

SVEUČILIŠTE U ZAGREBU  
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

Predmet:

**Bioinformatika**

Mile Šikić

Mirjana Domazet-Lošo

Zagreb, prosinac 2013.

Ova skripta je slobodno dostupna i može se preuzeti kao PDF datoteka s adrese <http://www.fer.unizg.hr/predmet/bio>. Skripta je napravljena kao prateći materijal za predmet Bioinformatika koji se održava u sklopu diplomskoga studija na Fakultetu elektrotehnike i računarstva.

Slobodno smijete umnožavati, distribuirati i javnosti priopćavati djelo pod sljedećim uvjetima (<http://creativecommons.org/licenses/by-nc-nd/3.0/hr/>):

**Imenovanje** Morate priznati i označiti autorstvo djela na način kako je specificirao autor ili davatelj licence (ali ne na način koji bi sugerirao da Vi ili Vaše korištenje njegova djela imate njegovu izravnu podršku).

**Nekomercijalno** Ovo djelo ne smijete koristiti u komercijalne svrhe.

**Bez prerada** Ne smijete mijenjati, preoblikovati ili prerađivati ovo djelo.

U slučaju daljnog korištenja ili distribuiranja morate drugima jasno dati do znanja licencne uvjete ovog djela.

Od svakog od navedenih uvjeta moguće je odstupiti samo ako dobijete dopuštenje nositelja autorskog prava.

©2013. Mile Šikić, Mirjana Domazet-Lošo.

## Sadržaj

1.	Genomika .....	7
1.1.	Biološke osnove.....	7
1.1.1.	Središnja dogma .....	8
1.1.2.	Geni .....	9
1.2.	Sekvenciranje.....	10
1.2.1.	Projekt određivanja ljudskog genoma i razvoj metoda za sekvenciranje....	11
1.2.2.	Metode sekvenciranja.....	12
1.3.	Formati podataka .....	16
1.3.1.	FASTA format podataka .....	16
1.3.2.	FASTQ format podataka .....	17
1.3.3.	Zapisи genoma .....	18
1.4.	Literatura .....	20
2.	Optimalno poravnavanje sljedova .....	21
2.1.	Poravnanje .....	21
2.2.	Algoritam za računanje udaljenosti uređivanja .....	22
2.3.	Needleman-Wunsch algoritam .....	26
2.4.	Algoritam preklapanja.....	27
2.5.	Lokalno poravnanje .....	30
2.6.	Optimizacije primjenjive na algoritme za poravnanje sljedova .....	32
2.6.1.	Linearna memorijska složenost .....	33
2.6.2.	Dinamičko programiranje s ograničenim pojasom.....	36
2.7.	Biološka usporedba nizova.....	37
2.7.1.	Matrice sličnosti .....	37
2.7.2.	Praznine i procijepi.....	38
2.8.	Literatura .....	40

3.	Pretraživanje baze podataka sljedova. BLAST .....	41
3.1.	Usporedba bioloških sljedova s bazom podataka korištenjem heurističkog pristupa .....	42
3.2.	Program BLAST.....	44
3.3.	BLAST - postupak pretraživanja baze podataka.....	50
3.4.	Literatura .....	53
4.	Sufiksno stablo i sufiksno polje.....	54
4.1.	Sufiksno stablo .....	55
4.1.1.	Osnovni pojmovi .....	56
4.1.2.	Struktura podataka sufiksno stablo.....	56
4.1.3.	Memorijski zahtjevi.....	58
4.1.4.	Izgradnja sufiksnog stabla u vremenu $O(n^2)$ .....	59
4.1.5.	Implicitno sufiksno stablo .....	63
4.1.6.	Ukkonenov algoritam – „naivna“ izvedba.....	64
4.1.7.	Ukkonenov algoritam u linearnom vremenu .....	68
4.1.8.	Poopćeno sufiksno stablo .....	72
4.2.	Sufiksno polje .....	73
4.2.1.	Sufiksno polje i sufiksno stablo.....	74
4.2.2.	Izgradnja sufiksnog polja u linearnom vremenu Kärrkäinen-Sandersovim algoritmom .....	74
4.2.3.	Izgradnja sufiksnog polja u linearnom vremenu Nong-Zhang-Chanov algoritmom .....	78
4.3.	Literatura .....	86
5.	Poravnavanje dva ili više sljedova odjednom.....	87
5.1.	Heuristički pristup poravnavanju para sljedova .....	87
5.1.1.	MUMmer.....	87
5.2.	Poravnavanje više sljedova odjednom.....	88

5.2.1.	Metoda sume parova.....	89
5.2.2.	Poravnavanje više sljedova odjednom dinamičkim programiranjem.....	91
5.2.3.	Heuristički pristup poravnjanju više sljedova odjednom.....	94
5.3.	Literatura .....	97
6.	Samostojni indeksi.....	98
6.1.	Određivanje entropije znakovnog niza .....	99
6.1.1.	Shannonova entropija .....	99
6.1.2.	Shannonova entropija znakovnog niza .....	100
6.1.3.	Entropija višeg reda.....	101
6.2.	Burrows-Wheelerova transformacija (BWT) .....	102
6.2.1.	Konstrukcija BWT rotacijom znakovnog niza .....	102
6.2.2.	Konstrukcija BWT korištenjem sufiksnog polja .....	103
6.2.3.	Reverzibilnost BWT-a.....	104
6.3.	FM-indeks.....	108
6.3.1.	Interval sufiksnog polja .....	108
6.3.2.	Traženje podniza u nizu korištenjem FM-indeksa .....	109
6.3.3.	Prostorna i vremenska složenost FM-indeksa .....	113
6.4.	Literatura .....	114
7.	Filogenija.....	115
7.1.1.	Povijesni razvoj ideja .....	115
7.2.	Filogenija i filogenetsko stablo .....	116
7.2.1.	Broj mogućih stabala u ovisnosti o broju taksonomske jedinice .....	119
7.3.	Evolucijski DNA modeli .....	119
7.3.1.	Jukes-Cantorov model .....	120
7.4.	Metode za izgradnju filogenetskih stabala .....	124
7.4.1.	Mjera udaljenosti .....	125

7.5.	Metode temeljene na udaljenostima između taksona .....	125
7.5.1.	UPGMA.....	125
7.5.2.	Metoda povezivanja susjeda.....	128
7.6.	Metode temeljene na obilježjima.....	131
7.6.1.	Metoda najmanjeg broja evolucijskih promjena .....	131
7.7.	Literatura .....	136
8.	Sastavljanje genoma .....	137
8.1.	Izazov sastavljanja.....	137
8.2.	Osnove sastavljanja .....	139
8.3.	Algoritmi za sastavljanje .....	140
8.3.1.	Traženje najkraćeg zajedničkog nadniza.....	140
8.3.2.	Algoritmi nad grafovima .....	143
8.4.	Preklapanje-Razmještanje-Konsenzus pristup .....	145
8.4.1.	Preklapanje .....	145
8.4.2.	Razmještanje.....	149
8.4.3.	Konsenzus.....	153
8.5.	Sastavljanje koristeći <i>de Bruijn</i> grafove .....	155
8.5.1.	Problem sastavljanja genoma kratkim očitanjima .....	155
8.5.2.	Eulerova staza i <i>de Bruijn</i> grafovi .....	155
8.5.3.	Sastavljanje genoma .....	159
8.6.	Literatura .....	163

# 1. Genomika

## 1.1. Biološke osnove

Genomika je grana genetike koja primjenjuje metode DNA sekvenciranja i bionformatike u cilju sekvenciranja, sastavljanja i analize funkcija i strukture genoma.

Za funkcioniranje svih poznatih živih organizama presudne su makromolekule nukleinskih kiselina, proteina i ugljikohidrata – takozvani biopolimeri. Dvije temeljne vrste nukleinskih kiselina su RNA i DNA. DNA služi kao spremište instrukcija za razvoj i funkcioniranje svih živih organizama, s izuzetkom RNA virusa. Postoji nekoliko vrsta RNA molekula koje dijelimo ovisno o njihovim funkcijama. Najvažnije su glasnička RNA (mRNA), transportna RNA (tRNA), ribosomska (rRNA) te regulacijske RNA poput mikro RNA (miRNA), male jezgrene RNA (snRNA) i male interferirajuće RNA (siRNA). Jednim imenom sve RNA koje možemo naći u stanici nazivamo transkriptom.

Osnovna jedinica nasljeđivanja u živim organizmima je gen. Gen je dio DNA ili RNA koji kodira informaciju za proizvodnju proteina ili RNA lanaca koji imaju aktivnu funkciju u organizmu.

Većina DNA molekula su dvostrukе uzvojnice koje se sastoje od manjih gradivih jedinica - nukleotida. Svaki nukleotid se sastoji od nukleinskih baza (adenin, citozin, gvanin i timin) koje označavamo slovima A, C, G i T, te okosnice koja je građena od naizmjeničnoga niza šećera i fosfatne skupine. Ta dva lanca su međusobno povezana na način da se adenin spaja s timinom, a gvanin sa citozinom. Lunci se protežu u suprotnim smjerovima i zbog toga kažemo da su anti-paralelni. Te lance nazivamo 3' i 5' zavisno o smjeru trećeg i petog atoma ugljika molekule šećera na okosnici. DNA je upakirana u strukture koje nazivamo kromosomima.

RNA molekule imaju ulogu u kodiranju, dekodiranju, regulaciji i ekspresiji gena. RNA je poput DNA građena od nukleotida, ali za razliku od DNA je najčešće građena od jednog lanca. Nukleinske baze koje uz šećer i fosfatnu grupu tvore lanac RNA su adenin, citozin, gvanin i uracil koje označavamo sa slovima A, C, G i U.

Proteini su makromolekule sastavljene od jednoga ili više lanaca aminokiselina, a unutar živih organizama obavljaju mnogobrojne funkcije: služe kao gradivi blokovi, ubrzavaju metaboličke reakcije, služe za transport molekula, umnažanje i prepisivanje DNA te odgovaranje na podražaje. Jedan od najpoznatijih proteina je hemoglobin koji prenosi atome kisika u krvi. Popis funkcija proteina dug je i s novim biološkim spoznajama konstantno raste.

### 1.1.1. Središnja dogma

Prvi puta opisana 1956 (Crick, 1956), središnja dogma molekularne biologije objašnjava način na koji se genetski nacrt živog organizma, sadržan u njegovom DNA, „pretvara“ u funkcionalne jedinice koje mogu obavljati biološke funkcije u organizmu – a to su proteini i RNA. Danas je ta definicija proširena i opisuje sve načine protoka informacija između DNA, RNA i proteina.

Postoji 9 objasnjivih prijenosa informacija prikazanih u Tablica 1.1. Dogma ih dijeli u 3 grupe po 3: 3 opća prijenosa (odvijaju se u većini stanica), 3 specijalna prijenosa (odvijaju se samo pod specifičnim uvjetima i to obično u virusima ili su pokazani u laboratoriju) i 3 nepoznata (vjerojuće se da se ne događaju).

Tablica 1.1 Središnja dogma – 3 klase prijenosa informacija

<i>Opći</i>	<i>Specijalni</i>	<i>Nepoznat</i>
DNA → DNA	RNA → DNA	protein → DNA
DNA → RNA	RNA → RNA	protein → RNA
RNA → protein	DNA → protein	protein → protein

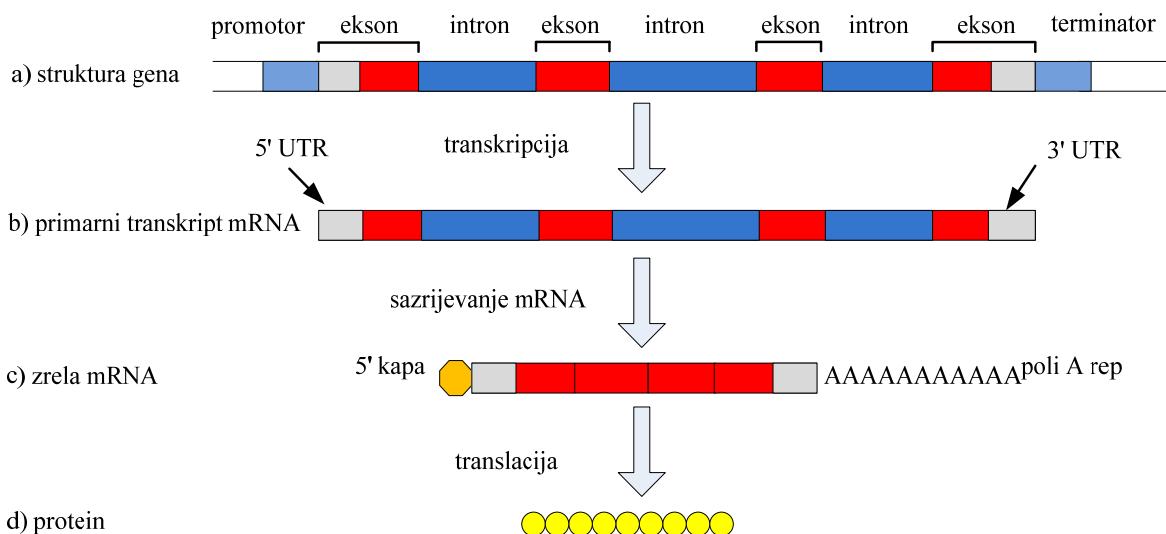
Opći prijenosi su umnažanje DNA, kopiranje informacije iz DNA u mRNA (transkripcija) i sintetiziranje proteina iz mRNA. Dodatno, informacije iz DNA se mogu kopirati i u ostale tipove RNA kao što su npr. tRNA, rRNA. Specijalni oblici prijenosa su: kopiranje informacija iz RNA u DNA (obrnuta transkripcija), kopiranje jedne RNA u drugu (RNA umnažanje), te direktna translacija DNA u proteine koja je pokazana samo u laboratorijskim uvjetima.

### **1.1.2. Geni**

Gen je osnovna molekularna jedinica nasljeđivanja svih živih organizama. Geni su milijardama godina usavršavani nacrti za proizvodnju bioloških mašina – proteina i funkcionalnih RNA lanaca. Mogu biti dugi od nekoliko desetaka nukleotida pa do nekoliko milijuna. Izmjena jednog nukleotida u ključnom genu može dovesti do potpuno nefunkcionalne jedinke. Kod većine živih bića, geni se nalaze pohranjeni u dvostrukoj DNA uzvojnici. Na jednom lancu te uzvojnici nalazi se genetski nacrt opisan slijedom nukleotida potrebnim za proizvodnju proteina ili RNA. Na komplementarnom lancu nalazi se kalup pomoću kojeg je moguće proizvesti produkt tog gena.

Ekspresija gena (Sl. 1.1) je proces u kome se informacije iz gena koriste za sintezu funkcionalnog produkta gena – najčešće proteina preko mRNA, iako neki geni kodiraju rRNA, tRNA ili snRNA.

Prvi korak ekspresije je transkripcija - prepisivanje DNA u mRNA. Kod prokariota (živih bića bez stanične jezgre poput bakterija) transkripcija gena koji kodiraju proteine direktno kreira mRNA koja je spremna za translaciju u protein. Eukarioti, (živa bića sa staničnom jezgrom poput životinja, biljaka i gljiva) imaju složeniju strukturu gena – umjesto da je čitava informacija gena kodirana u jednom slijedu nukleotida, eukariotski gen je ispresijecan nekodirajućim regijama (intronima) koji omogućuju da se isti gen različitim spajanjem kodirajućih regija (eksona) iskoristi za više proteinskih proizvoda. Zbog tog se kod njih proces transkripcije odvija u nekoliko koraka. U prvom koraku prepisivanjem informacija s DNA nastaje primarni transkript od RNA (pre-mRNA). Primarni transkript sadrži eksone i introne, kao i dijelove na 3' i 5' kraju transkripta koji se ne prevode u protein (UTR, od engl. *untranslated regions*).



Sl. 1.1 Struktura eukariotskog gena i slijed genske ekspresije. a) Struktura gena: promotor, eksoni, introni, terminator. b) Transkripcija započinje na promotoru, a završava na terminatoru. Produkt transkripcije je primarni transkript mRNA, a sastoji se od eksona, introna, 5' UTR i 3' UTR (engl. *untranslated regions*; dijelovi koji se ne prevode u protein). c) Doradom primarnoga transkripta procesima prekrajanja introna, dodatku 5' kape i poli A repa nastaje zrela mRNA. d) Nakon izlaska iz jezgre, mRNA ribosomu u procesu translacije daje uputu za sintezu proteina.

Nakon dorade primarnog transkripta mRNA procesima izrezivanja introna, dodavanja 5' kape te poli A repa (sastoji se od dugačkoga niza adenina) nastaje zrela mRNA koja po izlasku iz jezgre na ribosomu kao kalup diktira sintezu proteina. Taj proces nazivamo prevođenje (translacija) određenog niza nukleotida mRNA u protein.

Sveukupna nasljedna informacija nekoga organizma naziva se genom. Genom sadrži i gene i sve druge dijelove RNA i DNA, bila njihova funkcija poznata ili ne.

## 1.2. Sekvenciranje

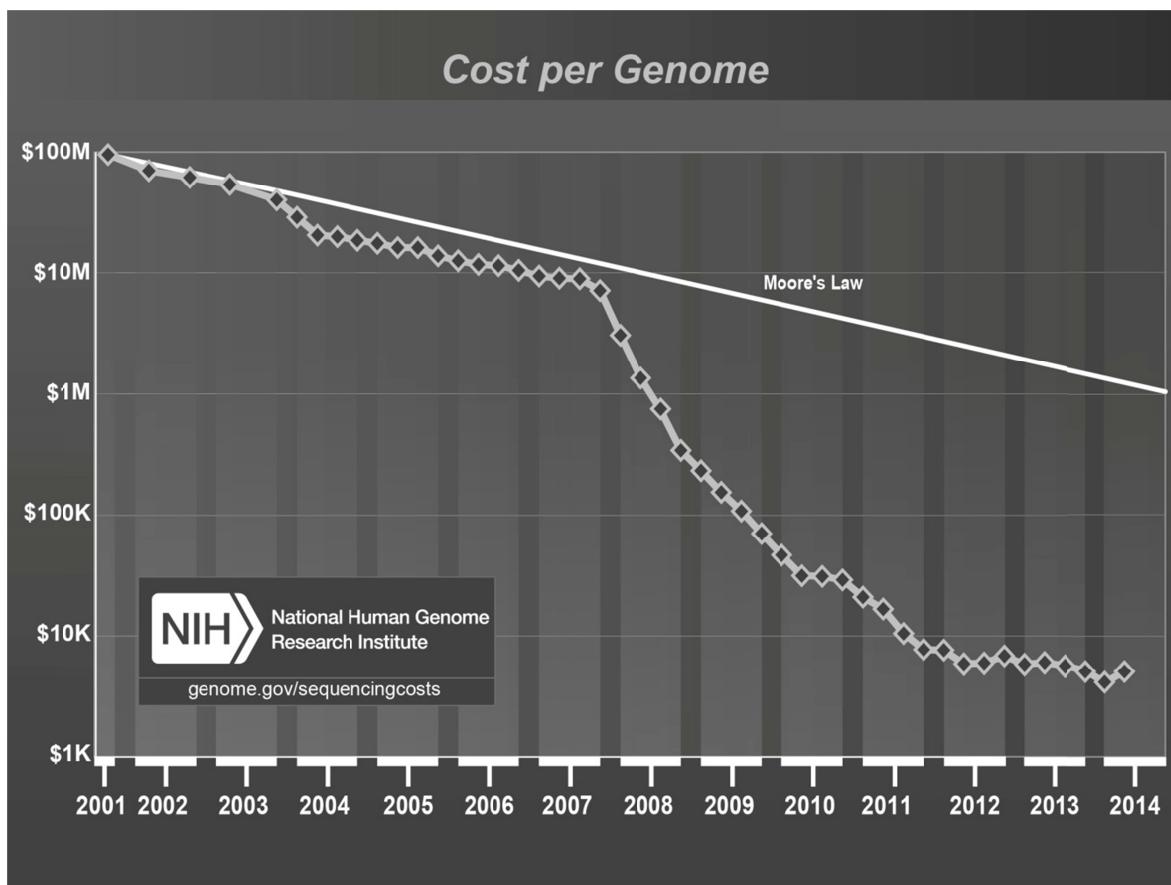
Kako je sva genetska informacija sadržana u lancu DNA ili RNA, koristi od određivanja tog slijeda su ogromne – nude nam bolje razumijevanje nasljednih bolesti, patogenih toksina, infekcija, tumora i manje-više svih bioloških interakcija organizama. Sekvenciranje je proces određivanja poretku pojedinih nukleinskih baza – adenina, citozina, gvanina, timina i uracila – unutar RNA ili DNA lanca. Rezultati sekvenciranja se koriste za utvrđivanje samoga genoma, utvrđivanje srodnosti i evolucije vrsta, utvrđivanje novih gena i njihovo pridruživanje bolestima, utvrđivanje potencijalnih ciljeva za lijekove te metagenomiku - istraživanje zajednica mikroba direktno iz okoliša.

### **1.2.1. Projekt određivanja ljudskog genoma i razvoj metoda za sekvenciranje**

Razvoj sekvenciranja potaknut je radom Fredericka Sangera koji je razvio brze metode sekvenciranja sredinom 1970-tih. Ključna godina za razvoj sekvenciranja je 1990. kada su Ministarstvo energetike (*Department of Energy*) i Nacionalni institut za zdravlje (*National Institute of Health*) vlade SAD-a zajednički pokrenuli projekt određivanja ljudskoga genoma (engl. *Human genome project – HGP*). Ciljevi projekta su bili: određivanje i skladištenje slijeda svih parova baza ljudske DNA, transfer tehnologije u privatni sektor i odgovaranje na etička, socijalna i pravna pitanja koja mogu nastati iz projekta. U lipnju 2000-te završena je radna verzija „skice“ cjelokupnoga ljudskoga genoma koja je pokrivala >90% genoma. Radna verzija je objavljena u veljači 2001, a projekt je proglašen završenim u travnju 2003.

Rezultati projekta određivanja ljudskoga genoma su pokazali da on sadrži 3 milijarde parova baza (A, C, G i T); da se prosječni gen sastoji od 3000 baza, dok duljina može varirati i do 2,4 milijuna baza; ukupan broj gena je oko 20500 što je višestruko manje od procijenjenih 80000 – 140000; 99.9 % baza su iste u svim ljudima. Cijena projekta je bila oko 3 milijarde američkih dolara.

Rezultati projekta su potaknuli snažan razvoj metoda za sekvenciranje i ubrzan pad cijena. Sl. 1.2 prikazuje taj dramatičan pad cijena sekvenciranja i njegovo približavanje dugo sanjanoj granici sekvenciranja ljudskoga genoma za 1000 američkih dolara ili manje. Taj cilj je konačno ostvaren u siječnju 2014. kada je tvrtka Illumina objavila da njihovi novi uređaji mogu sekvencirati ljudski genom pri čemu ukupan trošak neće prijeći 1000 dolara.



Sl. 1.2 Cijena sekvenciranja ljudskoga genoma i njena usporedba njezina pada s Mooreovim zakonom (izvor: <http://www.genome.gov/sequencingcosts/>)

Ovim padom cijena težište se pomiče na algoritme za sastavljanje i analizu genoma.

### 1.2.2. Metode sekvenciranja

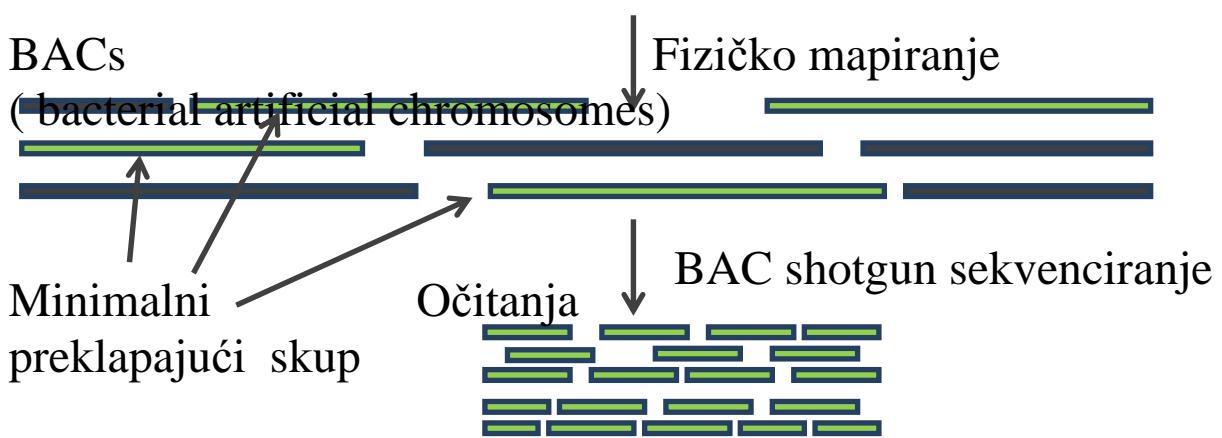
Metode za sekvenciranje mogu pročitati ograničen broj nukleotida. Taj se broj, zavisno o izvedbi, kreće od 50 do nekoliko desetaka tisuća. Iz tog je razloga prije sekvenciranja potrebno podijeliti dulje nukleotidne lance u kraće te ih potom očitati nekom od metoda sekvenciranja.

Danas dominantna strategija je takozvano *shotgun* sekvenciranje (sekvenciranje sačmaricom). Njegovom primjenom DNA se lomi na slučajan način u velik broj malih fragmenata koji se onda sekvenciraju i na taj način se dobivaju očitanja koja predstavljaju pročitani slijed nukleinskih baza. Višestruka očitanja DNA se dobivaju ponavljanjem fragmentacije i sekvenciranja. Na kraju, računalni programi koriste preklapajuće krajeve očitanja da bi ih sastavili u kontinuiranu sekvencu.

Metode sekvenciranja se mogu podijeliti ugrubo u dvije skupine: hijerarhijsko *shotgun* sekvenciranje i *shotgun* sekvenciranje cijelog genoma.

Kod hijerarhijskog *shotgun* sekvenciranja (Sl. 1.3) prvi korak je izrada fizičke mape genoma niske rezolucije. Iz te mape se odredi minimalan broj fragmenata koji je potreban za sekvenciranje cijelog genoma. Pri korištenju ove metode genom se dijeli u fragmente dugačke 50000 – 200000 nukleinskih parova baza (za jedan par nukleinskih baza koristi se jedinica b – engl. *base pair*). Dobiveni fragmenti umeću se u bakterije domaćine koristeći BAC-ove (engl. *Bacterial artificial chromosome*) i na taj način kloniraju. Na taj način dobijemo veći broj kopija originalnih fragmenata koje sada ponovo usitnjavamo na slučajan način na veličine pogodne za sekvenciranje (obično oko 2000 baza odnosno 2 kb). S obzirom da je lokacija BAC-ova na kromosomu poznata ova metoda pruža veliku točnost i omogućava sastavljanje cijelog kromosoma.

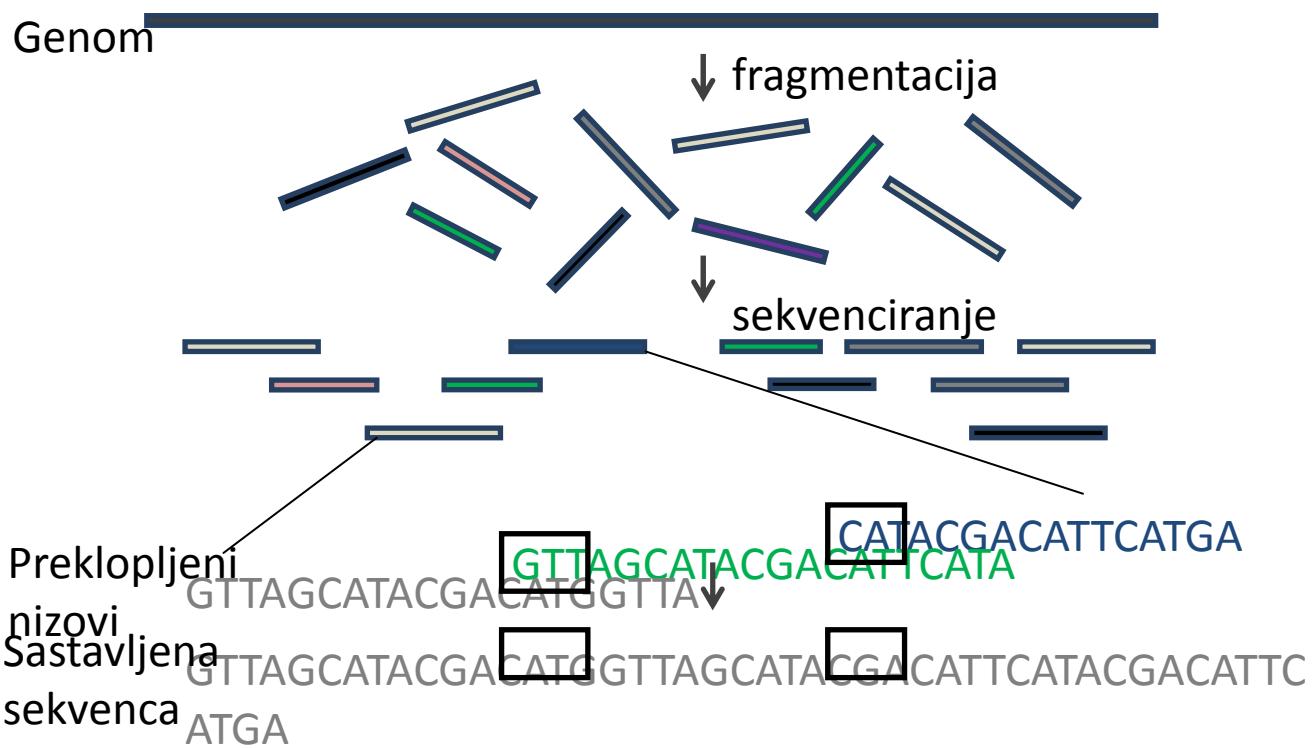
## Ljudski genom



Sl. 1.3 Hijerarhijsko *shotgun* sekvenciranje

Ideja *shotgun* sekvenciranja cijelog genoma je u preskakanju faze izrade fizičke mape. Izrada fizičke mape je spor i skup procesa umetanja BAC-ova u bakterije, čekanje da se one razmnože svojim prirodnim tempom, te naknadno uklanjanje istih regija iz bakterija. Zahtjeva stručno osoblje i mnoštvo skupe laboratorijske opreme. Bez izrade fizičke mape, sekvenciranje može obaviti needucirana osoba, proces postaj jeftiniji, brži i jednostavniji, ali se na taj način stavlju veći zahtjevi na računalnu obradu. Primjenom ove metode DNA se dijeli u slučajne fragmente različitih duljina (obično, 2, 10, 50 i 150 kb) koji se kloniraju. Nakon toga se ti fragmenti sekvenciraju. Sekvenciranje se obavlja uvijek u smjeru od 5' kraja prema 3' kraju. S obzirom da svaki lanac ima jedan 5' kraj to nam omogućava da čitamo s dva kraja. Originalni slijed se dobije rekonstrukcijom iz očitanja.

Iako olakšava sekvenciranje dugačkih regija DNA, nedostatak ove metode je u tome što, je mogućnost ove metode da korektno poveže te regije pod sumnjom, pogotovo za genome s ponavljajućim regijama.



Sl. 1.4 *Shotgun* sekvenciranje cijelog genoma

Za mjerjenje performansi uređaja za sekvenciranje ključni parametri su duljine dobivenih očitanja, točnost i cijena. Tehnološki se uređaji za sekvenciranje dijele na tri generacije: prvu, drugu (još se i naziva sekvenciranje sljedeće generacije prema engl. *Next generation sequencing*) i treću. Tablica 1.2 prikazuje usporednu analizu tri generacije uređaja za sekvenciranje.

Tablica 1.2 Usporedna analiza uređaja za sekvenciranje

Metoda	Duljina očitanja (bp)	Točnost (%)	Cijena za 1 milijun baza (USD)	Komentar
Sangerovo sekvenciranje <b>prva generacija</b>	400-900	99.9	2400	Duga očitanja, povoljna za mnoge primjene, no preskupa za veće projekte
Sekvenciranje sintezom (Illumina) <b>druga generacija</b>	50 -300	98	0.05 – 0.15	Točna, dobra za velike projekte, no uređaji mogu biti jako skupi.
Ion poluvodičko sekvenciranje (Ion Torrent) <b>druga generacije</b>	do 400	98	1	Manje skupa oprema.
Sekvenciranje jedne DNA molekule u realnom vremenu (Pacific Bio) <b>treća generacija</b>	5500-8500 u prosjeku >30000 max	87%	0.33 – 1	Najdulja očitanja, no velika greška i uređaji su skupi
Nanopore sekvenciranje (Oxford Nanopore) <b>treća generacija</b>	do 50k	60 – 70 %	Nepoznata	Uskoro kreće u testnu fazu. Mali uređaji koji se mogu ukopčati preko USB konektora

Osnovna mjera kvalitete sekvenciranja je pokrivenost (engl. *coverage*) koje se ponekad naziva i dubina očitanja. Pokrivanost je prosječan broj očitanja koji predstavljaju dani nukleotid u rekonstruiranom slijedu. Ukoliko sekvenciramo segment genoma duljine  $G$  i time dobijemo  $N$  očitanja, duljine  $L$  pokrivenost računamo kao:

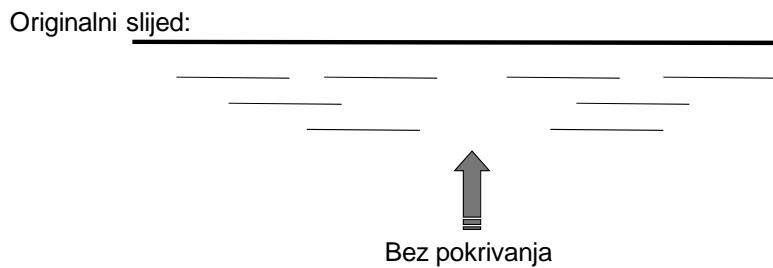
$$C = NL/G \quad (1.1)$$

Pokrivenost obično nije uniformno i njegove varijacije su unesene slučajno, varijacijom u broju staničnih kopija izvornih DNA molekula i pristranošću metode sekvenciranja. Vrlo niska pokrivenost proizvodi procijepu u sastavljenom genomu (Sl. 1.5) i time se ne može dobiti jedinstveni slijed već je on razlomljen u više dijelova. Glavni cilj svakog sekvenciranja je zatvoriti sve procijepu. U cilju da matematički postave problem

sekvenciranja i sastavljanja genoma, Lander i Waterman u svome radu (Lander & Waterman, 1988) definiraju model preko kojeg pokušavaju doći do očekivanog broja dijelova sastavljenoga genoma. Njihova teorija taj broj aproksimira s:

$$E\langle \text{kontiga} \rangle = Ne^{-\sigma C} \quad (1.2)$$

pri čemu je  $\sigma$  udio duljine u kome se dva susjedna očitanja ne poklapaju.



Sl. 1.5 Primjer sekvenciranja u kome pojedini dijelovi originalnoga slijeda nisu pokriveni

## 1.3. Formati podataka

### 1.3.1. FASTA format podataka

Najčešći oblik podataka u bioinformatici je FASTA tekstualni format koji služi za prikazivanje sljedova proteina ili nukleinskih kiselina pri čemu je svaki nukleotid ili aminokiselina prikazan jednim slovom. Format je nastao prešutnim konsenzusom bioinformatičke zajednice, pa za njega ne postoji referentna specifikacija, iako je dobro opisan na stranicama NCBI-ja (<http://blast.ncbi.nlm.nih.gov/blastcgihelp.shtml>). Za prikaz nukleinskih kiselina osim slova koje predstavljaju nukleinske baze (A, C, G, T i U) koriste se i posebni znakovi različitoga značenja kao što su npr. znakovi za slučaj kada se ne može jednoznačno odrediti o kojoj se bazi radi i znakovi za procjepe u slijedu. Tablica 1.3 prikazuje potpuni prikaz kodova za

Tablica 1.3 Kodovi za prikaz nukleinskih kiselina

Kod nukleinske kiselina	Značenje
A	Adenin
C	Citozin
G	Gvanin
T	Timin
U	Uracil
R	A ili G

Y	C, T ili U
K	G, T ili U
M	A ili C
S	C ili G
W	A, T ili U
B	ne A (npr. C, G, T ili U)
D	ne C (npr. A, G, T ili U)
H	ne G (npr. A, C, T ili U)
V	niti T niti U (npr. A, C ili G)
N	A C G T U
X	Maskiranje
-	Procjep neodređene duljine

nukleinske kiseline.

U FASTA datoteci može biti više zapisa, a svaki zapis sastoji se od linije zaglavlja i jedne ili više linija samog slijeda. Isto tako mogu postojati komentari koji počinju sa znakom „“, ali se oni rijetko koriste.

Primjer FASTA datoteke:

```
>Slijed1 blok primjer 1
AGCTAGCT-CATAT
>Slijed2
AGCTATAGCTAGAC
>Slijed3
AGCTATANNNNNNNNNNNNGCTAGAC
```

Linija zaglavlja se razlikuje od linije slijeda time što počinje sa znakom veće od („>“) u prvom stupcu. Riječ koja slijedi nakon znaka „>“ je identifikator slijeda, a ostatak linije je opcionalan. Važno je napomenuti da ne smije biti razmaka između „>“ i prvoga slova identifikatora. Slijed može biti proizvoljne duljine i prostirati se u više linija. Slijed završava krajem datoteke ili sljedećom linijom koja počinje i predstavlja početak narednog zapisu. Ukoliko postoji samo jedan zapis u slijedu onda taj zapis ne mora imati liniju zaglavlja.

### 1.3.2. FASTQ format podataka

FASTQ format podataka koristi se za prikaz podataka na izlazu uređaja za sekvenciranja. Vrlo je sličan FASTA formatu, no dodatno ima oznaku kvalitete svakoga očitanja.

Primjer FASTQ zapisu:

```
@SEQ_ID
GATTTGGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTGTTCAACTCACAGTTT
```

```
+  
! ' ' * ( ( ( * * * + ) % % % + + ) ( % % % ) . 1 * * * - + * ' ' ) ) * * 55CCF>>>>>CCCCCCCC65
```

FASTQ zapis obično koristi 4 tipa linija po slijedu. Linije su:

- Linija 1 počinje sa znakom „@“, nakon njega je identifikator slijeda i opcionalan opis.
- Linija 2 je sam slijed.
- Linija 3 počinje s „+“ znakom, a opcionalno iza njega može biti isti identifikator slijeda (i bilo koji opis ponovo).
- Linija 4 predstavlja vrijednost kvalitete za slijed iz linije 2, s time da broj znakova mora biti jednak broju znakova u slijedu.

Isto kao i kod FASTA formata može biti više zapisa u datoteci, a sam slijed i niz znakova kvalitete se mogu prostirati u više redaka. Važno je napomenuti da treba biti pažljiv pri izvlačenju informacija iz ovoga tipa podataka zato što se znakovi „@“ i „+“ također mogu nalaziti u nizu znakova kvalitete.

Uređaji za sekvenciranje za svaku pojedinu bazu daju ocjenu kvalitete očitanja. Ta vrijednost se obično izražava kao vjerojatnost netočnoga očitanja  $p$ . Ta vrijednost se pretvara u takozvani Sangerov format kvalitete izrazom

$$Q_{\text{Sanger}} = -10 \log_{10} p$$

Na taj način dobijemo iznose kvalitete u rasponu od 0 do 93. S obzirom da su prvi 32 ASCII znakova kontrolni, a 32. znak razmak, kodiramo kvalitetu na način da dobivenom izrazu dodamo 33 i tako dobijemo znakove u rasponu od 33 do 126.

Za programsku analizu navedenih formata postoje stabilni i testirani programski paketi za nekoliko jezika – BioPython<sup>1</sup>, BioPerl<sup>2</sup>, BioJava<sup>3</sup>, BioRuby<sup>4</sup>.

### 1.3.3. Zapisи genoma

Jednom kad se genom organizma sekvencira i sastavi, može se predstaviti jednim dugačkim slijedom nukleotida. Duljina genoma mjeri se u jedinici bp (engl. *base pairs*).

---

<sup>1</sup> [http://biopython.org/wiki/Main\\_Page](http://biopython.org/wiki/Main_Page)

<sup>2</sup> [http://www.bioperl.org/wiki/Main\\_Page](http://www.bioperl.org/wiki/Main_Page)

<sup>3</sup> [http://biojava.org/wiki/Main\\_Page](http://biojava.org/wiki/Main_Page)

<sup>4</sup> <http://bioruby.org/>

Sastavljeni genom obično se sastoje isključivo od nukleotidnih baza A, T, G i C (bez više značnih baza iz Tablice 1.3). Iz toga je jasno da je za računalno kodiranje jedne baze potrebno 2 bita podataka. Neki komprimirani formati za zapis nukleotidnih sljedova koriste upravo 2 bita. Ipak, većina formata je tekstualna i služi se duljinom nukleotida od 8 bita, što je veličina podataka za kodiranje ASCII znakova.

## **1.4. Literatura**

Crick FHC (1956). Ideas on protein synthesis.

Available at: [http://profiles.nlm.nih.gov/SC/B/B/F/T/\\_scbbft.pdf](http://profiles.nlm.nih.gov/SC/B/B/F/T/_scbbft.pdf) (acknowledged by Crick in 1958)

Lander,E. & Waterman,M. (1988) Genomic mapping by fingerprinting random clones: a mathematical analysis. *Genomics*, **2**, 231–239.

## 2. Optimalno poravnavanje sljedova

### 2.1. Poravnanje

Poravnanje bioloških sljedova je najčešći prvi korak u bioinformatičkog analizi – bilo da je u pitanju pronalazak evolucijski očuvanih regija među vrstama, analiza genetske bolesti ili kreiranje rodovskog stabla. Upravo zbog svoje široke primjene, poravnanje dva biološka slijeda predstavlja jedan od najstarijih i najviše istraživanih problema u bioinformatici.

Problem je prvi put definiran 1950. godine u području teorije informacije kad je Richard Hamming definirao mjeru udaljenosti između dva niza znakova jednake duljine. Ako promatramo dva niza znakova  $s$  i  $t$ , njihova Hammingova udaljenost jednaka je broju pozicija gdje na kojima se ti znakovi razlikuju. Takva mjera udaljenosti se može razmatrati i kao minimalan broj zamjena znakova koje je potrebno napraviti da se niz  $s$  pretvori u niz  $t$ . Promotrimo, primjerice, nizove  $s=borba$  i  $t=torba$ . Naočigled je jasno da se iz niza  $s$  do niza  $t$  lako može doći zamjenom prvoga znaka niza  $s$ , te je stoga njihova međusobna udaljenost jednaka jedan.

Ovakva mjera udaljenosti korisna je za dva niza jednake duljine, ali to zasigurno nije prikladno ograničenje široko uporabljivanoj tehnički bioinformatičke analize. Vladimir Levenshtein je 1965. popravoio ovaj problem za dva niza proizvoljnih duljina. Mjera udaljenosti između znakova  $s$  i  $t$  po Levenshteinu je minimalni broj potrebnih modifikacija nad jednim znakom (umetanje, brisanje, izmjena) potrebnih za pretvaranje niza  $s$  u niz  $t$ , i naziva se udaljenost uređivanja (engl. „edit distance“). Tri dozvoljene operacije nad jednim znakom su:

- Zamjena – promjena jednoga znaka iz niza  $s$  u znak koji se nalazi na odgovarajućoj poziciji u nizu  $t$ .

Npr. *most* u *mast*

- Umetanje – umetanje jednoga znaka u niz  $s$  u cilju da odgovara nizu  $t$ .

Npr. *sir* u *svir*

- Brisanje – brisanje jednoga znaka iz niza  $s$  u cilju da odgovara nizu  $t$ .

Npr. *brod* u *rod*

Trošak svih uređivanja je isti i iznosi jedan.

Za primjer računanja udaljenosti uređivanja uzmimo dva niza  $s=TGCATAT$  i  $t=ATCCGAT$ .

Možemo primijetiti da niz  $s$  možemo pretvoriti u niz  $t$  kroz pet koraka:

- $TGCATAT$  – obrišemo znak  $T$  s kraja niza
- $TGCATA$  – obrišemo znak  $A$  s kraja niza
- $ATGCAT$  – umetnemo znak  $A$  na početak niza
- $ATGCGAT$  – umetnemo znak  $G$  ispred zadnjeg  $A$
- $ATGCTAT$  – zamijenimo znak  $G$  sa  $C$
- $ATCCGAT$  – kraj

Ovime smo dobili da je udaljenost između dva niza 5. Međutim, pretvorbu možemo obaviti i u samo četiri koraka:

- $TGCATAT$  – obrišemo znak  $A$  u sredini niza
- $ATGCTAT$  – umetnemo znak  $A$  na početak niza
- $ATGCTAT$  – zamijenimo  $G$  sa  $C$
- $ATCCTAT$  – zamijenimo  $T$  s  $G$
- $ATCCGAT$  – kraj

Na ovaj način smo dobili da je udaljenost određivanja 4.

Levenshtein nikada nije predložio algoritam za određivanje udaljenosti uređivanja. Algoritam za rješavanje ovoga problema je otkrivan i ponovo otkrivan u raznim područjima od automatskoga prepoznavanja govora do bioinformatike. Iako se zavisno o primjeni pojedine izvedbe algoritma razlikuju, zajedničko im je da je algoritam temeljen na dinamičkom programiranju – koje svoje ponešto neobično ime duguje Richardu Bellmanu, koji se za njega odlučio iz zanimljivih razloga.<sup>5</sup>

## 2.2. Algoritam za računanje udaljenosti uređivanja

Predstavimo poravnjanje nizova  $s$  i  $t$  duljina  $|s| = n$  i  $|t| = m$  s matricom od dva retka u kojoj prvi redak sadrži znakove niza  $s$  koji imaju zadržani poredak, drugi redak sadrži znakove niza  $t$  koji isto tako imaju zadržan poredak, dok se praznine mogu nalaziti na različitim mjestima unutar nizova. Kao rezultat znakovi u svakom nizu zadržavaju

---

<sup>5</sup> [http://en.wikipedia.org/wiki/Dynamic\\_programming#History](http://en.wikipedia.org/wiki/Dynamic_programming#History)

poredak, no nisu nužno susjedni. Praznine označavamo sa znakom razmaka “-“ i umećemo ih ako je došlo do operacija brisanja ili umetanja. Npr. prepostavimo da se radi o nizovima iz gornjega primjera gdje je  $s=TGCATAT$  i  $t=ATCCGAT$  i kako nema  $A$  na početku niza  $s$  potrebno ga je tamo umetnuti. Tu operaciju označimo s prazninom u nizu  $s$ . Možemo prepostaviti da ne postoji stupac u matrici poravnanja koji sadrži praznine u oba retka jer takva operacija ne bi imala smisla. Iz toga slijedi da poravnanje ima maksimalno  $n + m$  stupaca.

-	T	G	C	A	T	A	T
A	T	C	C	G	-	A	T

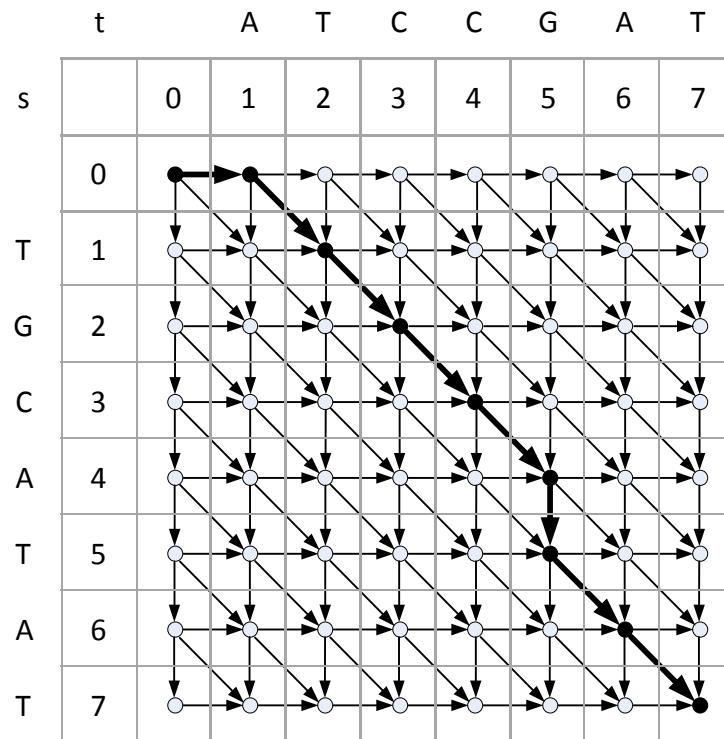
U stupcima u kojima su samo slova moglo je doći do zamjene slova. Slučaj kada se nije dogodila zamjena, odnosno slova su ista nazivamo slaganje (engl. *match*). Slučaj kada se dogodila zamjena zovemo neslaganje (engl. *mismatch*). Broj slaganja između nizova  $s$  i  $t$  je 4, a broj neslaganja je 2. S obzirom da želimo preuređiti niz  $s$  u niz  $t$ , prazninu u gornjem retku nazivamo umetanje znakova, a prazninu u donjem retku nazivamo brisanjem. Ukupno imamo dvije praznine što zbrojeno s dva neslaganja daje ukupnu udaljenost uređivanja 4.

Preklapanje možemo predstaviti i tako da svakom slovu u  $s$  i  $t$  dodijelimo indeks zavisno o njegovoj poziciji u nizu, a svaki redak predstavimo trenutnim indeksom za dani niz. Tako prvi redak  $-TGCATAT$  predstavimo s 01234567, a drugi redak  $ATCCG-AT$  s 12345567. Nula kod prvoga niza je zbog toga što počinjemo redak s prazninom. Predstavimo oba retka predstavimo nizom uređenih parova indeksa na svakoj poziciji. Dobijemo (0,1), (1,2), (2,3), (3,4), (4,5), (5,5), (6,6) i (7,7). Ono što možemo vidjeti je da u bilo kojem slučaju poravnanja najmanji indeks u oba niza će biti jednak nuli, a najveći će biti jednak duljini niza. Stoga možemo nacrtati mrežu dimenzija  $(m + 1) \times (n + 1)$  u kojoj će s indeksima biti označeno poravnanje koje predstavlja put:

$$(0,1) \rightarrow (1,2) \rightarrow (2,3) \rightarrow (3,4) \rightarrow (4,5) \rightarrow (5,5) \rightarrow (6,6) \rightarrow (7,7)$$

u mreži. Isto tako možemo vidjeti da svaki mogući put u mreži od  $(0,0)$  do  $(n,m)$  odgovara nekom poravnaju dva slijeda kao i da svako poravnanje odgovara nekom putu u mreži od  $(0,0)$  do  $(n,m)$ .

Mreža poravnjanja zapravo tvori graf u kome se kretanje između vrhova (nalaze se na križanju indeksa jednoga i drugoga niza) odvija preko bridova koji predstavljaju operacije uređivanja. Iz svakoga vrhova možemo se kretati u 3 smjera (desno, dolje, dijagonalno). Kretanjem desno ili dolje pomičemo se u jednom nizu, a da u drugom nizu ostajemo na istom mjestu. Ukoliko želimo niz  $s$  čiji indeksi označavaju redove pretvoriti u niz  $t$  čiji indeksi predstavljaju stupce, onda kretanje udesno predstavlja umetanje u niz  $s$  jer on tim kretanjem zadržava svoj indeks, dok se pomičemo za jedno mjesto u nizu  $t$ . Slično kretanje prema dolje, predstavlja brisanje jednoga znaka iz niza  $s$ . Kretanje dijagonalno može predstavljati slaganje ili neslaganje između dva niza. Svakom od ovih kretanja možemo dodijeliti određenu težinu koja predstavlja cijenu koštanja te operacije. U slučaju udaljenosti određivanja, najčešće ta cijena iznosi 1 za svako brisanje, umetanje i neslaganje.



Sl. 2.1 Mreža poravnjanja za nizove  $s=TGCATAT$  i  $t=ATCCGAT$ . Svako poravnanje odgovara nekom putu u mreži poravnjanja od  $(0,0)$  do  $(n,m)$  i svaki put od  $(0,0)$  do  $(n,m)$  odgovara jednom poravnjanju

Cilj je pronaći ono optimalno poravnanje za koje će udaljenost određivanja biti najmanja. S obzirom da svako poravnanje odnosno put završavaju u točki, odnosno vrhu  $(n,m)$ , problem je odrediti minimalnu cijenu puta do toga vrha. Možemo primjetiti da smo u taj

vrh mogli doći iz tri smjera: od lijevo ( $n, m-1$ ), od gore ( $n-1, m$ ) ili dijagonalno od lijevo gore ( $n-1, m-1$ ). Ukupna cijena u vrhu ( $n,m$ ) jednaka je minimumu cijena puta do tih vrhova plus troškovi puta od tih vrhova do vrha ( $n,m$ ). Cijenu puta do tih vrhova možemo odrediti na isti način gledajući njima lijeve, gornje i dijagonalno lijeve vrhove. Postupak se rekurzivno nastavlja. Važno je primijetiti da svi putovi imaju ishodište u vrhu  $(0,0)$  i da bi smo pronašli optimalno poravnanje ili put, moramo riješiti cijenu puta od njega do svih ostalih vrhova. Ako postavimo da je cijena puta u vrhu  $(0,0)$  0, cijenu puta za njemu susjedne vrhove (desno, dolje i desno dolje) možemo odrediti dodajući cijene putova do tih vrhova. Lako možemo nakon toga odrediti cijene puta za njima susjedne vrhove i u konačnici na taj način možemo odrediti cijenu puta do svakoga vrha u mreži. Algoritmi kojima prvo rješavamo manje podprobleme i progresivno rješavamo do krajnjega rješenja koristeći rezultate podproblema nazivamo dinamičko programiranje. Računanje poravnjanja se svodi na izračun vrijednosti u matrici dimenzija  $(m + 1) \times (n + 1)$ . Nazovimo tu matricu  $V$ , s  $d$  označimo cijenu kretanja desno i dolje (umetanje ili brisanje), a sa  $w(s_i, s_j)$  cijenu zamjene dva znaka odnosno kretanja dijagonalno dolje. Sl. 2.1 pokazuje da možemo, osim za prvi stupac i prvi redak, vrijednost pojedinog člana matrice izračunati poznavanjem vrijednosti za njegove susjede lijevo, gore i lijevo gore. Vrijednosti za prvi redak i stupac su tzv. početni uvjeti koji se lako dobiju računanjem udaljenosti od  $(0,0)$ . Cijena svakoga pomaka je  $d$  tako da su vrijednosti za rubne uvjete linearni s  $d$ . Stoga za rješavanje matrice  $V$  možemo koristiti relaciju:

$$V(i,j) = \begin{cases} 0 & i = 0 \wedge j = 0 \\ d * i & j = 0 \\ d * j & i = 0 \\ \min \begin{cases} V(i-1, j-1) + w(s_i, t_j) \\ V(i-1, j) + d \\ V(i, j-1) + d \end{cases} & \text{inače} \end{cases} \quad (2.1)$$

Udaljenost poravnjanja će biti rezultat u  $(m,n)$ , no nemamo informaciju o putu do toga vrha, odnosno poravnjanju. Za tu informaciju potrebno je za svaki element matrice zapamtiti smjer od kuda se do njega došlo koristeći relaciju (2.1). Pretragom unatrag od elementa  $(m,n)$  do  $(0,0)$  i koristeći informaciju o smjerovima možemo rekonstruirati poravnanje.

Kôd 2.1 prikazuje pseudo kod algoritma za računanje udaljenosti određivanja dva niza  $s$  i  $t$  bez rekonstrukcije. Za rekonstrukciju potrebno je pamtitи za svaki element matrice dodatnu

informaciju o smjeru iz kojega smo došli u taj element. Koristeći tu informaciju vraćajući se iz točke  $(m,n)$  možemo napraviti potpunu rekonstrukciju.

