UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Miloš D. Lončarević

ALGORITMI ZA PORAVNANJE BIOLOŠKIH SEKVENCI - ELEKTRONSKA LEKCIJA

master rad

Beograd, 2022.

Mentor:

dr Jovana Kovačević, docent
Matematički fakultet, Univerzitet u Beogradu

Članovi komisije:

dr Mirjana Maljković, docent
Matematički fakultet, Univerzitet u Beogradu

Aleksandar Veljković, asistent
Matematički fakultet, Univerzitet u Beogradu

Datum odbrane: septembar 2022.

Najmilijima

Naslov master rada: Algoritmi za poravnanje bioloških sekvenci - elektronska lekcija

Rezime: U ovom radu su prikazani algoritmi koji se koriste za poravnanje bioloških sekvenci i implementirana je elektronska lekcija koja pokriva temu poravnanja bioloških sekvenci kao bioinformatičku nastavnu jedinicu. Lekcija se sastoji iz detaljnog teorijskog i interaktivnog dela. Interaktivni deo omogućava korisniku da pokrene nekoliko algoritama za poravnanje sekvenci i da prati izvršavanje algoritama korak po korak. U radu je svaki algoritam detaljno opisan uz primere konkretnih primena datih algoritama. Elektronska lekcija bi mogla da posluži kao nastavno sredstvo za predavače i kao sredstvo za učenje za studente.

Ključne reči: proteini, aminokiseline, genetski kod, nukleotid, dinamičko programiranje, poravnanje sekvenci, globalno poravnanje, lokalno poravnanje, afine kazne, višestruko poravnanje.

Sadržaj

Glava 1

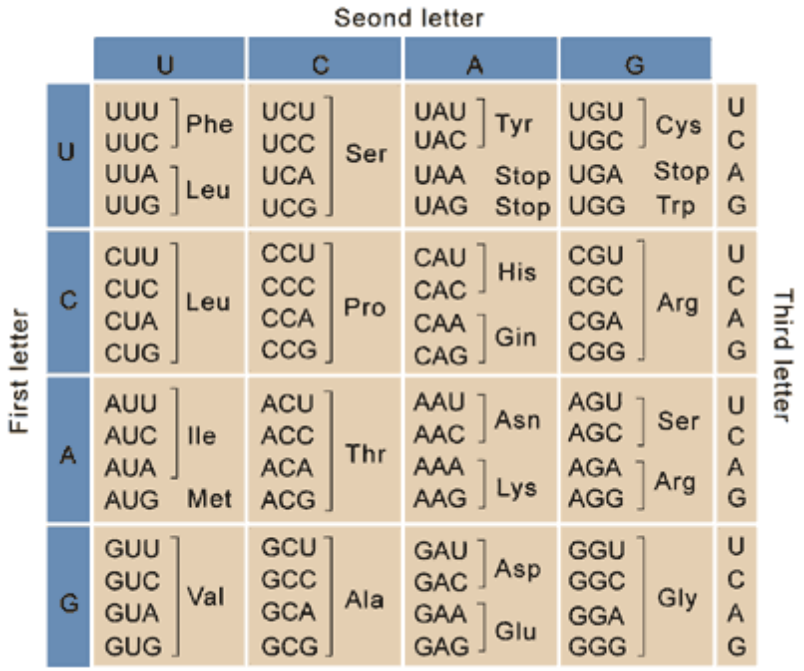
Uvod

1.1 Sinteza ribozomalnih proteina

Proteini su veliki, složeni molekuli, neophodni za funkcionisanje svih živih organizama. Proteini se sastoje od stotina hiljada manjih jedinica, zvanih aminokiseline. Postoji 20 različitih aminokiselina koje učestvuju u građi proteina.

Dezoksiribonukleinska kiselina (DNK) i ribonukleinska kiselina (RNK) su odgovorne za prenos genetskih informacija unutar ćelije. DNK je odgovorna za čuvanje i transfer genetskih informacija, dok je RNK odgovorna za kodiranje proteina i predstavlja vezu između DNK i ribozoma u kojima se generišu proteini. DNK i RNK predstavljaju nizove nukleotida, i to adenina (A), guanina (G), citozina (C) i timina (T) kod DNK ili uracila (U) kod RNK. Unutar svake DNK se nalaze segmenti na osnovu kojih se unutar ćelije sintetišu proteini. Ovi segmenti se nazivaju genima.

Generisanje proteina na osnovu DNK je kompleksan proces koji se sastoji iz dva koraka, transkripcije i translacije. Tokom transkripcije, informacije skladištene u DNK se prenose molekulu RNK tako što se ceo lanac DNK komplementarno prepisuje na molekul RNK. Tokom translacije, niz nukleotida u RNK se dešifruje unutar ribozoma, kako bi se dobio specifičan protein na osnovu pravila koja su definisana u genetskom kodu. Genetskim kodom je određeno preslikavanje između nukleotidnih tripleta (kodona) i aminokiselina (slika 1.1). Sklapanje proteina se nastavlja dok ribozomi ne dešifruju stop kodon (triplet nukleotida koji ne odgovara ni jednoj aminokiselini).



Slika 1.1: Genetski kod [1]

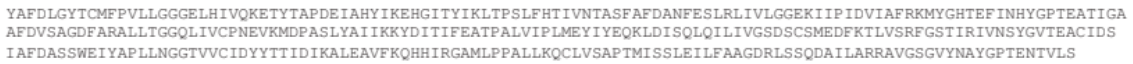
1.2 Sinteza neribozomalnih proteina

Sinteza neribozomalnih proteina (NRP) se odvija nezavisno od ribozoma, za razliku od ostalih proteina. Ovi proteini se sintetišu bez oslanjanja na ribozome i na genetski kod. Umesto toga, njihova sinteza se vrši uz pomoć ogromnih proteinskih kompleksa pod imenom **NRP sintetaze**. NRP sintetaze se generišu u okviru ribozoma, a nakon toga se na osnovu njih, pomoću neribozomalnog koda, generišu NRP-ovi (dijagram 1.1). Svaki NRP ima svoju NRP sintetazu, a svaka NRP sintetaza se sastoji od različitih segmenata, koji se nazivaju domeni adenilacije (A domeni). Svaki A domen određuje jednu aminokiselinu. Na primer, NRP sintetaza koja dešifruje 10 aminokiselina dug antibiotik Tirocidin B1, uključuje 10 A domena. Svaki A domen ima dužinu od oko 500 aminokiselina, i odgovoran je za jednu aminokiselinu u Tirocidinu B1.

$$DNK \xrightarrow{\text{genetski kod}} RNK \xrightarrow{\text{ribozom}} Protein \xrightarrow{\text{neribozomalni kod}} NRP \quad \text{(Neribozomalna proteinska sintetaza)}$$

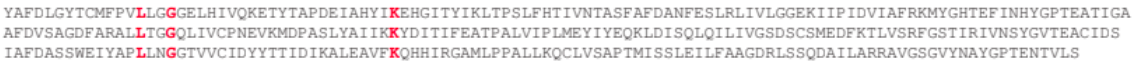
Dijagram 1.1: Sinteza NRP-ova

Kako svi A domeni imaju sličnu funkciju (dodavanje aminokiseline na rastući peptid), različiti A domeni imaju slične delove. A domeni bi takođe trebalo da imaju i različite delove koji odgovaraju različitim aminokiselinama. Na slici 1.3 možemo videti fragmente tri A domena (iz različitih bakterija), koji kodiraju asparaginsku kiselinu (Asp), orintonin (Orn) i valin (Val).



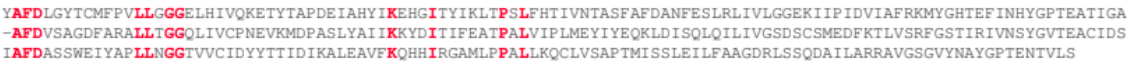
Slika 1.3: Fragmenti - asparaginska kiselina, orintonin i valin [2]

Na slici 1.4 možemo videti da u ovim A domenima imamo jedino 3 pozicije poklapanja (pozicije unutar sekvenci na kojima se nalaze iste aminokoseline).



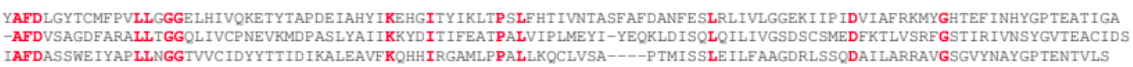
Slika 1.4: 3 pozicije poklapanja u A domenima [2]

Međutim, ako pomerimo drugu sekvencu za jedno mesto u desno (dodavanjem praznog simbola − na početak sekvence), onda vidimo da imamo 11 pozicija poklapanja (slika 1.5).

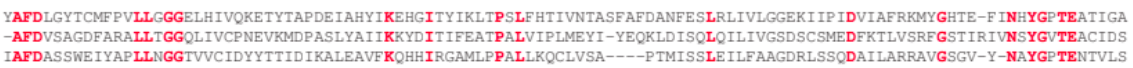


Slika 1.5: 11 pozicija poklapanja u A domenima [2]

Dodavanjem još nekoliko praznih simbola, otkrivamo 14 pozicija poklapanja (slika 1.6). I dodavanjem još malo praznih simbola, dobijamo 19 pozicija poklapanja (slika 1.7).



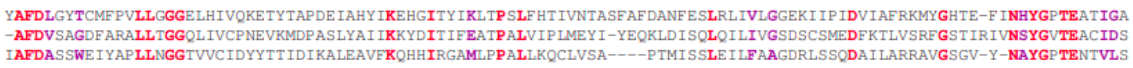
Slika 1.6: 14 pozicija poklapanja u A domenima [2]



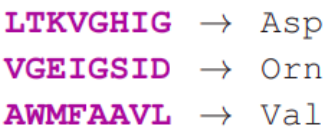
Slika 1.7: 19 pozicija poklapanja u A domenima [2]

Ispostavlja se da crvene pozicije predstavljaju **konzervirano jezgro** koje dele mnogi A domeni. Nakon dugogodišnjih istraživanja u kojima je od velikog značaja

bilo pronalaženje konzerviranih pozicija u A domenima iste NRP sintetaze, otkriveno je da su aminokiseline kodirane nizom nesusednih aminokiselina koje se nalaze na istim pozicijama u nekonzerviranim delovima. Na slici 1.8 možemo videti neribozomalni kod dužine 8 aminokiselina za *Asp*, *Orn* i *Val*, koji je označen ljubičastom bojom. Neribozomalni kod definisan nizom od 8 aminokiselina se naziva **signatura**. Na slici 1.9 možemo videti izdvojeno signature za aminokiseline *Asp*, *Orn* i *Val*.



Slika 1.8: Neribozomalni kod označen ljubičastom bojom [2]



Slika 1.9: Signature za Asp, Orn i Val [2]

Kao što za proteine koji nastaju standardnim postupkom postoji genetski kod, tako za neribozomalne proteine postoji neribozomalni kod kojim su aminokiseline koje čine protein kodirane. Neribozomalni kod nije toliko dobro izučen kao standardni genetski kod i istraživanja su još uvek u toku.

Glava 2

Poravnanje sekvenci

2.1 Poređenje sekvenci

Biološke sekvence su podložne promenama. Na primer, kroz proces ćelijske deobe dolazi do mutacija (zamena jednog nukleotida drugim), insercija (ubacivanje nukleotida) i delecija (brisanje nukleotida). Neke promene mogu da prouzrokuju izmene u funkcionisanju ćelije, a neke mogu da budu neutralne. Jedan od razloga zbog koga vršimo poređenje sekvenci je da bismo utvrdili da li se proteinska ili genomska sekvenca jedne jedinke razlikuje od sekvence istog proteina druge jedinke iste vrste ili od genoma referentnog za tu vrstu. Razlike koje uočimo mogu da pomognu u pronalaženju uzroka promena koje se pojavljuju kod date jedinke. Pored toga, upravo je poređenje sekvenci, konkretno A domena, dovelo do otkrića neribozomalnog genetskog koda za neke aminokiseline, kao što je navedeno u prethodnom poglavlju.

Sekvence poredimo tako što ih poravnavamo. Poravnanje podrazumeva da uparimo što više karaktera jedne sekvence sa karakterima druge sekvence, pri čemu težimo ka tome da upareni karakteri budu jednaki. Ukoliko nije moguće upariti sve karaktere, mogu se upariti različiti karakteri ili ostaviti neupareni karakteri navođenjem praznih simbola ($-$). Na taj način bismo dobili dvorednu matricu karaktera. Prilikom ubacivanja praznih simbola, nastojimo da, kad god je to moguće, u jednoj koloni budu isti karakteri. Posmatrajmo primer poravnanja sekvenci ATGCATGC i TGCATGCA. Kada ne ubacimo prazne simbole, sekvence ATGCATGC i TGCATGCA nemaju kolone koje se poklapaju:

$$\begin{array}{cccccccc} A & T & G & C & A & T & G & C \\ T & G & C & A & T & G & C & A \end{array} \tag{2.1}$$

Ali ako ih poravnamo malo drugačije, vidimo da imaju 6 pozicija koje se poklapaju:

$$\begin{array}{ccccccccc} A & T & G & C & A & T & G & C & - \\ - & T & G & C & A & T & G & C & A \end{array} \tag{2.2}$$

Ali imaju i malo manje uočljive sličnosti, kao na primer:

$$\begin{array}{ccccccccc} A & T & G & C & - & T & T & A & - \\ - & T & G & C & A & T & T & A & A \end{array} \tag{2.3}$$

Poravnanje zapravo predstavlja jedan mogući scenario kako je jedna sekvenca evoluirala u drugu. Ubacivanjem praznog simbola unutar poravnanja modelujemo insercije i delecije, a kolonama u kojima se karakteri ne poklapaju modelujemo mutacije. Postoji više načina na koje možemo poravnati dve sekvence pa zbog toga poravnanja ima smisla nekako rangirati i među njima, prema nekom kriterijumu definisati optimalno poravnanje. Jedan takav kriterijum tj. jedna mera kvaliteta poravnanja je **Hamingovo rastojanje**, koje predstavlja broj nepoklapanja u dve sekvence, pretpostavljajući da i-ti simbol jedne sekvence poredimo sa i-tim simbolom druge sekvence. Prethodni primeri nas navode da definišemo optimalno poravnanje kao ono koje ima najveći broj poklapanja, tj. kao ono kod koga je Hamingovo rastojanje najmanje.

2.2 Optimalno poravnanje sekvenci

Poravnanje dve sekvence možemo definisati kao dvorednu matricu, pri čemu prvi red odgovara prvoj, a drugi red drugoj sekvenci. Prazan simbol − može biti umetnut na bilo koje mesto u obe sekvence, uz pravilo da dva prazna simbola − ne budu poravnata. Primer poravnanja za sekvence ATGTTATA i ATCGTCC možemo videti na sledećoj slici (slika 2.1).

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

Slika 2.1: Dvoredna matrica poravnanja niski ATGTTATA i ATCGTCC [2]

U svakoj matrici poravnanja razlikujemo sledeće tipove kolona:

- poklapanja, kada se u jednoj koloni nalaze isti simboli
- promašaje (nepoklapanja), kada se u jednoj koloni nalaze različiti simboli
- insercije, kada se u prvoj koloni nalazi simbol −
- delecije, kada se u drugoj koloni nalazi simbol −

Poklapanja u poravnanju dve sekvence formiraju njihovu zajedničku podsekvencu. Na primer, za poravnanje sekvenci ATGTTATA i ATCGTCC sa slike 2.1, zajednička podsekvenca je ATGT a njihovo Hamingovo rastojanje je 5. S obzirom da želimo da u poravnanju imamo što više poklapanja, problem poravnanja možemo svesti na problem nalaženja najduže zajedničke podsekvence (problem 1).

PROBLEM 1 (PROBLEM NAJDUŽE ZAJEDNIČKE PODSEKVENCE)

problem: *Naći najdužu zajedničku podsekvencu dve niske.*
ulaz: *Dve niske.*
izlaz: *Najduža zajednička podsekvenca ovih niski.*

Glava 3

Algoritamske tehnike za rešavanje sličnih jednostavnijih problema

Problemom kusura i problemom turiste na Menhetnu se služimo da bismo potencijalna algoritamska rešenja ilustrovali i analizirali najpre na jednostavnijim problemima a zatim primenili na problem poravnanja. Objasnićemo rešenja koristeći rekurzivni pristup i pristup dinamičkim programiranjem i kako ovi pristupi mogu poboljšati pohlepni pristup.

3.1 Problem kusura

Objasnimo problem kusura na primeru. Ako je račun u prodavnici \$69.24, koji kupac želi da plati sa \$70, to znači da prodavac mora da vrati kusur u iznosu od 76 centi. Na prodavcu je da donese odluku kako da vrati kusur, da li će dati 76 novčića od 1 cent ili na primer samo 4 novčića ($25 + 25 + 25 + 1 = 76$). Ovaj jednostavan primer nam pomaže u definisanju opštijeg problema: kako prodavac može da vrati kusur koristeći najmanji broj novčića (problem 2).

Problem 2 (Problem vraćanja kusura)

---

problem: *Naći minimalan broj novčića neophodnih za vraćanje kusura.*

ulaz: *Ceo broj* $\boldsymbol{money}$ *(iznos kusura) i niz pozitivnih celih brojeva* $(\boldsymbol{coin_1}, \boldsymbol{coin_2}, ..., \boldsymbol{coin_d})$ *koji predstavlja raspoložive novčiće.*

izlaz: *Minimalni broj novčića koji rasitnjava sumu* $\boldsymbol{money}$.

