SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 4585

PORAVNANJE DUGAČKIH RNA OČITANJA

Antonio Jurić

Zagreb, lipanj 2016.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA ZAVRŠNI RAD MODULA

Zagreb, 17. ožujka 2016.

ZAVRŠNI ZADATAK br. 4585

Pristupnik: **Antonio Jurić (0036477605)**
Studij: Računarstvo
Modul: Računarska znanost

Zadatak: **Poravnanje dugačkih RNA očitanja**

Opis zadatka:

Potrebno je u postojeći GraphMap alat za poravnanje dugačkih DNA očitanja dodati mogućnost poravnanja dugačkih RNA očitanja. Posebnu pažnju posvetiti očitanjima koja se mapiraju na više međusobno udaljenih eksona. Istestirati na odabranom skupu simuliranih očitanja te usporediti performanse (točnost, količinu potrebne RAM memorije i brzinu izvođenja) s BBMap i Star alatima. Rješenje mora biti prilagođeno paralelnoj arhitekturi i napisano u jeziku C++, po C++11 standardu. Programski kod je potrebno komentirati i pri pisanju pratiti stil opisan u Googleovom C++ vodiču. Napisati iscrpne upute za instalaciju i izvođenje. Kompletnu aplikaciju postaviti na Github pod jednom od OSI-odobrenih licenci.

Zadatak uručen pristupniku: 18. ožujka 2016.
Rok za predaju rada: 17. lipnja 2016.

Mentor:

Izv. prof. dr. sc. Mile Šikić

Djelovođa:

Doc. dr. sc. Tomislav Hrkać

Predsjednik odbora za završni rad modula:

Prof. dr. sc. Siniša Srbljić

Posebne zahvale za izradu ovog završnog rada Mili Šikiću, Krešimiru Križanoviću te Ivanu Soviću

SADRŽAJ

1. UVOD

Bioinformatika jest znanstveno područje u kojem se istražuju genetički i drugi biološki podaci pomoću računalne tehnologije i statističkih metoda. Stoga, to je pravo interdisciplinarno područje koje povezuje biologiju, računarsku znanost te statistiku u analiziranju i interpretiranju tih bioloških podataka.

Sekvenciranje i poravnanje (engl. *alignment*) nizova DNA i RNA jest jedno od područja bioinformatike. Sekvenciranje jest očitavanje nizova nukleotidnih baza iz genetskog materijala. Poravnanje jest, od dobivenih očitanja nizova nukleotidnih baza, što točnije i brže sastaviti cjelokupni genom (u slučaju rada s DNA nizovima) ili pronaći gdje se ti očitani nizovi nukleotidnih baza nalaze na referentnom genomu (u slučaju rada s RNA nizovima).

Poravnanje nizova DNA omogućuje rekonstrukciju cjelokupnog genoma promatranog živog bića dok poravnanje nizova RNA omogućuje poznavanje izraženosti gena u pojedinim stanicama promatranog živog bića.

Ovaj rad bavi se rješavanjem problema poravnanja dugačkih RNA očitanja, tj. pronalaskom mjesta na referentnom genomu s kojeg dolaze očitani dijelovi nizova RNA. Zbog specifičnih svojstava RNA, poravnanje dugačkih RNA očitanja jest bitno kompliciranje od poravnanja dugačkih DNA očitanja. Te razlike i specifičnosti poravnanja RNA predstavljene su u radu.

Drugo poglavlje daje uvid u problematiku poravnanja RNA nizova opisujući svojstva RNA zbog koji se poravnanje razlikuje od poravnanja DNA nizova. Nanopore, alat koji radi očitanja za poravnanje, opisan je u trećem poglavlju uz svojstva njegovih očitanja. Četvrto poglavlje sadrži rješenje problema ovog rada uz objašnjenja korištenih algoritama. Opis procesa implementacije dan je u petom poglavlju. Simulirani podaci korišteni prilikom testiranja programskog rješenja opisani su u šestom poglavlju. Sedmo poglavlje sadrži rezultate i opis rezultate te moguća buduća poboljšanja.

2. RNA-SEQ

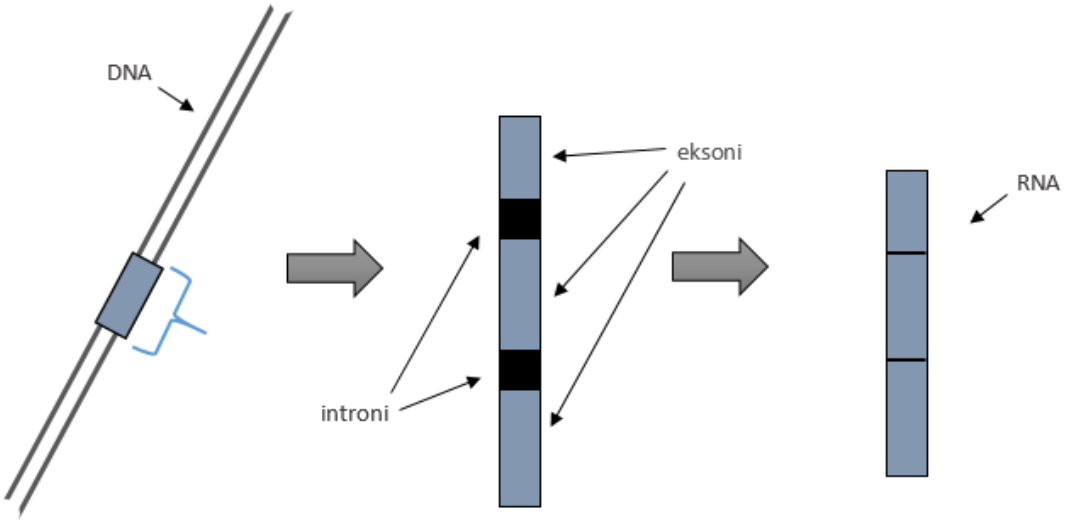
Poravnavanje nizova RNA omogućuje rekonstrukciju izraženosti gena u pojedinoj stanici živog organizma. Poznavanje izraženosti gena u pojedinim stanicama omogućuje detekciju pojedini genskih nepravilnosti (mutacija), pronalazak mogućih bolesti uzrokovanih naslijeđenim genetskim materijalom, promatranje aktivnosti gena ovisno o dobu dana, starosti, i sl., pronalasku kodirajućih regija proteina u genomu, sastavljanje prehrane na temelju poznavanja izraženosti gena (ovo je jedna od neuobičajenijih primjena poznavanja genetskog materijala kojom se bavi nutrigenomika[1]), detekciju i razlikovanje stanica raka od normalnih stanica tkiva, i drugo.

Postupak poravnanja DNA nakon faze sekvenciranja[2] dobivena očitanja poravnava na pripadajuća mjesta u genomu. Postupak poravnanja RNA nakon faze sekvenciranja također pokušava dobivena očitanja poravnati na pripadajuća mjesta u genomu, no ovaj postupak je nešto kompliciraniji od postupa s DNA-om. Razlog većoj složenosti ovog postupka u odnosu na DNA poravnanja nalazi se u samom svojstu RNA.

U pojednostavljenom postupku stvaranja RNA enzim sintetizira („prepisuje“) RNA iz lanca DNA. Taj proces naziva se transkripcija. Nakon prepisivanja RNA događa se proces izbacivanja introna: iz RNA se izbacuju nizovi baza koji su nekodirajući – ti nizovi nazivaju se introni. Preostali nizovi koji formiraju RNA nazivaju se eksoni. Slika 1. prikazuje proces nastanka RNA te izbacivanja introna.

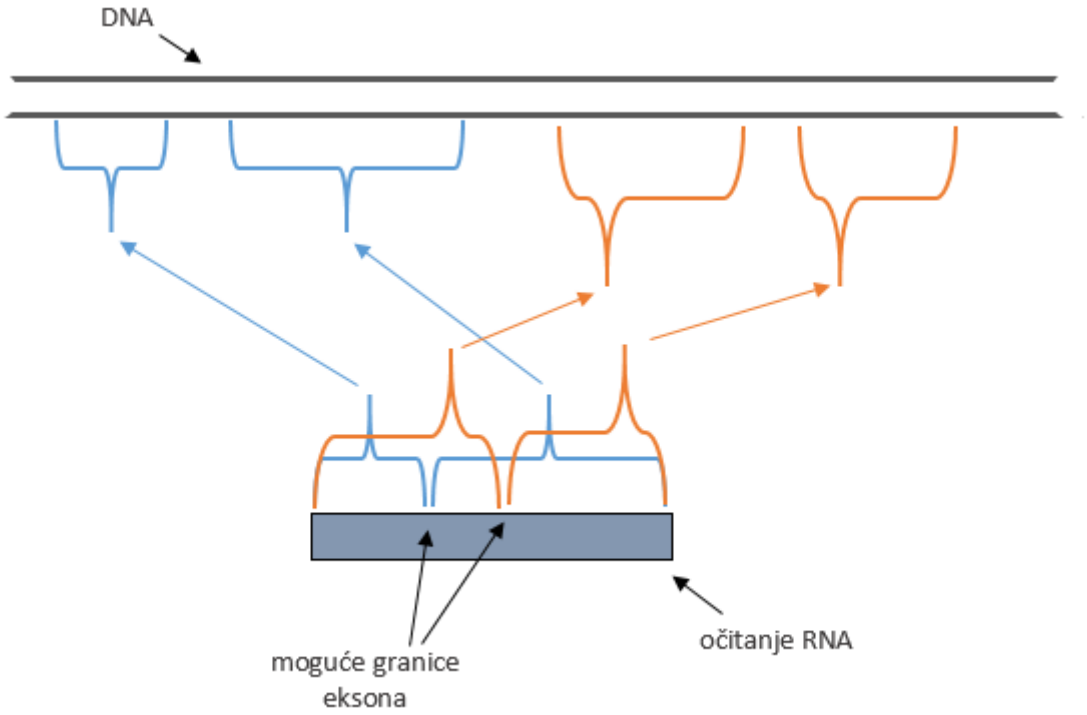
[1] Nutrigenomika je znanost koja proučava način na koji pojedini nutrijenti ili njihovi metaboliti međudjeluju s genomom pri regulaciji ekspresije gena. Cilj nutrigenomike je odrediti tip prehrane koji pojedincu najbolje odgovara na osnovi njegova genoma kako bi održao zdravlje i poboljšao kvalitetu svog života. [9]

[2] Sekvenciranje – očitanje nizova nukleotidnih baza iz genetskog materijala



Slika 1. **Proces nastanka RNA**

Ovo svojstvo izbacivanja introna bitno komplicira postupak poravnanja RNA: kada pokušavamo pronaći odgovarajuće mjesto u genomu za očitanje (engl. *read*) RNA, moguće je da imamo više mjesta na koje bi trebalo odgovarajuće dijelove (eksone) RNA poravnati na genom zbog izbačenih introna, ali ne znamo kako su postavljene granice između eksona očitanja.



Slika 2. **Nejasne granice eksona na RNA**

Neki od postojećih alati koji su razvijeni za postupak poravnanja RNA su Star ([20]) i BBMap([17]). S njima ćemo u usporediti naše rješenje. Glavna razlika između alata za poravanje jest u tome što rade s različitim tipovima podataka.

3. NANOPORE PODACI

MinION(proizvod tvrtke Oxford Nanopore Technologies) jest alat za sekvenciranje (alat koji daje očitanja nizova DNA i RNA) čija očitanja obrađuje GraphMap, alat čiju jednu od faza obrade poravnanja RNA očitanja obrađuje ovaj rad. On spada u trenutno zadnju (treću) generaciju alata za sekvenciranje te se od prethodnih alata razlikuje po duljini očitanja te preciznosti podataka.

