju svi elementi koji se trenutno nalaze u segmentu koji se ekspandirao. Gledajući prvi sljedeći bit otiska prsta, određuje se hoće li otisak prsta ostati u trenutnom segmentu ili će se prebaciti u segment dobiven ekspanzijom.




Slika 2: Postavljanje otiska prsta sa slike 1 u pripadajući segement i kanticu.

Na slici 2 se vidi struktura samog filtra. Gledajući vrijednost raspršivanja iz slike 1 vidi se da je indeks pripadajućeg segmenta 11 (dekadski 3), što znači da se otisak prsta postavlja u segment s indeksom 3. Na isti način se dolazi do indeksa 3 i za pripadajuću kanticu. Element se postavlja na prvo slobodno mjesto u kantici s indeksom 3., u segmentu s indeksom 3. Ako se želi pretražiti postoji li element s vrijednosti raspršivanja sa slike 1 u filteru, izračuna se otisak prsta, indeks segmenta, indeks kantice i alternativni indeks kantice te se pomoću tih veličina pretražuje tablica filtra. Na slici 2 se vidi da element s vrijednošću raspršivanja prikazanoj na slici 1 postoji u filteru, s obzirom da se u segmentu s indeksom 3 i kantici s indeksom 3 nalazi otisak prsta 101. U slučaju da trebamo umetnuti otisak prsta u kanticu koja je puna, uzimamo nasumičan otisak prsta iz te kantice. Za uzeti otisak prsta računamo indeks alternativne kantice. Umjesto uzetog otiska prsta umećemo otisak prsta koji orginalno

nismo mogli umetnuti radi popunjene kantice. Za uzeti otisak prsta provjeravamo ima li kantica s alternativnim indeksom za taj otisak prsta slobodno mjesto, a ako ima umećemo otisak prsta u tu kanticu. Ako ta kantica nema slobodnog mjesta ponavljamo postupak fiksan broj puta za idući nasumičan otisak prsta. U slučaju da i dalje nismo našli alternativnu kanticu sa slobodnim mjestom, za zadani otisak prsta rekonstruiramo vrijednost raspršivanja i postavljamo ga u dio prelijeva (*engl*. overflow). Svaki segment ima svoj dio prelijeva, kao što je vidljivo na slici 2. U teoriji, taj dio može biti beskonaćne veličine, ali se to u praksi neće dogoditi. Prilikom svake ekspanzije se provjerava za segment koji se ekspandira možemo li njegove elemente unutar dijela prelijeva preraspodijeliti u njega samoga ili u segment dobiven ekspanzijom (ovisno o prvom sljedećem indeksu otiska prsta), s obzirom da će se dio mjesta u kanticama osloboditi jer će se dio elemanta prebaciti iz segmenta koji se ekspandira u segment dobiven ekspanzijom. U slučaju da se i dalje elementi iz dijela prelijeva nemogu preraspodijeliti unutar samog segmenta koji se ekspandira ili segementa dobivenog ekspanzijom, oni ostaju u dijelu prelijeva trenutnog segmenta ili se prebacuju u dio prelijeva segmenta dobivenog ekspanzijom, ovisno o prvom sljedećem bitu fingertipa.

## 2.2. Funkcije umetanja, pretraživanja i ekspanzije

U ovom poglavlju dat je opis funkcija umetanja, pretraživanja i ekspanzije. Funkcije umetanja ima vremensku složenost $O(1)$, dok funkcije pretraživanja i ekspanzije imaju vremensku složenost $O(k)$, gdje je $k$ broj elemenata u dijelu prelijeva segmenta u kojem se nalazi element. U praksi k nije velik, stoga je vremenska cijena funkcija pretraživanja i ekspanzije vrlo mala. U nastavku će se opisivat pseudokodovi navedenih funkcija. Svi pseudokodovi su preuzeti iz [3], ali sama implementacija ima detalja koji će biti opisani ispod samih pseudokodova.

### 2.2.1. Funkcija umetanja

Na slici 3 dan je pseudokod funkcije umetanja. Kao što je prethodno navedeno, njena vremenska složenost je $O(1)$.

