SVEUČILIŠTE U ZAGREBU

**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

**Bioinformatika 1**

**High-speed and high-ratio referential genome compression algorithm**

*Katarina Mišura, Marko Marfat*

Voditelj: *Mirjana Domazet-Lošo*

Zagreb, svibanj, 2022.

**Sadržaj**

[Uvod 1](#_Toc104160650)

[1. Koraci algoritma 2](#_Toc104160651)

[1.1. Pretprocesiranje 2](#_Toc104160652)

[1.2. Pohlepno podudaranje pomoću *hash* tablice 3](#_Toc104160653)

[1.3. Postprocesiranje 5](#_Toc104160654)

[1.4. Dekompresija 6](#_Toc104160655)

[2. Rezultati testiranja 7](#_Toc104160656)

[Zaključak 9](#_Toc104160657)

[Literatura 10](#_Toc104160658)

# Uvod

U ovome će se projektu, iz predmeta Bioinformatika 1, implementirati algoritam za kompresiju genoma. Puni naziv algoritma je „High-speed and high-ratio referential genome compression“ ili „HiRGC“, a cilj mu je uvesti poboljšane performanse u odnosu na već dostupne algoritme za kompresiju genoma [1].

Motivacija za uvođenje ovakvog algoritma je sve veći broj sekvenciranih genoma koji zahtijevaju velike količine memorijskog prostora za pohranu. Neke od tradicionalnih metoda kompresije poput „gzip-a“ (<http://www.gzip.org/>) i „bzip2-a“ (<http://www.bzip.org>) nisu dovoljno učinkovite u kompresiji genomskih podataka. Razlog tomu je što ovakve metode ne iskorištavaju intrinzična svojstva DNA sekvenci poput male abecede, čestih ponavljanja i palindroma. Preferirani pristup je kompresija bez gubitaka temeljena na pohranjivanju razlike između ciljnog genoma i referentnog genoma. Upravo to je pristup na kojemu se zasniva ovaj algoritam.

# Koraci algoritma

Pri pokretanju program kao argumente prima imena referentnog i ciljnog FASTA dokumenta. Nakon provjere točnosti unesenih argumenata može se započeti daljnji proces. Cijeli algoritam možemo podijeliti u četiri cjeline: pretprocesiranje, pohlepno podudaranje pomoću *hash* tablice, postprocesiranje i dekompresija. Svaka cjelina je opisana u nastavku uz pripadajući primjer.

## **Pretprocesiranje**

Pretprocesiranje je proces pripreme sekvenci referentnog i ciljnog FASTA dokumenta za daljnje uspoređivanje i kompresiju. Cilj pretprocesiranja je izdvojiti znakove A,C,G i T iz sekvenci ciljnog i referentnog FASTA dokumenta. Prilikom uklanjanja preostalih znakova iz ciljne sekvence, potrebno je sačuvati poziciju intervala tih znakova i duljinu intervala. Pri obradi referentne sekvence, potrebno je ukloniti preostale znakove, ali nije potrebno sačuvati informacije o njihovoj poziciji. Očuvane pomoćne informacije o znakovima koji se ne nalaze u skupu znakova A,C,G,T bit će potrebne kasnije pri dekompresiji i ponovnom sastavljanju cjelokupne ciljne sekvence. Pretprocesiranje možemo podijeliti u tri dijela. Prvi dio je procesiranje ciljne sekvence, drugi dio je procesiranje referentne sekvence, a treći dio je spremanje pomoćnih informacija o ciljnoj sekvenci u dokument.