Pod prvom generacijom alata za sekvenciranje podrazumijevamo metode i postupke koji su, između ostalog, korišteni i u Human Genome Projectu, velikom projektu sastavljanja ljudskog genoma za kojeg je bilo potrebno skoro 15 godina te više od tri milijarde dolara.

Druga generacija alata temelji sa na puno kratkih i preciznih očitanja. Problem koji se događa kod kratkih očitanja jest taj što se u genomu nalaze ponavljajuće regije baza pa nije jasno s koje regije je pojedino očitanje došlo. Ovaj problem komplicira cijeli proces poravnanja.

Treća generacija alata proizvodi očitanja veće duljine nego prijašnje generacije, ali se kod njih pojavljuje određena nepreciznost u očitanjima. Ipak, ta nepreciznost je ispod određene granice koja osigurava da rezultati poravnanja s takvim podacima budu prihvatljivi.

Tablica 1. prikazuje svojstva pojedinih generacija ([18]). Broj očitanja odnosi očitanja koja dobijemo u jednom procesu sekvenciranju.

Tablica 1. **Svojstva strojeva za sekvenciranje**

	Duljina očitanja (bp[3])	**Broj očitanja**	**Pogreška (%)**
Prva generacija	800	1	0.3
Druga generacija	[50 - 700]	$[10^5 - 5 \cdot 10^9]$	< 2
Treće generacija	> 5000	$[6 \cdot 10^4 - 1 \cdot 10^6]$	[10 - 34]

Na konferenciji London calling conference održanoj 26. – 27. svibnja 2016. iznesenoj je da je pogreška Nanoporea prilikom sekvenciranja svedena na prosječno 10%, ali i da se očekuje da će se daljnjim radom na usavršavalju tehnologije pogreška spustiti na $1 - 4\%$ ([12]).

[3] bp – engl. „base-pair“, hrv. „parova baza“

4. METODA

4.1. FAZE POSTUPKA

GraphMap ([19]) je alat za poravnanje DNA očitanja (*engl. read*) na referentni genom. Trenutno je cilj dodavanje mogućnosti poravnanja RNA očitanja na referentni genom bez prethodnog predznanja o granicama eksona. Cijeli postupak poravnanja provodi se u nekoliko faza, no da bismo opisali faze potrebno je znati kako se referentni genom obradi prije pokretanja postupka poravnanja.

Referentni genom indeksira se za brže pretraživanje, a svako očitanje u ulaznom skupu poravnava se zasebno i neovisno o drugim očitanjima.

Indeks referentnog genoma izveden je kao sažetak kratkih podsekvenci (zrna, engl. *seed*). Podsekvence se dobiju tako da se pomičnim prozorom (duljine oko 13 baza) prođe jednom kroz genom, na svakom pomaku se iz baza u prozoru generira ključ sažetka, a u sažetak se za taj ključ pohrane lokacije gdje se to zrno nalazi na referentnom genomu.

Faza poravnanja ima nekoliko:

1. Odabir regija - Referentni genom podijeli se na niz malih nepreklapajućih regija veličine: $\mathrm{duljina\_očitanja}/3$. Za neko očitanje pomičnim se prozorom odrede sva zrna. Za svako zrno napravi se pretraživanje sažetka i odrede se lokacije pogodaka. Za svaki pogodak nađe se odgovarajuća regija te se poveća brojač pogodaka u njoj. Ovaj korak služi da se smanji prostor pretraživanja za sljedeće korake. Regije se sortiraju po broju pogodaka te se procesiraju slijedno.

2. Mapiranje na graf - Ovaj dio koristi puno manju duljinu zrna ($\sim 6\mathrm{bp}$) te je puno osjetljiviji, a zrna automatski povezuje u duže nizove (fragmenti).

3. LCSk - Idealno, fragmenti koje dobijemo u prethodnom koraku bili bi svi na istom pravcu. No zbog ponavljanja u sekvencama u pravilu uvijek postoji puno veći broj anchora od očekivanog. LCSk nalazi najdulji niz fragmenata s monotono rastućim koordinatama i na očitanju i na referenci.

4. Filtriranje i grupiranje - LCSk pokušava globalno poravnati sekvence te ne generira nužno niz fragmenata koji su na pravcu, a neki fragmenti koje je odabrao mogu biti i pogrešni. Filtriranje se provodi metodom ulančavanja (povezuju se disjunktni i poredani fragmenti). Nakon što se fragmenti filtriraju, preostali fragmenti uglavnom su pouzdani. No u nekim slučajevima ne dobije se linearan pravac fragmenata, nego se na nekom mjestu može nalaziti velika rupa (umetanje ili u

očitanju ili u referenci). U tom slučaju fragmente grupiramo u linearne podgrupe. Izračunate grupe sadrže sljedeće podatke:

- početna (*readStart*) i završna (*readEnd*) koordinata na očitanju
- početna (*refStart*) i završna (*refEnd*) koordinate na referenci
- broj fragmenata koji su sadržani u toj grupi
- broj baza na očitanju (*coveredBases*) koje fragmenti pokrivaju

5. Poravnanje - Najbolje regije se poravnaju koristeći Myersov algoritam globalnog poravnanja po grupama (engl. *cluster*) i između njih.

4.2. DEFINICIJA PROBLEMA

U ovoj petoj fazi želimo od svih grupa odabrati one koje će najviše pokriti trenutni očitanje.

Definicija tog problema jest sljedeća: *uz dan skup regija i uz dan skup detektiranih grupa za svaku regiju, potrebno je odrediti optimalnu kombinaciju grupa za najvjerojatnije poravnanje očitanja.*

Parametri optimalnosti su sljedeći:

- koordinate grupa moraju biti slijedne i na očitanju i na referenci;

- potrebno je što više popuniti očitanje grupama;

- odabrati samo one grupe koje daju bolje rješenje ako postoji više grupa koje pokrivaju isti dio očitanja.

Drugi uvjet optimalnosti odnosni se na podatak grupa *coveredBases*.

Problem možemo matematički apstrahirati: *za dani skup dužina S u dvodimenzionalnom koordinatnom sustavu od kojih svaka ima zadani parametar dobrote potrebno je odabrati podskup S' dužina koje su nepreklapajuće, imaju rastuće koordinate te je suma dobrote tog podskupa najveća moguća*, tj:

Neka je $T_i \in \mathbb{N}^2$ te (T_{si}, T_{ei}, d_i) uređena trojka u kojoj je:

- T_{si} početna točka i-*te* dužine,
- T_{ei} konačna točka i-*te* dužine,
- d_i parametar dobrote i-*te* dužine.

Iz

$$S = \{(T_{si}, T_{ei}, d_i), i \in \mathbb{N}^2\}$$

treba pronaći

$$S' = \{(T_{si1}, T_{ei1}), (T_{si2}, T_{ei2}), \ldots, (T_{sik}, T_{eik})\}, S' \subseteq S$$

td.

$$T_{ei1} \leq T_{si2}, T_{ei2} \leq T_{si3}, \ldots, T_{eik-1} \leq T_{sik}$$

te

$$max \sum_{k=1}^{|S\prime|} d_{ik}$$

gdje relacija $\leq (T_1, T_2)$ jest relaclija za koju mora vrijediti $T_{1x} \leq T_{2x}$ i $T_{1y} \leq T_{2y}$.

Ovaj je problem zapravo jedan od oblika knapsack problema koji spada u poznate kombinatorne probleme. U nastavku slijede opisi knapsack problema i logaritamske strukture čije ideje koristimo u rješenjima problema.

4.3. KNAPSACK

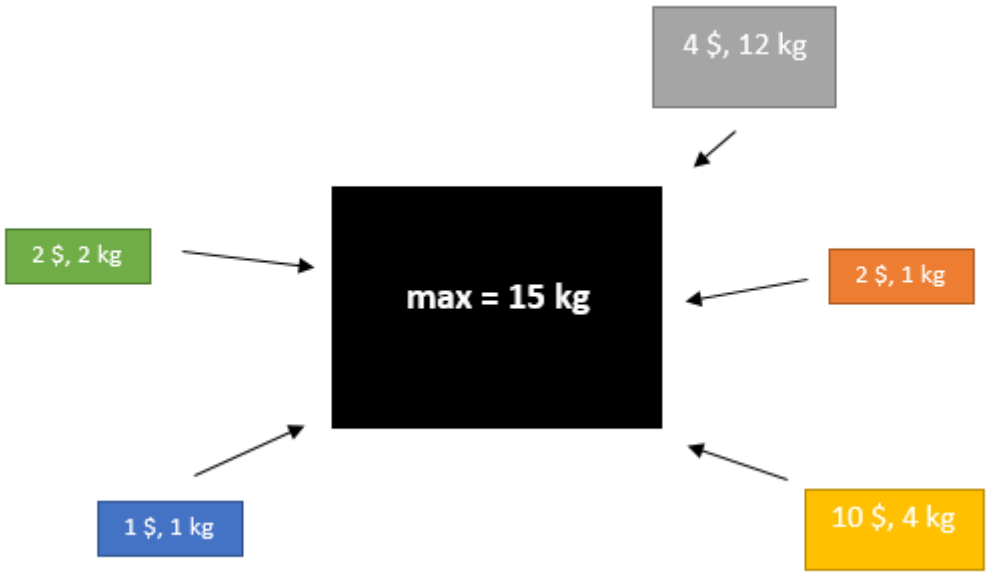
Jednostavnu verziju knapsack problema [2] možemo opisati na sljedeći način: za dani skup od N elemenata numeriranih od 1 do N, svaki težine t_i i vrijednosti v_i te za ograničenje ukupne težine T potrebno je pronaći podskup elemenata čija suma težina ne prelazi T, a daje najveću sumu vrijednosti, tj. formalnije:

$$max \sum_{i=1}^{N} v_i \cdot x_i$$

uz uvjet

$$\sum_{i=1}^{N} t_i \cdot x_i \leq T, x_i \in \{0, 1\}.$$

Ovdje x_i predstavlja broj koliko se puta *i-ti* element pojavljuje u rješenju. Ograničivši se na vrijednosti *0* i *1* zapravo određujemo da se svaki element može pojaviti najviše jednom, tj. da nema ponavljanja elementa.



Slika 3. **Primjer knapsack problema**

Jedna od metoda rješavanja ovog problema jest dinamičko programiranje koje rješava problem svodeći ga na manje i rješive probleme. Ono definira stanje $dp[i, t]$ u kojem za neku ukupnu težinu t koristeći samo prvih i elemenata pokušava pronaći najbolje rješenje (najveću sumu vrijednosti). Iterirajući redom po broju korištenih elemenata te krenuvši od najmanje moguće ukupne težine 0 pa do najveće T, algoritam pokušava popraviti rješenje trenutnog potproblema preko prijašnjih rješenja isprobavajući ubaciti svaki od elemenata te pazeći da suma težina odabranih elemenata ne prelazi T:

- $dp[0, t] = 0$ (početni slučaj – za prvih 0 elementa vrijednost je 0 jer nismo ništa uzeli)
- $dp[i, t] = dp[i - 1, t]$ (ako je t_i (težina elementa kojeg pokušava ubaciti) veća od trenutnog ograničenja na težinu t)
- $dp[i, t] = max(dp[i – 1, t], dp[i – 1, t - t_i] + vi)$.

Konačno rješenje zapisano je u $dp[N, T]$ koje predstavlja koja je najveća moguća ukupna vrijednost ako koristimo sve elemente i ukupna težina nije veća od T.

