Vedran Sabadoš

0036422816

PROJEKT

Iz predmeta Bioinformatika

FM-indeks - brojanje pojava upitnog slijeda

Zagreb, siječanj, 2014

Sadržaj

[1. Uvod 1](#_Toc378126206)

[2. Opis rada aplikacije FMIndexCount 2](#_Toc378126207)

[3. Teorija rada 6](#_Toc378126208)

[3.1 Burrows-Wheeler transformacija 6](#_Toc378126209)

[3.2 Binarni niz sa brzim prebrojavanjem 8](#_Toc378126210)

[3.3 Valićno stablo 10](#_Toc378126211)

[3.4 Kompaktni kod 11](#_Toc378126212)

[4. Opis programskih modula aplikacije FMIndexCount 12](#_Toc378126213)

[4.1 Klasa FMIndexCountFrame 12](#_Toc378126214)

[4.2 Klasa ReferenceSequence 12](#_Toc378126215)

[4.2.1 Konstruktor 12](#_Toc378126216)

[4.2.2 Funkcija saveReferenceSequenceIndex 14](#_Toc378126217)

[4.2.3 Funkcija countQuerySequenceOccurrences 15](#_Toc378126218)

[4.3 Klasa ReferenceSequenceIndex 15](#_Toc378126219)

[4.4 Klasa AlphabetDictionary 15](#_Toc378126220)

[4.5 Klasa FMIndexWaveletTreeNode 16](#_Toc378126221)

[4.5.1 Konstruktori 16](#_Toc378126222)

[4.5.2 Funkcija Occ 16](#_Toc378126223)

[4.5.3 Funkcija getMemoryConsumption 16](#_Toc378126224)

[4.6 Klasa FMIndexBitString 17](#_Toc378126225)

[4.6.1 Konstruktor 17](#_Toc378126226)

[4.6.2 Funkcije setBit i resetBit 17](#_Toc378126227)

[4.6.3 Funkcije setBitNoBucket i resetBitNoBucket 17](#_Toc378126228)

[4.6.4 Funkcija refreshBuckets 17](#_Toc378126229)

[4.6.5 Funkcija rank1 17](#_Toc378126230)

[4.6.6 Funkcija getMemoryConsumption 17](#_Toc378126231)

[5. Testiranje i rezultati mjerenja 18](#_Toc378126232)

[5.1 Testiranje 18](#_Toc378126233)

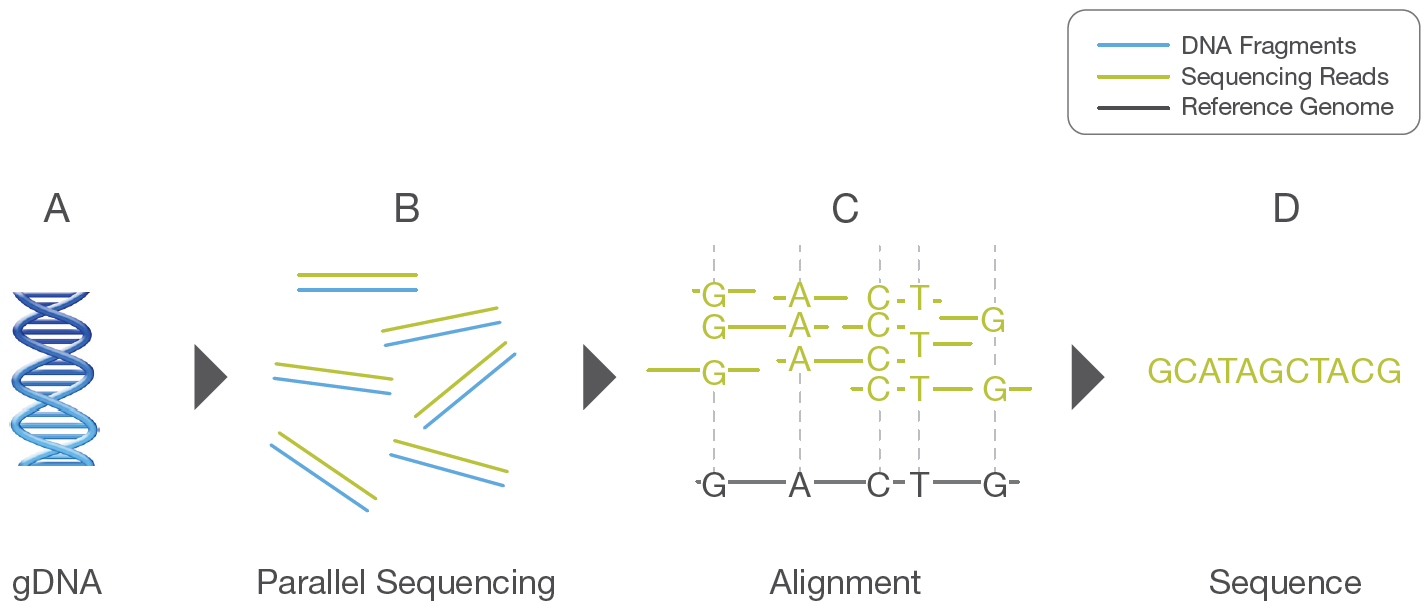
[5.2 Rezultati mjerenja 18](#_Toc378126234)

[6. Zaključak 21](#_Toc378126235)

[7. Literatura 22](#_Toc378126236)

# Uvod

Nakon otkrića da ključnu ulogu u nasljeđivanju svojstava živih organizama igra dezoksiribonukleinska kiselina (DNA), počelo je njeno intenzivno proučavanje. Koristi od tog istraživanje su nesumnjive, počevši od medicine, pa sve do forenzike. Kako su molekule DNA vrlo dugačke, njihovo istraživanje uvijek je bilo praćeno potrebom za odgovarajućim informatičkim alatima. Nakon usvajanja tehnologije druge generacije sekvenciranja (Next-Generation Sequencing), glad za pratećom informatičkom podrškom postala je još mnogo veća. Princip određivanja slijeda nukleotida ovom metodom prikazan je na slici 1.1. Najprije se cijeli genom (A) razbije na slučajan način u veliki broj malih fragmenata (B) koji se zatim očitavaju paralelnim postupkom. U jednom prolazu moguće je paralelno obrađivati više tisuća, pa i miliona fragmenata. Ta očitanja se zatim elektronički obrađuju i uparuju (C) da bi se na kraju dobio polazni genom (D). Jasno je da je u postupku rekombinacije potrebno uložiti znatnu procesorsku snagu, odnosno razviti odgovarajuće algoritme i alate kako bi elektronička obrada mogla pratiti brzinu prikupljanja uzoraka. Jedan od tih alata je pronalaženje odnosno brojanje ponavljanja manjeg slijeda nukleotida unutar velikog referentnog slijeda, a jedan od algoritama koji se za to koristi je upravo FM-indeks.



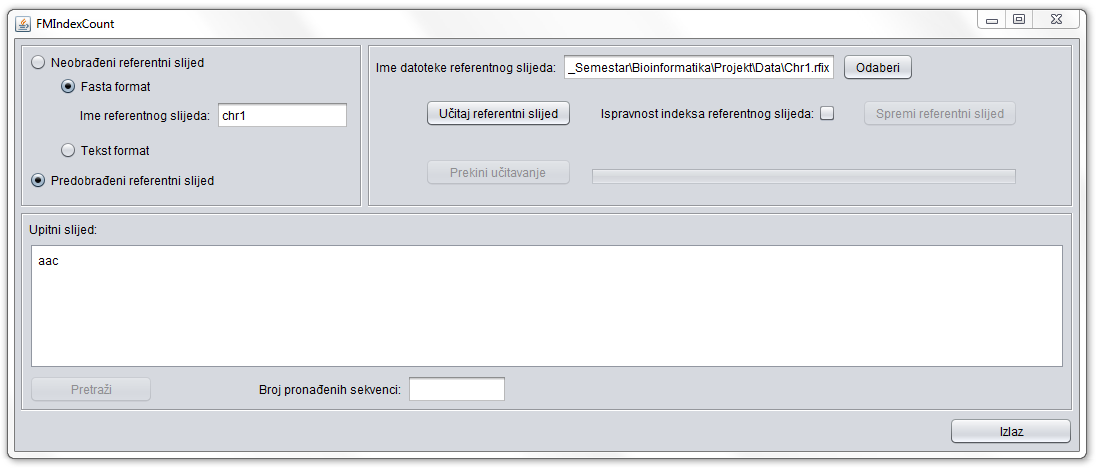
Slika 1.1 – princip sekvenciranja genoma metodom druge generacije

Indeksiranje je česta strategija prilikom pretraživanja i traženja određenih uzoraka u tekstu. Kada se radi o velikim tekstovima, javlja se potreba za komprimiranjem ne samo teksta već i pripadnog indeksa. FM-indeks sam po sebi vrši kompresiju i podataka i indeksa istovremeno.