**Algorithm 1:** Insert($e$)

```
   Input: The element to insert e
1  Calculate f, I_b, I_b', I_s as per Equation (1) - (4);
2  if bucket I_b or I_b' in segment I_s has an empty entry then
3  |   Store f in an empty entry;
4  |   return Success;
5  end
6  Randomly select a bucket i_b from I_b and I_b';
7  for loop = 0; loop < M_e; loop++ do
8  |   Randomly select an entry e from bucket i_b;
9  |   Swap f and the fingerprint stored in entry e;
10 |   Calculate another candidate bucket i_b' of f by Equation
   |   (3);
11 |   if bucket i_b' has an empty entry then
12 |   |   Store f in an empty entry;
13 |   |   return Success;
14 |   end
15 |   i_b = i_b';
16 end
   // Insert f into segment I_s's overflow
      segment
17 Segment[I_s].overflow.Insert(e);
18 if expand condition is triggered then
19 |   call Expand();
20 end
21 return Success;
```

Slika 3: Psuedokod funkcije umetanja, preuzeto iz [3]

Funkcija umetanja u implementaciji ovog rada kao ulaz prima znakovni niz. Na temelju znakovnog niza izračunaju se otisak prsta (1), indeks kantice (2), alternativni indeks kantice (3) te indeks segmenta (4). U slučaju da segment s izračunatim indeksom segmenta ne postoji, vrši se korekcija indeksa segmenta (5).

$$f(e) = (hash(e) \gg (l_b + l_{s0}) \,\%\, 2^{l_f} \quad (1)$$

$$I_b(e) = hash(e) \,\%\, 2^{l_b} \quad (2)$$

$$I_b'(e) = \left(I_b \oplus \mathrm{f(e)}\right) \% \, 2^{l_b} \quad (3)$$

$$I_s(e) = (hash(e) \gg I_b) \,\%\, \left(2^{i+1}N_0\right) \quad (4)$$

$$I_s(e) = (hash(e) \gg l_b) \,\%\, \left(2^{i+1}N_0\right) - 2^i N_0 \quad (5)$$

Nakon izračuna potrebnih veličina, provjerava se ima li kantica s indeksom $I_b$ unutar segmenta s indeksom $I_s$ slobodnog mjesta. Ukoliko ima umeće se otisak prsta na prvo slobodno mjesto u kantici. Za razliku od pseudokoda sa slike 3, u našoj implementaciji se prije linije 4 provjeri treba li pozvati funkciju ekspanzije ukoliko je uvijet ekspanzije zadovoljen. U slučaju da kantica s indeksom $I_b$ unutar segemnta $I_s$ nema praznog mjesta, provjerava se kantica s alternativnim indeksom kantice $I_b'$ unutar istog segmenta. Ukoliko ta kantica ima praznog mjesta, umeće se otisak prsta na prvo slobodno mjesto u kantici te se kao i u prethodnom slučaju provjerava je li zadovoljen uvjet ekspanzije te ako je, zove se ekspanzija. Ukoliko u niti jednoj od dvije kantice nismo pronašli prazno mjesto, vršimo operaciju po kojoj Bamboo filter podsjeća na Cuckoo filter. Prvo se odabere na nasumičan način indeks kantice. Pri tome se bira jedan od dva indeks; $I_b$ ili $I_b'$. Nakon toga se izvršava petlja koja je po broju iteracija ograničena s brojem *Me*. Petlja se mora ograničiti kako bi se zadržala *O*(1) složenost algoritma. Iteracija petlje funkcionira ovako: odabere se nasumičan element iz kantice s indeksom *ib* i indeksom segmenta $I_s$. Taj se element zamijeni s trenutnim otiskom prsta koji nismo uspjeli smjestiti u kantice s indeksima $I_b$ i $I_b'$. Za otisak prsta koji je do tada bio na tom mjestu u toj kantici sada treba dobiti novo mjesto jer si ne možemo priuštiti gubitak elemenata. Stoga za taj element izračunamo alternativni indeks kantice po formuli (3). Ukoliko ta kantica ima slobodnog mjesta, element koji smo prethodno izbacili sa svojeg orginalnog mjesta stavljamo u kanticu s alternativnim indeksom. Nakon uspješnog ubacivanja provjeravamo uvjet ekspanzije i ako je zadovoljen zovemo ekspanziju, slično kao u prethodnim slučajevima, te izlazimo iz funkcije. Ako ni u alternativnoj kantici nema mjesta, *ib* se postavlja na vrijednost alternativnog indeksa kantice izbačenog elementa te započinjemo novu iteraciju petlje. U slučaju da petlja odradi svih *Me* iteracije i da se za neki izbačeni otisak prsta ne uspije pronaći mjesto, taj se element ubacuje u dio prelijeva (*engl.* overflow). Prije nego što se izbačeni otisak prsta ubaci u indeks prelijeva, rekonstruira mu se vrijednost raspršivanja, što nije prikazano u pseudokodu radi implementacijskih detalja. Na kraju funkcije insert se još jednom provjeri uvjet ekspanzije (za slučaj kada je rekonstruirana vrijednost raspršivanja otiska prsta dodana u dio prelijeva) i ako je zadovoljen, vrši ekspanzija.

