SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

HRCM algoritam

Paulo Erak, Lara Grgurić
Voditelj: *Mirjana Domazet-Lošo*

Zagreb, svibanj 2023.

SADRŽAJ

1. Uvod **1**

2. Algoritam **2**

2.1. Vađenje podataka 2

2.2. Podudaranje podataka 4

2.2.1. Podudaranje prve razine 4

2.2.2. Podudaranje druge razine 5

2.2.3. Podudaranje malih znakova 7

2.3. Kodiranje podataka 8

2.4. Dekompresija 8

3. Implementacija **10**

3.1. Vađenje podataka 10

3.2. Podudaranje podataka 11

3.3. Kodiranje podataka 11

3.4. Dekompresija 11

4. Rezultati testova **12**

4.1. Sintetičke sekvence 12

4.2. *Escherichia coli* 13

5. Zaključak **14**

1. Uvod

Jedan od bitnih problema u području bioinformatike je sažimanje i dekompresija genomskih podataka. Ovaj problem je aktualan jer je spremanje podataka skupo, a s razvojem tehnologija povećava se važnost i količina podataka [**?**]. Jedna od metoda kompresije velikih genomskih podataka je metoda hibridne referentne kompresije koju smo odlučili implementirati. Cilj projekta je napraviti vlastitu implementaciju metode hibridne referentne kompresije po uzoru na rad [**?**] te testirati na sintetski stvorenim podacima i podacima korištenima u sklopu spomenutog rada.

2. Algoritam

Algoritam koji implementiramo opisan je u radu [**?**] te ćemo dati kratki osvrt na njega. Metoda hibridne referentne kompresije podijeljena je na procese kompresije i dekompresije. Proces kompresije podijeljen je na tri dijela ovim redoslijedom izvođenja: vađenje, podudaranje i kodiranje podataka. Ulaz u proces kompresije su jedna datoteka referentnog genoma te jedna ili više datoteka genoma koje treba sažeti.

2.1. Vađenje podataka

Ulazi za ovaj dio kompresije su datoteke koje sadrže genome. Vađenje podataka sastoji se od nekoliko koraka, međutim koraci potrebni za vađenje podataka iz referentnog genoma su samo oni koji spremaju informaciju o malim znakovima i vraćaju ACGT sekvencu. Koraci redom su:

(i) Izvođenje prvog reda kao identifikatora sekvence

(ii) Zapis duljine pročitanog reda

(iii) Pretvorba malih u velika slova i bilježenje pozicija i duljina dijelova genoma zabilježenih malim slovima

(iv) Spremanje informacija o pozicijama i duljinama segmenata sačinjenih od N znakova te informacija o pozicijama i duljinama segmenata sačinjenih od specijalnih znakova

(v) Izbacivanje svih znakova iz sekvence osim ACGT znakova

Primjer

Dobijemo dvije FASTA datoteke koje sadrže sljedeće sekvence od kojih je prva referentna, a druga sekvenca koju želimo sažeti:

>Xchr1.fa
AGCTGGGCCCTTaaggNNNnnnXXX
TTTCCCGGGAAAaaaTTTccctttg
>Ychr1.fa
AGCTGGGCCCTTaaggtttnnnXXX
TTTCCCGGGNNNaaaTTTccctttg

Program učitava sekvencu X kao referentnu te krene s obradom kako bi se podaci dobiveni obradom koristili tijekom podudaranja i enkripcije. Kod reference je jedino bitno uzeti id, dobiti pozicije i duljine nizova malih znakova i čisti ACGT-niz. Njih spremamo u različite spremnike i čuvamo za daljnju obradu. U ovom primjeru dobivamo:

Id: Xchr1.fa
Čista sekvenca ACGT:
AGCTGGGCCCTTAAGGTTTCCCGGGAAAAAATTTCCCTTTG
Pozicije početka nizova malih znakova:
12, 3, 15, 3
Duljine nizova malih znakova:
4, 3, 3, 7