U okviru ovog projekta, potrebno je izraditi alat za prebrojavanje ponavljanja zadanog slijeda u referentnom slijedu.

# Opis rada aplikacije FMIndexCount

Aplikacija za prebrojavanje ponavljanja upitnog slijeda u referentnom slijedu naziva se FMIndexCount. Napisana je u programskom jeziku Java, a njeno korisničko sučelje realizirano je u obliku forme i prikazano je na slici 2.1.

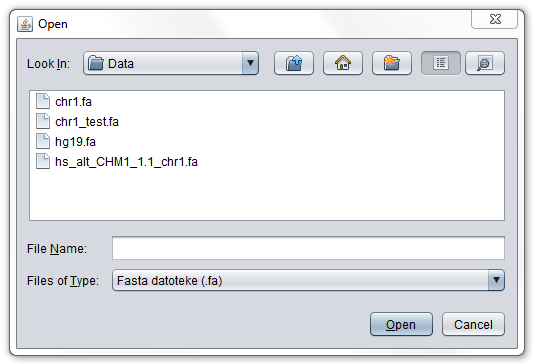


Slika 2.1 – Korisničko sučelje programa FMIndexCount

Aplikacija ima dva osnovna dijela. Prvi dio vezan je uz učitavanje i kreiranje indeksa referentnog slijeda, i uz njega su vezane gornje dvije grupe kontrola. Drugi dio namijenjen je osnovnoj funkciji aplikacije, a to je pronalaženje i prebrojavanje ponavljanja upitnog slijeda u referentnom slijedu. Uz ovaj dio aplikacije vezane su kontrole iz donje grupe.

Referentni slijed učitava se iz datoteke čije se ime upisuje u tekstualni okvir pod imenom "Ime datoteke referentnog slijeda". Ime datoteke može se upisati pomoću tipkovnice ili odabrati pritiskom na kontrolu "Odaberi". Kontrola "Odaberi" prikazuje standardni Java-in dijalog prozor za otvaranje datoteka prikazan na slici 2.2. Odabirom datoteke, njezino puno ime upisuje se u tekstualni okvir "Ime datoteke referentnog slijeda".

Referentni slijed može biti zadan u "sirovom" tekstualnom obliku, koji je potrebno prije upotrebe obraditi kako bi se dobio indeks referentnog slijeda, ili kao već pripremljeni indeks koji se može odmah koristiti. Za odabir tipa datoteke referentnog slijeda služi lijeva gornja grupa kontrola forme. Ako je izabrana kontrola "Predobrađeni referentni slijed", program učitava datoteku sa već pripremljenim indeksom referentnog slijeda. Datoteke ovog tipa imaju "default" ekstenziju ".rfix". U protivnom, ako je izabrana kontrola "Neobrađeni referentni slijed", učitava se datoteka referentnog slijeda u tekstualnom obliku.

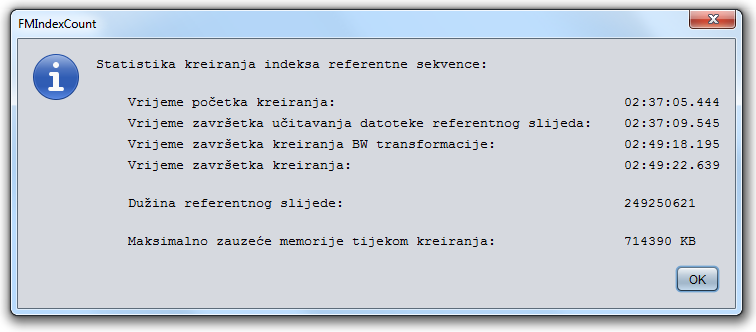


Slika 2.2 – Dijalog prozor za otvaranje datoteka

Kada se referentni slijed učitava u tekstualnom obliku, ponovno je moguć izbor između dva tipa ulaznih datoteka. Ako je izabrana kontrola "Fasta format", aplikacija očekuje da ulazna datoteka ima fasta format. Ovaj format datoteke dolazi iz programskog paketa "Fasta", ali je sada postao standard na području bioinformatike. U sljedovima nukleotida ili peptida, pojedini nukleotidi ili aminokiseline prikazuje se jednim jedinim znakom. Datoteka može sadržavati više sljedova, a svaki od njih je prethođen nazivom slijeda. Naziv željenog slijeda upisuje se u tekstualni okvir "Ime referentnog slijeda". Datoteke ovog tipa imaju "default" ekstenziju ".fa". Aplikacija pretvara mala slova iz ulaznog slijeda u velika.

Izborom kontrole Tekst format odabire se opći oblik tekstualne datoteke. Smatra se da je datoteka kodirana UTF-8 kodom, odnosno ASCII kodom, a dozvoljeni su znakovi u rasponu od 32 do 126, to jest znakovi koji se mogu prikazati na zaslonu odnosno otisnuti na papir (eng. printable characters). U ovoj inačici aplikacije, nedozvoljeni, kontrolni znakovi naprosto se izbacuju iz teksta, dok se mala slova pretvaraju u velika. Ovakva obrada ulaznog slijeda prilično je dvojbena, i prvenstveno ovisi o pravoj svrsi aplikacije, ali kako je ovaj projekt rađen u sklopu predmeta "Bioinformatika", nisam dublje ulazio u problematiku. Datoteke ovog tipa imaju "default" ekstenziju ".txt".

Proces učitavanja, odnosno kreiranja, indeksa referentnog slijeda započinje pritiskom na kontrolu "Učitaj referentni slijed". Kako ovaj proces može biti dugotrajan, naročito kada se kreira indeks vrlo dugih tekstualnih datoteka, aktivira se kontrola za prekid: "Prekid učitavanja", a stupanj gotovosti prikazuje se na traci napretka "progress bar". Pred kraj obrade, na zaslonu se pojavljuje dijalog prozor u kojem su prikazani podaci vezani uz proces kreiranja indeksa referentnog slijeda. Tu su prikazana vremena početka i završetka kreirana, vremena završetka pojedinih faza, dužina referentnog slijeda, kao i količina utrošene memorije tijekom obrade. Ovaj prozor prikazan je na slici 2.3.

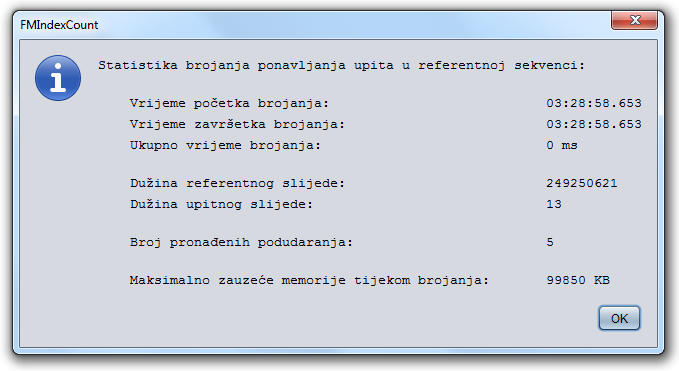


Slika 2.3 – Statistika kreiranja indeksa referentnog slijeda.

