

# Sažimanje genoma korištenjem referentnog genoma

---

**Palić, Magdalena**

**Undergraduate thesis / Završni rad**

**2024**

*Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj:* **University of Zagreb, Faculty of Electrical Engineering and Computing / Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva**

*Permanent link / Trajna poveznica:* <https://um.nsk.hr/um:nbn:hr:168:149953>

*Rights / Prava:* [In copyright](#)/[Zaštićeno autorskim pravom.](#)

*Download date / Datum preuzimanja:* **2025-03-21**



*Repository / Repozitorij:*

[FER Repository - University of Zagreb Faculty of Electrical Engineering and Computing repository](#)



SVEUČILIŠTE U ZAGREBU  
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 445

**SAŽIMANJE GENOMA KORIŠTENJEM REFERENTNOG  
GENOMA**

Magdalena Primorac

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU  
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 445

**SAŽIMANJE GENOMA KORIŠTENJEM REFERENTNOG  
GENOMA**

Magdalena Primorac

Zagreb, lipanj 2022.

## ZAVRŠNI ZADATAK br. 445

Pristupnica: **Magdalena Primorac (0036524547)**  
Studij: Elektrotehnika i informacijska tehnologija i Računarstvo  
Modul: Računarstvo  
Mentorica: izv. prof. dr. sc. Mirjana Domazet-Lošo

Zadatak: **Sažimanje genoma korištenjem referentnog genoma**

### Opis zadatka:

U okviru ovoga završnog rada potrebno je proučiti problem sažimanja skupa genoma korištenjem referentnog genoma. Potrebno je proučiti i implementirati algoritam SCCG koji koristi referentni genom za sažimanje genoma. Implementaciju je potrebno napraviti u programskom jeziku C++ te ju usporediti s originalnom implementacijom u postupcima komprimiranja i dekomprimiranja skupa genoma.

Rok za predaju rada: 10. lipnja 2022.

*Zahvaljujem se mentorici doc. dr. sc. Mirjani Domazet-Lošo na vodstvu i pomoći te obitelji i prijateljima na podršci tijekom pisanja ovoga rada.*

# SADRŽAJ

|  |    |
|--|----|
| 1. <a href="#">Uvod</a> .....  | 2  |
| 2. <a href="#">DNA</a> .....   | 5  |
| 2.1. <a href="#">Struktura</a> .....   | 5  |
| 2.2. <a href="#">Mutacije</a> .....  | 6  |
| 2.3. <a href="#">Fasta</a> .....   | 7  |
| 3. <a href="#">Kodiranje</a> .....   | 8  |
| 3.1. <a href="#">Aritmetičko kodiranje</a> .....                                     | 8  |
| 3.2. <a href="#">Predviđanje djelomičnim podudaranjem</a> .....                      | 12 |
| 3.3. <a href="#">Kompresija genoma</a> .....   | 15 |
| 4. <a href="#">Tablica raspršenog adresiranja</a> .....                              | 17 |
| 5. <a href="#">SCCG</a> .....  | 18 |
| 5.1. <a href="#">Ulaz i preprocesiranje</a> .....                                    | 18 |
| 5.2. <a href="#">Lokalno podudaranje</a> .....                                       | 19 |
| 5.3. <a href="#">Globalno podudaranje</a> .....                                      | 20 |
| 5.4. <a href="#">Postprocesiranje</a> .....  | 21 |
| 5.5. <a href="#">Dekompresija</a> .....  | 23 |
| 5.6. <a href="#">Primjer kompresije i dekompresije</a> .....                         | 23 |
| 6. <a href="#">Implementacija u C++-u</a> .....                                      | 28 |
| 6.1. <a href="#">Lokalno i podudaranje</a> .....                                     | 30 |
| 6.2. <a href="#">Razlučivanje između faze lokalnog i globalnog podudaranja</a> ..... | 33 |
| 6.3. <a href="#">Dekompresija</a> .....  | 35 |
| 7. <a href="#">Rezultati</a> .....   | 35 |
| 7.1. <a href="#">Usporedba SCCGC implementacije i originala</a> .....                | 35 |
| 7.2. <a href="#">Izvedba SCCGD implementacije</a> .....                              | 40 |
| 8. <a href="#">Zaključak</a> .....   | 41 |
| 9. <a href="#">Literatura</a> .....  | 42 |
| 10. <a href="#">Sažetak</a> .....  | 43 |

# 1.Uvod

Deoksiribonukleinska kiselina, poznatija kao DNA, složena je molekula koja sadrži sve informacije potrebne za izgradnju i održavanje organizma. Ona također služi kao primarna jedinica nasljeđivanja kod organizmima svih vrsta.

Određivanje redoslijeda nukleotida u biološkim uzorcima, sastavni je dio širokoga spektra znanstvenih istraživanja. DNA sadrži vrlo korisne informacije koje se mogu upotrijebiti za predviđanje bolesti kod pacijenata i pravovremeno reagiranje na njih. Zbog toga je tijekom posljednjih pedeset godina veliki broj znanstvenika razrađivao nove tehnike sekvenciranja odnosno određivanja sljedova nukleinskih baza molekula DNA i RNA.

1953. godine zabilježeno je prvo sekvenciranje biološke molekule. Frederick Sanger je po prvi put uspio sekvencirati dva lanca inzulinskog proteina. To je otvorilo put sekvenciranju DNA [\[1\]](#). Sekvenciranje DNA započelo je Sangerovim radom „*DNA sequencing with chain-terminating inhibitors*“ 1977. godine kada je i prvi put sekvenciran potpuni genom nekog organizma, u ovom slučaju to je bio genom bakteriofaga. Iste godine su Allan Maxam i Walter Gilbert objavili rad „*DNA sequencing by chemical degradation*“ [\[2\]](#). Sangerova metoda je ipak bila prihvaćenija radi veće efikasnosti i korištenja manje toksičnih kemikalija u procesu sekvenciranja.

Napredak tehnologije u 80-tim i 90-tim godinama prošlog stoljeća potakao je pokretanje međunarodnog Projekta ljudskog genoma koji je za cilj imao utvrđivanje sekvence cjelovitog ljudskog genoma [\[1\]](#). Procijenjen trošak projekta je trebao biti ukupno 3 milijarde dolara, no projekt je na kraju koštao oko 2,7 milijardi dolara [\[3\]](#).

Jedno od najvećih otkrića bioloških znanosti bila je objava sekvence ljudskog genoma. Objavile su je dvije nezavisne skupine na čelu s Craigom Venterom iz Celera Genomics 2001. godine [\[2\]](#). Na početku 2000-tih godina sekvenciranje jednog ljudskog genoma koštalo je čak preko 100 000 000\$ [\[4\]](#). Današnja cijena sekvenciranja ljudskog genoma je manja od 1000\$ te je sekvenciranje postalo vrlo

pristupačno [4]. Time se povećala i potrošnja memorije s obzirom na to da je veličina datoteke koju generira jedno sekvenciranje ljudskog genoma oko 100 GB [5]. Zabilježena je iznimno velika potražnja za sekvenciranjem DNA te se očekuje da će do 2028. godine, globalno tržište sekvenciranja DNA dosegnuti cijenu od čak 12,55 milijardi \$ i zabilježiti prosječnu godišnju stopu rasta od 11.3% [6]. Potrebno je još više memorije zbog dodatnih podataka koji se prikupljaju iz raznih tkiva kako bi se vršile usporedbe zdravih i bolesnih stanja pacijenata te kako bi se prikupljali i ostali slični podatci [7].

Cijena spremanja neobrađenih podataka sekvenciranja je prerasla cijenu samog sekvenciranja DNA. Trenutna je uobičajena praksa brisanje neobrađenih slikovnih datoteka nakon što su iste obrađene, kako bi se proizveo relativno mali tekstualni slijed i datoteke s kvalitetnim podacima. Iako je dugotrajno pohranjivanje datoteka tekstualnog niza izvedivo korištenjem tehnologije traka i diskova, takav način pohrane predstavlja izazov za održavanje podataka jer nije u obliku koji je pristupačan korisnicima za lako ispitivanje [8].

Slijed ljudskog genoma proglašen je dovršenim 2004. godine, međutim zbog prisutnosti osebujnih sekvenci ponavljajuće DNA, koje je teško mapirati i sekvencirati, slijed je prekinut s 341 prazninom za koje se procjenjuje da su veličine 225 Mbp gdje je 1 Mb(megabaza) = 1 000 000 parova baza. Gotova sekvenca sadrži 2.85 Gbp (1 Gbp = 1 000 000 000 parova baza). To zajedno daje procijenjenu veličinu ljudskog genoma oko 3.1 Gbp odnosno ljudski genom sadrži oko 3 milijarde parova baza [9]. Pošto parovi baza dolaze u obliku jednog od 4 slova (A, C, G T), svaka baza se može kodirati pomoću 2 bita što bi rezultiralo pohranom cijelog ljudskog genoma u datoteku veličine 750 MB [10]. Međutim, za svaki ljudski genom, proces sekvenciranja generira još dodatne podatke koji zauzimaju oko 100 GB prostora, a posljedica su višestrukih očitavanja i pozivanja baza te poravnavanja [5].

Treća generacija sekvenciranja i mapiranja DNA donijela je procvat u visokokvalitetnom sekvenciranju. Druga generacija sekvenciranja proizvodi kratka čitanja od nekoliko stotina parova baza, a za razliku od nje, tehnologije treće generacije s jednom molekulom generiraju čitanje preko 10 000 parova baza [11]. U treću



generaciju sekvenciranja spadaju tehnologije poput „Helicos single molecule fluorescent sequencing“, „Pacific Biosciences“ i „Oxford Nanopore“.

Zbog svega navedenog, javila se potreba za sažimanjem genoma poradi smanjenja prostora potrebnog za pohranu istoga. To bi nam omogućilo brže pretraživanje genoma, a u isti mah i manju cijenu pohrane. Ovaj rad bavit će se SCCG algoritmom [\[18\]](#) za sažimanje skupa genoma pomoću referentnog genoma. U radu će se predstaviti implementacija ovoga algoritma, kompresija genoma i dekompresija genoma.

## 2.DNA

### 2.1. Struktura

DNA, odnosno deoksiribonukleinska kiselina, nasljedni je materijal kod ljudi i gotovo svih drugih organizama. Skoro svaka stanica u ljudskome tijelu sadrži DNA, a svaki lanac molekule DNA sastoji se od monomernih nukleotida. Nukleotidi DNA sastoje se od molekule šećera deoksiriboze na koju se veže fosfatna skupina i jedna od četiri dušične baze: adenin (A), gvanin (G), citozin (C) i timin (T). Takva struktura sačinjava jedan lanac DNA koji se veže u spiralni oblik s još jednim lancem [\[12\]](#).

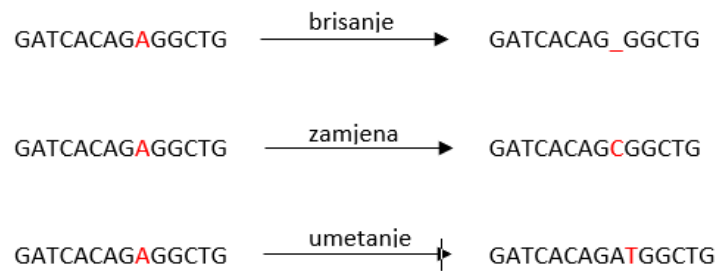
Dva lanca spajaju se preko parova baza adenin i timin te citozin i gvanin, a osim toga javlja se i pojava zvana „base-stacking“ koja je u svojoj biti dodatno učvršćivanje povezanosti dvaju lanaca slabim vodikovim vezama. Znatnu prednost u ulozi povezivanja lanaca ima povezivanje preko parova baza [\[12\]](#). Zbog takvog načina spajanja, dovoljne su nam informacije o samo jednom lancu kako bismo znali potpunu strukturu DNA, odnosno potrebno je znati duplo manje informacija za konstrukciju cjelovite DNA.

Geni su segmenti DNA koji sadrže upute za stvaranje molekula (uglavnom proteina) potrebnih za funkcioniranje tijela te se nasljeđuju od roditelja. Genom je sav genetski materijal u stanici organizma i sastoji se od približno 3 200 000 000 nukleotida [\[9\]](#). Kod ljudi i ostalih eukariotskih organizama, genom se nalazi u jezgri stanice. Kako bi sav genetski materijal stao u jezgri i bio kompaktan, on se u ljudskom organizmu dijeli u linearne molekule i nakuplja se u kromosome. Kada se govori o broju kromosoma karakterističnom za čovjeka, govori se o duplom ili diploidnom broju  $2n$ . Ljudi imaju  $2n=46$  kromosoma. Od tih 46 kromosoma, 44 su autosomi, a 2 su spolni kromosomi. U spolnim (haploidnim) stanicama nalaze se 22 autosoma i 1 spolni kromosom [\[12\]](#).

Proučavajući DNA osobe, liječnici mogu vidjeti kojim bolestima je ta osoba sklona te se može spriječiti njihov razvoj ili brzo reagirati kod već postojećih bolesti.

## 2.2. Mutacije

Mutacija je promjena u DNA koja nastaje prilikom replikacije, rekombinacije ili nekih drugih vanjskih faktora. Manifestira se brisanjem (delecija), zamjenom (supstitucija) ili umetanjem (insercija) parova baza [19].



