STU FIIT
SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Dátové štruktúry a algoritmy

Algoritmy s reťazcami

12. 12. 2017

zimný semester 2017/2018

Rozbehový problém

- **Daný je reťazec nájsť najdlhší palindróm…**
- Palindróm je slovná hračka: postupnosť písmen, ktorá znie rovnako pri čítaní spredu ako pri čítaní zozadu.
- Používal už 3. st. pred n.l. básnik Sotades z Maronei
- Nájdeme ich vo všetkých jazykoch

Chrám Svätej Múdrosti
Konštantínopol

- Napr. veľmi známy grécky
  ΝΙΨΟΝ ΑΝΟΜΗΜΑΤΑ ΜΗ ΜΟΝΑΝ ΟΨΙΝ
  Nipson anomemata me monan opsin
  (Zmy aj hriechy, nielen tvár)

Sv. Gregor Naziánsky

Žartovnejšie palindrómy

- Náhrobok: **A man, a plan, a canal, Panama.**
- Aké auto si kúpim?
  A Toyota! Race fast...safe car: a Toyota.
- Morálna dilema: **Borrow or rob?**
- Niečo si vyjasnime: **Madam, in Eden, I'm Adam.**
- Bratia Česi sa nezaprú: **Jelenovi pivo nelej** ...
- Zo slovenskej kuchyne: **Má diery syr eidam?**
  Ráno kuchár hodí do hrachu konár ...
- Jankovia: **Jano**, **konaj**! **Nájde med Ján?**
- Iné: **more za jazerom, seno dones, Edo vo vode,**
  Kobyla má malý bok, Nitra Martin ...

Najdlhší palindróm

- Majme slovo W, otočené slovo označíme W^R
  W=AABBACCD W^R=DCCABBAA
- Palindróm je také slovo, pre ktoré W = W^R
  AABBACCDDCCABBAA
- Ako v texte T (dĺžky N písmen) nájdem najdlhší palindróm? AABACADDACABDABAC
- Prvý algoritmus:
  - Preskúmam všetky súvislé postupnosti písmen v T a pre každú skontrolujem, či je palindróm.
  - Koľko je všetkých súvislých postupností písmen v T? $\binom{N}{2}$
  - Postupnosť písmen dĺžky K skontrolujem v čase O(K)
  - Celková zložitosť: $O(N^3)$

Najdlhší palindróm (2)

- Palindróm je také slovo, pre ktoré $W = W^R$
  AABBACCDDCCABBAA
- Všimnem si, že palindróm má stred, okolo ktorého sú písmená symetricky:
  AABBACCD|DCCABBAA
- Stred môže byť:
  - jedno písmeno: A**B**A, alebo
  - medzi susednými písmenami: AB|BA
- Transformujem text na vstupe, tak, aby:
  - bol stred vždy jedno písmeno, a
  - súčasne sa nezhoršila výpočtová zložitosť

Najdlhší palindróm (3)

- Medzi každé susedné písmenká pridám pomocný znak, ktorý sa nenachádza inde v texte, napr. +
- Stred palindrómov pôvodného textu potom bude vždy v nejakom písmenku transformovaného textu
- Párna dĺžka: A+B**+**B+A
  Nepárna dĺžka: A+**B**+A
- Druhý algoritmus:
  - Pre každý možný stred S, určím najdlhší palindróm so stredom S (postupne kontrolujem písmená od stredu S k okraju textu)
  - Počet možných stredov palindrómu? O(N)
  - Určenie dĺžky palindrómu okolo jedného stredu? O(N)
  - Celková zložitosť : $O(N^2)$

Najdlhší palindróm (4)

- Je možné to ešte zrýchliť ?
- Všimneme si, že každý palindróm v sebe obsahuje vnorené menšie palindrómy: AAB**BACCDDCCAB**BAA
- Dajme si najvyššiu métu:
  Vytvoríme O(N) algoritmus
- Čo to znamená?
- Nejaké základné predstavy ako prvý návrh:
  - Musíme prejsť písmená každé práve raz
  - Prechádzať budeme zľava doprava
  - Budeme určovať polomer P[i] palindrómu so stredom v t_i
  - Uvažujme len palindrómy so stredom v nejakom písmene (spravíme transformáciu, aby stredy neboli medzi písmenami)

Najdlhší palindróm – Lineárny algoritmus

- Predpokladajme, že sme už určili P[0], P[1], …, P[i-1], chceme teraz určiť P[i]. Využijeme princíp **zrkadiel**:

palindróm

`T: A A` **`B A C A B`** `D` **`B A C A B`** `D A B`

j

`P: 1 1 2 1 3 1 1 6 1 1` **`3`** `. . . . .`

z=j-(i-j) **i**

- Všimneme si palindróm polomeru P[j] pre stred j < i, ktorý zasahuje najďalej do nasledujúceho textu:
- Sú dve možnosti:
  - **P[j] zasahuje aj do i-teho znaku** – využijeme hodnotu polomeru palindrómu zo stredom v zrkadlovom obraze z podľa znaku j: P[j-(i-j)]
  - **P[j] nezasahuje do i-teho znaku (končí skôr)** – nevyužijeme ho

Najdlhší palindróm – Lineárny algoritmus (2)

- Pri určovaní P[i] začneme od hodnoty polomeru palindrómu v zrkadlenom písmene, ktorá je ešte v najďalej-zasahujúcom palindróme:

palindróm

```
T: A A B A C A B D B A C A B D A B
                =
P: 1 1 2 1 3 1 1 6 1 1 3 . . . . .
```

- Kontrolujeme ďalšie písmená – rozširujeme stred palindrómu
- <u>Ak najďalej-zasahujúci palindróm zasahuje aj do i-teho znaku</u>: tak začneme porovnávať písmená od hodnoty polomeru v zrkadlenom písmene, **ktorá je pokrytá v najďalej-zasahujúcom palindróme**)

Najdlhší palindróm – Lineárny algoritmus (3)

- Pri určovaní P[i] začneme od hodnoty polomeru palindrómu v zrkadlenom písmene, ktorá je ešte v najďalej-zasahujúcom palindróme:

```
T: A A B A C A B D B A C A B D A B
P: 1 1 2 1 3 1 1 6 1 1 4 . . . . .
```

- Kontrolujeme ďalšie písmená
- Rozširujeme stred palindrómu ...

Najdlhší palindróm – Lineárny algoritmus (4)

- Pri určovaní P[i] začneme od hodnoty polomeru palindrómu v zrkadlenom písmene, ktorá je ešte v najďalej-zasahujúcom palindróme:

Starý najďalej-zasahujúci palindróm

```
T: A A B A C A B D B A C A B D A B
```

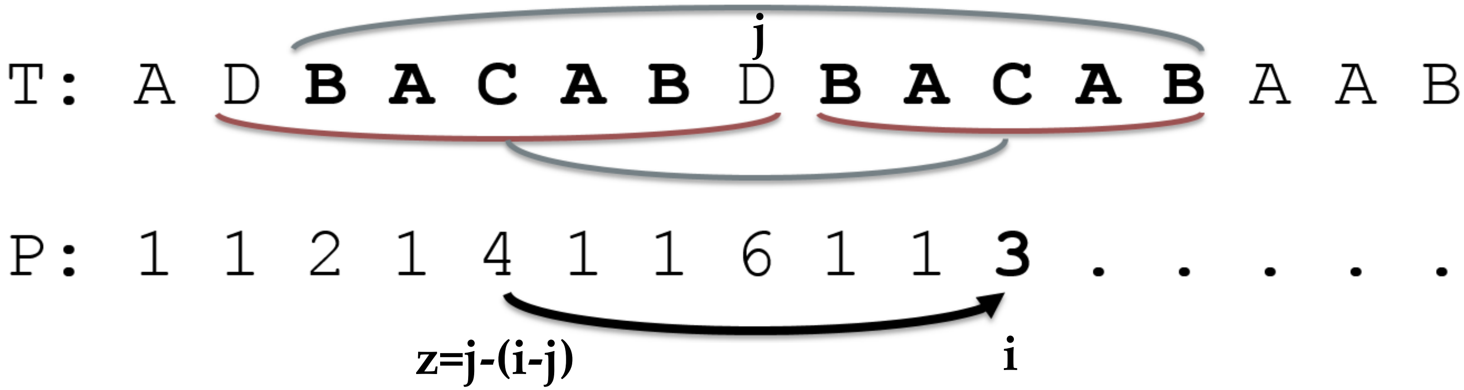
Nový najďalej-zasahujúci palindróm

```
P: 1 1 2 1 3 1 1 6 1 1 4 ? . . . .
```

- Kontrolujeme ďalšie písmená
- Nový palindróm presahuje doteraz najďalej-zasahujúci, a teda ho dokážeme neskôr viac (ako starý) využiť pri určovaní hodnôt P pre ďalšie písmená

Najdlhší palindróm – Lineárny algoritmus (5)

- Palindróm so stredom v zrkadlovom obraze z=j-(i-j) môže smerom doľava presahovať najďalej-zasahujúci palindróm, príklad nižšie:



V takomto prípade, začneme od hodnoty polomeru, **ktorá je ešte pokrytá najďalej-zasahujúcim palindrómom,** teda: od hodnoty, o ktorej máme ešte informácie z predchádzajúcich porovnaní. (V ukážke začneme od hodnoty 3, napriek tomu, že v zrkadlovom obraze je polomer palindrómu 4.)

