Bioinformatika

2. april 2020..

Glava 1

Kako poredimo biološke sekvence?

1.1 Biološki uvid u poređenje sekvenci

Posmatrajmo sledeće tri proteinske sekvence:

```
YAFDLGYTCMFPVLLGGGELHIVQKETYTAPDEIAHYIKEHGITYIKLTPSLFHTIVNTASFAFDANFESLRLIVLGGEKIIPIDVIAFRKMYGHTEFINHYGPTEATIGA
AFDVSAGDFARALLTGGQLIVCPNEVKMDPASLYAIIKKYDITIFEATPALVIPLMEYIYEQKLDISQLQILIVGSDSCSMEDFKTLVSRFGSTIRIVNSYGVTEACIDS
IAFDASSWEIYAPLLNGGTVVCIDYYTTIDIKALEAVFKQHHIRGAMLPPALLKQCLVSAPTMISSLEILFAAGDRLSSQDAILARRAVGSGVYNAYGPTENTVLS
```

Možemo uočiti da se na istim pozicijama u sekvencama uglavnom nalaze različite aminokiseline, osim na sledeće tri pozicije:

```
YAFDLGYTCMFPVLLGGGELHIVQKETYTAPDEIAHYIKEHGITYIKLTPSLFHTIVNTASFAFDANFESLRLIVLGGEKIIPIDVIAFRKMYGHTEFINHYGPTEATIGA
AFDVSAGDFARALLTGGQLIVCPNEVKMDPASLYAIIKKYDITIFEATPALVIPLMEYIYEQKLDISQLQILIVGSDSCSMEDFKTLVSRFGSTIRIVNSYGVTEACIDS
IAFDASSWEIYAPLLNGGTVVCIDYYTTIDIKALEAVFKQHHIRGAMLPPALLKQCLVSAPTMISSLEILFAAGDRLSSQDAILARRAVGSGVYNAYGPTENTVLS
```

Pozicije na kojima se nalaze iste aminokiseline zovemo *konzerviranim* pozicijama i kažemo da čine *konzerviranu* kolonu. Primetimo da ako drugu sekvencu pomerimo za jednu poziciju udesno dodavanjem simbola **-**, dobijamo veći broj konzerviranih kolona, ukupno 11:

```
YAFDLGYTCMFPVLLGGGELHIVQKETYTAPDEIAHYIKEHGITYIKLTPSLFHTIVNTASFAFDANFESLRLIVLGGEKIIPIDVIAFRKMYGHTEFINHYGPTEATIGA
-AFDVSAGDFARALLTGGQLIVCPNEVKMDPASLYAIIKKYDITIFEATPALVIPLMEYIYEQKLDISQLQILIVGSDSCSMEDFKTLVSRFGSTIRIVNSYGVTEACIDS
IAFDASSWEIYAPLLNGGTVVCIDYYTTIDIKALEAVFKQHHIRGAMLPPALLKQCLVSAPTMISSLEILFAAGDRLSSQDAILARRAVGSGVYNAYGPTENTVLS
```

Ubacivanjem simbola **-** unutar sekvenci možemo dobiti još više konzerviranih kolona, ukupno 19:

```
YAFDLGYTCMFPVLLGGGELHIVQKETYTAPDEIAHYIKEHGITYIKLTPSLFHTIVNTASFAFDANFESLRLIVLGGEKIIPIDVIAFRKMYGHTE-FINHYGPTEATIGA
-AFDVSAGDFARALLTGGQLIVCPNEVKMDPASLYAIIKKYDITIFEATPALVIPLMEYI-YEQKLDISQLQILIVGSDSCSMEDFKTLVSRFGSTIRIVNSYGVTEACIDS
IAFDASSWEIYAPLLNGGTVVCIDYYTTIDIKALEAVFKQHHIRGAMLPPALLKQCLVSA----PTMISSLEILFAAGDRLSSQDAILARRAVGSGV-Y-NAYGPTENTVLS
```

U nukleotidnim sekvencama, a time i u proteinskim, često dolazi do mutacija odnosno do zamene jednog nukleotida drugim nukleotidom. Kada poredimo više sekvenci, važno je da utvrdimo koje su pozicije konzervirane jer one imaju veliki biološki značaj. Na primer, sekvence koje smo prethodno razmatrali predstavljaju delove takozvanih adenilacionih domena (skraćeno A-domeni) koji se nalaze unutar NRP sintetaza. Kao što je bilo reči u prethodnom poglavlju, NRP sintetaze su enzimi uz čiju pomoć nastaju neribozomalni proteini. Sastav ovih proteina nije zapisan u DNK kao što je to slučaj sa drugim proteinima, već ga određuje sastav NRP sintetaze koja omogućava njegovu sintezu. Međutim, NRP sintetaza je takođe protein pa se zbog toga postavlja pitanje kako je unutar jednog proteina kodiran sastav drugog proteina. Svaka NRP sintetaza se sastoji od modula od kojih svaki određuje po jednu aminokiselinu proteina koji se sintetiše. Dalje, svaki modul se sastoji od nekoliko podjedinica od kojih su za sintezu najznačajniji upravo A-domeni. Nakon dugogodišnjih istraživanja u kojima je od velikog značaja bilo poznavanje konzerviranih kolona u A-domenima iste NRP sintetaze, otkriveno je da su aminokiseline kodirane nizom *nesusednih* aminokiselina koje se nalaze na istim pozicijama u nekonzerviranim kolonama:

```
YAFDLGYTCMFPVLLGGGELHIVQKETYTAPDEIAHYIKEHGITYIKLTPSLFHTIVNTASFAFDANFESLRLIVLGGEKIIPIDVIAFRKMYGHTE-FINHYGPTEATIGA
-AFDVSAGDFARALLTGGQLIVCPNEVKMDPASLYAIIKKYDITIFEATPALVIPLMEYI-YEQKLDISQLQILIVGSDSCSMEDFKTLVSRFGSTIRIVNSYGVTEACIDS
IAFDASSWEIYAPLLNGGTVVCIDYYTTIDIKALEAVFKQHHIRGAMLPPALLKQCLVSA----PTMISSLEILFAAGDRLSSQDAILARRAVGSGV-Y-NAYGPTENTVLS
```

Niz od 8 aminokiselina koje čine kod za drugu aminokiselinu naziva se *signatura*. Sa prethodne slike možemo videti signature za aminokiseline aspartat (Asp), valin (Val) i nestandardnu aminokiselinu ornitin (Orn):

```
LTKVGHIG  Asp
VGEIGSID  Orn
AWMFAAVL  Val
```

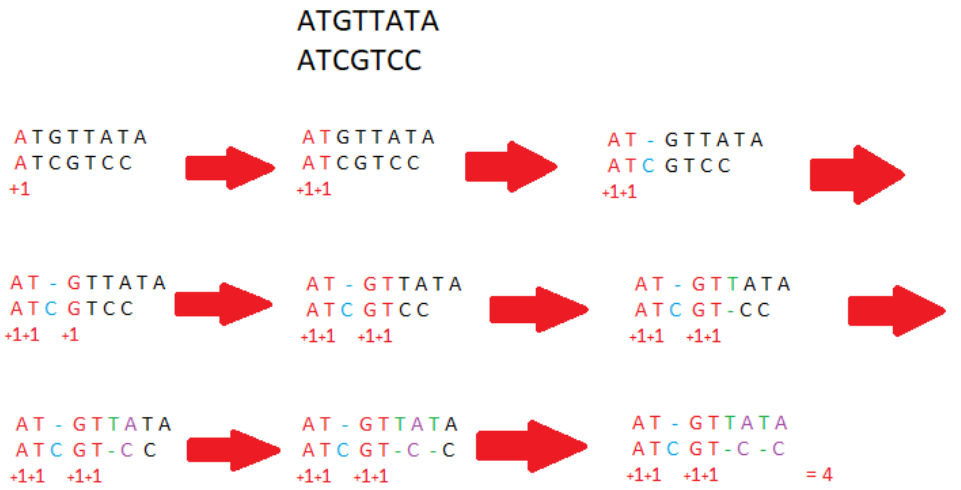
Kao što za proteine koji nastaju standardnim postupkom postoji genetski kod, tako za neribozomalne proteine postoji neribozomalni kod kojim su aminokiseline koje čine protein kodirane. Neribozomalni kod nije toliko dobro izučen kao standardni genetski kod i istraživanja su još uvek u toku. U ovom trenutku (2020.) je poznat neribozomalni kod za 14 standardnih aminokiselina.

Razbijanje neribozomalnog koda je samo jedno od brojnih bioloških istraživanja koje je uključivalo poređenje bioloških sekvenci. U nastavku ćemo formalizovati problem poređenja bioloških sekvenci i predložiti različite algoritme za njegovo rešavanje.

1.2 Igra poravnanja i najduža zajednička podsekvenca

Pretpostavimo da su date dve nukleotidne sekvence i da je potrebno *poravnati* ih tako da broj konzerviranih kolona bude što veći. Ovaj postupak možemo nazvati *igrom poravnanja* u kojoj je cilj ukloniti sve simbole iz sekvenci, jedan po jedan, sleva nadesno, tako da pritom sakupimo što više poena. U igri su mogući sledeći potezi:

- Uklanjanje prvog simbola sleva iz *obe* sekvence
  - 1 poen ako se simboli poklapaju,
  - 0 ako se simboli ne poklapaju
- Uklanjanje prvog simbola sleva iz *jedne* sekvence i dodavanje simbola - u *drugu* sekvencu
  - 0 poena

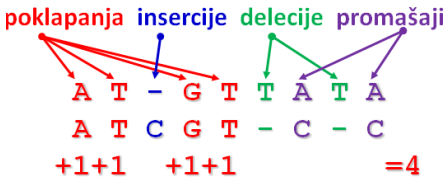


Slika 1.1: Igra poravnanja

Na slici 1.1 prikazana je jedna partija igre poravnanja za sekvence ATGTTATA i ATCGTCC.

Poravnanjem dve sekvence nazivamo matricu koja se sastoji iz dva reda:

1. red: simboli prve sekvence (redom) eventualno sa ubačenim simbolom -
2. red: simboli druge sekvence (redom) eventualno sa ubačenim simbolom -



Slika 1.2: Poravnanje

Na slici 1.2 prikazano je poravnanje iz odigrane partije. U svakoj matrici poravnanja razlikujemo sledeće tipove kolona:

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

U matrici poravnanja dve sekvence nema smisla da se u istoj koloni nađu dva simbola -.

Poklapanja u poravnanju dve sekvence (u primeru 1.2 to je ATGT) formiraju njihovu *zajedničku podsekvencu*. Primetimo da se pojmovi podsekvenca i podniska razlikuju. Simboli koji čine podsekvencu ne moraju biti susedni u sekvencama pa podsekvenca ne mora biti podniska datih sekvenci.

S obzirom da želimo da u poravnanju imamo što više poklapanja, problem poravnanja možemo svesti na problem nalaženja najduže zajedničke podsekvence.