Program slično radi i za sekvence koje želimo sažeti, ali za njih još program vadi podatke o poziciji i duljini znakova N i ostalih znakova. Oni se neće koristiti prilikom podudaranja ali su ključni za sastavljanje početne sekvence prilikom dekompresije. Prolaskom sekvence koju želimo sažeti kroz funkciju za vađenje informacija dobivamo sljedeće:

Id: Ychr1.fa
Čista sekvenca ACGT:
TGGGCCCTTAAGGTTTTTTCCCGGGAAATTTCCCTTTGG
Pozicije početka nizova malih znakova:
12, 15, 3
Duljine nizova malih znakova:
10, 3, 7
Pozicije početka nizova N znakova:
19, 12
Duljine nizova N znakova:
3, 3
Pozicije ostalih znakova:
22, 0, 0
Ostali znakovi:
X, X, X
Duljina linija: 25

Sekvence ACGT-a i informacije o malim slovima idu dalje na podudaranje, a ostale informacije se pamte za ponovno sastavljanje sekvenci.

2.2. Podudaranje podataka

Proces podudaranja podataka obavlja se kroz dva dijela: podudaranje ACGT sekvenci i podudaranje informacija o malim znakovima. Podudaranje ACGT sekvenci temelji se na podudaranju k-mera sekvence koja se sažima s k-merima referentne sekvence te se izvodi na dvije razine.

2.2.1. Podudaranje prve razine

Za podudaranje prve razine potrebni su referentna sekvenca i sekvenca koju želimo sažeti. Prvi korak izrada je hash tablice na temelju k-mera referentne sekvence. Pri tome koristimo dvije podatkovne strukture: H i L. U strukturu H upisujemo vrijednosti oblika: $H[vrijednost_i] = i$. U slučaju pojave k-mera iste hash vrijednosti, vrijednost zapisana u $H[vrijednost_i]$ zapisuje se na poziciju trenutačnog indeksa u strukturu L. Znači da su nam za ostvarenje ovog podudaranja potrebne dvije formule: $L[i] = H[vrijednost_i]$ i $H[vrijednost_i] = i$. Na taj način stvaramo lanac indeksa pozicija k-mera identične hash vrijednosti. Drugi je korak pretraga sekvence koju

želimo sažeti za najduljim podnizom koji se poklapa s referentnom sekvencom. To izvodimo izračunavanjem hash vrijednosti k-mera te pretraživanjem lanaca stvorenih strukturama H i L s ciljem pronalaska najduljeg niza uzastopnih vrijednosti indeksa tih k-mera u referentnoj sekvenci. Pri tom pretraživanju dok se pronalaze podudaranja, bilježi se duljina niza, a pri pronalasku nepodudaranja, bilježi se jedan ili više znakova nepodudaranja. Time dobivamo tuple vrijednost (pozicija, duljina) koji služi kao zamjena za pronađeni segment sekvence.

Primjer

Iz prethodnog koraka dobivene su referentna ACGT sekvenca i ACGT sekvence koje trebamo sažeti. Za ovaj primjer odredit ćemo k=3. Prvo stvorimo hash tablicu od svih podnizova duljine k referentne sekvence. Nakon toga prolazimo kroz sekvencu koju trebamo sažeti te tražimo najdulji podniz koji je također podniz referentne sekvence. Nakon što pronađemo najdulji podniz, zapišemo jedan ili više znakova koji slijede iza spomenutog podniza i ne podudaraju se s referentnom sekvencom. Opisanim postupkom dobivamo sljedeće tupleove:

Prva sekvenca za sažimanje:
TGGGCCCTTAAGGTTTTTTCCCGGGAAATTTCCCTTTGG
Druga sekvenca za sažimanje:
TGGGCCCTTAAGGTTTTTTCCCGGGAAATTTCCCCTTGG
Informacije o podudaranju prve razine za prvu sekvencu za sažimanje:
[3, 16, T], [17, 11, T], [32, 9, G]
Informacije o podudaranju prve razine za drugu sekvencu za sažimanje:
[3, 16, T], [17, 11, T], [32, 6, C], [26, 3, G]

