SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Vlastita implementacija: HRCM algoritam

Leon Hegedić, Ivan Terzić
Voditelj: *Mirjana Domazet-Lošo*

Zagreb, lipanj 2024.

SADRŽAJ

1. Uvod

U području bioinformatike čest je rad s dugačkim nizovima znakova. Neka taj dugi niz znakova predstavlja niz baza A, C, G i T u DNA molekuli. Ta sekvenca baza određuje genetsku informaciju koja je cijelom svojom duljinom bitna [1], tj. ne smije se izostaviti niti jedna baza u zapisu baze. Ako se uzme u obzir da se tim nizom može predstaviti i cijeli genom pojedine vrste koji može biti dug od 50 000 pa sve do preko 100 000 000 000 znakova, odnosno baza (za primjer, genom čovjeka sastoji se od 3 000 000 000 baza). [2] Ako se uzme u obzir da se slova u današnjim računalima pohranjuju s jednim bajtom, odnosno 8 bitova, neki niz baza dug 1 000 000 znakova zauzeo bi 1MB, a 100 000 000 gotovo 100MB, a to nije ni približno dovoljno za pohranu genoma na primjer kralježnjaka. Jasno je da je potreban efikasan način za pohranu tih niza baza zbog samog prijenosa, preuzimanja, ali i pohrane na uređajima. Ovaj se rad bavi upravo takvim algoritmom: HRCM algoritmom. Algoritam radi kompresiju datoteka genoma pomoću referentnog niza baza te pohranjuje, s manjim memorijskim zauzećem, inicijalnu datoteku genoma u drugom obliku sa svim potpunim informacijama. Rad se bazira na proučavanju algoritma, opisu teoretskih pojmova nužnih za razumijevanje algoritma, opisu samog algoritma te dokumentiraju testiranja algoritma.

2. Opis algoritma [3]

HRCM algoritam sastoji se od dvije glavne faze: kompresija i dekompresija. U fazi kompresije se sekvence spremaju u datoteke sažetog oblika (manjeg memorijskog zauzeća) koje sadržavaju potpunu informaciju o sekvenci, dakle ne gube se informacije ni o malim slovima, ni specijalnim znakovima niti „X" znakovima. U fazi dekompresije se ti pohranjeni podatci vraćaju u svoj izvorni oblik, kako je primljen na ulazu u algoritam.

2.1. Kompresija

Algoritam na svoj ulaz u fazi kompresije prima referentnu datoteku genoma (referentnu sekvencu) i jednu ili više sekvenci koje se trebaju komprimirati, sve datoteke u FASTA formatu. Sekvence za komprimiranje komprimiraju se pomoću referentnog genoma. Sama faza kompresije algoritma dijeli se u 3 glavna koraka: izvlačenje informacija sekvenci, pronalazak podudarnih informacija u nizovima te kodiranje informacija o sekvencama.

2.2. Izvlačenje informacija o sekvencama

U prvom koraku kompresije, na ulazu samog algoritma, nalaze se datoteke sekvenci namijenjene za kompresiju te referentna sekvenca u FASTA formatu. To znači da je u prvom redu identifikator genoma te u svakom redu ispod niz znakova fiksne duljine za svaki red koji se sastoji od osnovnih baza, malih slova, koje označavaju nesigurnost kod očitanja, slova N i slova X ili procijepa -. Za svaku se sekvencu pohranjuju sljedeće informacije: identifikator, sekvenca osnovnih baza (A, C, G i T), informacije o pozicijama malih slova, informacije o slovima N, informacije o ostalim znakovima te broj koji označava duljinu retka. Za referentnu sekvencu pohranjuju se samo sekvenca osnovnih baza i informacije o malim slovima, ona ne mora pamtiti informacije o ostalim slovima, itd. Sam proces obavlja se na sljedeći način: obzirom na FASTA format,

