Metódy v bioinformatike

CB #01 Informatika pre biológov

Jana Černíková

FMFI UK

26/09/2024

Dnešné ciele

1. Pojem problému, algoritmu
2. Ukážka prevodu biologického problému na informatický
3. Efektivita algoritmu, pojem časovej zložitosti, O-notácia
4. NP-ťažké algoritmy

Formulácia problému, algoritmus

- *Formulácia problému*: jasne definované vstupné a výstupné dáta a aký výstup očakávame pre každý vstup.
- Formulácia neuvádza *akým spôsobom* sa majú zo vstupov vypočítať výstupy.
- *Správny algoritmus*: Postup, ktorý určuje *spôsob*, akým pre každý vstup vypočítame príslušný výstup.

Biologický problém

Pomocou hmotnostného spektrometra (mass spectrometer) sme odmerali vo vzorke peptid s hmotnosťou K. Máme databázu proteínov a chceme zistiť, ktorý z proteínov obsahuje peptid s touto hmotnosťou.

Informatický problém

Vstup je postupnosť n kladných čísel $a[1], a[2], \ldots, a[n]$ a číslo K. Nájdite súvislý úsek tejto postupnosti $a[i], a[i+1], \ldots, a[j]$, ktorý svojim súčtom dáva číslo K.

Príklad

```
K=19
3 4 6 3 6 4 9 2 8
    ^^^^^^^
```

Otázka na zamyslenie

Informatický problém

Vstup je postupnosť n kladných čísel $a[1], a[2], \ldots, a[n]$ a číslo K. Nájdite súvislý úsek tejto postupnosti $a[i], a[i+1], \ldots, a[j]$, ktorý svojim súčtom dáva číslo K.

Príklad

```
K=19
3 4 6 3 6 4 9 2 8
    ^^^^^^^
```

Ako túto úlohu vyriešiť?

Informatický problém

Vstup je postupnosť n kladných čísel $a[1], a[2], \ldots, a[n]$ a číslo K. Nájdite súvislý úsek tejto postupnosti $a[i], a[i+1], \ldots, a[j]$, ktorý svojim súčtom dáva číslo K.

Triviálne riešenie

Skúšame všetky možnosti

```
pre každé i od 1 po n
|   pre každé j od i po n
|   |   suma := 0;
|   |   pre každé u od i po j
|   |   |  suma := suma + a[u]
|   |   ak suma = K, vypíš i,j
```

```
K=19
3 4 6 3 6 4 9 2 8
    i     j
```

Ako dlho takýto program pobeží?

- Naimplementovať do počítača a odmerať
- Na akom počítači? Na akých vstupoch?
- *Časová zložitosť* - počet operácií, ktoré program vykoná, v závislosti od množstva dát.
- Pre každú veľkosť vstupu *odhadneme najhorší možný prípad*

```
pre každé i od 1 po n
|   pre každé j od i po n
|   |   suma := 0;
|   |   pre každé u od i po j
|   |   |  suma := suma + a[u]
|   |   ak suma = K, vypíš i,j
```

Výpočet časovej zložitosti

```
pre každé i od 1 po n
|   pre každé j od i po n
|   |   suma := 0;
|   |   pre každé u od i po j
|   |   |  suma := suma + a[u]
|   |   ak suma = K, vypíš i,j
```

Počet operácií := a +

$$T(n) = \sum_{i=1}^{n} \left(\sum_{j=i}^{n} \left(1 + \sum_{u=i}^{j} 2 \right) \right) = \ldots = \frac{1}{6}n^3 - n^2 + \frac{5}{6}n$$

Zaujíma nás *najvýznamnejší* člen tejto sumy, a to je $\frac{1}{6}n^3$. Navyše, nezaujíma nás konštanta pri tom člene. Výsledok takéhoto "zjednodušenia" píšeme ako $O(n^3)$ a hovoríme, že daný algoritmus má *kubickú časovú zložitosť*.

Prečo používame O-notáciu?

Úlohou O-notácie je odpovedať na otázky typu "ak budem mať X krát viac dát, koľkokrát dlhšie budem čakať na výsledok?"

Napríklad, $T(10^5) = \frac{1}{6} \cdot 10^{15} - 10^{10} + \frac{5}{6} \cdot 10^5 = 166656666750000$

$T(2 \cdot 10^5) = \frac{1}{6} \cdot 2^3 \cdot 10^{15} - 2^2 \cdot 10^{10} + 2 \cdot \frac{5}{6} \cdot 10^5 = 1333293333500000$

$$\frac{1333293333500000}{166656666750000} = 8.000240011400564$$

Tento výpočet môžem spraviť len s najvýznamnejšími členmi:
$\frac{\frac{1}{6} \cdot 2^3 \cdot 10^{15}}{\frac{1}{6} \cdot 10^{15}} = 2^3 = 8$. Všimnite si, že na konštante $\frac{1}{6}$ nezáleží.

Prečo používame O-notáciu?

Teda, namiesto porovnávania presných funkcií stačí porovnať ich pomocou "zjednodušených" funkcií:

$$\frac{T(X \cdot n)}{T(n)} \approx \frac{(X \cdot n)^3}{n^3} = \frac{X^3 \cdot n^3}{n^3} = X^3$$

Ak by časová zložitosť bola napríklad $O(2^n)$, tak by zmena času behu algoritmu vyzerala následovne:

$$\frac{2^{X \cdot n}}{2^n} = 2^{(X-1) \cdot n}$$

Všimnite si, že pri exponenciálnej zložitosti nárast závisí nielen od X, ale aj od pôvodnej veľkosti vstupu n.

Merania

		$O(n)$	$O(n^2)$	$O(n^3)$	$O(2^n)$
Čas na	10	ε	ε	ε	ε
vyriešenie	50	ε	ε	ε	2 weeks
problému	100	ε	ε	ε	2800 univ.
veľkosti	1000	ε	0.02s	4.5s	—
...	10000	ε	2.1s	75m	—
	100000	0.04s	3.5m	52d	—
	1 mil.	0.42s	5.8h	142yr	—
	10 mil.	4.2s	24.3d	140000yr	—
Max veľkosť	1s	2.3 mil.	6900	610	33
problému	1m	140 mil.	53000	2400	39
vyriešená za	1d	200 bil.	2 mil.	26000	49
Zvýšenie	$+1$	—	—	—	$\times 2$
času so	$\times 2$	$\times 2$	$\times 4$	$\times 8$	—
zvýšeným n					

Efektívnejší algoritmus

Informatický problém

Vstup je postupnosť n kladných čísel $a[1], a[2], \ldots, a[n]$ a číslo K. Nájdite súvislý úsek tejto postupnosti $a[i], a[i+1], \ldots, a[j]$, ktorý svojim súčtom dáva číslo K.

Skúsme počítať sumy $a[i] + \ldots + a[j]$ rýchlejšie.

- Nech $S[i] = a[1] + a[2] + \ldots + a[i]$, $S[0] = 0$.
- Ak hodnoty $S[i]$ poznáme, ako vieme rýchlo zrátať súčet $a[i] + \ldots + a[j]$?

Efektívnejší algoritmus

Informatický problém

Vstup je postupnosť n kladných čísel $a[1], a[2], \ldots, a[n]$ a číslo K. Nájdite súvislý úsek tejto postupnosti $a[i], a[i+1], \ldots, a[j]$, ktorý svojim súčtom dáva číslo K.

Skúsme počítať sumy $a[i] + \ldots + a[j]$ rýchlejšie.

- Nech $S[i] = a[1] + a[2] + \ldots + a[i]$, $S[0] = 0$.
- Ak hodnoty $S[i]$ poznáme, ako vieme rýchlo zrátať súčet $a[i] + \ldots + a[j]$? $a[i] + \ldots + a[j] = S[j] - S[i-1]$

Efektívnejší algoritmus

Informatický problém

Vstup je postupnosť n kladných čísel $a[1], a[2], \ldots, a[n]$ a číslo K. Nájdite súvislý úsek tejto postupnosti $a[i], a[i+1], \ldots, a[j]$, ktorý svojim súčtom dáva číslo K.

Skúsme počítať sumy $a[i] + \ldots + a[j]$ rýchlejšie.

- Nech $S[i] = a[1] + a[2] + \ldots + a[i]$, $S[0] = 0$.
- Ak hodnoty $S[i]$ poznáme, ako vieme rýchlo zrátať súčet $a[i] + \ldots + a[j]$? $a[i] + \ldots + a[j] = S[j] - S[i-1]$
- Ako vieme spočítať hodnoty $S[i]$?

Efektívnejší algoritmus

Informatický problém

Vstup je postupnosť n kladných čísel $a[1], a[2], \ldots, a[n]$ a číslo K. Nájdite súvislý úsek tejto postupnosti $a[i], a[i+1], \ldots, a[j]$, ktorý svojim súčtom dáva číslo K.

Skúsme počítať sumy $a[i] + \ldots + a[j]$ rýchlejšie.

- Nech $S[i] = a[1] + a[2] + \ldots + a[i]$, $S[0] = 0$.
- Ak hodnoty $S[i]$ poznáme, ako vieme rýchlo zrátať súčet $a[i] + \ldots + a[j]$? $a[i] + \ldots + a[j] = S[j] - S[i-1]$
- Ako vieme spočítať hodnoty $S[i]$?

```
S[0] := 0
pre každé i od 1 po n:
|    S[i] := S[i-1] + a[i]
```

Efektívnejší algoritmus

Informatický problém

Vstup je postupnosť n kladných čísel $a[1], a[2], \ldots, a[n]$ a číslo K. Nájdite súvislý úsek tejto postupnosti $a[i], a[i+1], \ldots, a[j]$, ktorý svojim súčtom dáva číslo K.

Skúsme počítať sumy $a[i] + \ldots + a[j]$ rýchlejšie.

- Nech $S[i] = a[1] + a[2] + \ldots + a[i]$, $S[0] = 0$.
- Ak hodnoty $S[i]$ poznáme, ako vieme rýchlo zrátať súčet $a[i] + \ldots + a[j]$? $a[i] + \ldots + a[j] = S[j] - S[i-1]$
- Ako vieme spočítať hodnoty $S[i]$?

```
S[0] := 0
pre každé i od 1 po n:
|    S[i] := S[i-1] + a[i]
```

- Akú má časovú zložitosť výpočet $S[i]$?

Efektívnejší algoritmus

Informatický problém

Vstup je postupnosť n kladných čísel $a[1], a[2], \ldots, a[n]$ a číslo K. Nájdite súvislý úsek tejto postupnosti $a[i], a[i+1], \ldots, a[j]$, ktorý svojim súčtom dáva číslo K.

Skúsme počítať sumy $a[i] + \ldots + a[j]$ rýchlejšie.

- Nech $S[i] = a[1] + a[2] + \ldots + a[i]$, $S[0] = 0$.
- Ak hodnoty $S[i]$ poznáme, ako vieme rýchlo zrátať súčet $a[i] + \ldots + a[j]$? $a[i] + \ldots + a[j] = S[j] - S[i-1]$
- Ako vieme spočítať hodnoty $S[i]$?

```
S[0] := 0
pre každé i od 1 po n:
|    S[i] := S[i-1] + a[i]
```

- Akú má časovú zložitosť výpočet $S[i]$? $O(n)$

Efektívnejší algoritmus

Informatický problém

Vstup je postupnosť n kladných čísel $a[1], a[2], \ldots, a[n]$ a číslo K. Nájdite súvislý úsek tejto postupnosti $a[i], a[i+1], \ldots, a[j]$, ktorý svojim súčtom dáva číslo K.

- Máme spočítané $S[i]$ pre všetky $i = 1, \ldots, n$ (výpočet má časovú zložitosť $O(n)$)

Efektívnejší algoritmus

Informatický problém

Vstup je postupnosť n kladných čísel $a[1], a[2], \ldots, a[n]$ a číslo K. Nájdite súvislý úsek tejto postupnosti $a[i], a[i+1], \ldots, a[j]$, ktorý svojim súčtom dáva číslo K.

- Máme spočítané $S[i]$ pre všetky $i = 1, \ldots, n$ (výpočet má časovú zložitosť $O(n)$)
- Chceme pre všetky dvojice i, j spočítať $a[i] + \ldots + a[j] = S[j] - S[i-1]$ a porovnať túto hodnotu s K

Efektívnejší algoritmus

Informatický problém

Vstup je postupnosť n kladných čísel $a[1], a[2], \ldots, a[n]$ a číslo K. Nájdite súvislý úsek tejto postupnosti $a[i], a[i+1], \ldots, a[j]$, ktorý svojim súčtom dáva číslo K.

- Máme spočítané $S[i]$ pre všetky $i = 1, \ldots, n$ (výpočet má časovú zložitosť $O(n)$)
- Chceme pre všetky dvojice i, j spočítať $a[i] + \ldots + a[j] = S[j] - S[i-1]$ a porovnať túto hodnotu s K

```
pre každé i od 1 po n
|   pre každé j od i po n
|   |   ak S[j]-S[i-1] = K, vypíš i,j
```

Efektívnejší algoritmus

Informatický problém

Vstup je postupnosť n kladných čísel $a[1], a[2], \ldots, a[n]$ a číslo K. Nájdite súvislý úsek tejto postupnosti $a[i], a[i+1], \ldots, a[j]$, ktorý svojim súčtom dáva číslo K.

- Máme spočítané $S[i]$ pre všetky $i = 1, \ldots, n$ (výpočet má časovú zložitosť $O(n)$)
- Chceme pre všetky dvojice i, j spočítať $a[i] + \ldots + a[j] = S[j] - S[i-1]$ a porovnať túto hodnotu s K

```
pre každé i od 1 po n
|   pre každé j od i po n
|   |   ak S[j]-S[i-1] = K, vypíš i,j
```

- Aká bude časová zložitosť?

Efektívnejší algoritmus

Informatický problém

Vstup je postupnosť n kladných čísel $a[1], a[2], \ldots, a[n]$ a číslo K. Nájdite súvislý úsek tejto postupnosti $a[i], a[i+1], \ldots, a[j]$, ktorý svojim súčtom dáva číslo K.

- Máme spočítané $S[i]$ pre všetky $i = 1, \ldots, n$ (výpočet má časovú zložitosť $O(n)$)
- Chceme pre všetky dvojice i, j spočítať $a[i] + \ldots + a[j] = S[j] - S[i-1]$ a porovnať túto hodnotu s K

```
pre každé i od 1 po n
|   pre každé j od i po n
|   |   ak S[j]-S[i-1] = K, vypíš i,j
```

- Aká bude časová zložitosť? kvadratická, alebo $O(n^2)$

Problém # 2: Najkratšie spoločné nadslovo

Formulácia problému

- Vstup: niekoľko reťazcov
- Výstup: najkratší reťazec, ktorý obsahuje všetky vstupné reťazce ako súvislé podreťazce

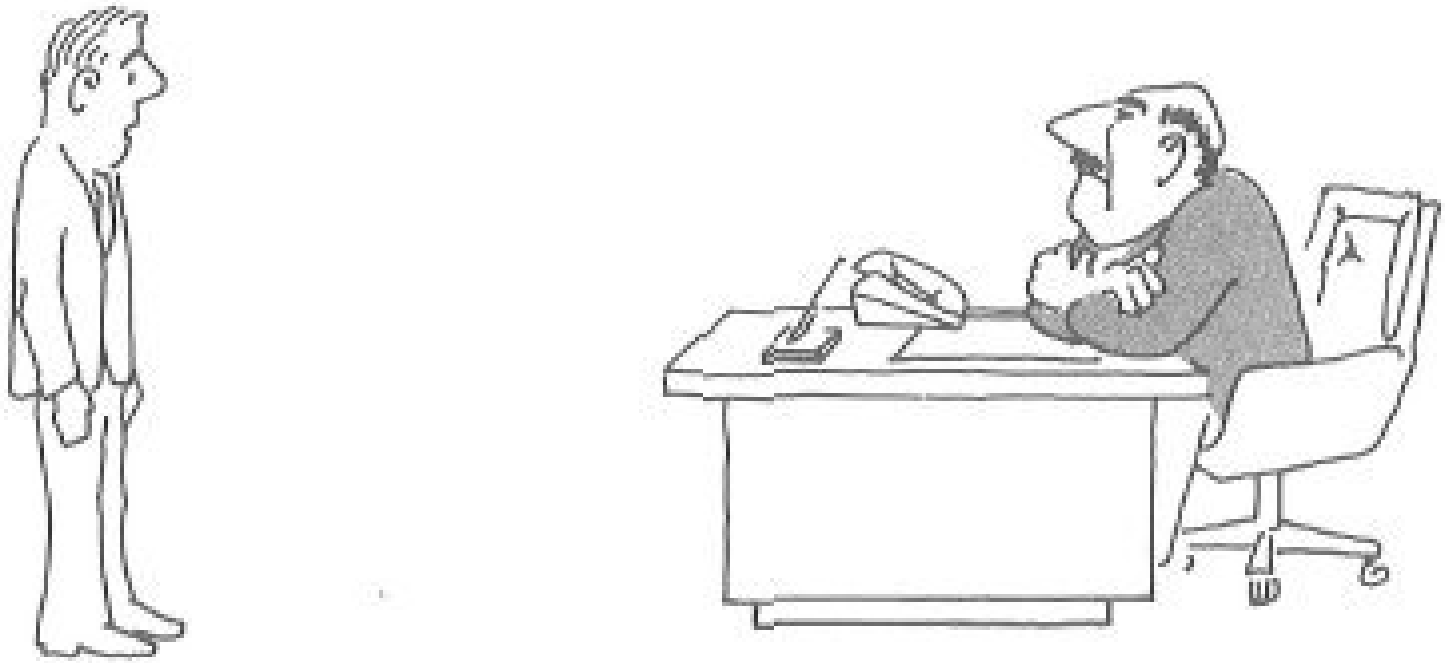
Príklad

Vstup: GCCAAC, CCTGCC, ACCTTC
Výstup: CCTGCCAACCTTC (najkratšie možné)

Problém # 2: Najkratšie spoločné nadslovo

Najlepší algoritmus?

- Nepoznáme algoritmus, ktorý by bežal v polynomiálnom čase t.j. $O(n^k)$ pre nejakú konštantu k.
- Daný problém je *NP-ťažký*.



"I can't find an efficient algorithm, I guess I'm just too dumb."

"I can't find an efficient algorithm, because no such algorithm is possible!"

"I can't find an efficient algorithm, but neither can all these famous people."

Ako sa vysporiadať s NP-ťažkými problémami?

Heuristické algoritmy

- Nájde *aspoň nejaké riešenie*, aj keď nie nutne optimálne
- Nejde teda o správny algoritmus riešiaci náš problém, lebo pre niektoré vstupy dáva zlú odpoveď
- Radšej ale horšia odpoveď rýchlo, ako perfektná o milión rokov

Príklad

Heuristika pre najkratší spoločný nadreťazec: v každom kroku zlepíme dva reťazce s najväčším prekryvom
Príklad: CATATAT, TATATA, ATATATC
Optimum: CATATATATC, dĺžka 10
Heuristika: CATATATCTATATA, dĺžka 14

Ako so vysporiadať s NP-ťažkými problémami?

Aproximačný algoritmus

Často vieme dokázať, že nejaká heuristika sa vždy priblíži k optimálnemu riešeniu aspoň po určitú hranicu

Príklad

Heuristika pre najkratší spoločný nadreťazec: v každom kroku zlepíme dva reťazce s najväčším prekryvom
Je dokázané, že vždy nájde najviac 3,5-krát dlhší reťazec ako najlepšie riešenie.
Informatici predpokladajú, že v skutočnosti najviac 2-krát dlhší, ale nevieme to dokázať.

Ako so vysporiadať s NP-ťažkými problémami?

Exaktný výpočet pomocou iného problému

- Preformulovať do podoby jedného z dobre známych NP-ťažkých problémov (napr. celočíselné lineárne programovanie, a pod.)
- Múdri ľudia napísali programy, ktoré vedia riešiť tieto známe problémy *aspoň v niektorých prípadoch* (CONCORD, CPLEX, a pod.)

Preformulovať problém

- Je toto skutočne jediná rozumná formulácia biologického problému ktorý chceme vyriešiť?

Zhrnutie

- Problémy zo skutočného života je dobré najskôr sformulovať tak, aby bolo jasné, aké výsledky očakávame pre každý možný vstup.
- Takáto formulácia by mala byť oddelená od postupu (algoritmu) riešenia.
- Informatici merajú čas v O-čkach, ktoré abstrahujú od detailov konkrétneho počítača.
- Vytvorenie efektívneho algoritmu je umenie! Časť z toho sú finty (ako napr. dynamické programovanie).
- Pre niektoré problémy poznáme iba Nechutne Pomalé algoritmy (NP-ťažké problémy).
- Aj napriek tomu vo veľa prípadoch vieme pomôcť.

Metódy v bioinformatike

CB #2 Úvod do dynamického programovania

Jana Černíková

FMFI UK

3/10/2024

Problém platenia minimálnym počtom mincí

Vstup: hodnoty k mincí $m_1, m_2, \ldots, m_k$ a cieľová suma X (všetko kladné celé čísla).

Výstup: najmenší počet mincí, ktoré potrebujeme na zaplatenie X.

Problém platenia minimálnym počtom mincí

Vstup: hodnoty k mincí $m_1, m_2, \ldots, m_k$ a cieľová suma X
(všetko kladné celé čísla).
Výstup: najmenší počet mincí, ktoré potrebujeme na zaplatenie X.

Príklad: $k = 3$, $m_1 = 1$, $m_2 = 2$, $m_3 = 5$, $X = 13$.

Odbočka: ešte matematickejšia formulácia bez slov minca, suma, ...

Vstup: kladné celé čísla $m_1, m_2, \ldots, m_k$ a X.
Výstup: celé číslo n a n čísel $x_1, \ldots, x_n$, pre ktoré platia nasledujúce podmienky:

- $x_i \in \{m_1, m_2, \ldots, m_k\}$ pre každé $i = 1, 2, \ldots, n$.
- $\sum_{i=1}^{n} x_i = X$.
- n je najmenšie možné.

Problém platenia minimálnym počtom mincí

Vstup: hodnoty k mincí $m_1, m_2, \ldots, m_k$ a cieľová suma X (všetko kladné celé čísla).
Výstup: najmenší počet mincí, ktoré potrebujeme na zaplatenie X.

Príklad: $k = 3,\ m_1 = 1,\ m_2 = 2,\ m_3 = 5,\ X = 13.$
Príklad: $k = 3,\ m_1 = 1,\ m_2 = 3,\ m_3 = 4,\ X = 6.$

Jednoduchý spôsob riešenia

- opakuj kým $X > 0$:
  - použi najväčšiu mincu, ktorá je najviac X
  - odčítaj hodnotu mince od X

Jednoduchý spôsob riešenia

- opakuj kým $X > 0$:
  - použi najväčšiu mincu, ktorá je najviac X
  - odčítaj hodnotu mince od X

príklad pre $k = 3$, $m_1 = 1$, $m_2 = 2$, $m_3 = 5$, $X = 13$:

Jednoduchý spôsob riešenia

- opakuj kým $X > 0$:
  - použi najväčšiu mincu, ktorá je najviac X
  - odčítaj hodnotu mince od X

príklad pre $k = 3$, $m_1 = 1$, $m_2 = 2$, $m_3 = 5$, $X = 13$:

- použijeme 5, X = 8

Jednoduchý spôsob riešenia

- opakuj kým $X > 0$:
  - použi najväčšiu mincu, ktorá je najviac X
  - odčítaj hodnotu mince od X

príklad pre $k = 3$, $m_1 = 1$, $m_2 = 2$, $m_3 = 5$, $X = 13$:

- použijeme 5, X = 8
- použijeme 5, X = 3

Jednoduchý spôsob riešenia

- opakuj kým $X > 0$:
  - použi najväčšiu mincu, ktorá je najviac X
  - odčítaj hodnotu mince od X

príklad pre $k = 3$, $m_1 = 1$, $m_2 = 2$, $m_3 = 5$, $X = 13$:

- použijeme 5, X = 8
- použijeme 5, X = 3
- použijeme 2, X = 1

Jednoduchý spôsob riešenia

- opakuj kým $X > 0$:
  - použi najväčšiu mincu, ktorá je najviac X
  - odčítaj hodnotu mince od X

príklad pre $k = 3$, $m_1 = 1$, $m_2 = 2$, $m_3 = 5$, $X = 13$:

- použijeme 5, X = 8
- použijeme 5, X = 3
- použijeme 2, X = 1
- použijeme 1, X = 0

Jednoduchý spôsob riešenia

- opakuj kým $X > 0$:
  - použi najväčšiu mincu, ktorá je najviac X
  - odčítaj hodnotu mince od X

príklad pre $k = 3$, $m_1 = 1$, $m_2 = 2$, $m_3 = 5$, $X = 13$:

- použijeme 5, X = 8
- použijeme 5, X = 3
- použijeme 2, X = 1
- použijeme 1, X = 0

Problém s týmto riešením?