S obzirom da za računanje udaljenosti određivanja treba izračunati elemente matrice veličine  $(m + 1) \times (n + 1)$ , vremenska složenost algoritma je  $O(n*m)$ . Iz izraza (2.1) može se vidjeti da je za računanje vrijednosti u svakom elementu dovoljno znati samo elemente iz njegovog retka te retka iznad što nam daje linearnu memorijsku složenost  $O(\max(n,m))$ . Međutim ako nam samo iznos poravnjanja nije dovoljan zbog potreba rekonstrukcije, memorijska složenost je  $O(n*m)$ .

```

EDITDISTANCE( s , t )
1   V[ 0 , 0 ] = 0
2   for i ← 1 to | s | do
3       V[ i , 0 ] = d * i
4   end
5   for j ← 1 to | t |
6       V[ 0 , j ] = d * j
7   end
8   for i ← 1 to | s | do
9       for j ← 1 to | t | do
10          MATCH = V[ i - 1 , j - 1 ] + w( s[ i ] , t[ j ] )
11          INSERTION = V[ i , j - 1 ] + d
12          DELETION = V[ i - 1 , j ] + d
13          V[ i , j ] = min( MATCH , INSERTION , DELETION )
14      end
15  end

```

Kôd 2.1 Pseudo kod algoritma za računanje udaljenosti uređivanja

## 2.3. Needleman-Wunsch algoritam

Važno je napomenuti da algoritam za računanje udaljenosti uređivanja nije bio prvi algoritam koji je efikasno riješio problem poravnanja dva niza. Rješenje je došlo iz sasvim drugoga smjera. Zbog potreba razvoja biologije, došlo je do potrebe za poravnanjem bioloških nizova. Biologe je prvenstveno zanimala maksimalna sličnost dvaju bioloških sljedova. Taj problem su prvi formulirali Needleman i Wunsch (Needleman and Wunsch, 1970) te predložili prvi algoritam za njegovo rješavanje temeljen na dinamičkom programiranju. Zanimljivo je da je taj prvi algoritam imao kubnu složenost. Prvi algoritam s kvadratnom složenošću je predložio David Sankoff (Sankoff, 1972). Slijedeći važan

korak je bio rad Petera H. Sellersa (Sellers, 1974) koji je pokazao da su problemi maksimiziranja sličnosti i minimiziranja udaljenosti uređivanja ekvivalentni.

Danas algoritam kojeg nazivamo Needleman-Wunschovim za računanje sličnosti nizova s i t je temeljen na relaciji (2.2). Ako pogledamo tu relaciju možemo vidjeti da je jako slična relaciji koju koristimo za određivanje udaljenosti određivanja. Temeljna razlika je u tome što maksimaliziramo vrijednosti u matrici te da su kažnjavanja negativne vrijednosti. Isto tako, dok kod udaljenosti uređivanja cijene slaganja su nula i obično koristimo istu vrijednost za kažnjavanje zamjene, brisanja i umetanja i najčešće je ta vrijednost 1, kod Needleman-Wunschovog algoritma, zbog bioloških razloga, te vrijednosti se razlikuju. Slaganja imaju pozitivne vrijednosti dok kažnjavanja uslijed neslaganja te umetanja i brisanja imaju negativne vrijednosti. Vrijednosti slaganja i kažnjavanja mogu varirati zavisno o pojedinim nukleotidima odnosno parovima nukleotida. Dodatno se te vrijednosti razlikuju od vrijednosti za umetanje ili brisanje.

$$V(i,j) = \begin{cases} 0 & i = 0 \wedge j = 0 \\ d * i & j = 0 \\ d * j & i = 0 \\ \max \begin{cases} V(i-1, j-1) + w(s_i, t_j) \\ V(i-1, j) + d \\ V(i, j-1) + d \end{cases} & \text{inače} \end{cases} \quad (2.2)$$

Kod algoritma je sličan kodu algoritma za računanje udaljenosti određivanje jedino se početni uvjeti i elementi matrice računaju prema relaciji (2.2).

S obzirom da se i u ovome algoritmu traži put odnosno poravnanje između (0,0) i (n,m) odnosno od početka do kraja oba slijeda, ovaj algoritam nazivamo algoritam za globalno poravnanje. Njegove vremenske i memorejske složenosti su također  $O(n*m)$ , s time da ako računamo isključivo sličnost, ne i poravnanje, onda je njegova memorejska složenost linearна.

## 2.4. Algoritam preklapanja

Čest problem u biologiji je potreba za traženje preklapanja zadanoga uzorka s dugačkim DNA slijedom (Sl. 2.2). Primjeri takvih poravnjanja su traženje eksona u DNA slijedu, mapiranje kratkih očitanja na poznati genom, traženja preklapanja među očitanjima u cilju sastavljanja genoma i slično. Ovaj problem se može svesti na traženje preklapanja dva

niza, odnosno poravnanje sufiksa jednoga niza s prefiksom drugoga ili poravnanja jednoga s podnizom drugoga. Preklapanje dva slijeda možemo definirati kao poravnanje u kome su praznine (brisanja i umetanja) na početku i kraju nizova zanemarene.



Sl. 2.2 Traženje preklapanja zadanog uzorka s dugačkim DNA slijedom

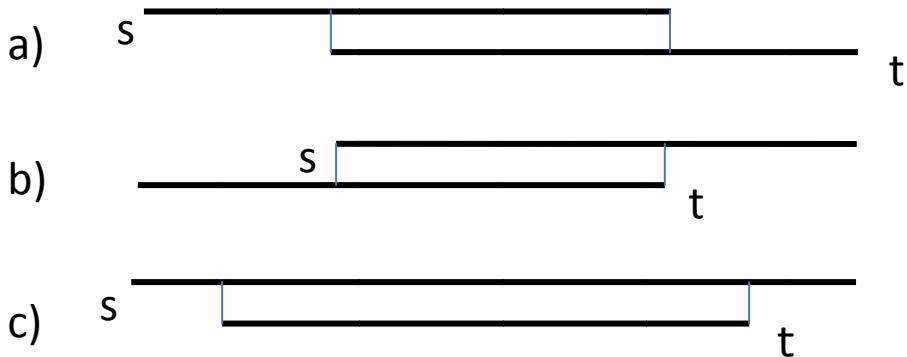
Ako za primjer uzmemo dva slijeda  $s=CAGCACTTGGATTCTCGG$  i  $t=CAGCGTGG$ , njihovo globalno poravnanje je:

```
CAGCACTTGGATTCTCGG
CAGC----G--T----GG
```

Međutim, ono što bi željeli postići je:

```
CAGCA-CTTGGATTCTCGG
---CAGCGTGG-----
```

Sl. 2.3 prikazuje različita moguća preklapanja dvaju nizova  $s$  i  $t$ . Radi pojednostavljenja za sada prepostavljamo da se oba niza prostiru u istom smjeru, od desna na lijevo.



Sl. 2.3 Primjeri različitih preklapanja dvaju nizova  $s$  i  $t$ . a) Sufiks od  $s$  se preklapa s prefiksom od  $t$ .

b) Sufiks od  $t$  se preklapa s prefiksom od  $s$ . c) Niz  $t$  se u cijelosti preklapa s podnizom od  $s$  (ili obrnuto).

S obzirom da obično želimo poravnati samo dio jedne ili obje sekvene, ovo poravnanje nazivamo polu-globalno poravnanje ili preklapanje. Kod preklapanja je bitno primijetiti da za razliku od klasičnog globalnog poravnanja koristeći udaljenost uređivanja ili Needleman-Wunschov algoritam, u ovom slučaju ne želimo kažnjavati praznine na

početku ili kraju pojedinoga niza. Praznine na početku niza možemo omogućiti tako da prilikom inicijalizacije ih ne penaliziramo. Praznine na kraju niza možemo omogućiti tako da kao kraj poravnanja uzimamo maksimalnu vrijednost u zadnjem retku ili zadnjem stupcu.

Zavisno o zahtjevima možemo definirati nekoliko slučajeva i njima pripadnih algoritama preklapanja za nizove  $s$  i  $t$  (niz  $s$  se nalazi u stupcu, a niz  $t$  u retku):

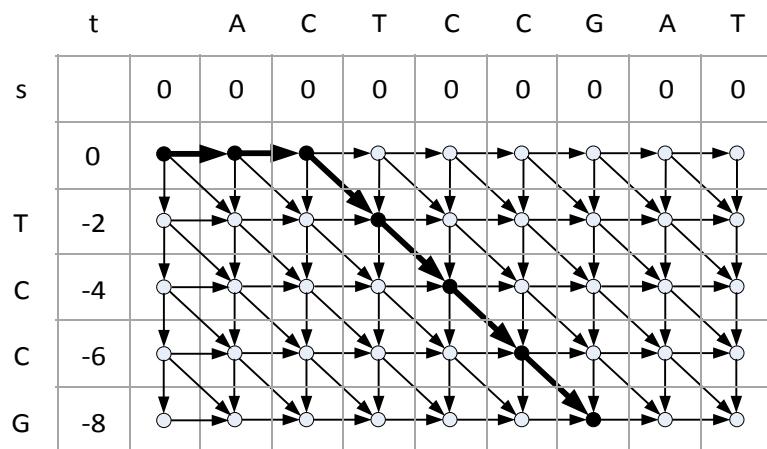
1. Preklapanje prefiksa niza  $s$  sa sufiksom niza  $t$  – inicijaliziramo prvi stupac s  $d^*j$ , prvi redak s nulama, a kraj poravnanja tražimo u maksimumu zadnjeg stupca matrice.
2. Preklapanje sufiksa niza  $s$  sa prefiksom niza  $t$  – inicijaliziramo prvi stupac s nulama, prvi redak s  $d^*j$ , a kraj poravnanja tražimo u maksimumu zadnjeg retka.
3. Preklapanje sufiks-prefiks dva niza, pri čemu oba niza mogu imati i sufiks i prefiks poravnanja – inicijaliziramo i prvi redak i prvi stupac s nulama, a kraj poravnanja tražimo u maksimumu zadnjeg retka ili stupca zavisno koji je veći.
4. Preklapanje u potpunosti niza  $s$  s podnizom niza  $t$  – inicijaliziramo prvi stupac s  $d^*j$ , prvi redak s nulama, a kraj poravnanja tražimo u maksimumu zadnjega retka.

Sl. 2.4 prikazuje primjer poravnanja za slučaj 4 za nizove  $s=TCCG$  i  $t=ACTCCGAT$ .

Dobiveno poravnanje je:

--TCCG--

ACTCCGAT



Sl. 2.4 Preklapanje dva niza  $s$  i  $t$  pri čemu želimo cijeli niz  $s$  poravnati s podnizom niza  $t$ . U ovom slučaju koristimo modificirani Needleman-Wunsch algoritam (cijene: slaganje +4, neslaganje -1, praznina -2). S obzirom da želimo poravnati cijeli niz  $s$  penaliziramo inicijalna brisanja. Kod niza  $t$

želimo omogućiti poravnanje podniza stoga ne penaliziramo inicijalna umetanja. Kako ne želimo penalizirati praznine na kraju niza  $t$  u zadnjem retku odabiremo element matrice s maksimalnim rezultatom i od njega praćenjem unatrag računamo put, odnosno poravnanje

## 2.5. Lokalno poravnanje

Prva poravnaja bioloških sljedova rađena su početkom 1970-tih. U to doba metode za određivanje nizova nukleinskih kiselina nisu bile toliko razvijene pa su se uglavnom poravnavali proteinski sljedovi. U tim ranim danima uspoređivali su proteini podjednake duljine. Međutim, uskoro su se pojavili slučajevi u kojima su proteini dijelili samo izolirane regije sličnosti. Za poravnanje takvih regija mjeru globalnog poravnanja i Needleman-Wunschov algoritam nisu bili dobro prilagođeni. Stoga se pojavila potreba za novom definicijom lokalne sličnosti kao i novim algoritmima za pronađak optimalnoga lokalnog poravnanja.

Kod lokalnog poravnanja važno je utvrditi regije čije će poravnanje imati najveći rezultat. Za utvrđivanja takvih regija puno je pogodnija maksimizacija sličnosti nego minimizacija udaljenosti određivanja. Kada promatramo sličnost dva slijeda, ta vrijednost pada i raste kako se krećemo kroz oba slijeda. U slučaju kada sličnost padne ispod nula, nema više smisla govoriti o sličnosti. Ako promatramo put poravnanja u matrici od neke krajnje točke prema početku i nađemo na element čija je vrijednost manja od nule možemo utvrditi da poravnanja prije te točke u smjeru prema  $(0,0)$  ne bi povećalo ukupan rezultat nego ga umanjilo. Stoga u takvim elementima možemo zamijeniti tu vrijednost s nula. Time dobivamo da poravnanje počinje od te točke nadalje. Kod udaljenosti određivanja je teško definirati vrijednost ekvivalentnu negativnoj vrijednosti kod određivanja sličnosti jer tamo vrijednost monotono raste. S obzirom da želimo odrediti lokalno poravnanje, nema potrebe za penaliziranjem početnih praznina.

Algoritam lokalnog poravnanja možemo konstruirati koristeći razmatranja iz prošloga odlomka:

- Ne penaliziramo praznine na početku sljedova.
- U slučaju da vrijednost u nekom elementu matrice padne ispod nula tu vrijednost zamijenimo s nula.
- Nađemo maksimalnu vrijednost u cijeloj matrici i od nje tražimo poravnanje praćenjem unatrag.

Korištenje ovoga načina računanja poravnanja nazivamo Smith-Waterman algoritam (Smith and Waterman, 1981). Ovaj algoritam je jedan od najpoznatijih u bioinformatici. Računanje matrice sličnosti definiramo na slijedeći način:

$$V(i,j) = \begin{cases} 0 & i = 0 \vee j = 0 \\ \max \begin{cases} V(i-1, j-1) + w(s_i, t_j) \\ V(i-1, j) + d \\ V(i, j-1) + d \end{cases} & \text{inače} \end{cases} \quad (2.3)$$

Kôd 2.2 prikazuje pseudo kod Smith-Waterman algoritama za dva niza  $s$  i  $t$ . Pseudo kod ne sadrži dio kojim se određuje samo poravnanje. Poravnanje se dobije praćenjem unatrag od elementa matrice koji ima maksimalnu vrijednost.

```

SMITHWATERMAN( s , t )
1   M = 0
2   V[ 0 , 0 ] = 0
3   for i←1 to |s| do
4       V[ i , 0 ] = 0
5   end
6   for j←1 to |t|
7       V[ 0 , j ] = 0
8   end
9   for i←1 to |s| do
10      for j←1 to |t| do
11          MATCH = V[ i-1 , j-1 ] + w( s[ i ] , t[ j ] )
12          INSERTION = V[ i , j-1 ] + d
13          DELETION = V[ i-1 , j ] + d
14          V[ i , j ] = max( 0 , MATCH , INSERTION , DELETION )
15      end
16      M = max( M , V[ i , j ] )
17  end

```

Kôd 2.2 Pseudo kod Smith-Waterman algoritma bez utvrđenja puta poravnanja

Sl. 2.5 prikazuje primjer poravnanja za nizove  $s=ACCTAAGG$  i  $t=GGCTCAATCA$  koristeći +2 za podudaranje, -1 za zamjenu i -2 za prazninu. U svaki element matrice se spremaju trenutni rezultat i smjer od kuda se došlo. U varijablu  $M$  sprema se trenutno najveća vrijednost u matrici. U slučaju primjera sa slike 2.5 najveća vrijednost je 6 i element s tom vrijednošću predstavlja kraj poravnanja koje se dobije vraćanjem unatrag od toga elementa. Za ovaj primjer poravnanje je:

CTC  
 || |  
 CT-AA

U prikazu poravnjanja između dva niza često koristimo oznake “|“ koje povezuju pojedine znakove u slučaju da je došlo do slaganja.

	G	G	C	T	C	A	A	T	C	A
A	0	0	0	0	0	0	0	0	0	0
C	0	0	0	2	0	2	0	1	1	2
C	0	0	0	2	1	2	1	0	0	3
T	0	0	0	0	4	2	1	0	2	1
A	0	0	0	0	2	3	4	3	1	1
A	0	0	0	0	0	1	5	6	4	2
G	0	2	2	0	0	0	3	4	5	3
G	0	2	4	-2	0	0	1	2	3	4

Sl. 2.5 Primjer poravnjanja dva niza koristeći Smith-Waterman algoritam. Poravnavamo nizove  $s=ACCTAAGG$  i  $t=GGCTCAATCA$  koristeći +2 za podudaranje, -1 za zamjenu i +2 za prazninu. Prvi redak i prvi stupac su inicijalizirani s nulama. Crvene strelice pokazuju smjerove u kojima je moguće ići pretraživanjem unatrag za sve elemente matrice koji nisu došli od elementa s vrijednošću 0. Narančasto je označeno optimalno poravnanje.

Da bi smo dobili lokalno poravnanje dvaju nizova duljine  $m$  i  $n$  korištenjem Smith-Waterman algoritma potrebno je kao i kod globalnoga poravnjanja riješiti sve elemente matrice dimenzija  $(m+1) \times (n+1)$ , te sačuvati sve vrijednosti smjerova. Stoga je također i vremenska i memorijska složenost  $O(nm)$ .

## 2.6. Optimizacije primjenjive na algoritme za poravnanje sljedova

Kvadratna vremenska i memorijska složenost algoritama za poravnanje sljedova temeljenih na dinamičkom programiranju predstavlja velik problem za dulje sljedove. Tijekom godina razvilo se nekoliko algoritama koji smanjuju vremensku ili memorijsku

složenost, a u ovom poglavlju naglasak će biti na algoritmima za globalno poravnjanje iako se većina ovih tehnika može primijeniti i za lokalno i polu-globalno poravnjanje.

### 2.6.1. Linearna memorijska složenost

S obzirom na današnja računala nešto veći izazov za poravnjanje dva niza predstavlja kvadratna memorijska složenost od one vremenske. Npr. poravnjanje ljudskoga genoma s genomom miša (oba duljine  $3 \times 10^9$  nukleotida) bi zahtijevalo  $9 \times 10^{18}$  mjesta u matrici.

Razlog za kvadratnu memorijsku složenost je potreba za pamćenjem smjerova za svaki element matrice za potrebe određivanja poravnanja. Stoga se svi algoritmi za smanjenje memorijske složenosti temelje na određivanju poravnanja bez prethodnoga poznavanja cijele matrice poravnanja.

Vrlo je važno primijetiti da za poravnanje dva niza dinamičkim programiranjem vrijedi sljedeće:

- Za izračun svakog elementa matrice potrebno je samo znati vrijednosti u trenutnom i prethodnom retku. Dodatno, za rješavanje cijelog retka dovoljno je poznavati vrijednosti u prethodnom retku i vrijednost rubnog uvjeta u prvom stupcu.
- Ako globalno poravnavamo niz  $s = s_1s_2s_3\dots s_m$  i niz  $t = t_1t_2t_3\dots t_n$  i kao rezultat dobijemo poravnanje  $p = p_1p_2p_3\dots p_k$  pri čemu je  $k \leq m + n$ . Ako s  $s^r$  označimo reverzni niz od  $s$  tako da je  $s^r = s_ms_{m-1}s_{m-2}\dots s_1$  i analogno označimo  $t^r$  i  $p^r$  možemo lako pokazati da je  $p^r$  optimalno poravnanje od  $s^r$  i  $t^r$ . Kao posljedicu toga možemo reći da je rezultat poravnanja dva niza jednak ukoliko rješavamo matricu od elementa u  $(0,0)$  ili elementa  $(m,n)$ .

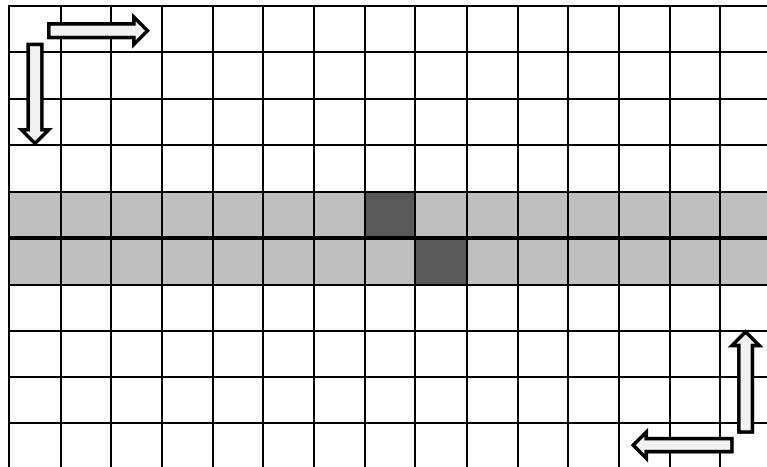
Uvezši u obzir gornje navode najjednostavniji algoritam za izračun poravnanja sa složenošću manjom od kvadratne možemo dobiti ako pamtimo samo svaki  $k$ -ti redak matrice. Pri tome je dovoljno da pamtimo samo vrijednosti elemenata, ne i smjer. Kada smo završili s računanjem svih vrijednosti u matrici krenemo ponovno računati od zadnjeg zapamćenoga retka prema kraju s time da ovaj put pamtimo samo smjer za svaki element. Kada dođemo do kraja vraćanjem unatrag dođemo do poravnanja između zadnjeg zapamćenoga retka i kraja. Analogno rješavamo za svih  $k$  zapamćenih redaka. Između bilo koja dva susjedna zapamćena retka ostalo je za izračunati  $\frac{m}{k}$  redaka. Nakon što izračunamo put poravnanja između njih te nam vrijednosti ne trebaju više u memoriji stoga nam je za računanje u svakom trenutku potrebno zapamtiti maksimalno  $k + \frac{m}{k}$  redaka. Ukoliko

odaberemo  $k = \sqrt{m}$ , onda nam je ukupna memorijska složenost  $O(n\sqrt{m})$ . S obzirom da krećemo ponovo iz svakoga  $k$ -tog retka, cijelu matricu rješavamo dva puta te nam je stoga ukupna vremenska složenost povećana dva puta.

Algoritam s linearnom memorijskom složenošću je predstavio Dan Hirschberg (Hirschberg, 1975). Osnovna ideja algoritma je slijedeća. Promatramo dva niza  $s$  i  $t$  duljina  $m$  i  $n$  te njihove reverzne nizove  $s^r$  i  $t^r$ . S  $V(i,j)$  označimo optimalan rezultat poravnjanja  $s_1...s_i$  s  $t_1...t_j$ , a s  $V^r(i,j)$  optimalan rezultat poravnjanja  $s^r_1...s^r_i$  s  $t^r_1...t^r_j$ . Lako se može pokazati da je  $V^r(i,j)$  jednak poravnaju  $s_{m-i+1}...s_m$  s  $t_{n-j+1}...t_n$ . Pojednostavljeni kažemo da rezultat poravnjanje zadnjih  $i$  znakova iz niza  $s$  i zadnjih  $j$  znakova iz niza  $t$  je jednak rezultatu reverznog poravnjanja tih znakova. Koristeći to uz pretpostavku da je  $m$  paran možemo pokazati da je

$$V(m, n) = \max_{k=0...n} (V\left(\frac{m}{2}, k\right) + V^r\left(\frac{m}{2}, n - k\right)) \quad (2.4)$$

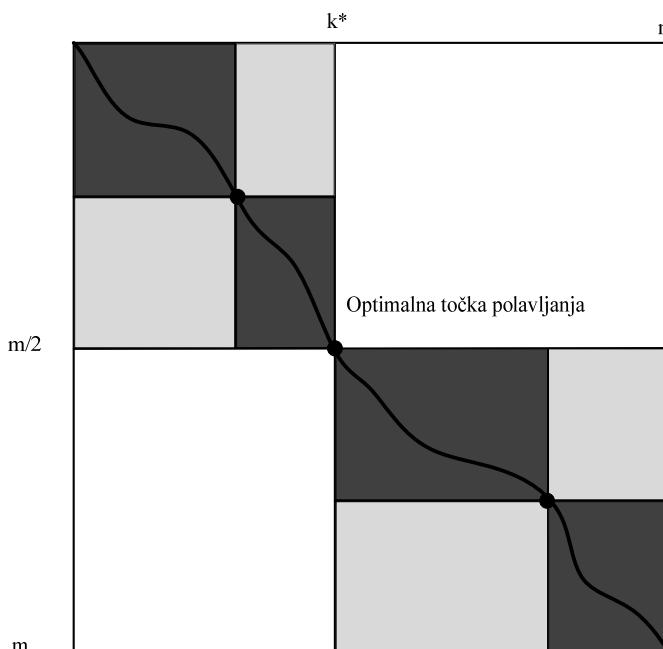
odnosno da ukupno poravnanje možemo gledati kao zbroj poravnanja prve polovice niza  $s$  s nizom  $t$  i reverznog poravnanja druge polovice niza  $s$  s nizom  $t$ . Cilj je pronaći indeks  $k$  niza  $t$  za koji je zbroj dva poravnanja maksimalan. Taj indeks obilježavamo s  $k^*$ . Praktično algoritam dijeli matricu rješavanje na dvije polovice te radi na principu spajanja polovica putova rekonstrukcije u jednu cjelinu. Sl. 2.6. pokazuje prvi korak u nakon što riješimo



Sl. 2.6 Prvi korak Hirschbergova algoritma. Odvojeno se rješavaju gornja i donja polovica matrice, s time da donju polovicu rješavamo u reverznom smjeru. Nakon toga promatramo zadnje redove poravnjanja (označeni sivo). U tim zadnjim redovima odabiremo one susjedne elemente matrice čiji zbroj rezultata je najveći (označeni tamno sivo).

gornju i donju polovicu matrice promatramo zadnje retke. Kod gornje polovice to je njezin zadnji redak, kod donje polovice to je prvi redak, odnosno zadnji kada rješavamo reverzno iz donjeg desnoga kuta.

S obzirom da gledamo ukupno poravnanje tražimo rezultat koji će biti ukupno maksimalan, stoga promatramo susjedne elemente matrice (elemente koji su mogli nastati procesom poravnjanja – za elemente gornje polovice to su elementi ispod i desno ispod) i tražimo one čija je suma rezultata poravnjanja maksimalna. Kada odredimo te susjedne elemente znamo da se oni nalaze na putu optimalnoga poravnjanja i onda rekursivno dalje odredimo sve preostale dijelove poravnjanja.



Sl. 2.7 Hirschbergov algoritam. Algoritam u prvom koraku traži na polovici matrice indeks  $k^*$  niza  $t$  za kojeg je zbroj poravnanja obje polovice matrice maksimalan. Nakon toga se postupak rekursivno ponavlja u dobivenim podmatricama i na taj način dobivamo točku po točku poravnjanja.

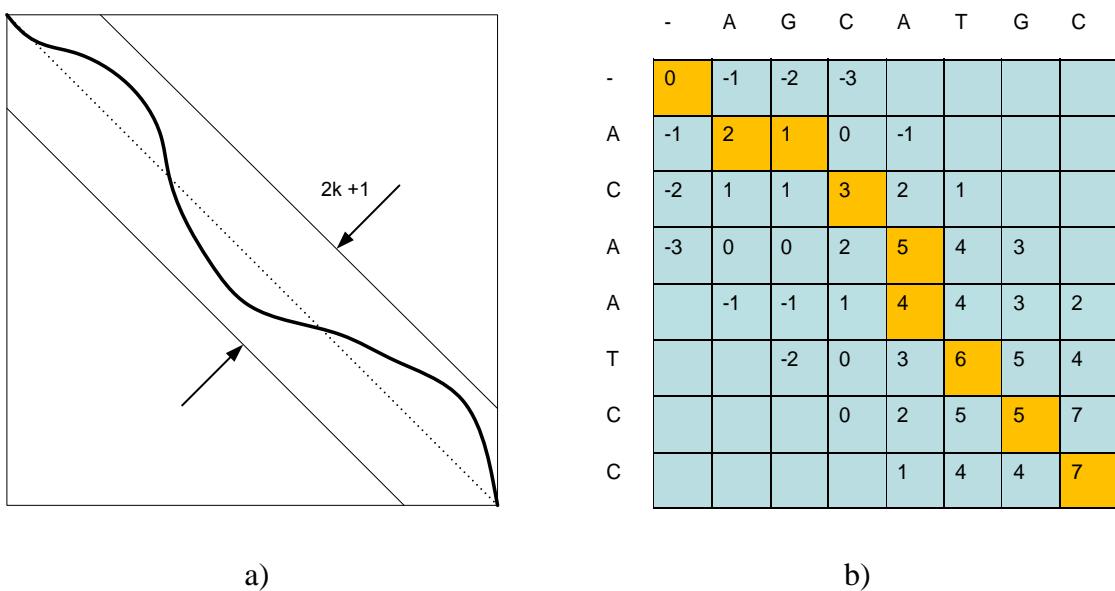
Sl. 2.7 prikazuje korake Hirschbergovog algoritma. Važno je primijeti da nakon prvoga koraka površine označene bijelom bojom ne treba rješavati jer se poravnanje ne nalazi u tim dijelovima matrice. Njihova površina je jednaka polovici ukupne površine matrice. U prvom smo koraku morali izračunati vrijednosti za sve elemente matrice, u drugom je dovoljno izračunati samo za pola i tako dalje rekursivno. Iz ovoga možemo vidjeti da je ukupno vrijeme izvođenje proporcionalno:

$$(m * n) * \left(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots\right) = (m * n) * \sum_{i=0}^{\infty} \frac{1}{2^i} = (m * n) * 2 \quad (2.5)$$

što znači da je ovaj algoritam samo dva puta sporiji od Needleman-Wunschovog algoritma. Kako je dovoljno čuvati maksimalno dva prethodna retka u tablici ( $2n$ ), te moramo odrediti maksimalno poravnanje dugačko  $(m+n)$  ukupna memorija složenost je  $O(3n+m) = O(n+m)$ .

### 2.6.2. Dinamičko programiranje s ograničenim pojasom

Kao što je pokazano algoritmi dinamičkoga programiranja uzimaju  $O(nm)$  vremena. Ako želimo s tim algoritmima poravnati genome čovjeka i miša duge nekoliko milijardi baza trebat će nam minimalno  $9 \times 10^{18}$  operacija. Uz radni takt od 1GHz za računanje ovoga poravnanja trebat će nam preko 200 godina. Dodatno, pokazano je da se općenita vremenska složenost poravnanja dinamičkim programiranje teško može popraviti. Međutim, postoje posebni slučajevi u kojima je to moguće. Jedan specijalan slučaj je ograničenje maksimalnoga broja unesenih praznina pri poravnanju. Označimo taj broj s  $k$ . Očito je da vrijedi  $0 < k \leq n + m$ . Podsetimo se da u matrici  $V$  umetanje odgovara vodoravnom pomicanju, a brisanje okomitom. Ako smo ograničili poravnanja s  $k$  praznina onda to poravnanje mora biti unutar pojasa širine  $(2k + 1)$ . Sl. 2.8 prikazuje primjer dinamičkoga programiranja s ograničenim pojasm.



Sl. 2.8 Poravnanje s ograničenim pojasmom. a) Prikaz  $2k + 1$  pojasa unutar matrice dinamičkog programiranja. b) Primjer poravnanja koristeći ograničeni pojaz  $k=3$

S obzirom da smo definirali ograničeni pojas nema potrebe ni rješavati dijelove matrice izvan pojasa. Površina matrice  $V$  koju je potrebno izračunati je  $nm - (n-k)(m-k) = mk + nk - k^2$ . Za  $k \ll \min(m, d)$  i pretpostavku da za popunjavanja svakoga elementa matrice treba  $O(1)$ , vremenska složenost algoritma je  $O((n+m)k)$ . Pažljivom izvedbom postiže se ista memorijска složenost.

Ovakvi algoritmi se danas često koriste u slučajevima kada znamo da je udaljenost određivanja između dva slijeda mala. Najčešća primjena je u algoritmima za poravnjanja očitanja dobivenih DNA sekvenciranjem na već poznati, tzv. referentni genom.

Dodatno se ovaj algoritam može dodatno ubrzati i smanjiti korištenje memorije (Myers, 1986; Ukkonen, 1985). Isto tako danas se često koriste SIMD (engl. *single instruction multiple data*) instrukcije u procesorima pa se time postižu dodatna ubrzanja (Myers, 1999).

## 2.7. Biološka usporedba nizova

Konstruiranje smislenoga poravnjanja dvaju nizova zahtijeva korištenje odgovarajućih funkcija zamjene između svih parova simbola unutar abecede, te kažnjavanja praznina. Funkcije zamjene su obično realizirane kao matrice sličnosti.

### 2.7.1. Matrice sličnosti

Zbog toga što se DNA može naći samo četiri vrste nukleotida, te su matrice puno jednostavnije od matrica sličnosti za proteine. Najjednostavnije matrice za Needleman-Wunschov algoritam pridjeljuju +1 za slaganje, a -1 za neslaganje baza, dok za udaljenost uređivanja obično koristimo 0 za slaganje i +1 za neslaganje. Složenije matrice koriste dodatna biološka znanja. Između ostalog nukleotide može klasificirati u purine (adenin i gvanin) ili pirimidine (citozin i timin). Mutacija nukleotida pri čemu ostajemo u istoj skupini odnosno mutacija purin→purin ili pirimidin→pirimidin se naziva tranzicija. Mutacija u kojoj se promijeni skupina odnosno purin→pirimidin ili pirimidin→purin se naziva transverzija. Biološki su tranzicije češće nego transverzije te se zbog toga i više kažnjavaju.

Sl. 2.9 prikazuje dio EDNAFull matrice koja se jako često koristi za slučaj kada koristimo algoritme koji traže optimalnu sličnost. Sl. 2.10 prikazuje matricu koju koristimo kada želimo odrediti sličnost dva slijeda, a različito kažnjavamo tranziciju i transverziju.

	A	T	C	G
A	5	-4	-4	-4
T	-4	5	-4	-4
C	-4	-4	5	-4
G	-4	-4	-4	5

Sl. 2.9 Dio EDNAFull matrice koji se najčešće koristi s algoritmima koji maksimiziraju sličnost. Ostatak matrice se odnosi na kodove koji predstavljaju situacije u kojima je nejasno o kojem se točno nukleotidu radi.

	A	T	C	G
A	0	5	5	1
T	5	0	1	5
C	5	1	0	5
G	1	5	5	0

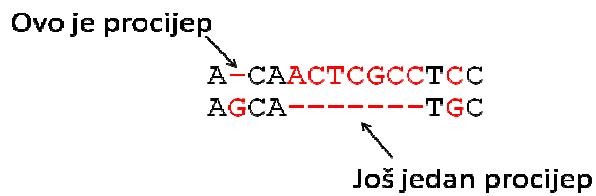
Sl. 2.10 Matrica koja se koristi s algoritmom uređivanje udaljenosti i kada se vodi računa o tome da li dolazi do tranzicije ili transverzije.

Kod proteinskih sekvenci matrice su komplikiranije. Koriste se PAM (engl. *point accepted mutations*) i BLOSUM (engl. *block substitution*) matrice koje odražavaju frekvenciju s kojom pojedina aminokiselina  $x$  je zamijenjena aminokiselinom  $y$  u evolucijski srodnim sljedovima. Danas se najčešće koristi BLOSUM 62 matrica. Procedura (Henikoff and Henikoff, 1992) korištena za procjenu BLOSUM matrica je bila vrlo jednostavna. Henikoffovi su uzeli veliku bazu pouzdanih poravnanja i samo brojali poravnanja sljedova koja su imala postotak identičnosti jednak ili manji od nekoga praga. Tako je npr. prag od 62% identičnosti korišten za izračun frekvencija zamjena za BLOSUM 62 matricu.

## 2.7.2. Praznine i procijepi

Mutacije u DNA su obično posljedica pogrešaka prilikom umnažanja DNA. U prirodi se vrlo često brišu ili umeću cijeli podnizovi, umjesto umetanja ili brisanja pojedinih nukleotida. Procijep u poravnaju definiramo kao neprekidni niz praznina u jednom od redaka poravnanja (Sl. 2.11). S obzirom da su umetanja i brisanja evolucijski česta, kažnjavanje procijepa duljine  $x$  sa  $xd$  (gdje je  $d$  cijena praznine) je prestroga. Stoga mnogi

praktični algoritmi koriste blaži pristup za kažnjavanje procijepa i kažnjavaju procijep duljine  $x$  s funkcijom koja puno sporije raste nego suma kažnjavanja za  $x$  praznina.



Sl. 2.11 Poravnanje dva niza s označenim procijepima. Procijep se definira kao neprekinuti niz poravnanja.

Iako bi idealna funkcija kažnjavanja procijepa bila konveksna (npr. logaritamska) takve funkcije se ne koriste jer algoritmi koji to podržavaju imaju  $O(n^3)$  vremensku složenost što ih čini presporima za praktične svrhe. Stoga se danas često konveksna funkcija, aproksimira s afinom funkcijom. Takva funkcija ima oblik  $d+xe$ , pri čemu je  $d$  cijena otvaranja novoga procjepa, a  $e$  cijena svakog produženja procjepa.

## 2.8. Literatura

- Henikoff,S. and Henikoff,J.G. (1992) Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci. U. S. A.*, **89**, 10915–10919.
- Hirschberg,D.S. (1975) A Linear Space Algorithm for Computing Maximal Common Subsequences. *Commun. ACM*, **18**, 341–343.
- Myers,E.W. (1986) An O(ND) Difference Algorithm and Its Variations. *Algorithmica*, **1**, 251–266.
- Myers,E.W. (1995) Toward simplifying and accurately formulating fragment assembly. *J. Comput. Biol.*, **2**, 275–90.
- Needleman,S.B. & Wunsch,C.D. (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*, **48**, 443–53.
- Sankoff,D. (1972) Matching Sequences under Deletion/Insertion Constraints. *Proc. Natl. Acad. Sci.*, **69**, 4–6.
- Sellers,P. (1974) On the Theory and Computation of Evolutionary Distances. *SIAM J. Appl. Math.*, **26**, 787–793.
- Smith,T.F. & Waterman,M.S. (1981) Identification of common molecular subsequences. *J. Mol. Biol.*, **147**, 195–197.
- Ukkonen,E. (1985) Algorithms for approximate string matching. *Inf. Control*, **64**, 100–118.

### 3. Pretraživanje baze podataka sljedova. BLAST

Temeljni oblik bioinformatičke analize je pretraživanje baza podataka bioloških sljedova u potrazi za sličnim sljedovima, gdje sličnost nije slučajna: nukleotidni ili aminokiselinski slijed nekog organizma uspoređuje se s bazom sljedova kako bi se pronašli slični sljedovi u drugim organizmima. Sličnost upućuje na zajedničko podrijetlo organizama (*homologija*), a homologija na sličnost biološke funkcije. Kao primjer možemo navesti sličnost između gena: ako neki slijed predstavlja *gen* nekog organizma (gen je dio DNA koji kodira za *protein*), onda se za njemu sličan slijed iz drugog organizma može prepostaviti da također predstavlja gen, odnosno, da oba gena potječu od gena - zajedničkog pretka. Zbog homologije možemo prepostaviti i sljedeće: ako je za određeni protein poznata funkcija u organizmu, onda se slična funkcija može prepostaviti i za njemu slične proteine. Ta se ideja često primjenjuje kod novosekvenciranih organizama: usporedbom s postojećim sljedovima utvrđuje se je li neki slijed gen i je li sličan genima iz baze podataka za čije je proteine poznata funkcija. Takvim analitičkim pristupom mogu se dijelom izbjegći skupi i dugotrajni eksperimenti u kojima bi se inače moralo izravno utvrditi koji je dio DNA gen i koja je funkcija proteina za koji taj gen kodira.

Baze podataka nukleotidnih sljedova mogu sadržavati cijele ili samo dijelove genoma. S razvojem i padom cijene suvremenih tehnologija sekvenciranja, broj dostupnih genoma i transkriptoma eksponencijalno raste.

Najčešće korištene javno dostupne baze podataka nukleotidnih sljedova su:

- NCBI-GenBank baza podataka (NCBI; *National Center for Biotechnology Information*)
- EMBL (*European Molecular Biology Laboratory*) baza podataka
- DDBJ (*DNA Data Bank of Japan*) baza podataka

Najpoznatija i najveća javno dostupna baza podataka koja sadrži proteinske sljedove je UniProt, koja objedinjava i koristi podatke iz drugih velikih proteinskih baza podataka:

- Swiss-Prot
- TrEMBL
- Protein Information Resource

Za ilustraciju veličine baze podataka bioloških sljedova pogledajmo primjer GenBank baze podataka iz listopada 2014.: nukleotidni sljedovi zauzimaju oko 750 GB (Growth of GenBank and WGS). Veličinu ovih podataka lakše razumijemo pogledamo li prosječne veličine genoma pojedinih organizama: npr. red veličine bakterijskih genoma je  $10^6$ - $10^7$  bp. Genomi kukaca su reda veličine  $10^8$ - $10^9$  bp, genomi sisavaca su oko  $10^9$  bp (npr. čovjekov genom je  $\approx 3 \cdot 10^9$  bp), dok genomi cvjetnica dosežu i  $10^{11}$  (Gregory, 2014). Trenutno se procjenjuje da je sekvencirano blizu 20 000 genoma različitih vrsta, od toga oko 18 000 bakterija (Azvolinsky, 2014).

### **3.1. Usporedba bioloških sljedova s bazom podataka korištenjem heurističkog pristupa**

Prepostavimo da želimo usporediti upitni sliked (engl. *query*) s bazom podataka sljedova (engl. *target*). Neka upitni sliked sadrži  $m$  nukleotida, a baza podataka ukupno  $n$  nukleotida. Određivanje optimalnog poravnjanja dinamičkim programiranjem (poglavlje 2) obavlja se u  $O(mn)$  vremenu, što je vrlo sporo za velike baze podataka. Kao alternativa dinamičkom programiranju, za velike se skupove podataka koriste programi koji imaju heuristički pristup rješavanju problema. Njihova je osobina značajno ubrzanje u odnosu na programe koji koriste dinamičko programiranje, ali pronalazak optimalnog rješenja ovakvim pristupom nije zajamčen<sup>6</sup>. Heuristički pristup koriste i najpoznatiji programi za usporedbu bioloških sljedova s bazom podataka: BLAST (Altschul *et al.*, 1990) i FASTA (Lipman and Pearson, 1985). Oba programa koriste princip lokalnog poravnanja - dijelovi upitnog slikeda pokušavaju se na lokalno najbolji način poravnati sa sljedovima iz baze podataka. Novije inačice BLAST-a, Gapped BLAST, PSI-BLAST (Altschul *et al.*, 1997) glavni su alati NCBI-a za usporedbe proteinskih i nukleotidnih sljedova. Osim njih, danas se često koriste i MegaBLAST (Zhang *et al.*, 2000) i PHI-BLAST (Zhang *et al.*, 1998). O popularnosti BLAST-a svjedoči i njegova citiranost: radovi Altschul *et al.* (1990, 1997) zajedno su imali preko 70 000 citata do kraja 2013. godine, a gotovo 77 000 citata u listopadu 2014. godine (Web of Science). S obzirom da je BLAST danas najrašireniji alat za usporedbu upitnog slikeda s bazom podataka, u dalnjem će tekstu biti opisan princip

---

<sup>6</sup> Ovakav pristup predstavlja odvagivanje (eng. *trade-off*) između brzine, kojom se dolazi do rješenja i osjetljivosti (eng. *sensitivity*) dobivenog rješenja.

njegova rada (na Sl. 3.1 je prikazano sučelje NCBI-BLAST programa za rad korištenjem preglednika).

The screenshot shows the NCBI BLAST search interface. At the top, there's a navigation bar with links for Home, Recent Results, Saved Strategies, Help, My NCBI, Sign In, and Register. Below the navigation bar, the title "Standard Nucleotide BLAST" is displayed, along with tabs for blastn, blastp, blastx, tblastn, and tblastx. A sub-header indicates "BLASTN programs search nucleotide databases using a nucleotide query." The main search area has a text input field for "Enter Query Sequence" containing a sample nucleotide sequence: CTTATAGAGAGTGTATTTTTTTTTATGTAATTACCTCCAAATCAATTGATAACCAA GTATTATCATAAACGAGTAAAAAGATATTGATAACGCIT. There are buttons for "Pretraži..." and "Datoteka nije odabrana." (File not selected). To the right, there are "Query subrange" options with "From" and "To" fields. Below the sequence input, there's a section for "Job Title" and a checkbox for "Align two or more sequences".

**Choose Search Set**

**Database**: Human genomic + transcript, Mouse genomic + transcript, Others (nr etc.); Nucleotide collection (nr/nt)

**Organism**: Enter organism name or id—completions will be suggested, Exclude (+)

**Exclude**: Models (XM/XP), Uncultured/environmental sample sequences

**Limit to**: Sequences from type material

**Entrez Query**: Enter an Entrez query to limit search (YouTube Create custom database)

**Program Selection**

**Optimize for**: Highly similar sequences (megablast) (selected), More dissimilar sequences (discontiguous megablast), Somewhat similar sequences (blastn), Choose a BLAST algorithm

**Algorithm parameters**

**General Parameters**

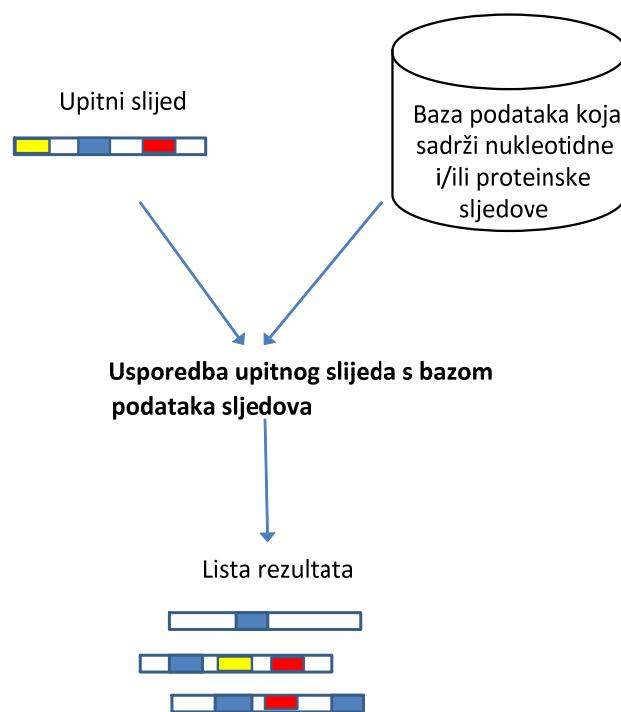
Below the search form, a summary box says: "Search database Nucleotide collection (nr/nt) using Megablast (Optimize for highly similar sequences)" with a link to "Show results in a new window".

Sl. 3.1 NCBI-BLAST sučelje. Korisnik može zadati upitni slijed (engl. *Query Sequence*) upisivanjem slijeda u FASTA formatu ili upisivanjem *gi* broja (jedinstvenog identifikatora slijeda).

Zatim se zadaje podskup baze podataka koji će se pretraživati (engl. *Search Set*), izabiru se parametri po kojima će se određivati rezultat poravnjanja (engl. *Algorithm Parameters*) te se odabire jedan od potprograma BLAST-a, koji će koristiti za pretraživanje (npr. *Megablast*).

## 3.2. Program BLAST

Da bi se izbjeglo poravnjanje jako dugih sljedova, BLAST koristi sljedeću heuristiku: iz upitnog se slijeda određuju podnizovi duljine  $l$ , gdje je  $l$  tipično  $\sim 10$  za nukleotidne sljedove ili 2-3 za aminokiselinske sljedove. Ove podnizove se naziva začetcima (engl. *seeds*). Baza sljedova se pretražuje korištenjem tih kratkih podnizova. Nakon što su podnizovi pronađeni u bazi, potrebno je odlučiti koji sljedovi iz baze podataka imaju dovoljno podnizova lokalno poravnatih s podnizovima-začetcima poredanih u pravilnom redoslijedu te s prihvatljivim međusobnim udaljenostima da bi mogle činiti „kvalitetno“ lokalno poravnjanje s upitnih sljedom. BLAST zatim koristi egzaktno poravnjanje dinamičkim programiranjem između upitnog slijeda i odabranih sljedova iz baze podataka. Pojednostavljeni, kao rezultat odabiru su oni sljedovi za koje je statistički procijenjeno da je mala vjerojatnost da je „kvaliteta“ njihovog poravnanja s upitnim slijedom posljedica slučajnosti, a ne homologije.



Sl. 3.2 Usporedba upitnog slijeda s bazom podataka nukleotidnih ili aminokiselinskih sljedova

Ideja osnovnog BLAST algoritma te njegove verzije koja uključuje procijepe (*Gapped BLAST*) prikazana je na Sl. 3.3 te je zatim detaljnije pojašnjena u tekstu koji slijedi.

### **Algoritam BLAST (*List, Scan, Extend*)**

1. Iz upitnog slijeda odrediti listu riječi  $L$ . *Riječi* su kratki podnizovi ulaznog niza unaprijed zadane duljine (za aminokiseline: obično 3 znaka; za nukleotide: obično 11 znakova).
2. Za svaku riječ  $w$  iz  $L$  pronaći slične riječi, tj. izgraditi listu riječi  $L'$  čije poravnanje s  $w$  daje rezultat  $\geq T$ , gdje je  $T$  programski definiran prag. Rezultat se poravnaja kod nukleotidnih sljedova određuje bodovanjem podudaranja/zamjene nukleotida, a kod aminokiselinskih sljedova korištenjem supstitucijskih matrica.
3. Pretražiti bazu podataka kako bi se pronašle sljedovi koji sadrže riječi iz  $L'$ ; pronađeni podnizovi u bazi podataka nazivaju se pogodcima (engl. *hits*).

#### Osnovni BLAST

4. Proširiti poravnanje oko pogodaka u lijevo i u desno, sve dok rezultat poravnaja ne počne padati ispod zadanog praga. Tako dobivena područja nazivaju se područja s visokim rezultatom poravnaja (**HSP**; engl. *High-scoring Segment Pair*).
5. Odabratи HSP-ove s najvišim rezultatom i odrediti njihov statistički značaj ( $E$  i  $p$  vrijednosti). Prikazati ih kao konačne rezultate u listi.

#### BLAST koji dozvoljava procijepu (Gapped BLAST)

4. Povezati 2 ili više inicijalnih pogodaka koji se nalaze na istoj dijagonali i međusobno su udaljeni  $< A$ . Između njih izabrati statistički značajna podudaranja.
5. Statistički značajna podudaranja se ponovno poravnavaju korištenjem SW algoritma. Takva područja predstavljaju konačni rezultat koji se onda pokazuje u listi rezultata zajedno sa statističkim pokazateljima ( $E$  i  $p$  vrijednost).

Sl. 3.3 Algoritam BLAST – osnovni algoritam i algoritam koji uključuje procijepu

Podnizovi, koji se u programu BLAST koriste kao začetak duljih poravnaja, nazivaju se *rijeci* (engl. *words*) i njihovu duljinu korisnik zadaje unaprijed, ovisno o tome sadrži li upitni slijed nukleotide ili aminokiseline. U slučaju nukleotidnog upitnog slijeda, prepostavljena duljina riječi je 11 (obično se koriste vrijednosti od 7 do 14). U slučaju aminokiselina, prepostavljena duljina riječi je 3 (ponekad se koristi i 2 kada se pretražuju kraći sljedovi aminokiselina, npr. peptidi). Primjer izgradnje liste riječi s kojima se zatim pretražuje baza podataka prikazan je na Sl. 3.4.

**Primjer: Izgradnja liste riječi duljine  $n$  iz upitnog slijeda**

Neka je zadan ulazni aminokiselinski slijed **GEIIGCT**.

Neka je za aminokiseline zadana duljina riječi  $n = 3$ . S obzirom da ukupno postoji 20 aminokiselina, broj mogućih riječi od 3 znaka (aminokiseline) je  $20^3 = 8000$ . Neka je zadan prag  $T = 12$ .  $T$  predstavlja vrijednost poravnjanja između 2 riječi korištenjem zadane supstitucijske matrice (ovdje: BLOSUM62).

- Iz upitnog se slijeda **GEIIGCT** stvara lista riječi  $L$ . Riječi iz  $L$  su podnizovi ulaznog niza **GEIIGCT** duljine 3 znaka, dakle:

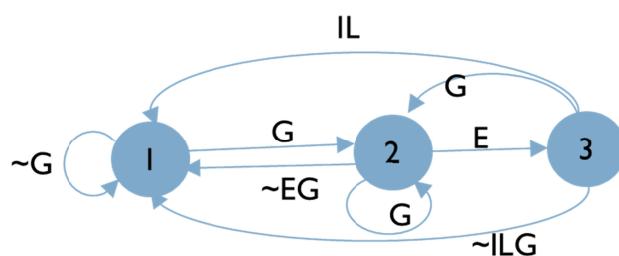
$$L = \{GEI, EII, IIG, IGC, GCT\}$$

- Svaka riječ  $r$  iz  $L$  promatra se u odnosu na sve moguće riječi od 3 znaka (aminokiseline);  $r'$ . Za svaku riječ  $r$  određuje se rezultat poravnjanja s  $r'$ . Ako je rezultat poravnanja korištenjem BLOSUM62 matrice  $\geq T$ , onda se nova riječ  $r'$  dodaje u  $L'$ , gdje je  $L'$  lista riječi s kojima će se pretraživati baza podataka.

Npr. za  $r = GEI$ , zbog jednostavnosti, promatramo samo riječi GEI, GEE i GEL. Rezultat poravnanja  $r$  s GEI je  $6 + 5 + 4 = 15 \geq T$ , rezultat poravnanja  $r$  i GEE je  $6 + 5 - 3 = 8 < T$ , a rezultat poravnanja  $r$  i GEL je  $6 + 5 + 2 = 13 \geq T$ . Dakle, u listi  $L'$  nalazit će se GEI i GEL.

S1. 3.4. Primjer izgradnje liste riječi iz upitnog slijeda

Za riječi iz liste  $L' = \{GEI, GEL\}$  gradi se automat s konačnim brojem stanja koji za slijed iz baze podataka može dojaviti je li prepoznata neka od riječi iz liste  $L'$  (S1. 3.5). Početno stanje, a ujedno i završno stanje, označeno je s 1. Prijelaz u stanje 2 moguć je samo ako se učita znak G. Prijelaz u stanje 3 je moguć, ako se učita znak E. Prijelaz iz stanja 3 u završno stanje (1) događa se ako se učita I ili L, nakon čega se za ulazni niz dojavljuje da odgovara nizovima iz  $L'$ .



S1. 3.5 Primjer automata s konačnim brojem stanja koji prepoznaće riječi iz liste  $L' = \{GEI, GEL\}$ . Početno stanje je 1, što je ujedno i završno stanje. Oznaka  $\sim$  označava negaciju, npr.  $\sim G$  označava bilo koji znak osim G;  $\sim ILG$  označava bilo koji znak osim znakova I, L i G.

Nad sljedovima iz BP, za koje je automat dojavio da sadrže riječi iz  $L'$ , obavlja se daljnja usporedba s ulaznim nizom, odnosno prelazi se na 4. korak algoritma (Sl. 3.3). Na Sl. 3.6 opisan je primjer određivanja područja s visokim rezultatom poravnjanja (engl. *High-scoring Segment Pair*; HSP), nastalog proširenjem inicijalnog pogotka (engl. *hit*).

#### **Primjer: Određivanje područja s visokim rezultatom poravnjanja**

Neka je zadan upitni slijed  $Upit = EETPQIAVE$ . Slijed  $Upit$  rastavlja se na podnizove (rijec) od 3 znaka koje će sačinjavati skup  $L$ . Npr. riječ  $PQI \in L$ .

Nadalje, neka je zadan prag  $T = 9$ , tako da sve riječi od 3 znaka, koje poravnate s rijećima iz  $L$  daju rezultat poravnjanja  $\geq T$ , sačinjavaju skup  $L'$ . Za određivanje poravnjanja koristimo supstitucijsku matricu BLOSUM62. Npr. riječ  $PQE \in L'$ , jer je rezultat poravnjanja PQI i PQE jednak 9, tj. kraće pišemo  $PQI \parallel PQE = 9$ .

Neka je u bazi podataka pronađen slijed  $LITPQELVC$ , koji želimo poravnati sa slijedom  $Upit$  proširenjem pronađenog pogotka (PQE).

$Upit = E E T P Q I A V E$

*Slijed iz BP = L I T P Q E L V C*

Pogodak se proširuje uljevo i udesno sve dok trenutno najbolji rezultat poravnjanja ne padne za  $\geq X = 3$ . Za određivanje poravnjanja koristimo supstitucijsku matricu BLOSUM62.