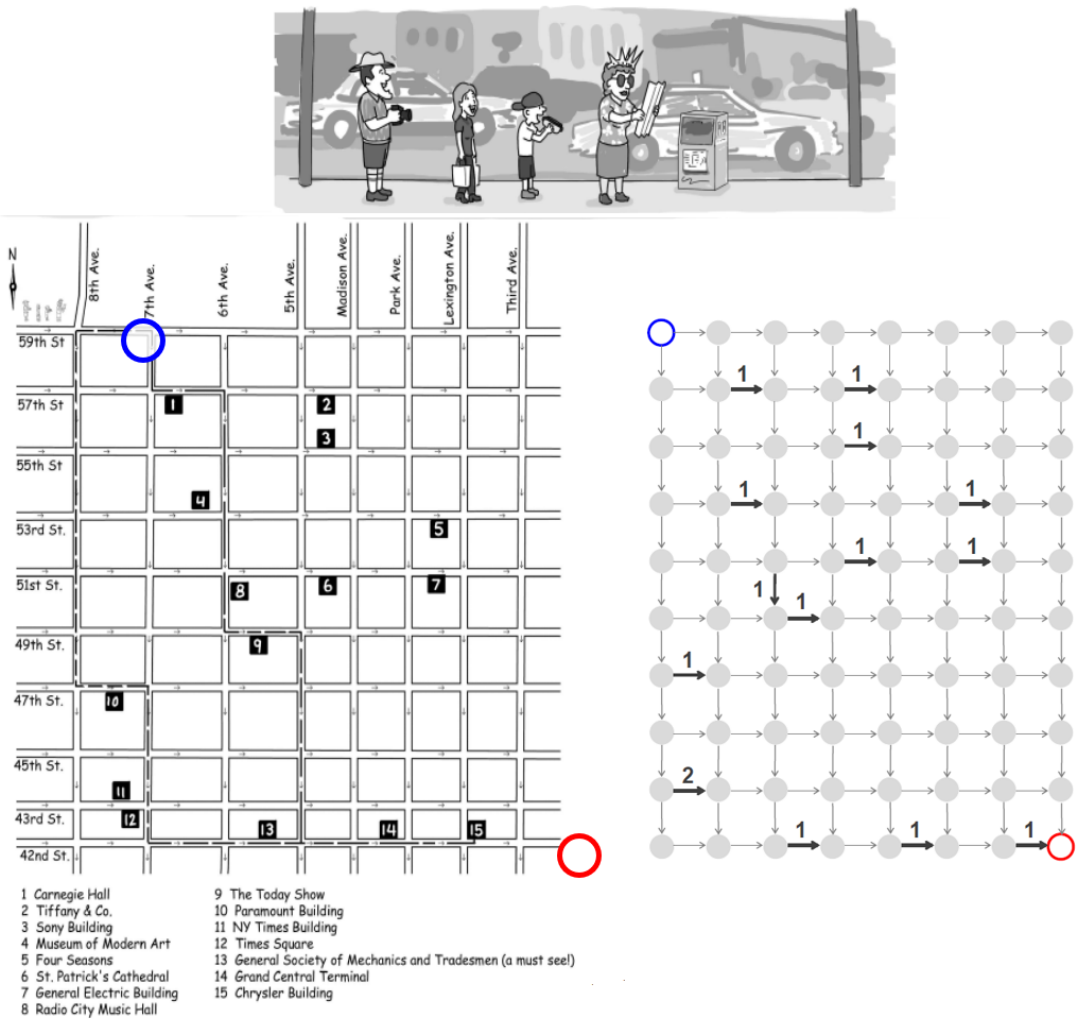
Problem 1 (Problem najduže zajedničke podsekvence)**.** *Naći najdužu zajedničku podsekvencu dve niske.*
Ulaz: Dve niske.
Izlaz: Najduža zajednička podsekvenca ovih niski

1.3 Problem turiste na Menhetnu

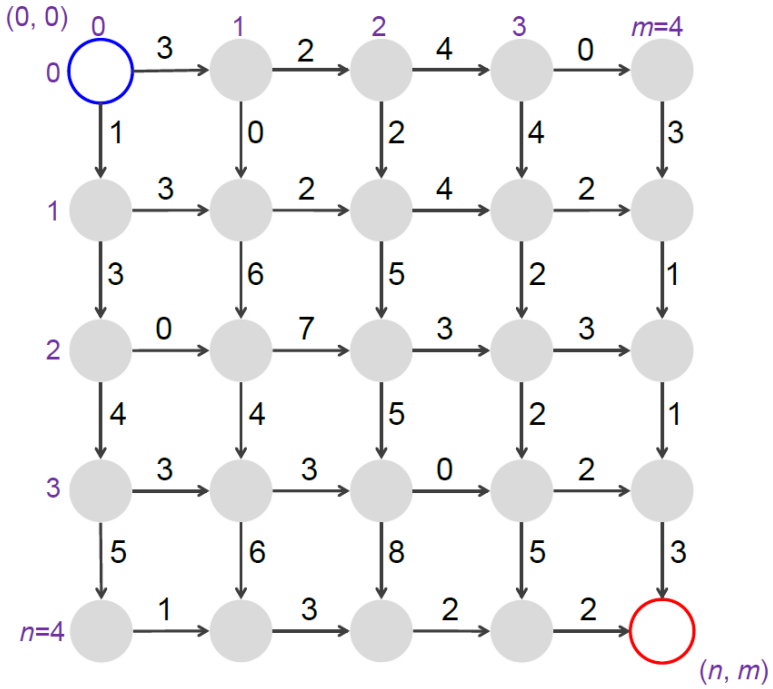
Pre nego što počnemo sa rešavanjem problema najduže zajedničke podsekvence, razmotrimo jedan specijalan slučaj tog problema pod nazivom problem turiste na Menhetnu. Pretpostavimo da se jedan turista nalazi na Menhetnu gde se ulice seku pod pravim uglom i tako formiraju pravougaonu mrežu. Turista se nalazi u gornjem levom uglu pravougaone mreže i želi da dođe do donjeg desnog ugla ali da na tom putu vidi što više znamenitosti. Na mapi Mehnetna (slika 1.3 levo) je u svakom delu ulice označen broj znamenitosti koje se tu nalaze. Pravougaonu mrežu ulica možemo predstaviti usmerenim težinskim grafom 1.3 desno). Podrazumevamo da će se turista u svakom trenutku kretati prema cilju (na dole ili na desno) i da se neće vraćati unazad, što modelujemo usmeravanjem grana grafa.

Sada možemo da definišemo problem turiste na Menhetnu:

Problem 2 (Problem turiste na Menhetnu)**.** *Naći najdužu putanju između dva čvora usmerenom težinskom pravougaonom mrežnom grafu.*
Ulaz: Usmeren težinski pravougaoni mrežni graf.
Izlaz: Najduža putanja od početnog (source) do krajnjeg čvora (sink) u usmerenom težinskom pravougaonom mrežnom grafu.



Slika 1.3: Mapa Menhetna

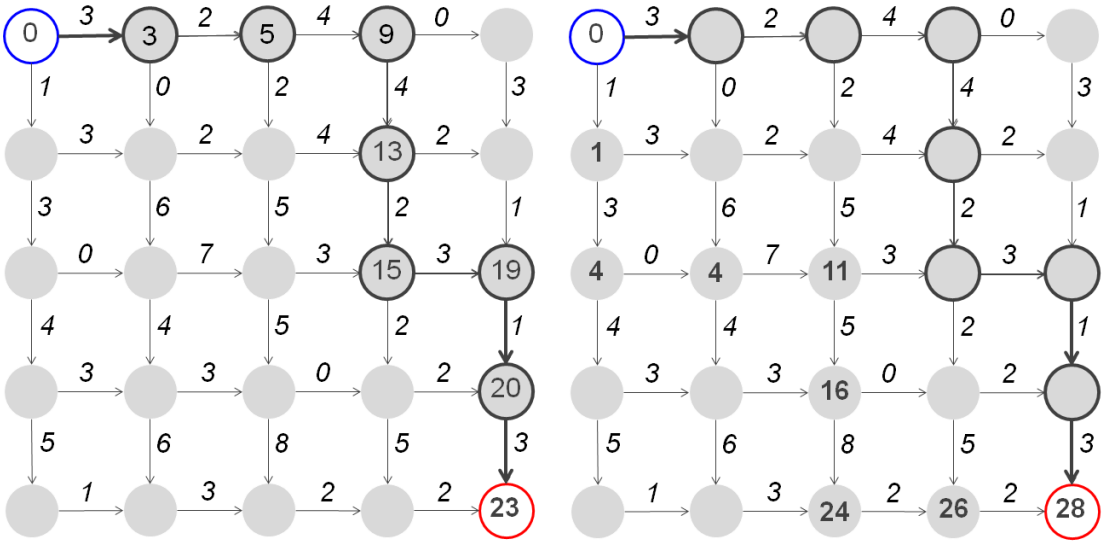


Slika 1.4: Usmereni težinski pravougaoni mrežni graf

Na slici 1.4 prikazan je usmereni težinski pravougaoni mrežni graf sa označenim početnim i krajnjim čvorom. Postavlja se pitanje kako u njemu možemo pronaći najdužu putanju od početnog do krajnjeg čvora. Razmatramo dva naivna pristupa:

- pristup grubom silom bi bio nepraktičan jer je ukupan broj svih mogućih putanja između početnog i krajnjeg čvora ogroman
- pohlepni pristup bi mogao da propusti optimalno rešenje (slika 1.5)

Očigledno, navedenim naivnim pristupima ne možemo rešiti problem turiste sa Menhetna i zbog toga je potrebno razmotriti druge pristupe (u nastavku).

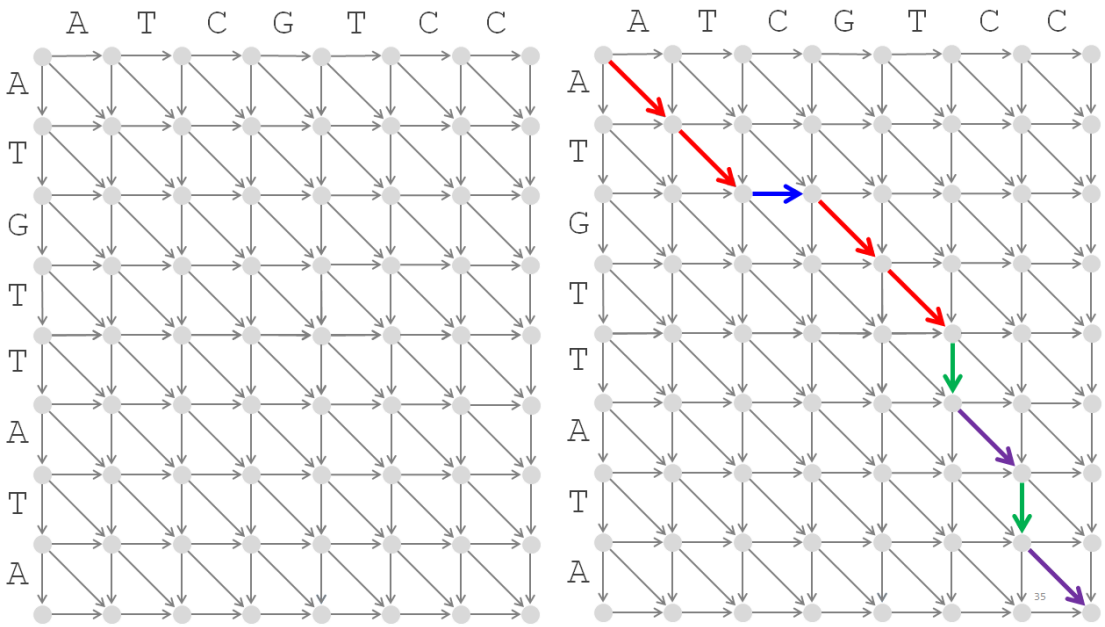


Slika 1.5: Pohlepni pristup propušta optimalno rešenje

Vratimo se na problem poravnanja. Ako smo mogli problem turiste sa Menhetna da modelujemo grafom, da li bismo mogli da slično modelujemo i problem poravnanja? Za dve date sekvence, mrežni graf koji modeluje problem poravnanja možemo izgraditi na sledeći način:

- Kolone mreže označimo aminokiselinama iz prve niske
- Vrste mreže označimo aminokiselinama 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 otežamo koeficijentom 1, ostale koeficijentom 0

Na slici 1.6 predstavljen je usmereni mrežni graf za poređenje sekvenci `ATGTTATA` i `ATCGTCC`. Da bismo pronašli njihovo optimalno poravnanje (ono koje bi u igri poravnanja osvojilo najviše bodova), potrebno je da nađemo najdužu putanju između početnog i krajnjeg čvora. Primetimo da je u odnosu na problem turiste sa Menhetna ovaj graf nešto složeniji jer umesto horizontalnih i vertikalnih ima i dijagonalne grane. Takav graf i dalje ima mrežnu topologiju s tim što mreža nije pravougaona.



Slika 1.6: Problem poravnanja predstavljen usmerenim mrežnim grafom

Neka je dato jedno poravnanje sekvenci `ATGTTATA` i `ATCGTCC`:

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

Za dato poravnanje u grafu možemo konstruisati putanju između početnog i krajnjeg čvora na sledeći način: počev od prve kolone poravnanja, za svako poklapanje i promašaj u putanju dodamo dijagonalnu granu, za svaku inserciju horizontalnu a za svaku deleciju vertikalnu. Na ovaj način, svakom poravnanju odgovara tačno jedna putanja u grafu. Analogno, važi i obrnuto, to jest za datu putanju možemo konstruisati odgovarajuće poravnanje. Na taj način, problem nalaženja optimalnog poravnanja može se svesti na problem nalaženja najduže putanje u usmerenom težinskom mrežnom grafu. Problem turiste sa Menhetna predstavlja specijalni slučaj ovog problema kada je mreža pravougaona.