Ako bi problem rješavali pohlepnim algoritmom, vjerojatno bismo vrlo brzo zaglavili s nekim podoptimalnim rješenjem. Uzevši problem sa slike 3., to bi vjerovatno bio sivi element i još neki s manjom težinom, dok najbolje rješenje jest

uzeti upravo sve elemente umjesto tog što metoda dinamičkog programiranja i pronalazi (uz redoslijed elemenata *(4, 12), (2, 1), (10, 4), (1, 1), (2, 2)*):

i\t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	4	4	4	4
2	0	2	2	2	2	2	2	2	2	2	2	2	4	6	6	6
3	0	2	2	2	10	12	12	12	12	12	12	12	12	12	12	12
4	0	2	3	3	10	12	13	13	13	13	13	13	13	13	13	13
5	0	2	3	4	10	12	13	14	15	15	15	15	15	15	15	15

Slika 4. **Izračun knapsack problema**

U *dp[5, 15]* zapisano je rješenje 15.

Izračun koji elementi ulaze u rješenje obično radimo pamćenjem indeksa onih elemenata pomoću kojih smo popravili neko od međurješenja. U našem slučaju, prilikom $dp[i, t] = max(dp[i - 1, t], dp[i - 1, t - t_i] + vi)$ u situaciji da je $dp[i - 1, t]$ veći, zapamtili bismo indeks tog međurješenja, a inače bismo za ovo međurješnje postavili novi indeks: i. Nakon izvršenja algoritma jednostavnim linearnim prolaskom po zapamćenim indeksima možemo rekonstruirati koji elementi su ušli u rješenje.

Vremenske i prostorne složenosti ovog algoritma su jednake: $O(N \cdot T)$.

4.4. LOGARITAMSKA STRUKTURA

Logaritamska struktura [5] jest podatkovna struktura koja omogućuje izvršavanje operacija nad intervalima podataka u logaritamskoj složenosti:

- operacija upisivanja elementa te
- operacija upita za zadani interval

obje se izvršavaju u logaritamskoj složenosti ovisno o najvećem dopuštenom broju podataka.

Valja napomenuti da je interval u operaciji upita često ograničen na način da mu je lijeva granica jednaka početku niza, tj. promatra se interval od početka do i-*tog* mjesta u nizu. To kod nekih operacija ne predstavlja problem jer je moguće pomoću postojećih operacija konstruirati složenu operaciju koja može dati odgovor za proizvoljan interval dok kod drugih to nije moguće (npr. minimum na intervalu).

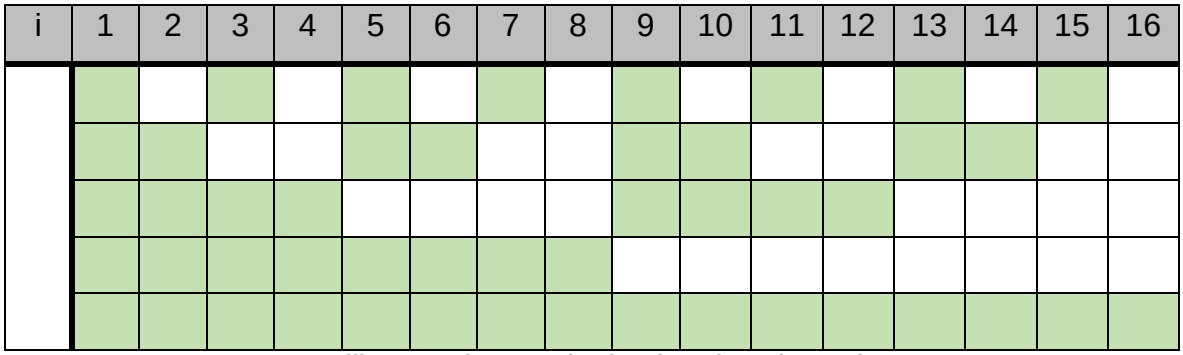
Logaritamska struktura može podržavati razne operacije na intervalu:

- suma intervala (proizvoljan interval),
- minimum na intervalu (isključivo od početka niza do i-*tog* mjesta),

- maksimum na intervalu (isključivo od početka niza do *i-tog* mjesta). Pokazat ćemo primjer logaritamske strukture s upitom za maksimum na intervalu budući da ćemo takvu logaritamsku strukturu i koristiti u rješenju. Konstrukcije svih ostalih upita su sliče i ovdje ih nećemo navoditi.

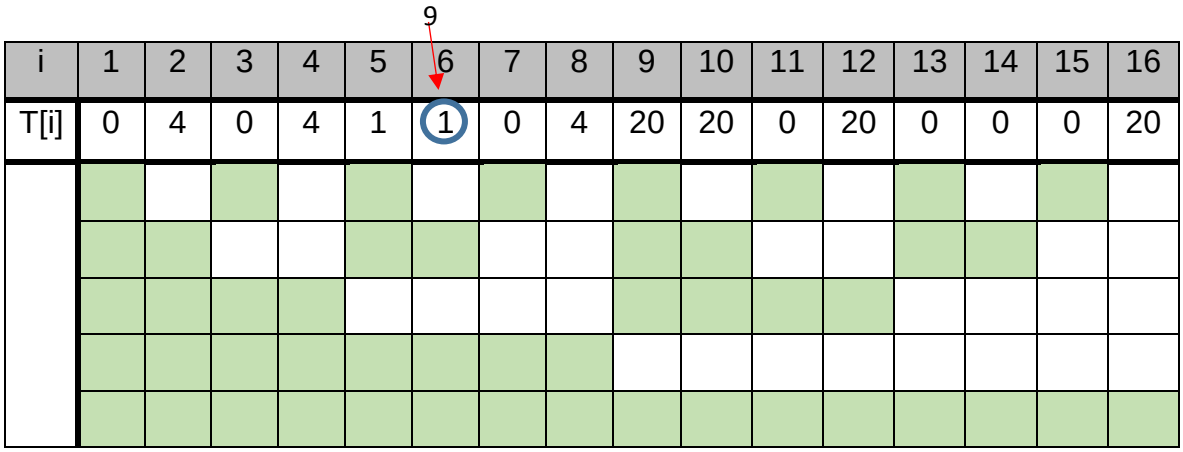
Logaritamska struktura s upitom za maksimum na intervalu predstavljena je jednostavnim nizom u kojem svaki element pamti maksimum na intervalu za koji je odgovoran. Interval svakog elementa određuje se na temelju najmanje potencije broja *2* koji se pojavljuje u binarnom zapisu indeksa tog elementa: interval *i-tog* elementa jest $[i - 2^k + 1, i]$ gdje je 2^k najmanja potencija broja *2* u binarnom zapisu broja *i*.

Stoga, kod operacije upisivanja na *i-to* mjesto potrebno je ažurirati sve intervale koji pokrivaju ovu poziciju, a kod operacije upita za maksimum na intervalu *[1, i]* potrebno je obići intervale koji pokrivaju ovo područje. Broj takvih intervala u oba slučaja iznosi najviše *log(N)* gdje je *N* ukupan broj elemenata. Sljedeće slike ilustriraju intervale elemenata, operaciju upisivanja te operaciju upita s nizom od *16* elemenata.

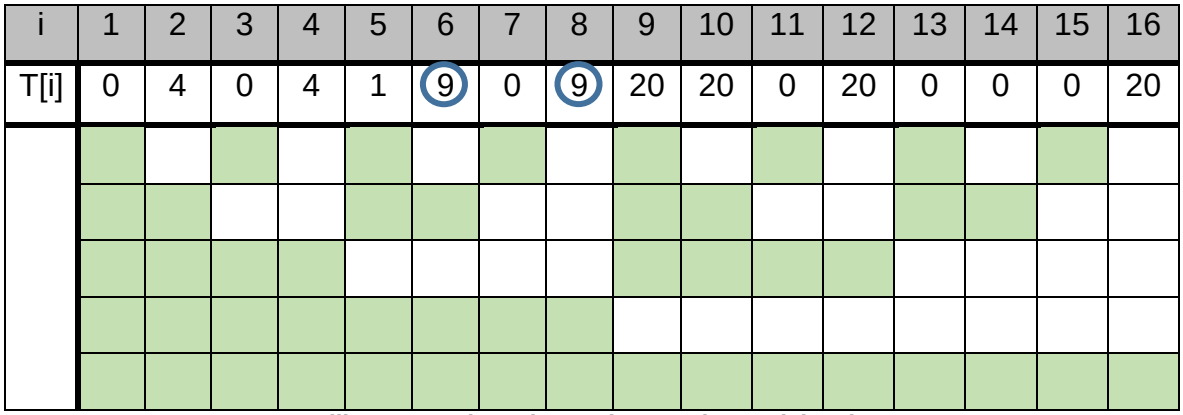


Slika 5. **Struktura podataka - logaritamska struktura**

Zelena polja na slici 5. označavaju interval koji pokriva *i-ti* element. Tako element na poziciji *6* pokriva interval *[5, 6],* element na poziciji *11* pokriva samo sebe *[11, 11]*, dok element na poziciji *16* pokriva interval *[1, 16]*. Iskoristimo li formulu $[i - 2^k + 1, i]$, vidimo da se navedeni intervali poklapaju s danom formulom za određivanje intervala $(6: [6 - 2^1 + 1, 6], 11: [11 - 2^0 + 1, 11], 16: [16 - 2^4 + 1, 16])$.

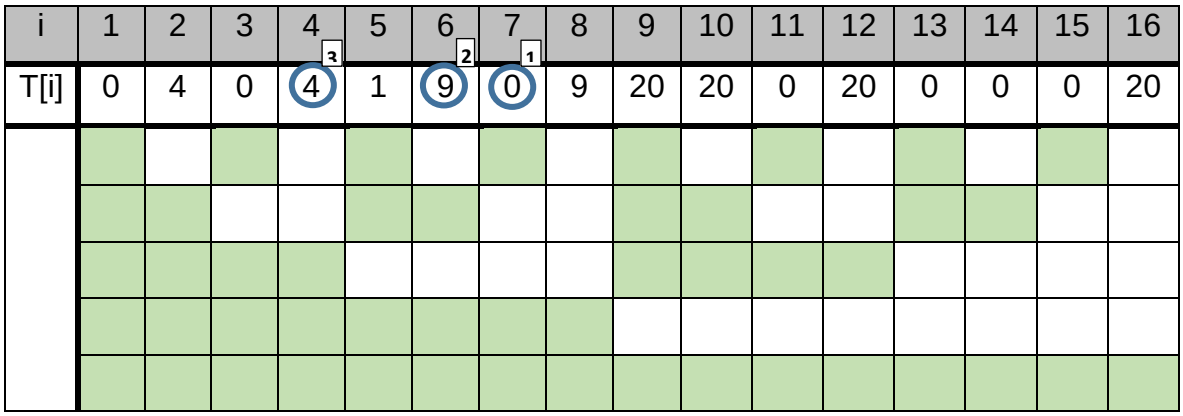


Slika 6. **Upisivanje u logaritamsku strukturu**



Slika 7. **Logaritamska struktura nakon upisivanja**

Slika 6. prikazuje stanje logaritamske strukture (*T[]* je niz u kojem računamo logaritamsku strukturu) u kojoj su ubačeni elementi *4* na poziciju *2*, *1* na poziciju *5* te *20* na poziciju *9*. Primijetimo kako odgovarajući intervali koji pokrivaju pozicije tih elemenata također sadrže njihove vrijednosti: *4* sadrži i pozicije *4* i *8*, *1* i pozicija *6*, a *20* i pozicije *10, 12* te *16*. U logaritamsku strukturu želimo na poziciju *6* ubaciti broj *9*. Na slici *7*. prikazano je koje su vrijednosti logaritamske strukture promijenjene: na poziciju *6* upisan je broj *9*, nakon toga je popravljena vrijednost na pozicij *8*, a dalje nisu popravljane vrijednosti jer, iako je interval elementa na pozicij *16* odgovoran (pokriva) poziciju *6*, vrijednosti na tim pozicijama su veće od broja *9* (u našem slučaju to je broj *20*).