### 2.2.2. Funkcija ekspanzije

Na slici 4 dan je pseudokod funkcije ekspanzije. Kao što je prethodno navedeno, njena vremenska složenost je *O*(*k*), gdje je *k* broj elemenata u dijelu prelijeva. U praksi *k* nije velik, stoga je vremenska cijena funkcije ekspanzije vrlo mala.

**Algorithm 2:** Expand($F$)

```
   Input: The filter to be expanded F
1  p = index of next segment to be split;
2  Add a segment p' to the F's hash table;
3  for each fingerprint f in segment p do
4  |  if f[i] == 1 then
   |  |  // The i-th bit of the fingerprint
   |  |     is 1
5  |  |  Insert the fingerprint f into segment p';
6  |  end
7  end
8  p = p + 1;
9  if p == 2^i N_0 then
   |  // The (i+1)-th round of expansion
   |     finished
10 |  p = 0;
11 end
12 return Success;
```

Slika 4: Pseudokod funkcije ekspand, preuzeto iz [3]

Na temelju sljedećeg segmenta za ekspanziju s indeksom *p* kreira se novi segment, kao što je vidljivo i u linijama 1 i 2 u pseudokodu sa slike 4. Zatim se iterira po svim otiscima prsta iz segementa s indeksom *p*. Za svaki otisak prsta se gleda *i*-ti bit te ako je taj bit jednak 1, otisak prsta se seli u novi segment, inače ostaje u trenutnom. U slučaju selidbe, u segmentu s indeksom *p* se oslobađa mjesto na kojem je bio smješten otisak prsta te se smješta u kanticu s indeksom $I_b$ u novom segementu s indeksom *p'*. Ukoliko to mjesto u novom segmentu nije slobodno, smješta se u kanticu s alternativnim indeksom $I'_b$ ili u dio prelijeva novog segmenta. Ukoliko i to mjesto u novom segmentu nije slobodno, smješta se u dio prelijeva novog segmenta. Prije pomaka na liniju 8, u ovoj implementaciji Bamboo filtera se također provjeravaju i svi elementi u dijelu prelijeva segmenta *p*. Pošto su unutra smještene

rekonstruirane vrijednosti raspršivanja elemenata, na temelju njih za svaki element računamo veličine potrebne za određivanje pozicije otiska prsta, kao i sam otisak prsta. Ovisno o indeksu segmenta, za svaki element prelijeva se proverava stane li u svoju običnu ili alternativnu kanticu. Ukoliko ne stane u niti jednu od te dvije, ostaje u dijelu prelijeva segmenta *p* ili *p'*, ovisno o $I_s$. Na kraju se povećava *p* kako bi pokazivao na idući segment za ekspanziju te se provjerava je li uvjet kraja runde ekspanzije zadovoljen, a u slučaju da je, p se postavlja na 0 te će se s idućim pozivom funkcije ekspanzije započeti nova runda ekspanzije.

### 2.2.3. Funkcija pretraživanja

Na slici 5 dan je pseudokod funkcije pretraživanja. Kao što je prethodno navedeno, njena vremenska složenost je *O*(*k*), gdje je *k* broj elemenata u dijelu prelijeva. U praksi *k* nije velik, stoga je vremenska cijena funkcije pretraživanja vrlo mala.