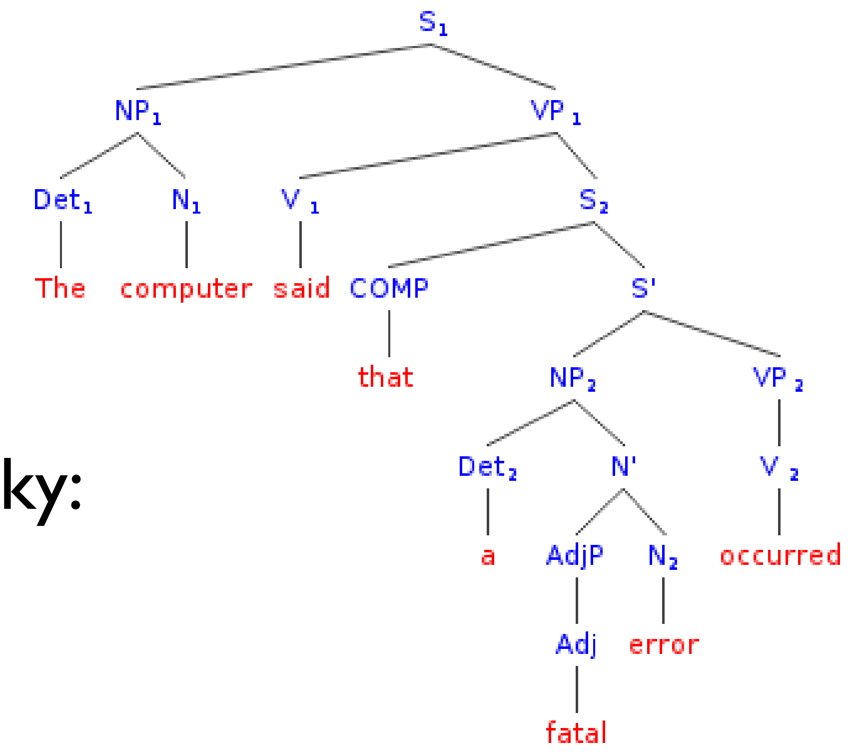
Najdlhší palindróm – Lineárny algoritmus (6)

- Lineárny algoritmus: Pre každý stred t_i určíme polomer palindrómu P[i] so stredom v t_i, využijeme: najďalej-zasahujúci palindróm a princíp zrkadiel:
  - pre ďalšie spracované písmeno t_i vieme zo zrkadlového obrazu zistiť minimálny polomer palindrómu so stredom t_i taký, že je ešte pokrytý najďalej-zasahujúcim palindrómom
  - Od tejto hodnoty začneme porovnávať písmená – rozširovať palindróm okolo t_i od okraja najďalej-zasahujúceho palindrómu
- Prečo má tento algoritmus lineárnu zložitosť O(n)?
- Pretože v každom písmene nastane jedna z možností:
  - alebo urobíme len jedno porovnanie písmen (nerozšírime stred)
  - alebo rozšírime najďalej-zasahujúci palindróm (čo môže nastať celkovo v texte najviac n krát)

Spracovanie prirodzeného textu

- Všeobecný problém – syntaktická analýza (parsing): Daná je gramatika jazyka a slovo (postupnosť písmen) určiť strom odvodenia slova v tejto gramatike

- Napr. slovo:
  The computer said that a fatal error occured.



- Pre bezkontextové gramatiky:
  Cocke-Younger-Kasami (CYK) algoritmus

Cocke-Younger-Kasami (CYK) algoritmus

- Daná je bezkontextová gramatika v Chomského normálnom tvare ($A \rightarrow BC$, $A \rightarrow a$, $S \rightarrow \varepsilon$)
- Slovo: $a_1 a_2 ... a_n$.
- Neterminálne symboly $R_1, ... R_r$
- Pre každdú súvislú podpostupnosť slova určiť, či je možné ju odvodiť z R_k:

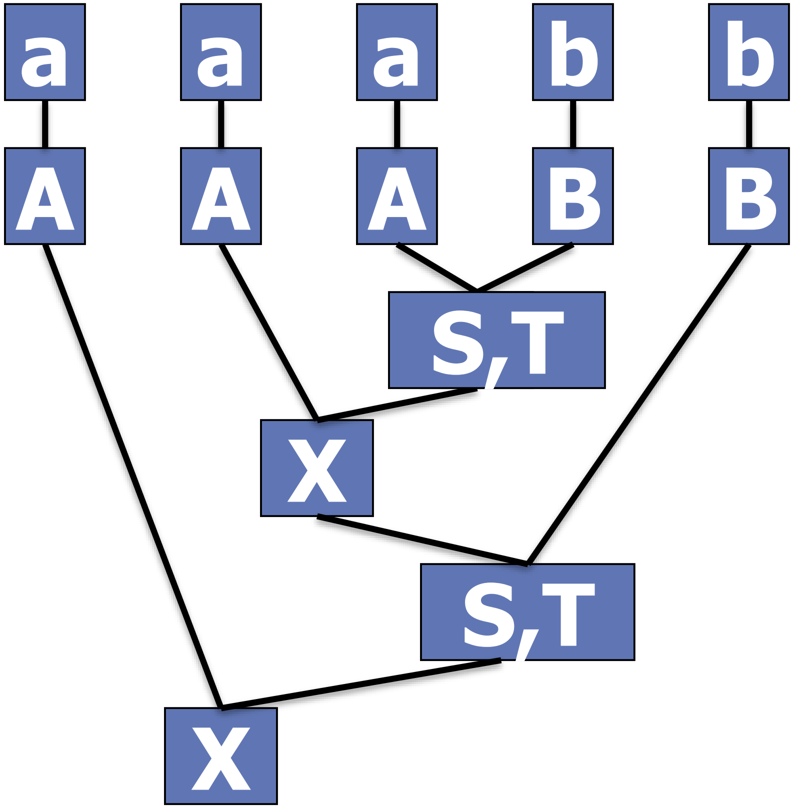
$$
\mathbf{P(i,j,k)} = \begin{cases} \textbf{1, ak slovo } \mathbf{a_i ... a_j} \textbf{ je možné odovodiť z } \mathbf{R_k} \\ \textbf{0, inak.} \end{cases}
$$

Cocke-Younger-Kasami (CYK) algoritmus (2)

- Slovo: $a_1a_2...a_n$. Neterminálne symboly R_1, ... R_r
- Pre každú súvislú podpostupnosť slova určiť, či je možné ju odvodiť z R_k: P(i,j,k).
- Postup: určiť hodnoty P(i,j,k) pre podslová dĺžky postupne 1, 2, ... n.
  - Pre podslová dĺžky 1 je to triviálne.
  - Pre podslová dĺžky 2 a viac skúsiť každé možné rozdelenie na dve časti X a Y: pričom, ak existuje produkčné pravidlo $P \rightarrow QR$, také, že X sa dá odvodiť z Q a zároveň Y sa dá odvodiť z R, tak potom aj celé slovo XY sa dá odvodiť z P.

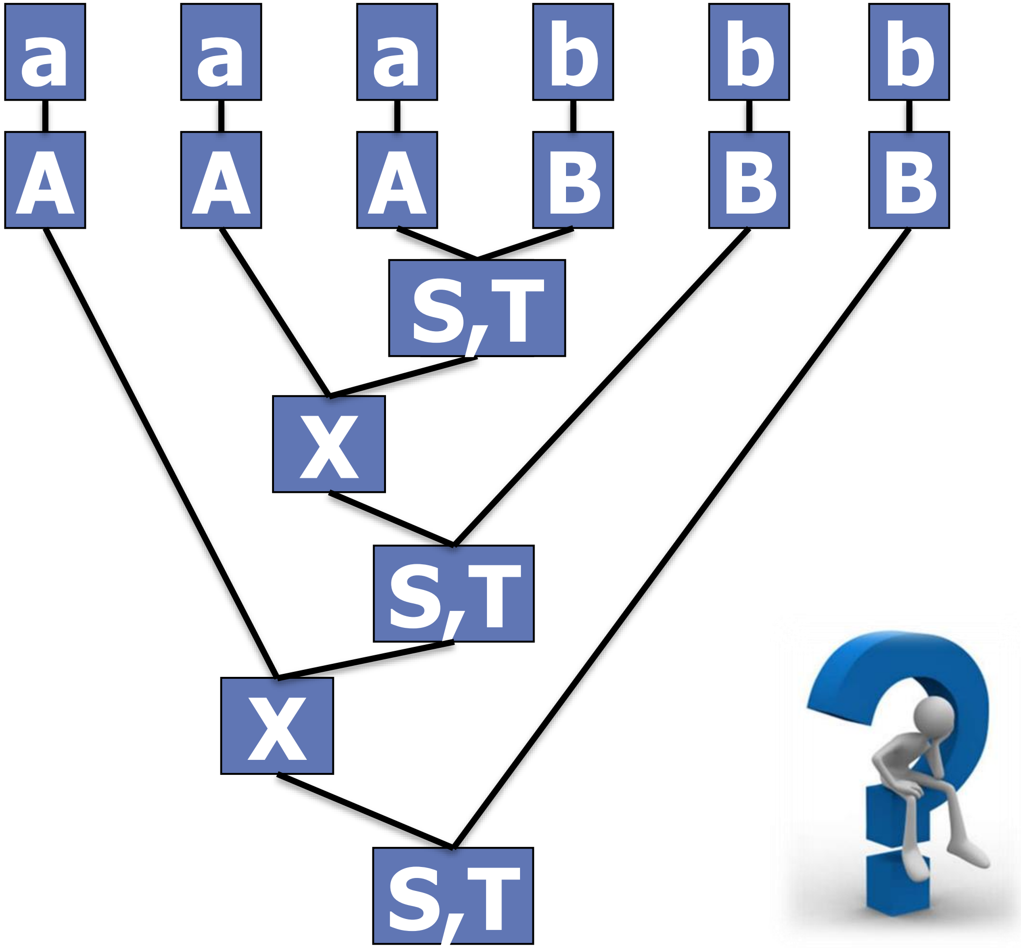
CYK algoritmus – Ukážka

- Napr. gramatika G:
  $S \rightarrow \varepsilon \mid AB \mid XB$
  $T \rightarrow AB \mid XB$
  $X \rightarrow AT$
  $A \rightarrow a$
  $B \rightarrow b$
- Slovo: aaabb
- **Výsledok:**
  aaabb $\notin$ L(G)
  (lebo aaabb NIE JE
  možné odvodiť
  z počiatočného S)



CYK algoritmus – Ukážka (2)