Već smo razmotrili dva naivna pristupa rešavanju specifičnijeg problema turiste sa Menhetna i uvideli da neće dati dobre rezultate. Zato u nastavku razmatramo nove strategije za rešavanje.

1.4 Problem kusura

Razmotrimo sada nekoliko pristupa na jednostavnom problemu vraćanja kusura:

Problem 3 (Problem vraćanja kusura)**.** *Naći minimalan broj novčića neophodnih za vraćanje kusura.*
Ulaz: Ceo broj money i niz pozitivnih celih brojeva $(coin_1, coin_2, ..., coin_d)$.
Izlaz: Minimalan broj novčića iz datog niza koji rasitnjava sumu money.

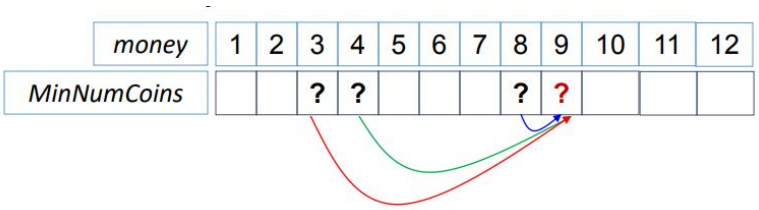
1.4.1 Pohlepni prustup

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 GreedyChange(money)
2 begin
3     change ← prazna kolekcija novcica
4     while money > 0
5         coin ← najveci novcic manji od money
6         add coin to change
7         money ← money - coin
8     return change
9 end
```

1.4.2 Rekurzivni pristup

Pokušajmo sada da problem rešimo rekurzivno. Za zadate apoene 6, 5, 1, koji je najmanji broj novčića neophodnih za vraćanje kusura od 9 centi?



Slika 1.7: Vraćanje kusura - rekurzija

Problem rešavamo tako što prvo od 9 oduzmemo 6 i dobijemo 3 kao ostatak kusura. Dakle, 9 se može vratiti od jednog novčića od 6 apoena i jos plus broj novčiča koji je potreban za preostali deo kusura. U istoj iteraciji analogno računamo za preostale apoene (5 i 1) (slika 1.7). Od tri dobijena rešenja, uzimamo najmanje.

$$\text{MinNumCoins}(9) = \min \begin{cases} MinNumCoins(9-6)+1 = MinNumCoins(3)+1 \\ MinNumCoins(9-5)+1 = MinNumCoins(4)+1 \\ MinNumCoins(9-1)+1 = MinNumCoins(8)+1 \end{cases}$$

U opštem slučaju, rekurentna formula za ovaj rekurzivni postupak je:

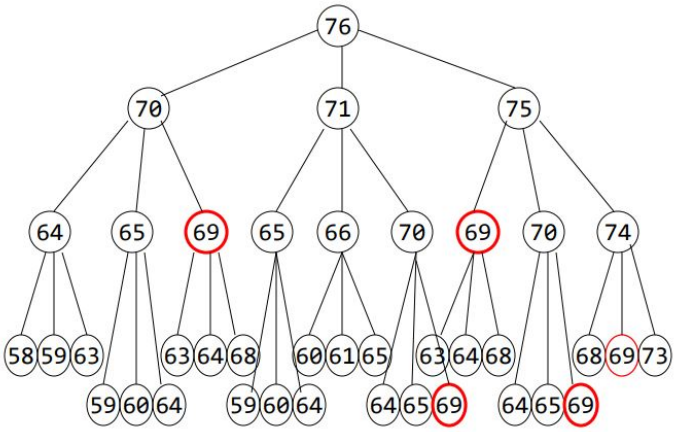
$$\text{MinNumCoins}(\text{money}) = \min \begin{cases} MinNumCoins(money - coin_1)+1 \\ \dots \\ MinNumCoins(money - coin_d)+1 \end{cases}$$

Algoritam koji implementira ovu strategiju dat je u nastavku:

```
1 RecursiveChange(money, coins)
```

```
2  begin
3      if money = 0
4          return 0
5      MinNumCoins ← infinity
6      for i ← 1 to |coins|
7          if money ≥ coin_i
8              NumCoins ← RecursiveChange(money - coin_i, coins)
9              if NumCoins + 1 < MinNumCoins
10                 MinNumCoins ← NumCoins + 1
11     return MinNumCoins
12 end
```

Na ovaj način ćemo ispitati sva moguća rešenja i doći do optimalnog rešenja. Razmotrimo sada koliko je brz RecursiveChange na sledećem primeru. Potrebno je vratiti kusur od 76 centi u apoenima 6, 5 i 1. Ponašanje algoritma RecursiveChange je predstavljeno na slici 1.8.

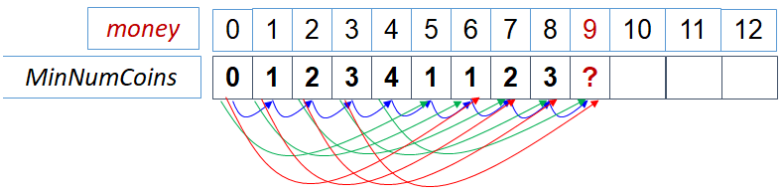


Slika 1.8: Vracanje kusura - ponašanje rekurzivnog algoritma

Možemo primetiti da se algoritam više puta izvršava za istu vrednost, na primer za vrednost od 69 centi, čak 6 puta. Daljim procenama možemo doći do zaključka da se optimalna kombinacija novčića za vraćanje kusura od 30 centi izračunava milijardama puta. Dakle, rekurzivni pristup nije efikasan za rešavanje ovakvih problema.

1.4.3 Pristup dinamičkim programiranjem

Kako bismo izbegli višestruka izračunavanja za istu vrednost, ideja je da počnemo od najmanje vrednosti kusura (1) i redom izračunamo kusur za sve vrednosti do tražene vrednosti (slika 1.9).



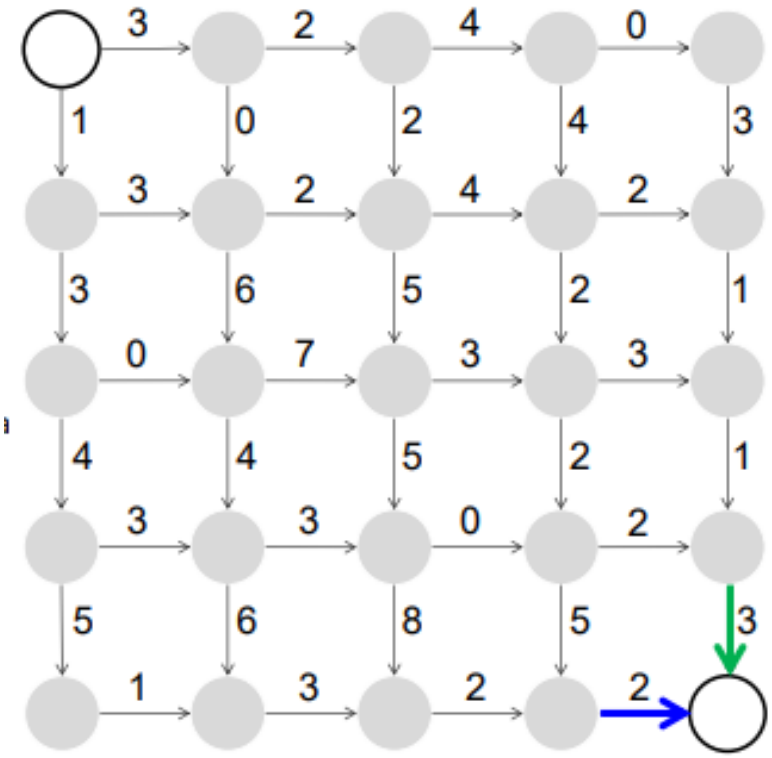
Slika 1.9: Vraćanje kusura - dinamičko programiranje

Sve izračunate vrednosti pamtimo i koristimo ih po potrebi, pa tako umesto vremenski zahtevnih poziva $RecursiveChange(money - coin_i, coins)$ jednostavno bismo potražili vrednosti iz unapred izračunate tabele $MinNumCoins(money - coin_i)$.

```
1  DPChange(money, coins)
2  begin
3      MinNumCoins(0) ← 0
4      for m ← 1 to money
5          MinNumCoins(m) ← infinity
6          for i ← 1 to |coins|
7              if m ≥ coin_i
8                  if MinNumCoins(m - coin_i) + 1 < MinNumCoins(m)
9                      MinNumCoins(m) ← MinNumCoins(m - coin_i) + 1
10     return MinNumCoins(money)
11 end
```

1.5 Dinamičko programiranje i putokazi za povratak

Primenimo sada prethodna razmatranja sa problema vraćanja kusura na problem turiste sa Menhetna. Najpre rekurzivni pristup. Doo krajnjeg čvora (*sink*) možemo doći samo na dva načina: kretanjem na dole (južno ↓) ili kretanjem na desno (istočno →). Rekurzivnim pristupom bismo pošli od krajnjeg čvora i za svaki njemu susedni čvor rekurzivno pozivali funkciju koja pronalazi najdužu putanju od početnog čvora (slika 1.10).



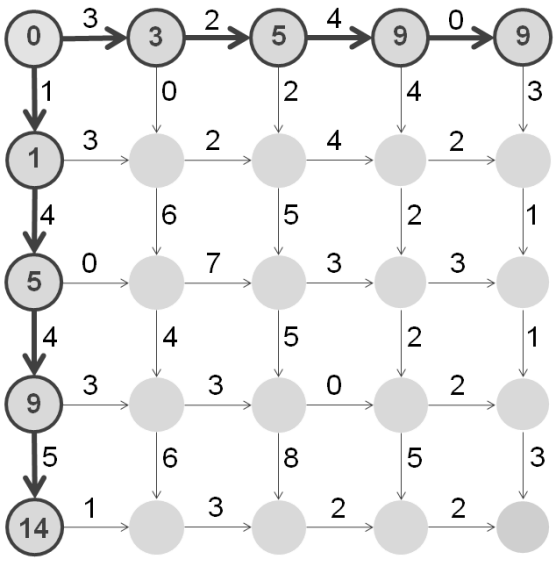
Slika 1.10: Južno ili istočno?

Prvo probamo da rešimo problem rekurzivno:

```
SouthOrEast(n, m)
if n=0 and m=0
    return 0
x ← -infinity, y ← -infinity
if n > 0
    x ← SouthOrEast(n-1,m)+weight of edge "↓" into (n, m)
if m > 0
    y ← SouthOrEast(n,m-1)+ weight of edge "→" into (n,m)
return max{x, y}
```

Ovaj algoritam se poziva za svaki čvor u grafu veličine $m \times n$, a pri tom se dešava da za jedan isti čvor algoritam poziva više puta. Zbog toga je ovaj pristup očekivano previše spor, pa prelazimo na dinamičko programiranje.

Krenućemo od početnog čvora. Zatim, u čvor (i, j) upisujemo dužinu maksimalne putanje od (0,0) do (i,j). Prvo izračunamo za čvorove na obodu grafa jer oni imaju samo po jednu ulaznu granu (slika 1.11).



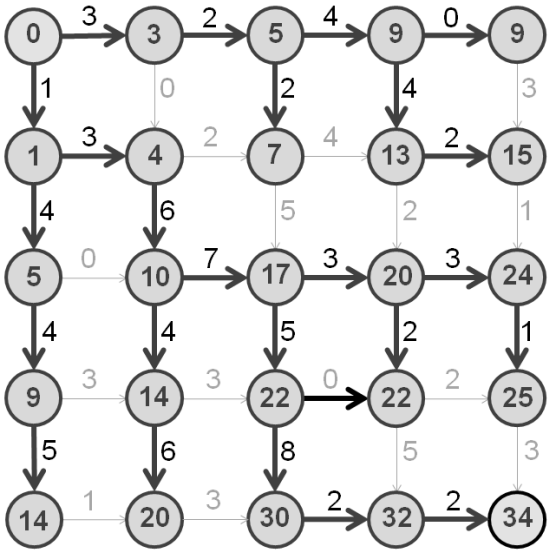
Slika 1.11: Izračunavanje najduže putanje od početnog čvora za čvorove na obodu grafa

Zatim, kolonu po kolonu, izračunavamo odgovarajuće dužine $s_{i,j}$ za preostale čvorove prateći sledeću rekurentnu vezu:

$$s_{i,j} = \max \begin{cases} s_{i-1,j} + tezina \downarrow prema(i,j) \\ s_{i,j-1} + tezina \rightarrow prema(i,j) \end{cases}$$

Unutrašnji čvorovi imaju po dve ulazne grane pa je potrebno zapamtiti kojom granom smo došli do datog čvora, to jest koja grana od dve ulazne učestvuje u najdužoj putanji. Ove grane su obeležene podebljano na slici 1.12. Putanju je moguće rekonstruisati ako krenemo od krajnjeg čvora podebljanim granama u suprotnom smeru. Navedeni postupak je predstavljen pseudokodom `ManhattanTourist` gde su $down_{i,j}$ i $right_{i,j}$ težine odgovarajućih grana.