2.2.2. Podudaranje druge razine

Podudaranje druge razine kompleksnije je od podudaranja prve razine zbog više razloga. Umjesto korištenja referentne sekvence, koriste se isključivo rezultati podudaranja prve razine. Kako bi se to ostvarilo, potrebno je odabrati postotak svih sekvenci za sažimanje od kojih će svaka imati ulogu referentne sekvence za sve sekvence koje slijede iza nje. Postupak također započinje izgradnjom hash tablice, ali u ovom je koraku izračun hash vrijednosti kompleksniji jer u obzir uzima poziciju, duljinu i hash

vrijednost znaka nepodudaranja. Podudaranje hash vrijednosti ne znači nužno da su ta dva tuplea identična, što znači da je potrebna usporedba svakog elementa kako bi se potvrdilo podudaranje. Za svaku sekvencu potrebno je usporediti podudaranja sa svim referentnim sekvencama te odabrati najdulje podudaranje. U slučaju podudaranja, zapisuje se tuple (id sekvence, pozicija i duljina), pri čemu vrijednost id sekvence označava indeks referentne sekvence. U slučaju nepodudaranja, tuple kojemu ne možemo pronaći podudarajući tuple u referentnoj sekvenci prepisujemo.

Primjer

U podudaranju druge razine kao referentne sekvence uzimamo rezultate podudaranja prve razine koje ulaze u definirani postotak referentnih sekvenci. Za svaku referentnu sekvencu potrebno je napraviti hash tablicu koristeći hash vrijednosti izračunate iz vrijednosti pozicije, duljine i znakova nepodudaranja za svaki tuple u rezultatu podudaranja. Sve ostale rezultate prve razine podudaranja uspoređujemo s prethodećim rezultatima referentnih sekvenci i tražimo najdulji podniz. U primjeru prikazat ćemo algoritam na tri sekvence. Rezultate zapisujemo u tupleovima (identifikator sekvence, početna pozicija, duljina podniza). Pretpostavimo da su ovo rezultati podudaranja prve razine:

Informacije o podudaranju prve razine za prvu sekvencu za sažimanje:
[3, 16, T], [17, 11, T], [32, 9, G]
Informacije o podudaranju prve razine za drugu sekvencu za sažimanje:
[3, 16, T], [17, 11, T], [32, 6, C], [37, 3, G]
Informacije o podudaranju prve razine za treću sekvencu za sažimanje:
[3, 16, G], [17, 11, T], [32, 6, C], [37, 3, G]

Prvo rezultate podudaranja prve razine za drugu sekvencu uspoređujemo s rezultatima podudaranja prve razine za prvu sekvencu te zapišemo rezultate podudaranja. Nepodudarajuće tupleove, u ovom primjeru na indeksima 2 i 3, prepisujemo.

Informacije o podudaranju druge razine za drugu sekvencu za sažimanje:
[0, 0, 2], [32, 6, C], [37, 3, G]

Nakon toga rezultate podudaranja prve razine za treću sekvencu uspoređujemo s rezultatima podudaranja prve razine za prvu i drugu sekvencu te tražimo najdulji podniz. U ovom primjeru nepodudarajući tuple za treću sekvencu je na indeksu 0, zbog čega ga prepisujemo.