$PQI \parallel PQE = 7 + 5 - 3 = 9$  (pronađen je pogodak za PQI)

$TPQIA \parallel TPQEL = 5 + 7 + 5 - 3 - 1 = 13$  (poravnavaju se dodatno T i T te A i L)

$ETPQIAV \parallel ITPQELV = -3 + 5 + 7 + 5 - 3 - 1 + 4 = 14$  (maksimalan rezultat poravnjanja)

$EETPQIAVE \parallel LITPQELVC = -3 - 3 + 5 + 7 + 5 - 3 - 1 - 4 = 7$  (rezultat poravnjanja je pao za  $7 \geq X = 3$ )

Sl. 3.6 Primjer proširenja pogotka (engl. *hit*) i generiranje područja s visokim rezultatom poravnjanja (engl. *High-scoring Segment Pair*; HSP)

U u slučaju osnovnog BLAST algoritma (Altschul *et al.*, 1990), riječi iz upitnog slijeda za koje su pronađeni pogodci (engl. *hits*) u BP, pokušavaju se proširiti uljevo i udesno sve dok rezultat poravnjanja dobiven proširenjem ne počne padati ispod unaprijed zadano praga u odnosu do tada maksimalnu vrijednost poravnjanja, kao što je prikazano na Sl. 3.6. Tako se dobivaju područje s visokim rezultatom poravnjanja (*HSP*).

Između svih pronađenih parova segmenata, ispisuju se ona koja daju statistički najznačajniji rezultat ( $E$  i  $p$  vrijednost, koje su pojašnjene kasnije u tekstu). Određivanje proširenja pogodaka zahtijeva oko 90% ukupnog vremena rada programa.

U slučaju verzije BLAST-a koja omogućuje rad s procijepima, *Gapped* BLAST (Altschul *et al.*, 1997), pronađeni parovi riječi se pokušavaju međusobno spojiti ako se nalaze na istoj dijagonali i međusobno su udaljeni za manje od  $A$  znakova. Riječi se spajaju tako da se područje između njih optimalno poravna dinamičkim programiranjem. Među tako poravnatim područjima pronalaze se statistički najznačajnija, koja se onda ponovno poravnavaju korištenjem Smith-Watermanovog algoritma (Smith & Waterman, 1981).

Kod poravnавања нуклеотида користе се unaprijed zadane vrijedности за бодовање подударанja (engl. *match*) и неподударанja (тј. замјене) нуклеотида (engl. *mismatch*) (нпр. +1 за подударanje и -2 за замјену). За математички приказ бодовања појаве процјепа (празнине) у поравњању (engl. *gap cost*) користи се линеарна или афина функција (поглавље 2.7.2). Код поравњања аминокиселина, користи се бодовање према supstitucijskim матрицама<sup>7</sup>. У програму BLAST могу се користити PAM матрице (Dayhoff *et al.*, 1978) или BLOSUM матрице (Henikoff & Henikoff, 1992).

PAM (или APM,engl. *Accepted Point Mutation*) матрице темеље се на филогенетском стаблу изграђеном из поравњања проматраних слједова аминокиселина близко сродних врста. PAM 1 матрица садржи цјелобројне vrijednosti које се односе на мутацијске стопе између аминокиселина у времену у којем је промјенијено 1% аминокиселина. Све остale PAM матрице с већим бројем (нпр. PAM30, PAM250) рачунaju се из PAM1 матрице и представљају промјене у дужим временским раздobljima, што одговара слједовима, који су међусобно еволуцијски удалjeniji.

BLOSUM (engl. *BLOck SUbstitution Matrix*) матрице изграђене су темељем поравњања врло очуваних подручја између удалjenijih слједова и дјају боље резултате за успоређивање удалjenijih протеина. У програму BLAST предпостављено је кориштење BLOSUM62 матрице, а још су понуђене матрице PAM30, PAM70, PAM250, BLOSUM45,

---

<sup>7</sup> Supstitucijska матрица је матрица димензија  $20 \times 20$ , где је 20 број аминокиселина. Полje  $(i, j)$  у supstitucijskoj матрици садржи цјели број који представља стопу, односно, вјеројатност мутације аминокисeline  $i$  у аминокиселину  $j$  у неком времену. Вјеројатности мутација одговарају проматраним хемијским и физичким својствима аминокиселина (нпр. већа је вјеројатност да ће хидрофилна аминокиселина мутирати у хидрофилну, а мања да ће мутирати у хидрофобну киселину).

BLOSUM50, BLOSUM80, BLOSUM90. Kod poravnanja blisko srodnih proteina obično se koriste PAM1 i BLOSUM80 matrice, kod srednje udaljenih proteina PAM120 i BLOSUM62, a za udaljene proteine: PAM250 i BLOSUM45.

Pri korištenju BLAST-a, rezultati uvijek nose informaciju o tzv. *E-vrijednosti* (engl. *E-value*). Ta vrijednost ukazuje na statistički značaj resultantnog poravnjanja. Što je ona manja, to je manja i vjerojatnost da je poravnanje nastalo slučajno.

Vjerojatnost da je poravnanje nastalo slučajno ovisit će o nekoliko stvari. Jednostavno je za primijetiti da je u slučaju pretraživanja veće baze, veća i vjerojatnost da će doći do slučajnoga „kvalitetnog“ poravnanja s nekim od sljedova iz baze. Isto vrijedi i za kratak upitni slijed kraći: za ulazni slijed duljine 30, veća je vjerojatnost da će se pronaći poravnanje u bazi sljedova, nego za upitni slijed duljine 300. Pokazano je da se rezultati Smith–Watermanovog lokalnog poravnanja ponašaju prema distribuciji ekstremne vrijednosti (engl. *Extreme Value Distribution*; EVD) (Karlin and Altschul, 1990), što omogućuje procjenu vjerojatnosti u kojoj je mjeri rezultat poravnanja posljedica slučajnosti. Zbog toga se i u programu BLAST statistički značaj svakog poravnanja procjenjuje primjenom EVD-a. Temeljni izraz je određivanje vjerojatnosti ( $p$ ) da je rezultat poravnanja dva slučajna slijeda ( $S$ ) veći ili jednak  $x$ :

$$P(S \geq x) = 1 - \exp(-Kmn \cdot e^{-\lambda x}) \quad (3.1)$$

gdje je  $m$  duljina upitnog niza,  $n$  duljina svih sljedova u bazi, a  $\lambda$  i  $K$  su Karlin-Altschulovi statistički parametri (Altschul *et al.*, 1990, 1997) određeni odabranim parametrima i svojstvima podataka u bazi podataka, koje program automatski određuje.

Bitovni rezultat poravnanja  $S'$  određuje se iz  $S$  kao:

$$S' = (\lambda S - \ln K) / \ln 2 \quad (3.2)$$

Očekivanje da ćemo slučajnim putem pronaći slijed u bazi podataka kojemu je bitovni rezultat poravnanja s ulaznim slijedom jednak  $S'$  je:

$$E = mn \cdot 2^{-S'} \quad (3.3)$$

Odnosno, vjerojatnost  $p$  da je dobiveni rezultat poravnjanja  $S$  slučajan jest:

$$p = 1 - e^{-E} \quad (3.4)$$

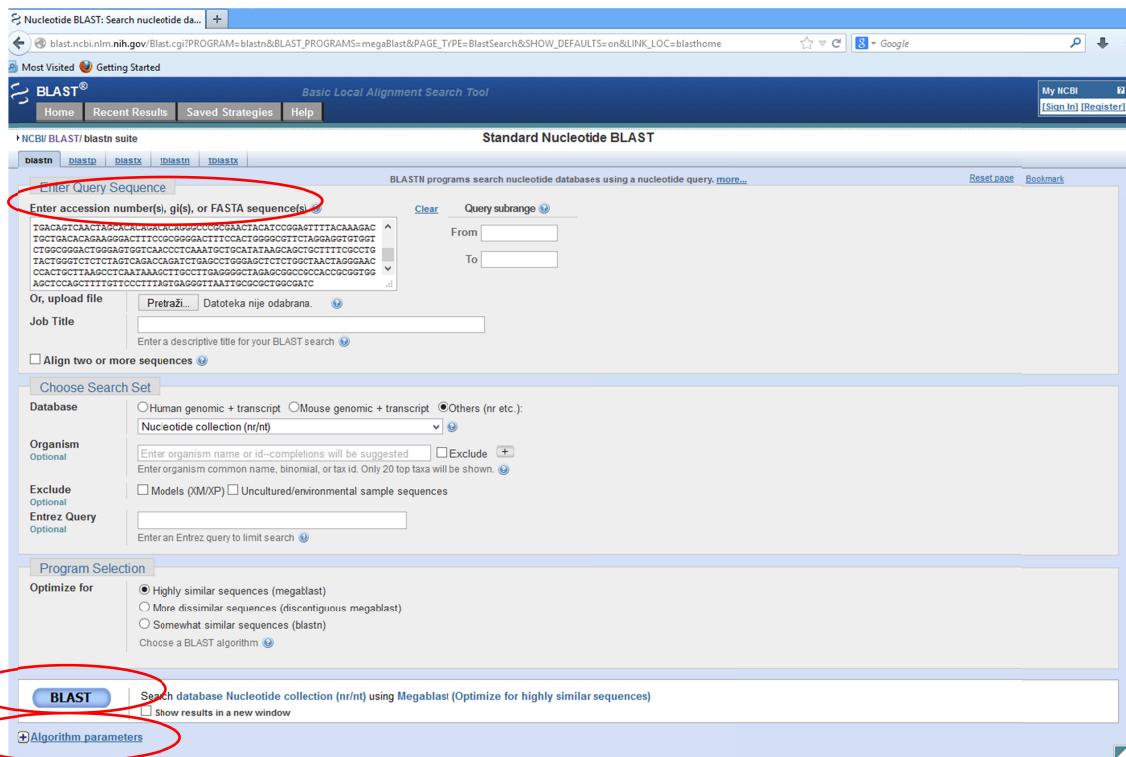
Za  $E \leq 0.05$ ,  $E$  i  $p$  su približno jednaki. Npr. za  $E = 0.05$ ,  $p = 0.04877058$ . Kao statistički značajan rezultati može se uzeti  $E \leq 0.05$ , ali u praktičnoj primjeni relevantnost  $E$  ovisi o tome koliko su srodni sljedovi koji se uspoređuju, njihovoj duljini i ukupnoj veličine baze podataka. Obično, za vrlo srodne sljedove može biti  $E \leq 10^{-20}$ , pa i  $E \approx 0$ . Kod usporedbi genoma mikroorganizama pokazalo se da su značajne vrijednosti oko  $10^{-4}$  i manje (Pevsner, 2009a).

### 3.3. BLAST - postupak pretraživanja baze podataka

Pretraživanje baze podataka korištenjem programa BLAST može se obavljati kroz *web-sučelje*<sup>8</sup> (Sl. 3.1) ili korištenjem zasebne aplikacije.

---

<sup>8</sup> [blast.ncbi.nlm.nih.gov](http://blast.ncbi.nlm.nih.gov)



Sl. 3.7 BLAST web-sučelje. Oznakama A, B i C označene su osnovne opcije: A) unos upitnog slijeda; B) odabir podskupa baze podataka, koji će se pretraživati; C) parametri BLAST algoritma, koji će se koristiti pri pretraživanju.

Radi jednostavnosti, korake u postupku pretraživanja pokazat ćemo korištenjem mrežnog sučelja programa BLAST na stranicama NCBI-a:

Odabrati jedan od BLAST programa, ovisno o tome sastoji li se upitni slijed od nukleotida ili aminokiselina i s kojim tipom sljedova (proteinskim ili nukleotidnim sljedovima) se želi uspoređivati upitni slijed. Ponuđeni su sljedeći programi: blastn i njegova nadgradnj (megablast), blastp i njegove nadgradnje (psi-blast), blastx, tblastx, tblastn (Tablica 3.1). Na Sl. 3.7 odabran je program blastn (kartica *blastn* pri vrhu sučelja).

Tablica 3.1 Temeljni BLAST programi

Program	Upitni slijed	Baza podataka sljedova
blastp	Protein	Protein
blastn	DNA	DNA
blastx <sup>9</sup>	DNA (6 mogućih čitanja)	Protein
tblastn <sup>10</sup>	Protein	DNA (6 mogućih čitanja)
tblastx <sup>11</sup>	DNA (6 mogućih čitanja)	DNA (6 mogućih čitanja)

2. Odabratи upitni slijed (engl. *query*) zadavanjem jednog od identifikacijskih brojeva (*gi* identifikator ili *accession number* identifikator) ili unosom slijeda u formatu FASTA. Na Sl. 3.7 unijet je upitni slijed u formatu FASTA (A).
3. Odabratи bazu podataka za pretraživanje, npr. *nonredundant database* (*nr*). Na Sl. 3.7 (B) odabrana je *nonredundant database*, što je i inicijalno ponuđeno.
4. Odabratи postavke pretraživanja baze podataka (engl. *algorithm parameters*) (Sl. 3.7; C) i pritisnuti gumb BLAST.
5. Pregled prikazanih rezultata: rezultat je lista nukleotidnih sljedova koji su najbolje rangirani prema parametrima vezanim uz bodovanje poravnjanja između ulaznog i rezultantnog slijeda (*Max score*, *Total score*, *Query cover*, *Ident*) te statističkom značaju rezultantnog poravnjanja (*E-value*) (primjer ispisa sljedova iz baze podataka koji imaju najveći rezultat poravnjanja sa zadanim upitnim slijedom je prikazan na Sl. 3.8)



Description		Max score	Total score	Query cover	E value	Ident	Accession
<input type="checkbox"/>	HIV-1 strain 97CN001 from China, complete genome	16188	16188	98%	0.0	99%	AF286226.1
<input type="checkbox"/>	HIV-1 strain 98CN009 from China, complete genome	15653	15653	98%	0.0	98%	AF286230.1
<input type="checkbox"/>	HIV-1 strain CNGL179 from China, complete genome	15531	15531	97%	0.0	98%	AF503396.1
<input type="checkbox"/>	HIV-1 isolate Sichuan_2006_SC025 from China gag protein (gag) gene, complete cds; pol protein (pol) gene, partial cds; vif protein (vif), vpr prot	14846	14846	98%	0.0	97%	JX392381.1

Sl. 3.8 BLAST: ispis dijela rezultata.

<sup>9</sup> **blastx** – nukleotidni upitni slijed je dinamički translatiran prema 6 okvira čitanja u 6 proteinskih sljedova i onda se svaki od tih sljedova uspoređuje s bazom podataka proteina

<sup>10</sup> **tblastn** - DNA baza podataka je translatirana po 6 okvira čitanja i onda se svaki od dobivenih proteinskih sljedova uspoređuje s ulaznim proteinom

<sup>11</sup> **tblastx** – 36 mogućih proteinsko-proteinskih kombinacija

### 3.4. Literatura

- Altschul,S.F. *et al.* (1990) Basic local alignment search tool. *J. Mol. Biol.*, **215**, 403–410.
- Altschul,S.F. *et al.* (1997) Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res.*, **25**, 3389–3402.
- Azvolinsky,A. (2014) Sequencing the Tree of Life.
- Dayhoff,M.O. *et al.* (1978) A Model of Evolutionary Change in Proteins. *Atlas Protein Seq. Struct.*, **5**, 345–352.
- Gregory,T.R. (2014) Animal Genome Size Database. <http://www.genomesize.com>.
- Growth of GenBank and WGS <http://www.ncbi.nlm.nih.gov/genbank/statistics>.
- Henikoff,S. and Henikoff,J.G. (1992) Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci.*, **89**, 10915–10919.
- Karlin,S. and Altschul,S.F. (1990) Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proc. Natl. Acad. Sci. U. S. A.*, **87**, 2264–2268.
- Lipman,D. and Pearson,W. (1985) Rapid and sensitive protein similarity searches. *Science*, **227**, 1435–1441.
- Pevsner,J. (2009) Bioinformatics and functional genomics 2nd ed. Wiley-Blackwell, Hoboken, N.J.
- Smith,T.F. and Waterman,M.S. (1981) Identification of common molecular subsequences. *J. Mol. Biol.*, **147**, 195–197.
- Web of Science.
- Zhang,Z. *et al.* (2000) A greedy algorithm for aligning DNA sequences. *J. Comput. Biol. J. Comput. Mol. Cell Biol.*, **7**, 203–214.
- Zhang,Z. *et al.* (1998) Protein sequence similarity searches using patterns as seeds. *Nucleic Acids Res.*, **26**, 3986–3990.

## 4. Sufiksno stablo i sufiksno polje

U 2. poglavlju naveli smo problem optimalnog poravnjanja dugačkih sljedova, koje je vremenski i memorijski zahtjevno za duge sljedove. Također smo vidjeli kako se može ubrzati poravnavanje sljedova korištenjem heuristike (program BLAST; 3. poglavlje). U ovom će poglavlju biti opisane strukture podataka koje omogućuju brz pronađak kratkih podnizova u zadanom slijedu ili sljedovima. Nakon što su pronađeni podnizovi, moguće je analizirati njihove međusobne udaljenosti i na temelju njih odlučiti o isplativosti određivanja optimalnog poravnjanja dinamičkim programiranjem na cijelom području. Pronađeni podnizovi se onda koriste kao začetci (engl. *seeds*) lokalnih ili globalnih poravnjanja.

Problem pronađaka identičnih ili sličnih podnizova u promatranim sljedovima odgovara standardnom problemu *pronađaka uzorka u tekstu* (engl. *pattern matching*). Ako je s  $P$  označen *uzorak* (tj. kraći podniz, engl. *pattern*), a s  $T$  *tekst* (tj. dulji niz; engl. *text*), gdje je obično  $|T| \gg |P|$ , tada je naš problem pronaći svih  $z$  pojavljivanja  $P$  u  $T$ . Naivno rješenje ovog problema može se pronaći u  $O(|P| \cdot |T|)$  vremenu, što je, naravno, neprihvatljivo za dulje nizove (npr. ako je  $T$  genom bakterije, onda je riječ o duljini  $|T|$  od  $10^6\text{-}10^7$  bp, a za složenije organizme duljina  $|T|$  je i nekoliko redova veličine veća).

Algoritamska rješenja problema pronađaka uzorka u tekstu možemo podijeliti u dvije skupine:

1. Rješenja u kojima se prvo indeksira podniz  $P$ , a onda se indeks izgrađen nad  $P$  koristi za pretraživanje teksta  $T$ .
2. Rješenja u kojima se prvo indeksira teksta  $T$ , gdje se onda indeks izgrađen nad  $T$  koristi za pretraživanje  $P$ .

U prvom pristupu, izgradnja indeksa nad podnizom  $P$  je brza (vremenske složenosti  $O(|P|)$ ), a sam indeks zauzima vrlo malo mesta. Međutim, pretraživanje teksta  $T$  korištenjem takvog indeksa obavlja se u najgorem slučaju u vremenu  $O(|T|)$ . To je neprihvatljivo za dugačke tekstove  $T$ , posebno kada se pretraživanjem  $T$  želi pronaći više različitih podnizova. Klasični algoritmi koji koriste ovaj pristup su Knuth-Morris-Prattov algoritam (Knuth *et al.*, 1977) i Boyer-Mooreov algoritam (Boyer & Moore, 1977).

U bioinformatici je promatrani tekst  $T$  vrlo dugačak i gotovo uvijek postoji potreba da se  $T$  pretražuje s puno različitih podnizova. Zbog toga se najčešće koristi drugi pristup u kojem se tekst  $T$  indeksira. Traženje podniza  $P$  u  $T$  se obavlja korištenjem izgrađenog indeksa, što se može postići u vremenu  $O(|P|)$ . U ovom će poglavlju biti prikazani algoritmi i podatkovne strukture koje se koriste za indeksiranje dugačkih  $T$ : sufiksno stablo i sufiksno polje.

## 4.1. Sufiksno stablo

Sufiksno stablo (engl. *suffix tree*) je indeksna struktura podataka namijenjena učinkovitom obavljanju različitih operacija nad znakovnim nizovima i često se koristi u bioinformatici (Gusfield, 1997).

Sufiksno stablo na elegantan način rješava problem pronalaska podniza u nizu. Traženje podniza u nizu odgovara traženju pojma u knjizi. Ako knjiga sadrži indeks pojmove, jednostavno je pronaći traženi pojam u knjizi: umjesto da se pretražuje cijela knjiga, pojam se traži pretraživanjem indeksa. Pretraživanje je dodatno ubrzano s obzirom da su pojmovi abecedno poredani. Ako je pojam zabilježen u indeksu, uz pojam su navedene i stranice knjige na kojima se pojam spominje, pa se pronalaskom pojma u indeksu izravno dolazi i do mjesta gdje se pojam spominje u knjizi. Analogno indeksu u knjizi, indeksna struktura sufiksno stablo omogućuje brz pronalazak podniza u promatranom tekstu - podniz je moguće pronaći u vremenu proporcionalnom duljini podniza. Korištenjem sufiksnog stabla moguće je riješiti i mnoštvo drugih složenih operacija nad znakovnim nizovima, poput pronalaska najduljeg zajedničkog podniza dvaju nizova u vremenu proporcionalnom zbroju njihovih duljina (Gusfield, 1997).

Sufiksno stablo može biti izgrađeno u vremenu linearno ovisnom o duljini ulaznog niza. Prvi algoritam koji je to omogućio bio je Weinerov algoritam (1973), a zatim i McCreightov algoritam (1976). Današnji je standard Ukkonenov algoritam iz 1995. godine (Ukkonen, 1995) i njegove će osnovne postavke biti izložene u potpoglavlju 4.1.6. To je bio prvi *on-line* algoritam za izgradnju sufiksnog stabla: umjesto da se sufiksno stablo gradi odjednom iz cijelog ulaznog niza (pri čemu cijeli ulazni niz mora biti poznat na samom početku izgradnje), sufiksno stablo se proširuje za svaki znak u nizu.

### 4.1.1. Osnovni pojmovi

Neka je  $S$  niz znakova duljine  $n$ , tj.  $|S| = n$ , gdje su znakovi niza  $S$  elementi abecede  $\Sigma$ . Neka  $\sigma$  označava broj znakova abecede, tj.  $|\Sigma| = \sigma$ . Znak na  $i$ -tom mjestu u nizu  $S$  će biti označen s  $S[i]$  ili kraće  $s_i$ , gdje  $1 \leq i \leq n$ . Neka je  $\$$  poseban znak koji se dodaje na kraj niza  $S$  i ne postoji u abecedi  $\Sigma$ , a leksički (abecedno) je manji od svakog znaka iz  $\Sigma$ .

Uvodimo i sljedeće oznake:

- *podniz* niza  $S$  je  $S[i, j] = S_i S_{i+1} \dots S_j$ ,  $1 \leq i \leq j \leq n$
- *prefiks* niza  $S$  je podniz  $S[1, j]$ ,  $1 \leq j \leq n$
- *sufiks* niza  $S$  je podniz  $S[i, n]$  (kraće,  $s_i$ ),  $1 \leq i \leq n$

### 4.1.2. Struktura podataka sufiksno stablo

Neka je zadan niz  $S$  duljine  $n$ , gdje su znakovi niza iz abecede  $\Sigma$ . Sufiksno stablo (engl. *suffix tree*) izgrađeno nad  $S$  je ukorijenjeno stablo  $T$  (engl. *rooted tree*) koji sadrži sve sufiksa niza  $S\$$ .

Listovi (engl. *leaf; terminal node*) stabla  $T$  označeni su brojevima 1 do  $n$ , a odgovaraju početnim mjestima sufiksa u nizu  $S$ . Iz korijena sufiksnog stabla izlazi točno onoliko grana (ili bridova; engl. *branch; edge*) koliko je znakova abecede te još jedna grana za znak  $\$$ . Svaki unutarnji čvor (engl. *inner node; branch node*) ima dvoje ili više čvorova-djece.

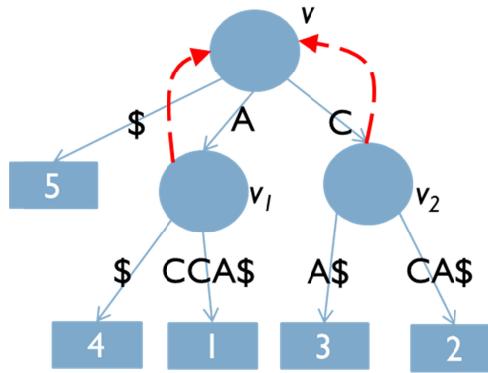
Oznaka grane (engl. *label*) može sadržavati jedan ili više znakova koji odgovaraju nekom podnizu niza  $S$ . Oznake grana koje izlaze iz istog čvora moraju započinjati s različitim znakovima.

Znakovna dubina (engl. *string depth*) nekog čvora je zbroj duljina svih oznaka grana na putu od korijena do tog čvora. Dubina čvora  $x$  (engl. *node-depth*) je broj čvorova na putu od korijena do  $x$ .

#### Primjer:

Zadan je ulazni niz  $S = \text{ACCA}$ . Na Sl. 4.1 prikazano je sufiksno stablo za niz  $S\$ = \text{ACCA}\$$ . Znak  $\$$  je dodan na kraj niza  $S$ , čime je svaki sufiks  $S[i, n]$ ,  $1 \leq i \leq 5$  i  $n = 5$ , jednoznačno određen pripadajućim listom  $i$ . Kada  $\$$  ne bi bio dodan na kraj niza  $S$ , onda bi sufiks, koji je prefiks nekog drugog sufiksa, završavao u unutarnjem čvoru stabla, a ne u listu stabla.

## $S\$ = ACCA\$$



Sl. 4.1 Sufiksno stablo za niz  $S\$ = ACCA\$$ . Na kraj niza  $S$  dodaje se znak  $\$$  (znak koji ne postoji nigdje u  $S$  i abecedno je manji od svih znakova iz  $S$ ), čime je osigurano da svaki sufiks niza  $S$  završava u listu stabla. Sufiksne veze su prikazane crvenim crtanim strelicama.

Sufiksi sadržani u prikazanom sufiksnom stablu su:

$$s_1 = S[1, n+1] = S[1, 5] = ACCA\$$$

$$s_2 = S[2, n+1] = S[2, 5] = CCA\$$$

$$s_3 = S[3, n+1] = S[3, 5] = CA\$$$

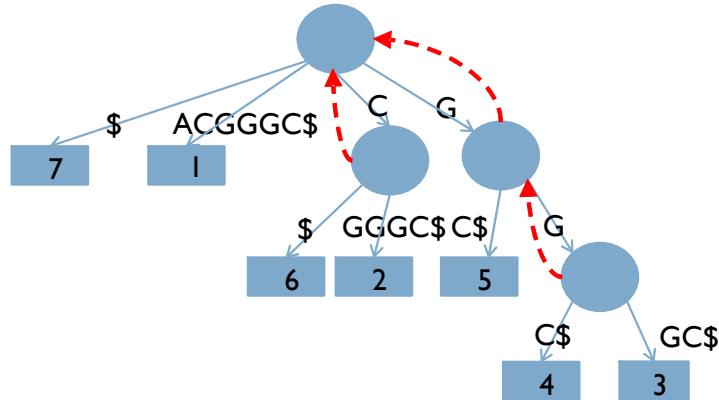
$$s_4 = S[4, n+1] = S[4, 5] = A\$$$

$$s_5 = S[n+1, n+1] = S[5, 5] = \$$$

Na primjer, možemo za sufiks  $s_2 = CCA\$$  provjeriti postoji li odgovarajući list u stablu. Provjeru obavljamo tako da slijedimo put od korijena (čvor  $v$ ) prema listovima stabla (Sl. 4.1). Krećemo od korijena i uspoređujemo prvi znak sufiksa  $s_2$  (C) sa svim prvim znakovima na oznakama grana koje kreću od korijena prema čvorovima-djeci  $v_1$  i  $v_2$  (iz korijena kreću tri grane čije su oznake: "\$", "A" i "C"). Krajnje desna grana na **Error!** **Reference source not found.** označena je s "C", pa put nastavljamo tom granom do čvora  $v_2$ . Iz  $v_2$  vode dvije grane: lijeva grana označena je s "A\$", a desna s "CA\$". Uspoređujemo drugi znak iz  $s_2$  (C) s prvim znakom u granama koje vode iz čvora  $v_2$ . S obzirom da oznaka desne grane počinje znakom C, nastavljamo s usporedbama znakova iz  $s_2$  i znakova na oznaci te grane: uspoređujemo treći znak iz  $s_2$  s drugim znakom na oznaci desne grane. S obzirom da se znakovi podudaraju, uspoređujemo onda i sljedeći par znakova: četvrti znak sufiksa  $s_2$  i treći znak oznake grane. S obzirom da se i taj par znakova podudara,

nastavljamo put tom granom. Kako više nije ostao niti jedan znak u  $s_2$ , a put istodobno završava u listu s oznakom 2, potvrdili smo da sufiks  $s_2$  ima pripadajući list u stablu  $T$ .

Na **Error! Reference source not found.** i Sl. 4.2 prikazane su *sufiksne veze* (engl. *suffix links*) kao crvene crtkane strjelice. Među ostalim, koriste se kod izgradnje sufiksnog stabla u linearnom vremenu (poglavlje 4.1.6).



Sl. 4.2 Sufiksno stablo za niz  $S\$ = \text{ACGGGC\$}$  ( $\$$  je leksikografski manji od znakova iz  $S$ ). Sufiksne veze označene su crvenim crtkanim strelicama.

Sufiksnu vezu (engl. *suffix link*) definiramo kao *pokazivač* od čvora  $x$  do čvora  $s(x)$ , uz uvjet da je:

1. Niz znakova  $zp$  oznaka puta do  $x$ , gdje je  $z$  znak, a  $p$  podniz ( $p$  može biti i prazan niz; tj.  $p = \epsilon$ ).
2. Niz znakova  $p$  oznaka puta do  $s(x)$ .

U primjeru na **Error! Reference source not found.** vidimo da je oznaka puta do čvora  $v_1$  "A", a oznaka puta do korijena  $v$  je  $\epsilon$  (prazan niz). Zbog toga je dodana sufiksna veza od čvora  $v_1$  do čvora  $v$  ( $z = \text{A}$ ,  $p = \epsilon$ ). Analogno, dodana je i sufiksna veza od čvora  $v_2$  do čvora  $v$  ( $z = \text{C}$ ,  $p = \epsilon$ ). Naravno, sufiksne veze se dodaju i između unutarnjih čvorova, što možemo vidjeti na složenijem primjeru (Sl. 4.2).

U primjeru na Sl. 4.2 neka je  $s x$  označen čvor na dubini dva, a to je čvor čija je oznaka puta  $zp = \text{GG}$ . Iz čvora  $x$  vodi sufiksna veza do čvora  $s(x)$  čija je oznaka puta  $p = \text{"G"}$ .

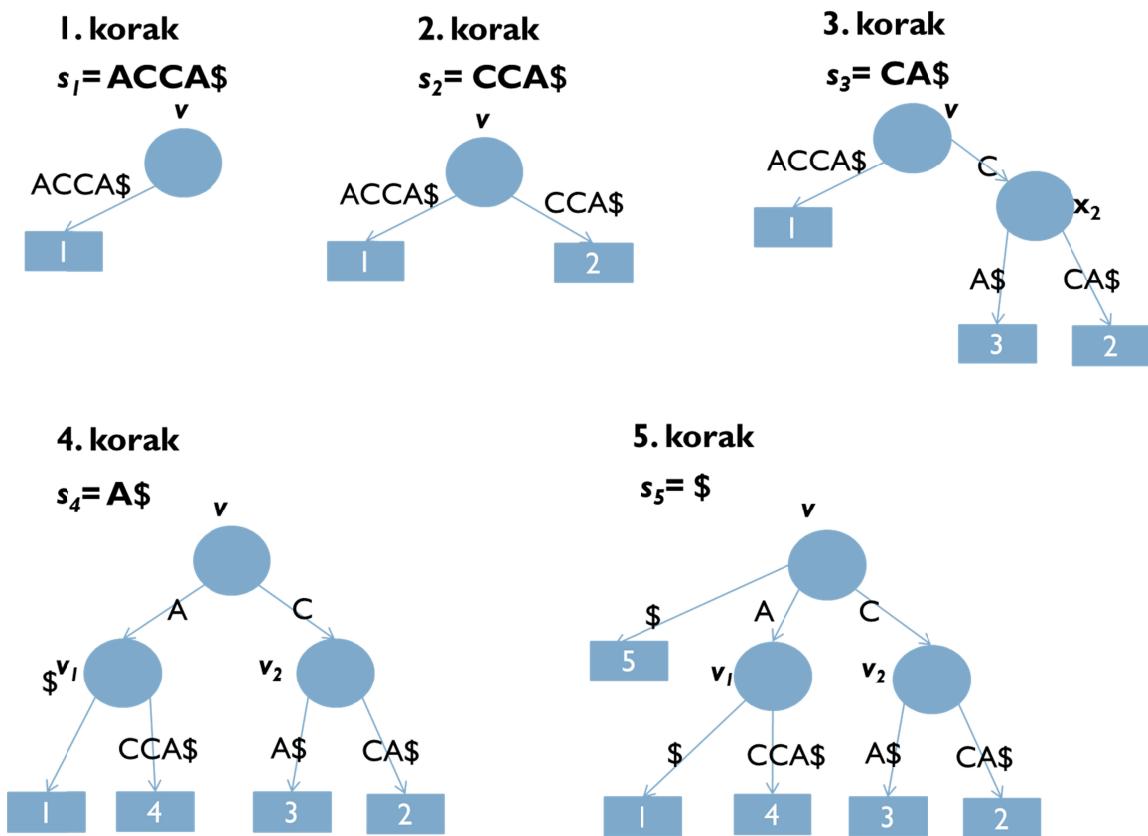
### 4.1.3. Memorijski zahtjevi

Sufiksno stablo ima široku primjenu u bioinformatici. Mnoštvo primjena opisano je posebno početkom 2000.-ih (npr. Kurtz *et al.*, 2004; Bray & Pachter, 2004). No, s vremenom je alternativna struktura podataka, sufiksno polje (poglavlje 4.2), zamijenilo

sufiksno stablo. Glavni razlog tome je što u praksi sufiksno stablo za ulazni niz od  $n$  znakova zahtijeva najmanje  $10n$  okteta (engl. *byte*) memorijskog prostora, a često  $15n$  -  $20n$  okteta (Kurtz, 1998a). S druge strane, sufiksno polje zauzima teorijski zauzima  $\Theta(n \log n)$  bita, a u praksi obično  $4n$  okteta.

#### 4.1.4. Izgradnja sufiksnog stabla u vremenu $O(n^2)$

Sufiksno stablo može se izgraditi u vremenu koje kvadratno ovisi o duljini ulaznog niza, tj. u vremenu  $O(n^2)$  za niz duljine  $n$ . Ideja je jednostavna ("naivni" algoritam): u stablu dodajemo sufiks po sufiks, počevši od sufiksa  $s_1$ , zatim  $s_2$  i tako dalje do  $s_n$ . Općenito, u  $i$ -tom koraku dodajemo sufiks  $s_i$  tako da za njega u sufiksno stablo dodajemo pripadajući list (s oznakom  $i$ ). Pri tome, ako je potrebno, dodajemo nove unutarnje čvorove i/ili nove grane koje vode iz postojećih ili dodanih čvorova.



Sl. 4.3 Izgradnja sufiksnog stabla „naivnim“ algoritmom za niz  $S = ACCA\$$  (prepostavljamo da je znak  $\$$  leksikografski manji od znakova iz  $S$ ). Vremenska složenost „naivnog“ algoritma je  $|S|$ .

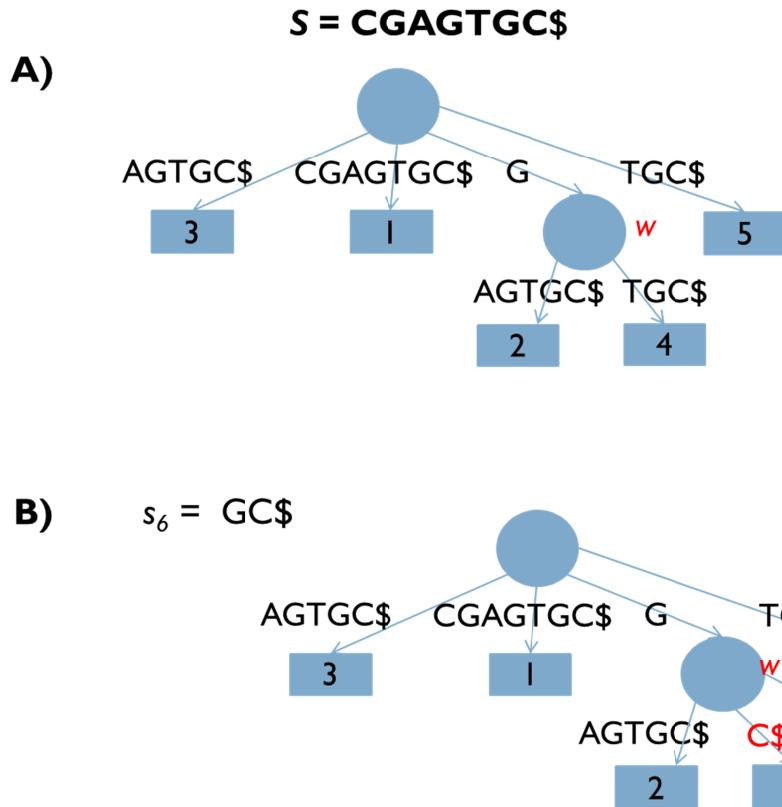
Primjer izgradnje sufiksno stabla "naivnim algoritmom" za niz ACCA\$ prikazan je na Sl. 4.3. U prvom koraku dodaje se u prazno stablo sufiks  $s_1$ : korijen  $v$  se spaja s listom označenim "1". Oznaka grane koja vodi do lista "1" označena je cijelim sufiksom  $s_1$ , tj. s "ACCA\$". Zatim u drugom koraku dodajemo sufiks  $s_2$ . S obzirom da oznaka jedine grane u stablu (koja spaja korijen  $v$  i list "1") počinje s "A", u stablo je potrebno dodati novu granu čija će oznaka početi s "C", a ujedno će sadržavati i sve ostale znakove iz sufiksa  $s_2$ . Ta grana će voditi od korijena  $v$  do lista označenog s "2".

U 3. koraku dodajemo sufiks  $s_3 = CA$$ , koji počinje s "C". Kako sada već postoji grana čija oznaka počinje s "C", sljedimo tu granu. Međutim, sljedeći znak na oznaci te grane je "C", što ne odgovara drugom znaku sufiksa  $s_3$  (koji je "A"), tj. došlo je do nepodudaranja znakova. Stoga, dodajemo novi čvor na mjestu gdje je došlo do nepodudaranja, tj. iza prvog znaka na oznaci postojeće grane (iza znaka "C"). Taj čvor ćemo označiti s  $v_2$  (kako bi notacija odgovarala Sl. 4.1). Iz čvora  $v_2$  će sada izlaziti dvije nove grane: jedna do postojećeg lista označenog s "2" (oznaka grane će biti "CA\$") i jedna do novog lista, kojeg ćemo označiti s "3", jer je se odnosi na sufiks  $s_3$ . Na granu do lista "3" upisujemo oznaku grane koja će sadržavati preostale znakove sufiksa  $s_3$ , tj. podniz  $s_3[2, 3] = A$$ .

U sljedećem koraku dodajemo sufiks  $s_4$ , koji počinje s "A". Ponovno polazeći od korijena  $v$ , tražimo postoji li grana koja počinje s "A". Pronalazimo granu koja vodi od  $v$  do lista "1". Kako ta grana počinje s "A", ali drugi znak na oznaci grane ne odgovara drugom znaku  $s_4$ , trebamo uvesti novi čvor, koji ćemo označiti s  $v_1$  i iz kojeg će izlaziti dvije nove grane. Jedna će nova grana voditi do postojećeg lista "1" i njezinu oznaku čine znakovi sufiksa  $s_1$  koji se nalaze nakon prvog znaka "A", tj. podniz "CCA\$". Na drugoj novoj grani će biti preostali znak sufiksa  $s_4$  (podniz  $s_4[2, 2] = \$$ ) i ta će grana voditi do novog lista označenog s "4".

Naposljetku, dodajemo sufiks  $s_5$ , čiji je jedini znak "\$". Kako iz korijena  $v$  ne izlazi niti jedna grana koja počinje s \$, dodajemo novu granu (oznaka grane je "") od  $v$  do novog lista "5". Time smo dodali sve sufikse u sufiksno stablo, odnosno, za svaki smo sufiks dodali pripadajući list i izgradili put do lista.

Općenito, kada dodajemo čvor  $s_i$  krećemo od korijena stabla i promatramo za svaki znak sufiksa  $s_i$  odgovara li oznakama po postojećim granama stabla sve dok ne najđemo na  $j$ -ti znak sufiksa  $s_i$  za koji ne postoji odgovarajuće podudaranje.



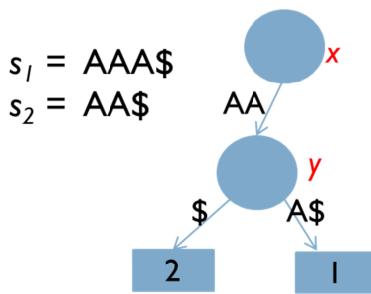
Sl. 4.4 "Naivni" algoritam za izgradnju sufiksnog stabla za niz  $S = CGAGTGC\$$  u kvadratnom vremenu. A) Sufiksno stablo nakon što su dodani sufiksi  $s_1$  do  $s_5$ . B) Dodavanje sufiksa  $s_6 = GC\$$  u sufiksno stablo: dodaje se nova grana s oznakom "C\$" iz čvora  $w$  koja vodi do lista 6.

1. Ako se nepodudaranje znakova dogodilo na nekom čvoru  $w$ , onda se od čvora  $w$  uvodi nova grana do novog lista s oznakom  $i$ . Znakovi koji će činiti oznaku nove grane su preostali znakovi sufiksa  $s_i$ , počevši od  $j$ -tog znaka sufiksa  $s_i$ , dakle:  $s_i[j, /s_6/]$ .

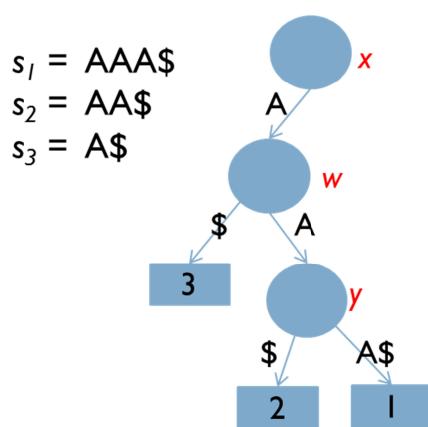
**Primjer:** želimo izgraditi sufiksno stablo nad nizom  $S = CGAGTGC\$$  korištenjem "naivnog" algoritma u vremenu  $O(|S|^2)$ . Na Sl. 4.4A) prikazano je stablo nakon što su dodani sufiksi  $s_1$  do  $s_5$ . Sl. 4.4B) prikazuje stablo nakon što je dodan sufiks  $s_6 = GC\$$ , za koji je dodana nova grana iz čvora  $w$  do lista označenog sa 6, a oznaka te grane je  $s_6[2, 3] = C\$$ .

$$S = AAA\$$$

A)



B)



Sl. 4.5 "Naivni" algoritam za izgradnju sufiksнog stabla za niz  $S = AAA\$$  u kvadratnom vremenu.

A) Sufiksno stablo nakon što su dodani sufiksi  $s_1$  i  $s_2$ . B) Dodavanje sufiksa  $s_3$  u sufiksno stablo tako da se dodaje novi čvor  $w$  te dvije nove grane: jedna grana će voditi do lista 3 (oznaka te grane je "\$"), a druga grana će povezivati čvorove  $w$  i  $y$  (oznaka te grane je "A").

2. Ako se nepodudaranje dogodilo između  $j$ -tog znaka sufiksa  $s_i$  i nekog znaka unutar oznake grane koja je povezivala postojeće čvorove  $x$  i  $y$ , onda se uvodi novi čvor  $w$  na grani između  $x$  i  $y$  (Sl. 4.5). Čvor  $w$  će ostati povezan s čvorom  $x$  postojećom granom, ali će njegina oznaka biti skraćena tako da završava sa znakom  $s_i[j - 1]$ . Iz čvora  $w$  će sada ići dvije grane: jedna do postojećeg čvora  $y$  i njegina će oznaka sadržavati preostale znakove iz stare oznake na grani ( $x, y$ ), počevši od znaka na kojem je došlo do nepodudaranja sa znakom  $s_i[j]$ . Druga će grana biti nova grana do novog lista s oznakom  $i$ , a oznaka na toj grani će biti niz  $s_i[j, /s_i/]$ .

**Primjer:** želimo izgraditi sufiksno stablo nad nizom  $S = AAA\$$  korištenjem "naivnog" algoritma u vremenu  $O(|S|^2)$ . Na Sl. 4.5A) prikazano je stablo nakon što su dodani sufiksi  $s_1$  i  $s_2$ . Na Sl. 4.5B) prikazano je stablo nakon što je dodan sufiks  $s_3$ . U stablu je dodan novi unutarnji čvor ( $w$ ) iz kojega vode dvije grane: grana s oznakom "A" vodi od čvora  $w$  do  $y$ , a grana s oznakom "\$" vodi od  $w$  do lista s oznakom 3. Grana od kojeg se korijenja do čvora  $w$  ima oznaku "A", jer je to najdulji prefiks sufiksa  $s_3$  i sufiksa koji su već bili u stablu ( $s_1$  i  $s_2$ ).

Broj sufiksa, odnosno listova, koje dodajemo u sufiksno stablo jednak je duljini ulaznog niza, tj.  $n$ . Neka je najdulji zajednički prefiks između  $s_i$  i sufiksa koji su već u stablu ( $s_1, s_2, \dots, s_{i-1}$ ) duljine  $k-1$ , a  $k$ -ti znak sufiksa  $s_i$  je prvi znak u kom se  $s_i$  razlikuje od prethodno dodanih sufiksa. Zbog toga je, u  $i$ -tom koraku kada dodajemo sufiks  $s_i$ , potrebno obaviti  $k$

usporedbi,  $k \leq |s_i|$ , Preostali znakovi sufiksa  $s_i$ , tj. podniz  $s_i[k, /s_i/]$ , se dodaju na oznaku nove grane koja vodi do lista označenog s  $i$ . Ukupno vrijeme potrebno za  $i$ -ti korak algoritma je  $\Theta(i)$ , a za dodavanje svih  $n$  sufiksa je  $\sum_{i=1}^n \Theta(i) = \Theta(i^2)$ .

#### 4.1.5. Implicitno sufiksno stablo

U sufiksnom stablu izgrađenom za neki niz  $S\$$ , zbog dodavanja jedinstvenog znaka  $\$$  na kraj niza  $S$ , svaki sufiks niza  $S\$$  u stablu ima svoj pripadajući list. Svaki se sufiks se može odrediti tako da se spoje svi znakovi na putu od korijena stabla prema listu.

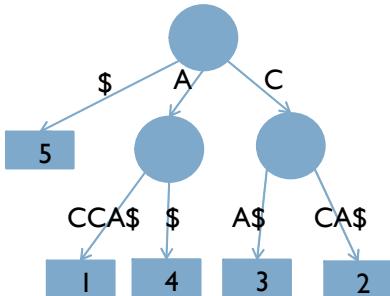
Implicitno sufiksno stablo (engl. *implicit suffix tree*) je stablo izgrađeno za neki niz  $S$ , gdje  $S$  ne završava posebnim znakom  $\$$  ( $\$$  ne postoji nigdje u  $S$ ) (Sl. 4.6). Zbog toga neki sufiksi niza  $S$  neće imati pripadajuće listove u stablu, nego će završavati u nekom unutarnjem čvoru stabla (na Sl. 4.6 vidljivo je da u implicitnom sufiksnom stablu izgrađenom za niz  $S=ACCA$  sufiks  $s_4=A$  ne završava u listu, nego unutar oznake grane koja vodi od korijena stabla do lista 1).

Ovo je postupak kojim se iz sufiksnog stabla  $T$ , izgrađenog za neki niz  $S\$$ , konstruira implicitno sufiksno stablo (Sl. 4.6):

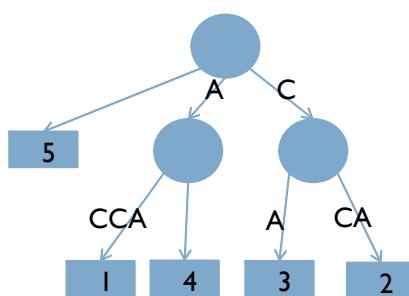
1. Obrisati znak  $\$$  sa svih oznaka grana u  $T$ .
2. Sve grane, kojima su oznake grana prazan niz, uklanjuju se iz stabla (na primjer: na Sl. 4.6 oznaka grane koja vodi od korijena do lista 5 je prazan niz, pa će se ta grana obrisati iz stabla).
3. Svi unutarnji čvorovi, koji više nemaju 2 djeteta, uklanjuju se iz stabla.

Implicitno sufiksno stablo se koristi u Ukkonenovom algoritmu (Ukkonen, 1995) za izgradnju sufiksnog stabla (poglavlje 4.1.6).

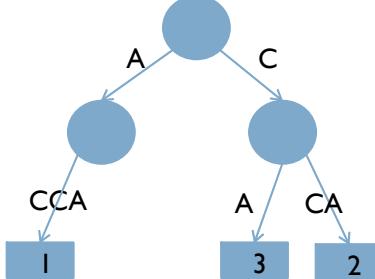
$S\$ = ACCA\$$



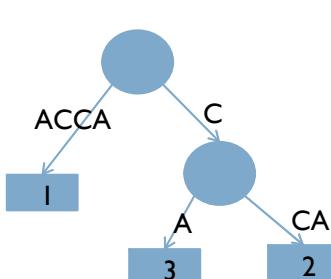
I. Obrisati oznake za kraj niza \$



2. Ukloniti grane koje sadrže samo prazan niz



3. Sve unutarnje čvorove, koji više nemaju 2 djjeteta, obrisati iz stabla



Sl. 4.6 Izgradnja implicitnog sufiksnog stabla iz sufiksnog stabla. Slika lijevo gore prikazuje sufiksno stablo za niz  $S\$ = ACCA\$$ . Slika desno dolje prikazuje implicitno sufiksno stablo za niz  $S = ACCA$ . Uočiti da sufiks  $s_4 = A$  (niza  $S = ACCA$ ) ne završava u listu, nego unutar grane koja vodi od korijena stabla do lista s oznakom 1.

#### 4.1.6. Ukkonenov algoritam – „naivna“ izvedba

U ovom će poglavlju biti izložena osnovna ideja izgradnje sufiksnog stabla „naivnom“ izvedbom Ukkonenovog algoritma (Ukkonen, 1995). Kao što je već spomenuto, Ukkonenov algoritam je tzv. *on-line* algoritam za koji ulazni niz  $S$  duljine  $n$  ne treba biti poznat u trenutku kada krećemo s izgradnjom stabla, nego se u svakom koraku gradi trenutno implicitno sufiksno stablo za onoliko znakova koliko je poznato u tom trenutku.

Počinje se s prefiksom  $S[1, 1]$  za koji se gradi implicitno sufiksno stablo  $T_1$ . Zatim se za sljedeći prefiks  $S[1, 2]$  gradi stablo  $T_2$  tako da se proširi postojeće stablo  $T_1$ , tj. u stablu  $T_1$  dodaju se sufiksi niza  $S[1, 2]$ :  $S[1, 2]$  i  $S[2, 2]$ .

Općenito, u  $(i+1)$ -tom koraku algoritma gradi se *implicitno sufiksno stablo*  $T_{i+1}$  za prefiks  $S[1, i+1]$  tako da se proširuje stablo iz prethodnog koraka,  $T_i$ . Postupak se ponavlja sve do zadnjeg prefiksa  $S[1, n]$ . Kada se izgradilo implicitno sufiksno stablo  $T_n$ , gradi se implicitno sufiksno stablo  $T_{n+1}$  za  $S\$$ , čime se na kraj svakog sufiksa dodaje jedinstveni

znak za kraj niza \$. Time je osigurano da implicitno sufiksno stablo  $T_{n+1}$  bude ujedno i obično sufiksno stablo.  $T_{n+1}$  je konačno rješenje, odnosno sufiksno stablo za niz S\$.

Ključni postupak u  $(i+1)$ -tom koraku algoritma je proširenje stabla  $T_i$  u stablo  $T_{i+1}$  dodavanjem svih sufiksa prefiksa  $S[1, i+1]$  u stablo  $T_i$ . Prefiks  $S[1, i+1]$  ima  $i+1$  sufiksa ( $S[1, i+1], S[2, i+1], \dots, S[i+1, i+1]$ ) za koja je možda potrebno napraviti proširenje (engl. *extension*) u stablu  $T_i$  (ili se ne obavlja ništa).

U izgradnji stabla  $T_{i+1}$  (iz stabla  $T_i$ ) sa svakim sufiksom  $S[j, i+1]$ ,  $1 \leq j \leq i+1$ , postupa se prema jednom od tri sljedeća pravila (Sl. 4.7):

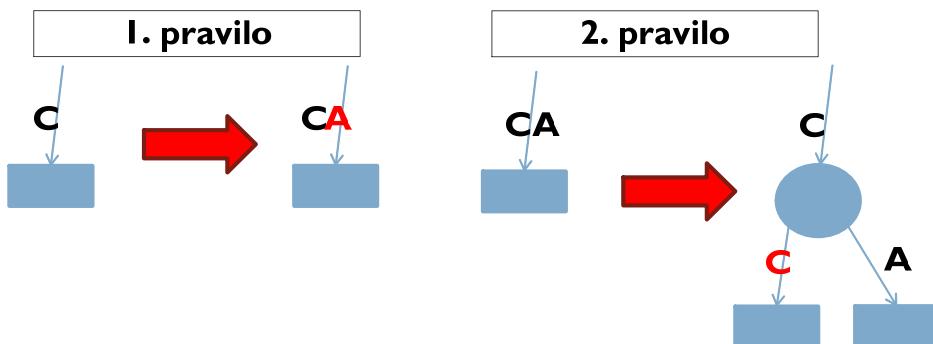
1. Za sufiks se produljuje postojeća grana do lista (tj. proširuje se oznaka grane).
2. Za sufiks se u stablo dodaju novi list i grana koja vodi do lista.  
Napomena: ovo je jedino pravilo prema kojemu se može dodati novi list u stablo.
3. Ako sufiks već postoji u stablu, ne obavlja se ništa.

### 1. pravilo

Neka u stablu postoji put  $S[j, i]$  koji završava kao list. Tada se dodaje znak  $S[i+1]$  na kraj oznake grane koja vodi do tog lista. Ovo pravilo se još zove "jednom list, uvijek list" (engl. *once a leaf, always a leaf*).

**Primjer** (Sl. 4.7; lijevo):

Neka u koraku  $i$  (izgradnja stabla  $T_i$ ) sufiks  $S[j, i] = C$  završava u listu stabla. U  $(i+1)$ -tom koraku algoritma (u kojem stablo  $T_i$  proširujemo u stablo  $T_{i+1}$ ) dodajemo u  $T_i$  sufiks  $S[j, i+1] = CA$  i to tako da znak "A" dodajemo na kraj postojeće oznake grane "C". Sada je nova oznaka grane "CA" i time smo osigurali da i novi sufiks  $S[j, i+1]$  završava u listu stabla  $T_{i+1}$ .



Sl. 4.7 Ukkonenov algoritam: prikaz pravila 1 i 2 za određivanje proširenja sufiksa (pojašnjjenje vidjeti u tekstu).

## 2. pravilo

Ako u stablu  $T_i$  postoji put  $S[j, i]$  i iz njega se ne nastavlja niti jedan put koji počinje znakom  $S[i+1]$ , onda trebamo dodati novu granu koja će voditi do novog lista s oznakom  $j$ . Oznaka grane će biti  $S[i+1, i+1]$ .

Pri tome, ako  $S[j, i]$  završava unutar grane, onda trebamo dodati i novi unutarnji čvor u stablu.

