Kako poredimo biološke sekvence?

Dinamičko programiranje i algoritmi podeli-pa-vladaj

Bioinformatics Algorithms:
an Active Learning Approach
~Poglavlje 5~

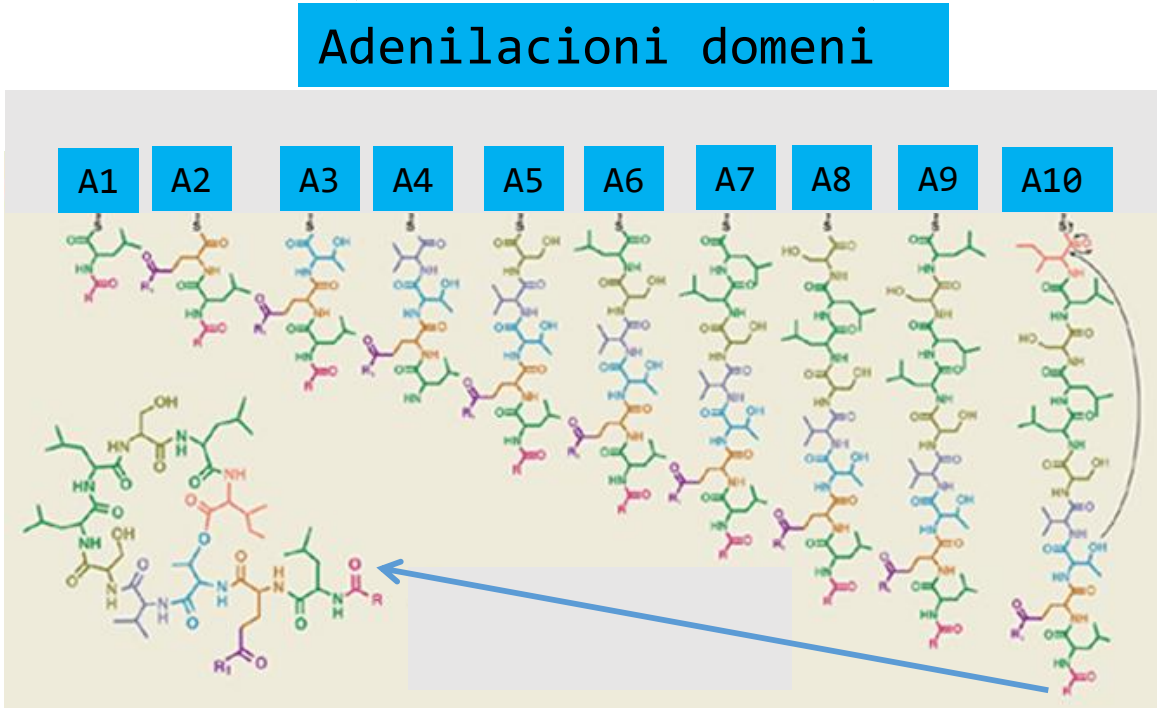
Pregled

- **Biološki uvid u poređenje sekvenci**
- Igra poravnanja i najduža zajednička podsekvenca
- Problem turiste na Menhetnu
- Problem kusura
- Dinamičko programiranje i putokazi za povratak
- Od Menhetna do grafa poravnanja
- Od globalnog do lokalnog poravnanja
- Kažnjavanje insercija i delecija u poravnanju sekvenci
- Prostorno efikasno poravnanje sekvenci
- Višestruko poravnanje sekvenci

Od genetskog do neribozomalnog koda

NRP sintetaza: mašina za sklapanje molekula

- Svaka NRP sintetaza se sastoji od modula koji određuju koju aminokiselinu treba dodati na peptid koji se sintetiše
- Svaki modul se sastoji od nekoliko različitih podjedinica (domena) od kojih su za sintezu najznačajniji adenilacioni domeni (skraćeno A-domeni)



NRP sintetaza dodaje jednu po jednu aminokiselinu

Ova tri A-domena ne izgledaju slično

YAFDLGYTCMFPVLLGGGELHIVQKETYTAPDEIAHYIKEHGITYIKLTPSLFHTIVNTASFAFDANFESLRLIVLGGEKIIPIDVIAFRKMYGHTEFINHYGPTEATIGA

AFDVSAGDFARALLTGGQLIVCPNEVKMDPASLYAIIKKYDITIFEATPALVIPLMEYIYEQKLDISQLQILIVGSDSCSMEDFKTLVSRFGSTIRIVNSYGVTEACIDS

IAFDASSWEIYAPLLNGGTVVCIDYYTTIDIKALEAVFKQHHIRGAMLPPALLKQCLVSAPTMISSLEILFAAGDRLSSQDAILARRAVGSGVYNAYGPTENTVLS

Ova tri A-domena ne izgledaju slično

YAFDLGYTCMFPVLLGGGELHIVQKETYTAPDEIAHYIKEHGITYIKLTPSLFHTIVNTASFAFDANFESLRLIVLGGEKIIPIDVIAFRKMYGHTEFINHYGPTEATIGA

AFDVSAGDFARALLTGGQLIVCPNEVKMDPASLYAIIKKYDITIFEATPALVIPLMEYIYEQKLDISQLQILIVGSDSCSMEDFKTLVSRFGSTIRIVNSYGVTEACIDS

IAFDASSWEIYAPLLNGGTVVCIDYYTTIDIKALEAVFKQHHIRGAMLPPALLKQCLVSAPTMISSLEILFAAGDRLSSQDAILARRAVGSGVYNAYGPTENTVLS

samo 3 konzervirane kolone

Da li sada izgledaju slično?

YAFDLGYTCMFPVLLGGGELHIVQKETYTAPDEIAHYIKEHGITYIKLTPSLFHTIVNTASFAFDANFESLRLIVLGGEKIIPIDVIAFRKMYGHTEFINHYGPTEATIGA

-AFDVSAGDFARALLTGGQLIVCPNEVKMDPASLYAIIKKYDITIFEATPALVIPLMEYIYEQKLDISQLQILIVGSDSCSMEDFKTLVSRFGSTIRIVNSYGVTEACIDS

IAFDASSWEIYAPLLNGGTVVCIDYYTTIDIKALEAVFKQHHIRGAMLPPALLKQCLVSAPTMISSLEILFAAGDRLSSQDAILARRAVGSGVYNAYGPTENTVLS

11 konzerviranih kolona

A sada?

YAFDLGYTCMFPVLLGGGELHIVQKETYTAPDEIAHYIKEHGITYIKLTPSLFHTIVNTASFAFDANFESLRLIVLGGEKIIPIDVIAFRKMYGHTE-FINHYGPTEATIGA

-AFDVSAGDFARALLTGGQLIVCPNEVKMDPASLYAIIKKYDITIFEATPALVIPLMEYI-YEQKLDISQLQILIVGSDSCSMEDFKTLVSRFGSTIRIVNSYGVTEACIDS

IAFDASSWEIYAPLLNGGTVVCIDYYTTIDIKALEAVFKQHHIRGAMLPPALLKQCLVSA----PTMISSLEILFAAGDRLSSQDAILARRAVGSGV-Y-NAYGPTENTVLS

19 konzerviranih kolona!

Crvene pozicije kodiraju konzervirano jezgro A-domena

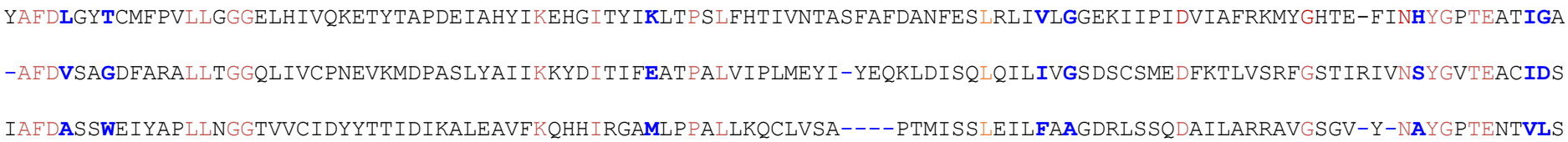
YAFDLGYTCMFPVLLGGGELHIVQKETYTAPDEIAHYIKEHGITYIKLTPSLFHTIVNTASFAFDANFESLRLIVLGGEKIIPIDVIAFRKMYGHTE-FINHYGPTEATIGA

-AFDVSAGDFARALLTGGQLIVCPNEVKMDPASLYAIIKKYDITIFEATPALVIPLMEYI-YEQKLDISQLQILIVGSDSCSMEDFKTLVSRFGSTIRIVNSYGVTEACIDS

IAFDASSWEIYAPLLNGGTVVCIDYYTTIDIKALEAVFKQHHIRGAMLPPALLKQCLVSA----PTMISSLEILFAAGDRLSSQDAILARRAVGSGV-Y-NAYGPTENTVLS

Koje pozicije kodiraju aminokiseline Asp, Orn, Val u ovim A-domenima?

Plave pozicije u A-domenima definišu neribozomalni kod



LTKVGHIG	**Asp**
VGEIGSID	**Orn**
AWMFAAVL	**Val**

Pregled

- Biološki uvid u poređenje sekvenci
- **Igra poravnanja i najduža zajednička podsekvenca**
- Problem turiste na Menhetnu
- Problem kusura
- Dinamičko programiranje i putokazi za povratak
- Od Menhetna do grafa poravnanja
- Od globalnog do lokalnog poravnanja
- Kažnjavanje insercija i delecija u poravnanju sekvenci
- Prostorno efikasno poravnanje sekvenci
- Višestruko poravnanje sekvenci

Igra poravnanja

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

Igra poravnanja (cilj je ukloniti sve simbole iz sekvenci tako da pritom sakupimo što više poena):

- Uklanjanje prvog simbola sleva iz obe sekvence
  - 1 poen ako se simboli poklapaju, 0 ako se ne poklapaju
- Uklanjanje prvog simbola sleva iz jedne sekvence
  - 0 points

Igra poravnanja

A T G T T A T A
A T C G T C C
+1

Igra poravnanja

A T G T T A T A
A T C G T C C
+1+1

Igra poravnanja

A T - G T T A T A
A T C G T C C
+1+1

Igra poravnanja

```
 A T - G T T A T A
 A T C G T C C
+1+1  +1
```

Igra poravnanja

```
A T - G T T A T A
A T C G T C C
+1+1  +1+1
```

Igra poravnanja

```
A T - G T T A T A
A T C G T - C C
+1+1  +1+1
```

Igra poravnanja

```
A T - G T T A T A
A T C G T - C C
+1+1  +1+1
```

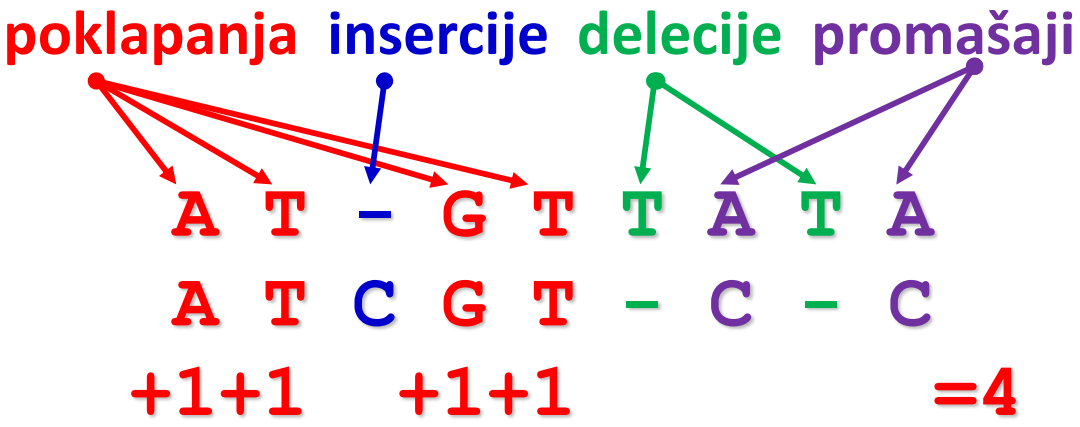
Igra poravnanja

```
A T - G T T A T A
A T C G T - C - C
+1+1  +1+1
```

Igra poravnanja

```
A T - G T T A T A
A T C G T - C - C
+1+1  +1+1       =4
```

Šta je poravnanje sekvenci?



Poravnanje dve sekvence predstavlja matricu koja ima dva reda:

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

Najduža zajednička podsekvenca

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

Poklapanja (matches) u poravnanju dve sekvence (**ATGT**) formiraju njihovu **zajedničku podsekvencu**

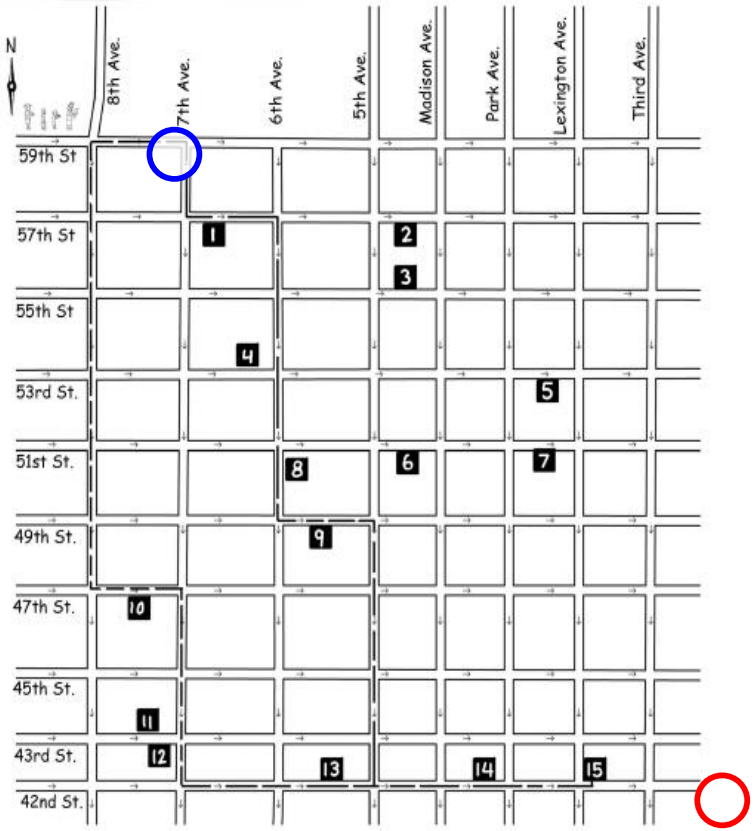
Problem najduže zajedničke podsekvence: Naći najdužu zajedničku podsekvencu dve niske.

- **Ulaz:** Dve niske.
- **Izlaz:** Najduža zajednička podsekvenca ovih niski.

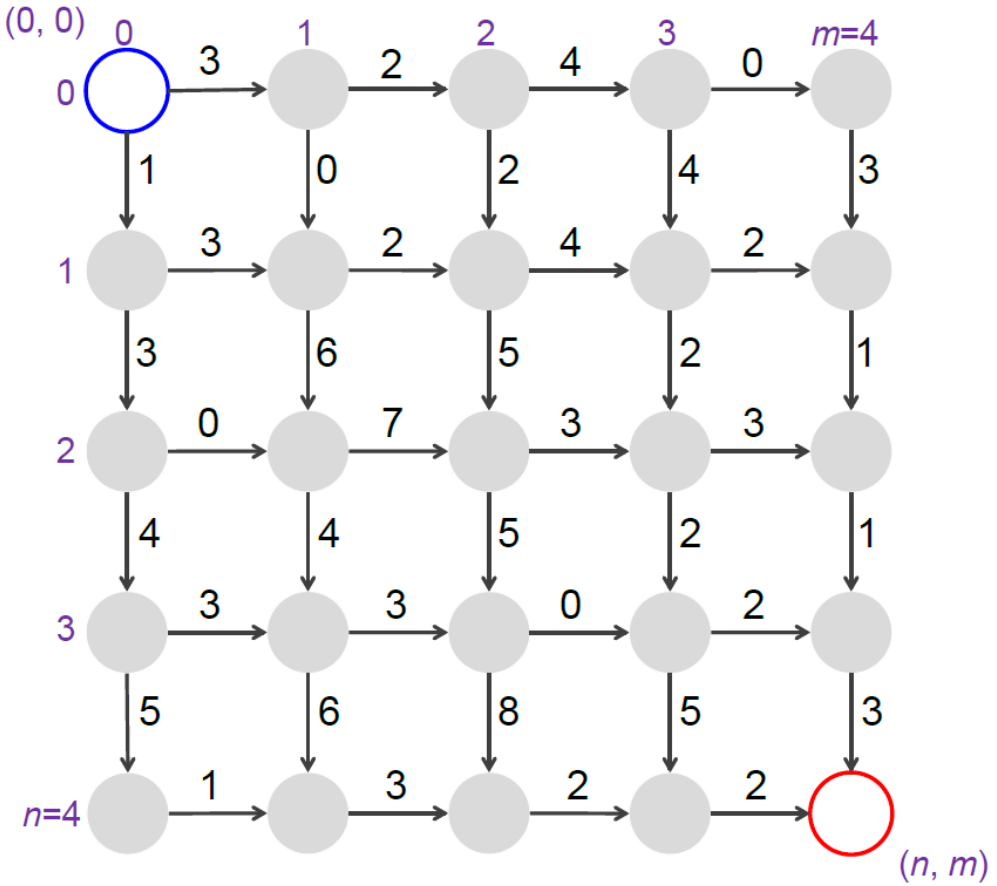
Pregled

- Biološki uvid u poređenje sekvenci
- Igra poravnanja i najduža zajednička podniska
- **Problem turiste na Menhetnu**
- Problem kusura
- Dinamičko programiranje i putokazi za povratak
- Od Menhetna do grafa poravnanja
- Od globalnog do lokalnog poravnanja
- Kažnjavanje insercija i delecija u poravnanju sekvenci
- Prostorno efikasno poravnanje sekvenci
- Višestruko poravnanje sekvenci

Od plana Menhetna do mrežnog grafa



1 Carnegie Hall
2 Tiffany & Co.
3 Sony Building
4 Museum of Modern Art
5 Four Seasons
6 St. Patrick's Cathedral
7 General Electric Building
8 Radio City Music Hall
9 The Today Show
10 Paramount Building
11 NY Times Building
12 Times Square
13 General Society of Mechanics and Tradesmen (a must see!)
14 Grand Central Terminal
15 Chrysler Building



Problem turiste na Menhetnu

Problem turiste na Menhetnu: Naći najdužu putanju u pravougaonoj mreži gradskih ulica.

- **Ulaz:** Usmeren težinski mrežni graf.
- **Izlaz:** Najduža putanja od početnog (*source*) do krajnjeg čvora (*sink*) u mrežnom grafu.

Problem turiste na Menhetnu

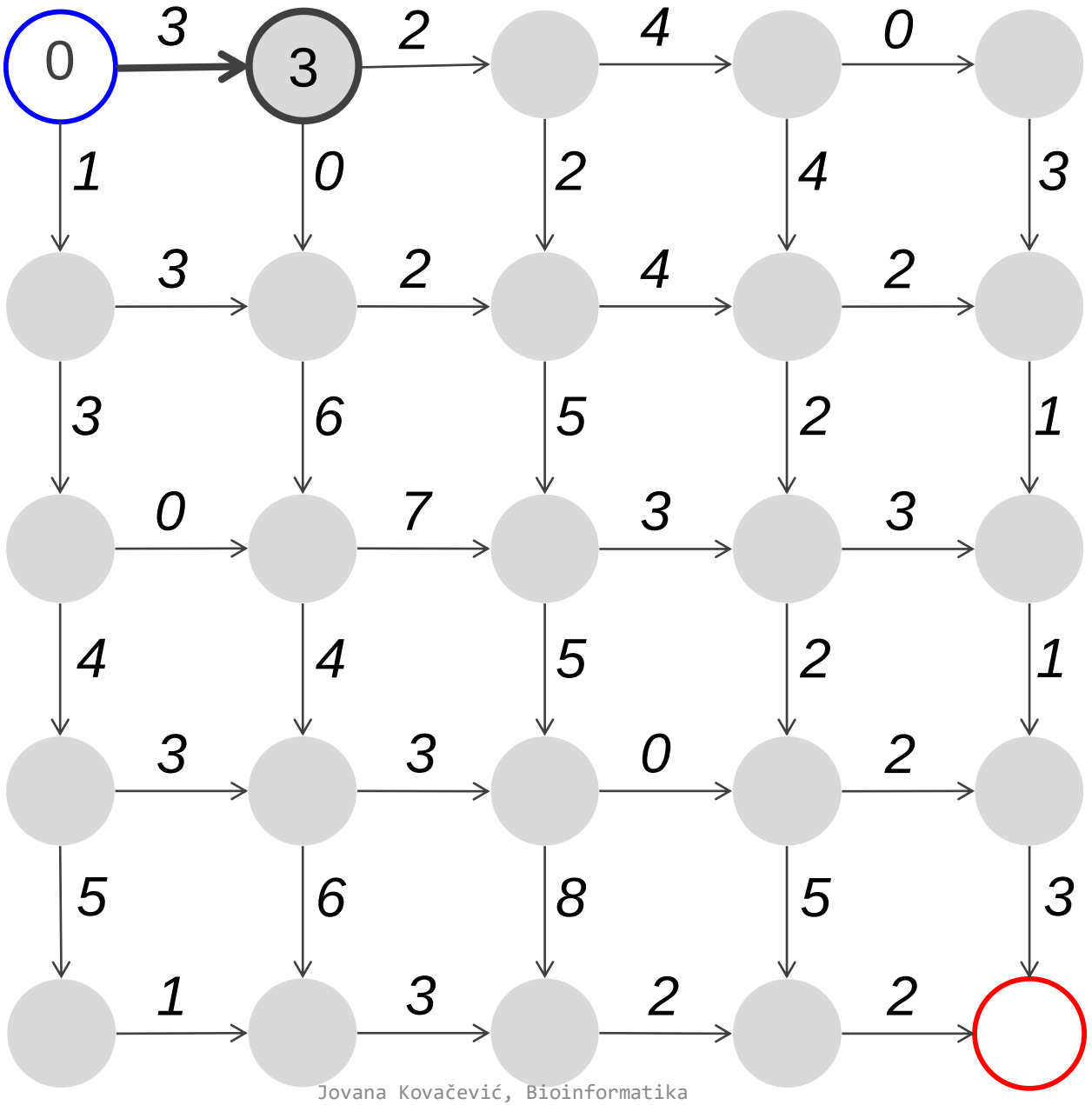
Problem turiste na Menhetnu: Naći najdužu putanju u pravougaonoj mreži gradskih ulica.

- **Ulaz:** Usmeren težinski mrežni graf.
- **Izlaz:** Najduža putanja od početnog (*source*) do krajnjeg čvora (*sink*) u mrežnom grafu.

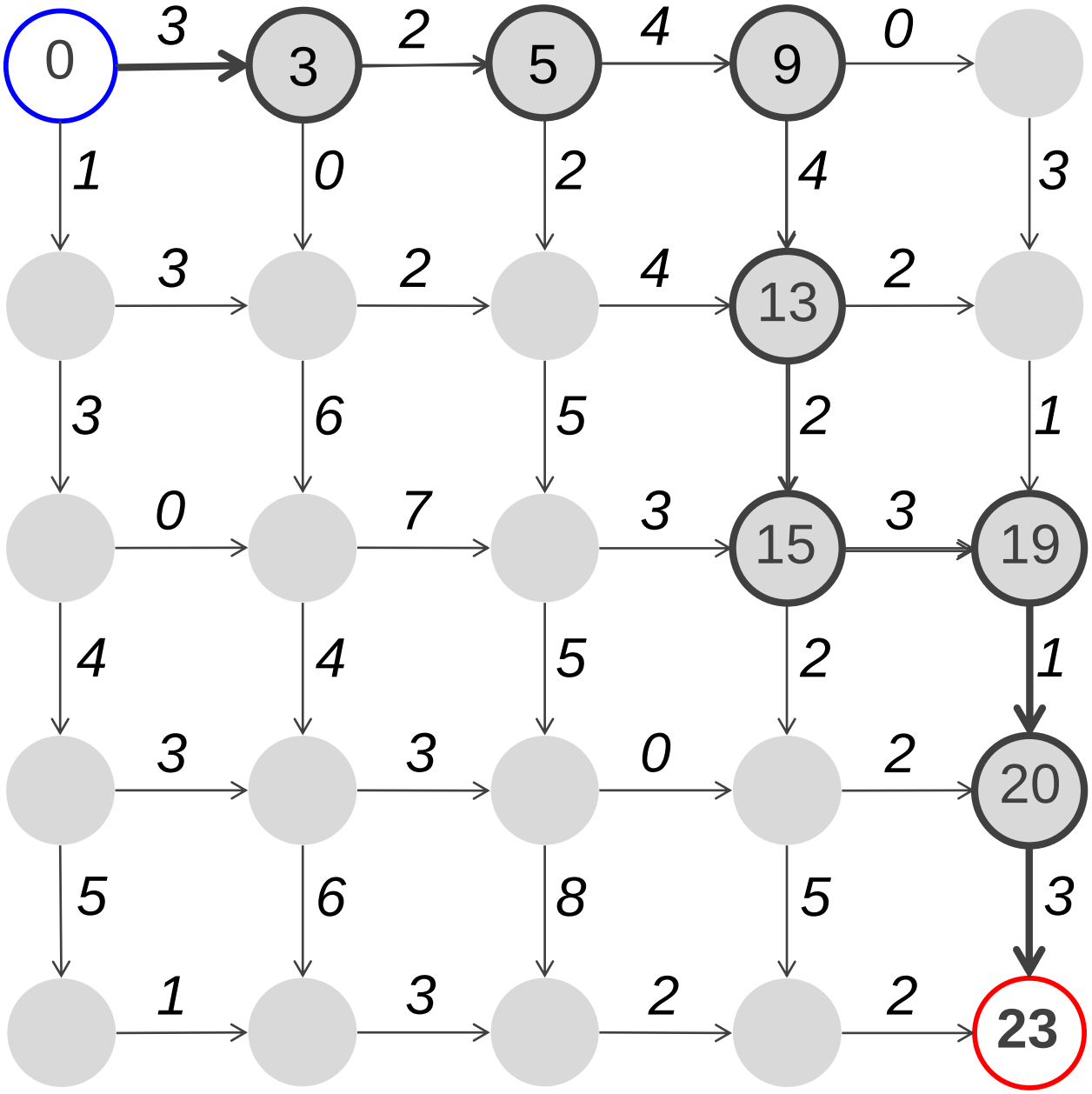
Mogući pristupi za rešavanje:

- Gruba sila
  - ogroman broj svih mogućih putanja
- Pohlepni pristup
  - moguće je propustiti najdužu putanju

Pohlepni algoritam?

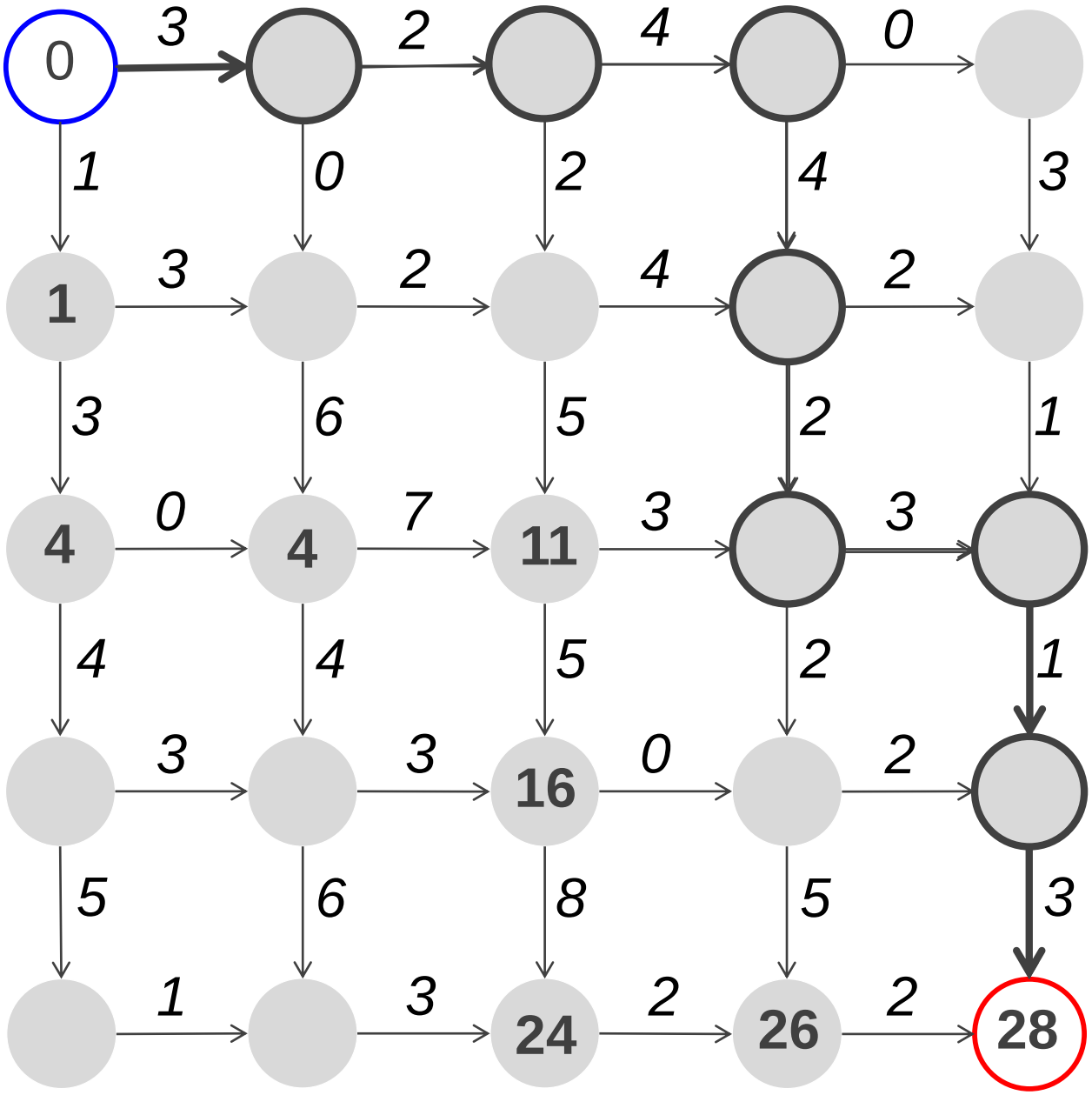


Pohlepni algoritam?

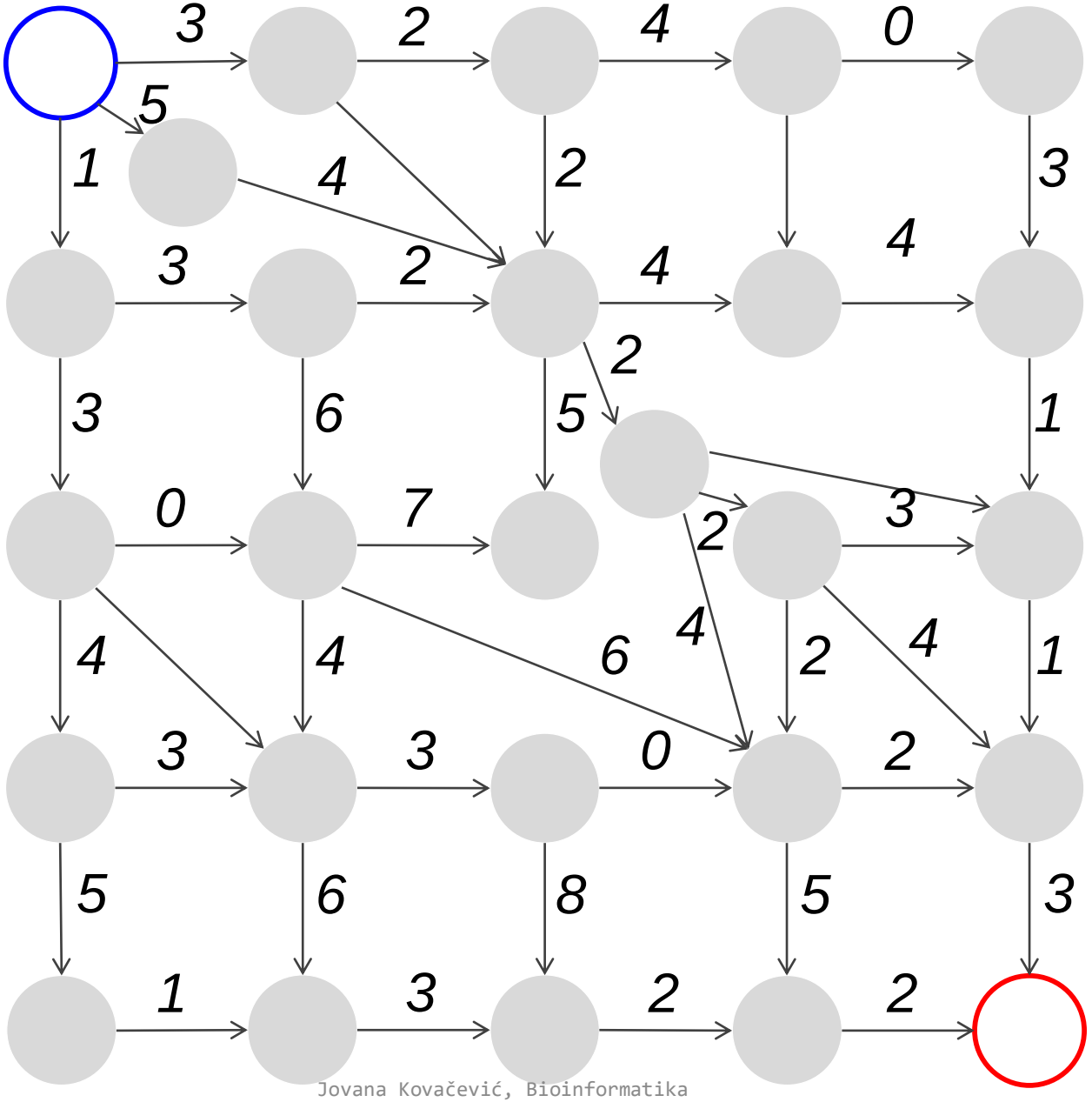


Pohlepni algoritam?

Nije pronašao najdužu putanju



Uopštenje: graf ne mora imati topologiju pravougaone mreže



Problem Turiste sa Menhetna je specijalni slučaj problema najduže putanje u usmerenom grafu

Problem najduže putanje u usmerenom grafu: Naći najdužu putanju između dva čvora u težinskom usmerenom grafu.

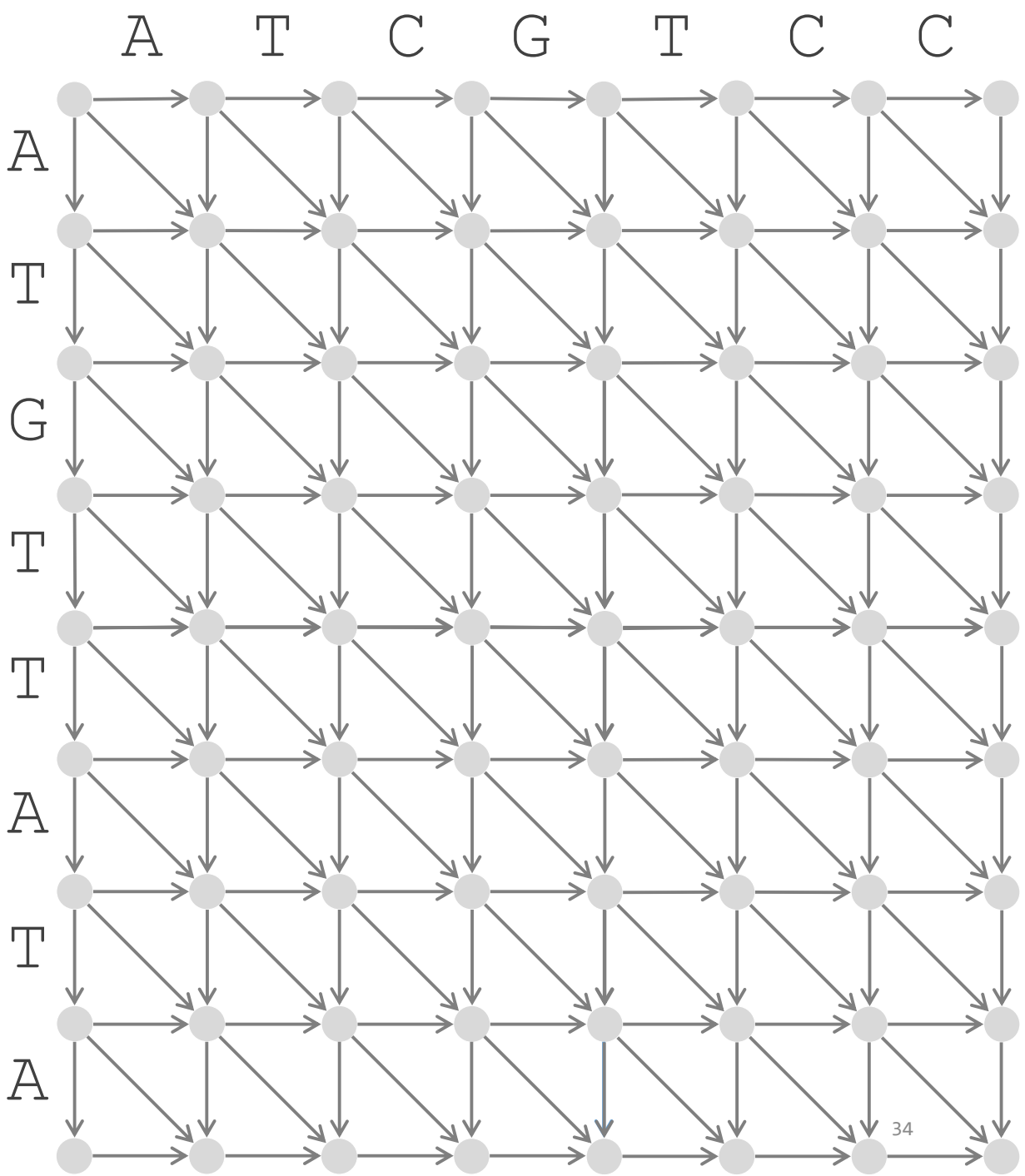
- **Ulaz:** Usmereni težinski graf sa označenim čvorovima *source* i *sink*.
- **Izlaz:** Najduža putanja od čvora *source* do čvora *sink* u usmerenom težinskom grafu.