Funkcija *target\_preprocess()* kao argument prima niz znakova *file\_name* koji predstavlja ime FASTA dokumenta iz kojeg učitavamo ciljnu sekvencu. Funkcija započinje čitanjem redaka iz dokumenta i izdvajanjem prvog retka u kojem se nalaze informacije o sekvenci u niz znakova *id\_tg*. Ostali redci se čitaju jedan po jedan i spremaju se u niz znakova *t\_seq\_L*, a duljina svakog retka se sprema u vektor brojeva *t\_seq\_len*. Nakon učitavanja svih redaka iz dokumenta u niz znakova zatvara se pristup dokumentu. Prvo se pretražuju sva mala slova u sekvenci, sprema se pozicija prvog pronađenog malog slova u intervalu i duljina intervala. Svi znakovi koji su slova se pretvore u velika slova i prepisuju u novi niz znakova *t\_seq\_L1* . Nastavlja se pretraga i spremanje pozicije prvog pronađenog znaka N i duljina intervala. Uklanjaju se znakovi N i preostali znakovi se prepisuju u novi niz znakova *t\_seq\_L2*. Posljednji prolaz kroz sekvencu sprema pozicije prvog pronađenog znaka koji nije A,C,G,T, sprema sve znakove u intervalu i duljinu tog intervala. Prepisivanjem znakova A,C,G,T u novi niz znakova *t\_seq\_L3* dobije se konačna sekvenca. Radi lakše usporedbe i uštede memorije, nad sekvencom se vrši enkodiranje znakova prema pravilu: znak A se zamjenjuje znakom 0, znak C znakom 1, znak G znakom 2 i znak T znakom 3. Konačna sekvenca koja se koristi za daljnju usporedbu nalazi se u nizu znakova *t\_final*.

Funkcija *reference\_preprocess()* kao argument prima niz znakova *file\_name* koji predstavlja ime FASTA dokumenta iz kojeg učitavamo referentnu sekvencu. Izdvojimo prvi redak s informacijama o referentnoj sekvenci, a ostale retke spremimo u niz znakova *r\_seq\_L*. Vrši se pretvorba znakova u velike znakove i prepisivanje u novi niz znakova *r\_seq\_L1*. Potrebno je ukloniti sve znakove koji nisu A,C,G ili T, stoga se samo ti znakovi prepisuju u novi niz znakova *r\_seq\_L3*. Konačno se vrši enkodiranje prema istom pravilu za enkodiranje ciljne sekvence. Konačna sekvenca koja se koristi u daljnjoj usporedbi spremljena je u niz znakova *r\_final*.

Funkcija *saveDataToFile()* kao argument prima pokazivač na izlazni tok datoteke *&myfile*, a služi za spremanje dodatnih informacija o ciljnoj sekvenci. U dokument prvo se upisuje dužina intervala malih znakova i njihova duljina. Zatim se u sljedeći red upisuje dužina intervala znakova N i njihova duljina. Par interval i njegova duljina međusobno su odvojeni znakom „-“, a parovi su odvojeni razmakom. Posljednje se upisuje dužina intervala znakova koji nisu A,C,G,T ili N i svi znakovi u intervalu. Par dužina intervala i pripadajući znakovi koji čine interval odvojeni su znakom „-“, a parovi su odvojeni razmakom.

## **Pohlepno podudaranje pomoću *hash* tablice**

Nakon pretprocesiranja znakovi A, C, G i T poprimaju sljedeće vrijednosti:

Sljedeći je korak pohraniti takav niz cijelih brojeva veličine 2 • k bitova u strukturu n-torki. Te n-torke se računaju na sljedeći način:

gdje predstavlja niz cijelih brojeva (), a gdje

(n - k + 1) predstavlja broj preklapajućih n-torki. Sada je, pomoću ove konstrukcije, moguće inicijalizirati *hash* tablicu pomoću koje će se kasnije provoditi *greedy matching* algoritam. Ta tablica se inicijalizira na sljedeći način [1]:

1: za i = 1 do (nr - k + 1):