```
ManhattanTourist(n, m, Down, Right)
s_{0,0} ← 0
```



Slika 1.12: Izračunavanje najduže putanje od početnog čvora za sve čvorove grafa

```
3  for i ← 1 to n
4      s_{i,0} ← s_{i−1,0} + down_{i,0}
5  for j ← 1 to m
6      s_{0,j} ← s_{0,j−1} + right_{0,j}
7  for i ← 1 to n
8      for j ← 1 to m
9          s_{i,j} ← max { s_{i−1,j} + down_{i,j}, s_{i,j−1} + right_{i,j} }
10 return s_{n,m}
```

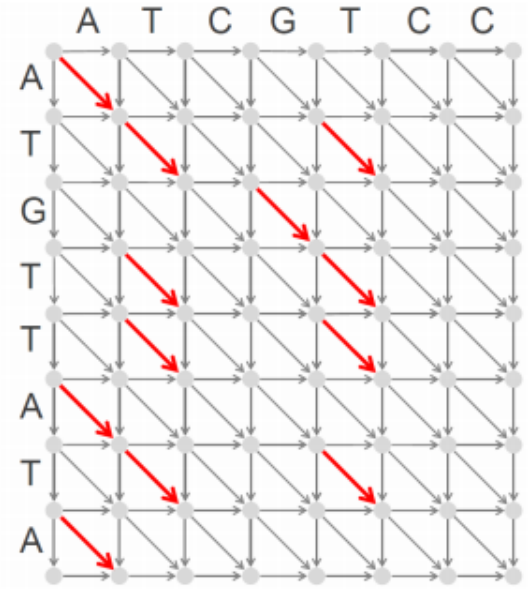
1.6 Od Menhetna do grafa poravnanja

Da bi se algoritam `ManhattanTourist` prilagodio za problem poravnanja, neophodno je izmeniti rekurentnu vezu kojom se računa najduža putanja od početnog čvora do čvora i, j:

$$s_{i,j} = \max \begin{cases} s_{i-1,j} + tezina" \downarrow " prema\ (i,j) \\ s_{i,j-1} + tezina" \rightarrow " prema\ (i,j) \\ s_{i-1,j-1} + tezina\ of\ edge\ " \searrow " prema\ (i,j) \end{cases}$$

Ako uzmemo u obzir da su težine grana kao na slici 1.13, tada se rekurentna veza svodi na:

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



Slika 1.13: Crvene grane imaju težinu 1, ostale težinu 0, v_i i w_j oznake vrste i kolone

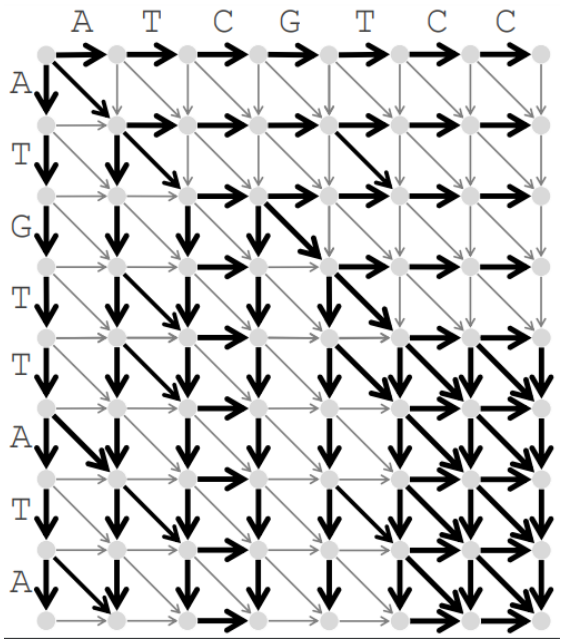
Niz grana koje učestvuju u najdužoj putanji od početnog čvora grafa do svog izlaznog čvora može se dobiti prateći sledeću rekurentnu relaciju polaskom od krajnjeg čvora:

$$backtrack_{i,j} \leftarrow \max \begin{cases} " \rightarrow ", s_{i,j} = s_{i,j-1} \\ " \downarrow ", s_{i,j} = s_{i-1,j} \\ " \searrow ", otherwise \end{cases}$$

Matricu *backtrack* možemo formirati istovremeno kad i matricu $s_{i,j}$ na isti način kao i kod problema turiste sa Menhetna, tako što ćemo prvo obraditi čvorove na obodu grafa a potom unutrašnje čvorove. U ovoj matrici se za čvor na poziciji i, j nalazi informacija kojom granom smo došli do tog čvora. Ovaj postupak je opisan algoritmom *Longest Common Sequence BackTrack* (`LCSBackTrack`) u nastavku.

```
1  LCSBackTrack (v,w)
2  for i ← 0 to |v|
3       s_{i,0} ← 0
4  for j ← 0 to |w|
5       s_{0,j} ← 0
6  for i ← 0 to |v|
7       for j ← 0 to |w|
8            match = 0
9            if v_i = w_j
10                match=1
11           s_{i,j} ← max{s_{i-1,j} , s_{i,j-1} , s_{i-1,j-1} +match}
12           if s_{i,j} = s_{i-1,j}
13                backtrack_{i,j} ← ↓
14           else if s_{i,j} = s_{i,j-1}
15                backtrack_{i,j} ← →
16           else
17                backtrack_{i,j} ← ↘
18 return backtrack
```

Navedenim algoritmom kreirana je matrica *backtrack* na osnovu koje je moguće rekonstruisati tačno jednu najdužu putanju između početnog i krajnjeg čvora. Međutim, takvih putanja može biti i više kada je više od jedne vrednosti na osnovu koju računamo $s_{i,j}$ jednaka maksimalnoj. Sve ove putanje imaju istu dužinu i sa ovakvim težinama određuju najdužu zajedničku podnisku iste dužine. S obzirom na to, svejedno je koju ćemo vrednost izabrati kada više grana daje isti maksimum i u ovoj implementaciji algoritma dajemo prednost grani ↓ bez posebnog razloga. Ukoliko je za pojedinačne primene neophodno dati prednost drugim granama, redosled uslova u naredbi grananja se može lako promeniti. Takođe, algoritam je moguće prilagoditi tako da vraća sve moguće putanje maksimalne dužine ukoliko je to neophodno. Na slici 1.14 su prikazane podebljano grane izdvojene primenom algoritma prilagođenog za kreiranje matrice *backtrack* na osnovu koje je moguće rekonstruisati sve putanje maksimalne dužine.



Slika 1.14: Putokazi za rekonstrukciju najduže putanje

Nakon što odredimo matricu *backtrack*, možemo rekonstruisati *najdužu zajedničku podsekvencu* sledećim rekurzivnim algoritmom:

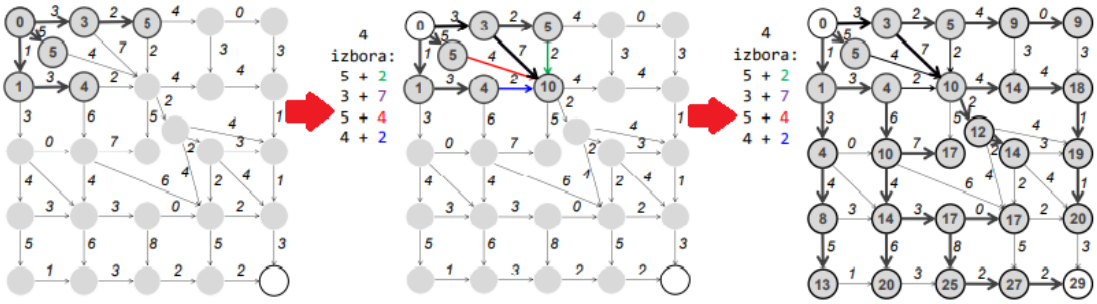
```
1  OutputLCS (backtrack, v, i, j)
2  if i = 0 or j = 0
3      return ""
4  if backtrack_{i,j} = "↓"
5      return OutputLCS (backtrack, v, i-1, j)
6  else if backtrack_{i,j} = "→"
7      return OutputLCS (backtrack, v, i, j-1)
8  else
9      return OutputLCS (backtrack, v, i-1, j-1) + v[i-1]
```

`OutputLCS` vraća najdužu zajedničku podsekvencu za prefikse niski `v` i `w` dužina redom `i` i `j` za izračunatu matricu `backtrack`. Poziv `OutputLCS(backtrack, v, |v|, |w|)` vraća najdužu zajedničku podsekvencu za cele niske `v` i `w`.

1.7 Generalizacija rešenja za proizvoljni usmereni graf

Predstavljeni algoritmi rešavaju problem poravnanja dve sekvence. Pored ove vrste poravnanja, postoje i razne druge, kompleksnije vrste poravnanja koja se ne mogu modelovati mrežnim grafovima. Zbog toga se postavlja pitanje da li se predložena rešenja mogu generalizovati tako da važe za proizvolje usmerene grafove bez mrežne topologije?

Posmatrajmo usmereni graf na slici 1.15 levo. Hteli bismo da primenimo sličan algoritam za označavanje čvorova i grana kako bismo izračunali dužinu najduže putanje od početnog do krajnjeg čvora i obeležili putokaze za njeno nalaženje. Posmatrajmo čvor označen sa 10 na 1.15 sredina. Kako se rekurentna relacija za računanje dužine najduže putanje menja kada nemamo samo dijagonalnu, horizontalnu i vertikalnu ulaznu granu već ih je proizvoljno mnogo?



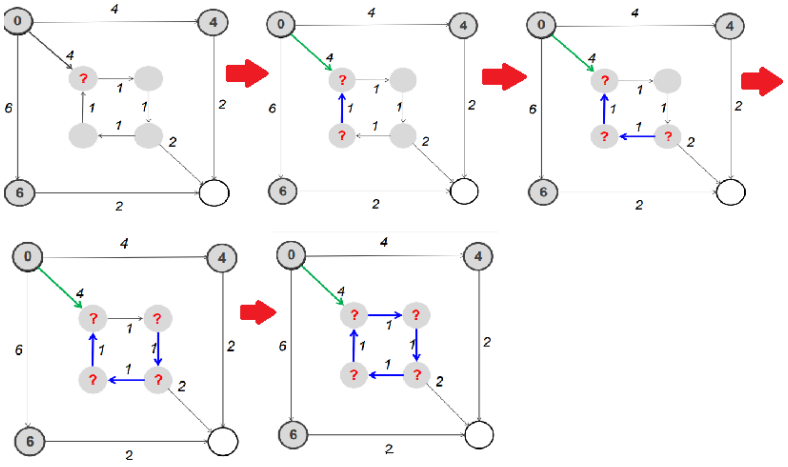
Slika 1.15

Ako posmatrani čvor označimo sa a, tada važi:

$$s_a = max_{svi\ preci\ b\ cvora\ a}\{s_b + tezina\ grane\ od\ b\ do\ a\}$$

Ova rekurentna relacija je zapravo uopštenje rekurentne relacije koja je važila za graf poravnanja na proizvoljni usmereni graf. Ako bismo zamenili rekurentnu relaciju u algoritmu `LCSBackTrack`, mogli bismo da izračunamo dužinu najduže putanje od početnog čvora do bilo kog čvora u grafu, uključujući i krajnji čvor, kao i da označimo grane koje u toj putanji učestvuju (1.15 desno).

Važan detalj na koji je prilikom modifikacije ovog algoritma neophodno obratiti pažnju je redosled obilaska čvorova. Kod mrežnog grafa prvo smo obilazili čvorove na obodu pa onda u unutrašnjosti i time smo implicitno osigurali da smo obišli sve pretke jednog čvora pre nego što taj čvor dođe na red. Postavlja se pitanje kako da ovakav uslov ispunimo kada čvor nema pravilnu topologiju da možemo da obilazimo po vrstama ili po kolonama i da li je ovakav uslov moguće ispuniti za proizvoljni usmereni graf.



Slika 1.16: Začarani krug

Posmatrajmo usmereni graf na slici 1.16 gore levo. Želeli bismo da izračunamo dužinu najduže putanje od početnog čvora do čvora označenog upitnikom. Da bismo primenili rekurentnu relaciju, moramo da znamo odgovarajuće dužine za sve njegove pretke. Međutim, ako pokušamo da izračunamo odgovarajuće dužine za sve pretke posmatranog čvora, vratićemo se upravo do njega (slika 1.16 dole desno). Možemo zaključiti da se rekurentna relacija ne može primeniti na sve grafove već samo na grafove koji ne sadrže cikluse, odnosno na *usmerene acikličke grafove* (često se ovakav graf naziva DAG, skraćeno od engleskog naziva *directed acyclic graph*).

Redosled obilaska čvorova koji će obezbediti da pre svakog čvora obiđemo sve njegove pretke postiže se *topološkim* sortiranjem čvorova grafa. Topološki redosled čvorova grafa podrazumeva da će se čvor i nalaziti ispred čvora j ako u grafu postoji grana $i \rightarrow j$ Može se pokazati da svaki DAG ima bar jedan topološki redosled koji se može konstruisati u vremenu proporcionalnom broju grana u grafu ($O(\#egdes)$). Nakon topološkog sortiranja niza čvorova grafa, čvorovi se obilaze u dobijenom redosledu. Algoritam `LongestPath` prikazuje postupak za računanje najduže putanje u proizvoljnom DAG-u sa označenim početnim i krajnjim čvorom.