Nakon uspješnog završetka učitavanja, odnosno kreiranja, indeksa referentnog slijeda, u okviru "Ispravnost indeksa referentnog slijeda" pojavljuje se kvačica kao oznaka postojanja ispravnog indeksa. Istovremeno se omogućava kontrola "Spremi referentni slijed". Pritiskom na ovu kontrolu ponovo se otvara Java-in dijalog prozor za otvaranje datoteka putem kojeg se može odrediti mapa i naziv datoteke. Indeks referentnog slijeda sprema se pritiskom na kontrolu "Save".

Grupa kontrola vezana za upitni slijed, koja se nalazi u donjem dijelu forme, blokirana je dok aplikacija nema ispravan indeks referentnog slijeda. Ove kontrole deblokiraju se nakon što se u okviru "Ispravnost indeksa referentnog slijeda" pojavi kvačica. Upitni slijed unosi se putem tipkovnice ili "Copy/Paste" procedurom u tekstualni okvir za unos višerednog teksta "Upitni slijed", a pretraga započinje pritiskom na kontrolu "Pretraži". Nakon završetka pretrage, prikazuje se dijalog prozor prikazan na slici 2.4 u kojem su prikazani podaci vezani uz tu pretragu. Tu se nalaze vremena početka i završetka pretrage, odnosno trajanje pretrage, dužine referentnog i upitnog slijeda, broj pronađenih podudaranja, te maksimalno zauzeće memorije tijekom pretrage. Broj pronađenih podudaranja upisuje se i u tekstualni okvir "Broj pronađenih podudaranja" glavne forme.

Aplikacija završava s radom pritiskom na kontrolu "Izlaz" glavne forme.

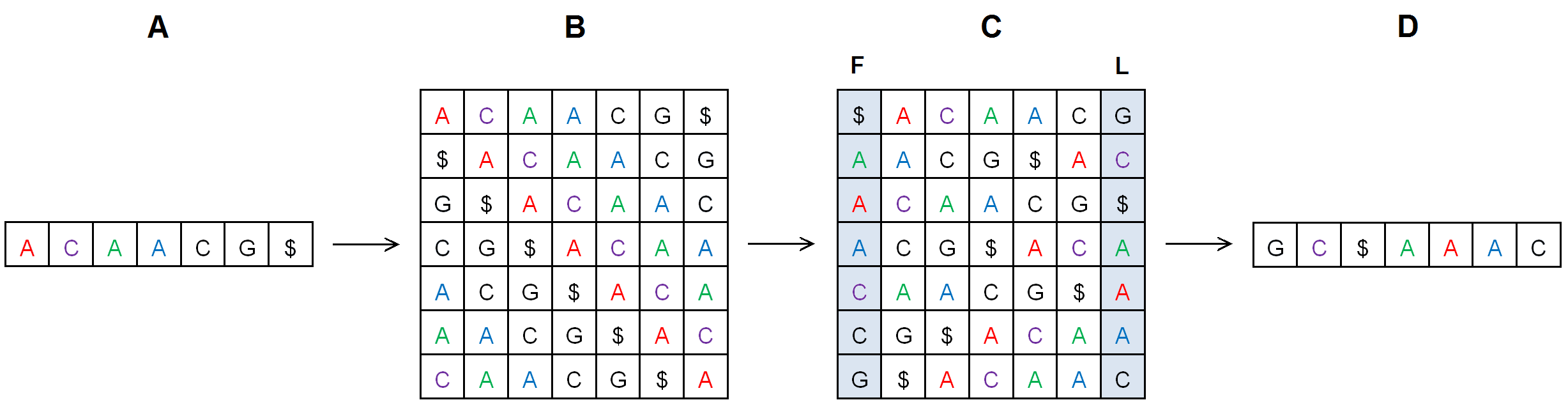


Slika 2.4 – Statistika pretrage za upitni slijed.

# Teorija rada

## Burrows-Wheeler transformacija

Burrows-Wheeler transformacija ili skraćeno BWT pokazala se kao vrlo važan postupak u različitim obradama tekstualnih podataka. Postupak dobivanja BW transformacije iz tekstualnog slijeda prikazan je primjerom na slici 3.1.1.

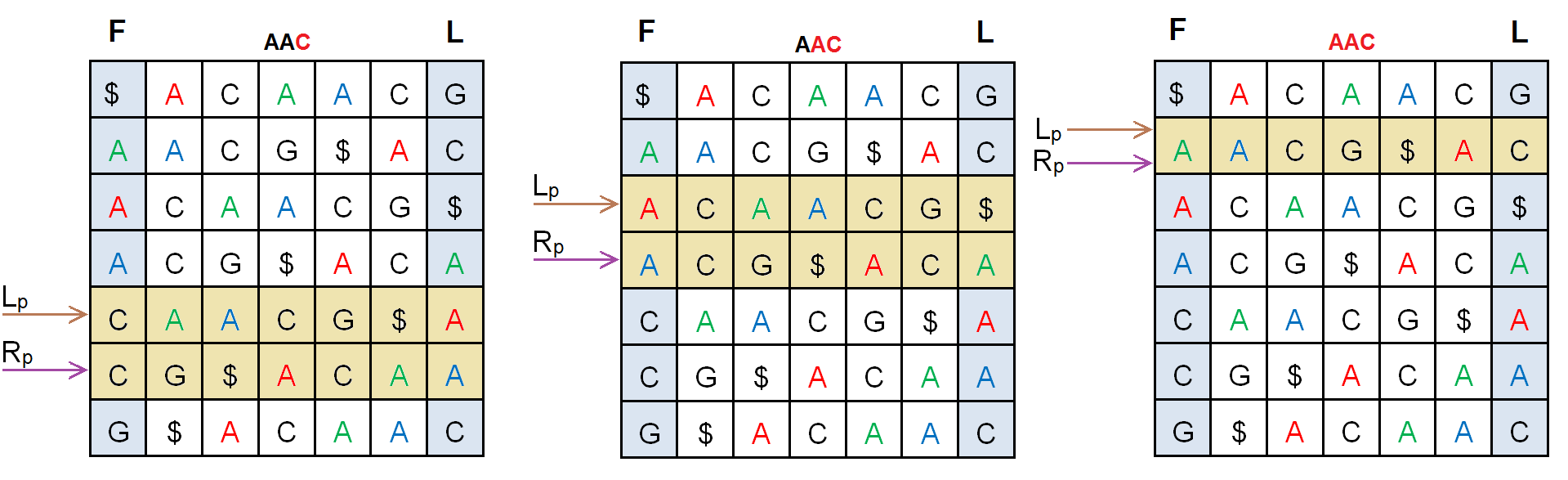


Slika 3.1.1 – Postupak dobivanja Burrows-Wheler transformacije

Od ulaznog tekstualnog slijede (A) naprave se sve moguće cikličke rotacije i poredaju jedna ispod druge tako da čine tablicu prikazanu pod (B). Zatim se cikličke rotacije poredaju abecednim redom tako da se dobije tablica sortiranih cikličkih rotacija (C). Pri tome znak "$", koji je umetnut na kraj ulaznog slijeda, ima najmanju vrijednost u odnosu na sve druge znakove u zadanoj abecedi. Zadnji stupac u dobivenoj strukturi (označen sa L) predstavlja Burrows-Wheeler transformaciju ulaznog slijeda (D).

Može se pokazati da je za efikasno brojanje podudaranja upitnog slijeda sa referentnim slijedom dovoljno poznavati prvi (označen sa F) i zadnji (označen sa L) stupac tablice sortiranih cikličkih rotacija. Za to se koriste dva važna svojstva ove tablice. Zbog toga što tablicu čine cikličke rotacije ulaznog slijeda, znak u zadnjem stupcu tablice neposredno prethodi znaku iz prvog stupca u ulaznom slijedu. Drugo svojstvo govori da je redoslijed pojavljivanja istog znaka u prvom i zadnjem stupcu sačuvan. To znači da se ljubičasti znak "C" (onaj koji se nalazi između crvenog i zelenog znaka "A") nalazi ispred crnog znaka "C" i u prvoj i u zadnjoj koloni sa slike 3.1.1. Isto vrijedi i za crveni, zeleni i plavi znak "A".