Veza između problema turiste sa Menhetna i igre poravnanja

A T - G T T A T A
A T C G T - C - C
↘ ↘ → ↘ ↘ ↓ ↘ ↓ ↘

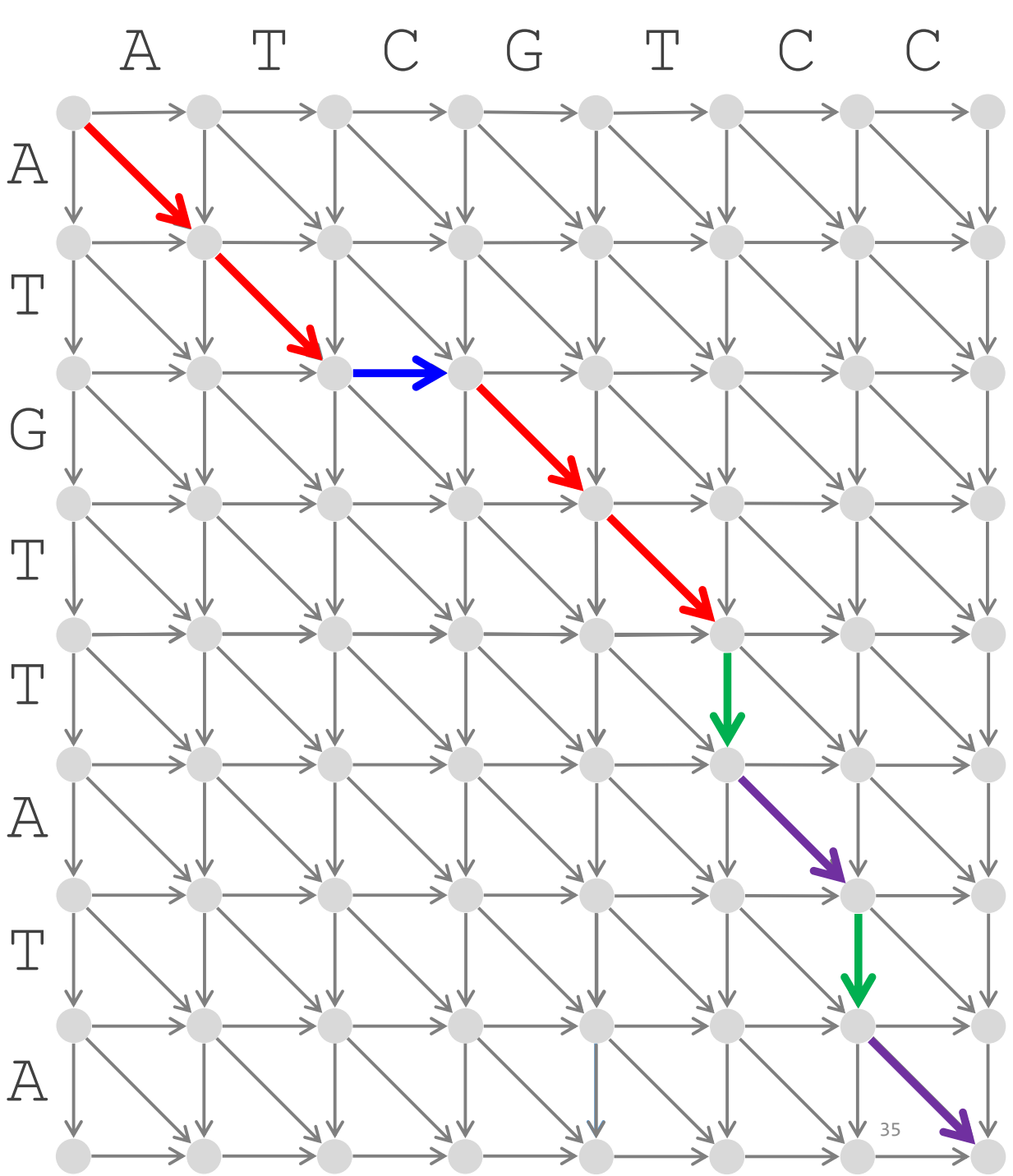
poravnanje → putanja

A T - G T T A T A
A T C G T - C - C
↘ ↘ → ↘ ↘ ↓ ↘ ↓ ↘



poravnanje → putanja

```
A T - G T T A T A
A T C G T - C - C
↘ ↘ → ↘ ↘ ↓ ↘ ↓ ↘
```



?

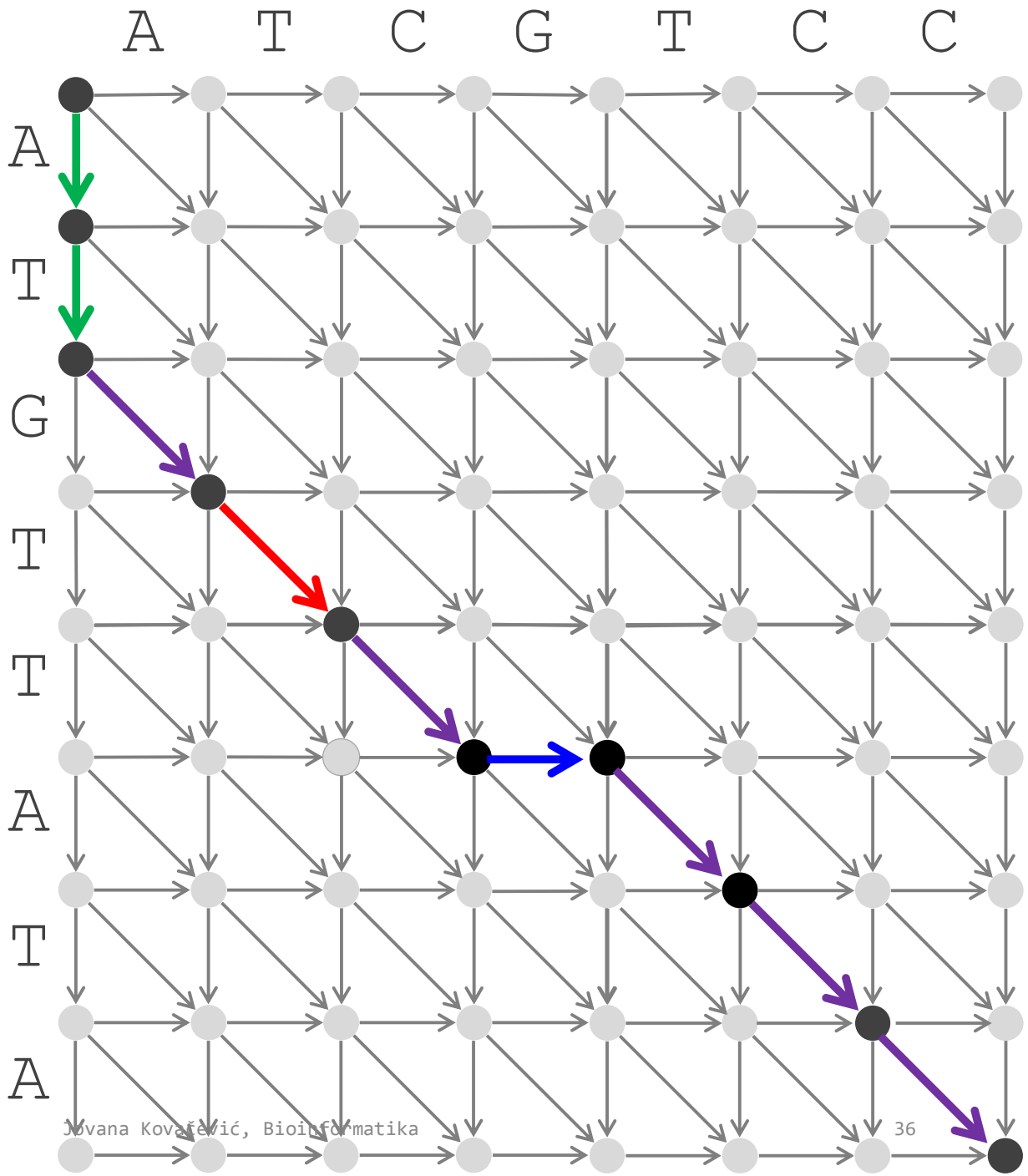
putanja → poravnanje

A T G T T - A T A

- - A T C G T C C

↓ ↓ ↘ ↘ ↘ → ↘ ↘ ↘

poravnanje
najvišeg skora
=
najduža putanja u
mrežnom grafu

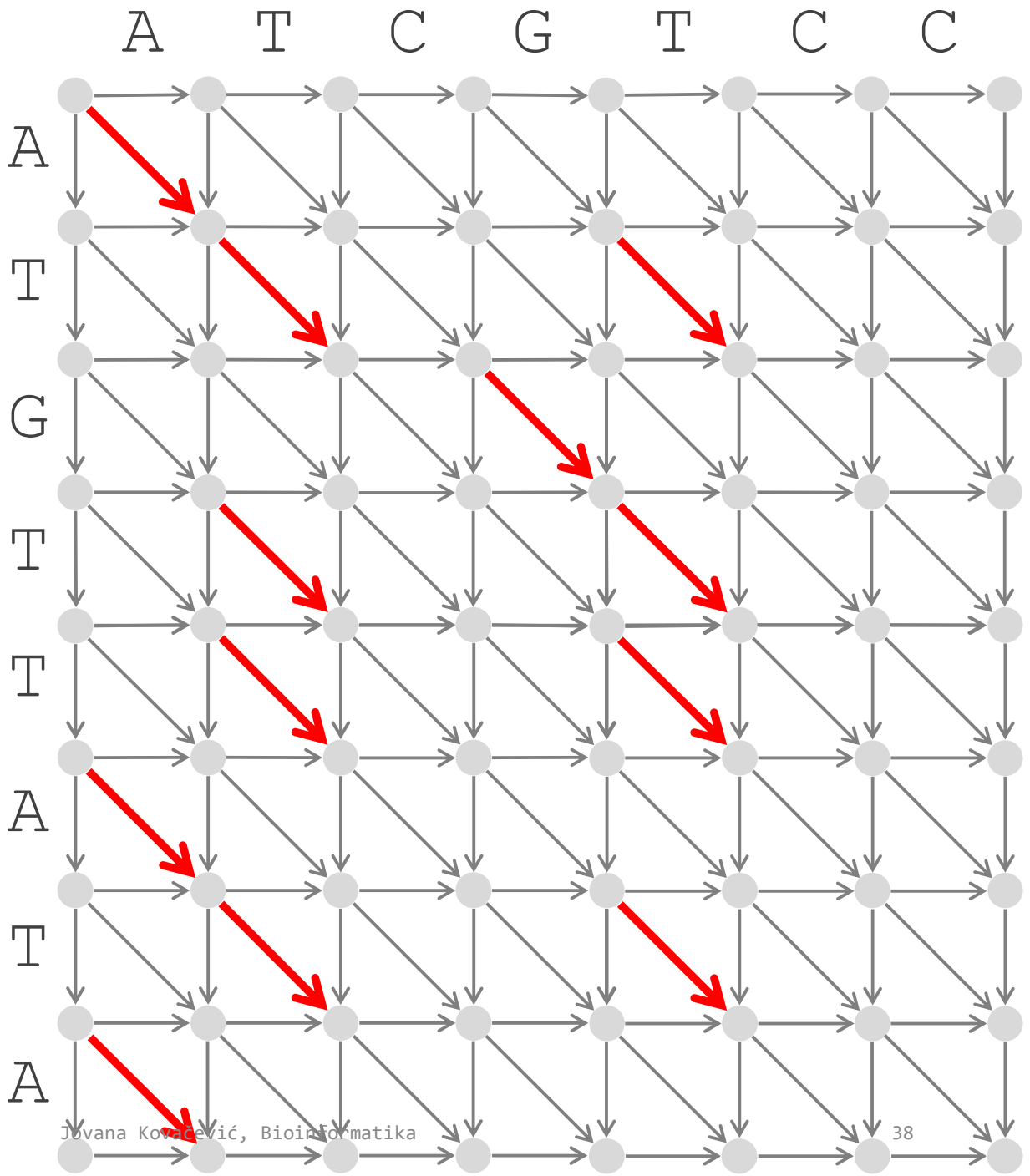


Kako izgraditi Menhetn graf za igru poravnanja i za problem najduže podsekvence?

- Vrste označimo aminokiselinama iz prve niske
- Kolone označimo aminokiselinama iz druge niske
- U svaku presečnu tačku postavimo jedan čvor
- Gde god je moguće, postaviti vertikalne (insercija), horizontalne (delecija) i dijagonalne grane (*match* ili *mismatch*)
- Dijagonalne grane otežati koeficijentom 1, ostale koeficijentom 0
- Problem najduže zajedničke podsekvence se svodi na problem nalaženja najduže putanje između dva data čvora u usmerenom grafu

Kako izgraditi Menhetn graf za igru poravnanja i za problem najduže podsekvence?

Dijagonalne crvene grane odgovaraju poklapanju simbola i imaju skor 1



- Prikazaćemo neka rešenja problema nalaženja najduže zajedničke podniske tehnikama dinamičkog programiranja

- Da bismo približili ove tehnike, najpre predstavljamo problem vraćanja kusura

Pregled

- Biološki uvid u poređenje sekvenci
- Igra poravnanja i najduža zajednička podsekvenca
- Problem turiste na Menhetnu
- **Problem kusura**
- Dinamičko programiranje i putokazi za povratak
- Od Menhetna do grafa poravnanja
- Od globalnog do lokalnog poravnanja
- Kažnjavanje insercija i delecija u poravnanju sekvenci
- Prostorno efikasno poravnanje sekvenci
- Višestruko poravnanje sekvenci

Problem vraćanja kusura

Problem vraćanja kusura: Naći minimalan broj novčića neophodnih za vraćanje kusura.

- **Ulaz:** Ceo broj *money* i niz pozitivnih celih brojeva ($coin_1$, . . . , $coin_d$).
- **Izlaz:** Minimalni broj novčića iz datog niza koji rasitnjava sumu *money*.

Pohlepno vraćanje kusura

```
GreedyChange(money)
    change ← empty collection of coins
    while money > 0
        coin ← largest denomination that does
   not exceed money
        add coin to change
        money ← money – coin
    return change
```

Vraćanje kusura u Tanzaniji

40 cents = 25+10+5

Pohlepno

Vraćanje kusura u Tanzaniji: *GreedyChange* ne daje optimalno rešenje

40 cents = 25+10+5 = 20+20

Pohlepno nije **Optimalno**

Rekurzivno vraćanje kusura

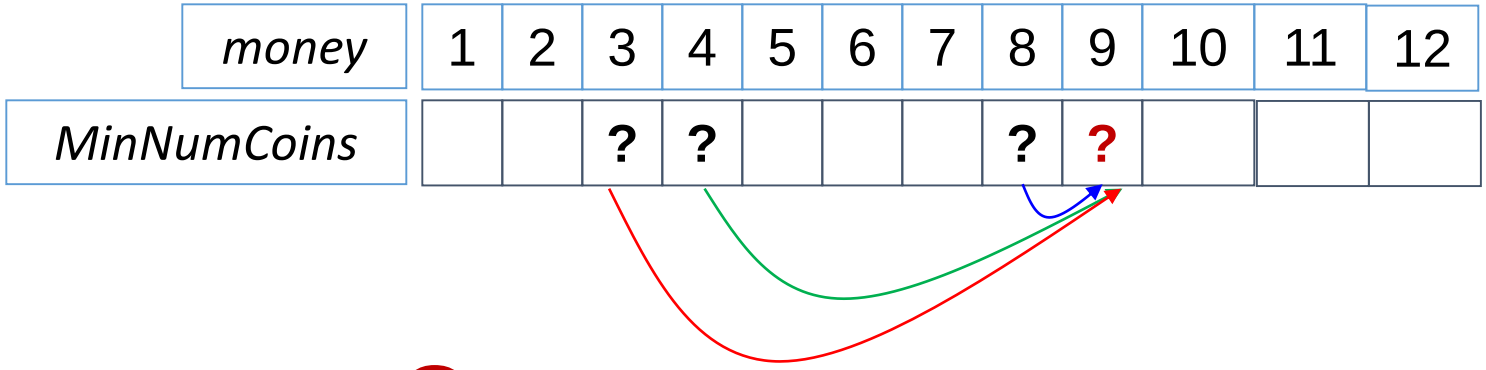
Za date apoene 6, 5 i 1, koji je najmanji broj novčića neophodnih za vraćanje kusura od 9 centi?

money	1	2	3	4	5	6	7	8	9	10	11	12
MinNumCoins									?			

MinNumCoins(9)= **?**

Rekurzivno vraćanje kusura

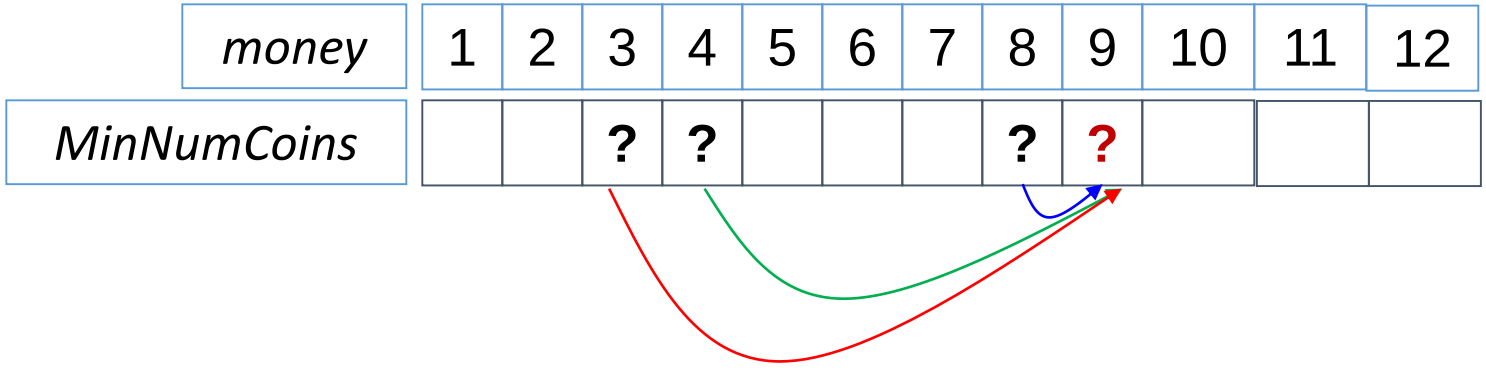
Za date apoene 6, 5 i 1, , koji je najmanji broj novčića neophodnih za vraćanje kusura od 9 centi?



MinNumCoins(9)= **?**

Rekurzivno vraćanje kusura

Za date apoene 6, 5 i 1, , koji je najmanji broj novčića neophodnih za vraćanje kusura od 9 centi?



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

Rekurzivno vraćanje kusura

Za date apoene 6, 5 i 1, koji je najmanji broj novčića neophodnih za vraćanje kusura od 9 centi?

money	1	2	3	4	5	6	7	8	9	10	11	12
MinNumCoins			?	?				?	?			

MinNumCoins(3)=
MinNumCoins(4)=
MinNumCoins(8)=

?

Rekurzivno vraćanje kusura

Za date apoene 6, 5 i 1, koji je najmanji broj novčića neophodnih za vraćanje kusura od 9 centi?

money	1	2	3	4	5	6	7	8	9	10	11	12
MinNumCoins		?	?	?			?	?	?			

MinNumCoins(3)=
MinNumCoins(4)=
MinNumCoins(8)=

?

Rekurzivno vraćanje kusura

Za date apoene 6, 5 i 1, koji je najmanji broj novčića neophodnih za vraćanje kusura od 9 centi?

money	1	2	3	4	5	6	7	8	9	10	11	12
MinNumCoins		?	?	?			?	?	?			

$$MinNumCoins(money) = \min \begin{cases} MinNumCoins(money-6) + 1 \\ MinNumCoins(money-5) + 1 \\ MinNumCoins(money-1) + 1 \end{cases}$$

Rekurzivno vraćanje kusura

Za date apoene 6, 5 i 1, koji je najmanji broj novčića neophodnih za vraćanje kusura od 9 centi?

money	1	2	3	4	5	6	7	8	9	10	11	12
MinNumCoins		?	?	?			?	?	?			

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

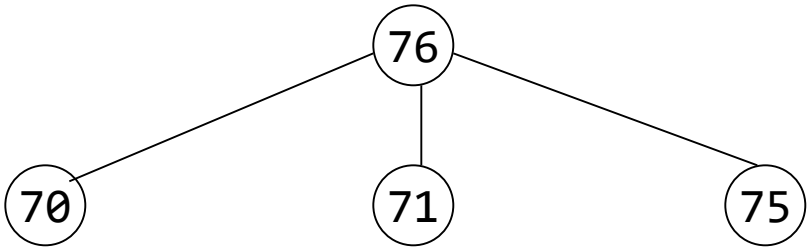
RecursiveChange

```
RecursiveChange(money, coins)
    if money = 0
        return 0
    MinNumCoins ← infinity
    for i ← 1 to |coins|
        if money ≥ coin_i
            NumCoins ← RecursiveChange(money-
coin_i, coins)
            if NumCoins + 1 < MinNumCoins
                MinNumCoins ← NumCoins + 1
    return MinNumCoins
```

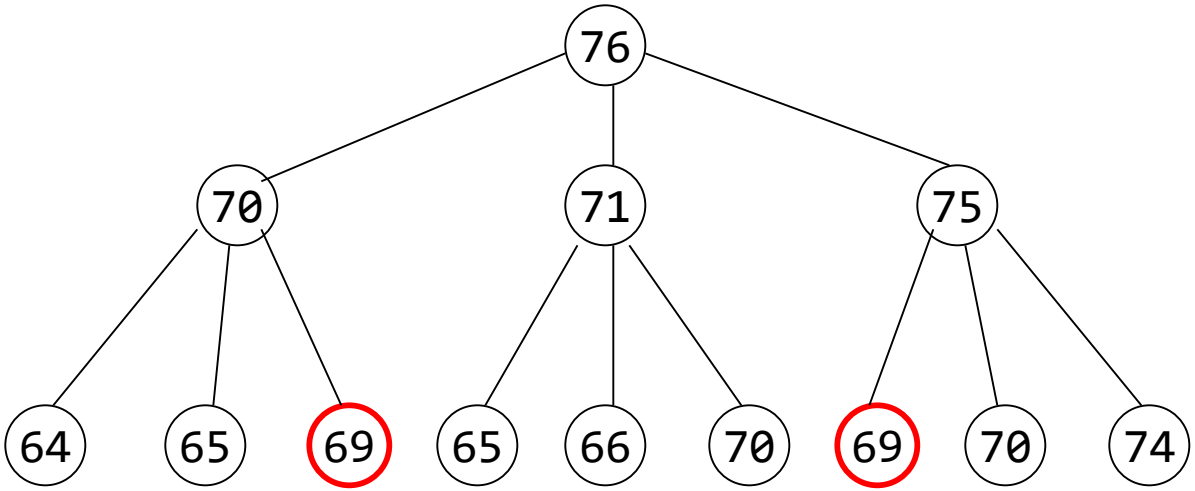
Koliko je brz *RecursiveChange*?

76

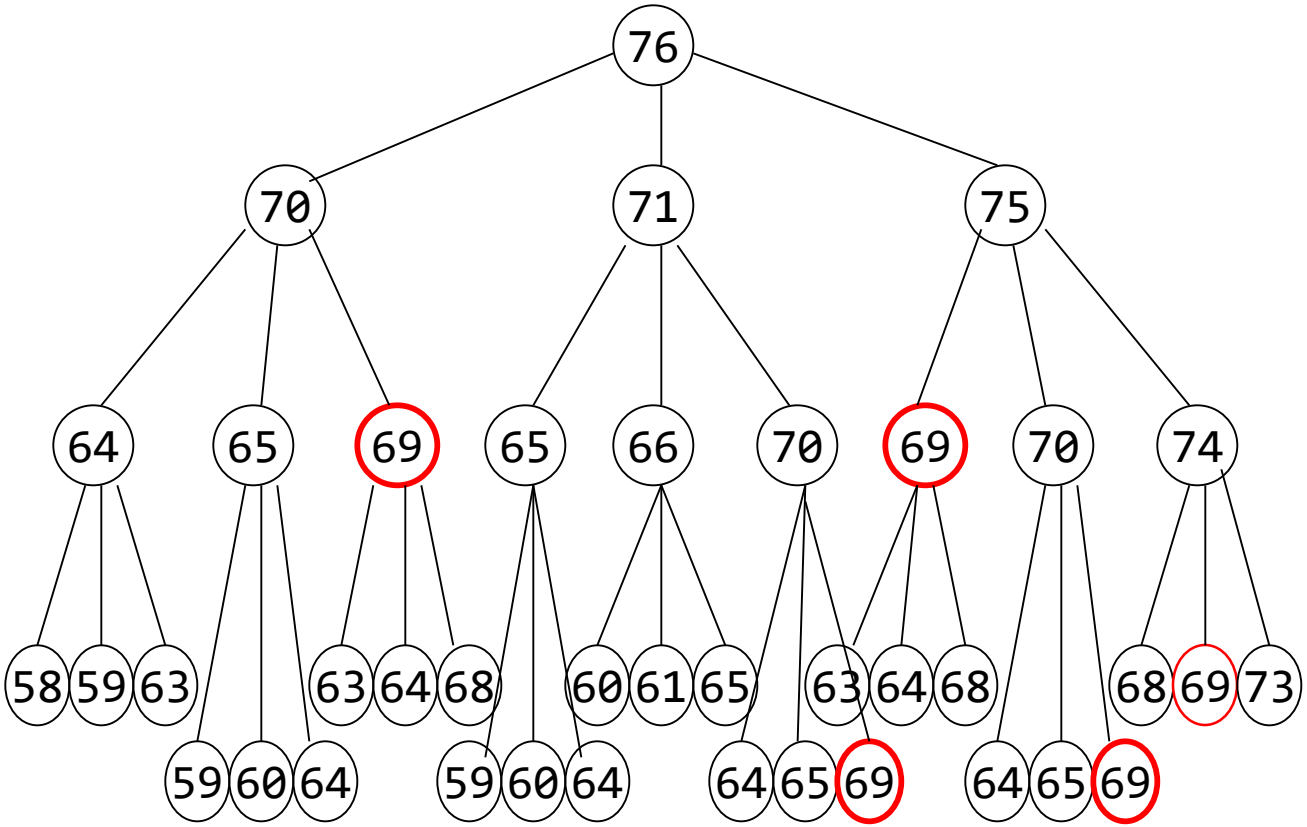
Rekurzivno stablo



Rekurzivno stablo



Rekurzivno stablo



Optimalna kombinacijal novčića za 69 centi se izračunava **6** puta!

Optimalna kombinacijal novčića za 30 centi se izračunava **milijardama** puta!

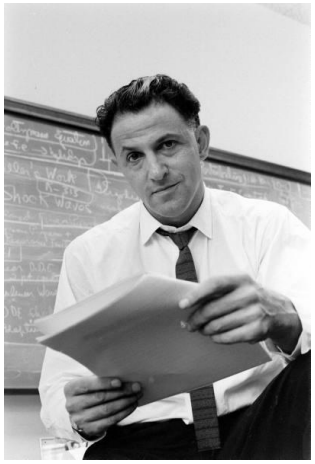
Vraćanje kusura dinamičkim programiranjem

Nagoveštaj. Da li možemo izračunati sve vrednosti

$$MinNumCoins(money - coin_i)$$

unapred, pre računanja

$$MinNumCoins(money)?$$



Richard Bellman

Umesto vremenski zahtevnih poziva:

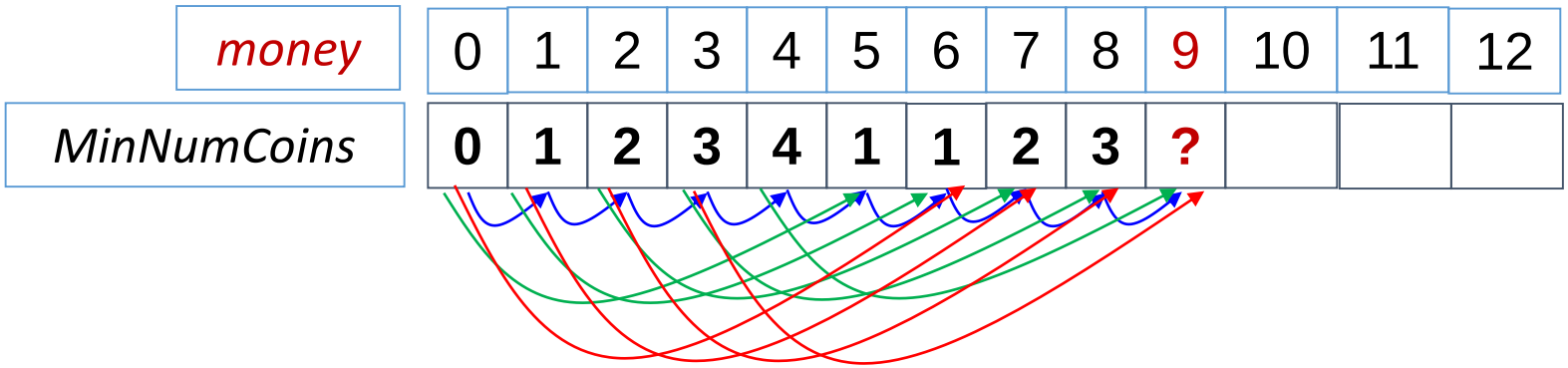
RecursiveChange$(money\text{-}coin_i,$ ***Coins***$)$

Jednostavno bismo potražili vrednost iz unapred izračunate tabele

$$MinNumCoins(money - coin_i)$$

Vraćanje kusura dinamičkim programiranjem

Za date apoene 6, 5 i 1, , koji je najmanji broj novčića neophodnih za vraćanje kusura od 9 centi?



DPChange

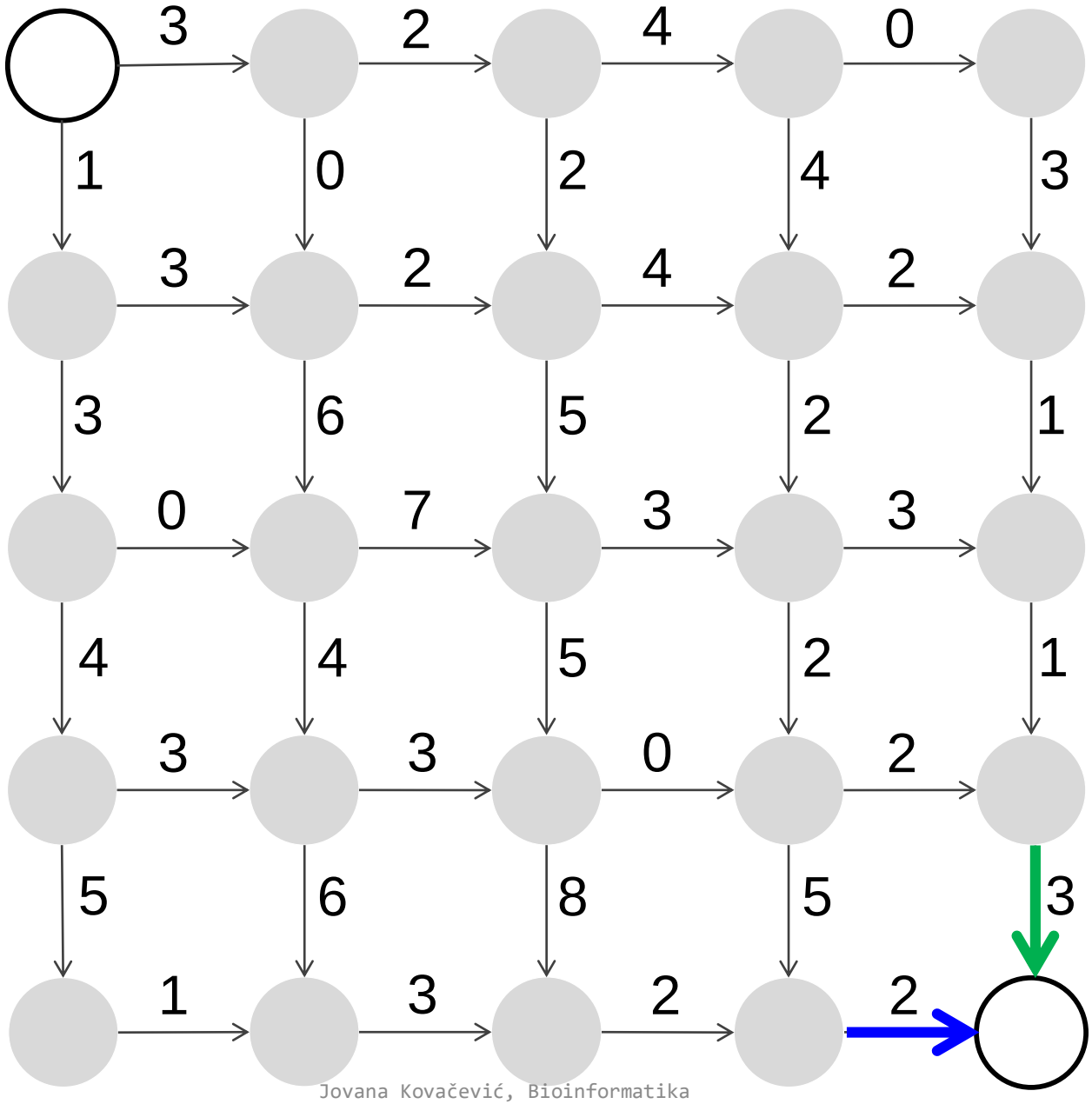
```
DPChange(money, coins)
 MinNumCoins(0) ← 0
 for m ← 1 to money
   MinNumCoins(m) ← infinity
   for i  ← 1 to |coins|
     if m ≥ coin_i
      if MinNumCoins(m – coin_i) + 1 < MinNumCoins(m)
         MinNumCoins(m) ← MinNumCoins(m – coin_i)+ 1
   return MinNumCoins(money)
```

Pregled

- Biološki uvid u poređenje sekvenci
- Igra poravnanja i najduža zajednička podniska
- Problem turiste na Menhetnu
- Problem kusura
- **Dinamičko programiranje i putokazi za povratak**
- Od Menhetna do grafa poravnanja
- Od globalnog do lokalnog poravnanja
- Kaznjavanje insercija i delecija u poravnanju sekvenci
- Prostorno efikasno poravnanje sekvenci
- Višestruko poravnanje sekvenci

- Primenimo rekurzivni pristup i pristup dinamičkim programiranjem za rešavanje problema turiste sa Menhetna

Posmatramo **jednostavniji, Menhetn graf**: pretpostavimo da do čvora *sink* možemo doći samo na dva načina: kretanjem **južno ↓** ili kretanjem **istočno →**



južno ili istočno ?