prva linija u datoteci je tzv. identifikator, on se izvlači i sprema kao identifikator pojedine datoteke. Nakon toga se, za prvu liniju koja sadrži niz baza, gleda njena duljina te pohranjuje kao duljina retka. Nakon toga se u nizu baza redom bilježe pozicije malih slova te potom i N slova i posebnih slova. Obzirom da je očekivano da će algoritam raditi s izuzetno dugim nizovima znakova, pohrana informacija o malim i N slovima se radi pomoću pohrane relativnog indeksa (udaljenosti) od prošle pojave malog ili N slova (naravno, prošla pojava malog slova se gleda od prošle pojave malog slova, itd.) i pohrane duljine tog niza znakova. Tako se izbjegava pohrana velikih brojeva kao pozicija, što bi bilo očekivano za slučaj kad bi se koristili apsolutni indeksi u nizu znakova. Nakon pohrane tih informacija, znakovi se spremaju u jedan niz znakova, brišu se posebni znakovi i N znakovi te se dobije čist niz znakova A, C, G ili T, odnosno baza. Za referentu sekvencu postupak je analogan, osim što se spremaju samo podatci potrebni za taj niz, dakle niz baza i podaci o malim slovima. Iako se informacije ne spremaju, također se brišu N znakovi i specijalni znakovi. Ovako zapisani nizovi znakova ulaze u drugu fazu kompresije: pronalazak podudarnih informacija.

2.3. Pronalazak podudarnih informacija

U drugoj fazi kompresije, pronalasku podudarnih informacija, kao ulaz primamo nizove znakova zapisane kako je opisano u prošlom odlomku, točno 1 referentan niz znakova i barem jednu sekvencu koja je namijenjena za kompresiju. U ovoj fazi cilj je pronaći podudarnosti između referentne sekvence i sekvenci koje se komprimiraju kako bi se što više smanjilo memorijsko zauzeće zapisa. Ova se faza sastoji od pronalaska podudaranja u nizu baza te u malim slovima. Ukoliko postoji više od jedne sekvence koje se trebaju komprimirati, algoritam koristi i tzv. podudaranje drugog reda (engl. second-level matching). Prvo se u ovoj fazi radi pronalazak podudaranja u sekvencama pojedine sekvence za kompresiju i referentne sekvence. Za to, potrebno je napraviti hash tablicu pojedinih k-mera u referentnoj sekvenci. Ovaj postupak funkcionira kao posmični prozor koji ide nad nizom te onda izračunava sažetak, odnosno hash vrijednost za pojedini k-mer. A, C, G i T se kodiraju kao 0, 1, 2 i 3, time se svako slovo može kodirati s 2 bita jer su 00, 01, 10, 11 vrijednosti koje predstavljaju te brojeve u oblicima bitova. Izračunava se hash vrijednost za prvi k-mer u prozoru na način da se gleda vrijednost k-mera tako da su vrijednosti znaka koji je prije na značajnijem mjestu, to se ostvaruje posmakom za dva bita ulijevo kad se dodaju znakovi. Kreiranje hash tablica odvija se pomoću dva polja: H i L, koje se prije početka kreacije tablice moraju inicijalizirati. H tablica je veličine mogućih hash vrijednosti što