Postupak brojanja podudaranja upitnog slijeda sa referentnim slijedom pokazan je primjerom sa slike 3.1.2, gdje se broje podudaranja upitnog slijeda "AAC" sa referentnim "ACAACG$" . Postupak započinjemo posljednjim znakom iz upitnog slijeda, znakom "C". Zbog sortiranosti tablice sortiranih cikličkih rotacija, sve cikličke rotacije koje započinju znakom "C" nalaze se jedna do druge. Prvoj od tih rotacija pridružuje se indeks Lp, dok se zadnjoj pridružuje indeks Rp. Kad bi se upitni slijed sastojao samo od jednog znaka C, postupak bi bio gotov, a rezultat bi se dobio kao Rp ‑ Lp + 1.



Slika 3.1.2 – Primjer brojanja ponavljanja upitnog slijeda

Kako znaku "C" u upitnom slijedu prethodi znak "A", iskoristit ćemo svojstvo posljednjeg stupca koje kaže da se u njemu nalaze znakovi koji neposredno prethode znakovima iz prvog stupca. Vidimo da se u cikličkim rotacijama koje započinju znakom "C" (u prvoj tablici rotacije označenim smeđom bojom), u zadnjoj koloni nalaze crveni i plavi znak "A". To znači da su cikličke rotacije koje započinju crvenim i plavim znakom "A" ujedno i jedine koje započinju slijedom "AC". Rotacija koja započinje zelenim znakom "A" ne započinje sa slijedom "AC", već sa slijedom "AA", pa nas ne zanima.

U drugom koraku (srednja tablica) želimo izolirati cikličke rotacije koje započinju slijedom "AC". To radimo tako da indekse Lp i Rp postavimo da pokazuju na prvi, odnosno posljednji znak "A" u prvoj koloni tablice. Zatim prebrojimo u prethodnoj tablici koliko se puta znak A pojavljuje u posljednjem stupcu prije pripadnog indeksa Lp. To je upravo zeleni znak A koji započinje slijed "AA". Kako je redoslijed znakova u prvoj i zadnjoj koloni sačuvan, prebrojeni znakovi (u našem slučaju to je samo zeleni znak "A") nalaze se na početku niza rotacija izabranih u drugom koraku, pa za taj iznos povećavamo vrijednost novog indeksa Lp. Isto tako, potrebno je prebrojati i znakove "A" koji se nalaze u zadnjoj koloni prethodne tablice iza indeksa Rp, pa za taj iznos smanjiti novi indeks Rp. Kako u našem slučaju nema takvih znakova, vrijednost novog indeksa Rp ostaje nepromijenjena. Vidimo da sada indeksi Lp i Rp pokazuju na prvu, odnosno posljednju cikličku rotaciju koja započinje sa slijedom "AC".

Ova procedura ponavlja se tako dugo dok ne dođemo do prvog znaka upitnog niza. U našem slučaju, to je treći korak u kojem izoliramo cikličke rotacije koje započinju sa slijedom "AAC". U prethodnoj tablici prebrojimo znakove "A" koji se nalaze u zadnjem stupcu prije, odnosno poslije cikličkih rotacija označenih pripadnim indeksima Lp i Rp. To su u našem slučaju crveni i plavi znak "A", koji se nalaze iza indeksa Rp iz prethodnog koraka, pa za taj iznos smanjujemo novi indeks Rp. Vidimo da sad oba indeksa pokazuju na jedinu postojeću cikličku rotaciju koja započinje slijedom "AAC". I ovaj put broj ponavljanja upitnog slijeda u referentnom slijedu dobije se kao Rp ‑ Lp + 1. Ukoliko se upitni slijed ne nalazi niti jednom u referentnom slijedu, Lp će biti veći od Rp i u tom slučaju rezultat je 0.

Iz prethodnog primjera vidjeli smo da je za brojanje podudaranja dva tekstualna slijeda dovoljno imati informacije o prvom i posljednjem stupcu tablice sortiranih cikličkih rotacija. Pošto su u prvom stupcu jednaki znakovi grupirani zajedno i sortirani po abecedi, on se može sažeti u tablicu koja pamti indekse prvog pojavljivanja nekog znaka, ili točnije rečeno, ukupan broj znakova u početnom slijedu koji su po abecedi manji od nekog znaka abecede. Ta se tabela zove tabela suma prefiksa (eng. prefix-sum-table). Već smo prije vidjeli da je zadnji stupac tabele sortiranih cikličkih rotacija u stvari Burrows-Wheeler transformacija. Ona se može i dalje mijenjati ovisno o primjeni. U našem slučaju, ona se dalje transformira u valićno stablo (eng. wavelet tree) bazirano na binarnom nizu koji omogućava brzo prebrojavanje bitova postavljenih u jedinicu (eng. rank support bit string). Općenito, podatkovna struktura koja čuva informaciju o broju pojavljivanja nekog znaka do zadanog indeksa, zove se tabela pojavljivanja (eng. occurrence table).

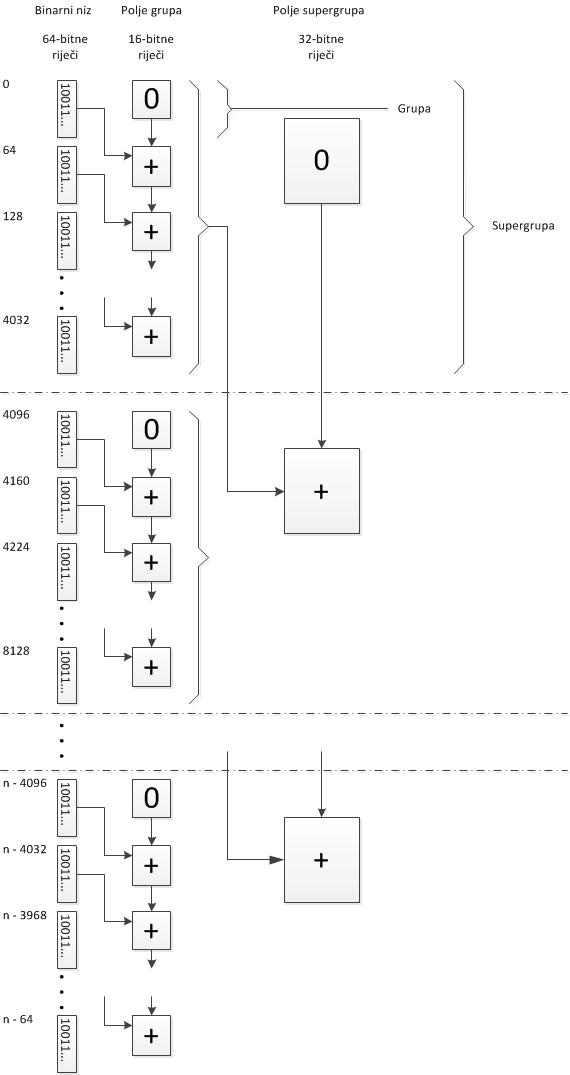
## Binarni niz sa brzim prebrojavanjem

Binarni niz sa brzim prebrojavanjem (eng. Rank support bit string) ubraja se u jezgrovite podatkovne strukture (eng. succinct data structure), odnosno podatkovne strukture koje omogućavaju komprimiranje podataka ali i korištenje tih podataka bez prethodne dekompresije. U našem slučaju, zahtijeva se postojanje funkcije rankq(x) koja vraća broj elemenata q (q {0, 1}) do pozicije x u konstantnom vremenu, neovisno o veličini binarnog niza. Da bi to bilo moguće, pored samog binarnog niza podatkovna struktura u našem slučaju sadrži dodatne podatke i prikazana je na slici 3.2.1.

Podaci koje čuva binarni niz (niz nula i jedinica) podijeljeni su u grupe (eng. bucket) od po 64 bita i pohranjeni u polje 64-bitnih riječi (longword). Binarni je niz dodatno podijeljen u super grupe (eng. super bucket) koje se sastoje od po 64 grupe, odnosno 4096 bita. Za svaku grupu od 64 bita, postoji pripadna 16-bitna riječ, koja se čuva u polju grupa, a koja sadrži zbroj svih jedinica pohranjenih u binarnom nizu od početka super grupe, pa do uključujući grupe koja prethodi trenutnoj grupi. To znači da unutar super grupe, 1. riječ u polju grupe sadrži 0 (nema prethodne grupe), 2. riječ sadrži zbroj jedinica u 1. grupi, 3. riječ sadrži zbroj jedinica u 1. i 2. grupi, dok posljednja, 64. riječ sadrži zbroj jedinica u prve 63 grupe.