3.1.1 Pohlepan pristup

Ako pretpostavimo da na raspolaganju imamo apoene $50, 25, 20, 10, 5, 1$, pohlepnim pristupom bismo kusur u vrednosti od 40 vratili kao $25 + 10 + 5$. Međutim, manje novčića bismo upotrebili ako bismo izabrali dva novčića od po $20 : 20 + 20$. Ovakav pristup problemu vraćanja kusura opisan je sledećim algoritmom 1.

Algoritam 1 Pohlepan pristup

```
function GreedyChange(money)
    change ← empty collection of coins
    while money > 0 do
        change ← largest denomination that is less than or equal to money
        add a coin with denomination coin to the collection of coins change
        money ← money − coin
    return change
```

3.1.2 Rekurzivni pristup

Pošto pohlepan pristup može da propusti tačno rešenje, pokušaćemo drugačije. Na primer, potrebno je da vratimo kusur od 76 centi, ali jedino imamo dostupne apoene vrednosti $5, 4$ i 1 cent. Minimalna kolekcija novčića koja u zbiru daje 76 mora biti jedna od tri:

- minimalna kolekcija novčića koja u zbiru daje 75 centi, plus novčić od 1 cent
- minimalna kolekcija novčića koja u zbiru daje 72 centa, plus novčić od 4 centa
- minimalna kolekcija novčića koja u zbiru daje 71 cent, plus novčić od 5 centi

Za niz celih brojeva $Coins = (coin_1, coin_2, ..., coin_d)$, minimalni broj novčića, $MinNumCoins(money)$, je jednak minimumu sledećih d brojeva:

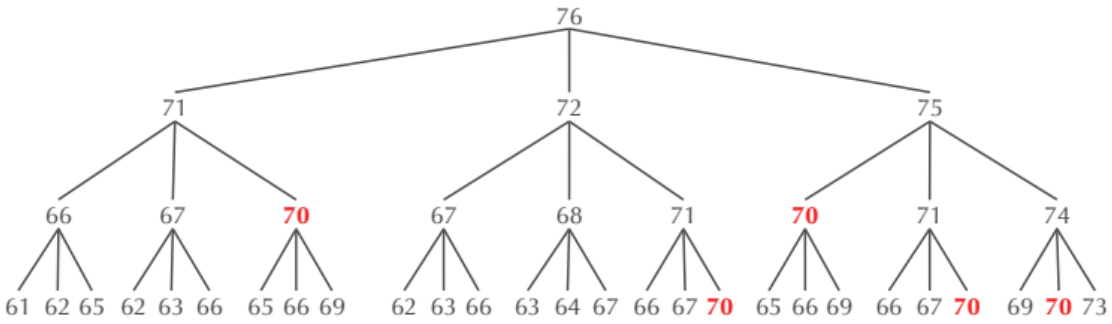
$$MinNumCoins(money) = min \begin{cases} MinNumCoins(money - coin_1) + 1 \\ \vdots \\ MinNumCoins(money - coin_d) + 1 \end{cases} \tag{3.1}$$

Pomoću rekurentne relacije 3.1, definišemo algoritam 2 za rešavanje problema kusura. Možemo primetiti da se algoritam više puta izvršava za istu vrednost. Na primer, za vrednost od 70 centi, algoritam se izvršava čak 6 puta (slika 3.1). Daljim procenama možemo doći do zaključka da se optimalna kombinacija novčića za

Algoritam 2 Rekurzivni pristup

```
function RecursiveChange(money, coins)
    if money = 0 then return 0
    minNumCoins ← ∞
    for i ← 1 to |Coins| do
        if money ≥ coin_i then
            numCoins ← RecursiveChange(money − coin_i, coins)
            if numCoins + 1 < minNumCoins then
                minNumCoins ← numCoins + 1
    return minNumCoins
```

vraćanje kusura od 30 centi izračunava milijardama puta. Možemo zaključiti da rekurzivan pristup nije efikasan za rešavanje ovakvih problema.



Slika 3.1: Primer vraćanja kusura rekurzivnim pristupom, crvenom bojom su označena izračunavanja za vrednost od 70 centi [2]

3.1.3 Pristup dinamičkim programiranjem

Kako bismo izbegli mnogo rekurzivnih poziva, koristićemo tehniku memorizacije. Bilo bi idealno kada bismo znali sve vrednosti $MinNumCoins(money - coin_i)$ u vreme izračunavanja $MinNumCoins(money)$. Suština dinamičkog programiranja je u koraku koji ne deluje toliko intuitivno. Umesto rekurzivnog računanja $MinNumCoins(M)$ za svaku vrednost M od 76 do 1, uradićemo potpuno suprotno, izračunaćemo $MinNumCoins(M)$ za svaku vrednost M od 1 do 76. Sve vrednosti pamtimo i koristimo po potrebi pa bismo tako, umesto vremenski zahtevnih rekurzivnih poziva, jednostavno potražili vrednosti iz unapred izračunate tabele. Rekurentna relacija za izračunavanje $MinNumCoins(money)$ ostaje ista kao kod rekurzivnog pristupa (3.1).

Pristup dinamičkog programiranja rešavanju problema kusura je opisan algoritmom 3. Vremenska složenost ovog algoritma za računanje $MinNumCoins(money)$ je $O(money \cdot |Coins|)$.

Algoritam 3 Dinamičko programiranje

```
function DPChange(money, coins)
    minNumCoins(0) ← 0
    for m ← 1 to money do
        minNumCoins(0) ← ∞
        for i ← 1 to |Coins| do
            if m ≤ coin_i then
                if minNumCoins(m − coin_i) + 1 < minNumCoins(m) then
                    minNumCoins(m) ← minNumCoins(m − coin_i) + 1
    return minNumCoins(money)
```

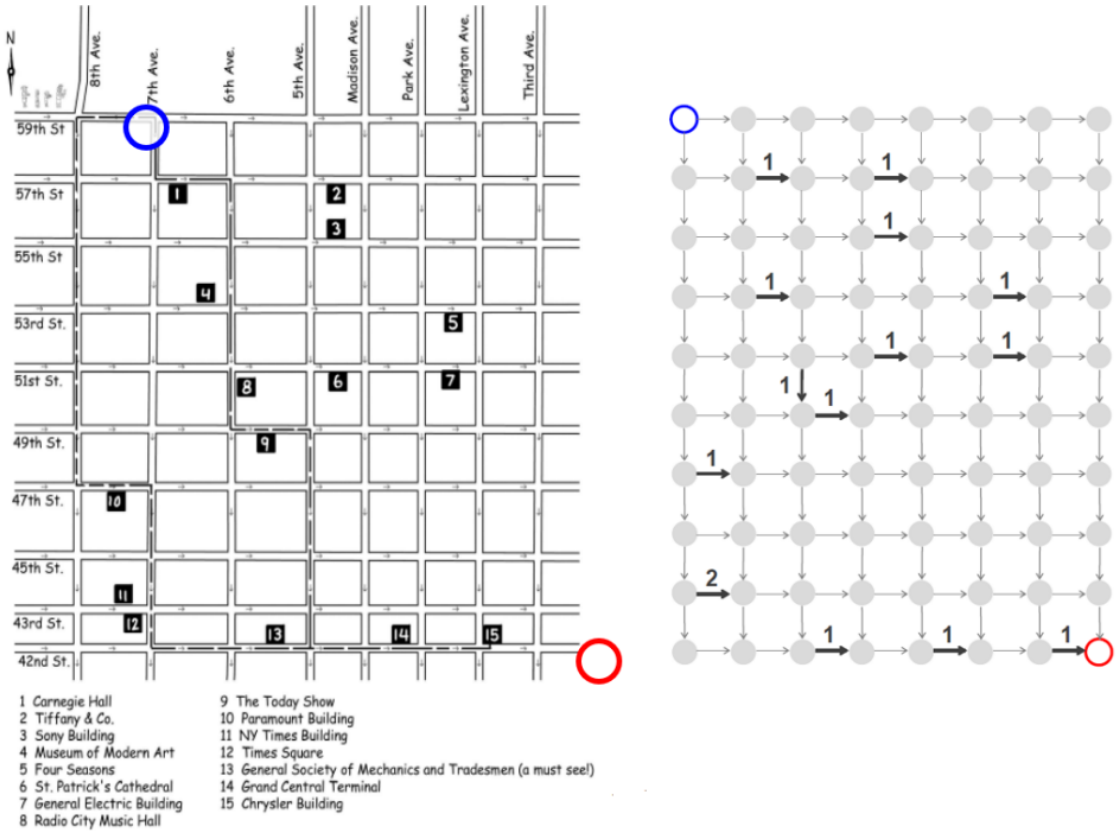
3.2 Problem turiste na Menhetnu

Pretpostavimo da turista na Menhetnu želi da poseti što više znamenitosti za što manje vremena. Na Menhetnu se sve ulice seku pod pravim uglom i pravilo je da se turista na svakoj raskrsnici može kretati jedino južno ($\downarrow$) ili istočno ($\rightarrow$), što znači da ne može da se vrati nazad. Turista može birati između više različitih putanja, ali ne postoji putanja kojom će posetiti sve znamenitosti. Problem pronalaženja putanje kroz grad kojom će turista posetiti najviše znamenitosti se zove Problem turiste na Menhetnu.

Mapu Menhetna možemo predstaviti kao usmeren graf, u kome svaka raskrsnica predstavlja čvor, a svaka ulica sa naznačenim smerom kretanja (južno ili istočno) predstavlja granu u grafu. Svakoj grani pridružimo težinu jednaku broju znamenitosti koje se nalaze u toj ulici. Graf koji posmatramo je usmeren, pravougaoni, težinski i mrežni i takav graf ćemo zvati **Menhetn graf** (slika 3.2). Da bismo rešili ovaj problem, zapravo moramo pronaći putanju maksimalne težine u Menhetn grafu. S obzirom na dozvoljene smerove kretanja, čvor sa koordinatama $(0, 0)$ možemo označiti kao početni, a čvor sa koordinatama (n, m) kao krajnji (slike 3.2 i 3.3a). Na taj način ćemo definisati problem turiste na Menhetnu (problem 3).

PROBLEM 3 (PROBLEM TURISTE NA MENHETNU)

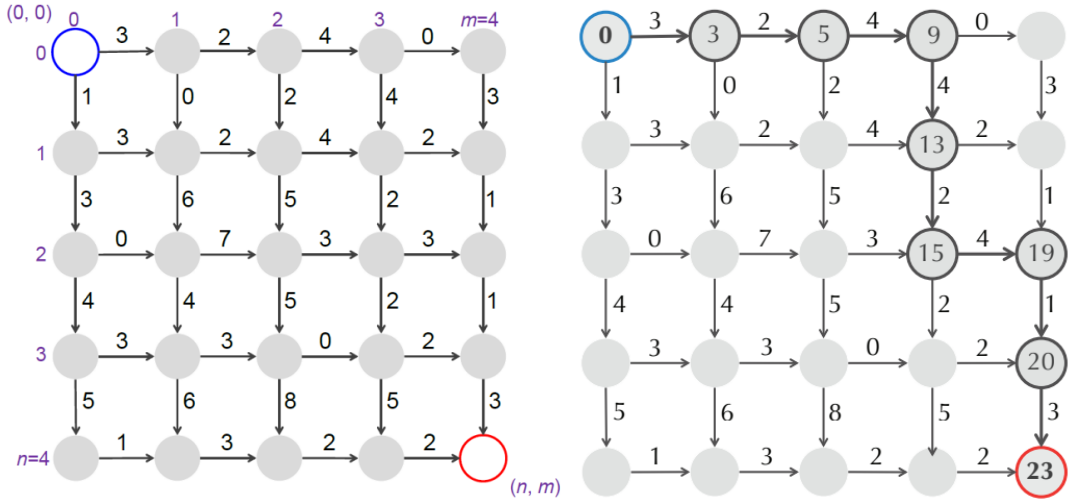
problem: *Naći najtežu putanju u Menhetn grafu.*
ulaz: *Menhetn graf dimenzije $n \times m$*
izlaz: *Putanja maksimalne težine od početnog (source) do krajnjeg (sink) čvora u Menhetn grafu.*



Slika 3.2: Mapa Menhetna i usmereni graf koji je opisuje [2]

3.2.1 Pohlepan pristup

Primena grube sile na ovaj problem zahteva mnogo vremena zato što je ukupan broj svih mogućih putanja veliki. Pohlepan pristup bi izabrao jedan od dva pravca (južno ili istočno), u zavisnosti koliko znamenitosti možemo posetiti krećući se južno ili istočno. Postoji mogućnost da ovakav pristup zanemari najtežu moguću putanju. Na primer, posmatrajmo sliku 3.3b gde vrednosti unutar čvorova predstavljaju vrednost najteže putanje do tog čvora. Vrednost najteže putanje do krajnjeg

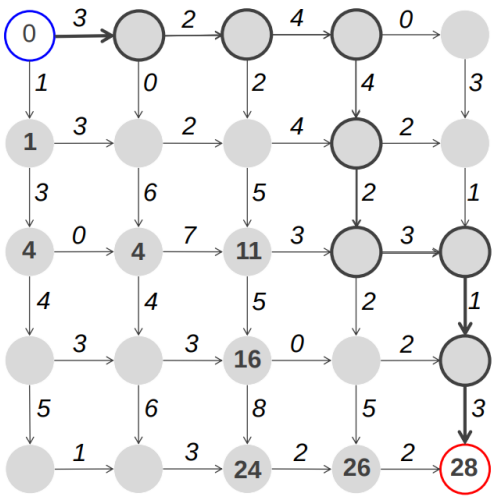


(a) Menhetn graf [3]

(b) Putanja maksimalne težine u Menhetn grafu dobijena primenom pohlepnog pristupa [2]

Slika 3.3

čvora dobijena pohlepnim pristupom iznosi 23. Međutim, to nije putanja maksimalne težine. Putanju maksimalne težine možemo videti na slici 3.4.



Slika 3.4: Putanja maksimalne težine u Menhetn grafu [4]

3.2.2 Rekurzivni pristup

Algoritam 4 izračunava najtežu putanju do čvora (i, j) u Menhetn grafu uz ograničenje da do čvora (i, j) jedino možemo doći krećući se južno (iz čvora $(i-1, j)$)

ili istočno (iz čvora $(i, j-1)$). Slično kao kod rekurzivnog pristupa vraćanju kusura, algoritam 4 se izvršava više puta za istu vrednost.

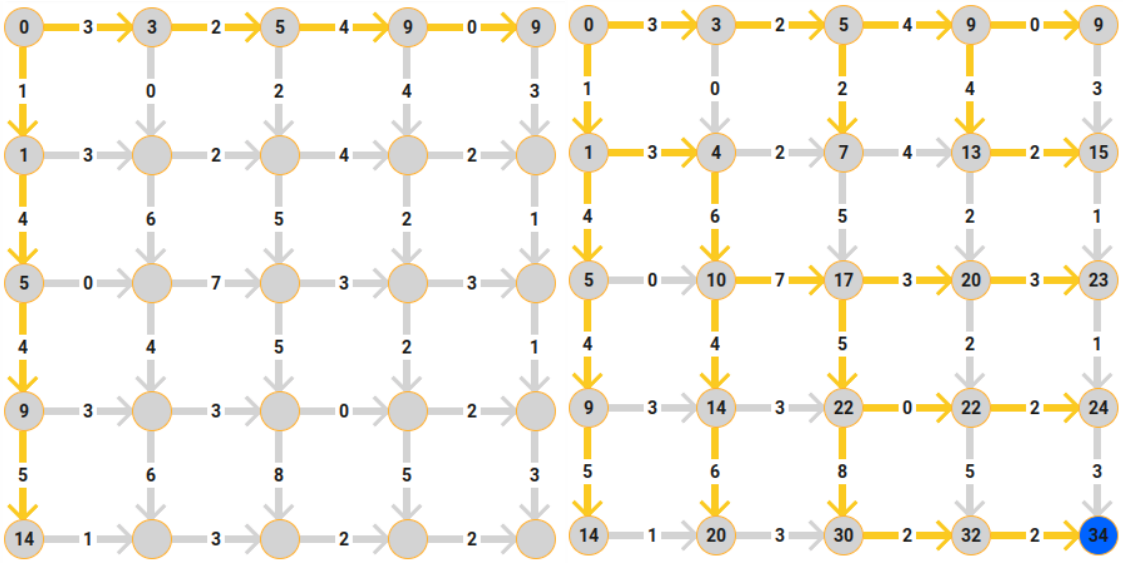
Algoritam 4 Menhetn turista rekurzivnim pristupom

```
function ManhattanRecursive(i, j)
    if i = 0 and j = 0 then return 0
    x ← −∞
    y ← −∞
    if i > 0 then
        x ←ManhattanRecursive(i − 1, j) + weight of vertical edge into (i, j)
    if j > 0 then
        y ←ManhattanRecursive(i, j − 1) + weight of horizontal edge into (i, j)
    return max{x, y}
```

3.2.3 Pristup dinamičkim programiranjem

Pristupom dinamičkog programiranja, da bismo pronašli najtežu putanju od početnog čvora $(0, 0)$ do krajnjeg (n, m), prvo ćemo pronaći težine svih putanja od početnog do svih čvorova (i, j) u mreži, krećući se od početnog pa sve do krajnjeg čvora. Na prvi pogled, kao kod vraćanja kusura, može nam se činiti da rešavamo $m \times n$ drugih problema, umesto jednog problema. Međutim, ideja iza pristupa dinamičkim programiranjem je da prvo rešimo svaki od manjih problema jednom, umesto milionima puta. Označićemo sa $S_{i,j}$ vrednost najteže putanje od početnog čvora do čvora (i, j) u grafu (vrednost $S_{n,m}$ označava vrednost najteže putanje od početnog do krajnjeg čvora). Prvo računamo težine svih putanja do čvorova po obodu grafa (slika 3.5a), zato što oni imaju samo po jednu ulaznu granu. Nakon toga, red po red, izračunavamo odgovarajuće težine $S_{i,j}$ za ostale čvorove prateći relaciju 3.2.