- Napr. gramatika G:
  $S \rightarrow \varepsilon \mid AB \mid XB$
  $T \rightarrow AB \mid XB$
  $X \rightarrow AT$
  $A \rightarrow a$
  $B \rightarrow b$
- Slovo: aaabbb



- **Výsledok: aaabbb ∈ L(G)**
  (lebo aaabbb JE možné odvodiť z počiatočného S)

CYK algoritmus – Reprezentácia v tabuľke

začína v \ končí v	1: aaabbb	2: aaabbb	3: aaabbb	4: aaabbb	5: aaabbb	6: aaabbb
0:aaabbb	*A*	-	-	-	*X*	*S*,*T*
1:aaabbb		*A*	-	*X*	*S*,*T*	-
2:aaabbb			*A*	*S*,*T*	-	-
3:aaabbb				*B*	-	-
4:aaabbb					*B*	-
5:aaabbb						*B*

- Implementácia: Dynamické programovanie $O(N^3)$

Problém vyhľadania vzorky (string matching)

- Základný (hlavný) problém spracovania reťazcov
- Daný je text T a vzorka P, nájdi všetky výskyty P v T
- Označenie:
  - n dĺžka T, m dĺžka P
  - Znaky: abeceda ∑ (konštantnej veľkosti), písmená a, b, c, ... slová (postupnosti písmen) x, y, z
  - i-ty znak vzorky P_i (indexujeme od 1)
- Príklad:

  T = AGCATGCTGCAGTCATGCTTAGGCTA

  P = GCT

  P sa vyskytuje tri krát v T:

  AGCAT<u>GCT</u>GCAGTCAT<u>GCT</u>TAG<u>GCT</u>A

Základné pojmy

- **Slovo (reťazec)** S = AGCTTGA
- |S|=7, **dĺžka slova (reťazca)** S

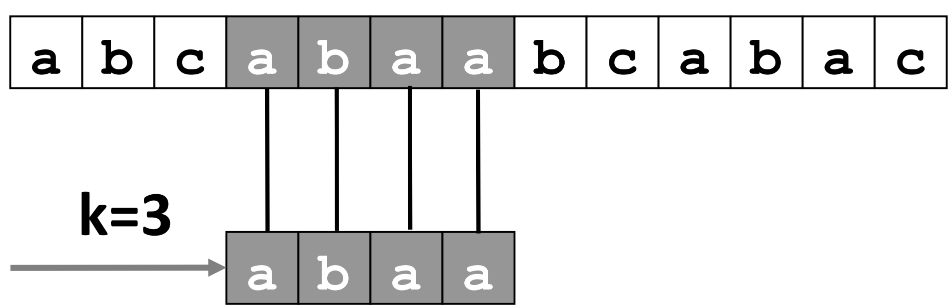
- **Podslovo (súvislý podreťazec)**: $S_{i,j} = S_i S_{i+1} \ldots S_j$
  napr. $S_{2,4}$ = GCT (A<u>GCT</u>TGA)

- **Podpostupnosť reťazca S**: vymazaním niekoľkých (vrátane žiadneho) znakov z S
  napr. ACT (<u>A</u>G<u>CT</u>TGA), GCTT (A<u>GCTT</u>GA)

Základné pojmy – Výskyt vzorky

- Uvažujme teraz reťazce:
  - Text T[1...n], ktorý má dĺžku n
  - Vzorku P[1...m], ktorá má dĺžku m
- Vzorka P sa nachádza v texte T s posunutím *k* ak platí:
  - T[k+1…k+m] = P[1…m]

    napr. T = abcabaabcabac, P = abaa
    m=4, n=13, k=3

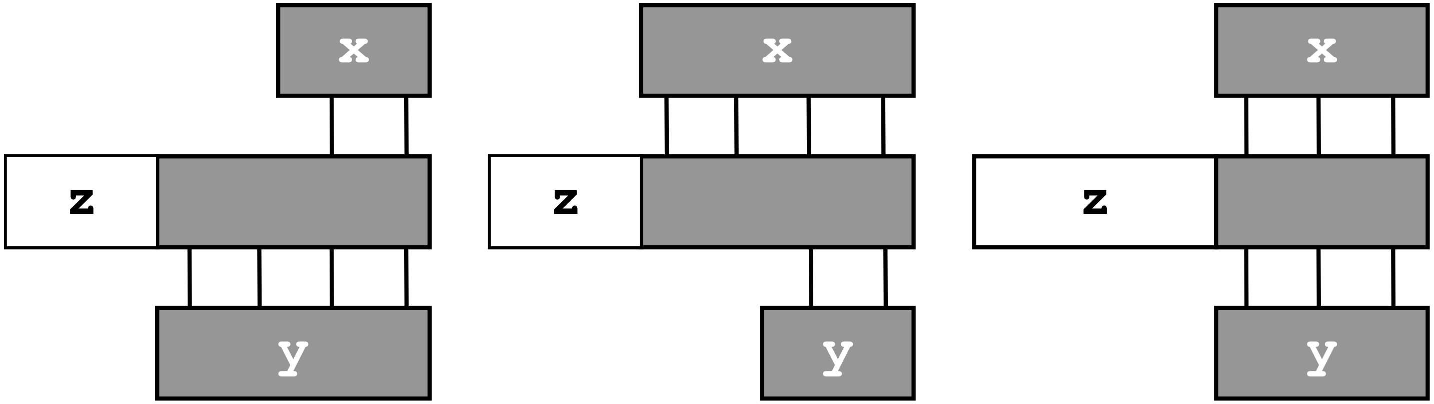


Základné pojmy – Prefix a sufix

- Prefix (predpona) S: podreťazec tvaru $S_{1,k}$
  napr. AGCT (<u>AGCT</u>TGA)
  - Reťazec *w* je predponou reťazca *x*, ak (existuje reťazec y taký, že) *x* = *wy*, kde *y* je akýkoľvek reťazec z použitej abecedy *Σ*, t.j. prvok z množiny *Σ**
    Napr: pre(ab,abcca)
- Sufix (prípona) S: podreťazec tvaru $S_{h,|S|}$
  napr. CTTGA (AG<u>CTTGA</u>)
  - Reťazec *w* je príponou reťazca *x*, ak (existuje reťazec y taký, že) *x* = *yw*, kde *y* je akýkoľvek reťazec z použitej abecedy *Σ*, t.j. prvok z množiny *Σ**
    Napr: suf(cca,abcca)

Dôležitá vlastnosť sufixov (aj prefixov)

- Predpokladajme, že x, y a z sú reťazce, pre ktoré platí suf(x, z) a suf(y, z), potom:
  - ak $|x| \leq |y|$, tak suf(x,y)
  - ak $|x| \geq |y|$, tak suf(y,x)
  - ak $|x| = |y|$, tak $x = y$



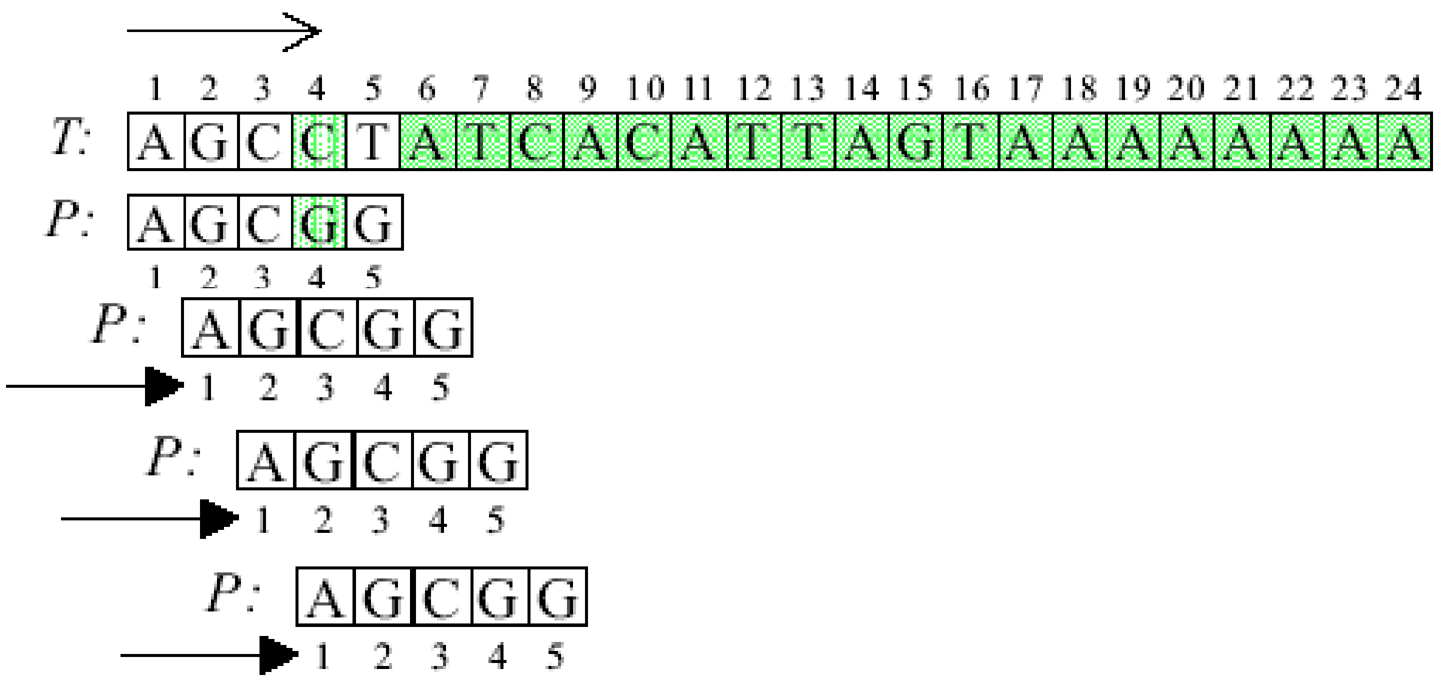
Prvé riešenie

- Formulácia problému:
  Daný je text T a vzorka P, nájdi všetky výskyty P v T:
  T = AGCAT<u>GCT</u>GCAGTCAT<u>GCT</u>TAG<u>GCT</u>A
  P = GCT

- Prvé riešenie (tzv. naivné):
  - Pre každé písmeno textu T vyskúšaj porovnať vzorku P, či v ňom začína.
  - Pre každý začiatok môže porovnanie trvať O(m), napr. pre:
    T = AAAAAAAAAAAAAAAAAAAAAAAAAH
    P = AAAAAH
  - Celková zložitosť: O(nm)

Prvé riešenie (2)

- Naivný algoritmus sa pri nájdení rozdielu posunie vždy o 1 doprava:



- Vedeli by sme sa posunúť aj o viac ako o 1?