2: izračunaj // za referentnu sekvencu

3: osvježi vrijednosti u *hash* tablici

loc(i) = point( mod s), point( mod s) = i

4: kraj

Nakon što je na ovaj način generirana *hash* tablica, potrebno je implementirati i sam *greedy matching* algoritam. Cilj ovog algoritma je pronaći mjesta na kojima se poklapaju ciljna i referentna sekvenca. Način na koji to radi je taj da prvo računa i njenu odgovarajuću *hash* vrijednost, ali ovoga puta za ciljnu sekvencu. Zatim provjerava postoji li takva n-torka igdje u referentnoj sekvenci. S obzirom na to da *hash* tablica ne garantira da ako dvije n-torke imaju jednake *hash* vrijednosti da će biti identične mi zapravo gledamo „koševe“ (engl. *bucket*) unutar kojih se nalazi nekoliko potencijalnih kandidata (ili niti jedan). U slučaju da u košu ne postoji niti jedan kandidat, n-torka iz ciljne sekvence se ne poklapa s referentnom sekvencom te algoritam zatim prelazi na (i+1) n-torku. U drugom slučaju, ako postoje kandidati, algoritam pregledava poklapaju li se brojevi iz ciljne i referentne sekvence. Algoritam radi tu provjeru dok god se poklapaju brojevi te tako povećava najduži poklapajući niz. U slučaju kada algoritam naiđe na dva jednako duga poklapajuća niza, on uzima onaj koji je prvi. Svaki takav niz se zapisuje u datoteku u obliku (pozicija poklapanja, duljina poklapanja). Nizovi koji nemaju poklapanje zapisuju se u datoteku u svom originalnom obliku. Taj proces se može opisati sljedećim pseudokodom [1]:

1: postavi i = 0 i p = 0

2: dok i > (nt - k + 1):

3: izračunaj

4: postavi j = point( mod s), pmax = -1 i lmax = 0

5: dok j ≠ 0:

6: postavi l = 0

7: uvjet1 = (j + l) < len(t) // t-target

8: uvjet2 = (j + l) < len(r) // r-reference

9: uvjet3 = r(j + 1) = t(i + l)

10: dok (uvjet1 i uvjet2 i uvjet3):

11: l = l + 1

12: završi dok

13: ako (l >= k i l > lmax):

14: pmax = j, lmax=l

15: završi ako

16: osvježi j = loc[j]

17: ako (lmax > 0):

18: podniz t(p,....,i-1) je nepoklapajuć

19: (pmax, lmax) je zapis poklapajućeg niza

20: osvježi p = i + lmax

21: završi ako

22: osvježi i = i + lmax + 1

23: završi dok

24: preostali niz je nepoklapajuć

Primjenom ovog algoritma na primjeru ispod postižemo sljedeći rezultat:

r = 0032302230103002021

t = 003210223012130020022

Rezultat = [(1, 4), 1, (6, 6), 21, (14, 5), 0, (7, 2)]

## **Postprocesiranje**

Nakon primjene *greedy matching* algoritma i pohranjivanja rezultata u datoteku (uz ostale informacije) tu datoteku je potrebno i na neki način kompresirati. U radu se spominju tri pristupa: delta enkodiranje, *run length encoding* i PPMd kompresija. S obzirom da autori nisu primijenili delta enkodiranje [2] ono neće biti primijenjeno ni u našem radu. PPMd je način kompresije te je dostupan u programu 7zip (<https://www.7-zip.org/>) i on će se koristiti, ali se neće dodatno implementirati već će se koristiti gotova implementacija iz 7zip programa. Prije same kompresije još je potrebno provesti *run length encoding*. To je kodiranje koje nastoji smanjiti količinu ponovljene informacije, a njegov rad se najbolje može opisati na primjeru.

Recimo da imamo sljedeći niz:

5 5 5 5 5 5 4 4 4 4 5 5 5 5 2

Umjesto da se zapisuju svi znakovi (njih 15, ako se ne broje razmaci) oni se efikasnije mogu zapisati na sljedeći način:

5-6 4-4 5-4 2

Prvi broj predstavlja znak, a broj nakon crtice predstavlja broj ponavljanja tog znaka. Ovaj zapis sada sadrži svega 10 znakova što je poboljšanje od 33%. Autori su u svome radu odlučili primijeniti ovo kodiranje samo na završetke redaka što se može ilustrirati sljedećim primjerom.

aaaaaaa\n

aaaaaaa\n

aaaaaaa\n

aaaaaaa\n

aaaaaaa\n

aaaaaaaaa\n

Umjesto da se pohranjuju svi znakovi koji označavaju sljedeći redak (\n), radije će biti zapisani u sljedećem formatu:

8-5 10-1

Prvi broj predstavlja mjesto znaka novog retka, a drugi broj predstavlja koliko redova nakon ima jednako mjesto novog retka. Na ovako malom primjeru ne vidi se velika prednost ovakvog zapisa, ali je ona itekako prisutna u slučajevima s puno redaka.