Kako bi na kraju bilo moguće rekonstruisati putanju, pošto ulazni čvorovi imaju po dve ulazne grane, potrebno je i zapamtiti kojom granom smo došli do datog čvora. Na slici 3.5b su ove grane označene žutom bojom. Da bismo rekonstruisali putanju, dovoljno je da krenemo žutim granama od krajnjeg čvora u suprotnom smeru. Ako sa $down_{i,j}$ i $right_{i,j}$ označimo težine odgovarajućih grana, rešenje problema turiste na Menhetnu dinamičkim pristupom možemo opisati algoritmom 5.



(a) Težine za čvorove po obodu grafa

(b) Težine za sve čvorove grafa i najteža putanja od početnog do krajnjeg čvora

Slika 3.5

$$S_{i,j} = max \begin{cases} S_{i-1,j} + \text{težina vertikalne grane od } (i-1,j) \text{ do } (i,j) \\ \vdots \\ S_{i,j-1} + \text{težina horizontalne grane od } (i,j-1) \text{ do } (i,j) \end{cases} \tag{3.2}$$

Algoritam 5 Menhetn turista dinamičkim pristupom

```
function ManhattanDynamic(n, m, down, right)
    S_{0,0} ← 0
    for i ← 1 to n do
        S_{i,0} ← S_{i-1,0} + down_{i,0}
    for j ← 1 to m do
        S_{0,j} ← S_{0,j-1} + right_{0,j}
    for j ← 1 to m do
        for i ← 1 to n do
            S_{i,j} ← max{S_{i-1,j} + down_{i,j}, S_{i,j-1} + right_{i,j}}
    return S_{n,m}
```

Glava 4

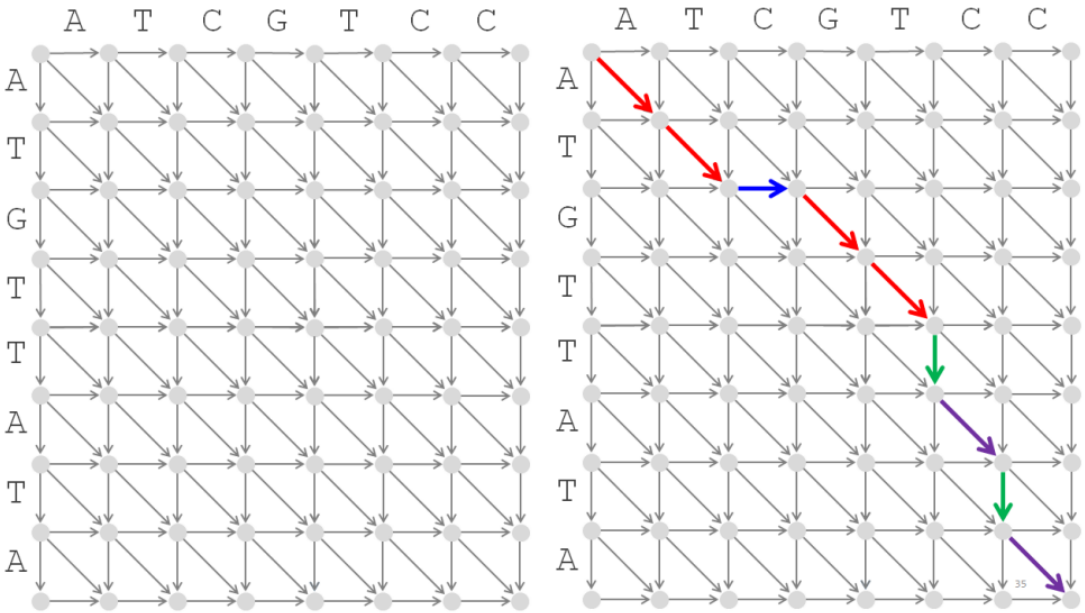
Problem poravnanja i algoritmi za njegovo rešavanje

U prethodnoj glavi smo videli kako se problem turiste na Menhetnu modeluje kao graf. Razmotrimo da li na sličan način možemo da modelujemo graf koji bi odgovarao problemu poravnanja. Za datu matricu poravnanja, graf koji bi bio analogon Menhetn grafu možemo da konstruišemo na sledeći način:

- Kolone grafa označimo karakterima iz prve niske
- Vrste grafa označimo karakterima iz druge niske
- U svaku presečnu tačku postavimo po jedan čvor
- Gde god je moguće, postavimo vertikalne (insercija), horizontalne (delecija) i dijagonalne grane (poklapanje ili promašaj)
- Dijagonalne grane koje odgovaraju poklapanjima otežamo koeficijentom 1, ostale grane koeficijentom 0 (suština je da povoljniji ishod (poklapanje) otežamo više nego nepovoljniji (promašaj), a same težine se mogu izabrati na razne načine o čemu će biti reči u nastavku)
- Čvor $(0, 0)$ označimo kao početni čvor u grafu
- Čvor (n, m) označimo kao krajnji čvor u grafu

Na slici 4.1 predstavljen je graf poravnanja za sekvence ATGTTATA i ATCGTCC. Da bismo pronašli njihovo optimalno poravnanje, potrebno je da nađemo najtežu

putanju između početnog i krajnjeg čvora. Kada poredimo ovaj graf sa grafom problema turiste sa Menhetna (slika 3.3a), vidimo da pored vertikalnih i horizontalnih, ovaj graf sadrži i dijagonalne grane. Problem turiste na Menhetnu predstavlja specijalan slučaj problema najduže zajedničke podsekvence (problem 1).



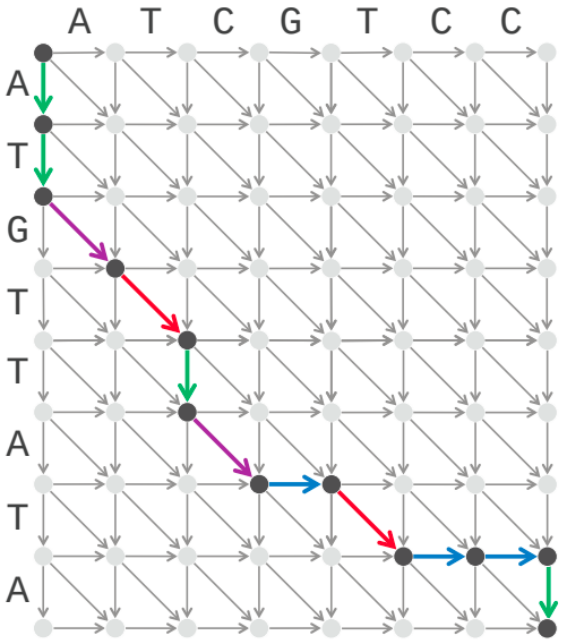
Slika 4.1: Problem poravnanja predstavljen usmerenim mrežnim grafom [3]

Za svako dato poravnanje između dve sekvence, putanju između početnog i krajnjeg čvora u grafu možemo konstruisati na sledeći način: počevši od prve kolone poravnanja, za svako poklapanje i promašaj u putanju dodamo dijagonalnu granu, za svaku inserciju horizontalnu a za svaku deleciju vertikalnu granu. Na primer, neka su sekvence ATGTTATA i ATCGTCC poravnate na sledeći način:

$$\begin{array}{cccccccccc} A & T & G & T & T & - & A & T & A \\ - & - & A & T & C & G & T & C & C \end{array} \tag{4.1}$$

Za dato poravnanje, na slici 4.2 možemo videti konstruisanu putanju u grafu poravnanja. Možemo primetiti da svakom poravnanju odgovara tačno jedna putanja u grafu, ali i da svakoj putanji odgovara tačno jedno poravnanje.

Primetimo da su problem turiste na Menhetnu i problem poravnanja zapravo specijalni slučajevi problema pronalaženja najteže putanje u usmerenom težinskom acikličnom grafu, koji nije mrežni i koji ima proizvoljno postavljene grane. O tome zašto graf ne sme sadržati cikluse i koji uslovi moraju biti ispunjeni da bismo mogli da pronađemo najtežu putanju biće reči u nastavku.



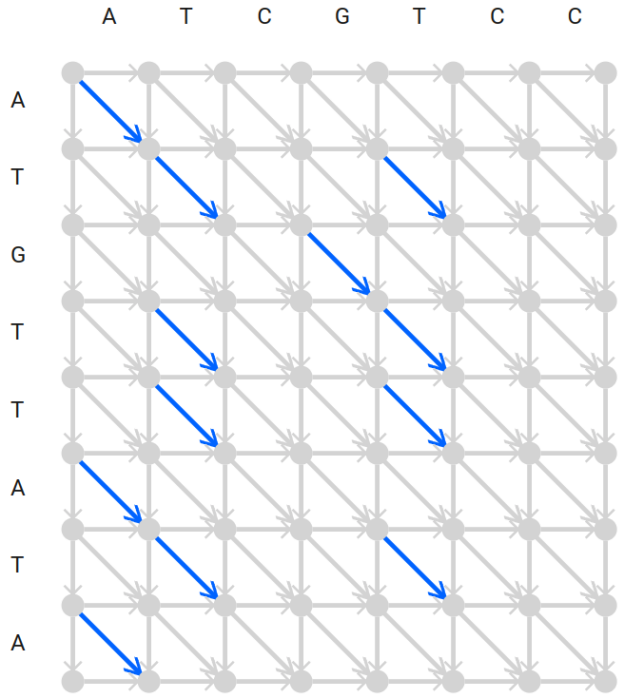
Slika 4.2: Putanja za jedno poravnanje sekvenci ATGTTATA i ATCGTCC [2]

4.1 Dinamički pristup problemu poravnanja

Princip dinamičkog programiranja primenjen na problem Menhetn turiste možemo primeniti i za pronalaženje najteže putanje u grafu poravnanja. Da bi se algoritam 5 primenio na problem poravnanja, neophodno je da izmenimo rekurentnu relaciju kojom se računa najteža putanja do čvora (i, j). Kao i kod problema Menhetn turiste, sa $S_{i,j}$ ćemo označiti vrednost najteže putanje od početnog čvora do čvora (i, j) u grafu (vrednost $S_{n,m}$ označava vrednost najteže putanje od početnog do krajnjeg čvora). Definisaćemo rekurentnu relaciju 4.2, uzimajući u obzir da su težine grana kao na slici 4.3, a gde su v i w sekvence koje poredimo.

$$S_{i,j} = max \begin{cases} S_{i-1,j} + 0 \\ S_{i,j-1} + 0 \\ S_{i-1,j-1} + 1, v_i = w_j \\ S_{i-1,j-1} + 0, v_i \neq w_j \end{cases} \tag{4.2}$$

Slično kao kod Menhetn turiste, potrebno je da rekonstruišemo najtežu putanju. U slučaju grafa poravnanja, rekonstruisana putanja predstavlja najdužu zajedničku podsekvencu niski v i w. Najtežu putanju ćemo dobiti pomoću matrice *backtrack* koja se računa uz pomoć rekurentne relacije 4.3 polaskom od krajnjeg čvora. U ovoj



Slika 4.3: Plave grane (dijagonalne grane koje odgovaraju istim karakterima) imaju težinu 1, ostale težinu 0, v_i i w_j oznake vrste i kolone

matrici, za čvor (i, j) se na poziciji (i, j) nalazi informacija kojom granom smo došli do tog čvora (vrednosti matrice su: $\downarrow$, $\rightarrow$ i $\searrow$). Matrice *backtrack* i S formiramo u isto vreme na način prikazan algoritmom 6. Algoritam vraća vrednost matrice *backtrack*, koju dalje koristimo za pronalaženje najduže zajedničke podsekvence.

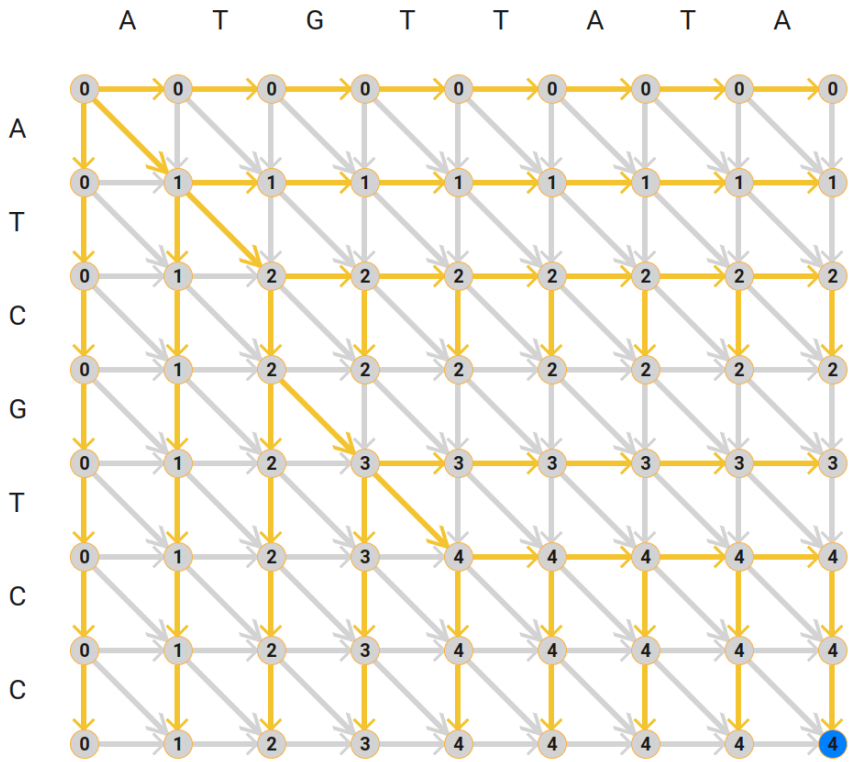
$$backtrack_{i,j} = max \begin{cases} \rightarrow & , S_{i,j} = S_{i,j-1} \\ \downarrow & , S_{i,j} = S_{i-1,j} \\ \searrow & , otherwise \end{cases} \tag{4.3}$$

Na primer za niske ATCGTCC i ATGTTATA na slici 4.4 možemo videti vrednosti matrice *backtrack* obeležene žutom bojom. Na osnovu matrice *backtrack* potrebno je rekonstruisati najtežu putanju i najdužu zajedničku podsekvencu, što je opisano algoritmom 7.

Poziv funkcije OutputLCS($backtrack, v, i, j$), na osnovu prethodno izračunate matrice *backtrack*, vraća najdužu zajedničku podsekvencu za prefikse niski v i w redom dužina i i j, a poziv OutputLCS($backtrack, v, |v|, |w|$) vraća najdužu zajedničku podsekvencu za niske v i w.

Algoritam 6 Putokazi za povratak

function LCSBackTrack(v, w)
  for $i \leftarrow 0$ to $|v|$ **do**
    $S_{i,0} \leftarrow 0$
  for $j \leftarrow 0$ to $|w|$ **do**
    $S_{0,j} \leftarrow 0$
  for $j \leftarrow 1$ to $|v|$ **do**
    for $i \leftarrow 1$ to $|w|$ **do**
      $match = 0$
      if $v_i = w_j$ **then**
        $match = 1$
      $S_{i,j} \leftarrow max\{S_{i-1,j}, S_{i,j-1}, S_{i-1,j-1} + match\}$
      if $S_{i,j} = S_{i-1,j}$ **then**
        $bakctrack_{i,j} \leftarrow \downarrow$
      else if $S_{i,j} = S_{i,j-1}$ **then**
        $bakctrack_{i,j} \leftarrow \rightarrow$
      else
        $bakctrack_{i,j} \leftarrow \searrow$
  return *backtrack*



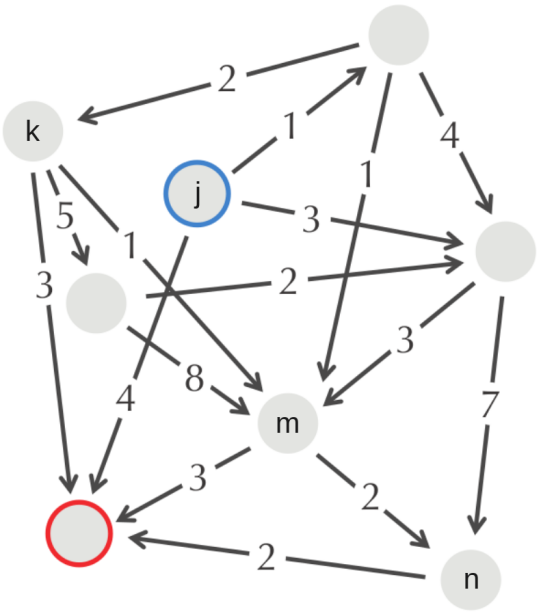
Slika 4.4: Putokazi za rekonstrukciju najteže putanje

Algoritam 7 Najduža zajednička podsekvenca

```
function OutputLCS(backtrack, v, i, j)
    if i = 0 or j = 0 then
        return ""
    if backtrack_{i,j} = ↓ then
        return OutputLCS (backtrack, v, i − 1, j)
    else if backtrack_{i,j} = → then
        return OutputLCS(backtrack, v, i, j − 1)
    else
        return OutputLCS(backtrack, v, i − 1, j − 1) + v[i − 1]
```

4.2 Najteža putanja u proizvoljnom usmerenom ackiličnom grafu

Pored poravnanja konstruisanog pomoću algoritma 6, postoje i razne druge, kompleksnije vrste poravnanja koja se ne mogu modelovati mrežnim grafovima. Zbog toga ćemo, uz pomoć principa dinamičkog programiranja, predstaviti rešenje za opštiji problem koji podrazumeva pronalaženje najteže putanje u usmerenom težinskom grafu koji nema mrežnu topologiju (slika 4.5).