Jednoduchý spôsob riešenia

- opakuj kým $X > 0$:
  - použi najväčšiu mincu, ktorá je najviac X
  - odčítaj hodnotu mince od X

príklad pre $k = 3$, $m_1 = 1$, $m_2 = 2$, $m_3 = 5$, $X = 13$:

- použijeme 5, X = 8
- použijeme 5, X = 3
- použijeme 2, X = 1
- použijeme 1, X = 0

Problém s týmto riešením? nefunguje vždy

Jednoduchý spôsob riešenia

- opakuj kým $X > 0$:
  - použi najväčšiu mincu, ktorá je najviac X
  - odčítaj hodnotu mince od X

Problém s týmto riešením: nefunguje vždy

Jednoduchý spôsob riešenia

- opakuj kým $X > 0$:
  - použi najväčšiu mincu, ktorá je najviac X
  - odčítaj hodnotu mince od X

Problém s týmto riešením: nefunguje vždy

príklad:

- mince hodnôt 1,3,4
- X = 6

Jednoduchý spôsob riešenia

- opakuj kým $X > 0$:
  - použi najväčšiu mincu, ktorá je najviac X
  - odčítaj hodnotu mince od X

Problém s týmto riešením: nefunguje vždy

príklad:

- mince hodnôt 1,3,4
- X = 6
- algoritmus:

Jednoduchý spôsob riešenia

- opakuj kým $X > 0$:
  - použi najväčšiu mincu, ktorá je najviac X
  - odčítaj hodnotu mince od X

Problém s týmto riešením: nefunguje vždy

príklad:

- mince hodnôt 1,3,4
- X = 6
- algoritmus: $4 + 1 + 1$

Jednoduchý spôsob riešenia

- opakuj kým $X > 0$:
  - použi najväčšiu mincu, ktorá je najviac X
  - odčítaj hodnotu mince od X

Problém s týmto riešením: nefunguje vždy

príklad:

- mince hodnôt 1,3,4
- X = 6
- algoritmus: $4 + 1 + 1$
- optimum:

Jednoduchý spôsob riešenia

- opakuj kým $X > 0$:
  - použi najväčšiu mincu, ktorá je najviac X
  - odčítaj hodnotu mince od X

Problém s týmto riešením: nefunguje vždy

príklad:

- mince hodnôt 1,3,4
- X = 6
- algoritmus: $4 + 1 + 1$
- optimum: $3 + 3$

Riešenie dynamickým programovaním

- zrátame najlepší počet mincí nielen pre X, ale pre všetky možné cieľové sumy $1, 2, 3, ..., X-1, X$
- vyrobíme si tabuľku A, do ktorej si pre všetky sumy $i = 1, 2, 3, ..., X-1, X$ uložíme najmenší počet mincí, ktorými ich vieme zaplatiť
  - A[i] = najmenší počet mincí, ktoré treba na zaplatenie sumy i

Riešenie dynamickým programovaním

- zrátame najlepší počet mincí nielen pre X, ale pre všetky možné cieľové sumy $1, 2, 3, ..., X-1, X$
- vyrobíme si tabuľku A, do ktorej si pre všetky sumy $i = 1, 2, 3, ..., X-1, X$ uložíme najmenší počet mincí, ktorými ich vieme zaplatiť
  - A[i] = najmenší počet mincí, ktoré treba na zaplatenie sumy i

mince 1, 3, 4

i	0	1	2	3	4	5	6	7	8	9
$A[i]$										

Riešenie dynamickým programovaním

- zrátame najlepší počet mincí nielen pre X, ale pre všetky možné cieľové sumy $1, 2, 3, ..., X-1, X$
- vyrobíme si tabuľku A, do ktorej si pre všetky sumy $i = 1, 2, 3, ..., X-1, X$ uložíme najmenší počet mincí, ktorými ich vieme zaplatiť
  - A[i] = najmenší počet mincí, ktoré treba na zaplatenie sumy i

mince 1, 3, 4

i	0	1	2	3	4	5	6	7	8	9
$A[i]$	0									

Riešenie dynamickým programovaním

- zrátame najlepší počet mincí nielen pre X, ale pre všetky možné cieľové sumy $1, 2, 3, ..., X-1, X$
- vyrobíme si tabuľku A, do ktorej si pre všetky sumy $i = 1, 2, 3, ..., X-1, X$ uložíme najmenší počet mincí, ktorými ich vieme zaplatiť
  - A[i] = najmenší počet mincí, ktoré treba na zaplatenie sumy i

mince 1, 3, 4

i	0	1	2	3	4	5	6	7	8	9
$A[i]$	0	1								

Riešenie dynamickým programovaním

- zrátame najlepší počet mincí nielen pre X, ale pre všetky možné cieľové sumy $1, 2, 3, ..., X-1, X$
- vyrobíme si tabuľku A, do ktorej si pre všetky sumy $i = 1, 2, 3, ..., X-1, X$ uložíme najmenší počet mincí, ktorými ich vieme zaplatiť
  - A[i] = najmenší počet mincí, ktoré treba na zaplatenie sumy i

mince 1, 3, 4

i	0	1	2	3	4	5	6	7	8	9
$A[i]$	0	1	2							

Riešenie dynamickým programovaním

- zrátame najlepší počet mincí nielen pre X, ale pre všetky možné cieľové sumy $1, 2, 3, ..., X-1, X$
- vyrobíme si tabuľku A, do ktorej si pre všetky sumy $i = 1, 2, 3, ..., X-1, X$ uložíme najmenší počet mincí, ktorými ich vieme zaplatiť
  - A[i] = najmenší počet mincí, ktoré treba na zaplatenie sumy i

mince 1, 3, 4

i	0	1	2	3	4	5	6	7	8	9
$A[i]$	0	1	2	1						

Riešenie dynamickým programovaním

- zrátame najlepší počet mincí nielen pre X, ale pre všetky možné cieľové sumy $1, 2, 3, ..., X-1, X$
- vyrobíme si tabuľku A, do ktorej si pre všetky sumy $i = 1, 2, 3, ..., X-1, X$ uložíme najmenší počet mincí, ktorými ich vieme zaplatiť
  - A[i] = najmenší počet mincí, ktoré treba na zaplatenie sumy i

mince 1, 3, 4

i	0	1	2	3	4	5	6	7	8	9
$A[i]$	0	1	2	1	1					

Riešenie dynamickým programovaním

- zrátame najlepší počet mincí nielen pre X, ale pre všetky možné cieľové sumy $1, 2, 3, ..., X-1, X$
- vyrobíme si tabuľku A, do ktorej si pre všetky sumy $i = 1, 2, 3, ..., X-1, X$ uložíme najmenší počet mincí, ktorými ich vieme zaplatiť
  - A[i] = najmenší počet mincí, ktoré treba na zaplatenie sumy i

mince 1, 3, 4

i	0	1	2	3	4	5	6	7	8	9
$A[i]$	0	1	2	1	1	2				

Riešenie dynamickým programovaním

- zrátame najlepší počet mincí nielen pre X, ale pre všetky možné cieľové sumy $1, 2, 3, ..., X-1, X$
- vyrobíme si tabuľku A, do ktorej si pre všetky sumy $i = 1, 2, 3, ..., X-1, X$ uložíme najmenší počet mincí, ktorými ich vieme zaplatiť
  - A[i] = najmenší počet mincí, ktoré treba na zaplatenie sumy i

mince 1, 3, 4

i	0	1	2	3	4	5	6	7	8	9
$A[i]$	0	1	2	1	1	2	2			

Riešenie dynamickým programovaním

- zrátame najlepší počet mincí nielen pre X, ale pre všetky možné cieľové sumy $1, 2, 3, ..., X-1, X$
- vyrobíme si tabuľku A, do ktorej si pre všetky sumy $i = 1, 2, 3, ..., X-1, X$ uložíme najmenší počet mincí, ktorými ich vieme zaplatiť
  - A[i] = najmenší počet mincí, ktoré treba na zaplatenie sumy i

mince 1, 3, 4

i	0	1	2	3	4	5	6	7	8	9
$A[i]$	0	1	2	1	1	2	2	2		

Riešenie dynamickým programovaním

- zrátame najlepší počet mincí nielen pre X, ale pre všetky možné cieľové sumy $1, 2, 3, ..., X-1, X$
- vyrobíme si tabuľku A, do ktorej si pre všetky sumy $i = 1, 2, 3, ..., X-1, X$ uložíme najmenší počet mincí, ktorými ich vieme zaplatiť
  - A[i] = najmenší počet mincí, ktoré treba na zaplatenie sumy i

mince 1, 3, 4

i	0	1	2	3	4	5	6	7	8	9
$A[i]$	0	1	2	1	1	2	2	2	2	

Riešenie dynamickým programovaním

- zrátame najlepší počet mincí nielen pre X, ale pre všetky možné cieľové sumy $1, 2, 3, ..., X-1, X$
- vyrobíme si tabuľku A, do ktorej si pre všetky sumy $i = 1, 2, 3, ..., X-1, X$ uložíme najmenší počet mincí, ktorými ich vieme zaplatiť
  - A[i] = najmenší počet mincí, ktoré treba na zaplatenie sumy i

mince 1, 3, 4

i	0	1	2	3	4	5	6	7	8	9
$A[i]$	0	1	2	1	1	2	2	2	2	3

Riešenie dynamickým programovaním

A[i] = najmenší počet mincí, ktoré treba na zaplatenie sumy i

mince 1, 3, 4

i	0	1	2	3	4	5	6	7	8	9	10
$A[i]$	0	1	2	1	1	2	2	2	2	3	?

X = 10, mince: 1, 3, 4

prvá minca	1	3	4

Riešenie dynamickým programovaním

A[i] = najmenší počet mincí, ktoré treba na zaplatenie sumy i

mince 1, 3, 4

i	0	1	2	3	4	5	6	7	8	9	10
$A[i]$	0	1	2	1	1	2	2	2	2	3	?

X = 10, mince: 1, 3, 4

prvá minca	1	3	4
X - prvá			

Riešenie dynamickým programovaním

A[i] = najmenší počet mincí, ktoré treba na zaplatenie sumy i

mince 1, 3, 4

i	0	1	2	3	4	5	6	7	8	9	10
$A[i]$	0	1	2	1	1	2	2	2	2	3	?

X = 10, mince: 1, 3, 4

prvá minca	1	3	4
X - prvá	10 - 1 = 9		

Riešenie dynamickým programovaním

A[i] = najmenší počet mincí, ktoré treba na zaplatenie sumy i

mince 1, 3, 4

i	0	1	2	3	4	5	6	7	8	9	10
$A[i]$	0	1	2	1	1	2	2	2	2	3	?

X = 10, mince: 1, 3, 4

prvá minca	1	3	4
X - prvá	10 - 1 = 9	10 - 3 = 7	

Riešenie dynamickým programovaním

A[i] = najmenší počet mincí, ktoré treba na zaplatenie sumy i

mince 1, 3, 4

i	0	1	2	3	4	5	6	7	8	9	10
$A[i]$	0	1	2	1	1	2	2	2	2	3	?

X = 10, mince: 1, 3, 4

prvá minca	1	3	4
X - prvá	10 - 1 = 9	10 - 3 = 7	10 - 4 = 6

Riešenie dynamickým programovaním

A[i] = najmenší počet mincí, ktoré treba na zaplatenie sumy i

mince 1, 3, 4

i	0	1	2	3	4	5	6	7	8	9	10
$A[i]$	0	1	2	1	1	2	2	2	2	3	?

X = 10, mince: 1, 3, 4

prvá minca	1	3	4
X - prvá	10 - 1 = 9	10 - 3 = 7	10 - 4 = 6
# mincí ešte potrebujeme			

Použijeme tabuľku

Riešenie dynamickým programovaním

A[i] = najmenší počet mincí, ktoré treba na zaplatenie sumy i

mince 1, 3, 4

i	0	1	2	3	4	5	6	7	8	9	10
$A[i]$	0	1	2	1	1	2	2	2	2	3	?

X = 10, mince: 1, 3, 4

prvá minca	1	3	4
X - prvá	10 - 1 = 9	10 - 3 = 7	10 - 4 = 6
# mincí ešte potrebujeme	3		

Použijeme tabuľku

Riešenie dynamickým programovaním

A[i] = najmenší počet mincí, ktoré treba na zaplatenie sumy i

mince 1, 3, 4

i	0	1	2	3	4	5	6	7	8	9	10
$A[i]$	0	1	2	1	1	2	2	2	2	3	?

X = 10, mince: 1, 3, 4

prvá minca	1	3	4
X - prvá	10 - 1 = 9	10 - 3 = 7	10 - 4 = 6
# mincí ešte potrebujeme	3	2	

Použijeme tabuľku

Riešenie dynamickým programovaním

A[i] = najmenší počet mincí, ktoré treba na zaplatenie sumy i

mince 1, 3, 4

i	0	1	2	3	4	5	6	7	8	9	10
$A[i]$	0	1	2	1	1	2	2	2	2	3	?

X = 10, mince: 1, 3, 4

prvá minca	1	3	4
X - prvá	10 - 1 = 9	10 - 3 = 7	10 - 4 = 6
# mincí ešte potrebujeme	3	2	2

Použijeme tabuľku

Riešenie dynamickým programovaním

A[i] = najmenší počet mincí, ktoré treba na zaplatenie sumy i

mince 1, 3, 4

i	0	1	2	3	4	5	6	7	8	9	10
$A[i]$	0	1	2	1	1	2	2	2	2	3	?

X = 10, mince: 1, 3, 4

prvá minca	1	3	4
X - prvá	10 - 1 = 9	10 - 3 = 7	10 - 4 = 6
# mincí ešte potrebujeme	3	2	2
dokopy mincí			

Riešenie dynamickým programovaním

A[i] = najmenší počet mincí, ktoré treba na zaplatenie sumy i

mince 1, 3, 4

i	0	1	2	3	4	5	6	7	8	9	10
$A[i]$	0	1	2	1	1	2	2	2	2	3	?

X = 10, mince: 1, 3, 4

prvá minca	1	3	4
X - prvá	10 - 1 = 9	10 - 3 = 7	10 - 4 = 6
# mincí ešte potrebujeme	3	2	2
dokopy mincí	4		

Riešenie dynamickým programovaním

A[i] = najmenší počet mincí, ktoré treba na zaplatenie sumy i

mince 1, 3, 4

i	0	1	2	3	4	5	6	7	8	9	10
$A[i]$	0	1	2	1	1	2	2	2	2	3	?

X = 10, mince: 1, 3, 4

prvá minca	1	3	4
X - prvá	10 - 1 = 9	10 - 3 = 7	10 - 4 = 6
# mincí ešte potrebujeme	3	2	2
dokopy mincí	4	3	

Riešenie dynamickým programovaním

A[i] = najmenší počet mincí, ktoré treba na zaplatenie sumy i

mince 1, 3, 4

i	0	1	2	3	4	5	6	7	8	9	10
$A[i]$	0	1	2	1	1	2	2	2	2	3	?

X = 10, mince: 1, 3, 4

prvá minca	1	3	4
X - prvá	10 - 1 = 9	10 - 3 = 7	10 - 4 = 6
# mincí ešte potrebujeme	3	2	2
dokopy mincí	4	3	3

Riešenie dynamickým programovaním

A[i] = najmenší počet mincí, ktoré treba na zaplatenie sumy i

mince 1, 3, 4

i	0	1	2	3	4	5	6	7	8	9	10
$A[i]$	0	1	2	1	1	2	2	2	2	3	3

X = 10, mince: 1, 3, 4

prvá minca	1	3	4
X - prvá	10 - 1 = 9	10 - 3 = 7	10 - 4 = 6
# mincí ešte potrebujeme	3	2	2
dokopy mincí	4	3	3

Riešenie dynamickým programovaním

A[i] = najmenší počet mincí, ktoré treba na zaplatenie sumy i

mince 1, 3, 4

i	0	1	2	3	4	5	6	7	8	9	10
$A[i]$	0	1	2	1	1	2	2	2	2	3	3

X = 10, mince: 1, 3, 4

prvá minca	1	3	4
X - prvá	10 - 1 = 9	10 - 3 = 7	10 - 4 = 6
# mincí ešte potrebujeme	3	2	2
dokopy mincí	4	3	3

$$A[10] = min\{A[9] + 1, A[7] + 1, A[6] + 1\}$$

Riešenie dynamickým programovaním

A[i] = najmenší počet mincí, ktoré treba na zaplatenie sumy i

mince 1, 3, 4

i	0	1	2	3	4	5	6	7	8	9	10
$A[i]$	0	1	2	1	1	2	2	2	2	3	3

X = 10, mince: 1, 3, 4

prvá minca	1	3	4
X - prvá	10 - 1 = 9	10 - 3 = 7	10 - 4 = 6
# mincí ešte potrebujeme	3	2	2
dokopy mincí	4	3	3

$$A[10] = min\{A[9] + 1, A[7] + 1, A[6] + 1\}$$

$$A[i] = min\{A[i-1] + 1, A[i-3] + 1, A[i-4] + 1\}$$

Algoritmus pre všeobecnú sústavu k mincí $m_1, m_2, \ldots, m_k$

Podproblém A[i]

$$A[i] = 1 + \min\{A[i - m_1], A[i - m_2], \ldots, A[i - m_k]\}$$

```
m = [1,3,4]
X = 11
k = len(m)
nekonecno = math.inf
A = [0]
for i in range(1, X + 1):
    min = nekonecno
    for j in range(k):
        if i >= m[j] and A[i - m[j]] < min:
        min = A[i - m[j]]
    A.append(1 + min)
print(A)
```

Ako nájsť ktoré mince použiť?

Pridáme druhú tabuľku B, kde v B[i] si pamätáme, ktorá bola najlepšia prvá minca, keď sme počítali A[i]
(ak je viac možností, zoberieme ľubovoľnú, napr. najväčšiu)

Ako nájsť ktoré mince použiť?

Pridáme druhú tabuľku B, kde v B[i] si pamätáme, ktorá bola najlepšia prvá minca, keď sme počítali A[i]
(ak je viac možností, zoberieme ľubovoľnú, napr. najväčšiu)

i	0	1	2	3	4	5	6	7	8	9	10
$A[i]$	0	1	2	1	1	2	2	2	2	3	3
$B[i]$	-	1	1	3	4	4	3	4	4	4	4

Ako nájsť ktoré mince použiť?

Pridáme druhú tabuľku B, kde v B[i] si pamätáme, ktorá bola najlepšia prvá minca, keď sme počítali A[i]
(ak je viac možností, zoberieme ľubovoľnú, napr. najväčšiu)

i	0	1	2	3	4	5	6	7	8	9	10
$A[i]$	0	1	2	1	1	2	2	2	2	3	3
$B[i]$	-	1	1	3	4	4	3	4	4	4	4

Rekonštrukcia riešenia pre sumu 10:

Ako nájsť ktoré mince použiť?

Pridáme druhú tabuľku B, kde v B[i] si pamätáme, ktorá bola najlepšia prvá minca, keď sme počítali A[i]
(ak je viac možností, zoberieme ľubovoľnú, napr. najväčšiu)

i	0	1	2	3	4	5	6	7	8	9	10
$A[i]$	0	1	2	1	1	2	2	2	2	3	3
$B[i]$	-	1	1	3	4	4	3	4	4	4	4

Rekonštrukcia riešenia pre sumu 10:

- B[10] = 4, zostane nám zaplatiť 6

Ako nájsť ktoré mince použiť?

Pridáme druhú tabuľku B, kde v B[i] si pamätáme, ktorá bola najlepšia prvá minca, keď sme počítali A[i]
(ak je viac možností, zoberieme ľubovoľnú, napr. najväčšiu)

i	0	1	2	3	4	5	6	7	8	9	10
$A[i]$	0	1	2	1	1	2	2	2	2	3	3
$B[i]$	-	1	1	3	4	4	3	4	4	4	4

Rekonštrukcia riešenia pre sumu 10:

- B[10] = 4, zostane nám zaplatiť 6
- B[6] = 3, zostane nám zaplatiť 3

Ako nájsť ktoré mince použiť?

Pridáme druhú tabuľku B, kde v B[i] si pamätáme, ktorá bola najlepšia prvá minca, keď sme počítali A[i]
(ak je viac možností, zoberieme ľubovoľnú, napr. najväčšiu)

i	0	1	2	3	4	5	6	7	8	9	10
$A[i]$	0	1	2	1	1	2	2	2	2	3	3
$B[i]$	-	1	1	3	4	4	3	4	4	4	4

Rekonštrukcia riešenia pre sumu 10:

- B[10] = 4, zostane nám zaplatiť 6
- B[6] = 3, zostane nám zaplatiť 3
- B[3] = 3, zostáva 0

Ako nájsť ktoré mince použiť?

Pridáme druhú tabuľku B, kde v B[i] si pamätáme, ktorá bola najlepšia prvá minca, keď sme počítali A[i]
(ak je viac možností, zoberieme ľubovoľnú, napr. najväčšiu)

i	0	1	2	3	4	5	6	7	8	9	10
$A[i]$	0	1	2	1	1	2	2	2	2	3	3
$B[i]$	-	1	1	3	4	4	3	4	4	4	4

Rekonštrukcia riešenia pre sumu 10:

- B[10] = 4, zostane nám zaplatiť 6
- B[6] = 3, zostane nám zaplatiť 3
- B[3] = 3, zostáva 0
- riešenie: 4 + 3 + 3

Program aj s výpisom mincí

```
m = [1,3,4]
X = 11
k = len(m)
nekonecno = 1000000
A = [0]
B = [-1]
for i in range(1, X + 1):
    min = nekonecno
    min_minca = -1
    for j in range(k):
        if i >= m[j] and A[i - m[j]] < min:
        min = A[i - m[j]]
        min_minca = m[j]
    A.append(1 + min)
    B.append(min_minca)

while X > 0:
    print(B[X])
    X = X - B[X]
```

Dynamické programovanie vo všeobecnosti

Dynamické programovanie vo všeobecnosti

- Okrem riešenia celého problému riešime aj menšie problémy (nazývame ich podproblémy).

Dynamické programovanie vo všeobecnosti

- Okrem riešenia celého problému riešime aj menšie problémy (nazývame ich podproblémy).
- Riešenia podproblémov ukladáme do tabuľky a používame pri riešení väčších podproblémov.

Dynamické programovanie vo všeobecnosti

- Okrem riešenia celého problému riešime aj menšie problémy (nazývame ich podproblémy).
- Riešenia podproblémov ukladáme do tabuľky a používame pri riešení väčších podproblémov.
- Technika dynamického programovania sa používa na viacero problémov v bioinformatike.
  - napr. hľadanie zarovnaní sekvencií

Metódy v bioinformatike

Zarovnávanie sekvencií

Jana Černíková

FMFI UK

10/10/2024

Problém: Lokálne zarovnávanie (local alignment)

```
ggcccttggagttgactgtcctgctgctccttgagg
ccattctcagagagaggaagtggcctcattttaatc
cgcttcccacagccttgtcctttccagacccatggg
agagggaggggctgagggtgtggctgagcccaccca
agtcacgcgtcactctgcaggtccctctcccccaag
gccgtggccttgggagcccgtggatcccagtgagtg
acgcctccacccccgccctactcgggcagtttaac
ccttgttgttcacttgcagacatcgtgaacacggcc
cggcccgacgagaaggccataatgacctatgtgtcc
agcttctaccatgccttttcaggagcgcagaaggta
ccgagcagggccaggcaggccctcctcgccgccacc
gcgcaatgccgccgctgcctctcgcctcccgtgctc
acctcatttctcttgcagacggcagtggcctctctc
caactggaagccacccccagctccct...
```

```
tgatgccgaggatgtgttcgtcgagcatccggacga
gaagtccatcacctacgtggtcacctactatcacta
ctttagcaaactcaagcaggagacggtgcagggcat
aagcgtatcggtaaggtggtcggcattgccatggag
aacgacaaaatggtccacgactacgagaacttcaca
agcgatctgctcaagtggatcgaaacgaccatccag
tcgctgggcgagcgggagttcgaaaactcgctggcc
ggcgtccaagggcagttggcccagttctccaactac
cgcaccatcgagaagccgcccaagtttgtggaaaag
ggcaacctcgaggtgctccttttcaccctgcagtcc
aagatgcgggccaacaaccagaagccctacacaccc
aaagagggcaagatgatttcggacatcaacaaggcc
tgggagcgtctggagaaggccgagcacgaacgcgaa
ttggccctgcgcgaggagctcatccg...
```

Vstup: dve sekvencie

Problém: Lokálne zarovnávanie (local alignment)

Výstup: podobné úseky (zarovnania, alignments).

```
CCCGACGAGAAGGCCATAATGACCTATGTGTCCAGCTTCTACCATGCCTTT
||  |||||||||| ||||    |||||  |||  ||  ||  |||  ||    ||||
CCGGACGAGAAGTCCAT---CACCTACGTGGTCACCTACTATCACTACTTT
```

Vlož pomlčky (medzery, gaps) tak, aby rovnaké bázy boli pod sebou.
Dobré zarovnanie má veľa zarovnaných rovnakých báz, málo medzier.

Na čo sú dobré zarovnania?