Knuth-Morris-Prattov (KMP) algoritmus

- Hlavná myšlienka: **Preskočiť nejaké porovnávania využitím prechádzajúcich výsledkov porovnávaní**
- Využíva predpočítané pole π[1..m] definované :

  π[i] je najväčšie číslo menšie ako i také, že

  prefix $P_1 \ldots P_i$ je sufixom $P_1 \ldots P_i$

i	1	2	3	4	5	6	7	8	9	10
P_i	a	b	a	b	a	b	a	b	c	a
$\pi[i]$	0	0	1	2	3	4	5	6	0	1

π[6]=4 pretože abab je najdlhší (kratší) sufix ababab

π[9]=0 – neexistuje prefix kratší ako 9 znakov končiaci v c

KMP algoritmus – ukážka

- `T = ABC ABCDAB ABCDABCDABDE`
  `P = ABCDABD, π = (0,0,0,0,1,2,0)`
- Začneme porovnávať na prvom znaku T_1:

```
12345678901234567890123
ABC ABCDAB ABCDABCDABDE
ABCDABD
1234567
```

- Nezhoda na 4. znaku vzorky P:
  - Súhlasia k=3 znaky, pričom π[k]=0, a teda nemá zmysel porovnávať od druhého T_2 ani tretieho znaku T_3.
  - Posunieme P o k-π[k]=3 znaky dopredu...

KMP algoritmus – ukážka (2)

- Posunieme P o k-π[k]=3 znaky dopredu...

```
12345678901234567890123
ABC ABCDAB ABCDABCDABDE
   ABCDABD
   1234567
```

- Opäť nezhoda na 4. znaku vzorky P:
  - Súhlasí k=0 znakov, posunieme P o k-π[k]=1 znak dopredu... (definujeme π[0]=-1)

```
12345678901234567890123
ABC ABCDAB ABCDABCDABDE
    ABCDABD
    1234567
```

KMP algoritmus – ukážka (3)

- $\pi[6]=2$ znamená že prefix P_1P_2 je (najdlhším kratším) sufixom $P_1P_2P_3P_4P_5P_6$

```
12345678901234567890123
ABC ABCDAB ABCDABCDABDE
    ABCDABD
          ||
          ABCDABD
          1234567
```

- Nemá zmysel posúvať o 1, 2 alebo 3 znaky, posunieme o $6-\pi[6]=4$ znaky ...

KMP algoritmus – ukážka (4)

- Znovu nezhoda na T_{11}:

```
12345678901234567890123
ABC ABCDAB ABCDABCDABDE
          ABCDABD
          1234567
```

- Teraz súhlasia k=2 znaky, posunieme o: 2-π[2]=2 znakov

```
12345678901234567890123
ABC ABCDAB ABCDABCDABDE
            ABCDABD
            1234567
```

- Posunieme o 0-π[0]=1

KMP algoritmus – ukážka (5)

- Ďalšia nezhoda na T_{18}:

```
12345678901234567890123
ABC ABCDAB ABCDABCDABDE
           ABCDABD
           1234567
```

- Súhlasí k=6 znakov, posunieme o: 6-π[6]=4 znaky

```
12345678901234567890123
ABC ABCDAB ABCDABCDABDE
               ABCDABD
               1234567
```

- Našli sme výskyt, posunieme vzorku o 7-π[7]=7 znakov, a hľadáme ďalší možný výskyt ...

KMP algoritmus – výpočet π

- Pozorovanie č.1 (zjavné): **ak $P_1 \ldots P_{\pi[i]}$ je sufix $P_1 \ldots P_i$, potom $P_1 \ldots P_{\pi[i]-1}$ je sufix $P_1 \ldots P_{i-1}$**
- Pozorovanie č.2: **všetky prefixy P, ktoré sú zároveň aj sufixami $P_1 \ldots P_i$, môžeme získať opakovaným (rekurzívnym) aplikovaním funkcie π na i. $P_1 \ldots P_{\pi[i]}$, $P_1 \ldots P_{\pi[\pi[i]]}$, $P_1 \ldots P_{\pi[\pi[\pi[i]]]}$, … sú sufixy $P_1 \ldots P_i$**
  - Označme $\pi^{(k)}[i]$ k-krát aplikované π na i, napr. $\pi^{(2)}[i] = \pi[\pi[i]]$
  - Potom: $\pi[i] = \pi^{(k)}[i-1] + 1$ pre najmenšie také k, pre ktoré platí: $P_{\pi^{(k)}[i-1]+1} = P_i$
- Intuitívne (určenie hodnoty $\pi[i]$): **Najdlhší taký prefix P, ktorý je sufixom $P_1 \ldots P_{i-1}$ taký, že nasledujúce písmenko v texte súhlasí s P_i**

KMP algoritmus – Implementácia

- Výpočet π:

```
π[0] = -1
k = -1
for (i = 1...m) do
  while (k >= 0 && P[k+1] != P[i]) do
    k = π[k]
  π[i] = ++k
```

- Vyhľadanie výskytov vzorky:

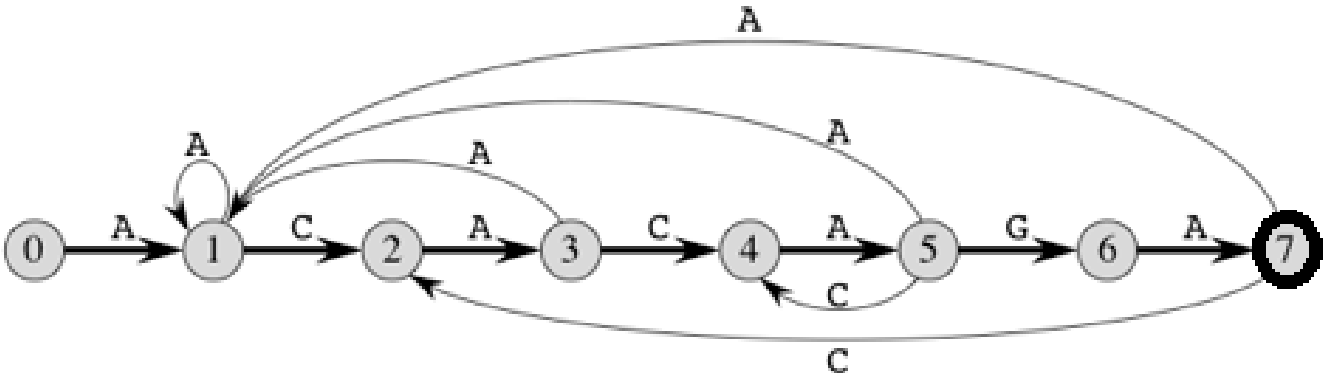
```
k = 0
for (i = 1...n) do
  while (k >= 0 && P[k+1] != T[i]) do
    k = π[k]
  k++
  if (k == m) then
    OUTPUT: P matches T[i-m+1..i]
    k = π[k]
```

KMP algoritmus – Zložitosť

- Vstup: text T[1...n] a vzorka P[1...m]
- Výpočet funkcie π
  - Hlavný for cyklus sa opakuje m krát
  - Vnorený while cyklus sa vracia celkovo najviac toľko krát, koľko krát sme sa „posunuli dopredu“ (k++)
  - Celkovo sme sa posunuli m-krát dopredu
  - Celkovo O(m)
- Vyhľadanie výskytov vzorky P v texte T
  - Hlavný for cyklus sa opakuje n krát
  - Vnorený while cyklus sa vracia celkovo najviac toľko krát, koľko krát sme sa „posunuli dopredu“ (n krát)
  - Celkovo O(n)
- Celková zložitosť O(m+n)

Konečný automat

- Vyhľadávanie výskytu vzorky deterministickým konečno-stavovým automatom



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

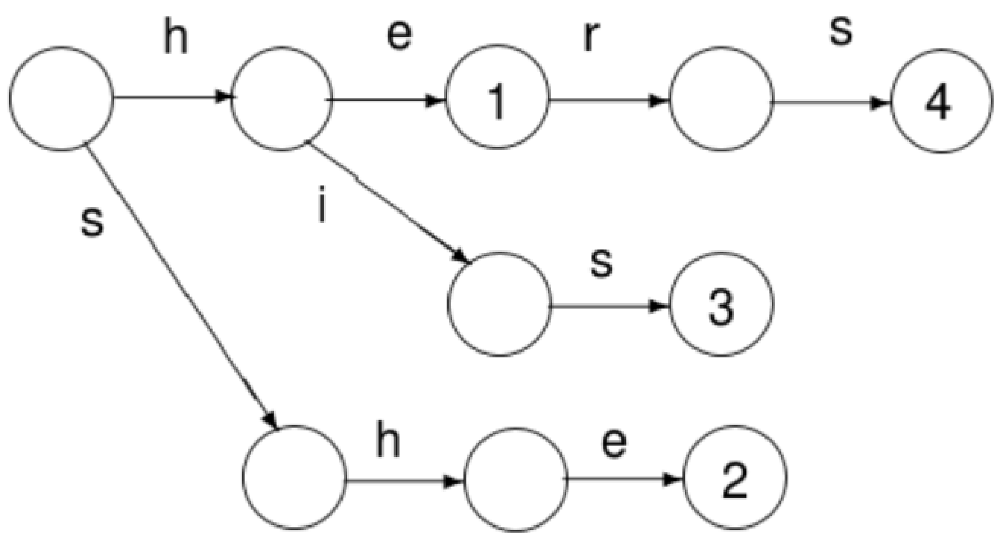
- Princíp:
  - čítame písmená (z textu)
  - stavy automatu zodpovedajú dĺžke najdlhšieho prekryvu vzorky v mieste práve čítaného textu
  - prechody zostrojíme podľa funkcie π