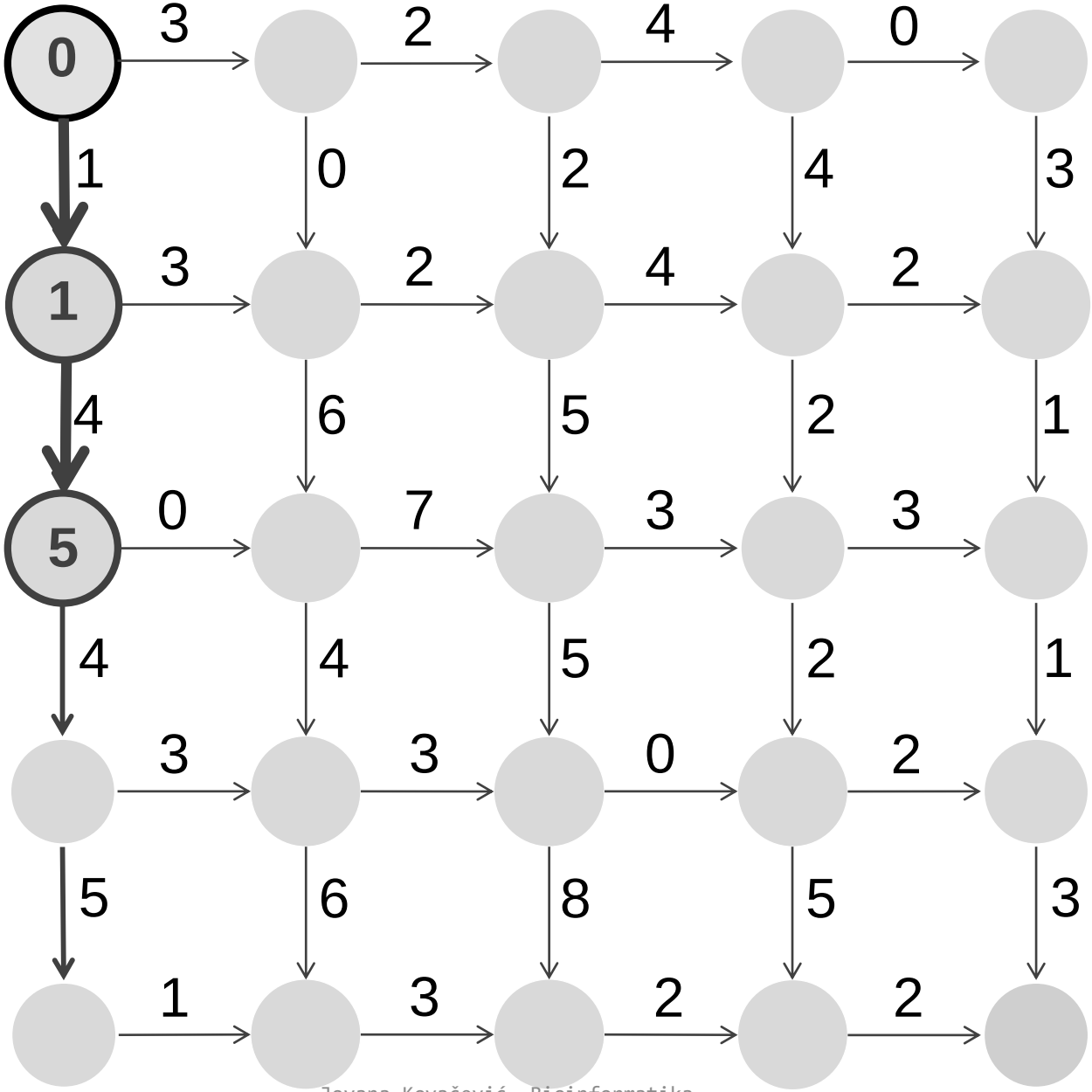
Rekurzivni pristup

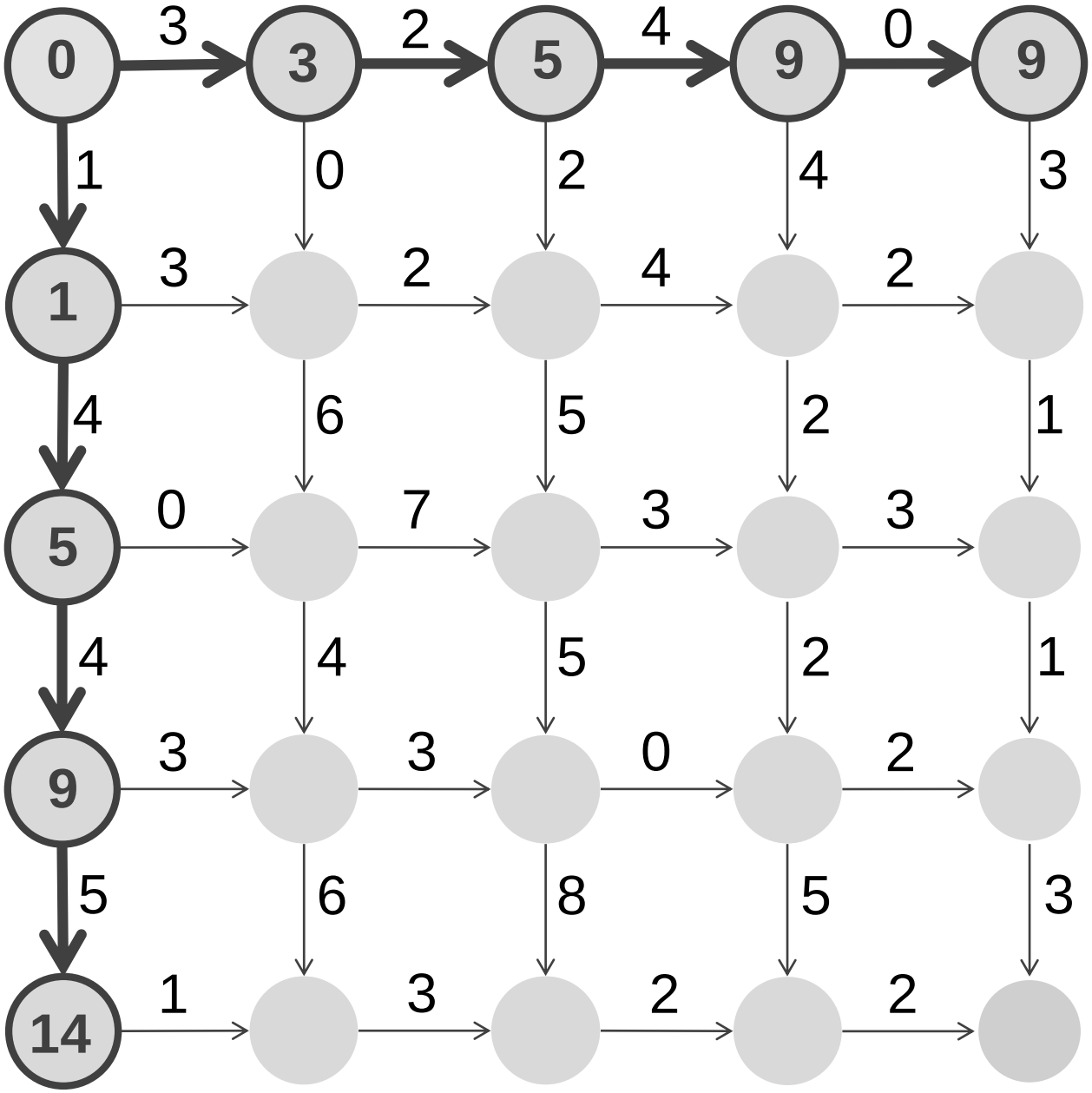
```
SouthOrEast(n,m)
    if n=0  and m=0
        return  0
    x ← -infinity, y ← -infinity,
    if n>0
        x ← SouthOrEast(n-1,m)+weight of edge "↓"into (n,m)
    if m>0
        y ← SouthOrEast(n,m-1)+ weight of edge "→"into (n,m)
    return max{x,y}
```

- SouthOrEast pozivamo za svaki čvor u grafu (*m*x*n*) puta i pritom se kao kod *RecursiveChange* dešava da jedan isti čvor računamo više puta
- Rekurzivni pristup je, kao i kod problema vraćanja kusura, suviše spor. Prelazimo na dinamičko programiranje

Pristup dinamičkim programiranjem

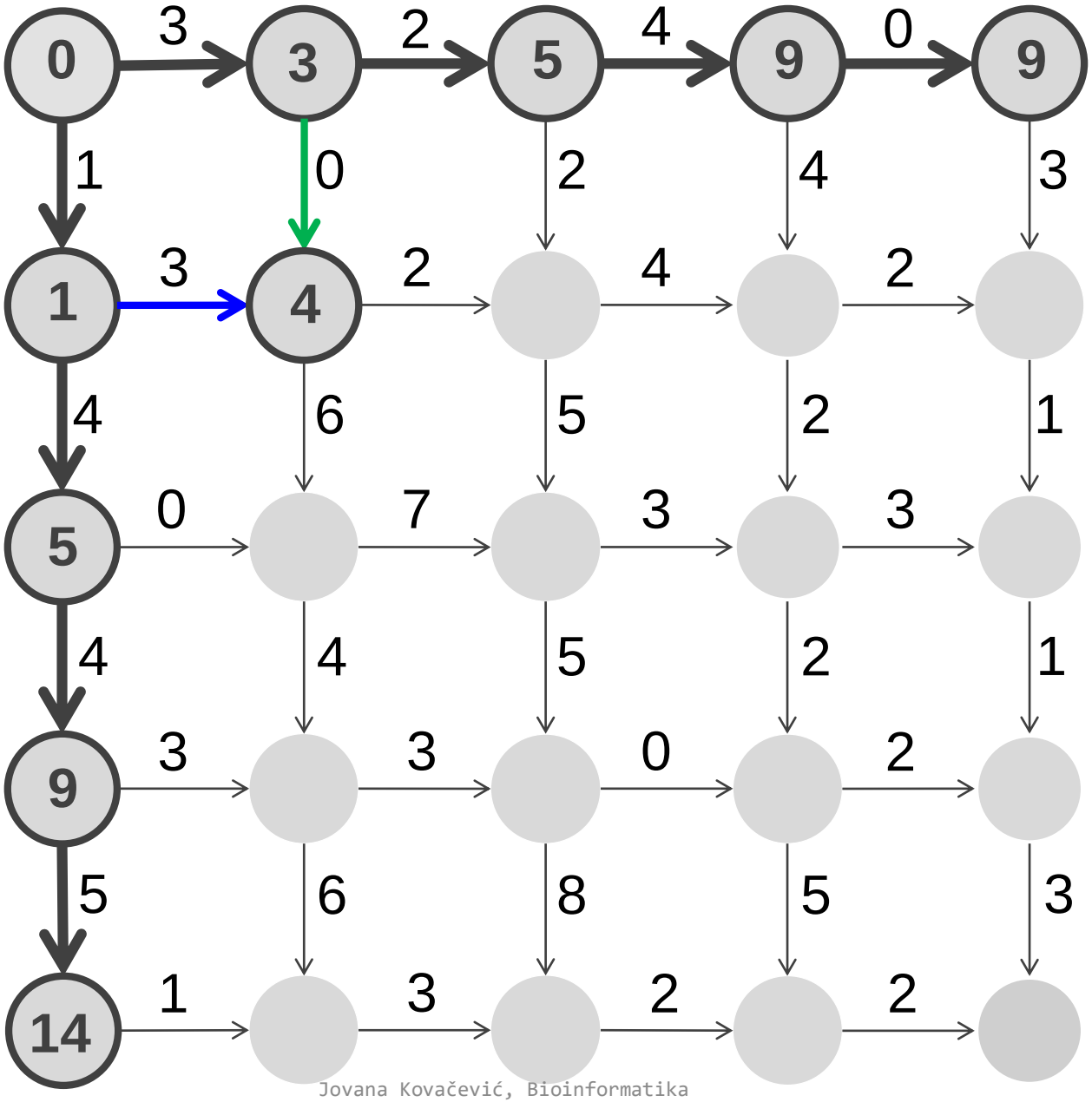
- Kao što smo kod *Change* problema umesto od krajnje vrednosti kusura krenuli od najmanje vrednosti apoena, tako i ovde umesto od krajnjeg cvora (*sink*) krećemo od početnog (*source*).
- U čvor (i,j) upisujemo dužinu maksimalne putanje od (0,0) do (i,j)
- prvo izračunamo za čvorove na obodu grafa a zatim, kolonu po kolonu, za preostale čvorove


0
3
2
4
0
1
0
2
4
3
1
3
2
4
2
4
6
5
2
1
5
0
7
3
3
4
4
5
2
1
3
3
0
2
5
6
8
5
3
1
3
2
2

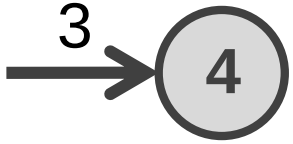

0
3
2
5
4
9
0
9
1
0
2
4
3
1
3
2
4
2
4
6
5
2
1
5
0
7
3
3
4
4
5
2
1
9
3
3
0
2
5
6
8
5
3
14
1
3
2
2

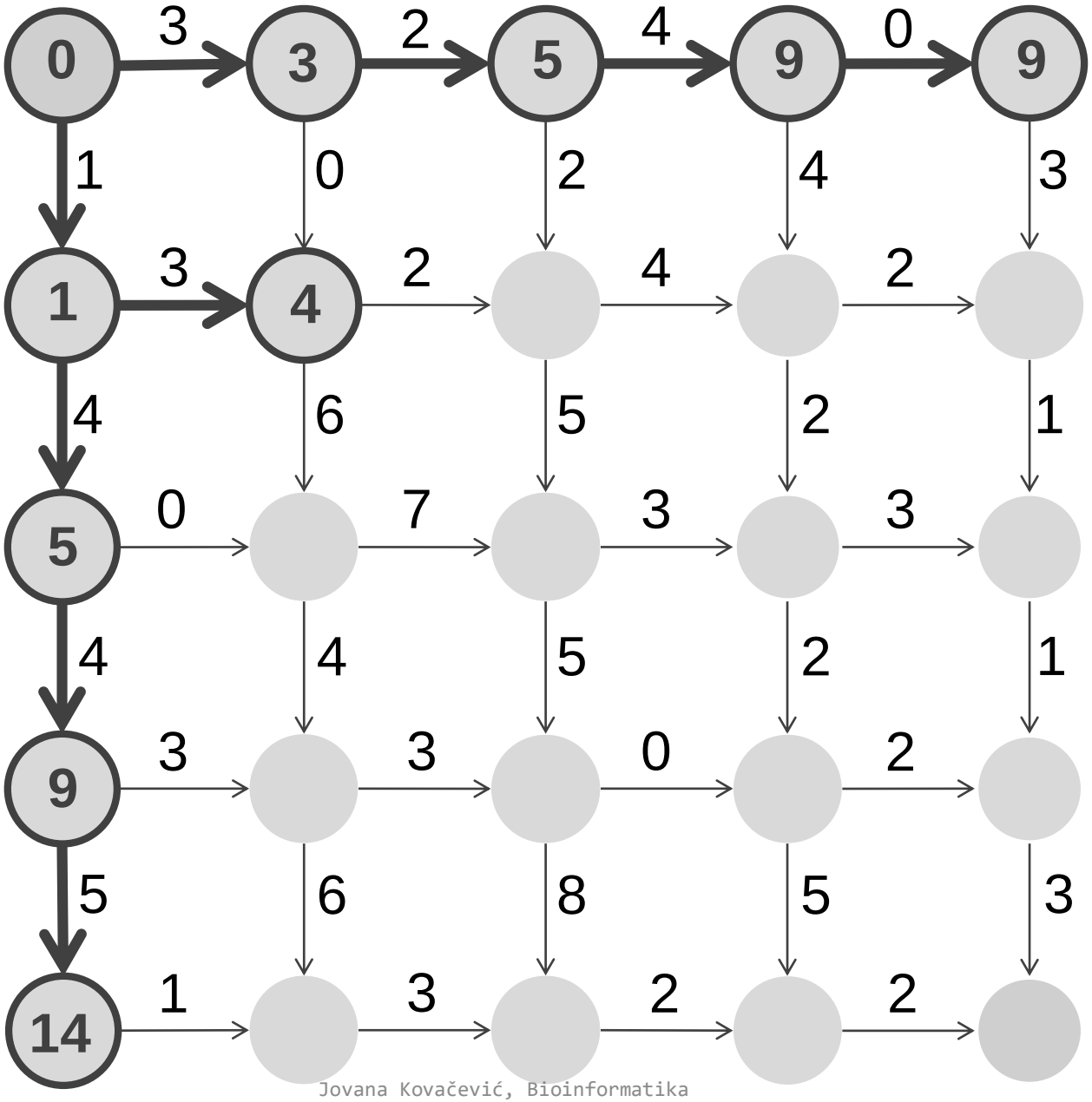
južno
ili
istočno?

1+3 >
3+0



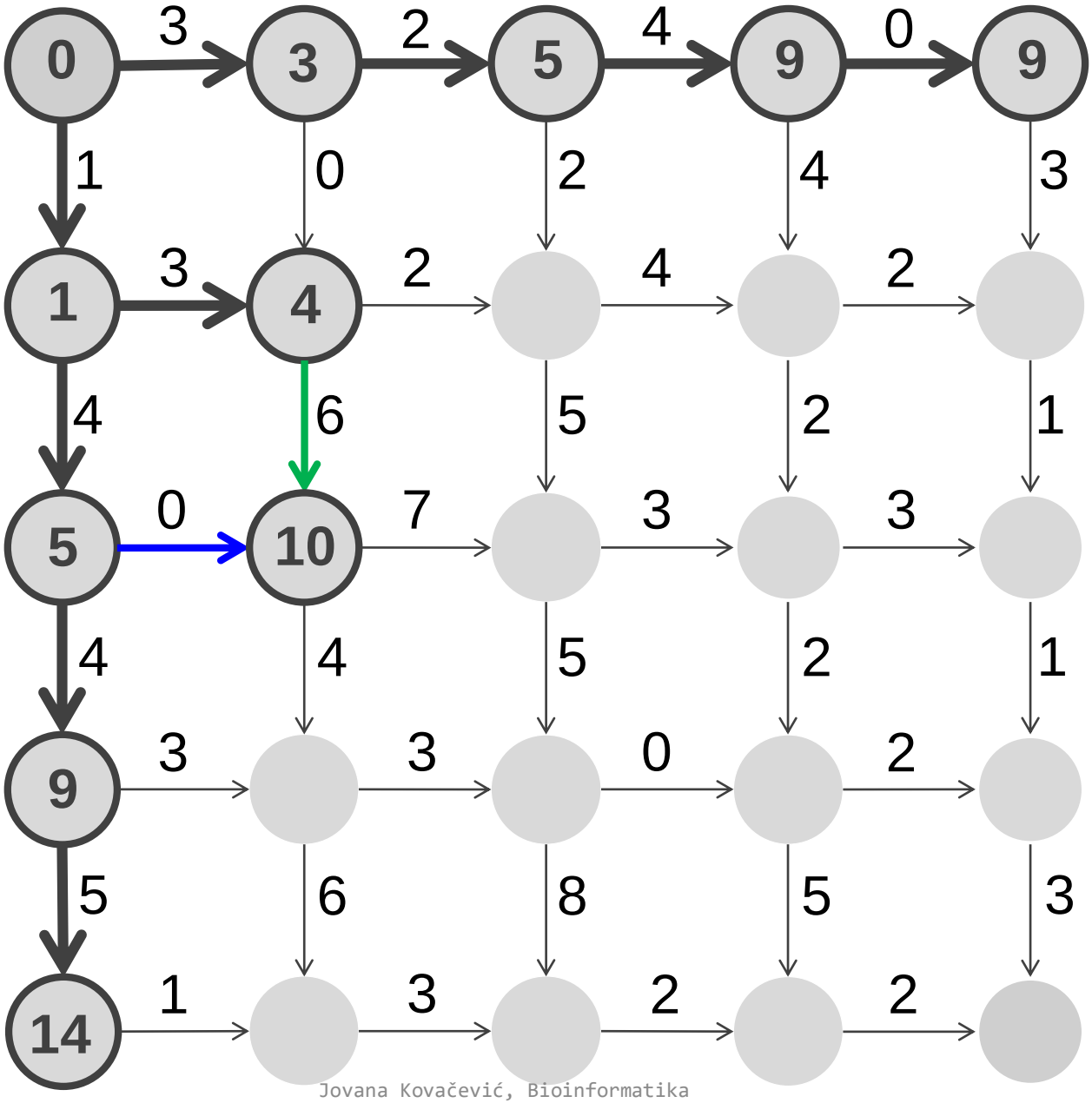
Došli smo do (1,1) preko **podebljane** grane:



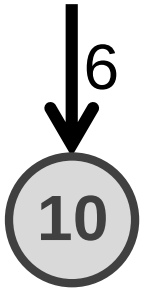


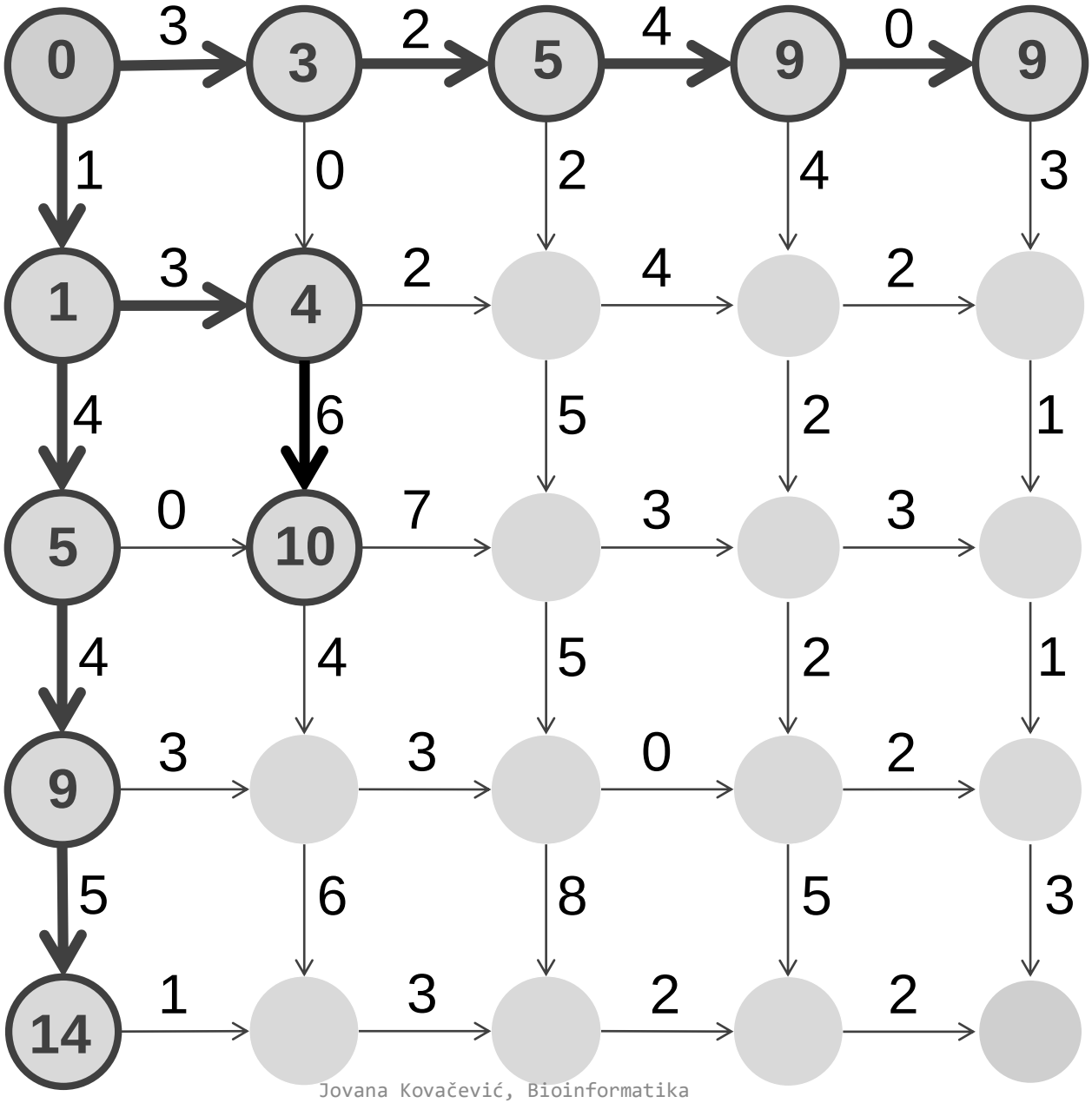
južno
ili
istočno?

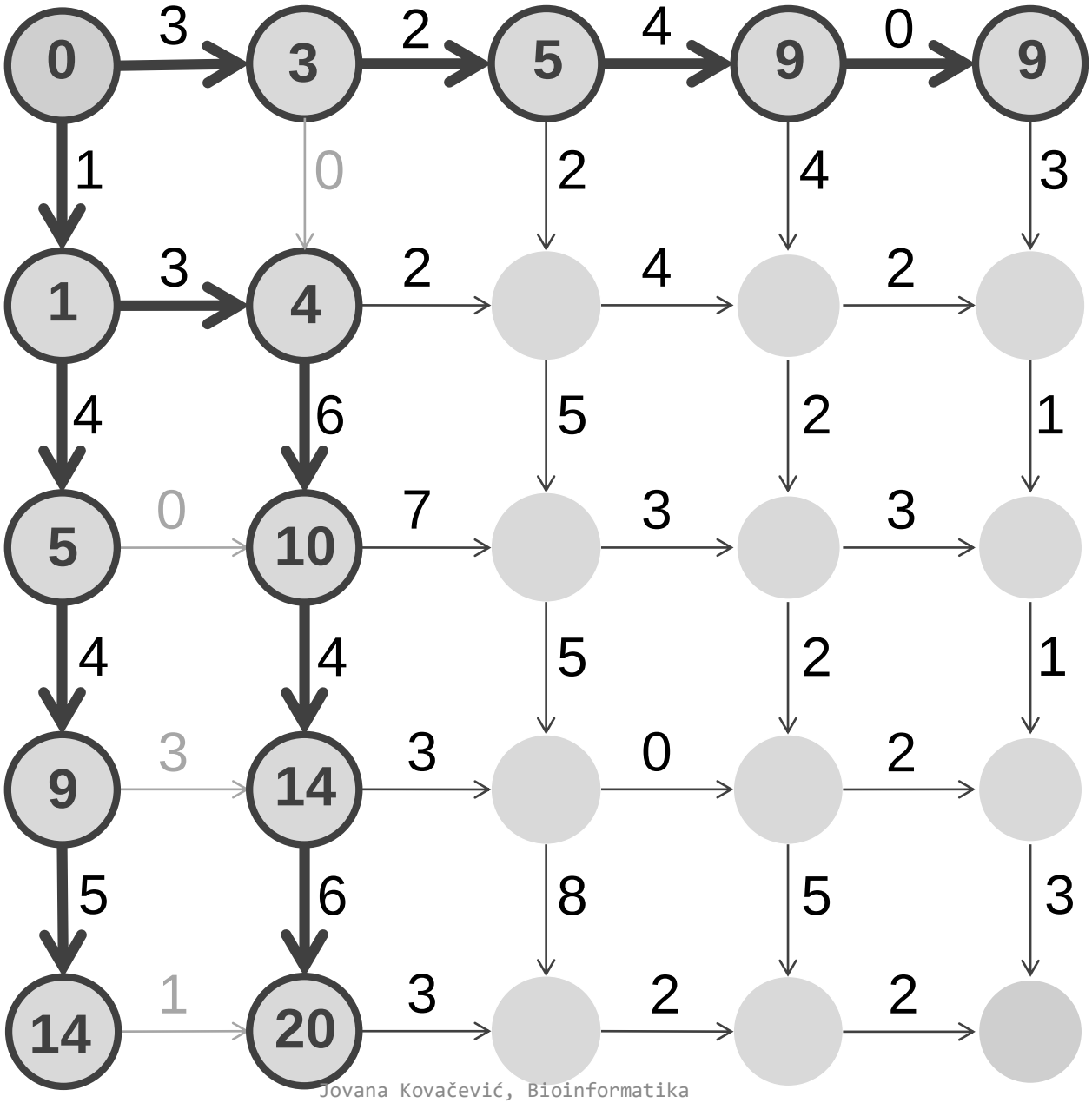
5+0 <
4+6



Došli smo do (2,1) preko **podebljane** grane

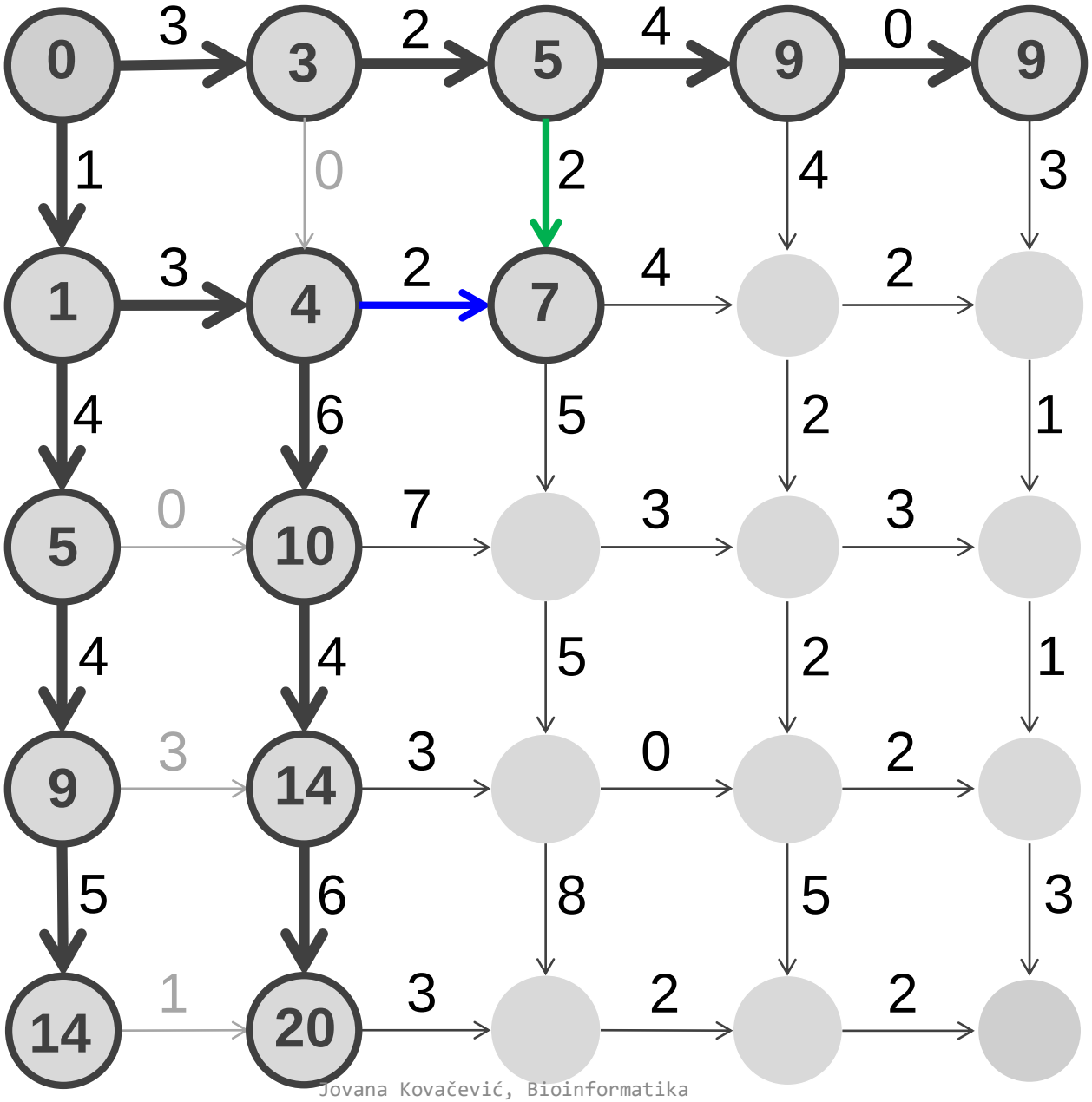





0
3
3
2
5
4
9
0
9
1
0
2
4
3
1
3
4
2
4
2
4
6
5
2
1
5
0
10
7
3
3
4
4
5
2
1
9
3
14
3
0
2
5
6
8
5
3
14
1
20
3
2
2

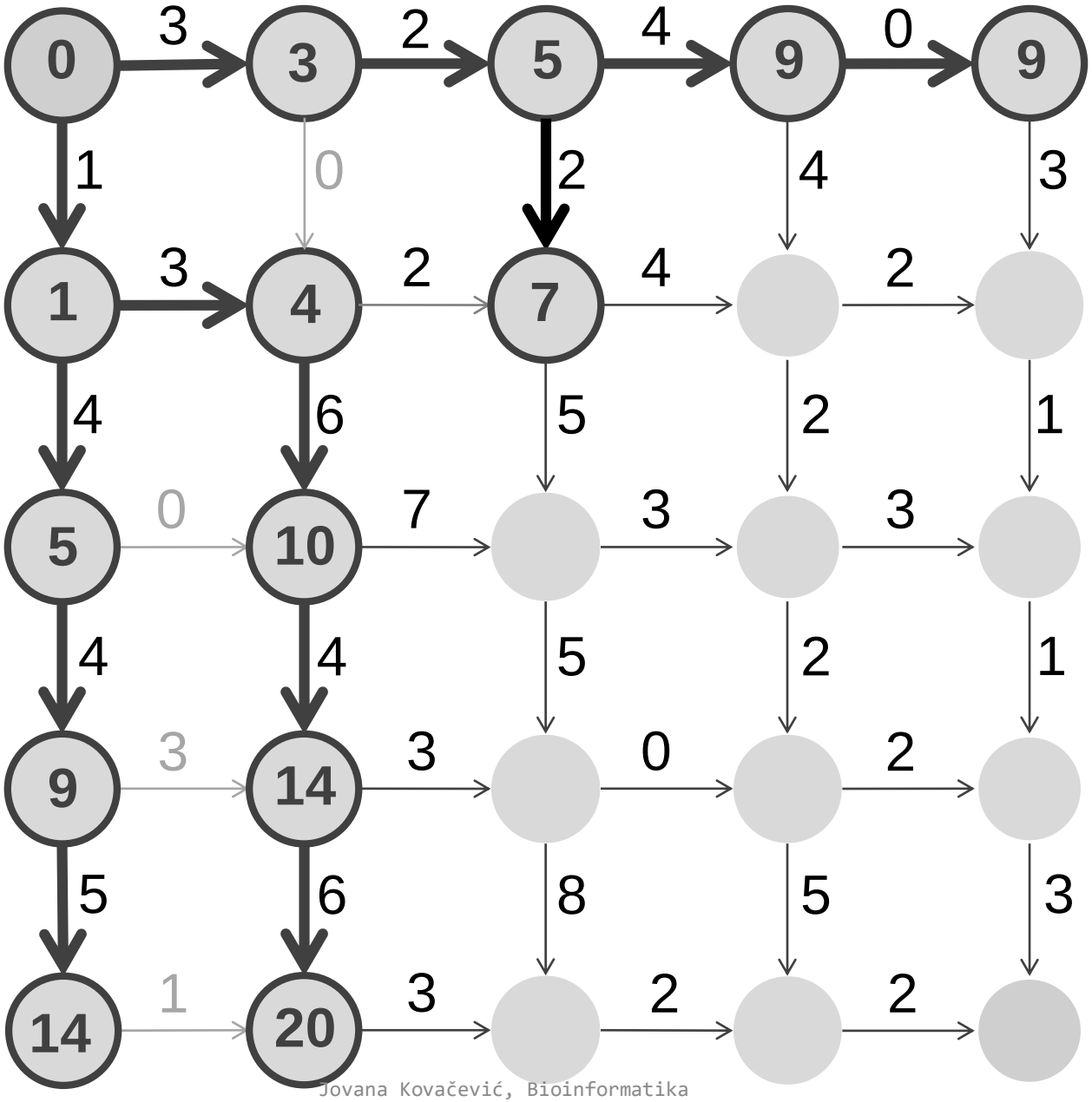
južno
ili
istočno?