**Primjer** (Sl. 4.7; desno):

Neka u koraku  $i$  (izgradnja stabla  $T_i$ ) niz  $S[j, i] = C$  završava u grani s oznakom "CA". U  $(i+1)$ -tom koraku algoritma (u kojem stablo  $T_i$  proširujemo u stablo  $T_{i+1}$ ) dodajemo u  $T_i$  sufiks  $S[j, i+1] = CC$ . S obzirom da u stablu  $T_i$  ne postoji put "CC" (tj.  $S[j, i]C$ ), trebamo postojecu granu s oznakom "CA", podijeliti u dvije grane te dodati treću granu, novi unutarnji čvor i novi list. U stablo dodajemo novi čvor *iza* znaka  $S[j, i]$ , iz kojeg će sada ići dvije grane: jedna s oznakom "A" do postojecog lista, a druga s oznakom "C" do novog lista (koji odgovara sufiksu  $S[j, i+1] = CC$ ). Do novog čvora će voditi grana s oznakom "C".

## 3. pravilo

Ako u stablu  $T_i$  postoje putovi koji se nastavljaju na  $S[j, i]$ , a jedan od njih počinje znakom  $S[i+1]$ , onda ne treba učiniti ništa, jer sufiks  $S[j, i+1]$  već postoji kao put u stablu u  $T_i$ , a time i u stablu  $T_{i+1}$ .

Ukkonenov algoritam u naivnoj izvedbi prikazan je na Sl. 4.8.

```
Izgradi  $T_1$ 
za  $i = 1$  do  $n - 1$  radi
/* korak  $i+1$ : izgradnja  $T_{i+1}$  iz  $T_i$  */
za  $j = 1$  do  $i + 1$  radi /*  $j$ -to proširenje */
    /* osigurati da  $S[j, i+1]$  bude u stablu korištenjem pravila 1, 2 ili 3 */
    U trenutnom stablu pronaći kraj puta koji počinje u korijenu, a označen
    je s  $S[j, i]$ .
    Ako je potrebno, dodati na kraj puta znak  $S[i+1]$ .
    kraj
kraj
```

Sl. 4.8 Ukkonenov algoritam – naivna izvedba

Primjer izgradnje sufiksnog stabla za niz  $S=ACCA\$$  „naivnom“ izvedbom Ukkonenovog algoritma prikazan je na Sl. 4.9.

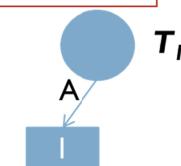
$S = ACCA\$$

Pravilo 1: produljivanje oznake grane do lista

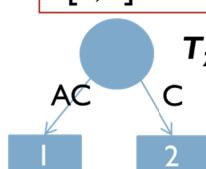
Pravilo 2: dodavanje nove grane

Pravilo 3: ako sufiks već postoji u stablu, ne obavlja se ništa

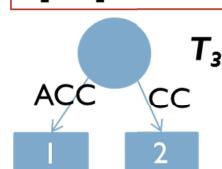
$S[1, 1] = A$



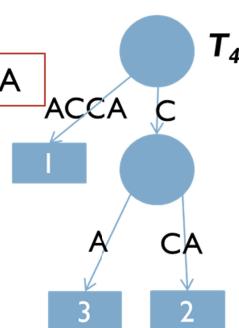
$S[1, 2] = AC$



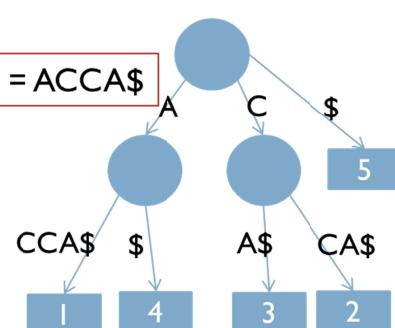
$S[1, 3] = ACC$



$S[1, 4] = ACCA$



$S[1, 5] = ACCA\$$



Sl. 4.9 Izgradnja sufiksnog stabla za niz  $S = ACCA\$$  „naivnom“ izvedbom Ukkonenovog algoritma.

### Vremenska složenost „naivne“ izvedbe Ukkonenovog algoritma

Neka je zadan niz  $S$  duljine  $n$  koji završava jedinstvenim znakom za kraj niza ( $\$$ ). Za  $S$  se gradi ukupno  $n$  implicitnih sufiksnih stabala ( $T_1$  do  $T_n$ ), gdje u izgradnji stabla  $T_{i+1}$  ima ukupno  $(i+1)$  koraka (zbog dodavanja  $i+1$  sufiksa prefiksa  $S[1, i+1]$ ).

U  $(i+1)$ -koraku dodaju se sufiksi  $S[j, i+1]$ , gdje je  $1 \leq j \leq i+1$ . Za svaki sufiks  $S[j, i+1]$  potrebno je u stablu pronaći put  $S[j, i]$  na koji se dodaje znak  $S[i+1]$  (pravilo 1 ili 2), ako  $S[j, i+1]$  već ne postoji u stablu (pravilo 3). Traženje  $S[j, i]$  obavlja se u vremenu  $O(i+1-j)$ .

Dakle, ukupna je složenost:  $\sum_{i=1}^n \sum_{j=1}^{i+1} O(i+1-j) = O(n^3)$

Korištenjem poboljšanja pravila 1-3 i sufiksnih veza, složenost se smanjuje na  $O(n)$  (potpoglavlje 4.1.7).

#### 4.1.7. Ukkonenov algoritam u linearном vremenu

Pravila 1-3, koja se koriste u naivnoj izvedbi Ukkonenovog algoritma (potpoglavlje 4.1.6), mogu se unaprijediti i ta su unaprjeđenja temelj Ukkonenovog algoritma u linearnom vremenu.

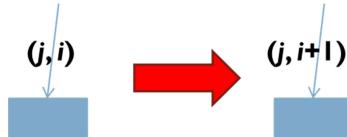
##### Unaprjeđenje pravila

###### 1. pravilo (implicitno proširenje) - unaprjeđenje

Grane, koje vode do listova, označavaju se, umjesto znakovima, indeksima  $p$  i  $k$ , tj. kao  $(p, k)$ , gdje su  $p$  i  $k$  početak i kraj podniza  $S[p, k]$  koji je oznaka te grane (engl. *edge-label compression*), čime se postiže memorijska ušteda (Sl. 4.10).

Posljedica je da su, u  $(i+1)$ -koraku, oznake grana koje vode do listova oblika  $(p, K = i+1)$ . Postavljanjem globalnog indeksa  $K$  na  $i+1$  u  $O(1)$  vremenu, implicitno se ažuriraju sve oznake grana koje vode do listova.

Štoviše, kroz svih  $n$  koraka (izgradnja stabala  $T_1$  do  $T_n$ ), za ukupno sva implicitna proširenja potrebno vrijeme je  $O(n)$ .



Sl. 4.10 Promjena oznake grane koja vodi do lista iz  $(j, K = i)$  u  $i$ -tom koraku, u  $(j, K = i+1)$  u  $(i+1)$ -tom koraku, gdje je  $K$  globalna varijabla. Promjena se obavlja u  $(i+1)$ -koraku za sve listove odjednom u vremenu  $O(1)$ .

###### 2. pravilo (eksplicitno proširenje) - unaprjeđenje

Neka indeks  $j_{\text{prev}}$  označava posljednji list koji je eksplicitno dodan u  $i$ -tom koraku (izgradnja stabla  $T_i$ ). Tada su "kandidati" za eksplicitno proširenje, tj. za dodavanje novog lista u stablo  $T_{i+1}$ , samo oni sufiksi  $S[j, i+1]$  za koje vrijedi  $j \geq j_{\text{prev}} + 1$ . Svi ostali sufiksi ( $j \leq j_{\text{prev}}$ ) su već dodani u stablo i za njih se samo može proširiti oznaka grane, odnosno, na njih se može samo primijeniti pravilo 1.

S obzirom da Ukkonenov algoritam ukupno ima  $n$  koraka, tj. ukupno postoji samo  $n$  listova koji se mogu dodati u stablo, tada je  $j$  ograničen s  $n$ . Dakle, obavlja se ukupno  $n$  eksplicitnih proširenja kroz svih  $n$  koraka zajedno.

### 3. pravilo (pravilo zaustavljanja) - unaprjeđenje

Promatramo  $(i+1)$ -korak, tj. izgradnju stabla  $T_{i+1}$ . Ako u stablu postoje putevi koji se nastavljaju na  $S[j, i]$ , a jedan od njih počinje s  $S[i+1]$ , onda ne treba učiniti ništa, tj.  $S[j, i+1]$  već postoji u stablu.

Posljedica je da su onda također i podnizovi  $S[j+1, i+1], \dots, S[i+1, i+1]$  već u stablu (sufiksi se u stablo unose slijeva nadesno, pa ako u stablu postoji  $S[j, i+1]$ , onda postoje i svi njegovi sufiksi). Stoga nam ovo pravilo omogućuje završetak  $(i+1)$ -koraka (engl. "showstopper").

Dakle,  $(i+1)$ -korak završavamo kada je  $j > i + 1$  ili za prvi  $j$  za koji vrijedi pravilo 3.

Korištenjem unaprjeđenja pravila 1-3 te sufiksnih veza kod eksplisitnih proširenja (pravilo 2) osigurava se izvođenje Ukkonenovog algoritma u vremenu  $O(n)$  za niz duljine  $n$  (Sl. 4.11).

#### Ukkonenov algoritam

**Ulaz:** niz  $S$ ,  $|S| = n$

**Izlaz:** sufiksno stablo izgrađeno nad nizom  $S$

Izgradi  $T_1$

$j_{\text{pret}} = 1$

**za**  $i = 1$  **do**  $n - 1$  **radi** /\* ukupno:  $n$  eksplisitnih proširenja,  $|S| = n$  \*/

/\* korak  $i+1$ : izgradnja  $T_{i+1}$  iz  $T_i$  \*/

Obavi implicitna proširenja, tj. postavi  $K = i+1$  /\* Pravilo 1 za  $j \leq j_{\text{pret}}$  \*/

**za**  $j = j_{\text{pret}} + 1$  **do**  $i + 1$  **radi** /\*  $j$ -to proširenje \*/

/\* osiguravamo da je  $S[j, i+1]$  u stablu: Pravilo 2 ili 3 \*/

U trenutnom stablu pronađi kraj puta koji počinje u korijenu, a označen je s  $S[j, i]$ .

Ako je potrebno, dodaj na kraj puta znak  $S[i+1]$ . /\* Pravilo 2 \*/

$j_{\text{pret}} := j$  /\* priprema za sljedeći korak \*/

**ako** je primjenjeno Pravilo 3

**onda**  $j_{\text{pret}} := j - 1$  i završi  $(i+1)$ -korak

**kraj**

**kraj**

Sl. 4.11 Ukkonenov algoritam za izgradnju sufiksнog stabla za niz  $S$

## Korištenje sufiksnih veza kod dodavanja proširenja

Promatramo  $(i+1)$ -korak kada želimo dodati sufiks  $S[j, i+1]$  u stablu (Sl. 4.12).

Za  $j=1$ , potrebno je naći kraj puta čija je oznaka  $S[1, i]$  u stablu i onda ga proširujemo za znak  $S[i+1]$ . Za drugo (i svako sljedeće) proširenje u  $(i+1)$ -koraku, umjesto da krećemo od korijena tražiti podniz  $S[j, i+1]$ , tražimo čvor  $v$  ispod kojeg završava put označen s  $S[j-1, i+1]$ , a jedino ako nema  $v$ , onda krećemo od korijena.

Kada smo pronašli  $v$ , onda slijedimo sufiksnu vezu  $(v, s(v))$ , gdje je  $s(v)$  čvor ispod kojeg završava  $S[j, i+1]$ . Traženje kraja puta s oznakom  $S[j, i]$  nastavljamo od  $s(v)$  i onda dodajemo znak  $S[i+1]$  iza  $S[j, i]$ . Tako smo preskočili znakove na putu od korijena do  $s(v)$ .

Na putu od  $s(v)$  do  $S[j, i]$  "skače" se od čvora do čvora (na Sl. 4.12 označeno zelenim, crtkanim strjelicama), kako bi se preskočili znakovi na granama i ubrzao postupak traženja (engl. *skip/count trick*).

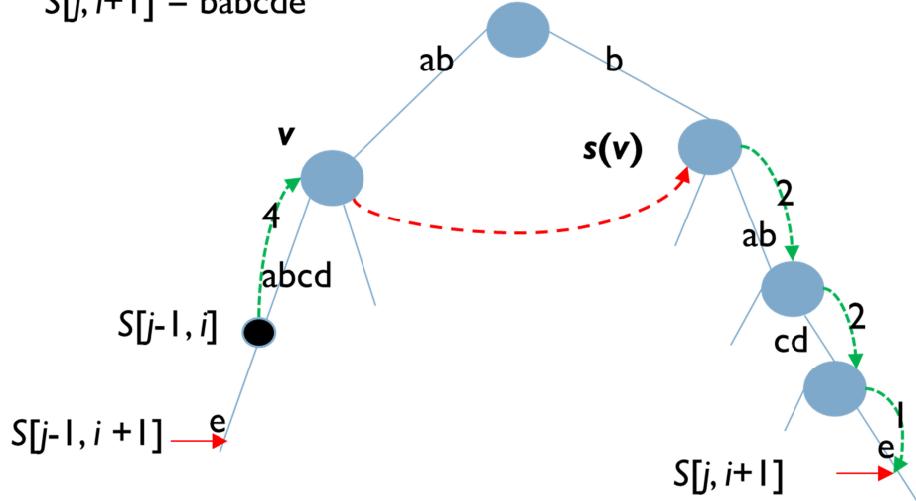
Kako smo znali trebamo li skočiti na sljedeći čvor ili stati usred grane? Za svaku granu koja kreće iz nekog čvora pohranjuju se podatci o znaku s kojim počinje oznaka grane te koliko ima znakova na toj grani (Sl. 4.12). Da bi znali kojom granom trebamo krenuti, dovoljno je provjeriti prvi znak na oznaci grane, a da bi znali trebamo li stati unutar grane ili preskočiti granu, trebamo provjeriti je li broj znakova na grani veći od broja znakova koje još trebamo provjeriti na putu do  $S[j, i]$ . Ako je broj znakova manji, onda "hodamo" po grani dok ne nađemo na mjesto gdje bi trebali dodati novi list, tj. znak  $S[i+1]$ . Ako je veći, preskačemo do čvora-djeteta i onda tamo opet provjeravamo kojom granom ćemo krenuti usporedbom s prvim znakom na oznakama grana iz čvorova-djeteta.

Na Sl. 4.12, nakon čvora  $s(v)$  preskočena su dva čvora dok se nije došlo do grane na koju treba dodati znak  $S[i+1]$ , čime je osigurano da sufiks  $S[j, i+1]$  bude u stablu.

Ovaj postupak zahtijeva  $O(n)$  "hodanja" s čvora na čvor prema dolje (engl. *down-walk*), jer je  $n$  maksimalna dubina čvora u stablu. Štoviše, kroz svih  $n$  eksplicitnih proširenja postupak ovaj zahtijeva  $O(n)$  vremena (Gusfield, 1997).

$$S[j-1, i+1] = ababcde$$

$$S[j, i+1] = babcde$$



Sl. 4.12 Korištenje sufiksne veze kod eksplisitnog proširenja. U ovom se primjeru dodaje novi list za sufiks  $S[j, i+1]$  (pojašnjeno u tekstu).

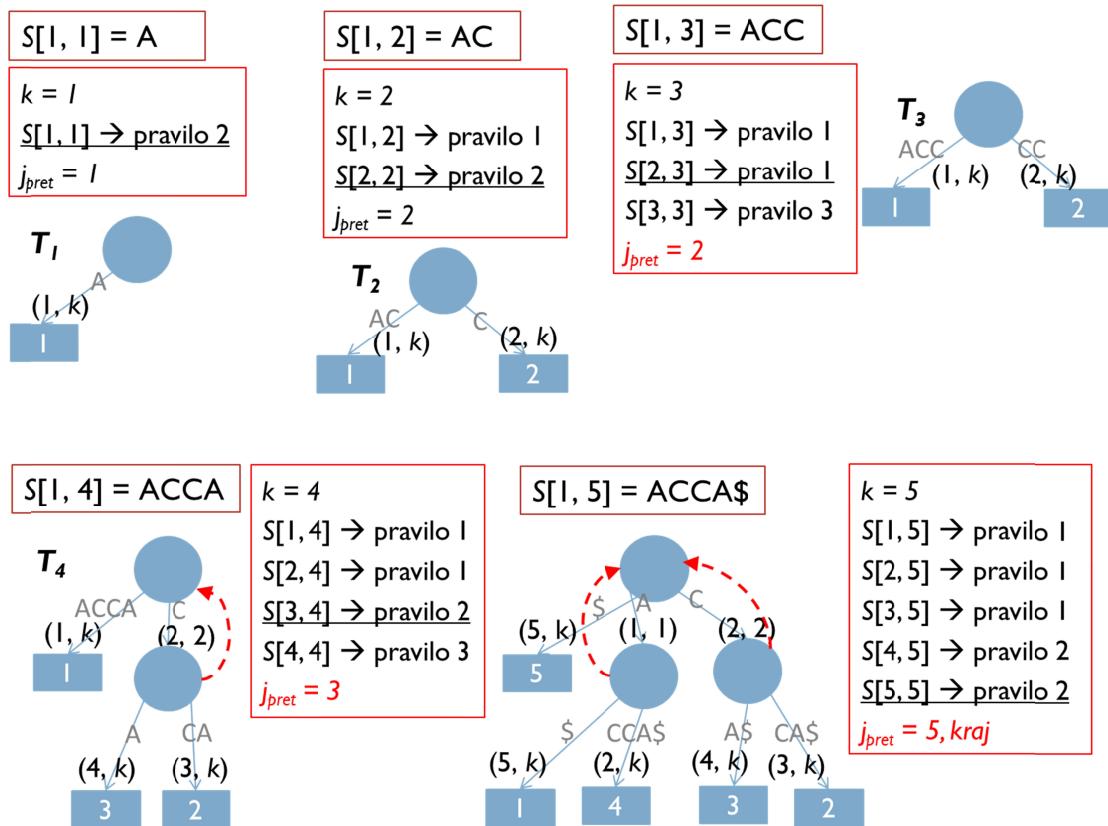
Na Sl. 4.13 prikazan je primjer izgradnje sufiksnog stabla za niz  $S = \text{ACCA\$}$  Ukkonenovim algoritmom u vremenu  $O(|S|)$ . Izgradnja stabla odvija se u pet koraka, jer se u stablo dodaje pet prefiksa niza  $S$ :  $S[1, 1], S[1, 2], \dots, S[1, 5]$  te se za svaki prefiks gradi implicitno sufiksno stablo  $T_1$  do  $T_5$ . Stablo  $T_{i+1}$  gradi se proširenjem stabla  $T_i$ .

Prvi je korak izgradnja stabla  $T_1$  (Sl. 4.13) u kojem se dodaje jedini sufiks prefiksa  $S[1, 1]$ , a to je  $S[1, 1]$ . Za  $S[1, 1]$  se u stablo dodaje list s oznakom 1 (primjena pravila 2: eksplisitno dodavanje lista u stablo). Ujedno se i vrijednost  $j_{\text{prev}}$  postavlja na indeks sufiksa koji je posljednji dodan u stablo, tj.  $j_{\text{prev}} = 1$ . Indeks  $j_{\text{prev}}$  nam služi kako bi u sljedećem koraku (izgradnja  $T_2$ ) znali koji sufiksi su „kandidati“ za dodavanje u stablo, a to su oni sufiksi kojima je indeks početka na poziciji  $\geq j_{\text{prev}} + 1$  (Sl. 4.11).

Stablo  $T_2$  gradi se proširenjem stabla  $T_1$  tako da se dodaju sufiksi prefiksa  $S[1, 2]: S[1, 2]$  i  $S[2, 2]$ . Podniz  $S[1, 2]$  dodaje se implicitno u stablo tako da se indeks završnog znaka ( $k$ ) na grani koja vodi do lista s oznakom 1 promijeni s  $k = 1$  na  $k = 2$ . Sufiks  $S[2, 2]$  se dodaje primjenom pravila 2: dodaje se novi list s oznakom 2, a početni indeks sufiksa (ujedno i oznaka lista) koji je posljednji dodan u stablo postavlja se na  $j_{\text{prev}} = 2$ , kako bi se u stablu  $T_3$  krenulo s dodavanjem sufiksa od  $j_{\text{prev}} + 1$ .

Kada se u stablu  $T_4$  dodaje novi unutarnji čvor, tada je za taj čvor potrebno dodati sufiksnu vezu na korijen. Općenito, za svaki unutarnji čvor  $v$ , koji se dodaje pri dodavanju sufiksa  $S[j, i+1]$ , najkasnije u sljedećem koraku (dodavanje sufiksa  $S[j+1, i+1]$ ), dodat će se

sufiksna veza na čvor  $s(v)$ , jer će najkasnije tada biti dodan čvor  $s(v)$  (Gusfield, 1997). Čvor  $s(v)$  je korijen stabla, ako je  $|S[j, i+1]| = 1$  ( $S[j, i+1]$  je samo jedan znak), odnosno ako je  $S[j+1, i+1] = \epsilon$  (prazan niz). U primjeru za  $T_4$  dodaje se novi unutarnji čvor kada se dodaje sufiks  $S[3, 4] = C$ . S obzirom da je tada  $S[4, 4] = \epsilon$ , dodaje se sufiksna veza iz unutarnjeg čvora prema korijenu stabla. Kod izgradnje  $T_5$  dodaje se i druga sufiksna veza iz unutarnjeg čvora prema korijenu.



Sl. 4.13 Izgradnja sufiksnog stabla za niz  $S = \text{ACCA\$}$  Ukkonenovim algoritmom u vremenu  $O(|S|)$ .

#### 4.1.8. Poopćeno sufiksno stablo

Mnogi problemi vezani uz znakovne nizove, kao npr. traženje najduljeg zajedničkog podniza dvaju (ili više) nizova zahtijevaju izgradnju sufiksnog stabla za dva (ili više) nizova. Sufiksno stablo koje je izgrađeno nad skupom ulaznih nizova  $\{S_1, \dots, S_k\}$  zove se *poopćeno sufiksno stablo* (engl. *generalized suffix tree*).

Poopćeno sufiksno stablo  $T$  gradi se nad konkatenacijom nizova  $S_1\$ \dots S_k\$$ , gdje svaki niz  $S_i$  završava jedinstvenim znakom za kraj niza  $\$$ , a za koje vrijedi  $\$_i \neq \$_j$ ,  $1 \leq i, j \leq n$ ,  $i \neq j$ . Stablo  $T$  sadrži sve sufikse nizova nad kojim je izgrađeno. Izgradnja poopćenog sufiksnog stabla obavlja se u vremenu  $O(|S_1| + \dots + |S_k|)$ .

## 4.2. Sufiksno polje

Manber i Myers su 1990. (Manber & Myers, 1990) predložili novu strukturu podataka kao memorijski manje zahtjevnu alternativu sufiksnom stablu: sufiksno polje (engl. *suffix array*) (detaljnije u potpoglavlju 4.1.3). Sufiksno polje je polje ne-negativnih cijelih brojeva, koji predstavljaju početne pozicije abecedno poredanih sufiksa ulaznog niza.

### Primjer:

Neka je zadan ulazni niz  $S = \text{ACCA}$ . Analogno, postupku izgradnje sufiksnog stabla, na kraj niza  $S$  dodajemo jedinstveni znak  $\$$ , koji se ne nalazi u abecedi  $\Sigma$  nad kojom je  $S$  izgrađen. Prepostavljamo da je  $\$$  abecedno manji od znakova iz  $\Sigma$ .

Sufiksi niza  $S\$$  su:  $s_1 = \text{ACCA}\$$ ,  $s_2 = \text{CCA}\$$ ,  $s_3 = \text{CA}\$$ ,  $s_4 = \text{A}\$$ ,  $s_5 = \$$ .

Abecedno poredani sufiksi su:

$$s_5 = \$$$

$$s_4 = \text{A}\$$$

$$s_1 = \text{ACCA}\$$$

$$s_3 = \text{CA}\$$$

$$s_2 = \text{CCA}\$$$

Dakle, sufiksno polje SA sadrži početne pozicije abecedno poredanih sufiksa:  $\text{SA} = [5, 4, 1, 3, 2]$ .

Sufiksno polje može zamijeniti sufiksno stablo, odnosno, svaki problem koji se može riješiti korištenjem sufiksnih stabala, može se riješiti i korištenjem sufiksnog polja s istom asimptotskom složenošću (Abouelhoda *et al.*, 2004; Puglisi *et al.*, 2006)

Mnoštvo je algoritama koji omogućuju konstrukciju sufiksnog polja (engl. *Suffix Array Construction Algorithm; SACA*). Originalni algoritam, koji su osmislili Manber i Myers (1990), omogućavao je izgradnju sufiksnog polja u vremenu  $O(n \log n)$  za ulazni niz od  $n$

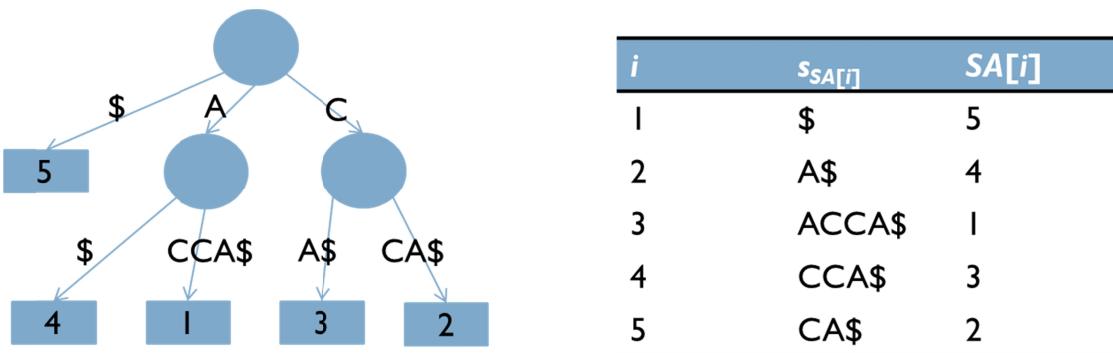
znakova. Danas postoje algoritmi koji sufiksno polje mogu konstruirati u linearnom vremenu (Kärkkäinen & Sanders, 2003; Ko & Aluru, 2003; Nong *et al.*, 2011). Također postoje i algoritmi koji imaju supralinearne vrijeme izvođenja za najgori slučaj, ali u praksi mogu biti podjednako brzi kao najbolji algoritmi s linearnim vremenom konstrukcije, a u nekim slučajevima mogu i biti brži od njih (Puglisi *et al.*, 2006; Nong *et al.*, 2011).

Kao što je već spomenuto, teorijsko memorijsko zauzeće sufiksog polja je  $\Theta(n \log n)$  bita, dok je to u praksi obično  $4n$  okteta. Najbolji algoritmi za izgradnju sufiksog polja zahtijevaju ukupno tek nešto više od  $5n$  okteta memorijskog prostora za ulazni niz duljine  $n$ : od toga ulazni niz i sufiksno polje zauzimaju zajedno  $5n$  okteta, a ostatak je za pomoćne strukture (Nong *et al.*, 2011).

#### 4.2.1. Sufiksno polje i sufiksno stablo

Neka je zadan niz  $S$  duljine  $n$ . Sufiksno polje  $SA$  može se izravno dobiti iz sufiksog stabla *preorder* obilaskom stabla (Sl. 4.14). Svaki list stabla nosi oznaku pripadajućeg sufiksa. Primjer: prvi list slijeva je list s oznakom "5", što znači da je abecedno najmanji sufiks  $s_5$ , odnosno da je  $SA[1] = 5$ . Sljedeći list na koji nailazimo obilaskom stabla je list s oznakom "4", što znači da je sljedeći po abecedno

m redu sufiks  $s_4$ , tj.  $SA[2] = 4$ , itd.



Sl. 4.14 Sufiksno stablo i sufiksno polje (polje  $SA$ ) za niz  $S = ACCA\$$ . Član polja  $SA[i]$  odgovara listu u sufiksnom stablu s oznakom  $i$ .

#### 4.2.2. Izgradnja sufiksog polja u linearnom vremenu Kärkkäinen-Sandersovim algoritmom

Osnovna ideja postojećih algoritama za konstrukciju sufiksog polja u linearnom vremenu je podjela ulaznog niza u dva dijela prema nekom pravilu, npr. u nizove  $S'$  i  $S''$ . Jedan niz,

npr.  $S'$ , se sortira (tj. odredi se  $\text{SA}(S')$ ) u rekurzivnom pozivu, a onda se drugi niz  $S''$  sortira korištenjem prvog (tj. odredi se  $\text{SA}(S'')$ ). Na kraju se odredi spojeno sufiksno polje  $\text{SA}(S)$  korištenjem  $\text{SA}(S')$  i  $\text{SA}(S'')$ .

Ta ideja koristi se i kod Kärkkäinen-Sandersovog algoritma (Kärkkäinen & Sanders, 2003): iz ulaznog niza  $S$ , tvore se nizovi  $S'$  i  $S''$ . Zatim se određuje  $\text{SA}(S')$ , a na temelju  $\text{SA}(S')$  generira se  $\text{SA}(S'')$ , a zatim i  $\text{SA}(S)$  u linearном vremenu.

Niz  $S'$  se sastavlja od trojki (podnizova iz  $S$ ) tako da je  $|S'| < 2|S|/3$ . Ukupno vrijeme izvođenja algoritma je  $\Theta(n)$  za niz  $S$  duljine  $n$ .

### **Kärkkäinen-Sandersov algoritam:**

1. Iz niza  $S$  duljine  $n$  oblikovati nizove  $S'$  i  $S''$ .
2. Sortirati  $S'$ , što ujedno znači rekurzivno sortirati  $2n/3$  sufiksa niza  $S$  koji počinju na poziciji  $i \bmod 3 \neq 0$ .
3. Sortirati  $S''$ ; što ujedno znači sortirati preostale sufikse ( $n/3$  sufiksa) koji počinju na poziciji  $i \bmod 3 = 0$  koristeći rezultat 2. koraka.
4. Napraviti spajanje (engl. *merge*) sortiranih sufiksa i sufiksnih polja dobivenih u koracima (1) i (2).

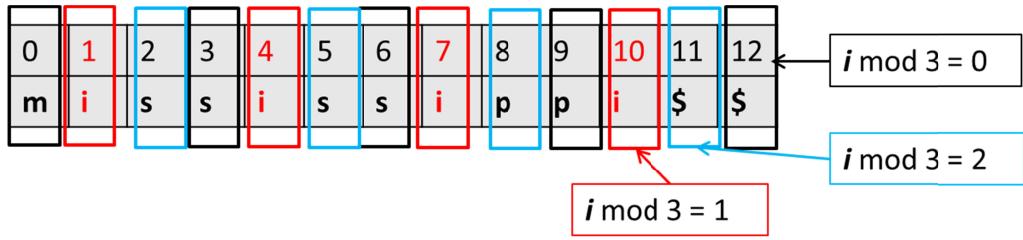
Niz  $S'$ , koji se još naziva *1,2-niz*, tvori se na sljedeći način: svaki znak  $S'[i]$  gradi se iz tri uzastopna znaka iz  $S$ , tj. iz podniza oblika  $S[i, i+2]$ , gdje je  $i \bmod 3 \neq 0$  (ili  $i = 1, 2$ ). Indeks  $i$  odabire se sljedećim redoslijedom:  $i = 1, 4, 7, \dots, 2, 5, 8, \dots$ , tj. prvo se za izgradnju  $S'$  dodaju svi podnizovi  $S[i, i+2]$  gdje je  $i \bmod 3 = 1$ , a onda svi podnizovi  $S[i, i+2]$  gdje je  $i \bmod 3 = 2$ .

### **Primjer**

Neka je zadan  $S = \text{mississippi} \$\$$  (na kraj niza su dodana 2 znaka \$, kako bi zadnji indeks znaka iz  $S$  bio višekratnik od 3).

Prvo se iz niza  $S$  tvori niz  $T'$ , gdje je  $T'$  konkatenacija podnizova  $S[i, i+2]$ ,  $i \bmod 3 \neq 0$  i to tako da je  $i$  odabran sljedećim redoslijedom:  $i = 1, 4, 7, \dots, 2, 5, 8, \dots$

Ovdje je  $T' = \text{ississippi} \$\$ \text{ssissippi}$  (Sl. 4.15).  $S'$  se tvori iz  $T'$  tako da se svaki podniz  $T'[3i-2, 3i]$  zamjeni jedinstvenim leksikografskim imenom  $S'[i]$  uz uvjet:  $S'[i] \leq S'[j]$ , ako i samo ako  $T'[i, i+2] \leq T'[j, j+2]$ . Na primjer,  $T'[0, 2] = \text{iss}$  zamjenjujemo znakom "C" (**Error! Reference source not found.B**).



B)  $T' = \text{ississippi\$\$ssissippi}$

$S' = \text{CCBAEED}$

iss	iss	ipp	i\$	ssi	ssi	ppi
C	C	B	A	E	E	D

$S' = \text{CCBAEED}$

4	3	2	1	7	6	5
---	---	---	---	---	---	---

Sufiksno polje od  $S'$

AEED	4
BAEED	3
CBAEED	2
CCBAEED	1
D	7
ED	6
EED	5

Sl. 4.15 Kärrkäinen-Sandersov algoritam A) Tvorba niza  $T' = \text{ississippi\$\$ssissippi}$  iz niza  $S = \text{mississippi\$\$}$ . Niz  $T'$  čine konkatenirani podnizovi oblika  $S[i, i+2]$ , tako da su prvo dodani podnizovi gdje je  $i \bmod 3 = 1$ , a zatim podnizovi gdje je  $i \bmod 3 = 2$  (tj.  $i = 1, 4, 7, 10, 2, 5, 8, 11$ ).

B) Svaki jedinstveni podniz  $T'[3i-2, 3i]$  zamjenjuje se jedinstvenim znakom (leksikografskim imenom)  $S'[i]$ ; npr.  $T'[1, 3] = \text{iss}$  zamjenjuje se znakom „C“. Podnizovi iz  $T'$  su sortirani *radix*-sortiranjem, te je svaki podniz zamijenjen svojim rangom predstavljenim odgovarajućim leksikografskim imenom. U ovom se primjeru niz  $S'$  ne sastoji od jedinstvenih znakova (leksikografskih imena), pa je redoslijed sufiksa iz  $S'$  potrebno odrediti u rekurzivnom pozivu.

Ovakvim se postupkom povezuje svaka trojka iz  $T'$  sa svojim rangom: leksikografsko ime iz  $S'$  odgovara rangu pripadajuće trojke iz  $T'$ . Ako su leksikografska imena u  $S'$  jedinstvena, tj. niti jedno ime se ne ponavlja, tada je  $T'$  sortiran, a onda su automatski sortirani i pripadajući sufiksi iz  $S$ . Poredak trojki iz  $T'$  određuje se *radix*-sortiranjem, a onda se njihov rang zamjenjuje leksikografskim imenom, odnosno novim znakom. Na primjer,  $T'[0, 2] = \text{iss}$  zamjenjujemo znakom "C", što odgovara rangu 3 (znak "A" ima rang 1, znak "B" ima rang 2, itd.). Ako leksikografska imena u  $S'$  nisu jedinstvena (tj. više trojki iz  $T'$  ima isti rang), onda se poredak sufiksa iz  $S'$  određuje u rekurzivnom pozivu.

Važno je primijetiti da sortirani sufiksi od  $S'$  imaju isti poredak kao i sufiksi od  $T'$ , odnosno temeljem  $\text{SA}(S')$  određuje se  $\text{SA}(T')$ .

U 3. koraku algoritma redoslijed sufiksa iz  $S''$  određuje se korištenjem prethodno sortiranih sufiksa niza  $S'$ . Prvo se iz niza  $S$  oblikuje niz  $T''$  koji čine trojke  $S[i, i+2]$ ,  $i \bmod 3 = 0$ . Ovdje je  $T'' = \text{mississippi\$}$ .

Trojke iz  $T''$  se zatim sortiraju korištenjem radix-sortiranja (Sl. 4.16A)), gdje se dvije trojke  $t_i = S[i, i+2]$  i  $t_j = S[j, j+2]$  promatraju kao  $S[i]s_{i+1}$  u odnosu na  $S[j]s_{j+1}$ , a  $s_{i+1}$  i  $s_{j+1}$  su sufksi čiji poredak već znamo iz  $S'$ . Preciznije, poredak  $t_i$  i  $t_j$  je određen s  $S[i]$  i  $S[j]$ , ako  $S[i] \neq S[j]$ . Ako  $S[i] = S[j]$ , onda je poredak  $t_i$  i  $t_j$  određen s  $s_{i+1}$  i  $s_{j+1}$ , čiji međusobni poredak znamo iz  $S'$ . To je zapravo radix-sortiranje parova  $(S_i, \text{rang}(s_{i+1}))$ .

Na primjer (Sl. 4.16A)), ako promatramo  $T''[1, 3] = \text{mis}$  i  $T''[4, 6] = \text{sis}$ , onda se ta dva podniza razlikuju po prvom znaku. Podnizovi  $T''[4, 6] = \text{sis}$  i  $T''[7, 9] = \text{sip}$  imaju isti prvi znak, ali međusobni poredak sufiksa  $s_5$  i  $s_8$  znamo iz  $S'$ , gdje je  $s_8$  leksikografski prije  $s_5$ .

A)  $T' = \text{ississippi\$\$ssissippi}$   
 $T'' = \text{mississippi\$}$

	1	4	7	10		1	10	7	4
$T''$	mis	sis	sip	pi\$		mis	pi\$	sip	sis

### B) Spajanje $S'$ i $S''$

S1. 4.16 Kärkkäinen-Sandersov algoritam (nastavak). **A)** Sortiranje  $T''$  (odnosno  $S''$ ) radix-sortiranjem uz korištenje ranga (poretka) trojki iz  $T'$  (odnosno ranga sufiksa iz  $S'$ ). **B)** Određivanje sufiksнog polja  $SA(S)$  prema sortiranim sufiksima u  $T'$  (tј.  $S'$ ) i  $T''$  (tј.  $S''$ ).

U 4. koraku spajamo sortirana polja  $S'$  i  $S''$  (Sl. 4.16B)): usporedba 2 sufiksa obavlja se u vremenu  $O(1)$  kako bi se spajanje cijelih polja obavilo u vremenu  $O(n)$ . Koristimo sljedeća pravila:

1. Sufiksi  $s_i$  ( $i \bmod 3 = 0$ ) i  $s_j$  ( $j \bmod 3 = 1$ ) uspoređuju se tako da se usporedi podnizovi  $S[i, i+2]$  i  $S[j, j+2]$ . Ako je  $S[i] \neq S[j]$ , onda je leksikografski  $s_i$  ispred  $s_j$ . Ako  $S[i] =$

$S[j]$ , onda se uzima redoslijed (rang) sufiksa  $s_{i+1}$  i  $s_{j+1}$ , koji se preuzimaju iz sortiranog polja  $S'$ .

2. Sufiksi  $s_i$  ( $i \bmod 3 = 2$ ) i  $s_j$  ( $j \bmod 3 = 0$ ) se uspoređuju tako da se usporedi njihova prva 2 znaka. Analogno prethodnom slučaju, ako su prva 2 znaka ista za oba sufiksa, onda se uspoređuju sortirani rangovi  $s_{i+2}$  i  $s_{j+2}$ , koji se preuzimaju iz sortiranog polja  $S'$ .

#### 4.2.3. Izgradnja sufiksnog polja u linearном vremenu Nong-Zhang-Chanov algoritmom

Trenutno najpopularniji i najbrži algoritam za izgradnju sufiksnog polja u linearnom vremenu je Nong-Zhang-Chanov SA-IS algoritam (Nong *et al.*, 2009, 2011). U tom se algoritmu koriste neke ideje iz Ko-Alurovog algoritma (Ko and Aluru, 2003). Međutim, uvođenjem LMS-podniza (pojašnjeno kasnije u tekstu) ostvarena su značajna poboljšanja u brzini izvođenja i memorijskoj potrošnji u odnosu na Ko-Alurov algoritam, ali i druge postojeće algoritme za određivanje sufiksnog polja u linearном vremenu. Također, originalna verzija programskog koda Nong-Zhang-Chanovog SA-IS algoritma (Nong *et al.*, 2009, 2011) ima samo stotinjak linija koda u programskom jeziku C.

##### Osnovni pojmovi i definicije

Neka je  $S$  niz sastavljen od znakova iz abecede  $\Sigma$  i neka  $S$  završava znakom  $\$$ , koji je abecedno manji od ostalih znakova iz  $\Sigma$  i ne pojavljuje se nigdje unutar  $S$ . Kao i do sada, sufiks koji počinje na  $i$ -tom mjestu u nizu  $S$  označavat ćemo  $s_i = S[i, |S|]$ .

##### 1. $S$ -tip sufiksa (engl. $S$ -type)

$s_i$  je  $S$ -tip sufiksa

- ako je  $s_i < s_{i+1}$  ili
- ako je  $s_i = \$$

##### 2. $L$ -tip sufiksa (engl. $L$ -type)

$s_i$  je  $L$ -tip sufiksa

- ako je  $s_i > s_{i+1}$

##### 3. $S$ -tip znaka

$S[i]$  je S-tip znaka

- ako je  $S[i] < S[i + 1]$  ili
- ako je  $S[i] = S[i + 1]$  i  $s_{i+1}$  je S-tip

#### 4. L-tip znaka

$S[i]$  je L-tip znaka:

- ako je  $S[i] > S[i + 1]$  ili
- ako je  $S[i] = S[i + 1]$  i  $s_{i+1}$  je L-tip

#### Primjer

Neka je zadan niz  $S = \text{ATTAGCGAGCG\$}$ . Za svaki znak u nizu  $S$  određujemo je li S-tip ili L-tip znaka prolaskom kroz niz  $S$  zdesna na lijevo, tj. od zadnjega prema prvome znaku (Tablica 4.1). Oznaka je li znak S-tip ili L-tip pohranjuje se u polje  $t$ . Implementacijski se  $t$  pohranjuje kao polje bitova, gdje 0 označava L-tip znaka, a 1 S-tip znaka.

Tablica 4.1 Određivanje tipa znaka (S-tip ili L-tip) za svaki znak niza  $S = \text{ATTAGCGAGCG\$}$

	0	1	2	3	4	5	6	7	8	9	10	11
$S$	A	T	T	A	G	C	G	A	G	C	G	\$
$t$	S	L	L	S	L	S	L	S	L	S	L	S

#### 5. LMS-znak (engl. leftmost S-type)

$S[i]$  je LMS-znak ako je  $S[i]$  S-tip i  $S[i - 1]$  L-tip (za  $i \geq 1$ )

#### 6. LMS-sufiks

$s_i$  je LMS-sufiks ako je  $S[i]$  LMS-znak

#### 7. LMS-podniz

$S[i, j]$  je LMS-podniz

- ako je  $S[i, j]$  znak za kraj niza ( $i = j$ ) ili
- ako su  $S[i]$  i  $S[j]$  LMS-znakovi ( $i \neq j$ ), a između njih nema drugih LMS-znakova

#### 8. Poredak LMS-podnizova

Poredak između dva LMS-podniza određuje se njihovom usporedbom slijeva nadesno: za svaki par znakova promatra se prvo njihov leksikografski poredak, a ako imaju istu leksikografsku vrijednost, onda se promatra jesu li znakovi S-tipa ili L-tipa, gdje S-tip ima viši prioritet od L-tipa (tj. S-tip je leksikografski nakon L-tipa).

Dva LMS-podniza su jednaka, ako su jednake duljine i ako su im svi znakovi leksikografski jednaki te istog tipa.

## 9. *LMS-prefiks*

LMS-prefiks je

- LMS-znak ili
- prefiks  $S[i, k]$  sufiksa  $s_i$  takav da je  $S[k]$  prvi LMS-znak nakon  $S[i]$

LMS-prefiks je S-tipa, ako je sufiks  $s_i$  S-tipa.

LMS-prefiks je L-tipa, ako je sufiks  $s_i$  L-tipa. Svaki LMS-prefiks L-tipa ima najmanje 2 znaka.

Svaki LMS-podniz je ili je znak za kraj niza ili LMS-prefiks duljine veće od jedan.

### Primjer

Neka je zadan niz  $S = \text{ATTAGCGAGCG\$}$ . Kako bi odredili LMS-podnizove za  $S$ , koristimo polje  $t$  koje sadrži oznake je li neki znak iz  $S$  L-tip ili S-tip (Tablica 4.1). Indekse LMS-podnizova pohranjujemo u polje  $P_1$ . Dakle, u ovom slučaju  $P_1 = \{3, 5, 7, 9, 11\}$ . Na primjer, prvi LMS-podniz gledano slijeva počinje na poziciji 3 i završava na poziciji 5. Znakovi na mjestima 3 i 5 su LMS-znakovi, a između njih nema niti jedan drugi LMS-znak. Zadnji LMS-podniz je po definiciji znak za kraj niza, tj.  $S[11]$ .

Možemo primijetiti da su LMS-podnizovi  $S[3, 5]$  i  $S[7, 9]$  jednaki, jer su jednaki po duljini, a svi parovi znakovi su međusobno jednaki i abecedno i po tipu znaka.

### SA-IS algoritam ( $S, SA$ ):

*Ulasz:* niz  $S$  ( $|S| = n$ );  $S$  je sastavljen iz znakova iz abecede  $\Sigma$  i završava sa znakom  $\$$ .

*Izlaz:* sufiksno polje  $SA$ .

*Pomoćna polja:*  $t, P_1, B$

$B$  je polje pokazivača na početak ili kraj pojedinog pretinca (engl. *bucket*), takav da pojedini pretinac uključuje sve sufikse s istim početnim slovom,  $|B| = \Sigma$ .

- Za ulazni niz  $S$  odrediti pomoćna polja  $t$  (polje tipova znakova: S-tip/L-tip) i polje  $p$  (polje početnih pozicija LMS-podnizova).

$$|t| = n$$

$$|P_I| = n_I \text{ (} n_I \text{ – broj LMS-podnizova od } S \text{)}$$

- Inducirano sortirati LMS-podnizove korištenjem  $P_I$  i  $B$ .
- Imenovati svaki LMS-podniz prema pripadajućem rangu, čime se dobije novo polje  $S_I$ .
- Ako svaki znak iz  $S_I$  ima jedinstveno ime, onda se izravno određuje  $SA_I$  za  $S_I$ , a inače pozvati  $SA\text{-IS}(S_I, SA_I)$ .
- Odrediti  $SA$  iz  $SA_I$ .

Napomena: algoritam na više mesta koristi svojstvo da je sufiks S-tipa abecedno iza sufiksa L-tipa kada oba počinju istim početnim znakom. Na primjer, za niz  $S = GTTGC\$$  (Tablica 4.1) sufiks  $s_0 = GTTGC\$$  je S-tip, a sufiks  $s_3 = GC\$$  je L-tip i leksikografski je  $s_0$  iza  $s_3$ .

To pravilo se koristi kada se sufiksi s istim početnim znakom  $z$  sortiraju unutar pretinca: S-sufiksi dodaju se na kraj pretinca za znak  $z$ , a L-sufiksi na početak pretinca za znak  $z$ .

#### *Korak 2: Inducirano sortiranje LMS-podnizova korištenjem $P_I$ i $B$ (Sl. 4.17)*

- Postaviti sve članove polja  $SA$  na -1.

Postaviti pokazivač  $B[k]$  na *kraj*  $k$ -tog podpolja od  $SA$  za svaki  $k = \Sigma[0], \dots, \Sigma[|\Sigma|]$ .

Proći kroz polje  $S$  slijeva nadesno:

- dodavati indekse LMS-sufiksa iz  $S$  u pripadajuće podpolje od  $SA$  od kraja podpolja prema početku. Nakon svakog dodavanja, pomaknuti pokazivač  $B[k]$  za jedno mjesto ulijevo.

- Postaviti pokazivač  $B[k]$  na *početak*  $k$ -tog podpolja od  $SA$  za svaki  $k = 0, \dots, |\Sigma|$ . Proći kroz polje  $SA$  slijeva nadesno:

- za svaki  $SA[i] > 0$ : ako je znak  $k = S[SA[i] - 1]$  L-tip, onda dodati  $SA[i] - 1$  na početak  $k$ -tog podpolja i pomaknuti  $B[k]$  jedno mjesto udesno.

- Postaviti pokazivač  $B[k]$  na *kraj*  $k$ -tog podpolja od  $SA$  za svaki  $k = 0, \dots, |\Sigma|$ . Proći kroz polje  $SA$  zdesna nalijevo:

- za svaki  $SA[i] > 0$ : ako je znak  $k = S[SA[i] - 1]$  S-tip, onda dodati  $SA[i] - 1$  na kraj  $k$ -tog podpolja i pomaknuti  $B[k]$  jedno mjesto ulijevo

	0	1	2	3	4	5	6	7	8	9	10	11
<b>S</b>	A	T	T	A	G	C	G	A	G	C	G	\$
<b>t</b>	S	L	L	S	L	S	L	S	L	S	L	S
<b>P<sub>1</sub></b>	3	5	7	9	11							

$$B['\$'] = 0; \quad B['A'] = 3; \quad B['C'] = 5; \quad B['G'] = 9;$$

$$B['T'] = 11;$$

$$\text{1. korak: } S = \text{ATTAGCGAGCG\$}$$

$$\begin{array}{ccccccc} \$ & A & & C & G & & T \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 \end{array}$$

$$SA = \{-1\} \{1, -1, -1\} \{1, -1\} \{1, -1, -1, -1\} \{1, -1\}$$

$$SA = \{11\} \{1, 7, 3\} \{9, 5\} \{1, -1, -1, -1\} \{1, -1\}$$

$$B['\$'] = -1; \quad B['A'] = 1; \quad B['C'] = 3; \quad B['G'] = 9;$$

$$B['T'] = 11;$$

2. korak:  $S = \text{ATTAGCGAGCG\$} \xrightarrow{\leftarrow}$

$$B['\$'] = 0; \quad B['A'] = 1; \quad B['C'] = 4; \quad B['G'] = 6; \quad B['T'] = 10$$

$$\begin{array}{ccccccc} \$ & A & C & G & & & T \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 \end{array}$$

$$SA = \{11\} \{1, 7, 3\} \{9, 5\} \{1, -1, -1, -1\} \{1, -1\}$$

$$SA = \{11\} \{1, 7, 3\} \{9, 5\} \{10, -1, -1, -1\} \{1, -1\} \quad i = 0 \rightarrow \text{dodati } 10, B['G'] = 7$$

$$SA = \{11\} \{1, 7, 3\} \{9, 5\} \{10, 6, -1, -1\} \{1, -1\} \quad i = 2 \rightarrow \text{dodati } 6, B['G'] = 8$$

$$SA = \{11\} \{1, 7, 3\} \{9, 5\} \{10, 6, -1, -1\} \{2, -1\} \quad i = 3 \rightarrow \text{dodati } 2, B['T'] = 11$$

$$SA = \{11\} \{1, 7, 3\} \{9, 5\} \{10, 6, 8, -1\} \{2, -1\} \quad i = 4 \rightarrow \text{dodati } 8, B['G'] = 9$$

$$SA = \{11\} \{1, 7, 3\} \{9, 5\} \{10, 6, 8, 4\} \{2, -1\} \quad i = 5 \rightarrow \text{dodati } 4; B['G'] = 10;$$

$$SA = \{11\} \{1, 7, 3\} \{9, 5\} \{10, 6, 8, 4\} \{2, 1\} \quad i = 10 \rightarrow \text{dodati } 1; B['T'] = 12;$$

3. korak:  $S = \text{ATTAGCGAGCG\$} \xleftarrow{\leftarrow}$

$$B['\$'] = 0; \quad B['A'] = 3; \quad B['C'] = 5; \quad B['G'] = 9; \quad B['T'] = 11$$

$$\begin{array}{ccccccc} \$ & A & C & G & & & T \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 \end{array}$$

$$SA = \{11\} \{1, 7, 3\} \{9, 5\} \{10, 6, 8, 4\} \{2, 1\}$$

$$SA = \{11\} \{1, 7, 0\} \{9, 5\} \{10, 6, 8, 4\} \{2, 1\} \quad i = 11 \rightarrow \text{dodati } 0; B['A'] = 2;$$

$$SA = \{11\} \{-1, 3, 0\} \{9, 5\} \{10, 6, 8, 4\} \{2, 1\} \quad i = 9 \rightarrow \text{dodati } 3; B['A'] = 1;$$

$$SA = \{11\} \{7, 3, 0\} \{9, 5\} \{10, 6, 8, 4\} \{2, 1\} \quad i = 8 \rightarrow \text{dodati } 7; B['A'] = 0;$$

$$SA = \{11\} \{7, 3, 0\} \{9, 5\} \{10, 6, 8, 4\} \{2, 1\} \quad i = 7 \rightarrow \text{dodati } 5; B['C'] = 4;$$

$$SA = \{11\} \{7, 3, 0\} \{9, 5\} \{10, 6, 8, 4\} \{2, 1\} \quad i = 6 \rightarrow \text{dodati } 9; B['C'] = 3$$

Sl. 4.17 Inducirano sortiranje LMS-podnizova korištenjem  $P_1$  i  $B$  za niz  $S = \text{ATTAGCGAGCG\$}$ .

Niz  $B$  je niz pokazivača na početak ili kraj pretnica (unutar  $SA$ ) za pojedini znak abecede.

## Primjer

Na Sl. 4.17 prikazan je postupak induciraniog sortiranja LMS-podnizova (2. korak SA-IS algoritma) za niz  $S = \text{ATTAGCGAGCG\$}$ . U ovom se postupku zapravo sortiraju LMS-prefiksi i to tako da se u prvom koraku sortiraju LMS-prefiksi S-tipa duljine 1, zatim se iz njih sortiraju LMS-prefiksi L-tipa (2. korak), a u trećem se koraku sortiraju svi LMS-prefiksi dulji od jednog znaka tako da se dodaju LMS-prefiksi S-tipa dulji od jednog znaka. Dokaz je izložen u (Nong *et al.*, 2009). Kako je svaki LMS-podniz ujedno i LMS-prefiks duljine veće od jedan, tako sortiranjem LMS-prefiksa sortiramo i LMS-podnizove.

Na kraju 2. koraka SA-IS algoritma sufiksno polje možemo prikazati na sljedeći način:  $SA = \{\{11\} \{7, 3, 0\} \{9, 5\} \{10, 6, 8, 4\} \{2, 1\}\}$ , gdje su pojedina podpolja (pretnici) unutar  $SA$  koji sadrže indekse sufiksa koji počinju istim početnim znakom. Npr. podpolje

$\{11\}$  sadrži samo indeks sufiksa  $\$$ ; podpolje  $\{7, 3, 0\}$  sadrži indekse tri sufiksa ( $s_7, s_3, s_0$ ) koja počinju slovom A, itd.

U ovome trenutku  $SA$  još nije konačan, tj. sufiksi unutar pojedinih pretinaca (podpolja od  $SA$ ) nisu međusobno dobro poredani.

### *Korak 3: Imenovanje LMS-podnizova*

Svakom LMS-podnizu pridjeljuje se novo leksikografsko ime (cijeli broj) tako da taj broj odgovara rangu LMS-podniza. Ako su LMS-podnizovi jednaki, tj. ako su jednake duljine i na svim mjestima u oba LMS-podniza su jednaki znakovi i ti su znakovi istog tipa, onda imaju i ista pridijeljena imena.

Leksikografska imena pridijeljena LMS-podnizovima oblikuju novi niz  $S_I$ .

Imena se pridjeljuju na sljedeći način:

Prolazi se kroz polje  $SA$  dobiveno u drugom koraku SA-IS algoritma (u  $SA$  su sada svi LMS-prefiksi sortirani) za  $i = 0$  do  $n - 1$ . Imena se pridjeljuju počevši od 0. Međusobni redoslijed dva LMS-podniza, koja su susjedna u  $SA$ , određuje se njihovom usporedbom (uspoređuju se parovi znakova i njihovi tipovi). Svakom LMS-podnizu pridjeljuje se odgovarajuće ime: ili novo ili isto kao prethodniku (ako su međusobno jednaki). Provjera je li promatrani sufiks iz  $SA$  ujedno i LMS-podniz, obavlja se u  $O(1)$  vremenu tako da se provjeri je li početni znak sufiksa LMS-znak.

### **Primjer**

Za niz  $S = \text{ATTAGCGAGCG\$}$  na Sl. 4.17 prikazano je polje  $SA = \{\{11\} \quad \{7, 3, 0\} \quad \{9, 5\} \quad \{10, 6, 8, 4\} \quad \{2, 1\}\}$  i polje  $P_I = \{3, 5, 7, 9, 11\}$ . LMS-podnizovi se imenuju prolaskom kroz polje  $SA$  slijeva nadesno.