Aho-Corasickovej (AC) algoritmus

- Daný text T a vzorky $P_1, P_2, \ldots, P_k$ nájdi všetky výskyty všetkých vzoriek v texte T, pričom $m=|P_1|+|P_2|+\ldots+|P_k|$
- Ak by sme použili k-krát KMP, máme zložitosť O(nk+m)
- Mohli by sme použiť hashovanie (Rabin-Karpov algoritmus) ale tam je najhoršia zložitosť podobná...
- AC algoritmus beží v zložitosti O(n+m+z), kde z je celkový počet výskytov slov v texte.
- Hlavná myšlienka (AC algoritmu): **Zostrojíme <u>jeden</u> konečný automat pre všetky slová :)**
- Prechod v automate je potom v zložitosti O(n+z)

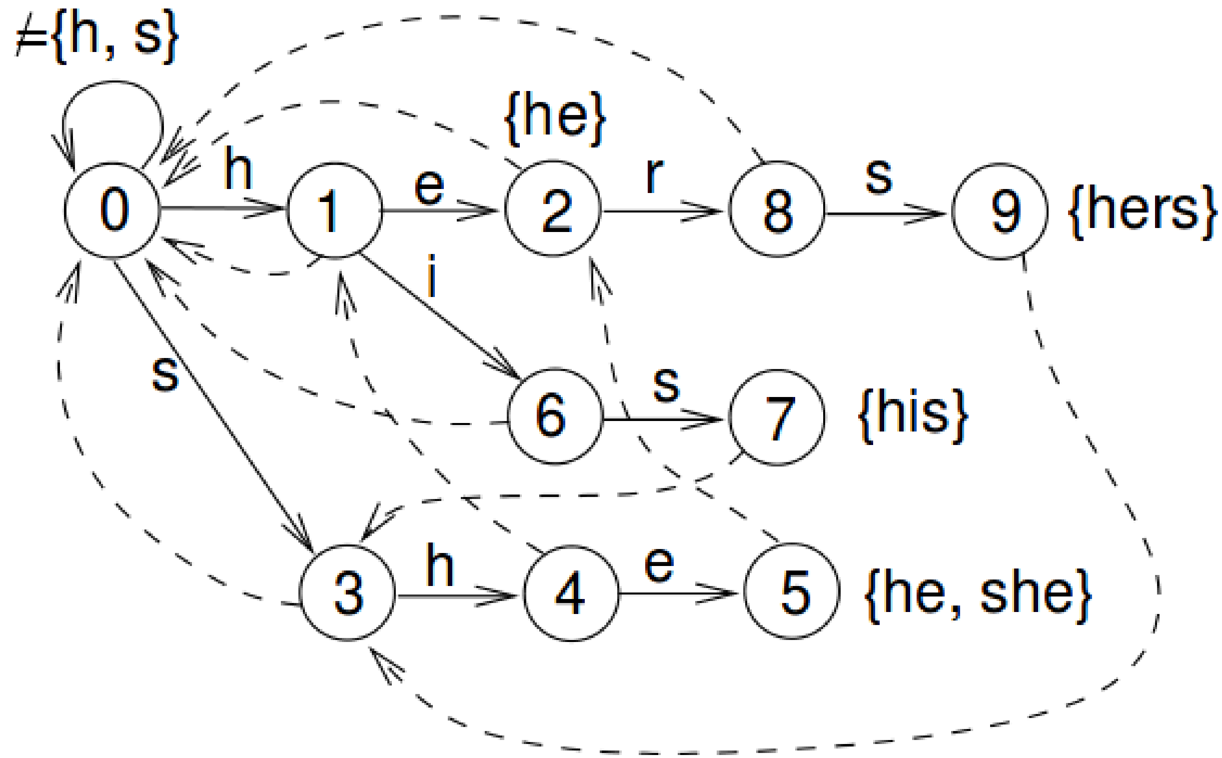
AC algoritmus – Konštrukcia automatu

- Konštrukcia prebehne v dvoch krokoch
- Prvý krok: **zostrojíme písmenkový strom (trie)**
  Napr. pre {he, she, his, hers}



AC algoritmus – Konštrukcia automatu (2)

- Druhý krok: Prehľadávaním do šírky doplníme spätné hrany (prechody automatu), ktoré sa použijú v prípade nezhody písmena: **najdlhší prefix, ktorý je zároveň sufixom**...



Existuje ešte niečo rýchlejšie?

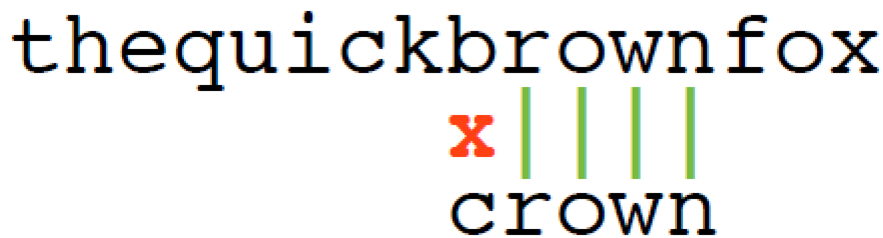
- Text T dĺžky n znakov
- Vzorka (resp. vzorky) dĺžky celkovo m znakov
- Knuth-Morris-Prattov / Aho-Corasickovej algoritmy majú zložitosť O(n+m) – teda, preskúmam každé písmeno „raz“ ...
- Na zamyslenie:
  Je možné, aby existovalo niečo rýchlejšie?
- Áno je to možné!! Ako?

Existuje ešte niečo rýchlejšie? (2)

- Ak to má byť rýchlejšie, musí to preskúmať MENEJ ako každé písmenko raz
- Niektoré písmenká teda nemôže preskúmať
- Huh?
- Je možné, že nepreskúmam celý vstup, a napriek tomu nájdem správne riešenie vzhľadom k celému vstupu?
- Samozrejme ...
- Dokonca: **Šokujúco, čím dlhšia vzorka tým rýchlejšie algoritmus pracuje!!** (zvykne pracovať)

Boyer-Moorov (BM) algoritmus

- Začneme porovnávať vzorku od začiatku textu
- Pre konkrétne posunutie vzroky porovnávaj znaky **SPRAVA DOĽAVA**



- Dve pravidlá pre posun:
  - Pravidlo zlého znaku
  - Pravidlo dobrého sufixu

BM algoritmus – pravidlo zlého znaku

- Uvažujeme znak, na ktorom sa porovnávanie nezhoduje
- Nájdeme nasledujúci výskyt tohto znaku vo vzorke vľavo a vzorka posunieme (dopredu) na tento znak:

```
- - - - X - - K - - -
A N P A N M A N A M -
- N N A A M A N - - -
- - - N N A A M A N -
```

- Ak sa znak nenachádza vľavo vo vzorke, posunieme o CELÚ dĺžku vzorky! (niektoré posuny sme preskočili)

BM algoritmus – pravidlo dobrého sufixu

- Pre nejaký posun, sa zhoduje sufix t, ale nezhoda nastáva pri nasledujúcom znaku vľavo.
- Nájdi najpravejšiu kópiu t’ podreťazca t v P, ktorá nie je sufixom P a znak vľavo od t’ je iný ako znak vľavo od t.
- A) Posuň vzorku dopredu, aby t’ sa zarovnalo s t v T:

```
----X--K------
MANPANAMANAP-
ANAMPNAM-----
----ANAMPNAM-
```

  B) Ak neexistuje, tak posuň tak, aby najdlhší prefix P súhlasil so sufixom t:

```
----X--K-------
MANPANAMANAP--
AMAMPNAM-------
-------AMAMPNAM
```

- C) Ak to nie je možné, posuň o m znakov dopredu.

BM algoritmus – Pseudokód

```
k = 1
while (k < |T| - |P| + 1) do
  porovnaj po znakoch P s T[k..|P|] sprava doľava
  s = max { pravidlo zlého znaku, pravidlo dobrého sufixu, 1 }
  k += s
```

- Najhorší prípad O(nm)
- Veľa rôznych rozšírení a optimalizácií
  - Boyer-Moore-Horspool, Apostolico–Giancarlo, Raita, ...
- Rozšírenia (napr. Galilovo pravidlo) zaručujú najhorší prípad O(n+m)
- **V praxi je to najrýchlejší algoritmus, pretože väčšinu textu preskočí ... O(n/m)**
- Pomalší, keď je malá abeceda.

Bioinformatika – Zostavovanie genómov

- Genóm kravy:
  2,86 miliardy písmen

TATGGAGCCAGGTGCCTGGGGCAACAAGACTGTGG
TCACTGAATTCATCCTTCTTGGTCTAACAGAGAAC
ATAGAACTGCAATCCATCCTTTTTGCCATCTTCCT
CTTTGCCTATGTGATCACAGTCGGGGGCAACTTGA
GTATCCTGGCCGCCATCTTTGTGGAGCCCAAACTC
CACACCCCCATGTACTACTTCCTGGGGAACCTTTC
TCTGCTGGACATTGGGTGCATCACTGTCACCATTC
CTCCCATCTGGCCTGTCTCCTGACCCACCAATGCC
GGGTTCCCTATGCAGCCTGCATCTCACAGCTCTTTTTTTCCACCTCCTGGCTGGAGTGGACTGTCACCTC
CTGACAGCCATGGCCTACGACCGCTACTGGCCATTTGCCAGCCCCTCACCTATAGCATCCGCATGAGCCG
TGACGTCCAGGGAGCCCTGTGGCCGTCTGCTGCTCCATCTCCTTCATCAATGCTCTGACCCACACAGTGG
CTGTGTCTGTCTGGACTTCTGCGGCCCTAACGTGGTCAACCACTTCTACTGTGACCTCCCGCCCCTTTTC
CACTCTCCTGCTCCAGCATCCACCTCAACGGGCAGCTACTTTTCGTGGGGGCCACCTTCATGGGGTGGTC
CCCATGGTCTTCATCTCGGTATCCTATGCCCACGTGGCAGCCGCATCCTGCGGATCCGCTCGGCAGAGGG
CAGGAAGAAAGCCTTCTCCACGTGTGCTCCCACCTCACCGTGGTCTGCATCTTTTATGGAACCGGCTTCT
TCAGCTACTGCGCCTGGGCTCCGTGTCCGCCTCAGACAAGGACAAGGGCATTGGCATCCCAACACTGTCA
TCAGCCCCATGCTGAACCCACTCATCTACAGCCTCCGGAACCCTGATGTGCAGGGCGCCCTGAAGAGGTT
GCTGACAGGGAAGCGGCCCCCGGAGTG ...