Polje super grupa sastoji se od 32-bitnih riječi, a svaka od njih čuva ukupan zbroj svih jedinica u svim prethodnim super grupama od početka binarnog niza. Slično kao i u polju grupa, prvi podatak u polju sadrži 0 jer nema prethodnih super grupa.

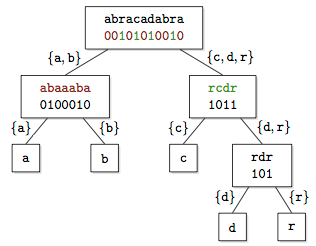
Uz ovakvu strukturu podataka, dohvaćanje broja jedinica od početka binarnog niza, pa do zadane pozicije vrlo je jednostavno i brzo. Naprosto treba zbrojiti pripadni podatak u polju super grupa i u polju grupa, te tome dodati broj jedinica od početka pripadne grupe binarnog niza, pa do zadane pozicije. Za brojanje jedinica unutar grupe binarnog niza koristi se popcount algoritam (eng. population count) koji je ugrađen u statičku funkciju java.lang.Long.bitCount, a koji se odvija u konstantnom vremenu.



Slika 3.2.1 – Podatkovna struktura binarnog niza sa brzim prebrojavanjem

## Valićno stablo

Kao što je u poglavlju 3.1 rečeno, Burrows-Wheeler transformacija referentnog slijeda može se i dalje transformirati u svrhu komprimiranja podataka i omogućavanja brzog prebrojavanja pojedinih znakova. Podatkovna struktura koju smo upotrijebili u ovom projektu je valićno stablo (eng. wavelet tree), bazirano na binarnom nizu sa brzim prebrojavanjem. Valićno stablo isto spada u grupu jezgrovitih podatkovnih struktura jer omogućava prebrojavanje pojedinih znakova bez prethodne dekompresije. Valićno stablo prikazano je na slici 3.3.1.



Slika 3.3.1 – Valićno stablo

Način na koji se tekstualni slijed čuva u valićnom stablu najbolje je objasniti na primjeru. U primjeru sa slike 3.3.1, znakovi tekstualnog slijeda "abracadabra" u korijenskom čvoru dijele se u dvije grupe i to u grupu sa leksikografski nižim (a, b) i u grupu sa leksikografski višim (c, d, r) znakovima. Granica među grupama može biti ovisna o primjeni, ali najčešće se postavlja tako da u obje grupe bude podjednaki broj znakova. U tom slučaju stablo se naziva balansiranim. Pripadajući binarni niz dugačak je onoliko koliko je dugačak i tekstualni slijed koji se pohranjuje. Znakovi iz prve grupe kodiraju se sa "0", dok se znakovi iz druge grupe kodiraju sa "1". Ukoliko pojedina grupa sadrži više od jednog znaka, formira se nova grana sa znakovima iz samo te grupe. U novom čvoru znakovi imaju međusobni redoslijed isti kao i u originalnom tekstu. Procedura se ponavlja tako dugo dok se broj znakova u nekoj grupi ne smanji na 1. U tom slučaju nije potrebno formirati novu granu jer se do broja znakova može doći prebrojavanjem znakova grupe.

Kako u pozadini svakog čvora stoji binarni niz sa brzim prebrojavanjem, vrijeme prebrojavanja znakova pojedinog čvora je konstantno. Međutim da se prebroji broj ponavljanja nekog znaka, potrebno se spustiti kroz pojedine grane do krajnjeg čvora. Broj prolaznih čvorova ovisi o bruju znakova abecede ||, pa u slučaju balansiranog stabla, vremenska je kompleksnost O(log ||).

## Kompaktni kod

Kada se referentni slijed učitava u tekstualnom obliku (bilo u fasta formatu, bilo u općem tekstualnom formatu), aplikacija prihvaća kodove između 32 i 126, odnosno tekst kodiran ASCII kodom, ili UTF-8 inačicom unicod-a koji se u prvih 128 znakova poklapa sa ASCII kodom. To znači da će u velikoj većini slučajeva postojati velike praznine između upotrijebljenih kodova. Radi jednostavnije kasnije obrade, ulazni se tekst na samom početku obrade prekodira u "kompaktni" kod koji počinje sa vrijednošću "1" i nema praznina između upotrijebljenih kodova. Pri tom je očuvan leksikografski odnos među znakovima. Kod "0" rezerviran je za znak kojim se označava kraj referentnog slijeda. Vrijednosti kodova računaju se za svaki učitani referentni slijed, a rječnik abecede čini dio strukture podataka koja se pamti zajedno s tabelom suma prefiksa i tabelom pojavljivanja.

Kako je kod procesiranja pojedinog upita potrebno na isti način prekodirati i upitni slijed, na prvi se pogled čini da je to nepotreban posao u trenutku kada se traži brzina. Međutim dobra praksa programiranja nalaže provjeru ispravnosti i upitnog slijeda prije obrade, a prekodiranje predstavlja vrlo efikasan način da se to provede. Stoga je upotreba kompaktnog koda opravdana i u ovom slučaju.

# Opis programskih modula aplikacije FMIndexCount

Aplikacija FMIndexCount izvedena je u programskom jeziku Java. To je objektno orijentiran programski jezik, baziran na klasama i dizajniran tako da aplikacije mogu raditi na različitim platformama bez potrebe za novim kompiliranjem . Java kompajler prevodi aplikaciju u "bytecode", set instrukcija koje onda izvodi "Java virtual machine" (JVM), prilagođena za rad na platformi na kojoj se aplikacija izvodi.

U sklopu aplikacije razvijeno je nekoliko klasa koje čine funkcionalnost sustava:

* FMIndexCountFrame – korisničko sučelje
* ReferenceSequence – funkcionalnost vezana uz referentni slijed
* ReferenceSequenceIndex – podaci potrebni za obradu upitnog slijeda
* AlphabetDictionary – funkcionalnost vezana uz abecedu referentnog slijeda
* FMIndexWaveletTreeNode – funkcionalnost čvora valićnog stabla
* FMIndexBitString – funkcionalnost binarnog niza sa brzim prebrojavanjem

## Klasa FMIndexCountFrame

Klasa FMIndexCountFrame nasljeđuje klasu javax.swing.JFrame i daje funkcionalnost za korisničko sučelje koje je izvedeno u obliku forme. Kroz dizajn klase, na glavnu formu programa postavljene su kontrole čija je funkcionalnost opisana u poglavlju 2. Pored funkcija koje obrađuju događaje na pojedinim kontrolama, klasa sadrži i statičku funkciju "main" kojom započinje izvođenje aplikacije. Klasa instancira i objekt tipa ReferenceSequence preko kojeg ima pristup referentnom slijedu i njegovim funkcijama.

## Klasa ReferenceSequence

Kroz ovu ostvarena je funkcionalnost referentnog slijeda, uključujući njegovo kreiranje, pohranjivanje, te obradu upitnih zahtjeva.

### Konstruktor

Klasa ima javni konstruktor koji kao parametre prihvaća ime datoteke referentnog slijeda, njen tip te ime slijeda koje se koristi samo kada je datoteka u fasta formatu. Konstrukcija objekta započinje kreiranjem objekt **referenceSequenceIndex** tipa **ReferenceSequenceIndex** koji će na kraju sadržavati sve podatkovne strukture potrebne za obradu upitnih zahtjeva. Ti su podaci izdvojeni u poseban objekt kako bi se olakšala njihova pohrana. Nakon toga započinje učitavanje iz datoteke referentnog slijeda. Taj se posao obavlja u funkcijama **preprocssFastaReferenceSequence**, **preprocssTextReferenceSequence** ili **readPreprocssedReferenceSequence** ovisno o tipu ulazne datoteke. Ako se radi o predprocesiranoj datoteci, nakon učitavanja **ReferenceSequence** objekt je uspješno kreiran. Tijekom učitavanja datoteke u fasta ili tekst formatu, stvara se i rječnik abecede **alphabetDictionary** koji je sastavni dio objekta **referenceSequenceIndex**. Podaci se učitavaju u polje **compactCodeBuffer**,a nakon završetka se prekodiraju u kompaktni kod.