## **Dekompresija**

S obzirom na to da su svi opisani algoritmi reverzibilni (PPMd, *run length encoding*, *greedy matching*, pretprocesiranje), sve što je potrebno napraviti za dekompresiju je rekonstruirati originalnu datoteku pomoću svih pohranjenih informacija. S obzirom na to da su svi algoritmi već detaljno opisani u postupku kompresije, smatramo da nije potrebno dodatno objašnjavati provođenje dekompresije s obzirom da ona samo prati obratni slijed koraka.

# Rezultati testiranja

Preostalo je provesti testiranje naše implementacije i usporediti rezultate s originalnom implementacijom. Testiranje se provodi u virtualnom okruženju na Ubuntu operativnom sustavu s 3 GB RAM memorije. Zbog ovoga ograničenja bilo je potrebno prilagoditi maksimalnu veličinu *hash* tablice s 1 << 30 na 1 << 20, odnosno veličina *hash* tablice je smanjena s 4 GB na 4 MB. To ne igra ulogu u primjerima koji su se ovdje testirali.

Prvi skup primjera koji se testirao su sintetski generirani podaci pomoću *python* bilježnice dostupne u repozitoriju projekta. Generirane su različite sekvence veličina od 103 sve do veličine 107. Drugi skup primjera uključivao je sekvencu bakterije *escherichia coli*, točnije sljedeće dvije sekvence ([referentna sekvenca](http://bacteria.ensembl.org/Escherichia_coli_gca_003194125/Info/Index) i [ciljna sekvenca](http://bacteria.ensembl.org/Escherichia_coli_str_k_12_substr_mg1655_gca_000005845/Info/Index)). Rezultati su ostvareni tako što se svaki program pokrenuo 5 puta za svaki primjer te se uzela prosječna vrijednost tih 5 pokretanja. U nastavku su prikazani rezultati testiranja (nažalost, za primjere 106 i 107 originalna implementacija vraća *segmentation fault*) :

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Duljina sekvence | Vrijeme kompresije (ms) (naša implementacija) | Vrijeme kompresije (ms) (originalna implementacija) | Vrijeme dekompresije (ms) (naša implementacija) | Vrijeme dekompresije (ms) (originalna implementacija) | Veličina nakon kompresije (kB) (naša implementacija) | Veličina nakon kompresije (kB) (originalna implementacija) |
| 300 | 7.9114 | 7.9938 | 7.1904 | 5.1584 | 0.304 | 0.321 |
| 3000 | 8.4224 | 8.5328 | 7.9286 | 5.8628 | 1.511 | 1.253 |
| 30000 | 18.7852 | 18.903 | 21.9018 | 16.2914 | 12.763 | 9.677 |
| 300000 | 72.9728 | 95.7856 | 121.8048 | 72.3948 | 125.273 | 89.789 |
| 600 | 7.7298 | 7.7478 | 7.502 | 5.3824 | 0.433 | 0.424 |
| 6000 | 9.06 | 8.4782 | 9.5186 | 6.2274 | 2.7 | 2.181 |
| 60000 | 28.8628 | 25.0906 | 36.5192 | 21.424 | 25.474 | 19.06 |
| 600000 | 162.973 | 168.354 | 198.7546 | 103.7524 | 247.318 | 174.463 |
| 3000000 | 966.5432 | SEG\_FAULT | 879.88 | SEG\_FAULT | 1197.671 | SEG\_FAULT |
| 30000000 | 800074.2534 | SEG\_FAULT | 8039.407 | SEG\_FAULT | 11429.277 | SEG\_FAULT |
|  |  | |  | |  | |
| **Razlika** | ≈7.63% brža | | ≈73.84% sporija | | ≈39.91% veće datoteke | |