ovisi o duljini k-mera, a L tablica veličine svih mogućih indeksa k-mera u sekvenci, tj. obadvije vrijednosti ovise o duljini k-mera, ali se računaju po različitim formulama. Nakon izračuna sažetka k-mera, recimo da je njegova vrijednost v, njegov indeks se pohranjuje u polje H pod indeksom v, a originalna vrijednost koja je bila u H se sprema u L na indeksu i. Time efektivno polje H drži zadnji indeks pojave k-mera s određenim sažetkom, a L „pokazivač“ na prethodni indeks čime to polje efektivno postaje lista s pokazivačima na prethodni indeks k-mera s istom vrijednosti sažetka. Time se na efektivan način kreira hash struktura koja će znatno ubrzati proces pronalaženja najduljih podnizova koji se podudaraju među dvije sekvence znakova. U procesu pronalaženja najduljih podnizova koji se podudaraju, za svaki se k-mer neke sekvence za kompresiju računa sažetak na isti način, s posmakom, kao i kod izrade hash tablice. Nakon izračuna sažetka se gleda u tablicu H, ako je zapis ekvivalentan onom kod inicijalizacije, u ovom slučaju -1, očito je po procesu izrade hash tablice da onda ne postoji u referentnoj sekvenci k-mer, odnosno podniz, koji se podudara s danim podnizom. Stoga se taj dio sekvence, taj znak, sprema u informacije o znakovima koji se ne podudaraju. U suprotnom, ako postoji indeks u tablici H, postoji bar jedan k-mer koji se podudara između dva niza i onda se ulazi u proces pronalaska najduljeg podudaranja. Tijekom ovog se procesa prolazi kroz sve relevantne k-mere pomoću polja L te se gleda za koju pojavu k-mera se može naći najdulji zajednički niz između referentne sekvence i one koja se komprimira. Informacije se spremaju kao trojka: pozicija (opet relativna u odnosu na prošlu poziciju), duljina tog najvećeg pronađenog podudaranja i niz znakova koji se ne podudaraju. Kada algoritam za kompresiju primi više datoteka, automatski prelazi u način batch kompresije. Ovdje se rezultati prve razine podudaranja koriste za daljnje podudaranje u drugoj razini, kako bi se dodatno smanjili zahtjevi na memorijsko zauzeće korištenjem više sekvenci za pronalazak podudaranja. Podudaranje drugog reda, kao i ono prvog reda, također koristi hash podudaranje, ali umjesto načina koji je implementiran u prvom redu (izvlačenje sažetka iz kodiranih baza) koristi složenije entitete koji uključuju poziciju, duljinu i nepodudarnosti, odnosno izlaz iz podudaranja prve razine, kompresija se izvodi po nizu struktura podataka za podudarnosti iz podudaranja prve razine. Prije tog mora se provesti pretraga najduljih zajedničkih nizova u svim predanim sekvencama za kompresiju. Kako bi se limitirala potrošnja memorije, može se postaviti postotak druge razine podudaranja, p, što znači da se p% sekvenci koje treba komprimirati koristi kao referentne sekvence druge razine. Prvo se stvara hash tablica podataka o podudaranju, outputa iz first-level matchinga. U kreaciju sažetka uključuju se sva tri elementa pojedinog objekta toga izlaza, dakle pozicija, duljina i svaku nepodudarnu bazu. Veliki prosti broj se koristi u množenju tih podataka

za sažetak. U slučaju da postoji konflikt, kao i u podudaranju prvog reda, koristi se odvojeno ulančavanje s H i L listama. Kod faze pretrage, nakon kreacije hash tablice, se na identičan način kao kod podudaranja prvog reda, naravno koristeći ispravnu hash funkciju, računaju saželi i traži se identičan zapis u hash tablici. Međutim, identična hash vrijednost ovdje ne znači i identičan entitet jer preslikavanje između entiteta i hash vrijednosti u ovom slučaju nije jednoznačno za razliku od prve razine. Kada se pronađe identična hash vrijednost, potrebno je provjeriti jesu li entiteti identični na razini samog entiteta, ne samo hash vrijednosti. Ako jesu isti, pretraga se nastavlja kao kod prve razine uzastopno dok se ne pronađe nepodudarni entitet, a zatim se bilježe ID sekvence, pozicija i duljina podudarnog segmenta. Nakon pretrage svih referentnih sekvenci, najduži podudarni podniz uzima se za zamjenu i sprema se, ponovo, trojka ID sekvence, pozicija, duljina. Ako se identičan entitet ne pronađe nakon pretrage svih referentnih sekvenci, segment se sprema kao nepodudaran segment. Neovisno o podudaranjima prvog i drugog reda odvija se treća glavna faza drugog dijela kompresije: podudaranje informacija o malim slovima. Na ovaj se način jednostavno smanjuje prostor za pohranu podataka na način da, ako za sekvencu za određen zapis o malim slovima taj zapis već postoji u zapisu o malim slovima referentne sekvence, pohranjuje se lokacija tog zapisa, inače ako ne postoji ostaje inicijalan zapis.