**Algorithm 3:** Lookup($e$)

**Input:** The element to lookup $e$

```
1 Calculate f, I_b, I_b', I_s as per Equation (1) - (4);
2 if bucket I_b or I_b' in segment I_s contains f then
3 |   return True;
4 end
5 if the overflow chain of segment I_s is not empty then
6 |   return segment[I_s].overflow.Lookup(e);
7 end
8 return False;
```

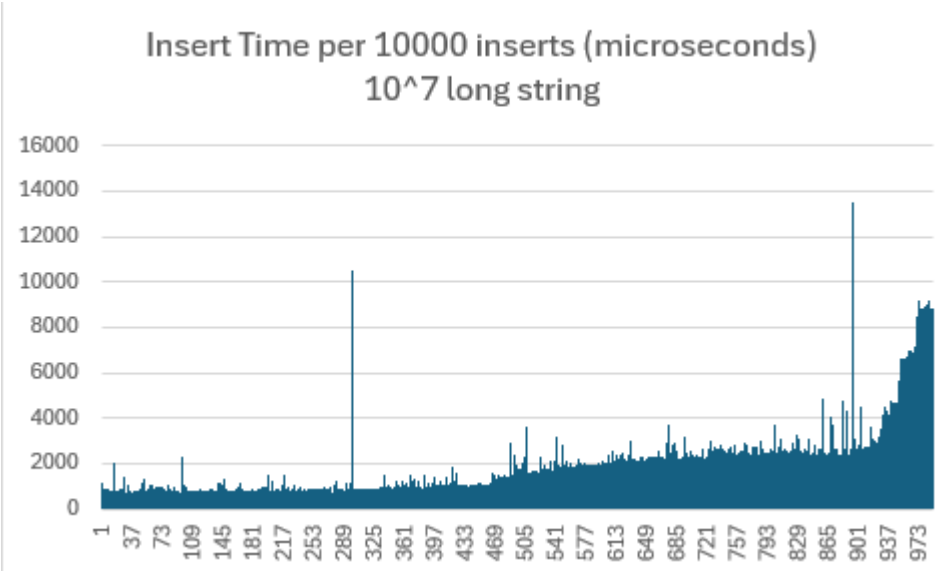
Slika 5: Pseudokod funkcije pretraživanja, preuzeto iz [3]

Funkcija pretraživanja kao ulaz prima znakovni niz. Prvo izračuna sve potrebne veličine s formulama (1) – (5). Zatim provjeri sadrže li kantice $I_b$ ili $I_b'$ u segmentu $I_s$ otisak *f*. U slučaju da sadrže, funcija vraća istinu te se element smatra pronađenim. U slučaju da kantice ne sadrže otisak *f*, rekonstruira se vrijednost raspršivanja na dva načina; prvo s indeksom $I_b$, zatim s indeksom $I_b'$ (ovo je implementacijski detalj te nije naveden u pseudokodu). Nakon toga se pretražuje dio prelijeva segmenta $I_s$, tj. gleda se sadrži li dio prelijeva jednu od dvije

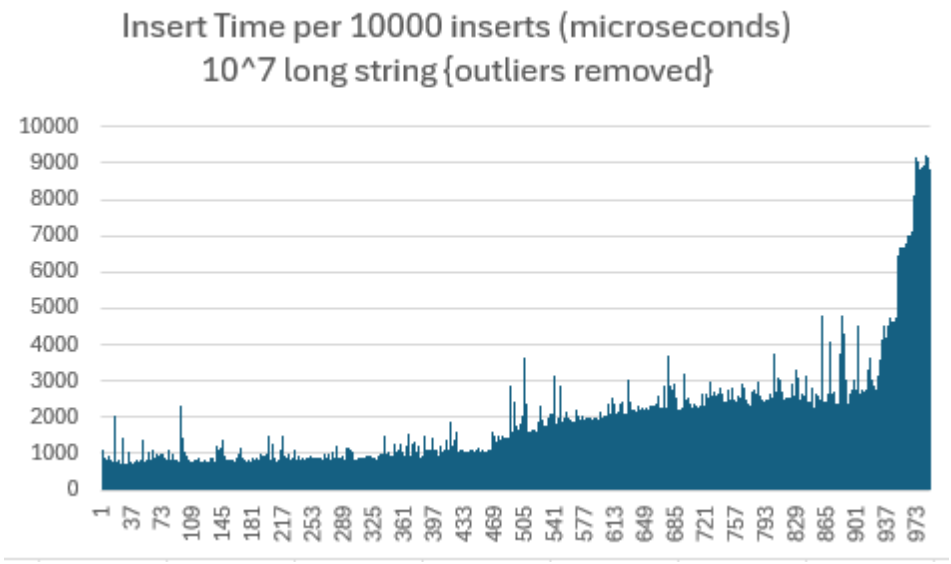
vrijednosti raspršivanja. U slučaju da sadrži, vraća se istina i element se smatra pronađenim. Inače se vraća laž i smatra se da element ne postoji u filteru.