Na primjer, prvi član  $SA$  slijeva nadesno je  $SA[0] = 11$ , što je ujedno i indeks jednog od LMS-podnizova. Zato se tome LMS-podnizu (" $\$$ ") pridjeljuje najmanji leksikografski broj (što je onda i rang tog LMS-podniza): 0. Sljedeći član  $SA$  koji sadrži indeks LMS-podniza je  $SA[1] = 7$ . LMS-podnizu koji počinje na mjestu 7 ("AGC") pridjeljuje se onda rang 1. Isti rang će se pridijeliti i LMS-indeksu koji počinje na poziciji 3 ("AGC"), jer su ta dva LMS-podniza jednaka: i po duljini, i po znakovima i po tipovima znakova. Konačni izgled

polja  $S_I$ , koje sadrži leksikografska imena pridijeljena LMS-podnizovima, je sljedeći  $S_I = "13120"$ .

#### *Korak 4: Određivanje $SA_I$ za $S_I$*

Ako svaki znak iz  $S_I$  ima jedinstveno ime, onda se izravno određuje  $SA_I$  za  $S_I$ , a inače pozvati  $SA\text{-IS}(S_I, SA_I)$ . U slučaju rekurzivnog poziva, implementacijski se  $S_I$  i  $SA_I$  pohranjuju unutar originalnog  $SA$  (zato jer je  $S_I$  sigurno najviše pola duljine od  $S$ ).

#### **Primjer**

U primjeru za niz  $S = \text{ATTAGCGAGCG\$}$  (Sl. 4.17) svi znakovi u  $S_I$  nisu jedinstveni, kao što smo pokazali u primjeru uz 3. korak, jer je  $S_I = "13120"$  (znak "1" se ponavlja dva puta u nizu  $S_I$ ). Prema tome, za  $S_I$  se ne može izravno odrediti  $SA_I$ , pa treba pozvati  $SA\text{-IS}(S_I, SA_I)$ . Rekurzivnim pozivom određuje se  $SA_I = \{4, 2, 0, 3, 1\}$ .

#### *Korak 5: Odrediti $SA$ iz $SA_I$*

- Postaviti sve članove  $SA$  na -1. Postaviti  $B[k]$  na indeks zadnjeg mesta u pretincu (podpolju od  $SA$ ) za svaki znak  $k$  iz abecede  $\Sigma$ .

Prolazeći kroz  $SA_I$  od kraja prema početku, staviti  $P_I[SA_I[i]]$  na kraj pretinca za pripadajući sufiks i pomaknuti  $B[k]$  za jedno mjesto ulijevo.

- (isto kao u 2. koraku, 2. stavka) Postaviti pokazivač  $B[k]$  na *početak*  $k$ -tog podpolja od  $SA$  za svaki  $k = 0, \dots, |\Sigma|$ . Proći kroz polje  $SA$  slijeva nadesno:

- za svaki  $SA[i] > 0$ : ako je znak  $k = S[SA[i] - 1]$  L-tip, onda dodati  $SA[i] - 1$  na početak  $k$ -tog podpolja i pomaknuti  $B[k]$  jedno mjesto udesno.

- (isto kao u 2. koraku, 3. stavka) Postaviti pokazivač  $B[k]$  na *kraj*  $k$ -tog podpolja od  $SA$  za svaki  $k = 0, \dots, |\Sigma|$ . Proći kroz polje  $SA$  zdesna nalijevo:

- za svaki  $SA[i] > 0$ : ako je znak  $k = S[SA[i] - 1]$  S-tip, onda dodati  $SA[i] - 1$  na kraj  $k$ -tog podpolja i pomaknuti  $B[k]$  jedno mjesto ulijevo

#### **Primjer**

Za niz  $S = \text{ATTAGCGAGCG\$}$  (Sl. 4.17), odredili smo polje  $P_I = \{3, 5, 7, 9, 11\}$ . U prethodnom smo primjeru (za 4. korak) vidjeli da LMS-podnizovi iz  $S$  nemaju jedinstvena

leksikografska imena, tj. da je niz  $S_I = "13120"$ . Zbog toga je bilo potrebno pozvati rekurzivno  $SA\text{-IS}(S_I, SA_I)$  čime smo izračunali  $SA_I = \{4, 2, 0, 3, 1\}$ . Sada koristimo  $SA_I$  kako bi odredili konačan rezultat, tj.  $SA$ .

Prvo, postavljamo sve članove  $SA$  na -1 ( $SA$  je prikazan kao skup podpolja/pretinaca, gdje se svako podpolje odnosi na sufikse koji počinju istim prvim znakom iz abecede  $\Sigma$ ):

$$SA = \{-1\} \ \{-1, -1, -1\} \ \{-1, -1\} \ \{-1, -1, -1, -1\} \ \{-1, -1\}$$

Zatim se svaki element  $B[k]$  iz polja pokazivača na pretince  $B$  postavlja tako da  $B[k]$  pokazuje na kraj podpolja iz  $SA$  za svaki znak  $k$  iz abecede  $\Sigma$ , tj.:

$$B['\$'] = 0; B['A'] = 3; B['C'] = 5; B['G'] = 9; B['T'] = 11$$

Sada je potrebno proći kroz  $SA_I$  od kraja prema početku te staviti indeks LMS-podniza  $P_I[SA_I[i]]$  na kraj pretinca za pripadajući sufiks i pomaknuti  $B[k]$  za jedno mjesto ulijevo. Npr. za  $i = 0$ :  $SA_I[0] = 4$ ,  $P_I[4] = 11$ ; 11 se dodaje na kraj pretinca za znak \$, a  $B['\$']$  se pomiče za jedno mjesto ulijevo, ali kako ga nema, onda postavljamo  $B['\$'] = -1$ .

Dakle,  $SA$  je:

$$SA = \{11\} \ \{-1, 7, 3\} \ \{9, 5\} \ \{-1, -1, -1, -1\} \ \{-1, -1\}$$

$$B['\$'] = -1; B['A'] = 1; B['C'] = 3; B['G'] = 9; B['T'] = 11$$

U drugome koraku algoritma dodajemo LMS-prefikse L-tipa (*induceSAI*) u  $SA$  (isti kao 2. korak na Sl. 4.17):

$$SA = \{11\} \ \{-1, 7, 3\} \ \{9, 5\} \ \{10, 6, 8, 4\} \ \{2, 1\}$$

I na kraju, u trećem koraku dodajemo LMS-prefikse S-tipa duljine veće od jedan (*induceSAs*) (isti kao 3. korak na Sl. 4.17) i dobivamo:

$$SA = \{11\} \ \{7, 3, 0\} \ \{9, 5\} \ \{10, 6, 8, 4\} \ \{2, 1\}$$

što je ujedno i konačno rješenje.

U drugom i trećem koraku dobiveni su isti rezultati kao i u 2. i 3. koraku algoritma za inducirano sortiranje LMS-podnizova (Sl. 4.17). U općenitom slučaju to ne mora biti tako.

### Vremenska složenost SA-IS algoritma

Ako je niz  $S$  duljine  $n$  izgrađen nad konstantnom ili cjelobrojnom abecedom, tada je vremenska složenost određivanja  $SA$ :  $O(n)$  (Nong *et al.*, 2009). Naime, u svakom koraku rekurzije problem se smanjuje barem za pola (jer je LMS-znakova u nizu  $S$  najviše pola od ukupnog broja znakova), tj. broj znakova koje se promatra u sljedećoj iteraciji je  $\lfloor n/2 \rfloor$ .

Dakle, ukupno vrijeme izvođenja SA-IS algoritma je  $T(n) = T(\lfloor n/2 \rfloor) + O(n) = O(n)$ .

### 4.3. Literatura

- Abouelhoda,M.I. *et al.* (2004) Replacing suffix trees with enhanced suffix arrays. *J. Discrete Algorithms*, **2**, 53–86.
- Boyer,R.S. and Moore,J.S. (1977) A fast string searching algorithm. *Commun. ACM*, **20**, 762–772.
- Bray,N. and Pachter,L. (2004) MAVID: Constrained Ancestral Alignment of Multiple Sequences. *Genome Res.*, **14**, 693–699.
- Gusfield,D. (1997) Algorithms on strings, trees, and sequences: computer science and computational biology Cambridge University Press, Cambridge [England] ; New York.
- Kärkkäinen,J. and Sanders,P. (2003) Simple Linear Work Suffix Array Construction. In, Baeten,J.C.M. *et al.* (eds), *Automata, Languages and Programming*. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 943–955.
- Knuth,D.E. *et al.* (1977) Fast Pattern Matching in Strings. *SIAM J. Comput.*, **6**, 323–350.
- Ko,P. and Aluru,S. (2003) Space efficient linear time construction of suffix arrays. In, *Journal of Discrete Algorithms*. Springer, pp. 200–210.
- Kurtz,S. (1998) Reducing the Space Requirement of Suffix Trees. *Softw. – Pract. Exp.*, **29**, 1149–1171.
- Kurtz,S. *et al.* (2004) Versatile and open software for comparing large genomes. *Genome Biol.*, **5**, R12.
- Manber,U. and Myers,G. (1990) Suffix Arrays: A New Method for On-line String Searches. In, *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '90. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp. 319–327.
- McCreight,E.M. (1976) A Space-Economical Suffix Tree Construction Algorithm. *J. ACM*, **23**, 262–272.
- Nong,G. *et al.* (2009) Linear Suffix Array Construction by Almost Pure Induced-Sorting. IEEE, pp. 193–202.
- Nong,G. *et al.* (2011) Two Efficient Algorithms for Linear Time Suffix Array Construction. *IEEE Trans. Comput.*, **60**, 1471–1484.
- Puglisi,S.J. *et al.* (2006) Suffix Arrays: What Are They Good for? In, *Proceedings of the 17th Australasian Database Conference - Volume 49*, ADC '06. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, pp. 17–18.
- Ukkonen,E. (1995) On-line construction of suffix trees. *Algorithmica*, **14**, 249–260.
- Weiner,P. (1973) Linear pattern matching algorithms. IEEE, pp. 1–11.

## 5. Poravnavanje dva ili više sljedova odjednom

### 5.1. Heuristički pristup poravnavanju para sljedova

U poglavlju 3 prikazano je poravnavanje ulaznog slijeda s bazom podataka sljedova gdje je duljina ulaznog slijeda značajno manja od duljine baze. Objasnjen je princip rada programa BLAST, kao najčešće upotrebljavanog alata. Korišteno poravnanje lokalnog je tipa: pokušavaju se poravnati dijelovi ulaznog slijeda s dijelovima sljedova iz baze podataka čije su duljine sumjerljive s duljinom ulaznog niza. Takav je pristup prikladan kada je za novi gen ili protein potrebno vidjeti postoje li homolozi u bazi podataka. Međutim, program BLAST nije učinkovit kada želimo usporediti cijele genome. Za taj se problem koriste posebni programi koji koriste heuristički pristup globalnom poravnanju para sljedova (engl. *pairwise sequence alignment*). Neki od alata koji se koriste u tu svrhu su: MegaBLAST (Zhang *et al.*, 2000) i MUMmer (Kurtz *et al.*, 2004), koji se koriste za usporedbe srodnih dugačkih sljedova, npr. za usporedbe kromosoma i genoma eukariotskih organizama.

Ovdje ćemo prikazati heuristički pristup poravnanju koji se koristi kod MUMmer-a.

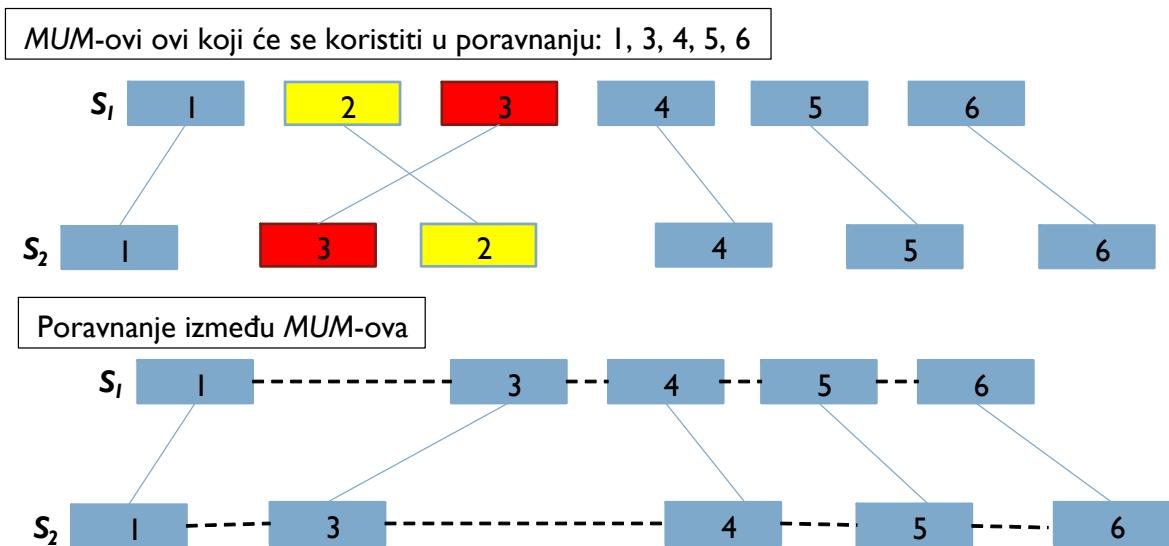
#### 5.1.1. MUMmer

Osnovna ideja koja se koristi kod MUMmer-a je pronaći dugačke podnizove nukleotida (ili aminokiselina) koji su zajednički nizovima koje uspoređujemo, a zatim ih pokušati proširiti korištenjem dinamičkog poravnanja (Kurtz *et al.*, 2004).

Dugački podniz koji je zajednički uspoređivanim ulaznim sljedovima naziva se *MUM* (engl. *Maximal Unique Match*) i to je najdulji jedinstveni podniz koji predstavlja podudaranje 2 ulazna niza. Jedinstvenost se u ovom slučaju ne odnosi na cijele dužine ulaznih nizova, nego na najdulji podniz unutar jednog područja, gdje se takav podniz ne može proširiti niti u lijevo niti u desno da bi se dobio dulji jedinstveni podniz. *MUM*-ovi se onda koriste kao ishodišne točke (engl. *anchor*) oko kojih se onda pokuša napraviti dulje lokalno poravnanje (Sl. 5.1).

#### MUMmer – algoritam

- Odrediti *MUM*-ove za dva ulazna niza (npr. genoma)  $S_1$  i  $S_2$ . Prvo se za nizove  $S_1$  i  $S_2$  izgradi poopćeno sufiksno stablo  $T$ . Zatim se obilaskom stabla  $T$  određuju *MUM*-ovi: pronalaze se unutarnji čvorovi  $v_1, v_2, \dots$  takvi da imaju 2 djeteta-lista koja od kojih jedan pripada ulaznom nizu  $S_1$ , a drugi nizu  $S_2$ . Svaki takav čvor  $v_i$  odnosi se na jedan *MUM*, jer je podniz koji vodi od korijena stabla  $T$  do čvora  $v_i$  jedinstven za nizove  $S_1$  i  $S_2$  promatrajući cijelo stablo. Takav podniz najdulji je u smislu da nema duljeg zajedničkog prefiksa sufiksa koji završavaju u listovima čvora  $v_i$ .
- Odabratи *MUM*-ove koji će biti korišteni za poravnanje. Odabiru se oni *MUM*-ovi koji sačinjavaju uzlazno rastući slijed brojeva (Sl. 5.1), što nam ujedno govori da se ti *MUM*-ovi nadovezuju jedan na drugog (engl. *Longest Increasing Subsequences*).
- Područja između tako odabranih *MUM*-ova poravnavaju se korištenjem Smith-Watermanovog algoritma (Smith & Waterman, 1981).



Sl. 5.1 Prikaz osnovne ideje programa MUMmer za poravnanje para sljedova  $S_1$  i  $S_2$ . Prvo se pronalaze *MUM*-ovi, a zatim se odabiru *MUM*-ovi koji će se koristiti u poravnanju (ovdje označeni brojevima 1, 3, 4, 5, 6). Nапослјетку, *MUM*-ovi 1, 3, 4, 5 i 6 pokušavaju se povezati tako da se područja između njih lokalno poravnaju.

## 5.2. Poravnanje više sljedova odjednom

Poravnanje više sljedova odjednom (engl. *multiple sequence alignment*; MSA) ima smisla kada su sljedovi ili neki njihovi dijelovi homologni. Potrebno je naglasiti da se ovdje polazi od prepostavke da su sljedovi koje želimo poravnati homologni, za razliku od

usporedbe ulaznog slijeda s bazom podataka, gdje se homologija između ulaznog slijeda i sljedova u bazi podataka tek određuje.

Ako su sljedovi homologni cijelom svojom dužinom i podjednake duljine, govorimo o globalnom poravnanju više sljedova odjednom (npr. poravnanje kromosoma čovjeka i čimpanze; poravnavanje genoma više sojeva neke bakterije, itd.) Ako sljedovi nisu homologni cijelom svojom dužinom, onda se mogu poravnati po dijelovima koji su ostali sačuvani (engl. *conserved regions*) i tada govorimo o lokalnom poravnanju (npr. poravnanje homolognih proteina između različitih vrsta).

Ideja poravnanja tri ili više sljedova odjednom je smještanje homolognih mesta u promatranim sljedovima u iste stupce. Formalno se to može izreći na sljedeći način: Neka je zadan skup od  $n$  sljedova:  $S_1, \dots, S_n$  nad istom abecedom (znakovi abecede mogu biti nukleotidi ili aminokiseline). Poravnanje više sljedova odjednom je skup sljedova  $S'_1, \dots, S'_n$ , gdje vrijedi:

1. Duljine poravnatih sljedova su jednake, tj.  $|S'_1| = \dots = |S'_n|$ .
2. Uklanjanjem procijepa (praznina; engl. *gap*) iz  $S'_i$  dobiva se  $S_i$  za  $i = 1, \dots, n$ .

Nakon poravnanja, modificirane ulazne sljedove  $S'_i$  predstavljamo kao matricu od  $n$  redaka i  $m$  stupaca, gdje je  $n$  broj sljedova, a  $m$  duljina svakog od sljedova nakon poravnanja, tj.  $m = |S'_i| \geq |S_i|$  za svaki  $i = 1, \dots, n$ . Duljine poravnatih sljedova mogu biti dulje od originalnih sljedova, ako su u originalne sljedove dodani procijepi. Procijepi ili praznine se dodaju ako slijed  $S_i$  u određenom stupcu (homolognom mjestu) nema odgovarajući znak (nukleotid ili aminokiselina). To može biti posljedica npr. brisanja (engl. *deletion*) tog znaka u slijedu  $S_i$ , dok je u drugim sljedovima taj znak ostao zadržan ili umetanje (engl. *insertion*) tog znaka na  $i$ -to mjesto u ostalim sljedovima, dok u  $S_i$  nije došlo do umetanja tog znaka.

### 5.2.1. Metoda sume parova

S obzirom na mnoštvo kombinacija mogućih poravnanja, konačno rješenje, odnosno najbolje od svih mogućih poravnanja određuje se prema unaprijed zadanoj mjeri. Mjeru možemo definirati na sljedeći način: neka je poravnanje predstavljeno kao matrica  $A$  dimenzija  $n \times m$ . Elementi matrice su oblika  $A_{i,j}$ , gdje  $i$  predstavlja indeks slijeda, a  $j$  indeks stupca u matrici poravnanja.

Neka je  $s(A_{k,j}, A_{b,j})$  je rezultat poravnanja (engl. *score*):

- para nukleotida  $A_{k,j}$  i  $A_{l,j}$  prema zadanom bodovanju za par nukleotida (podudaranje/nepodudaranje) ili bodovanju para nukleotid-procijep, ili
- para aminokiselina  $A_{k,j}$  i  $A_{l,j}$  prema zadanoj supstitucijskoj matrici (PAM ili BLOSUM) ili bodovanju para aminokiselina-praznina

*Metoda sume parova* (engl. *Sum-of-Pairs Scoring Method*) budiće cijelo poravnanje

$n$  sljedova tako da se prvo odrede rezultati bodovanja za svaki  $A_j$  stupac matrice,  $SP(A_j)$  (5.1):

$$SP(A_j) = \sum_{k < l} s(A_{k,j}, A_{l,j}) \quad (5.1)$$

Ukupan rezultat bodovanja cijelog poravnjanja je  $SP(A)$  (3.1):

$$SP(A) = \sum_j SP(A_j) \quad (5.2)$$

### Primjer

Neka su zadani ulazni nukleotidni sljedovi AGC, AGGG, ATGC, koje želimo poravnati. Neka je zadano sljedeće bodovanje: +1 za podudaranje nukleotida; -1 za nepodudaranje nukleotida; -2 za prazninu.

Promatramo jedno od mogućih poravnanja prikazano matricom  $A$  koja ima 3 retka i 4 stupca:

<b>A</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>
<b>1</b>	A	-	G	C
<b>2</b>	A	G	G	G
<b>3</b>	A	T	G	C

Vrijednost sume parova za poravnanje prikazano matricom  $A$  računamo na sljedeći način:

$$SP(A_1) = s(A_{1,1}, A_{2,1}) + s(A_{1,1}, A_{3,1}) + s(A_{2,1}, A_{3,1}) = 1 + 1 + 1 = 3$$

$$SP(A_2) = s(A_{1,2}, A_{2,2}) + s(A_{1,2}, A_{3,2}) + s(A_{2,2}, A_{3,2}) = -5$$

$$SP(A_3) = s(A_{1,3}, A_{2,3}) + s(A_{1,3}, A_{3,3}) + s(A_{2,3}, A_{3,3}) = 3$$

$$SP(A_4) = s(A_{1,4}, A_{2,4}) + s(A_{1,4}, A_{3,4}) + s(A_{2,4}, A_{3,4}) = -1$$

$$SP(A) = SP(A_1) + SP(A_2) + SP(A_3) + SP(A_4) = 3 - 5 + 3 - 1 = 0$$

Metoda sume parova ima nedostatak u neproporcionalno većoj težini kojom se ocjenjuju mutacije u odnosu na izostanak mutacije, što u velikoj mjeri smanjuju vrijednost konačnog zbroja. Složenost operacije sume parova za jedan stupac matrice poravnjanja je  $O(k^2)$ , gdje je  $k$  broj sljedova u matrici.

### Primjer

Promatramo poravnanje 4 nukleotidna slijeda. Neka su vrijednosti sljedova u  $i$ -tom stupcu: C, C, G i C. Dakle, jedino se nukleotid G u 3. slijedu razlikuje od ostalih. Neka je zadano bodovanje: +1 za podudaranje nukleotida; -1 za nepodudaranje nukleotida. Suma poravnjanja za stupac  $A_i$  je:

$$SP(A_i) = 1 - 1 + 1 - 1 + 1 - 1 = 0$$

Za usporedbu, ako promatramo 4 nukleotidna slijeda za koje je isti nukleotid C u  $i$ -tom stupcu u svim sljedovima, onda je suma poravnjanja za  $i$ -ti stupac  $A_i$ :

$$SP(A_i) = 1 + 1 + 1 + 1 + 1 + 1 = 6$$

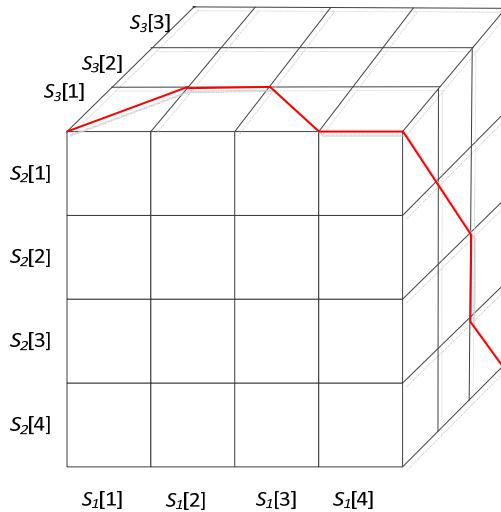
## 5.2.2. Poravnanje više sljedova odjednom dinamičkim programiranjem

Globalno optimalno poravnanje dva slijeda dinamičkim programiranjem opisano je u poglavlju 2.3. Needleman-Wunschov algoritam (Needleman & Wunsch, 1970) uspoređuje svaki znak iz jednog slijeda duljine  $m$  sa svakim znakom iz drugog slijeda duljine  $n$ . Bodovanje poravnjanja spremi se u dvodimenzionalno polje (matricu) dimenzija  $m \times n$ . Element matrice u donjem desnom kutu sadrži konačnu vrijednost poravnjanja, a samo poravnanje određuje se kao put od elementa u donjem desnom kutu matrice prema elementu u gornjem lijevom kutu rekurzivno slijedeći put najboljeg rezultata. Optimalno poravnanje obično se nalazi oko dijagonale matrice.

Globalno optimalno poravnanje  $k$  sljedova je poopćenje Needleman-Wunschovoga algoritma za dva slijeda. Vrijednosti poravnjanja za neku kombinaciju znakova  $k$  sljedova spremaju se u  $k$ -dimenzionalno polje, tj. poliedar (engl. *polyhedron*). Kod poravnjanja dva nukleotidna niza cijena koraka poravnjanja određena je cijenom zamjene, umetanja i brisanja, a kod aminokiselinskih sljedova vrijednostima matrice zamjene. Korak

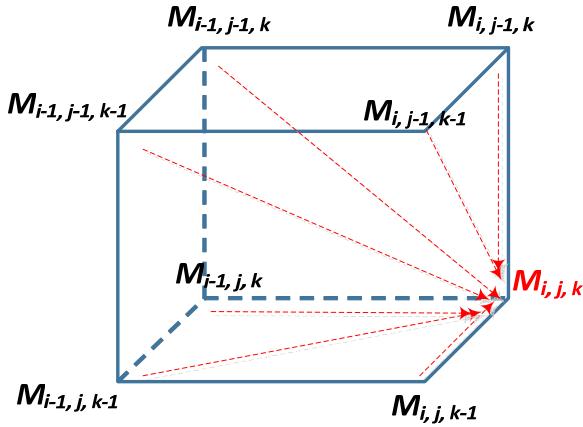
poravnjanja  $k$  sljedova uključuje  $2^k - 1$  kombinacija i svaka od njih evaluira se metodom sume parova (pojašnjeno kasnije u tekstu). Optimalno rješenje nalazi se oko  $(k - 1)$ -dimenzionalnog poliedra (*dijagonale*) unutar  $k$ -dimenzionalnog poliedra (Sl. 5.2). Kako bi se ubrzalo traženje optimalnog puta kroz poliedar, pretraživanje se može koncentrirati samo na područje oko dijagonale, odnosno na vrijednosti u području tzv. Carrillo-Lipmanove granice (engl. *Carrillo-Lipman Bound*) (Carrillo & Lipman, 1988).

Razmotrimo slučaj optimalnog poravnjanja 3 slijeda:  $S_1$ ,  $S_2$  i  $S_3$ , gdje su duljine sljedova redom 4, 4 i 3. Računanje poravnjanja za 3 slijeda pohranjujemo u trodimenzionalno polje  $M$  (Sl. 5.2). Elementi polja označeni su s  $M_{i,j,k}$  gdje su  $i$ ,  $j$  i  $k$  indeksi nizova  $S_1$ ,  $S_2$  i  $S_3$ . Izračun elemenata polja  $M$  počinje od elementa  $M_{1,1,1}$  i ide prema zadnjem elementu poliedra –  $M_{4,4,3}$  (za primjer na Sl. 5.2).



Sl. 5.2 Računanje poravnjanja tri slijeda  $S_1$ ,  $S_2$ ,  $S_3$  odjednom. Rješenje se obično nalazi oko dijagonale trodimenzionalnog polja (izlomljena crta označena crvenom bojom).

Vrijednosti elemenata  $M_{i,j,k}$ , računaju se kao i u dvodimenzionalnom Needleman-Wunschovom slučaju – potrebno je izračunati cijene mogućih koraka poravnjanja i odabrati korak koji maksimizira vrijednost elementa  $M_{i,j,k}$ .



Sl. 5.3 Računanje elementa  $M_{i,j,k}$  kod poravnanja tri slijeda  $S_1, S_2, S_3$ .

Konkretno, element  $M_{i,j,k}$  matrice (Sl. 5.3) računa se prema izrazu (5.3):

$$M_{i,j,k} = \max \begin{cases} M_{i-1,j,k} + S(S_1[i], -, -) \\ M_{i,j-1,k} + S(-, S_2[j], -) \\ M_{i,j,k-1} + S(-, -, S_3[k]) \\ M_{i-1,j-1,k} + S(S_1[i], S_2[j], -) \\ M_{i-1,j,k-1} + S(S_1[i], -, S_3[k]) \\ M_{i,j-1,k-1} + S(-, S_2[j], S_3[k]) \\ M_{i-1,j-1,k-1} + S(S_1[i], S_2[j], S_3[k]) \end{cases} \quad (5.3)$$

gdje je  $S$  oznaka prethodno opisane sume parova. Za određivanje elementa  $M_{i,j,k}$  potrebno je ispitati sedam kombinacija. Nije dopušteno (s obzirom da nema smisla) da su u nekom stupcu poravnanja sve praznine te je zato ukupan broj kombinacija  $2 \cdot 2 \cdot 2 - 1 = 7$ , a među sedam kombinacija odabire se kombinacija koja daje maksimalan rezultat.

Općenito, ako računamo poravnanje  $k$  sljedova, gdje je svaki slijed duljine  $n$ , vrijednost poravnanja pohranjuje se u  $k$ -dimenzionalno polje  $M$ . Dakle,  $M$  se sastoji od  $n^k$  elemenata. Svaki element  $M_{i_1, i_2, \dots, i_k}$  računa se prema izrazu (5.4), gdje se indeksi  $i_1, i_2, \dots, i_k$  odnose na indekse promatranih znakova u sljedovima  $S_1, S_2, \dots, S_k$ , tj. na znakove  $S_1[i_1], S_2[i_2], \dots, S_k[i_k]$ . Za slučaj poravnanja 3 niza u svakom je koraku (tj. za svaki element iz  $M$ ) bilo potrebno razmotriti  $2^3 - 1$  kombinacija. U slučaju  $k$  sljedova potrebno je razmotriti  $2^k - 1$  kombinacija (zbrojeva) za svaki element iz  $M$  (za svaki od sljedova na promatranom mjestu može biti ili nukleotid/aminokiselina ili praznina). Ako se svaki zbroj  $S$  računa metodom sume parova (poglavlje 5.2.1), vrijeme potrebno za računanje jednog zbroja je  $O(k^2)$ , pa je ukupno vrijeme potrebno za računanje jednog elementa matrice  $M$ :  $O(k^2 2^k)$ .

$$M_{i_1, i_2, \dots, i_k} = \max \begin{cases} M_{i_1-1, i_2-1, \dots, i_k-1} + S(S_1[i_1], S_2[i_2], \dots, S_k[i_k]) \\ M_{i_1, i_2-1, \dots, i_k-1} + S(-, S_2[i_2], \dots, S_k[i_k]) \\ \dots \\ M_{i_1, i_2, \dots, i_k-1} + S(-, -, \dots, S_k[i_k]) \end{cases} \quad (5.4)$$

Vremenska složenost izračuna  $k$ -dimenzionalnog polja  $M$  za  $k$  sljedova duljine  $n$  iznosi  $O(k^2 2^k n^k)$ , gdje je  $n^k$  broj elemenata u  $k$ -dimenzionalnom polju  $M$ , a  $O(k^2 2^k)$  je vrijeme potrebno za računanje jednog elementa polja  $M$ . Prostora složenost je  $O(n^k)$ , zbog prostora potrebnog za pohranu  $k$ -dimenzionalnog polja  $M$ .

Zbog vremenske i prostorne složenosti koje eksponencijalno ovise o broju sljedova ( $k$ ), dinamičko programiranje optimalnog poravnanja tri ili više sljedova vrlo je zahtjevno i u praksi se ne primjenjuje. Kao alternativa, koristi se heurističko poravnavanje više sljedova odjednom.

### 5.2.3. Heuristički pristup poravnanju više sljedova odjednom

Nekoliko je osnovnih heurističkih pristupa poravnanju više sljedova odjednom:

1. progresivno poravnavanje (engl. *progressive alignment*)
2. iterativne metode (engl. *iterative methods*)
3. metode temeljene na skrivenim Markovljevim modelima (engl. *Hidden Markov Model; HMM*)
4. metode temeljene na genetskim algoritmima
5. metode temeljene na filogenetskom predznanju (engl. *phylogeny-aware methods*)

Ovdje ćemo ukratko objasniti dvije metode: progresivnu i iterativnu. Kao primjer progresivne metode objasnit ćemo način rada popularnog alata za poravnavanje više sljedova odjednom: ClustalW/ClustalX (Thompson *et al.*, 1994; Larkin *et al.*, 2007) te njegove višeprocesorske verzije, Clustal Omega (Sievers *et al.*, 2011). Kao primjer korištenja iterativne metode ukratko ćemo objasniti princip rada programa MUSCLE (Edgar, 2004).

Clustal skupina programa kao ulaz prima skup nukleotidnih ili aminokiselinskih sljedova. Sljedovi se inicijalno poravnavaju u parovima. Zatim se prema inicijalnim poravnanjima odrede udaljenosti na temelju kojih se onda gradi pomoćno filogenetsko stablo (engl. *guide tree*). Stablo se gradi korištenjem metode najbližih susjeda (engl. *neighbor joining*; potpoglavlje 0). Prema stablu se odabiru najsličniji sljedovi i oni se prvi dodaju u konačno

poravnanje. Zatim se postupak ponavlja tako da se jedan po jedan preostali sljedovi dodaju u konačno poravnanje. Postupak završava kada su svi sljedovi dodani u poravnanje. Konačno se poravnanje može dodatno ručno prilagoditi (Sl. 5.4). Nedostatak metode je što konačni rezultat ovisi o paru sljedova koji su prvi odabrani, a što se naknadno ne može mijenjati.

#### Algoritam ClustalW

Inicijalno poravnati sve parove sljedova

Odabratи prvi par sljedova

**Ponavljati** dok se ne dodaju svi preostali sljedovi

Odabratи sljedeći slijed ili grupu sljedova koji će se dodati prethodno poravnatim sljedovima

Poravnati novododani slijed ili grupu s prethodno poravnatim sljedovima

Ručno prilagoditi izračunata poravnanja

Sl. 5.4 Princip rada programa ClustalW/ClustalX

#### Primjer

Zadani su sljedovi  $S_1 = \text{AGC}$ ,  $S_2 = \text{ATGC}$ ,  $S_3 = \text{AGGG}$ . Prvo se naprave inicijalna poravnanja parova sljedova. Rezultat poravnanja para sljedova npr. može biti:

1. Mjera sličnosti (engl. *similarity measure*) između sljedova, definirana kao broj podudaranja u poravnanju podijeljeno s ukupnim brojem stupaca koji ne sadrže niti jednu prazninu
2. Mjera udaljenosti (engl. *distance measure*) između sljedova, definirana kao broj nepodudaranja u poravnanju podijeljeno s ukupnim brojem stupaca koji ne sadrže niti jednu prazninu

U ovom ćemo primjeru računati rezultat poravnanja kao mjeru sličnosti sljedova (Sl. 5.5).

Vidimo da između 3 moguća poravnanja, najveću sličnost ima par  $(S_1, S_2)$ , čija je sličnost 1. S obzirom da su ta dva slijeda međusobno najsličnija, njih prvo grupiramo, a na kraju dodajemo  $S_3$ .

Rezultat( $S_1, S_2$ ) = 1

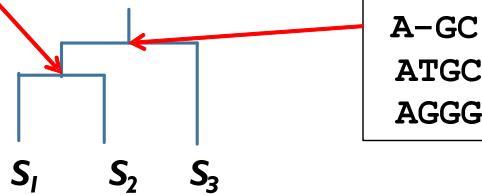
$S_1$	A	-	G	C
$S_2$	A	T	G	C

Rezultat( $S_1, S_3$ ) = 2/3

$S_1$	A	G	C	-
$S_3$	A	G	G	G

Rezultat( $S_2, S_3$ ) = 2/4

$S_2$	A	T	G	C
$S_3$	A	G	G	G



Sl. 5.5 Odredivanje poravnanja sljedova  $S_1, S_2$  i  $S_3$ .

### 5.3. Literatura

- Carrillo,H. and Lipman,D. (1988) The Multiple Sequence Alignment Problem in Biology. *SIAM J Appl Math*, **48**, 1073–1082.
- Edgar,R.C. (2004) MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Res.*, **32**, 1792–1797.
- Kurtz,S. *et al.* (2004) Versatile and open software for comparing large genomes. *Genome Biol.*, **5**, R12.
- Larkin,M.A. *et al.* (2007) Clustal W and Clustal X version 2.0. *Bioinforma. Oxf. Engl.*, **23**, 2947–2948.
- Needleman,S.B. and Wunsch,C.D. (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*, **48**, 443–453.
- Sievers,F. *et al.* (2011) Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega. *Mol. Syst. Biol.*, **7**.
- Smith,T.F. and Waterman,M.S. (1981) Identification of common molecular subsequences. *J. Mol. Biol.*, **147**, 195–197.
- Thompson,J.D. *et al.* (1994) CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res.*, **22**, 4673–4680.
- Zhang,Z. *et al.* (2000) A greedy algorithm for aligning DNA sequences. *J. Comput. Biol. J. Comput. Mol. Cell Biol.*, **7**, 203–214.

## 6. Samostojni indeksi

Indeks je struktura podataka koja omogućuje učinkovit dohvati i pretraživanje podataka. *Potpuni indeks* (engl. *full-text index*) je indeks koji omogućuje dohvati ili pretraživanje cijelog teksta ili bilo kojeg njegovog dijela; takvi indeksi su npr. sufiksno stablo i sufiksno polje.

Neka je zadan niz  $S$  izgrađen nad abecedom  $\Sigma$  i duljine  $n$ . Za pohranu niza  $S$  potrebno je  $O(n \log |\Sigma|)$  bita, gdje je  $n$  broj znakova niza, a  $\log|\Sigma|$  minimalan broj bitova potrebnih za pohranu znakova iz abecede  $\Sigma$  (ako sve znakove iz  $\Sigma$  kodiramo s jednakim brojem bitova). Na primjer, za abecedu koja se sastoji od samo 4 znaka, dovoljno je  $\log_2 4 = 2$  bita za jedinstveni prikaz svakog od znakova. Za pohranu potpunog indeksa izgrađenog nad nizom  $S$  potrebno je  $\Theta(n \log n)$  bita, gdje je  $\log n$  minimalan broj bitova potreban za pohranu jednog broja  $\leq n$ .

Međutim, u praksi je za pohranu niza  $S$  obično potrebno  $n$  okteta (engl. *byte*). Za pohranu potpunog indeksa potrebno je od  $4n$  okteta (u slučaju sufiksнog polja) do barem  $10n$  okteta u slučaju sufiksнog stabla (Kurtz, 1998b).

Kao memorijski učinkovitija alternativa, 2000. godine se pojavio prvi *samostojni indeks* (engl. *self-index*) (Ferragina & Manzini, 2000), čije je memorijsko zauzeće proporcionalno veličini komprimiranog teksta, odnosno sublinearno u odnosu na originalni tekst. Takav indeks zamjenjuje tekst, tj. sam indeks omogućuje pristup tekstu ili dijelovima teksta (podnizovima) nad kojima je indeks izgrađen.

Prvi takav indeks bio je FM-indeks (Ferragina & Manzini, 2000), koji se temelji na Burrows-Wheelerovoј transformaciji (BWT) ulaznog niza (Burrows & Wheeler, 1994). Osim samostojnih indeksa koji se temelje na BWT, postoji grupa samostojnih indeksa koja koristi komprimirana sufiksna polja, npr. (Sadakane, 2003) te indeksi koji koriste LZ-sažimanje (Ziv & Lempel, 1977, 1978), npr. (Navarro, 2009). Iako samostojni indeksi zauzimaju znatno manje memorijskog prostora u odnosu na potpune indekse, za njihovu izgradnju potrebno je  $5n$ - $9n$  okteta za ulazni niz duljine  $n$  (Ferragina *et al.*, 2008).

U ovom će poglavlju biti izložen FM-indeks, koji se najviše primjenjuje u bioinformatici. S obzirom da je memorijski prostor, koji je potreban za pohranu tog indeksa, ovisan o  $k$ -toj entropiji ulaznog teksta, najprije ćemo ukratko objasniti kako se računa entropija

znakovnog niza (6.1), a zatim i Burrows-Wheelerovu transformaciju (6.2) te na kraju sam FM-indeks.

## 6.1. Određivanje entropije znakovnog niza

### 6.1.1. Shannonova entropija

Shannon je 1948. godine uveo koncept statističke entropije u teoriju informacija (Shannon, 1948). Entropija u teoriji informacija je *mjera količine informacije* sadržane u određenoj poruci, a ovisi o razdiobi vjerojatnosti pojedinih simbola u abecedi poruke. Ukoliko izvor informacije ima abecedu od samo jednog slova (čime je vjerojatnost odašiljanja tog slova 1), poruke takvog izvora ne nude nikada novu informaciju; njihov informacijski sadržaj ili entropija je nula. Najgušće pakirane informacije (one s najvećom entropijom) imaju izvori čija je vjerojatnost odašiljanja svih simbola abecede podjednaka.

Slova abecede možemo promatrati kao ishode različitih događaja. Neka je  $p_i$  vjerojatnost  $i$ -tog ishoda. Tada je Shannonova entropija  $H$  definirana kao:

$$H = -\sum_i p_i \log(p_i) \quad (6.1)$$

#### Primjer:

Promatramo bacanje kocke, gdje kocka može pasti na jednu od 6 strana označenih brojevima: 1, 2, 3, 4, 5 i 6. Dakle, bacanje kocke može imati jedan od 6 mogućih ishoda. Zbroj vjerojatnosti svih ishoda je:

$$P(X = 1) + P(X = 2) + P(X = 3) + P(X = 4) + P(X = 5) + P(X = 6) = 1$$

Ako je kocka nepristrana, onda je vjerojatnost da kocka padne na bilo koju stranu jednaka, odnosno,

$$P(X = 1) = P(X = 2) = P(X = 3) = P(X = 4) = P(X = 5) = P(X = 6) = \frac{1}{6}$$

Entropija u tom slučaju iznosi:

$$H = -\sum_i p_i \log(p_i) = -6 \cdot \frac{1}{6} \cdot \log_2 \frac{1}{6} = 2.58$$

Za bazu logaritma obično koristimo 2, jer time dobivamo informacijski sadržaj u bitovima, ali je moguće koristiti i druge vrijednosti.

Ako je kocka pristrana, npr. kocka uvijek pada na stranu označenu sa 6, onda je:

$$P(X = 6) = 1, \quad P(X \neq 6) = 0$$

Također, pretpostavljamo da vrijedi  $0 \cdot \log 0 = 0$ , s obzirom da nema smisla da ishodi s vjerojatnošću nula doprinose ukupnom informacijskom sadržaju.

Entropija je za primjer pristrane kocke jednaka:

$$H = -1 \cdot \log_2 1 - 5 \cdot \log_2 0 = 0$$

Dakle, u slučaju nepristrane kocke entropija je maksimalna (nije moguće predvidjeti ishod sljedećeg bacanja), odnosno, u slučaju pristrane kocke koja uvijek pada na stranicu označenu sa 6, entropija je minimalna (uvijek je moguće predvidjeti ishod sljedećeg bacanja).

### 6.1.2. Shannonova entropija znakovnog niza

Neka je zadan niz znakova  $S$ , duljine  $n$ . Neka je  $p_i$  vjerojatnost pojavljivanja  $i$ -tog znaka u  $S$ , gdje se  $p_i$  određuje kao učestalost pojavljivanja tog znaka u  $S$ , tj.  $p_i = n_i / n$ . Shannonova entropija niza  $S$  definirana je tada kao:

$$H(S) = - \sum_i p_i \log(p_i) = - \sum_i \frac{n_i}{n} \log \frac{n_i}{n}, \quad (6.2)$$

Ovako definirana entropija se još zove *nulta entropija* niza  $S$  ili  $H_0(S)$ .

**Primjer:**

Neka su zadani nizovi  $S_1 = \text{ACCA}$  ( $n = 4$ ,  $n_A = n_C = 2$ ) i  $S_2 = \text{ACCC}$  ( $n = 4$ ,  $n_A = 1$ ,  $n_C = 3$ ). Potrebno je odrediti nultu entropiju nizova  $S_1$  i  $S_2$  ( $\log$  je  $\log_2$ ):

$$H_0(S_1) = - \left( \frac{n_A}{n} \log \frac{n_A}{n} + \frac{n_C}{n} \log \frac{n_C}{n} \right) = - \left( \frac{2}{4} \log \frac{2}{4} + \frac{2}{4} \log \frac{2}{4} \right) = 1$$

$$H_0(S_2) = - \left( \frac{n_A}{n} \log \frac{n_A}{n} + \frac{n_C}{n} \log \frac{n_C}{n} \right) = - \left( \frac{1}{4} \log \frac{1}{4} + \frac{3}{4} \log \frac{3}{4} \right) = 0.811$$

Rezultat možemo tumačiti na sljedeći način: niz  $S_1$  ima veću entropiju od niza  $S_2$ , odnosno teže je predvidjeti sljedeći znak u nizu u slučaju niza  $S_1$ .

Za niz koji bi se sastojao od samo jednog znaka, uvijek bi bilo moguće predvidjeti sljedeći znak, tj. entropija takvog niza bila bi 0. Takav bi se niz mogao vrlo lako sažeti (komprimirati): npr. cijeli niz bi se mogao komprimirati tako da se pohrani samo znak koji se ponavlja i broj ponavljanja znaka u nizu. Suprotno, što je niz složeniji, tj. ako se sastoji od više znakova abecede koji su slučajno poredani, to je takav niz teže sažeti.

### 6.1.3. Entropija višeg reda

Osim nulte entropije, za neki se niz mogu definirati i entropije višeg, odnosno,  $k$ -tog reda,  $H_k(S)$ ,  $k > 0$ . Nadovezujući se na prethodni primjer, ovdje ćemo ukratko izložiti vezu između entropije i sažimanja (engl. *compression*) teksta te ulogu entropije višeg reda u sažimanju.

Naime, svaki je znak u računalu pohranjen kao niz bitova. Za pohranu znakova neke abecede  $\Sigma$ , želimo osigurati da je svaki znak pohranjen jedinstvenom kombinacijom 0 i 1 (engl. *codeword*). Ako je svaki znak pohranjen s jednakim brojem bitova, onda je za pohranu svakog znaka potrebno  $\lceil \log_2 |\Sigma| \rceil$  bitova. Općenito, veličina idealno komprimiranog niza je  $nH_0(S)$ , gdje se za pohranu svakog znaka koristi  $-\log(n_i/n)$  bitova (Manzini, 2001). Dakle,  $H_0(S)$  može predstavljati prvu aproksimaciju kod sažimanja teksta.

Međutim, ako se uoče neke pravilnosti u tekstu, znakove je moguće pohraniti i korištenjem manjeg broja bitova. Takvo sažimanje teksta odnosi se na entropije višeg reda,  $H_k$ ,  $k > 0$ .

Neka je  $\Sigma^k$  je skup svih nizova duljine  $k$  čiji su znakovi iz  $\Sigma$ . Promatramo kontekst (engl. *context*)  $con$ , koji je podniz niza  $S$  duljine  $k$ . Neka je  $S^{con}$  niz konateniranih znakova koji se pojavljuju u  $S$  iza  $con$  gledano s lijeva na desno. Tada definiramo  $k$ -ti red entropije niza  $S$  (engl. *the  $k$ -th order entropy of  $S$* ),  $H_k(S)$ ,  $k \geq 0$ :

$$H_k(S) = \frac{1}{n} \sum_{con \in \Sigma^k} |S^{con}| H_0(S^{con}) \quad (6.3)$$

Možemo uočiti da je:  $0 \leq H_k(S) \leq H_{k-1}(S) \leq \dots \leq H_1(S) \leq H_0(S) \leq \log|\Sigma|$ .

#### Primjer:

Neka je zadan niz  $S = \text{ACCA}$ . Promatramo  $con_1 = \text{A}$  i  $con_2 = \text{C}$  (podnizovi niza  $S$  duljine 1). Tada je niz  $S^{con_1}$  konatenacija znakova koji slijede iza podniza  $con_1 = \text{A}$  u  $S$ :  $S^{con_1} = \text{C}$ . Niz  $S^{con_2}$  koji je konatenacija znakova koji slijede iza podniza  $con_2$ :  $S^{con_2} = \text{AC}$ . Koristimo (6.2) i (6.3):

$$H_0(S) = -\left(\frac{n_A}{n} \log \frac{n_A}{n} + \frac{n_C}{n} \log \frac{n_C}{n}\right) = -\left(\frac{2}{4} \log \frac{2}{4} + \frac{2}{4} \log \frac{2}{4}\right) = 1$$

$$H_0(S^{con1}) = -\left(\frac{1}{1} \log \frac{1}{1}\right) = 0$$

$$H_0(S^{con2}) = -2\left(\frac{1}{2} \log \frac{1}{2}\right) = 1$$

$$H_I(S) = \frac{1}{4}(|S^{con1}|H_0(S^{con1}) + |S^{con2}|H_0(S^{con2})) = 0.5$$

Prema očekivanju, vidimo da je entropija višeg reda  $H_I(S)$  manja od  $H_0(S)$ . Također  $H_0(S^{con2})$  je 1, a  $H_0(S^{con1}) = 0$ , jer se samo znak A pojavljuje iza *con1*, a iza *con2* mogu se pojaviti dva znaka: A ili C, pa je u tom slučaju entropija veća.

## 6.2. Burrows-Wheelerova transformacija (BWT)

Burrows i Wheeler su 1994. osmislili transformaciju teksta (Burrows & Wheeler, 1994), koja je vrlo prikladna za sažimanje teksta. Među ostalim, koristi se npr. u programu bzip2 (Seward, 2007). Osim pod nazivom Burrows-Wheelerova transformacija (kraće, BWT), poznata je i pod nazivom *block-sorting compression*.

Neka je zadan niz  $S$ . Niz  $S' = \text{BWT}(S)$  može se odrediti na dva načina: (i) rotacijom niza  $S$ , ili (ii) sufiksnog polja  $\text{SA}(S)$ .

### 6.2.1. Konstrukcija BWT rotacijom znakovnog niza

Neka je zadan niz  $S$  duljine  $n$  nad abecedom  $\Sigma$ . Na kraju niza  $S$  nalazi se jedinstveni znak koji predstavlja oznaku kraja niza,  $\$$ . Prepostavljamo da je  $\$$  leksikografski manji od svih ostalih znakova iz  $\Sigma$ . Prvi znak niza  $S$  nalazi se na poziciji 0, a zadnji na poziciji  $n - 1$ .

Da bi se izgradio niz  $B = \text{BWT}(S)$ , potrebno je obaviti sljedeće korake (Tablica 6.1):

1. Napraviti sve cikličke rotacije  $CR$  (permutacije) niza  $S$ , tj. formirati nizove oblika

$$CR(i) = S[i, n - 1]S[0, i - 1] \text{ za } i \geq 1 \text{ i } CR(0) = S:$$

$$CR(0) = S, CR(1) = S[1, n - 1]S[0, 0], \text{ itd.}$$

2. Sve cikličke rotacije potrebno je leksikografski poredati u tablici.
3. Zadnji stupac znakova u tablici leksikografski poredanih cikličkih rotacija predstavlja  $B = \text{BWT}(S)$ .

**Primjer:**

Neka je zadan niz  $S = \text{ACCA\$}$  ( $n = 5$ ). Prepostavljamo da je znak leksikografski  $\$$  manji od svih znakova abecede nad kojom je izgrađen  $S$ . Potrebno je odrediti  $B = \text{BWT}(S)$ .

Prvo određujemo cikličke rotacije ( $CR$ ) niza  $S$  (drugi stupac Tablica 6.1). Npr. ciklička rotacija  $CR(0) = \text{ACCA\$}$ ;  $CR(1) = \text{CCA\$A}$ , jer je  $CR(1) = S[1, 4]S[0, 0]$ , itd.

Zatim se cikličke rotacije niza  $S$  poredaju abecedno (treći stupac, Tablica 6.1). Naposljetku se konkateniraju zadnji znakovi abecedno poredanih cikličkih rotacija (četvrti stupac Tablica 6.1) niza  $S$ , čim se dobije niz  $B = \text{BWT}(S) = \text{AC\$CA}$ .

Tablica 6.1 Izgradnja niza  $\text{BWT}(S)$  za niz  $S = \text{ACCA\$}$  cikličkim rotacijama.

$i$	Cikličke rotacije ( $CR$ ) niza $S$	Abecedno poredane cikličke rotacije niza $S$	$B[i]$
0	ACCA\$	\$ACCA	A
1	CCA\$A	A\$ACC	C
2	CA\$AC	ACCA\$	\$
3	A\$ACC	CA\$AC	C
4	\$ACCA	CCA\$A	A

### 6.2.2. Konstrukcija BWT korištenjem sufiksnog polja

Neka je izgrađeno sufiksno polje  $\text{SA}$  nad nizom  $S$ . Tada niz  $B = \text{BWT}(S)$  definiramo na sljedeći način:

1.  $B[i] = \$$ , ako je  $\text{SA}[i] = 0$
2.  $B[i] = S[\text{SA}[i] - 1]$ , inače.

**Primjer:**

Zadan je niz  $S = \text{ACCA\$}$  ( $n = 5$ ). Prepostavljamo da je znak  $\$$  leksikografski manji od znakova abecede nad kojom je izgrađen  $S$ . Određujemo  $B = \text{BWT}(S)$  korištenjem sufiksnog polja  $\text{SA}$  izgrađenog nad  $S$ .

Prvo smo abecedno poredali sufikse od  $S$  (drugi stupac Tablica 6.2). Npr., abecedno je najmanji sufiks  $s_4 = \$$ , zatim slijedi  $s_0 = \text{ACCA\$}$ , itd. Sufiksno polje  $\text{SA} = \{4, 3, 0, 2, 1\}$  (treći stupac, Tablica 6.2). Na kraju, korištenjem pravila da je  $B[i] = \$$ , ako je  $\text{SA}[i] = 0$ , a inače  $B[i] = S[\text{SA}[i] - 1]$ , dobijemo  $B = \text{BWT}(S) = \text{AC\$CA}$  (četvrti stupac, Tablica 6.2).

Tablica 6.2 Izgradnja niza  $B = \text{BWT}(S)$  korištenjem sufiksnog polja  $\text{SA}$  od  $S$ .

$i$	Abecedno poredani sufiksi od $S$	$\text{SA}[i]$	$\text{BWT}[i]$
0	$\$$	4	A
1	A\\$	3	C
2	ACCA\$	0	\$
3	CA\$	2	C
4	CCA\$	1	A

Lako je uočiti vezu između postupka izgradnje niza  $B = \text{BTW}(S)$  korištenjem sufiksnog polja i cikličkih rotacija: svaka abecedno poredana ciklička rotacija (treći stupac, Tablica 6.1) do uključivo znaka  $\$$  odgovara sufiksu niza  $S$  na istoj poziciji (drugi stupac, Tablica 6.2).

### 6.2.3. Reverzibilnost BWT-a

Važno svojstvo BWT transformacije je njezina reverzibilnost, koja će biti pojašnjena u ovom poglavlju.

Nakon što smo nad nizom  $S$  obavili BWT transformaciju, zadnji stupac tablice sortiranih rotacija je  $B = \text{BWT}(S)$  (Tablica 6.3). Zadnji stupac, kao i ostali stupci ove tablice, sadrži sve znakove izvornog niza  $S$ . Također je važno uočiti da prvi stupac sortiranih rotacija niza  $S$  uvijek sadrži leksikografski sortirane znakove niza  $S$ , što je ujedno i niz dobivenim leksikografskim sortiranjem niza  $B = \text{BWT}(S)$  (nazvat ćemo ga  $B_{sort}$ ).

## Primjer

Neka je zadan niz  $S = \text{ACCA\$}$ . Tada je  $B = \text{BWT}(S) = \text{AC\$CA}$ , a sortiranjem znakova iz  $B$  (ili  $S$ ) dobivamo niz  $B_{\text{sort}} = \$\text{AACC}$  (Tablica 6.3).