***Slika 8.* Izvršavanje operacije upita u logaritamskoj strukturi**

Slika 8. prikazuje upit za maksimum na intervalu *[1, 7]*. Prilikom izračuna maksimuma obilazimo samo one pozicije čiji intervali pokrivaju traženi interval: pozicija *7* nakon koje je maksimalna vrijednost *0*, pozicija *6* nakon koje je maksimalna vrijednost *9* te pozicije *4* nakon koje maksimalna vrijednost ostaje *9*. Slijede pseudokodovi operacija upisivanja i upita u logaritamsku strukturu (N je veličina niza):

```
Function SET(i, value, T, N)
  for x ← i to N step lobit(x)
    T[x] = max(T[x], value)

Function GET(i, T)
  ret = -∞
  for x ← i to 0 step –lobit(x):
    ret = max(ret, T[x])
  return (ret)

Function LOBIT(x)
  return (x & -x)
```

Algoritam 1. Operacije upisivanja i upita logaritamske strukture

Primijetimo da ako prilikom unosa na neku poziciju želimo pregaziti staru vrijednost s upisom broja koji je manji od trenutnog broja, operacija neće uspjeti, tj. ostat će zapamćene stare vrijednosti većeg broja (zbog naredbe u petlji u set funkciji). Algoritmu u kojem ćemo ga obliku mi koristiti to neće smetati jer ćemo uvijek ubacivati samo veće vrijednosti.

Vremenska složenost ovog algoritma za obje operacije jest $O(\log(N))$, a prostorna složenost jest $O(N)$.

Logaritamska struktura se s jednodimenzionalnog niza lako može proširiti na višedimenzionalne nizove i tako umjesto jednodimenzionalnih rješavati višedimenzionalne probleme. U našem algoritmu koristi ćemo dvodimenzionalni oblik logaritamske strukture.

Jednodimenzionalni problem možemo zamisliti kao vrijednosti za neku fiksnu x koordinatu za koju obrađujemo sve y vrijednosti. Sukladno tome, za svaku x koordinatu potrebno je stvoriti poseban niz koji će pamtiti vrijednosti za njezine y vrijednosti. Operacija upisivanja ubacuje element na određenu poziciju te ažurira pozicije koje intervalom pokrivaju poziciju ubačenog elementa (za svaki x odgovarajuće y pozicije). Operacija upita vraća element koji je maksimum na intervalu *[1 .. x, 1 .. y]*. Pseudokodovi su slični (sada je *T[,]* dvodimenzionalno polje, *lobit(x)* je isti):

```
Function SET(x, y, value, N, T)
  for i ← x to N step lobit(i)
    for j ← y to N step lobit(j)
      T[i, j] = max(T[i, j], value)

Function GET(x, y, T)
  ret = -∞
  for i ← x to 0 step –lobit(i):
    for j ← y to 0 step –lobit(j):
      ret = max(ret, T[i, j])
  return (ret)
```

Algoritam 2. **Operacije upisivanja i upita 2D logaritamske strukture**

Složenost operacija dvodimenzionalne logaritamske strukture iznosi $O(log^2(N))$ (općenito, to je umnožak logaritamskih vrijednosti dimenzija *log(N)*), dok prostorna složenost iznosi $O(N^2)$ (općenito, to je umnožak dimenzija).

4.5. PRIPREMA PODATAKA I RJEŠENJE

Prije samog računanja rješenja potrebno je pripremiti podatke. Grupe sortiramo uzlazno po obje koordinate (prvo po početnoj pa po završnoj točki dužine), a unutar koordinata po pokrivenosti bazama. Korak sortiranja jest potreban kako bismo kasnije u računanju rješenja mogli iskoristiti prednosti sortiranih podataka i ubrzati izvršavanje. Zajedno sa sortiranjem radimo i izbacivanje duplikata grupa koje su nastali u prvoj fazi cjelokupnog procesa poravnanja: regije smo prije procesiranja proširili s obje strane za duljinu očitanja kako bismo pokušali svakom regijom pokriti cijelo očitanje (budući da je početna duljina regije jednaka trećini očitanja).

Tim proširenjem moguće je da su se susjedne regije koje imaju puno hitova poklopile pa smo u kasnijem koraku dobili jednake grupa. Nadalje, potrebno je grupe razdvojiti po lancu DNA (engl. *strand*) jer u jednom poravnanju mogu biti isključivo grupe s istog lanca. Uzimamo u obzir da očitanja mogu biti poravnana na oba lanca jer kada dobijemo očitanje zapravo ne znamo s kojeg je lanca on sekvenciran. Konačno, grupe koji pripadaju različitim referencama odvajamo te odvojeno računamo rješenja za svaku referencu (u isto rješenje ne mogu ući grupe koji pripadaju različitim referentnim nizovima DNA). Grupe koje pripadaju različitim referentnim nizovima DNA dobijemo u prijašnjim fazama cijelog postupka poravnanja zbog toga što se genom koji koristimo kao referentni (na kojeg pokušavamo napraviti poravnanje) može sastojati od više kromosoma (npr. ljudski se genom sastoji od dvadeset i tri kromosoma). Za pojedino očitanje ne znamo s kojeg je kromosoma došao pa tražimo najbolje mjesto za njega pretražujući sve kromosome (u našem algoritmu to nazivamo referencama). Pseudokod je prikazan u nastavku.

```
Function MAP_CLUSTERS(data)
  for i ← 0 to size(data.references)
    clusters = getClusters(data.references[i])
    forwardStrand, reverseStrand = separateStrands(removeDuplicates(sort(clusters)))
    sol = max(sol, mappingAlg(forwardStrand), mappingAlg(reverseStrand))
  return (reconstructUsedClusters(sol))
```

Algoritam 3. **Računanje rješenja**

Pseudokod pokazuje kako odvojeno računamo poravnanje za različite lance DNA (mappingAlg(forwardStrand) i mappingAlg(reverseStrand)) i uzimamo bolje od ta dva rješenja te od dosadašnjeg najbolje. Nakon pronalaska najboljeg rješenja po

svim referencama (a unutar njih po lancima), rekonstruiramo koje su grupe ušle u rješenje. Funkcija mappingAlg(), koja prima grupe s jednog od lanaca, poziva jedan od dva algoritma s kojima računamo poravnanje: ovisno o ukupnom broju grupa poziva se algoritam koji koristi dinamičko programiranje (knapsack problem) ili algoritam koji koristi 2D logaritamsku strukturu.

4.6. ALGORITAM 1 – KNAPSACK

Ideja predstavljenog rješenja knapsack problema koristi se u prvom algoritmu početnog problema. Paralele su sljedeće: elementi su grupe (dužine), vrijednosti elemenata su pokrivenosti baza (dobrote dužina), težine elemenata su koordinate grupa (početna i završna točka dužine), a ukupna dopuštena težina T jest duljina očitanja. Pseudokod algoritma prikazan je u nastavku.

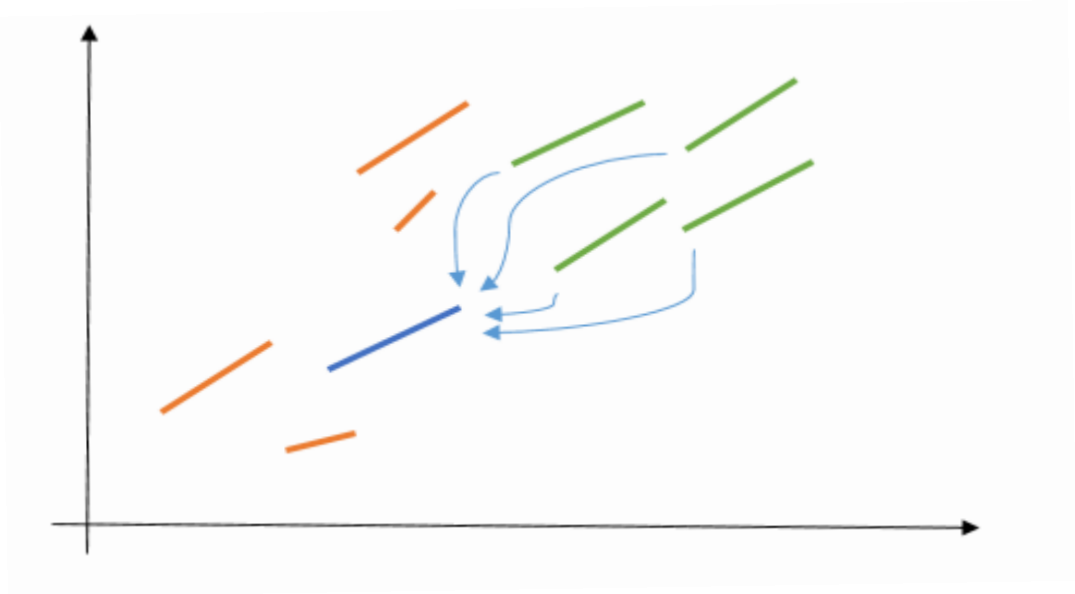
```
Function CALCULATE_DP(clusters, dp, backtrack)
  init(dp), init(backtrack)
  for i ← size(clusters) to 0:
    for j ← i to size(clusters):
      if intersects(clusters[i], clusters[j]) then continue
      if clusters[j].coveredBases + dp[j] > dp[i] then
        dp[i] = clusters[j].coveredBases + dp[i]
        backtrack[i] = j
```

Algoritam 4. **Knapsack**

Na početku je potrebno ispravno inicijalizirati strukture podataka. Polje *dp[]* inicijaliziramo na *0*. U *dp[i]* jest zapisano koja je najveća moguća pokrivenost ako su koordinate *i-te* grupe zadnje. Polje *backtrack[]* inicijaliziramo na *-1*. U *backtrack[i]* zapisano je koja se grupa nastavlja na *i-tu* grupu u rješenju. Ovo polje potrebno je postaviti na *-1* kako bismo kasnije u fazi rekonstrukcije imali uvjet zaustavljanja prilikom rekonstrukcije poravnanih grupa (ona grupa koji ima polje *backtrack[]* postavljeno na *-1* jest zadnja u rekonstrukciji).

Fiksiranjem indeksa *i* odabiremo svaku grupu čije koordinate postavljamo kao zadnje. Bitno je ovo činiti unatrag zbog ispravnosti algoritma. Iteriranjem po indeksu *j* koji kreće od *i* pa do kraja popravljamo međurezultat *dp[i]* tako da svaka grupa koja je u sortiranom redoslijedu (sortiranje smo obavili prije pokretanja ovog algoritma) poslije *i-te* grupe, tj. desno od nje, i koja se ne presijeca s *i-tom* grupom pokušamo nadodati na *i-tu* grupu. Ako je to rješenje bolje od trenutnog, zapišemo ga u *dp[i]* te zapamtimo u *backtrack[i]* da se na *i-tu* grupu nastavlja *j-ta* grupa. Na

slici 9. demonstrirano je popravljanje rezultata u $i\text{-}tom$ koraku: plavom bojom označena je $i\text{-}ta$ grupa, zelenom bojom označene su grupe koji se mogu nastaviti na nju, a narančastom bojom označene su grupe koji se ili sijeku s njom ili su im koordinate manje od koordinata $i\text{-}te$ grupe.



Slika 9. **Popravljanje rezultata u i-tom koraku knapsacka**

Vremenska složenost ovog algoritma jest $O(N^2)$ dok je prostorna $O(N)$ gdje je N broj grupa. Konstanta vremenske složenosti iznosi $½$ zbog toga što u unutarnjoj iteraciji ne prelazimo sve grupe, nego krećemo od $i\text{-}te$ pa do kraja (što smo omogućili sortiranjem grupa) čime smo ubrzali vrijeme izvođenja.