## 2.3. Rezultati

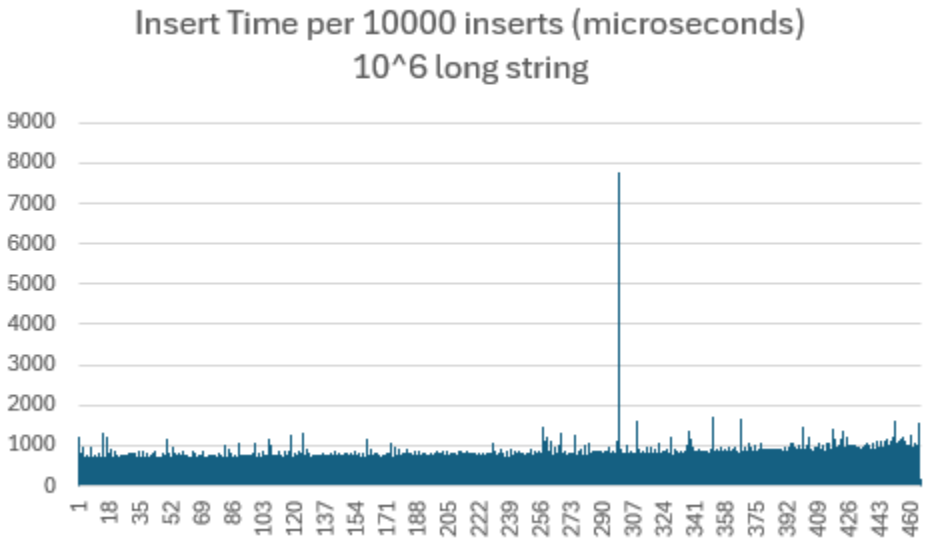
Ovo poglavlje proći će kroz podatke i grafove vezano uz ovaj rad. Statistika insert i lookup funkcija rađenaj je na sljedeći način, vrijeme je opisano kao trajanje u mikrosekundama za obaviti insert ili lookup 10000 stringova, ovaj broj odabran je eksperimentalno jer za vrijednosti exponenata od 10 to je prvi koji ima ugodno malu količinu zapisa kao 0. Napravljena su 2 grafa za insert I lookup jedan za string duljine 10^7 znakova drugi za cijeli string koji pretstavlja bakteriju ecoli. Grafovi insert imaju neocekivane siljke u podatcima koji su maknuti u alternativnoj verziji grafa radi preglednosti. Na oba grafa vidljivo je kako vrijeme inserta postaje dulje kako se inserta vise podataka jer se cesce moraju odradivati operacije izbacivanja elemenata iz bucketa I trazenje njihovih alternativnih bucketa kako bi se mogli ubaciti novi podatci u filter, takoder u kasnijim iteracijama inserta cesce se mora expendati sto povecava trajanje operacije insert. Za operaciju lookup vidljivo je kako se povecanjem broja lookup poziva ne mijenja vrijeme potrebno za pretrazivanje zapisa u stringu sto je I ocekivano ponasanje posto je lookup ima O(1) veliko O notaciju. Uz podatke prilozene u grafovima uz ovaj rad bit ice I prilozena zip arhiva sa svim csv vrijednostima za sve pokrete programa. Program se pokretao mijenjajuci duljinu k-mera te mijenjajuci duljinu stringa koja se stavlja u filter. Imena datoteka imaju suffix oblika _duljinaKmera_duljinaStringa.