- **Orientácia v obrovských databázach.**
  Genbank WGS má vyše 22 TB sekvencií.
  Napr. z ktorého genómu (a odkiaľ) pochádza daná sekvencia?
- **Prekryvy čítaní** pri skladaní genómov, mapovanie čítaní
- **Určovanie funkcie (napr. proteínu).**
  Podobné sekvencie často majú rovnakú/podobnú funkciu.
- **Štúdium evolúcie.**
  Hľadáme homológy: sekvencie, ktoré sa vyvinuli z tej istej sekvencie v spoločnom predkovi.
  V ideálnom prípade medzery zodpovedajú inzerciám a deléciám, zarovnané bázy zachovaným bázam a substitúciám.

Zarovnávanie sekvencií ako optimalizačný problém

- **Cieľ:** nájdi páry homologických sekvencií
  (tých, čo pochádzajú z rovnakého spoločného predka)
- **Modelovacia fáza:** vytvor skórovaciu schému, ktorá
  – skutočným homologickým párom dáva vysoké skóre
  – falošne pozitívnym párom dáva nízke skóre
- **Optimalizačná fáza:**
  pre dané dve vstupné sekvencie, nájdi zarovnanie s najlepším skóre
  (Optimalizačná fáza je téma dnešnej prednášky.)

Formulácia problému

Skórovanie zarovnania: napr. zhoda +1, nezhoda -1, medzera -1.

```
GAGAAGGCCATAATGACCTATGTGTCCAGCT
|||||| ||||     ||||| |||  || ||
GAGAAGTCCAT---CACCTACGTGGTCACCT
```

22 zhôd, 6 nezhôd, 3 medzery $\rightarrow$ skóre 13.
V praxi zložitejšie skórovanie.

Problém 1: globálne zarovnanie (global alignment)
Vstup: sekvencie $X = x_1x_2 \ldots x_n$ a $Y = y_1y_2 \ldots y_m$.
Výstup: zarovnanie X a Y s najvyšším skóre.

Problém 2: lokálne zarovnanie (local alignment)
Vstup: sekvencie $X = x_1x_2 \ldots x_n$ a $Y = y_1y_2 \ldots y_m$.
Výstup: zarovnania podreťazcov $x_i \ldots x_j$ a $y_k \ldots y_\ell$ s najvyšším skóre.

Dynamické programovanie pre globálne zarovnanie (Needleman, Wunsch 1970)

Podproblém: $A[i,j]$: najvyššie skóre globálneho zarovnania reťazcov $x_1x_2\ldots x_i$ a $y_1y_2\ldots y_j$.

Jeden z reťazcov dĺžky 0: druhý reťazec je zarovnaný s medzerou. $A[0,j] = -j$, $A[i,0] = -i$.

Všeobecný prípad, $i > 0$**,** $j > 0$**:**

- ak $x_i = y_j$ sú zarovnané $A[i,j] = A[i-1,j-1] + 1$
- ak $x_i \neq y_j$ sú zarovnané $A[i,j] = A[i-1,j-1] - 1$
- ak x_i je zarovnané s medzerou $A[i,j] = A[i-1,j] - 1$
- ak y_j je zarovnané s medzerou $A[i,j] = A[i,j-1] - 1$

Dynamické programovanie pre globálne zarovnanie

Podproblém: $A[i,j]$: najvyššie skóre globálneho zarovnania reťazcov $x_1x_2\ldots x_i$ a $y_1y_2\ldots y_j$.

Všeobecný prípad, $i > 0$, $j > 0$:

- ak $x_i = y_j$ sú zarovnané $A[i,j] = A[i-1,j-1] + 1$
- ak $x_i \neq y_j$ sú zarovnané $A[i,j] = A[i-1,j-1] - 1$
- ak x_i je zarovnané s medzerou $A[i,j] = A[i-1,j] - 1$
- ak y_j je zarovnané s medzerou $A[i,j] = A[i,j-1] - 1$

Rekurencia:

$$A[i,j] = \max \begin{cases} A[i-1,j-1] + s(x_i, y_j), \\ A[i-1,j] - 1, \\ A[i,j-1] - 1 \end{cases}$$

kde $s(x,y) = 1$ ak $x = y$ $\quad s(x,y) = -1$ ak $x \neq y$

Príklad globálneho zarovnania

CATGTCGTA vs CAGTCCTAGA

		C	A	G	T	C	C	T	A	G	A
	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10
C	-1	1	0	-1	-2	-3	-4	-5	-6	-7	-8
A	-2	0	2	1	0	-1	-2	-3	-4	-5	-6
T	-3	-1	1	1	?						
G	-4										
T	-5										
C	-6										
G	-7										
T	-8										
A	-9										

$$A[i,j] = \max \begin{cases} A[i-1,j-1] + s(x_i, y_j), \\ A[i-1,j] - 1, \\ A[i,j-1] - 1 \end{cases}$$

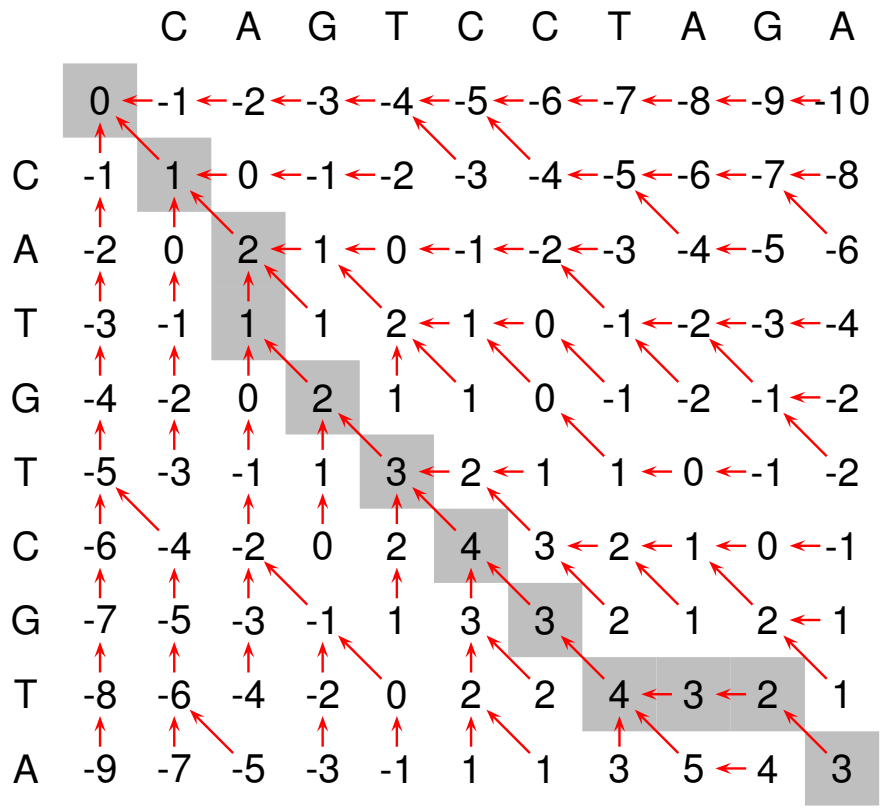
Príklad globálneho zarovnania

CATGTCGTA vs CAGTCCTAGA

		C	A	G	T	C	C	T	A	G	A
	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10
C	-1	1	0	-1	-2	-3	-4	-5	-6	-7	-8
A	-2	0	2	1	0	-1	-2	-3	-4	-5	-6
T	-3	-1	1	1	2	1	0	-1	-2	-3	-4
G	-4	-2	0	2	1	1	0	-1	-2	-1	-2
T	-5	-3	-1	1	3	2	1	1	0	-1	-2
C	-6	-4	-2	0	2	4	3	2	1	0	-1
G	-7	-5	-3	-1	1	3	3	2	1	2	1
T	-8	-6	-4	-2	0	2	2	4	3	2	1
A	-9	-7	-5	-3	-1	1	1	3	5	4	3

$$A[i,j] = \max \begin{cases} A[i-1,j-1] + s(x_i, y_j), \\ A[i-1,j] - 1, \\ A[i,j-1] - 1 \end{cases}$$

Ako získať zarovnanie?




```
CA-GTCCTAGA
CATGTCGT--A
```

Časová zložitosť celého algoritmu $O(nm)$

Dynamické programovanie pre lokálne zarovnanie (Smith, Waterman 1981)

Podproblém: $A[i, j]$: najvyššie skóre lokálneho zarovnania reťazcov $x_1 x_2 \ldots x_i$ a $y_1 y_2 \ldots y_j$, ktoré obsahuje bázy x_i a y_j, alebo je prázdne.

Jeden z reťazcov dĺžky 0: prázdne zarovnanie
$A[0, j] = A[i, 0] = 0$

Všeobecný prípad, $i > 0$**,** $j > 0$**:**

- ak x_i a y_j sú zarovnané $A[i, j] = A[i-1, j-1] + s(x_i, y_j)$
- ak x_i je zarovnané s medzerou $A[i, j] = A[i-1, j] - 1$
- ak y_j je zarovnané s medzerou $A[i, j] = A[i, j-1] - 1$
- ak x_i a y_j nie sú časťou zarovnania s kladným skóre $A[i, j] = 0$

Dynamické programovanie pre lokálne zarovnanie (Smith, Waterman 1981)

Podproblém: $A[i,j]$: najvyššie skóre lokálneho zarovnania reťazcov $x_1x_2\ldots x_i$ a $y_1y_2\ldots y_j$, ktoré obsahuje bázy x_i a y_j, alebo je prázdne.

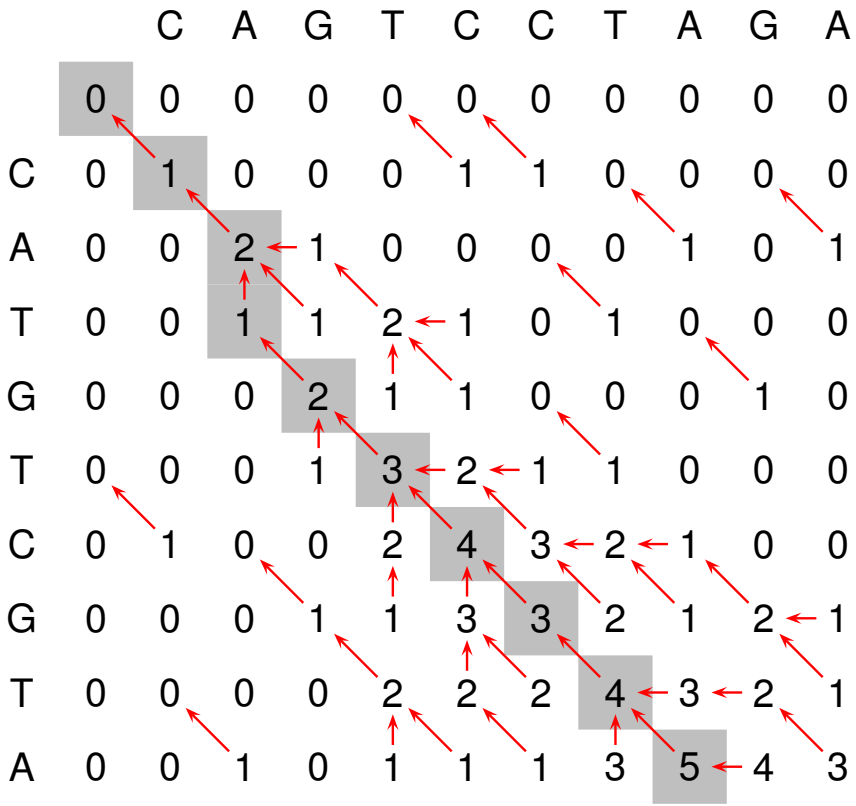
Všeobecný prípad, $i > 0$, $j > 0$:

- ak x_i a y_j sú zarovnané $A[i,j] = A[i-1,j-1] + s(x_i, y_j)$
- ak x_i je zarovnané s medzerou $A[i,j] = A[i-1,j] - 1$
- ak y_j je zarovnané s medzerou $A[i,j] = A[i,j-1] - 1$
- ak x_i a y_j nie sú časťou zarovnania s kladným skóre $A[i,j] = 0$

Rekurencia:

$$A[i,j] = \max \begin{cases} 0, \\ A[i-1,j-1] + s(x_i, y_j), \\ A[i-1,j] - 1, \\ A[i,j-1] - 1 \end{cases}$$

Príklad lokálneho zarovnania



CATGTCGTA

CA-GTCCTA

Časová zložitosť celého algoritmu $O(nm)$

Zložitejšie skórovanie

Problémy $+1, -1$ skórovania:

- Je skutočne jedna nezhoda alebo medzera až taká zlá v porovnaní s jednou zhodou?
- Čo urobíme pre zarovnávanie proteínov?
  (20 prvková abeceda $\approx$ 200 parametrov)

Úloha skórovacej schémy:

- Chceme vedieť rozlíšiť *lepšie zarovnania* od *horších zarovnaní*:
  - Ktoré usporiadania pomlčiek dávajú väčší zmysel
- Chceme vedieť, či dané zarovnanie *má biologický význam*:
  - Ide o homológy, alebo sú zarovnané len náhodou?

Povedali sme si:

- Globálne a lokálne zarovania
- Needlemanov-Wunschov a Smithov-Watermanov algoritmus

Pokračovanie prednášky

`https://youtu.be/0GkhkRiqbl4?feature=shared&t=2227`

- Skórovanie zarovnaní pomocou porovnávania modelov
- Proteínové BLOSUM matice
- Afínne skórovanie medzier

Metódy v bioinformatike

CB #3 Zarovnávanie sekvencií

Jana Černíková

FMFI UK

10/10/2024

Globálne zarovnanie

Uvažujme skórovanie zhoda +3, nezhoda -1, medzera -2
Reťazce TAACGG a CACACT

$$A[i,j] = \max \begin{cases} A[i-1,j-1] + s(x_i, y_j), \\ A[i-1,j] - 2, \\ A[i,j-1] - 2 \end{cases}$$

$s(x_i, y_j) = 3$ ak $x_i = y_j$,
$s(x_i, y_j) = -1$ ak $x_i \neq y_j$

$A[i,0] = -2i$,
$A[0,j] = -2j$

Globálne zarovnanie

		C	**A**	**C**	**A**	**C**	**T**
	0	-2	-4	-6	-8	-10	-12
T	-2						
A	-4						
A	-6						
C	-8						
G	-10						
G	-12						

Globálne zrovnanie

		0	1	2	3	4	5	6
			C	A	C	A	C	T
0		0	-2	-4	-6	-8	-10	-12
1	T	-2	-1	-3	-5	-7	-9	-7
2	A	-4	-3	2	0	-2	-4	-6
3	A	-6	-5	0	1	3	1	-1
4	C	-8	-3	-2	3	1	6	4
5	G	-10	-5	-4	1	2	4	5
6	G	-12	-7	-6	-1	0	2	3

CACACT-
TA-ACGG
alebo
CACAC-T
TA-ACGG

Lokálne zarovnanie

Uvažujme skórovanie zhoda +3, nezhoda -1, medzera -2
Reťazce TAACGG a CACACT

$$A[i,j] = \max \begin{cases} 0, \\ A[i-1,j-1] + s(x_i, y_j), \\ A[i-1,j] - 2, \\ A[i,j-1] - 2 \end{cases}$$

$s(x_i, y_j) = 3$ ak $x_i = y_j$,
$s(x_i, y_j) = -1$ ak $x_i \neq y_j$

$A[i,0] = 0$,
$A[0,j] = 0$

Lokálne zarovnanie

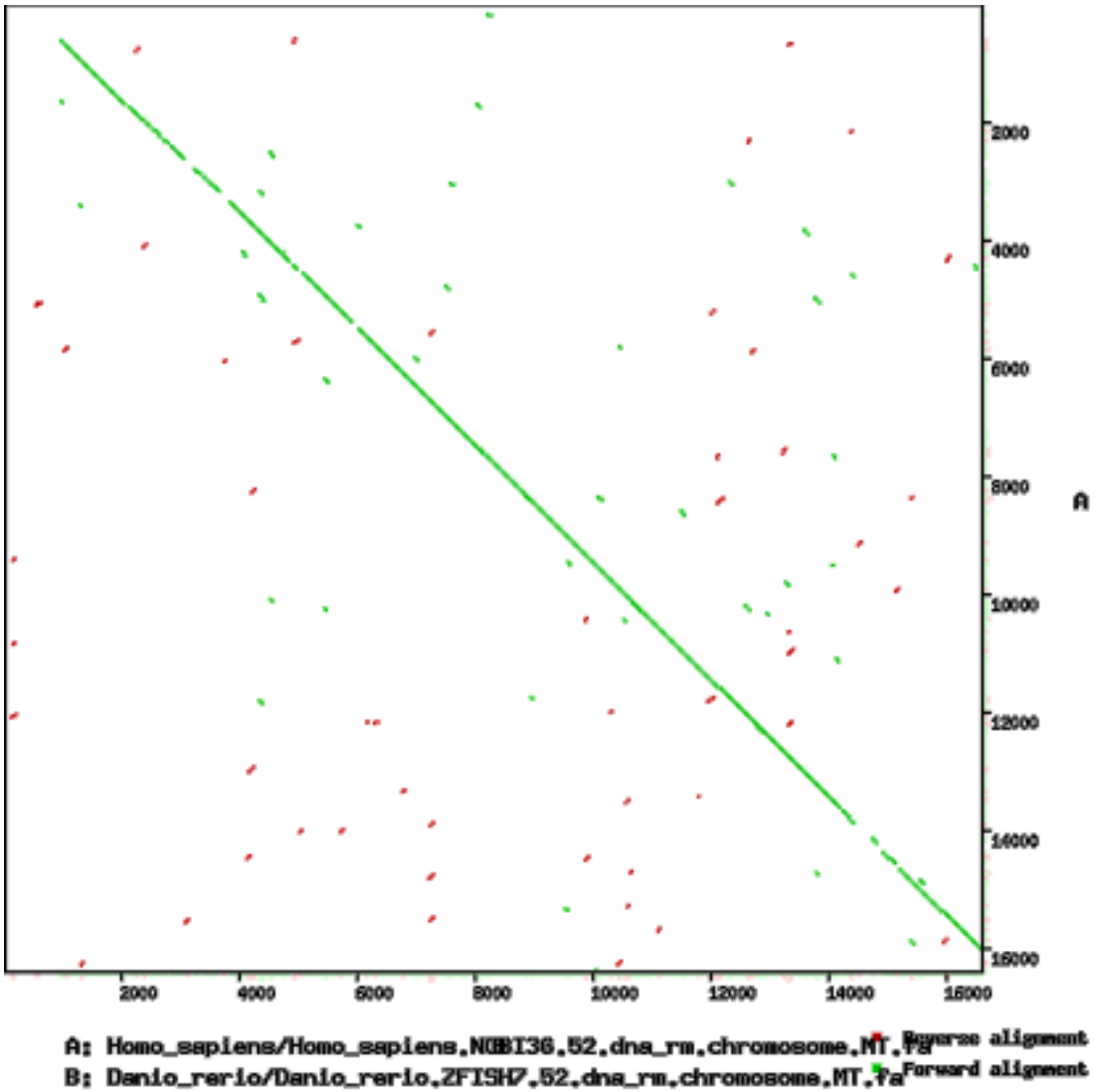
		C	**A**	**C**	**A**	**C**	**T**
	0	0	0	0	0	0	0
T	0						
A	0						
A	0						
C	0						
G	0						
G	0						

Lokálne zarovnanie

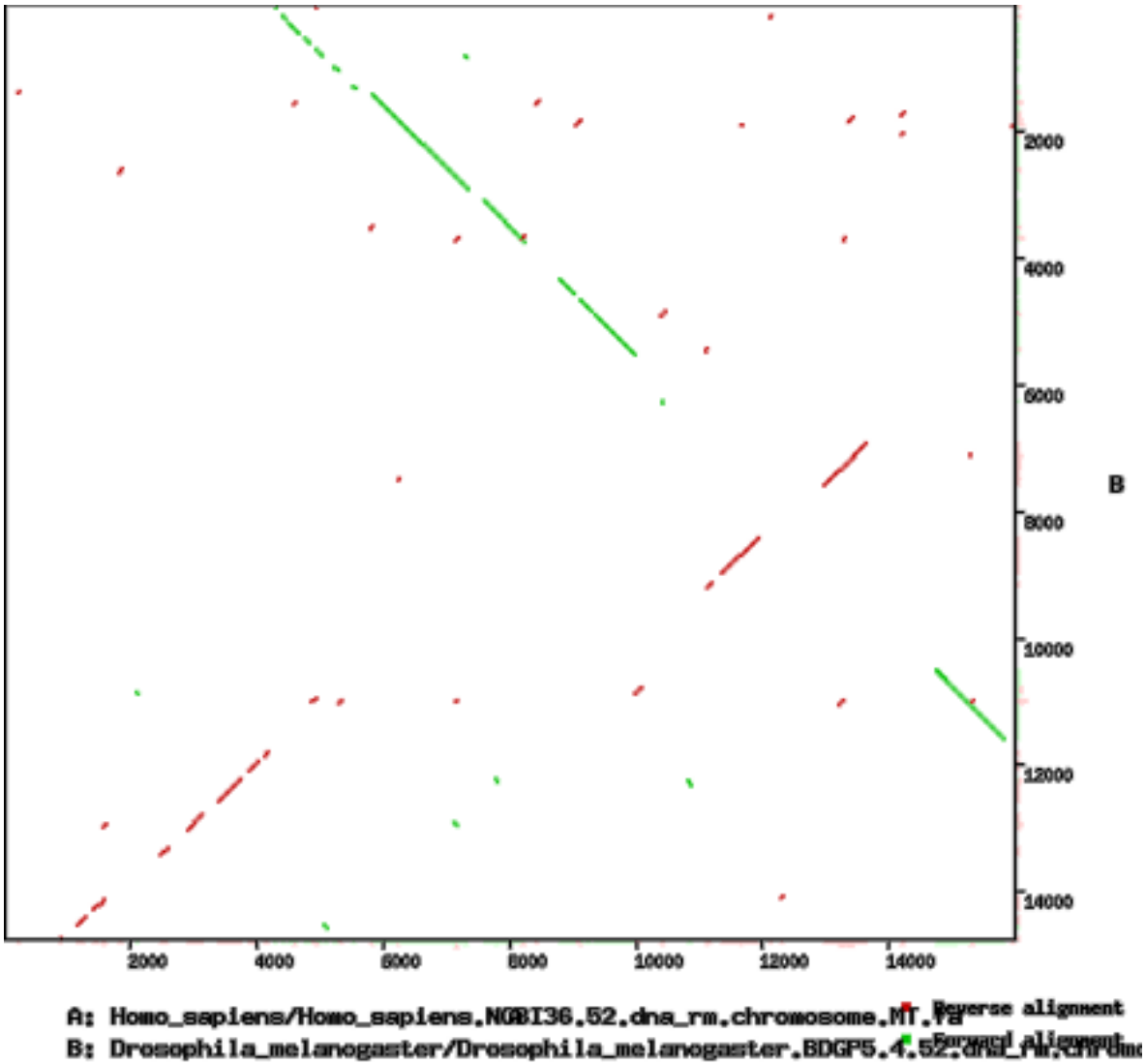
		0	1	2	3	4	5	6
			C	A	C	A	C	T
0		0	0	0	0	0	0	0
1	T	0	0	0	0	0	0	3
2	A	0	0	3	1	3	1	1
3	A	0	0	3	2	4	2	0
4	C	0	3	1	6	4	7	5
5	G	0	1	2	4	5	5	6
6	G	0	0	0	2	3	4	4