```
1  LongestPath(Graph, source, sink)
2  for each node a in Graph
3        s_a ← -∞
4  s_source ← 0
5  topologically order Graph
6  for each node a (from source to sink in topological order)
7       s_a ← max_{all predecessors b of node a} {s_b + weight of edge from b to a}
8  return s_sink
```

1.8 Skor poravnanja i vrste poravnanja

Do sada smo u igri poravnanja podrazumevali najjednostavniji slučaj kada se poklapanja boduju sa 1 a ostalo sa 0. To ne mora biti tako. Možemo imati skor različit od nule za promašaje i za indele. Ako ih označimo redom sa μ i σ, onda skor poravanja možemo računati kao $\#matches - \mu * \#mismatches - \sigma * \#indels$. Na slici 1.17 prikazan je primer računanja skora za $\mu = 2, \sigma = 3$.

```
A T - G T T A T A
A T C G T - C - C
+1+1-2+1+1-2-3-2-3=-7
```

Slika 1.17: Računanje skora poravnanja

U opštem slučaju, svaki promašaj i svaki indel se može različito ceniti. Tada ove vrednosti ima smisla predstaviti matrično i takvu matricu nazivamo matricom skora. Jedna takva matrica prikazana je na slici 1.18. Obratimo pažnju da takva matrica ne mora biti simetrična, jer se na primer češće dešavati da G mutira u T nego obrnuto pa ćemo takve situacije u poravnanju manje kažnjavati. Ovakve matrice se koriste i kod poravnanja proteinskih sekvenci.

	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	

Slika 1.18: Matrica poravnanja

Označimo matricu poravnanja sa *score*. Uzevši u obzir izmenu u načinu računanja skora poravnanja, rekurentna relacija za računanje dužine najduže putanje s u grafu poravnanja se menja:

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

Sada kada smo formalno definisali skor poravnanja, možemo formalno definisati i problem poravnanja o kom smo do sada govorili. Takvo poravnanje je samo jedna od vrsta poravnanja i naziva se *globalno poravnanje*.

Problem 4 (Problem globalnog poravnanja)**.** *Naći poravnanje sa najvišim skorom između dve niske za datu matricu skora.*
Ulaz: Niske v i w, kao i matrica skora score
Izlaz: Poravnanje niski v i w čiji je skor poravnanja (prema matrici skora) maksimalan od svih mogućih poravnanja v i w.

Posmatrajmo sledeća dva poravnanja u kojima poklapanja, promašaji i indeli vrede po -1:

```
GCC-C-AGT--TATGT-CAGGGGGCACG--A-GCATGCAGA-
GCCGCC-GTCGT-T-TTCAG----CA-GTTATG--T-CAGAT
```

Slika 1.19
score = 22(matches) - 2(indent) = 2