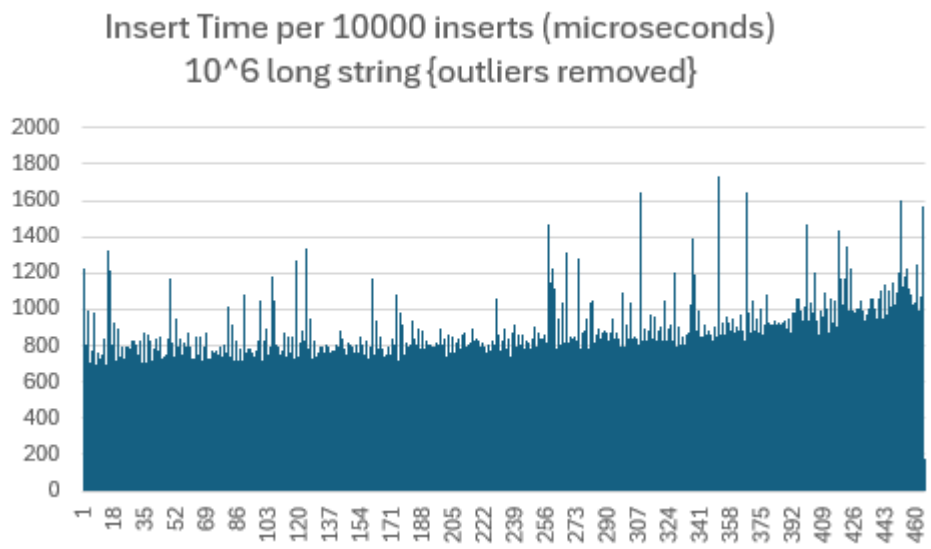

Slika 6: Vrijeme umetanja po 10000 umetanja (mikrosekunde), znakovni niz veličine 10^7



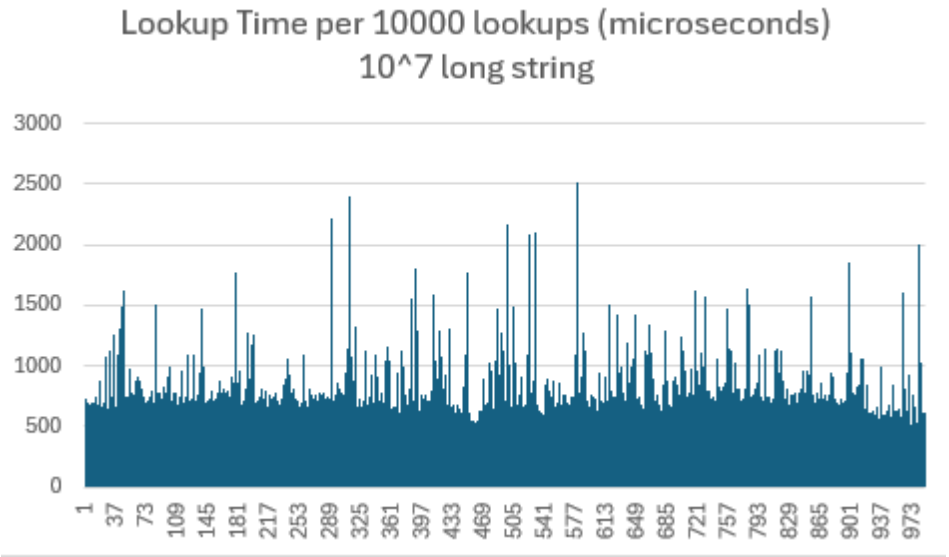

Slika 7: Vrijeme umetanja po 10000 umetanja (mikrosekunde), znakovni niz veličine 10^7 sa oduzetim odstupajućim vrijednostima