Slika 1: mutacije

Ukoliko se mutacija dogodi na autosomima, ona se nalazi samo u nekim stanicama tijela te je često bezopasna osim kada dovodi do odumiranja stanica ili razvoja tumora. Mutacije nisu nužno loše pojave nego su i često korisne za evoluciju tako što se javljaju kao posljedice prilagodbe organizma na određene uvjete te tako jedinka ima veće šanse za preživljavanjem. Ako su nastale na spolnim kromosomima, tada su mutacije nasljedne i budući potomci će imati te mutacije u svakoj stanici svoga organizma.

## 2.3. FASTA

Nakon što smo definirali DNA, njezinu strukturu i mutacije, shvaćamo zbog čega je toliko potrebno sekvenciranje i proučavanje genoma za podizanje životnog standarda te za sprječavanje i pravovremeno reagiranje na bolesti. Pohranjene informacije o genomu spremaju se u tekstualne datoteke od kojih je među poznatijima FASTA oblik datoteke.

Format datoteke FASTA koristi se za zapisivanje sekvence uvezenog genoma. FASTA datoteka jest tekstualna datoteka koja predstavlja sekvencu za jedan kromosom. Svaka sekvenca započinje jednom linijom sa znakom „>“ i opisom niza, nakon čega slijede linije podataka o sekvenci. FASTA datoteke najčešće završavaju nastavcima .fa ili .fasta [\[13\]](#).

Svaka baza nukleotida predstavljena je pripadnim slovom: A, C, G, T i U, a koristi se i slovo N koje označava bilo koju bazu nukleotida te znak „-“ koji označava brisanje jednog slova [\[13\]](#).

Primjer:

```
>chr4
```

```
ACCCTAACCCCTNACCCTAACCCCTA-CCCTAACCCCTACCCTAACCCCTAACCC  
CTTAACCCTTAACCCTAACCCCTAACCCCTAACNNNAACCCTAACCCCTAACCC
```

```
...
```

## 3. Kodiranje

Kodiranje je proces sastavljanja niza znakova u format specijaliziran za prijenos i pohranu. Kodiranje se često koristi za redukciju veličine memorije koju zauzimaju velike datoteke. To se čini tako da se signalima/znakovima poruke dodjeljuju kodne riječi i tako je omogućeno sažimanje zbog tendencije ponavljanja sekvenci poruke. Dekodiranje jest proces vraćanja kodirane forme u originalnu poruku.

Efikasnost sažimanja mjeri se omjerom kompresije koji je definiran kao omjer broja bitova kodirane poruke naspram broju bitova originalne poruke. Idealno je koristiti tehniku kodiranja podataka koja nema gubitke, odnosno onu koja stvara kodiranu poruku koja će prilikom dekodiranja stvarati poruku identičnu originalnoj. To nije uvijek slučaj pa metode kodiranja dijelimo na one bez gubitaka i one s gubitcima [\[14\]](#).

Skupina metoda kodiranja kojom ćemo se mi baviti jest entropijsko kodiranje. To su metode kodiranja bez gubitaka što je poželjno kod kompresije i dekompresije genoma. Entropijsko kodiranje se fokusira na originalnu poruku i stvara kodne riječi za ponovljene sekvence poruke [\[14\]](#).

### 3.1. Aritmetičko kodiranje

Jedna od metoda entropijskog kodiranja jest aritmetičko kodiranje. To je metoda kodiranja koja originalnu poruku predstavlja kao interval realnih brojeva između 0 i 1. Što je poruka dulja, interval kojim se kodira postaje manji jer se svakim novim simbolom poruke, prethodni interval dodatno dijeli u manje intervale te se među njima odabire odgovarajući interval. U isto vrijeme, što je poruka dulja to se broj

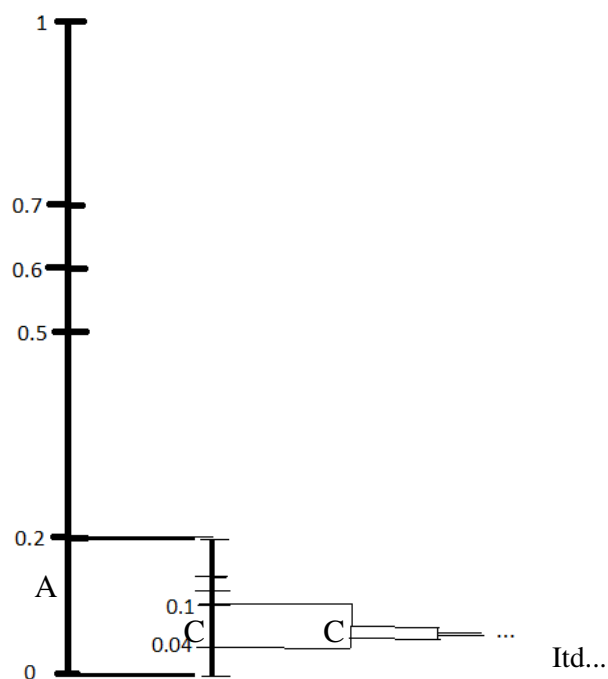
bitova potreban za prikaz i specifikaciju intervala povećava. Simboli s većom vjerojatnošću pojavljivanja smanjuju raspon intervala manje nego ostali simboli te time i manje bitova dodaju kodiranoj poruci [15].

Na početku je raspon intervala  $[0, 1)$ . Kako se čitaju svi simboli ulazne poruke, tako se svakome simbolu dodjeljuje vjerojatnost pojavljivanja te svaki simbol ima veličinu intervala razmjernu njegovoj vjerojatnosti.

| simbol | vjerojatnost | interval     |
|--------|--------------|--------------|
| A      | 0.2          | $[0, 0.2)$   |
| C      | 0.3          | $[0.2, 0.5)$ |
| G      | 0.1          | $[0.5, 0.6)$ |
| T      | 0.1          | $[0.6, 0.7)$ |
| N      | 0.3          | $[0.7, 1)$   |

Tablica 1.: Primjer za poruku ACCATGNNCN

Nakon toga se odabere onaj interval kojemu pripada prvi simbol, a zatim se odabrani interval dijeli na isti omjer kao i početni te se bira opet interval koreliran sa sljedećim simbolom u poruci i tako do kraja.



Slika 2.: prikaz aritmetičkog kodiranja za poruku ACCATGNNCN

Za navedeni primjer poruke ACCATGNNCN proces kodiranja izgleda ovako:

Početni interval: [0, 1)

Nakon A: [0, 0.2)

Nakon C: [0.04, 0.1)

Nakon C: [0.052, 0.07)

Nakon A: [0.052, 0.0556)

Nakon T: [0.05416, 0.05452)

Nakon G: [0.05434, 0.054376)

Nakon N: [0.0543652, 0.054376)

Nakon N: [0.05437276, 0.054376)

Nakon C: [0,054373408, 0,05437438)

Nakon N: [0,0543740884, 0,05437438)

Konačni interval: [0,0543740884, 0,05437438)

Formula kojom se računa interval  $[X', Y')$  iz prethodnog intervala  $[X, Y)$ , pomoću intervala  $[p1, p2)$  navedenog u tablici za svaki znak poruke je sljedeća:

$$X' = X + (X - Y) * p1$$

$$Y' = X + (X - Y) * p2$$

Primjerice početni interval  $[X, Y)$  je  $[0, 1)$ . Nakon pročitanoog znaka A, uzimamo odgovarajući interval iz tablice  $[p1, p2) = [0, 0.2)$ .

$$X = 0 + (1 - 0) * 0 = 0$$

$$Y = 0 + (1 - 0) * 0.2 = 0.2$$

Pa je  $[X', Y') = [0, 0.2)$

Poruku ne moramo kodirati s obje granice konačnog intervala, nego ju kodiramo samo jednim brojem unutar tog intervala. U danom primjeru, kodirana poruka može biti 0,0543741.

Proces dekodiranja obavlja se na sličan način: kodirana poruka u obliku realnog broja locira se unutar odgovarajućeg intervala te se odgovarajući simbol piše kao prvi znak originalne poruke. Zatim se unutar tog intervala opet pronalazi interval naše poruke i simbol se zapisuje. Ovo može ići u beskonačnost te nam zbog toga treba duljina originalne poruke kako bismo znali kada stati. Umjesto duljine poruke može se i staviti dodatan simbol na kraju kodirane poruke koji označava kraj poruke i proces dekodiranja staje kada naiđe na taj simbol [15].

Primjer dekodiranja: (nastavak na prethodni primjer)

Dekoderu se prosljeđuje broj 10 kao broj simbola originalne poruke te se za kodiranu poruku 0,0543741 traži kojem od početnih intervala ona pripada. Ona pripada intervalu  $[0, 0.2)$ . U tablici se pogleda kojem simbolu taj interval odgovara i taj simbol se zapisuje kao prvi znak poruke. U ovom slučaju to je simbol A.

Zatim se interval  $[0, 0.2)$  dijeli na intervale istih omjera kao i početni te se opet među njima traži kojem intervalu pripada kodirana poruka 0,0543741.

Intervali na koje se interval  $[0, 0.2)$  dijeli navedeni su u tablici 2:

| simbol | vjerojatnost | interval       |
|--------|--------------|----------------|
| A      | 0.2          | $[0, 0.04)$    |
| C      | 0.3          | $[0.04, 0.1)$  |
| G      | 0.1          | $[0.1, 0.12)$  |
| T      | 0.1          | $[0.12, 0.14)$ |
| N      | 0.3          | $[0.14, 0.2)$  |

Tablica 2.: Primjer za poruku ACCATGNNCN

Među njima se vidi da broj 0,0543741 spada u interval  $[0.04, 0.1)$  te je sljedeći znak poruke C. Tako se nastavlja sve dok se ne zapiše 10 simbola poruke.



## 3.2. Predviđanje djelomičnim podudaranjem

Predviđanje djelomičnim podudaranjem (PPM) je prilagodljiva tehnika kompresije koja se temelji na kontekstualnom modeliranju i predviđanju. PPM modeli koriste skup simbola u nekomprimiranom toku simbola kako bi predvidjeli sljedeći simbol u nizu [16]. SCCG algoritam koristi PPMd algoritam za kompresiju.

PPMd algoritam koristi Markovljev lanac reda  $N$ , gdje je  $N$  broj već pročitanih simbola. Pri čitanju novoga simbola, algoritam na temelju već pročitanih  $N$  simbola određuje vjerojatnost pojavljivanja tog simbola u budućnosti nakon istog konteksta. Ako pročitani simbol do sada nije bio pročitani, algoritam mu dodjeljuje vjerojatnost  $p = \frac{1}{\text{veličina abecede}}$ . Ako je određena vjerojatnost 0, algoritam će uzeti u obzir  $N-1$  pročitanih simbola u kontekst pa opet odrediti vjerojatnost. Tako će nastaviti smanjivati kontekst sve dok vjerojatnost ne postane veća od 0. Zatim se pročitani simbol kodira aritmetičkim kodiranjem [17].

Primjer:

Poruka: xzzxyxzy

$N = 9$

Slijedeći znak: x

$N'$  = broj koji određuje koliko će se pročitanih znakova nalaziti u kontekstu

-----

U tablici 3 su zapisi u obliku: *kontekst -> znak (broj pojavljivanja znaka iza konteksta)*.

Postupak PPM algoritma je slijedeći:

Za kontekst  $N'=9$  nema niti jednog pojavljivanja znaka, pa je vjerojatnost pojavljivanja znaka x iza konteksta xzzxyxzy 0. odnosno tražena vjerojatnost  $p(\text{xzzxyxzy} \rightarrow \text{x})=0$

$N'$  se smanjuje za jedan i nastavljamo postupak:

$N'=8$ :  $p(\text{zzxyxzy} \rightarrow \text{x})=0$

$N'=7$ :  $p(\text{zxyxzy} \rightarrow \text{x})=0$

$N'=6$ :  $p(\text{xyxzy} \rightarrow \text{x})=0$

$$N'=5: p(yxzzy \rightarrow x)=0$$

$$N'=4: p(xzzy \rightarrow x)=0$$

$$N'=3: p(zzy \rightarrow x)=0$$

$$N'=2: p(zy \rightarrow x)=0$$

$$N'=1: p(y \rightarrow x)=1/9$$

Nađeno je ponovljeno pojavljivanje znaka x iza konteksta y s vjerojatnošću  $p=1/9$  te se pročitani simbol x kodira aritmetičkim kodiranjem sa tom vjerojatnošću, a tablica konteksta se nadograđuje kao što je prikazano u tablici 4.

| <b>N'=8</b>     | <b>N'=7</b>                      | <b>N'=6</b>                                     | <b>N'=5</b>  |
|-----------------|----------------------------------|---|--|
| xzzxyxzz->y (1) | xzzxyxz->z (1)<br>zzxyxzz->y (1) | xzzxyx->z (1)<br>zzxyxz->z (1)<br>zxyxzz->y (1) | xzzxy->x (1)<br>zzxyx->z (1)<br>zxyxz->z (1)<br>xyxzz->y (1) |