```
---G----C-----C--CAGTTATGTCAGGGGGCACGAGCATGCAGA
GCCGCCGTCGTTTTCAGCAGTTATGTCAG-----A------T-----
```

Slika 1.20
score = 17 (matches) - 30(indent) = -13

Kao što vidimo, levo poravnanje ima mnogo veći skor nego desno poravnanje. Sa druge strane, u desnom poravnanju postoji veći broj uzastopnih poklapanja. To znači da su ove dve sekvence u jednom delu veoma slične a u ostalim delovima nema sličnosti. Ovakva poravnanja imaju veliki biološki značaj. Ona se često dešavaju kada poravnavamo dva ista gena različitih vrsta. Isti geni imaju konzervirane regione koji su vrlo slični, sa velikim brojem poklapanja, dok su ostali delovi različiti. Jedan takav primer je *homeobox* gen koji se pojavljuje u različitim vrstama (na primer, kod čoveka i kod muve) kao različita sekvenca, a jedan njegov deo, *homeodomen* je u svim vrstama isti. U ovakvim primerima, umesto da tražimo globalno poravnanje sa najvećim skorom, želimo da pronađemo globalno poravnanje između svih mogućih podniski jedne i druge sekvence sa najvećim skorom. Ovakvo poravnanje se naziva *lokalno poravnanje*. Na slici 1.21 je prikazano globalno i lokalno poravnanje za dati primer.

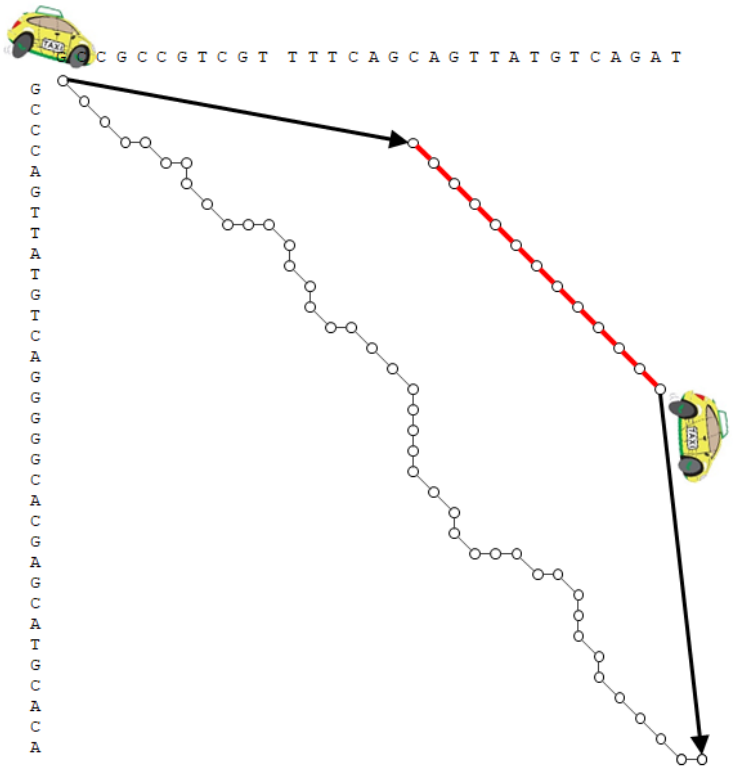
Slika 1.21: Globalno i lokalno poravnanje

Problem lokalnog poravnanja sada možemo i formalizovati.

Problem 5 (Problem lokalnog poravnanja)**.** *Naći lokalno poravnanje najvećeg skora između dve niske.*
Ulaz: Niske v i w, kao i matrica skora score
Izlaz: Podniske niski v i w čije je globalno poravnanje (prema matrici skora) maksimalno među svim globalnim poravnanjima svih podniski niski v i w.

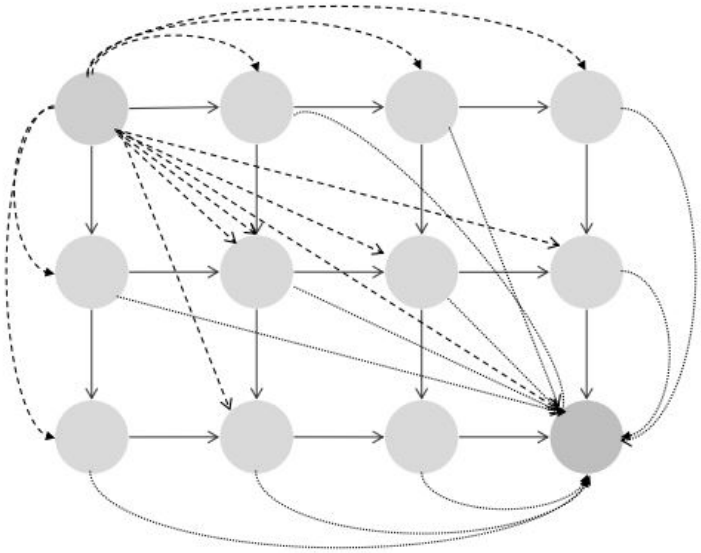
Problem globalnog poravnanja smo rešavali nalaženjem najduže putanje između početnog (*source*) i krajnjeg čvora (*sink*). Primena takvog rešenja za problem lokalnog poravnanja bi značila da za svaka dva čvora u grafu poravnanja pronađemo najdužu putanju koja ih povezuje i od svih njih odaberemo putanju sa najvećim skorom poravnanja. Ako pretpostavimo da u grafu poravnanja imamo ukupno $\#nodes$ čvorova, onda bismo algoritam za traženje najduže putanje morali da pokrenemo $O(\#nodes^2)$ puta. U nastavku ćemo pokušati da pronađemo efikasniji način.

Na slici 1.21 smo videli da će poravnanje sa najboljim skorom, računatim tako što se svaka kolona u poravnanju posebno boduje odgovarajućim skorom iz matrice poravnanja, biti globalno poravnanje. Razmotrimo kako možemo da promenimo način računanja skora poravnanja tako da poravnanje sa najboljim skorom bude lokalno poravnanje. Da bi skor bio maksimalan kod lokalnog poravnanja, ima smisla ukinuti kažnjavanje od početnog čvora do prvog čvora poklapanja i od poslednjeg čvora poklapanja do krajnjeg čvora. Na slici ... ovaj princip je prikazan kroz `besplatne vožnje`: možemo započeti poklapanje na bilo kojoj poziciji i dolazak do te pozicije ne košta ništa, kao ni od završetka poklapanja do poslednjeg čvora.



Slika 1.22: Besplatne vožnje

Kada je lokalno poravnanje poznato, jasno je između kojih čvorova su potrebne besplatne vožnje. Kada lokalno poravnanje nije poznato, potrebno je dopustiti besplatne vožnje do i od svakog čvora u grafu. Da bismo graf poravnanja prilagodili ovom konceptu, potrebno je da dodamo grane od početnog čvora do svih drugih čvorova, kao i od svakog čvora do krajnjeg čvora grafa. Na slici 1.23 je zbog preglednosti prikazan Mehnetn graf sa dodatim granama koje modeluju besplatne vožnje.



Slika 1.23: Menhetn graf za problem lokalnog poravnanja

Računanje najduže putanje u DAG-u je proporcionalno broju grana. U grafu poravnanja sekvenci v i w, ukupan broj grana je $|v| * |w|$, a sa dodavanjem besplatnih vožnji, ukupan broj će biti $O(|v| * |w|)$ što je i vremenska složenost algoritma za nalaženje najduže putanje. Rekurentna relacija će se izmeniti na sledeći način:

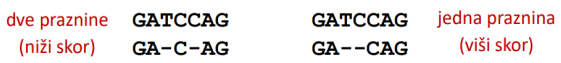
$$s_{i,j} = \max \begin{cases} 0 \\ s_{i-1,j} + score(v_i, -) \\ s_{i,j-1} + score(-, w_j) \\ s_{i-1,j-1} + + score(v_i, w_j) \end{cases}$$

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

$$s_{n,m} = \max_{0 \le i \le n, 0 \le j \le m} s_{i,j}$$

1.9 Kažnjavanje insercija i delecija u poravnanju sekvenci

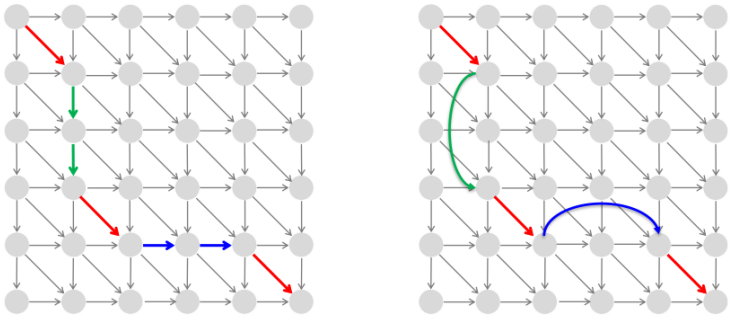
Za globalno poravnanje korišćen je linearni model za računanje skora poravnanja gde se svaki indel posebno računa. Kod lokalnog poravnanja ovaj princip nije odgovarajuć jer ne želimo da kažnjavamo duge nizove praznina. Naime, uzastopni nizovi praznina su većeg biološkog značaja jer predstavljaju jedan isti evolucioni događaj. Želeli bismo da skor poravnanja bude takav da skor poravnanja na slici 1.24 levo bude manji nego na slici desno.



Slika 1.24

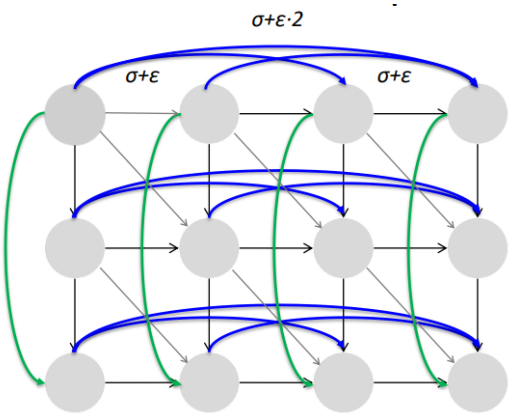
Zbog toga se za lokalno poravnanje uvodi afina kazna za praznine dužine k: $\sigma + \epsilon(k-1)$, gde je σ kazna za otvaranje praznine a ϵ kazna za proširenje praznine, pri čemu je $\sigma > \epsilon$.

Da bismo graf poravnanja prilagodili afinoj kazni za praznine, potrebno je da dodamo nove horizontalne i vertikalne grane između svaka dva čvora grafa gde je to moguće. Ove takozvane *duge grane* će predstavljati praznine dužine k i ilustrovane su na slici 1.25.



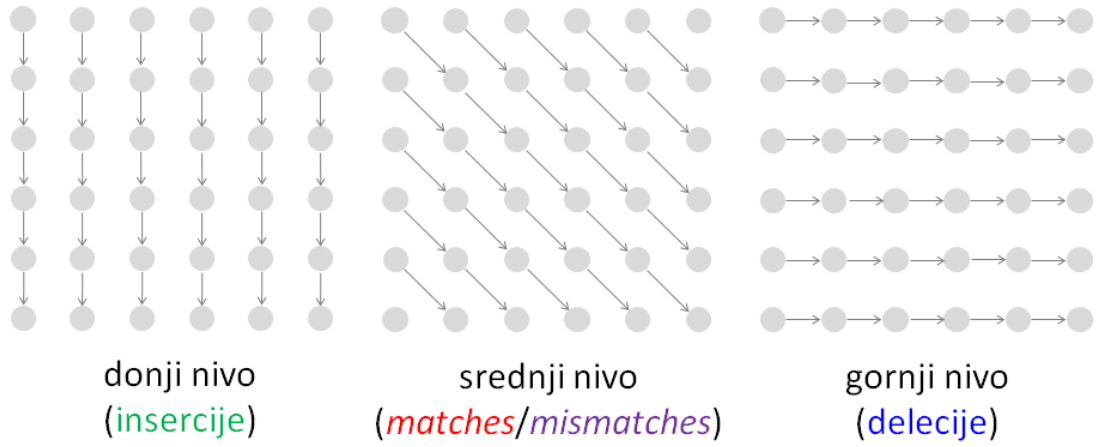
Slika 1.25: Modelovanje afinih kazni za praznine pomoću dugih grana

Preciznije, za svaki čvor (i, j) grafa poravnanja dodajemo grane ka čvorovima $(i+k, j)$ i $(i, j+k)$ sa težinama $\sigma + \epsilon(k-1)$ za sve moguće vrednosti k (slika 1.26). Za dve sekvence dužine n, broj grana u grafu ovim dodavanjem raste sa $O(n^2)$ na $O(n^3)$.



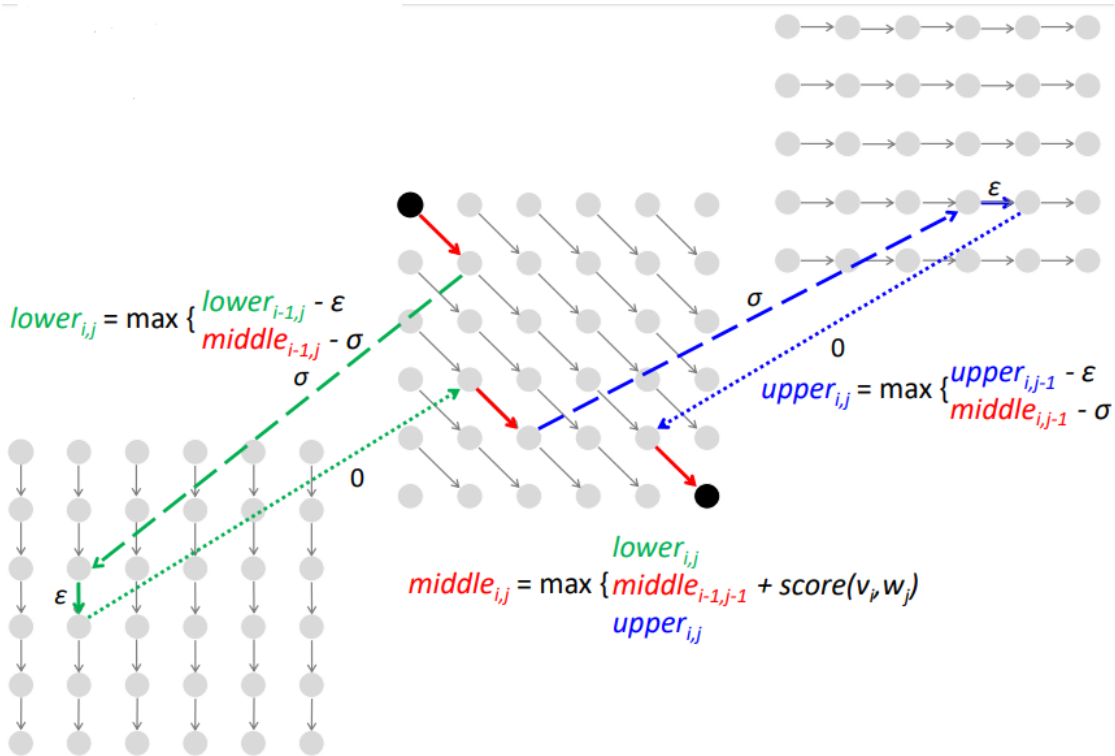
Slika 1.26: Duge grane i njihove težine

S obzirom da je vremenska složenost određivanje skora poravnanja proporcionalna broju grana koji je ovde kubni u odnosu na dužinu jedne sekvence, potrebno je smanjiti broj grana u grafu poravnanja. Broj čvorova ne figuriše u ovoj vremenskoj složenosti pa ga možemo povećati. Graf poravnanja možemo predstaviti na tri nivoa tako što ćemo za svaki čvor (i, j) uvesti tri nova čvora: $(i, j)_{lower}, (i, j)_{middle}, (i, j)_{upper}$ (slika 1.27). Srednji nivo sadrži dijagonalne grane težine određene matricom skora $score(v_i, w_j)$. Niži nivo sadrži samo vertikalne čvorove težine ϵ i predstavlja praznine duže od 1 u sekvenci v (insercije), a viši samo horizontalne čvorove iste težine koji predstavljaju praznine duže od 1 u sekvenci w (delecije).