Slika 4.5: Usmereni težinski aciklični graf [2]

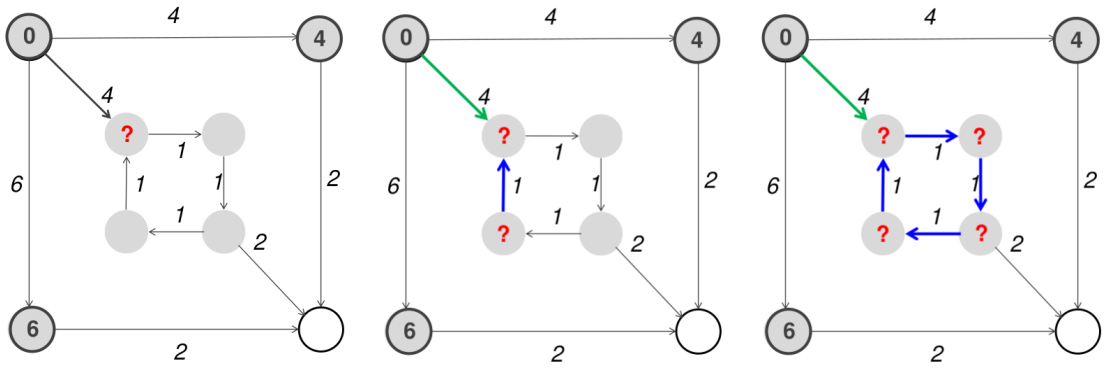
Ako čvor u grafu označimo slovom b, onda ćemo sa S_b označiti najtežu putanju od početnog čvora do čvora b. Za svaki čvor a kažemo da je predak čvora b ako u

grafu postoji grana koja povezuje čvorove a i b. Slično kao kod grafa poravnanja, definisaćemo rekurentnu relaciju 4.4 za računanje vrednosti S_b za proizvoljni usmereni aciklični graf.

$$S_b = max_{all\ predecessors\ a\ of\ node\ b}\{S_a + weight\ of\ edge\ from\ a\ to\ b\} \tag{4.4}$$

Na primer, ako posmatramo čvor označen crvenom bojom na slici 4.5, vidimo da do ovog čvora možemo doći iz četiri druga čvora, označena slovima k, j, m i n. Na osnovu relacije 4.4, najteža putanja do crvenog čvora je $max\{S_k + 3, S_j + 4, S_m + 3, S_n + 2\}$.

Posmatrajmo graf na slici 4.6. Želeli bismo da izračunamo težinu najteže putanje od početnog čvora do čvora označenog upitnikom. Da bismo izračunali najtežu putanju do ovog čvora, moramo obići sve njegove pretke. Ako pokušamo da izračunamo težinu za sve pretke, vratićemo se do posmatranog čvora. Možemo zaključiti da se rekurentna relacija ne može primeniti na sve grafove već samo na grafove koji ne sadrže cikluse, odnosno na usmerene aciklične grafove.



Slika 4.6: Usmereni težinski graf koji sadrži ciklus [4]

Topološko sortiranje

Posmatrajmo ponovo sliku 4.5, gde je plavom bojom označen početni, a crvenom krajnji čvor. Postavljamo pitanje kako da pronađemo najtežu putanju do krajnjeg čvora kod grafa koji nije mreža. Kod mrežnog grafa, prvo bismo obilazili sve čvorove po obodu, pa onda po unutrašnjosti, i time bismo bili sigurni da smo obišli sve pretke datog čvora. Pitamo se kako da budemo sigurni da smo obišli sve pretke čvora u grafu, ako taj graf nema mrežnu topologiju. Da bismo pronašli najtežu putanju, prvo moramo da definišemo redosled čvorova tako da za čvor b računamo vrednost

S_b nakon svih njegovih predaka. Tako definisan redosled, koji obezbeđuje da pre svakog čvora obiđemo sve njegov pretke, naziva se **topološko sortiranje** čvorova grafa. Ako za niz čvorova $(a_1, ..., a_k)$ važi da će se svaki čvor a_i nalaziti ispred čvora a_j ako u grafu postoji grana od a_i do a_j za $i < j$, onda kažemo da je takav niz topološki sortiran. Može se dokazati da za svaki usmereni aciklični graf postoji topološko sortiranje, i da takvo sortiranje može biti konstruisano u vremenu proporcionalnom broju grana u grafu ($O(\#edges)$). Nakon ovako definisanog redosleda čvorova, možemo definisati algoritam 8 za računanje najteže putanje u proizvoljnom usmerenom acikličnom grafu.

Algoritam 8 Najteža putanja

function LONGESTPATH(*Graph*, *source*, *sink*)
  for each node b in *Graph* **do**
    $S_b \leftarrow -\infty$
  $S_{source} \leftarrow 0$
  topologically order *Graph*
  for each node b in *Graph* (following the topological order) **do**
    $S_b \leftarrow max_{\textit{all predecessors } a \textit{ of node } b}\{S_a + \text{weight of edge from } a \text{ to } b\}$
  return S_{sink}

Glava 5

Vrste poravnanja

Do sada smo podrazumevali da je optimalno poravnanje dve sekvence ono koje ima najveći broj poklapanja, tj. ono kod koga je Hamingovo rastojanje najmanje. Problem poravnanja smo svodili na problem pronalaženja najduže zajedničke podsekvence. Ovako konstruisano poravnanje ima veći broj poklapanja, ali i insercija i delecija. Poravnanje u biološkom smislu može imati različita značenja. Važno je da možemo da kontrolišemo da li u optimalnom poravnanju želimo da dobijemo manje ili više insercija/delecija kao i da li da budu na početku/kraju niske ili u sredini. To možemo postići pomoću mere kvaliteta poravnanja koja je kompleksnija a time i informativnija od Hamingovog rastojanja. Uvešćemo pojam **skora poravnanja**, na osnovu koga ćemo definisati različite vrste poravnanja.

5.1 Globalno i lokalno poravnanje

Ako skor za promašaj (*mismatch*) označimo sa μ, a skor za insercije i delecije (*indel*) sa σ, onda skor poravnanja možemo računati kao:

$$\#matches - \mu \cdot \#mismatches - \sigma \cdot \#indels$$

Pošto su neke mutacije više verovatne od drugih, to znači da svaki promašaj, inserciju i deleciju možemo različito ceniti zavisno koji simboli su uključeni. Tada ima smisla ove vrednosti predstaviti matrično i takvu matricu nazivamo **matricom skora**. Na slici 5.1a možemo videti matricu skora DNK sekvenci kada su promašaji kažnjeni skorom μ a insercije i delecije skorom σ. Ovakve matrice ne moraju biti simetrične, jer se na primer češće dešava da G mutira u T nego obrnuto, na slici 5.1b možemo videti primer jedne takve matrice. Slične matrice se koriste i kod

poravnanja proteinskih sekvenci. Primer takve matrice možemo videti na slici 5.2 (PAM matrica).

	A	C	G	T	–
A	+1	-μ	-μ	-μ	-σ
C	-μ	+1	-μ	-μ	-σ
G	-μ	-μ	+1	-μ	-σ
T	-μ	-μ	-μ	+1	-σ
–	-σ	-σ	-σ	-σ	

(a) Simetrična matrica skora DNK sekvenci [2]

	A	C	G	T	–
A	+1	−3	−5	−1	−3
C	−4	+1	−3	−2	−3
G	−9	−7	+1	−1	−3
T	−3	−5	−8	+1	−4
–	−4	−2	−2	−1	

(b) Nesimetrična matrica skora DNK sekvenci [3]

Slika 5.1

		Ala	Arg	Asn	Asp	Cys	Gln	Glu	Gly	His	Ile	Leu	Lys	Met	Phe	Pro	Ser	Thr	Trp	Tyr	Val
		A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
Ala	A	13	6	9	9	5	8	9	12	6	8	6	7	7	4	11	11	11	2	4	9
Arg	R	3	17	4	3	2	5	3	2	6	3	2	9	4	1	4	4	3	7	2	2
Asn	N	4	4	6	7	2	5	6	4	6	3	2	5	3	2	4	5	4	2	3	3
Asp	D	5	4	8	11	1	7	10	5	6	3	2	5	3	1	4	5	5	1	2	3
Cys	C	2	1	1	1	52	1	1	2	2	2	1	1	1	1	2	3	2	1	4	2
Gln	Q	3	5	5	6	1	10	7	3	7	2	3	5	3	1	4	3	3	1	2	3
Glu	E	5	4	7	11	1	9	12	5	6	3	2	5	3	1	4	5	5	1	2	3
Gly	G	12	5	10	10	4	7	9	27	5	5	4	6	5	3	8	11	9	2	3	7
His	H	2	5	5	4	2	7	4	2	15	2	2	3	2	2	3	3	2	2	3	2
Ile	I	3	2	2	2	2	2	2	2	2	10	6	2	6	5	2	3	4	1	3	9
Leu	L	6	4	4	3	2	6	4	3	5	15	34	4	20	13	5	4	6	6	7	13
Lys	K	6	18	10	8	2	10	8	5	8	5	4	24	9	2	6	8	8	4	3	5
Met	M	1	1	1	1	0	1	1	1	1	2	3	2	6	2	1	1	1	1	1	2
Phe	F	2	1	2	1	1	1	1	1	3	5	6	1	4	32	1	2	2	4	20	3
Pro	P	7	5	5	4	3	5	4	5	5	3	3	4	3	2	20	6	5	1	2	4
Ser	S	9	6	8	7	7	6	7	9	6	5	4	7	5	3	9	10	9	4	4	6
Thr	T	8	5	6	6	4	5	5	6	4	6	4	6	5	3	6	8	11	2	33	6
Trp	W	0	2	0	0	0	0	0	0	1	0	1	0	0	1	0	1	0	55	1	0
Tyr	Y	1	1	2	1	3	1	1	1	3	2	2	1	2	15	1	2	2	3	31	2
Val	V	7	4	4	4	4	4	4	4	5	4	15	10	4	10	5	5	5	72	4	17

Slika 5.2: Primer matrice skora koja se koristi za poravnanje proteinskih sekvenci (PAM matrica) [5]

Sada ćemo definisati problem globalnog poravnanja (problem 4) koji pored dve niske, za ulaz prima i matricu skora.

PROBLEM 4 (PROBLEM GLOBALNOG PORAVNANJA)

problem: *Pronaći poravnanje sa najvišim skorom između dve niske za datu matricu skora.*

ulaz: *Dve niske* $\boldsymbol{v}$ *i* $\boldsymbol{w}$*, kao i matrica skora* ***score****.*

izlaz: *Poravnanje niski* $\boldsymbol{v}$ *i* $\boldsymbol{w}$ *čiji je skor poravnanja (defnisan matricom skora* ***score****) maksimalan od svih mogućih poravnanja za* $\boldsymbol{v}$ *i* $\boldsymbol{w}$*.*

Označimo matricu skora sa *score*. Da bismo rešili ovaj problem, uzevši u obzir izmenu u načinu računanja skora poravnanja, definisaćemo rekurentnu relaciju 5.1 za računanje najteže putanje S u grafu poravnanja.

$$S_{i,j} = max \begin{cases} S_{i-1,j} + score(v_i, -) \\ S_{i,j-1} + score(-, w_j) \\ S_{i-1,j-1} + score(v_i, w_j) \end{cases} \tag{5.1}$$

Jedan od primera gde globalno poravnanje ne uspeva da otkrije biološke povezanosti je poređenje *homeobox* gena. Ovi geni su odgovorni za razvoj embriona i prisutni su u velikom broju vrsta. *Homeobox* geni su dugački i dosta se razlikuju između vrsta, ali oko 60 aminokiselina dug region u svakom proteinu koji ovi geni kodiraju je najčešće konzerviran i naziva se **homeodomen**. Na slici 5.3 možemo videti prikazane homeodomene čoveka i miša.

```
...ARRSRTHFTKFQTDILIEAFEKNRFPGIVTREKLAQQTGIPESRIHIWFQNRRARHPDPG...
...ARQKQTFITWTQKNRLVQAFERNPFPDTATRKKLAEQTGLQESRIQMWFQKQRSLYLKKS...
```

Slika 5.3: Homeodomeni čoveka i miša [2]

Globalno poravnanje traži sličnosti između cele dve niske karaktera. Međutim kada tražimo homeodomene, od značaja su manji lokalni regioni sličnosti. Na slici 5.4 možemo videti globalno poravnanje za dve niske karaktera, a na slici 5.5 možemo videti drugačije poravnanje koje sadrži konzerviran region između dve podniske CAGTCTATGTCAG i CAGTTATGTTCAG.

```
GCCGCCGTCGTTT-TCAG----CA-GTT-ATGTTCAGAT
GCC-CAGTC-TATGTCAGGGGGCACGAGCATG--CACA-
```

Slika 5.4: Globalno poravnanje niski karaktera

```
GCCGCCGTCGTTTTCAGCAGT-TATGTTCAG-----A------T-----
-------------GC-CCAGTCTATGT-CAGGGGGCACGAGCATGCACA
```

Slika 5.5: Lokalno poravnanje niski karaktera

Kada su biološki značajne sličnosti prisutne u nekim delovima sekvenci v i w, umesto da tražimo globalno poravnanje sa najvećim skorom, želimo da pronađemo globalno poravnanje između svih podniski jedne i druge sekvence sa najvećim skorom. Ovakvo poravnanje se naziva lokalno poravnanje.

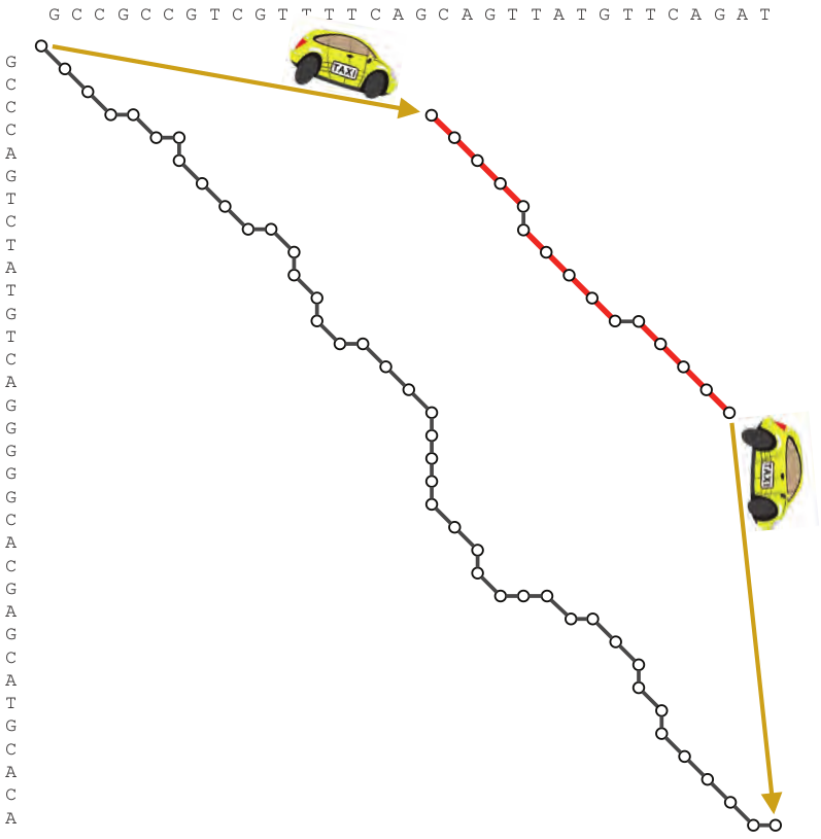
PROBLEM 5 (PROBLEM LOKALNOG PORAVNANJA)

problem: *Pronaći lokalno poravnanje sa najvišim skorom između dve niske za datu matricu skora.*

ulaz: *Dve niske* $\boldsymbol{v}$ *i* $\boldsymbol{w}$*, kao i matrica skora* ***score****.*

izlaz: *Poravnanje niski* $\boldsymbol{v}$ *i* $\boldsymbol{w}$ *čije je globalno poravnanje (defnisano matricom skora* ***score****) maksimalno od svih mogućih globalnih poravnanja za sve moguće podniske niski* $\boldsymbol{v}$ *i* $\boldsymbol{w}$*.*

Problem lokalnog poravnanja možemo rešiti tako što ćemo pronaći globalno poravnanje za svaka dva čvora u grafu poravnanja i izabrati ono sa najvećim skorom. Ako u grafu poravnanja imamo ukupno $\#nodes$ čvorova, onda bismo algoritam za pronalaženje globalnog poravnanja morali da pokrenemo $O(\#nodes^2)$ puta. Pokušaćemo da pronađemo efikasniji način. Zamislimo besplatnu vožnju od početnog čvora u grafu poravnanja do čvora koji označava početak konzerviranog intervala, i takođe zamislimo besplatnu vožnju od čvora koji označava kraj konzerviranog intervala i krajnjeg čvora u grafu poravnanja (slika 5.6).