Tablica 3.: tablica konteksta i vjerojatnosti (prvi dio)

| <b>N'=4</b> | <b>N'=3</b> | <b>N'=2</b> | <b>N'=1</b> | <b>N'=0</b> |
|-------------|-------------|-------------|-------------|-------------|
| xzzx->y (1) | xzz->x (1)  | xz->z (2)   | x->z (2)    | x (3)       |
| zzxy->x (1) | xzz->y (1)  | zz->x (1)   | z->z (2)    | y (2)       |
| zxyx->z (1) | zzx->y (1)  | zz->y (1)   | z->x (1)    | z (4)       |
| xyxz->z (1) | zxy->x (1)  | zx->y (1)   | z->y (1)    |             |
| yxzz->y (1) | xyx->z (1)  | xy->x (1)   | y->x (1)    |             |
|             | yxz->z (1)  | yx->z (1)   |             |             |

Tablica 3.: tablica konteksta i vjerojatnosti (drugi dio)

| <b>N'=9</b>        | <b>N'=8</b>        | <b>N'=7</b>    | <b>N'=6</b>   | <b>N'=5</b>  |
|--------------------|--------------------|----------------|---------------|--------------|
| xzzxyxzz->x<br>(1) | xzzxyxzz->y<br>(1) | xzzxyxz->z (1) | xzzxyx->z (1) | xzzxy->x (1) |
|                    | zzxyxzz->x<br>(1)  | zzxyxzz->y (1) | zzxyxz->z (1) | zzxyx->z (1) |
|                    |                    | zxyxzz->x (1)  | zxyxzz->y (1) | zxyxz->z (1) |
|                    |                    |                | xyxzz->x (1)  | xyxzz->y (1) |
|                    |                    |                |               | yxzz->x (1)  |

| <b>N'=4</b> | <b>N'=3</b> | <b>N'=2</b> | <b>N'=1</b> | <b>N'=0</b> |
|-------------|-------------|-------------|-------------|-------------|
| xzzx->y (1) | xzz->x (1)  | xz->z (2)   | x->z (2)    | x (4)       |
| zzxy->x (1) | xzz->y (1)  | zz->x (1)   | z->z (2)    | y (2)       |
| zxyx->z (1) | zzx->y (1)  | zz->y (1)   | z->x (1)    | z (4)       |
| xyxz->z (1) | zxy->x (1)  | zx->y (1)   | z->y (1)    |             |
| yxzz->y (1) | xyx->z (1)  | xy->x (1)   | y->x (2)    |             |
| xzzy->x (1) | yxz->z (1)  | yx->z (1)   |             |             |
|             | zzy->x (1)  | zy->x (1)   |             |             |

Tablica 4.: nova tablica konteksta i vjerojatnosti

### 3.3. Kompresija genoma

Zbog velike potrebe za sekvenciranjem i pohranom sekvenciranog genoma, razvili su se razni algoritmi namijenjeni specifično za kompresiju i dekompresiju genoma. Genomi su veliki nizovi sastavljeni od četiri simbola odnosno slova (A, C, G i T) te imaju iznimnu tendenciju ponavljanja niza simbola. Algoritmi temeljeni na stvaranju rječnika izvrsno iskorištavaju svojstvo ponavljanja nizova simbola u sekvenci genoma.

Najpoznatiji takvi algoritmi su LZ77, LZ78 i LZW [14]. Oni funkcioniraju tako da kada naiđu na niz koji se ponavlja, upisuju ga u rječnik i njime zamjenjuju svaku pojavu toga niza [14].

Primjer rada LZW kodera:

Poruka: ATTATATAC

| indeks | riječ |
|--------|-------|
| (1)    | A     |
| (2)    | T     |
| (3)    | C     |

Tablica 5.: početni rječnik

| korak | Ulaz (mjesto u nizu) | (indeks) Sadržaj rječnika | izlaz |
|-------|----------------------|---------------------------|-------|
| (1)   | A (1)                | (4) AT                    | (1)   |
| (2)   | T (2)                | (5) TT                    | (2)   |
| (3)   | T (3)                | (6) TA                    | (2)   |
| (4)   | A (4)                | (7) ATA                   | (4)   |
| (5)   | A (6)                | (8) ATAC                  | (7)   |
| (6)   | C (9)                |                           | (3)   |

Tablica 6.: nadogradnja rječnika

Tablica 6 se popunjava po sljedećem principu: Na ulazu čitamo prvi simbol poruke i to postaje radna riječ. Ako ima još simbola iza pročitano simbola, čita se sljedeći simbol i niz od ta dva simbola sačinjavaju novu radnu riječ. U rječniku se traži postoji li ista takva riječ. Ukoliko postoji, čita se sljedeći simbol iz poruke i pročitani simboli tvore novu radnu riječ. Postupak se ponavlja sve dok se u rječniku više ne može naći odgovarajuća riječ. Kada se u rječniku više ne pronalazi radna riječ, tada se radna riječ zapisuje u rječnik kao nova riječ, a na izlaz kodera šalje se kod za zadnju pronađenu radnu riječ u rječniku. Postupak stvaranja nove radne riječi ponavlja se sa početkom u simbolu koji je zadnji bio pročitani. Iz danog primjera u koraku 1 radna riječ je simbol A, ona postoji u rječniku pod indeksom (1) pa se čita sljedeći simbol T i radna riječ postaje AT. Ustanovi se da AT ne postoji u rječniku te se ona zapisuje kao nova riječ rječnika pod indeksom (4), a na izlaz se šalje indeks (1). Zatim se u drugom koraku uzima već pročitani simbol T koji čini novu radnu riječ. Postupak se ponavlja do kraja poruke.

Druga vrsta algoritama za sažimanje genoma temelji se na korištenju referentnog genoma pomoću kojeg sažimamo ciljani genom. Ljudski genomi su slični u 99% baza [20] pa je potrebno samo spremati razlike dvaju genoma kako bismo maksimizirali uštedu memorije. Traže se najdulji identični podnizovi te se oni zapisuju pod „match“ nizove, a oni koji se ne podudaraju spadaju u „mismatch“ nizove [18].

Primjer:

Referentni genom: ACCACTAGGC

Ciljani genom: ACCTTAGGA

„match“ oznaka niza: M(početni indeks, duljina)

„mismatch“ oznaka niza: Mi

Sažeti genom: M(1, 3), Mi(T), Mi(T), M(6, 4), Mi(A)

Najviše specijaliziranih algoritama za sažimanje genoma upravo koristi metodu sažimanja pomoću referentnog genoma zbog njezine velike efikasnosti.

## 4. Tablica raspršenog adresiranja

Raspršeno adresiranje je tehnika koja se koristi za jedinstvenu identifikaciju određenog objekta iz grupe sličnih objekata. Za pohranu objekata možemo koristiti jednostavne strukture podataka poput polja. Za nesortirana polja duljine  $n$ , vremenska složenost pretraživanja polja je  $O(n)$ , a za sortirana polja  $O(\log n)$ . U slučajevima kada su podatci velike vrijednosti ili kada je mnogo elemenata potrebno pohraniti, ovakve strukture podataka su dosta ne efikasne te bi se trebalo koristiti raspršivanje čija je vremenska složenost  $O(1)$ .

U raspršivanju, hash funkcija preslikava objekte u indekse pod kojima ćemo spremati te objekte u hash tablici. Hash funkcija pozvana nad objektom većinom uzima ASCII vrijednosti znakova tog niza i primjenjuje određene matematičke operacije nad njima. Jedan jednostavan primjer raspršenog adresiranja prikazan je na slici 3 gdje hash funkcija zbraja ASCII vrijednosti svih znakova u nizu te primjenjuje modulo( $n$ ) nad tim zbrojem, gdje je  $n$  broj mjesta u tablici.

Niz znakova  $x=x_0x_1\dots x_{m-1}x_m$

Duljina hash tablice  $n=6$

$\text{hashFunction}(x)=\sum_{i=0}^m \text{ASCII}(x_i)\%n$

-----

$\text{hashFunction}(\text{Fakultet})=4$

$\text{hashFunction}(\text{elektrotehnike})=2$

$\text{hashFunction}(i)=3$

$\text{hashFunction}(\text{racunarstva})=0$

|   |                |
|---|----------------|
| 0 | racunarstva    |
| 1 |                |
| 2 | elektrotehnike |
| 3 | i              |
| 4 | Fakultet       |
| 5 |                |

Slika 3.: primjer raspršenog adresiranja

Ponekad više objekata može imati istu hash vrijednost i ta se pojava zove kolizija. Kolizija se rješava tako da objekte istih hash vrijednosti ulančavamo u strukture liste na odgovarajućem indeksu u hash tablici. Primjer rješavanja kolizije dat je na slici 4.

Niz znakova  $x=x_0x_1\dots x_{m-1}x_m$

Duljina hash tablice  $n=6$

$$\text{hashFunction}(x)=\sum_{i=0}^m \text{ASCII}(x_i)\%n$$

-----  
 $\text{hashFunction}(\text{Fakultet})=4$

$$\text{hashFunction}(\text{elektrotehnike})=2$$

$$\text{hashFunction}(i)=3$$

$$\text{hashFunction}(\text{racunarstva})=0$$

$$\text{hashFunction}(\text{FER})=5$$

$$\text{hashFunction}(\text{bioinformatika})=5$$

|   |                        |
|---|------------------------|
| 0 | racunarstva            |
| 1 |                        |
| 2 | elektrotehnike         |
| 3 | i                      |
| 4 | Fakultet               |
| 5 | FER ● → bioinformatika |

Slika 4.: primjer raspršenog adresiranja i kolizije

## 5. SCCG

SCCG algoritam [18] je jedan od algoritama kompresije genoma koji koristi referentni genom. Upravo se proučavanjem i implementacijom SCCG algoritma bavi ovaj rad.

Prednost i važna značajka ovoga algoritma jest to što se ne koristi isključivo hash tablica za segmente, niti isključivo globalna hash tablica za sekvence, nego se kombiniraju obje. Kada je velika sličnost između ciljne i referentne sekvence, tada se koristi lokalno podudaranje, a inače globalno. [18]

### 5.1. Ulaz i pretprocesiranje

Za ulazne podatke upisujemo putanju do fasta datoteke referentne sekvence genoma (R) i do sekvence ciljanog genoma (T). Zatim se ta dva niza obrađuju tako da se sva slova pretvaraju u velika slova, a položaj malih slova bilježi se u međudatoteku.

## 5.2. Lokalno podudaranje

Nakon što su R i T pretprocesirani, algoritam kreće tražiti podudaranja među njima. Uvijek prvo primjenjujemo strategiju lokalnog podudaranja zbog pretpostavke da su R i T dovoljno slični, ukoliko se ispostavi suprotno onda se primjenjuje strategija globalnog podudaranja. Pretprocesirani T i R se dijele na segmente:  $t_1, t_2, t_3, \dots, t_n$  i  $r_1, r_2, r_3, \dots, r_m$ . Svi segmenti osim  $t_n$  i  $r_m$  imaju određenu duljinu L, a  $t_n$  i  $r_m$  su segmenti duljine manje ili jednake L.

Za svaki podniz u  $t_i$  tražimo najdulje podudaranje u  $r_i$ , a za to se koristi hash tablica odnosno tablica raspršenog adresiranja s obzirom na to da je za takvo pretraživanje vremenska složenost skoro  $O(1)$  što rezultira bržim rezultatima. No hash tablica zauzima dosta memorije pa se stoga za svaki ulaz u tablicu uzima najmanje jedan niz znakova duljine k koji se naziva k-mer. Parsiranjem od prvog simbola u  $r_i$  računamo hash vrijednost odgovarajućeg k-mera hash funkcijom, a početak k-mera se zapisuje u hash tablicu pod indeksom pripadajuće hash vrijednosti. Taj proces se ponavlja i za drugi simbol u  $r_i$ , i tako se nastavlja dok se ne obradi L-k+1 k-mera. Kada se isti k-mer pojavi na različitim pozicijama u  $r_i$ , svi se njihovi položaji upisuju na isti indeks u hash tablici i nižu se u strukturu liste.

Nakon što je hash tablica dovršena, tada krećemo na izračunavanje hash vrijednosti prvog k-mera u  $t_i$  segmentu. Ukoliko u hash tablici postoji ista hash vrijednost to znači da postoji barem jedan k-mer u  $r_i$  koji je identičan promatranom k-meru u  $t_i$ . Tada se duljina podudarnih podnizova može povećavati podudaranjem uzastopnih znakova u  $t_i$  i  $r_i$  odmah nakon spomenutih k-mera sve dok ne naiđemo na simbole koji se više ne podudaraju. Time se završava podudaranje podniza.

Ukoliko se za k-mer u  $t_i$  nađe više identičnih k-mera u  $r_i$  tada se opisani postupak podudaranja podnizova ponavlja za svaki od tih k-mera kako bi se našlo najdulje podudaranje. Ako se pronade više najduljih podnizova, tada se odabire onaj čija je početna pozicija najbliža završnoj poziciji prethodnog podudaranja jer se nakon pretraživanja podudaranja izvodi delta kodiranje pozicija svih podudaranja te se bolji rezultat postiže kada su početne pozicije uzastopnih podudaranja blizu.