ACAC
A-AC

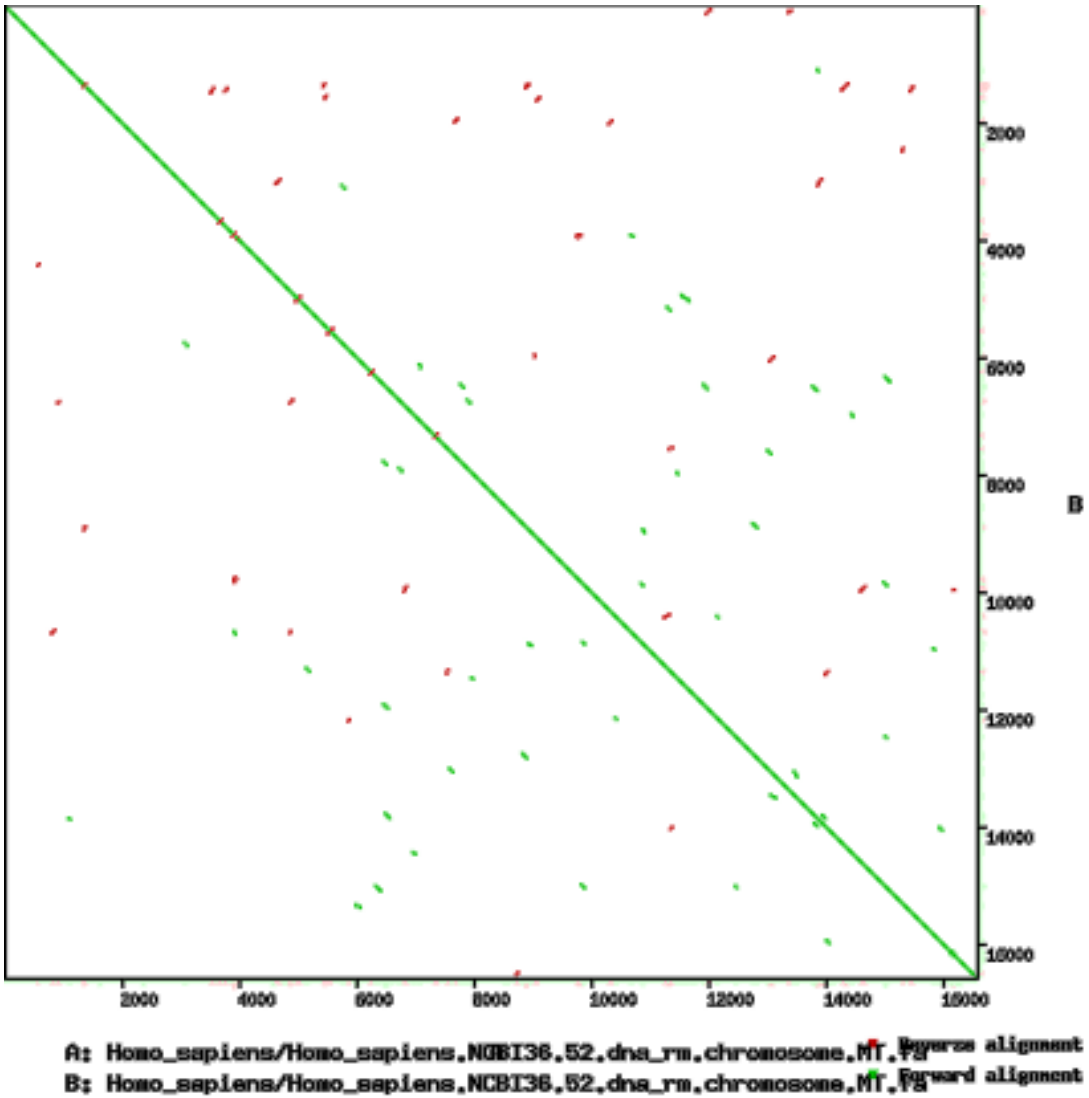
Mitochondriálny genóm človeka vs. ryba Danio rerio



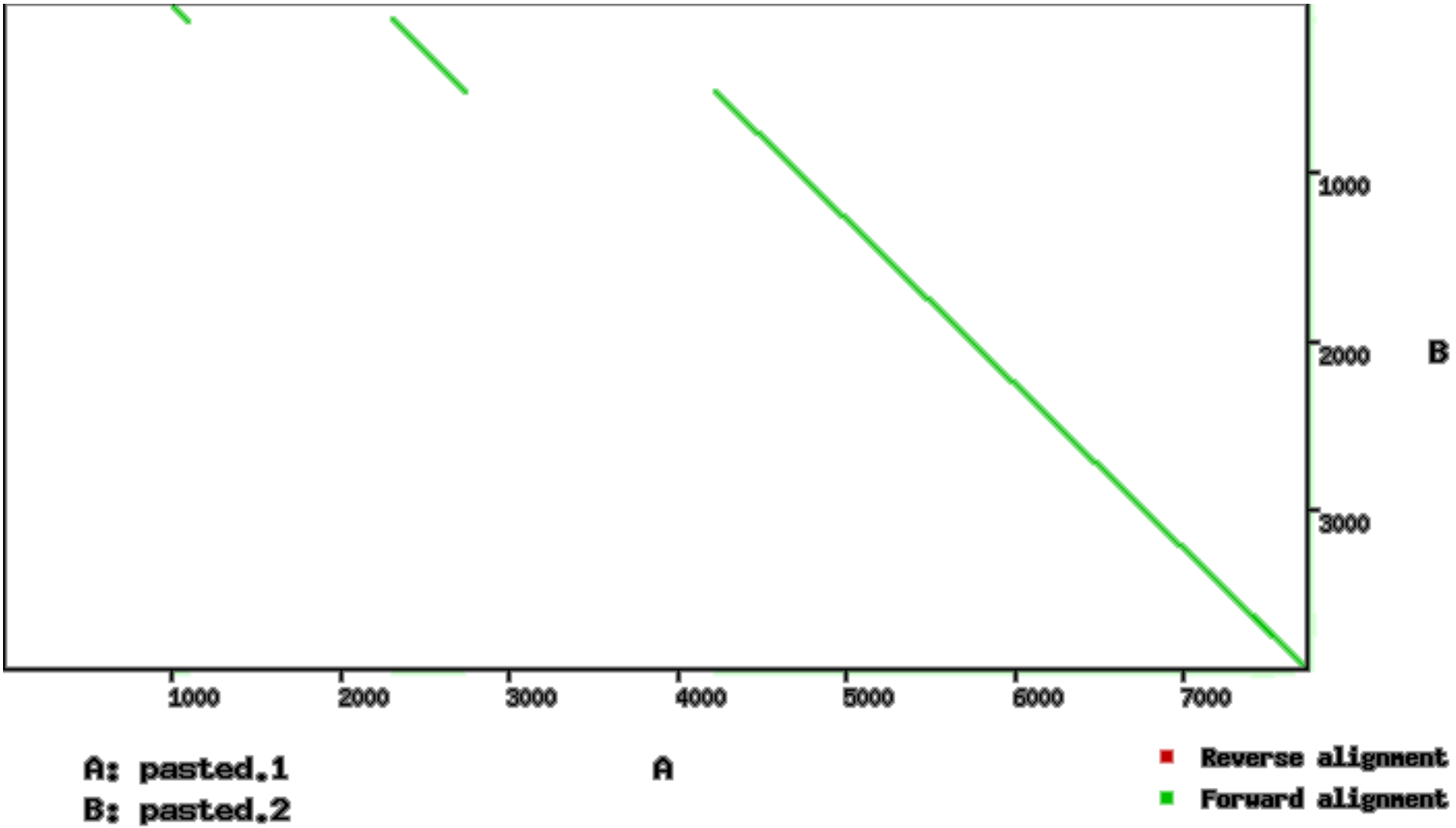
Mitochondriálny genóm človeka vs. Drosophila melanogaster



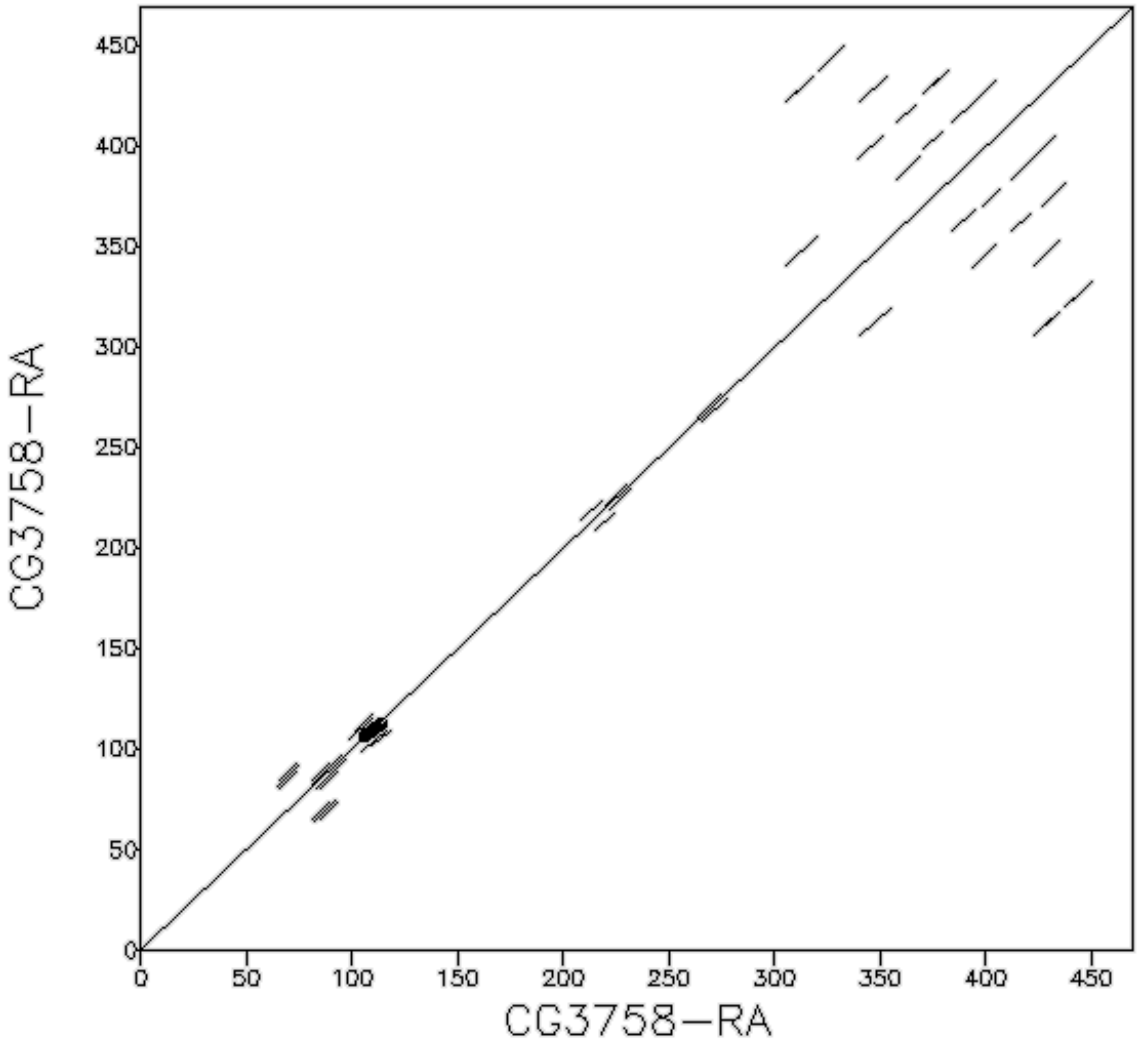
Mitochondriálny genóm človeka vs. to isté



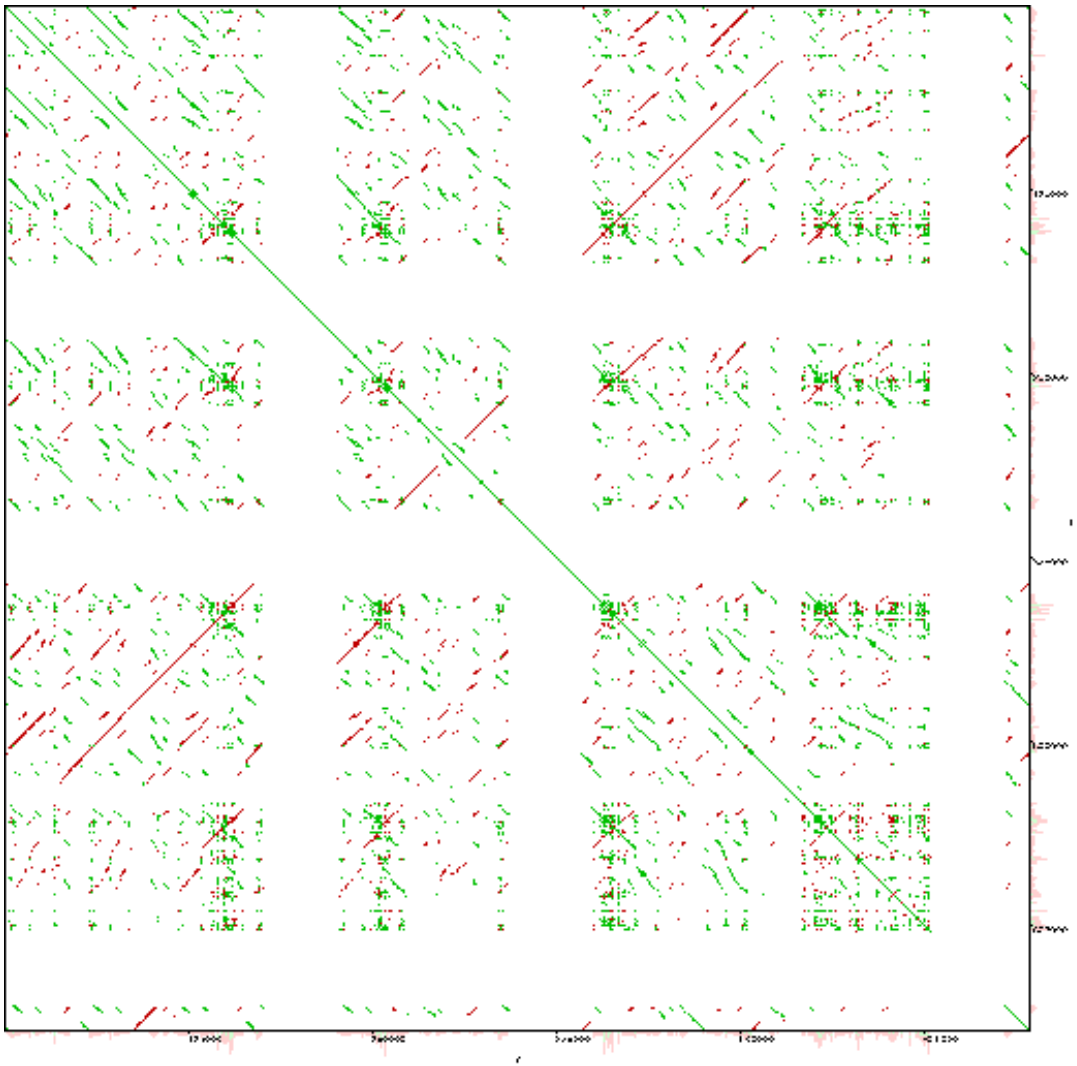
Drosophila mRNA Oaz zinc finger vs. genomický usek (časť chr2R)



Drosophila proteín Escargot zinc finger vs. to isté



Zhluk génov PRAME v človeku vs. to isté



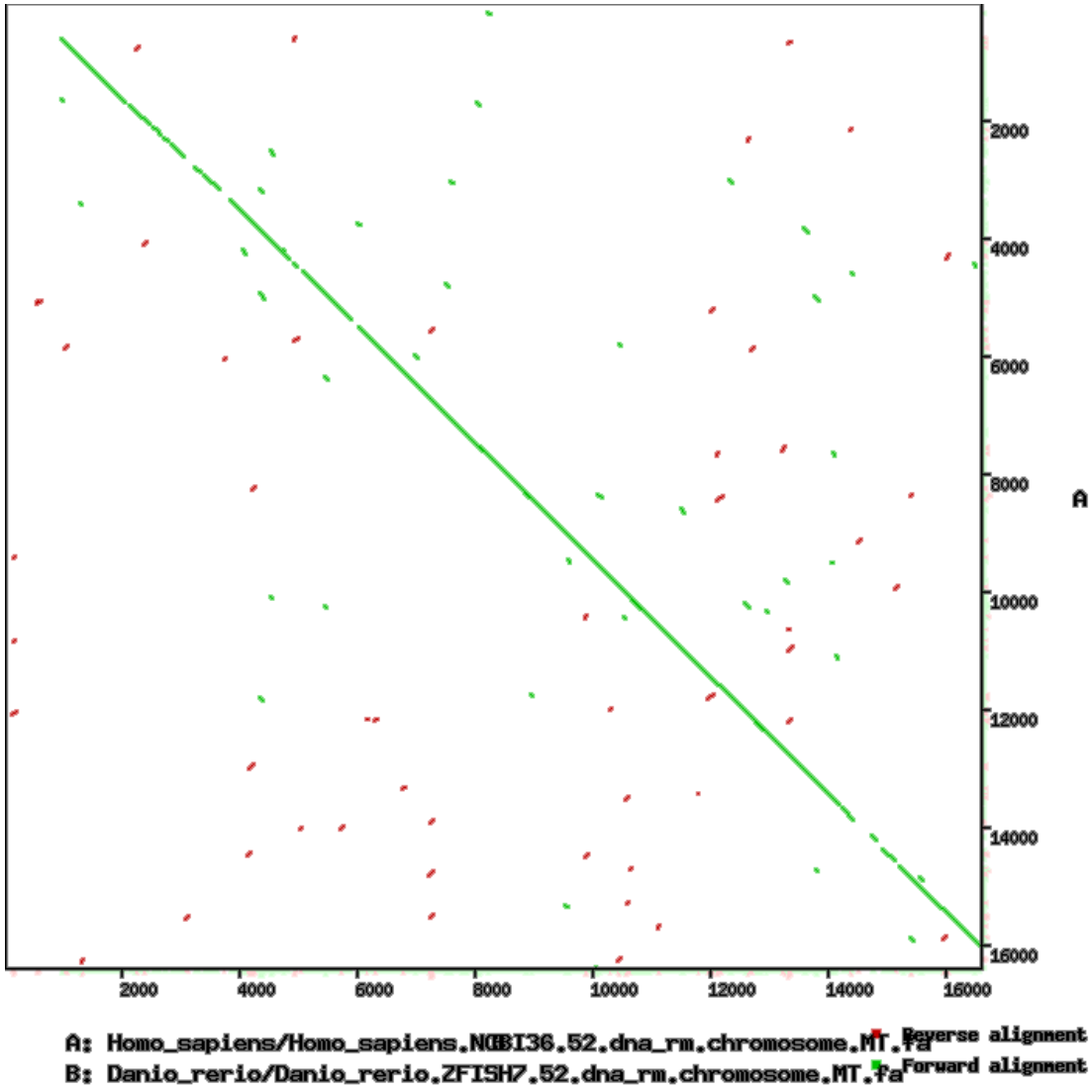
Zarovnávanie sekvencií

(cvičenie)

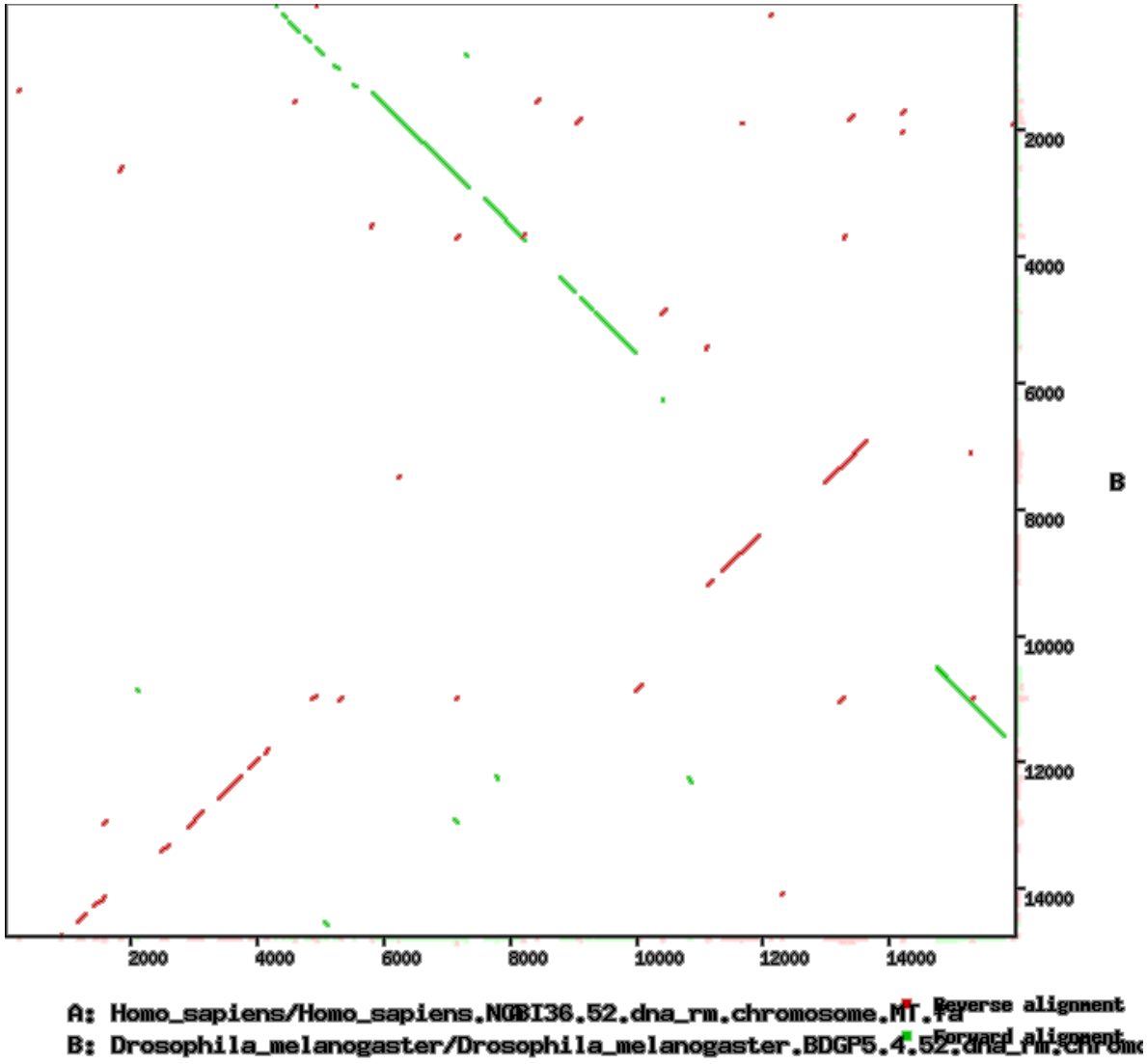
Broňa Brejová

8.10.2021

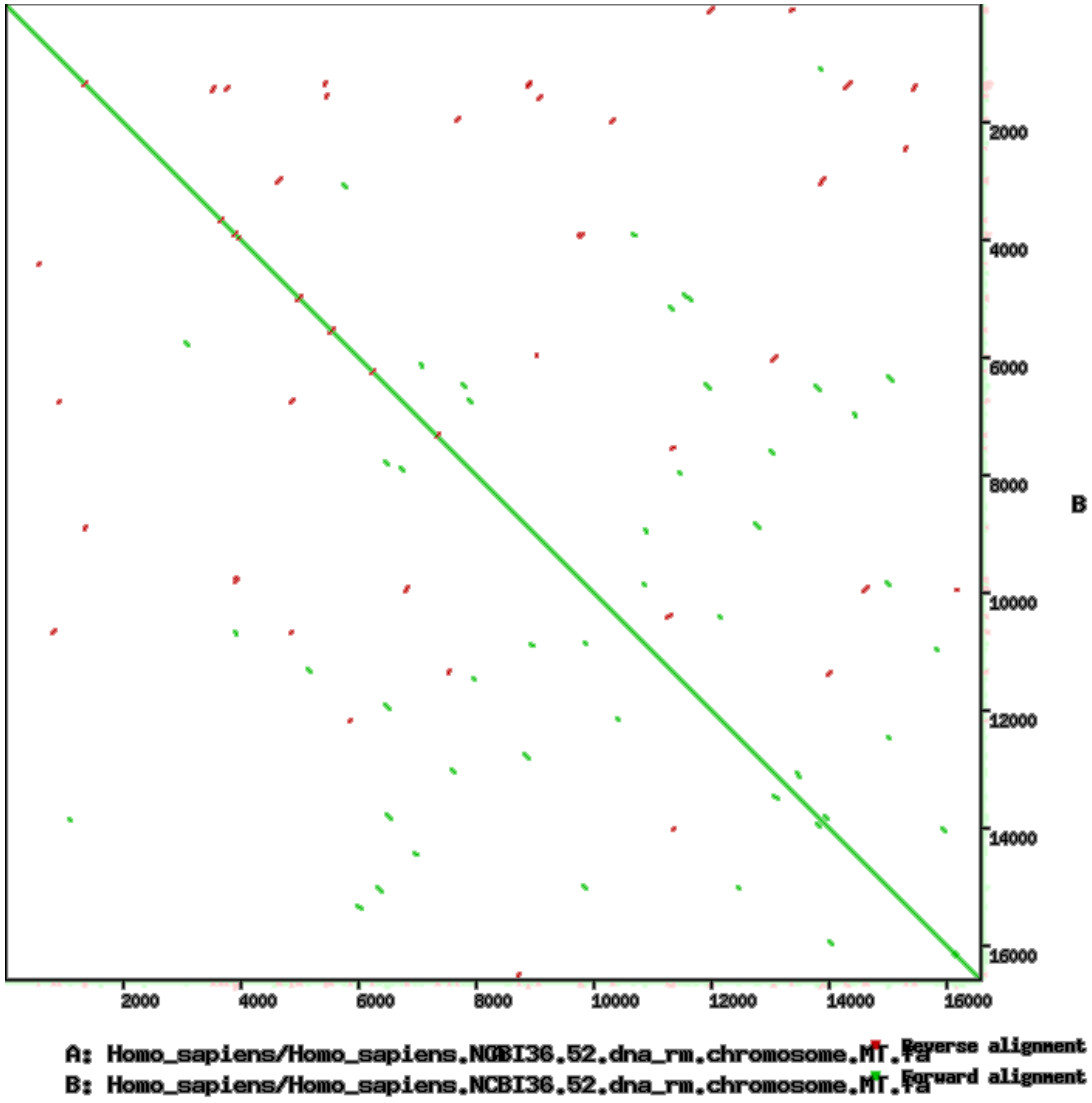
Mitochondriálny genóm človeka vs. ryba Danio rerio