Slika 5.6: Besplatne vožnje [2]

Ako bismo pronašli način kojim bismo detektovali te besplatne vožnje, tj. ukinuli kažnjavanje od početnog čvora do prvog čvora konzerviranog intervala i od poslednjeg čvora konzerviranog intervala do krajnjeg čvora, došli bismo do lokalnog poravnanja. Da bismo graf poravnanja prilagodili na ovaj način, dovoljno je da dodamo grane od početnog do svih drugih čvorova, kao i od svakog čvora do krajnjeg čvora grafa. Za sekvence v i w ukupan broj grana u ovakvom grafu sa dodavanjem besplatnih vožnji je $O(|v| \cdot |w|)$ što je i vremenska složenost algoritma za nalaženje najteže putanje. Izmenićemo rekurentnu relaciju za izračunavanje vrednosti $S_{i,j}$, sada umesto 3 imamo 4 ulazne grane u čvor (i, j) (relacija 5.2).

$$S_{i,j} = max \begin{cases} 0 \\ S_{i-1,j} + score(v_i, -) \\ S_{i,j-1} + score(-, w_j) \\ S_{i-1,j-1} + score(v_i, w_j) \end{cases} \tag{5.2}$$

Pomoću relacije 5.2 modelovane su besplatne vožnje od početnog ali ne i do krajnjeg čvora. Pošto postoje grane od svakog do krajnjeg čvora, $S_{n,m}$ možemo izračunati kao maksimum po svim ostalim $S_{i,j}$:

$$S_{n,m} = max_{0 \leq i \leq n, 0 \leq j \leq m} S_{i,j}$$

5.2 Afine kazne za praznine u poravnanju sekvenci

Kod globalnog poravnanja korišćen je linearni model za računanje skora: ako je σ kazna za insercije i delecije, onda je $k \cdot \sigma$ kazna za interval od k insercija i delecija. Mutacije su često prouzrokovane greškama prilikom replikacije DNK, koje se često sastoje iz insercije ili delecije celog intervala od k nukleotida. Zbog toga je preterano kažnjavati takve mutacije sa kaznom $k \cdot \sigma$. Na primer, ako posmatramo dva poravnanja sa slike 5.7, želeli bismo da skor poravnanja sa leve strane slike bude manji nego na desnoj strani slike.

```
GATCCAG        GATCCAG
GA-C-AG        GA--CAG
```

Slika 5.7: Dva poravnanja sa istim skorom [2]

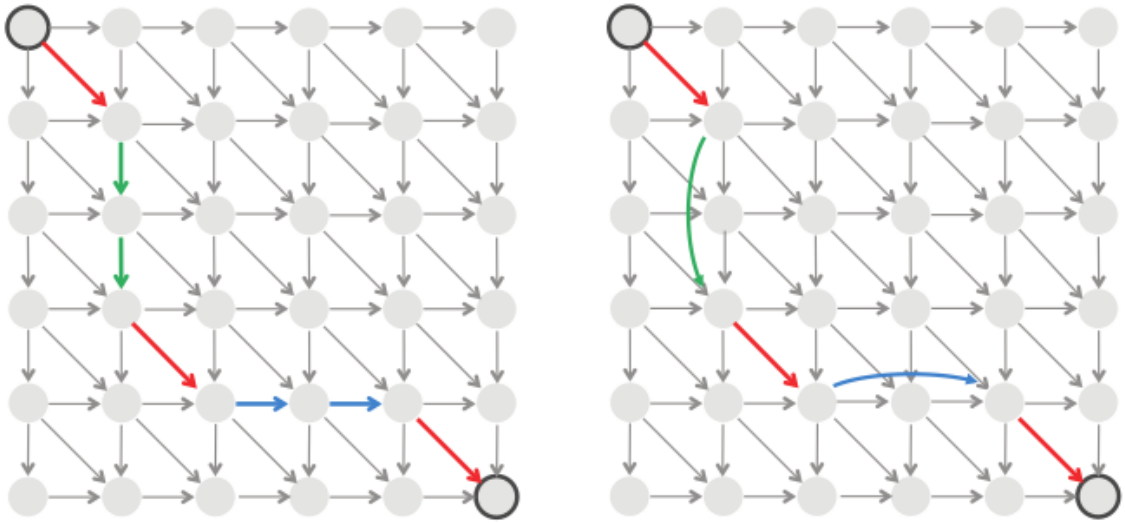
Definišimo prazninu kao sekvencu uzastopnih $-$ simbola u poravnanju. Za prazninu dužine k ćemo definisati **afinu kaznu** koja iznosi $\sigma + \epsilon \cdot (k-1)$, gde je σ kazna

za otvaranje praznine, a ϵ kazna za proširenje praznine, pri čemu je $\sigma > \epsilon$. Definisaćemo problem poravnanja (problem 6), koji pored matrice skora za ulaz prima i brojeve σ i ϵ.

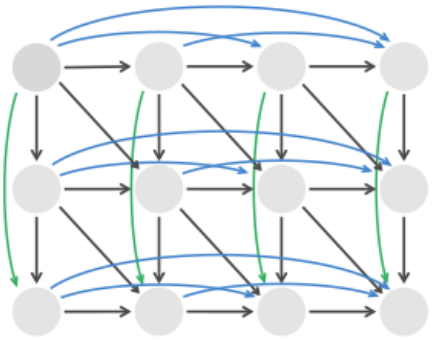
Na slici 5.8 prikazano je kako graf poravnanja možemo prilagoditi afinoj kazni za praznine. Potrebno je da dodamo novu vrstu grana, koje nazivamo duge grane, za svaku prazninu. Pošto ne znamo koliko praznina postoji, moramo da dodamo grane za svaku moguću prazninu. Zbog toga ćemo dodati nove horizontalne i vertikalne grane između svaka dva čvora gde je to moguće. Konkretno, za svaki čvor (i, j) grafa poravnanja dodajemo grane ka čvorovima $(i + k, j)$ i $(i, j + k)$ sa težinama $\sigma + \epsilon \cdot (k - 1)$ za sve moguće praznine dužine k (slika 5.9). Za dve sekvence dužine n, broj grana u grafu ovim dodavanjem raste sa $O(n^2)$ na $O(n^3)$. Kako je složenost algoritma za određivanje skora poravnanja proporcionalna broju grana, potrebno je pronaći efikasnije rešenje.

PROBLEM 6 (PROBLEM PORAVNANJA SA AFINIM KAZNAMA)

problem:	*Pronaći poravnanje sa najvišim skorom između dve niske (sa afinim kaznama).*
ulaz:	*Dve niske* $\boldsymbol{v}$ *i* $\boldsymbol{w}$*, matrica skora* **score***, kao i brojevi* $\boldsymbol{\sigma}$ *i* $\boldsymbol{\epsilon}$*.*
izlaz:	*Poravnanje niski* $\boldsymbol{v}$ *i* $\boldsymbol{w}$ *čiji je skor poravnanja (defnisan matricom skora* **score** *i kaznama za otvaranje i proširenje praznine* $\boldsymbol{\sigma}$ *i* $\boldsymbol{\epsilon}$*) maksimalan od svih mogućih poravnanja za niske* $\boldsymbol{v}$ *i* $\boldsymbol{w}$*.*



Slika 5.8: Graf poravnanja prilagođen afinoj kazni za praznine [2]



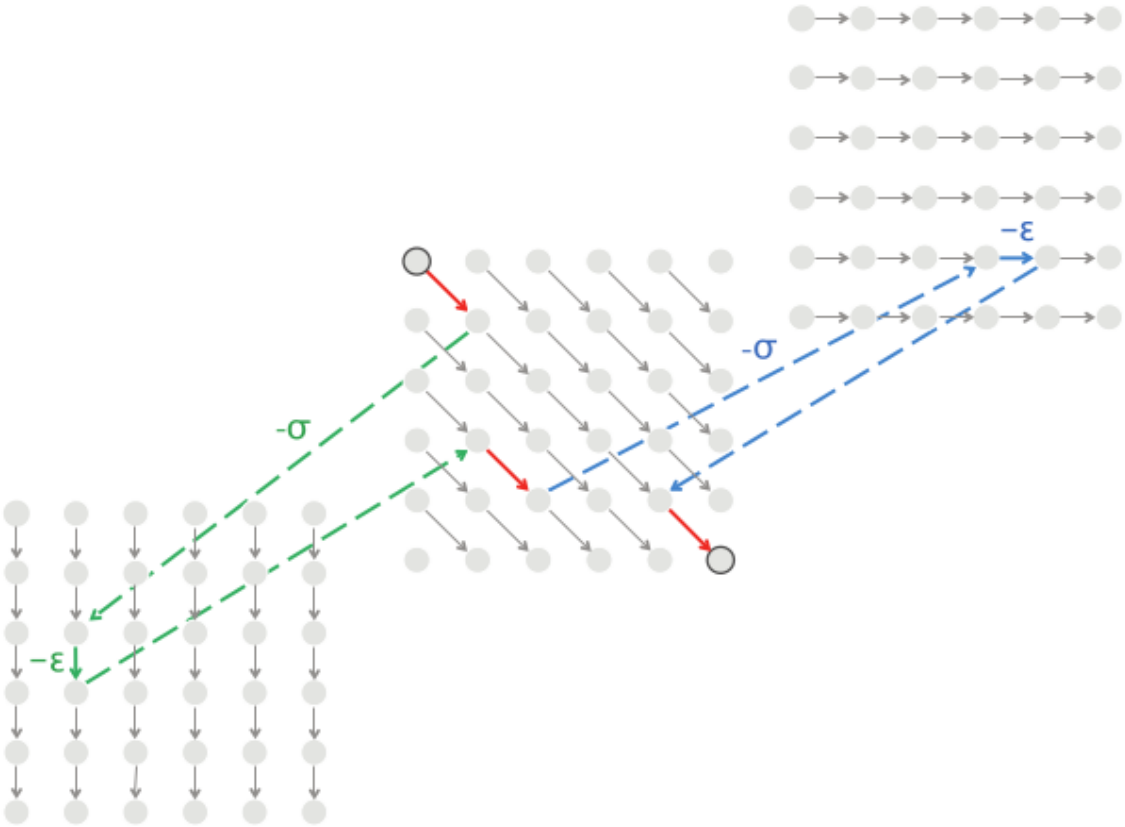
Slika 5.9: Graf poravnanja sa svim dugim granama [2]

Kako je broj grana kubni u odnosu na dužinu jedne sekvence, potrebno je smanjiti broj grana u grafu poravnanja. Pošto broj čvorova u grafu ne utiče na vremensku složenost algoritma, možemo ga povećati. Izgradićemo graf poravnanja na tri nivoa (slika 5.10). Za svaki čvor (i, j) (označimo ga sa $(i, j)_{middle}$) uvešćemo dva dodatna čvora $(i, j)_{lower}$ i $(i, j)_{upper}$. Donji nivo sadrži samo vertikalne čvorove težine $-\epsilon$ i predstavlja praznine u sekvenci v, a gornji nivo sadrži samo horizontalne čvorove težine $-\epsilon$ i predstavlja praznine u sekvenci w. Srednji nivo sadrži dijagonalne grane težine određene matricom skora $score(v_i, w_j)$, tj. poklapanja i promašaje.

Slika 5.10: Tri nivoa grafa poravnanja [2]

Trenutno ne postoji način za povezivanje tri nivoa grafa. Da bismo rešili taj problem, dodaćemo grane koje će predstavljati otvaranje i zatvaranje praznina. Za otvaranje praznina, povezaćemo svaki čvor $(i, j)_{middle}$ sa čvorovima $(i + 1, j)_{lower}$ i $(i, j + 1)_{upper}$ granama težine $-\sigma$. Zatvaranje praznina ne želimo da kažnjavamo, pa ćemo dodati grane težine 0 između svakog čvora $(i, j)_{lower}$ i $(i, j)_{upper}$ i odgovarajućeg čvora $(i, j)_{middle}$. Na ovaj način, praznina dužine k počinje i završava se na srednjem

nivou i kažnjava se sa $-\sigma$ za prvi simbol $-$, $-\epsilon$ za svaki sledeći i 0 za zatvaranje (slika 5.11). Ovim dobijamo kaznu $\sigma + \epsilon \cdot (k-1)$ za prazninu dužine k.



Slika 5.11: Graf poravnanja sa granama između nivoa [2]

Iako se na prvi pogled konstrukcija ovakvog grafa čini komplikovana, zapravo ovakav graf ima samo $O(nm)$ grana za sekvence dužine n i m. Najteža putanja u ovakvom grafu odgovara optimalnom poravnanju sa afinom kaznom za praznine. Potrebno je da definišemo rekurentnu relaciju za pronalaženje najteže putanje u ovako konstruisanom grafu. U ovom slučaju, umesto jedne kao što je do sada bio slučaj, definisaćemo tri rekurentne relacije, za računanje $lower_{i,j}$, $upper_{i,j}$ i $middle_{i,j}$, (relacije 5.3, 5.4 i 5.5). $lower_{i,j}$, $upper_{i,j}$ i $middle_{i,j}$ predstavljaju redom težine najtežih putanja od početnog čvora do čvorova $(i,j)_{lower}$, $(i,j)_{upper}$ i $(i,j)_{middle}$.

$$lower_{i,j} = max \begin{cases} lower_{i-1,j} - \epsilon \\ middle_{i-1,j} - \sigma \end{cases} \tag{5.3}$$

$$upper_{i,j} = max \begin{cases} upper_{i,j-1} - \epsilon \\ middle_{i,j-1} - \sigma \end{cases} \tag{5.4}$$

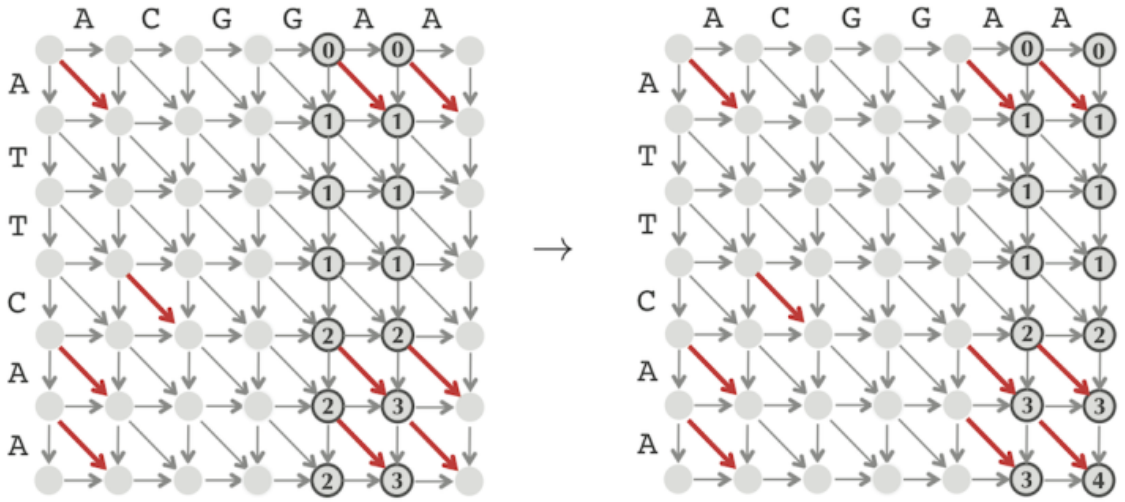
$$middle_{i,j} = max \begin{cases} lower_{i,j} \\ middle_{i-1,j-1} + score(v_i, w_j) \\ upper_{i,j} \end{cases} \tag{5.5}$$

5.3 Prostorno efikasni algoritmi za poravnanje sekvenci

Proteinske sekvence mogu sadržati i do nekoliko desetina hiljada aminokiselina. Na primer, NRP sintetaza koja se može naći u bakteriji *Bacilus brevis* sadrži oko 20000 aminokiselina. Ako bismo pokušali da konstruišemo poravnanje dve sekvence dužine od po nekoliko desetina hiljada karaktera, ne bismo uspeli, zato što bi količina memorije za čuvanje matrice poravnanja bila prevelika.

U slučaju dinamičkih algoritama poravnanja, vremenska složenost je proporcionalna broju grana ($O(nm)$). Da bismo rekonstruisali najtežu putanju u grafu poravnanja, potrebno je čuvati i putokaze od krajnjeg do početnog čvora, pa je i prostorna složenost ovog algoritma proporcionalna broju čvorova ($O(nm)$). Konstruisaćemo prostorno efikasniji algoritam za poravnanje čija će prostorna složenost biti linearna ($O(n)$), a vremenska složenost će biti duplirana, tj. i dalje će ostati $O(nm)$. Algoritam će biti konstruisan pomoću strategije *podeli-pa-vladaj*.

Za računanje skora najteže putanje nije potrebno čuvati celu matricu poravnanja. Za računanje skora jedne kolone, dovoljno je čuvati vrednosti skorova prethodne kolone, u svakom trenutku dovoljno je čuvati matricu od $2n = O(n)$ elemenata. Za računanje skorova u koloni j dovoljni su nam skorovi iz kolone $j - 1$ (slika 5.12). Međutim, pronalaženje najteže putanje zahteva čuvanje cele matrice poravnanja, što dovodi do kvadratne prostorne složenosti. Da bismo smanjili prostornu složenost, nećemo čuvati matricu poravnanja, ali ćemo potrošiti više vremena.