Slika 1.27: Tri nivoa grafa poravnanja

Za povezivanje tri nivoa grafa, potrebno je da dodamo grane koje će modelovati otvaranje i zatvaranje praznina. Za modelovanje otvaranja praznina, svaki čvor $(i,j)_{middle}$ povezaćemo i sa $(i+1,j)_{lower}$ i sa $(i,j+1)_{upper}$ i otežati ove grane sa $-\sigma$. Zatvaranje praznine ne želimo da kažnjavamo i zato uvodimo grane težine 0 između $(i,j)_{lower}$ i (i,j) i između $(i,j)_{upper}$ i (i,j). Na taj 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, čime smo dobili $\sigma + \epsilon(k-1)$ kao što smo i želeli (slika 1.28).



Slika 1.28: Graf poravnanja sa granama između nivoa

Konstruisani graf ima svega $O(nm)$ grana za sekvence dužina n i m i najduža putanja u grafu odgovara optimalnom poravnanju sa afinom kaznom za praznine. Ovaj trostruki graf se može predstaviti sistemom od tri rekurentne veze:

$$
\begin{aligned}
lower_{i,j} &= \max \begin{cases} lower_{i-1,j} & -\epsilon \\ middle_{i-1,j} & -\sigma \end{cases} \\
upper_{i,j} &= \max \begin{cases} upper_{i,j-1} & -\epsilon \\ middle_{i,j-1} & -\sigma \end{cases} \\
middle_{i,j} &= \max \begin{cases} lower_{i,j} \\ middle_{i-1,j-1} & + score(v_i, w_j) \\ upper_{i,j} \end{cases}
\end{aligned}
$$

Slika 1.29: Rekurentne veze za trostruki graf poravnanja ($lower_{i,j}, middle_{i,j}$ i $upper_{i,j}$ predstavljaju dužine najdužih putanja od početnog čvora do čvorova $(i,j)_{lower}, (i,j)_{middle}$ i $(i,j)_{upper}$, redom)

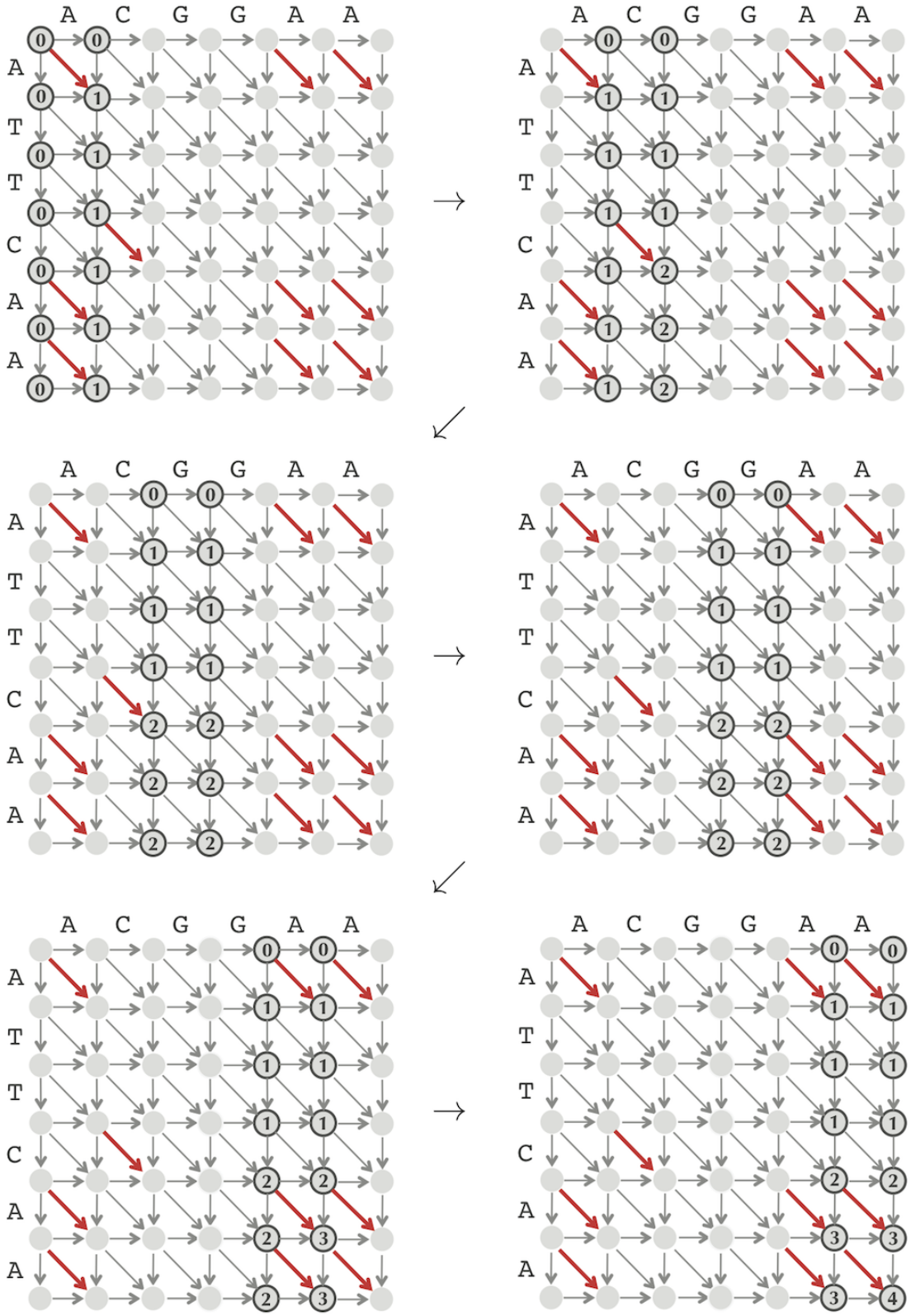
Primetimo da vrednost $lower_{i,j}$ predstavlja skor optimalnog poravnanja između i-tog prefiksa sekvence v i j-tog prefiksa sekvence w koji se završava delecijom (to jest vertikalnom granom), dok vrednost $upper_{i,j}$ predstavlja skor optimalnog poravnanja između ovih prefiksa koji se završava insercijom (to jest horizontalnom granom) a $middle_{i,j}$ računa skor optimalnog poravnanja koje se završava poklapanjem ili promašajem. Prvi slučaj u rekurentnim relacijama za $lower_{i,j}$ i $upper_{i,j}$ odgovara proširenju praznine dok drugi slučaj odgovara otvaranju praznine.

1.10 Poboljšanje prostorne složenosti poravnanja sekvenci

Proteini su sekvence dužine od nekoliko desetina do nekoliko desetina hiljada. Na primer, NRP sintetaze mogu sadržati i do 20 000 aminokiselina. Kao što je bilo prethodno reči, vremenska složenost dinamičkog algoritma poravnanja je proporcionalna broju grana ($O(nm)$). Razmotrimo i prostornu složenost. S obzirom da je za rekonstrukciju najduže putanje u grafu poravnanja potrebno čuvati putokaze od krajnjeg do početnog čvora, prostor potreban za njihovo smeštanje proporcionalan je broju čvorova ($O(nm)$). Ako poravnavamo duge sekvence od nekoliko desetina hiljada aminokiselina, matrica u kojoj bi se čuvali putokazi bi bila ogromna.

U nastavku ćemo pokazati kako možemo da konstruišemo poravnanje u linearnom prostoru ($O(n)$) uz udvostručavanje vremena izvršavanja (koje i dalje ostaje kvadratno) pomoću strategije *podeli-pa-vladaj*.

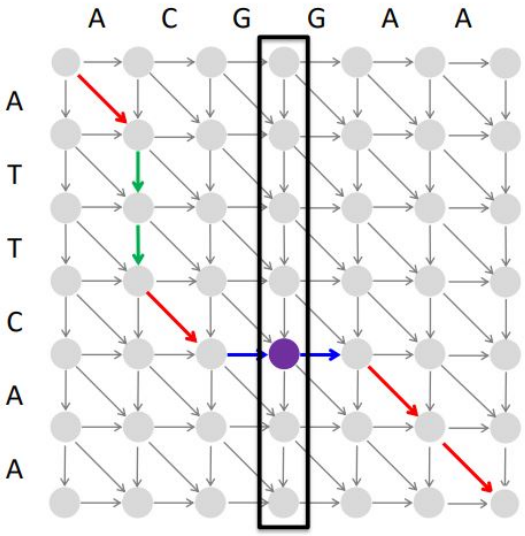
Primetimo da za računanje skora najduže putanje nije neophodno čuvati celu matricu poravnanja. Naime, za računanje skorova za čvorove jedne kolone dovoljni su čvorovi prethodne kolone, što znači da je u svakom trenutku dovoljno čuvati matricu od $2n = O(n)$ elemenata (slika 1.30).



Slika 1.30: Računanje skora poravnanja po kolonama. Za računanje skora za čvorove jedne kolone potrebne su samo vrednosti prethodne kolone. Prikazan je graf za globalno poravnanje gde grane koje odgovaraju poklapanju imaju težinu 1 (crvene), dok ostale imaju težinu 0.

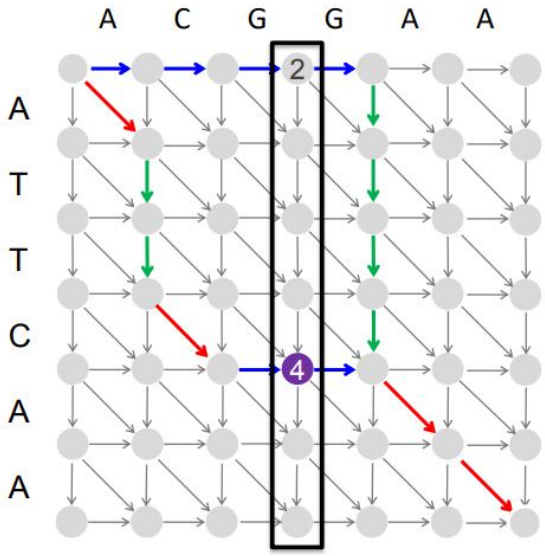
1.10.1 Srednji čvor

Neka je dat graf poravnanja za sekvence v dužine n i w dužine m. Označimo sa $middle = \lfloor m/2 \rfloor$. Srednjom kolonom zvaćemo kolonu na poziciji $middle$. Najduža putanja mora negde preseći srednju kolonu i taj čvor preseka ćemo zvati srednjim čvorom (slika 1.31).



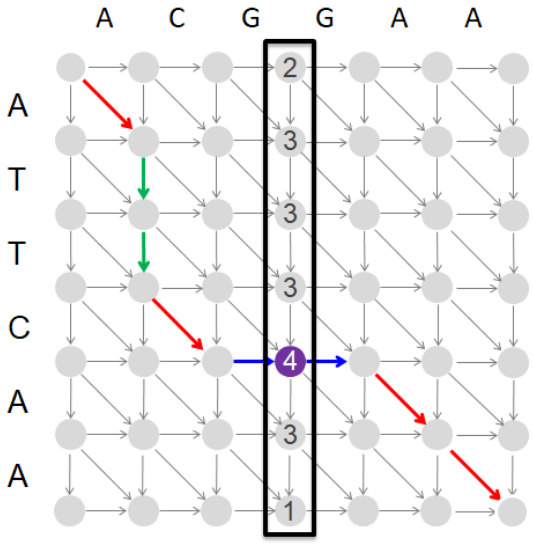
Slika 1.31: Srednja kolona i čvor poravnanja

Primetimo da za *nalaženje srednjeg čvora nije neophodno da rekonstruišemo putanju u grafu poravnanja*. Putanju od početnog do krajnjeg čvora zvaćemo i-putanjom ako prolazi kroz srednju kolonu u čvoru i. Na primer, putanja na slici 1.31 je 4-putanja jer sadrži 4. čvor srednje kolone (numeracija čvorova kreće od nule). Dužina ove i-putanje je 4. Pretpostavimo da imamo način da izračunamo dužine najdužih i-putanja za svako i (označićemo ih sa $length(i)$). Na slici 1.32 su prikazane 0-putanja i 4-putanja a njihove dužine su upisane u odgovarajuće srednje čvorove.



Slika 1.32: Najduže i-putanje za $i = 0$ i $i = 4$

Na slici 1.33 su prikazane dužine svih najdužih i-putanja upisanim u odgovarajućim čvorovima srednje kolone. Primetimo da od svih najdužih i-putanja, ona sa maksimalnom dužinom prolazi kroz srednji čvor.

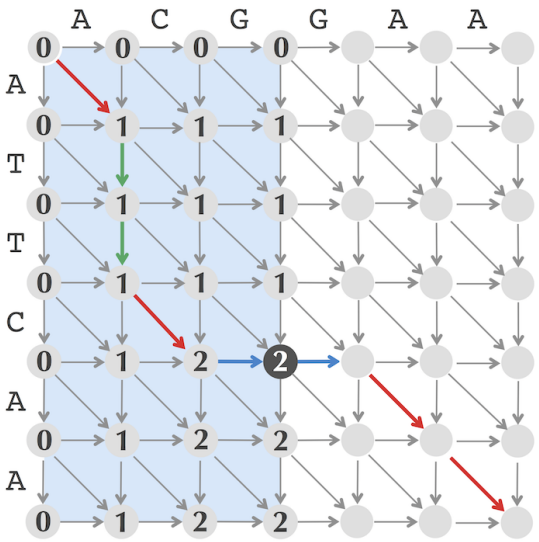


Slika 1.33: Dužine najdužih i-putanja za svako i

Razmotrimo sada kako bismo računali $length(i)$. Dužinu svake i-putanje možemo podeliti na dužinu putanje od početnog čvora do čvora i ($fromSource(i)$) i od čvora i do krajnjeg čvora ($toSink(i)$) pa važi:

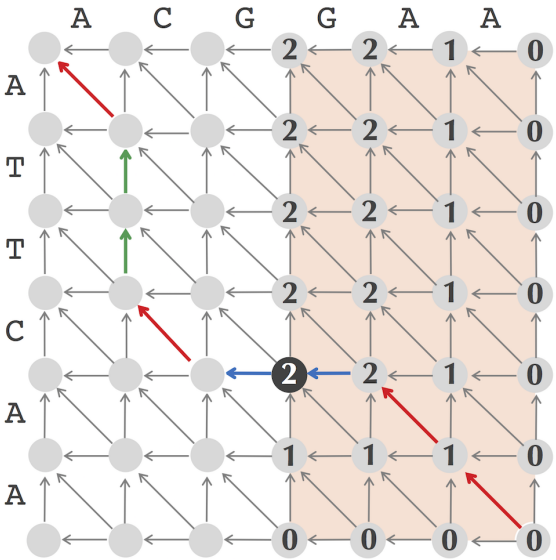
$$length(i) = fromSource(i) + toSink(i)$$

Veličina $fromSource(i)$ predstavlja najdužu putanju od početnog čvora do čvora $(i, middle)$ i odgovara veličini $s_{i,middle}$. Već smo razmotrili da se skor poravnanja, a time i cela matrica s, može izračunati u linearnom prostoru. što se vremenske složenosti tiče, ona je proporcionalna broju grana, a kako ih ovde ima polovina od ukupnog broja, ona iznosi $nm/2$ (slika 1.34).



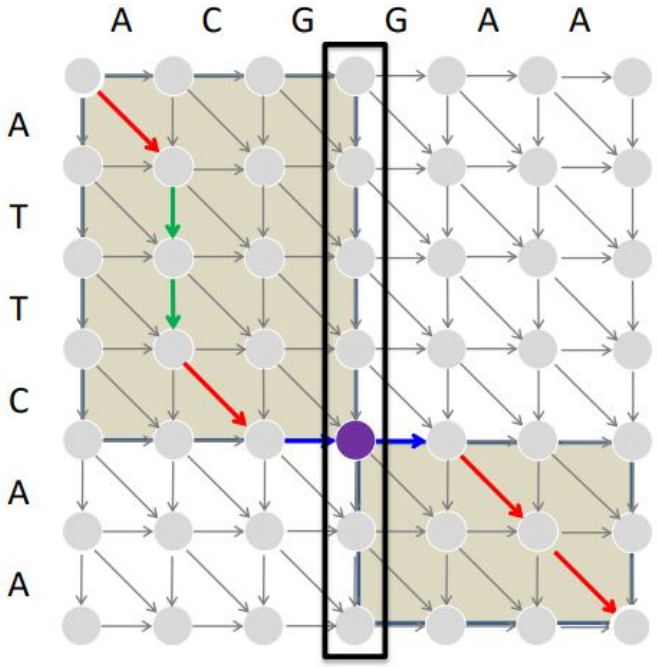
Slika 1.34: Računanje $fromSource(i)$

Računanje veličine $toSink(i)$ svodi se na računanje veličine $toSink(i)$ ako obrnemo smer svake grane (slika 1.35). Možemo reći da je za računanje srednjeg čvora potrebno $O(n)$ prostora i $O(nm)$ vremena.



Slika 1.35: Računanje $toSink(i)$

Poznavanje srednjeg čvora pruža informaciju o delu grafa gde će se nalaziti najduža putanja. Kao što je prikazano na slici **??**, putanja mora sadržati čvorove iz dva pravougaonika, gde je prvi određen početnim i srednjim čvorom a drugi srednjim i krajnjim čvorom. Njihova ukupna površina odgovara polovini površine grafa poravnanja.