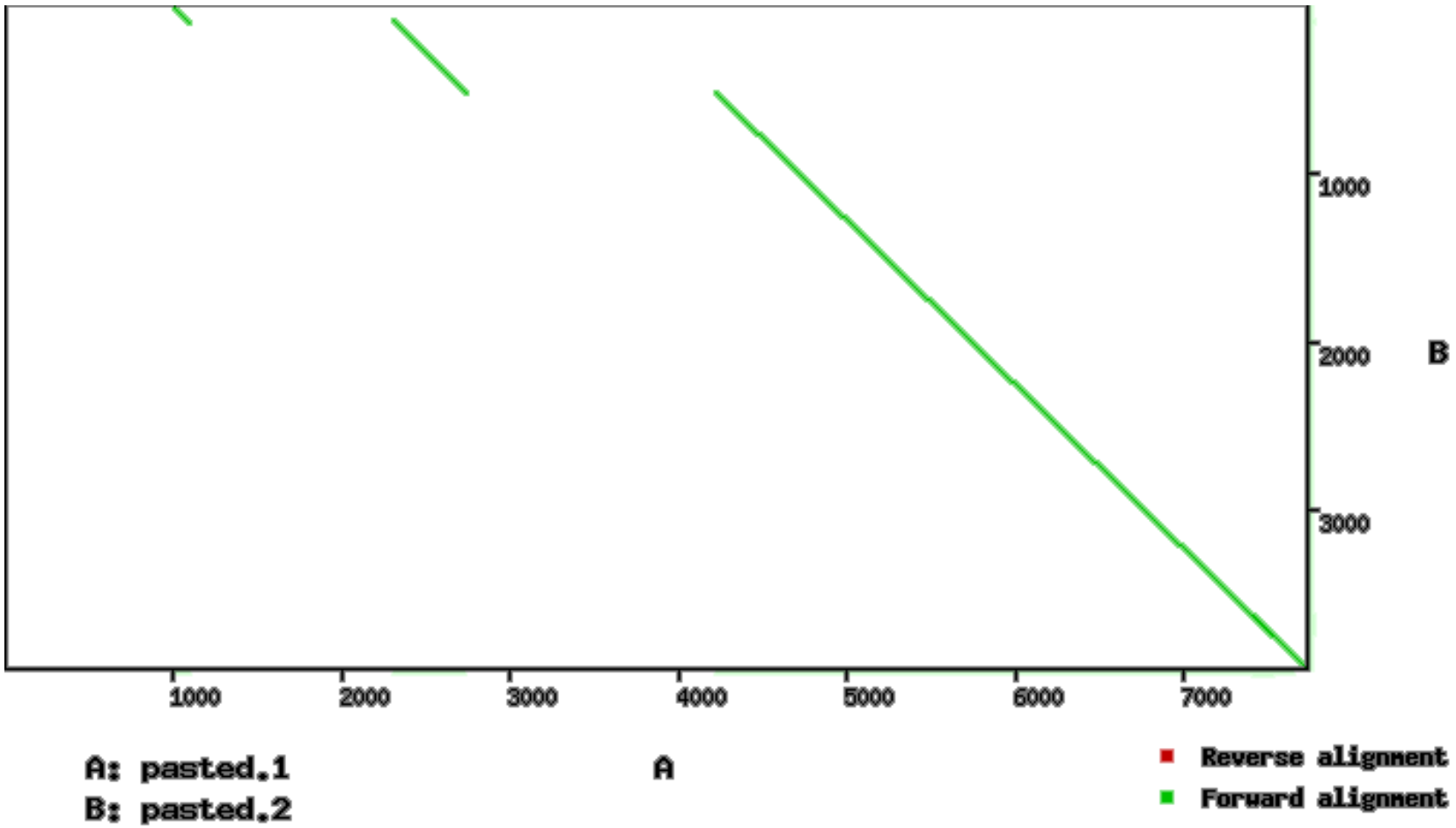


Mitochondriálny genóm človeka vs. Drosophila melanogaster

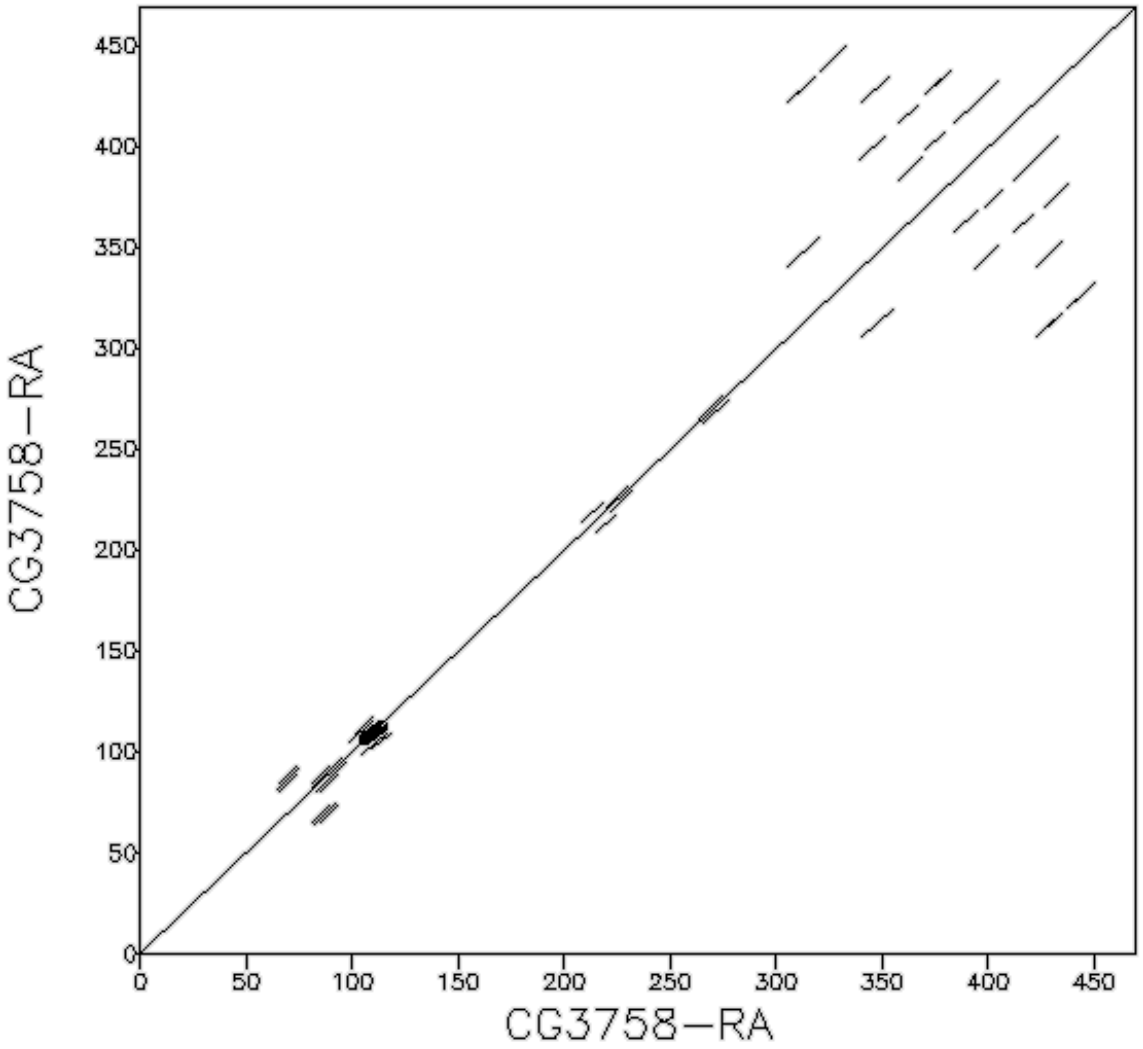


Mitochondriálny genóm človeka vs. to isté




1000
2000
3000
B
1000
2000
3000
4000
5000
6000
7000
A
A: pasted.1
B: pasted.2
Reverse alignment
Forward alignment

Drosophila proteín Escargot zinc finger vs. to isté

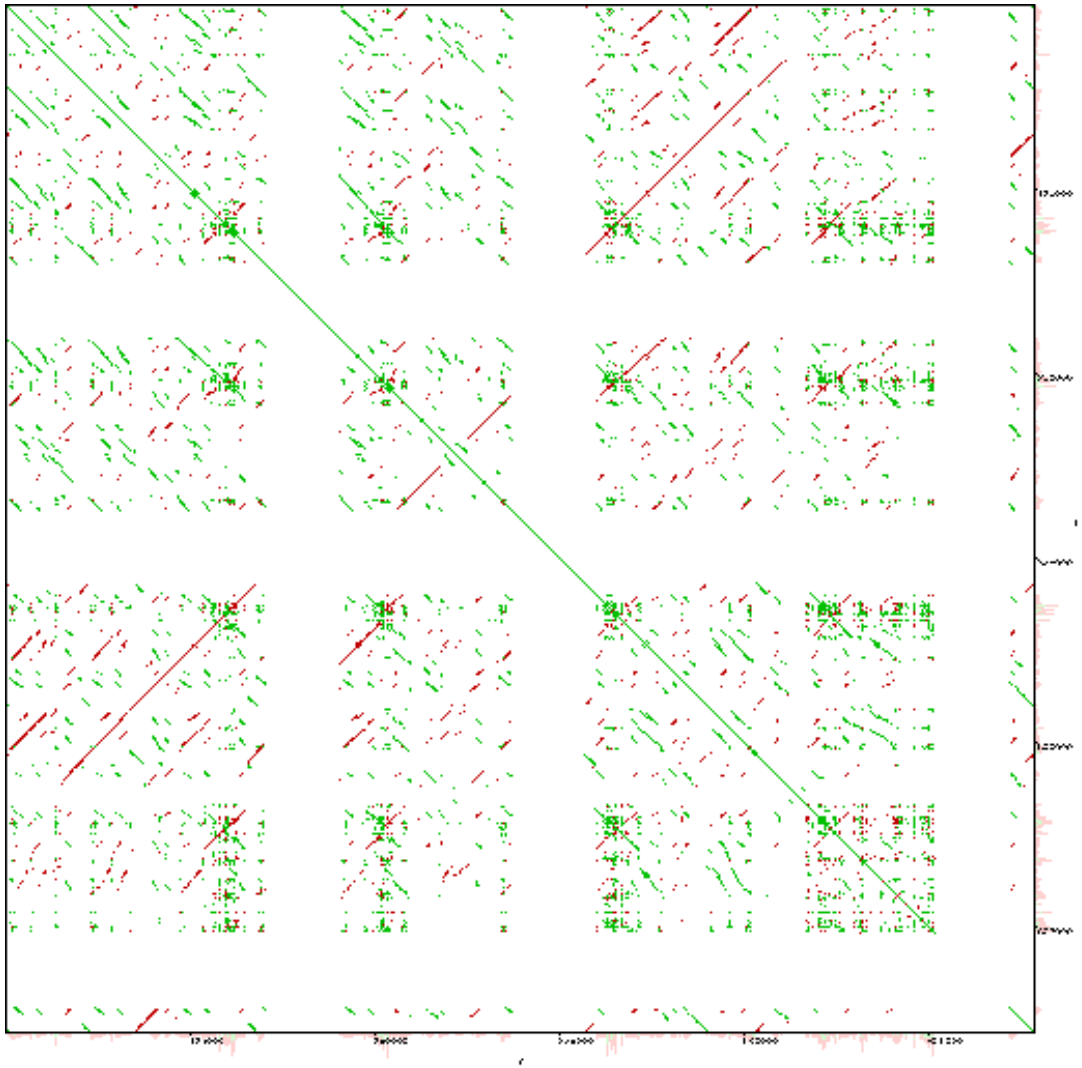


Drosophila proteín Escargot zinc finger

Description:	Protein escargot
Source organism:	Drosophila melanogaster (Fruit fly) View Pfam proteome data.
Length:	470 amino acids

Pfam domains

Source	Domain	Start ▲	End
Pfam B	Pfam-B_18487	8	270
low_complexity		71	95
low_complexity		101	129
low_complexity		163	177
low_complexity		244	263
low_complexity		264	274
Pfam A	zf-C2H2	309	332
Pfam A	zf-C2H2	344	366
Pfam A	zf-C2H2	370	392
Pfam A	zf-C2H2	398	420
low_complexity		445	460



Metódy v bioinformatike

CB05: HMM, E-value

Jana Černíková

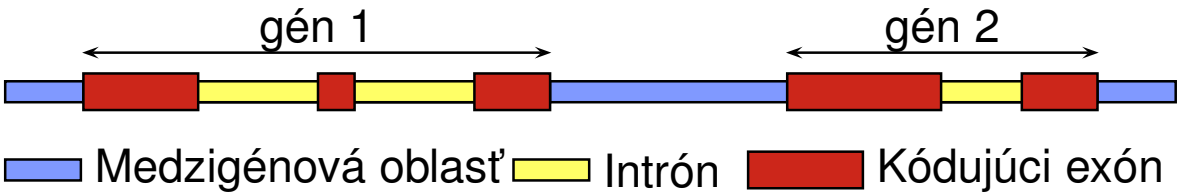
FMFI UK

24/10/2024

Bioinformatický problém: Hľadanie génov

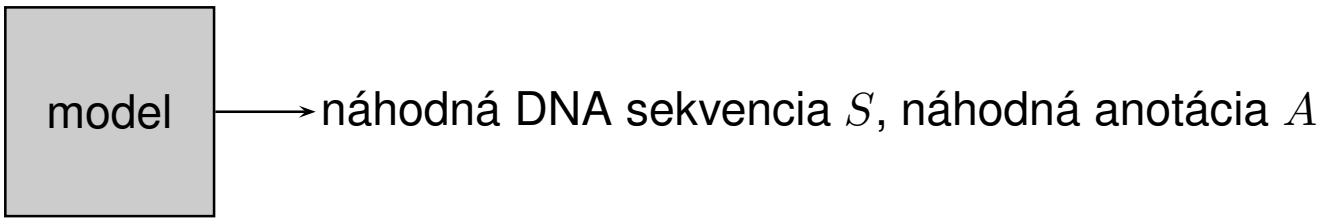
Vstup: DNA sekvencia
Cieľ: označ každú bázu ako intrón/exón/medzigénovú oblasť (anotácia)
Výstup: anotácia s maximálnym skóre
(segmentácia pôvodnej sekvencie na neprekrývajúce sa regióny, ktoré reprezentujú intróny, exóny a medzigénové úseky, pre ktorú dostaneme max. skóre na základe pravdepodobnostného modelu)

cggtgaaactgcacgattgttgctggcttaaagatagaccaatcagagtgtgtaacgtca
tatttagcgtcttctatcatccaatcactgcactttacacactataaatagagcagctca
tgggcgtatttgcgctagtgttgggtgttccgctgtgctgttttccgtcatggctcgca
ctaagcaaactgctcggaagtctactggtggcaaggcgccacgcaaacagttggccacta
aggcagcccgcaaaagcgctccggccaccggcggcgtgaaaaagccccaccgctaccggc
cgggcaccgtggctctgcgcgagatccgccgttatcagaagtccactgaactgcttattc
gtaaactacctttccagcgcctggtgcgcgagattgcgcaggactttaaaacagacctgc
gtttccagagctccgctgtgatggctctgcaggaggcgtgcgaggcctacttggtagggc
tatttgaggacactaacctgtgcgccatccacgccaagcgcgtcactatcatgcccaagg
acatccagctcgcccgccgcatccgcggagagagggcgtgattactgtggtctctctgac



Pravdepodobnostný model génov

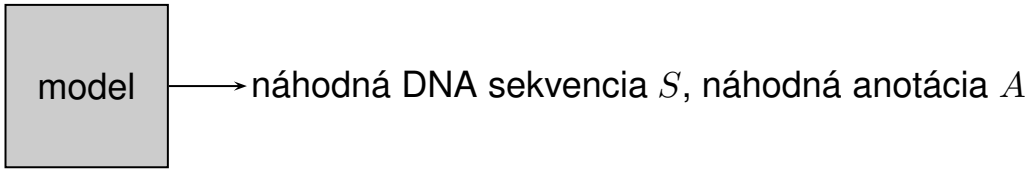
Žiadna informácia nám neumožňuje jednoznačne určiť, čo je gén.
Skombinujeme dostupnú informáciu pravdepodobnostným modelom.



$\Pr(S, A)$ – pravdepodobnosť, že model vygeneruje pár (S, A).
Model zostavíme tak, aby páry s vlastnosťami podobnými skutočným génom mali veľkú pravdepodobnosť.

Použitie: pre novú sekvenciu S nájdi najpravdepodobnejšiu anotáciu $A = \arg\max_A \Pr(A|S)$

Pravdepodobnostný model génov



Použitie: pre sekvenciu S nájdi najpravdepodobnejšiu anotáciu A

Hračkársky príklad modelu: sekvencie dĺžky 2

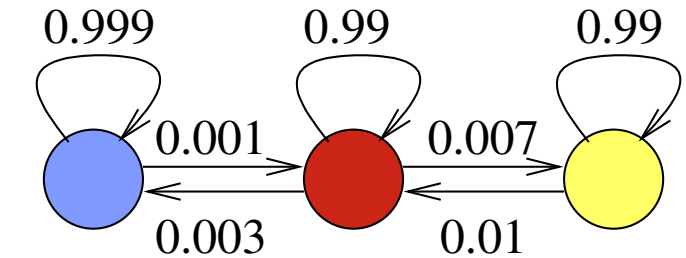
Tabuľka pravdepodobností pre 16 sekvencií, 9 anotácií (súčet 1)

Najpravdepodobnejšia anotácia pre S =aa je aa.

aa	0.008	ac	0.009	ag	0.0085 ...
aa	0	ac	0	...	
aa	0.011	...			
aa	0				
aa	0.009				
aa	0				
aa	0.007				
aa	0				
aa	0.010				

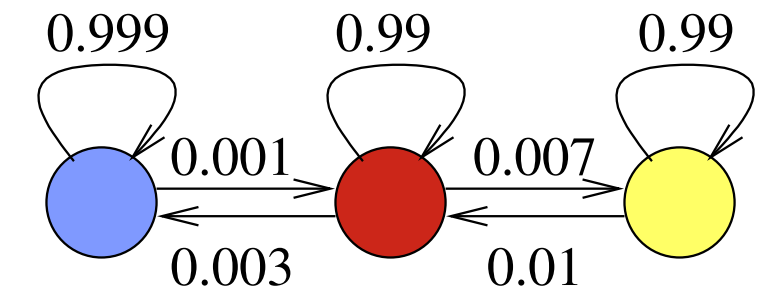
Skrytý Markovov model, hidden Markov model (HMM)

Spôsob, ako zadefinovať model pre dlhšie sekvencie.



- Konečný automat, stavy napr. exón, intrón, medzigénová oblasť
- Sekvenciu aj anotáciu generuje bázu po báze
- V každom kroku je v jednom stave a náhodne vygeneruje jednu bázu podľa tabuľky v stave
- Potom sa presunie do ďalšieho stavu podľa pravdepodobností na hranách

Skrytý Markovov model (HMM)



Predpokladajme, že model vždy začína v modrom stave.

Príklad:

$$\Pr(\text{aca}) = 0.27 \cdot 0.001 \cdot 0.27 \cdot 0.99 \cdot 0.24 = 0.000017$$

$$\Pr(\text{aca}) = 0.27 \cdot 0.999 \cdot 0.23 \cdot 0.999 \cdot 0.27 = 0.017$$

Príklady stavových automatov pre HMM

Uvažujme HMM
so špeciálnym začiatočným stavom b a koncovým stavom e,
ktoré nič negenerujú.

Príklady stavových automatov pre HMM

Uvažujme HMM
so špeciálnym začiatočným stavom b a koncovým stavom e,
ktoré nič negenerujú.

- **Úloha 1:** Nakreslite HMM (stavový diagram), ktorý generuje sekvencie, ktoré začíajú niekoľkými červenými písmenami a potom obsahujú niekoľko modrých

Príklady stavových automatov pre HMM

Uvažujme HMM
so špeciálnym začiatočným stavom b a koncovým stavom e,
ktoré nič negenerujú.

- **Úloha 1:** Nakreslite HMM (stavový diagram), ktorý generuje sekvencie, ktoré začíajú niekoľkými červenými písmenami a potom obsahujú niekoľko modrých
- **Úloha 2:** Ako treba zmeniť HMM, aby dovoloval ako "niekoľko" aj nula?

Príklady stavových automatov pre HMM

Uvažujme HMM
so špeciálnym začiatočným stavom b a koncovým stavom e,
ktoré nič negenerujú.

- **Úloha 1:** Nakreslite HMM (stavový diagram), ktorý generuje sekvencie, ktoré začíajú niekoľkými červenými písmenami a potom obsahujú niekoľko modrých
- **Úloha 2:** Ako treba zmeniť HMM, aby dovoloval ako "niekoľko" aj nula?
- **Úloha 3:** Ako treba zmeniť HMM, aby počet červených aj modrých bol vždy parne číslo?

Príklady stavových automatov pre HMM

Uvažujme HMM
so špeciálnym začiatočným stavom b a koncovým stavom e,
ktoré nič negenerujú.

- **Úloha 1:** Nakreslite HMM (stavový diagram), ktorý generuje sekvencie, ktoré začíajú niekoľkými červenými písmenami a potom obsahujú niekoľko modrých
- **Úloha 2:** Ako treba zmeniť HMM, aby dovoloval ako "niekoľko" aj nula?
- **Úloha 3:** Ako treba zmeniť HMM, aby počet červených aj modrých bol vždy parne číslo?
- **Úloha 4:** Ako zmeniť HMM, aby sa striedali červené a modré kusy párnej dĺžky?

Príklady stavových automatov pre HMM

V ďaľších príkladoch uvažujeme aj to, ktoré písmená su v ktorom stave povolené (pravdepodobnosť emisie > 0) a ktoré sú zakázané

Príklady stavových automatov pre HMM

V ďaľších príkladoch uvažujeme aj to, ktoré písmená su v ktorom stave povolené (pravdepodobnosť emisie > 0) a ktoré sú zakázané

- **Úloha 5:** Model generujúci červené sekvencie dĺžky dva, ktoré začínajú na A

Príklady stavových automatov pre HMM

V ďaľších príkladoch uvažujeme aj to, ktoré písmená su v ktorom stave povolené (pravdepodobnosť emisie > 0) a ktoré sú zakázané

- **Úloha 5:** Model generujúci červené sekvencie dĺžky dva, ktoré začínajú na A
- **Úloha 6:** Model generujúci červené sekvencie dĺžky dva, ktoré môzu byť čokoľvek iné ako AA

Príklady stavových automatov pre HMM

V ďalších príkladoch uvažujeme aj to, ktoré písmená su v ktorom stave povolené (pravdepodobnosť emisie > 0) a ktoré sú zakázané

- **Úloha 5:** Model generujúci červené sekvencie dĺžky dva, ktoré začínajú na A
- **Úloha 6:** Model generujúci červené sekvencie dĺžky dva, ktoré môzu byť čokoľvek iné ako AA
- **Úloha 7:** Rozšírte predošlý model na sekvencie dĺžky 3 bázy, tak aby to nemohli byť stop kodóny TAA, TAG, TGA

Príklady stavových automatov pre HMM

V ďalších príkladoch uvažujeme aj to, ktoré písmená su v ktorom stave povolené (pravdepodobnosť emisie > 0) a ktoré sú zakázané

- **Úloha 5:** Model generujúci červené sekvencie dĺžky dva, ktoré začínajú na A
- **Úloha 6:** Model generujúci červené sekvencie dĺžky dva, ktoré môzu byť čokoľvek iné ako AA
- **Úloha 7:** Rozšírte predošlý model na sekvencie dĺžky 3 bázy, tak aby to nemohli byť stop kodóny TAA, TAG, TGA

  toto sa dá rozšíriť na HMM, ktorý reprezentuje ORF (open reading frame): začína štart kodónom, potom niekoľko bežných kodónov, ktoré nie sú stop kodónom a na koniec stop kodón

Iný príklad použitia HMM: Topológia transmembránových proteínov

Chceme označiť aminokyseliny v proteíne – vonku z bunky, v membráne, vo vnútri bunky
Nie každá postupnosť označení dáva zmysel – napr. vonku—vnútri alebo vonku—v_membráne—vonku

Iný príklad použitia HMM: Topológia transmembránových proteínov

Chceme označiť aminokyseliny v proteíne – vonku z bunky, v membráne, vo vnútri bunky
Nie každá postupnosť označení dáva zmysel – napr. vonku—vnútri alebo vonku—v_membráne—vonku

Čo by reprezentovali stavy v tomto prípade?