Informacije o podudaranju prve razine za treću sekvencu za sažimanje:
[3, 16, G], [1, 1, 3]

2.2.3. Podudaranje malih znakova

Podudaranje malih znakova provodi se prolaskom kroz tuplove malih znakova sekvenci koje želimo sažeti. Tu vrstu tupleova dobili smo tijekom procesa vađenja podataka i sastoje se od pozicije i duljine niza malih znakova od te pozicije. Slične tupleove dobijemo i za referentnu sekvencu. Oni su također ključni u ovom procesu podudaranja. Inicijaliziramo niz koji je iste duljine kao i broj tuplea sekvence koju želimo sažeti. On će za sad na svim mjestima imati poseban znak ili broj (npr. -1) koji će indicirati da se točno taj tuple, tj. ta kombinacija pozicije i duljine, ne može naći u referentnoj sekvenci. Sada počinjemo prolaziti kroz tupleove vrijednosti malih znakova sekvence koju želimo sažeti i uspoređujemo svakog od njih sa svakim tupleom malih znakova referentne sekvence. Ako se podudaraju stavimo u inicijaliziranu listu za taj tuple redni broj tupla referentne sekvence s kojom se podudara. U suprotnom zapisujemo taj tuple u niz tuplea razlike.

Primjer

Možemo zapisati tupleove u ovom slučaju poput kombinacije dva broja *[pozicija, duljina]* i to ćemo koristiti u ovom primjeru za bolju vizualizaciju. Obradom naših sekvenci i vađenjem podataka o malim znakovima dobili smo sljedeće tupleove:

Informacije o malim znakovima reference:
[12, 4], [3, 3], [15, 3], [3, 7]
Informacije o malim znakovima sekvence za sažimanje:
[12, 10], [15, 3], [3, 7]

Možemo vidjeti da imamo dva tuplea koji se nalaze i u jednom i u drugom nizu [15, 3] i [3, 7]. Iskoristiti ćemo tu informaciju i napraviti dva niza. Prvi u kojem ćemo čuvati redni broj tuplea iz reference koji se (i ako se) pojavljuje u sekvenci za sažima-

nje. Drugi u koji ćemo spremati tupleove koji se ne pojavljuju u referenci i to onim redom kako se pojavljuju u sekvenci za sažimanje. Prvi niz će imati onoliko elemenata koliko imamo mjesta s malim znakovima u sekvenci za sažimanje. Sva mjesta će biti inicirana s posebnim znakom ili brojem koji će indicirati da se taj tuple ne pojavljuje u referenci. Sad prolazimo kroz svaki tuple sekvence za sažimanje i uspoređujemo ga sa svim tupleovima iz reference. Ako se tupleovi podudaraju zapiše se redni broj tuplea iz reference na mjesto tuplea iz sekvence za sažimanje u nizu, inače ostaje znak nepodudaranja. Sad ćemo prikazati popunjavanje nizova na način da ćemo proći tuple po tuple iz sekvence za sažimanje:

Početak
Niz sekvence za sažimanje:
[-1, -1, -1]
Niz razlike:
[]

---

1. tuple
Niz sekvence za sažimanje:
[-1, -1, -1]
Niz razlike:
[[12,10]]

---

2. tuple
Niz sekvence za sažimanje:
[-1, 2, -1]
Niz razlike:
[[12,10]]

---

3. tuple
Niz sekvence za sažimanje:
[-1, 2, 3]
Niz razlike:
[[12,10]]

2.3. Kodiranje podataka

Ovim procesom kodiraju se sve informacije. Identifikator sekvence i duljine redova kodiraju se RLE (run-length encoding) metodom [**?**], dok se posebni znakovi kodiraju statičnim entropijskim kodiranjem [**?**]. Vrijednosti pozicije kodiraju se prediktivnim inkrementalnim kodiranjem [**?**] tako da se pozicija sljedećeg entiteta predviđa na temelju vrijednosti pozicije i duljine trenutačnog entiteta. Sve kodirane informacije kodiraju se PPMD koderom koristeći alat 7-zip [**?**].