Nakon što se pronade najdulji podniz koji se podudara, njegova se početna pozicija (p) iz referentnog segmenta i duljina (l) zapisuju u međudatoteku. Zapisuje se



i simbol iz  $t_i$  koji je prekinuo podudaranje nizova. Zatim se uzima sljedeći simbol u segmentu i ponovno se u hash tablici traži hash vrijednost k-mera koji započinje tim simbolom. Ukoliko se ta hash vrijednost ne nalazi u tablici, tada se pohranjuje prvi simbol u k-meru. Algoritam se ponovno pozicionira na sljedeći simbol u nizu i ponavlja postupak. Ovaj postupak traje sve dok se ne nađe k-mer ciljnog segmenta čija se hash vrijednost nalazi u hash tablici. Tada se opet pokreće postupak produljenja podudarnih podnizova. U međudatoteci u kojoj se pohranjuju rezultati. Dva susjedna podudaranja su predstavljena parovima  $(p, l)$  koji su odvojeni barem jednim simbolom nepodudaranja.

Ako niti jedan k-mer iz ciljnog segmenta  $t_i$  ne može pronaći svoju hash vrijednost u hash tablici referentnih k-mera iz  $r_i$ , tada smanjujemo duljinu k-mera sa  $k$  na  $k'$  kako bismo povećali vjerojatnost pronalaženja podudarnih k-mera. Ako ni nakon ovog koraka nema podudarnih k-mera, tada se cijeli segment  $t_i$  zapisuje u međudatoteku. Ako se mnogo cijelih segmenata  $t_i$  direktno zapisuje u međudatoteku tada je lokalno podudaranje slabe efikasnosti te valja prijeći na globalno podudaranje.

[\[18\]](#)

### 5.3. Globalno podudaranje

U ovoj fazi podudaranja, referentna i ciljna sekvenca nisu segmentirane. Tako se za svaki podniz u  $T$  može tražiti odgovarajuće podudaranje po cijeloj  $R$  sekvenci. Time se povećava mogućnost pronalaženja podudarnog niza u  $R$ . Za sekvence koje dijele visok stupanj podudarnosti globalno će podudaranje davati mnogo lošije rezultate nego lokalno podudaranje, a delta kodiranje ne može učinkovito kodirati takve rezultate. Zbog toga se koristi pažljivo osmišljen kriterij kojim se određuje treba li ciljna sekvenca biti obrađena u fazi globalnog ili lokalnog podudaranja.

Tijekom lokalnog podudaranja se prati omjer broja izravno pohranjenih simbola naspram duljini segmenta. Ukoliko je taj omjer veći od određenog praga  $T_1$  tada će se taj segment smatrati slabo podudarnim. Ako zbroj slabo podudarnih segmenata i nepodudarnih segmenata dosegne određeni prag  $T_2$ , tada se smatra da su te dvije sekvence slabo podudarne. Osim simbola A, C, G i T, pojavljuje se i simbol N koji označava bilo koju dušičnu bazu. Zbog toga se ne broje pojavljivanja za one

parove uzastopno nepodudarajućih segmenata u kojima je jedan od segmenata sačinjen od samo N simbola.

Ukoliko se utvrdilo da je potrebno prijeći na globalnu strategiju podudaranja, osim promjene svih malih slova u velika slova, brišu se svi N simboli iz oba niza. Položaji malih slova i N simbola se zapisuju u međudatoteku.

Hash tablica se stvara iz cijele sekvence R te time zauzima znatno više prostora nego hash tablica u lokalnom podudaranju. Metode raspršenog adresiranja k-mera iz T sekvence i produljenja podudarajućih podnizova su u biti temeljene na onima iz lokalnog podudaranja s jednom razlikom, a to je da tijekom pretraživanja najduljeg podudaranja isprva ograničavamo raspon pretraživanja ponajviše radi izbjegavanja pronalaženja podudaranja koje je predaleko od prethodnog podudaranja. To je preventivna mjera da ne bismo dobili rezultate korištenjem pohlepne strategije podudaranja. Pohlepni algoritmi ciljaju na lokalni optimum koji često ne vodi do globalnog optimuma i zbog toga nisu poželjni. Ako unutar definiranog ograničenja pretraživanja nije pronađeno podudaranje, tada se ograničenje uklanja. [18]

## 5.4. Postprocesiranje

Nakon pretraživanja podudarnih podnizova, uzastopno ponavljajući parovi  $(p_n, l_n)$  koji se pojavljuju kada se podudarajući podnizovi nalaze preko granica segmenata, bit će spojeni, a zatim će pozicije  $p_n$  u tim parovima biti delta kodirane. Delta kodiranjem ćemo zabilježiti samo relativni prirast pozicije  $p_{n+1}$ , u odnosu na poziciju  $p_n$ , a ne apsolutnu poziciju  $p_{n+1}$ . Broj bitova potreban za zapisivanje indeksa pozicija raste porastom indeksa te se ovim pristupom relativnog adresiranja smanjuje broj bitova potreban za zapisivanje pozicija što za sobom povlači manje zauzimanja memorije i veću efikasnost.

Tako obrađena međudatoteka se zatim kompresira pomoću PPMd algoritma. PPMd je optimizirana varijanta algoritma Predviđanja djelomičnim podudaranjem koji je dio 7-zip alata.

```

static void use7zip(string filename){

    struct stat buffer;

    if (stat("result", &buffer) != 0) {
        system("mkdir result");
    }

    string executeline = "7z a " + filename + ".7z " + filename + " -m0=PPMD";
    system(executeline.c_str());
}

```

Slika 5.: implementacija PPMd algoritma

Algoritam PPMd se poziva iz terminala kao što je prikazano na slici 5. Primjerice za file chr19.fa, naredba koja će se pozvati glasi „7z a chr19.fa.7z chr19.fa -m0=PPMD“ te se tako chr19.fa kompresira u datoteku chr19.fa.7z.

```

2,232
A
2,669
A
1,214
G
2,44
T
2,1054
G
1,247
G
2,257
A
2,73
A
2,1176
CGTCTGG
45,222
A
5,485
T
2,51
A

```

Slika 6.: Primjer zapisa nakon postprocesiranja.

Na slici 6 vide se parovi  $(p_n, l_n)$ , iza kojih slijedi jedan ili više nepodudarnih simbola. Primjerice prvi par na slici  $(2, 232)$  označava da je pozicija početnog simbola tog podudaranja udaljena za 2 od pozicije prošlog podudaranja, a duljina podudaranja iznosi 232. Podudaranje je terminirano prvim nepodudarajućim simbolom A.

## 5.5. Dekompresija

Kako bismo dekodirali naš dobiveni kompresirani genom, potreban nam je referentni genom korišten za kompresiju ciljnog genoma i kompresirani ciljni genom. Ciljni genom se rekonstruira izvlačenjem nizova iz  $R$  pomoću  $(p_n, l_n)$  parova te zapisivanjem nepodudarnih simbola i segmenata između tih nizova. Brzina, ali i složenost algoritma za dekompresiju je znatno manja od kompresije.

## 5.6. Primjer kompresije i dekompresije

```

genome > hg19 > chr19.fa
8039 GCGGGT CCTCCCGCCACCCACCCACGATGCGCTTAATCAACAGGCCCTG
8040 CCTGCGGTGTAACCTGGAGAGAGGCTCTGGACACAGTCACTGTACAGT
8041 AGGGTAAACTGAGGCTCAGGCGCCGGCCCGGGAAATGGCCGCCCTCGT
8042 GGGAGGAGCATGTGCGGGT CCGGCT CCGCCT CCTCTTTTATTGAAG
8043 TCATTTTTCTCCTCCTGT CAGCGGAAC TGGCCGTGGGCAGCCCTCCTC
8044 CCACCCCCAGGAGCTGAGAAGAAAAAAAAGCTTAATCGAATCCGGCACT
8045 GGGGCGGTGGTGCCGCAGAGAGAGGCCGAGGGAGCCCGCCCTGGGTGG
8046 GTCTGAGGCCGCTGTAATTATGGCCCTGATGTGATTATTAATTATCG
8047 TCGCTATTTTGTGCCGCTTCCGCCGGCGAGCCTGTGGATTGAGCTCA
8048 CTGCACACAAGCCGCCAGGGCTCATCACAGACAGCCCTCGGCCCGGG
8049 AGCCTGAGACCCAGAGAAGCCACGCAC TGGCCGGGCCACACCCACGTTG
8050 GAAGGTGGGGCTTAAAAC TACCCTGCGGCTCCCCACACAGTCAGGGTT
8051 AGAGGTTCAAGGCCCAAGAGAGGACAGGCCGCAAGGCAGGGGCCCTGGGA
8052 AGGTGTTACTCCTCCTCCTGGATGGGAGAGCCCACTGGATCAGTGTTC
8053 TGTACCTTAAAAAATGAATCAGccggacgtggggcgaagcctgtaaat
8054 cccagcactttggaggccgaggcaggcgtatcacctgaggtcaggaatt
8055 tgagacgagcctggccaaacttggtaaaacccgtctctactaaaaataca
8056 aaaaaactagctgggtgtggggcggcctgtaactcagctatttggg
8057 aggtggggcagggaatcgcttgaacccctggaagaagaggtacagtgga
8058 gccgagactgcacctgtactccagcctggcgacacagcaagactccg
8059 tctcaaaaaa caa caaa caaa caaa caaa caaa AAACTGAAA caagccg
8060 gcttgagcaaacaggcactttaccggccaccctctgagctgaaatggt
8061 ggcagttatggacttcaggcatagctggatccaggtgtccgcaatgtct
8062 ccaggcttctgactcttcatctgttttccctatgatggctcccctctga
8063 agaaggGCATCTTTTGTAAATTTGCACCAAAACGGGGT GCCCCCCGCTTG
8064 TGGT GCCCTGGCCTT CCTGAGGGT CGGGAGACTCAGGATGGCCGCCAGT
8065 CTGAAAGGAGAGACTGCTGGCAGGTGTGCTGCCTCTTCAACGCACAGC
8066 ACCCACTGGGGCCGAGCGCTCTGTGGCGCCAGGGATGCTGGAGGTGCAAA
8067 CAGTAGATT CGTTCTGCTCAGGATCAACATCA CAAGGCAGCAAA CGGCCA
8068 CAGCCAGGAGGGAACAAGGGCATCTGACTCCCCGGGGTGGGGGCTGACC
8069 TGAAAGGCAGTGGGCCAAGACCTGAGGCAGGGCGAGACACCAAGCAGCGG
8070 CAGAATCCCTGGATGACGCCACTTGGAAAGTGGAGGGGGAGGGCATGGG
8071 CAGCTGACGCTCCAGACCAAGcaca caa caagcaca caa caagtcac
8072 accagcagcaca caggcaca caagcaca cagggtcaca catgcat
8073 acaaggctcagtgcttacagtgca cagcagcaca cagcagcaca cagcagcaca
  
```

Slika 7.: zapis chr19.fa prije kompresije

Na slici 7 prikazan je odsječak kromosoma chr19.fa iz skupa podataka hg19 (Human genome verzija 19) [23]. To nam je ciljani odsječak genoma, a za referentni ćemo uzeti chr19.fa iz skupa podataka hg18 (Human genome verzija 18) [22].

```
magdalena@DESKTOP-GC408P0:/mnt/c/Users/User/Desktop/3.GODINA/ZAVRSNI/moj_zavrzni/kod$ make SCCGC
g++ SCCGC.cpp -o SCCGC
magdalena@DESKTOP-GC408P0:/mnt/c/Users/User/Desktop/3.GODINA/ZAVRSNI/moj_zavrzni/kod$ ./SCCGC ./genome/hg18/chr19.fa ./genome/hg19/chr19.fa ./result
./genome/hg19/chr19.fa is compressing...
Local attempt time: 289.468seconds

7-Zip [64] 16.02 : Copyright (c) 1999-2016 Igor Pavlov : 2016-05-21
p7zip Version 16.02 (locale=C.UTF-8,Utf16=on,HugeFiles=on,64 bits,4 CPUs Intel(R) Core(TM) i5-6300U CPU @ 2.40GHz (406E3),ASM,AES-NI)

Scanning the drive:
1 file, 5829 bytes (6 KiB)

Creating archive: ./result/chr19.fa.7z

Items to compress: 1

Files read from disk: 1
Archive size: 1879 bytes (2 KiB)
Everything is Ok
Compressed time: 376.328 seconds
Compressed file is: ./result/chr19.fa.7z
Done
```

Slika 8.: pozivanje SCCGC programa iz Ubuntu terminala

Za navedeni referentni i ciljani kromosom naredba koja pokreće SCCGC algoritam je „./SCCGC ./genome/hg18/chr19.fa ./genome/hg19/chr19.fa ./result“. U terminalu se prvo ispisuje dojava o početku rada algoritma „./genome/hg19/chr19.fa is compressing“. Algoritam pokušava naći lokalno podudaranje. Ako se procijeni da to podudaranje nije efikasno, on ispisuje liniju „Local attempt time: “ i vremenski period izvođenja lokalnog podudaranja. Nakon toga algoritam kreće tražiti globalno podudaranje.

Nakon što je kodiranje uspješno obavljeno ispisuje se poruka o 7-Zip kompresiranju i dobivena fasta datoteka je kompresira u .7z datoteku. Na posljetku dobivamo poruku o završetku rada programa, o ukupnom proteklom vremenskom periodu te o relativnoj putanji do konačne datoteke.