Nakon učitavanja, referentni slijed se podvrgava Burrows-Wheeler transformaciji, a podaci se pohranjuju u polju **bwtBuffer**. Tijekom transformacije stvara se i polje suma prefiksa **prefixSumTable** koje je sastavni dio objekta **referenceSequenceIndex**.

Zadnji korak u kreaciji referentnog slijeda je komprimiranje BW transformacije u valićno stablo **occurrenceWaveletTree** iz objekta **referenceSequenceIndex**. Konstrukcija objekta završava prikazivanjem statistike kreiranja indeksa referentnog slijeda koja sadrži vremena značajnijih faza, dužinu referentnog slijeda te procjenu zauzeća memorije tijekom kreacije.

#### Privatna funkcija **createBwt**

Funkcija **createBwt** provodi Burrows-Wheeler transformaciju referentnog slijeda. Ovaj proces je vrlo zahtjevan i sa stajališta utrošenog vremena i sa stajališta utrošene memorije. Kako danas još uvijek prevladavaju računala sa 32-bitnim operativnim sustavom koji podržava do 4GB memorije, glavni cilj bio je smanjiti potrošnju memorije. Veličina ljudskog kromosoma 1 (University of California, Santa Cruz – UCSC), na kojem sam vršio ispitivanja, je oko 250 MB. Upotreba potpunog sufiksnog polja zahtijeva 4n bajtova memorije. Pored toga, istovremeno moraju egzistirati i polje sa referentnim slijedom i polje sa BW transformacijom, svako veličine 1n. Ukupno zauzeće memorije bilo bi, u tom slučaju, veće od 1,5 GB. Kako pod 32-bitnim operacionim sustavom Windows, JVM može u praksi ostvariti "heap" memoriju veličine do maksimalno 1,6 GB, metode sufiksnih polja nisu se mogle upotrijebiti za kreiranje BW transformacije. Umjesto toga, referentni slijed se razbija u dijelove u kojim svaka ciklička rotacija ima jednaku početnu sekvencu, a zatim se dio po dio sortira. Veličina parcijalnog polja cikličkih rotacija ovisi o veličini početne sekvence. Povećanjem veličine početne sekvence smanjuje se veličina polja parcijalnih cikličkih rotacija, ali se produžuje ukupno vrijeme sortiranja cijelog niza. Ova funkcija određuje trenutnu početnu sekvencu i poziva funkciju **findRotations** koja kreira polje cikličkih rotacija i zatim ga sortira.

Prilikom izdvajanja cikličkih rotacija, mora se paziti na rotacije sa kraja referentnog slijeda koje su kraće od početne sekvence. Te rotacije ne spadaju niti u jednu izdvojenu grupu. Stoga se te rotacije obrađuju posebno, pozivom funkcije **checkTail**.

#### Privatna funkcija **findRotations**

Funkcija **findRotations** izdvaja iz skupa svih cikličkih rotacija referentnog slijeda one koje započinju zadanom početnom sekvencom, pohranjuje ih u polje parcijalnih cikličkih rotacija, sortira ih, a zatim popunjava polje BW transformacije. Početna sekvenca pohranjena je u polju **startSequenceBuffer**, a njena dužina dana je kao ulazni parametar. Ulazni parametri funkcije još su i pozicija u referentnom slijedu od koje treba započeti pretragu, te polje popunjenih polja izdvojenih cikličkih rotacija koje je u prvom koraku prazno.

Za pohranu izdvojenih rotacija kreira se polje fiksne dužine. Ukoliko se u referentnom slijedu nalazi više rotacija sa zadanom početnom sekvencom nego što stane u za to kreirano polje, povećava se dužina početne sekvence, a zatim se funkcija poziva rekurzivno za svaku moguću novu početnu sekvencu. Prije toga se provjerava kraj referentnog slijeda funkcijom **checkTail**, a popunjeno se polje pohranjuje u za to predviđeno polje polja izdvojenih cikličkih rotacija (ulazni parametar).

Kada se sa pretragom dođe do kraja referentnog slijeda, to jest, kada su izdvojene sve odgovarajuće cikličke rotacije, formira se novo polje koje objedinjava sve izdvojene cikličke rotacije (iz ove i iz svih prethodnih rekurzija), te se sortira pozivom funkcije **sortRotations**. Nakon toga se iz sortiranog polja popunjava odgovarajući segment polja BW transformacije pozivom funkcije **insertBwt**.

Nedostatak ovog algoritma je mogućnost postojanja dugačkih sekvenci istog znaka u referentnom slijedu. U tom slučaju povećanje dužine početne sekvence ne pridonosi smanjenju polja izdvojenih cikličkih rotacija. Sve takve rotacije moraju biti sortirane u istom prolazu.

#### Privatna funkcija **sortRotations**

Funkcija **sortRotations** sortira polje cikličkih rotacija koje je primila kao ulazni parametar. Funkcija kao ulazni parametar prihvaća još i podatak o dužine polja cikličkih rotacija, te dužinu početne sekvence.

Za sortiranje se koristi poznati algoritam "heapsort". To je prilično brz algoritam s vremenskom kompleksnošću O(n log n), koji za sortiranje ne treba dodatne memorije, već za to koristi samo polazno polje. Za usporedba dvije cikličke rotacije koristi se funkcija **isRotationGreater**.

#### Privatna funkcija **isRotationGreater**

Funkcija **isRotationGreater** uspoređuje dvije cikličke rotacije, dane kao ulazni parametar. Uspoređuju se znakovi sa početka rotacija, tako dugo dok se ne naiđe na razliku, i tada se odlučuje koja je rotacija leksikografski veća. Usporedba započinje tek od znaka na poziciji koja je isto dana kao ulazni parametar (sve rotacije imaju jednaku početnu sekvencu).

I u ovoj funkciji dugački sljedovi istog znaka stvaraju probleme, ovaj put u vremenu izvođenja. Da se prebrodi ovaj problem, prije početka sortiranja, kreira se baza takvih sljedova (dužih od 100 znakova) pozivom funkcije **detectLongRuns**. Tijekom usporedbe provjerava se, pozivom funkcije **checkLongRunSequence**, da li se obje rotacije nalaze u nekom od ovih sljedova, pa ako je tako, može se odmah preskočiti na kraj kraćeg.

### Funkcija saveReferenceSequenceIndex

Kada je indeks referentnog slijeda jednom kreiran, može ga se pohraniti u ".rfix" datoteku. Funkcija koja obavlja pohranu zove se **saveReferenceSequenceIndex** i prihvaća ime datoteke za pohranu kao parametar. Pohranjuje se jedino **referenceSequenceIndex** objekt jer on sadrži sve relavantne podatke potrebne za kasniju obradu upitnih zahtjeva. Za pohranu se koristi standardna JAVA-ina procedura za serijalizaciju putem objekta tipa ObjectOutputStream i njegove funkcije writeObject.

### Funkcija countQuerySequenceOccurrences

Funkcija **countQuerySequenceOccurrences** obrađuje upitni zahtjev, odnosno prebrojava podudaranja upitnog sa referentnim slijedom. Funkcija prihvaća upitni slijed (String) kao parametar. Obrada započinje provjerom i transformacijom upitnog slijeda. Izbacuju se znakovi manji od 32 i veći od 126, mala se slova pretvaraju u velika, a kod se transformira u kompaktni kod putem rječnika abecede **alphabetDictionary** iz **referenceSequenceIndex** objekta. Nakon toga, broje se podudaranja upitnog slijeda sa referentnim slijedom procedurom opisanom u poglavlju 3.1. Obrada završava prikazom statistike obrade koja sadrži vrijeme početka i završetka obrade, odnosno ukupno trajanje obrade, dužinu referentnog i upitnog slijeda, broj pronađenih podudaranja, te procjenu zauzeća memorije.