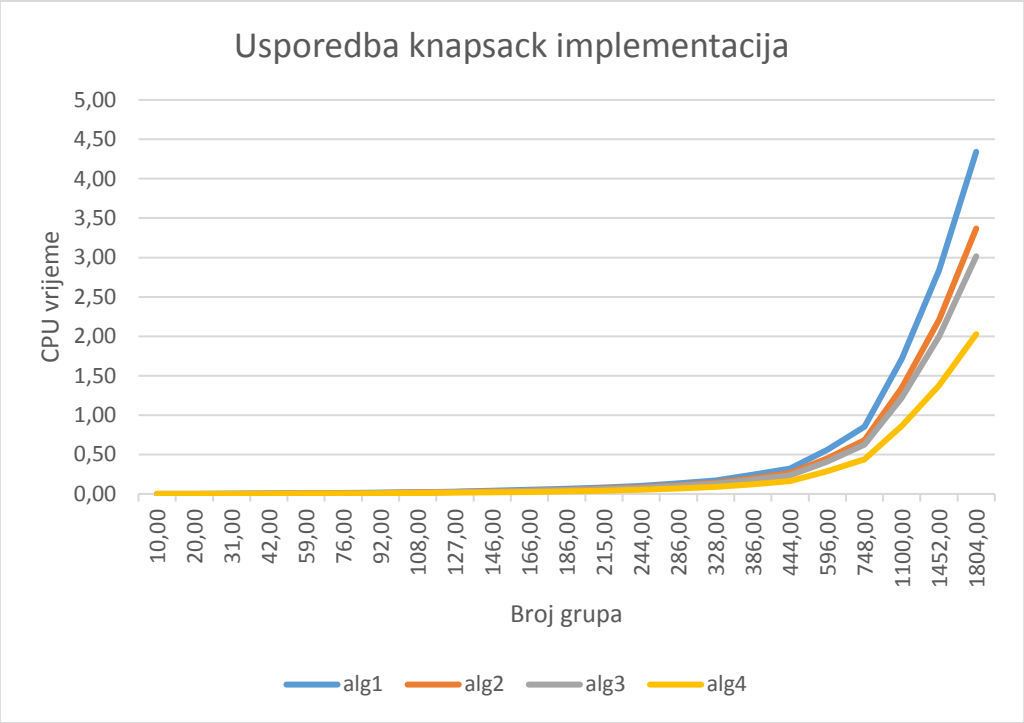
Implementacija ovog algoritma prošla je kroz četiri iteracije. Svaka iteracija ima istu vremensku ($O(N^2)$) i prostornu ($O(N)$) složenost, ali su im konstante kod vremenske složenosti različite. Iteracije implementacija su sljedeće:

- *alg1*: rekurzivna verzija s memoizacijom,
- *alg2*: rekurzija verzija s memoizacijom i sortiranjem prije pokretanja i „pametnim iteriranjem“,
- *alg3*: iterativna verzija,
- *alg4*: iterativna verzija sa sortiranjem prije pokretanja i „pametnim iteriranjem“.

„Pametno iteriranje“ odnosi se na to da unutarnja petlja kreće od i umjesto da ide isto po svim grupama kao vanjska što možemo učiniti zbog sortiranosti podataka. Uočimo da u slučaju da radimo „pametno iteriranje“, a grupe nisu sortirani, u općenitom slučaju bismo izgubili pravo rješenje. Slika 10. prikazuje

vremena izvršavanja ovih implementacija na serveru s performansama *Intel(R) Xeon(R) CPU E5645 @2.4GHz, cpu cores: 6, cache size: 12 MB, RAM: 297GB*.

Ukupno vrijeme izvršavanja pojedine implementacije na pojedinom skupu podataka (grupama) dobiveno je kao suma vremena izvršavanja pokrenutog 1000 puta za tu implementaciju na tom skupu podataka. Najbolje (najkraće) vrijeme izvođenja ima četvrta iteracija (iterativna verzija sa sortiranjem prije pokretanja i „pametnim iteriranjem“) stoga je ona uvrštena u konačno rješenje i dan je njezin pseudokod.



Slika 10. **Vremena izvršavanja implementacija knapsacka**

4.7. ALGORITAM 2 – 2D LOGARITAMSKA STRUKTURA

Ovo rješenje zapravo je samo ubrzanje jednog dijela u prvom rješenju (knapsacku): dijela u kojem za $i\text{-}tu$ grupu tražimo desno od nje najbolju grupu koju bismo na nju nastavili uz uvjet da se ne preklapa s njom. Uočimo da u ovom koraku tražimo samo jedan grupu i to onu koja bi dala najbolje produljenje (suma pokrivenosti bi bila najveća). Tu informaciju nam upravo može dati 2D logaritamska struktura s upitom za maksimum.

Grupe obilazimo s lijeva nadesno. Za svaku grupu pozivamo operaciju upita pomoću koje saznajemo maksimum na intervalu $[1 .. readStart, 1 ... refStart]$. Budući da ovim saznajemo koja je grupa do sada najbolja na koju možemo nastaviti

trenutnu grupu i ne preklapa se s njom (jer smo upit radili s *readStart* i *refStart*), dovoljno je samo na pokrivenost dobivenu iz upita nadodati pokrivenost trenutne grupe te u 2D logaritamsku strukturu operacijom upisivanja na poziciju (*readEnd*, *refEnd*) upisati tu novu pokrivenost. Bitno je novu dobivenu pokrivenost upisivati na poziciju (*readEnd*, *refEnd*) da bismo se osigurali da će operacija upita uvijek vraćati rezultat za nepreklapajuće grupe kada je za neku grupu pozovemo s njezinim (*readStart*, *refStart*).

Primijetimo da bi za ovakav oblik algoritma, zbog veličine koordinata grupa, bila potrebna 2D logaritamska struktura s jako velikom količinom memorije. No, najveći ukupan broj grupa u podacima će biti reda veličine 10^3 i to nas dovodi do sljedeće optimizacije: budući da imamo najviše $2 \cdot 10^3$ različitih koordinata, moguće je napraviti kompresiju koordinata. Kompresija koordinata podrazumijeva da sve koordinate preslikamo u interval *[1, M]* gdje je *M* broj različitih koordinata održavajući njihov relativan odnos. Ovim postupkom nismo, zbog održanja relativnog odnosa koordinata, izgubili nikakvu informaciju, a dobili smo smanjenje potrebnog memorijskog prostora na približno $(2 \cdot 10^3)^2$. Pseudokod prikazuje ovaj algoritam.

```
Function CALCULATE_FNW (clusters, F)
  init(F), init(sol)
  for cluster in clusters:
    currMax = F.get(cluster.readStart, cluster.readEnd)
    newValue = (currMax.coveredBases + cluster.coveredBases, cluster.index)
    sol = max(sol, newValue)
    F.set(newValue)
  return (sol)
```

Algoritam 5. **2D logaritamska struktura**

U pseudokodu valja primijetiti da u 2D logaritamsku strukturu uz podatak pokrivenosti baza zapisujemo i podatak o indeksu grupe: taj podatak, jednako kao u prošlom algoritmu, služi za rekonstrukciju grupa koji su ušli u poravnanje.

Vremenska složenost ovog algoritma jest $O(N \cdot log^2(N))$ (N za fiksiranje svake grupe te $log^2(N)$ za operacije upisivanja i upita u 2D logaritamsku strukturu) dok prostorna složenost iznosi $O(N^2)$ gdje je N broj grupa. Vremenska složenost kompresije koordinata iznosi $O(N \cdot log\,(N))$ zbog sortiranja i preslikavanja koordinata u interval *[1, N]*, no recimo da je ta složenost zanemariva zbog veće (iako ne puno) složenosti glavnog postupka.

4.8. KOMBINACIJA ALGORITAMA

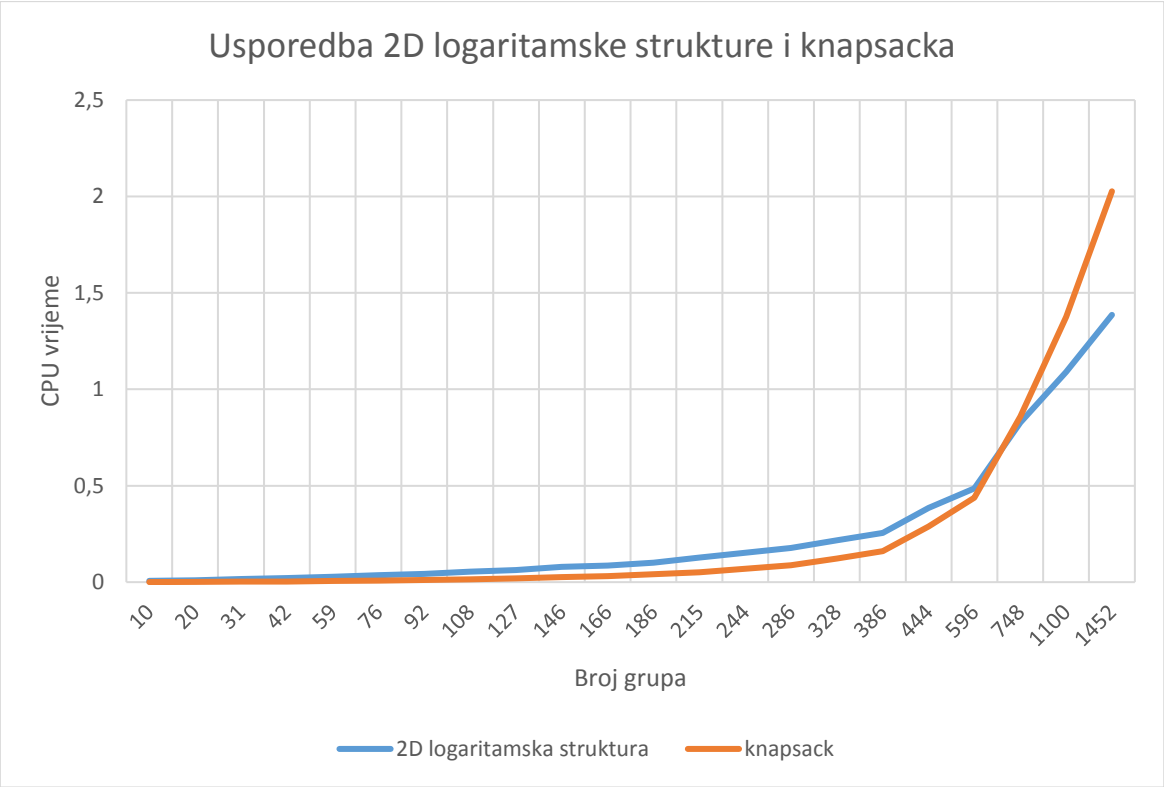
Iako drugi algoritam (2D logaritamska struktura) ima bolju vremensku složenost od prvog algoritma (knapsacka), prvi algoritam ima u određenim situacijama puno bolje vrijeme izvršavanja. Razlog možemo tražiti u prostornoj složenosti. Dok je prvi algoritam prostorne složenosti $O(N)$, drugi ima prostornu složenost $O(N^2)$. Prvi algoritam koristi jednodimenzionalan niz kako bi pamtio međurješenja prilikom izračuna dinamičkog programiranja, dok drugi algoritam koristi dvodimenzionalni niz kako bi pamtio strukturu potrebnu za 2D logaritamsku strukturu. Upravo to je razlog bržeg izvršavanja prvog algoritma u određenim situacijama: prilikom pristupanju jednodimenzionalnom nizu procesor puno češće potrebne podatke može naći već i priručnoj memoriji (engl. cache), dok se kod dvodimenzionalnog niza češće događa da mu potrebni elementi dvodimenzionalnog niza (jer je cijeli niz prevelik) nisu spremni u priručnoj memoriji već ih je potrebno prvo dohvatiti iz radne memorije što je skupa (vremenski zahtjevna) operacija. Ova situacija u kojoj procesor potrebne podatke ne pronalazi u priručnoj memoriji, nego ih prvo mora dohvatiti iz radne memorije, poznata je pod nazivom *cache miss* ([6]).