U chr19.fa.7z datoteci nalazi se chr19.fa datoteka koja sadrži zapis poput zapisa prikazanog na slikama 9 i 10. Slika 9 prikazuje početak dobivene datoteke chr19.fa. Na početku se nalazi znak „>“ iza kojega slijedi opis naše datoteke. U drugoj liniji je zapisana duljina niza znakova jedne linije iz originalnog ciljnog kromosoma. Na ovome primjeru piše 50, što znači da svaka linija u originalnom zapisu kromosoma chr19 ima 50 znakova. Treća linija u dobivenom zapisu je vrlo dugačka. Ona sadrži

zapisane pozicije malih slova. One su zapisane tako da prvi broj predstavlja prvo pojavljivanje niza malih slova, a drugi broj predstavlja duljinu tog niza. Zatim sljedeći broj predstavlja udaljenost od kraja prethodnog niza do novog pojavljivanja niza malih slova, a broj iza njega duljinu tog niza, i tako do kraja. Na ovome primjeru imamo prvi niz malih slova koji počinje indeksom 60514, i sadrži 141 znak. Sljedeći niz se pojavljuje 900 pozicija iza završetka prethodnog niza, a sadrži 460 znakova. To nam je potrebno kako bismo u procesu dekodiranja znali zamijeniti velika slova malima tamo gdje ona pripadaju.

Na slici 10. prikazana je četvrta linija koja sadrži isti princip zapisa kao i treća linija samo ne za pozicije malih slova, nego „N“ slova. Dakle u ovome primjeru prvi niz „N“ slova počinje indeksom 0 i duljine je 60000, sljedeći niz „N“ slova počinje indeksom udaljenim za 7286004 pozicija nakon završetka prethodnog niza i njegova duljina jest 50000 itd. Sve ostale linije u datoteci prikazuju parove  $(p_n, l_n)$  podudarajućih nizova i nepodudarajuće simbole između njih. Princip je već opisan uz sliku 6.

```

1 >chr19
2 50
3 60514 141 900 460 123 148 1270 126 1045 252 432 234 34 125 67 184 1908 50 177 61 147 262 3 81 132 249 315 181 208 80 1599 296 145 189 162 154 7 278 120
503 1087 196 11 408 10 43 22 510 7 60 96 118 90 150 3 356 296 276 1552 386 302 67 259 132 65 2038 39 297 3 98 2 163 486 905 85 134 739 2510 134 100 92
119 19 23 176 77 60 169 171 199 1325 1101 612 28 325 25 58 55 959 295 268 92 1509 35 685 437 2 373 3745 152 1171 307 1009 536 3 28 272 262 1639 239 1311
131 52 476 1367 247 261 169 1666 25 137 288 11 232 414 276 96 88 268 82 21 712 112 668 10 155 3 1049 3 2491 17 34 504 342 3 141 6 21 5 2170 78 979 3 570
16 3674 43 572 45 71 101 263 21 283 570 107 914 349 61 31 61 265 4 233 167 76 7 162 27 584 36 1036 51 740 35 2978 18 475 41 1447 3 854 3 368 26 436 8 93
87 97 448 828 8 120 2 488 16 899 20 1083 1170 6422 749 1504 276 107 2 555 2937 131 1479 100 406 141 10 47 17 246 50 136 847 83 521 308 391 189 113 53
1228 159 805 47 183 420 29 405 315 343 106 244 15 266 216 3198 5 160 3 361 136 304 498 69 131 498 1110 409 101 464 390 136 149 102 132 137 77 280 23 116
165 1225 3 489 283 723 333 44 85 380 1025 467 135 295 73 73 1364 54 40 96 365 70 7 112 4 558 164 301 1089 27 49 200 2 768 128 28 388 109 125 269 437 531
364 292 3 212 16 841 8 220 3092 308 5168 419 267 152 4 283 130 95 24 108 237 299 112 293 602 134 255 419 286 79 3 1776 37 182 276 103 296 107 34 124 14
431 303 347 2 763 361 229 28 87 70 158 2 293 254 273 11 372 3 104 60 410 35 729 22 301 4 114 185 307 212 93 146 322 2 131 100 607 2 571 28 350 4 1759 2
1062 46 194 215 63 378 119 3 300 40 119 143 38 71 307 48 312 4 242 43 102 221 107 12 144 343 283 2 35 323 223 1718 334 78 97 309 276 2 317 1137 256 26
24 285 336 312 303 746 278 34 133 634 302 62 294 696 301 129 427 31 305 177 296 745 443 759 50 687 80 1467 612 227 329 117 292 24 316 148 66 855 28 1196
34 129 219 83 279 349 197 230 39 9 465 5 2128 53 105 17 610 6 563 6 187 214 999 248 111 42 447 14 213 8 941 26 66 26 67 26 67 4 300 224 2280 869 444 166
66 498 180 247 418 631 272 8 259 3 294 12 535 102 362 33 302 5 337 4 80 4 355 41 241 15 289 2 69 5 329 10 318 11 299 9 1582 1512 29 440 300 22 1806 1222
435 3 302 24 745 23 1984 45 162 22 302 1708 295 957 304 69 184 14 152 7 294 1496 210 5 157 1705 69 480 199 53 1023 21 80 4 258 2 296 21 296 14 166 53

```

Slika 9.: Početak kompresiranog chr19.fa

```

3977 111 45 258 2 310 7 2702 9 1059 12 314 3 300 746 460 1413 97 117 272 20 289 926 146 49 517 2 45 6 124 11 298 3 226 171 799 3 962 3 1131 9 649 2 421
40 348 1358 682 509 99 953 293 224 538 1009 124 151 294 3429 453 11 474 14 318 1591 477 3 659 1754 508 377 1326 962 125 1049 280 8 29 3 304 408 308 22
381 314 126 15 2761 633 119 521 28 190 21 310 340 234 241 1545 74 3142 573 119 524 4 3113 3 208 2 297 18 224 35 104 8 207 12 91 4 303 71 113 73 317 10
303 102 299 62 313 51 294 321 278 242 117 530 25 576 336 765 200 1036 253 2402 61 1788 177 282 114 3131 302 2982 762 693 539 93 88 225 465 27 109 11 28
8 1003 38 123 37 168 2 299 21 302 25 31 38 289 84 65 4 135 151 289 5 143 604 62 2 244 62 696 46 263 9 294 95 316 4 292 7 295 1266 317 36 294 177 379 74
607 509 642 5 292 8 166 294 293 83 60 405 420 344 90 965 128 1907 87 472 27 70 303 85 54 12 157 50 158 14 158 14 167 68 369 95 302 111 32 8 305 13 1245
4 134 30 303 6 197 2 156 28 311 121 355 36 197 17 1469 27 317 3 307 880 309 1303 24 574 28 2512 294 138 138 3387 158 1266 1226 390 616 591 414 2 419 2
302 7 83 125 303 1041 197 8 52 4 301 15 307 3 160 301 215 323 645 3 623 55 265 9 1846 508 229 123 69 2 177 3 116 39 301 4 386 5 155 129 93 61 113 376
688 4 446 48 314 236 131 50 587 1166 300 60 138 4728 28 26 297 3 294 8 136 794 22 32 49 154 20 108 23 5038 37 2481 385 64 286 24 302 117 135 3261 33
1165 28 27 245 723 70 1662 103 6 272 418 103 1959 266 123 372 141 626 20 282 28 303 3 297 231 131 80 302 507 811 41 207 2911 289 1789 99 479 500 57 312
1130 596 28 293 121 136 11 299 18 160 451 191 221 857 188 381 163 504 1250 46 134 283 611 67 727 1235 116 74 108 137 4 96 25 173 523 145 269 198 357 273
84 219 129 36 1195 28 1037 102 4 409 242 118 750 443 869 2204 3 84 41 805 39 361 21 38 59 1279 41 466 607 446 4 67 26 67 26 67 26 67 26 67 26 67
26 67 26 67 26 937 7 217 13 451 42 111 250 1000 210 187 6 553 12 610 17 105 53 2028
4 0 60000 7286004 50000 1291194 50000 11786217 50000 4058367 3100000 31387201 10000
5 0,20345133
6
7 2,187
8
9 1,243
10
11 2,170
12
13 2,559
14
15 2,322
16

```

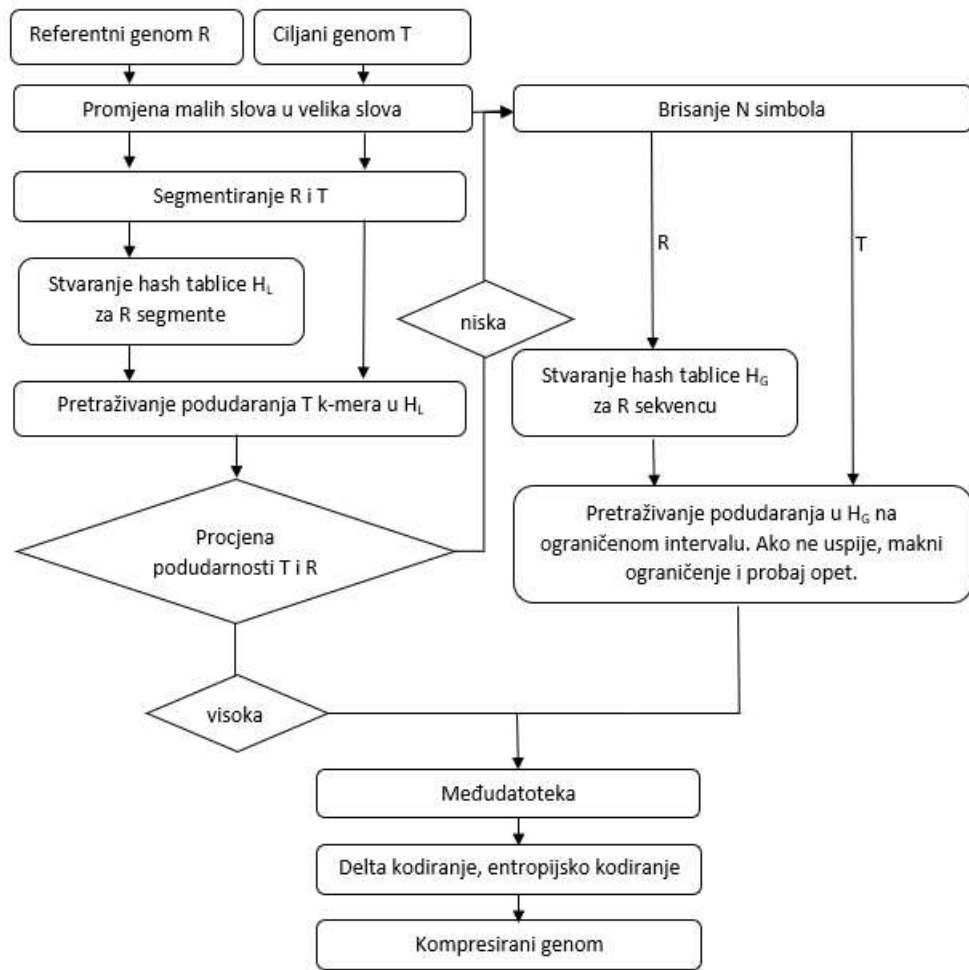
Slika 10.: Nastavak kompresiranog chr19.fa

Pokretanje algoritma SCCGD za dekompresiju se vrši pozivom naredbe „./SCCGD ./genome/hg18/chr19.fa ./result/chr19.fa ./dresult“ kao na slici 11. Prvo se ispisuje linija „./result/chr19.fa is decompressing...“ koja označava početak rada programa, a zatim nakon što je dekompresiranje gotovo, ispisuje se vrijeme koje je bilo potrebno za obavljanje dekompresije i relativna putanja do stvorene datoteke.

```
magdalena@DESKTOP-GC408P0: /mnt/c/Users/User/Desktop/FER/3.GODINA/ZAVRSNI/moj_zavrsni/kod$ ./SCCGD ./genome/hg18/chr19.fa ./result/chr19.fa ./dresult
./result/chr19.fa is decompressing...
Decompressed time: 5.86693 seconds
Decompressed time: ./dresult/chr19.fa
Done
-----
```

Slika 11.: pozivanje SCCGD programa iz Ubuntu terminala

Dobiveni rezultat je identična datoteka kao i početna datoteka ciljnog chr19 kromosoma.



slika 12.: Shema SCCGC algoritma za kompresiju



## 6. Implementacija u C++-u.

Program na početku očitava argumente koji su mu proslijeđeni u terminalu. Prvi argument je naziv programa koji pokreće. Drugi argument je putanja do referentnog genoma. Treći argument je putanja do ciljanog genoma i četvrti argument je folder u koji se rezultat kompresiranja sprema.