Slika 5.12: Računanje skora poravnanja po kolonama. Za računanje skora jedne kolone dovoljne su nam samo vrednosti iz prethodne kolone [3]

Srednji čvor

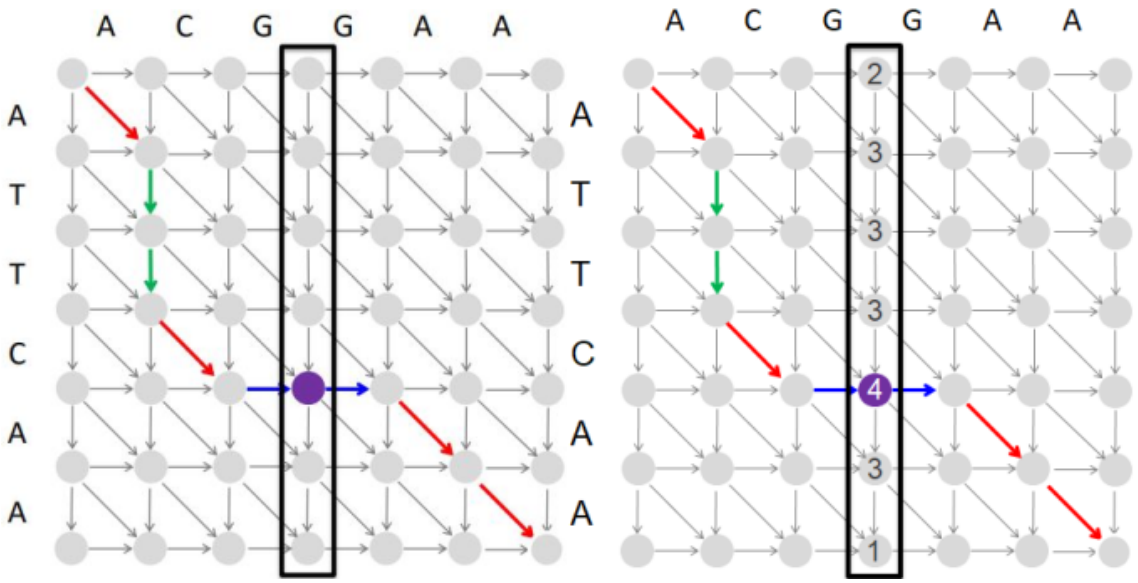
Neka je dat graf poravnanja za sekvence v dužine n i w dužine m. Označimo sa $middle = \lfloor m/2 \rfloor$. **Srednjom kolonom** ćemo zvati kolonu koja sadrži sve čvorove $(i, middle)$ za $0 \leq i \leq n$. Najteža putanja od početnog do krajnjeg čvora u grafu poravnanja mora negde preseći srednju kolonu. Čvor preseka ćemo zvati **srednji čvor**. Na slici 5.13a, srednja kolona je $middle = 3$, a najteža putanja preseca srednju kolonu u čvoru $(4, 3)$ (srednji čvor).

Primetimo da za pronalaženje srednjeg čvora nije neophodno da rekonstruišemo najtežu putanju u grafu poravnanja. Označimo putanju od početnog do krajnjeg čvora koja prolazi kroz srednju kolonu u čvoru i sa i-putanjom. Putanja na slici 5.13a je 4-putanja jer sadrži četvrti čvor srednje kolone (numeracija čvorova kreće od nule). Za svako i između 0 i n želeli bismo da nađemo težine najtežih i-putanja (označićemo ih sa $length(i)$). Od svih i-putanja, ona sa maksimalnom težinom prolazi kroz srednji čvor. Na slici 5.13b su prikazane težine svih najtežih i-putanja upisane u odgovarajuće čvorove srednje kolone.

Označimo sa $fromSource(i)$ težinu najteže putanje od početnog čvora do čvora $(i, middle)$ i sa $toSink(i)$ težinu najteže putanje od čvora $(i, middle)$ do krajnjeg čvora. Težinu najteže i-putanje $length(i)$ računamo na sledeći način:

$$length(i) = fromSource(i) + toSink(i)$$

Vrednost $fromSource(i)$ odgovara veličini $S_{i,middle}$. Već smo razmotrili da ma-



(a) Srednja kolona i srednji čvor $(4, 3)$ [3] (b) Težine najtežih i-putanja za svako i [3]

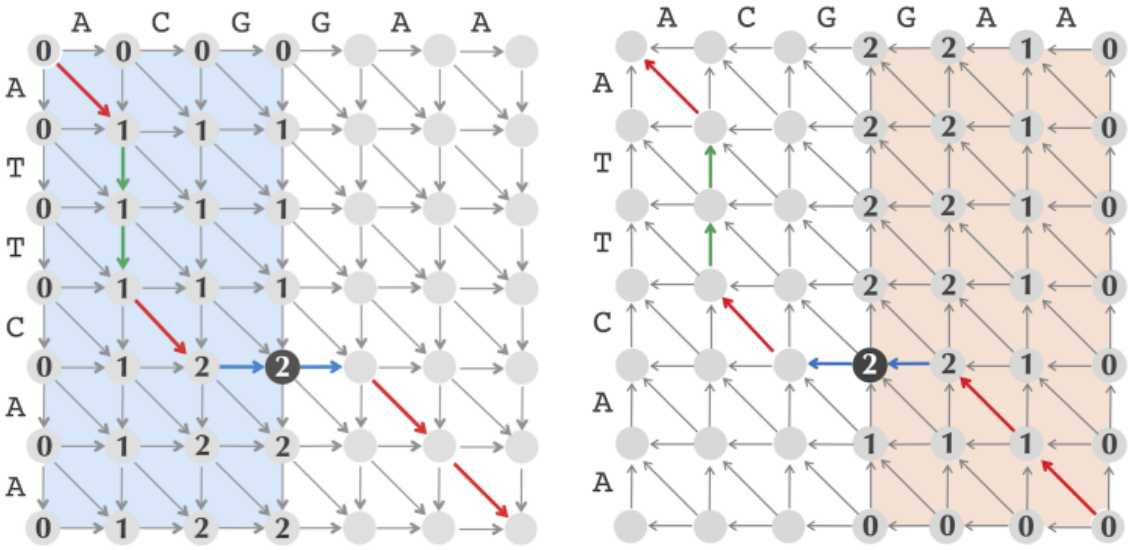
Slika 5.13

tricu S možemo izračunati u linearnom prostoru. Kako je vremenska složenost proporcionalna broju grana u grafu poravnanja, vremenska složenost algoritma za računanje $S_{i,middle}$ je proporcionalna polovini grafa poravnanja, tj. $nm/2$ (slika 5.14a).

Računanje vrednosti $toSink(i)$ se svodi na pronalaženje najteže putanje od krajnjeg čvora do čvora $(i, middle)$ ako obrnemo smer svake grane (5.14b). Umesto obrtanja smerova grana, možemo da obrnemo sekvence v i w. U ovom slučaju vrednost $toSink(i)$ odgovara veličini $S_{n-i,m-middle}$ u grafu poravnanja za obrnute sekvence v i w. Izračunavanje $toSink(i)$ je slično izračunavanju $fromSource(i)$ i može se izvršiti u linearnom prostoru i vremenu proporcionalnom polovini grafa poravnanja, tj. $nm/2$. Možemo reći da je za računanje svih vrednosti $length(i)$ (srednjeg čvora) potrebno $O(n)$ prostora i $O(nm)$ vremena $(nm/2 + nm/2 = nm)$.

Pronalaskom srednjeg čvora, automatski znamo dva dela grafa poravnanja (pravougaonika) kroz koje prolazi najteža putanja. Kao što je prikazano na slici 5.15a jedan od pravougaonika sadrži sve čvorove koji se nalaze iznad i levo od srednjeg čvora, a drugi pravougaonik sadrži sve čvorove dole i desno od srednjeg čvora. Njihova ukupna površina odgovara polovini površine grafa poravnanja.

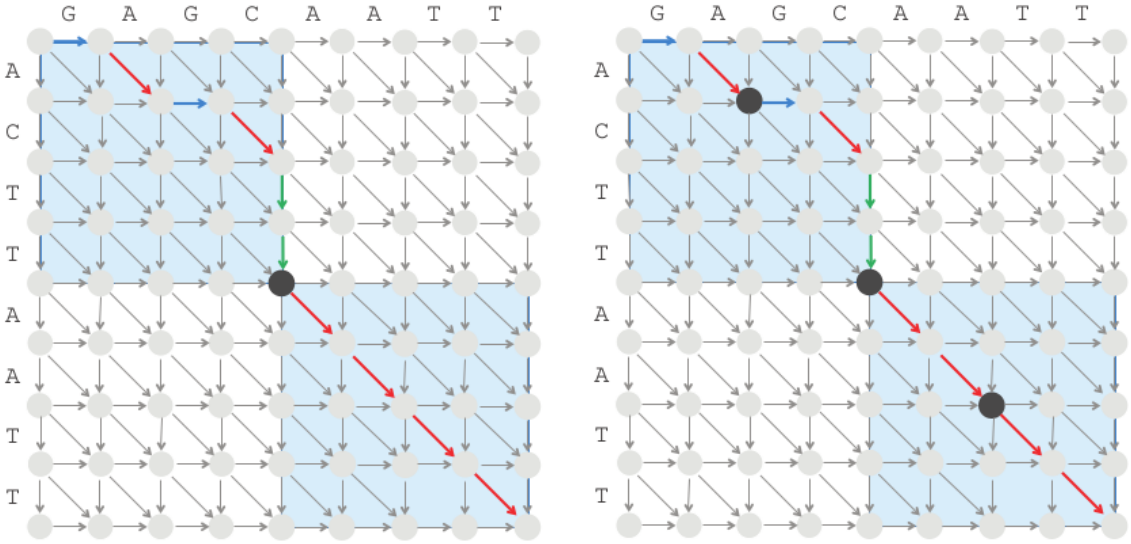
Sada možemo *podeliti* problem pronalaženja najteže putanje od čvora $(0, 0)$ do čvora (n, m) na dva potproblema. Prvi potproblem je pronalaženje najteže putanje od čvora $(0, 0)$ do srednjeg čvora, a drugi je pronalaženje najteže putanje od srednjeg čvora do čvora (n, m). Korak *vladaj* podrazumeva pronalaženje srednjeg čvora u



(a) Računanje $fromSource(i)$ [3] (b) Računanje $toSink(i)$ [3]

Slika 5.14

manjim pravougaonicima, za šta je vremenska složenost proporcionalna ukupnoj površini manjih pravougaonika, tj. $O(nm/2)$ (slika 5.15b). Ovim postupkom smo pronašli tri čvora najteže putanje. Postupak nastavljamo sve dok ne dođemo do pravougaonika dimenzije 1×1 i dok ne pronađemo sve čvorove najteže putanje.



(a) Delovi grafa kroz koje prolazi najteža putanja [2] (b) Srednji čvorovi pronađeni u manjim pravougaonicima [2]

Slika 5.15

Vremenska složenost nalaženja srednjeg čvora u svakom potproblemu je proporci-

onalna površini grafa poravnanja (pravougaonika). U svakom potproblemu površina pravougaonika se smanjuje dva puta dok ne dođemo do pravougaonika dimenzija 1×1. U prvom koraku imaćemo nm operacija, u drugom $nm/2$, u trećem $nm/4$... što znači da je vremenska složenost jednaka:

$$nm + \frac{nm}{2} + \frac{nm}{4} + \cdots < 2nm = O(nm)$$

Strategijom *podeli-pa-vladaj* je postignuto poboljšanje prostorne složenosti na linearnu, a vremenska složenost je duplirana, ali je ostala kvadratna.

5.4 Višestruko poravnanje sekvenci

Proteini koji obavljaju istu funkciju mogu imati donekle slične sekvence, ali ove sličnosti se veoma teško detektuju u slučaju različitih vrsta organizama. Ako je sličnost sekvenci slaba, onda poravnanje dve sekvence možda neće identifikovati njihovu biološku povezanost. Međutim, poređenje većeg broja sekvenci nam često može pomoći da pronađemo sličnosti koje se ne vide u poređenju dve sekvence.

Ako ponovo pogledamo poravnanje tri A domena iz poglavlja 1 (slika 5.16), vidimo da imamo 19 konzerviranih kolona.

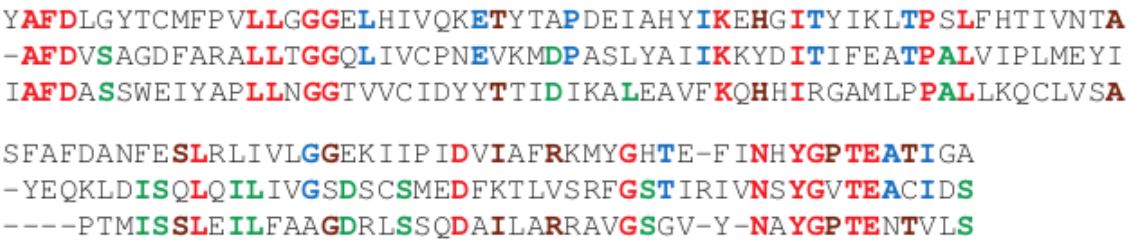
```
YAFDLGYTCMFPVLLGGGELHIVQKETYTAPDEIAHYIKEHGITYIKLTPSLFHTIVNTA
-AFDVSAGDFARALLTGGQLIVCPNEVKMDPASLYAIIKKYDITIFEATPALVIPLMEYI
IAFDASSWEIYAPLLNGGTVVCIDYYTTIDIKALEAVFKQHHIRGAMLPPALLKQCLVSA

SFAFDANFESLRLIVLGGEKIIPIDVIAFRKMYGHTE-FINHYGPTEATIGA
-YEQKLDISQLQILIVGSDSCSMEDFKTLVSRFGSTIRIVNSYGVTEACIDS
----PTMISSLEILFAAGDRLSSQDAILARRAVGSGV-Y-NAYGPTENTVLS
```

Slika 5.16: Poravnanje 3 A-domena sa konzerviranim kolonama [2]

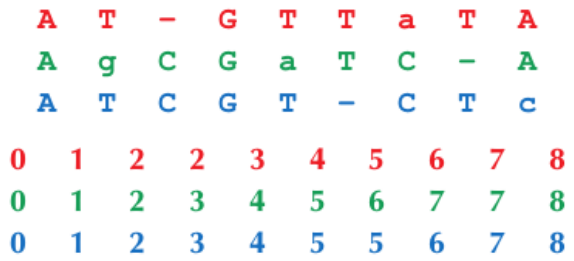
Međutim, ako pogledamo poravnanje po parovima, vidimo da se sličnosti između ovih A domena ne nalaze samo u 19 konzerviranih kolona. Poravnanje po parovima nam otkriva još $10 + 9 + 12 = 31$ polukonzerviranu kolonu gde se poklapaju po dve aminokiseline (slika 5.17).

Višestruko poravnanje t sekvenci $v^1, ..., v^t$ definisano je matricom od t redova. Na poziciji i u matrici se nalaze simboli iz sekvence v^i zajedno sa simbolima $-$. Podrazumevamo da nijedna kolona matrice na sadrži samo praznine. Na slici 5.18 možemo videti označene najpopularnije simbole u svakoj koloni (simboli koji se pojavljuju najviše puta u datoj koloni označeni velikim slovom).



Slika 5.17: Poravnanje 3 A-domena sa konzerviranim i polukonzerviranim kolonama [2]

Matrica poravnanja za više sekvenci predstavlja generalizaciju matrice poravnanja dve sekvence na proizvoljan broj sekvenci. U primeru na slici 5.18, ispod matrice poravnanja, je svakom simbolu različitom od praznine pridružen njegov indeks u sekvenci. Ovi indeksi pročitani po kolonama predstavljaju najtežu putanju u trodimenzionalnom grafu poravnanja za ove sekvence (slika 5.19).

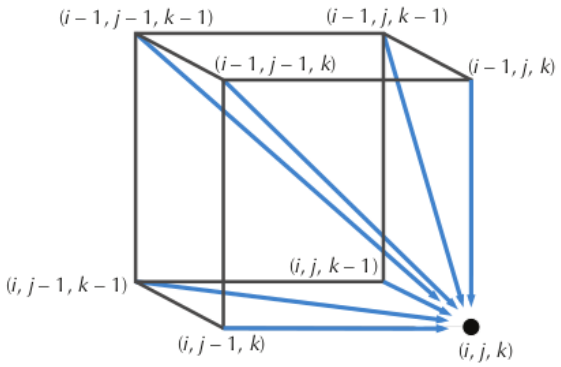


Slika 5.18: Matrica višestrukog poravnanja sa pridruženim indeksima svakom simbolu u sekvencama različitom od praznine [2]

$$(0,0,0) \to (1,1,1) \to (2,2,2) \to (2,3,3) \to (3,4,4) \to (4,5,5) \to (5,6,5) \to (6,7,6) \to (7,7,7) \to (8,8,8)$$

Slika 5.19: Najteža putanja u grafu višestrukog poravnanja [2]

Kod poravnanja dve sekvence, graf poravnanja je mreža kvadrata, a kod poravnanja tri sekvence, graf poravnanja je mreža kocki. Susedni čvorovi formiraju kocke i svaki ima sedam ulaznih grana (slika 5.20). Definisaćemo problem višestrukog poravnanja (problem 7).