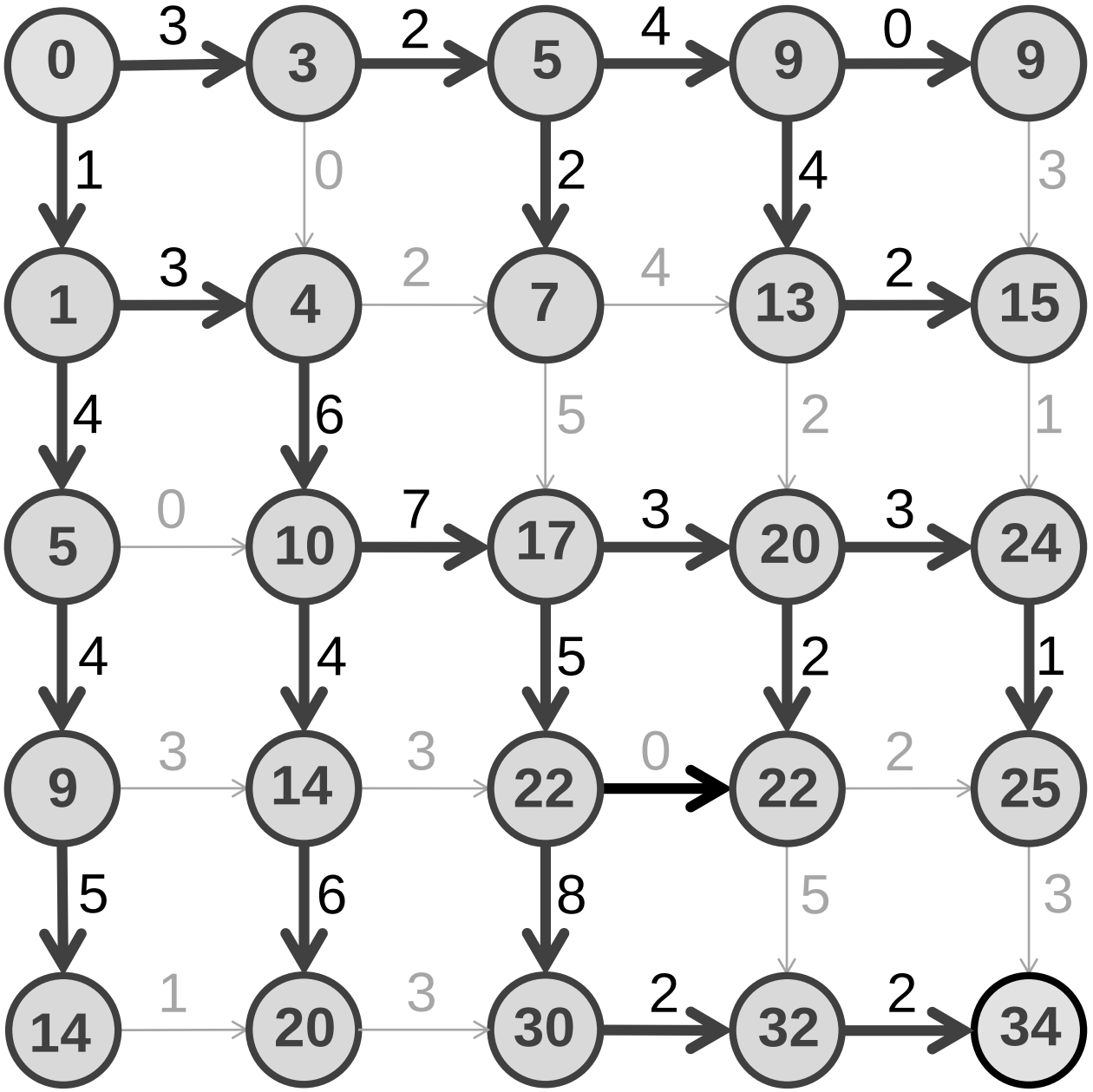
5+2 >
4+2



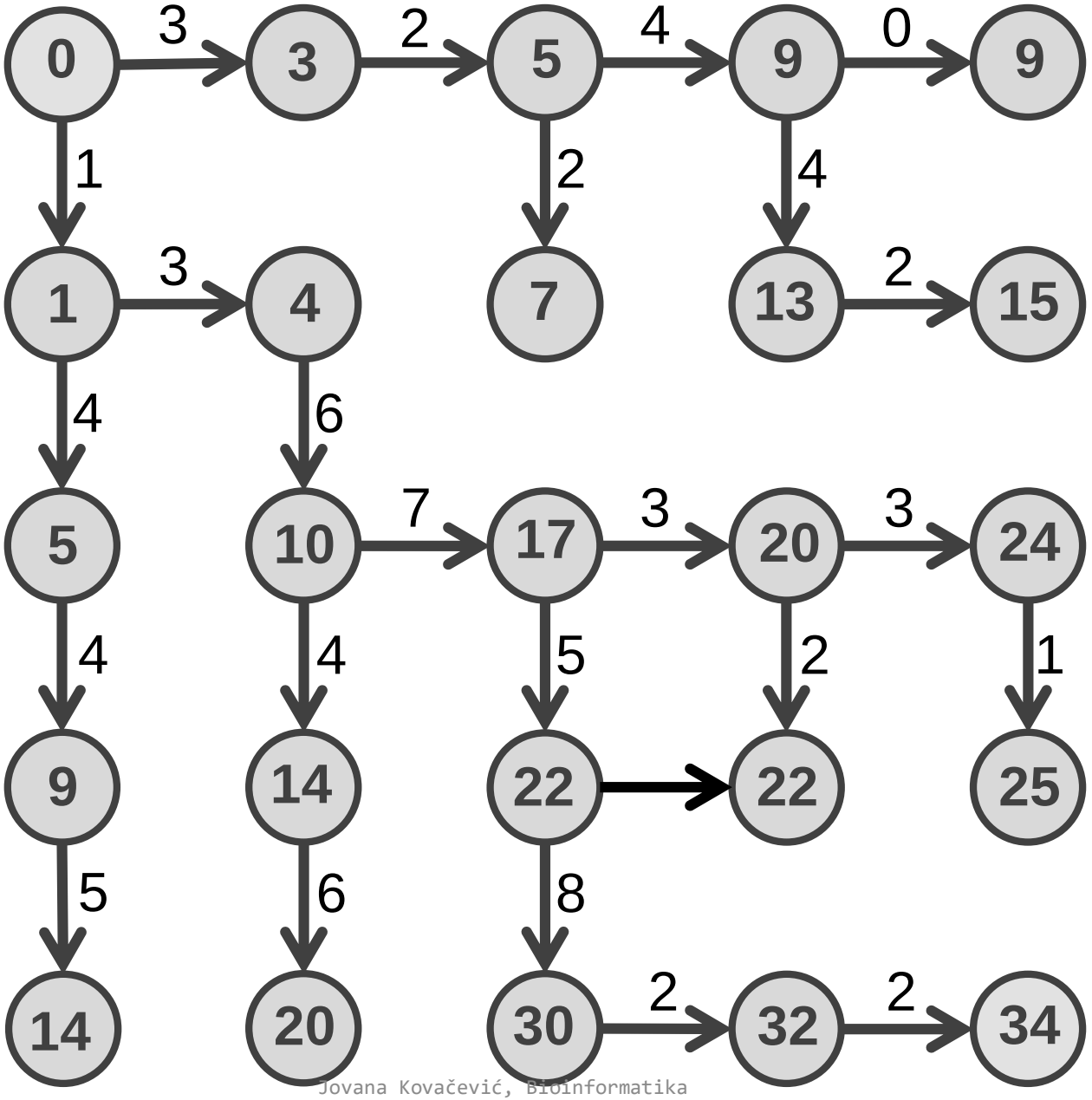
južno
ili
istočno?

5+2 >
4+2




0
3
5
9
9
1
4
7
13
15
5
10
17
20
24
9
14
22
22
25
14
20
30
32
34
3
2
4
0
1
0
2
4
3
1
3
2
4
2
4
6
5
2
1
0
7
3
3
4
4
5
2
1
3
3
0
2
5
6
8
5
3
1
3
2
2

Podebljane grane predstavljaju **putokaze za povratak** od čvora *sink* do čvora *source*: najbolji način da dođemo do čvora



Rekurentna relacija dinamičkog programiranja kod Menhetn grafa

$s_{i,\ j}$: the length of a longest path from (0,0) to (i,j)

$$s_{i,\ j} = \max\{ \begin{array}{l} s_{i-1,\ j} + \text{weight of edge “}\downarrow\text{”into } (i,j) \\ s_{i,\ j-1} + \text{weight of edge “}\rightarrow\text{”into } (i,j) \end{array}$$

MANHATTANTOURIST(n, m, *Down*, *Right*)

$s_{0,0} \leftarrow 0$

for $i \leftarrow 1$ to n

$s_{i,0} \leftarrow s_{i-1,0} + down_{i,0}$

for $j \leftarrow 1$ to m

$s_{0,j} \leftarrow s_{0,j-1} + right_{0,j}$

for $i \leftarrow 1$ to n

for $j \leftarrow 1$ to m

$s_{i,j} \leftarrow \max\{s_{i-1,j} + down_{i,j},\ s_{i,j-1} + right_{i,j}\}$

return $s_{n,m}$

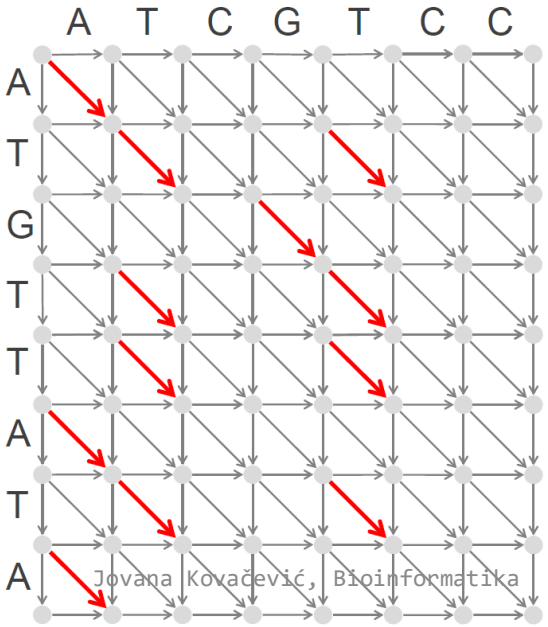
Pregled

- Biološki uvid u poređenje sekvenci
- Igra poravnanja i najduža zajednička podniska
- Problem turiste na Menhetnu
- Problem kusura
- Dinamičko programiranje i putokazi za povratak
- **Od Menhetna do grafa poravnanja**
- Od globalnog do lokalnog poravnanja
- Kažnjavanje insercija i delecija u poravnanju sekvenci
- Prostorno efikasno poravnanje sekvenci
- Višestruko poravnanje sekvenci

Rekurentna relacija dinamičkog programiranja kod grafa poravnanja

$s_{i,\,j}$: the length of a longest path from (0,0) to (i,j)

$$s_{i,\,j} = \max \begin{cases} s_{i-1,\,j} + \text{weight of edge “↓” into } (i,j) \\ s_{i,\,j-1} + \text{weight of edge “→” into } (i,j) \\ s_{i-1,\,j-1} + \text{weight of edge “↘” into } (i,j) \end{cases}$$

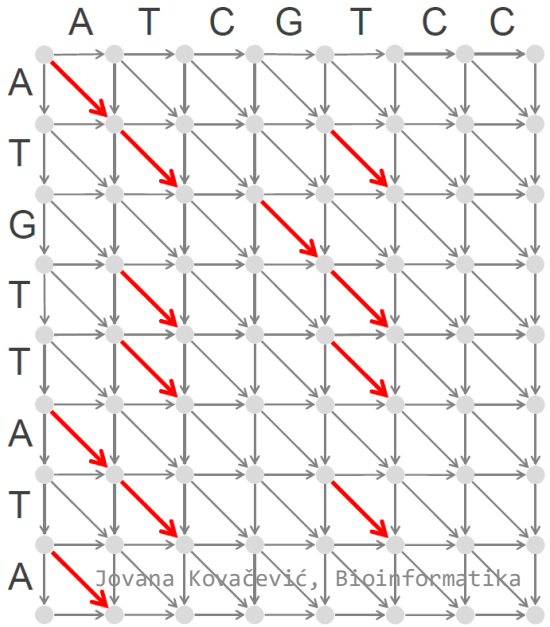


crvene grane ↘ – težina 1
ostale grane – težina 0

Rekurentna relacija dinamičkog programiranja kod nalaženja najduže zajedničke podsekvence

$s_{i,\,j}$: the length of a longest path from (0,0) to (i,j)

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



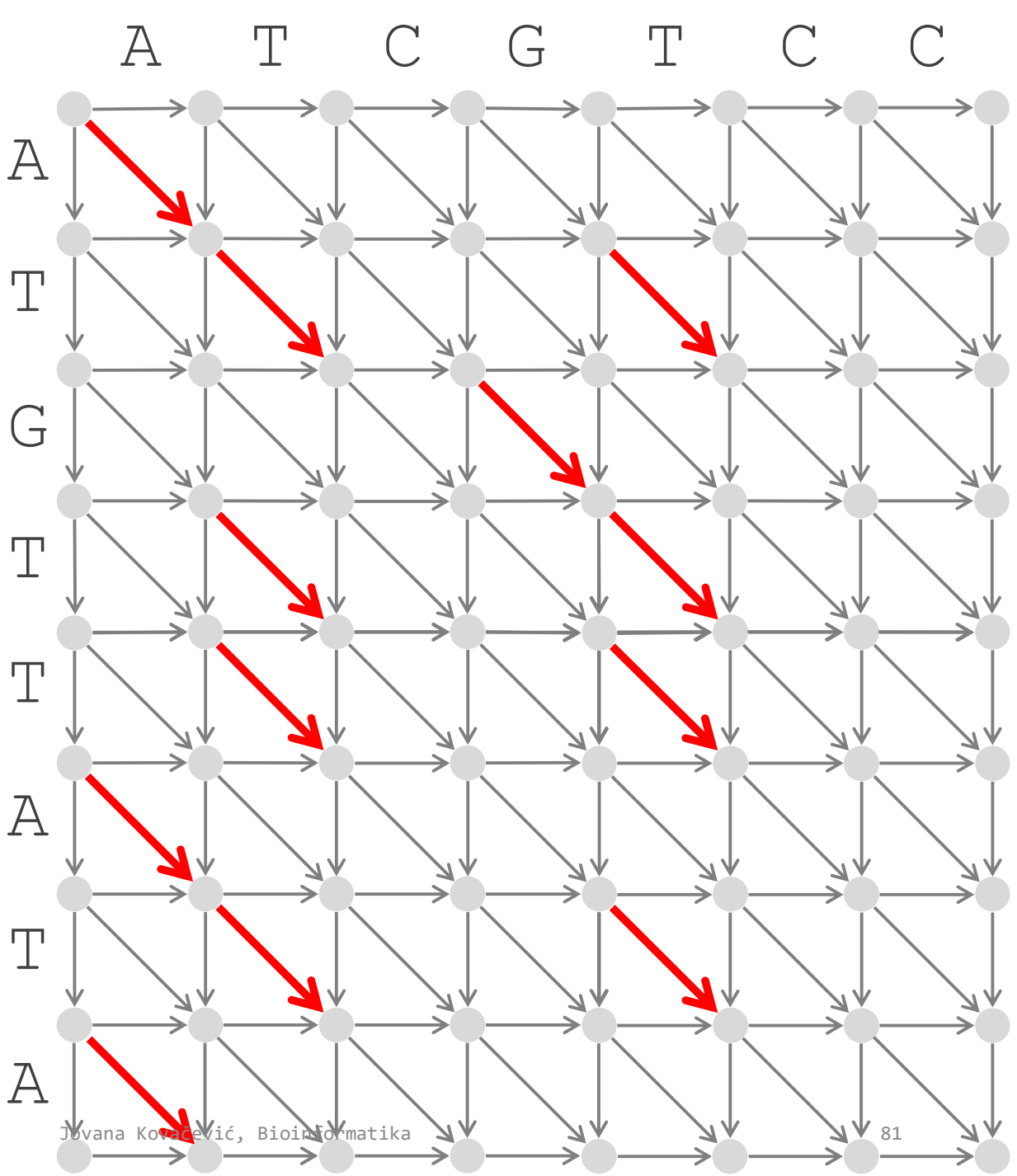
crvene grane ↘ – težina 1

ostale grane – težina 0

v_i, w_j – oznake vrste i kolone

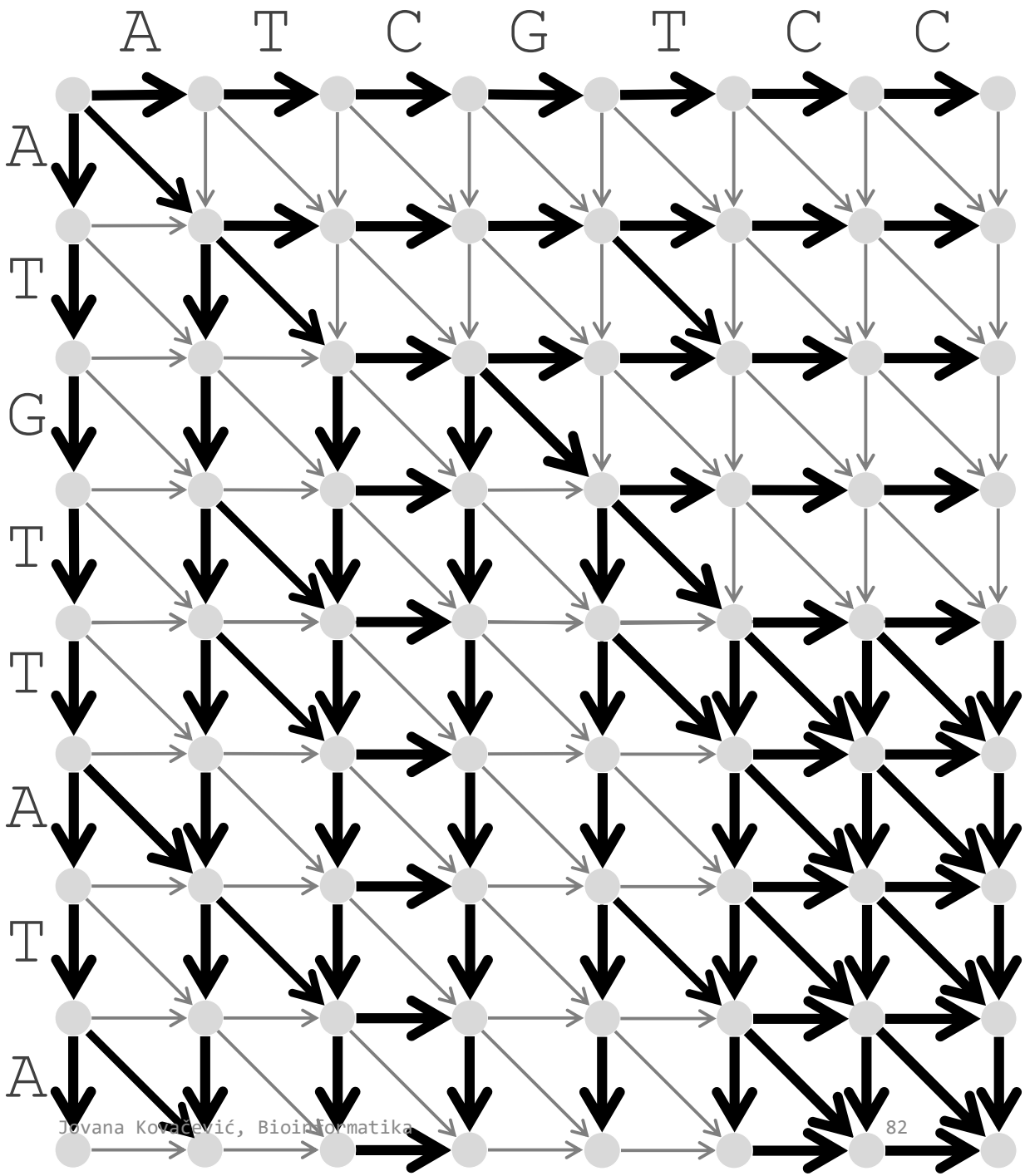
putokazi za povratak kod grafa za najdužu zajedničku podsekvencu

crvene grane ↘ – težina 1
ostale grane – težina 0



boldovane grane su nastale primenom pravila rekurentne relacije

one predstavljaju **putokaze za povratak** (***backtrack***) kod grafa za najdužu zajedničku podsekvencu



Računanje putokaza za povratak

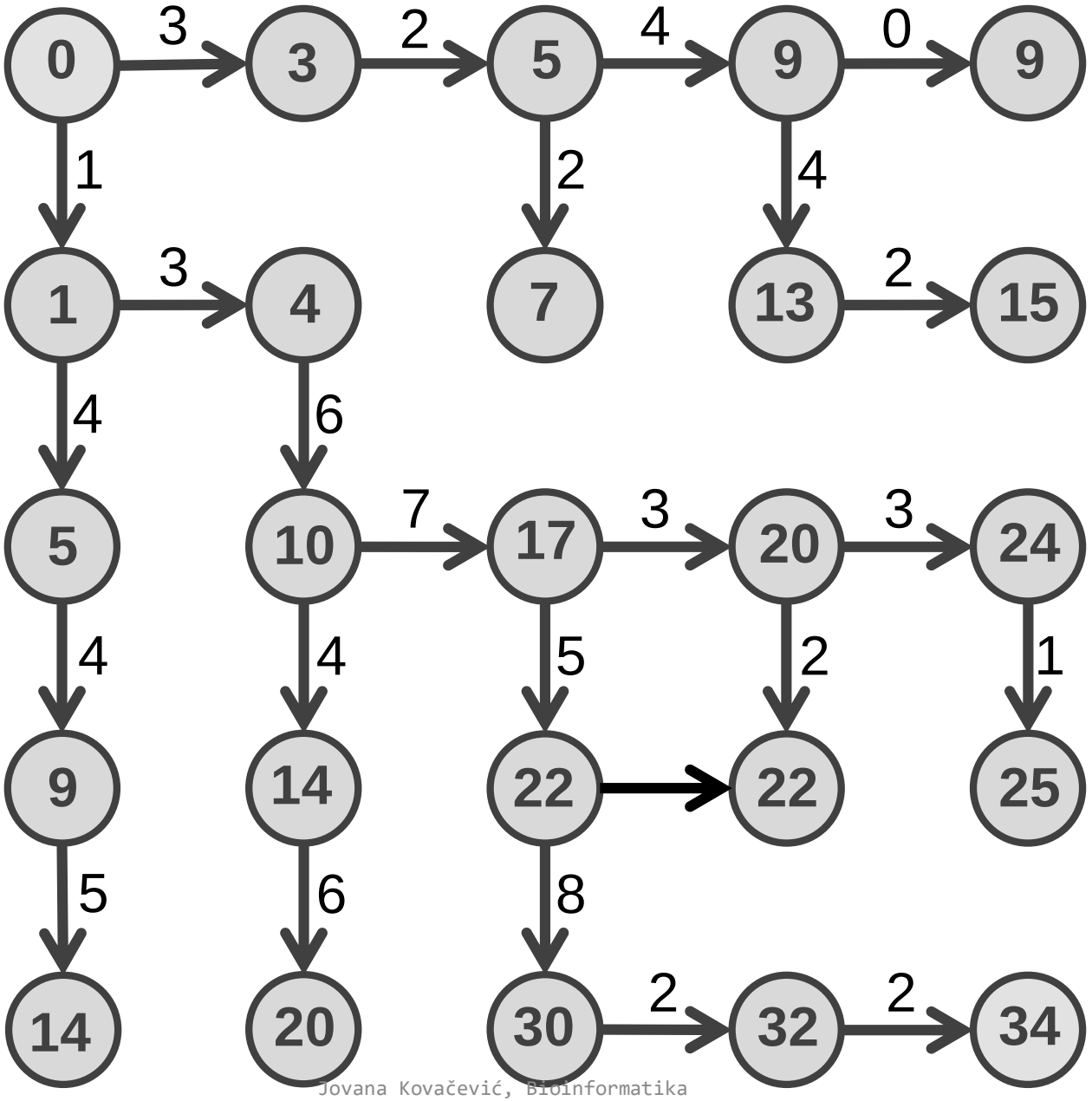
$$s_{i,j} \leftarrow \max\{ s_{i,j-1}+0,\ s_{i-1,j}+0,\ s_{i-1,j-1}+1 \text{ if } v_i=w_j,\ s_{i-1,j-1}+0 \text{ if } v_i\neq w_j \}$$

$$backtrack_{i,j} \leftarrow \{ \text{“}\rightarrow\text{”, if } s_{i,j}=s_{i,j-1},\ \text{“}\downarrow\text{”, if } s_{i,j}=s_{i-1,j},\ \text{“}\searrow\text{”, otherwise} \}$$

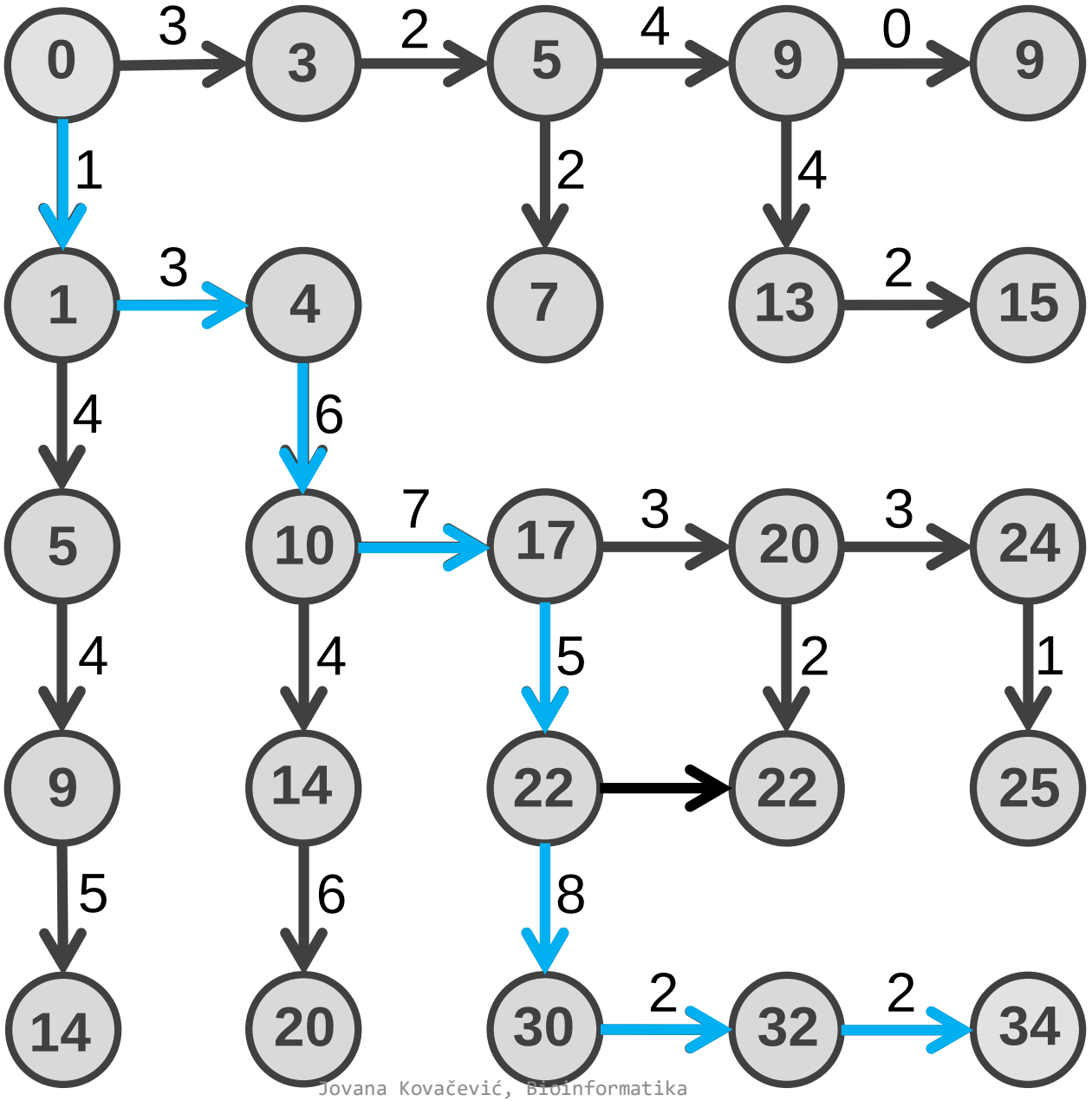
Longest Common Subsequence BackTrack

```
LCSBackTrack(v, w)
    for i ← 0 to |v|
        s_{i, 0} ← 0
    for j ← 0 to |w|
        s_{0, j} ← 0
    for i ← 1 to |v|
        for j ← 1 to |w|
            match ← 0
            if v_{i-1} = w_{j-1}
                match ← 1
            s_{i, j} ← max{s_{i-1, j} , s_{i,j-1} , s_{i-1, j-1} + match}
            if s_{i,j} = s_{i-1,j}
                Backtrack_{i, j} ← "↓"
            else if s_{i, j} = s_{i, j-1}
                Backtrack_{i, j} ← "→"
            else if s_{i, j} = s_{i-1, j-1} + match
                Backtrack_{i, j} ← "↘"
    return Backtrack
```

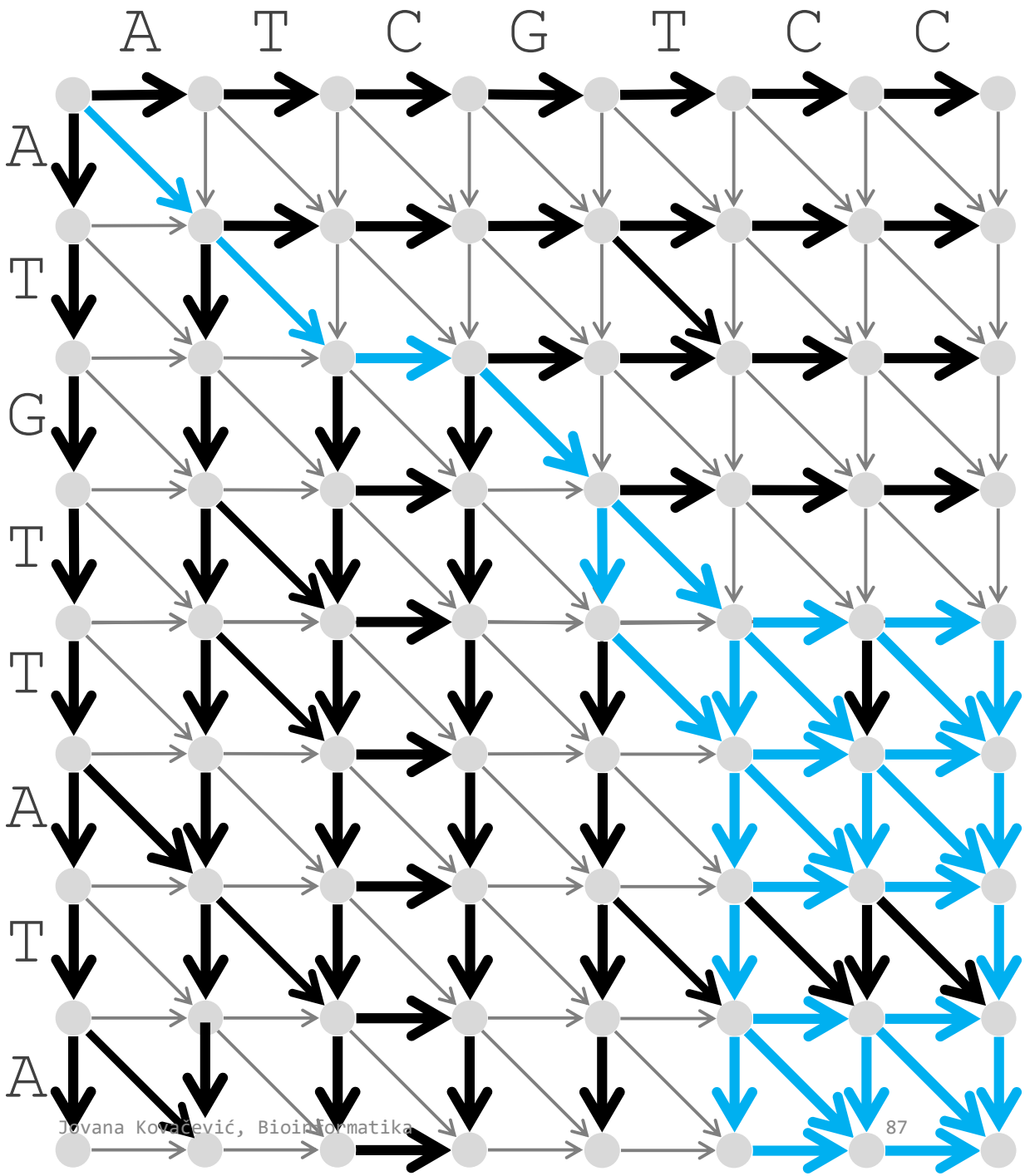
Podsetimo se kako smo rekontru-isali putanju preko putokaza kod Menhetn grafa



Krenuli bismo od krajnjeg čvora (*sink*) i pratili putokaze u obrnutom smeru do početnog čvora (*source*)



povratak (*backtracking*) kod grafa za najdužu zajedničku podsekvencu

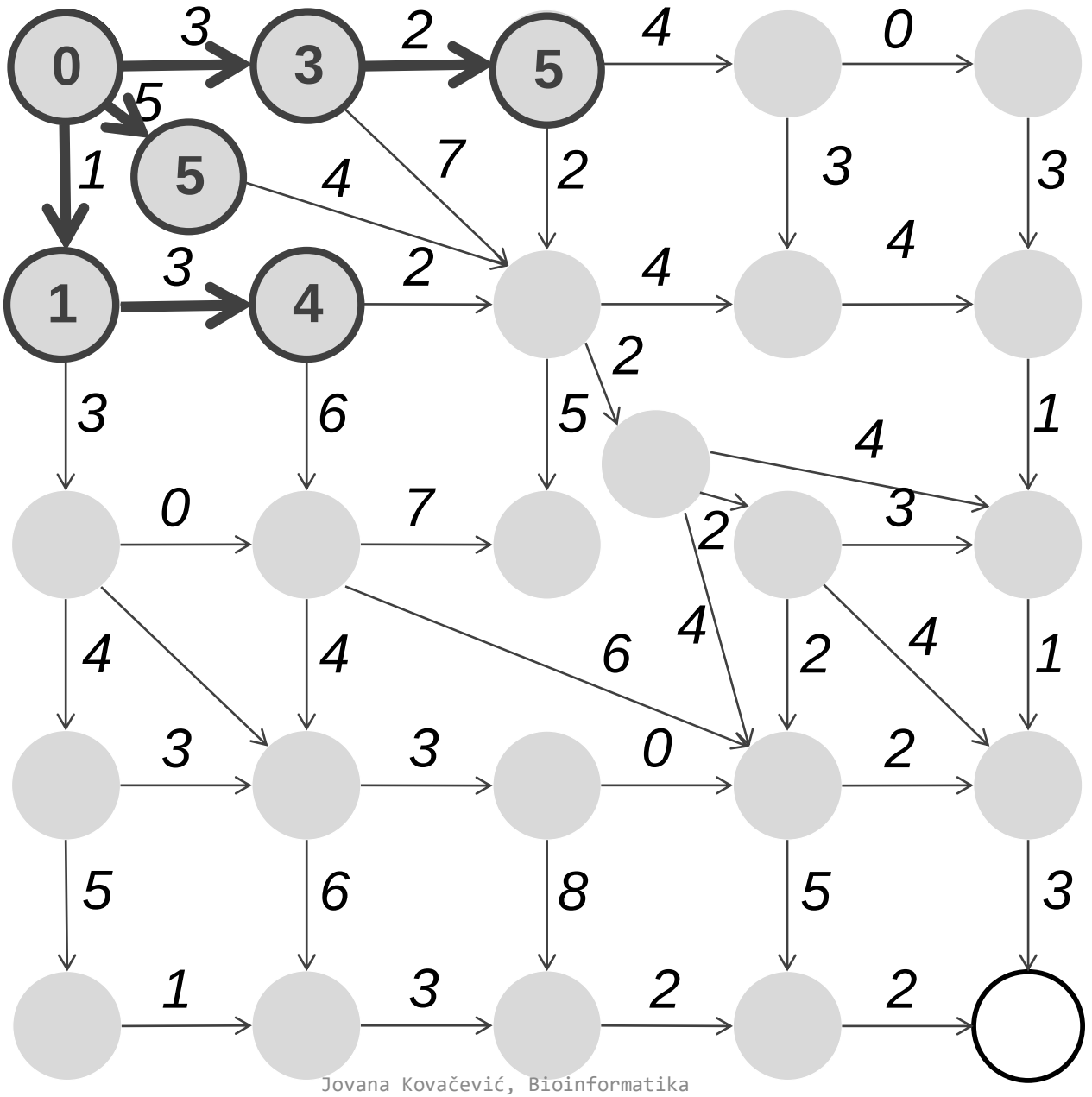


Određivanje najduže zajedničke podsekvence (*LCS – longest common subsequence*) korišćenjem putokaza za povratak

```
OutputLCS (backtrack, v, i, j)
  if i = 0 or j = 0
    return
  if backtrack_{i,j} = "→"
      OutputLCS (backtrack, v, i, j-1)
    else if backtrack_{i,j} = "↓"
      OutputLCS (backtrack, v, i-1, j)
    else
      OutputLCS (backtrack, v, i-1, j-1)
      output v_i
```

- Do sada smo pretpostavljali da graf u kom tražimo najdužu putanju ima samo tri vrste grana
- Da li se *OutputLCS* može generalizovati tako da važi i za grafove koji nemaju tako specifičnu topologiju?

Kako se rekuren-tna relacija dinami-čkog progra-miranja menja za ovakav graf?