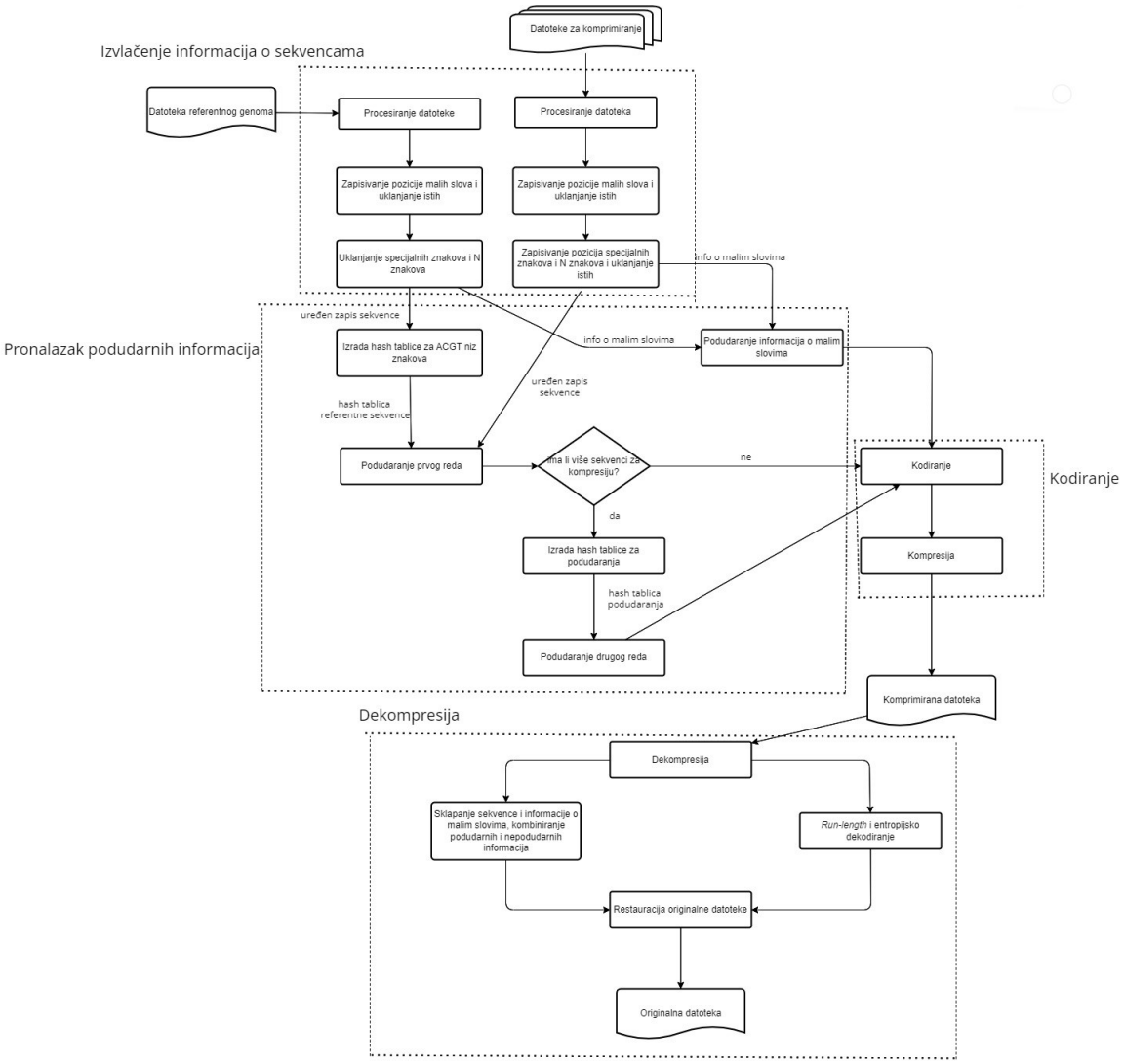
2.4. Kodiranje

S prethodno opisanim podatcima kao izlazima iz faza podudaranja, bilo nizova, bilo malih slova, ulazi se u fazu kodiranja, same kompresije. Duljine linija i identifikatori se kodiraju s tzv. run-length encodingom, specijalni znakovi s entropijskim kodiranje, a pozicije se kodiraju inkrementalnim kodiranje. Sve se tr informacije kodiraju i komprimiraju u datoteku arhive.