Bioinformatika – Zostavovanie genómov (2)

- Naraz vieme prečítať len krátku sekvenciu (cca 1000) písmen z náhodného miesta
- Analógia pre knihu Alica v krajine zázrakov:

```
good-natured, she thought: still
when it saw Alice. It looked
ought to be treated
good-natured, she thought, still
Cat only
a greet many
It looked good-
The Cat only grinned when it saw Alice.
be treated with respect.
still it had very long claws
claws and a great many teeth, so she
so she felt that it ought
```

Bioinformatika – Zostavovanie genómov (3)

- Potrebujeme algoritmy, ktoré to dajú dokopy:

```
The Cat only grinned when it saw Alice.
    Cat only         when it saw Alice. It looked
                                        It looked good-
```

```
good-natured, she thought: still
good-natured, she thought, still
                           still it had very long claws
```

```
claws and a great many teeth, so she
          a greet many        so she felt that it ought
```

```
ought to be treated
         be treated with respect.
```

Formulácia v zmysle spracovania reťazcov

- **Najkratšie spoločné nadslovo**
(shortest common superstring):
Daných je niekoľko reťazcov (čítaní), nájdite najkratší reťazec, ktorý obsahuje všetky vstupné reťazce ako súvislé podreťazce.
- Napr.
Vstup: GCCAAC, CCTGCC, ACCTTC
Výstup: CCTGCCAACCTTC

- V skutočnosti je to výrazne komplikovanejšie:
  - Chyby pri čítaní, neznáma orientácia čítaní, kontaminácia, ...

Bioinformatika – Objavovanie génov

- Objavovanie génov* v dlhých sekvenciách
  - Hľadanie sekvencií, ktoré sú podobné iným známym génom
  - Hľadanie sekvencií, ktoré „vyzerajú ako gény“

- Využitie algoritmov na spracovanie reťazcov
  - Potrebné vyhľadávať reťazce veľmi rýchlo, pričom sa môžu vyskytovať chyby v dátach (približné vyhľadávanie)
  - Vyhľadávať často sa opakujúce sekvencie znakov
  - Spracovanie veľmi dlhých postupností znakov

* Gén je úsek DNA (alebo RNA), ktorý kóduje informáciu pre tvorbu nejakého produktu (zvyčajne bielkovín). Gén je základná funkčná jednotka dedičnosti.

Porovnanie reťazcov

- Dané sú dva reťazce: $a=a_1a_2...a_m$ $b=b_1b_2...b_n$
- Abeceda napr. {A,C,G,T} (bioinformatika)
- Vypočítaj nakoľko sú podobné.

- **Editačná vzdialenosť** (Levenshteinova vzdialenosť) najmenší počet operácií potrebných na transformovanie $a \rightarrow b$, uvažujeme nasledovné operácie:
  - Zmena znaku (mutácia)
  - Odstránenie znaku
  - Pridanie znaku

$$\text{riddle} \longrightarrow \text{ridle} \longrightarrow \text{riple} \longrightarrow \text{triple}$$

Zarovnávanie sekvencií – ukážka

```
prin-ciple
|||| |||xx
prinncipal
(1 gap, 2 mm)
```

```
prin-cip-le
|||| ||| |
prinncipal-
(3 gaps, 0 mm)
```

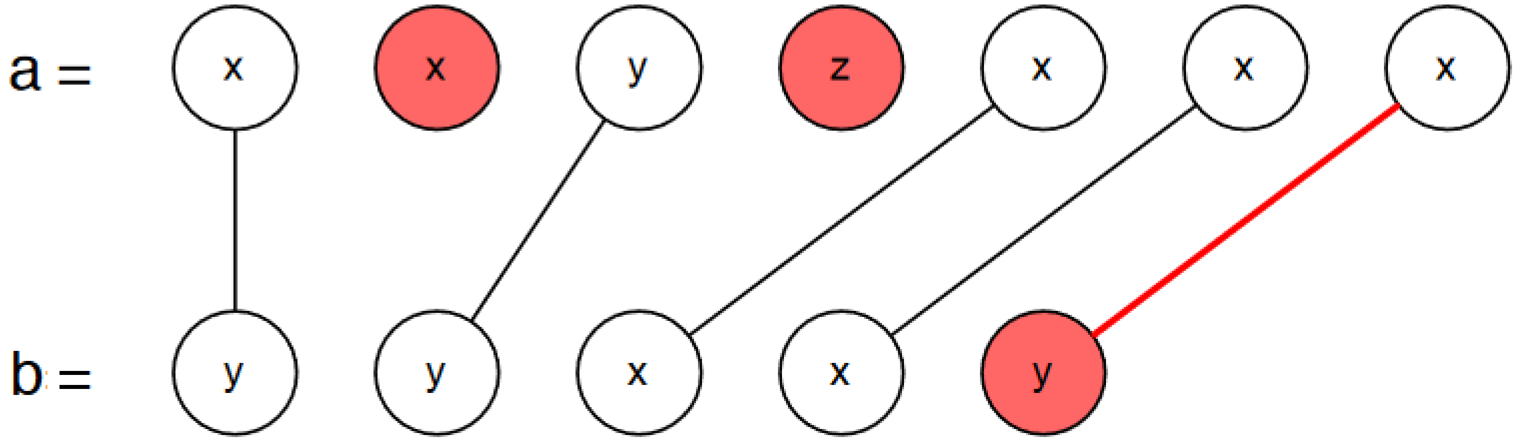
```
misspell
||| ||||
mis-pell
(1 gap)
```

```
prehistoric
   ||||||||
---historic
(3 gaps)
```

```
aa-bb-ccaabb
|x || | | |
ababbbc-a-b-
(5 gaps, 1 mm)
```

```
al-go-rithm-
|| xx ||x |
alKhwariz-mi
(4 gaps, 3 mm)
```

Zarovnávanie sekvencií ako párovanie



Cena párovania:

$$\text{gap} \times \#unmatched + \sum_{(a_i, b_j)} cost(a_i, b_j)$$

Zarovnávanie sekvencií – algoritmus

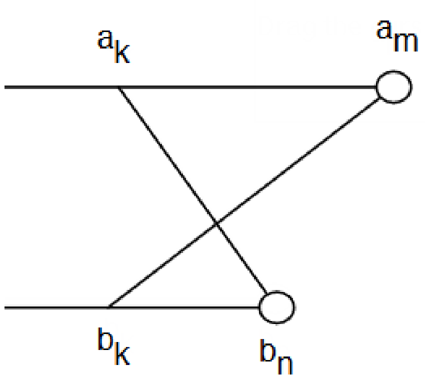
- Uvažuje posledný znak z každého reťazca:

  $a_1 a_2 \ldots a_{m-1} \mathbf{a_m}$

  $b_1 b_2 \ldots b_{n-1} \mathbf{b_n}$

- Nastane niektorá z nasledovných možností:
  1. Znaky sú spárované (cena hrany závisí od toho, či sú rovnaké)
  2. Znak a_m nie je spárovaný
  3. Znak b_n nie je spárovaný

  - Iná možnosť (4.) **nemôže nastať**, lebo by sa hrany párovania (priradené znaky) krížili.
    4. Znak a_m je spárovaný s nejakým b_j ($j \neq n$) a znak bn je spárovaný s nejakým a_k ($k \neq m$).



Zarovnávanie sekvencií – algoritmus (2)

- Rekurzívna definícia riešenia:

$$OPT(i,j) = \min \begin{cases} \text{cost}(a_i, b_j) + OPT(i-1, j-1) & a_i = b_j \\ \text{gap} + OPT(i-1, j) & a_i \text{ nespárovaný} \\ \text{gap} + OPT(i, j-1) & b_j \text{ nespárovaný} \end{cases}$$