## Klasa ReferenceSequenceIndex

Klasa **ReferenceSequenceIndex** sadrži sve relevantne podatke potrebne za obradu upitnog zahtjeva. To su objekti:

* **alphabetDictionary** tipa **AlphabetDictionary**
* **prefixSumTable** tipa **int[ ]**
* **occurrenceWaveletTree** tipa **FMIndexWaveletTreeNode**

Ovi se podaci popunjavaju tijekom kreiranja objekta tipa ReferenceSequence.

Pored toga ova klasa definira i funkciju **C** koja kao parametar uzima znak iz abecede, a vraća ukupni broj znakova sadržanih u referentnom slijedu, a koji su leksikografski niži od zadanog znaka, te funkciju **getCharacterTotal** koja isto tako kao parametar uzima znak iz abecede, ali vraća ukupni broj ponavljanja tog znaka u referentnom slijedu.

## Klasa AlphabetDictionary

Klasa **AlphabetDictionary** objedinjava sve aktivnosti vezane uz abecedu referentnog slijeda. U fazi učitavanja referentnog slijeda, svaki učitani znak se dodaje objektu ove klase funkcijom **addCharacter** kako bi se prikupili podaci o abecedi. Funkcija prihvaća tekstualni kod znaka kao ulazni parametar. Nakon završetka učitavanja, pokreće se funkcija **finalizeDictionary** čime se objektu dojavljuje da su svi znakovi učitani, i da može kreirati polje za konverziju iz tekstualnog u kompaktni kod. Nakon finalizacije može se pozivati funkcija **getCompactCode** koja kao ulazni parametar uzima tekstualni kod znaka, a vraća njegov kompaktni kod. Klasa još implementira i funkciju **getAlphabetSize** koja se može pozvati nakon finalizacije i vraća broj znakova u abecedi.

## Klasa FMIndexWaveletTreeNode

Klasa **FMIndexWaveletTreeNode** objedinjava sve aktivnosti oko izgradnje i korištenja valićnog stabla referentnog slijeda. Iako objekti ovog tipa predstavljaju po jedan čvor stabla, korisnik kreira samo korijenski čvor, a cijelo stablo će se izgraditi automatski. Slično tome, upiti se postavljaju samo korijenskom čvoru, a odgovor će uzeti u obzir cijelo stablo.

### Konstruktori

Implementirana su dva konstruktora ove klase. Prvi, premda je javni, namijenjen je prvenstveno za internu upotrebu i za ulazne parametre uzima polje sa BW transformacijom referentnog slijeda (ili neki drugi tekstualni niz kodiran kompaktnim kodom), tabelu sume prefiksa (koja mora biti usklađena sa prvim parametrom), te kompaktni kod leksikografski najmanjeg i najvećeg znaka koji se sprema u čvor. Drugi konstruktor namijenjen je kreiranju korijenskog čvora, odnosno cijelog stabla. Prva dva parametra su jednaka kao i u prvom slučaju, dok se kompaktni kodovi najmanjeg i najvećeg znaka ne daju kao parametar, već se podrazumijeva da se čvor kreira za cijelu abecedu.

Kada se konstruktor pokrene za korijenski čvor, on najprije izgradi sam sebe, a zatim rekurzivno pokrene konstruktore za čvorove u obje grane stabla (ako je potrebno). Na taj se način automatski izgradi cijelo stablo. Stablo je balansirano, što znači da je broj znakova u svakoj grani približno jednak.

U drugom koraku, nakon izgradnje stabla, ono se popunjava podacima. Za to se koristi funkcija **insertCharacter** koja prvo znak doda u vlastiti čvor, a zatim se rekurzivno poziva za čvor u grani kojoj taj znak pripada. Ova funkcija kao parametar uzima znak koji se dodaje.

### Funkcija Occ

Funkcija **Occ** vraća broj ponavljanja nekog znaka od početka tekstualnog niza pohranjenog u stablu, pa do određene pozicije. Funkcija kao ulazni parametar prihvaća kompaktni kod znaka za kojeg se računa broj ponavljanja i pozicija do koje se broji. Funkcija se poziva samo za korijenski čvor, a ona se dalje sama rekurzivno poziva za granu u kojoj se nalazi specificirani znak.

### Funkcija getMemoryConsumption

Funkcija **getMemoryConsumption** poziva se za korijenski čvor stabla i vraća procijenjeni utrošak memorije za cijelo stablo. Slično kao i kod ostalih funkcija, ona prvo računa utrošak memorije u vlastitom čvoru, a zatim se rekurzivno poziva za čvorove u granama ispod sebe.

## Klasa FMIndexBitString

Klasa **FmIndexBitString** objedinjava sve aktivnosti oko izgradnje i korištenja binarnog niza sa brzim prebrojavanjem.

### Konstruktor

Konstruktor za objekte ove klase prihvaća veličinu niza u bitovima kao ulazni parametar. Izgrađeni objekt ima sve podatkovne strukture korektno izgrađene, a ispunjen je samim nulama.

### Funkcije setBit i resetBit

Funkcije **setBit** i **resetBit** koriste se za postavljanje odnosno za brisanje pojedinačnog bita na poziciji danoj kao ulazni parametar. Promjene se provode na cijeloj podatkovnoj strukturi, a to znači da se moraju osvježiti za sve zbrojeve grupa i super grupa od zadane pozicije, pa do kraja niza. To je dugotrajna operacija i nije pogodna za inicijalno punjenje dugih nizova.

### Funkcije setBitNoBucket i resetBitNoBucket

Funkcije **setBitNoBucket** i **resetBitNoBucket** koriste se za postavljanje odnosno za brisanje pojedinačnog bita na poziciji danoj kao ulazni parametar. Promjene se provode samo na binarnom nizu, dok se zbrojevi grupa i super grupa ne diraju. Ove funkcije koriste se za inicijalno popunjavanje niza podacima. Kada je binarni niz popunjen, zbrojevi grupa i super grupa osvježavaju se u jednom prolazu pozivom funkcije **refreshBuckets**.

### Funkcija refreshBuckets

Funkcija **refreshBuckets** poziva se kada je binarni niz popunjen podacima. Funkcija na osnovu binarnog niza popunjava polja zbrojeva grupa i super grupa.

### Funkcija rank1

Funkcija **rank1** vraća ukupni broj jedinica u binarnom nizu od početka pa do pozicije dane kao ulazni parametar. Kao što je u poglavlju 3.2 rečeno, funkcija je vrlo brza i odvija se u konstantnom vremenu neovisno o veličini binarnog niza i poziciji do koje se broji.

### Funkcija getMemoryConsumption

Funkcija **getMemoryConsumption** vraća procijenjeni utrošak memorije binarnog niza.

# Testiranje i rezultati mjerenja

## Testiranje

Testiranje ispravnosti rada vršeno je primjenom kratkih, jednostavnih sljedova koji daju poznate posljedice. Budući da su pojedine funkcije, pa i čitavi algoritmi rađeni za obradu vrlo velikih sljedova, za njih ovakav sustav ispitivanja nije zadovoljavajući. Za takva ispitivanja korišteni su realni podaci koji su bili dobavljivi preko Interneta. Konkretno, korišten je slijed ljudskog kromosoma 1, dobavljen preko portala "UCSC Genome Bioinformatics" Kalifornijskog univerziteta iz Santa Cruza. Ovaj slijed dan je u fasta formatu i sastoji se od 249 250 621 znaka uključujući i znak N kojim se označavaju nepoznata područja. U nekim slučajevima, radi bržeg ispitivanja, koristila se je i skraćena, a katkada i editirana verzija ovog slijeda.