Prvi algoritam ima brže vrijeme izvođenja s manjim brojem grupa od drugog algoritma unatoč goroj vremenskoj složenosti. Razlog je jednostavan: manji broj cache misseva. Zbog ove situacije, konačno rješenje početnog problema koristit će kombinaciju ova dva algoritma. Svaki će se paliti ovisno o ukupnom broju grupa.

Tablica 2. **Prag za pokretanje algoritama**

Broj grupa	Algoritam
< 800	Algoritam 1 – knapsack
≥ 800	Algoritam 2 – 2D logaritamska struktura

Do ove granice (nazovimo je okidačem za algoritme) došli smo mjerenjem vremena izvršavanja oba algoritma nad različitim brojem grupa kako i prikazuje graf na slici 11. također mjereno na istom serveru kao i za implementacija knapsackova.



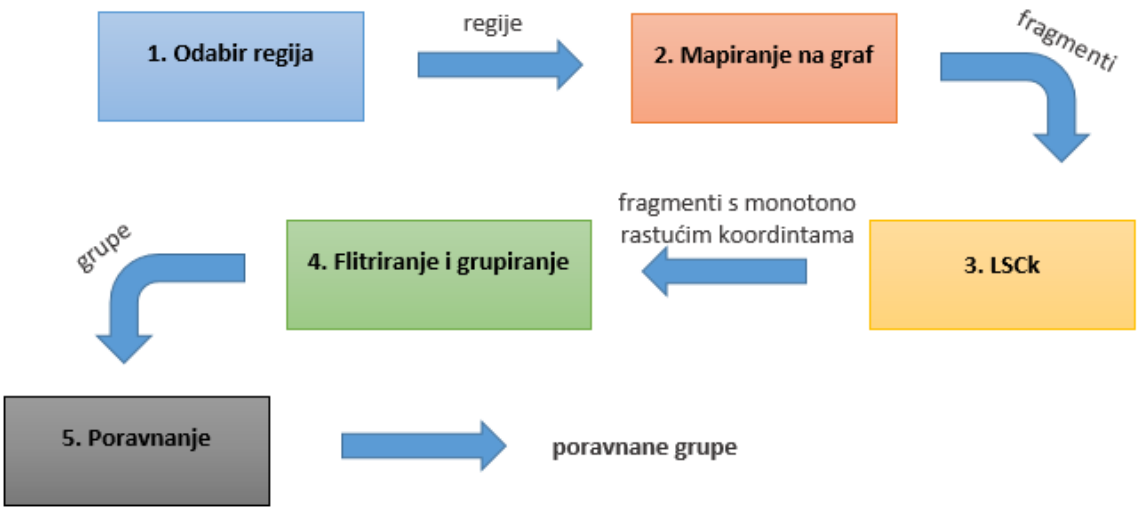
Slika 11. **Vremena izvršavanja knapsack i 2D logaritamske strukture**

5. IMPLEMENTACIJA

Prvotna rješenja implementirana su u Python skriptama u Python 2 verziji. Python je odabran zbog svoje jednostavnosti i velikog broja implementiranih biblioteka od kojih smo koristili *matplotlib*, biblioteku za vizualizaciju podataka.

Pravo rješenje implementirano je u C++-u u C++11 verziji u čemu je implementiran i GraphMap. Rezultat ovog završnog rada jest popunjena metoda *int GraphMap::RNAFilterClusters_(MappingData* mapping_data, const std::vector<Index *> &indexes, const SingleSequence* read, const ProgramParameters* parameters)* u datoteci rna.cc koja se može pronaći na poveznici *https://github.com/ajuric/graphmap/blob/rna/src/graphmap/rna.cc*.

Slika 12. grafički prikazuje faza poravnanja RNA i njihove izlaze. Izlaz pete faze jest rezultat rješenja početnog problema: grupe koje su ušle u poravnanje. Taj izlaz daje navedena metoda.



Slika 12. **Faze cjelokupnog postupka poravnanja**

Sam alat se za poravnavanje pokreće na sljedeći način:

```
$> make modules
$> make –j testing
$> ./graphmap-not_release align –r reference.fa –d reads.fastq –o
output.sam –x rnaseq
```

gdje je *align* modul Graphmapa za poravnanja, *-r* parametar za referencu, *-d* parametar za očitanja, *-o* parametar za izlaznu datoteku te *–x* parametar za pokretanje RNA poravnanja.

6. PODACI

Podaci za provjeru rada rješenja dobiveni su simulacijom. Umjesto pravih očitanja koristimo simulirana zbog toga da bismo mogli provjeriti točnost poravnanja. Kada bismo imali prava očitanja, ne bismo znali je li pojedino očitanje poravnano na pravo mjesto jer nemamo podatak dolazi li ono doista od tamo. Simuliranjem podataka, za svako očitanje znamo s koje pozicije je ono stvoreno pa taj podataka možemo koristiti za provjeru točnosti.

Podaci su simulirani pomoću alata PBSIM ([13]). Alat je namijenjen za simulaciju podataka dobivenih PacBio sekvencerom (jedan od PacBio sekvencera jest PacBioRS II, [14]), no mi ga koristimo zbog toga što su svojstva podataka koje daje PacBio veoma slična svojstvima podataka koje daje MinION. Uz to, PBSIM moguće je konfigurirati tako da simulirani podaci imaju zadana željena svojstva (duljine očitanja, postotak pogreške i slično). Konfiguracija koja odgovara MinION podacima prikazana je u tablici 3.

Tablica 3. **Konfiguracijski paramteri za PBSim**

Svojstvo	Vrijednost	Značenje
--data-type	*CLR*	Continuous Long Read – duga očitanja s većom pogreškom
--depth	*5, 20, 50*	prosječna pokrivenost očitanjima pojedine baze
--model_qc	*model_qc_clr*	model očitanja koji odgovara CLR podacima
--length-mean	*9753*	prosječna duljina očitanja
--length-sd	*4260*	standardna devijacija duljine očitanja
--length-min	*5*	najmanja moguća duljina očitanja
--length-max	*100000*	najveća moguća duljina očitanja
--accuracy-mean	*0.9*	postotak točnosti baza na očitanju
--accuracy-sd	*0.05*	standardna devijacija točnosti baza na očitanju
--accuracy-min	*0.7*	minimalni postotak točnosti baza na očitanju

--diference-ratio	*50:30:20*	udjeli pojedinih pogrešaka - mutacije[4]:umetanja[5]:brisanja[6]

PBSIM također kao ulaznu datoteku prima transkriptom[7] iz kojeg generiramo simulirana očitanja. Iz transkriptoma smo koristili samo *20%* ukupnog broja transkripata. Od svojstava navedenih u tablici 3. valja posebno objasniti svojstvo dubine (*engl. depth)* čije su korištene vrijednosti prilikom simuliranja očitanja *5*, *20* ili *50*. Svojstvo *depth* označava prosječnu pokrivenost svake baze transkripata[8] transkriptoma korištenih u simuliranju podataka.

Ovim simuliranjem dobili smo *4* skupa podataka. Tri skupa nastala su mijenjanjem svojstva *depth* (s vrijednostima *5, 20, 50*), dok četvrti skup podataka jednostavno sadrži sva tri prijašnja skupa. Tablica 4. pokazuje svojstva generiranih skupova podataka (podatak prosječne pokrivenosti baza za *sim4* označava da taj skup podataka sadrži sva tri prijašnja skupa).

Tablica 4. **Skupovi simuliranih podataka**

Skup podataka	Prosječna pokrivenost baza	Broj očitanja
sim1	*5*	*8 340*
sim2	*50*	*69 203*
sim3	*20*	*26 636*
sim4	*5 + 50 + 20*	*104 179*

[4] mutacija – promjena baza
[5] insertion – baza je ubačena
[6] deletion – baza je izbrisana
[7] transkriptom – skup svih transkripata pojedine stanice
[8] transkript – RNA koji se sastoji samo od eksona (introni su izbačeni)

7. REZULTATI

Rezultate poravnanja GraphMapa uspoređivali smo s poravnanjima koje daju postojeći alati za poravnanje: Star i BBMap, svaki s različitim svojstvima i prilagođen za poseban tip očitanja (kraća ili duža očitanja, pogreška mala ili velika i slično).

Alat Star nije namijenjen za rad s podacima kakve proizvodi MinION (duga očitanja s osjetljivom pogreškom $\sim 10\%$), već s kratkim i točnim očitanjima. Problem kod Stara s ovim očitanjima nije njihova duljina, već pogreška koja je za ovaj alat prevelika.

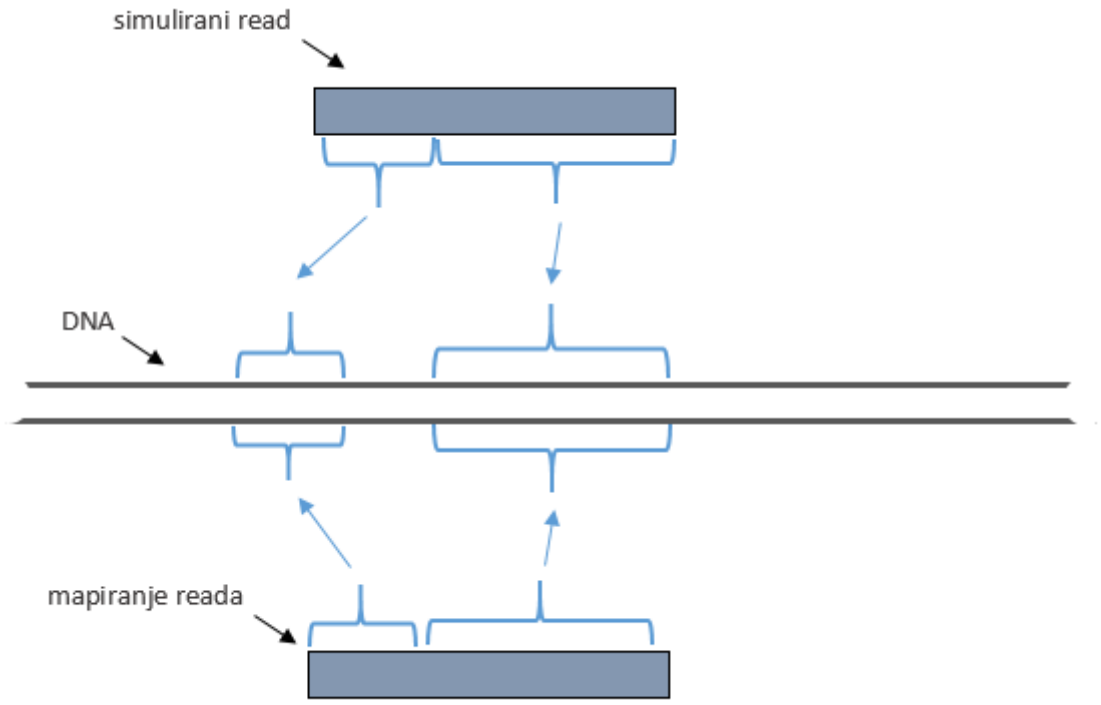
Alat BBMap prilagođen je za rad s očitanjima koja daje PacBio sekvencer, no zbog sličnih postotaka pogreške s podacima koje daje MinION, može se koristiti i za obradu MinION očitanja.. Valja napomenuti da skripta za BBMap za poravnanje dugih očitanja mapPacBio radi na principu da sva očitanja koje dobije ograničava na fiksnu duljinu: reže ih na točno 2000 baza tako da će očitanje, ako ima duljinu veću od 2000, rastaviti na više dijelova.