E-value: Hračkársky prípad

Dotaz: ATGCTCAAAC (dĺžka $m = 10$)
Databáza: (dĺžka $n = 300$)

accacttgcgcacgatttccagattcggtttccctgggcgcacgaagggc
ccacgaagcgGCTCAACccggagccttagttagaaggggggtctccgtca
agagagacggtaagttggagggtcactagcggtggactccgaatggaaac
actgaatagtggcagaacctaaacctcgttttggatttcctgaaaaaggc
aggcgctagaggaagaggcacgactgtgctagagataatcacttgtaaga
ccttggggatgggcttcgtatgcagaacgcgataaggtatcgaaaacgtg

Skórovacia schéma: zhoda $+1$, nezhoda -1, medzera -1
Lokálne zarovnanie so skóre $S = 6$

GCTCAAAC
GCTCA-AC

E-value: koľko očakávame lokálnych zarovnaní so skóre aspoň S v náhodnej databáze dĺžky n pri náhodnom dotaze dĺžky m

Náhodný dotaz a databáza

Dotaz: GTGCCTGCAG

Databáza:

cctctgatagccttgaaccgggcgagactcatacagacagtgctcctcgg
gcgataaccatgagatgacaggtccgatgctaatgttaacggacctacag
tgacatgttaaagtgtccattaagtttataccggaatcaacgagtgtccc
ccagcgcggcgaccgatggagccCCTGCAGgtatactcacttcaaggatt
accgctcggtgtaagttagtgttcagtcagactatactaagtattcagtt
atagagcgttagtaggtcgaccatgagcgggtaggGTGCCGAGatgtgaa

Počet výskytov: 2

Náhodný dotaz a databáza

Dotaz: `TCGACCGAAA`

Databáza:

```
tactccattagggattataacgactaaagcccgtcgtggcgggatcactt
tgagattcaactttaacgcatcacagaggaatctgagacaaagcaaaacc
gatcataatgatcgatccaggtaataagtctccttgatggcgttagactg
gaaataacagttgacttccgactatagtttaatgaacgttcgtaattaga
cgatcgtgtaacttaaccaaaggctgcccccaaactagctgagtaatagc
tcgtcctgagcatgtaagagtcagcctccacggaacactgcaacgttctt
```

Počet výskytov: 0

Náhodný dotaz a databáza

Dotaz: CCCGTCGTAG

Databáza:

cagcattagccccgttatttCGTCGTtctccaacgggtctgcctttctgg
aacgtggcgaaccttcacaggtcagtctgtcatcgcctgcgcttagagcg
gacggtactcgaaaggtcggttcagtgtggcgctggaaagaagaatagca
acacatgcactaatggaaggtcccagtggtgtgggacattctggaCCCGT
GTgtgccaacctatgtgagctccggcgttgactcggaggatgttaacaag
atcaagctgtaggcgacgatccccgccgggtttcctctactgcctcgagc

Počet výskytov: 2

Náhodný dotaz a databáza

Dotaz: AGGATGAGGA

Databáza:

ttatcgattctccggtgcgccagtacagcacaaggctcggatcctgtaaa
acactacaccttaaaaactaagtcAGGATGtgatctcccttaaGATGAGa
cagtctctaatgcggcgtagtgggaccctcgtgaccgagctaagcagttc
acaatgggcgctctgagcgattggctggagaccttgacttcccggtaggt
gtggtgttagttctgtgcccagagataaccatccaccgtaatggatctcg
taactttacGATGAAGAccggcatcatctcagttatatttctaggacggg

Počet výskytov: 3

Celkovo opakujeme 100 krát

$S = 6$, $m = 10$, $n = 300$, obsah GC 50%

Počet výskytov: 2, 0, 2, 3, 3, 1, 0, 1, 1, 1, 0, 0, 4, 2, 0, 1, 0, 1, 0, 0, 1, 0, 0, 4, 3, 1, 1, 0, 0, 0, 2, 3, 0, 0, 2, 1, 1, 1, 0, 0, 0, 0, 4, 1, 1, 0, 0, 1, 1, 1, 2, 2, 2, 0, 0, 2, 0, 1, 1, 0, 1, 2, 2, 1, 0, 0, 1, 1, 2, 0, 1, 0, 0, 1, 0, 3, 2, 0, 2, 2, 1, 0, 0, 2, 0, 0, 1, 2, 1, 1, 3, 2, 2, 1, 1, 0, 2, 0, 1, 3

Priemerný počet výskytov: 1.05

Keď celé opakujeme viackrát, dostávame hodnoty 0.99, 1.15, 1.02, 1.07, 0.98, . . .

Správna hodnota **E-value:** 0.99

K-means clustering

Tomáš Vinař

21.11.2024

Formulácia problému

Vstup: n-rozmerné vektory $x_1, x_2, \ldots, x_t$ a počet zhlukov k

Výstup: Rozdelenie vektorov do k zhlukov:

- priradenie vstupných vektorov do zhlukov zapísané ako čísla $c_1, c_2, \ldots, c_t$, kde $c_i \in \{1, 2, \ldots, k\}$ je číslo zhluku pre x_i
- centrum každého zhluku, t.j. n-rozmerné vektory $\mu_1, \mu_2, \ldots, \mu_k$

Hodnoty $c_1, \ldots, c_t$ a $\mu_1, \ldots, \mu_k$ volíme tak, aby sme minimalizovali súčet štvorcov vzdialeností od každého vektoru k centru jeho zhluku:

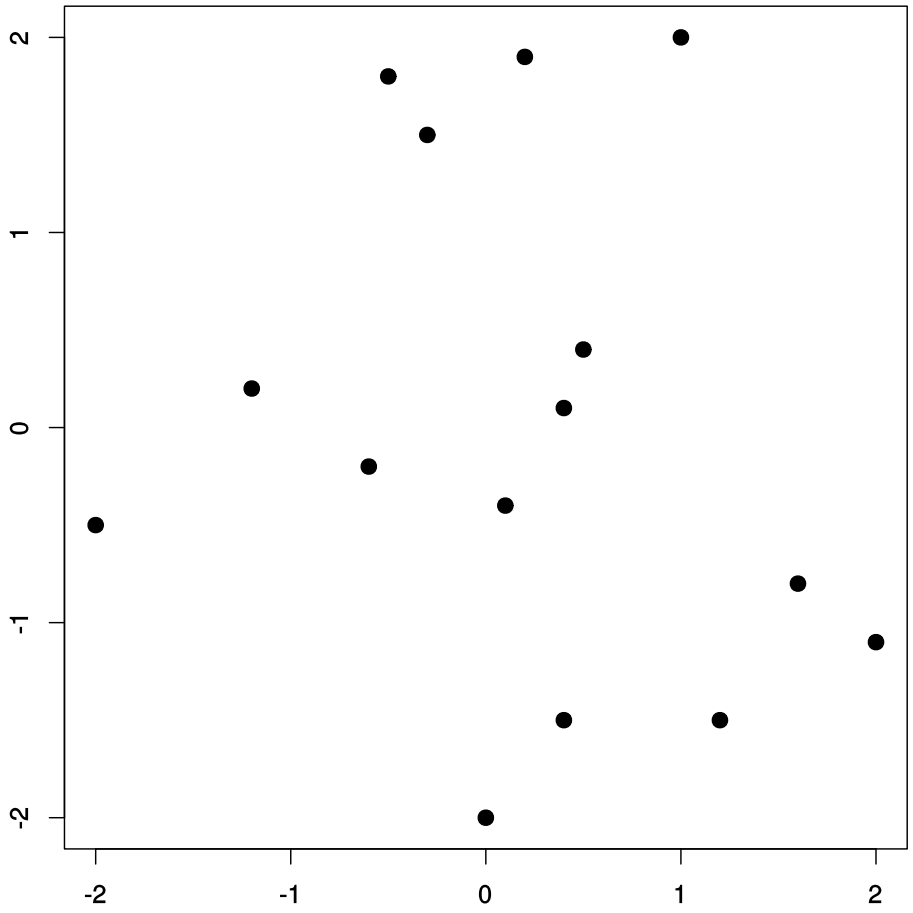
$$\sum_{i=1}^{t} \left\| x_i - \mu_{c_i} \right\|_2^2$$

Pre vektory $a = (a_1, \ldots, a_n)$ a $b = (b_1, \ldots b_n)$ je druhá mocnina vzdialenosti $\left\| a - b \right\|_2^2 = \sum_{i=1}^{n} (a_i - b_i)^2$

Príklad vstupu

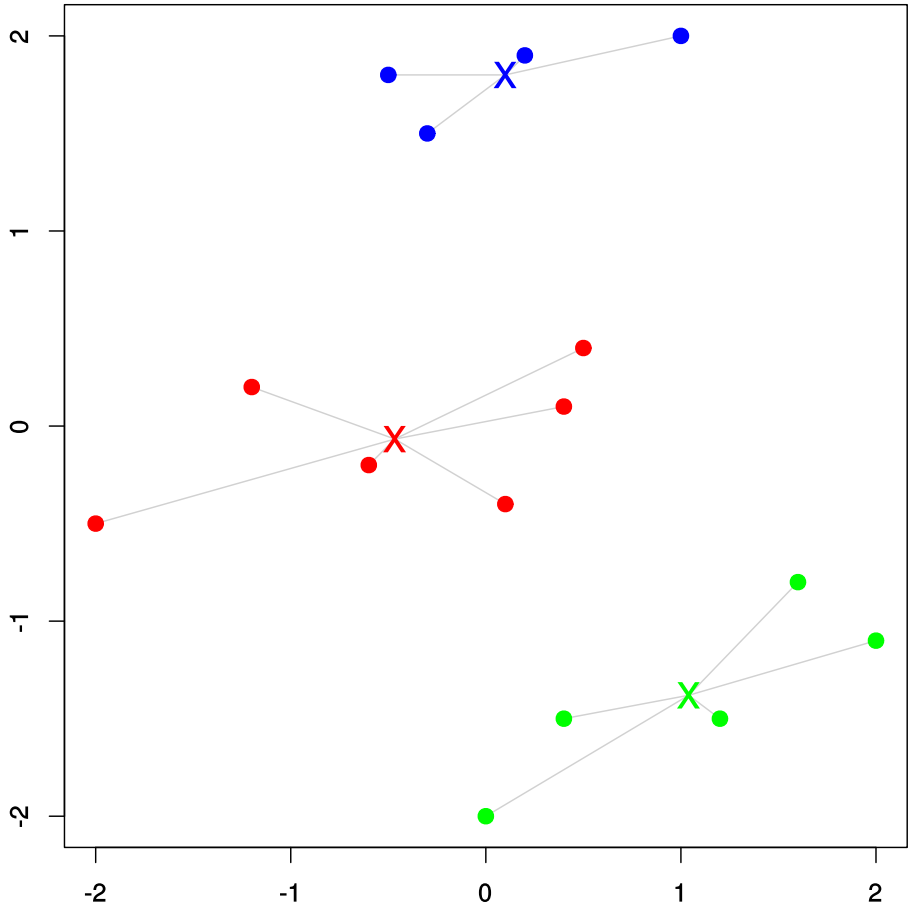
x_1	-2.00	-0.50
x_2	-1.20	0.20
x_3	-0.60	-0.20
x_4	-0.50	1.80
x_5	-0.30	1.50
x_6	0.00	-2.00
x_7	0.10	-0.40
x_8	0.20	1.90
x_9	0.40	0.10
x_{10}	0.40	-1.50
x_{11}	0.50	0.40
x_{12}	1.00	2.00
x_{13}	1.20	-1.50
x_{14}	1.60	-0.80
x_{15}	2.00	-1.10

$k = 3$



Príklad výstupu

x_1	-2.00	-0.50	**1**
x_2	-1.20	0.20	**1**
x_3	-0.60	-0.20	**1**
x_4	-0.50	1.80	**3**
x_5	-0.30	1.50	**3**
x_6	0.00	-2.00	**2**
x_7	0.10	-0.40	**1**
x_8	0.20	1.90	**3**
x_9	0.40	0.10	**1**
x_{10}	0.40	-1.50	**2**
x_{11}	0.50	0.40	**1**
x_{12}	1.00	2.00	**3**
x_{13}	1.20	-1.50	**2**
x_{14}	1.60	-0.80	**2**
x_{15}	2.00	-1.10	**2**
μ_1	**-0.47**	**-0.07**	
μ_2	**1.04**	**-1.38**	
μ_3	**0.10**	**1.80**	



Sum of squared distances: 10.61

Algoritmus

Heuristika, ktorá nenájde vždy najlepšie zhlukovanie.

Začne z nejakého zhlukovania a postupne ho zlepšuje.

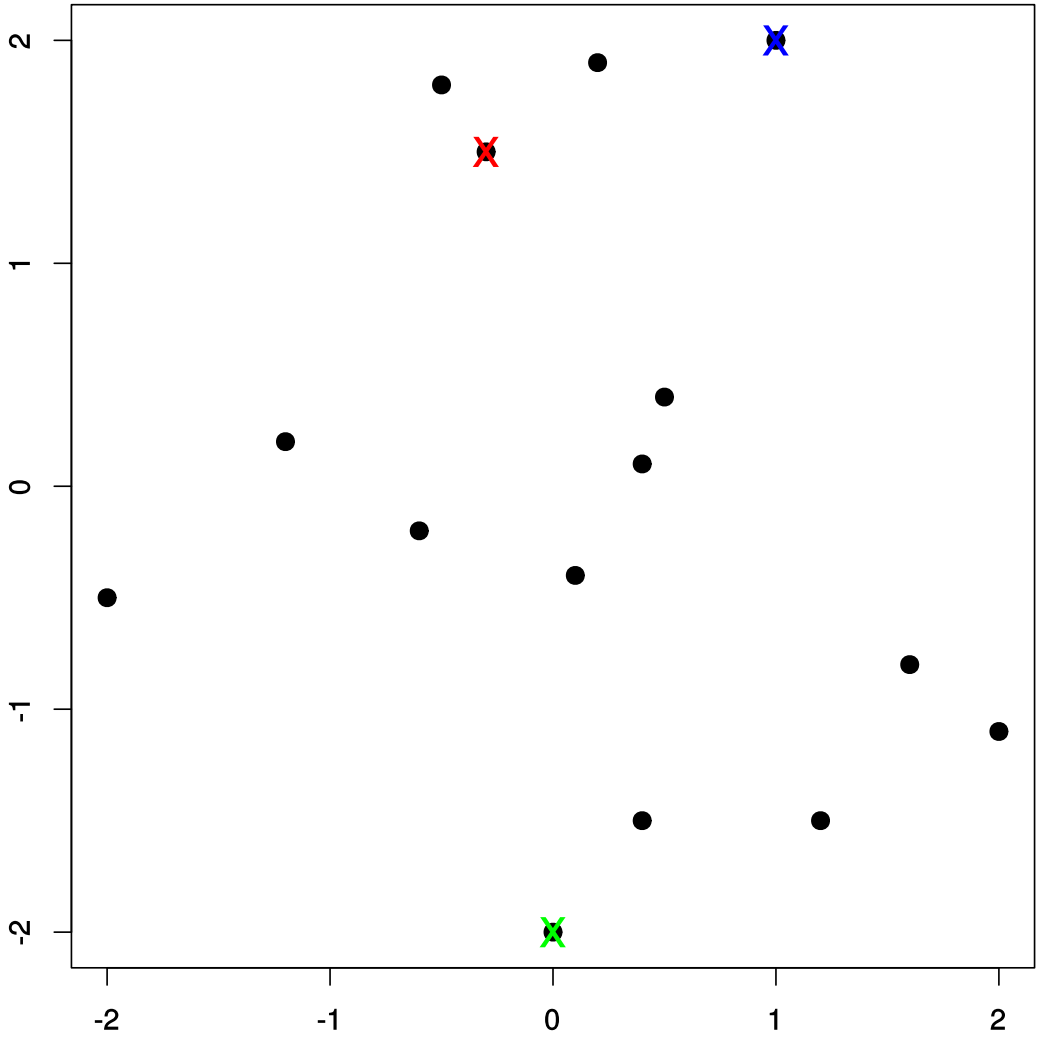
Inicializácia:

náhodne vyber k centier $\mu_1, \mu_2, \ldots, \mu_k$ spomedzi vstupných vektorov

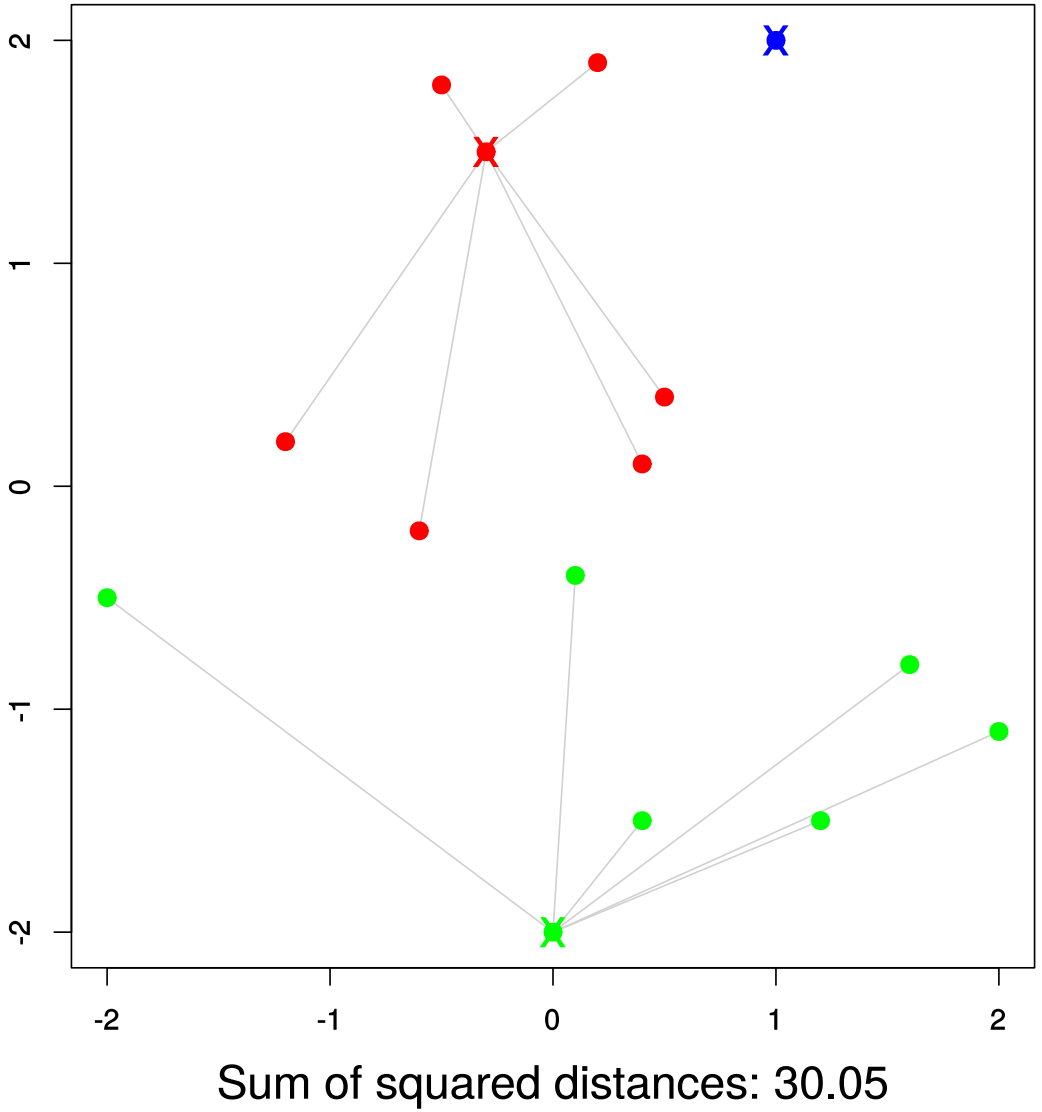
Opakuj, kým sa niečo mení:

- priraď každý bod najbližšiemu centru: $c_i = \arg\min_j \left\| x_i - \mu_j \right\|_2$
- vypočítaj nové centrá: μ_j bude priemerom (po zložkách) z vektorov x_i, pre ktoré $c_i = j$

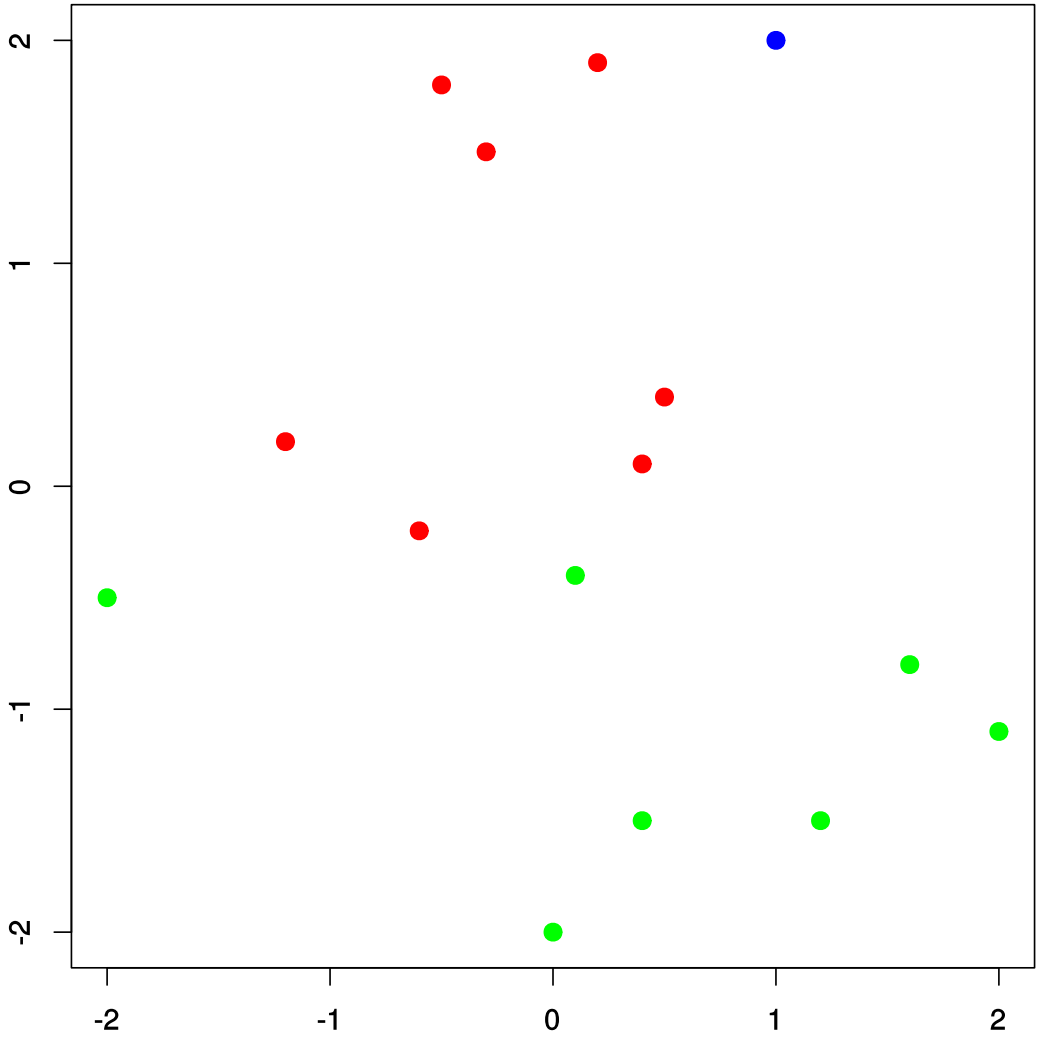
Zvolíme náhodné centrá μ_i



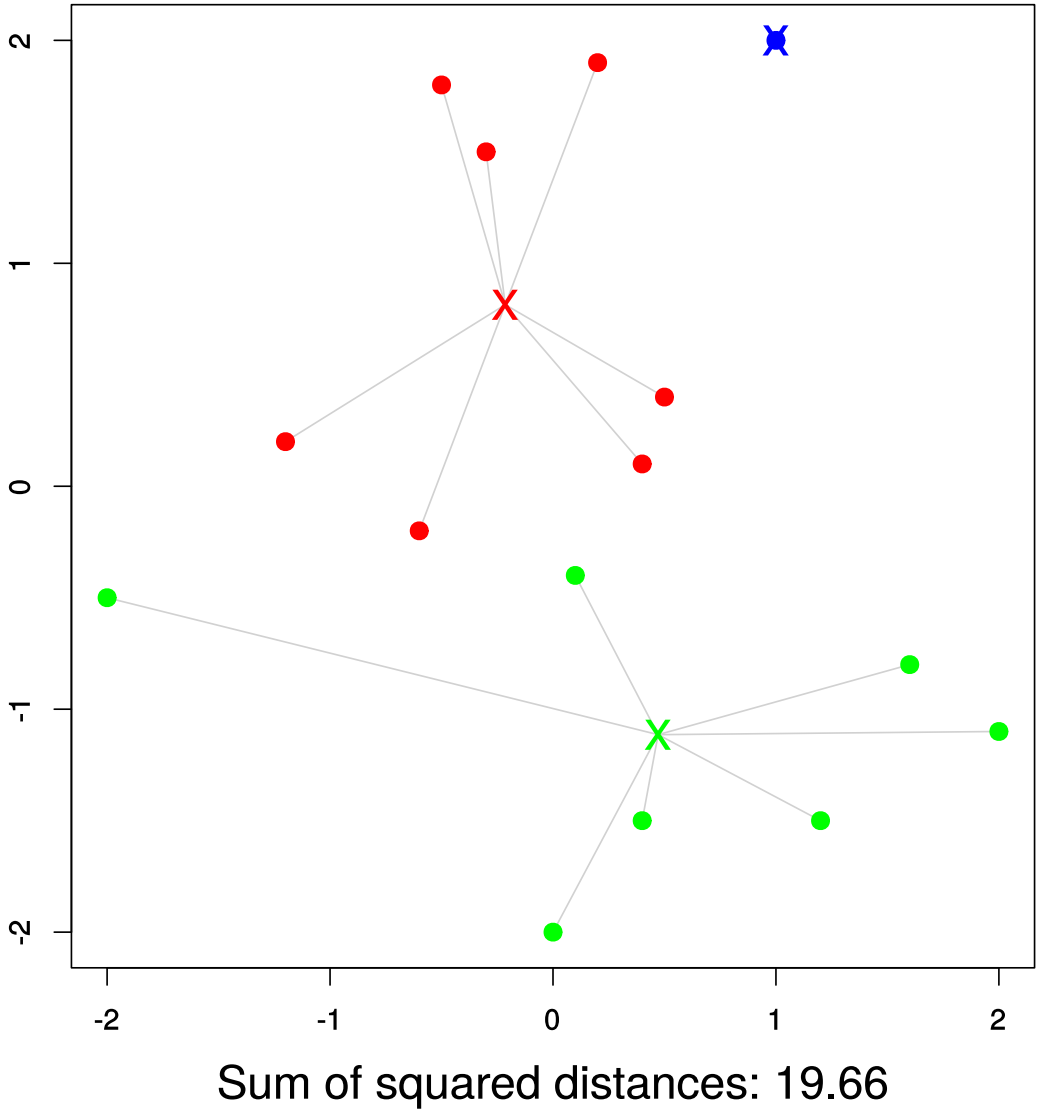
Vektory priradíme do zhlukov (hodnoty c_i)