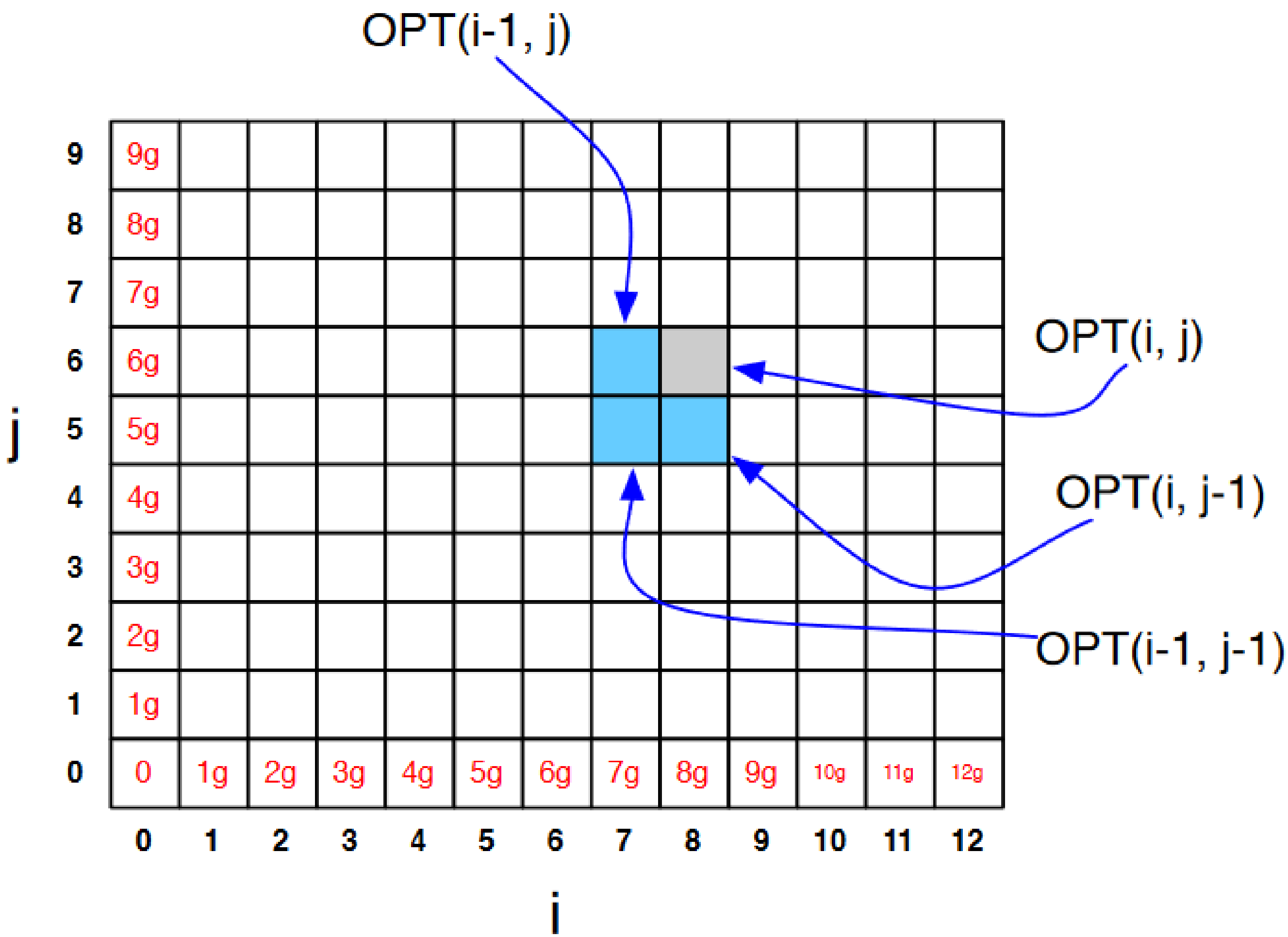
Optimálne zarovnanie $a_1 \ldots a_i$ a $b_1 \ldots b_j$

vyjadrené ako menšie problémy

- Pri výpočte vždy vyberieme ten lepší (min) ...

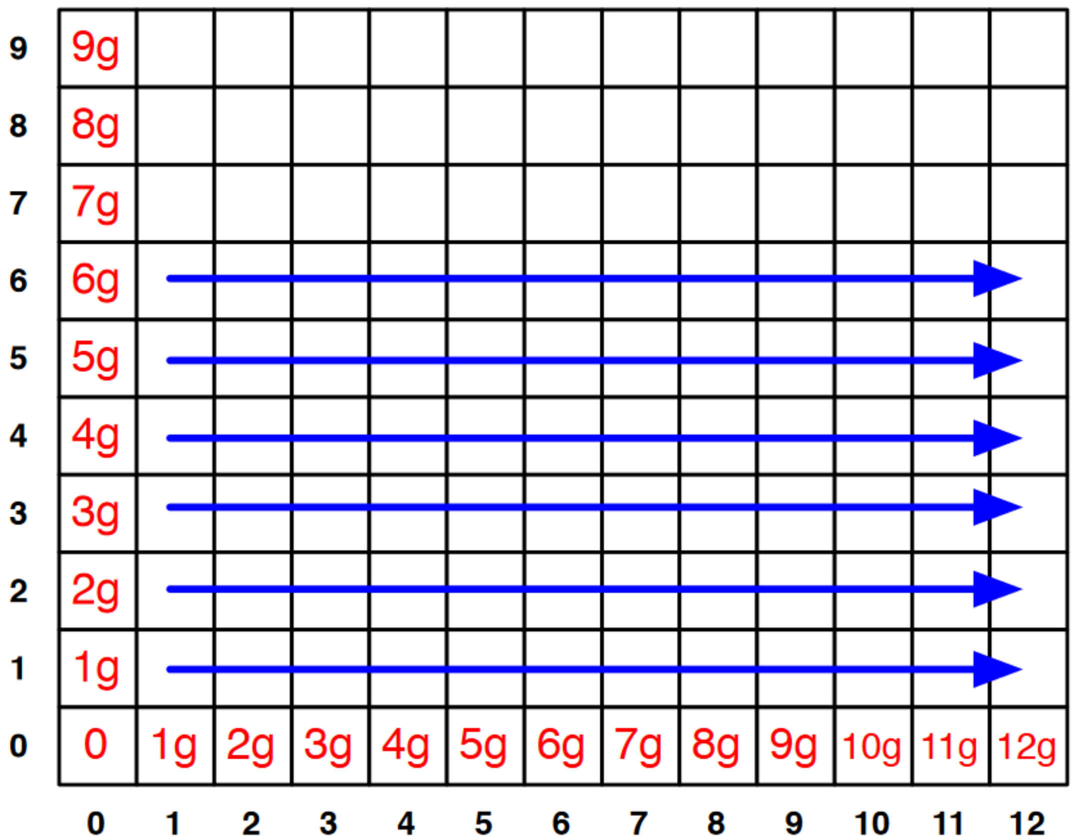
Zarovnávanie sekvencií – algoritmus (3)

- Riešenie reprezentované v tabuľke:



Zarovnávanie sekvencií – algoritmus (4)

- Tabuľku vypĺňame po riadkoch od najnižších indexov:



Zarovnávanie sekvencií – pseudokód

```
EditDistance(X,Y):
  for i = 1,...,m do A[i,0] = i*gap
  for j = 1,...,n do A[0,j] = j*gap

  for i = 1,...,m do
    for j = 1,...,n do
      A[i,j] = min(cost(a[i],b[j]) + A[i-1,j-1],
            gap + A[i-1,j],
            gap + A[i,j-1])
Return A[m,n]
```

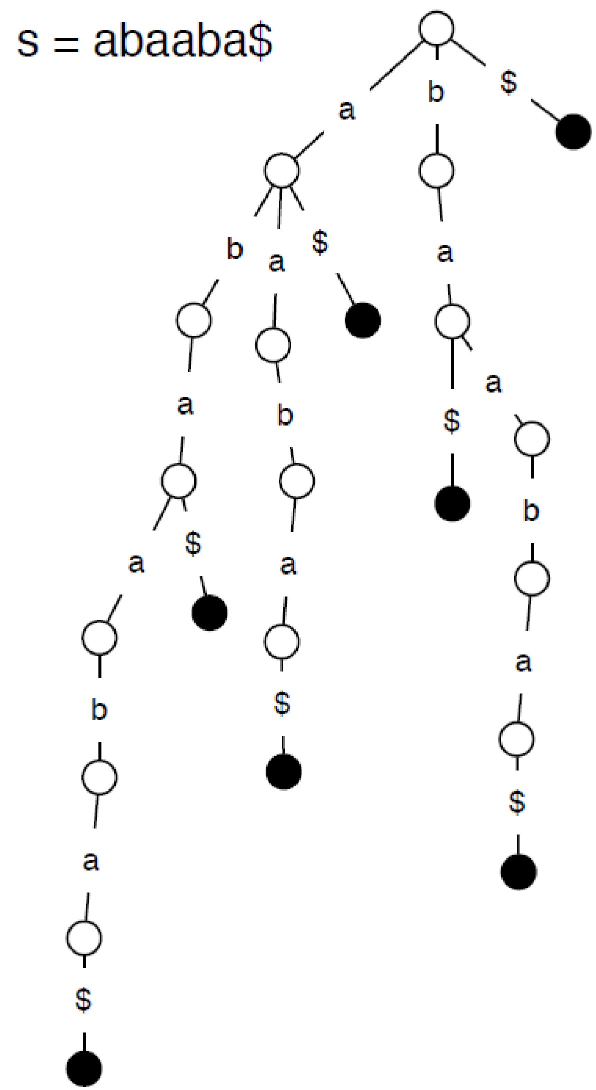
- Zložitosť výpočtu: O(nm)

Predspracovanie textu

- Typický scenár: dlhý nemeniaci sa reťazec-text (napr. genóm) a veľa rôznych (vopred neznámych) meniacich sa reťazcov-dopytov (query)
- Hlavná myšlienka: predspracovať text, tak aby sme vedeli rýchlo spracúvať dopyty nad týmto textom
- Ukážeme si dve základné dátové štruktúry
  - Sufixové stromy
  - Sufixové polia

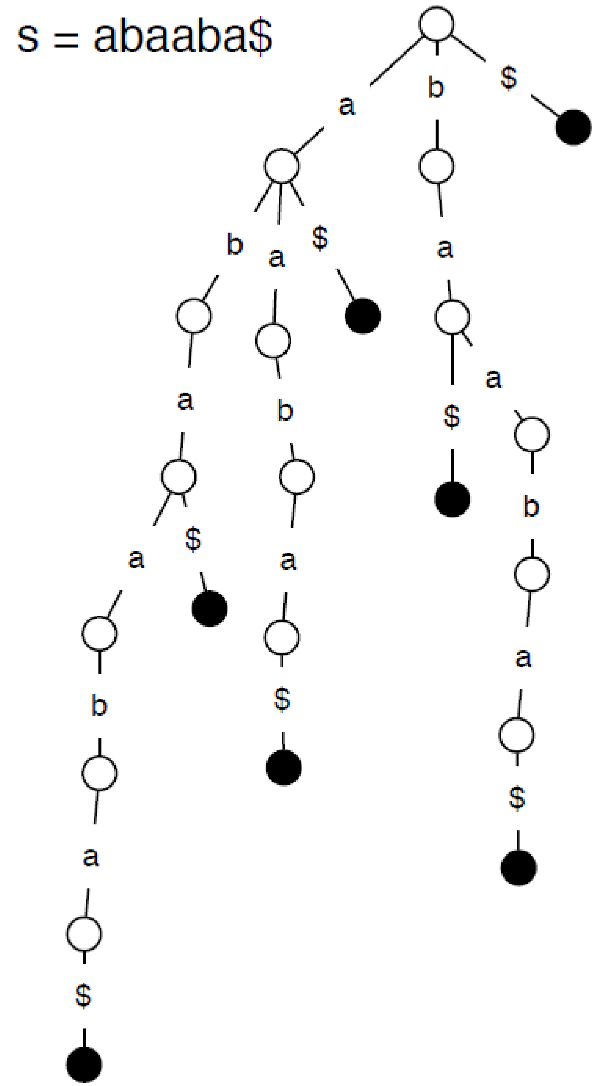
Sufixový strom (Suffix tree/trie)