Tablica 6.3 Zadnji i prvi stupac abecedno poredanih cikličkih rotacija za  $S = \text{ACCA\$}$ :  $B$  i  $B_{\text{sort}}$

$i$	Abecedno poredane cikličke rotacije (CR) niza $S$	$B[i]$	$B_{\text{sort}}[i]$
0	\$ACCA $\textcolor{red}{A}$	A	\$
1	A\$ACC $\textcolor{red}{C}$	C	A
2	ACCA\$ $\textcolor{red}{\$}$	\$	A
3	CA\$A $\textcolor{red}{C}$	C	C
4	CCA\$ $\textcolor{red}{A}$	A	C

Promatramo zadnji i prvi stupac abecedno poredanih cikličkih rotacija od  $S$ , tj. nizove  $B$  i  $B_{\text{sort}}$  (Tablica 6.3). Možemo uočiti sljedeće (jednostavno se vidi kada se stupac  $B_{\text{sort}}$  stavi zdesna stupcu  $B$ ):

1. U nizu  $S$  iza znaka A mogu doći znakovi \$ ili C
2. U nizu  $S$  iza znaka C mogu doći znakovi A ili C

Kako bi rekonstruirali  $S$  iz  $B$ , morali bi imati jednoznačnu informaciju o znaku koji slijedi iza promatranog znaka iz  $B$ . Na primjer, ne možemo jednoznačno odrediti koji znak dolazi iza  $B[0] = \text{A}$ : je li to \$ ili C? Da bi odgovorili na to pitanje, potrebno je prepoznati o kojem je slovu A iz niza S riječ. Zato ćemo ulazni niz napisati u sljedećem obliku:  $A_1C_1C_2A_2\$_1$ . Iz ovoga je vidljivo da iza prvoga slova A u nizu S (označenog s  $A_1$ ) slijedi slovo C (preciznije  $C_1$ ), a iza drugoga slova A u nizu S (označenog s  $A_2$ ) slijedi \$ (preciznije  $\$_1$ ). Dakle, ako u nizu  $B$  možemo prepoznati o kojem je slovu A riječ ( $A_1$  ili  $A_2$ ), onda ćemo znati i koji mu znak slijedi ( $C_1$  ili  $\$_1$ ).

Sljedeći primjer će nam ilustrirati povezanost između rednog broja pojavljivanja pojedinog znaka u nizovima  $S$  i  $B$ , odnosno,  $i$ -to pojavljivanje znaka  $c$  iz  $S$  odgovara  $i$ -tom pojavljivanju znaka  $c$  u nizu  $B$ .

Neka skup ciklički rotiranih sljedova sadrži sljedeće nizove:

$A_1BXXXXXX$

$A_2CXXXXXX$

$A_3DXXXXXX$

...

$BXXXXXXA_1$

...

$CXXXXXXA_2$

...

$DXXXXXXA_3$

Uočimo da su tri rotacije niza koje počinju istim slovom (slovom A) sortirane po drugom slovu niza, tj. redoslijed im je utvrđen slovima B, C i D. Cikličke rotacije koje počinju slovom A ( $A_1BXXXXXX$ ,  $A_2CXXXXXX$  i  $A_3DXXXXXX$ ) imaju različito indeksirana početna slova A ( $A_1$ ,  $A_2$  i  $A_3$ ), koja označavaju redni broj pojavljivanja početnih slova A u cikličkim rotacijama.

Kada se te rotacije ( $A_1BXXXXXX$ ,  $A_2CXXXXXX$  i  $A_3DXXXXXX$ ) ciklički rotiraju za jedno mjesto, sva tri A će se naći u zadnjem stupcu, dok će druga slova tih rotacija biti u prvom stupcu i određivati leksikografski poredak triju nizova. Uočimo da će upravo zato redoslijed slova A biti isti i u zadnjem i u prvom stupcu (tj. rotacija koja počinje slovom  $A_1$  u oba je stupca ispred rotacija koje počinju slovima  $A_2$  i  $A_3$ ). Ovo svojstvo očuvanosti porekla istih znakova u prvom i zadnjem stupcu BWT-a naziva se **LF-mapiranje** (engl. *LF-mapping*; *Last-to-Front mapping*; *Last-to-First mapping*). To svojstvo nam daje mogućnost rekonstrukcije niza  $S$  iz  $B$ , jer znamo koji od istih znakova trebamo odabrati (rekonstrukcija niza  $S$  iz  $B$  prikazana je na Sl. 6.1).

Formalnije rečeno, LF-mapiranje opisuje vezu, odnosno mapiranje, zadnjeg i prvog stupca u listi leksikografski sortiranih cikličkih rotacija niza  $S$ , a temelji se na sljedećem:  $i$ -ta pojava znaka  $c$  u zadnjem stupcu (stupcu  $L$ ) leksikografski sortiranih cikličkih rotacija odgovara  $i$ -toj pojavi znaka  $c$  u prvom stupcu (stupcu  $F$ ). Dakle, znak  $BWT[i]$  se nalazi u stupcu  $F$  na mjestu  $LF[i]$ .

LF-mapiranjem možemo iz niza  $B$ , uz prisutnost stupca  $F$ , izgraditi početni niz  $S$ . Prvi stupac  $F$  sačinjavaju leksikografski sortirani znakovi iz  $S$ , a time i  $B$ , što znači da nam je

samo  $B$  dovoljan da bi odredili  $S$ . Zbog toga kažemo da je BWT reverzibilna transformacija.

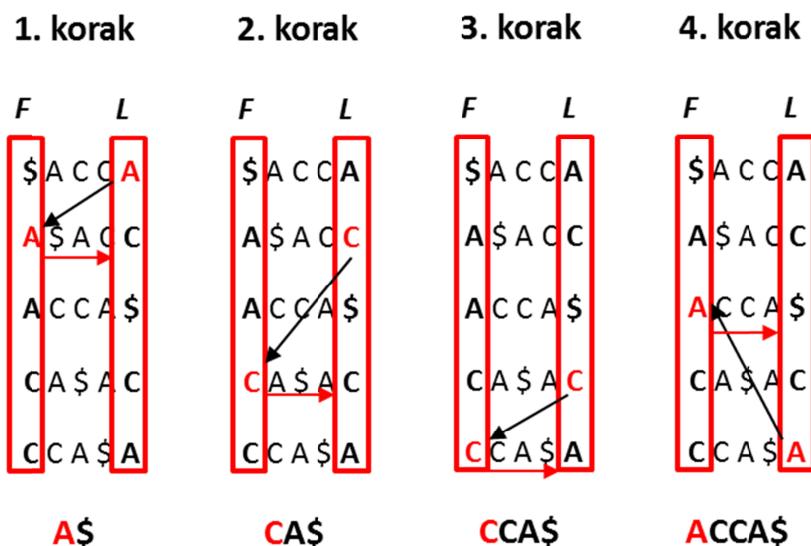
### Primjer:

Neka je zadan niz  $S = ACCA\$$ , gdje je  $\$$  leksikografski manji od ostalih znakova iz  $S$ . Niz  $B = \text{BWT}(S) = AC\$CA$  (Tablica 6.1).

Koristimo LF-mapiranje da bi iz niza  $B = AC\$CA$  izgradili niz  $S$  (Sl. 6.1). Na Sl. 6.1 zadnji stupac abecedno poredanih cikličkih rotacija od  $S$  označen je s  $L$ , a prvi s  $F$ . Stupac  $L$  je ujedno i  $B = \text{BWT}(S)$ . Da bi izgradili  $S$ , osim  $B$  (koji je pohranjen u stupcu  $L$ ), potreban nam je još stupac  $F$ . Stupac  $F$  sačinjavaju početni znakovi abecedno sortiranih sufiksa, a time i abecedno sortirani znakovi iz  $S$ , odnosno  $B$ .

U prvom koraku krećemo od prvog znaka u  $B$  (stupac  $L$ ), tj. od  $B[0] = A$ : znak  $A$  u stupcu  $L$  je označen crvenom bojom (Sl. 6.1). Znak  $A$  na početku stupca  $L$  je izravni prethodnik (ako promatramo niz zdesna nalijevo) znaka  $\$$  za koji znamo da je zadnji znak u  $S$ , pa ga onda dodajemo slijeva od  $\$$ , tj. rekonstruiramo niz  $S$  unazad.

S obzirom da promatrani znak  $A$  predstavlja prvo pojavljivanje znaka  $A$  u stupcu  $L$ , onda i u stupcu  $F$  tražimo prvo pojavljivanje znaka  $A$  (na Sl. 6.1 u 1. koraku prva pojavljivanja znakova  $A$  u stupcima  $L$  i  $F$  povezani su crnom strjelicom). Ovdje smo zapravo primijenili pravilo LF-mapiranja koje kaže da  $i$ -ta pojava znaka  $c$  u zadnjem stupcu ( $L$ ) odgovara  $i$ -toj pojavi znaka  $c$  u prvom stupcu ( $F$ ).



Sl. 6.1 Reverzibilnost BWT: iz niza  $B = AC\$CA$  u posljednjem stupcu ( $L$ ) LF-mapiranjem se dobije niz  $S = ACCA\$$

Na kraju 1. koraka određuje sljedeći znak u stupcu  $L$  (od kojeg ćemo nastaviti konstrukciju niza  $S$ ): to je znak C kojeg smo dobili tako da smo povukli crvenu strjelicu od prvog pojavljivanja znaka A u  $F$  prema znaku na istom mjestu u  $L$  (Sl. 6.1). To je moguće napraviti jer u originalnom nizu  $S$  znak u prvom stupcu slijedi znak u zadnjem stupcu ( $L[i]$  je ispred  $F[i]$ ); u ovom primjeru znak A slijedi znak C. To je vidljivo kada promatramo cijelu cikličku rotaciju na tom mjestu (A\$ACC), tj. ako A zarotiramo, onda će doći iza C.

Analogno 1. koraku rekonstruiramo i sve ostale znakove iz  $S$  (Sl. 6.1), sve do zadnjeg znaka u nizu, \$, kada zaustavljamo postupak (u 5. koraku smo pronašli zadnji, 5. znak u nizu  $S$ ).

## 6.3. FM-indeks

### 6.3.1. Interval sufiksnog polja

Neka je zadan niz  $S$  duljine  $n$  nad abecedom  $\Sigma$  ( $S$  završava znakom \$ koji je manji od svih ostalih znakova iz  $\Sigma$ ). Za niz  $S$  izgrađeno je sufiksno polje SA. Želimo pronaći sva pojavljivanja podniza  $P$  u  $S$ .

Promatramo abecedno poredane sufikse niza  $S$ , čiji redoslijed znamo prema SA. U sufiksnom se polju svi sufiksi, čiji je prefiks  $P$ , nalaze slijedno jedan iza drugog. Preciznije, indeksi tih sufiksa se nalaze jedan iza drugoga u SA, tj. određeni su *intervalom sufiksnog polja*  $[L_P, R_P]$ .  $L_P$  je indeks abecedno prvog sufiksa u SA kojemu je  $P$  prefiks, a  $R_P$  je indeks abecedno zadnjeg sufiksa u SA kojemu je  $P$  prefiks, tj.:

1.  $L_P = \min \{k: P \text{ je prefiks sufiksa } S_{SA[k]}\}$
2.  $R_P = \max \{k: P \text{ je prefiks sufiksa } S_{SA[k]}\}$

Za  $P = \emptyset$  vrijedi  $[L_P, R_P] = [0, n - 1]$ .

**Primjer:**

Zadan je niz  $S = ACCA\$$  i za niz  $S$  izgrađeno je sufiksno polje SA = {4, 3, 0, 2, 1} (Tablica 6.4). Tražimo podniz  $P = A$  u nizu  $S$ .

Svi sufiksi niza  $S$  koji počinju prefiksom  $P = A$  ( $s_3 = A\$$  i  $s_1 = ACCA$$ ), nalaze se slijedno jedan iza drugoga u listi abecedno sortiranih sufiksa. Indeksi abecedno sortiranih sufiksa koji počinju s  $P$  su elementi sufiksnog polja iz intervala SA[1, 2] = {3, 1}.

Tablica 6.4 Sufiksno polje SA izgrađeno za  $S = \text{ACCA\$}$ . Za podniz  $P = \text{A}$  vrijedi  $[L_P, R_P] = [1, 2]$ .

$i$	$\text{SA}[i]$	$S[\text{SA}[i], n - 1]$
0	4	\$
1	3	<b>A\$</b>
2	0	<b>ACCA\$</b>
3	2	CA\$
4	1	CCA\$

### 6.3.2. Traženje podniza u nizu korištenjem FM-indeksa

Neka je zadan niz  $S$  duljine  $n$  nad abecedom  $\Sigma$  ( $S$  završava znakom \$ koji ne postoji u  $\Sigma$ ). Za niz  $S$  su izračunati sufiksno polje SA i  $\text{BWT}(S)$ . FM-indeks, u kojemu se kombiniraju  $\text{BWT}$  i sufiksno polje za zadani znakovni niz  $S$  (Ferragina & Manzini, 2000), omogućuje učinkovito pretraživanje niza kako bi se odredilo postoji li podniz  $P$  u  $S$  te pronalazi sva pojavljivanja  $P$  u  $S$  (odnosno sve pozicije na kojima se  $P$  pojavljuje). FM-indeks zauzima najviše  $5 \cdot nH_k(S) + \varepsilon \cdot \log n$  (Ferragina & Manzini, 2000).

FM-indeks omoguće određivanje broja pojavljivanja  $P$  u  $S$  metodom *pretraživanja unatrag* (engl. *backward search*). Pretraživanje unatrag temelji se na LF-mapiranju.

Algoritam u kojemu se broje pojavljivanja podniza  $P$  u  $S$  zove se *BW\_Count* (Ferragina & Manzini, 2000). Algoritam koristi pomoćno polje  $C$  i funkciju  $Occ$  (ovisno o implementaciji,  $Occ$  može biti i dvodimenzionalno polje).

Polje  $C$  ima elemenata koliko je i znakova abecede  $\Sigma$ . Za svaki znak  $c$  abecede  $\Sigma$ , vrijednost  $C[c]$  jednaka je broju pojavljivanja znakova u nizu  $S$  koji su leksikografski ispred znaka  $c$ .  $Occ$  je funkcija (ili struktura podataka, ovisno o realizaciji) koja za zadani znak  $c$  i poziciju  $i$ ,  $Occ(c, i)$ , vraća broj pojavljivanja znaka  $c$  u prefiksnu  $B[0, i]$ , gdje je  $B = \text{BWT}(S)$ .

*BW\_Count* algoritam kreće s pretraživanjem niza  $S$  počevši od zadnjeg znaka podniza  $P$ . Za svaki promatrani sufiks podniza  $P$  (počevši od sufiksa koji sadrži samo zadnji znak iz  $P$ ) određuje se interval pozicija  $[L_P, R_P]$  na kojima se ti sufiksi pojavljuju. Postupak se

ponavlja sve dok se ne pregledaju svi znakovi iz  $P$ . Ako  $P$  postoji u  $S$ , algoritam vraća broj pojavljivanja  $P$  u  $S$ , koji iznosi:  $(R_P - L_P + 1)$ , a ako  $P$  ne postoji u  $S$ , vraća 0.

**Ulaz:**  $C, Occ, P$

**Izlaz:** broj pojavljivanja podniza  $P$  u nizu  $S$

$i = |P| - 1$  /\* kreće od zadnjeg mesta u  $P$  \*/

$c = P[|P| - 1]$  /\* zadnji znak u  $P$  \*/

$L_P = C[c]$

$R_P = C[cs/jed] - 1$  /\*  $cs/jed$  je znak nakon  $c$  u  $\Sigma$  \*/

**ponavljati za**  $((L_P \leq R_P) \text{ i } (i \geq 1))$

$i = i - 1$

$c = P[i];$  /\* sljedeći znak iz  $P$  \*/

$L_P = C[c] + Occ(c, L_P - 1)$

$R_P = C[c] + Occ(c, R_P) - 1$

**ako**  $(R_P < L_P)$  **onda vratiti** 0

**inače vratiti**  $(R_P - L_P + 1)$

Sl. 6.2 Algoritam BW\_Count (Ferragina & Manzini, 2000)

## Primjer

Neka je zadan  $S = ACCA\$$ ; pretpostavljamo da je znak \$ leksikografski manji od svih drugih znakova iz  $S$ . Duljina niza  $S$  je  $n = 5$ . Sufiksno polje SA izgrađeno nad  $S$  je  $SA = \{4, 3, 0, 2, 1\}$ . Želimo odrediti koliko se puta podniz  $P = CA$  pojavljuje u nizu  $S$ .

Kako bi primijenili algoritam  $BW\_Count$  na traženje  $P$  u  $S$ , prvo trebamo odrediti vrijednosti pomoćnog polja  $C$  i strukture  $Occ$ .  $Occ$  koristi  $B = BWT(S) = AC\$CA$  (Tablica 6.1).

Vrijednosti elemenata polja  $C$ ,  $C[c]$ , računaju se kao broj znakova u  $S$  koji su abecedno prije  $c$ :

- $C(' \$ ') = 0$ , jer ne postoji niti jedan znak u  $S$ , koji je abecedno manji od \$;
- $C(' A ') = 1$ , jer je znak \$ abecedno manji od znaka A, a pojavljuje se jedanput;
- $C(' C ') = 3$ , jer su znakovi \$ i A abecedno manji od C, a pojavljuju se ukupno tri puta (znak \$ se pojavljuje jedanput, a znak A dva puta).

Vrijednosti elemenata strukture  $Occ$ ,  $Occ(c, i)$ , računaju se kao broj pojavljivanja  $c$  u  $B[0, i]$  ( $B = AC\$CA$ ):

- $Occ('A', 0) = 1$ , jer se znak A pojavljuje jedanput u  $B[0, 0]$ ;
- $Occ('A', 1) = 1$ , jer se znak A pojavljuje jedanput u  $B[0, 1]$ ;  
itd.
- $Occ('C', 0) = 0$ , jer se znak C ne pojavljuje niti jednom u  $B[0, 0]$ ;
- $Occ('C', 1) = 1$ , jer se znak C pojavljuje jedanput u  $B[0, 1]$ ;
- $Occ('C', 2) = 1$ , jer se znak C pojavljuje jedanput u  $B[0, 2]$ ;  
itd.

Traženje  $P$  u  $S$  počinjemo sa zadnjim znakom niza  $P = CA$ , tj. s  $P[i = 1] = A$ . Određujemo inicijalne vrijednosti  $L_P$  i  $R_P$  (algoritam  $BW\_Count$ , Sl. 6.2) korištenjem prethodno izračunatih vrijednosti za polje  $C$ :

- $L_P = C['A'] = 1$
- $R_P = C(csljed) - 1 = C('C') = 2$

Ovime smo odredili da je interval pozicija leksikografski poredanih sufiksa koji počinju znakom A u sufiksnom polju  $SA(S)$ :  $[L_P, R_P] = [1, 2]$ . Ako pogledamo  $SA(S)$ , vidimo da se interval  $[L_P, R_P] = [1, 2]$  odnosi na elemente sufiksnog polja  $SA[1] = 3$  i  $SA[2] = 1$  što odgovara sufiksima  $s_3 = A\$$  i  $s_1 = ACCA\$$ .

U sljedećem koraku algoritma  $BW\_Count$  smanjujemo vrijednost varijable  $i$  za 1 ( $i$  postaje 0) i pretraživanje nastavljamo sljedećim znakom niza  $P$ ,  $P[0] = C$ . Među sufiksima iz intervala  $SA[1, 2]$  pronađenima u prethodnom koraku gledamo postoji li sufiks kojemu je prethodnik znak C (Tablica 6.5; crvenom strjelicom su povezani sufiks koji počinje slovom A, a kojemu prethodi znak C u originalnom nizu  $S$ ). U algoritmu, to se određuje računanjem novih vrijednosti  $L_P$  i  $R_P$ :

$$L_P = C['C'] + Occ('C', L_P - 1) = 3 + Occ('C', 0) = 3 + 0 = 3$$

$$R_P = C['C'] + Occ('C', R_P) - 1 = 3 + Occ('C', 2) - 1 = 3 + 1 - 1 = 3$$

Ovime smo odredili da je interval pozicija leksikografski poredanih sufiksa koji počinju nizom  $P = CA$  u sufiksnom polju  $SA(S)$ :  $[L_P, R_P] = [3, 3]$ .

S obzirom da je  $P[0] = C$  početni znak niza  $P$ , algoritam se ovdje zaustavlja i vraća vrijednost:

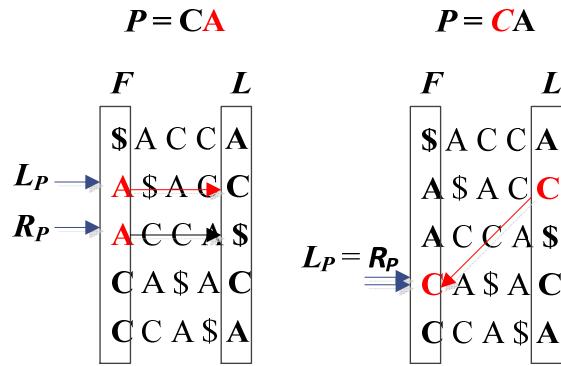
$$R_P - L_P + 1 = 3 - 3 + 1 = 1,$$

što znači da se  $P$  pojavljuje samo jedanput u  $S$ .

Tablica 6.5 Traženje niza  $P = CA$  u nizu  $S = ACCA\$$  prema algoritmu  $BW\_Count$

$i$	$SA[i]$	Abecedno poredane cikličke rotacije niza $S$	$B[i]$
0	4	\$ACCA	A
1	0	ACCA\$	C
2	3	A\$ACC	\$
3	2	CA\$AC	C
4	1	CCA\$A	A

Drugi način kako možemo odrediti postoji li  $P = CA$  u  $S = ACCA\$$  je korištenjem LF-mapiranja (Sl. 6.3). Kao što vidimo na Sl. 6.3, traženje  $P$  u  $S$  počinje sa zadnjim znakom niza  $P$ , tj. s A. U prvoj koraku u stupcu  $F$ , koji čine početni znakovi abecedno poredanih cikličkih rotacija niza  $S$ , pronalazimo interval  $[L_P, R_P] = [1, 2]$  koji se odnosi na cikličke rotacije niza  $S$  koje počinju znakom A. Te se cikličke rotacije zapravo odnose na sufikse niza  $S$  koji počinju znakom A (sufiks je prefiks dotičnog niza do terminalnog znaka \$).. Ovime smo utvrdili da znak A postoji u  $S$ .



Sl. 6.3 Traženje niza  $P = CA$  u nizu  $S = ACCA\$$  LF-mapiranjem (stupac  $L$  predstavlja niz  $B = \text{BWT}(S) = AC\$CA$ ).

Dalje nam valja utvrditi postoji li sufiks CA u nizu  $S$ . Kao što smo objasnili u prethodnom poglavlju (poglavlje 6.2.3), kada promatramo niz  $S$ , znak u nekom retku prvog stupca zapravo slijedi nakon znaka u zadnjem stupcu u istom retku, tj. za svaki  $i$  vrijedi da se  $F[i]$  u nizu  $S$  nalazi iza  $L[i]$ . Zato, ako želimo utvrditi postoji li podniz oblika CA, valja nam provjeriti retke prvog stupca u kojima se nalazi slovo A (a to su reci u intervalu  $[L_P, R_P]$ ),

nalazi li se slovo C u istom retku u zadnjem stupcu. Ako se C nalazi u zadnjem stupcu u nekom retku, onda to znači da C prethodi znaku A u nizu  $S$ . Za  $L[1] = A$  i  $F[1] = C$  je znak C doista traženi prethodnik znaka A, dok za  $L[2] = A$  i  $F[2] = \$$  to nije slučaj. Prema tome, pretraživanje nastavljamo samo od  $L[1]$ .

Kako LF-mapiranje znači da  $k$ -ta pojava znaka  $c$  u stupcu  $L$  odgovara  $k$ -toj pojavi znaka  $c$  u stupcu  $F$ , tako za znak C na mjestu  $L[1]$  (što predstavlja prvu pojavu znaka C u stupcu  $L$ ), tražimo prvu pojavu znaka C u stupcu  $F$ , tj. promatramo C na mjestu  $F[3]$ . Time dobivamo kao konačno rješenje interval  $[L_P, R_P] = [3, 3]$ , koji pokazuje na cikličku rotaciju CR[3] (tj. na sufiks  $s_3$  koji počinje prefiksom koji je jednak  $P$ ). S obzirom da je znak C ujedno i zadnji znak u nizu  $P$  gledano zdesna nalijevo, tu zaustavljamo pretraživanje i vraćamo interval  $[L_P, R_P] = [3, 3]$  kao konačno rješenje.

### 6.3.3. Prostorna i vremenska složenost FM-indeksa

FM-indeks je samostojni indeks čiji su prostorni zahtjevi blizu teoretskog optimuma, tj. proporcionalni su  $k$ -toj entropiji ulaznog niza. Teorijsko memorijsko zauzeće za niz  $S$  duljine  $n$  je  $O(nH_k(S)) + o(n)$  bita (Ferragina & Manzini, 2000).

Brojanje pojavljivanja podniza  $P$  u  $S$  zahtijevaju vrijeme  $O(|P|)$  u najboljem teoretskom slučaju (Ferragina & Manzini, 2000), ali je praktično to vrijeme veće i ovisi o implementaciji samog indeksa.

## 6.4. Literatura

- Burrows,M. and Wheeler,D.J. (1994) A block-sorting lossless data compression algorithm.
- Ferragina,P. *et al.* (2008) Compressed text indexes: From theory to practice. In, *ACM Journal of Experimental Algorithms*.
- Ferragina,P. and Manzini,G. (2000) Opportunistic Data Structures with Applications. In, *FOCS.*, pp. 390–398.
- Kurtz,S. (1998) Reducing the Space Requirement of Suffix Trees. *Softw. – Pract. Exp.*, **29**, 1149–1171.
- Navarro,G. (2009) Implementing the LZ-index: Theory Versus Practice. *J. Exp. Algorithmics*, **13**, 2:1.2–2:1.49.
- Sadakane,K. (2003) New Text Indexing Functionalities of the Compressed Suffix Arrays. *J. Algorithms*, **48**, 294–313.
- Seward,J. (2007) bzip2. <http://www.bzip.org>.
- Shannon,C. (1948) A Mathematical Theory of Communication. *Bell Syst. Tech. J.*, **27**, 379–423, 623–656.
- Ziv,J. and Lempel,A. (1977) A universal algorithm for sequential data compression. *IEEE Trans. Inf. Theory*, **23**, 337–343.
- Ziv,J. and Lempel,A. (1978) Compression of individual sequences via variable-rate coding. *IEEE Trans. Inf. Theory*, **24**, 530–536.

## 7. Filogenija

### 7.1.1. Povijesni razvoj ideja

Danas se smatra da je život na Zemlji počeo prije otprilike 3.8 milijardi godina. Današnji živući, a isto tako i izumrli organizmi, potomci su zadnjeg zajedničkog pretka: prve žive stanice. Genetički materijal prve žive stanice prenosio se na njezine potomke kopiranjem. Međutim, taj proces nije bio savršen: pratile su ga mutacije i rekombinacije. Sve takve promjene u genomima živih bića prenosile su se na određen broj sljedećih generacija, osim onih koje su u velikoj mjeri negativno utjecale na reproduktivnu spremu (engl. *fitness*), pa su vrlo brzo nestajale iz genoma procesom prirodne selekcije.

Teorija evolucije opisuje proces i objašnjava mehanizme kojima su se od zajedničkog pretka (engl. *common ancestor*) razvili živući i izumrli organizmi. Teoriju evolucije i utjecaj prirodne selekcije na razvoj živih bića izložili su Charles Darwin i Alfred Russel Wallace prvi puta 1858. godine u radu: *On the tendency of species to form varieties; and on the perpetuation of varieties and species by natural means of selection* (Darwin & Wallace, 1858). Sljedeće je godine C. Darwin objavio knjigu *On the Origin of Species* (naslov 1. izdanja: *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*) (Darwin, 1859) u kojoj je detaljno izložio teoriju evolucije te opisao svoja istraživanja i zapažanja za vrijeme petogodišnje plovidbe svjetom brodom *Beagle*. Zanimljivo je da je jedina slika u knjizi stablo koje prikazuje evolucijske odnose između pojedinih vrsta: u korijenu stabla je zajednički predak, a kroz grananje stabla opisan je razvoj potomaka. Danas takvo stablo nazivamo filogenetskim stablom, prema pojmu filogenija, koji je uveo Ernst Haeckel (Haeckel, 1866). E. Haeckel je objavio i prva filogenetska stabla kojima je povezao sve poznate oblike života (Haeckel, 1866; Dayrat, 2003)<sup>12</sup>. Prikaz odnosa organizama u stablu temelji

---

<sup>12</sup> E. Haeckel je, među ostalima, uveo i pojmove ekologija i ontogenija. On je prepostavljaо da je ontogenija (životni ciklus jedinke) rekapitulacija filogenije, odnosno evolucijske povijesti svih živih bića. Ta je teorija bila osporavana od početka 20. stoljeća, a danas se smatra da postoji složena veza između filogenije i ontogenije.

se na njihovim morfološkim i molekularnim sličnostima i razlikama, a međusobno srodniji organizmi grupirani su zajedno u podstabla.

Otprilike u isto vrijeme, Gregor Mendel istraživao je prijenos nasljednih osobina s roditelja na potomke i rezultate svojih eksperimenata križanja graška objavio je u radu *Versuche über Pflanzenhybriden* (Eksperimenti u hibridizaciji biljaka) (Mendel, 1865). On je pokazao da organizmi nasljeđuju osobine preko diskretnih, a ne kontinuiranih jedinica za nasljeđivanje (kako se tada smatralo) koje danas nazivamo genima. Iako je njegov rad prošao relativno nezapaženo u vrijeme kad je objavljen, ponovno je otkriven na prijelazu u 20. stoljeće i danas se Mendel smatra ocem suvremene genetike.

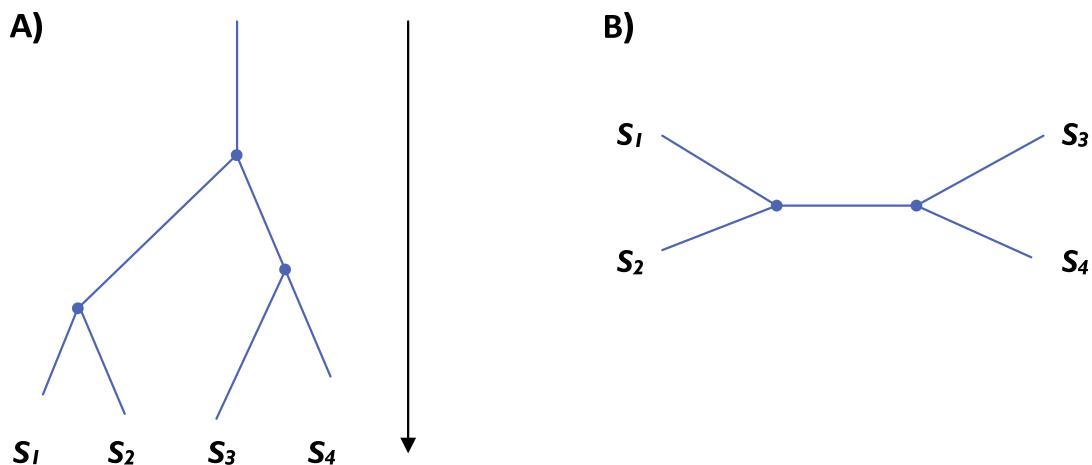
## 7.2. Filogenija i filogenetsko stablo

*Filogenija* (engl. *phylogeny*) je prikaz evolucijskih odnosa između organizama i, kao što je već spomenuto, ti se odnosi obično prikazuju *filogenetskim stablom* (engl. *phylogenetic tree*). Filogenetsko stablo uključuje topološku i, u slučaju ukorijenjenog stabla, vremensku dimenziju (Sl. 7.1). Organizmi prikazani u čvorovima filogenetskog stabla su sistematske jedinice: npr. vrste, populacije, rodovi, itd. Sistematske jedinice još se nazivaju i taksonomske jedinice ili taksoni, što dolazi od pojma taksonomija (znanstvena disciplina koja proučava klasifikaciju organizama).

Izgradnja filogenetskog stabla temeljem molekularnih sličnosti taksona odnosi se na izgradnju stabla temeljem sličnosti sljedova. Sljedovi mogu biti skupovi gena (ili proteina) ili cijeli genomi (ili proteomi). U prvome slučaju, govorimo o izgradnji genskih ili proteinskih stabala (engl. *gene tree; protein tree*), gdje se izgradnja stabla temelji na pretpostavci da promatrani geni imaju zajedničkog pretka. Kako bi se odredili filogenetski odnosi između vrsta, danas se izgrađuju i stabla koja koriste veći broj filogenetski informativnih dijelova genoma.

Na Sl. 7.1 prikazana su dva osnovna tipa filogenetskog stabla: ukorijenjeno (engl. *rooted*) i neukorijenjeno (engl. *unrooted*) filogenetsko stablo. Duljina neke grane u stablu odnosi se na relativni broj supstitucija koje su se dogodile između sljedova koji predstavljaju čvorove koje ta grana povezuje. Ako je supstitucijska stopa (engl. *substitution rate*) jednaka u svim granama kroz svo vrijeme, onda kažemo da vrijedi pretpostavka konstantne frekvencije molekularnog sata (engl. *molecular clock*). Ova pretpostavka općenito ne vrijedi.

*Ukorijenjeno* stablo pokazuje smjer evolucijskog procesa određujući odnos između pretka i potomaka, odnosno, ima prepostavljen smjer vremena. Na Sl. 7.1 A) prikazano je ukorijenjeno stablo koje ima četiri lista za četiri taksona – sljedove  $S_1$ ,  $S_2$ ,  $S_3$  i  $S_4$ . Unutarnji čvorovi stabla predstavljaju sljedove-pretke, a listovi predstavljaju sljedove-potomke. Pored stabla prikazana je strjelica koja prikazuje smjer vremena (vrijeme diferencijacije; engl. *differentiation time*) u evolucijskoj povijesti sljedova. Iz stabla se također može vidjeti da su sljedovi  $S_1$  i  $S_2$  međusobno srodniji nego ijedan od njih u odnosu na  $S_3$  i  $S_4$ . Iz stabla vidimo i da  $S_1$  i  $S_2$  imaju zajedničkog pretka, isto kao i  $S_3$  i  $S_4$  te da svi sljedovi zajedno imaju jednog zajedničkog pretka predstavljenog korijenom stabla.



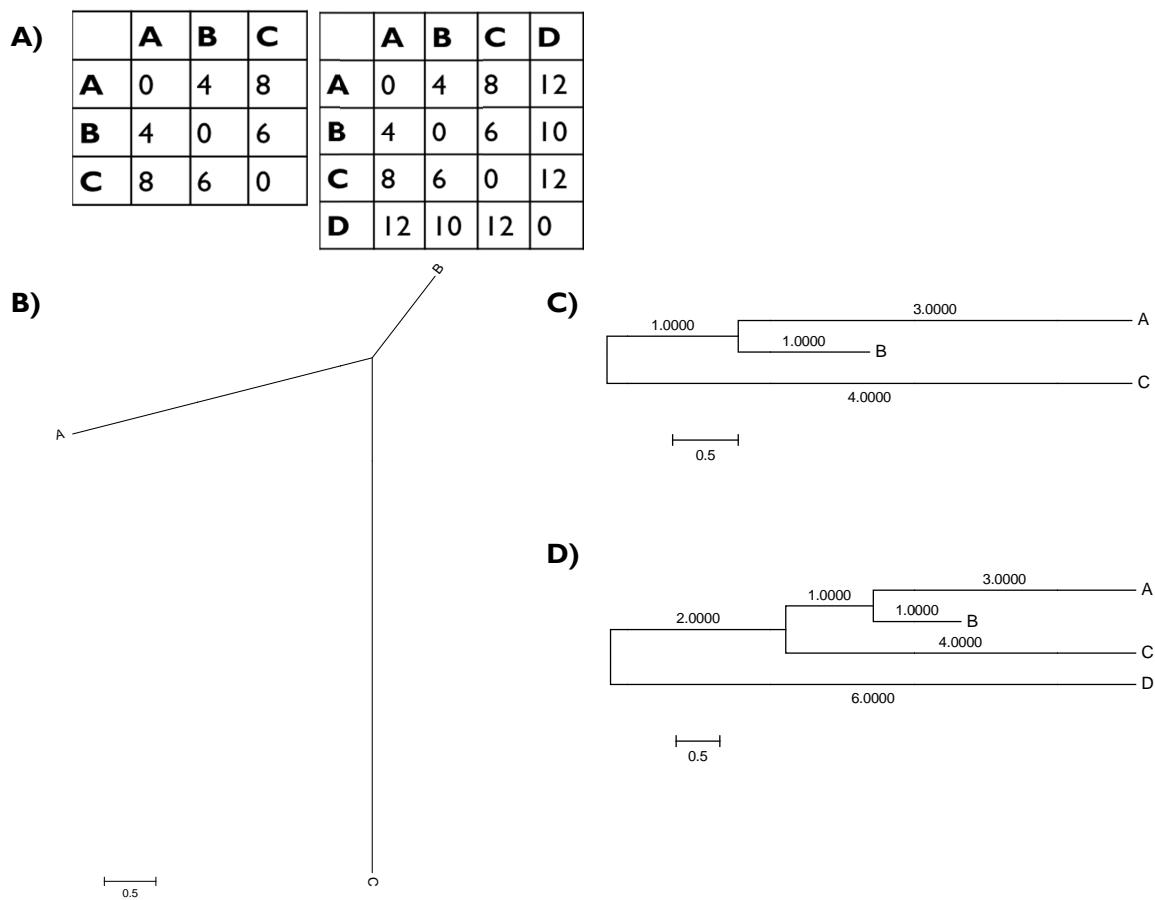
Sl. 7.1 A) Ukorijenjeno (engl. *rooted*) i B) neukorijenjeno (engl. *unrooted*) filogenetsko stablo za 4 sljeda  $S_1$ ,  $S_2$ ,  $S_3$  i  $S_4$ . Ukorijenjeno stablo ima ugrađen smjer vremena (od zajedničkog pretka u korijenu stabla prema potomcima u listovima stabla). Neukorijenjeno stablo nema ugrađen smjer vremena.

Na Sl. 7.1B) prikazano je neukorijenjeno stablo s četiri lista za 4 taksona – sljedove  $S_1$ ,  $S_2$ ,  $S_3$  i  $S_4$ . Neukorijenjeno stablo ne sadrži smjer evolucije (vremena) i nema korijen. Iz stabla možemo iščitati samo relativne odnose među taksonima: ovdje su  $S_1$  i  $S_2$  međusobno bliži u odnosu na  $S_3$  i  $S_4$ .

S obzirom da ukorijenjeno stablo sadrži više informacija od neukorijenjenoga (poznat je zajednički predak, odnosno smjer vremena u stablu), ponekad je prikladno neukorijenjeno stablo pretvoriti u ukorijenjeno. To se može obaviti na jedan od sljedećih načina (Sl. 7.2):

1. Dodavanjem vanjske taksonomske jedinice (engl. *outgroup rooting*) u postojeće neukorijenjeno stablo. Naknadno dodani takson treba biti najudaljeniji takson u odnosu na sve ostale taksone u početnom stablu.
2. Dodavanjem korijena na sredinu između grana koje povezuju dva najudaljenija lista postojećeg neukorijenjenog stabla (engl. *mid-point rooting*).

Na Sl. 7.2A) prikazane su matrice udaljenosti za taksone A, B i C, te za taksone A, B, C i D, gdje je D najudaljeniji takson u odnosu na A, B i C. Na Sl. 7.2B) prikazano je neukorijenjeno stablo za taksone A, B i C (duljine grana odgovaraju udaljenostima iz matrice). Na Sl. 7.2C) prikazano je ukorijenjeno stablo koje je dobiveno iz stabla na Sl. 7.2B) tako da se korijen dodao na polovicu grane između dvaju najudaljenijih taksona: A i C (engl. *mid-point rooting*). Na Sl. 7.2D) prikazano je ukorijenjeno stablo, gdje je korijen dodan između taksona D i podstabla koje sadrži taksone A, B i C (engl. *outgroup rooting*).



Sl. 7.2 Pretvaranje neukorijenjenog stabla u ukorijenjeno stablo za taksone A, B i C. **A)** Matrica udaljenosti za taksone A, B i C, te matrica udaljenosti za taksone A, B, C i D, gdje je D vanjski takson u odnosu na A, B i C. **B)** Neukorijenjeno stablo za taksone A, B i C. **C)** Ukorijenjeno stablo za taksone A, B i C koje se dobilo postavljanjem korijena na sredinu grane između dvaju

najudaljenijih taksona A i C (engl. *mid-point rooting*). **D)** Ukorijenjeno stablo koje se dobilo za A, B i C tako da je dodan vanjski takson D koji je od svih njih najudaljeniji (engl. *outgroup rooting*).

### 7.2.1. Broj mogućih stabala u ovisnosti o broju taksonomske jedinice

Neka je  $n$  broj taksonomske jedinice za koje želimo izgraditi filogenetsko stablo. Ako je  $n = 3$  i zadani su taksoni A, B i C, tada je moguće izgraditi samo jedno neukorijenjeno stablo (primjer na Sl. 7.2B)) ili 3 ukorijenjena stabla, ovisno o tome koji je par taksona najbliži. Na primjer, ako su A i B međusobno bliži u odnosu na C, onda su A i B zajedno grupirani, a C je vanjski takson (engl. *outgroup*); analogno za slučajeve kada su A i C najbliži ili kada su B i C najbliži.

Ako je  $n \geq 3$ , onda su broj mogućih neukorijenjenih stabla,  $N_U$ , i broj mogućih ukorijenjenih stabla,  $N_R$ , određeni sljedećim izrazima (Cavalli-Sforza & Edwards, 1967):

$$N_U = \frac{(2n-5)!}{2^{n-3}(n-3)!} \quad (7.1)$$

$$N_R = \frac{(2n-3)!}{2^{n-2}(n-2)!} \quad (7.2)$$

Kao što je vidljivo iz (7.1) i (7.2), broj mogućih stabala u oba slučaja raste u ovisnosti o faktorijelima, tako da je npr. za  $n = 5$ :  $N_R = 105$  i  $N_U = 15$ , a već za  $n = 20$ :  $N_R = 8 \cdot 10^{21}$  i  $N_U = 2 \cdot 10^{20}$ . Za  $n = 50$ :  $N_R$  je  $2.8 \cdot 10^{76}$  (Pevsner, 2009b). Jasno je, dakle, da je u praksi teško ili nemoguće ispitati sva moguća stabla već za desetak ili više taksona kako bi se odredilo najbolje stablo prema nekoj unaprijed zadanoj mjeri. Zbog toga se koristi heuristički pristup, kojim se smanjuje skup mogućih stabala koje treba pregledati.

## 7.3. Evolucijski DNA modeli

Kada promatramo 2 nukleotidna ili aminokiselinska slijeda  $S_1$  i  $S_2$ , želimo izračunati koliko su međusobno divergirali (odnosno, koliko je svaki od njih divergirao od slijeda  $S_0$  koji im

je bio zajednički predak, ako nam je  $S_0$  poznat). Kako bi se odredila evolucijska udaljenost između  $S_1$  i  $S_2$ , prvo se određuje vidljivi broj razlika između promatranih sljedova, što se obično računa iz njihova poravnanja. Zatim se iz vidljivih razlika određuje stvarna evolucijska udaljenost između sljedova primjenom određenog evolucijskog modela. Ono što je važno naglasiti jest da je broj vidljivih razlika u pravilu manji ili eventualno jednak broju promjena koje su se stvarno dogodile kroz njihovu evolucijsku povijest. Štoviše, što su  $S_1$  i  $S_2$  više divergirali, to je razlika između stvarnog broja evolucijskih promjena i vidljivog broja razlika veća. Uzmimo da je prvi nukleotid slijeda  $S_0$  bio A, a sada se na tom mjestu u  $S_1$  nalazi nukleotid C, dok se u slijedu  $S_2$  nalazi nukleotid A. To ne znači da je evolucija iz  $S_0$  u  $S_1$  uključivala samo jednu izmjenu nukleotida niti da slijed  $S_2$  na toj poziciji nije mutirao. Mi možemo, ovisno o tome koliko procjenjujemo da je vremena proteklo te kojoj brzinom procjenjujemo da dotični organizmi mutiraju, prepostaviti stvaran broj mutacija.

Kako bi se odredio stvaran broj evolucijskih promjena za promatrani par sljedova, razvijeni su mnogi matematički modeli. Najpoznatiji među njima su: Jukes-Cantorov model (Jukes & Cantor, 1969), Kimurin model (Kimura, 1980), itd. te najopćenitiji i najsloženiji model, GTR model (*Generalised time-reversible*) (Tavaré, 1986). Ovdje ćemo izložiti najjednostavniji, odnosno, Jukes-Cantorov model za DNA sljedove. Slični modeli, osim za DNA, postoje i za aminokiselinske sljedove.

### 7.3.1. Jukes-Cantorov model

Jukes-Cantorov model (Jukes & Cantor, 1969) je najjednostavniji model kojim se procjenjuje stvarna evolucijska udaljenost, odnosno stvarni broj promjena između promatranih sljedova. Zbog svoje jednostavnosti prikladan je samo za određivanje evolucijske udaljenosti između vrlo srodnih sljedova.

Osnovne prepostavke na kojima se temelji ovaj model su:

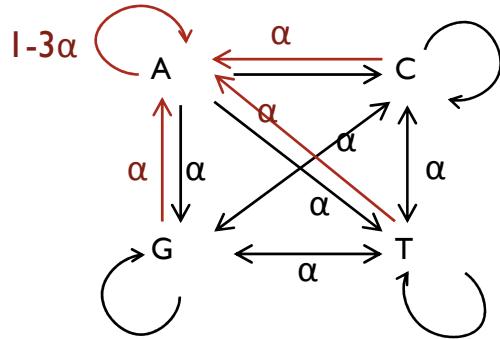
1. Jednaka je frekvencija svih nukleotida u promatranim sljedovima,<sup>13</sup> tj.

$$\pi_A = \pi_C = \pi_G = \pi_T = 0.25$$

---

<sup>13</sup> Prepostavke na kojima se temelji Juke-Cantorov model ne vrijede za mnoge organizme (kao npr. jednaka frekvencija svih nukleotida).

- Supstitucijska stopa u jedinici vremena (engl. *substitution rate*)  $\alpha$  jednaka je za sve nukleotide, tj. jednaka je vjerojatnost da će bilo koji nukleotid X mutirati u bilo koji drugi nukleotid Y ( $X \neq Y$ ).
- Nukleotidi u sljedovima su međusobno neovisni (engl. *independently identically distributed*).



Sl. 7.3 Jukes-Cantorov model: vjerojatnost prijelaza iz jednog stanja u drugo, tj. vjerojatnost prijelaza iz jednog nukleotida u drugi u jedinici vremena označena je s  $\alpha$  za sve nukleotide.

Vjerojatnost ostanka u istom stanju je  $1-3\alpha$ .

Na Sl. 7.3 prikazan je model prijelaza iz jednog stanja u drugo, odnosno vjerojatnost supsticije jednog nukleotida drugim.

Sada ćemo izvesti vjerojatnost da se na nekom mjestu u slijedu, gdje je u trenutku  $t = 0$  bio neki nukleotid, u nekom kasnijem trenutku  $t > 0$  na tom mjestu nalazi isti ili neki drugi nukleotid. Proizvoljno biramo da je početno stanje A tj. da za  $t = 0$  vrijedi  $P_{A(0)} = 1$ .

Zatim promatramo trenutak  $t = 1$ . Neka je  $P_{AA(1)}$  vjerojatnost prijelaza  $A \rightarrow A$  u trenutku  $t=1$ .  $P_{AA(1)}$  se još može kraći pisati  $P_{A(1)}$ , gdje je implicitno prepostavljeno početno stanje (A), pa se početno stanje ne navodi. Vrijedi (Sl. 7.3):

$$P_{AA(1)} = p_{AA}(t=1) = 1 - 3\alpha \quad (7.3)$$

U trenutku  $t = 1$ , ukupna vjerojatnost svih supsticija  $A \rightarrow Y$  ( $A \neq Y$ ) iznosi  $3\alpha$ , jer je  $p_{AC}(t=1) = p_{AG}(t=1) = p_{AT}(t=1) = \alpha$  (Sl. 7.3).

Općenito, neka je s  $P_{A(t)}$  označena vjerojatnost da je nukleotid u stanju A u trenutku  $t$ , a neka je s  $P_{A(t+1)}$  označena vjerojatnost da je nukleotid u stanju A u trenutku  $t+1$ . Vrijedi sljedeće:

1. Neka je u nekom trenutku  $t$  nukleotid u stanju A. Vjerojatnost da u trenutku  $t+1$  ostaje u stanju A je  $(1 - 3\alpha) \cdot P_{A(t)}$
2. Neka je u nekom trenutku  $t$  nukleotid u stanju  $Y \neq A$  (tj.  $Y = C, G$  ili  $T$ ). Vjerojatnost da će u sljedećem vremenskom trenutku  $t+1$  mutirati u A je  $\alpha(1 - P_{A(t)})$

Konačno, ukupna vjerojatnost da će na nekom mjestu u slijedu biti nukleotid A u trenutku  $t+1$  je  $P_{A(t+1)}$ :

$$P_{A(t+1)} = (1 - 3\alpha) P_{A(t)} + \alpha(1 - P_{A(t)}) \quad (7.4)$$

$$\Delta P_{A(t)} = P_{A(t+1)} - P_{A(t)} = -3\alpha P_{A(t)} + \alpha(1 - P_{A(t)}) = -4\alpha P_{A(t)} + \alpha \quad (7.5)$$

$\Delta P_{A(t)}$  možemo zamijeniti s  $dp/dt$ , a  $P_{A(t)}$  s  $p$ , pa izraz (7.5) možemo jednostavnije napisati kao:

$$dp / dt = -4\alpha p + \alpha \quad (7.6)$$

Iz (7.6) slijedi:

$$dt = dp / (-4\alpha p + \alpha) \quad (7.7)$$

$$\int dt = \int \frac{dp}{-4\alpha p + \alpha} \Rightarrow t = \frac{-1}{4\alpha} \ln(-4\alpha p + \alpha) + C \quad (7.8)$$

Konstantu  $C$  određujemo prema početnom uvjetu, tj.  $P_{A(0)} = 1$ , što uvrstimo u (7.8) i dobivamo:

$$C = \frac{1}{4\alpha} \ln(-3\alpha) \quad (7.9)$$

Zatim (7.9) uvrstimo u (7.8), iz čega slijedi:

$$t = \frac{-1}{4\alpha} \ln(-4\alpha p + \alpha) + \frac{1}{4\alpha} \ln(-3\alpha) \quad (7.10)$$

Konačno,

$$p = P_{A(t)} = P_{AA(t)} = \frac{1}{4} + \frac{3}{4} e^{-4\alpha t} \quad (7.11)$$

Vjerojatnost  $P_{AA(t)}$ , tj. vjerojatnost da je na nekom mjestu u slijedu, gdje je na početku bio nukleotid A, nakon vremena  $t$  ponovno nukleotid A opisana je izrazom (7.11). Iz  $P_{AA(t)}$  određujemo zatim  $P_{AY(t)}$  kao vjerojatnost da se iz početnog nukleotida A nakon vremena  $t$  na istom mjestu nalazi nukleotid Y  $\neq$  A (tj. Y = C, G ili T):

$$P_{AY(t)} = 1 - P_{AA(t)} = \frac{3}{4} - \frac{3}{4} e^{-4\alpha t} \quad (7.12)$$

Odnosno,

$$P_{AC(t)} = P_{AG(t)} = P_{AT(t)} = \frac{1}{4} - \frac{1}{4} e^{-4\alpha t} \quad (7.13)$$

Sada želimo odrediti evolucijsku udaljenost između dva slijeda. Prepostavit ćemo da je zajednički predak promatranog para sljedova u trenutku  $t=0$  na nekom mjestu u slijedu bio u stanju A. U trenutku  $t$  svaki od sljedova–potomaka bit će u stanju A s vjerojatnošću  $P_{AA(t)}$ . Vjerojatnost da su oba slijeda u stanju A određeno je s  $P_{AA(t)}^2$ . Analogno, oba slijeda će biti u stanju C s vjerojatnošću  $P_{AC(t)}^2$ , itd.

Neka je  $I_{(t)}$  vjerojatnost da se u oba slijeda–potomka u trenutku  $t$  u promatranom stupcu nalaze jednaki nukleotidi i da je na početku na tom mjestu bio nukleotid A. Tada vrijedi:

$$I_{(t)} = P_{AA(t)}^2 + P_{AC(t)}^2 + P_{AG(t)}^2 + P_{AT(t)}^2 = \frac{1}{4} + \frac{3}{4} e^{-8\alpha t} \quad (7.14)$$

Analogno, vjerojatnost da su u sljedovima–potomcima na promatranom mjestu različiti nukleotidi je:

$$p = 1 - I_{(t)} = \frac{3}{4}(1 - e^{-8\alpha t}) \Rightarrow 8\alpha t = -\ln\left(1 - \frac{4}{3}p\right) \quad (7.15)$$

Vjerojatnost  $p$  predstavlja vjerojatnost pojave različitih nukleotida u stupcima promatranog para sljedova.  $p$  možemo zamijeniti frekvencijom promjena, odnosno relativnim brojem razlika između promatranog para sljedova.

Neka je  $d$  označen relativni broj supstitucija od divergencije promatranog para sljedova. Ako je  $3\alpha t$  broj supstitucija u vremenu  $t$  na jednoj poziciji u jednom slijedu, tada je broj supstitucija u oba slijeda:

$$d = 2 \cdot 3\alpha t \quad (7.16)$$

Ako uvrstimo (7.15) u (7.16), dobit ćemo konačan izraz za evolucijsku udaljenost između 2 slijeda prema Jukes-Cantorovom modelu:

$$d = -\frac{3}{4} \ln\left(1 - \frac{4}{3}p\right) \quad (7.17)$$

Evolucijska udaljenost  $d$  predstavlja relativan broj supstitucija, a  $p$  se određuje kao relativan broj vidljivih razlika (odnosno, ukupan broj vidljivih mutacija podijeljen s duljinom poravnjanja).

## 7.4. Metode za izgradnju filogenetskih stabala

Metode za izgradnju filogenetskih stabala možemo podijeliti u dvije osnovne skupine:

1. Metode temeljene na udaljenostima između taksona
2. Metode temeljene na obilježjima taksona

*Metode temeljene na udaljenosti* polaze od prepostavke da su bliži (srodniji) organizmi (odnosno, njihovi sljedovi) međusobno sličniji. Određivanje evolucijske udaljenosti između parova sljedova polazi od relativnog broja razlika između promatranih sljedova (sličniji sljedovi imaju manji broj razlika) (poglavlje 7.3.). U ovoj su skupini najpoznatije metode:

1. UPGMA (engl. *Unweighted Pair Group Method with Arithmetic Mean*) (Sokal & Michener, 1958)
2. Metoda povezivanja susjeda (engl. *Neighbour-Joining*; NJ) (Saitou & Nei, 1987)

*Metode temeljene na obilježjima* mogu koristiti i morfološka obilježja (oblik i građa organizama) i molekularna obilježja (nukleotide ili aminokiseline). U ovoj skupini najpoznatije su:

1. Metoda najmanjeg broja evolucijskih promjena (engl. *maximum parsimony*)
2. Metoda najveće izglednosti (engl. *maximum likelihood*)

#### 7.4.1. Mjera udaljenosti

Neka je  $D(i, j)$  udaljenost između taksona  $i$  i  $j$ , što možemo kraće zapisati kao  $D_{ij}$ . Neka je  $X$  skup taksona. Da bi neka funkcija  $D: X \times X \rightarrow \mathbb{R}$  bila mjera udaljenosti (engl. *distance measure*), za sve  $i, j$  i  $k$  treba vrijediti sljedeće:

1.  $D_{ij} \geq 0$
2.  $D_{ij} = 0$  ako i samo ako vrijedi  $i = j$
3. simetričnost:  $D_{ij} = D_{ji}$
4. nejednakost trokuta:  $D_{ij} + D_{jk} \geq D_{ik}$

Dodatni uvjet za aditivnost je:

5. nejednakost četverokuta za taksone  $i, j, k, l$ :  $D_{ik} + D_{jl} = D_{il} + D_{jk} \geq D_{ij} + D_{kl}$

### 7.5. Metode temeljene na udaljenostima između taksona

#### 7.5.1. UPGMA

UPGMA metoda (*Unweighted Pair Group Method with Arithmetic Mean*) (Sokal & Michener, 1958) je metoda koja se oslanja na prepostavku molekularnog sata, odnosno da je evolucijska stopa izmjene konstantna u cijelom stablu. Kao posljedica te prepostavke, udaljenost svakog lista do korijena je jednaka.