2.5. Dekompresija

Dekompresija je jednostavniji postupak od kompresije te je u ovom slučaju samo reverzan postupak. Arhiva se prvo dekomprimira, te se s algoritmima opisanim u kodiranju dekodiraju pojedine informacije. Informacije o sekvencama i malim slovima se na reverzan način također rekonstruiraju u originalne vrijednosti koristeći i podudarne i nepodudarne informacije te se onda na taj način vraćaju u originalne datoteke genoma. Ovdje se kod obrnute faze podudaranja ne moraju koristiti hash vrijednosti jer su one

u fazi kompresije služile samo za pronalazak bržeg podudaranja, te tablice ovdje nisu bitne.



Slika 2.1: Odabir lokacije za niz AAC - prikaz po uzoru na [2]

3. Primjer izvođenja

Način rada implementiranog algoritma najbolje se vidi provođenjem na jednostavnom primjeru. Neka postoje 3 datoteke: Refseq, T1 i T2 gdje je Refseq datoteka referentne sekvence, a T1 i T2 datoteke za kompresiju. Neka je sadržaj datoteka sljedeći:

- Refseq:
  Refseq
  ACGAGTTCCCTTCCggNnNnnnXAC
  CCCAAAGGGAAAtttTaTccCtGAg

- T1:
  T1
  ACGAGGTCCCTTTCggAnNnnnXAC
  CCCAAAGTGAAAtttTATccCtGAg

- T2:
  T2
  ACGAGGTCCCTTTCggAnNnnnXAC
  CACAAAGTGAAAtttTATCCCtGAg

Datoteke ulaze u proces obrade gdje se one čitaju te obrađuju. Nakon toga slijedi zapisivanje i obrada pozicija malih slova te obrada ostalih znakova. Sukladno opisu algoritma, rezultat je sljedeći:

• Referentna sekvenca:
  - Sekvenca: ACGAGTTCCCTTCCGGACCCCAAAGGGAAATTTTATCCCT-GAG
  - Informacije o malim slovima: (14, 2), (1, 1), (1, 3), (15, 3), (1, 1), (1, 2), (1, 1), (2, 1)

• T1:

  - Identifikator: T1
  - Širina retka: 25
  - Sekvenca: ACGAGGTCCCTTTCGGAACCCCAAAGTGAAATTTTATCCC-TGAG
  - Informacije o malim slovima: (14, 2), (1, 1), (1, 3), (15, 3), (3, 2), (1, 1), (2, 1)
  - Informacije o N: (17, 5)
  - Informacije o posebnim znakovima: (22, X)
- T2:
  - Identifikator: T2
  - Širina retka: 25
  - Sekvenca: ACGAGGTCCCTTTCGGAACCCCAAAGTGAAATTTTATCCC-TGAG
  - Informacije o malim slovima: (14, 2), (1, 1), (1, 3), (15, 3), (6, 1), (2, 1)
  - Informacije o N: (17, 5)
  - Informacije o posebnim znakovima: (22, X)

Na ovom se primjeru vide sve navedene tvrdnje vezano za ovaj dio. Referentna sekvenca sadrži manje informacija, obadvije varijante ekstrakcije sadrže sekvencu i informacije o malim slovima dok identifikator, info o N znakovima i posebnim znakovima imaju samo sekvence za kompresiju. Također je očito da se ti znakovi i brišu iz sekvence te da se radi o relativnim indeksima. Npr. (14, 2) označava da se prvi niz malih slova duljine dva mala slova nalazi 14 znakova od početka, sljedeći par (1, 1) označava da se onda jedno mjesto od zadnje pojave malog slova (14+2) nalazi jedno malo slovo itd. Ovako obrađeni podatci idu u fazu pronalaska najduljeg podudaranja, ali kako bi se to moglo provesti, prvo se mora napraviti hash tablica nad referentnom sekvencom. U ovom su primjeru k-meri veličine 2, dakle imamo dva slova koja kodiramo sa po dva bita, kako je opisano u prošlom poglavlju. Sukladno tome, imamo 4 moguća bita i 16 mogućih hash vrijednosti, stoga će polje H biti veličine 16 kako bi moglo zapisati zadnju pojavu svake hash vrijendosti pojedinog k-mera. Recimo, prvi k-mer bio bi AC, drugi CG, itd... Te bi se redom računale hash vrijednosti i zapisivale u H, a prethodna vrijednost od H išla bi u listu L kao pokazivač na prethodni indeks k-mera tog hasha. Duljina polja L je broj k-mera, dakle duljinaSekvence – duljinaKmera + 1. Vrijednosti polja H i L zapisane su ispod. Za primjer, uzmimo u obzir hash na indeksu 15, s vrijednosti 32. Ta nam