Slika 5.20: Susedni čvorovi kod višestrukog poravnanja [2]

PROBLEM 7 (PROBLEM VIŠESTRUKOG PORAVNANJA)

problem: *Pronaći poravnanje sa najvišim skorom između t niski za datu t-dimenzionalnu matricu skora.*

ulaz: *t niski, kao i t-dimenzionalna matrica skora **score**.*

izlaz: *Višestruko poravnanje niski čiji je skor poravnanja (defnisan matricom skora **score**) maksimalan od svih mogućih poravnanja za sve niske.*

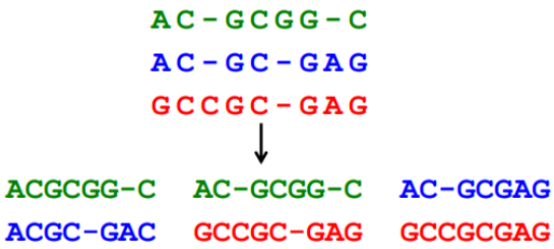
Rekurentna relacija za računanje skora višestrukog poravnanja predstavlja generalizaciju rekurentne relacije za dvostruko poravnanje. Za tri sekvence v, w i u definisaćemo rekurentnu relaciju za računanje vrednosti $S_{i,j,k}$ (relacija 5.6).

$$S_{i,j,k} = max \begin{cases} S_{i-1,j,k} + score(v_i, -, -) \\ S_{i,j-1,k} + score(-, w_j, -) \\ S_{i,j,k-1} + score(-, -, u_k) \\ S_{i-1,j-1,k} + score(v_i, w_j, -) \\ S_{i-1,j,k-1} + score(v_i, -, u_k) \\ S_{i,j-1,k-1} + score(-, w_j, u_k) \\ S_{i-1,j-1,k-1} + score(v_i, w_j, u_k) \end{cases} \tag{5.6}$$

Matrica *score* je u ovom slučaju trodimenzionalna. U slučaju t sekvenci dužine n, graf poravnanja sadrži n^t čvorova i svaki čvor ima $2^t - 1$ ulaznih grana. Kako je vremenska složenost dinamičkog algoritma poravnanja proporcionalna broju grana, vremenska složenost za višestruko poravnanje je $O(2^t n^t)$. Sa porastom broja sekvenci ovaj algoritam postaje prespor. Jedan od algoritama koji prevazilazi ovo usko grlo je pohlepni algoritam za višestruko poravnanje.

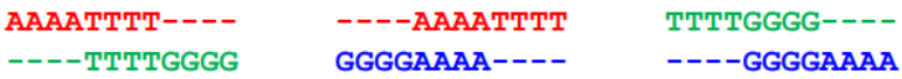
Pohlepni algoritam

Svako višestruko poravnanje sadrži i dvostruka poravnanja njegovih sekvenci koja ne moraju biti optimalna (slika 5.21).



Slika 5.21: Susedni čvorovi kod višestrukog poravnanja [2]

Proverimo da li važi obrnuto, tj. ako su data optimalna dvostruka poravnanja za parove od tri sekvence, da li se ona mogu kombinovati u višestruko poravnanje (slika 5.22).



Slika 5.22: Dvostruka poravnanja za tri sekvence [2]

Ne možemo uvek kombinovati optimalna dvostruka poravnanja u višestruko poravnanje zato što dvostruka poravnanja mogu biti nekompatibilna. Na osnovu prvog poravnanja, AAAA se pojavljuje pre TTTT u višestrukom poravnanju. Na osnovu drugog poravnanja, GGGG bi trebalo da se pojavi pre AAAA, a na osnovu trećeg, TTTT bi trebalo da se pojavi pre GGGG. Dolazimo do zaključka da se AAAA mora pojaviti pre TTTT, koje se mora pojaviti pre GGGG, koje se mora pojaviti pre AAAA, što dovodi do kontradikcije.

Kako bismo izbegli kontradikciju, pokušaćemo da rekonstruišemo višestruko poravnanje na osnovu dvostrukih poravnanja koja ne moraju biti optimalna. Pohlepnim pristupom se za početak od svih dvostrukih poravnanja bira ono sa najvećim skorom i generiše **profilna sekvenca**. Na taj način problem se svodi na poravnanje $t-2$ sekvence i profilne sekvence. Zatim se profilna sekvenca poredi sa preostale $t-2$ sekvence. Na osnovu $t-2$ poravnanja se bira ono sa maksimalnim skorom i ažurira profilna sekvenca i tako redom.

Ovaj algoritam pokazuje dobre rezultate za slične sekvence. Međutim, kada sekvence nisu slične, može se desiti da se inicijalno odabrano poravnanje u velikoj meri razlikuje od ostalih i da se greška nagomilava dodavanjem novih sekvenci.

Glava 6

Uputstvo za korišćenje elektronske lekcije

Aplikacija za potrebe elektronske lekcije je veb aplikacija koja se sastoji iz dve komponente, serverskog i klijentskog dela. Serverski deo je implementiran u razvojnom okviru *NestJS* [6] i struktuiran je kao veb *API*. Razvojni okvir *NestJS* se koristi za izradu serverskih aplikacija na *Node.js* platformi [7]. Klijentski deo je implementiran u razvojnom okviru *React* [8]. Kod aplikacije je javno dostupan na *GitHub* repozitorijumu [9].

Elektronska lekcija se sastoji iz detaljnog teorijskog i interaktivnog dela. Teorijski deo služi korisniku da se upozna sa ključnim problemima i da bolje razume pristupe koje smo koristili za rešavanje tih problema. Interaktivni deo služi korisniku da pokrene nekoliko algoritama za poravnanje sekvenci i da prati izvršavanje korak po korak.

6.1 Preduslovi i pokretanje aplikacije

Kao što je već pomenuto, aplikacija se sastoji iz dve komponente, serverskog i klijentskog dela. Za pokretanje obe komponente aplikacije, korisnik mora imati instaliran *Node.js*, verziju 16, na svom računaru. Preporuka je ne instalirati *Node.js* sa zvaničnog sajta, već koristiti *Node Version Manager* (*nvm*) [10]. Pomoću *nvm*-a možemo instalirati i koristiti više različitih verzija *Node.js*-a u zavisnosti od potreba različitih aplikacija. Još jedan od preduslova za pokretanje komponenti je i instaliran neki od menadžera paketa. Najčešće korišćeni su *npm* [11] i *yarn* [12]. *Npm* se instalira u sklopu *Node.js*-a, dok je *yarn* neophodno naknadno instalirati.

Za pokretanje serverskog dela aplikacije neophodno je pozicionirati se u */be* direktorijum. U zavisnosti od menadžera paketa, potrebno je pokrenuti `install` komandu (za *npm* dovoljno je pokrenuti `npm i` a za *yarn* dovoljno je pokrenuti samo `yarn`). Nakon završene instalacije paketa, za pokretanje serverskog dela aplikacije, dovoljno je izvršiti komandu (`npm` ili `yarn`) `start`. Serverski deo aplikacije će biti dostupan na portu 4000, na IP adresi mašine na kojoj je pokrenut server (lokalno, to je *http://localhost:4000*).

Za pokretanje klijentskog dela aplikacije neophodno je pozicionirati se u */fe* direktorijum. U zavisnosti od menadžera paketa, potrebno je pokrenuti `install` komandu (za *npm* dovoljno je pokrenuti `npm i` a za *yarn* dovoljno je pokrenuti samo `yarn`). Nakon završene instalacije paketa, za pokretanje klijentskog dela aplikacije, dovoljno je izvršiti komandu (`npm` ili `yarn`) `start`. Klijentski deo aplikacije će biti dostupan na portu 3000, na IP adresi mašine na kojoj je pokrenut server (lokalno, to je *http://localhost:3000*).

6.2 Statičke komponente aplikacije

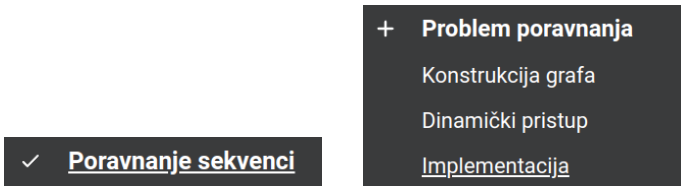
Postoje komponente u aplikaciji koje su zajedničke za svaku stranicu. Svaka od tih komponenti će biti posebno opisana.

Navigacioni meni

Navigacioni meni se nalazi sa leve strane na svakoj stranici. Meni služi korisniku da se lakše kreće kroz aplikaciju. Meni može sadržati dva tipa komponenti. Prvi tip je označen sa ikonicom ✔ (slika 6.1a). Klikom na ovaj tip komponente, korisnik će biti odmah prebačen na željenu stranicu. Drugi tip komponente je označen ikonicom ✚ . Klikom na ovaj tip komponente, korisnik otvara padajući podmeni (slika 6.1b). Klikom na neki od linkova u podmeniju, korisnik će biti prebačen na specifično mesto na stranici označeno naslovom podmenija.

Naslov i navigaciona dugmad

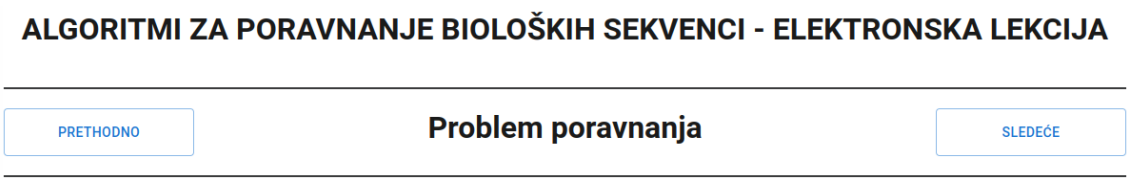
Na vrhu svake stranice se nalazi naslov elektronske lekcije. Ispod naslova i na kraju svake stranice, se leve i sa desne strane, se nalaze dugmad, *prethodno* i *sledeće*, koja nas vode na prethodnu ili sledeću stranicu u elektronskoj lekciji (slika 6.2).



(a) Prvi tip meni komponente

(b) Drugi tip meni komponente

Slika 6.1

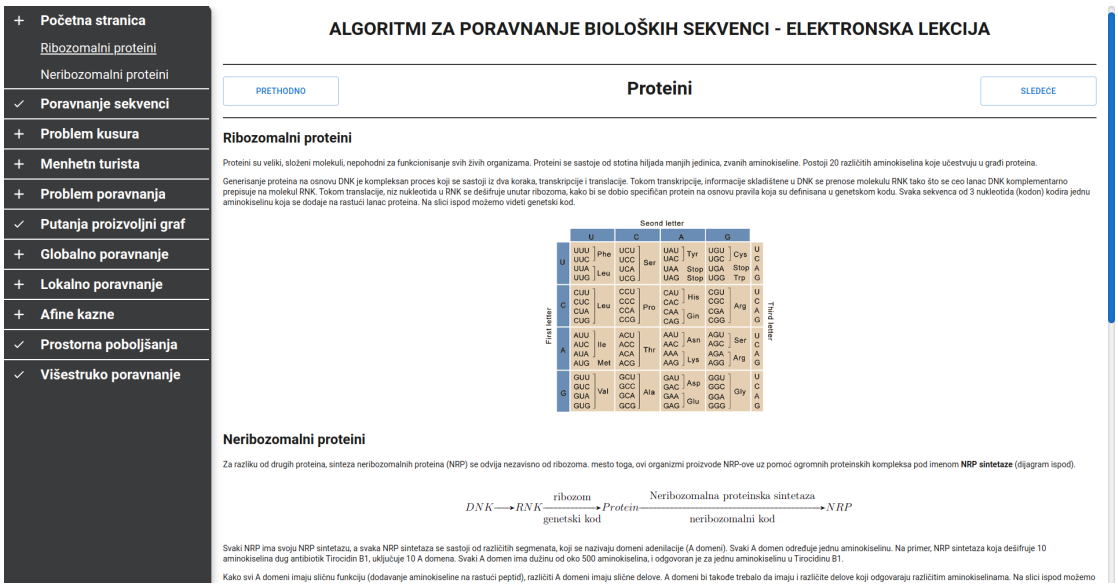


Slika 6.2: Naslov i dugmad *prethodno* i *sledeće*

6.3 Stranice

U aplikaciji razlikujemo dve vrste stranica, statičke stranice i interaktivne stranice.

Statičke stranice se sastoje samo iz teorijskog dela. Statičke stranice su: *Početna stranica*, *Poravnanje sekvenci*, *Problem kusura*, *Putanja proizvoljni graf*, *Prostorna poboljšanja* i *Višestruko poravnanje*. Primer statičke stranice možemo videti na slici 6.3.



Slika 6.3: Primer statičke stranice u aplikaciji (*Početna stranica*)

Na interaktivnim stranicama uz teorijski deo, postoji i interaktivni deo, koji omogućava korisniku da pokrene algoritam i da prati izvršavanje algoritama korak po korak. Interaktivne stranice su: *Menhetn turista*, *Problem poravnanja*, *Globalno poravnanje*, *Lokalno poravnanje* i *Afine kazne*.

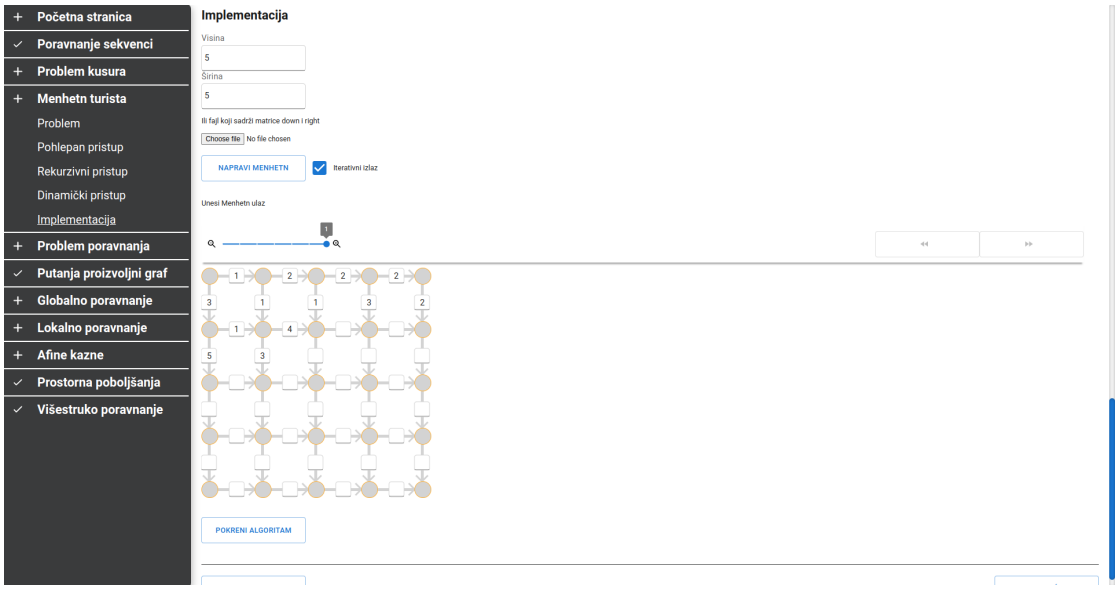
Pored unosa ulaznih parametara, za svaki algoritam, korisnik može da bira da li će se rezultat algoritma prikazati iterativno, pomoću *checkbox*-a. Na svakoj interaktivnoj stranici se nalazi komponenta koja omogućava korisniku da prilagodi prikaz izvršavanja algoritma (slika 6.4) i izlaz algoritma. Prvi slajder služi za umanjenje i uvećanje grafa na kojem se prikazuje izvršavanje algoritma. Drugi slajder služi za podešavanje brzine iterativnog prikaza izvršavanja algoritma. Prvo od tri dugmeta, označeno sa ⏸, omogućava korisniku da pauzira izvršavanje algoritma. Druga dva dugmeta označena sa ⏪ i ⏩, omogućavaju korisniku da pomera graf u levo ili u desno.