Primjer unosa kojim se pokreće program:

```
./SCCGC ./genome/hg18/chr19.fa ./genome/hg19/chr19.fa ./result
```

Prvi argument      Drugi argument      Treći argument      Četvrti argument

Kada se u string varijable pohrane argumenti, tada se pozivaju funkcije koje iz FASTA datoteka čitaju sekvence i pohranjuju ih u string varijable.

```
string reference_seq = LocalReadSeq(greference);  
string target_seq = LocalReadSeq(gtarget);
```

Slika 13.: Pohrana sekvenci iz FASTA datoteka

Kako bismo zabilježili pozicije k-mera, malih slova, N slova i podnizova, napravljena su dva razreda: razred Position i razred newkmer kao što je prikazano na slikama 14 i 15.

```
class Position{  
private:  
    int startinRef;  
    int endinRef;  
    int startinTar;  
    int endinTar;  
public:  
    int getstartinRef() {  
        return startinRef;  
    }  
    void setstartinRef(int startinRef) {  
        this->startinRef = startinRef;  
    }  
    int getendinRef() {  
        return endinRef;  
    }  
    void setendinRef(int endinRef) {  
        this->endinRef = endinRef;  
    }  
    int getstartinTar() {  
        return startinTar;  
    }  
    void setstartinTar(int startinTar) {  
        this->startinTar = startinTar;  
    }  
    int getendinTar() {  
        return endinTar;  
    }  
    void setendinTar(int endinTar) {  
        this->endinTar = endinTar;  
    }  
};
```

Slika 14.: razred Position

```

class newkmer {
private:
    string kmer;
    int kmerstart;
public:
    string getkmer() {
        return kmer;
    }

    void setkmer(string kmer) {
        this->kmer = kmer;
    }

    int getkmerstart() {
        return kmerstart;
    }

    void setkmerstart(int kmerstart) {
        this->kmerstart = kmerstart;
    }
};

```

Slika 15.: razred newkmer

Zatim se se zapisuju pozicije malih slova iz sekvence ciljnog genoma, a nakon toga i referentnu i ciljnu sekvencu obrađujemo pretvorbom svih malih slova u velika slova.

```
list<Position> L_list = lowercase_position(target_seq);
```

Slika 16.: Pohrana pozicija malih slova ciljnog genoma

```

static list<Position> lowercase_position(string sequence){
    list<Position> list;
    bool successive = false;
    int start = 0, end = 0;

    for(int i = 0; i < sequence.length(); i++){
        if(islower(sequence.at(i))){
            if(successive){
                end+=1;
            }else{
                start=i;
                end+=1;
                successive=true;
            }
        }else{
            if(successive){
                Position position;
                position.setstartinTar(start);
                position.setendinTar(end - 1);
                list.push_back(position);
            }
            successive = false;
            start = 0;
            end = i + 1;
        }
    }

    if(successive){
        Position position;
        position.setstartinTar(start);
        position.setendinTar(end - 1);
        list.push_back(position);
    }

    return list;
}

```

Slika 17.: metoda koja pamti pozicije malih slova

```
transform(reference_seq.begin(), reference_seq.end(),reference_seq.begin(), ::toupper);
transform(target_seq.begin(), target_seq.end(),target_seq.begin(), ::toupper);
```

Slika 18.: Pretvorba malih u velika slova

Referentna i ciljna sekvenca se dijele na segmente i kreće se s pozivanjem funkcije za lokalno podudaranje.

```
reference = reference_seq.substr(startOfReference, endOfReference-startOfReference);
target = target_seq.substr(startOfTarget, endOfTarget-startOfTarget);

//segmentation-based Local Matching Phase

list<Position> list = LocalMatching(reference, target, kmerLength);
```

Slika 19.: funkcija za lokalno podudaranje.

## 6.1. Lokalno i globalno podudaranje

Nakon pretprocesiranja, pozvana je metoda Lmatch za lokalno podudaranje R i T sekvenci. Na slici 21 prikazan je pseudokod algoritma za lokalno podudaranje. U koraku (1) određuje se duljina segment  $S_i$  koja može biti jednaka definiranoj duljini L ili manja od L. U koraku (2) kreira se tablica raspršenog adresiranja odnosno hash tablica H. Funkciju raspršenog adresiranja implementirala sam formulom:

$$hashFunction(key) = \sum_{i=0}^{n-1} key_i * 31^{n-1-i} \text{ [24].}$$

gdje je n duljina kmera, a  $key_i$  je znak u segmentu na poziciji „i“ počevši od početka segmenta. Funkcija je napravljena po uzoru na javinu implementaciju hashCode() funkcije [24].

```
int hashFunction(string key) {
    int hashCode = 0;
    for (int i = 0; i < key.length(); i++) {
        hashCode += key[i] * pow(PRIME_CONST, (key.length() - i-1));
    }
    return hashCode;
}
```

Slika 20.: implementacija hash funkcije.

**Ulaz:** referentni segment/sekvenca  $S_r$ , ciljni segment/sekvenca  $S_t$ , duljina  $k$  od  $k$ -mera, ograničenje intervala pretraživanja  $m$ , bool globalno (na početku je false).

```

(1) Pronađi L duljinu  $S_t$  segmenta.
(2) Parsiranje  $S_r$   $k$ -mera i spremanje u hash tablicu H. Postavi index = 0;
(3) while (index < L-k+1) do
(4)   Izdvoji  $k$ -mer' duljine  $k$  koji počinje indeksom u  $S_r$ ;
(5)   if (nema podudaranja za  $k$ -mer' u H)
(6)     Znak na poziciji index u  $S_t$  se smatra nepodudarnim;
(7)     index ++; continue;
(8)   else
(9)      $l_{max1}=0, l_{max2}=0, p_{n1}=0, p_{n2}=0, l_{n1}=0, l_{n2}=0$ ;
(10)    for (svaki  $k$ -mer koji je jednak  $k$ -mer') do
(11)      Produljij duljinu podudaranja  $l$  s početkom na početku zajedničkog  $k$ -mera u  $S_r$  i  $S_t$  sve dok se ne desi nepodudaranje.
(12)      if (globalno && udaljenost početne pozicije  $p$  trenutnog  $k$ -mera i završne pozicije prethodnog podudaranja je u intervalu od  $-m$  do  $m$ )
(13)        if ( $l == l_{max2}$ )
(14)          if ( $p$  trenutnog  $k$ -mera je najbliža završnoj poziciji prethodnog podudaranja)
(15)             $p_{n2}=p$ ;
(16)          end if
(17)        else if ( $l > l_{max2}$ )
(18)           $l_{max2}=l; p_{n2}=p; l_{n2}=l$ ;
(19)        end if
(20)      end if
(21)      if ( $l == l_{max1}$ )
(22)        if ( $p$  trenutnog  $k$ -mera je najbliža završnoj poziciji prethodnog podudaranja)
(23)           $p_{n1}=p$ ;
(24)        end if
(25)      else if ( $l > l_{max1}$ )
(26)         $l_{max1}=l; p_{n1}=p; l_{n1}=l$ ;
(27)      end if
(28)    end for
(29)    if (globalno &&  $p_{n2} != 0$ )  $p_n = p_{n2}, l_n = l_{n2}$ ;
(30)    else  $p_n = p_{n1}, l_n = l_{n1}$ ;
(31)    end if
(32)    zapiši ( $p_n, l_n$ ) i nepodudarne podnizove iz  $S_t$ ;
(33)    index = index +  $l_n + 1$ ;
(34)  end if
(35) end while

```

**Izlaz:** ( $p_n, l_n$ ) parovi i nepodudarni nizovi iz  $S_t$

Slika 21.: Algoritam 1

Početna duljina k-mera namještena je na 21 jer je ta duljina k-mera davala najbolje rezultate u originalnoj implementaciji [18]. Osim što je u originalnoj implementaciji duljina k-mera 21 davala najbolje rezultate, odlučila sam i sama testirati svoj kod i razmotriti brzinu izvođenja programa za različite duljine k-mera. Za referentnu sekvencu sam uzela sekvencu hg18/chrY.fa kromosoma, a za ciljnu sekvencu uzela sam sekvencu hg19/chrY.fa kromosoma. Uvjerila sam se da je duljina 21 zaista najbolja duljina za k-mere.

Rezultati se nalaze u tablici 7.

| Duljina k-mera | Vremenski period izvođenja programa [s] |
|----------------|---|
| 10             | 46.81                                   |
| 15             | 44.23                                   |
| 21             | 38.29                                   |
| 30             | 54.19                                   |
| 40             | 75.42                                   |

Tablica 7.: usporedba efikasnosti programa za različite k-mere

U koracima (4)-(7) na slici 21, za svaki indeks u segmentu  $S_t$  pronalazimo k-mer' s početkom u tom indeksu, primjenjujemo hash funkciju da dobijemo vrijednost k-mer'-a i tražimo ima li u H tablici već ta vrijednost. Ukoliko nema podudaranja, znak na početnom indeksu k-mer'-a se upisuje kao nepodudarajući, a početni indeks sljedećega k-mera' se povećava za 1. Kada k-mer' nađe podudarajuću hash vrijednost u H tablici prelazi se na korak (11).

U koraku (11) se kreće s postupkom produljenja podudaranja u kojemu se za podudarne kmere iz  $S_r$  i  $S_t$  nastavlja pretraživati jesu li podudarni i u znakovima koji slijede iza završetka k-mera. To se pretražuje sve dok se ne dođe do prvog nepodudarnog znaka koji se zapisuje kao nepodudarajući znak u konačnu datoteku.

U koracima (12)-(20) provjerava se ukoliko je podudaranje globalno ili lokalno i ako je globalno tada se provjerava jeli udaljenost dva susjedna podudaranja unutar

granica intervala ograničenja. Ukoliko je to slučaj i ako je aktualno podudaranje za određeni k-mer' najdulje pronađeno, tada se pamti pozicija i duljina tog podudaranja.

Koraci (21)-(27) se odnose na lokalno podudaranje u kojemu se provjerava jeli podudaranje koje se promatra najdulje do sada za promatrani k-mer i je li njegova početna pozicija najbliža završetku prethodnog podudaranja. Ako je to slučaj tada se pamte pozicija i duljina tog podudaranja.

Ostatak pseudokoda prikazuje zapisivanje (p, l) parova podudarajućih nizova i izravno zapisivanje nepodudarnih znakova u datoteku.

Ukoliko se podudaranje nije desilo, duljina k-mera se sa 21 spušta na 11 i ponavlja se postupak. Ako je podudaranje bilo lokalno, nakon traženja podudaranja provodi se provjera stupnja podudarnosti R i T. Ukoliko se odredi visoka podudarnost, postupak se završava, no ako se odredila slaba podudarnost, tada se algoritam prebacuje na globalno podudaranje.

## **6.2. Razlučivanje između faze lokalnog i globalnog podudaranja**

Pseudokod prikazan na slici 22 prikazuje postupak u kojemu se provjerava efikasnost lokalnog podudaranja i postupak pozivanja globalnog podudaranja.

U koracima (3)-(6) odvija se algoritam prikazan na slici 21 i provjerava se ukoliko je za ulazne segmente  $r_i$  i  $t_i$  bilo podudaranja. Ako to nije slučaj tada se kao što je već opisano, duljina kmera smanjuje sa 21 na 11 pošto je lakše naći podudarajuće nizove manje duljine.

Koraci (7)-(12) obavljaju se kada ni nakon smanjene duljine kmera, nije bilo podudaranja. Tada se broj nepodudaranja povećava za 1, ali samo kada niti  $r_i$  niti  $t_i$  segmenti nemaju u sebi N znakove. To je zato što N znakovi predstavljaju bilo koji od četiri valjana znaka (dušične baze) te postoji mogućnost da su upravo na mjestima gdje su N znakovi, mogle biti dušične baze koje bi dovele do podudarnih k-mera.

U koracima (14)-(16) računa se omjer broja izravno upisanih znakova naspram broju znakova originalne poruke. Ukoliko taj omjer prelazi granicu  $T_1$ , tada se broj

nepodudaranja povećava za jedan. U koraku (17) broj nepodudaranja se uspoređuje s granicom  $T_2$  i ukoliko je on veći od te granice, tada kreće faza globalnog podudaranja tako što se pokreće algoritam prikazan na slici 21 samo s definiranim ograničenjem intervala pretraživanja i sa bool vrijednošću koja označava globalno podudaranje.

**Ulaz:** referentna sekvenca  $R$ , ciljna sekvenca  $T$ ,  $k$ -mer duljine  $k$  i  $k'$  gdje je  $k' < k$ , segment duljine  $L$ , ograničenje pretraživanja  $m$ , pragovi podudarnosti  $T_1$  and  $T_2$ .

(1)  $R$  i  $T$  se podijele na segmente  $(r_1, r_2, r_3, \dots, r_m)$  i  $(t_1, t_2, t_3, \dots, t_n)$  istih duljina  $L$ , pretvori sva mala slova u velika i zapamti im pozicije u međudokumentu.

(2) **for**( $i = 1$  to  $n$ )**do**

(3) pozovi algoritam 1 a parametrima( $r_i, t_i, k, 0, false$ );

(4) **if** (nema podudaranja za  $(r_i, t_i)$ )

(5) pozovi algoritam 1 sa parametrima ( $r_i, t_i, k', 0, false$  );

(6) **end if**

(7) **if** (nema podudaranja za  $(r_i, t_i)$ )

(8) **if** (niti  $r_i$  niti  $t_i$  nemaju u sebi  $N$  znakove)

(9) broj\_nepodudaranja++;

(10) **end if**

(11) Spremi  $t_i$  u privremeni file; **continue**;

(12) **end if**

(13) Spremi  $(p_n, l_n)$  parove i nepodudarne znakove;

(14) **if** (postotak nepodudarnih znakova iz  $t_i > T_1$ )

(15) broj\_nepodudaranja++;

(16) **end if**

(17) **if** (broj\_nepodudaranja  $> T_2$ )

(17) **Go to** line 21;

(18) **end if**

(19) **end for**

(20) **Go to** line 25;

(21) očisti privremeni file;

(22) Pretvori sva mala slova u velika, izbriši sve 'N' znakove. Pohrani informacije o 'N' nizovima i pozicijama malih slova iz  $T$  u pomoćni file.