informacija govori da je zadnja pojava k-mera čiji je sažetak 15 na indeksu k-mera 32. Ako se prati lista L, isti se taj k-mer nalazi na indeksima 31, 30, 10, i 5. Također se može vidjeti da jedino za k-mer koji bi imao hash vrijednost 6 ne postoji zapis u H polju, time jednostavno vidimo da taj k-mer nije pronađen u ulaznoj sekvenci. Tako je efikasno stvorena struktura podataka hash tablice za podudaranje prvog reda.

H tablica	L tablica
H[0] = 28	L[0] = -1
H[1] = 20	L[1] = -1
H[2] = 40	L[2] = -1
H[3] = 33	L[3] = -1
H[4] = 16	L[4] = -1
H[5] = 37	L[5] = -1
H[6] = -1	L[6] = -1
H[7] = 35	L[7] = -1
H[8] = 41	L[8] = 7
H[9] = 13	L[9] = -1
H[10] = 25	L[10] = 5
H[11] = 39	L[11] = 6
H[12] = 34	L[12] = 8
H[13] = 38	L[13] = 1
H[14] = 4	L[14] = -1
H[15] = 32	L[15] = 2
	L[16] = 0
	L[17] = 12
	L[18] = 17
	L[19] = 18
	L[20] = -1
	L[21] = -1
	L[22] = 21
	L[23] = 3
	L[24] = 14
	L[25] = 24
	L[26] = 15
	L[27] = 22
	L[28] = 27
	L[29] = -1
	L[30] = 10
	L[31] = 30
	L[32] = 31
	L[33] = -1

H tablica	L tablica
	L[33] = -1
	L[34] = 29
	L[35] = 11
	L[36] = 19
	L[37] = 36
	L[38] = 9
	L[39] = -1
	L[40] = 26
	L[41] = 23

U nastavku algoritma, u podudaranju prvog reda sudjeluju i T1 i T2, svaki zasebno. Provodi se izračun vrijednosti sažetka svakog k-mera, kao i kod izrade hash tablic te se onda gleda postoji li zapis za taj k-mer u hash tablici. Ako ne, prvi se znak sprema u podatke o nepodudarnim informacijama, ako da ulazi se u proces pronalaska najduljeg podudaranja. Hash tablica izrađena u prethodnom koraku ovdje služi samo za brži pronalazak podudarnog k-mera bez da se pretražuje znak po znak. Za T1 niz izlaza iz podudaranja prvog reda je: (0, 5,), (1, 6, G), (1, 4, T), (0, 8, A), (1, 17, T), A za T2: (0, 5,), (1, 6, G), (1, 4, T), (2, 3, A), (-1, 4, C), (1, 17, T). Za podsjetnik, kako radi ova faza, pogledajmo sekvence Refseq i T1: ACGAGTTCCCTTCCGGACCCCAAAGGGAAATTTTATCCCTGAG ACGAGGTCCCTTTCGGAACCCCAAAGTGAAATTTTATCCCTGAG Kod prvog podudaranja se računa sažetak za k-mer „AC“. Izračunom se dolazi do Hash vrijednosti 4 i toga da je zadnji k-mer indeksa 16 za ovaj sažetak iz liste H. Prati se lista L i uspoređuje se za svaku pojavu tog k-mera koliko je dug niz podudaranja. Po L, to su k-meri 16 i 0. Obzirom da se na indeksu 16 podudara samo taj k-mer, očito je da će onaj na indeksu 0 dati dulje podudaranje od 5 znakova. Tako se radi za svaki k-mer te je tu od značajne vrijednosti hash tablica koja omogućava brzi pristup svim pozicijama (indeksima) k-mera koji imaju isti sažetak. U sljedećem koraku, algoritam nije mogao pronaći podudaranje za CG te je onda zapisan u nepodudarne informacije i priložen uz sljedeću trojku. Algoritam se nastavlja dalje tom logikom do kraja. Pošto dolazi do kompresije više datoteka nastavlja se na podudaranje drugog reda. T1 i T2, s fokusom na specifične segmente i obradu umetanja, brisanja i nepodudarnosti. Pogledaju li se izlazi iz podudaranja prvog reda, očito je da imamo podudaranja između izlaza za T1 i T2. Sukladno tome, izlaz iz podudaranja drugog reda je trojka (0,0,3) što znači da se T2 referencira na nultu sekvencu (T1), da s odmakom 0 u duljini