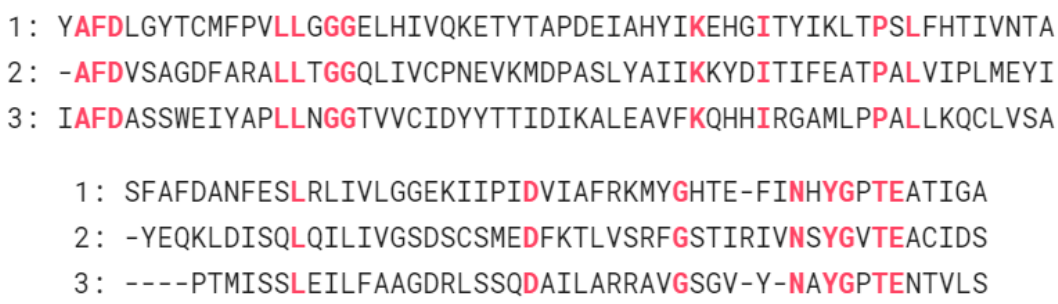
Slika 1.36: Izdvajanje delova grafa kroz koje prolazi najduža putanja

Sada možemo *podeliti* problem nalaženja najduže putanje od $(0,0)$ do (n,m) na dva potproblema: nalaženja najduže putanje od $(0,0)$ do srednjeg čvora i od srednjeg čvora do (n,m). Korak `vladaj` podrazumeva nalaženje srednjeg čvora u manjim pravougaonicima, za šta je potrebno vreme proporcionalno ukupnoj povr sini manjih pravougaonika, $O(nm/2)$. Svaki srednji čvor koji pronađemo je jedan čvor najduže putanje pa ćemo, ako nastavimo sa ovim postupkom, na kraju doći do pravougaonika dimenzije 1x1 i rekonstruisati sve čvorove putanje. Prostorna složenost celog postupka je kao što smo već analizirali $O(n)$. Razmotrimo sada vremensku složenost. Vremenska složenost nalaženje srednjeg čvora u svakom potproblemu je proporcionalna broju grana koji odgovara površini grafa poravnanja. U prvom koraku imaćemo nm operacija, u narednom $nm/2$, zatim $nm/4$, $nm/8\ldots$, što je ukupno $< 2nm \sim O(nm)$. Vidimo da je strategijom *podeli, pavladaj* postignuto željeno poboljšanje prostorne složenosti na linearnu, dok je vremenska složenost udvostručena ali je ostala kvadratna.

1.11 Višestruko poravnanje sekvenci

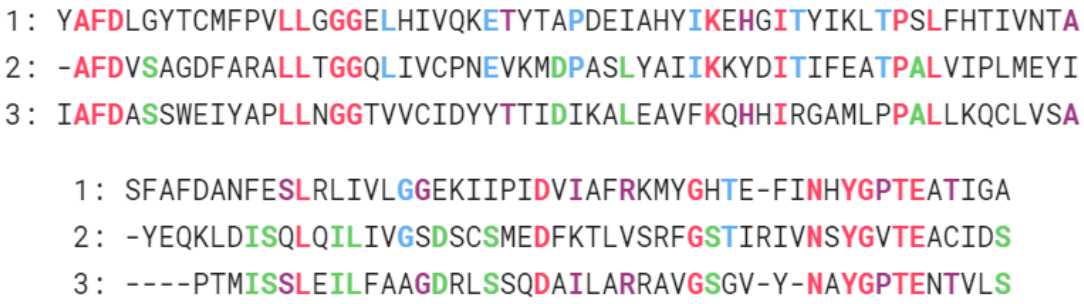
Proteini sa istom funkcijom mogu imati donekle sličan sastav ali je ove sličnosti teško detektovati u slučaju različitih vrsta organizama. Danas postoji veliki broj algoritama za poravnanje dve sekvence ali ako je njihova sličnost slaba, poravnanje dve sekvence možda neće identifikovati da su one biološki povezane. Sa druge strane, poređenje više sekvenci nam često može pružiti uvid u sličnosti koje se ne vide u poređenju dve sekvence.

Podsetimo se da smo u poravnanju tri A-domena pronašli 19 konzerviranih kolone (slika 1.37).



Slika 1.37: Poravnavanja 3 A-domena

Ako posmatramo poravnanja po parovima, pronaći ćemo još 10+9+12=31 polukonzerviranu kolonu gde se po dve aminokiseline poklapaju (slika 1.38).



Slika 1.38: Konzervirane i polukonzervirane kolone poravnavanja 3 A-domena

Višestruko poravnanje sekvenci $v_1, \ldots, v_t$ definisano je matricom od t redova gde se na poziciji (i, j) može naći simbol sekvence v_i ili prazninu (simbol -). Podrazumevano, nijedna kolona matrice višestrukog poravnanja ne sadrži samo praznine. Jedan primer je dat na slici 1.39.

Slika 1.39: Matrica višestrukog poravnanja

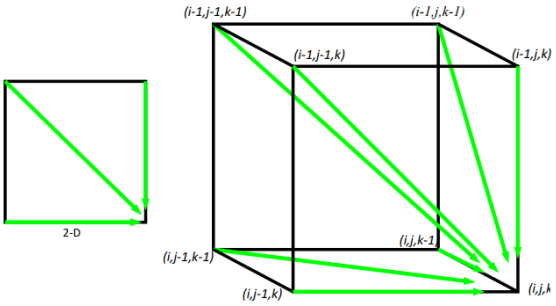
Višestruko poravnanje predstavlja generalizaciju poravnanja dve sekvence na proizvoljan broj sekvenci. Dimenzija grafa poravnanja se u tom slučaju menja i odgovara ukupnom broju sekvenci. U primeru na slici 1.40 je svakom simbolu različitom od praznine pridružen njegov indeks u sekvenci. Ako pročitamo ove indekse po kolonama kao torke, dobićemo koordinate čvorova najduže putanje u grafu poravnanja.

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

(0,0,0)→(1,1,0)→(1,2,1) →(2,3,2) →(3,3,3) →(4,4,4)

Slika 1.40: Najduža putanja u grafu višestrukog poravnanja

Kod poravnanja dve sekvence, susedni čvorovi su formirali kvadrate i svaki unutrašnji čvor je imao tri ulazne grane. Kod višestrukog poravnanja, susedni čvorovi formiraju kocke i svaki ima sedam grana (slika 1.41).



Slika 1.41: Susedni čvorovi kod dvostrukog i višestrukog poravnanja

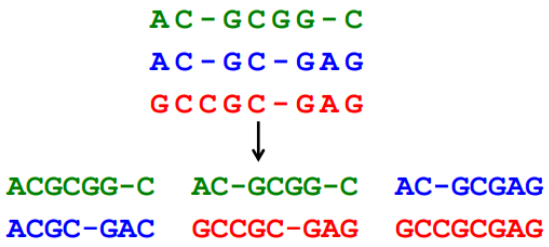
Skor višestrukog poravnanja predstavlja zbir režina grana u grafu poravnanja dok je optimalno poravnanje ono koje maksimizuje ovaj skor. Rekurentna relacija za računanje skora višestrukog poravnanja generalizuje rekurentnu relaciju za dvostruko poravnanje. Ako poravnavamo tri sekvence, ova relacija je:

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

Matrica *score* je u ovom slučaju trodimenzionalna i sadrži vrednosti za sve moguće kombinacije simbola odgovarajuće azbuke (nukleotida ili aminokiselina) uključujući i praznine. Vremenska složenost dinamičkog algoritma poravnanja prilagođenog za višestruko poravnanje t sekvenci je proporcionalna broju grana koji za jedan čvor iznosi najviše $2^t - 1$, a ako su sekvence dužine n graf sadrži n^t čvorova pa je ukupna složenost $O(2^t n^t)$. Kako raste broj sekvenci t, ovakav algoritam postaje prespor. Različiti heuristički algoritmi su predloženi kako bi se ovo usko grlo prevazišlo. Jedan od njih je pohlepni algoritam za višestruko poravnanje.

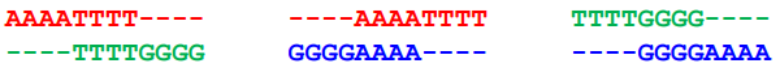
1.11.1 Pohlepni algoritam za višestruko poravnanje

Svako višestruko poravnanje uključuje i dvostruka poravnanja njegovih sekvenci (slika 1.42). Ta poravnanja ne moraju biti optimalna.



Slika 1.42

Razmotrimo da li važi obrnuto: ako su data optimalna dvostruka poravnanja za parove od tri sekvenci (slika 1.43), da li se ona mogu kombinovati u višestruko poravnanje?



Slika 1.43: Dvostruka poravnanja za tri sekvence

Na osnovu prvog poravnanja, AAAA se pojavljuje pre TTTT u višestrukom poravnanju konstruisanim od ovih dvostrukih poravnanja. Na osnovu trećeg poravnanja, TTTT treba da se pojavi pre GGGG u višestrukom poravnanju. Međutim, u drugom poravnanju vidimo da se GGGG mora pojaviti pre AAAA. Stoga, AAAA se mora pojaviti pre TTTT, koje se mora pojaviti pre GGGG, koje se mora pojaviti pre AAAA, što dovodi do kontradikcije.

Kako bi se izbegla ovakva nekompatibilnost, neki algoritmi za višestruko poravnanje pokušavaju da pohlepno konstruišu višestruko poravnanje od dvostrukih poravnanja koja ne moraju biti optimalna. Pohlepna heuristika za početak od svih dvostrukih poravnanja izabere ono sa najvećim skorom i izgradi *profilnu sekvencu*. Na taj način se problem poravnanja t sekvenci svodi na poravnanje $t-2$ sekvence i 1 profilne sekvence. Zatim se vrši poravnanje profilne sekvence i preostalih $t-2$ sekvenci, bira sekvenca koja je dala poravnanje sa najvećim skorom i ažurira profilna sekvenca i tako redom.

Pohlepni algoritam za višestruko poravnanje pokazuje dobre rezultate za slične sekvence. Ako sekvence nisu slične, može se desiti da su dve sekvence na osnovu kojih je odabrano inicijalno poravnanje u velikoj meri razlikuju od ostalih i da se greška nagomilava sa dodavanjem novih sekvenci.

Literatura