2.4. Dekompresija

Dekompresija se obavlja obrnutim postupkom od kompresije. Komprimirana datoteka dekomprimira se alatom 7-zip te RLE metodom, statičnim entropijskim kodiranjem [**?**] i prediktivnim inkrementalnim kodiranjem [**?**]. Sljedeće što je potrebno pri-

mijeniti je dekompresija dobivena procesom podudaranja. Kako bi se to ostvarilo, prva zapisana sekvenca dekodira se iščitavanjem podnizova u referentnom genomu koristeći informacije o pozicijama, duljinama i znakovima nepodudaranja. Sve ostale sekvence moraju se najprije dekomprimirati koristeći zapis rezultata podudaranja prve razine dobivenih za sve prethodeće sekvence. To se izvodi koristeći zapisane podatke o identifikatoru sekvence, pozicije i duljine podudarajućeg segmenta. Rezultat te dekompresije je rezultat podudaranja prve razine, koji se dekomprimira već opisanim postupkom. Naposlijetku, informacije o malim slovima, N znakovima i posebnim znakovima koriste se za sastavljanje originalnog zapisa sekvence.

3. Implementacija

Za implementaciju opisanog algoritma koristili smo programski jezik C++ te razvojno okruženje Visual Studio Code.

3.1. Vađenje podataka

Kod vađenja podataka koristili smo vektore za spremanje informacija o pozicijama i duljinama nizova malih znakova, N-ova i drugih znakova jer su nam potrebne veličine i količine podataka na početku nepoznate. Prvo smo izvadili informacije o malim znakovima krećući se znak po znak, bilježeći mjesto i duljinu niza. Mjesta, tj. pozicije koje će se spominjati su relativne na poziciju koja je došla prije njih, a ne apsolutne da označavaju na kojoj su udaljenosti od početka sekvence. Podaci o malim znakovima su uz čisti niz ACGT jedino što je potrebno dobiti obradom referentne sekvence tako da smo putem micali sve ostale znakove. Tijekom pomicanja po sekvencama male znakove pretvaramo u velike. Sva ostala obrada i vađenje podataka se odnosi samo na sekvence koje želimo sažeti. Potrebna su sad još dva prolaska po sekvenci velikih znakova kako bi dohvatili informacije o N znakovima i ostalim znakovima. Za N znakove smo uzeli pozicije početka niza N znakova te duljinu niza krećući se znak po znak. Za ostale znakove smo uzeli poziciju i o kojem znaku se radi ponovo krećući se znak po znak. Sad mičemo N i ostale znakove iz niza i imamo čisti niz sastavljen od A, C, G i T koji predajemo na podudaranja.

3.2. Podudaranje podataka

Za implementaciju podudaranja bilo je potrebno odabrati način na koji izračunati hash vrijednost ACGT niza za podudaranje prve razine te kombinacije pozicije, duljine i nepodudarajućih znakova za podudaranje druge razine. Metoda izračuna hash vrijednosti koju smo odabrali je polinomska rekurzivna hash funkcija. Koristili smo gotovi algoritam kao osnovu za implementaciju te funkcije [**?**] te smo joj dodali funkciju za kodiranje znakova A,C,G i T u brojeve 1,2,3 i 4. Za ostvarenje hash tablice koristili smo strukturu *unordered map* zbog konstantog vremena pretraživanja, dok smo za zapis vrijednosti pozicija i duljina segmenata, kao i nepodudarajućih znakova koristili vektore zbog mogućnosti mijenjanja veličine te manje prostorne složenosti od strukture *list*.

3.3. Kodiranje podataka

Id segmenta prepisuje se u novu datoteku. Zapisujemo i duljinu cijele sekvence prije vađenja podataka. Duljine linija zapisuju se pomoću *run-length-encoding*-a [**?**]. To znači da podatke zapisujemo u paru (*vrijednost broj*) gdje se *vrijednost* pojavljuje *broj* puta za redom. Zapis informacija o N i ostalim znakovima je sličan jer kombiniramo pozicije i duljine/znakove u jednu liniju za svaku od tih informacija. Kombinacija za N znakove je *pozicija duljina* po redu kako se pojavljuju u originalnoj sekvenci. Kombinacija za ostale znakove je *pozicija (vrijednost znaka umanjena za vrijednost "A")* po redu kako se pojavljuju u originalnoj sekvenci. Podudaranja malih znakova također prolazi kroz *run-length-encoding* [**?**] prije zapisa u datoteku, a nakon toga zapisujemo razlike u malim slovima u formatu *pozicija duljina*. Ostala podudaranja zapisuju se u formatu u kojem se nalaze nakon obrade, smatra se da su već u kodiranom obliku.