3 strukture podataka podudaranja prvog reda ima podudaranje. Vidimo da se i zadnji zapisi podudaraju, no za *second-level matching* potrebno je da se podudara više od jednog zapisa za redom, stoga postoji samo jedan zapis u podudaranju drugog reda za ovaj jednostavan primjer, u kompleksinijm primjerima naravno dolazi do više podudaranja. Nakon ove faze, slijedi podudaranje informacija o malim slovima, kao podsjetnik priložene su informacije o malim slovima za Refseq i T1: (14, 2), (1, 1), (1, 3), (15, 3), (1, 1), (1, 2), (1, 1), (2, 1) (14, 2), (1, 1), (1, 3), (7, 1), (7, 3), (1, 1), (4, 1), (2, 1). U ovoj je fazi cilj pronaći podudarnost informacija malih slova. Koristi se indeksiranje od 1 za redni broj te 0 ako ne postoji zapis o Za T1, izlaz je sljedeći: Mismatched lowercase information: (7, 1), (7, 3), (4, 1), (2, 1) Matched lowercase information: 1, 2, 5, 7, 0, 0, 7, 0, 0 Vidimo za koje je indekse pronađen ekvivalentan zapis u referentnoj sekvenci te za koje nije, ti za koje nije ostaju u obliku kakvom jesu. Za T2 izlaz je: Mismatched lowercase information: (7, 1), (7, 3), (4, 1), (2, 1) Matched lowercase information: 1, 2, 3, 0, 0, 5, 0, 0. Nakon svih tih koraka, postupcima u opisanim u prethodnom poglavlju se datoteke kodiraju i pohranjuju u svoj komprimirani oblik koji može varirati zavisno o implementaciji. Postupak dekompresije svodi se na reverzno provođenje opisanih postupaka.

4. Testiranje i analiza

Sva testiranja su provedena na računalu s Intel i7-9750H procesorom, koji ima 6 jezgri i 12 dretvi, te 16 GB RAM memorije. Operacijski sustav instaliran na tom računalu je Ubuntu 24.04 LTS.

Kako bismo temeljito ispitali naš kod, definirali smo sljedećih 9 slučajeva na kojima smo testirali vrijeme potrebno programu da se izvede, kao i najveće memorijsko zauzeće tijekom izvođenja:

1. Kompresija jedne male sekvence od dvadesetak parova baza (dozvoljene baze su A, C, G, T).
2. Kompresija (kao u prethodnom primjeru), ali s 5 takvih manjih parova baza.
3. Kompresija kratke sekvence s N-ovima, specijalnim znakovima i malim slovima.
4. Slično kao u prethodnom primjeru, ali kompresija 3 takve male sekvence.
5. Kompresija veće sekvence (oko 1000 znakova), ali samo s redovnim bazama i malim slovima.
6. Kao i u prethodnom primjeru, ali 3 takve sekvence.
7. Kompresija veće sekvence (malo više od 1000 znakova) sa svim dozvoljenim specijalnim znakovima, N-ovima i malim slovima.
8. Kompresija 3 sekvence slične kao u prethodnom zadatku.
9. Kompresija jedne veće sekvence genoma Escherichia coli (veličina 4.7 MB).

Važno je navesti još neke pojedinosti prije pokazivanja rezultata. Vrijeme izvođenja programa mjerilo se pomoću ugrađenih funkcija za mjerenje vremena unutar C++. Za mjerenje memorije koristio se vanjski alat Valgrind. Pokretanje programa unutar alata Valgrind izaziva veliko usporavanje rada programa, stoga smo zauzeće memorije i vrijeme potrebno za izvođenje programa mjerili nezavisno. Također, Valgrind kao

alat mjeri memorijsko zauzeće na gomili (eng. heap) pa smo to odlučili prikazati u tablicama.