Slika 6.4: Komponenta za prilagođavanje prikaza izvršavanja algoritma

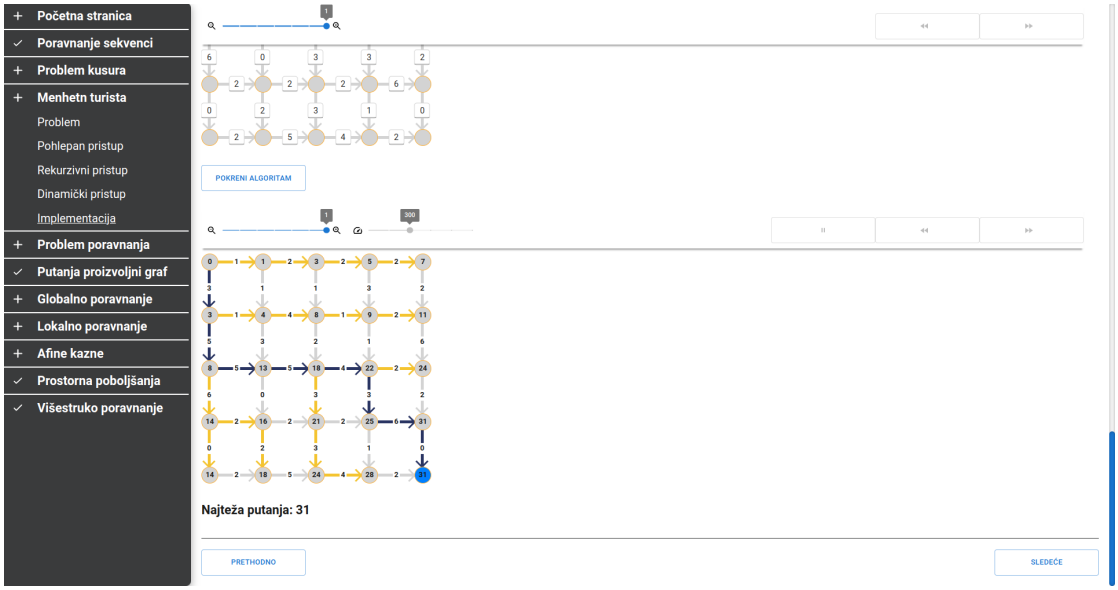
6.3.1 Menhetn turista

Na stranici *Menhetn turista* se nakon teorijskog dela nalaze polja u koja korisnik može da unese ulazne parametre za ovaj algoritam. Korisnik može da unese ulazne parametre na dva načina. Prvi način je da unese širinu i visinu Menhetn grafa i nakon toga da klikne na dugme `NAPRAVI MENHETN`. Nakon toga će se na ekranu pojaviti Menhetn graf koji na granama ima ulazna polja u koja korisnik može da unese težinu odgovarajuće grane. Klikom na dugme `POKRENI ALGORITAM` će se pokrenuti izvršavanje algoritma. Drugi način je da korisnik izabere *json* datoteku koja sadrži matrice *down* i *right* koje predstavljaju ulaz ovog algoritma. Klikom na dugme `NAPRAVI MENHETN` će se pokrenuti izvršavanje algoritma (slika 6.5).

U Menhetn grafu, grane obojene žutom bojom predstavljaju vrednosti matrice *backtrack*, grane obojene tamno plavom bojom predstavljaju najtežu putanju. Nakon Menhetn grafa se može videti izlaz algoritma koji predstavlja vrednost najteže putanje (slika 6.6).



Slika 6.5: Stranica *Menhetn turista* - unos ulaznih parametara



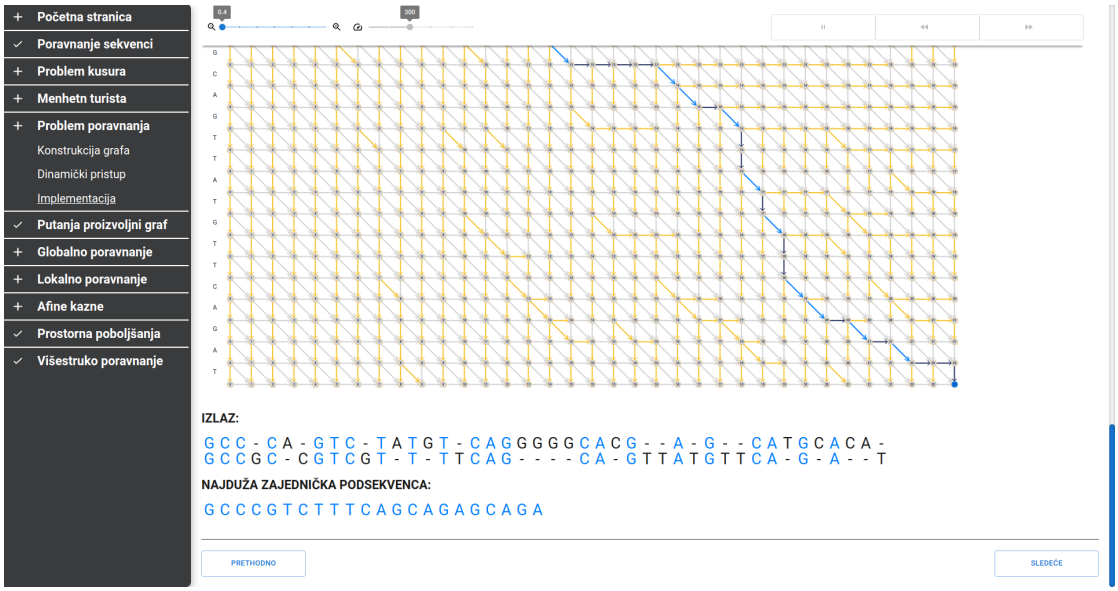
Slika 6.6: Stranica *Menhetn turista* - izlaz algoritma

6.3.2 Problem poravnanja

Na stranici *Problem poravnanja* se nakon teorijskog dela nalaze polja u koja korisnik može da unese dve niske karaktera. Klikom na dugme `POKRENI ALGORITAM` će se pokrenuti izvršavanje algoritma.

U grafu poravnanja, grane obojene žutom bojom predstavljaju vrednosti matrice *backtrack*, a grane obojene svetlo i tamno plavom bojom predstavljaju najtežu puta-

nju. Tamno plavom bojom su prikazane insercije i delecije, a svetlo plavom bojom su prikazana poklapanja. Nakon grafa poravnanja se može videti rezultat izvršavanja algoritma, poravnanje niski i najduža zajednička podsekvenca (slika 6.7).



Slika 6.7: Stranica *Problem poravnanja*

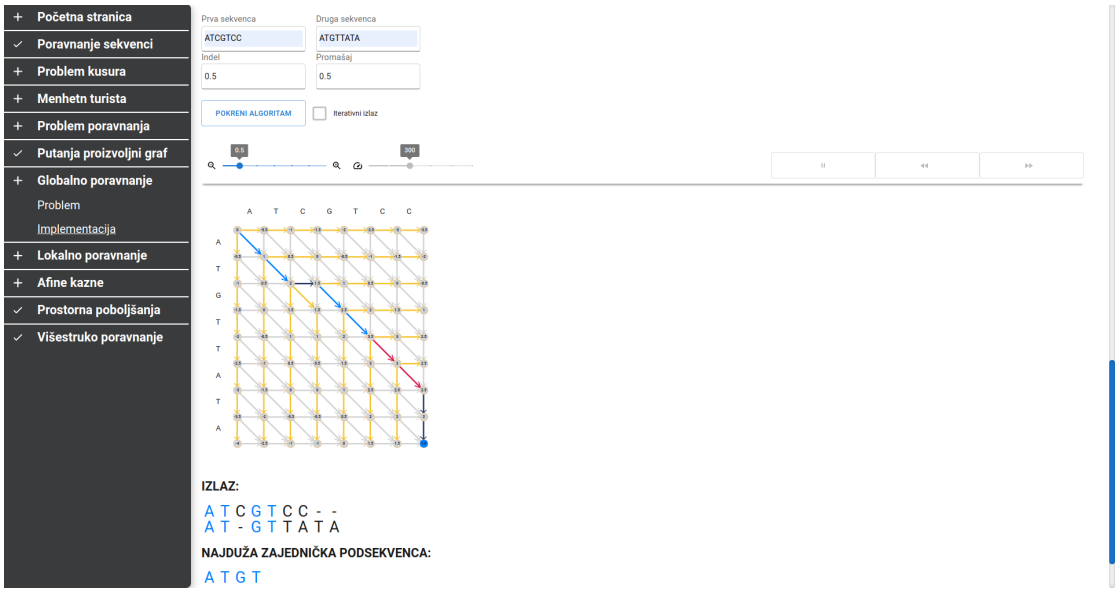
6.3.3 Globalno i lokalno poravnanje

Na stranicama *Globalno poravnanje* i *Lokalno poravnanje* se nalaze polja u koja korisnik može da unese dve niske karaktera, ali i odgovarajuće kazne za insercije/delecije i nepoklapanja. Klikom na dugme `POKRENI ALGORITAM` će se pokrenuti izvršavanje algoritma.

U grafu poravnanja, grane obojene žutom bojom predstavljaju vrednosti matrice *backtrack*, a grane obojene crvenom, svetlo plavom i tamno plavom bojom predstavljaju najtežu putanju. Tamno plavom bojom su prikazane insercije i delecije, crvenom bojom su prikazani promašaji, a svetlo plavom bojom su prikazana poklapanja. Nakon grafa poravnanja se može videti rezultat izvršavanja algoritma, poravnanje niski i najduža zajednička podsekvenca (slika 6.8).

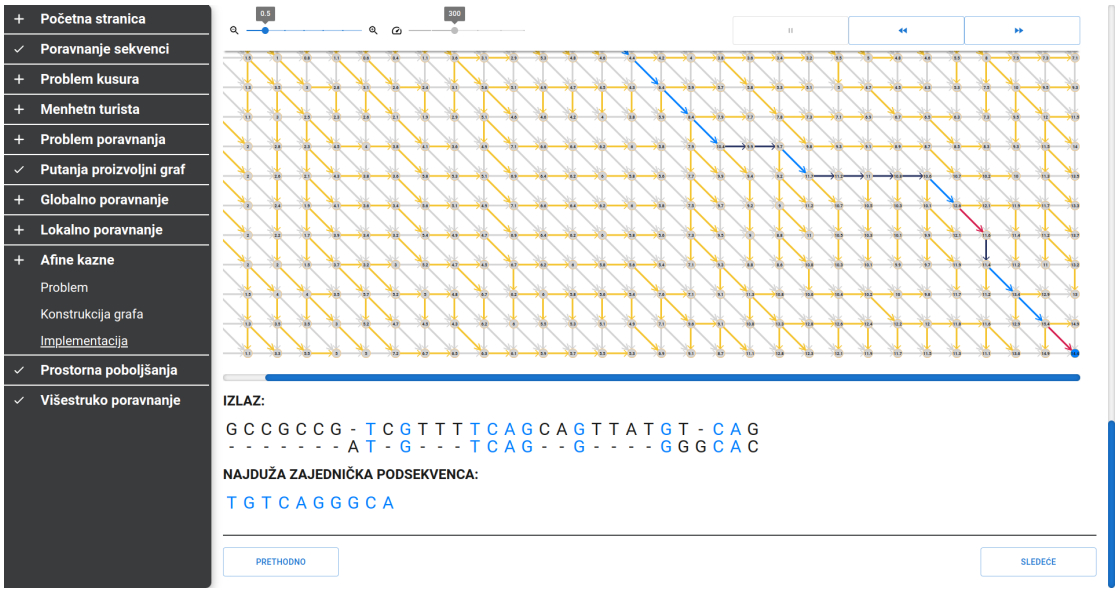
6.3.4 Afine kazne za praznine

Na stranici *Afine kazne* se nalaze polja u koja korisnik može da unese dve niske karaktera, ali i odgovarajuću kaznu za nepoklapanja i vrednosti σ i ϵ. Klikom na dugme `POKRENI ALGORITAM` će se pokrenuti izvršavanje algoritma.



Slika 6.8: Stranica *Globalno/Lokalno poravnanje*

U grafu poravnanja, grane obojene žutom bojom predstavljaju vrednosti matrice *backtrack*, a grane obojene crvenom, svetlo plavom i tamno plavom bojom predstavljaju najtežu putanju. Tamno plavom bojom su prikazane insercije i delecije, crvenom bojom su prikazani promašaji, a svetlo plavom bojom su prikazana poklapanja. Nakon grafa poravnanja se može videti rezultat izvršavanja algoritma, poravnanje niski i najduža zajednička podsekvenca (slika 6.9).



Slika 6.9: Stranica *Afine kazne*

Glava 7

Zaključak

U ovom radu je opisano nekoliko algoritama koji se koriste za poravnanje sekvenci. Takođe, opisani su jednostavniji problemi kako bismo najpre na njima ilustrovali tehinke za rešavanje koje će biti primenjene na problem poravnanja. Detaljno su objašnjene prostorna i vremenska složenost algoritama.

Osnovni cilj ovog rada je implementacija elektronske lekcije. Elektronska lekcija bi mogla da posluži i kao nastavno sredstvo za predavače i kao sredstvo za učenje za studente. Implementirana je veb aplikacija koja pored teorijskog ima i interaktivni deo. Teorijski deo služi korisniku da se upozna sa ključnim problemima i da bolje razume pristupe koje smo koristili za rešavanje tih problema. Interaktivni deo omogućava korisniku da iterativno posmatra način izvršavanja algoritama.

U glavi 1 su prikazane osnovne informacije o sintezi ribozomalnih i neribozomalnih proteina i pomenuta je jedna od primena algoritama poravnanja bioloških sekvenci. U glavi 2 je definisano optimalno poravnanje i problem najduže zajedničke podsekvence. U glavi 3 su definisana dva problema, problem turiste na Menhetnu i problem kusura. Na ovim problemima su demonstrirani različiti pristupi rešavanja problema koje ćemo koristiti za probleme poravnanja (pohlepni, rekurzivni i dinamički pristup). U glavi 4 je konstruisan graf za problem poravnanja i rešen je problem poravnanja dinamičkim pristupom. Takođe, rešenja problema Menhetn turiste i problema poravnanja su generalizovana na rešenje problema pronalaska najteže putanje u usmerenom acikličnom grafu. U glavi 5 su definisani globalno i lokalno poravnanje, kao i poravnanje sa afinim kaznama. Predstavljeni su i prostorno efikasan algoritam za poravnanje sekvenci i algoritam za višestruko poravnanje sekvenci. Na kraju, u glavi 6 je prikazano uputstvo za pokretanje i korišćenje elektronske lekcije.

Kada je reč o daljem radu, implementirani algoritmi imaju dosta poboljšanja i

adaptacija, koje nisu pokrivene ovim radom. U elektronsku lekciju može biti dodata implementacija algoritma koji kao ulaz prima matricu skora (trenutno je implementiran algoritam koji kao ulaz prima kazne za insercije, delecije i nepoklapanja). Uz ovaj dodatak, algoritam bi mogao da se koristi za poređenje proteinskih sekvenci. Takođe, može biti dodat algoritam za višestruko poravnanje sekvenci. Elektronska lekcija može biti proširena i sa drugim kompleksnijim vrstama poravnanja sekvenci.

Literatura

[1] *Cracking the genetic code.* URL: https://americanhistory.si.edu/blog/2010/02/cracking-the-genetic-code.html. (avgust 2022.)

[2] Phillip Compeau i Pavel Pevzner. *Bioinformatics Algorithms: An Active Learning Approach, 2nd Ed. Vol. 1.* Active Learning Publishers, 2015. (april 2022.)

[3] Jovana Kovačević. *Skripta za predavanja, Uvod u Bioinformatiku, Matematički fakultet.* URL: http://www.bioinformatika.matf.bg.ac.rs/predavanja/Chapter_5/Chapter_5_tekst.pdf. (april 2022.)

[4] Jovana Kovačević. *Prezentacija sa predavanja, Uvod u Bioinformatiku, Matematički fakultet.* URL: http://www.bioinformatika.matf.bg.ac.rs/predavanja/Chapter_5/Chapter_5_slajdovi.pdf. (april 2022.)

[5] *Pairwise Alignment.* URL: https://snipcademy.com/pairwise-alignment. (septembar 2022.)

[6] *NestJS dokumentacija.* URL: https://docs.nestjs.com/. (septembar 2022.)

[7] *Node.js dokumentacija.* URL: https://nodejs.org/en/. (septembar 2022.)

[8] *React dokumentacija.* URL: https://reactjs.org/. (septembar 2022.)

[9] *GitHub repozitorijum sa kodom aplikacije.* URL: https://github.com/milos494/kako-poredimo-bioloske-sekvence. (septembar 2022.)

[10] *Nvm dokumentacija.* URL: https://github.com/nvm-sh/nvm. (septembar 2022.)

[11] *Npm dokumentacija.* URL: https://www.npmjs.com/. (septembar 2022.)

[12] *Yarn dokumentacija.* URL: https://yarnpkg.com/. (septembar 2022.)

[13] Sabrina Mantaci, Antonio Restivo i Marinella Sciortino. „Distance measures for biological sequences: Some recent approaches". U: *International Journal of Approximate Reasoning* 47.1 (2008). Approximate Reasoning and Machine Learning for Bioinformatics, str. 109–124. DOI: https://doi.org/10.1016/j.ijar.2007.03.011. (jun 2022.)

[14] Mario Martínez-Núñez i Víctor López. „Nonribosomal peptides synthetases and their applications in industry". U: *Sustainable Chemical Processes* 4 (Dec. 2016). DOI: 10.1186/s40508-016-0057-6. (jun 2022.)

[15] *How do genes direct the production of proteins?* URL: https://medlineplus.gov/genetics/understanding/howgeneswork/makingprotein/. (jul 2022.)

Biografija autora

Miloš Lončarević rođen je 29. decembra 1994. u Čačku. Osnovnu školu i prirodno-matematički smer Čačanske gimnazije je završio kao nosilac Vukove diplome.

Smer matematika (računarstvo i informatika) na Matematičkom fakultetu Univerziteta u Beogradu upisao je 2013. godine, a završio 2017. godine sa prosečnom ocenom 8.45. Nakon toga je upisao master studije na istom smeru.

Od aprila 2018. do septembra 2020. godine je zaposlen na poziciji *Software* developer u frimi **Levi9**. Od septembra 2020. godine do maja 2022. je zaposlen na poziciji *Software* developer u firmi **HTEC**, a od maja 2022. godine do sada je zaposlen u istoj firmi na pozicji *Engineering lead*. Projekti na kojima je radio su uglavnom bili zasnovani na veb tehnologijama, a osnovni programski jezik u kojem je rađeno je *JavaScript*.