$s_a = \max_{\text{all predecessors } b \text{ of node } a}\{s_b + \text{weight of edge from } b \text{ to } a\}$

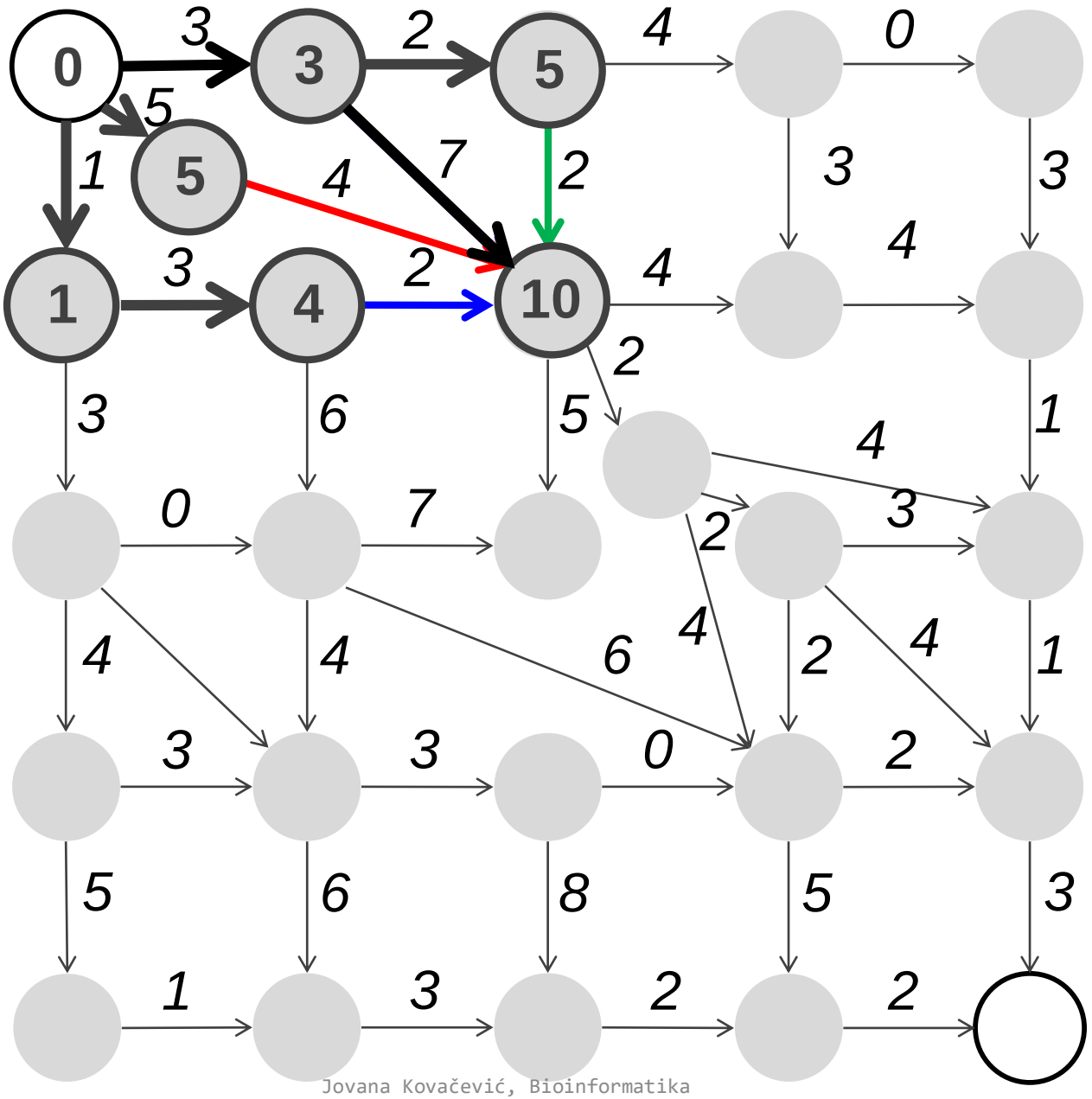
4 izbora:

5 + 2

3 + 7

5 + 4

4 + 2



$s_a = \max_{\text{all predecessors } b \text{ of node } a}\{s_b + \text{weight of edge from } b \text{ to } a\}$

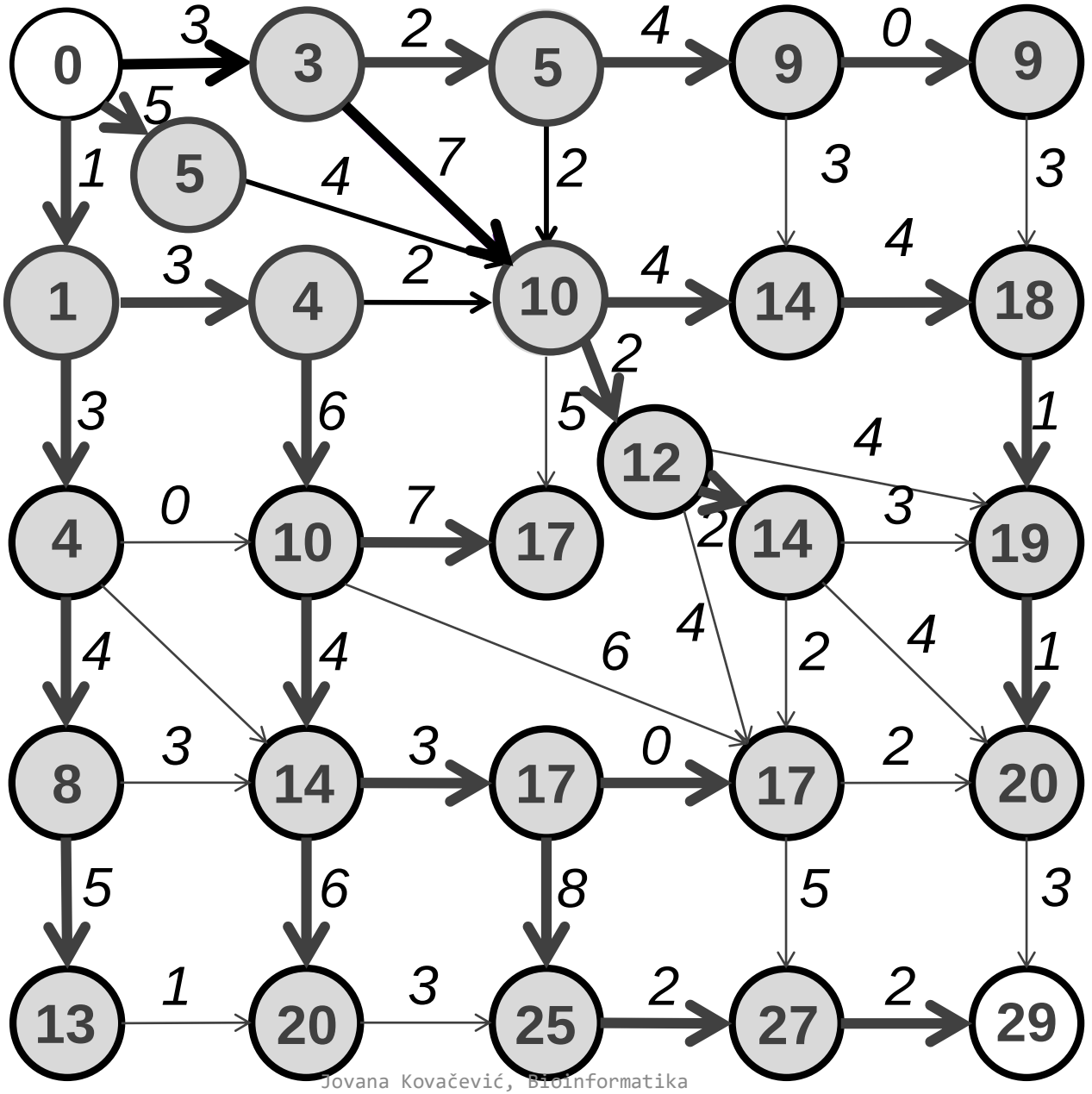
4 izbora:

5 + 2

3 + 7

5 + 4

4 + 2

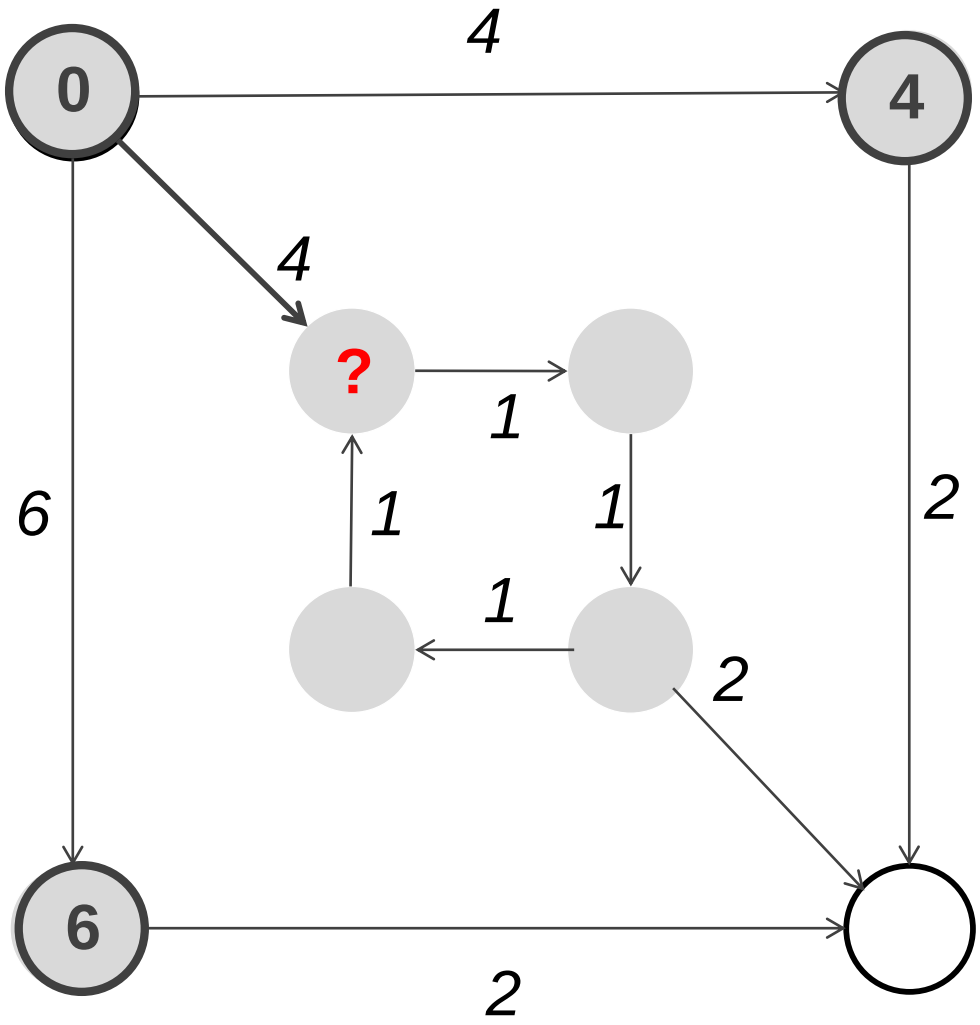


s_a = max$_{\text{all}}$ predecessors b of node a $\{s_b$ + weight of edge from b to $a\}$

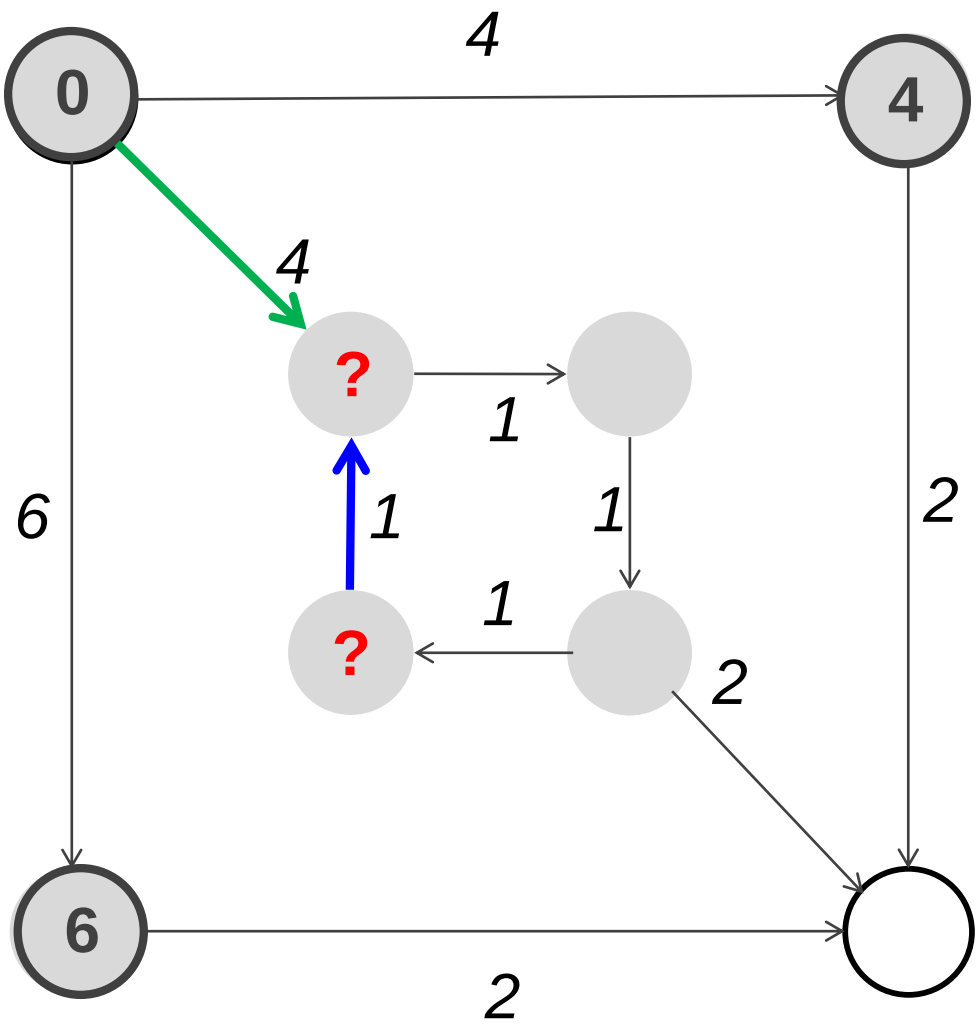
Pitanje:

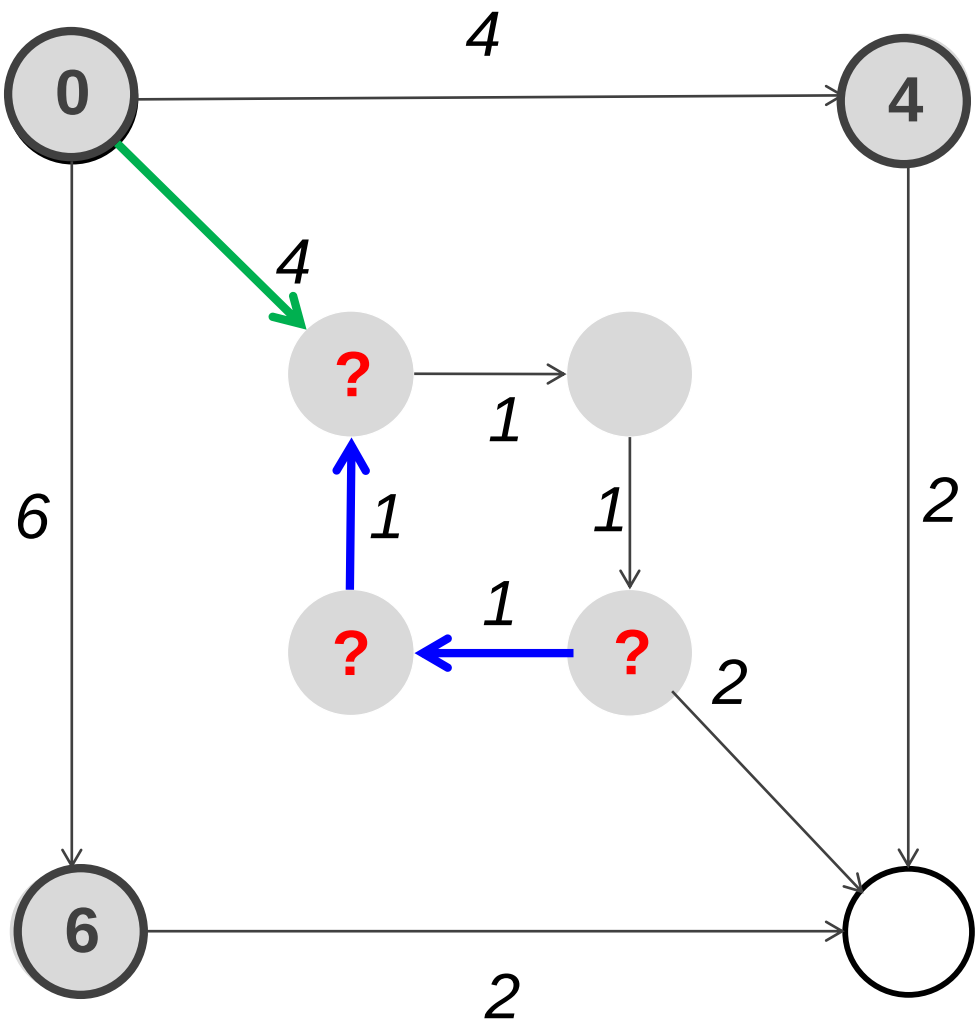
- Kod ovakve rekurentne relacije, važno je da pri računanju s_a imamo izračunate s_b za **sve** čvorove prethodnike b (čvorovi za koje postoji grana do čvora a)
- Da li je to moguće u bilo kom usmerenom težinskom grafu?

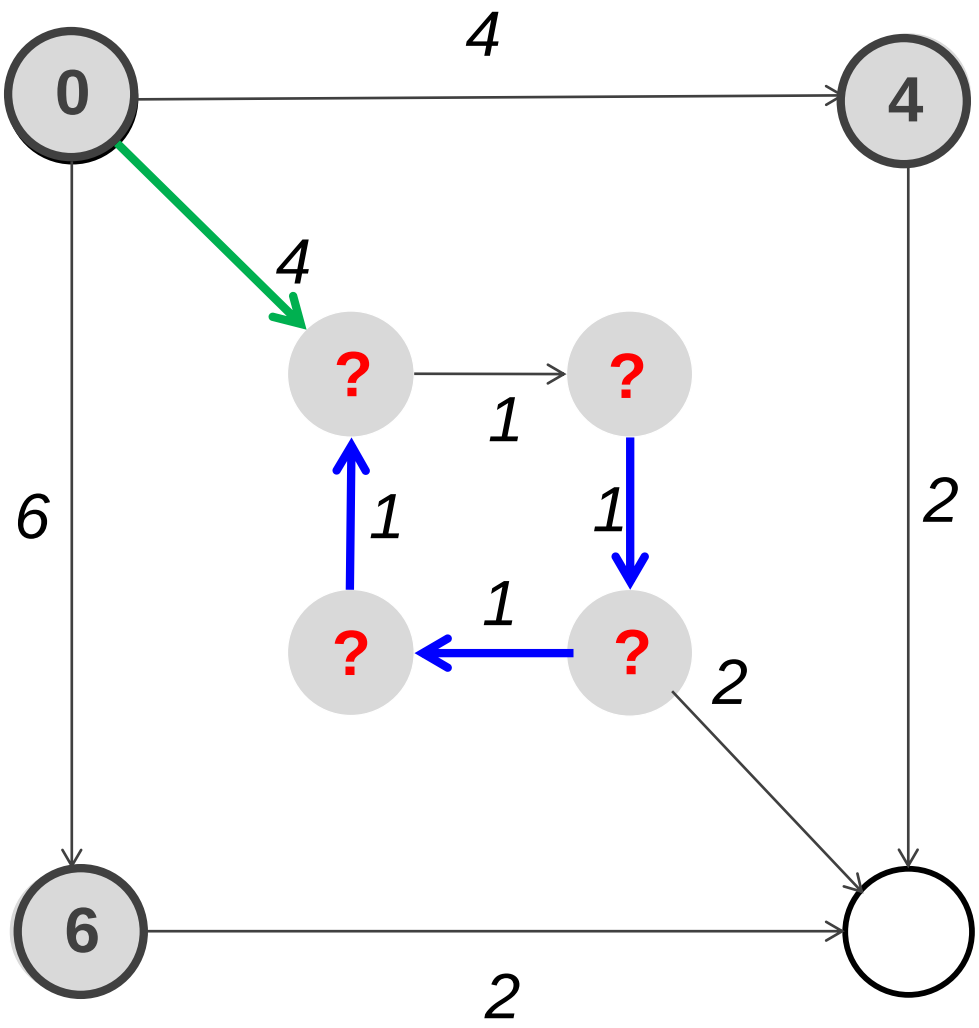
Računanje skora za SVE prethodnike



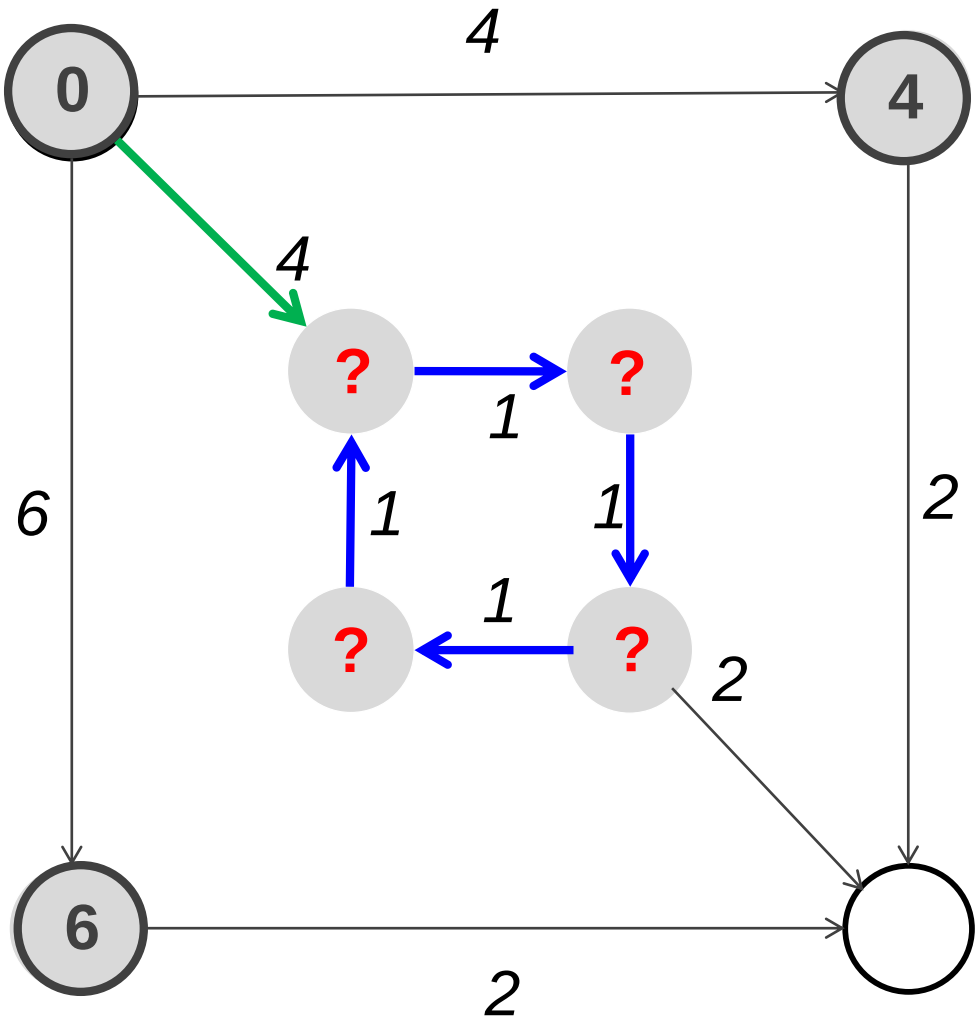
s_a = max$_{\text{all predecessors } b \text{ of node } a}$ {s_b + weight of edge from b to a}


0
4
4
?
1
6
1
1
2
1
?
2
6
2


0
4
4
?
1
6
1
1
2
1
?
?
2
6
2


4
0
4
4
?
?
1
6
1
1
2
1
?
?
2
6
2

Začarani krug



s_a = max$_{\text{all predecessors } b \text{ of node } a}\{s_b + \text{weight of edge from } b \text{ to } a\}$

Pitanje:

- Kod ovakve rekurentne relacije, važno je da pri računanju s_a imamo izračunate s_b za **sve** čvorove prethodnike b (čvorovi za koje postoji grana do čvora a)

- Da li je to moguće u bilo kom usmerenom težinskom grafu?

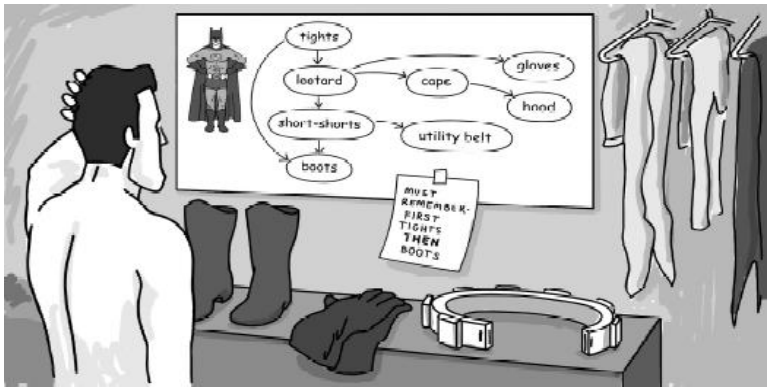
Odgovor:

- Nije. Da bismo kod svakog čvora mogli da mogli da izračunamo skor za sve njegove prethodnike, usmereni težinski graf mora biti *acikličan*
- *DAG (Directed Acyclic Graph)*

s_a = max$_{\text{all}}$ predecessors b of node a {s_b + weight of edge from b to a}

Pitanje:

- Ako je dat usmereni aciklični graf, da li njegove čvorove možemo poređati u niz tako da njihov redosled u nizu osigurava uslov da pri računanju s_a imamo izračunate s_b za **sve** čvorove prethodnike b (čvorovi za koje postoji grana do čvora a)?

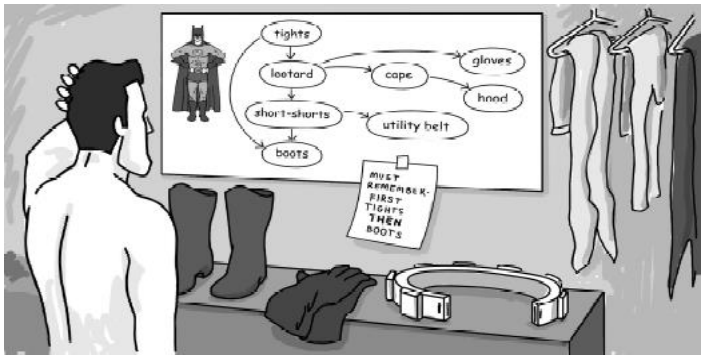


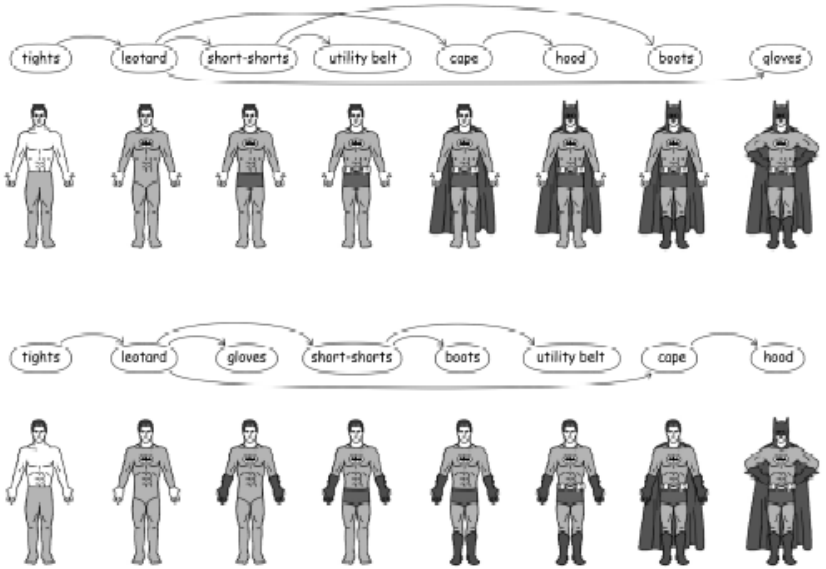
Odgovor:

- Da, moguće je poređati sve čvorove grafa u niz i taj niz *topološki sortirati*

Topološko sortiranje

- **Topološko sortiranje**: Sortiranje čvorova DAG-a u nizu tako da sve grane u takvom nizu idu sleva nadesno.





- **Teorema**: Svaki DAG se može topološki sortirati.
- Topološko sortiranje svakog DAG-a se obavlja za *O(#edges)* koraka

Algoritam za nalaženje najduže putanje u DAG-u

LongestPath(*Graph, source, sink*)
  for each node *a* in *Graph*
    $s_a \leftarrow$ -infinity
  $s_{source} \leftarrow 0$
  topologically order *Graph*
  for each node *a* (from *source* to *sink* in topological order)
    $s_a \leftarrow \max_{\text{all predecessors } b \text{ of node } a}\{s_b + \text{weight of edge from } b \text{ to } a\}$
  return s_{sink}

- Pošto svaka grana učestvuje tačno jednom, složenost je O(#edges)
- LongestPath vraća dužinu najdužeg zajedničkog podniza ali ne rekonstruiše putanju

Pregled

- Biološki uvid u poređenje sekvenci
- Igra poravnanja i najduža zajednička podsekvenca
- Problem turiste na Menhetnu
- Problem kusura
- Dinamičko programiranje i putokazi za povratak
- Od Menhetna do grafa poravnanja
- **Od globalnog do lokalnog poravnanja**
- Kažnjavanje insercija i delecija u poravnanju sekvenci
- Prostorno efikasno poravnanje sekvenci
- Višestruko poravnanje sekvenci

Skor poravnanja do sada

#matches

Skor sa *mismatch* i *indel* kaznama

#matches − μ · #mismatches − σ · #indels

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

	A	C	G	T	−
A	+1	$-\mu$	$-\mu$	$-\mu$	$-\sigma$
C	$-\mu$	+1	$-\mu$	$-\mu$	$-\sigma$
G	$-\mu$	$-\mu$	+1	$-\mu$	$-\sigma$
T	$-\mu$	$-\mu$	$-\mu$	+1	$-\sigma$
−	$-\sigma$	$-\sigma$	$-\sigma$	$-\sigma$	

Matrica skora

	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	

Još generalnija matrica skora

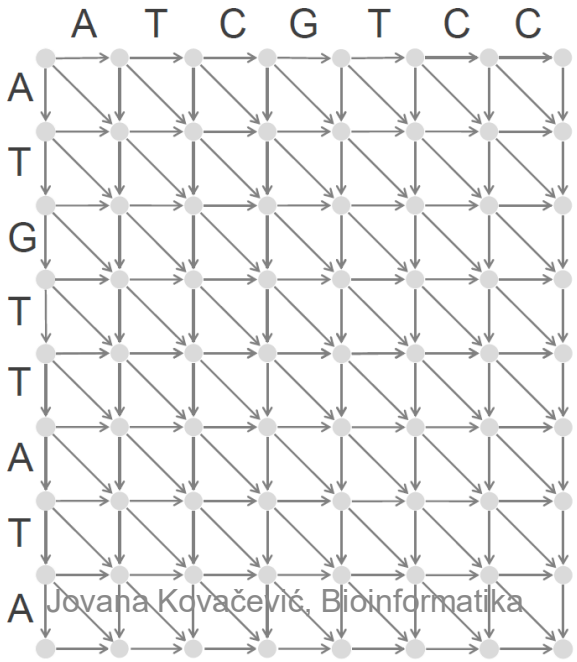
Primer matrice skora za proteinske sekvence

C Cys	12																		
S Ser	0	2																	
T Thr	-2	1	3																
P Pro	-3	1	0	6															
A Ala	-2	1	1	1	2														
G Gly	-3	1	0	-1	1	5													
N Asn	-4	1	0	-1	0	0	2												
D Asp	-5	0	0	-1	0	1	2	4											
E Glu	-5	0	0	-1	0	0	1	3	4										
Q Gln	-5	-1	-1	0	0	-1	1	2	2	4									
H His	-3	-1	-1	0	-1	-2	2	1	1	3	6								
R Arg	-4	0	-1	0	-2	-3	0	-1	-1	1	2	6							
K Lys	-5	0	0	-1	-1	-2	1	0	0	1	0	3	5						
M Met	-5	-2	-1	-2	-1	-3	-2	-3	-2	-1	-2	0	0	6					
I Ile	-2	-1	0	-2	-1	-3	-2	-2	-2	-2	-2	-2	-2	2	5				
L Leu	-6	-3	-2	-3	-2	-4	-3	-4	-3	-2	-2	-3	-3	4	2	6			
V Val	-2	-1	0	-1	0	-1	-2	-2	-2	-2	-2	-2	-2	2	4	2	4		
F Phe	-4	-3	-3	-5	-5	-5	-4	-6	-5	-5	-2	-4	-5	0	1	2	-1	9	
Y Tyr	0	-3	-3	**-5**	-3	-5	-2	-4	-4	-4	0	-4	-4	-2	-1	-1	-2	**7**	10
W Trp	-8	-2	-5	-6	-6	-7	-4	-7	-7	-5	-3	2	-3	-4	-5	-2	-6	0	0	17
	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W

Y (Tyr) često mutira u F (skor +7) ali retko u P (skor -5)

Rekurentna relacija dinamičkog programiranja kod grafa poravnanja

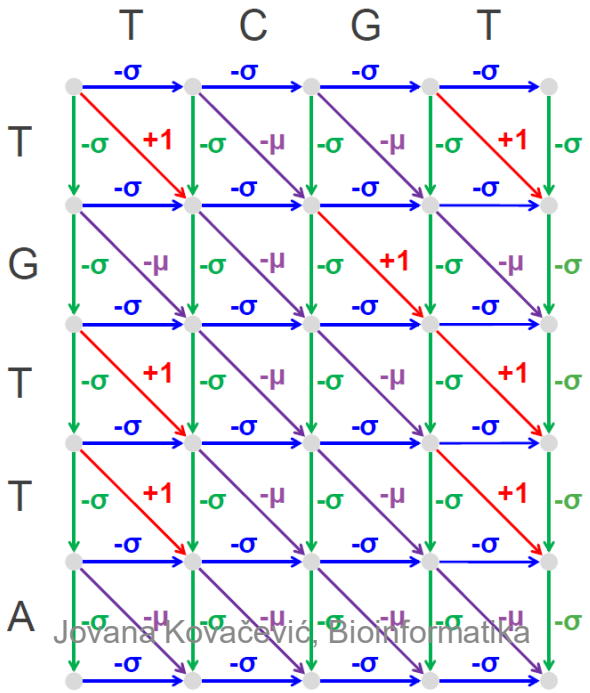
$$s_{i,j} = \max \begin{cases} s_{i-1,j} + \text{weight of edge “}\downarrow\text{” into } (i,j) \\ s_{i,j-1} + \text{weight of edge “}\rightarrow\text{” into } (i,j) \\ s_{i-1,j-1} + \text{weight of edge “}\searrow\text{” into } (i,j) \end{cases}$$



Rekurentna relacija dinamičkog programiranja kod grafa poravnanja

$$s_{i,j} = \max \begin{cases} s_{i-1,j} - \sigma \\ s_{i,j-1} - \sigma \\ s_{i-1,j-1} + 1, \text{ if } v_i = w_j \\ s_{i-1,j-1} - \mu, \text{ if } v_i \neq w_j \end{cases}$$

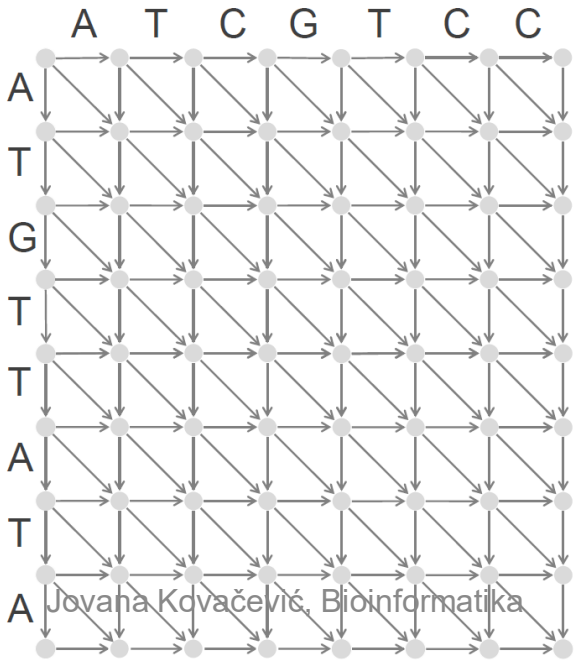
	A	C	G	T	–
A	+1	$-\mu$	$-\mu$	$-\mu$	$-\sigma$
C	$-\mu$	+1	$-\mu$	$-\mu$	$-\sigma$
G	$-\mu$	$-\mu$	+1	$-\mu$	$-\sigma$
T	$-\mu$	$-\mu$	$-\mu$	+1	$-\sigma$
–	$-\sigma$	$-\sigma$	$-\sigma$	$-\sigma$	



Rekurentna relacija dinamičkog programiranja kod grafa poravnanja

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

	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	



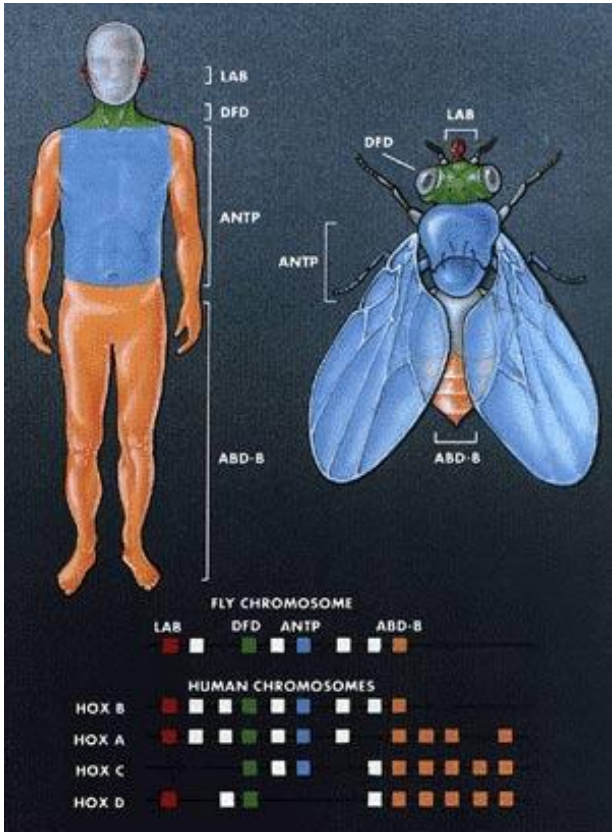
Globalno poravnanje

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 *score*) maksimalan od svih mogućih poravnanja v i w.