Slučajevi	**Originalna [ms]**	**Naša [ms]**
1.	505.137	768.273
2.	514.811	772.047
3.	509.262	768.705
4.	520.915	774.257
5.	516.102	771.032
6.	511.123	772.627
7.	509.690	766.818
8.	515.515	776.978
9.	-	2666.58

Tablica 4.1: Usporedba vremena izvođenja kompresije

Slučajevi	**Originalna [GB]**	**Naša [GB]**
1.	2.582	1.016
2.	2.582	1.016
3.	2.582	1.016
4.	2.582	1.016
5.	2.582	1.016
6.	2.582	1.016
7.	2.582	1.016
8.	2.582	1.016
9.	-	1.024

Tablica 4.2: Usporedba zauzeća memorije tijekom kompresije

Kao što je vidljivo iz rezultata u tablici 4.1, naša implementacija je u prosjeku manje od dvostruko sporija od originalne implementacije. Ovaj rezultat je ohrabrujući kada se uzme u obzir da smo većinom radili s dinamičkom memorijom. Također, u tablici 4.2 se vidi prava prednost dinamički alocirane memorije, manje zauzeće gomile. Vidimo da jedino veće zauzeće dolazi kada radimo s dužim sekvencama. (Napomena: u originalnom kodu danom uz rad, dobivali smo segmentation error tijekom pokretanja 9. slučaja. Zato smo stavili crticu kao nedostatak podataka.)

Sljedeće smo htjeli pokazati prednost ovog algoritma kada radimo s većim sekvencama. To smo odlučili pokazati kompresijom 1, 3, 10 i 22 sekvence Escherichia coli s veličinama većim od 4, ali manjim od 6 MB. Uspoređivali smo veličinu originalnih datoteka, njihovu direktnu ZIP kompresiju, te kompresiju pomoću HRCM algoritma. Rezultati su vidljivi u tablici dolje:

Br. datoteka	**Veličina [MB]**	**ZIP [MB]**	**HRCM [MB]**
1	4.7	1.4	1.6
3	15.3	4.6	3.4
10	54.1	16.4	5.0
22	120.4	36.5	8.3

Tablica 4.3: Usporedba zauzeća memorije

Kao što je prikazano u tablici 4.3, kompresija nije mnogo korisna kada radimo kompresiju jedne datoteke u usporedbi s ZIP kompresijom. Međutim, kompresijom više takvih dugačkih sekvenci drastično smanjujemo memorijsko zauzeće, čak i u usporedbi s direktnim zipanjem, postižući nekoliko puta bolju kompresiju.

5. Zaključak

Unatoč tome što je naša implementacija HRCM algoritma u prosjeku manje od dvostruko sporija od originalne implementacije, postižemo ohrabrujuće rezultate. Ovo je značajno jer smo većinom radili s dinamičkom memorijom, što može utjecati na brzinu izvođenja algoritma. Iako postoje prostori za poboljšanje u performansama, naši rezultati sugeriraju da je HRCM algoritam i dalje konkurentan izbor za kompresiju sekvenci, a posebno je učinkovit za kompresiju više sekvenci odjednom. Dodatno istraživanje i optimizacija mogli bi dodatno unaprijediti performanse algoritma u budućnosti.

6. Literatura

[1] F. Nositelji kolegija Bioinformatika 1, »Uvodno predavanje,« FER, [Mrežno]. Available: https://www.fer.unizg.hr/_download/repository/Bioinformatika_1%20-%20Uvodno_2023_2024.pdf . [Pokušaj pristupa 23. May 2024.].

[2] N. k. Bioinformatika, »Dinamičko programiranje,« FER, [Mrežno]. Available: https://www.fer.unizg.hr/_download/repository/Bioinformatika_%201_2-3_predavanje_Dinami%C4%8Dko_programiranje.pdf. [Pokušaj pristupa 23. May 2024.].

[3] Y. J. K. L. S. L. J. H. R. W. Haichang Yao, »HRCM: An Efficient Hybrid Referential Compression Method for Genomic Big Data,« 19 November 2019. [Mrežno]. Available: https://www.hindawi.com/journals/bmri/2019/3108950/. [Pokušaj pristupa 23. May 2024.].