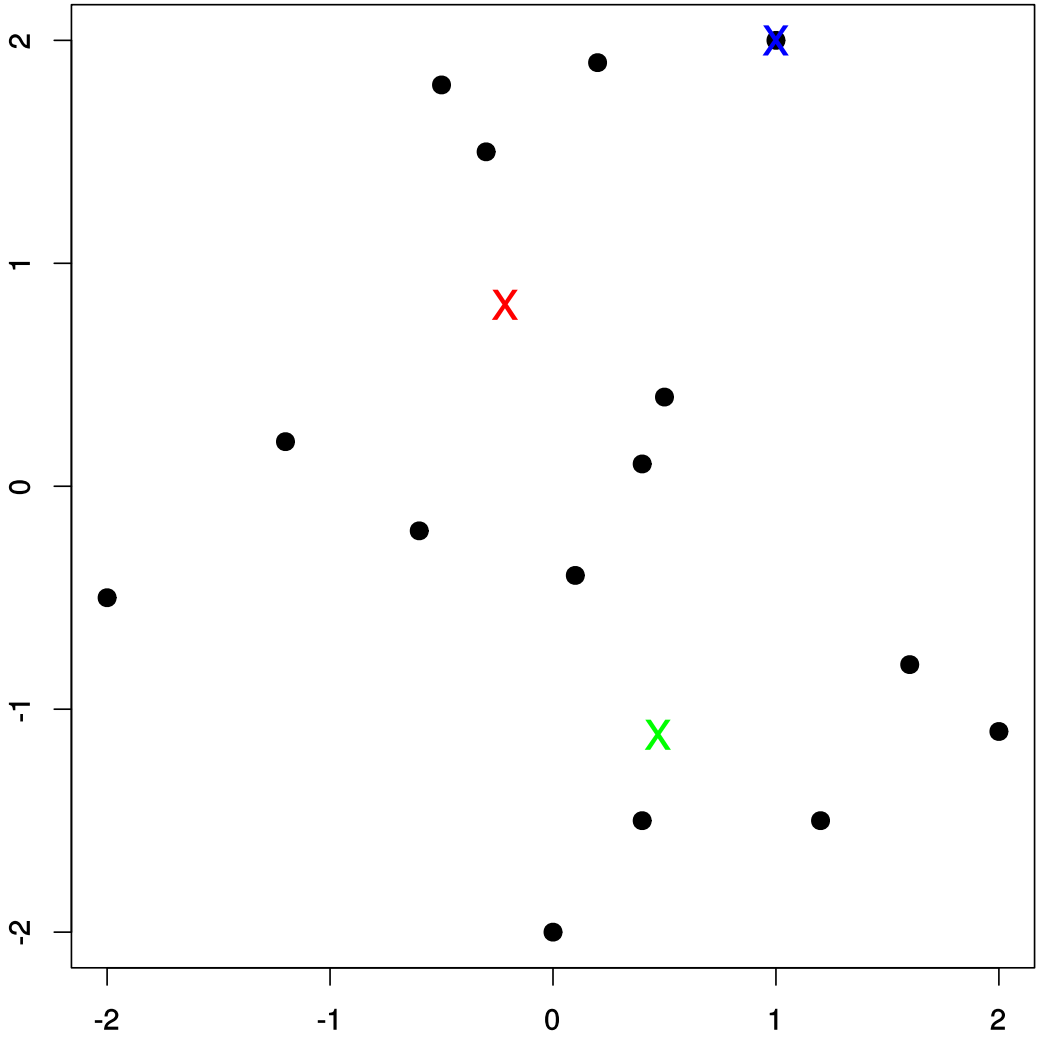
Zabudneme μ_i



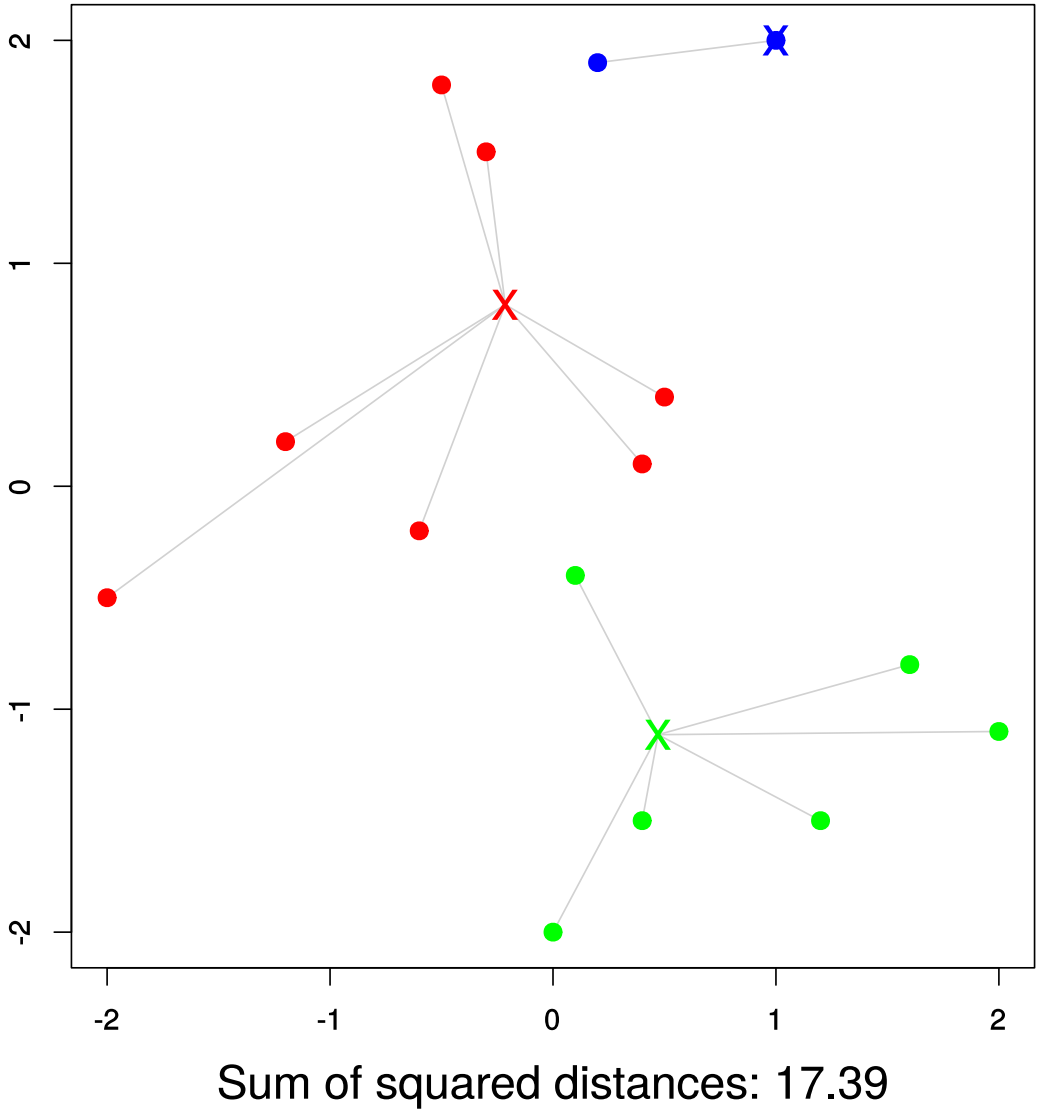
Dopočítame nové μ_i (suma klesla z 30.05 na 19.66)



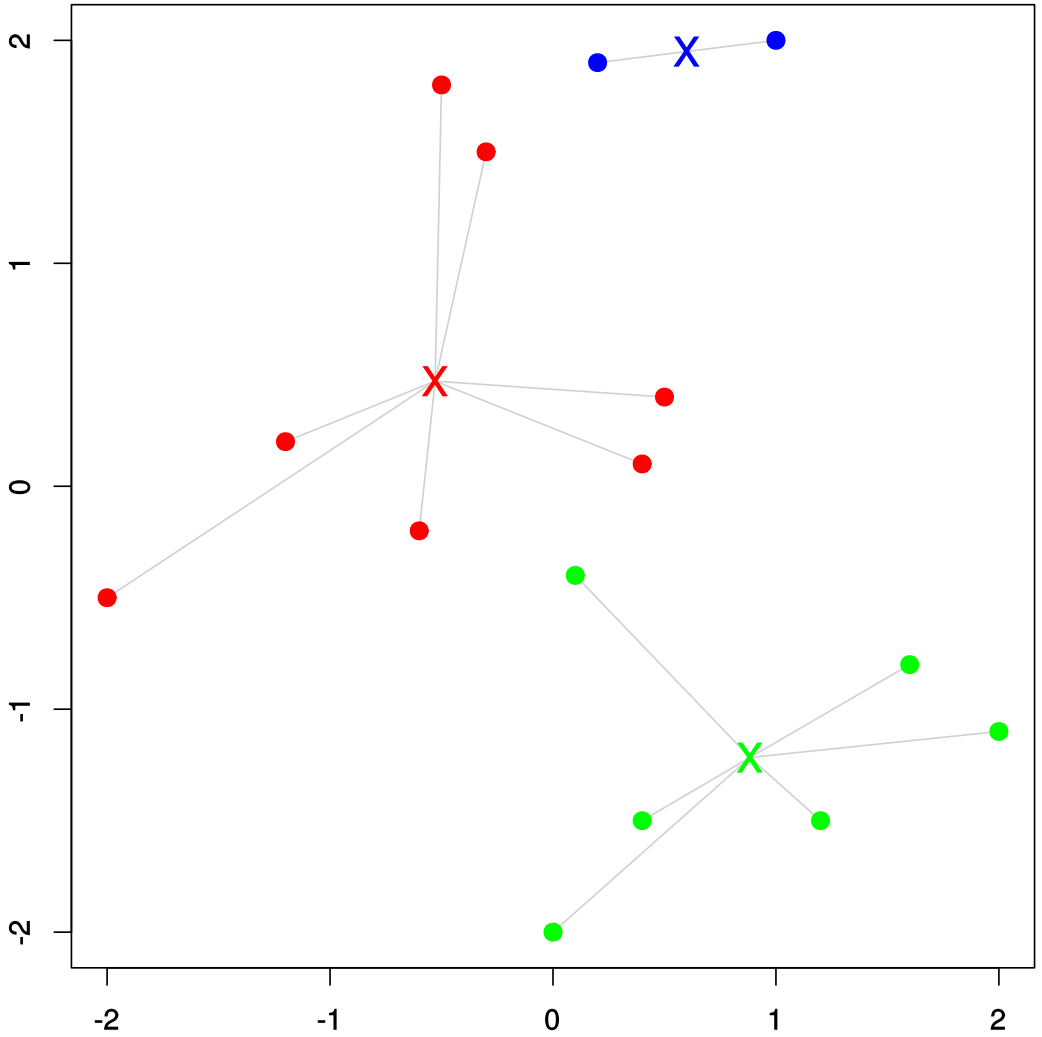
Zabudneme c_i



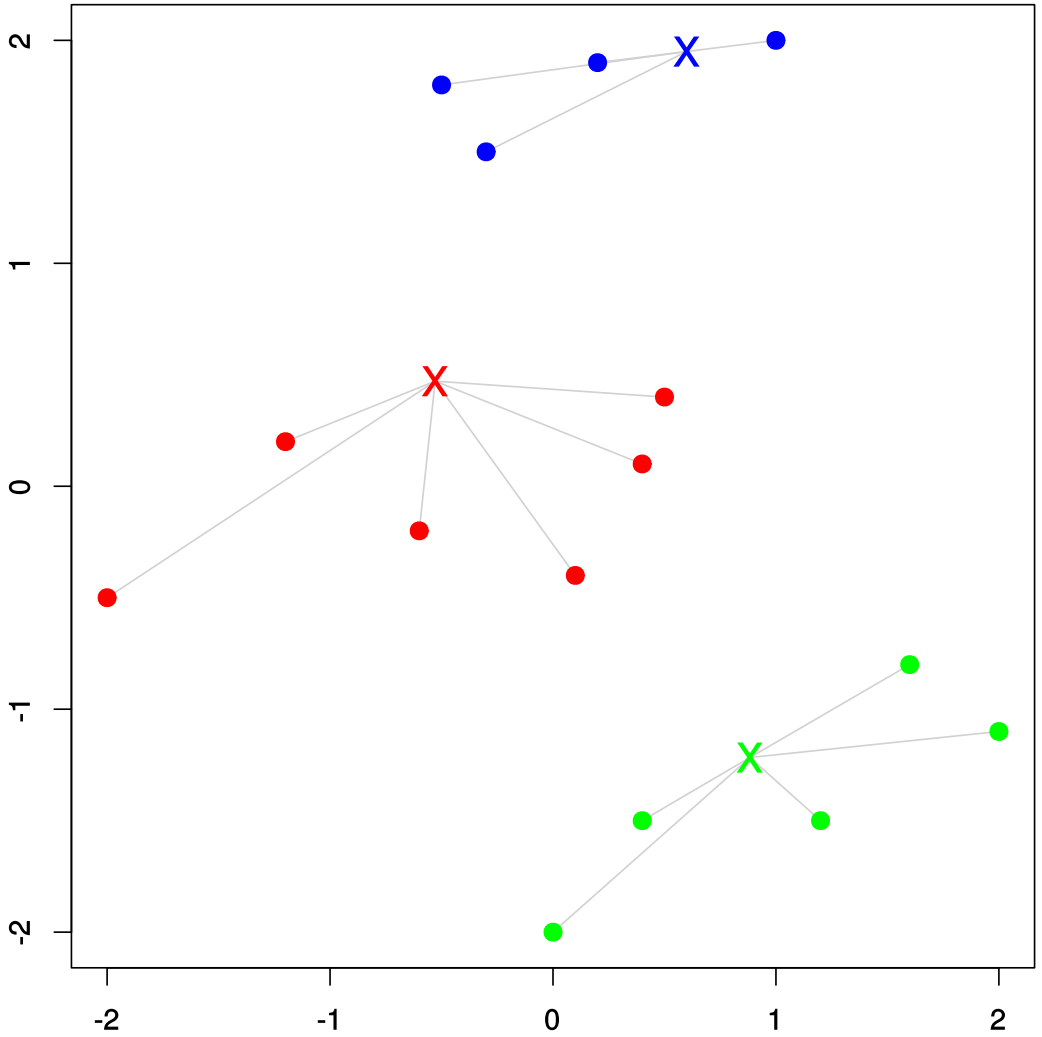
Dopočítame nové c_i (suma klesla z 19.66 na 17.39)



Prepočítame μ_i

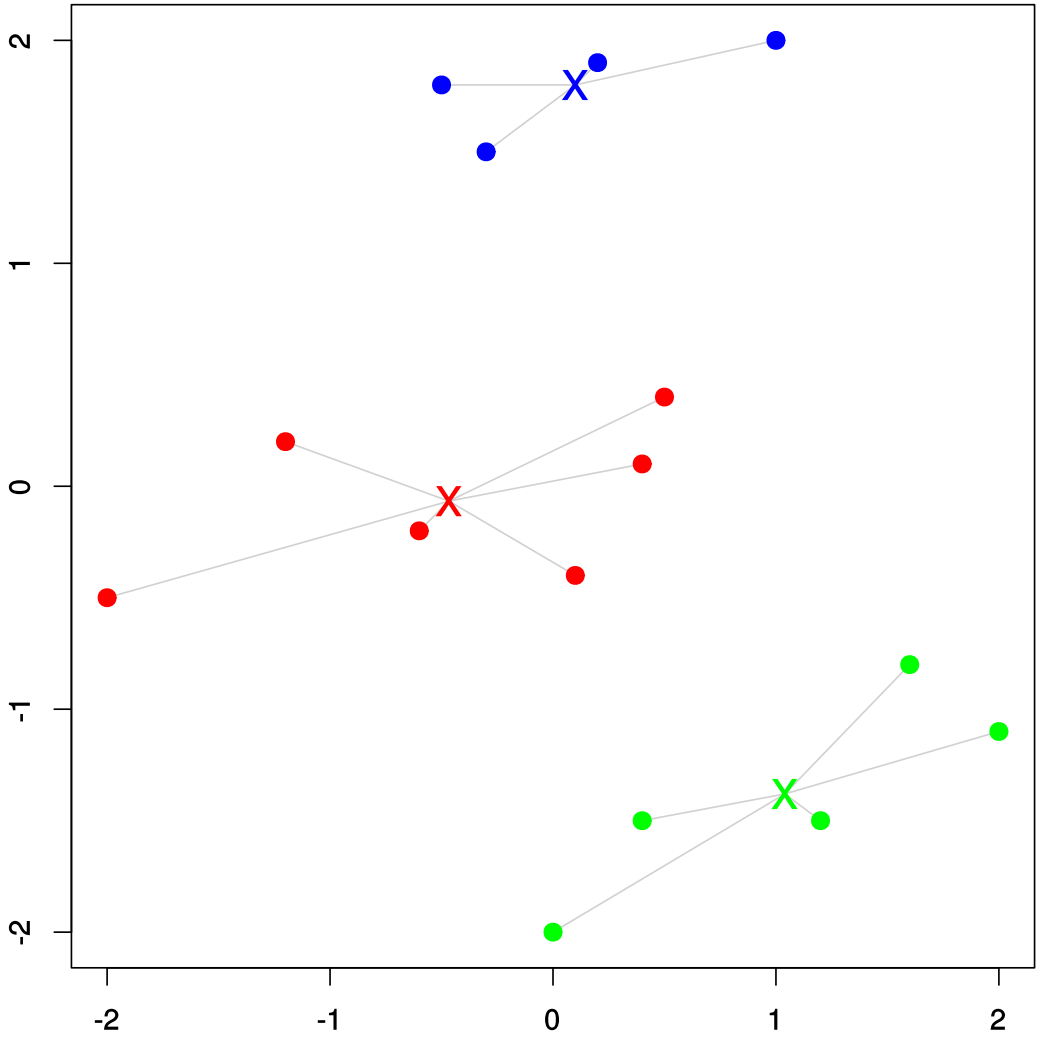


Prepočítame c_i

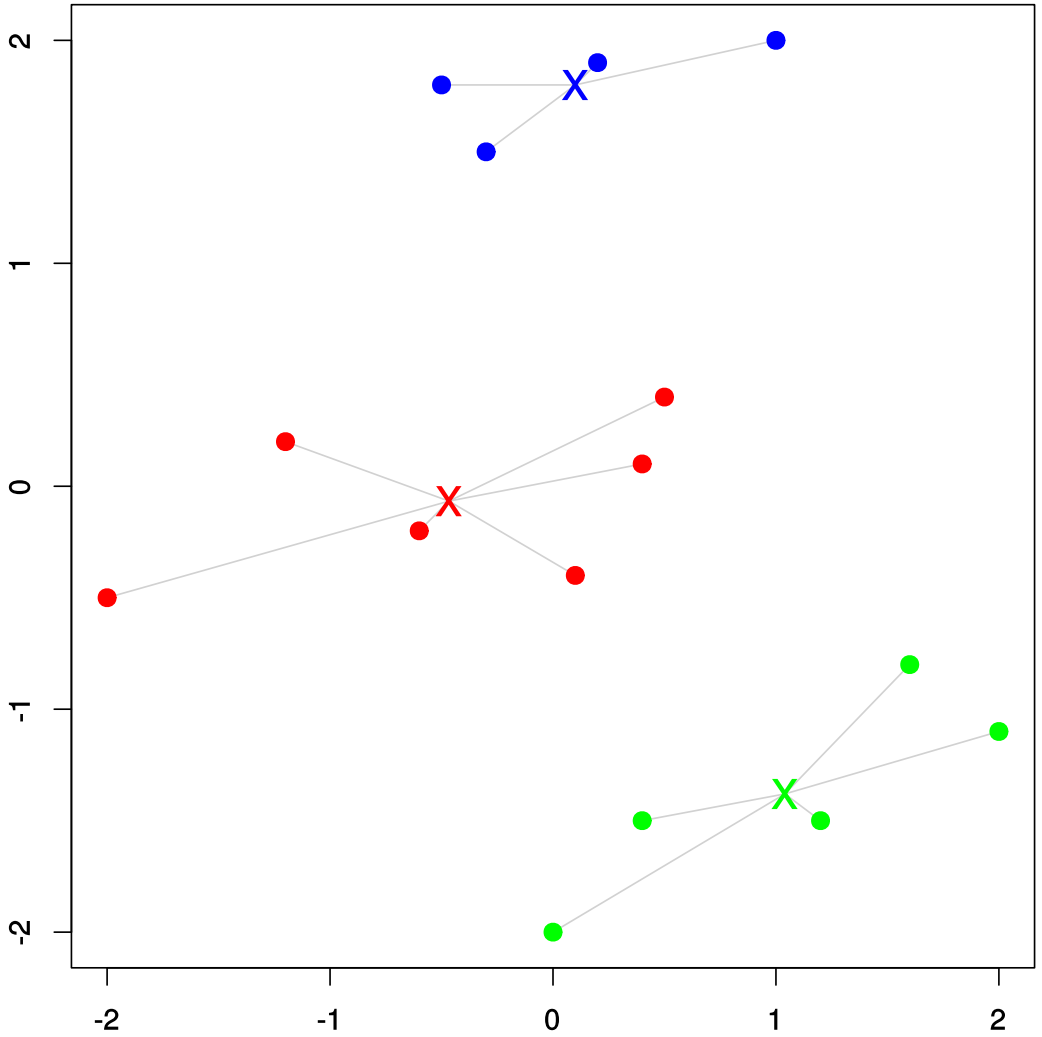


Sum of squared distances: 13.71

Prepočítame μ_i

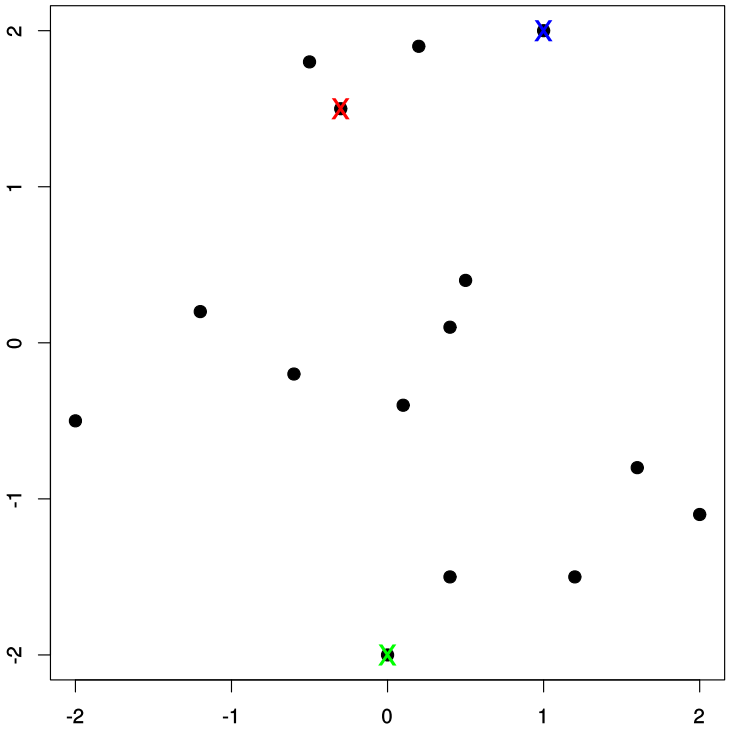


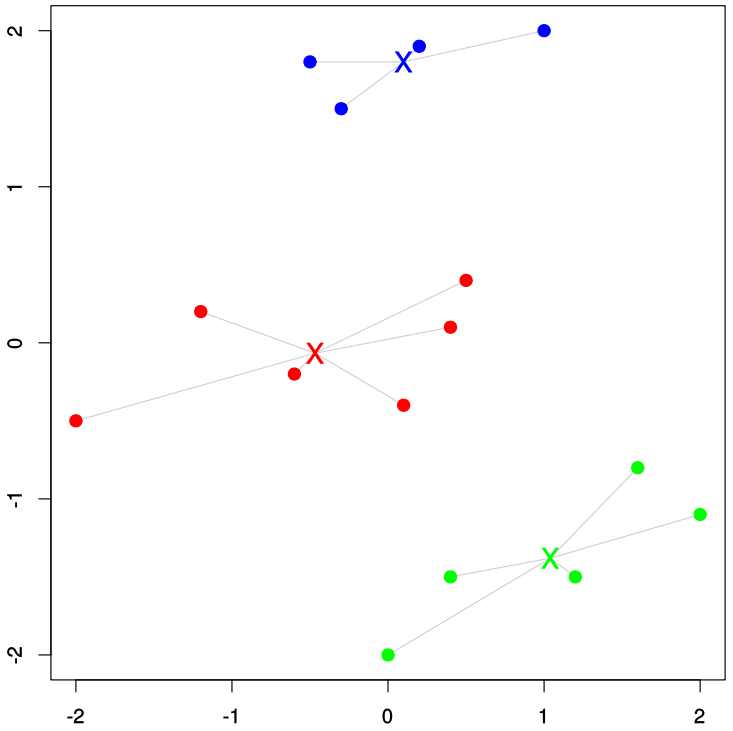
Prepočítame c_i (žiadna zmena, končíme)