Homeobox geni

- Dva gena u različitim vrstama mogu biti slična u kratkim, konzerviranim regionima a različita u ostalim delovima
- *Homeobox* geni sadrže kratak region **homeodomen** koji je čvrsto konzerviran među različitim vrstama
- Globalno poravnanje može da propusti nalaženje homeodomena jer pokušava da poravna sekvence u celosti



Koje poravnanje je bolje?

- Poravnanje 1: score = 22 (matches) - 20 (indels)=2.

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

- Poravnanje 2: score = 17 (matches) - 30 (indels)=-13.

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

Koje poravnanje je bolje?

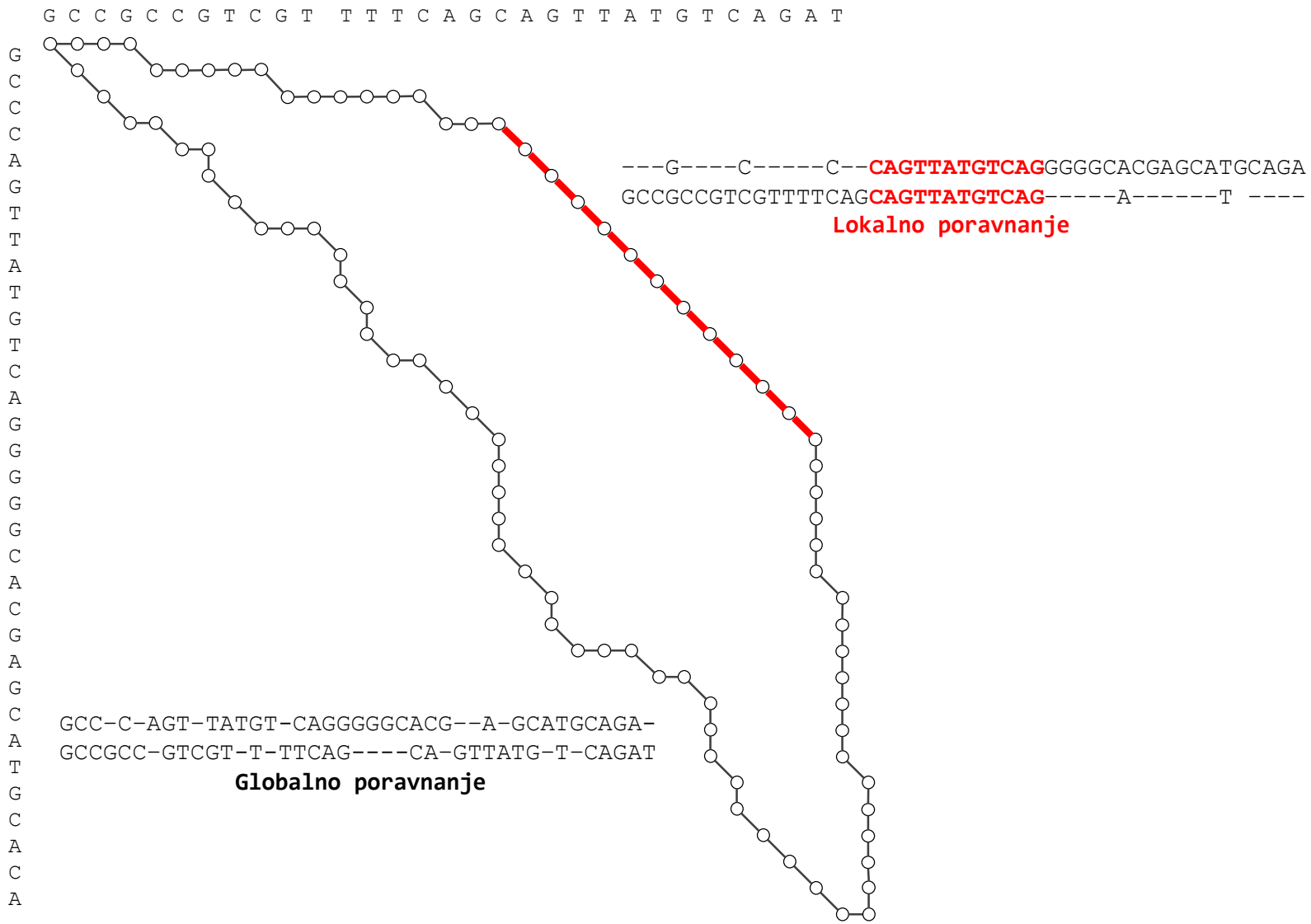
- Poravnanje 1: score = 22 (matches) - 20 (indels)=2.

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

- Poravnanje 2: score = 17 (matches) - 30 (indels)=-13.

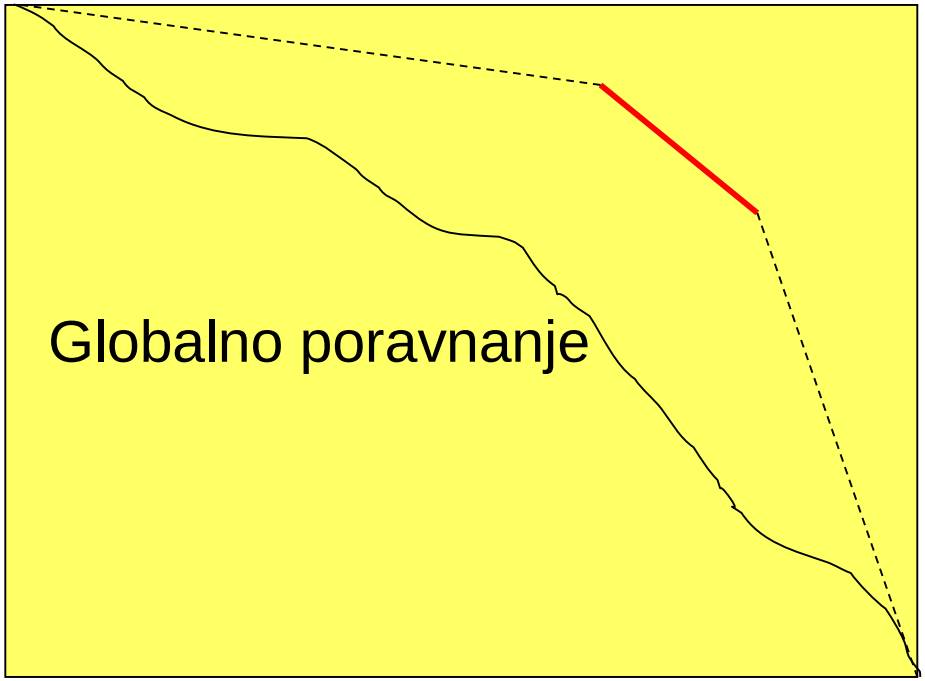
```
---G----C-----C--CAGTTATGTCAGGGGGCACGAGCATGCAGA
GCCGCCGTCGTTTTCAGCAGTTATGTCAG-----A------T-----
```

lokalno poravnanje

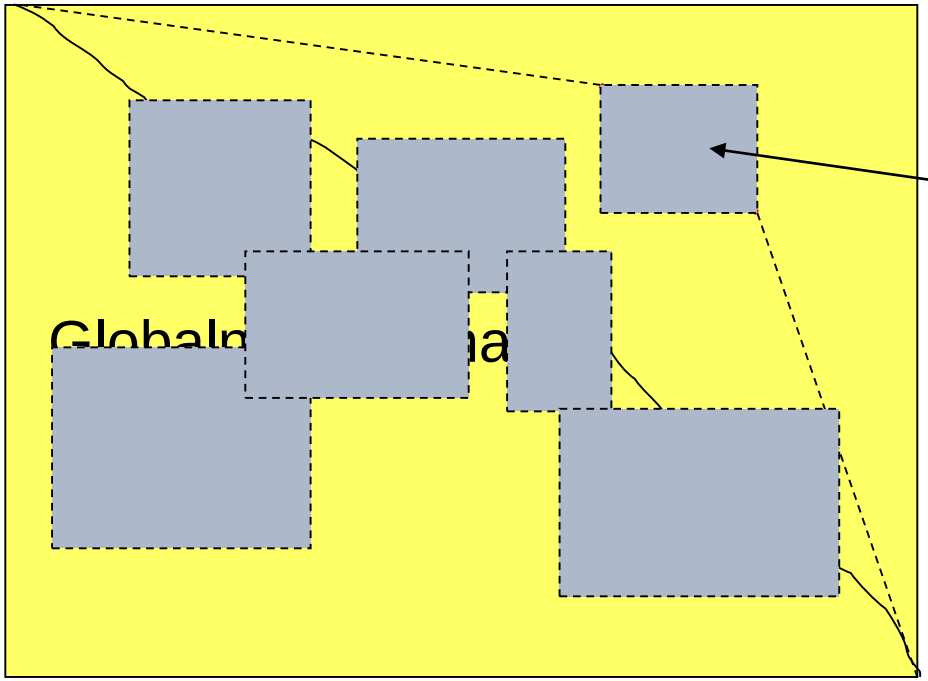

G C C G C C G T C G T T T T C A G C A G T T A T G T C A G A T
G C C C A G T T A T G T C A G G G G G C A C G A G C A T G C A C A
---G----C-----C--CAGTTATGTCAGGGGGCACGAGCATGCAGA
GCCGCCGTCGTTTTCAGCAGTTATGTCAG-----A------T ----
Lokalno poravnanje
GCC-C-AGT-TATGT-CAGGGGGCACG--A-GCATGCAGA-
GCCGCC-GTCGT-T-TTCAG----CA-GTTATG-T-CAGAT
Globalno poravnanje


G C C G C C G T C G T T T T C A G C A G T T A T G T C A G A T
G C C C A G T T A T G T C A G G G G G G C A C G A G C A T G C A C A
---G----C-----C--CAGTTATGTCAGGGGGCACGAGCATGCAGA
GCCGCCGTCGTTTTCAGCAGTTATGTCAG-----A------T ----
Lokalno poravnanje
GCC-C-AGT-TATGT-CAGGGGGCACG--A-GCATGCAGA-
GCCGCC-GTCGT-T-TTCAG----CA-GTTATG-T-CAGAT
Globalno poravnanje

Lokalno poravnanje



Lokalno poravnanje = Globalno poravnanje u pravougaoniku



Izračunamo globalno poravnanje u okviru svakog pravougaonika da bismo dobili lokalno poravnanje

- Algoritam globalnog poravnanja sprovodimo između svaka dva čvora, ne samo između početnog (*source*) i krajnjeg (*sink*)
- Potrebno je ponoviti algoritam za svaki par čvorova - #nodes^2 puta

Problem lokalnog poravnanja

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 *score*), maksimalno među svim globalnim poravnanjima svih podniski niski v i w.

Besplatne taksi vožnje

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

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

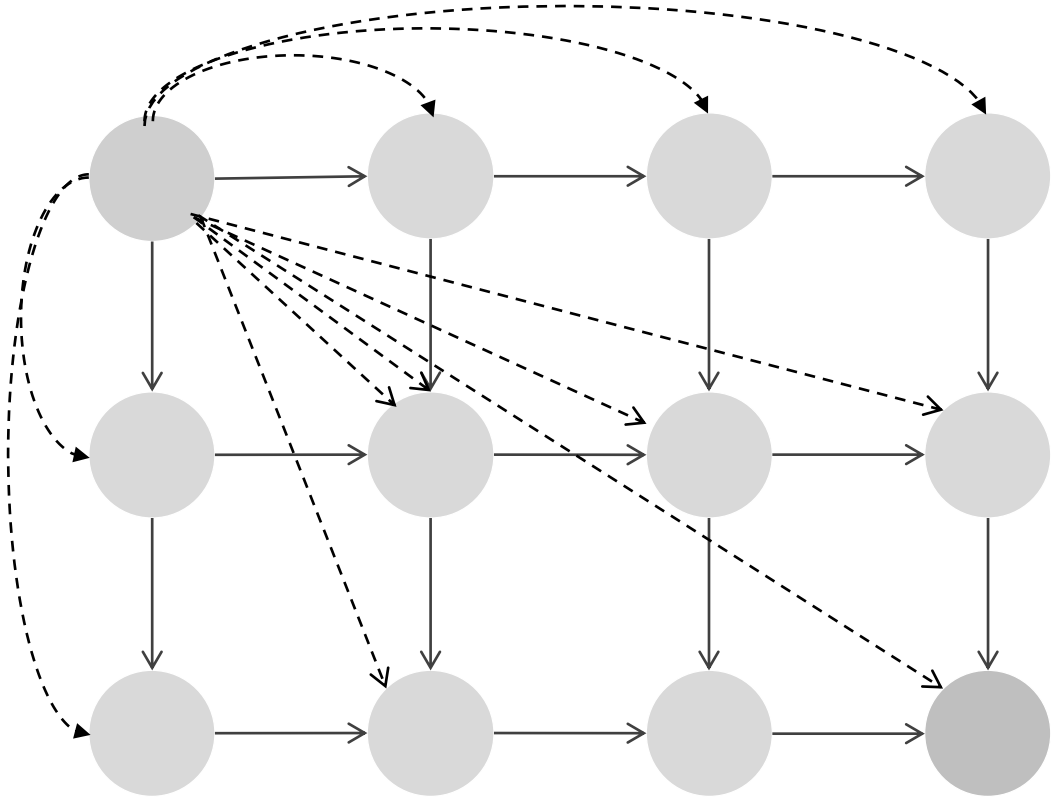
GCC-C-AGT-TATGT-CAGGGGGCACG--A-GCATGCACA-
GCCGCC-GTCGT-T-TTCAG----CA-GTTATG-T-CAGAT

Globalno poravnanje

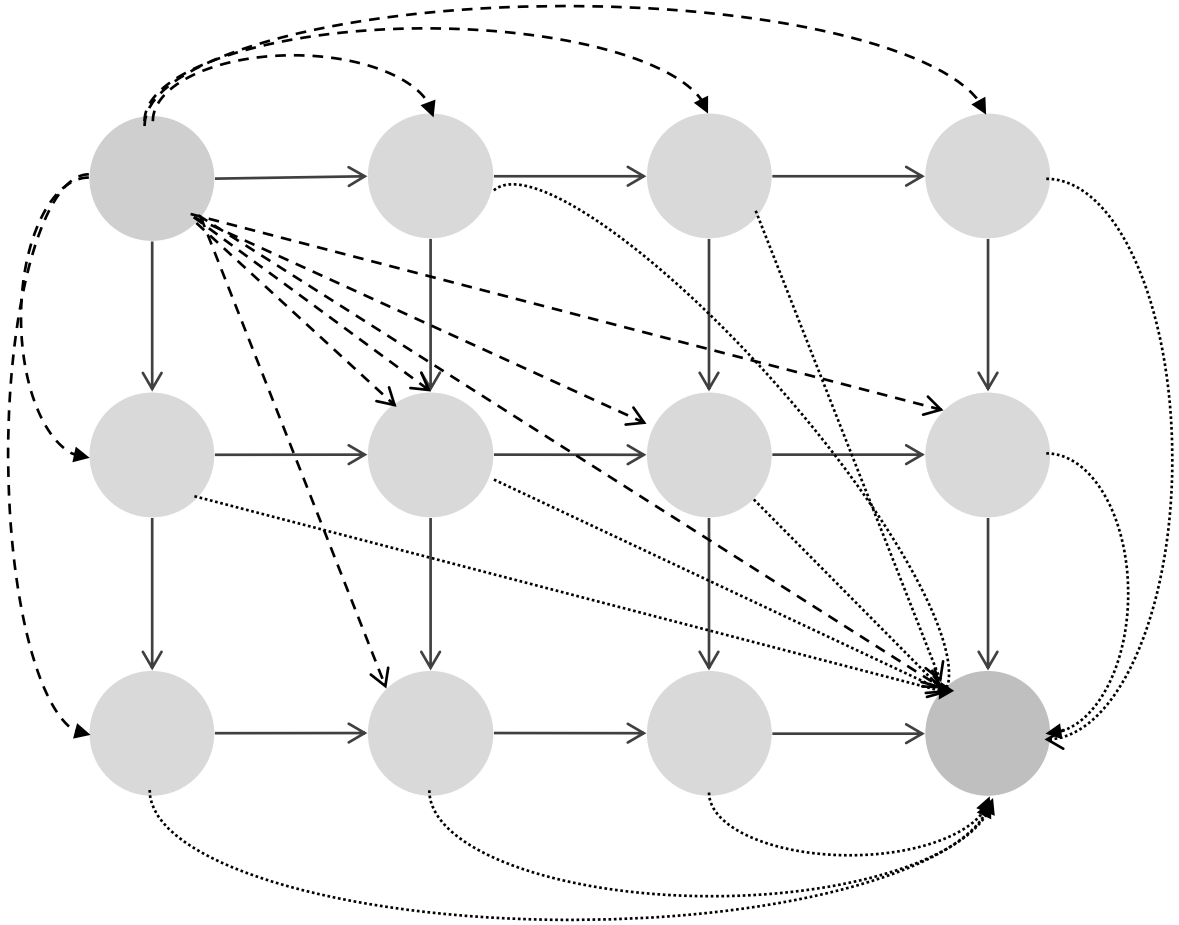
---G----C-----C--**CAGTTATGTCAG**GGGGCACGAGCATGCACA
GCCGCCGTCGTTTTCAG**CAGTTATGTCAG**-----A------T ----

Lokalno poravnanje

Šta su besplatne taksi vožnje na grafu poravnanja?



Izgradnja Menhetn grafa za problem lokalnog poravnanja



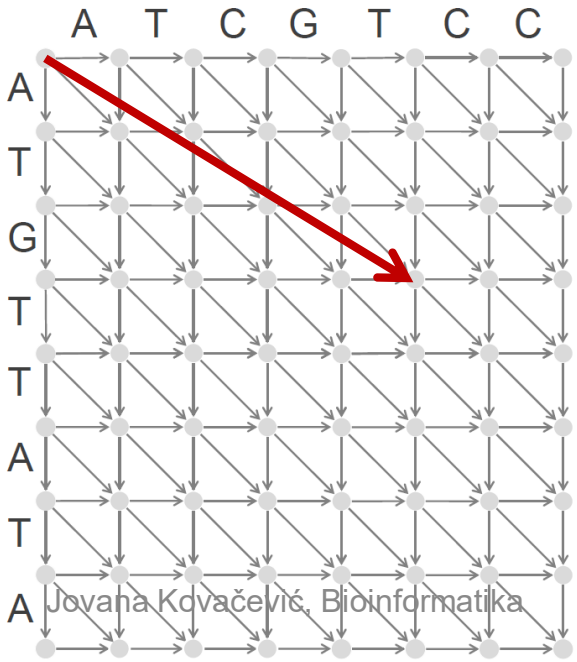
Koliko grana smo dodali?

O(|v|*|w|)

Dinamičko programiranje za lokalno poravnanje

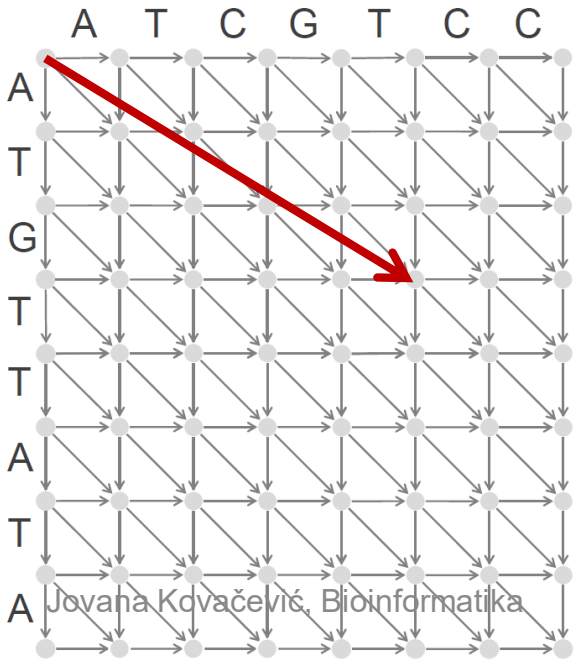
weight of edge from (0,0) to (i,j)

$$s_{i,j} = \max \begin{cases} s_{i-1,j} + \text{weight of edge “}\downarrow\text{” into } (i,j) \\ s_{i,j-1} + \text{weight of edge “}\rightarrow\text{” into } (i,j) \\ s_{i-1,j-1} + \text{weight of edge “}\searrow\text{” into } (i,j) \end{cases}$$



Dinamičko programiranje za lokalno poravnanje

$$s_{i,j} = \max \begin{cases} 0 \\ s_{i-1,j} + \text{weight of edge “}\downarrow\text{” into } (i,j) \\ s_{i,j-1} + \text{weight of edge “}\rightarrow\text{” into } (i,j) \\ s_{i-1,j-1} + \text{weight of edge “}\searrow\text{” into } (i,j) \end{cases}$$



Pregled

- Biološki uvid u poređenje sekvenci
- Igra poravnanja i najduža zajednička podsekvenca
- Problem turiste na Menhetnu
- Problem kusura
- Dinamičko programiranje i putokazi za povratak
- Od Menhetna do grafa poravnanja
- Od globalnog do lokalnog poravnanja
- **Kažnjavanje insercija i delecija u poravnanju sekvenci**
- Prostorno efikasno poravnanje sekvenci
- Višestruko poravnanje sekvenci

Kažnjavanje praznina

- U globalnom poravnanju je fiksna kazna σ bila dodeljena svakom indelu.
- Međutim, ova fiksna kazna može biti preoštra kod lokalnog poravnanja kada možemo imati 100 uzastopnih indela.
- Niz od k uzastopnih indela često predstavlja jedan isti evolucioni događaj, ne k različitih:

dve praznine (niži skor)	**GATCCAG** **GA–C–AG**	**GATCCAG** **GA––CAG**	jedna praznina (viši skor)

Adekvatnije kazne za praznine

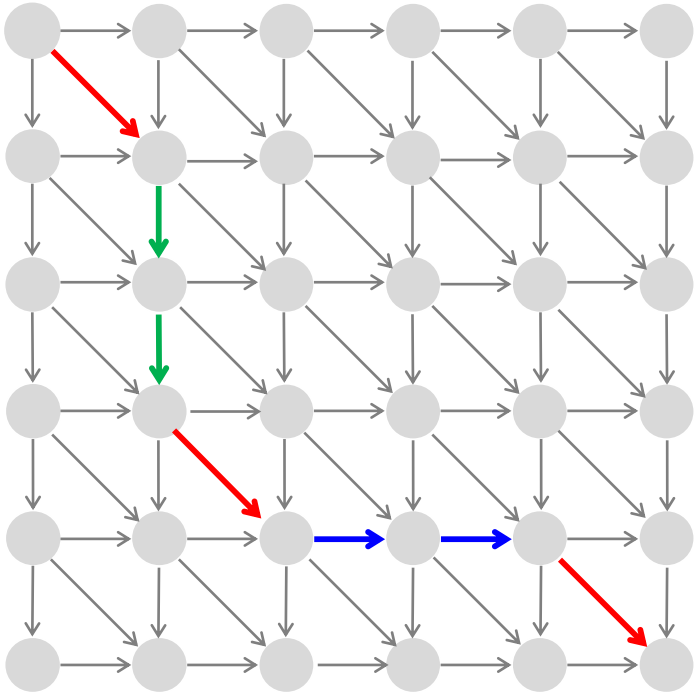
Afina kazna za praznine za prazninu dužine *k*: $\sigma+\varepsilon\cdot(k-1)$

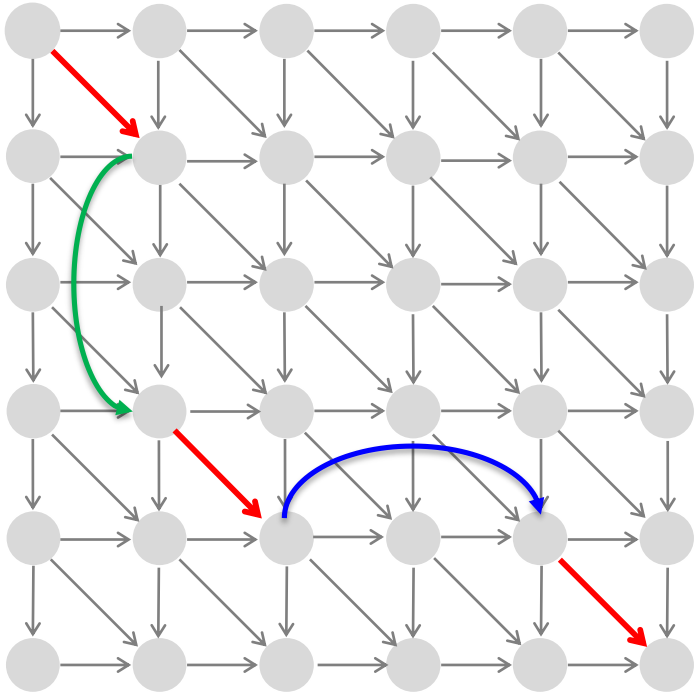
σ - kazna za **otvaranje** praznine

ε - kazna za **proširenje** praznine

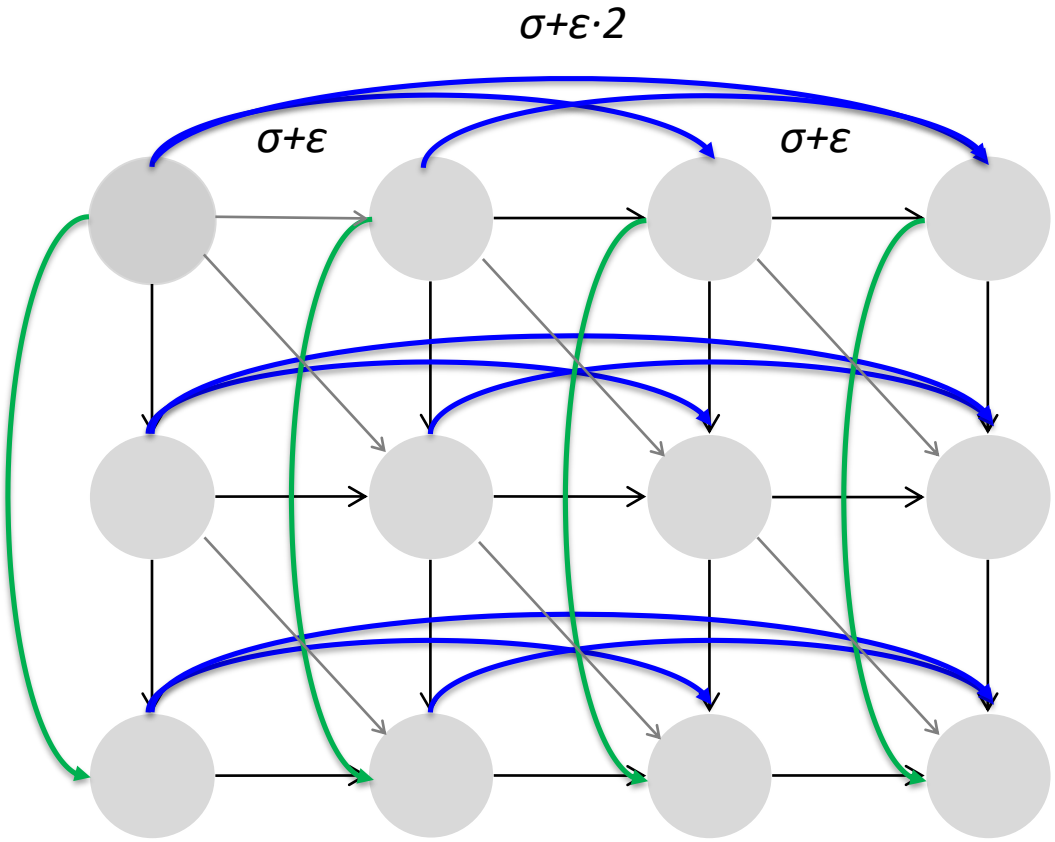
$\sigma > \varepsilon$, jer otvaranje praznine treba kazniti više nego njeno proširenje

Modelovanje afinih kazni za praznine pomoću dugih grana





Izgradnja Menhetn grafa sa afinim kaznama za praznine

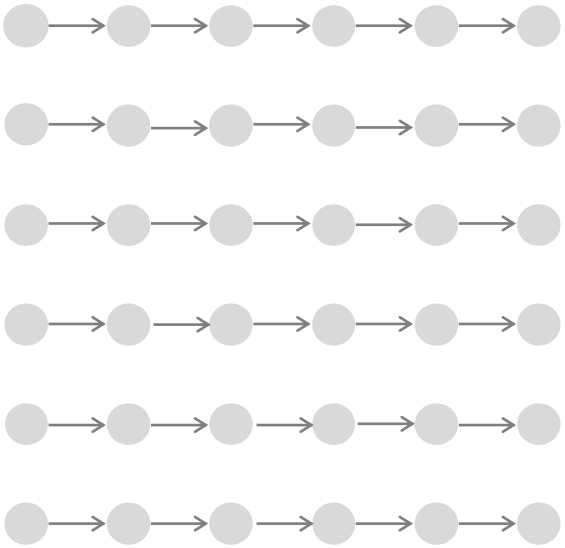


Koliko grana smo dodali?

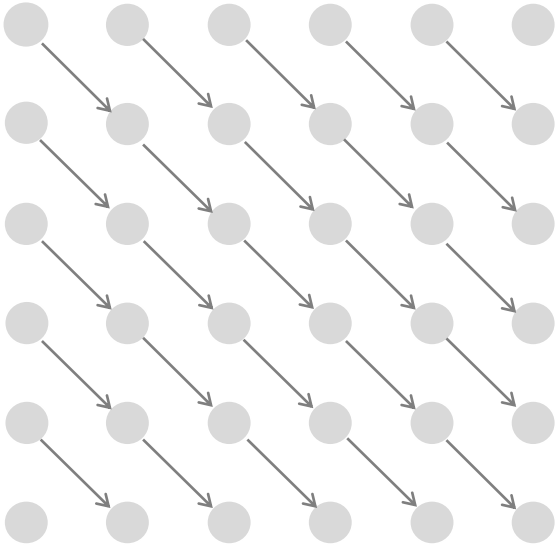
O(n^3)

- Vremenska složenost je direktno proporcionalna broju grana, zbog čega želimo da smanjimo broj grana u grafu (trenutno je O(n^3))
- Jedan način za smanjivanje broja grana je povećanje broja čvorova u grafu
- Zato delimo Menhetn graf na tri nivoa

Izgradnja Menhetn grafa na tri nivoa

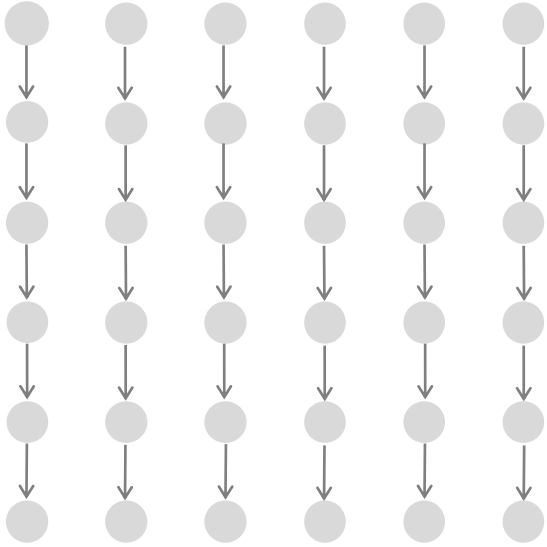


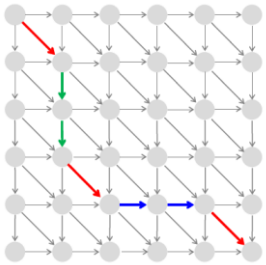
gornji nivo
(delecije)



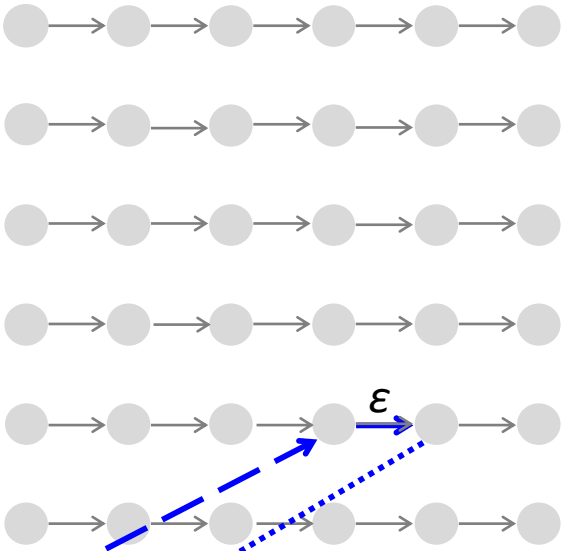
srednji nivo
(*matches*/*mismatches*)

donji nivo
(insercije)





Kako možemo da simuliramo ovu putanju na Menhetn grafu u 3 nivoa?



$lower_{i,j} = \max \{ lower_{i-1,j} - \varepsilon,\ middle_{i-1,j} - \sigma$

ε

σ

σ

0

$upper_{i,j} = \max \{ upper_{i,j-1} - \varepsilon,\ middle_{i,j-1} - \sigma$

0

$lower_{i,j}$

$middle_{i,j} = \max \{ middle_{i-1,j-1} + score(v_i, w_j)$

$upper_{i,j}$

Pregled

- Biološki uvid u poređenje sekvenci
- Igra poravnanja i najduža zajednička podsekvenca
- Problem turiste na Menhetnu
- Problem kusura
- Dinamičko programiranje i putokazi za povratak
- Od Menhetna do grafa poravnanja
- Od globalnog do lokalnog poravnanja
- Kažnjavanje insercija i delecija u poravnanju sekvenci
- **Prostorno efikasno poravnanje sekvenci**
- Višestruko poravnanje sekvenci

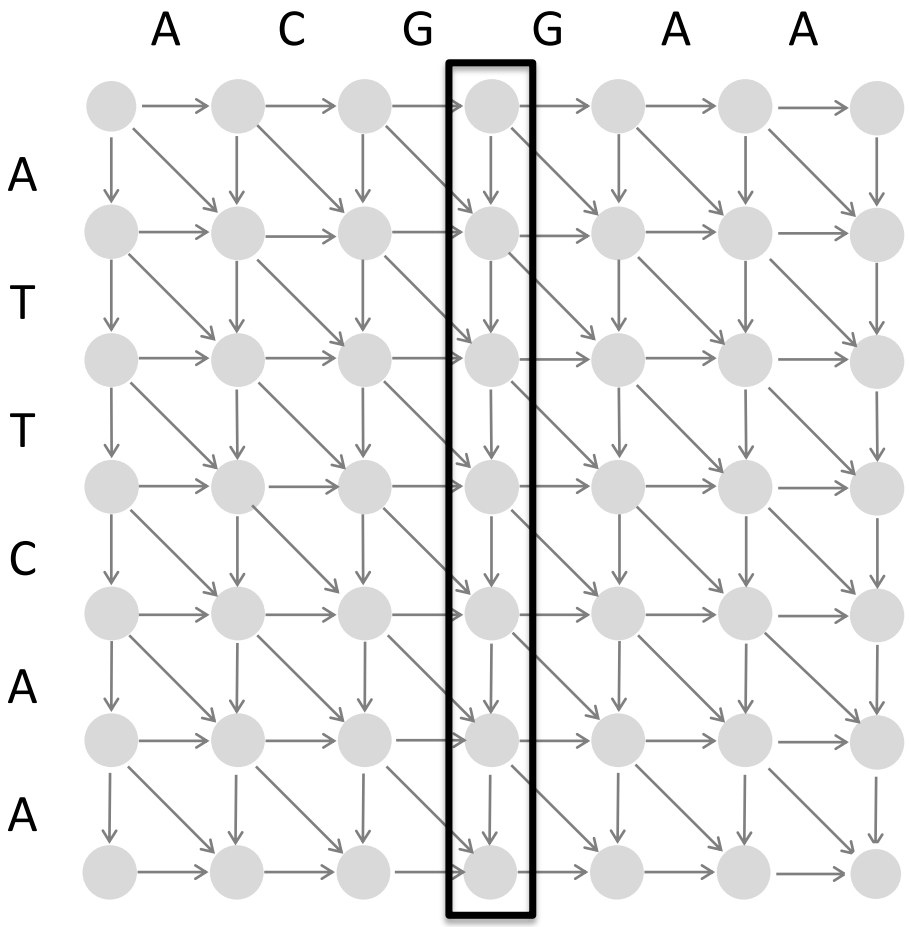
Možemo li poravnati NRP sintetaze iz dve različite bakterije?