Značenja podataka prikazanih u tablicama rezultata dana su u tablici 5. Slike od 13. do 17. prikazuju grafički pojedini podatak.

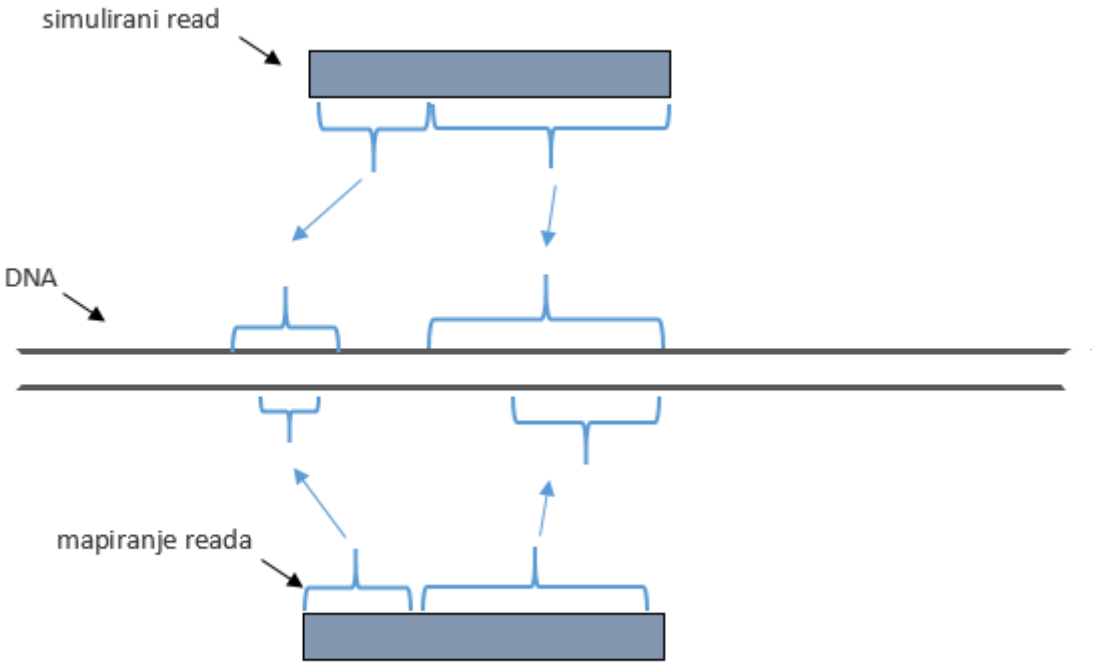
Tablica 5. **Podaci za mjerenje rezultata poravnanja**

Podatak	Značenje
Sva očitanja	
dobra poravnanja	broj očitanja kod kojih je točno pogođen početak i kraj svih dijelova očitanja uz dopuštenu netočnost od 5 baza
pogođeni svi dijelovi	broj očitanja čiji su svi dijelovi pogođeni, tj. dijelovi poravnanja se preklapaju s očitanjima
pogođen barem jedan dio	broj očitanja čiji je barem jedan dio pogođen
točno pogođen barem jedan dio	broj očitanja kod koji je točno pogođen početak i kraj barem jednog dijela očitanja

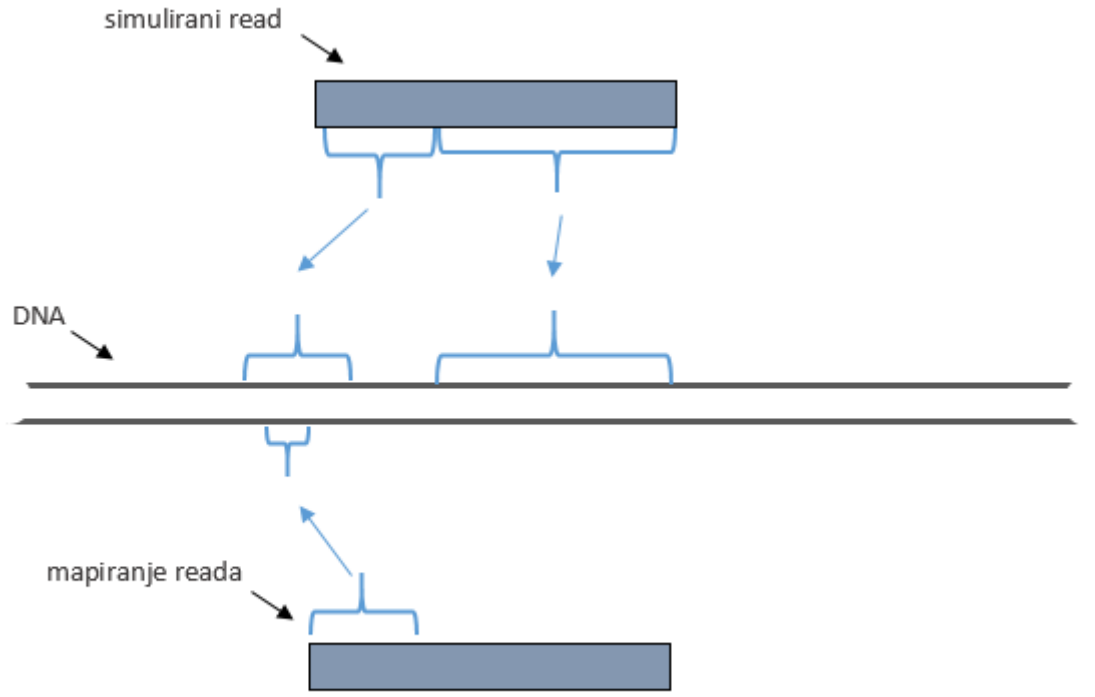
fragmentirana poravnanja	broj očitanja za koje postoji više poravnanja
Isključivo višedijelna očitanja	
broj višedijelnih očitanja	broj očitanja koji imaju više dijelova (višedijelnih očitanja zbog eksona)
pogođeni svi dijelovi višedijelnih očitanja	broj višedijelnih očitanja čiji su svi dijelovi pogođeni
pogođen barem jedan dio višedijelnih očitanja	broj višedijelnih očitanja kod kojih je pogođen barem jedan dio višedijelnog očitanja
točno pogođen barem jedan dio višedijelnog očitanja	broj višedijelnih očitanja kod kojih je točno pogođen početak i kraj barem jednog dijela višedijelnog očitanja
dobra poravnanja višedijelnih očitanja	broj višedijelnih očitanja koja su potpuno dobro poravnana



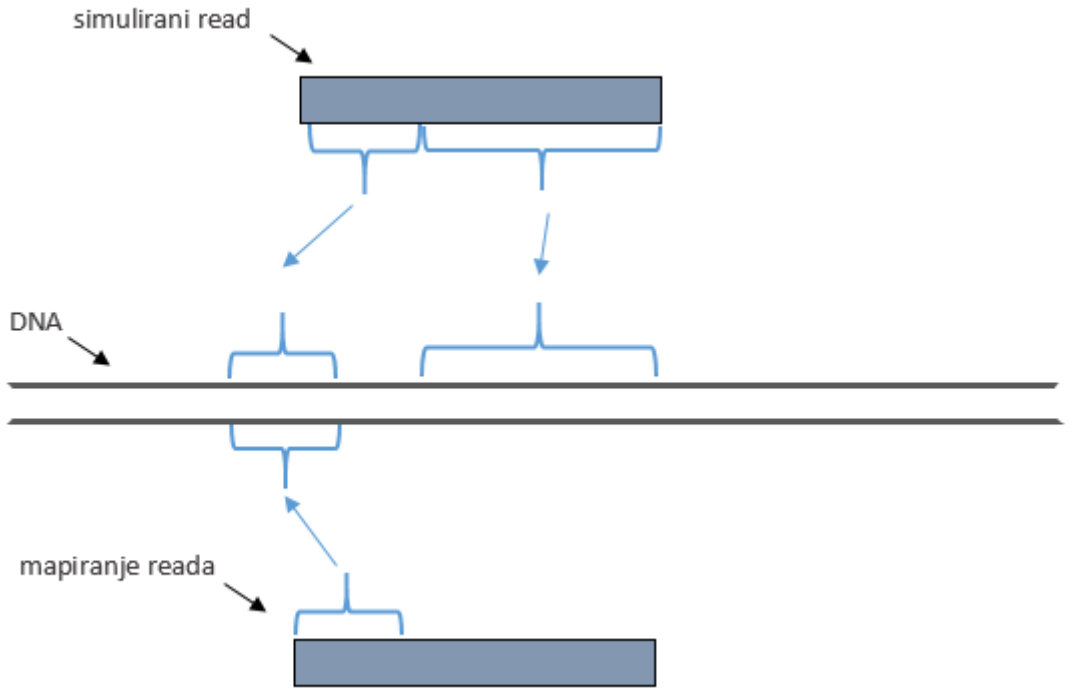
Slika 13. **Primjer za dobra poravnanja**



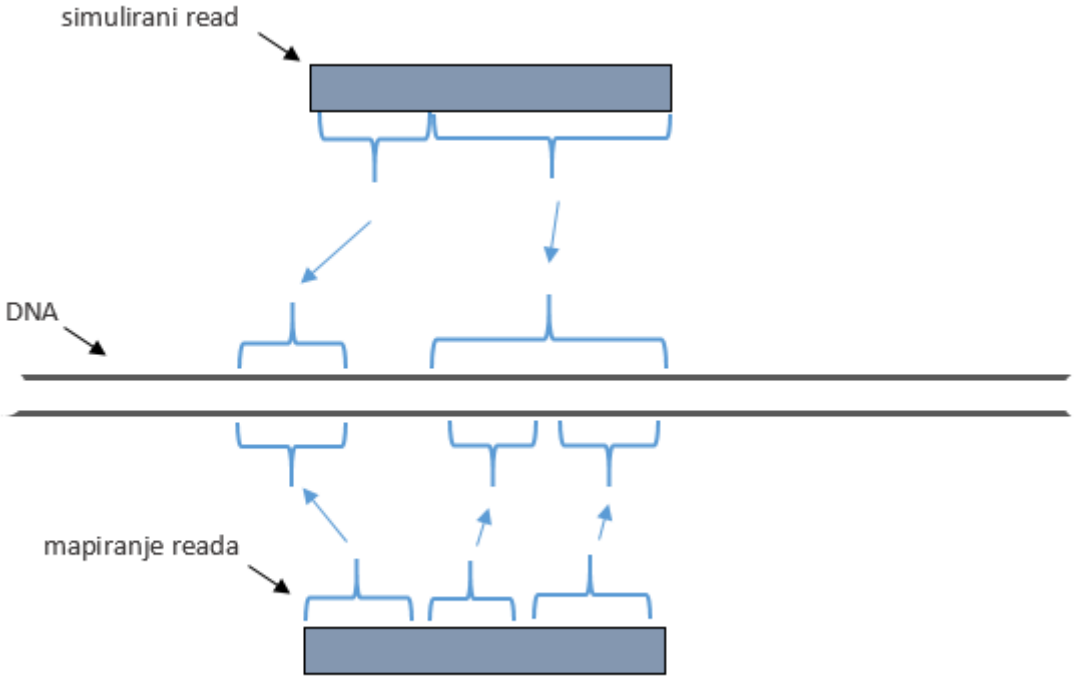
Slika 14. **Primjer za pogođeni svi dijelovi**