Sum of squared distances: 10.61

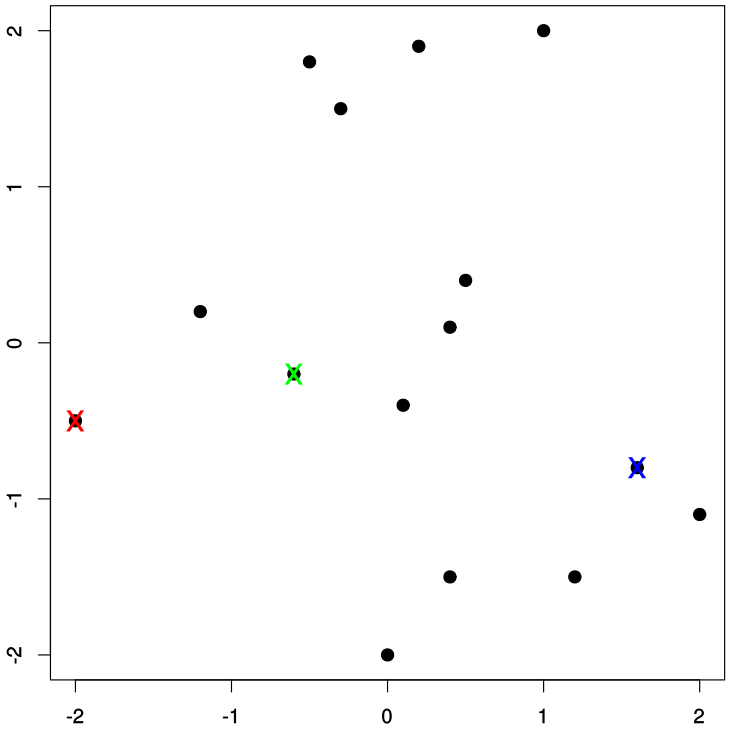
Príklady niekoľkých behov programu

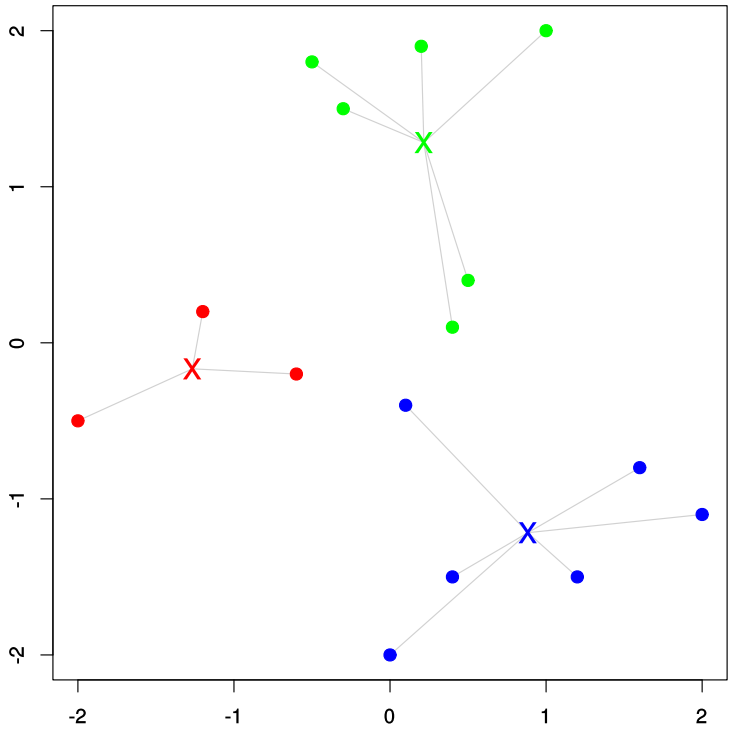




Sum of squared distances: 10.61

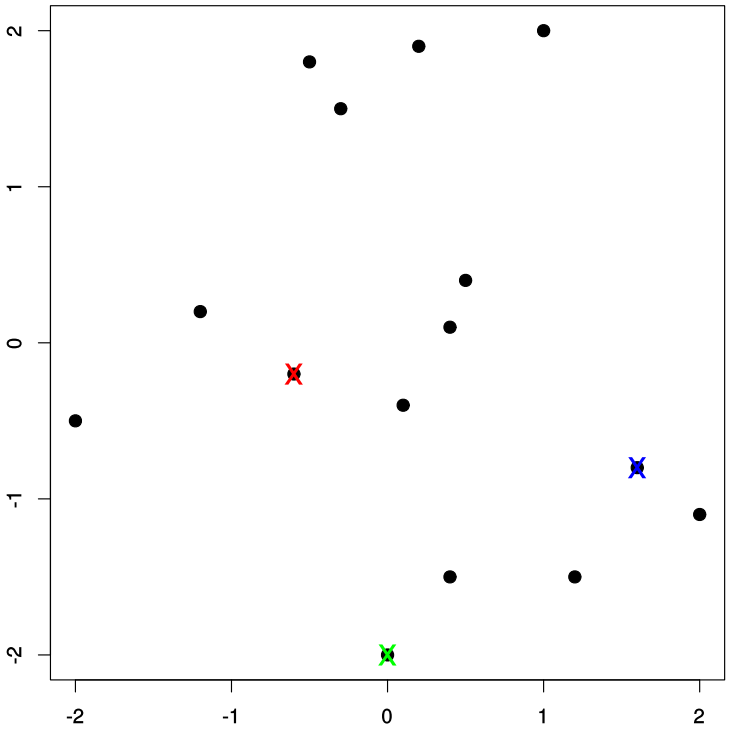
Príklady niekoľkých behov programu

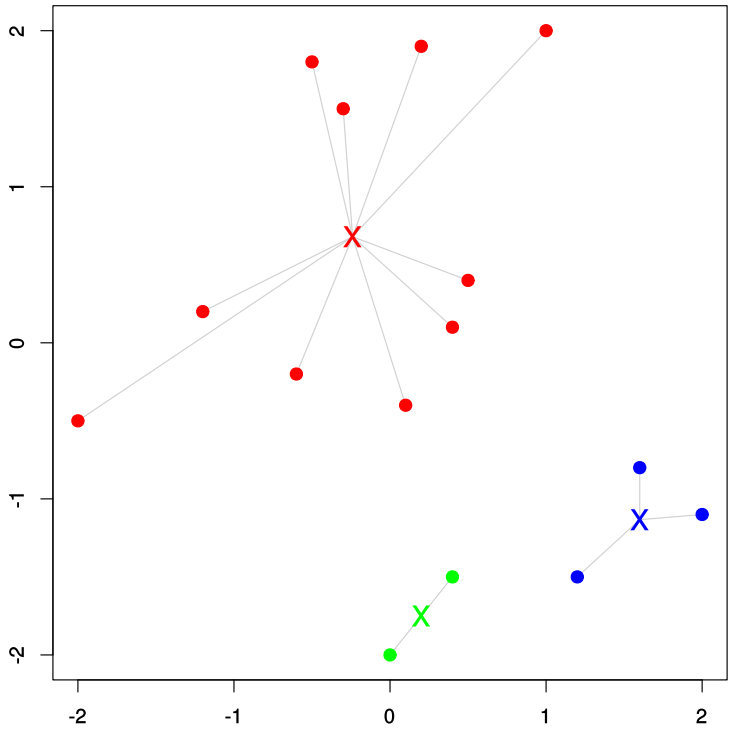




Sum of squared distances: 11.25

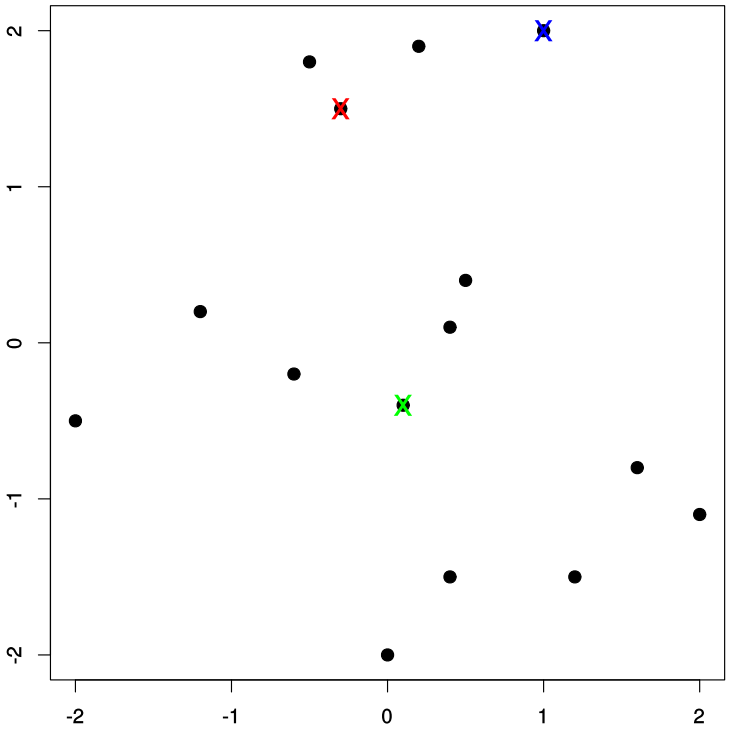
Príklady niekoľkých behov programu

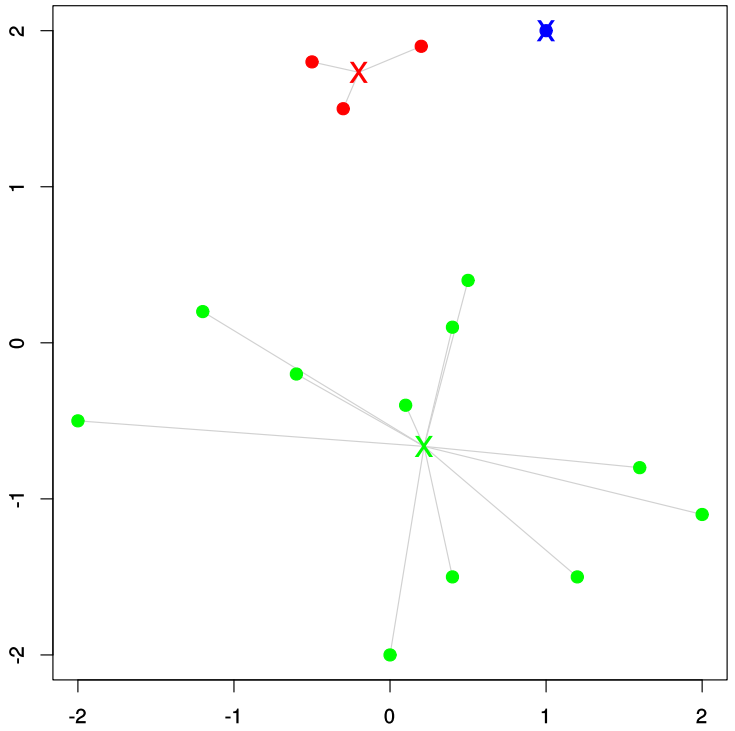




Sum of squared distances: 16.93

Príklady niekoľkých behov programu





Sum of squared distances: 20.37

Cvičenia pre biológov, 12.12.2024

Zhrnutie semestra

Tvorba bioinformatického nástroja

- Sformulujeme biologické ciele
  (aké máme dáta, aké typy otázok sa chceme pýtať).
- Sformulujeme informaticky/matematicky
  (napr. ako pravdepodobnostný model).
  Dostaneme informatické zadanie problému, v ktorom je presne daný vzťah medzi vstupom a želaným výstupom
  (napr. nájsť zarovnanie s max. skóre v určitej skórovacej schéme).
- Hľadáme efektívne algoritmy na riešenie informatického problému.
- Ak sa nám nepodarí nájsť dosť rýchly algoritmus, použijeme heuristiky, ktoré dávaju približné riešenia.
- Testujeme na reálnych dátach, či sú výsledky biologicky správne
  (či bol model dobre zvolený, či heuristiky dobre fungujú).

Použitie bioinformatického nástroja

- Sformulujeme biologické ciele
  (aké máme dáta, aké typy otázok sa chceme pýtať).
- Porozmýšľame, aký typ nástroja, resp. ich kombinácia by nám mohli pomôcť
- Alebo hľadáme v literatúre nástroj na typ problému, s ktorým sme sa ešte nestretli
- Pre správne nastavenie parametrov a interpretovanie výsledkov je dôležité poznať model, predpoklady, ktoré autori nástroja použili, resp. zdroj dát v príslušnej databáze
- Konkrétne nástroje a webstránky sa rýchlo menia, celkové princípy sa menia pomalšie

Prehľad preberaných tém

- Zostavovanie genómov (najkratšie spoločné nadslovo, heuristiky, de Bruijnov graf)
- Zarovnania (skórovanie ako pravdepodobnostný model, dynamické programovanie, heuristické zarovnávanie, E-value a P-value, lokálne vs. globálne, párové vs. viacnásobné, celogenómové)
- Evolúcia (pravdepodobnostné modely substitúcií, metóda maximálnej vierohodnosti, metóda maximálnej úspornosti, metóda spájania susedov)
- Hľadanie génov (skryté Markovove modely)
- Komparatívna genomika (hľadanie konzervovaných oblastí, komparatívne hľadanie génov, pozitívny výber, fylogenetické HMM, kodónové matice)

Prehľad preberaných tém (pokračovanie)

- Expresia génov (zhlukovanie, klasifikácia, regulačné siete, transkripčné faktory, hľadanie motívov)
- Proteíny (predikcia štruktúry, profily a profilové HMM rodín/domén)
- RNA štruktúra (dynamické programovanie, stochastické bezkontextové gramatiky)
- Populačná genetika (mapovanie asociácií, väzbová nerovnováha, genetický drift, štruktúra a história populácie)

Nahliadli sme do sveta informatiky

- Algoritmus, časová zložitosť
- NP-ťažké problémy, presné algoritmy, heuristiky, aproximačné algoritmy
- Dynamické programovanie
- Stromy, grafy
- Skryté Markovove modely a bezkontextové gramatiky

Ďalšie predmety

- **Genomika** N-mCBI-303, Nosek a kol. (LS, 2P, 3kr)
- **Linux pre používateľov** 1-AIN-500, Uhliarik (LS, 2K, 2kr) alebo
  Operačné systémy a počítačové siete 1-DAV-103 (ZS, 2P+2C, 5k)
- **Programovanie (1)** 1-MAT-130, Salanci (ZS, 2P+2C, 5kr) alebo
  Programovanie (1) 1-AIN-130 Blaho (ZS, 4P+4C, 9kr)

Pre pokročilejších

- **Seminár z bioinformatiky 1, 2** Brejová, Vinař (ZS/LS, 2S, 2kr)
  journal club o bioinformatických metódach
- **Úvod do bioštatistiky** 1-BMF-331 Waczulíková (LS, 2P+1C, 4kr)
  predmet pre biomedicínskych fyzikov a dátovú vedu
- **Vizualizácia dát** 1-DAV-105 Brejová, Bátorová (LS, 2P+2C, 5kr)
  vyžaduje základy Pythonu
- **Manažment dát** 1-DAV-105 Brejová, Boža, Vinař (LS, 1P+2C, 5kr)
  vyžaduje znalosť programovania, základy práce na príkazovom riadku

Teória grafov

Broňa Brejová

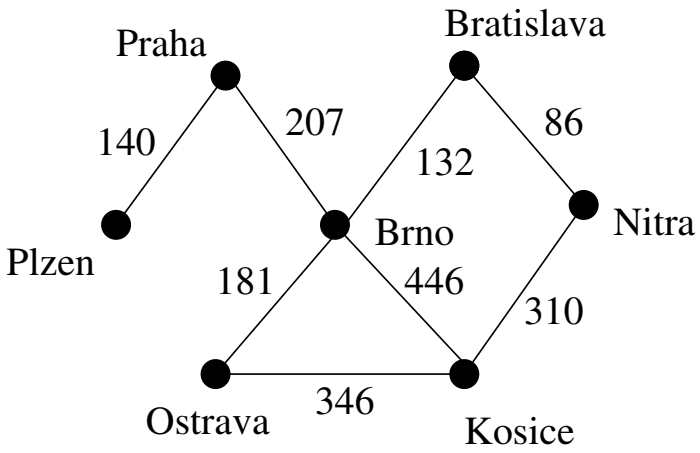
19.12.2020

Grafy a grafové algoritmy

Graf: 7 vrcholov (mestá), 8 hrán (cestné spojenia)

Počet vrcholov n, počet hrán m

Nezáleží na rozmiestnení vrcholov



Cesta: Postupnosť nadväzujúcich hrán, žiadny vrchol sa neopakuje

Napr. Plzeň–Praha–Brno–Bratislava je cesta

Brno–Ostrava–Košice–Brno–Praha nie je cesta

Najkratšia cesta z a do b: Cesta spájajúca vrcholy a a b s najmenším súčtom vzdialeností na hranách

Možno spočítať v čase $O(n^2)$ **Dijkstrovym algoritmom.**

Cyklus: Postupnosť nadväzujúcich hrán, ktorá sa vracia do východzieho bodu, nemá žiadne iné opakujúce sa vrcholy.

Proctor and Gamble súťaž, 1962

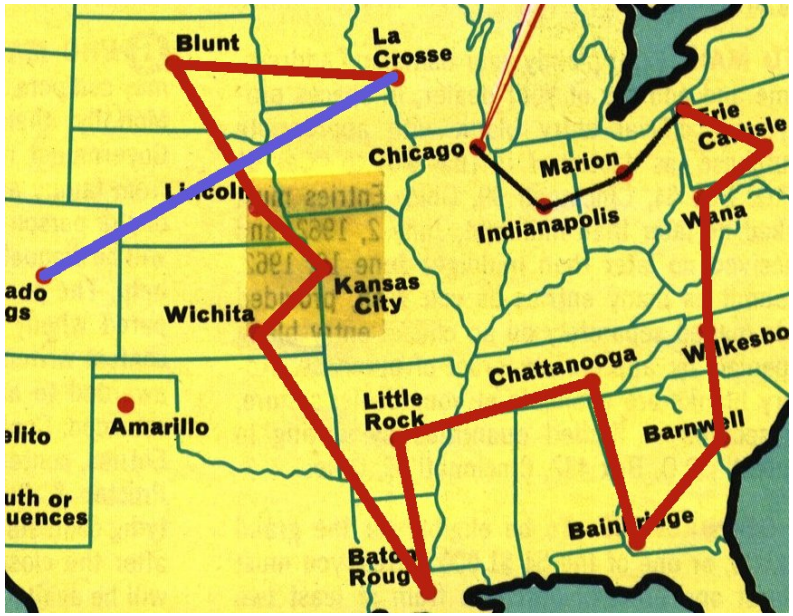
Problém obchodného cestujúceho

Vrcholy: mestá na mape

Hrany: medzi každými dvoma vrcholmi, váha je vzdušná vzdialenosť

Úloha: obcestovať všetky mestá tak, aby celková vzdušná vzdialenosť bola minimálna (**Hamiltonovská kružnica**)

Jednoduchá heuristika: Vždy pokračuj v najbližšom meste, ktoré sme ešte nenavštívili.



Správny a efektívny algoritmus? Nanešťastie, obchodný cestujúci je **NP-ťažký problém**.

Príklad: Sieť interakcií proteínov

Vrcholy: proteíny

Hrany: priame interakcie

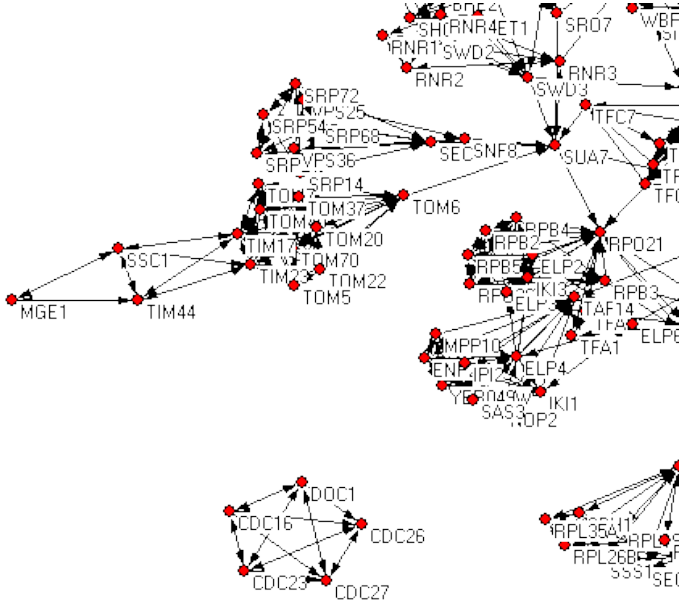
Metabolické dráhy zodp. **cestám**

Metabolické cykly zodp. **cyklom**

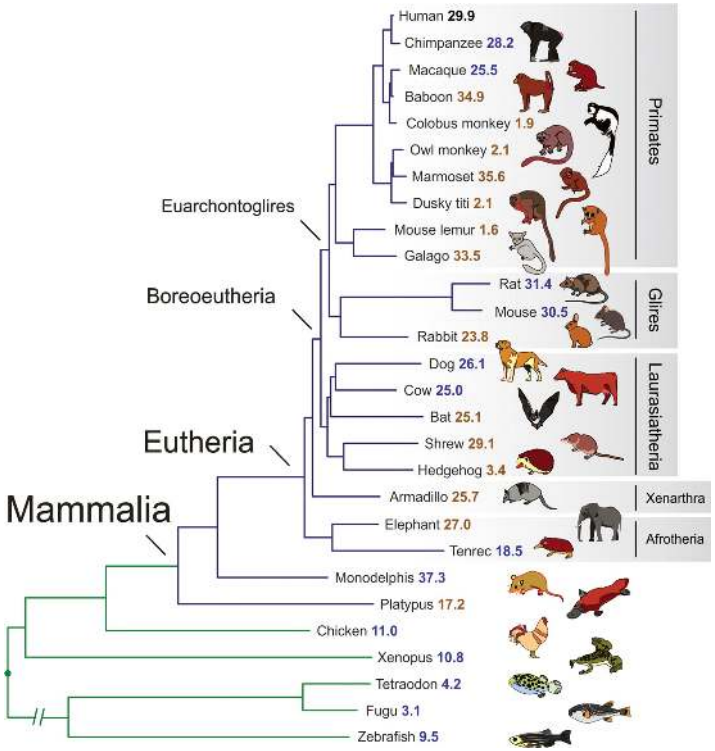
Kliky: Skupiny vrcholov priamo prepojené každý s každým

Komplexy zodpovedajú **klikám**

Komponenty súvislosti: Najväčšie skupiny vrcholov tak, aby sa v každom komponente dalo dostať z každého vrcholu do každého.



Príklad: Fylogenetický strom



- **Stromy** sú špeciálna podtrieda grafov (acyklické, súvislé)
- Vrcholy: listy, vnútorné (spolu n)
- Hrany: $n - 1$
- **Binárny strom:** každý vnútorný vrchol má 2 synov

Ďalšie príklady stromov: hierarchické zhlukovanie, dátové štruktúry na rýchle vyhľadávanie

Ďalšie príklady grafov: de Bruijnov graf, fylogenetická sieť (evolúcia s horizontálnym prenosom génov alebo rekombináciou), regulačné siete, hierarchia GO (gene ontology)