(23) pozovi algoritam 1 sa parametrima (procesirani  $R$ , procesirani  $T$ ,  $k$ ,  $m$ , true);

(24) Pohrani  $(p_n, l_n)$  parove i nepodudarne znakove iz  $T$ ;

(25) Pstprocesiraj  $p_n$  u  $(p_n, l_n)$  parovima delta kodiranjem;

**Output:** Kompresirani genom.

Slika 22.: Algoritam 2

### 6.3. Dekompresija

Za dekompresiju genoma kao parametre u terminalu prilikom pozivanja programa unosimo naziv programa, putanju do referentnog genoma korištenog za kompresiju ciljnog genoma, putanju do kompresiranog ciljnog genoma te folder u koji će se spremiti dekodirani ciljni genom.

Primjer unosa u terminal: `./SCCGD ./genome/hg18/chr19.fa ./result/chr19.fa ./dresult`

## 7. Rezultati

Testiranje koda radila sam na svome računalu u Ubuntu terminalu, detaljnije specifikacije računala navedene su u tablici 8.

|                    |  |
|--------------------|--|
| Operacijski sustav | 20.04.2 LTS  |
| Procesor           | Intel(R) Core(TM) i5-6300U CPU @<br>2.40GHz 2.50 GHz |
| RAM                | 16 GB  |

Tablica 8.: Specifikacije računala



## 7.1. Usporedba SCCGC implementacije i originala

U ovome dijelu rada uspoređivat ću originalnu implementaciju i moju implementaciju SCCG algoritma za kompresiju. Uspoređivat ću performanse mjerene na više ciljnih i više referentnih FASTA datoteka ljudskih kromosoma različitih veličina.

Dobiveni rezultati se nalaze u tablici 9. Za parove ciljnih i referentnog kromosoma sam radila više mjerenja i srednju vrijednost upisala u tablicu.

| <b>Ciljni kromosom:</b>      | <b>Referentni kromosom:</b>  | <b>Brzina originala[s]</b> | <b>Brzina implementacije [s]</b>         |
|------------------------------|------------------------------|----------------------------|--|
| hg18/chr19.fa<br>(63 563kB)  | hg17/chr19.fa<br>(63 563 kB) | 19.21                      | 65.03                                    |
| hg19/chr19.fa<br>(58 899 kB) | hg18/chr19.fa<br>(63 563 kB) | 28.28                      | 422.64                                   |
| hg18/chr20.fa<br>(62 193 kB) | hg17/chr20.fa<br>(62 193 kB) | 22.86                      | 70.49                                    |
| hg19/chr22.fa<br>(51 105 kB) | hg18/chr22.fa<br>(49 498 kB) | 32.39                      | Terminirano (proteklo više po pola sata) |
| hg19/chr18.fa<br>(77 773 kB) | hg18/chr18.fa<br>(75 820 kB) | 67.26                      | 91.48                                    |
| hg19/chr20.fa<br>(62 780 kB) | hg18/chr20.fa<br>(62 193 kB) | 21.98                      | 564.37                                   |
| hg18/chr20.fa<br>(62 193 kB) | hg19/chr20.fa<br>(62 780 kB) | 22.98                      | 60.09                                    |

|                             |                             |       |        |
|-----------------------------|-----------------------------|-------|--------|
| hg19/chrY.fa<br>(59 142 kB) | hg18/chrY.fa<br>(57 548 kB) | 10.65 | 45.53  |
| hg18/chrY.fa<br>(57 548 kB) | hg19/chrY.fa<br>(59 142 kB) | 10.15 | 210.04 |

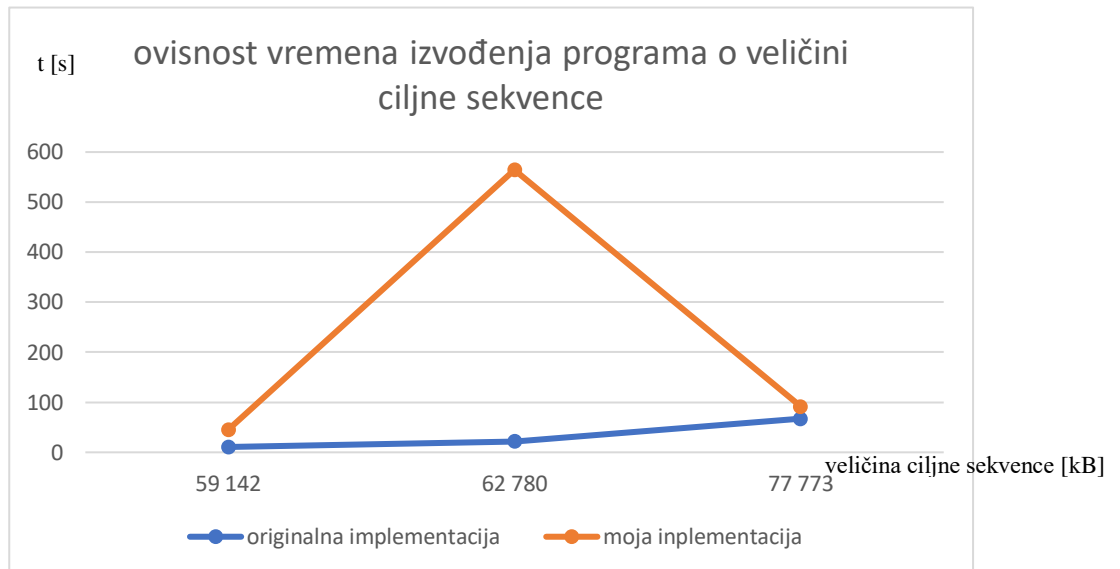
Tablica 9.: Usporedba SCCGC

Primjećuje se nerazmjerna razlika u brzinama izvođenja između ove dvije implementacije. Primijetila sam da su vremenski bolje izvedbe moje implementacije uvijek u korelaciji s time da datoteke referentne i ciljea sekvence zauzimaju istu količinu memorije. Kada se pogleda sažeti genom koji nastane, u takvim primjerima je zapisana samo jedna linija koja u kojoj je  $p=0$ , a  $l$ =veliki broj. Zbog toga shvaćam da su očito u pitanju identične sekvence kromosoma. No čak i u takvim slučajevima je originalna implementacija 3 puta brža od moje.

Iz rezultata u primjerima gdje kromosomi nisu identični, uspoređujući moju implementaciju i originalnu implementaciju vidimo da je moja puno sporije obrađivala podatke, čak 15 ili više puta sporije. Iz tablice se vidi kako na brzinu izvođenja utječe i veličina sekvence referentnog i ciljnog kromosoma. Primjerice kada mi je sekvenca referentnog kromosoma bila iščitana iz hg19/chrY.fa, a ciljnog iz hg18/chrY.fa, izvođenje programa je trajalo cca 5 puta dulje nego kada je hg18/chrY.fa bio zapis referentne sekvence, a hg19/chrY.fa ciljne. Zbog toga nije lako napraviti graf sa samo referentnim ili samo ciljnim kromosomima kao konstantama.

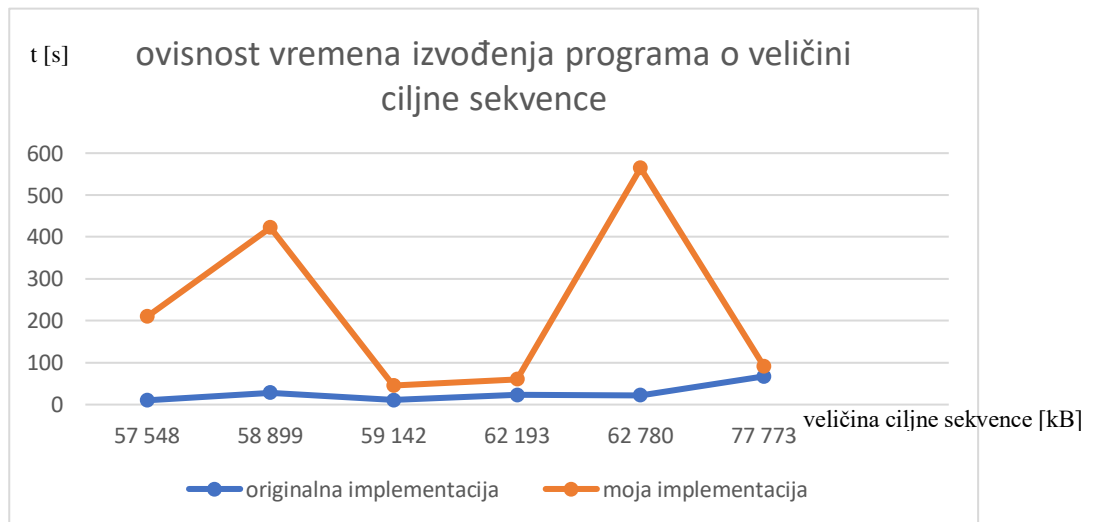
Iz tog razloga u grafu 1 prikazala sam slučajeve gdje je veličina datoteke ciljne sekvence kromosoma veća od datoteke referentne sekvence, pazeći na to da se te dvije veličine ipak ne razlikuju previše. Kao što se vidi, uvjet da ciljna sekvenca memorijski zauzima više prostora od referentne nema nikakav utjecaj na razmjernost brzine izvršavanja programa. Stoga sam u grafu 2 prikazala sve podatke sa sekvencama ciljnih kromosoma na x osi, čiji je jedini uvjet da su datoteke ciljne i referentne sekvence sličnih veličina, ali ne i identičnih. Na y-osi naravno, nalazi se vrijeme proteklo tijekom rada algoritma.

Kao što se vidi ni ovaj graf nije konzistentan za moju implementaciju. Oba grafa za originalnu implementaciju većinski konzistentno rastu s porastom veličine datoteke ciljnog kromosoma.



Graf 1.: prikaz srednje vrijednosti vremenskog trajanja s obzirom na ciljne kromosome

(Ciljni sekvenca memorijski veća od referentne)



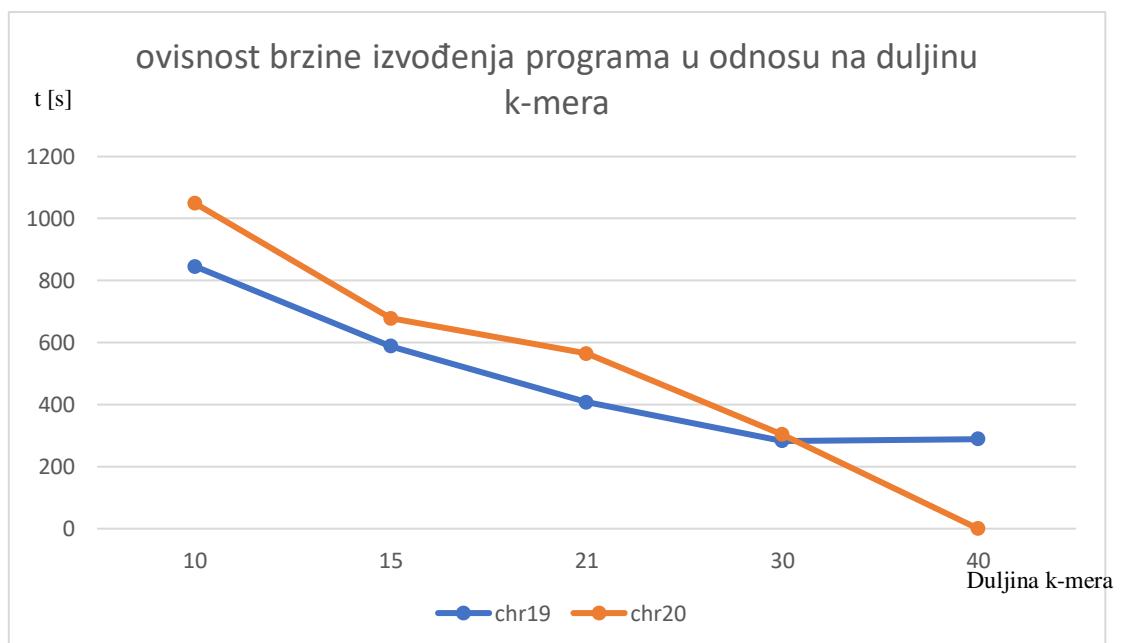
Graf 2.: prikaz srednje vrijednosti vremenskog trajanja s obzirom na ciljne kromosome

(ciljna i referentna sekvenca sličnih memorijskih veličina)

Ovi rezultati su vrlo nezadovoljavajući i nepredvidljivi. Iz tog razloga sam ispitala izvođenje programa za različite duljine k-mera za dva para R i T sekvenci koji su davali najlošije rezultate. Ishod je prikazan u tablici 10 i grafu 3.

| <b>Duljina k-mera:</b> | <b>Srednja vrijednost<br/>brzine izvođenja<br/>programa [s] za:<br/>Referentni kromosom:<br/>hg18/chr19.fa<br/>Ciljni kromosom:<br/>hg19/chr19.fa</b> | <b>Srednja vrijednost<br/>brzine izvođenja<br/>programa [s] za:<br/>Referentni kromosom:<br/>hg18/chr20.fa<br/>Ciljni kromosom:<br/>hg19/chr20.fa</b> |
|------------------------|---|---|
| 10                     | 845.87  | 1049.17   |
| 15                     | 588.05  | 678.43  |
| 21                     | 407.90  | 564.37  |
| 30                     | 282.60  | 303.92  |
| 40                     | 288.70  | 268.95  |

Tablica 10.: Analiza brzine izvođenja SCCGC algoritma za različite k-mere



Graf 3.: Analiza brzine izvođenja SCCGC algoritma za različite k-mere

Nakon ove analize, vidim da duljina k-mera značajno utječe na brzinu izvođenja programa za navedene sekvence. Zanimljivo je da sekvence koje su za duljinu k-mera 21 davale dobre rezultate, za veće duljine k-mera daju lošije rezultate, dok sekvence koje su za duljinu k-mera 21 davale loše rezultate, povećanjem duljine k-mera daju puno bolje rezultate. Iako i dalje brzine nisu slične brzinama koje daje originalna implementacija algoritma.