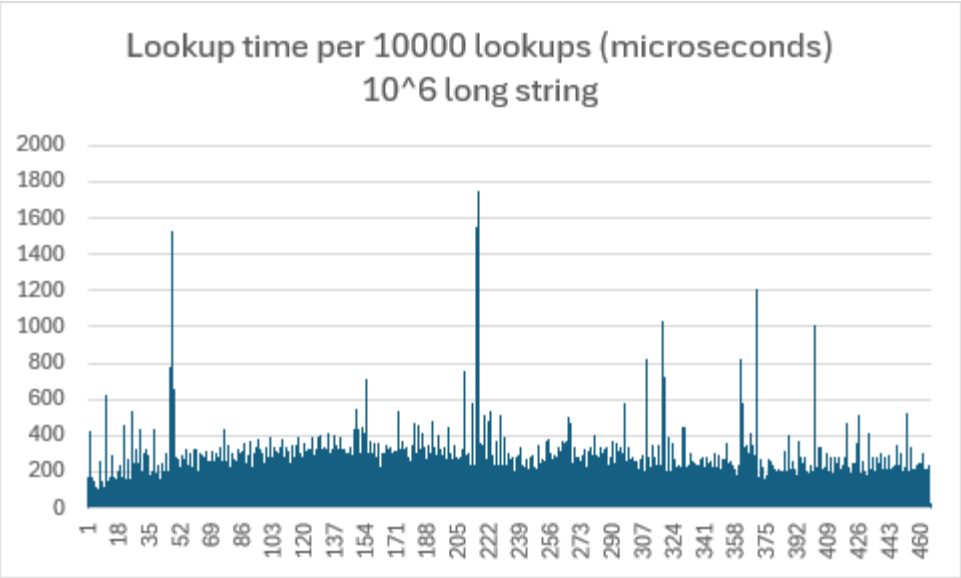

Slika 8: Vrijeme umetanja po 10000 umetanja (mikrosekunde), znakovni niz veličine 10^6




Slika 9: Vrijeme umetanja po 10000 umetanja (mikrosekunde), znakovni niz veličine 10^6 sa oduzetim odstupajućim vrijednostima




Slika 10: Vrijeme pretraživanja po 10000 pretraživanja (mikrosekunde), znakovni niz veličine 10^7




Slika 11: Vrijeme pretraživanje po 10000 pretraživanja (mikrosekunde), znakovni niz veliine 10^6

Za analizu prostorne memorije naše implementacije Bamboo filtra uzet ćemo u obzir k-mere duljine 50 s različitim duljinama stringa.

| Približna duljina stringa | 10^3 | 10^4 | 10^5 | 10^6 | 10^7 | 4.6*10^6 |
|---|---|---|---|---|---|---|
| Zauzeta količina RAM memorije pri izvođenju | 2.66 MB | 2.66 MB | 2.66 MB | 139.68 MB | 309.04 MB | 185.52 MB |

# 3. Zaključak

Bamboo Filter je brza i efikasna struktura podataka koja se može koristiti za pretraživanje znakovnih podnizova u znakovnom nizu genoma bakterije *E. coli*. Funkcija umetanja ima vremensku složenost $O(1)$, dok funkcije ekspanzije i pretraživanja imaju vremensku složenost $O(k)$, gdje je $k$ broj elemenata u dijelu prelijeva segmenta. Pošto je k obično vrlo malen, sve 3 funkcije su vremenski efikasne, tj. brze. Sami broj mogućih znakovnih podnizova u genomu bakterije *E. coli* dostiže red veličine 10^6, stoga je važno da se za potrebe pretraživanja koriste vremenski i prostorno efikasne podatkovne strukture, kao što je Bamboo filter. Rezultati testiranja ove implementacije Bamboo filtera ukazuju upravu na vremensku i prostornu efikasnost koja je bila i očekivana prije početka rada.

# 4. Literatura


[1] B. Fan, D. G. Andersen, M. Kaminsky, and M. Mitzenmacher, “Cuckoo f ilter: Practically better than bloom,” in Proceedings of ACM International Conference on Emerging Networking Experiments and Technologies. ACM, 2014, pp. 75–88

[2] B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors,” Communications of the ACM, vol. 13, no. 7, pp. 422–426, 1970.

[3] Wang H et al. Bamboo filters: make resizing smooth and adaptive *IEEE/ACM Trans. Netw*. 2024

# 5. Sažetak

Ovaj rad opisuje primjenu Bamboo filtera za pretraživanje podnizova znakova u nizu znakova genoma bakterije E. coli. Obrađeni su algoritmi umetanja, pretraživanja te ekspanzije uz osvrt na vremensku i prostornu analizu izvedenog rješenja.