3.4. Dekompresija

Dekompresija koristi sve korake ranije navedene te većinu istih struktura i algoritama, ali u obrnutom smjeru kako bi od formata što smo dobili kompresijom dobili originalne sekvence. Svaka dekomprimirana sekvenca zapisuje se u svoju datoteku.

4. Rezultati testova

Testiranje smo proveli nad 10 sintetičkih sekvenci duljina od 10^2 do 10^7 te na 4 sekvence duljine 10^3 znakova koje su uzete iz 4 genoma *Escherichia coli*. Testiranje je ostvareno višestrukim pokretanjem kompresije i bilježenjem srednje vrijednosti rezultata.

4.1. Sintetičke sekvence

Sintetičke sekvence				
Duljina sekvence	To-be-compressed veličine (KB)	Vrijeme kompresije (ms)	Vrijeme dekompresije (ms)	Veličina nakon kompresije (KB)
100	0.345	5	2	0.3
500	1.623	10	2	0.685
1000	3.12	23	23	1.15
5000	15.63	77	77	4.83
10000	31.2	101	101	9.31
50000	156	665	665	45
100000	312	1285	1285	88.2
500000	1560	10615	10615	440
1000000	3102,72	35224	2	894
5000000	15605.76	174433	2	4802

Tablica 4.1: Rezultati testova nad sintetskim sekvencama

4.2. *Escherichia coli*

Escherichia coli 10^3 sekvence					
Referenca	To-be-compressed Files	To-be-compressed veličine (KB)	Vrijeme kompresije (ms)	Vrijeme dekompresije (ms)	Veličina nakon kompresije (KB)
e.coli-1	e.coli-2, e.coli-3, e.coli-4	3.24	17	8	1.1
e.coli-2	e.coli-1, e.coli-3, e.coli-4	3.24	20	5	1.09
e.coli-3	e.coli-1, e.coli-2, e.coli-4	3.24	19	4	1.09
e.coli-4	e.coli-1, e.coli-2, e.coli-3	15.24	21	4	1.10

Tablica 4.2: Rezultati testova nad Escherichia coli 10^3 sekvencama

Program maksimalno koristi 2465.5 MB radne memorije prilikom kompresije i dekompresije sekvenci.

5. Zaključak

Potreba za dodatnim memorijskim prostorom postoji od kad postoje računala, a sad je još izraženija zbog silne količine podataka koje nam stižu svakodnevno. Isto vrijedi i za područje bioinformatike gdje se svakim danom sakuplja sve više različitih, ali i sličnih sekvenci. Upravo u toj sličnosti su tvorci originalnog rada na temu HRCM-a pronašli rješenje za problem memorije. Zašto ponovo unositi i spremati iste dijelove sekvence koji postoje u nekoj drugoj već spremljenoj sekvenci? Zašto ponovo spremiti cijelu sekvencu kad slična već postoji u našoj memoriji? Samo ju malo promijenimo. Ovo je jako lukav potez. Proći po sekvencama i samo reći po čemu se jedna razlikuje od druge u relativno par crtica koje zauzimaju djelić prostora u usporedbi s cijelim sekvencama. Naravno postupak nije savršen, a može se postići čak i suprotan učinak ako sekvence koje su korištene u algoritmu nisu dovoljno velike ili slične. Sam postupak je u trenucima dosta logički zahtjevan i postoji puno mjesta na kojima se može pogriješiti, ali ako se dobro izvede i nad pravim podacima učinak je itekako vidljiv.