Tablica 1 Rezultati testiranja na sintetičkim podacima

Nakon provedenih testova na sintetičkim podacima dobiveni su sljedeći rezultati:

* Naša implementacija kompresije je **≈7.63% brža** od originalne implementacije
  + Ovo nas je pomalo iznenadilo, ali razlika nije tako značajna da bi bila neuvjerljiva.
* Naša implementacije dekompresije je **≈73.84% sporija** od originalne implementacije
  + Razlog je malo „nespretna“ rekonstrukcija sekvence koja se oslanja na popunjavanje stringa znakovima '-' te se onda na mjesta tih znakova naknadno dodaju informacije iz kompresirane datoteke.
* Naša implementacija ima **≈39.91% veće datoteke** od originalne implementacije
  + Ovo uvelike ovisi o načinu spremanja *auxilliary* podataka što ovisi o implementaciji. Mi smo zadovoljni s našim rezultatom, a i također je u okviru dopuštenog prekoračenja.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Vrijeme kompresije (ms) (naša implementacija) | Vrijeme kompresije (ms) (originalna implementacija) | Vrijeme dekompresije (ms) (naša implementacija) | Vrijeme dekompresije (ms) (originalna implementacija) | Veličina nakon kompresije (kB) (naša implementacija) | Veličina nakon kompresije (kB) (originalna implementacija) |
|  | 3503.0516 | 3516.0648 | 781.1466 | 527.1134 | 1271.98 | 889.867 |
|  |  | |  | |  | |
| **Razlika** | ≈0.37% brža | | ≈48.19% sporija | | ≈42.94% veća datoteka | |

Tablica 2 Rezultati testiranja na sekvenci bakterije Escherichia coli

Nakon provedenih testova na sekvenci bakterije *escherichia coli* dobiveni su sljedeći rezultati:

* Naša implementacija kompresije je **≈0.37% brža** od originalne implementacije
  + Zanemariva razlika.
* Naša implementacije dekompresije je **≈48.19% sporija** od orginalne implementacije
  + U skladu sa sintetičkim podacima, doduše malo bolji rezultat.
* Naša implementacija ima **≈42.94% veće datoteke** od originalne implementacije
  + U skladu sa sintetičkim podacima.

Sva mjerenja su dostupna u datoteci *comparison* koja se nalazi u repozitoriju.

Izmjereno korištenje **RAM** memorije bilo je u rasponu od **85-93 Mb** za obje implementacije prilikom pokretanja programa za bakteriju *E-coli*.

# Zaključak

Sekvenciranje genoma vrlo je uzbudljivo i zanimljivo područje u današnjoj znanosti. Međutim, jedan od glavnih problema sekvenciranja je potreba za velikim memorijskim prostorom za pohranjivanje podataka. Prosječan ljudski genom ima otprilike 3 GB što ga čini podosta velikim, te pohranjivanje velikog broja takvih genoma jednostavno nije održivo. Na svu sreću, ljudski genomi su >99% slični i upravo tu činjenicu iskorištava algoritam koji se implementirao u ovome projektu. Algoritam *HiRGC* funkcionira tako da pohrani samo jednu referentnu sekvencu, te pomoću nje rekonstruira bilo koju drugu sličnu sekvencu, a pritom koristi trunku prostora koju bi koristile dvije ili više sekvenci. S obzirom na relativni novitet u području bioinformatike, ovaj projekt se ispostavio dosta zahtjevnim, ali smo na kraju uspjeli postići zadovoljavajući rezultat. Naravno, nismo postigli jednake performanse kao i autori originalnog rada, ali to je bilo i za očekivati. Zadovoljni smo postignutim rezultatima kao i stečenim znanjem.

# Literatura

[1] Liu, Y., Peng, H., Wong, L., Li, J. (2017). *High-speed and high-ratio referential genome compression*. Bioinformatics Vol. 33 Iss 21 pp. 3364-3372. Poveznica: <https://doi.org/10.1093/bioinformatics/btx412>; pristupljeno 11. svibnja 2022.

[2] Čeple, K. (2020). *Referential genome compression.* Fakultet elektrotehnike i računarstva. pristupljeno 11. svibnja 2022.