Slika 15. **Primjer za pogođen barem jedan dio**



Slika 16. **Primjer za točno pogođen barem jedan dio**



Slika 17. **Primjer za fragmentirana poravnanja**

Tablica 6. **Rezultati mjerenja za alat Star**

Star				
Sva očitanja				
	sim1 (8340 bp)	sim2 (69 203 bp)	sim3 (26 636 bp)	sim4 (104 179 bp)
dobra poravnanja	*267*	*1 202*	*462*	*1 931*
pogođeni svi dijelovi	*472*	*2 542*	*993*	*4 007*
pogođen barem jedan dio	*479*	*2 621*	*1 009*	*4 109*
točno pogođen barem jedan dio	*271*	*1 280*	*491*	*2 042*
fragmentirana poravnanja	*0*	*0*	*0*	*0*
Isključivo višedijelna očitanja				
broj višedijelnih očitanja	*12*	*122*	*37*	*171*
pogođeni svi dijelovi višedijelnih očitanja	*5*	*43*	*20*	*68*
pogođen barem jedan dio višedijelnih očitanja	*12*	*122*	*36*	*170*
točno pogođen barem jedan dio višedijelnog očitanja	*4*	*67*	*25*	*96*
dobra poravnanja višedijelnih očitanja	*0*	*0*	*0*	*0*

Tablica 7. **Rezultati mjerenja za BBMap**

BBMap				
Sva očitanja				
	sim1 (8340 bp)	sim2 (69 203 bp)	sim3 (26 636 bp)	sim4 (104 179 bp)
dobra poravnanja	*4 295*	*34 813*	*13 748*	*52 856*
pogođeni svi dijelovi	*7 420*	*66 451*	*25 546*	*99 411*
pogođen barem jedan dio	*7 499*	*67 301*	*25 806*	*100 600*
točno pogođen barem jedan dio	*4 320*	*35 056*	*13 807*	*53 183*

fragmentirana poravnanja	*0*	*1*	*0*	*1*
Isključivo višedijelna očitanja				
broj višedijelnih očitanja	*247*	*2 258*	*1 005*	*3 510*
pogođeni svi dijelovi višedijelnih očitanja	*162*	*1 365*	*734*	*2 261*
pogođen barem jedan dio višedijelnih očitanja	*241*	*2 215*	*994*	*3 450*
točno pogođen barem jedan dio višedijelnog očitanja	*25*	*243*	*59*	*327*
dobra poravnanja višedijelnih očitanja	*0*	*0*	*0*	*0*

Tablica 8. **Rezultati mjerenja za GraphMap**

GraphMap				
Sva očitanja				
	sim1 (8340 bp)	sim2 (69 203 bp)	sim3 (26 636 bp)	sim4 (104 179 bp)
dobra poravnanja	*2 120*	*17 986*	*6 928*	*27 033*
pogođeni svi dijelovi	*7 181*	*66 185*	*25 416*	*98 782*
pogođen barem jedan dio	*7 306*	*67 512*	*25 899*	*100 717*
točno pogođen barem jedan dio	*2 201*	*18 626*	*7 272*	*28 099*
fragmentirana poravnanja	*818*	*8 061*	*3 160*	*11 985*
Isključivo višedijelna očitanja				
broj višedijelnih očitanja	*252*	*2 270*	*1 011*	*3 533*
pogođeni svi dijelovi višedijelnih očitanja	*117*	*841*	*511*	*1 469*
pogođen barem jedan dio višedijelnih očitanja	*242*	*2 168*	*994*	*3 404*
točno pogođen barem jedan dio višedijelnog očitanja	*81*	*641*	*344*	*1 066*

dobra poravnanja višedijelnih očitanja	*0*	*0*	*0*	*0*

Rezultati u tablicama se interpretiraju na sljedeći način: veći broj je bolji. Iz tih rezultata vidimo da Star ima daleko najslabije rezultate poravnanja iz već navedenih razloga: ne može raditi s podacima koji imaju toliki postotak pogreške.

BBMap trenutno daje bolje rezultate od GraphMapa na jednodijelnim očitanjima, ali uopće ne može prepoznati višedijelna očitanja (koji se sastoje od više eksona) jer nikada ne uzima u obzir granice eksona, već uvijek reže očitanja na duljinu od *2000* baza ako su veća od tog broja.

GraphMap ima slabije rezultate od BBMapa na jednodijelnim očitanjima, ali daje nešto bolje rezultate na višedijelnim očitanjima, tj. općenito bolje pogađa eksone očitanja. GraphMap slabije pogađa početak i kraj dijelova očitanja te ima dosta fragmentiranih očitanja. Razlog za veliki broj fragmentiranih očitanja vjerovatno leži u fazi stvaranja grupa: kriterij za granice intervala grupa mogao bi biti prestrog pa se umjesto jedne veće grupe koja bi pokrila cijeli dio očitanja (ekson) stvori veći broj manjih grupa. Ovaj problem može se pokušati popraviti promjenom parametara kriterija stvaranja grupa i to je područje za budući razvoj cjelokupnog procesa poravnanja RNA GraphMap alata.

8. ZAKLJUČAK

Poravnanje RNA očitanja nije jednostavno što se može vidjeti iz cjelokupnog postupka opisanog u ovom radu: cijeli postupak GraphMapa trenutno se sastoji od pet faza od kojih je svaka posebne složenosti. U ovom radu posebno je razvijana peta faza cjelokupnog procesa: odabir najboljih grupa za poravnanje.

Algoritam koji je razvijen za problem odabira najboljeg podskupa grupa sadrži dva algoritma: knapsack složenosti $O(N^2)$ te 2D logaritamsku strukturu složenosti $O(N \cdot log^2(N))$ gdje je N broj grupa. Pojedini algoritam koristi se ovisno u ukupnom broju grupa i trenutno je granica 800: za broj grupa manji od 800 koristimo knapsack algoritam, u suprotnom koristimo 2D logaritamsku strukturu.

Rezultati poravnanja koje smo dobili s GraphMapom djelomično su slabiji od rezultata BBMapa: BBMap daje puno bolje rezultate za jednodijelna očitanja dok GraphMap daje malo bolje rezultate na višedijelnim očitanjima. Razlog trenutno slabijim rezultatima GraphaMapa vjerovatno treba tražiti u fazama stvaranja grupa i to je područje na kojem je moguće buduće istraživanje: popraviti proces stvaranja grupa.

9. LITERATURA

[1] Bioinformatics, 2016. *https://en.wikipedia.org/wiki/Bioinformatics,* pristupano: 9.6.2016

[2] Knapsack problem, 2016. *https://en.wikipedia.org/wiki/Knapsack_problem*, pristupano: 9.6.2016.

[3] Fenwick tree, 2016. *https://en.wikipedia.org/wiki/Fenwick_tree*, pristupano: 11.6.2016.

[4] Binary indexed trees, 2016. *https://www.topcoder.com/community/data-science/data-science-tutorials/binary-indexed-trees/,* pristupano: 11.6.2016.

[5] Logaritamska, 2012. *http://wiki.xfer.hr/logaritamska/*, pristupano: 11.6.2016.

[6] CPU cache, 2016. *https://en.wikipedia.org/wiki/CPU_cache#Cache_miss*, pristupano: 10.6.2016.

[7] GraphMap, 2016. *https://github.com/isovic/graphmap*, pristupano: 5.6.2016.

[8] GraphMap, 2016. *https://github.com/ajuric/graphmap*, pristupano 16.6.2016.

[9] Nutrigenomika, 2016. *http://www.pbf.unizg.hr/zavodi/zavod_za_biokemijsko_inzenjerstvo/kabinet_za_bioinformatiku/nutrigenomika*, pristupano: 12.6.2016.

[10] Ribonukleinska kiselina, 2016. *https://hr.wikipedia.org/wiki/Ribonukleinska_kiselina,* pristupano: 13.6.2016.

[11] RNA, 2016. *https://en.wikipedia.org/wiki/RNA,* pristupano 13.6.2016.

[12] London calling conference, 26. – 27.5.2016. *https://publications.nanoporetech.com/2016/05/31/clive-brown-cto-of-oxford-nanopore-talks-at-the-london-calling-conference/,* pristupano 13.6.2016.

[13] PBSim, 2016. https://code.google.com/archive/p/pbsim/, pristupano 15.6.2016.

[14] PacBioSystems, 2016. *http://www.pacb.com/products-and-services/pacbio-systems/rsii/*, pristupano: 14.6.2016.

[15] Engström P., Steijger T., Sipos B., Grant G., Kahles A., The RGASP Consortium, Rätsch G., Goldman N., Hubbard T., Harrow J., Guigó R., Bertone R., Systematic evaluation of spliced alignment programs for RNA-seq data, Nature Methods, vol. 10, no. 12, 1184 – 1193

[16] Finotello F., Di Camillo B., Measuring differential gene expression with RNA-seq: challenges and strategies for data analysis, Briefings in functional genomics, vol. 14, no. 2, 130 – 142
[17] Marić J., Long Read RNA-seq Mapper, Diplomski rad, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu, 2015.
[18] Jerzy K. Kulski, Next-Generation Sequencing — An Overview of the History, Tools, and “Omic” Applications, 2016. *http://dx.doi.org/10.5772/61964*, pristupano: 13.6.2016.
[19] Sović, I. *et al*. Fast and sensitive mapping of nanopore sequencing reads with GraphMap. *Nat. Commun.* 7:11307 doi: 10.1038/ncomms11307, 2016
[20] Dobin A. *et al.* STAR: ultrafast universal RNA-seq aligner, Bioinformatics Advance Access, 2012

PORAVNANJE DUGAČKIH RNA OČITANJA

Sažetak

Poravnanje dugačkih RNA očitanja jest problem pronalska s kojeg dijela reference je nastalo dobiveno RNA očitanje. Problem se razlikuje od poravnanja DNA nizova u kojem želimo sastaviti cjelokupni genom zbog posebnih stvojstava RNA koja bitno kompliciraju cijeli proces. U ovom radu opisana je problematika poravnanja nizova RNA. Alat za poravnanje nizova GraphMap problem poravnanja nizova RNA rješava u pet faza: posljednju fazu obrađuje ovaj rad. U toj fazi pokušavamo pronaći najbolje grupe (manje dijelove očitanja) za poravnanje pojedinog očitanja. Algoritam koji je razvijen za problem odabira najboljih grupa sadrži dva algoritma koji se pokreću ovisno u ukupnom broju grupa: knapsack složenosti $O(N^2)$ te 2D logaritamsku strukturu složenosti $O(N \cdot log^2(N))$ gdje je N broj grupa.

Ključne riječi: poravnanje RNA očitanja, knapsack, logaritamska struktura, GraphMap, bioinformatika

LONG RNA-SEQ ALIGNMENT

Abstract

Long RNA-seq alignment is problem of finding from which part of reference RNA read comes from. Problem is different from DNA alignment in which we want to compose whole genome because of special properties of RNA which make whole process more complicated. In this paper process of RNA alignment is described. Tool for alignment GraphMap solves the problem of alignment in five phases: last phase is described in this paper. In that phase we try to find best clusters (smaller parts of reads) for alignment of one read. Algorithm which is developed for problem of selecting best clusters is composed of two algorithms which are used depending on total number of clusters: knapsack with complexity of $O(N^2)$ and 2D logarithmic structure with complexity of $O(N \cdot log^2(N))$ where N is the number of clusters.

Keywords: RNA-seq alignment, knapsack, logarithmic structure, GraphMap, bioinformatics