## 7.1. Izvedba SCCGD implementacije

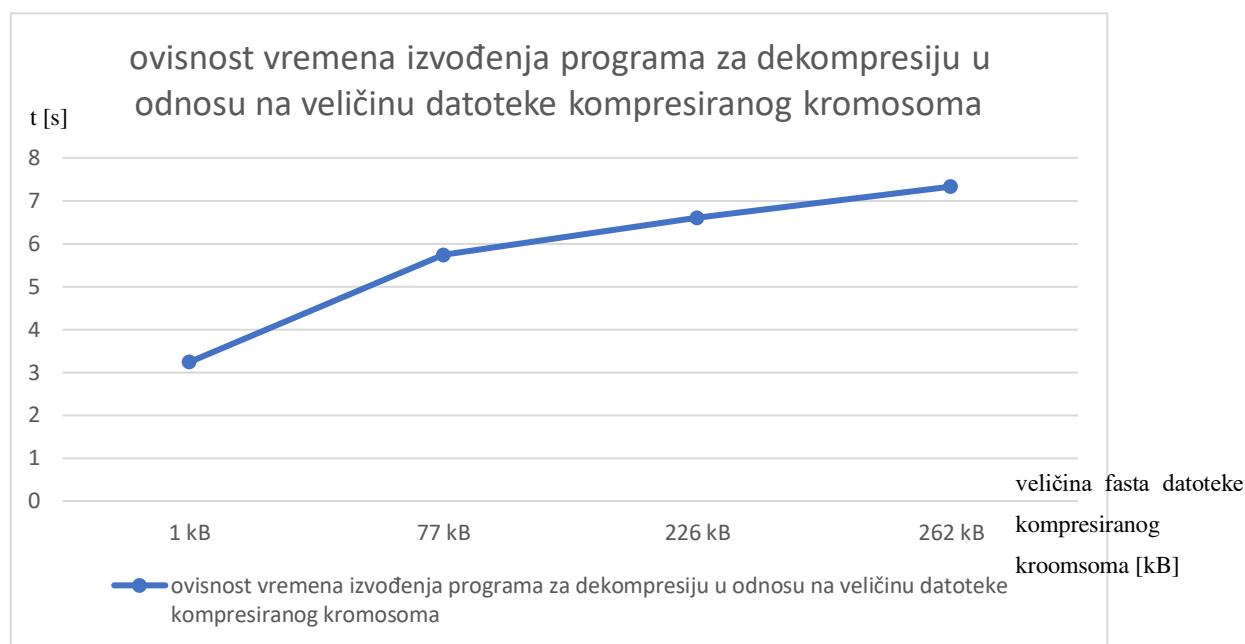
U ovom dijelu rada ću prokomentirati vlastitu implementaciju algoritma za dekompresiju. Algoritam za dekompresiju je dosta jednostavniji algoritam od onoga za kompresiju te dobri rezultati nisu iznenađujući. U ovome dijelu nisam uspoređivala svoju implementaciju sa originalnom jer su rezultati bili vrlo slični.

| <b>Referentni kromosom:</b>  | <b>Originalni ciljni kromosom:</b> | <b>Kompresirani kromosom:</b> | <b>Brzina izvođenja[s]:</b> |
|------------------------------|------------------------------------|-------------------------------|-----------------------------|
| hg18/chr19.fa<br>(63 563 kB) | hg19/chr19.fa<br>(58 899 kB)       | result/chr19.fa<br>(226 kB)   | 6.60                        |
| hg18/chr20.fa<br>(62 193 kB) | hg17/chr20.fa<br>(62 193 kB)       | result/chr20.fa<br>(1 kB)     | 3.24                        |
| hg18/chr18.fa<br>(75 820 kB) | hg19/chr18.fa<br>(77 773 kB)       | result/chr18.fa<br>(262 kB)   | 7.33                        |
| hg19/chrY.fa<br>(59 142 kB)  | hg18/chrY.fa<br>(57 548 kB)        | result/chrY.fa<br>(77 kB)     | 5.74                        |

Tablica 10.: Analiza brzine izvođenja SCCGD algoritma

Ovi rezultati dekompresiranja imaju približno linearnu ovisnost vremenskog trajanja o veličinama sažete datoteke. Stoga graf 4 na x-os ima veličine sažetih

kromosoma, a na y-osi vrijeme trajanja dekompresiranja istih i daje približno linearnu krivulju.



Graf 4.: prikaz vremenskog trajanja s obzirom na kompresirane kromosome

## 8. Zaključak

Zbog povećane potrebe za smanjenjem količine memorije korištene za spremanje sekvence ljudskog genoma, razvijen je algoritam SCCG kao jedno od rješenja problema. Algoritam koristi tablicu raspršenog adresiranja te entropijsko kodiranje. Algoritam je vrlo djelotvoran u svome cilju te je primjerice uspio sažeti sekvencu ljudskog kromosoma veličine 58 899 kB u kompresirani oblik od 6 kB. To je skoro 10000 manje zauzete memorije.

Moja implementacija algoritma, pisana u programskom jeziku c++, efikasno kompresira genome/kromosome i smanjuje količinu memorije koju zauzimaju njihovi zapisi kao što je već spomenuto. No mojoj je implementaciji mnogo vremena potrebno da napravi tu kompresiju dok originalnom algoritmu pisanom u javi nije potrebno toliko vremena. Vremenski je originalna implementacija superiornija te je, ovisno o ulaznim podatcima, od 3 do 15 puta efikasnija. U nekim slučajevima moja

implementacija čak ni nije obavila kompresiju do kraja u vremenskom periodu od pola sata te je terminirana.

Za kraj, zaključak je da bi bilo dobro ustanoviti zbog čega se implementacija SCCG algoritma u c++ nije najbolje realizirala i to u budućnosti popraviti. SCCG algoritam sam po sebi ima veliki potencijal za uspjeh.

## 9. Literatura

- [1] Long walk to genomics: History and current approaches to genome sequencing and assembly, Computational and Structural Biotechnology Journal, 2020, Pages 9-19, from <https://www.sciencedirect.com/science/article/pii/S2001037019303277>
- [2] W T. Godbey, Chapter 11 - Genetic engineering, Pages 247-284, Biotechnology and its Applications (Second Edition), Academic Press, 2022 from <https://www.sciencedirect.com/science/article/pii/B9780128177266000113>
- [3] Human Genome Project FAQ. (2022). Retrieved 23 June 2022, from <https://www.genome.gov/human-genome-project/Completion-FAQ>
- [4] The Cost of Sequencing a Human Genome. (2022). Retrieved 23 June 2022, from <https://www.genome.gov/about-genomics/fact-sheets/Sequencing-Human-Genome-cost>
- [5] The Importance of Data Compression in the Field of Genomics – EMBS. (2022). Retrieved 23 June 2022, from <https://www.embs.org/pulse/articles/the-importance-of-data-compression-in-the-field-of-genomics/>
- [6] BioSpace. (2021, December 28). *DNA sequencing market size to reach USD 12.55 billion in 2028, says reports and data*. BioSpace. Retrieved June 10, 2022, from <https://www.biospace.com/article/dna-sequencing-market-size-to-reach-usd-12-55-billion-in-2028-says-reports-and-data/>
- [7] Stephens, Z. D., Lee, S. Y., Faghri, F., Campbell, R. H., Zhai, C., Efron, M. J., Iyer, R., Schatz, M. C., Sinha, S., & Robinson, G. E. (n.d.). *Big data: Astronomical or genomics?* PLOS Biology. Retrieved June 10, 2022, from <https://journals.plos.org/plosbiology/article?id=10.1371%2Fjournal.pbio.1002195>
- [8] *Genome sequence data: Management, storage, and ... - future science*. (n.d.). Retrieved June 10, 2022, from <https://www.future-science.com/doi/full/10.2144/000113134>
- [9] Doležel, J., & Greilhuber, J. (2010). Nuclear genome size: are we getting closer?. *Cytometry Part A*, 77(7), 635-642.
- [10] The size of life: 350 kilobytes?, T. (2022). The size of life: 350 kilobytes?. Retrieved 23 June 2022, from <https://berthub.eu/dna-book/dna-size-matters/>

- [11] Lee, H., Gurtowski, J., Yoo, S., Nattestad, M., Marcus, S., Goodwin, S., ... & Schatz, M. C. (2016). Third-generation sequencing and the future of genomics. *BioRxiv*, 048603.
- [12] Brown TA. *Genomes*. 2nd edition. Oxford: Wiley-Liss; 2002. Chapter 1, The Human Genome. Available from: <https://www.ncbi.nlm.nih.gov/books/NBK21134/>
- [13] U.S. National Library of Medicine. (n.d.). *Blast topics*. National Center for Biotechnology Information. Retrieved June 10, 2022, from [https://blast.ncbi.nlm.nih.gov/Blast.cgi?CMD=Web&PAGE\\_TYPE=BlastDocs&DOC\\_TYPE=BlastHelp](https://blast.ncbi.nlm.nih.gov/Blast.cgi?CMD=Web&PAGE_TYPE=BlastDocs&DOC_TYPE=BlastHelp)
- [14] Pandžić, I. S. (2009). Uvod u teoriju informacije i kodiranje. *Element*.
- [15] Witten, I. H., Neal, R. M., & Cleary, J. G. (1987). Arithmetic coding for data compression. *Communications of the ACM*, 30(6), 520-540.
- [16] Wikimedia Foundation. (2021, July 4). *Prediction by partial matching*. Wikipedia. Retrieved June 10, 2022, from [https://en.wikipedia.org/wiki/Prediction\\_by\\_partial\\_matching](https://en.wikipedia.org/wiki/Prediction_by_partial_matching)
- [17] Google. (n.d.). *Handbook of Data Compression*. Google Knjige. Retrieved June 10, 2022, from <https://books.google.hr/books?id=LHCY4VbiFqAC&lpg=PR7&ots=N4x5NIH7h7&lr&hl=hr&pg=PA296#v=onepage&q&f=false>
- [18] Wei Shi, Jianhua Chen, Mao Luo, Min Chen, High efficiency referential genome compression algorithm, *Bioinformatics*, Volume 35, Issue 12, June 2019, Pages 2058–2065, from <https://academic.oup.com/bioinformatics/article/35/12/2058/5165377>
- [19] Mutation. (2022). Retrieved 9 June 2022, from <https://www.genome.gov/genetics-glossary/Mutation>
- [20] Kredens KV, Martins JV, Dordal OB, Ferrandin M, Herai RH, Scalabrin EE, et al. (2020) Vertical lossless genomic data compression tools for assembled genomes: A systematic literature review. *PLoS ONE* 15(5): e0232942. <https://doi.org/10.1371/journal.pone.0232942>
- [21] Index of /goldenPath/hg18/chromosomes. (2022). Retrieved 9 June 2022, from <https://hgdownload.soe.ucsc.edu/goldenPath/hg17/chromosomes/>
- [22] Index of /goldenPath/hg18/chromosomes. (2022). Retrieved 9 June 2022, from <https://hgdownload.soe.ucsc.edu/goldenPath/hg18/chromosomes/>
- [23] Index of /goldenPath/hg18/chromosomes. (2022). Retrieved 9 June 2022, from <https://hgdownload.soe.ucsc.edu/goldenPath/hg19/chromosomes/>
- [24] String (Java Platform SE 6). (2022). Retrieved 9 June 2022, from [https://docs.oracle.com/javase/6/docs/api/java/lang/String.html#hashCode\(\)](https://docs.oracle.com/javase/6/docs/api/java/lang/String.html#hashCode())



## 10. Sažetak

### Naslov

Sažimanje genoma pomoću referentnog genoma

### Sažetak

Razvojem medicine i tehnologije danas je moguće analiziranje DNA kao pomoć pri liječenju pacijenata ili u sprječavanju razvoja nasljednih bolesti. Dugačke genome (npr. Za pohranu genoma čovjeka potrebno je oko 3 GB) potrebno je sažeti prije spremanja. Ovaj rad se bavi SCCG algoritmom za sažimanje skupa genoma pomoću referentnog genoma. U radu će se predstaviti implementacija ovoga algoritma, kompresija genoma i dekompresija genoma.

**Ključne riječi:** kompresija genoma, referentni genom, SCCG algoritam, tablica raspršenog adresiranja, k-meri

## **Title**

Genome compression using a reference genome

## **Abstract**

With the development of medicine and technology, it is now possible to analyze DNA as an aid in treating patients or in preventing the development of hereditary diseases. Long genomes (e.g. it takes about 3 GB to store the human genome) need to be compressed before they are stored. This paper deals with the SCCG algorithm for genome compression using a reference genome. The paper will present the implementation of this algorithm, genome compression and genome decompression.

**Key words:** genome compression, refence genome, SCCG algorithm, hash table, k-mer