- NRP sintentaze su obično veoma dugi proteini (≈20 000 aminokiselina)
- Vremenska složenost poravnanja: ~#edges (kvadratna)
- Prostorna složenost poravnanja: ~#nodes (kvadratna)

Memorija je često usko grlo pri poređenju dugih sekvenci

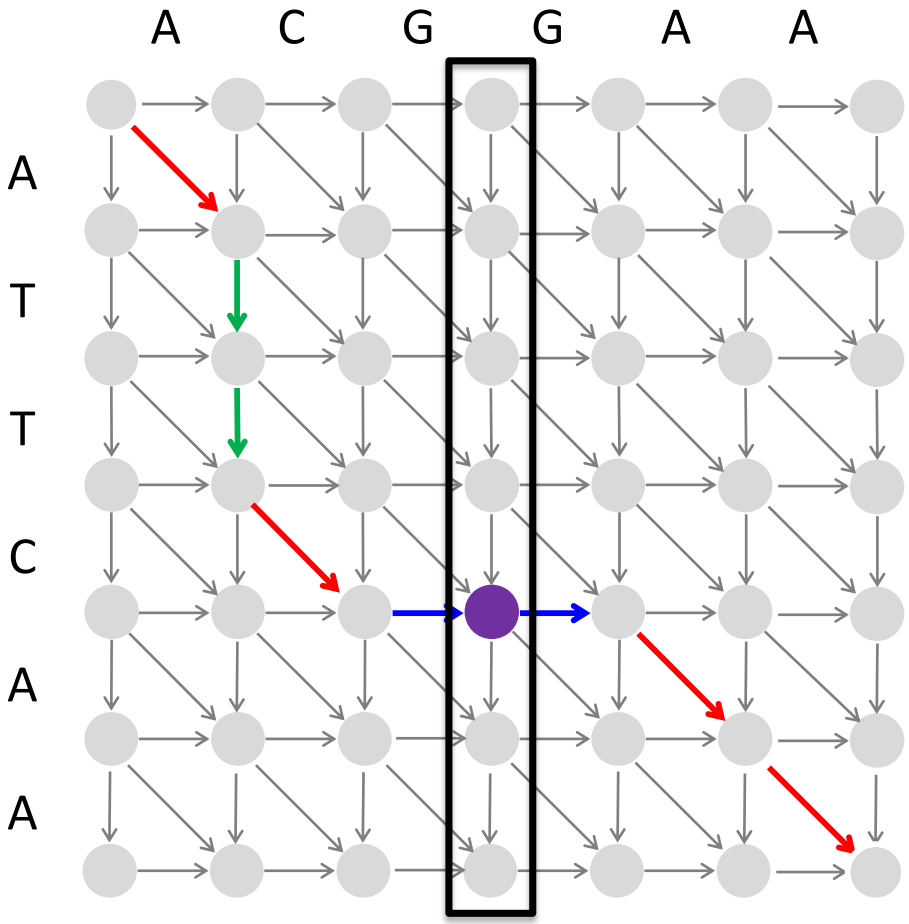
- Novi pristup: linearna prostorna složenost, udvostručena vremenska složenost (i dalje kvadratna)

Srednja kolona poravnanja



srednja kolona
(*middle*=#columns/2)

Srednji čvor poravnanja

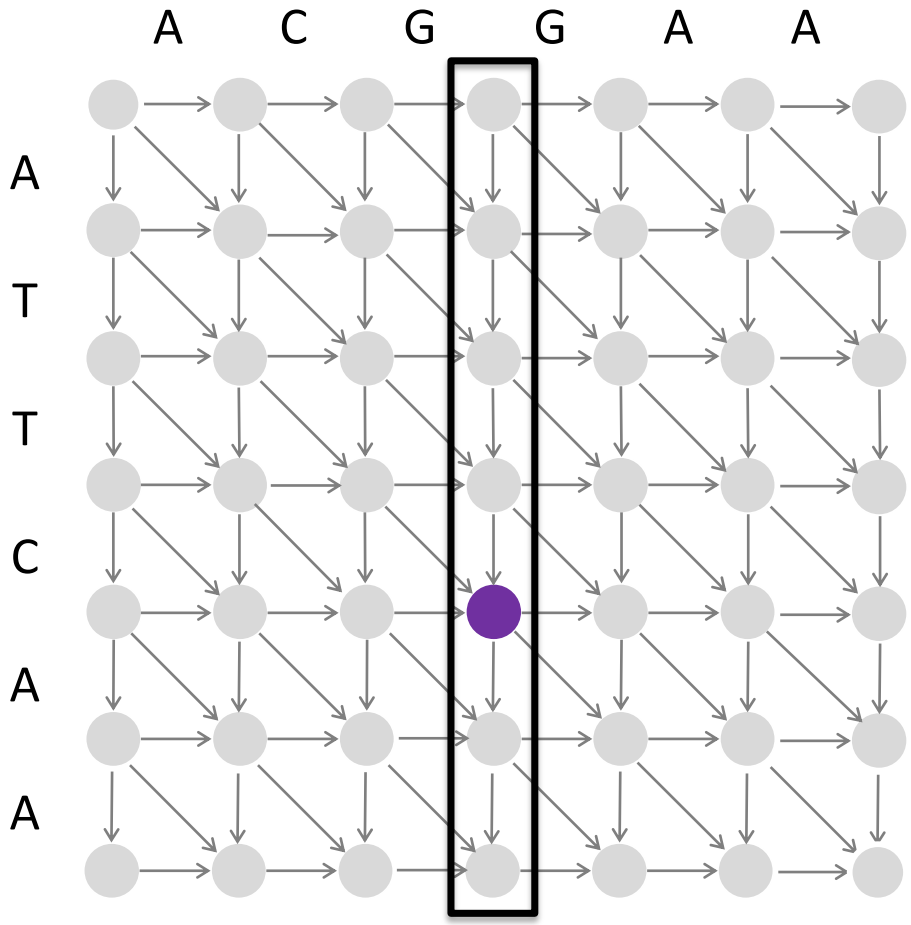


(čvor u preseku putanje optimalnog poravnanja i srednje kolone)

Algoritam *podeli-pa-vladaj* za poravnanje sekvenci

AlignmentPath(*source, sink*)

find *MiddleNode*

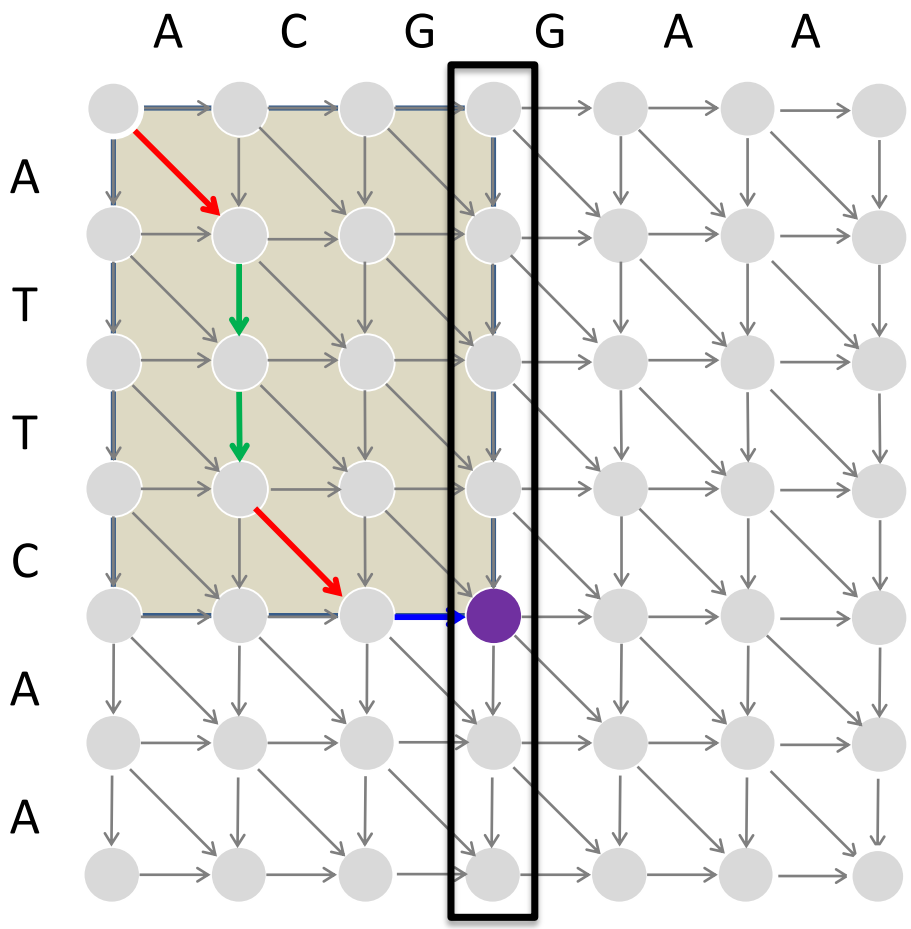


Algoritam *podeli-pa-vladaj* za poravnanje sekvenci

AlignmentPath(*source, sink*)

find *MiddleNode*

AlignmentPath(*source, MiddleNode*)



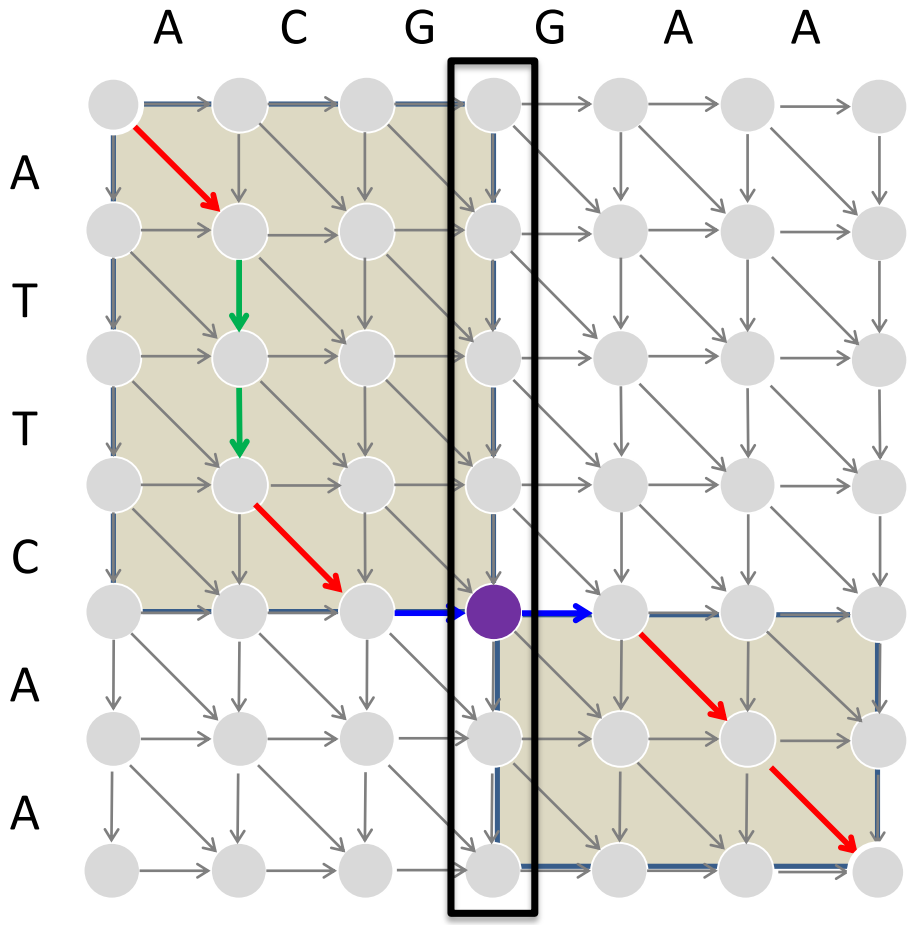
Algoritam *podeli-pa-vladaj* za poravnanje sekvenci

AlignmentPath(*source, sink*)

find *MiddleNode*

AlignmentPath(*source, MiddleNode*)

AlignmentPath(*MiddleNode, sink*)

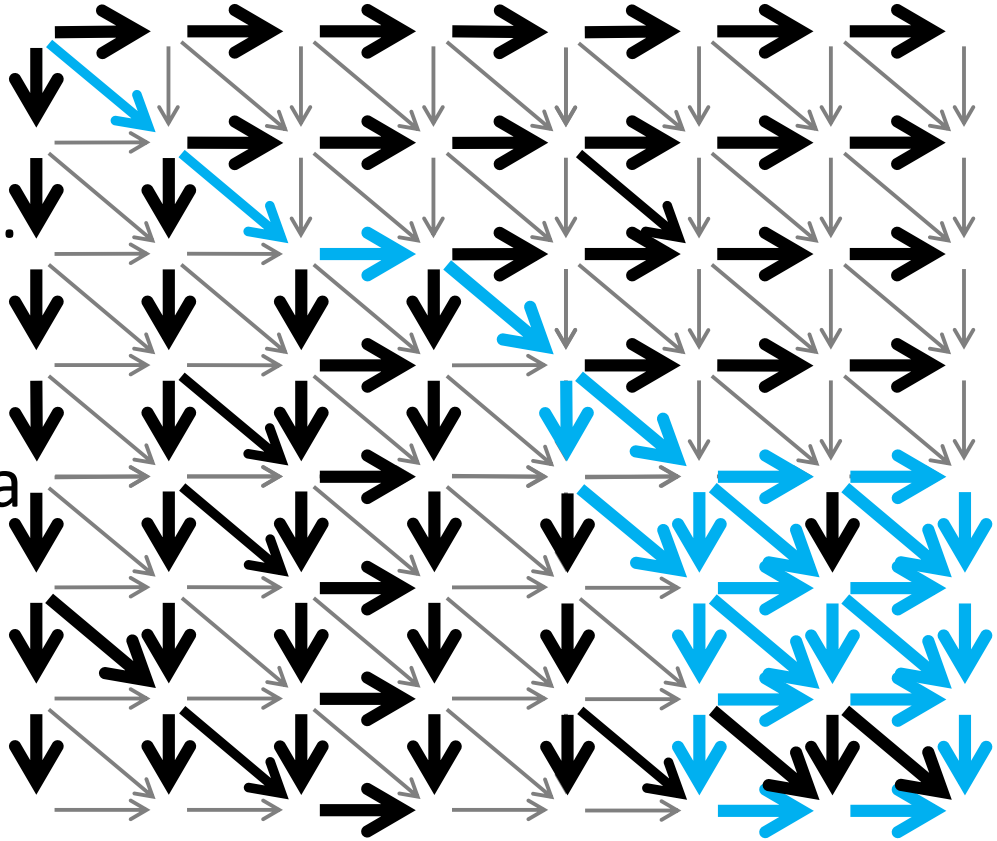


Kako naći srednji čvor u **linearnom prostoru**?

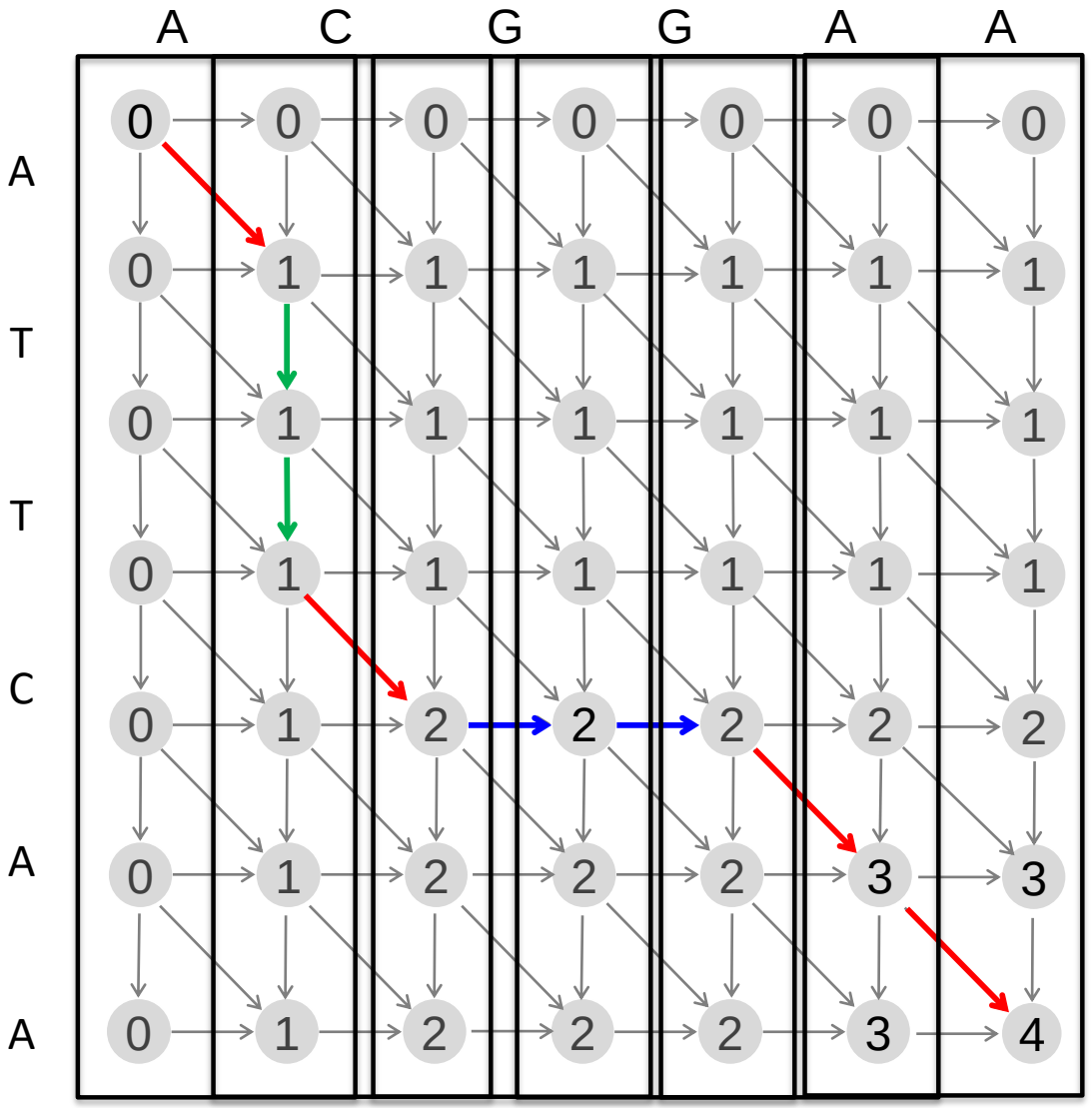
Računanje skora poravnanja u linearnom prostoru

Nalaženje **najduže putanje** u grafu poravnanja **traži** čuvanje svih putokaza za povratak – O(*nm*) prostora.

Nalaženje **dužine najduže putanje** u grafu poravnanja **ne traži** čuvanje putokaza – pokazaćemo da je potrebno O(*n*) prostora.

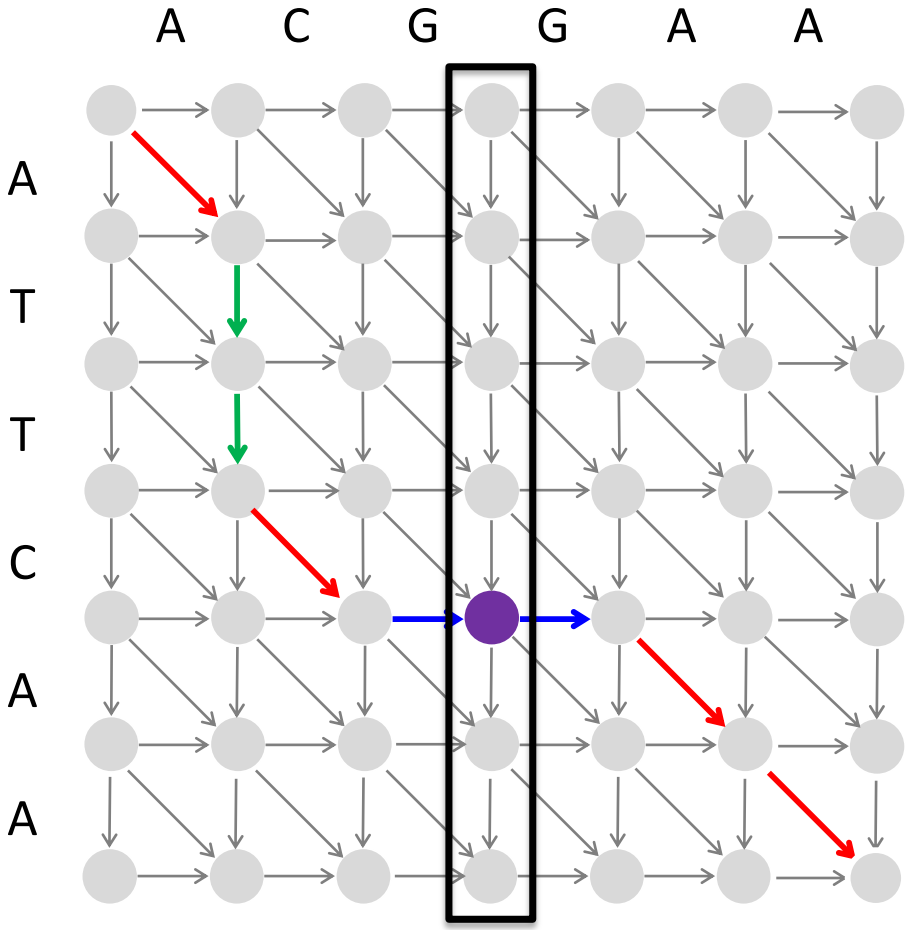


Recikliranje prostora za kolone u grafu poravnanja



- Pokazali smo da je prostor potreban za podeli-pa-vladaj poravnanje 2*n ~ O(n) => linearan
- Da vidimo kakva će biti vremenska složenost

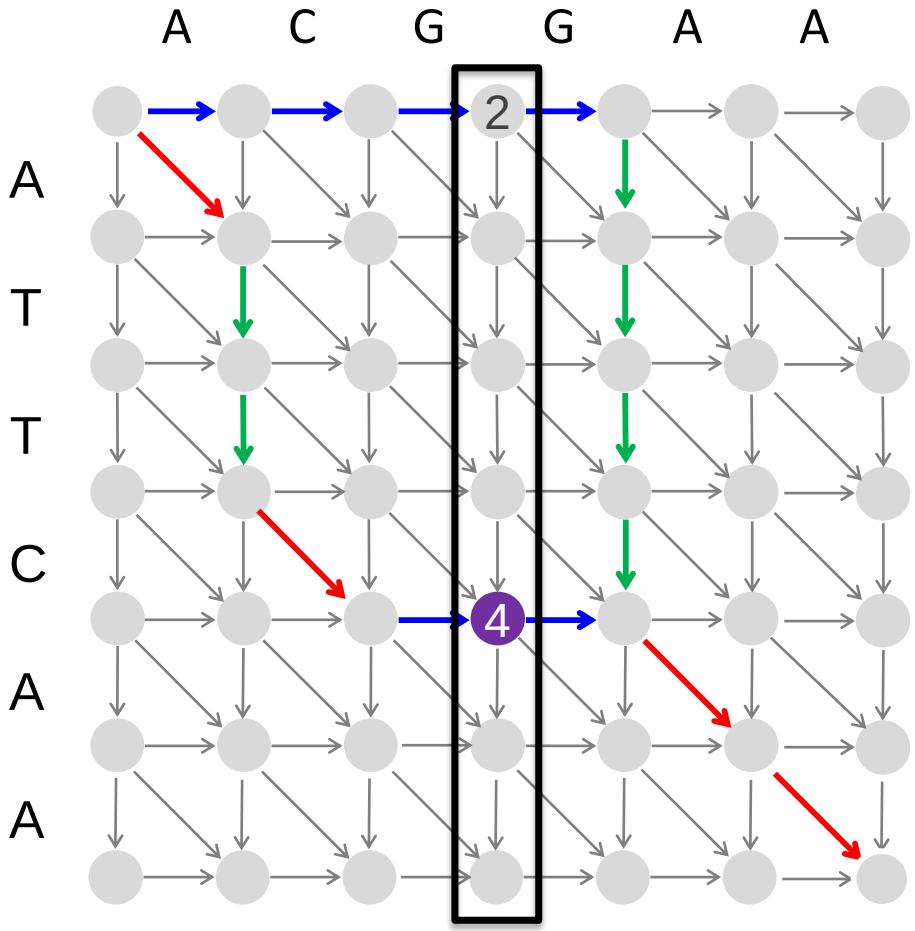
Da li možemo naći srednji čvor bez konstrukcije najduže putanje?



4-putanja posećuje čvor (4,*middle*) u srednjoj koloni

***i*-putanja** –putanja između početnog i krajnjeg čvora koja seče srednju kolonu u *i*-tom čvoru

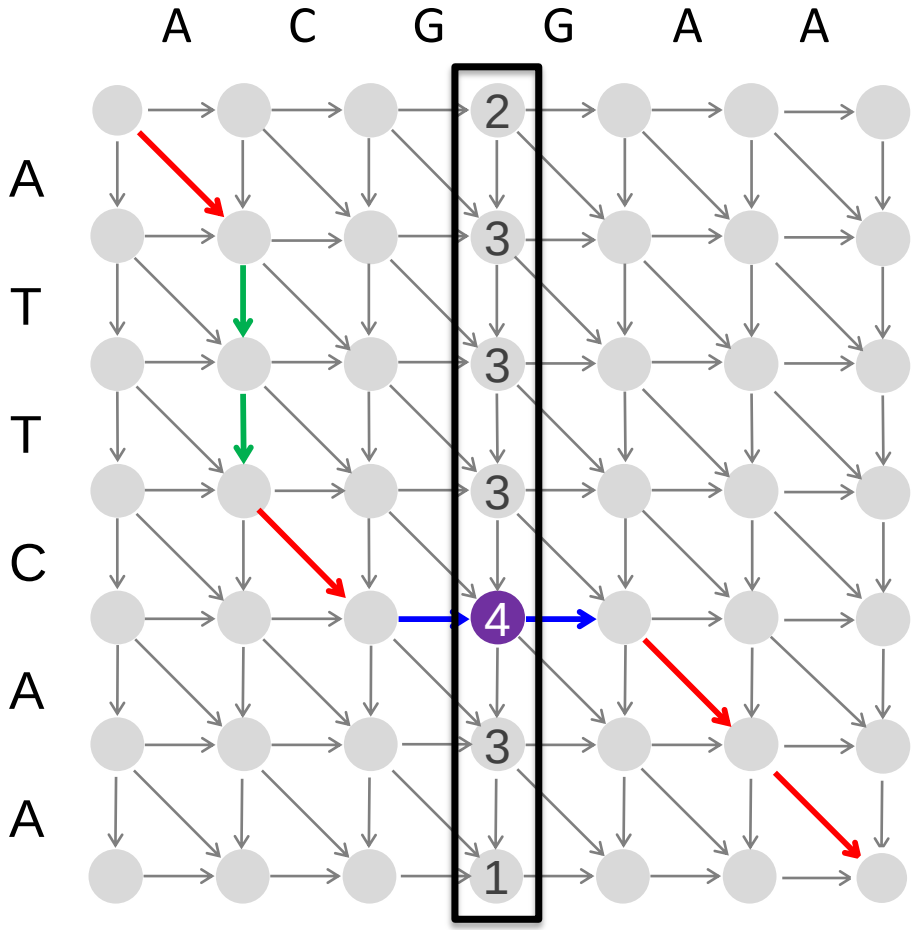
Možemo li naći dužine svih *i*-putanja bez konstrukcije najduže putanje?



length(0)=2
length(4)=4

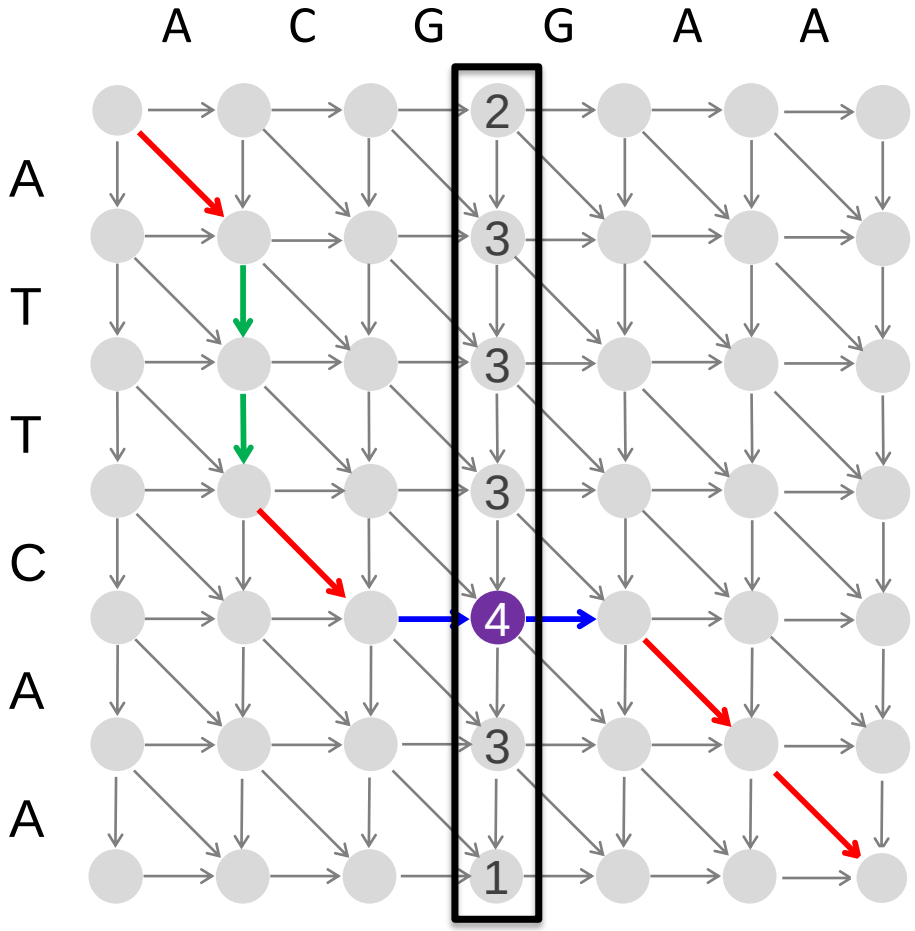
length(i) **–** dužina (skor) i-putanje

Možemo li naći dužine svih *i*-putanja bez konstrukcije najduže putanje?



Putanja sa ***maksimalnom*** **length(i)** po svim i sadrži **srednji čvor** poravnanja

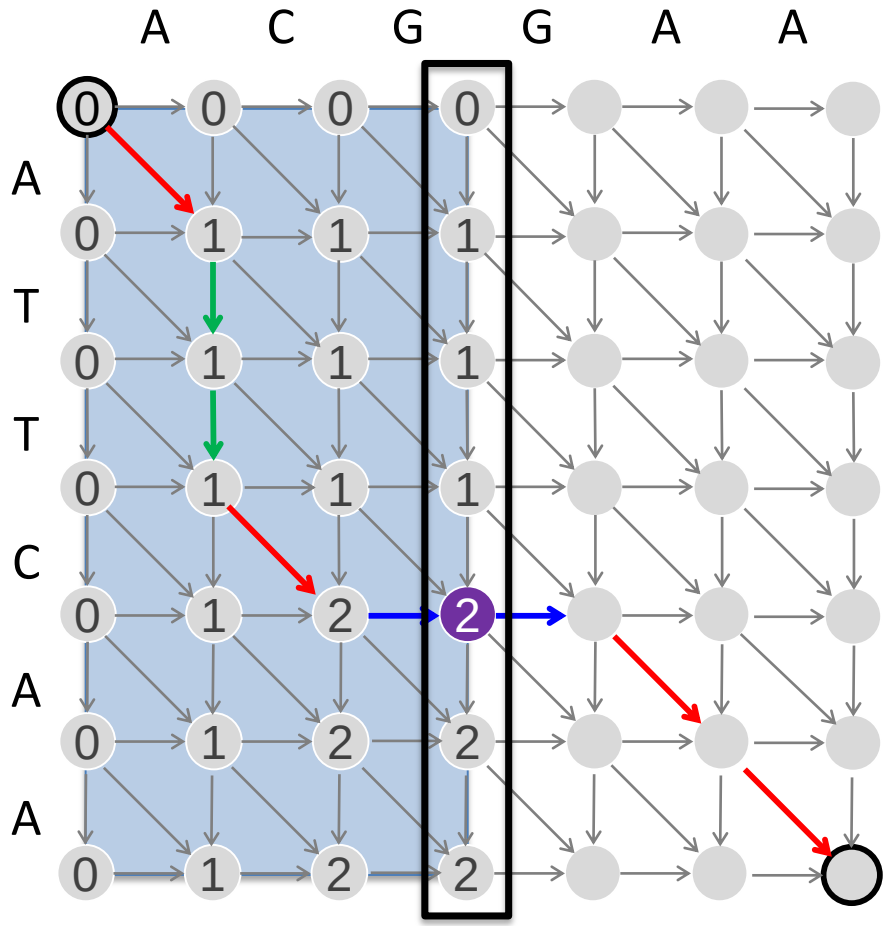
Možemo li naći dužine svih *i*-putanja bez konstrukcije najduže putanje?