UPGMA metoda kreće od dna prema vrhu (engl. *bottom-up*) i hijerarhijski grupira taksone. Na početku se grupiraju dva najsrodnija taksona, tj. dva taksona čija je međusobna udaljenost najmanja među svim parovima taksona. Osnovna ideja ove metode je da se u

svakom koraku ili dodaje novi takson postojećoj skupini već grupiranih taksona ili se dvije postojeće skupine međusobno spajaju (algoritam je prikazan na Sl. 7.4).

Neka je s  $d_{i,j}$  označena udaljenost između taksona  $i$  i  $j$ . Tada se sličnost između promatranih *skupina* taksona A i B određuje prema njihovoj međusobnoj udaljenosti, koja se računa kao prosjek svih međusobnih udaljenosti članova skupine A i članova skupine B:

$$\frac{1}{|A||B|} \sum_{i \in A} \sum_{j \in B} d_{i,j} \quad (7.18)$$

**za** svaki takson  $i$  /\* inicijalizacija \*/

$C_i = \{i\}$  /\* inicijalno svaka skupina  $C_i$  sadrži samo  $i$ -ti takson \*/

$d(C_i, C_j) = d_{ij}$

$h(i) = 0$  /\* visina  $i$ -tog taksona u stablu \*/

**kraj**

**Ponavljati** sve dok ima taksona koji nisu dodani u stablo

Među svim skupinama pronaći  $C_i$  i  $C_j$  tako da je  $d(C_i, C_j)$  minimalna

Dodati novu skupinu  $C_k$  koja zamjenjuje  $C_i$  i  $C_j$

Dodati u stablo novi čvor  $N_{ij}$  tako da je visina  $h(N_{ij}) = d(C_i, C_j)$  /\* izraz (1) \*/

$d(C_i, N_{ij}) = h(N_{ij}) - h(C_i)$  /\* povezati  $C_i$  i  $N_{ij}$  \*/ /\* izraz (2) \*/

$d(C_j, N_{ij}) = h(N_{ij}) - h(C_j)$  /\* povezati  $C_j$  i  $N_{ij}$  \*/ /\* izraz (3) \*/

**za** sve  $C_l \neq C_k$

$d(C_k, C_l) = (|C_i| \cdot d(C_i, C_l) + |C_j| \cdot d(C_j, C_l)) / (|C_i| + |C_j|)$  /\* izraz (4) \*/

**kraj**

**kraj**

Sl. 7.4 Algoritam UPGMA

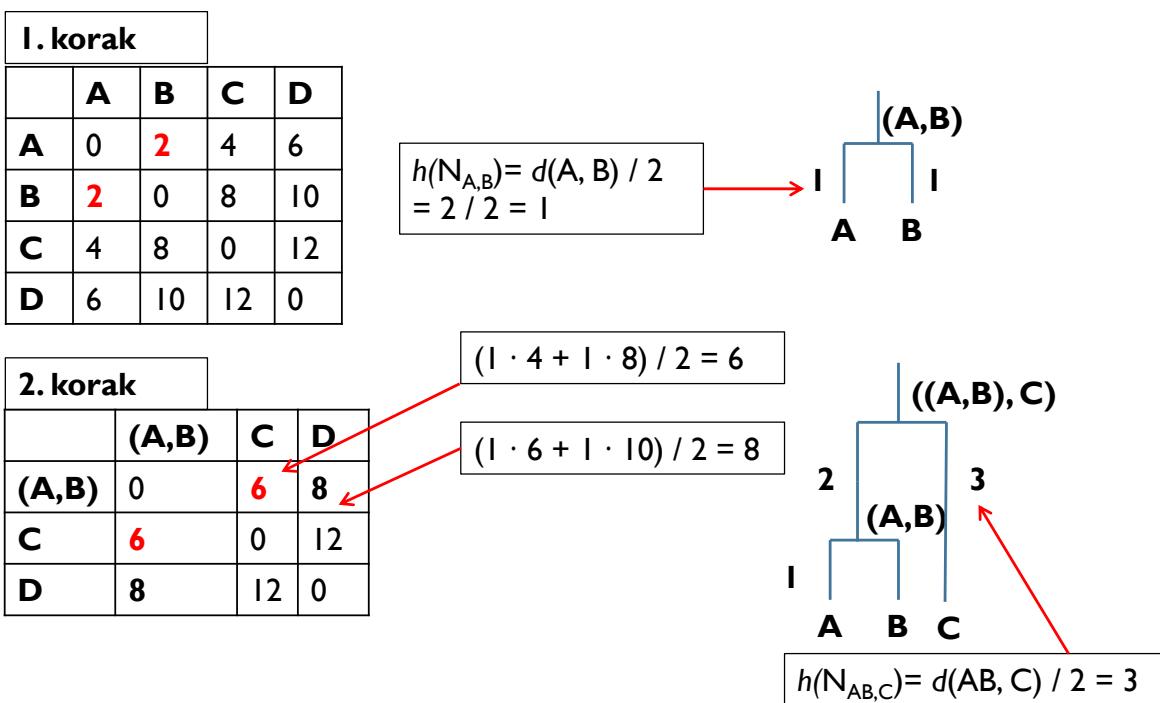
S obzirom da je ovo jedna od najjednostavnijih metoda za izgradnju filogenetskog stabla, koristi se npr. za izgradnju inicijalnog stabla (engl. *guide tree*) kao pomoć drugim složenijim i preciznijim metodama za izgradnju filogenetskog stabla ili za određivanje poravnanja više sljedova odjednom.

Vremenska složenost UPGMA algoritma je  $O(n^2)$  za  $n$  taksona.

## Primjer

Neka je zadan skup taksona A, B, C i D te matrica udaljenosti kao na Sl. 7.5. U prvoj koraku vidimo da je, među svim mogućim parovima taksona, najmanja udaljenost između taksona A i B, pa se oni prvi grupiraju zajedno. Kao rezultat dobivamo stablo s listovima A i B te čvorom (A, B), koji predstavlja zajedničkog pretka. Prema izrazima (1)-(3) iz UPGMA algoritma (Sl. 7.4) računamo visinu čvora (A, B) te udaljenosti od A i B do čvora (A, B), koje iznose polovicu njihove međusobne udaljenosti, odnosno  $d(A, B) / 2 = 1$ . Oba lista su jednakoudaljena od čvora (A, B), jer UPGMA koristi prepostavku molekularnog sata da je evolucijska stopa jednaka u vremenu i u svim granama, pa će u konačnom rezultatu udaljenost od svakog lista do korijena biti jednakna.

U 2. koraku, krećemo od matrice koja sada ima tri taksona (A, B), C i D, gdje se A i B promatraju zajedno kao jedna skupina (ili grozd; engl. *cluster*). Udaljenosti od čvorova C i D do novog čvora ((A, B), C) su izračunate prema izrazu iz UPGMA algoritma (Sl. 7.4; izraz (4)). U ovome se koraku u stablo dodaje novi čvor ((A, B), C), jer su, prema matrici udaljenosti, najmanje udaljeni takson (A, B) i takson C.

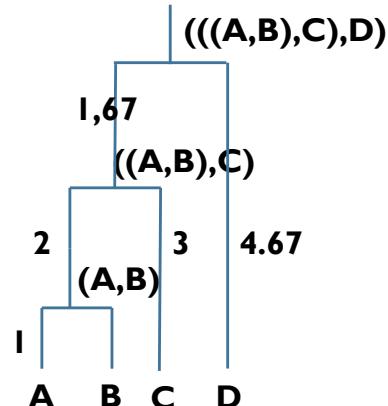


Sl. 7.5 UPGMA – primjer (1. i 2. korak)

U posljednjem, 3. koraku, (Sl. 7.6) u stablo se dodaje preostali takson (D) te korijen stabla. Kao što i očekujemo za UPGMA, svi listovi su jednakoudaljni od korijena (ta udaljenost iznosi 4.67).

3. korak		
	((A,B),C)	D
((A,B),C)	0	9,33
D	9,33	0

$$(2 \cdot 8 + 1 \cdot 12) / (2 + 1) = 9,33$$



Sl. 7.6 UPGMA – primjer (3. korak)

### 7.5.2. Metoda povezivanja susjeda

Metoda povezivanja najbližih susjeda (engl. *Neigbor-Joining*) (Saitou & Nei, 1987) razlikuje se od UPGMA po tome što osim topologije, određuje i duljinu grana filogenetskog stabla. U ovoj se metodi polazi od ideje da se u svakom koraku minimizira zbroj duljina grana, no konačan rezultat ne mora biti minimalan zbroj duljina svih grana.

Postupak kreće s inicijalnim stablom koje uključuje svih  $n$  taksona. Da bi se pronašla dva najbliža taksona (npr. A i B) potrebno je napraviti  $n(n-1)/2$  usporedbi. Ta će se dva taksona u dalnjim koracima algoritma tretirati kao jedan takson. Odabir susjednih (najbližih) taksona koji će biti spojeni, ovisi o mjeri udaljenosti  $M$  (Studier and Keppler, 1988), gdje se  $M$  računa iz inicijalnih udaljenosti taksona i njihovih prosječnih udaljenosti od ostalih taksona (izrazi (7.19) i (7.20)).

Nakon što su odabrani najbliži taksoni A i B, u stablo se dodaje novi čvor koji povezuje taj par: (A, B). Zatim se određuju udaljenost taksona A i B do novog čvora (A, B) te udaljenost svih ostalih čvorova X,  $X \neq A, B$ , do novog čvora (A, B). Postupak se ponavlja sve dok se ne dodaju svi taksoni, odnosno, dok se ne odrede sve grane (dok  $n$  ne postane 2). Vremenska složenost metode je  $O(n^3)$  za  $n$  taksona (Studier and Keppler, 1988).

Saitou i Nei su definirali susjedne (najbliže) taksone kao dva taksona za koja je minimalan zbroj svih duljina grana u stablu, tj. ako su  $i$  i  $j$  susjedni taksoni u stablu, onda je suma svih duljina grana u stablu manja u odnosu na bilo koju drugu kombinaciju odabranih taksona.

Ekvivalent toj mjeri je mjera  $M$ , koja se zbog bržeg računanja koristi umjesto originalnog minimalnog zbroja svih duljina grana u stablu (Studier and Keppler, 1988).

Neka je  $D_{ij}$  originalna udaljenost između taksona  $i$  i  $j$ . Neka je  $S_i$  zbroj udaljenosti čvora  $i$  do ostalih čvorova (analogno se određuje i  $S_j$ ). Prosječnu "korigiranu" udaljenost čvora  $i$  do ostalih čvorova dobijemo dijeljenjem  $S_i$  s  $n-2$  (7.19).

Originalne udaljenosti  $D_{ij}$  pretvaramo u udaljenosti za izgradnju filogenetskog stabla,  $M_{ij}$  (7.20). Mjera  $M_{ij}$  služe kako bi u svakom koraku izgradnje filogenetskog stabla (Sl. 7.8) izabrali susjedne taksone, tj. taksone koje ćemo spojiti (izrazi (7.19) i (7.20)). Odabiru se oni taksoni  $i$  i  $j$  za koje mjera  $M_{ij}$  ima minimalnu vrijednost. Ako ima više jednakih minimalnih  $M_{ij}$ , tada proizvoljno odabiremo bilo koji od njih.

$$S_i = \sum_{k=1}^n D_{ik} \quad (7.19)$$

$$M_{ij} = D_{ij} - \frac{1}{n-2} (S_i + S_j) \quad (7.20)$$

Ako su prema mjeri  $M$  najbliži taksoni A i B (tj.  $M_{AB}$  je minimalna vrijednost), onda njih spajamo i zamjenjujemo novim čvorom X. Nakon toga za A i B računamo udaljenosti do novog čvora X (7.21) te također nove udaljenosti svih ostalih taksona Y,  $Y \neq A, B$ , do X (7.22).

$$D_{AX} = \frac{1}{2} D_{AB} + \frac{1}{2(n-2)} (S_A - S_B) \quad (7.21)$$

$$D_{XY} = \frac{1}{2} (D_{AY} - D_{XA}) + \frac{1}{2} (D_{BY} - D_{XB}) = \frac{1}{2} (D_{AY} + D_{BY} - D_{AB}) \quad (7.22)$$

### Primjer

Neka je zadan skup taksona A, B, C i D ( $n = 4$ ) te njihove međusobne udaljenosti u matrici udaljenosti  $D$ , kao na Sl. 7.7 u 1. koraku. Kako bi se dobole vrijednosti koje predstavljaju elemente matrice  $M$ , prvo se izračunaju sume  $S_i$  (7.19), te zatim  $M_{ij}$  vrijednosti (Sl. 7.7, 1. korak). Između  $M_{ij}$  vrijednosti, odabire se najmanja. S obzirom da i  $M_{AD}$  i  $M_{BC}$  imaju

jednaku minimalnu vrijednost -19, proizvoljno odabiremo  $M_{AD}$ , što znači da smo u 1. koraku odabrali taksone A i D kao najbliže susjede, koje ćemo zamijeniti novim čvorom X.

U 2. koraku određujemo udaljenosti taksona A i D do zamjenskog čvora X (izraz (7.21)) te udaljenosti preostalih čvorova B i C do novog čvora X (izraz (7.22)) te s tim vrijednostima oblikujemo novu matricu udaljenosti koja sada ima tri taksona (tj.  $n' = 3$ ).

I. korak				
	A	B	C	D
A	0	6	10	6
B	6	0	8	10
C	10	8	0	12
D	6	10	12	0

$$n = 4, S_A = 22, S_B = 24, S_C = 30, S_D = 28$$

$$M_{AB} = D_{AB} - (S_A + S_B) / 2 = 6 - 23 = -17$$

$$M_{AC} = -16, M_{AD} = -19 \rightarrow \text{odaberemo npr. } M_{AD}$$

$$M_{BC} = -19, M_{BD} = -16$$

$$M_{CD} = -17$$

2. korak		
	X	B
X	0	5
B	5	0
C	8	8
		0

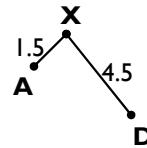
Uvodimo novi čvor X:

$$D_{AX} = D_{AD} / 2 + (S_A - S_D) / 4 = 1.5$$

$$D_{DX} = D_{AD} / 2 + (S_D - S_A) / 4 = 4.5$$

$$D_{CX} = (D_{AC} + D_{DC} - D_{XA} - D_{XD}) / 2 \\ = (10 + 12 - 1.5 - 4.5) / 2 = 8$$

$$D_{BX} = (D_{AB} + D_{DB} - D_{XA} - D_{XB}) / 2 \\ = (6 + 10 - 1.5 - 4.5) / 2 = 5$$



Sl. 7.7 Metoda povezivanja susjeda – primjer (1. i 2. korak)

U 3. koraku računamo nove  $M_{ij}$  vrijednosti, uz smanjeni broj taksona:  $n' = 3$ . S obzirom da su u ovom koraku sve  $M_{ij}$  vrijednosti jednake, proizvoljno odabiremo čvorove X i B koje ćemo u ovom koraku spojiti i zamijeniti novim čvorom Y (Sl. 7.8; 3. korak). U 4. koraku računamo udaljenost od čvorova X i B do novog čvora Y (izraz (7.21)) te udaljenost preostalog čvora C do čvora Y (izraz (7.22)). Time su određene duljine svih grana u stablu i broj promatranih taksona se smanjio na  $n'' = 2$ , čime završavamo postupak. Konačno stablo prikazano je na Sl. 7.8 (4. korak) i u stablu su ucrtane duljine grana. Ovakvo stablo je neukorijenjeno, odnosno ne poznajemo tijek vremena, nego samo topološke odnose između taksona.

3. korak			
	X	B	C
X	0	5	8
B	5	0	8
C	8	8	0

$$n' = 3, S_B = 13, S_C = 16, S_X = 13$$

$$M_{XB} = D_{XB} - (S_X + S_B) / l = 5 - (13 + 13) / l = -21$$

$$M_{XC} = D_{XC} - (S_X + S_C) / l = 8 - (13 + 16) / l = -21$$

$$M_{BC} = D_{BC} - (S_B + S_C) / l = 8 - (13 + 16) / l = -21$$

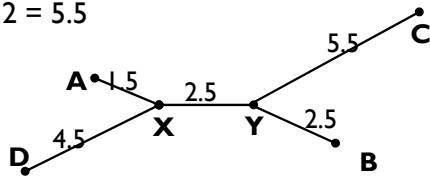
Uvodimo novi čvor Y:

4. korak		
	Y	C
Y	0	5.5
C	5.5	0

$$D_{XY} = D_{BX} / 2 + (S_X - S_B) / 2 = 2.5$$

$$D_{BY} = D_{BX} / 2 + (S_B - S_X) / 2 = 2.5$$

$$D_{CY} = (D_{BC} + D_{XC} - D_{BX}) / 2 = 5.5$$



Sl. 7.8 Metoda povezivanja susjeda – primjer (3. i 4. korak)

## 7.6. Metode temeljene na obilježjima

Prednost metoda temeljenih na obilježjima i statističkih metoda (metoda najmanjeg broja evolucijskih promjena, metoda najveće izglednosti i Bayesova metoda) u odnosu na metode temeljene na udaljenostima su značajno bolji rezultati, tj. rezultiraju u točnijim filogenetskim stablima i za DNA i za proteinske sljedove, dok su metode temeljene na udaljenostima značajno brže i memorijski manje zahtjevne (Hall, 2004).

### 7.6.1. Metoda najmanjeg broja evolucijskih promjena

Metoda najmanjeg broja evolucijskih promjena (mutacija) (engl. *maximum parsimony*) slijedi princip Occamove oštice (engl. *Occam's razor's principle*), tj. najjednostavnije rješenje je najvjerojatnije i ispravno.

U ovom je postupku cilj pronaći filogenetsko stablo kojime se mogu opisati promatrani sljedovi tako da izgrađeno stablo odražava minimalan broj evolucijskih promjena. S obzirom da je riječ o NP-problemu, ovaj postupak nije prikladan za velike skupove podataka. Također, ovaj postupak ne generira eksplicitne mјere udaljenosti među taksonima. Međutim, pokazano je da za proteinske sljedove ova metoda daje najbolje rezultate (tj. rezultira u najtočnijim filogenetskim stablima) među svim metodama (Hall,

2004) (za DNA sljedove bolje rezultate daju metoda najveće izglednosti i Bayesova metoda, koje su nešto sporije).

Osnovna ideja metode najmanjeg broja evolucijskih promjena je za zadani skup taksona (sljedova) izgraditi sva moguća filogenetska stabla te pronaći ono koje uključuje najmanji broj evolucijskih promjena. Zadani skup čine poravnati sljedovi, tj. matrica  $n \times m$ , gdje je  $n$  broj sljedova, a  $m$  duljina poravnatih sljedova. Za svaki promatrani stupac matrice, koja predstavlja poravnanje, potrebno je odabratи ono stablo koje uključuje minimalan broj promjena. Nakon što se za svaki stupac iz matrice poravnanja pronađe najbolje stablo, za konačan rezultat odabire se stablo za koje je ukupno bilo najmanje promjena po svim stupcima zajedno.

Kako bi se ubrzao postupak, u poravnanju se promatraju samo tzv. *informativni stupci* (engl. *informative site*), tj. oni za koje vrijede sljedeća dva uvjeta:

1. U tom stupcu su barem 2 tipa nukleotida.
2. Svaki od tipova nukleotida, koji se pojavljuje u stupcu, zastavljen je u barem 2 slijeda u poravnanju.

### Primjer

Zadana su 4 slijeda:  $S_1$ ,  $S_2$ ,  $S_3$  i  $S_4$  te njihovo poravnanje (Tablica 7.1). Stupci 1 i 3 su informativni, jer se u oba stupca pojavljuju po 2 tipa nukleotida (A i C) – svaki u dva slijeda. Na primjer, u prvome stupcu nukleotid A pojavljuje se u sljedovima  $S_1$  i  $S_2$ , a nukleotid C u  $S_3$  i  $S_4$ .

Stupac 4 nije informativan, jer su svi nukleotidi u stupcu isti, tj. imaju vrijednost C.

Stupac 2 nije informativan, jer se nukleotid A nalazi na tome mjestu u slijedu  $S_1$ , a u svim drugim sljedovima se nalazi nukleotid C. S obzirom da se koristi prepostavka da je najvjerojatniji događaj onaj koji ima najmanje promjena (tj. mutacija), ovdje možemo prepostaviti da je u slijedu  $S_1$  došlo do mutacije nukleotida C u A, pa bi svako od tri moguća stabla uključivalo promjenu na tom mjestu, što znači da nam takva promjena ne bi donijela nikakvu novu informaciju koje bi stablo bilo vjerojatnije.

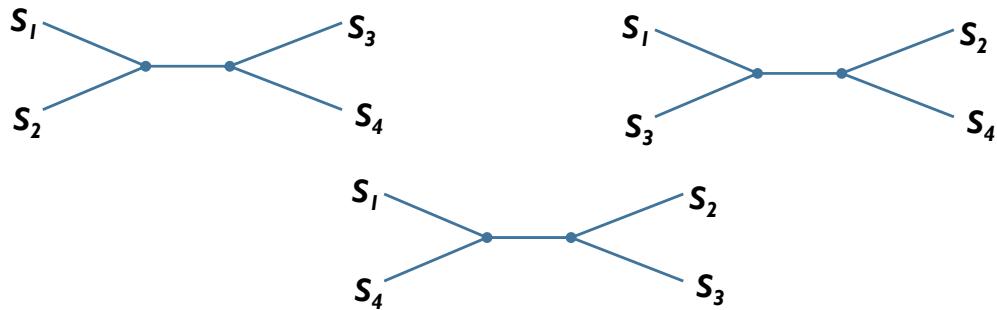
Tablica 7.1 Poravnanje sljedova  $S_1$ ,  $S_2$ ,  $S_3$  i  $S_4$ .

	1	2	3	4
--	---	---	---	---

$S_1$	A	A	C	C
$S_2$	A	C	A	C
$S_3$	C	C	A	C
$S_4$	C	C	C	C

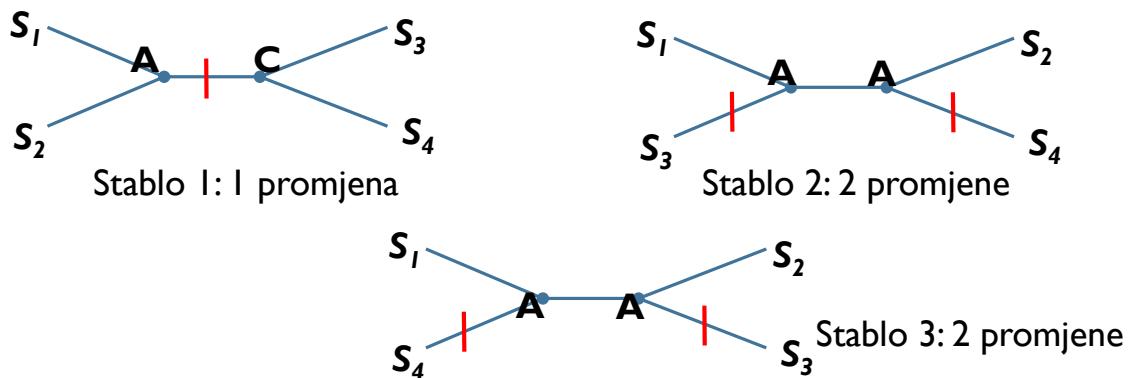
### Primjer

Zadana su 4 slijeda:  $S_1$ ,  $S_2$ ,  $S_3$  i  $S_4$  i njihovo poravnanje (Tablica 7.1). Za 4 zadana slijeda moguća su 3 neukorijenjena filogenetska stabla (Sl. 7.9). Svako od mogućih stabala promatrati ćemo za stupce 1, 2 i 3. Stupce 1 i 3 ćemo promatrati zato što su informativni, a stupac 2 kako bi prikazali da neinformativni stupci ne nose nikakvu informaciju.



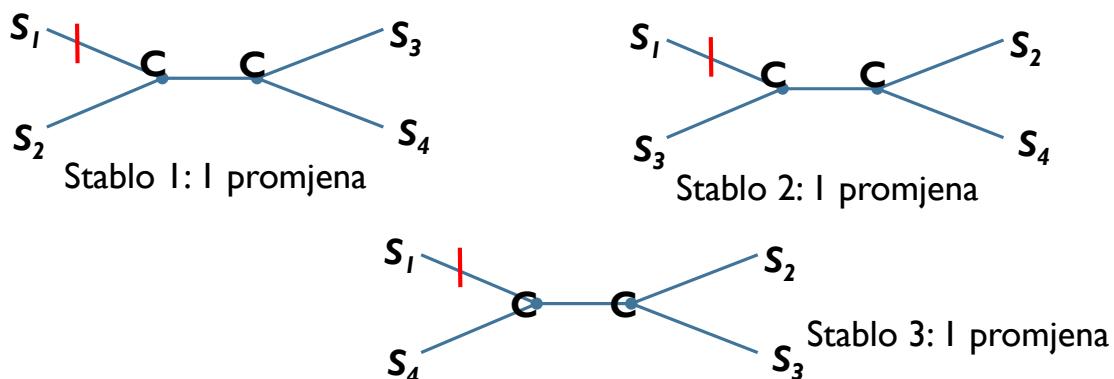
Sl. 7.9 Tri moguća neukorijenjena stabla za 4 slijeda:  $S_1$ ,  $S_2$ ,  $S_3$  i  $S_4$ .

Krećemo s 1. stupcem i promatramo minimalan broj promjena za svako od mogućih stabala. Kao što je vidljivo na Sl. 7.10, stablo 1 možemo konstruirati samo s jednom promjenom. U čvorovima su označeni nukleotidi koji su pretpostavljeni predci nukleotida u listovima tako da broj promjena bude minimalan. Npr. za stablo 1 u čvoru koji spaja sljedove  $S_1$  i  $S_2$  pretpostavljamo da se nalazi nukleotid A, jer time bi dobili najmanji broj promjena (preciznije: ne bi se dogodila niti jedna promjena s obzirom da je nukleotid A u 1. stupcu u sljedovima  $S_1$  i  $S_2$ ). Analogno tome, pretpostavljene minimalne promjene označene su u stablima 2 i 3 te pretpostavljene vrijednosti nukleotida u unutarnjim čvorovima. Vidljivo je (Sl. 7.10), dakle, da stablo 1 objašnjava stupac 1 s najmanjim brojem promjena (samo s jednom promjenom), dok stabla 2 i 3 imaju po 2 promjene.

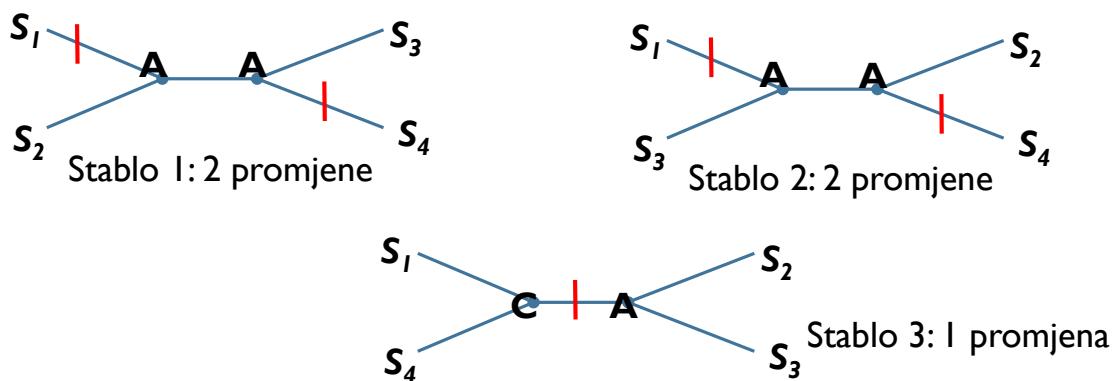


Sl. 7.10 Broj promjena za svako od tri moguća stabla za 1. stupac u poravnaju

Analogno 1. stupcu, promatramo sva tri moguća stabla i za 2. i za 3. stupac (Sl. 7.11; Sl. 7.12).



Sl. 7.11 Broj promjena za svako od tri moguća stabla za 2. stupac u poravnaju (ovaj stupac nije informativan, jer sva tri stabla imaju jednak broj promjena)



Sl. 7.12 Broj promjena za svako od tri moguća stabla za 3. stupac u poravnaju

Za 3. stupac vidimo na Sl. 7.12 da stabla 1 i 2 sadrže dvije promjene, a stablo 3 samo jednu promjenu. Dakle, 3. stupac u poravnanju najbolje je opisan stablom 3.

Ako sada prebrojimo sve promjene po informativnim stupcima (1. i 3. stupac) za sva moguća stabla, vidimo da stabla 1 i 3 uključuju ukupno 3 promjene, a stablo 2 uključuje ukupno 4 promjene. Iz toga zaključujemo da metodom najmanjeg broja evolucijskih promjena stabla 1 i 3 daju najbolja rješenja.

## 7.7. Literatura

- Cavalli-Sforza,L.L. and Edwards,A.W. (1967) Phylogenetic analysis. Models and estimation procedures. *Am. J. Hum. Genet.*, **19**, 233–257.
- Darwin,C. (1859) On the Origin of Species by Means of Natural Selection, Or, The Preservation of Favoured Races in the Struggle for Life J. Murray.
- Darwin,C. and Wallace,A. (1858) On the Tendency of Species to form Varieties; and on the Perpetuation of Varieties and Species by Natural Means of Selection. *J. Proc. Linn. Soc. Lond. Zool.*, **3**, 45–62.
- Dayrat,B. (2003) The Roots of Phylogeny: How Did Haeckel Build His Trees? *Syst. Biol.*, **52**, 515–527.
- Haeckel,E. (1866) Generelle morphologie der organismen ... Georg Reimer.
- Hall,B.G. (2004) Comparison of the Accuracies of Several Phylogenetic Methods Using Protein and DNA Sequences. *Mol. Biol. Evol.*, **22**, 792–802.
- Jukes,T. and Cantor,C. (1969) Evolution of protein molecules. In, *Mammalian Protein Metabolism*. Academic Press, New York, pp. 21–132.
- Kimura,M. (1980) A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. *J. Mol. Evol.*, **16**, 111–120.
- Mendel,J. (1865) Versuche über Pflanzenhybriden. *Verhandlungen Naturforschenden Vereines Brünn*, **IV**, 3–47.
- Pevsner,J. (2009) Bioinformatics and functional genomics 2nd ed. Wiley-Blackwell, Hoboken, N.J.
- Saitou,N. and Nei,M. (1987) The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Mol. Biol. Evol.*, **4**, 406–425.
- Sokal,R. and Michener,C. (1958) A statistical method for evaluating systematic relationships. *Univ. Kansas Science Bull.*, **38**, 1409–1437.
- Studier,J.A. and Keppler,K.J. (1988) A note on the neighbor-joining algorithm of Saitou and Nei. *Mol. Biol. Evol.*, **5**, 729–731.
- Tavaré,S. (1986) Some Probabilistic and Statistical Problems in the Analysis of DNA Sequences. In, *American Mathematical Society: Lectures on Mathematics in the Life Sciences*. Amer Mathematical Society, pp. 57–86.

## 8. Sastavljanje genoma

### 8.1. Izazov sastavljanja

Zamislimo da imamo 10 primjeraka diplomskoga rada naše majke pisanoga na pisaćoj mašini i kopiranog prije 25 godina. U obitelji se to čuva kao posebna vrijednost. Međutim, mlađi brat jednoga dana, želeći isprobati novu mašinu za sjeckanje papira, svih je 10 primjeraka sasjekao u sitne trake po retcima i odnio u kontejner za stari papir. Kasnije, kad je shvatio što je učinio brzo je odjurio i pokupio sve trake koje je uspio naći i sada ih želi natrag sastaviti. Problem je što je većina traka oštećena, neke su potrgane na više dijelova, neke neprepoznatljive, a neke su zaostale u kontejneru. Je li moguće sastaviti barem jednu ispravnu kopiju te istu skenirati ili prepisati i tako sačuvati za nasljednike? Sličan je posao sastavljanje niza ljudskoga genoma od sekvenciranih podataka, s tom razlikom što bi knjiga imala oko milijun stranica. Srećom, brat može računati na vaše poznavanje algoritama.

Čitanje genoma od otkrića strukture DNA predstavlja velik izazov za znanstvenike. S vremenom su razvijeni uređaji za očitanje DNA koji su s vremenom postajali sve brži i jeftiniji, no ostao je problem da su oni ograničeni na samo kraća očitanja. Stoga u zadnjih 20 godina velik izazov predstavlja kako sastaviti ta kratka očitanja u jedinstvenu sekvencu – genom. Isto tako, gotovo se isključivo koriste metode temeljene na *shotgun* sekvenciranju cijelog genoma pri čemu nemamo nikakvu informaciju o poretku pojedinih regija. Posao sastavljanja genoma dodatno otežava činjenica da se duljine očitanja prevladavajućih uređaja druge generacije kreću od nekoliko desetaka do par stotina nukleotida. Novi uređaji treće generacije koji proizvode dulja očitanja (reda veličine nekoliko tisuća nukleotida) imaju veći postotak greške (trenutno od 15 do čak 50 %), što pak zahtijeva ponavljanje postupka očitanja i sastavljanja genoma u cilju određivanja i potvrđivanja ispravnog poretku nukleotida u DNA slijedu. Čak i za dulja očitanja s dobrom kvalitetom postupak sastavljanja ostat će zahtijevan. Uzmimo da je potrebno sastaviti dosad čovjeku nepoznat genom. Taj se postupak naziva *de novo* sastavljanje. Uzmimo, nadalje, da imamo metagenomski uzorak u kom se nalazi DNA stotina različitih genoma, od kojih mnogi mogu biti dosad čovjeku nepoznati. Sastaviti ovako dobivene genome ostat

će težak zadatak i povećanjem duljine i kvalitete očitanja. U ovom slučaju postupak će moći biti značajno olakšan jedino pronalaskom tehnologija koja će moći pročitati očitanja dugačka reda veličine milijun nukleotida ili razdvojiti genome iz uzorka.

Kao što je navedeno u poglavlju 1.2.2 cilj algoritama za sastavljanja genoma je očitanja dobivena iz sekvencera pretvoriti u izlazni niz koristeći njihova međusobna preklapanja.

Preklapanje se svodi na preklapanje sufiksa jednoga niza s prefiksom drugoga. Promotrimo primjer preklapanja nekoliko očitanje:

```
AGCTGTCTGTC  
GCTGTCTGTCGTC  
GTCTGTCGTCATCGCATT  
GTCGTCATCGCATTCTGTCT
```

U idealnom slučaju mogli bismo pokušati sastaviti genom koristeći ovakva preklapanja, Nažalost, stvari nisu toliko jednostavne. Jedna od glavnih prepreka uspješnom sastavljanju genoma je velika prisutnost dugačkih identičnih ili gotovo identičnih sljedova u genomu. Te ponavljajuće regije karakteristične su za bakterije i biljke dok kod čovjeka i ostalih sisavaca one čine glavnu komponentu genoma. S obzirom da su mnoge od tih regija dulje nego trenutne duljine očitanja, njihova ispravna sastavljanje je otežano - njihovo prisustvo vrlo često vodi k tome da su te regije krivo sastavljene ili uopće ne mogu biti sastavljene, što pak dovodi do stvaranja procijepa u rezultantom slijedu. Druga su prepreka greške očitanja koje nastaju zbog brzine kojom novi uređaji identificiraju nukleotide u očitanjima. Alati za sastavljanje moraju dozvoljavati određenu razinu greške u očitanjima. Međutim, te greške mogu uzrokovati lažno pozitivna preklapanja pojedinih očitanja. Time nastaju tzv. kimerna sastavljanja kod kojeg su dva dijela sastavljenog genoma spojena makar su u prirodi međusobno jako udaljeni. Pojam kimerno odnosno kimera se često koristi u biologiji. Kimera (prema grč. Χίμαιρα: Himera), u biologiji, jedinka sastavljena od dijelova dviju ili više različitih rasa. Jedan od najpoznatijih primjera kimere je sfinga.

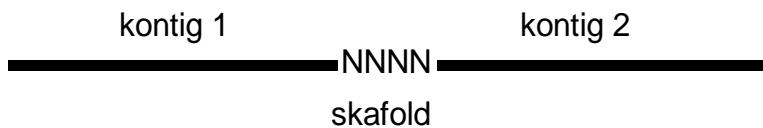
Zbog jednostavnosti većina alata za sastavljanje genoma koriste kraće podnizove očitanja koji se nazivaju *k-torke*. *K-torka* je niz od  $k$  očitanih nukleotida pri čemu je  $k$  bilo koji pozitivni cijeli broj. U preklapanjima se tada gleda dijele li dva očitanja *k-torke*, umjesto vremenski zahtjevnijeg preklapanja cijelih očitanja. Vjerojatnost da se pravo preklapanje prostire kroz dijeljene *k-torke* ovisi o vrijednosti  $k$ , duljini preklapanja i postotku pogreške očitanja. Prikladna vrijednost  $k$  treba biti dovoljno velika da eliminira slučajna preklapanja

nastala time što za dovoljno mali  $k$  raste vjerojatnost da će dva slučajno odabrana očitanja dijeliti dotičnu  $k$ -torku. Ipak, vrijednost  $k$  treba biti dovoljno mala da većina stvarnih preklapanja dijele  $k$ -torke, bez obzira na greške očitanja. Izbor  $k$  treba biti robustan na varijacije u pokrivanosti očitanja i točnosti.

Za sastavljanje genoma postoje dva osnovna pristupa. Prvi pristup koristimo kada radimo ponovno sekvenciranja genoma u slučaju da je genom te vrste već određen. Takav genom koji je obično nastao sastavljanjem očitanja više jedinki iste vrste nazivamo referentni genom. U ovom pristupu koristimo se činjenicom da se dva genoma iste vrste razlikuju u malom postotku nukleotida te je dovoljno da mapiramo nova očitanja na referentni genom. Druga metoda sastavljanja genoma je sastavljanje genoma kada referentni genom nije poznat. Ta metoda značajno je teža i naziva se *de novo* sastavljanje.

## 8.2. Osnove sastavljanja

U idealnom slučaju, sastavljeni genom bi imao jedan slijed bez dijelova nepoznatog sadržaja ili čak duljine. Nažalost, u praksi to često nije slučaj, a sastavljeni je genom predstavljen hijerarhijskom strukturom podataka koja mapira očitanja u prepostavljenu rekonstrukciju ciljanog genoma. U toj strukturi osnova su očitanja koja se grupiraju u kontige (engl. *contig*), a kotinzi u skafolde (engl. *scaffold*). Kontizi se sastoje od niza uzastopnih očitanja koja se preklapaju. Skafoldi se sastoje od više kontiga (među kojima često ima praznina) i definiraju njihov poredak, orientaciju i veličinu procijepa među njima. Alati za sastavljanje genoma najčešće na izlazu osim skafolda daju dodatno i skup očitanja koja se ne nalaze u sastavljenom genomu, te ona koja se nalaze samo djelomično. Najčešći izlazni format podataka je FASTA detaljnije pojašnjen u poglavљu 1.3.1. Osim A, C, T, G u izlaznim podacima nalazimo i druge specijalne znakove od kojih su najčešći “-“ (znak za prazninu) i N (u slučaju kada ne možemo odrediti o kojem je nukleotidu riječ). Praznine mogu predstavljati dodatne nukleotide koji se ne pojavljuju u konsenzusu poravnjanja, jer su prisutni tek u manjini očitanja, a uzastopni N-ovi obično predstavljaju procjepu između kontiga, te njihov broj predstavlja očekivanu duljinu procjepa.



Sl. 8.1 Primjer skafolda. U ovom slučaju skafold se sastoji od dva kontiga od kojih se kontig 1 nalazi ispred kontiga 2, a između njih je procijep duljine 4 znaka

Kvaliteta sastavljenog genoma se mjeri veličinom i točnošću njegovih kontiga i skafolda. Veličina sastavljenog genoma obično se definira raznim statistikama poput maksimalne i prosječne duljine, , kombinirane totalne duljina te mjere zvane N50. Da bismo izračunali posljednju mjeru prvo moramo sortirati kontige po duljini. Tako poredane kontige možemo podijeliti u dva skupa pri čemu su u prvom skupu svi kontizi dulji od neke granične duljine, a u drugom onim kraći. Pretpostavimo da je kontig granične duljine u skupu duljih kontiga i da taj kontig nazivamo granični kontig. Ako je zbroj duljina svih kontiga u duljem skupu veći ili jednak od 50 % duljine sastavljenoga genoma onda je mjera N50 duljina najduljeg graničnog kontiga za koji navedeni zahtjev vrijedi. Na primjer ako imamo duljine pojedinih kontiga 10, 8, 6, 5, 3, 3, 2, 1, 1, 1 onda je  $N50$  vrijednost jednaka 6. Zbrajanjem svih kontiga koji su dugački 6 i više dobijemo  $(10+8+6) = 24$ , što je više od 50% od ukupne duljine  $(10+8+6+5+3+3+2+1+1+1) = 40$ . Važno je primjetiti da  $N50$  kao mjeru usporedbe ima smisla koristiti jedino u slučaju kada raznim metodama pokušamo sastaviti isti genom, a referentni genom nam nije poznat. Općenito, najbolja ocjena kvalitete dobiva se poravnanjem sastavljenoga genoma s referentnim.

## 8.3. Algoritmi za sastavljanje

### 8.3.1. Traženje najkraćeg zajedničkog nadniza

Zamislimo da imamo originalni niz  $s=TATACATTAG$  i da iz njega dobijemo očitanja  $\{ACA, ATA, ATT, CAT, TAC, TAG, TAT, TTA\}$ . Očitanja smo naveli u abecednom poretku jer kod sekvenciranja nemamo informaciju o poretku. Kako iz ovih očitanja sastaviti početni niz? Očito je da ćemo koristiti sufikse i prefikse te gledati njihova preklapanja i tako progresivno pokušati doći do početnoga niza. Ovakav se problem naziva nalaženje najkraćega zajedničkog nadniza (engl. *Shortest common superstring – SCS*). Najkraći

zajednički nadniz je onaj niz kojem su sva očitanja podnizovi. Važno je napomenuti da pronađeni nadniz  $s'$  ne mora odgovarati početnom nizu  $s$ .

Pronalaženje nadniza svih podnizova nije teško, jer ih možemo sve jednostavno povezati jedan iza drugoga. Ono što je problematično je pronalazak najkraćeg takvoga niza. U stvari, pronalazak najkraćega zajedničkog nadniza je NP-težak problem i time za veći broj nizova predstavlja problem koje nije optimalno moguće riješiti u dostupnom vremenu. Stoga se koriste heuristički pristupi pri čemu dominiraju pohlepni (engl. *greedy*) pristupi.

Formuliramo problem tako da je za danu kolekciju od  $n$  nizova  $x^1, x^2, \dots, x^n$  nad skupom  $\Sigma$ , potrebno naći najkraći niz  $z$  takav da se svaki  $x^i$  u njem pojavljuje kao podniz (Frieze and Szpankowski, 1998). Uvezši u obzir greške sekvenciranja zbog kojih će se početni nizovi razlikovati od izvornog niza, u bioinformatici se obično koristi formulacija koju nazivamo približni SCS (engl. *shortest common superstring*). SCS sadrži nizove približno jednake originalnim nizovima (npr. u smislu Hammingove udaljenosti) kao podnizove. Radi jednostavnosti ćemo za sada koristiti prvu definiciju. Pohlepni algoritmi proizvode ukupno preklapanje SCS-a koje nazivamo  $O_n^{gr}$  i koje se obično razlikuje od optimalnoga poravnanja  $O_n^{opt}$ .

Prepostavimo da su  $s = s_1 s_2 \dots s_p$  i  $t = t_1 t_2 \dots t_r$  nizovi nad istom konačnom abecedom  $\Sigma = (\omega_1, \omega_2, \dots, \omega_K)$  gdje je  $K = |\Sigma|$  veličina abecede. Isto tako pišemo  $|x|$  za duljinu  $x$ . Definiramo preklapanje  $o(s, t)$  kao:

$$o(s, t) = \max\{j : t_i = s_{p-j+i}, 1 \leq i \leq j\}$$

Ako je  $s \neq t$  i  $l = o(s, t)$ , onda

$$s \oplus t = s_1 s_2 \dots s_p t_{l+1} t_{l+2} \dots t_r.$$

Ako je  $S$  skup svih nadnizova izgrađenih nad nizovima  $x^1, x^2, \dots, x^n$ . Tada,

$$O_n^{opt} = \sum_{i=1}^n |x^i| - \min_{z \in S} |z| \quad (8.1)$$

Osnovni algoritam za nalaženje SCS-a je

```

GREEDYSCS
1    $I \leftarrow \{x^1, x^2, \dots, x^n\}; O_n^{gr} \leftarrow 0;$ 
2   repeat
3       choose  $s, t \in I$  ( $s \neq t$ ) in order to maximise  $o(s, t)$ ;  $z = s \oplus t$ ;

```

```

4       $I \leftarrow I \setminus \{s, t\} \cup \{z\};$ 
5       $O_n^{gr} \leftarrow O_n^{gr} + o(s, t);$ 
6  until  $|I| = 1$ 

```

Kôd 8.1 Pseudokod pohlepnog traženja najduljeg zajedničkog nadniza

Ako se vratimo na početni niz  $s=TATACATTAG$  i njegova očitanja  $\{ACA, ATA, ATT, CAT, TAC, TAG, TAT, TTA\}$  i pokušamo koristeći algoritam iz Kôd 8.1 dobiti SCS:

```

ACA, ATA, ATT, CAT, TAC, TAG, TAT, TTA
ACA, ATA, ATT, CAT, TAC, TAG, TAT, TTA
ACAT, ATA, ATT, TAC, TAG, TAT, TTA
ACAT, ATAC, ATT, TAG, TAT, TTA
ACAT, ATAC, ATTA, TAG, TAT
ACAT, ATAC, ATTAG, TAT
ACAT, TATAC, ATTAG
ACATTAG, TATAC
TATACATTAG

```

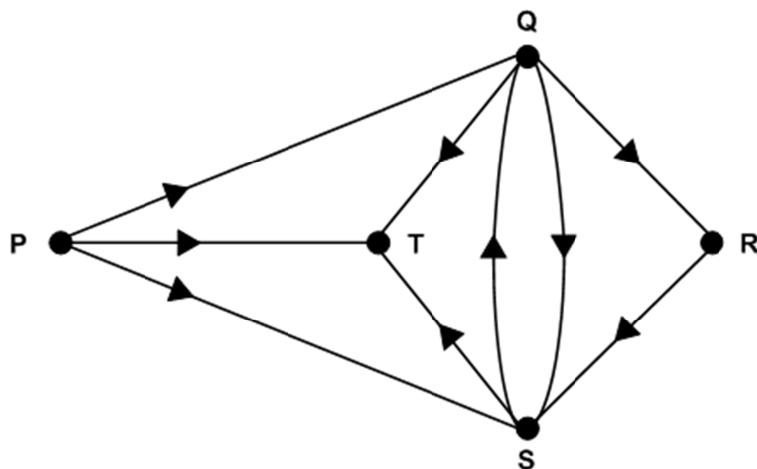
Dobijemo točan niz, no može se primijetiti da smo mogli dobiti krivo rješenje da samo u četvrtom koraku ujedinili nizove  $ACAT$  i  $ATAC$  i dobili bi  $ACATAC$  koji ne postoji u originalnom nizu i vrlo vjerojatno bi ostali bez preklapajućih nizova prije završetka. U tom slučaju bi se možda mogli vratiti korak unatrag i pokušati popraviti ili unaprijed pokušati predvidjeti ujedinjenja koja nas dovode u situaciju da nema daljih preklapanja. U ovom slučaju smo imali samo 8 početnih nizova i lako bi mogli dobiti na taj način točno rješenje, no za velik broj nizova bi to bilo nemoguće. Osim mogućih pogrešnih spajanja, sekvenciranja možemo očekivati greške u očitanjima, kao i to da će u nekim regijama očitanja nedostajati.

Prvi algoritmi za sastavljanje genoma bili su temeljeni na traženju SCS-a skupa očitanja unutar pogreške  $\varepsilon$ . Formalnije, tražimo najkraći niz  $R$  takav da za svako očitanje  $f_i$ , postoji takav podniz  $R[sp_i, ep_i]$  koji nije dulji za više od  $\varepsilon/f_i$  razlika od ili  $f_i$  ili  $f_i^c$ . Osnovni problem sa SCS algoritmima je u tome što u praksi daju loše rezultate s ponavljajućim regijama – i to na način da rezultirajući niz bude komprimirana, kraća verzija polaznog genoma. Prekomjerna kompresija može se primijetiti u područjima gdje ponavljajuće regije imaju neuobičajeno veliku pokrivenost u izlaznom rješenju. Ovo sugerira da se to može izbjegći formulirajući problem sastavljanja očitanja na način da se traži slijed koji maksimizira izglednost hipoteze da su njena očitanja uzorkovana duljinom početnoga slijeda s danom distribucijom.

### 8.3.2. Algoritmi nad grafovima

Moderni algoritmi za sastavljanje genoma temelje se na algoritmima nad grafovima. Zavisno o načinu na koji definiramo graf razlikujemo dvije osnovne metode: Preklapanje-Razmještaj-Konsenzus (engl. *Overlap-Layout-Consensus*, OLC) metode temeljene na grafu preklapanja i metode temeljene na *de Bruijn* grafovima.

Osnovu grafa čine vrhovi i bridovi koji povezuju vrhove. Vrhovima i bridovima možemo pridružiti različite atributе i semantike. U slučaju da se bridovima može proći samo u jednom smjeru takvi vrhovi i graf se nazivaju *usmjereni*, a strelicama prikazujemo smjer grafa. Sl. 8.2 prikazuje primjer usmjerenoga grafa s pet vrhova i bridovima između njih. Svaki usmjereni brid predstavlja vezu između izvorišnog i odredišnog vrha. Bridovi mogu imati i različite težine pa zavisno o tome imamo netežinske i težinske grafove. Skupina bridova tvori *šetnje* koje posjećuju vrhove u nekom redoslijedu tako da odredišni vrh jednoga brida tvori izvorište za sljedeće vrhove. Primjer šetnje je šetnja od vrha  $P$  do vrha  $R$ :  $(P,Q),(Q,R)$ . Isto tako možemo primijetiti da nema puta od vrha  $R$  do vrha  $P$ . Staza je šetnja pri čemu je svaki brid u šetnji posjećen samo jedanput. Primjer staze je:  $(P,Q),(Q,R),(R,S),(S,Q),(Q,S),(S,T)$ . Put je šetnja gdje je svaki vrh koji je dio šetnje posjećen samo jedanput.

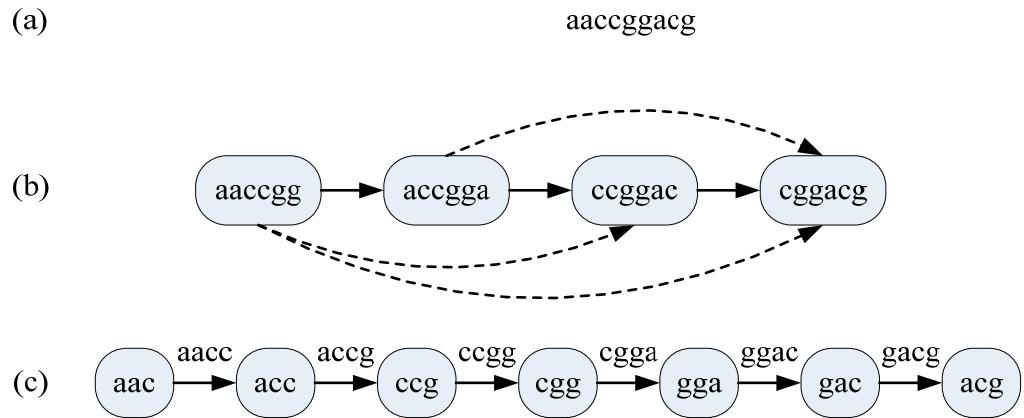


Sl. 8.2 Primjer grafa. Graf g s vrhovima (čvorovima)  $V:\{P,T,Q,S,R\}$  i bridovima

$$E:\{(P,T),(P,Q),(P,S),(Q,T),(S,T),(Q,S),(S,Q),(Q,R),(R,S)\}$$

Zatvorena staza pozitivne duljine čiji su vrhovi (osim krajeva) međusobno različiti zove se *ciklus*. Broj bridova kojima je povezan vrh naziva se *stupanj vrha*. Ukoliko su bridovi usmjereni onda razlikujemo *ulazni i izlazni stupanj*.

Za konstrukciju grafa od sekvenciranih očitanja koristimo dva pristupa (Miller *et al.*, 2010). U prvom pristupu očitanja su pridijeljena vrhovima, a ukoliko postoji preklapanje između dva vrha, između njih postoji brid. Takav graf nazivamo graf preklapanja. U drugom pristupu očitanja dijelimo na kraće  $k$ -torke i njih smještamo na bridove, a vrhovi predstavljaju prefiks i sufiks od  $k-1$  nukleotida  $k$ -torke. Na taj način radimo s  $k$ -torkama koje su obično kraće od duljine očitanja i sve su jednake duljine. Sl. 8.3 prikazuje primjere oba grafa za jedan slijed razlomljen u četiri očitanja.



Sl. 8.3 Očitanja predstavljeno grafovima. (a) Od početnoga slijeda radimo četiri očitanja duljine 6 (b) Graf preklapanja ima po jedan vrh za svako očitanje plus direktni brid za svaki par očitanja čiji se sufiks i prefiks preklapaju. S iscrtanim linijama označavamo preklapanja sadržana u drugim preklapanjima (tzv. tranzitivna preklapanja) (c) *de Buijn* graf ima brid za svaku k-torku, a u vrhovima se nalaze preklapanja od k-1 baza. U praksi se informacija u vrhovima ovoga grafa samo implicitno prepostavlja i sve je sadržano u bridu. Početna sekvenca se može lako rekonstruirati preko puta u oba grafa.

Graf preklapanja predstavlja sekvencirana očitanja i njihova preklapanja. U grafu vrhovi predstavljaju očitanja, a bridovi preklapanja. U praksi graf može imati različite elemente ili atributе za razlikovanje 5' i 3' krajeva očitanja, slijed nukleotida i njegov reverzni komplement za očitanja, duljinu očitanja i tip preklapanja (sufiks-prefiks ili jedno očitanje sadržava drugo). Putovi kroz graf su potencijalni kontizi koji mogu biti pretvoreni u sekvencu. Putovi moraju imati zrcalne slike koje predstavljaju slijed koji je reverzni komplement. Postoje dva načina za označavanje takvih grafova: sa zasebnim vrhovima za krajeve ili sa zasebnim bridovima. Ukoliko postoje zasebni vrhovi onda putovi moraju izaći na suprotnom kraju očitanja u koje su ušli. Ukoliko postoje zasebni bridovi za oba lanca, onda put mora izaći na istom lancu na kojem je i ušao.

Kod *De Bruijn* grafova koriste se  $k$ -torke očitanja fiksne duljine zbog prirode samih grafova. Najčešće se jedna  $k$ -torka predstavlja bridom dok se implicitno prepostavlja da su vrhovi koje povezuje taj brid  $k$ -1 prefiksi i  $k$ -1 sufiksi te  $k$ -torke.

## 8.4. Preklapanje-Razmještanje-Konsenzus pristup

Preklapanje-Razmještanje-Konsenzus (engl. *Overlap-Layout-Consensus*, OLC) pristup je bio tipičan za sastavljanje očitanja dobivenih uređajima temeljenim na Sangerovo tehnologiji prve generacije. Danas opet postaje popularan prije svega zbog uređaja treće generacije koji proizvode dulja očitanja. Najpoznatiji alati za sastavljanje temeljeni na ovome principu su Celera (Myers *et al.*, 2000) i SGA (Simpson and Durbin, 2012).