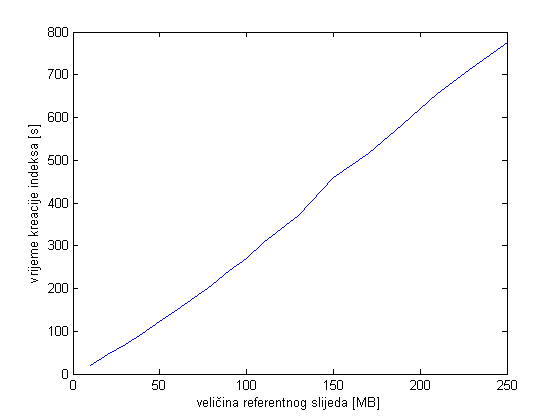
Pored ovih ispitivanja, rađene su intenzivna ispitivanja u ključnim točkama obrade. Jedno od takvih mjesta je završetak kreiranja BW transformacije kada se spoje svi parcijalni segmenti obrade. U tu svrhu, paralelno sa poljem BW transformacije, popunjavano je i testno polje sortiranih cikličkih rotacija. Potom je pokrenuta testna funkcija koja je provjeravala međusobni odnos svih parova susjednih rotacija u tom polju i ukupan broj kreiranih rotacija. Na kraju se provjerila i konzistentnost polja BW transformacije sa testnim poljem sortiranih cikličkih rotacija. Ispitivanja su rađena i na kraćim sljedovima, ali i na čitavom slijedu kromosoma 1 i nisu pokazala nikakva odstupanja od očekivanja.

Krajnja funkcionalnost aplikacije provjeravana je usporedbom rezultata dobivenih upotrebom aplikacije FMIndexCount i rezultata dobivenih upotrebom aplikacije "BLAT" sa portala "UCSC Genome Bioinformatics". Niti ovdje nisu otkrivena nikakva odstupanja od očekivanja.

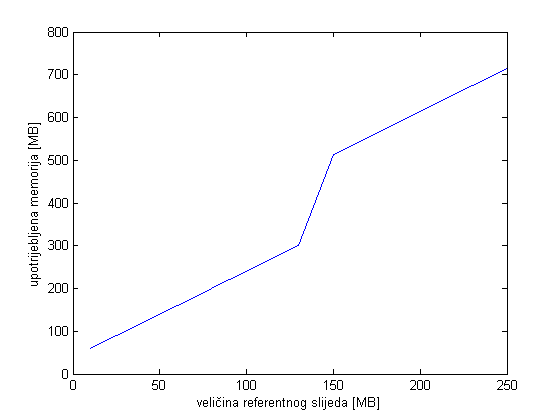
## Rezultati mjerenja

Nakon završetka izrade aplikacije, provedena su i mjerenja utrošenog vremena i memorije, i to i u fazi kreiranja indeksa referentnog slijeda i u fazi brojanja podudaranja upitnog slijeda sa referentnim. Korištene su različite dužine referentnog i upitnog slijeda. Rezultati su prikazani u dijagramima na slikama 5.2.1 do 5.2.4.

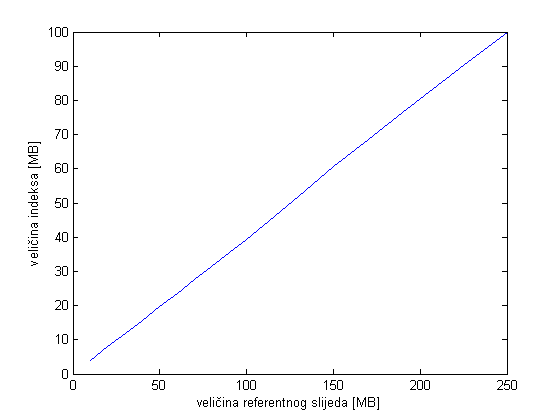
Vrijeme kreacije indeksa referentnog slijeda je gotovo linearno, iako se vidi lagano povećanje strmine s porastom veličine referentnog slijeda. Prosječni odnos iznosi oko 3,2 s/MB. Na krivulji utrošene memorije u ovisnosti o veličini referentnog slijeda vidi se nagli skok nakon 130 MB. Do skoka najvjerojatnije dolazi zbog koncentracije nepoznatih područja u tom dijelu referentnog slijeda. U ostalim dijelovima krivulje vidi se da upotrijebljena memorija raste u omjeru oko 2:1 prema veličini referentnog slijeda. To je zbog toga što se velika većina memorije troši za pohranu referentnog slijeda i njegove BW transformacije. Veličina indeksa referentnog slijeda u odnosu na veličinu samog indeksa je potpuno linearna. Postignuta kompresija iznosi 2,5. I vrijeme pretrage u odnosu na veličinu upitnog slijeda je linearno i iznosi oko 340 ns po znaku upitnog slijeda.



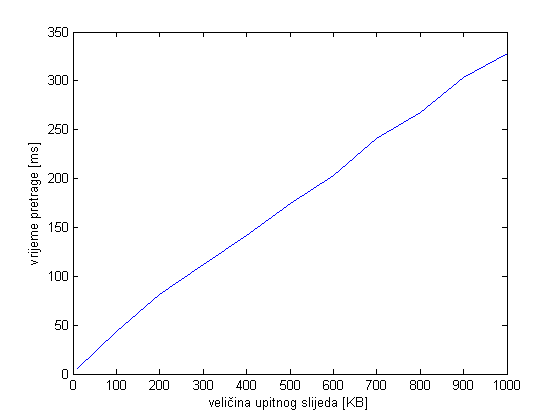
Slika 5.2.1 – Ovisnost vremena kreiranja indeksa o veličini referentnog slijeda



Slika 5.2.2 – ovisnost zauzeća memorije za vrijeme kreiranja indeksa o veličini referentnog slijeda



Slika 5.2.3 – ovisnost veličine indeksa o veličini referentnog slijeda



Slika 5.2.4 – Ovisnost vremena pretrage o veličini upitnog slijeda

# Zaključak

U okviru ovog projekta izrađena je aplikacija koja prikazuje broj ponavljanja upitnog tekstualnog slijeda u puno većem referentnom slijedu. Kroz izradu aplikacije pokazalo se da primjenom modernih rješenja na području obrade teksta mogu postići odlični rezultati u brzini obrade i količini upotrijebljene memorije. Ovakva rješenja su naročito bitna na području bioinformatike gdje je potrebno obrađivati vrlo dugačke tekstualne sljedove u što kraćem vremenu

Jedan od ciljeva je bio i ušteda memorije prilikom kreiranja indeksa referentne veličine do te mjere da se aplikacija može izvoditi i na 32-bitnim platformama koje su danas još uvijek najčešće u upotrebi. U tome se i uspjelo, pa je za izradu indeksa ljudskog kromosoma 1, veličine 250 MB, bilo utrošeno svega 714 MB memorije. Nakon izrade indeksa ta se brojka spustila na samo 100 MB.

Sama pretraga referentnog slijeda vrlo je brza i ne ovisi o njegovoj veličini. Postignute su brzine od oko 340 ns po znaku upitnog niza.

# Literatura

[1] FM-index, 14.12.2013, <http://en.wikipedia.org/wiki/FM-index>, 16.01.2014.

[2] FM-Indexes and Backwards Search, <http://alexbowe.com/fm-index/>, 16.01.2014.

[3] Jochen Singer, A Wavelet Tree Based FM-Indeks for Biological Sequences in SeqAn, 30.01.2012., <http://www.mi.fu-berlin.de/wiki/pub/ABI/FMIndexThesis/FMIndex.pdf>, 16.01.2014.

[4] An Introduction to Next-Generation Sequencing Technology, 16.10.2013., <http://res.illumina.com/documents/products/illumina_sequencing_introduction.pdf>, 16.01.2014.

[5] FASTA format, 10.01.2014, <http://en.wikipedia.org/wiki/FASTA_format>, 16.01.2014.

[6] Succinct dana structure, 28.12.2013, [http://en.wikipedia.org/wiki/Succinct\_data\_structure#Succinct\_dictionaries](http://en.wikipedia.org/wiki/Succinct_data_structure%23Succinct_dictionaries), 16.01.2014.

[7] Wavelet Tree, 22.03.2013, <http://en.wikipedia.org/wiki/Wavelet_Tree>, 16.01.2014.

[8] Heapsort, 07.01.2014, <http://en.wikipedia.org/wiki/Heap_sort>, 16.01.2014.