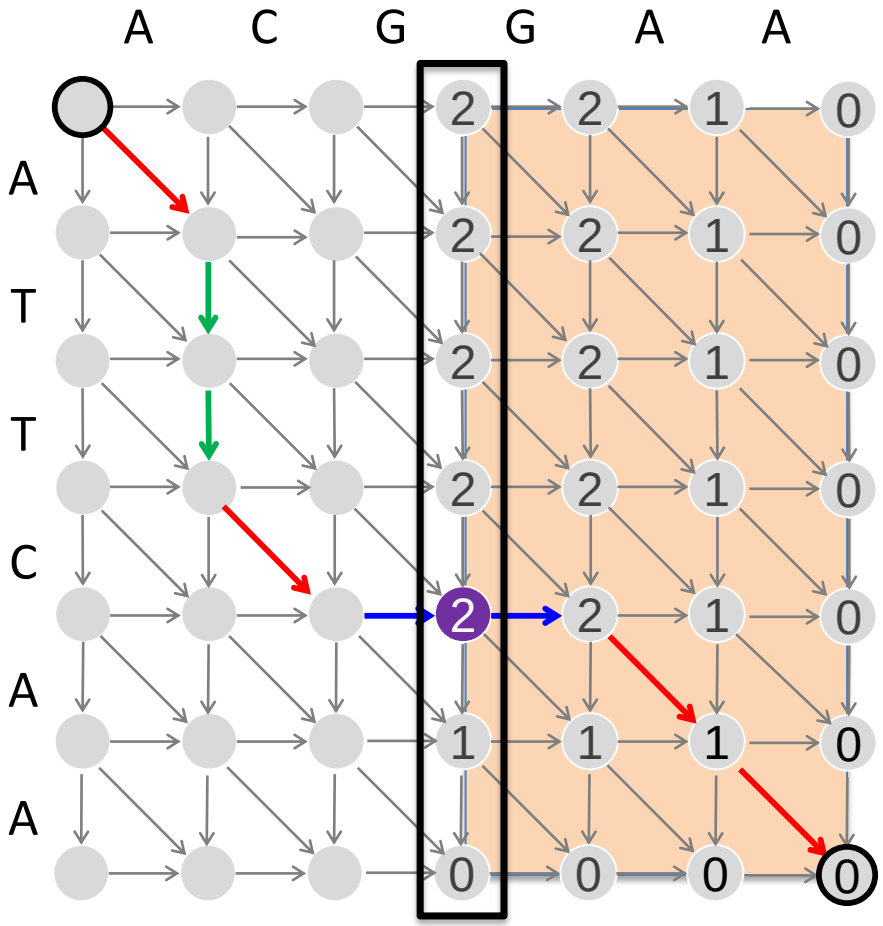


length(i):
dužina *i*-putanje

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

Računanje *FromSource* i *toSink*

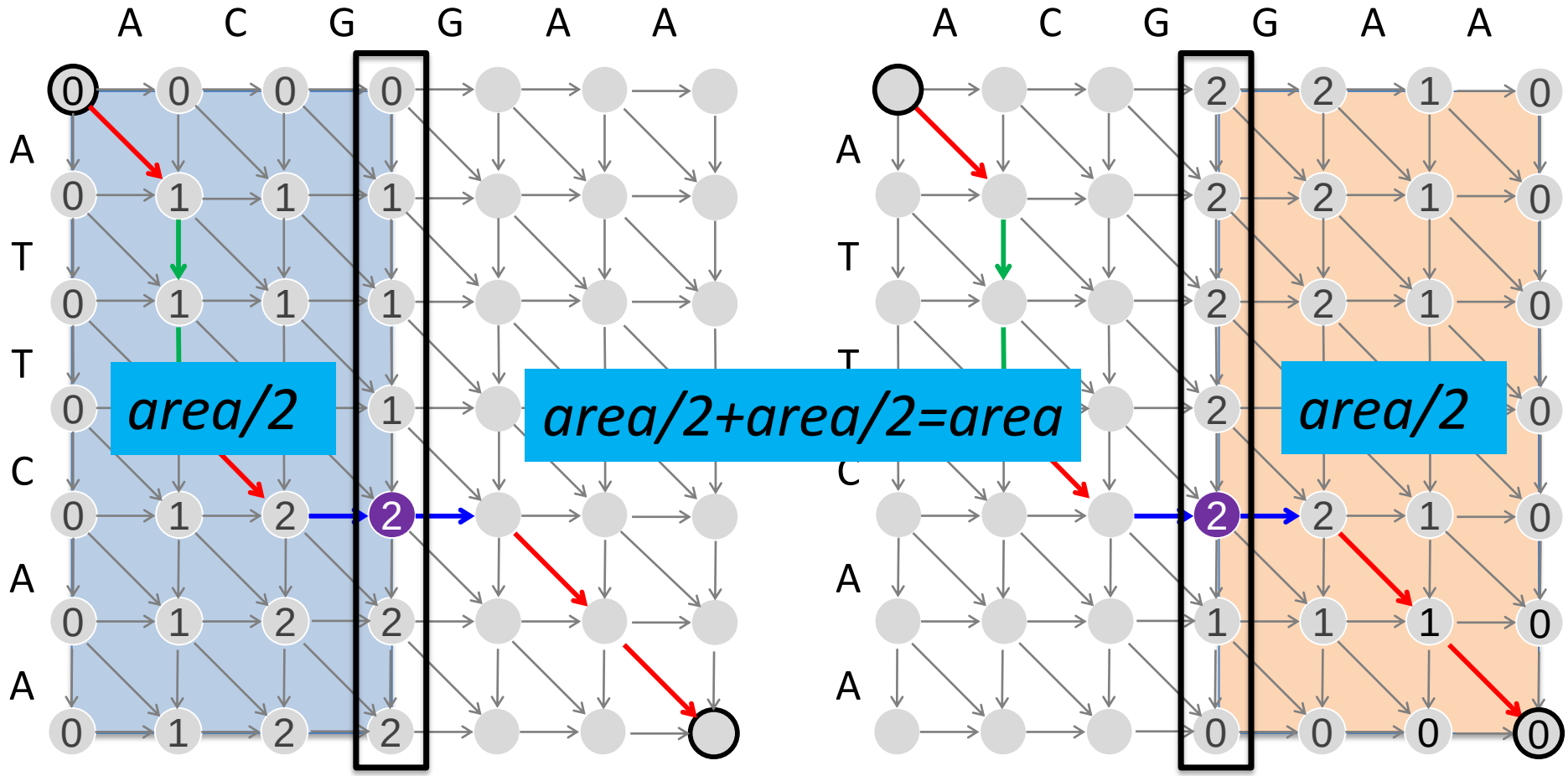




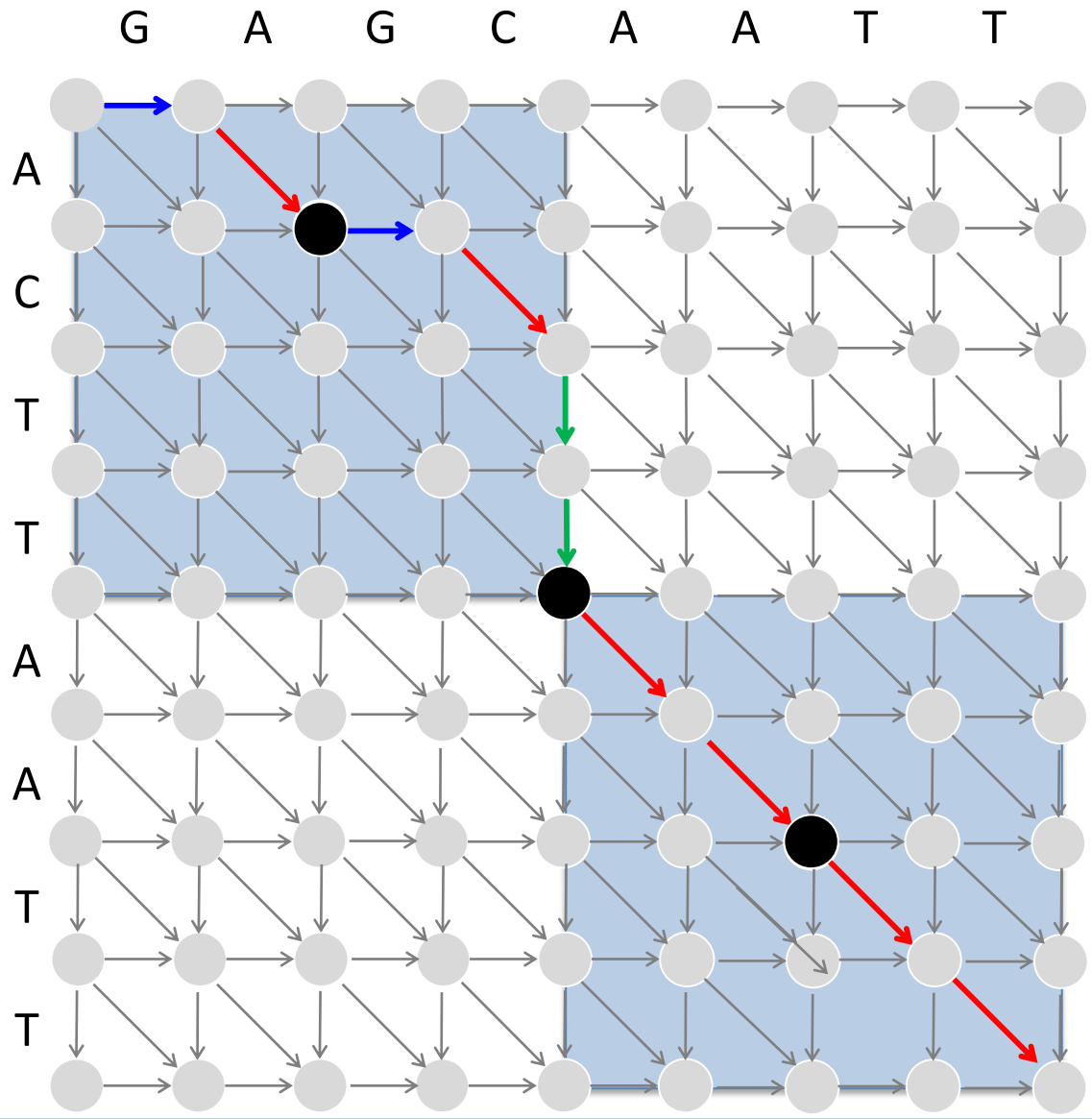
fromSource(i)

toSink(i)

Koliko vremena je potrebno za nalaženje srednjeg čvora ?



Čak O(*nm*) vremena za nalaženje samo **jednog** čvora!

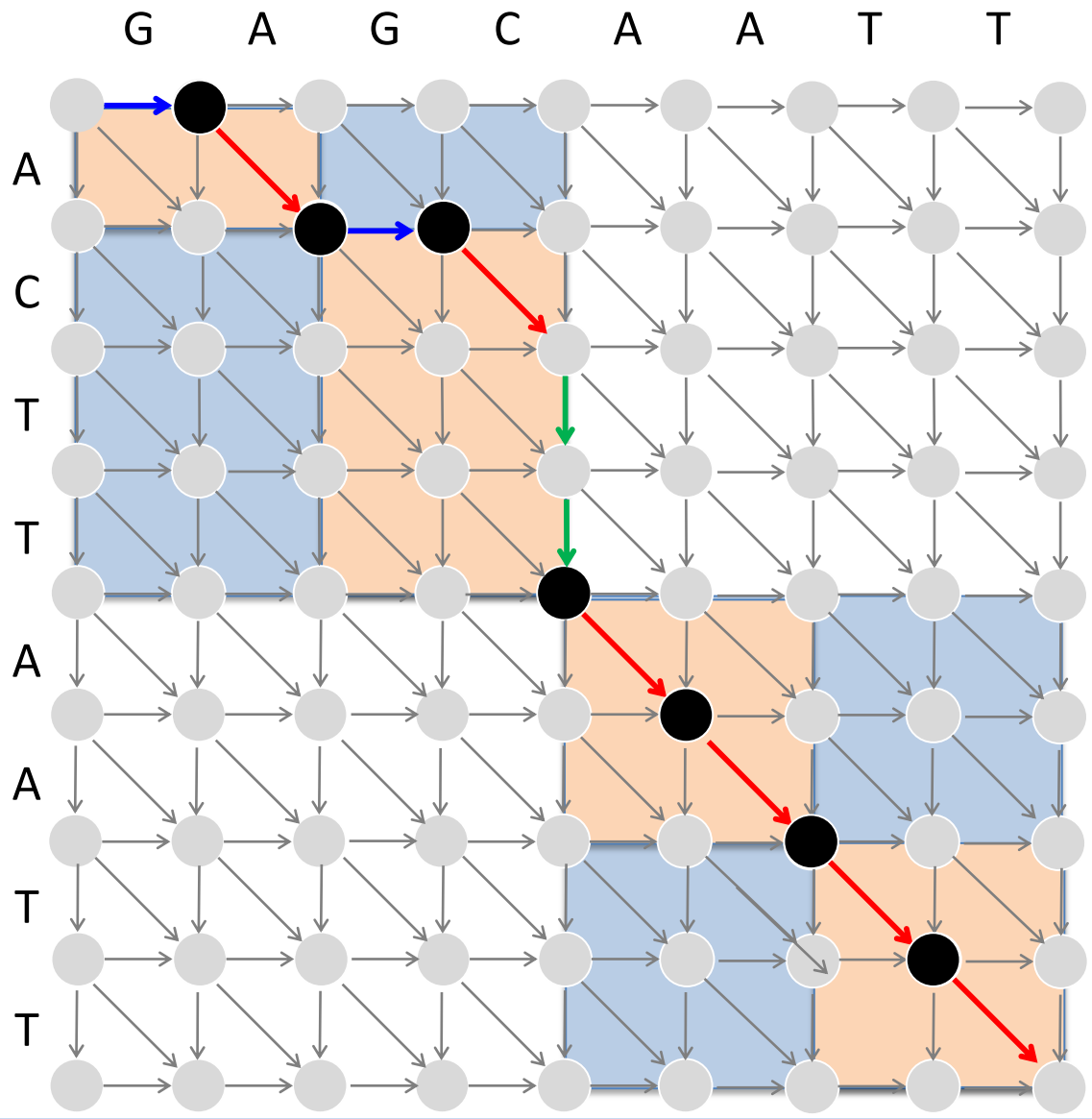


Svaki potproblem se može rešiti u vremenu proporcionalnom broju grana tj. površini koju zauzima:

area/4+area/4= ***area/2***

Koliko je vremena potrebno za rešavanje ova 2 potproblema?

Čak O(*nm+nm/2*) vremena za nalaženje **tri** čvora!

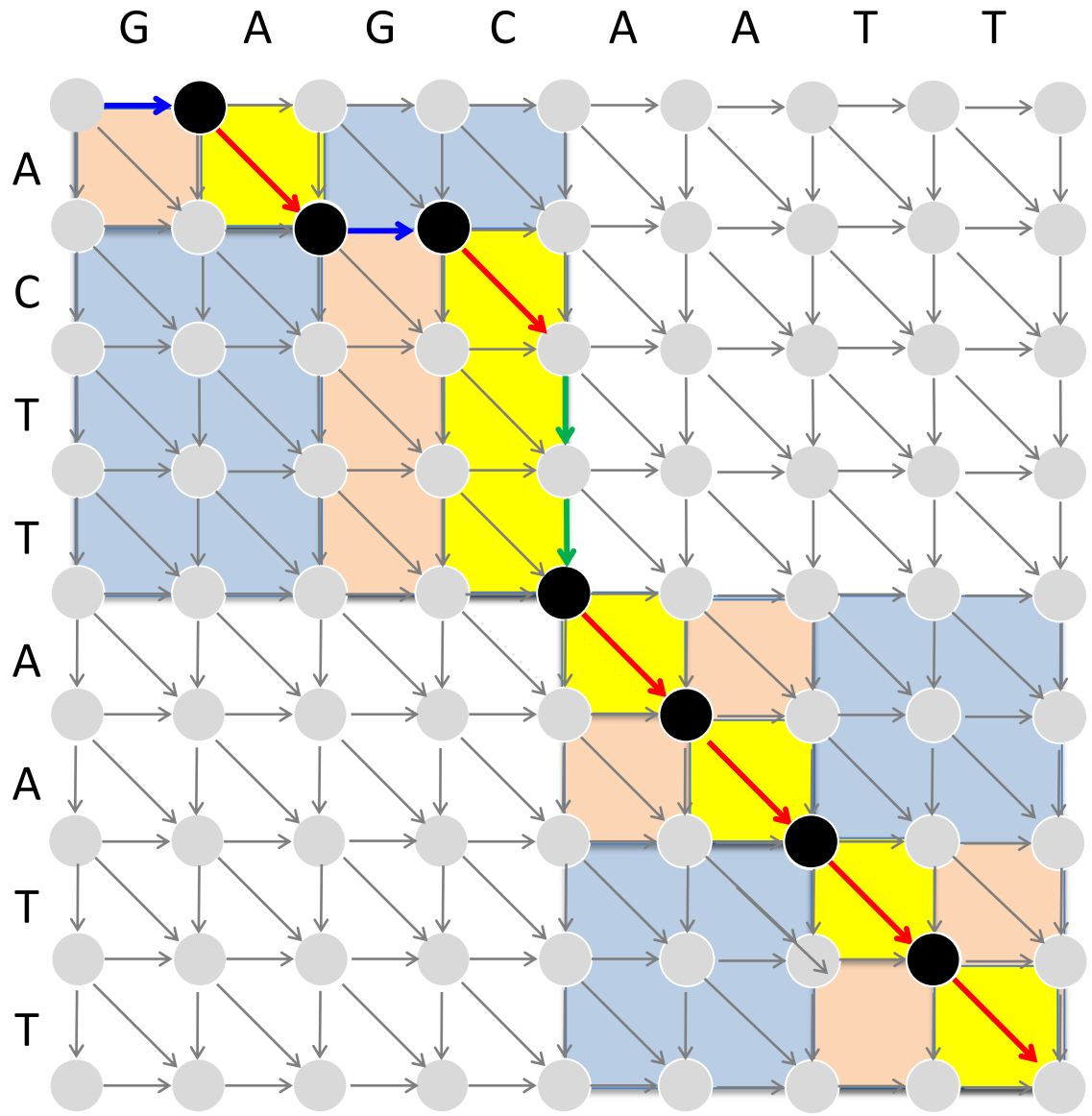


Svaki potproblem se može rešiti u vremenu proporcionalnom broju grana tj. površini koju zauzima:

area/8+area/8+ area/8+area/8= ***area/4***

Koliko je vremena potrebno za rešavanje ova 4 potproblema?

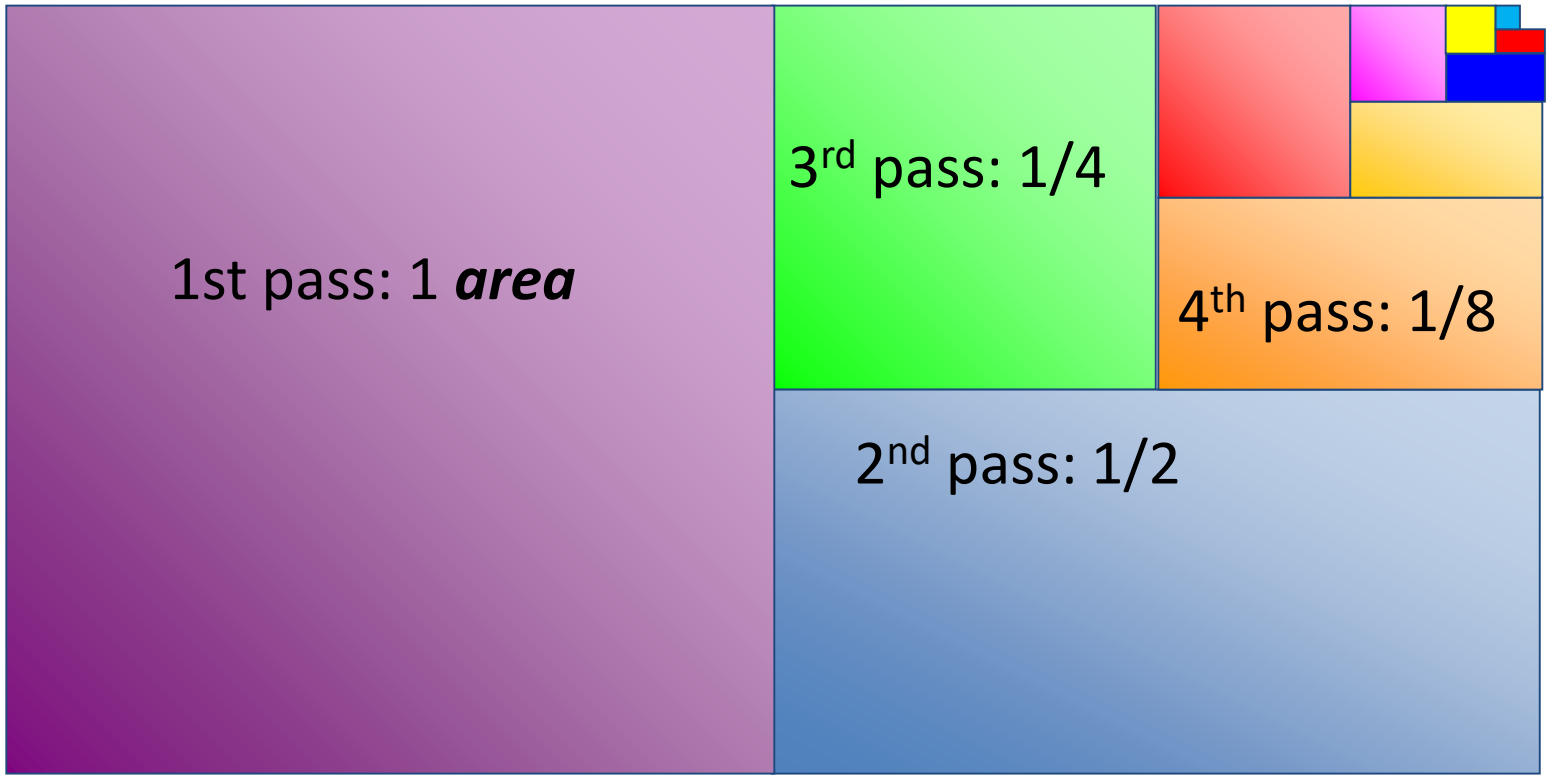
O(*nm+nm/2+nm/4*) vremena za nalaženje **skoro svih** čvorova!



area+
area/2
+area/4
+area/8
+area/16
+….+
<
2·area

Koliko je vremena potrebno za rešavanje svih potproblema?

Total Time: ***area+area/2+area/4+area/8+area/16+…***



$1 + ½ + ¼ + \ldots < 2$

- Pokazali smo da je vremenska složenost 2*m*n ~ O(m*n)

Pregled

- Biološki uvid u poređenje sekvenci
- Igra poravnanja i najduža zajednička podsekvenca
- Problem turiste na Menhetnu
- Problem kusura
- Dinamičko programiranje i putokazi za povratak
- Od Menhetna do grafa poravnanja
- Od globalnog do lokalnog poravnanja
- Kažnjavanje insercija i delecija u poravnanju sekvenci
- Prostorno efikasno poravnanje sekvenci
- **Višestruko poravnanje sekvenci**

Od dvostrukog do višestrukog poravnanja

- Do sada su u poravnanju učestvovale samo dve sekvence.
- Slaba sličnost između dve sekvence postaje značajna ako je prisutna i u drugim sekvencama
- Višestruka oravnanja mogu otkriti suptilne sličnosti koje dvostruka poravnanja ignorišu

Poravnanje tri A-domena

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

Poravnanje tri A-domena

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

Poravnanje tri A-domena

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

Generalizacija dvostrukog na višestruko poravnanje

- Poravnanje 2 sekvence je matrica od 2 reda
- Poravnanje 3 sekvence je matrica od 3 reda

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

- Funkcija skora treba da dodeljuje visok skor poravnanjima sa konzerviranim kolonama

Poravnanja = 3-D putanje

- Poravnanje sekvenci ATGC, AATC i ATGC

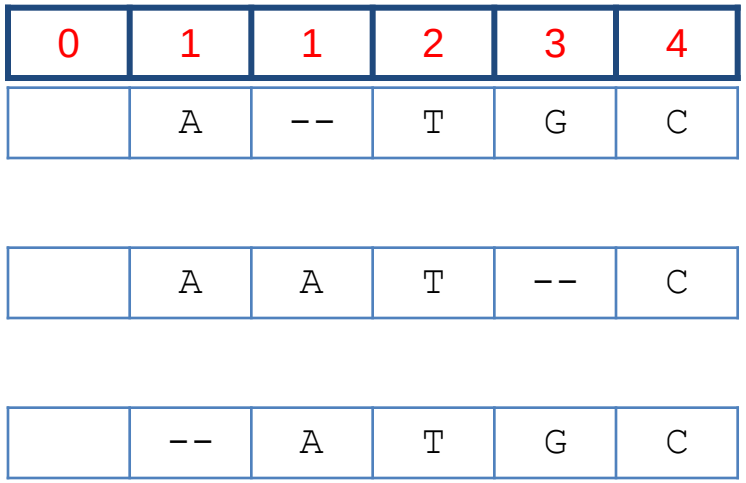
	A	--	T	G	C

	A	A	T	--	C

	--	A	T	G	C

Poravnanja = 3-D putanje

- Poravnanje sekvenci ATGC, AATC i ATGC



broj simbola do date pozicije

Poravnanja = 3-D putanje

- Poravnanje sekvenci ATGC, AATC i ATGC

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

	--	A	T	G	C

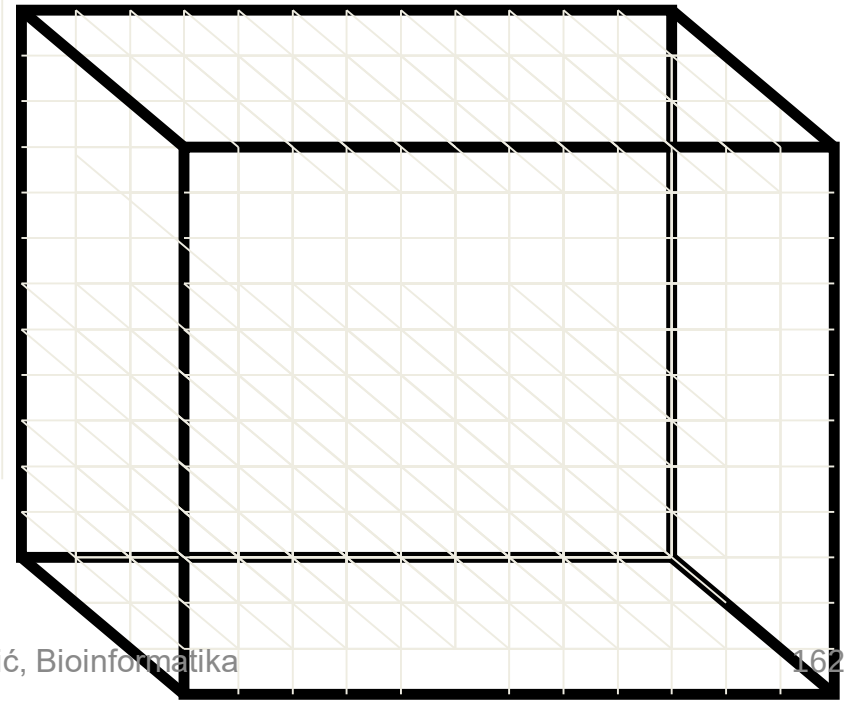
broj simbola do date pozicije

Poravnanja = 3-D putanje

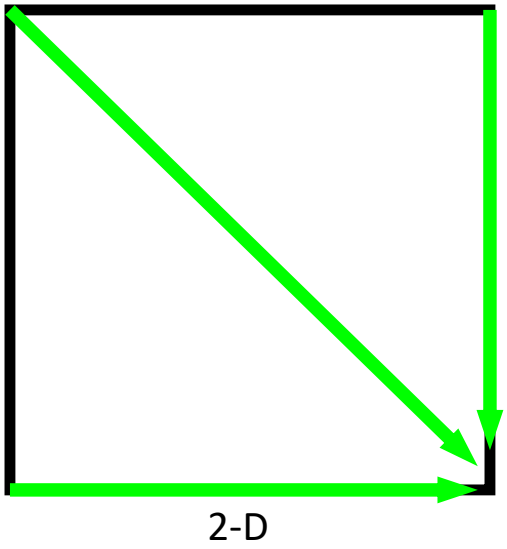
- Poravnanje sekvenci ATGC, AATC i ATGC

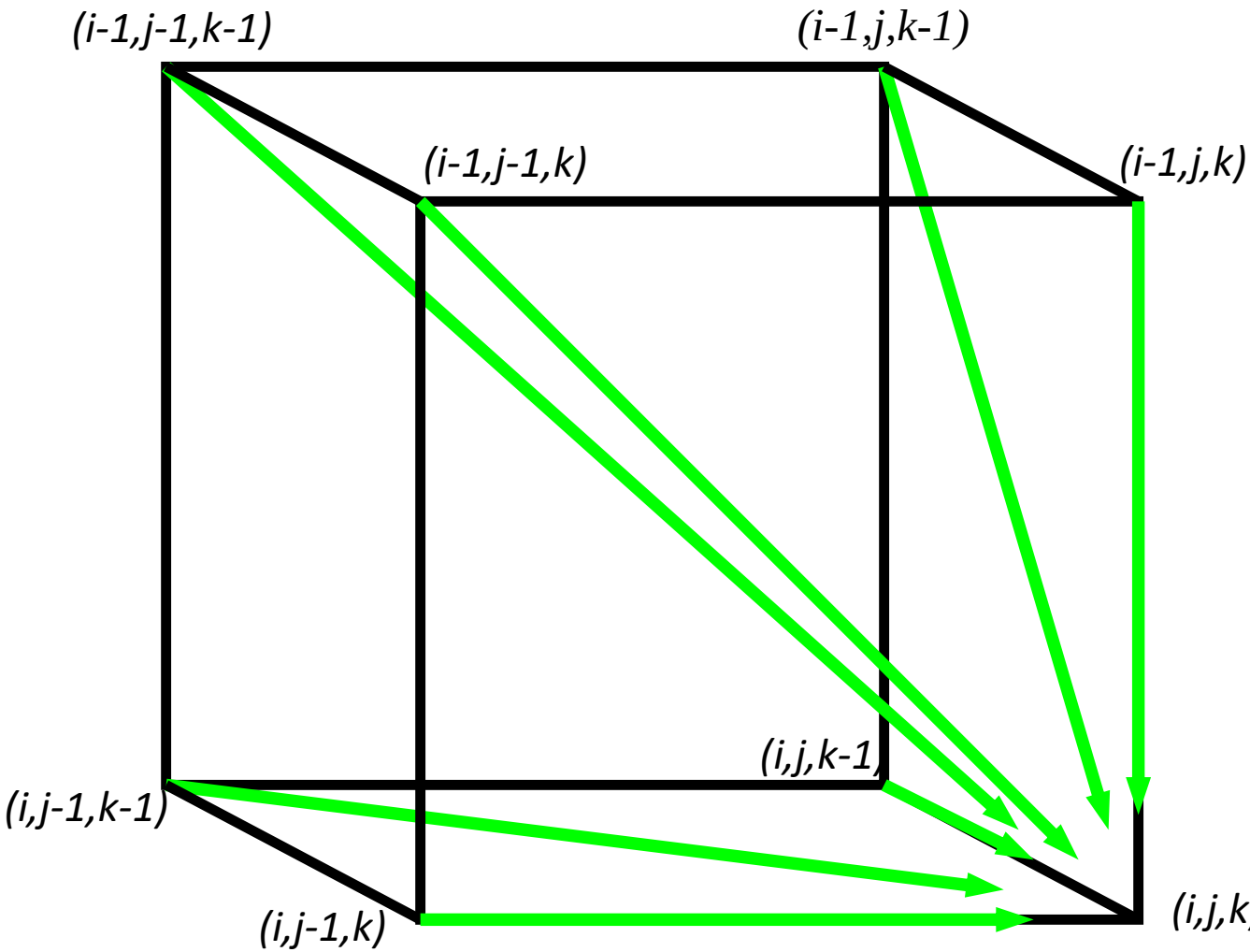
(0,0,0)→(1,1,0)→(1,2,1) →(2,3,2) →(3,3,3) →(4,4,4)

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



2-D poravnanje u odnosu na 3-D poravnanje





Rekurentna relacija dinamičkog programiranja za višestruko poravnanje

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

- $\delta(x, y, z)$ - element 3-D matrice skora

Vremenska složenost dinamičkog algoritma za višestruko poravnanje

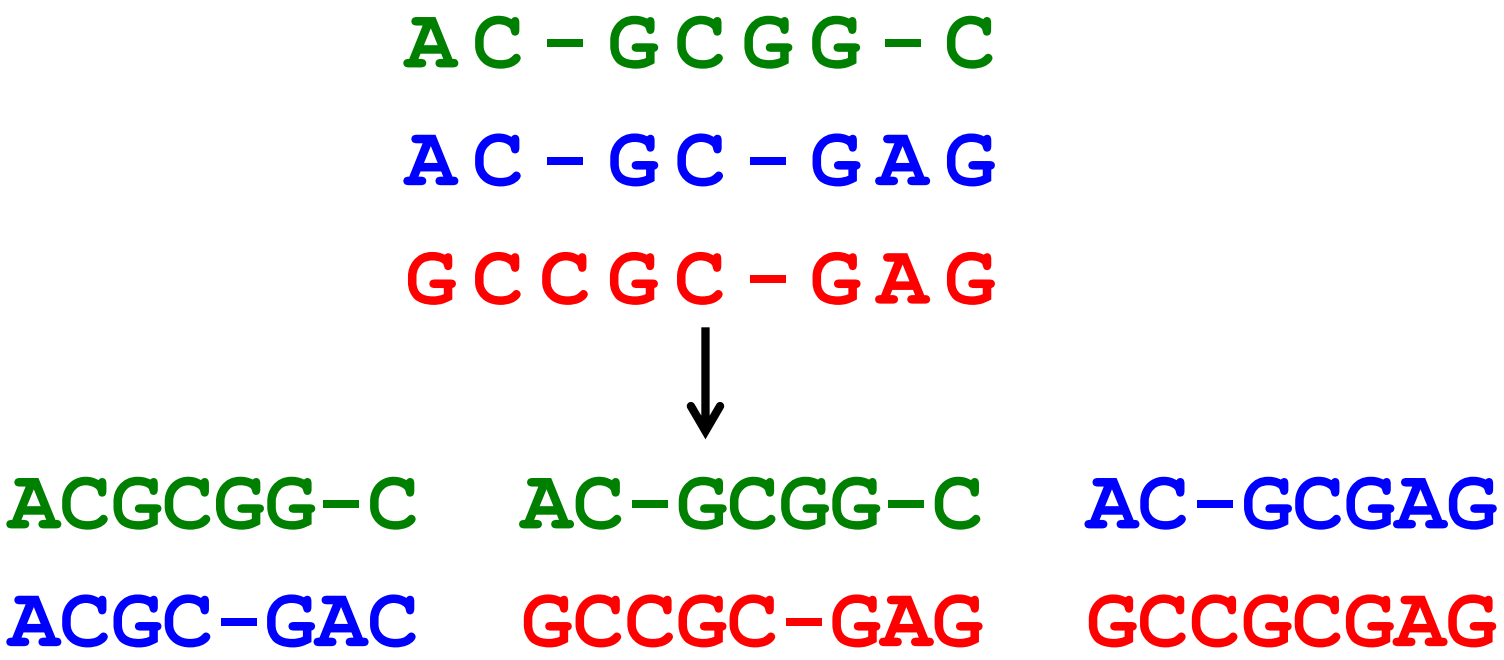
- Kao kod dvostrukog poravnanja, vremenska složenost je proporcionalna broju grana

  ~ O(#edges)

- Za 3 sekvence *n*, vremenska složenost je proporcionalna $7n^3$

- Za poravnanje *k* sekvenci, potrebno je izgraditi *k*-dimenzionalni Menhetn graf sa:
  - n^k čvorova
  - većina čvorova će imati $2^k - 1$ ulaznih grana.
  - Vremenska složenost: O($2^k n^k$)

Višestruko poravnanje uključuje i dvostruko poravnanje



Da li se višestruko poravnanje može izgraditi iz dvostrukog?

Za dati skup **optimalnih** dvostrukih poravnanja za 3 sekvence, možemo li odgovarajuće konstruisati višestruko poravnanje?

```
AAAATTTT----      ----AAAATTTT      TTTTGGGG----
----TTTTGGGG      GGGGAAAA----      ----GGGGAAAA
```

Profilna reprezentacija višestrukog poravnanja

	-	A	G	G	C	T	A	T	C	A	C	C	T	G
	T	A	G	–	C	T	A	C	C	A	-	-	-	G
	C	A	G	–	C	T	A	C	C	A	-	-	-	G
	C	A	G	–	C	T	A	T	C	A	C	–	G	G
	C	A	G	–	C	T	A	T	C	G	C	–	G	G
A	0	1	0	0	0	0	1	0	0	.8	0	0	0	0
C	.6	0	0	0	1	0	0	.4	1	0	.6	.2	0	0
G	0	0	1	.2	0	0	0	0	0	.2	0	0	.4	1
T	.2	0	0	0	0	1	0	.6	0	0	0	0	.2	0
-	.2	0	0	.8	0	0	0	0	0	0	.4	.8	.4	0

Poravnanje sekvence u odnosu na sekvencu

- Do sada smo poravnavali **sekvencu u odnosu na sekvencu**.

	-	A	G	G	C	T	A	T	C	A	C	C	T	G
	T	A	G	–	C	T	A	C	C	A	-	-	-	G
	C	A	G	–	C	T	A	C	C	A	-	-	-	G
	C	A	G	–	C	T	A	T	C	A	C	–	G	G
	C	A	G	–	C	T	A	T	C	G	C	–	G	G
A	0	1	0	0	0	0	1	0	0	.8	0	0	0	0
C	.6	0	0	0	1	0	0	.4	1	0	.6	.2	0	0
G	0	0	1	.2	0	0	0	0	0	.2	0	0	.4	1
T	.2	0	0	0	0	1	0	.6	0	0	0	0	.2	0
-	.2	0	0	.8	0	0	0	0	0	0	.4	.8	.4	0

Poravnanje sekvence u odnosu na profil

- Do sada smo poravnavali **sekvencu u odnosu na sekvencu**.
  - Možemo li poravnati **sekvencu u odnosu na profil**?

	-	A	G	G	C	T	A	T	C	A	C	C	T	G
	T	A	G	–	C	T	A	C	C	A	-	-	-	G
	C	A	G	–	C	T	A	C	C	A	-	-	-	G
	C	A	G	–	C	T	A	T	C	A	C	–	G	G
	C	A	G	–	C	T	A	T	C	G	C	–	G	G
A	0	1	0	0	0	0	1	0	0	.8	0	0	0	0
C	.6	0	0	0	1	0	0	.4	1	0	.6	.2	0	0
G	0	0	1	.2	0	0	0	0	0	.2	0	0	.4	1
T	.2	0	0	0	0	1	0	.6	0	0	0	0	.2	0
-	.2	0	0	.8	0	0	0	0	0	0	.4	.8	.4	0

Poravnanje profila u odnosu na profil

- Do sada smo poravnavali **sekvencu u odnosu na sekvencu**.
  - Možemo li poravnati **sekvencu u odnosu na profil**?
  - Možemo li poravnati **profil u odnosu na profil**?

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

A         0  1  0  0  0  0  1  0  0 .8  0  0  0  0
C        .6  0  0  0  1  0  0 .4  1  0 .6 .2  0  0
G         0  0  1 .2  0  0  0  0  0 .2  0  0 .4  1
T        .2  0  0  0  0  1  0 .6  0  0  0  0 .2  0
-        .2  0  0 .8  0  0  0  0  0  0 .4 .8 .4  0
```

Višestruko poravnanje: pohlepni pristup

- Izabrati najsličnije sekvence i kombinovati ih u profil
- Tako bismo smanjili broj sekvenci sa *k* na *k-2* i jedan profil
- Iterirati

Pohlepni pristup: primer

- Sekvence: GATTCA, GTCTGA, GATATT, GTCAGC.

- 6 dvostrukih poravnanja (match +1, indels i mismatches -1)

```
s2  GTCTGA
s4  GTCAGC   (score = 2)

s1  GAT-TCA
s2  G-TCTGA  (score = 1)

s1  GAT-TCA
s3  GATAT-T  (score = 1)

s1  GATTCA--
s4  G-T-CAGC (score = 0)

s2  G-TCTGA
s3  GATAT-T  (score = -1)

s3  GAT-ATT
s4  G-TCAGC  (score = -1)
```

Pohlepni pristup: primer

- Pošto su s_2 i s_4 najbliže, od njih pravimo profil:

$s2$ GTCTGA
$s4$ GTCAGC
} $s_{2,4}$ = GTCt/aGa/c

- Novi skup od 3 sekvence za poravnanje:

s_1 GATTCA
s_3 GATATT
$s_{2,4}$ GTCt/aGa/c

- Slajdovi pokrivaju poglavlje 5 knjige *Bioinformatics Algorithms: an Active Learning Approach*
- Sadržaj slajdova je preuzet sa zvaničnih prezentacija autora i dodatno prilagođen