Kod OLC pristupa, traženje puta u grafu svodi se na traženje puta koji prolazi kroz sve vrhove u grafu točno jedanput. Pronalaskom takvoga puta cijeli slijed bi bio poznat. Ono što je osnovni problem toga pristupa je da je problem traženja takvoga puta, koji se naziva Hamiltonov put je NP potpun problem i nije ga moguće obaviti u dostupnom vremenu. Iz tog se razloga koriste heuristike kojima će se graf što je moguće više pojednostaviti i na taj način doći do slijeda.

### 8.4.1. Preklapanje

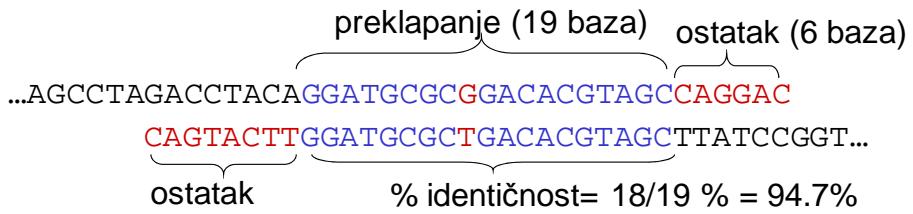
Preklapanje uključuje međusobnu usporedbu svih očitanja. Postoje dva glavna pristupa ovom problemu: koristiti dinamičko programiranje ili neku od heurističkih metoda uz „pronađi sjeme i proširi“ pristup.

U slučaju korištenja dinamičkoga programiranja poželjno je koristiti algoritam preklapanja objašnjen u poglavlju 2.4. Ono što je potrebno napraviti je za svaki par očitanja  $s$  i  $t$  naći njihovo preklapanje. Ako s  $l_i$  označimo pojedinačno očitanje, s  $L$  ukupnu duljinu genoma, složenost poravnjanja svih parova očitanja može se prikazati kao:

$$O\left(\sum_{i,j} l_i l_j\right) = O\left(\sum_i l_i \sum_j l_j\right) = O\left(\sum_i l_i L\right) = O(L^2)$$

Problem takvoga pristupa je što bi složenost takvoga pristupa bila kvadratna s duljinom genoma. Zbog sporosti dinamičkoga programiranja koriste se heurističke metode koje možemo podijeliti u dvije skupine. Prvu skupinu predstavljaju metode koji očitanja dijele u  $k$ -torke fiksne duljine, a drugu skupinu čine metode koje promatraju podnizove očitanja koji ne moraju biti fiksne duljine. Kod obje skupine princip je sličan: U prvom se koraku

napravi indeks za sve podnizove očitanja, odaberu kandidatna očitanja koja dijele podnizove i naprave poravnjana koristeći podnizove kao sjemena. Osjetljivost otkrivanja preklapanja ovisi o duljini  $k$ -torki, minimalnoj duljini poravnjana, dopuštenom neprekapanju na krajevima i minimalnom postotku sličnosti traženom za poravnanje (Sl. 8.4). Ti parametri također utječu na robusnost u slučaju veće greške očitanja i male pokrivenosti. S obzirom da su pojedina preklapanja neovisna, ova se faza može izvoditi paralelno.



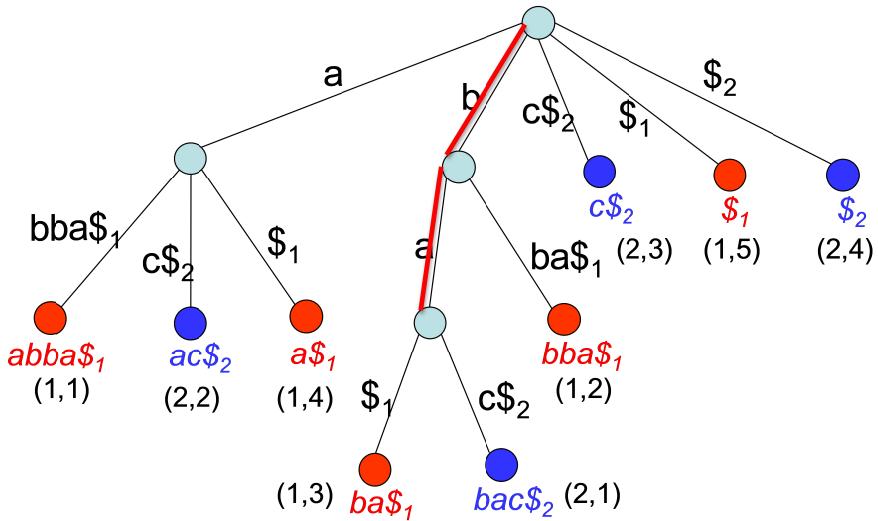
Sl. 8.4 Primjer preklapanja. Kod određivanja preklapanja potrebno je postaviti ograničenja na duljinu dijela koji se preklapa, identičnost nukleotida u tome dijelu te maksimalno dozvoljenom ostatku koji se ne preklapa na krajevima očitanja.

Za indeksiranje podnizova odnosno  $k$ -torki mogu se koristiti: raspršeno adresiranje, sufiksna stabla, sufiksna polja, i komprimirane strukture poput FM indeksa. Redoslijed kojim su navedene metode indeksiranja odgovara i redoslijedu prema brzini pristupa podacima, ali ne i utrošku memorije: pristup raspršenom adresiranju je brz, ali nam za njegovu izgradnju i smještaj treba najveća količina memorije, dok je pristup FM indeksu najsporiji, ali nam je potrebno najmanje memorije za njegovu izgradnju i smještaj. Odabir indeksa će prije svega ovisiti o veličini genoma, odnosno dostupnoj memoriji.

Jedna od prvih heurističkih metoda korištenih za preklapanja je metoda temeljena na poopćenim sufiksnim stablima (Gusfield, 1997). Metoda traži sve parove sufiks-prefiks slaganja. Problem traženja je definiran na sljedeći način. Za dva niza  $s_i$  i  $s_j$ , bilo koji sufiks od  $s_i$  koji se slaže s prefiksom od  $s_j$  naziva se sufiks-prefiks slaganje. Za danu kolekciju nizova  $S=s_1, s_2, \dots, s_k$  totalne duljine  $L$ , problem svih parova sufiks-prefiks je problem nalaženja, za svaki uređeni par  $s_i, s_j \in S$ , najduljeg sufiks-prefiks slaganja  $s_i$  i  $s_j$ .

Koristeći sufiksna stabla moguće je izračunati  $k^2$  parova preklapanja u  $O(L+k^2)$ , uz prepostavku da je abeceda fiksna. Za potrebe algoritma uvodimo termin *krajnjeg brida*. Brid nazivamo krajnjim ako je označen samo znakom za kraj niza. Jasno je da svaki krajnji brid ima list na svome kraju, ali nisu svi bridovi spojeni na listove krajnji bridovi.

Za rješavanje problema koristimo poopćeno sufiksno stablo  $T(S)$  za  $k$  nizova iz skupa  $S$ . Tijekom izrade stabla  $T(S)$ , algoritam također gradi listu  $N(v)$  za svaki unutarnji čvor  $v$ . Lista  $N(v)$  sadrži indeks  $i$  ako i samo ako je  $v$  vezan terminalnim bridom čiji list je označen sufiksom niza  $s_i$ . Pojednostavljeno,  $N(v)$  sadrži indeks  $i$  ako i samo ako je oznaka puta u  $v$  kompletan sufiks niza  $s_i$ . Sl. 8.5 prikazuje primjer poopćenog sufiksnog stabla za dva niza. Čvor s oznakom puta  $ba$  ima listu  $N$  koja sadrži samo jedan indeks 1. Isto tako čvor s oznakom puta  $a$  ima listu  $N$  koja također sadrži indeks 1. Sve ostale liste su prazne. U ovome slučaju nema niti jednoga unutarnjeg čvora koji bi u listi imao indeks 2. Lista se može konstruirati u linearnom vremenom za vrijeme ili nakon izgradnje  $T(S)$ .



Sl. 8.5 Primjer traženja maksimalnoga preklapanja nizova  $s=ABBA$  i  $t=BAC$  koristeći poopćeno sufiksno stablo. Crvenom bojom su označeni listovi u kojima završava niz  $s$ , a plavom oni listovi u kojima završava niz  $t$ . Bridovi koji opisuju preklapanje dodatno su obilježeni crveno.

Promotrimo niz  $s_j$  i fokusirajmo se na put od korijena  $T(S)$  do lista  $j$  koji predstavlja cijeli niz  $s_j$ . Ključno je primijetiti da ako je  $v$  čvor na tom putu i  $i$  je indeks u  $N(v)$ , onda oznaka puta od  $v$  je sufiks od  $s_i$  koji odgovara prefiksu od  $s_j$ . Na taj način za svaki indeks  $i$ , najdublji čvor  $v$  na putu do lista  $j$  takav da  $i \in N(v)$  identificira najdulje poklapanje sufiksa od  $s_i$  i prefiksa od  $s_j$ . Oznaka puta u  $v$  je najdulje sufiks-prefiks slaganje od  $(s_i, s_j)$ . Lako je vidjeti da jednom šetnjom od korijena do lista  $j$  možemo pronaći najdublje čvorove za sve  $1 \leq i \leq k$  ( $i \neq j$ ).

Možemo primijetiti da ovako dizajnirani algoritam može efikasno kupiti potrebna sufiks-prefiks slaganja koristeći pretraživanje prvo u dubinu (engl. *depth-first search* DFS) stabla  $T(S)$ . Za vrijeme pretrage on održava  $k$  stogova, jedan za svaki niz. Za vrijeme pretraživanja prvo u dubinu, gdje neki čvor  $v$  obilazimo prvi put, stavljamo  $v$  na  $i$ -ti stog,

za svaki  $i \in N(v)$ . Kada list  $j$  (koji predstavlja cijeli niz  $s_j$ ) je dostignut, provjerimo  $k$  stogova i zabilježimo za svaki indeks  $i$  trenutni vrh  $i$ -tog stoga. Nije teško vidjeti da vrh stoga  $i$  sadrži čvor  $v$  koji određuje sufiks-prefiks slaganje od  $(s_i, s_j)$ . Ako je  $i$ -ti stog prazan, onda ne postoji preklapanje između sufiksa niza  $s_i$  i prefiksa niza  $s_j$ . Kada se vraćamo natrag DFS-om i nađemo na čvor  $v$ , skinemo vrhove svih stogova čiji indeks je u  $N(v)$ .

Primijetimo da ukupan broj indeksa u svim listama  $N(v)$  je  $O(m)$ . Broj bridova u  $T(s)$  je također  $O(m)$ . Svako skidanje i stavljanje sa stoga je pridruženo listu  $T(s)$  i za svaki list imamo najviše jedno skidanje i jedno stavljanje na stog. Stoga obilazak  $T(s)$  i osvježavanje stogova uzima  $O(m)$  vremena. Spremanje svakog od  $O(k^2)$  odgovara se može napraviti u konstantnom vremenu. Na taj način lako izračunamo da je vremenska složenost ovoga algoritma  $O(m+k^2)$ .

U slučaju da imamo  $k' \leq k^2$  parova nizova koji imaju duljinu sufiksa i prefiksa različitu od nula, korištenjem dvostrukih veza, možemo održavati vezane liste nepraznih stogova. U tom slučaju kada list je na stablu dostignut tijekom obilaska, samo stogove na njegovoj listi treba ispitati. Na taj način, sva sufiks-prefiks slaganja mogu biti pronađena u  $O(m+k')$  vremenu. Važno je primijetiti da pozicija stoga unutar liste može varirati zbog toga što stog koji mijenja se iz praznog u neprazni se mora vezati na jedan kraj liste. Radi toga moramo u stogu čuvati ime niza pridruženog tom stogu.

S obzirom da sufiksno polje zajedno s lcp (engl. *longest common prefix*) strukturom je ekvivalentno sufiksnom stablu (Abouelhoda *et al.*, 2004; Ohlebusch and Gog, 2010), korištenjem takvih struktura možemo postići istu složenost pronalaska preklapanja kao za sufiksna stabla: uz nešto veću konstantu, ali uz manji utrošak memorije. Slično je i s FM indeksom s time da je to struktura koja ima najsporiji pristup.

Gornji algoritam temeljen na sufiksnom stablu je temeljen na potpunom egzaktnom poravnanju prefiksa i sufiksa, međutim to često nije slučaj. Stoga se danas primjenjuju algoritmi koji traže preklapanja i u slučaju nekoliko neslaganja u nukleotidima. Kao pristup se često koristi metoda da se očitanje podijeli u određeni broj  $k$ -torki i onda se pokušava napraviti ili potpuno ili djelomično preklapanje tih  $k$ -torki. Na osnovu broja prekloplojenih  $k$ -torki ocjenjuje se kvaliteta preklapanja (Välimäki *et al.*, 2012; Simpson and Durbin, 2010). Često se za određeni broj kandidatnih preklapanja napravi dodatna provjera dinamičkim programiranjem.

### 8.4.2. Razmještanje

U fazi preklapanja, svako se očitanje uspoređuje sa svim ostalim očitanjima (u obje orijentacije) u cilju pronaći mogućih odnosa između očitanja. U fazi razmještanja koja slijedi nakon toga odabire se podskup parova preklapanja što određuje poziciju svakoga očitanja u odnosu na sva ostala.

Neka se specifično rješenje opiše rekonstruiranim slijedom  $R$ , a razmještaj se sastoji od  $F$  parova cijelih brojeva,  $(s_i, e_i)_{i \in [1, F]}$  gdje  $1 \leq s_i, e_i \leq |R|$ .  $i$ -ti par razmještaja pokazuje da  $f_i$  je kopija podniza  $R[s_i, e_i]$  ako  $s_i \leq e_i$  ili  $R[s_i, e_i]^c$  inače. Na taj način poredak  $s_i$  i  $e_i$  određuje orijentaciju očitanja u razmještaju, što u stvari predstavlja je li  $f_i$  dobivena iz  $R$  ili njegova komplementarnog lanca. Kažemo da je početna točka  $sp_i$  (engl. *start-point*) očitanja  $f_i$  u razmještaju  $\min(s_i, e_i)$ , a njegova krajnja točka  $ep_i$  (engl. *end-point*) je  $\max(s_i, e_i)$ . Da bi bio  $\varepsilon$ -validan, razmještaj mora zadovoljavati sljedeća dva svojstva: (a) svako očitanje može biti poravnato sa svojim pridruženim podnizom s ne više od  $\varepsilon/|f_i|$  razlika i (b) svaki simbol od  $R$  mora biti pokriven nekim očitanjem. Potencijalni prostor rješenja je skup svih  $\varepsilon$ -razmještaja, od kojih se bira ono najbolje.

Za dani razmještaj, imamo opažene distribucije početnih točaka poravnanja:

$$D_{obs}(x) = |\{f_i : sp_i < x\}|/F$$

što predstavlja proporciju očitanja koja počinju prije pozicije  $x$ . Neka je poznata početna distribucija uzorkovanja  $D_{src}$ . Na osnovi gornjih pretpostavki možemo formulirati problem sastavljanja očitanja na sljedeći način. Uz dana očitanja i maksimalnu stopu pogreške  $\varepsilon \in [0, 1]$  pronaći rekonstrukciju  $R$  i  $\varepsilon$ -validnog razmještaja očitanja čija opažena distribucija očitanja  $D_{obs}$  ima minimum relativne devijacije od  $D_{src}$  (Myers, 1995)

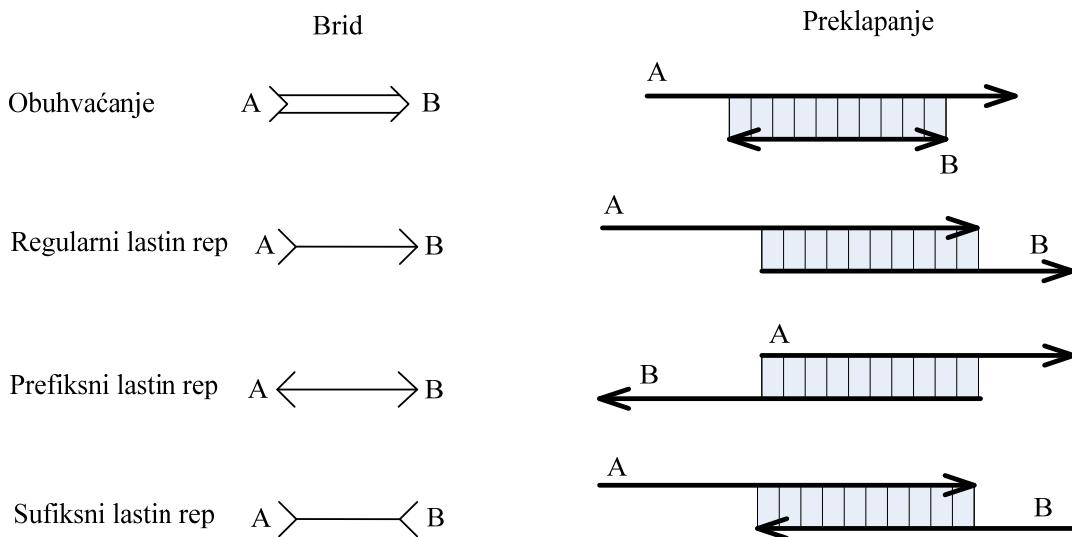
Važno je primijetiti da je problem opisan kao traženje maksimalne izglednosti  $\varepsilon$ -validnog razmještaja. Na taj način razmatra se prostor rješenja za „zašumljeni“ najkraći zajednički nadniz očitanja. Cilj je da radije nego da evaluiramo razmještaj na osnovu duljine rekonstrukcije, tražimo onaj najviše konzistentan s činjenicom da početne točke očitanja su uzete uzduž duljine ciljanog slijeda s distribucijom  $D_{src}$ . Važno je naglasiti da  $D_{src}$  može biti bilo koja distribucija.

Na osnovu gornje formulacije u fazi razmještanja određujemo parove  $(s_i, e_i)$  unutar točnosti koja je ovisna o  $\varepsilon$ . U konsenzus fazi formira se mjera konsenzusa višestrukog poravnanja u svim područjima gdje je prekrivanje dva ili više u cilju odabira konsenzus

znaka za svaku poziciju rezultirajući rekonstrukcijom  $R$ . Tek kada je  $R$  izračunat u zadnjoj fazi, točna vrijednost parova  $(s_i, e_i)$  razmještaja je poznata.

Graf preklapanja  $G$  rezultat je faze preklapanja. Taj graf modelira sva približna preklapanja dobivena nekim od algoritama za preklapanje. Svako se očitanje prikazuje vrhom, a svako preklapanje bridom. Graf može biti višestruki na način da može postojati više od jednog brida između parova očitanja  $A$  i  $B$ . Ovo je rezultat činjenice da može postojati značajno preklapanje između  $A$  i  $B$ , ali i između  $A$  i  $B^c$ . Graf preklapanja modeliramo preko dijelova očitanja  $\pi.A$  i  $\pi.B$  uključenih u preklapanje  $\pi$  i zbog mogućnosti reverznog preklapanja potrebno je imati direktni graf. Po konvenciji da preklapanje uvijek gledamo iz perspektive  $\pi.A$  koji ima orientaciju prema naprijed dok je orientacija  $\pi.B$  promjenjiva. Potpuna specifikacija preklapanja  $\pi$  postignuta je specificiranjem (a) podnizova  $\pi.A[\pi.s_A, \pi.e_A]$  i  $\pi.B[\pi.s_B, \pi.e_B]$  svakoga očitanja uključenog u poravnanje i (b) liste  $\pi.\Delta$  pozicija neporavnatih simbola svakoga očitanja u poravnanju preklapanja. Kako je  $\pi$  opisan iz perspektive  $\pi.A$  uvijek je slučaj da je  $\pi.s_A \leq \pi.e_A$ . Ako u specifikaciji razmještaja vrijedi  $\pi.s_B > \pi.e_B$ , to je znak da je došlo do preklapanja između  $\pi.A[\pi.s_A, \pi.e_A]$  i  $\pi.B[\pi.s_B, \pi.e_B]^c$ .  $\pi.\Delta$  se često naziva  $\Delta$ -kodiranje poravnjanja, i dok postoje drugi načini za kodiranje,  $\Delta$ -kodiranje je memorijski najefikasnije zbog toga što je  $\varepsilon$  rijetko veći od 10% .

Obzirom na očitanje  $B$  i podnizove uključene u preklapanje možemo definirati četiri kategorije bridova. Sl. 8.6 prikazuje te kategorije. Lako je uočiti da se sve ostale kategorije mogu opisati s ove četiri.

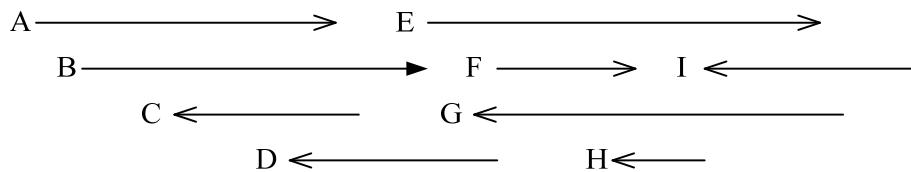


Sl. 8.6 Taksonomija tipova preklapanja. Kod obuhvaćanja smjer  $B$  očitanja nije bitan (strelica može gledati prema lijevo ili prema desno).

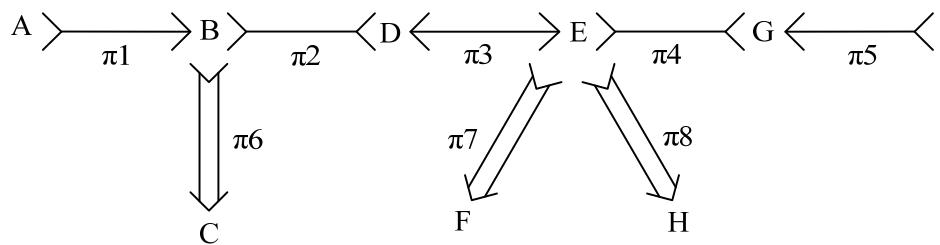
Brid obuhvaćanja modelira situacije kada je cijeli  $B$  poravnat s podnizom od  $A$  i označen je usmjerenum bridom s dvije linije od  $A$  prema  $B$ . Svi drugi bridovi imaju oblik lastinog repa pri čemu prefiks ili sufiks jednoga brida je poravnat s prefiksom ili sufiksom drugoga. Svaki takav brid u obliku lastinog repa je označen dvosmjernim bridom s jednom linijom. Za prefiksni lastin rep, strelice su usmjerene prema  $\pi.s_A$  i  $\pi.s_B$ . Za sufiksni lastin rep, obje strelice usmjerene su od  $\pi.s_A$  i  $\pi.s_B$ .

Kao rezultat preklapanja dobije se dvosmjerni višestruki graf obuhvaćanja i bridova u obliku lastina repa. Svaki brid/preklapanje je dalje označen egzaktnim podnizom uključenim u poravnanje preklapanja i njegovim  $\Delta$ -kodiranjem. Sl. 8.7 prikazuje primjer izgrađenog grafa na osnovu njegovih preklapanja.

Razmještaj:



Graf:



Sl. 8.7 Primjer izgradnje grafa iz preklapanja

Da bismo opisali sljedeći korak – pojednostavljenje grafa – potrebno je uvesti još nekolicinu oznaka. Pretpostavimo da je  $f \xleftrightarrow{\pi} g$  brid u  $G$ . Neka  $\pi.s_g$  označava  $\pi.s_A$  ako je  $\pi.A = g$  ili  $\pi.s_B$  inače. Slično definiramo  $\pi.e_g$ ,  $\pi.s_f$  i  $\pi.e_f$  tako da se ne moramo mučiti s time koje očitanje ima ulogu A očitanja u kodiranju  $\pi$ . Nadalje neka su  $\pi.lft_g = \min(\pi.s_g, \pi.e_g)$  i  $\pi.rgt_g = \max(\pi.s_g, \pi.e_g)$  uredeni graničnici podniza koji se preklapa npr  $g[\pi.lft_g, \pi.rgt_g]$ . Na kraju, neka je  $\pi.hang_g = |[1, |g|] - [\pi.lft_g, \pi.rgt_g]|$  broj simbola u  $g$  koji nisu u preklapanju i neka je  $\pi.suf_g$  istina ako i samo ako je preklapanje

podniza od  $g$  sufiks od  $g$ . Primijetiti da za bridove lastinog repa,  $\pi.suf_g$  je istina ako i samo ako je strelica u  $g$  usmjerena od  $g$ .

Nastavljamo s nizom redukcija grafa preklapanja na način da smanjujemo broj vrhova i bridova bez promjene prostora potencijalnih rješenja. Intuicija nalaže da sastavimo sve dijelove problema koji se ujedinjuju na jedinstveni način, i onda dalje radimo s jasnim kombinatornim izborima koji preostanu u reduciranim grafovima blokova.

Sl. 8.8 prikazuje tri osnovne transformacije grafa preklapanja. Transformacije su primjenjene neovisno i u sljedećem poretku:

1. *Uklanjanje već uključenih očitanja:* Svako očitanje već uključeno nekim drugim se uklanja iz grafa  $G$  zajedno sa svim bridovima spojenim s njime. Lista tih očitanja se spremi tako da mogu biti ponovo korišteni kasnije u konsenzus fazi. Nakon ovoga koraka ostaju samo čvorovi u obliku lastinog repa.
2. *Uklanjanje tranzitivnih bridova.* Ako su  $f \xleftrightarrow{\tau} g \xleftrightarrow{\tau'} h$  i  $f \xleftrightarrow{\pi} h$  međusobno konzistentna preklapanja između očitanja  $f$ ,  $g$  i  $h$ , onda se brid  $\pi$  uklanja. Preklapanja su međusobno konzistentna ako preklapanje između  $f$  i  $h$  podrazumijevano nastavljanjem poravnjanja  $\tau$  i  $\tau'$  je isto kao od  $\pi$  unutar stope pogreške  $\varepsilon$ . Jednostavna formulacija dovoljna za praktične primjene je sljedeća:

$$\begin{aligned}\tau.suf_g &\neq \tau'.suf_g \\ \pi.suf_f &= \tau.suf_f \\ \pi.suf_h &= \tau'.suf_h \\ \tau.hang_f + \tau'.hang_f &\in \pi.hang_f \pm (\varepsilon \cdot duljina(\pi) + \alpha) \\ \tau.hang_g + \tau'.hang_h &\in \pi.hang_h \pm (\varepsilon \cdot duljina(\pi) + \alpha)\end{aligned}$$

gdje je  $\alpha$  mala konstanta koja pomaže uhvatiti fluktuacije u distribuciji pogreške kada je duljina preklapanja  $\pi$  mala. Odabir  $\alpha$  da bude 3 je u praksi velikodušno.

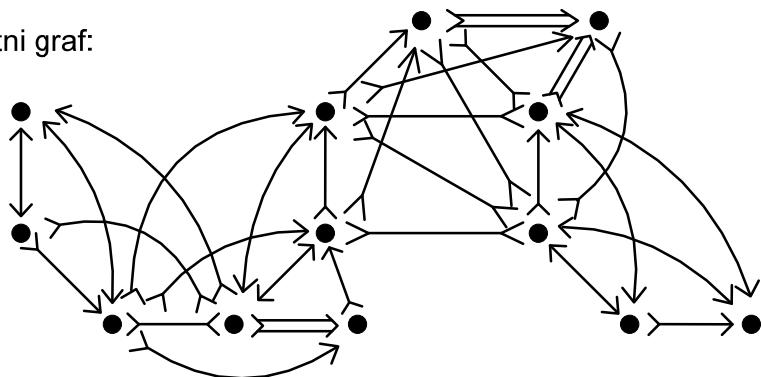
3. *Ujedinjavanje.* Ukoliko postoji brid  $f \xleftrightarrow{\pi} g$  takav da za svaki drugi brid  $x \xleftrightarrow{\pi} f$  pridružen  $f$ ,  $\pi.suf_f = \tau.suf_f$  i za svaki drugi brid  $x \xleftrightarrow{\pi} g$  pridružen  $g$   $\pi.suf_g = \tau.suf_g$ , onda ujedinjujemo  $f$  i  $g$  u jedan vrh koji predstavlja kontig od  $f$  preklopljen na  $g$  s  $\pi$ . Pojednostavljeni, uvjeti za ujedinjavanje su da strelice u  $f$  svih bridova pridruženih  $f$  osim  $\pi$  moraju pokazivati u suprotnom smjeru od onoga od  $\pi$ , i isto mora biti istina za strelice u  $g$ . Dobiveni graf se naziva graf blokova.

U drugom koraku treba biti pažljiv da se prvo obilježe svi tranzitivni vrhovi, a da se tek potom uklanja obilježene vrhove, iz razloga što tranzitivnost jednog brida može podrazumijevati tranzitivnost drugih bridova

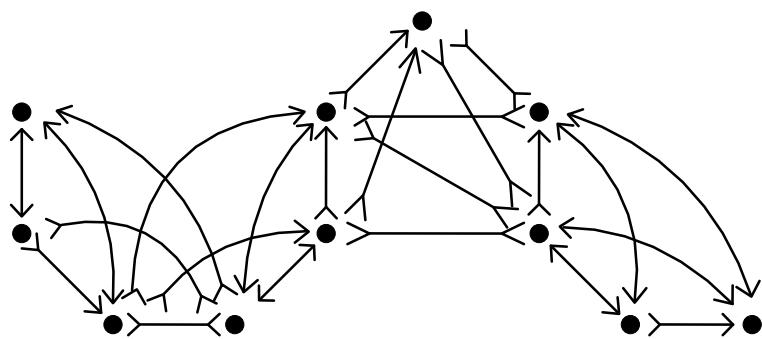
#### **8.4.3. Konsenzus**

Nakon razmještanja, OLC treba proglašiti konsenzusni slijed na osnovu višestrukog poravnjanja sljedova dobivenih u svakom bloku dobivenom u fazi razmještanja.

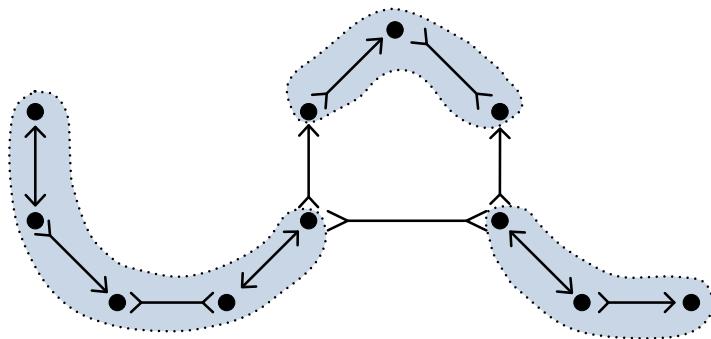
Početni graf:



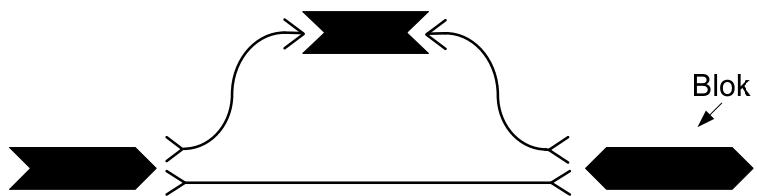
Nakon koraka 1:



Nakon koraka 2:



Nakon koraka 3:



Sl. 8.8 Redukcija grafa preklapanja (Myers, 1995)

## **8.5. Sastavljanje koristeći de Bruijn grafove**

### **8.5.1. Problem sastavljanja genoma kratkim očitanjima**

Pojavom novih tehnologija sekvenciranja (tzv. Sekvenciranje sljedeće generacije) koje su omogućavale brže sekvenciranje s manjim troškovima, javili su se i novi izazovi za sastavljanje genoma. Osnovna karakteristika ovih uređaja je da su izlazna očitanja kraća, nego ona dobivena Sangerovom metodom. Posljedično, potrebno je imati puno veću pokrivenost. Za razliku od pokrivenosti od  $8\times$  u projektima koji koriste Sangerovo sekvenciranje, projekti koji koriste očitanja sljedeće generacije često imaju pokrivenost  $30\times$ ,  $40\times$ ,  $50\times$  ili više.

U teoriji, algoritmi za sastavljanje kreirani za dulja očitanja (OLC) bi trebali raditi i za kraća očitanja. Principi detekcije preklapanja i izrade kontiga nisu različiti. Međutim, u praksi se pokazalo da korištenje tih metoda na vrlo kratkim očitanjima daje slabe rezultate. Neki od razloga su praktični kao to da te metode imaju ograničenja na minimalnu ulaznu duljinu očitanja ili zahtijevaju minimalnu duljinu preklapanja koja je predugačka za kratka očitanja. Drugi problemi su fundamentalniji.

Računalno najkritičniji korak u sastavljanju je preklapanje. Projekti s kratkim očitanja zahtijevaju redizajniranje toga koraka da ga se učini računalno izvedivim, posebno što im je potrebno puno više kratkih očitanja nego dugih za postizanje istoga preklapanja. Dodatno, za kraća očitanja trebamo puno veću pokrivenost.

Zbog navedenih razloga pojavila se nova generacija metoda za sastavljanje genoma razvijenih specifično za kraća očitanja. Najpoznatiji alati među njima su Velvet (Zerbino and Birney, 2008; Zerbino *et al.*, 2009), ALLPATHS (Butler *et al.*, 2008), ABySS (Simpson *et al.*, 2009) i SOAPdenovo (Luo *et al.*, 2012).

### **8.5.2. Eulerova staza i de Bruijn grafovi**

*Eulerovu stazu* definiramo kao šetnju grafom u kojoj svaki brid obiđemo točno jedanput. Slično, *Eulerov ciklus* je Eulerova staza koja počinje i završava u istom vrhu. Prvi se problemom traženja takvoga ciklusa bavio Leonhard Euler, jedan od najvećih matematičara u povijesti. Euler je pokušao riješiti problem koji je dugo zabavljao građane starog Königsberga, pruskoga grada na rijeci Pregel (danas Kaliningrad, Rusija). Grad se sastojao od četiri gradske četvrti međusobno povezane sa sedam mostova. Dvije središnje

četvrti su otoci na rijeci Pregel. Cilj je bio pronaći šetnju gradom tako da se krene iz jedne gradske četvrti, prođe svaki most jedanput i samo jedanput i vrati u istu četvrt. Problem je bio tim teži što u to doba nije postojala tehnika kojom bi se mogao takav problem riješiti. Euler je uočio da šetnja unutar iste četvrti nije važna, te da je jedino bitan slijed prelazaka mosta. To mu je omogućilo da postavi problem na način da označi samo četiri gradske četvrti i mostove između njih (Sl. 8.9). Gradske četvrti su postali vrhovi, a mostovi bridovi. Time je postavio temelje teorije grafova. Na kraju je Euler koristeći novu metodologiju pokazao da tražena šetnja ne postoji.



Sl. 8.9 Mostovi Königsberga. (a) Mapa starog Königsberga u kojoj je svaki dio grada označen drugačije obojanom točkom (b) Graf Königsbergovih mostova, stvoren tako da je svaki od četiri dijela grada predstavljen vrhom, a sedam mostova među njima bridom.

Euler je u svom radu pokazao da je nužan uvjet za postojanje Eulerovog ciklusa taj da svi vrhovi u grafu imaju paran stupanj i tvrdio bez dokaza da povezan graf sa svim vrhovima parnog stupnja ima Eulerov ciklus. Tu tvrdnju je kasnije dokazao Carl Hierholzer. U slučaju usmjerenih grafova potrebno je da svaki od vrhova ima jednak broj ulaznih i izlaznih bridova. Takav graf nazivamo i balansiranim grafom. Graf koji ima Eulerov ciklus ujedno ima i Eulerovu stazu.

Općenito, možemo reći da su nužni i dovoljni uvjeti da graf ima Eulerovu stazu sljedeći:

- Najviše jedan vrh u grafu ima  $(\text{izlazni stupanj} - \text{ulazni stupanj}) = 1$ .
- Najviše jedan vrh u grafu ima  $(\text{izlazni stupanj} - \text{ulazni stupanj}) = -1$ .
- Svi ostali vrhovi imaju  $(\text{izlazni stupanj} - \text{ulazni stupanj}) = 0$ .

Jednostavnije možemo reći da je nužan uvjet za postojanje Eulerove staze da i graf ima točno dva vrha neparnoga stupnja. Prvi efikasan algoritam za određivanje Eulerove staze u linearnoj složenosti razvio je Hierholzer po komu se danas taj algoritam i naziva.

#### Hierholzerov algoritam

- Odabratи почетни vrh  $v$  kojem je izlazni stupanj za jedan veći od ulaznog stupnja (ako takav ne postoji, krenuti od bilo kojeg) i pratiti stazu bridova od toga vrha sve dok se ne vratimo u njega ili ne dođemo do vrha koji ima za jedan veći ulazni stupanj od izlaznog stupnja  $z$  (ukoliko takav vrh postoji). Nije moguće da se zaglavimo niti u jednom drugom vrhu zato što paran stupanj svi vrhova osigurava da kada bridom dođemo u neki drugi vrh  $w$  mora postojati brid kojim možemo napustiti  $w$ . Šetnja formirana na ovaj način je zatvorena, ali ne prolazi nužno svim vrhovima i bridovima početnog grafa.
- Sve dok ima vrhova  $v$  koji pripadaju trenutnoj šetnji, ali imaju bridove koji joj ne pripadaju, krenuti novom šetnjom iz vrha  $v$ , prateći neiskorištene bridove sve dok se ne vratimo u  $v$  te nakon toga ujedinimo novu šetnju s postojećom.

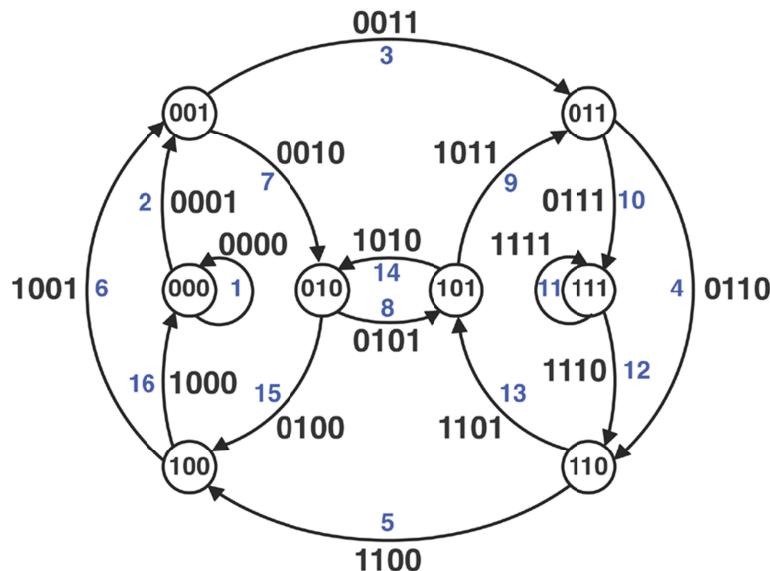
Sl. 8.10 Hierholzerov algoritam za računanje Eulerove staze u grafu

Algoritam se najčešće implementira koristeći dva stoga. U jednom spremamo put kojim idemo, a drugi koristimo za praćenje unatrag u cilju određivanja neposjećenih bridova.

Vrlo često pitanje je kako to da nije poznat polinomni algoritam za rješenje Hamiltonovog puta dok Eulerov put možemo pronaći u linearnoj složenosti. Prvi razlog je to što uvjeti da bi neki graf imao Hamiltonov ciklus nisu poznati, dok za postojanje Eulerova puta znamo nužne i dovoljne uvjete i lako ih je provjeriti. Dodatno, za pronalaženje Eulerovog puta možemo koristiti već riješene podprobleme i na taj način se približavamo rješenju originalnoga problema.

Nizozemski matematičar Nicolaas de Bruijn je prilagodio Eulerove ideje u cilju pronalaska cikličkih sljedova slova uzetih iz dane abecede za koju se svaka moguća riječ određene duljine ( $k$ ) pojavljuje kao niz uzastopnih znakova u cikličkom slijedu točno jedanput (Compeau *et al.*, 2011). Pojednostavljeni, uzmemu sve moguće nizove duljine  $k$  za danu abecedu i tražimo najkraći zajednički nadniz. U poglavlju 8.3.1 pokazali smo da traženje zajedničkoga nadniza je NP težak problem za koga postoje samo heuristička rješenja. Međutim, problem kako ga je definirao de Bruijn je specifičan jer postoje svi mogući nizove duljine  $k$ . Npr. ako se abeceda sastoji od znakova „0“ i „1“ i  $k=3$ , onda imamo 8

mogućih nizova: 000, 001, 010, 011, 100, 101, 110, 111. S obzirom da imamo sve kombinacije možemo primijetiti da uvjek možemo naći za svaki niz da njegov  $k-1$  sufiks odgovara  $k-1$  prefiksu nekoga drugoga niza. Koristeći tu činjenicu de Bruijn je konstruirao graf na način da je svakom vrhu pridružio  $(k-1)$ -torku, a dva vrha je povezao bridom ukoliko postoji  $k$ -torka kojoj je jedan vrh  $k-1$  prefiks, a drugi  $k-1$  sufiks. Bridovi su označeni  $k$ -torkama. Sl. 8.11 prikazuje graf za  $k = 4$ . De Bruijn je ovaj tip grafova nazvao  $B$  grafovima, no danas se u njegovo ime nazivaju de Bruijn grafovima. Ako promatramo s koliko drugih vrhova pojedini vrh može biti povezan, dođemo do toga da i ulazno i izlazno može biti povezan s brojem vrhova koji je jednak broju znakova u abecedi. Do toga dođemo uzimajući u obzir činjenicu da se dva vrha koja spajamo razlikuju u maksimalno jednom znaku. Stoga svaki vrh u grafu ima jednak ulazni i izlazni stupanj i možemo vidjeti da graf ima Eulerov ciklus. S obzirom da tražimo cirkularni ciklus možemo krenuti od bilo kojeg vrha. Rješenje zajedničkoga nadniza nađemo na način da zapamtimo prvi znak u svakom vrhu kroz koji prođemo ili prvi znak svakoga brida.



Sl. 8.11 De Bruijn graf. De Bruijn graf za  $k=4$  i abecedu koja se sastoji od „0“ i „1“. Ovaj graf ima Eulerov ciklus zato što svaki vrh ima ulazne i izlazne stupnjeve jednake 2. Slijedeći bridove označene plavim brojevima od 1 do 16 prođemo Eulerovim ciklusom **0000**, **0001**, **0011**, **0110**, **1100**, **1001**, **0010**, **0101**, **1011**, **0111**, **1111**, **1110**, **1101**, **1010**, **0100**, **1000**. Pamteći samo prve znakove u svakom brigu možemo dobiti ciklički nadniz **0000110010111101**.

Upravo ta ideja jednostavnog nalaska najkraćeg zajedničkoga nadniza koristeći *de Bruijnove* grafove danas se koristi u sastavljanju genoma, no da bismo bili sigurni da

Eulerov ciklus postoji moramo imati sve  $k$ -torke prisutne u genomu. Dodatno, u većini slučajeva mi nemamo grafove koji su cirkularni pa za razliku od traženja Eulerovog puta moramo tražiti Eulerovu stazu, no kao što je prije navedeno uz malu modifikaciju uvjeta (početni i završni vrh) prelazimo iz ciklusa u stazu.

### 8.5.3. Sastavljanje genoma

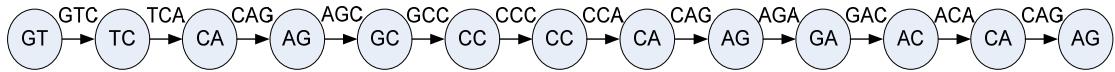
Važno je zapaziti da je de Bruijn grafove i pronalazak Eulerove staze moguće koristiti za sastavljanje genoma. Nužan uvjet je da u očitanjima imamo sve  $k$ -torke prisutne u genomu. Međutim, očitanja dobivena sekvenciranjem uređajima druge generacije, bez obzira na visoku točnost i veliku pokrivenost ipak ne sadrže sve  $k$ -torke. Zbog toga se očitanja lome u kraće  $k$ -torke, što rezultira da za dovoljno mali  $k$  imamo sve  $k$ -torke prisutne u grafu. Npr. ako imamo očitanja duljine 100 razlomimo ih u 51 preklapajuću  $k$ -torku duljine 50. Iako na prvu ovo izgleda kontraintuitivno, jer na ovaj način gubimo dio informacije upravo ova metoda nam omogućava ispravno sastavljanje genoma. Sl. 8.12 prikazuje primjer lomljenja očitanja u kome vidimo kako kao rezultat dobijemo savršenu pokrivenost ulaznog slijeda. Primijetiti da sa smanjenjem  $k$  postoji veća vjerojatnost potpune pokrivenosti, no isto tako dobiveni graf može biti veći (više vrhova i bridova) i zapetljaniji, te je teže pronaći pravu Eulerovu stazu. Stoga odabir  $k$  predstavlja kompromis između želje za što većom pokrivenosti i jednostavnosti grafa.

atgcgttatggacaacgact atgcgtatg gccgtatgga gtatggacaa gacaacgact	atgcgttatggacaacgact atgc tgccg gccgt ccgta cgtat gtatg tatgg atgga tggac ggaca gacaa acaac caacg aacga acgac cgact
---	---

Sl. 8.12 Lomljenje očitanja. Lijeva očitanja ne sadrže sve  $10$ -torke, no ako ih razlomimo na  $5$ -torke (desno) dobijemo savršenu pokrivenost.

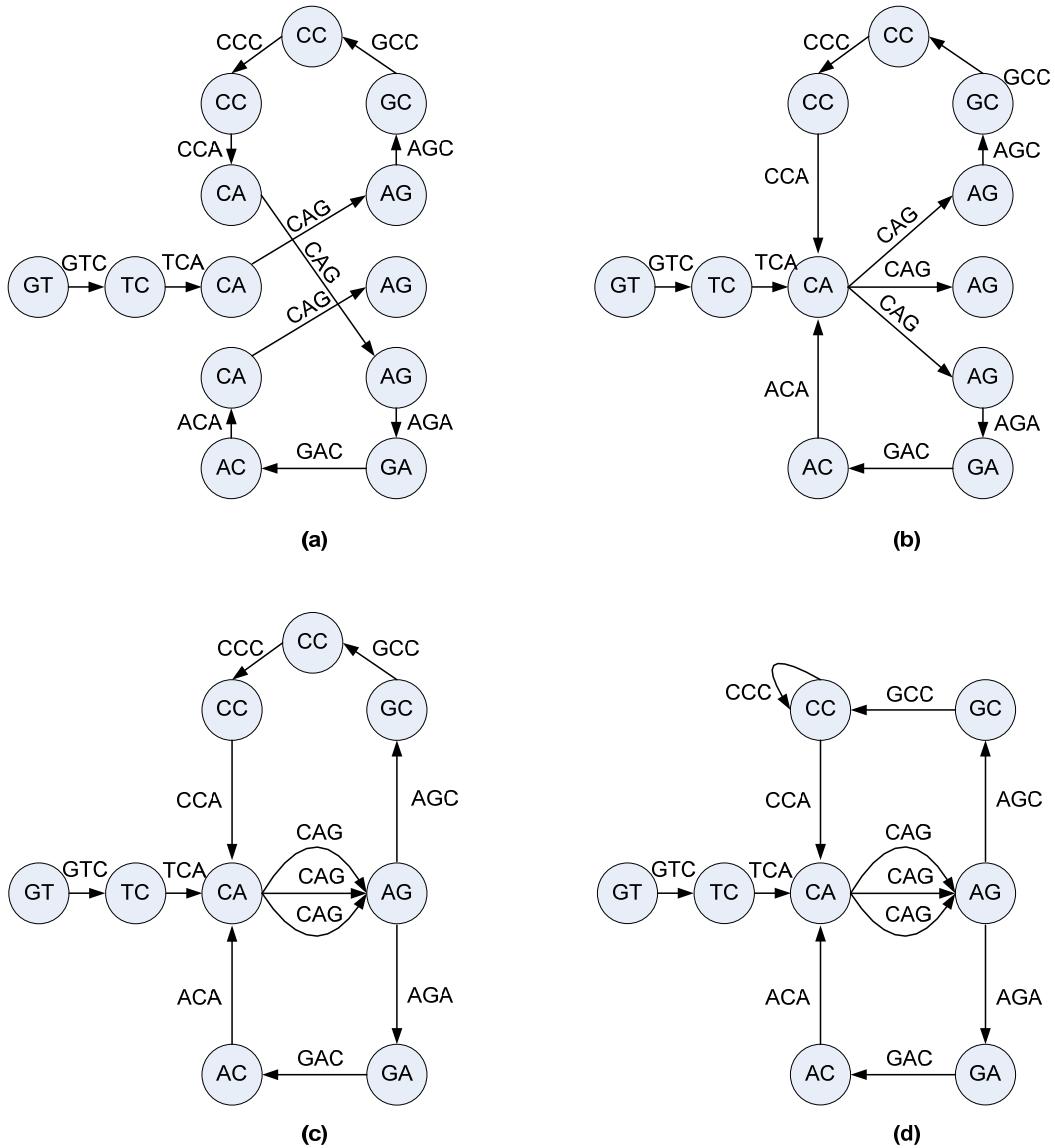
Ovakav pristup sastavljanju genoma se naziva de Bruijn graf pristupom (engl. *de Bruijn graph - DBG*) ili pristupom Eulerovog puta (Pevzner *et al.*, 2001).

Za primjer konstrukcije *de Bruijn* grafa uzmimo niz  $GT\textcolor{red}{CAGCC}CAGACAG$  i napravimo od njega sve  $k$ -torke ( $k=3$ ) i njih pridružimo bridovima, a njihove  $k-1$  sufikse i prefikse vrhovima između kojih se nalazi taj brid. Sa crvenom bojom su označene ponavljače regije. Sl. 8.13 prikazuje tako konstruiran graf.



Sl. 8.13 Slijed  $GTCAGCCCAGACAG$  sastavljen od  $k$ -torki duljine 3.  $K$ -torke se nalaze na bridovima, a vrhovi predstavljaju njihove  $k-1$  prefiks i sufikse.

Možemo primijetiti da postoji više vrhova s istim oznakama. Uz zadržavanje bridova prema njima susjednim vrhovima, možemo te vrhove slijepiti odnosno ujediniti u jedan vrh. Na taj način pojednostavljujemo graf (Sl. 8.14). Postoje dva slučaja kod lijepljenja vrhova. U prvom su svi vrhovi povezani s vrhovima drugačije oznake, dok su u drugom takvi vrhovi



Sl. 8.14 Pojednostavljenje grafa lijepljenjem vrhova s istom oznakom. (a) Grupiramo vrhove s istim oznakama. U ovom slučaju su to vrhovi CA i AG. (b) Lijepljenje vrhova CA. Primijetiti da njihovi bridovi prema drugim vrhovima su zadržani. (c) Lijepljenje vrhova AG. (d) Lijepljenje vrhova CC. S obzirom da je postojao brid između njih dodajemo petlju iz vrha u samog sebe.

međusobno povezani. U prvom slučaju nakon lijepljenja bridovi prema drugim vrhovima ostaju netaknuti. U drugom slučaju nakon ujedinjenja potrebno je napraviti petlje koje povezuju vrh sa samim sobom i time predstavljaju brid između dva identična vrha koja smo ujedinili lijepljenjem.

U metodi izrade grafa navedenoj gore smo prepostavili da znamo kako su bile povezane pojedine *k-torke*. Što ako nam ta informacija nije poznata, kao što neće ni biti pri sastavljanju genoma? Metoda koja se koristi u praksi za sastavljanje *de Bruijn*ova grafa je sljedeća. Za danu ulaznu kolekciju *k-torki* nađemo sve vrhove koji su jedinstvene

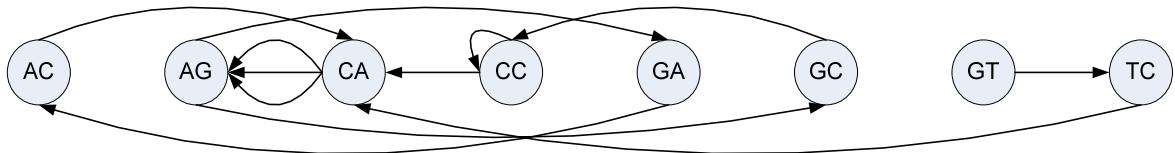
$(k-1)$ -torke. S obzirom da ne znamo njihov originalni poredak možemo ih poredati po abecedi. U našem primjeru  $k$ -torke su:

ACA AGA AGC CAG CAG CAG CCA CCC GAC GCC GTC TCA.

Jedinstveni vrhovi koji su njihovi  $k-1$  prefiksi ili sufiksi su:

AC AG CA CC GA GC GT TC.

Za svaku od ulaznih  $k$ -torki povežemo njen prefiks vrh sa sufiks vrhom direktnim bridom (Sl. 8.15).



Sl. 8.15 Izgradnja de Bruijn grafa kada ne znamo početni poredak  $k$ -torki (ACA, AGA, AGC, CAG, CAG, CAG, CCA, CCC, GAC, GCC, GTC i TCA) u slijedu. Za ulazne  $k$ -torke nađemo jedinstvene sufikse i prefikse koji definiraju vrhove. Na kraju povežemo vrhove koristeći početne  $k$ -torke.

Iako dobiveni graf izgleda na prvi pogled drugačije ako provjerimo sve bridove možemo primijetiti da se zapravo radi o istom grafu.

Nakon završetka izrade grafa traženi slijed se dobije traženjem Eulerove staze u grafu. S obzirom da između čvorova CA i AG možemo doći na tri načina postojat će više Eulerovih staza u grafu. U ovom slučaju, kao konačno rješenje možemo dobiti početni slijed  $GTCAGCCCAGACAG$ , ali i  $GTCAGCAGACCCAG$ . Na taj način nismo dobili jedinstveno rješenje.

## 8.6. Literatura

- Abouelhoda,M.I. *et al.* (2004) Replacing suffix trees with enhanced suffix arrays. *J. Discret. Algorithms*, **2**, 53–86.
- Butler,J. *et al.* (2008) ALLPATHS: de novo assembly of whole-genome shotgun microreads. *Genome Res.*, **18**, 810–20.
- Compeau,P.E.C. *et al.* (2011) How to apply de Bruijn graphs to genome assembly. *Nat. Biotechnol.*, **29**, 987–91.
- Frieze,A. & Szpankowski,W. (1998) Greedy Algorithms for the Shortest Common Superstring That Are Asymptotically Optimal. *Algorithmica*, **21**, 21–36.
- Gusfield,D. (1997) Algorithms on strings, trees, and sequences: computer science and computational biology Cambridge University Press.
- Luo,R. *et al.* (2012) SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler. *Gigascience*, **1**, 18.
- Miller,J.R. *et al.* (2010) Assembly algorithms for next-generation sequencing data. *Genomics*, **95**, 315–327.
- Myers,E.W. *et al.* (2000) A whole-genome assembly of Drosophila. *Science (80-. ).*, **287**, 2196–2204.
- Myers,E.W. (1995) Toward simplifying and accurately formulating fragment assembly. *J. Comput. Biol.*, **2**, 275–90.
- Ohlebusch,E. & Gog,S. (2010) Efficient algorithms for the all-pairs suffix-prefix problem and the all-pairs substring-prefix problem. *Inf. Process. Lett.*, **110**, 123–128.
- Pevzner,P. a *et al.* (2001) An Eulerian path approach to DNA fragment assembly. *Proc. Natl. Acad. Sci. U. S. A.*, **98**, 9748–53.
- Simpson,J.T. *et al.* (2009) ABYSS: a parallel assembler for short read sequence data. *Genome Res.*, **19**, 1117–23.
- Simpson,J.T. & Durbin,R. (2010) Efficient construction of an assembly string graph using the FM-index. *Bioinformatics*, **26**, i367–73.
- Simpson,J.T. & Durbin,R. (2012) Efficient de novo assembly of large genomes using compressed data structures. *Genome Res.*, **22**, 549–56.
- Välimäki,N. *et al.* (2012) Approximate all-pairs suffix / prefix overlaps. *Inf. Comput.*, **213**, 49–58.
- Zerbino,D.R. *et al.* (2009) Pebble and rock band: heuristic resolution of repeats and scaffolding in the velvet short-read de novo assembler. *PLoS One*, **4**, e8407.

Zerbino,D.R. & Birney,E. (2008) Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome Res.*, **18**, 821–9.