- Písmenkový strom obsahujúci všetky sufixy textu S
- Ak by boli kľúče celé reťazce: binárny vyhľadávací strom by porovnával celé reťazce
- Písmenkový strom skúma reťazec po písmenkách
- Hrany sú znaky z abecedy ∑
- Každá cesta do listu reprezentuje nejaký sufix S
- Každý sufix je v strome



Sufixový strom (2)

- Daný je sufixový strom
pre text S a reťazec q, ako:
  - zistíme, či q je podreťazec S?
  - zistíme, či q je sufixom reťazca S?
  - spočítame počet výskytov q v reťazci S?
  - nájdeme najdlhší opakujúci sa podreťazec v S?
- Hlavná myšlienka:
každý podreťazec S je prefixom nejakého sufixu S.



Sufixový strom – zložitosť

- Konštrukcia sufixového stromu je komplikovaná :(
  - Existujú algoritmy ktoré ho vytvoria v čase O(n), kde n je dĺžka reťazca, ale sú zložité...
- Pamäťová náročnosť reprezentácie stromu môže byť tiež relatívne vysoká: až 20 bytov / znak v reťazci
- Hľadáme úspornejšie alternatívy

Sufixové pole (Suffix array)

- Pre daný reťazec (text) sufixy usporiadame abecedne:

1	attcatg$
2	ttcatg$
3	tcatg$
4	catg$
5	atg$
6	tg$
7	g$
8	$

→

8	$
5	atg$
1	attcatg$
4	catg$
7	g$
3	tcatg$
6	tg$
2	ttcatg$

- A ďalej použijeme len indexy sufixov: 8,5,1,4,7,3,6,2

Sufixové pole – hľadanie výskytov vzorky

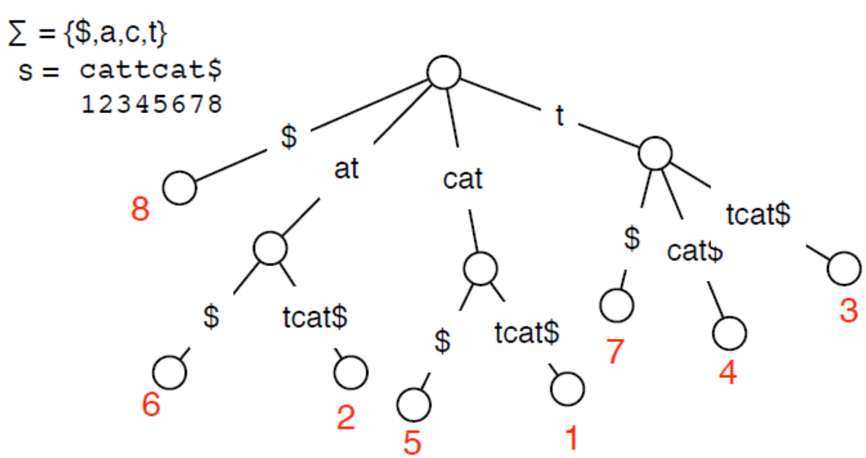
- Napr. hľadáme „at“
- Koľko krát sa nachádza v reťazci?
- Všetky výskyty budú v sufixovom poli vedľa seba...
- Postup:
  - Binárne vyhľadávanie nejakého sufixu začínajúceho na „at“
  - Spočítaj susedné sufixy, ktoré začínajú na „at“

s = cattcat$

8	$
6	at$
2	attcat$
5	cat$
1	cattcat$
7	t$
4	tcat$
3	ttcat$

Sufixové pole – konštrukcia

- Jednoduchý $O(n^2 \log n)$ algoritmus
  - Usporiadaj n sufixov:
    $O(n \log n)$ porovnaní, každé
    porovnanie dvoch sufixov je $O(n)$
- Viaceré algoritmy bežia v čase $O(n)$
  - napr. aj využitím sufixového stromu :)



s = cattcat$

8	$
6	at$
2	attcat$
5	cat$
1	cattcat$
7	t$
4	tcat$
3	ttcat$

Typy predspracovania

- **Predspracovanie vzorky P**
  (vzorka P sa nemení, text T sa mení)
  - Knuth-Morris-Prattov algoritmus (konštrukcia π)
  - Aho-Corasickovej algoritmus (konštrukcia automatu)
  - Boyer-Moorov algoritmus (konštrukcia indexov nad vzorkou pre výpočet pravidiel)
  - Rabin-Karpov (výpočet hashovacej hodnoty vzorky)

- **Predspracovanie textu T**
  (text T sa nemení, vzorka P sa mení)
  – Sufixový strom
  – Sufixové pole

STU FIIT

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Dátové štruktúry a algoritmy

Zhrnutie predmetu

12. 12. 2017

zimný semester 2017/2018

Zhrnutie predmetu – Čo sme preberali... (1)

- Vlastnosti algoritmov, zložitosť, asymptotické odhady
  - Odhady: Veľké „O“ (horný), Θ (tesný), Ω (dolný)
  - Najhorší prípad, priemerný, najlepší
- Abstraktná dátová štruktúra
  - Špecifikácia vs. Implementácia
- Správa pamäte pri vykonávaní programu
- Triedenie – Usporadúvanie
  - Porovnávaním
  - Spočítavaním

Zhrnutie predmetu – Čo sme preberali... (2)

- Vyhľadávanie
  - Lineárne, binárne, interpolačné
  - Tabuľka, strom, binárny vyhľadávací strom
  - Prehľadávanie stromov
  - Binárna halda (prioritný rad)
  - Vyvažovanie stromov (AVL, Splay, **Red-Black**)
  - B-stromy, Písmenkové stromy (trie), Binárne indexované stromy
- Hashovanie
  - Kolízie: Reťazenie, Otvorené adresovanie
  - Univerzálne, perfektné, konzistentné

Zhrnutie predmetu – Čo sme preberali... (3)

- Grafové algoritmy
  - Stupeň vrcholu, Eulerov ťah
  - Prehľadávanie grafov, mosty, artikulácie
  - Topologické usporiadanie
  - Najkratšie cesty: relaxácia, Dijkstra, Bellman-Ford, Floyd
  - Najlacnejšie kostry: Kruskal (+ Union-Find), Prim
  - Bludiská: prechod (konštrukcie grafov), generovanie
  - Bipartitné párovanie: Hopcroft-Karp, maximálny tok
  - Hamiltonovské grafy, TSP
  - Triedy zložitosti NP a P: NP-úplnosť, redukcie NP-úplných problémov
  - Aproximačné algoritmy, heuristiky

Zhrnutie predmetu – Čo sme preberali... (4)

- Výpočtová geometria
  - Analytická geometria: bod, úsečka, priamka, uhol, ...
  - Obsah mnohouholníka
  - Bod v mnohouholníku
  - Triangulácia: orezávanie uší
  - Konvexný obal: obaľovanie balíčka, horný a dolný obal
  - Zametanie: obsah zjednotenia N obdĺžnikov, priesečníky N úsečiek
  - Rozdeľuj a panuj: najbližšia dvojica bodov
  - Voronoiove diagramy a Delaunayova triangulácia
  - Vyhľadávanie vo viacrozmerných dátach (kD stromy)

Zhrnutie predmetu – Čo sme preberali... (5)

- Dynamické programovanie
  - 1D, 2D, intervalové, stromové, podmnožiny
- Algoritmy s reťazcami
  - Najdlhší palindróm
  - Syntaktická analýza: Cocke-Younger-Kasami
  - Vyhľadanie vzorky: Knuth-Morris-Pratt, konečný automat, Aho-Corasick, Rabin-Karp (hashovanie), Boyer-Moore
  - Bioinformatika: najkratšia spoločné nadslovo
    editačná vzdialenosť, zarovnávanie sekvencií
  - Sufixový strom
  - Sufixové pole

Zhrnutie predmetu – Čo si skúste odniesť...

- Prehľad nástrojov (algoritmov a dátových štruktúr) pre riešenie rozličných problémov
- Porozumieť vlastnostiam týchto nástrojov, najmä:
  - časovej efektívnosť (výpočtovej zložitosti)
  - pamäťovej efektívnosti (priestorovej zložitosti)
- Schopnosť aplikovať a prispôsobiť-upraviť tieto nástroje pre špecifické problémy
(za účelom dosiahnutia čo najlepšej efektívnosti)
- Otvorenosť pre ďalšie štúdium nových algoritmov a použitie efektívnych algoritmov a dátových štruktúr vo vašej ďalšej práci

Podmienky absolvovania

- Môžete získať až 100 bodov
- **Priebežne riešené úlohy** (zadania)
(max. 50 bodov: na cvičeniach 20 a doma 30):
  - na cvičení sa budú riešiť **malé úlohy**; (môže ich byť viac, každé najviac za 2 body), do konečného hodnotenia sa započítava najviac 20 bodov za všetky malé úlohy;
  - doma sa budú riešiť 3 **zadania**
každé max. 10 bodov, min. 4.
- **Priebežný test** (max. 15 bodov, treba získať min. 10)
- Podmienky udelenia zápočtu:
  - minimálne 25 bodov z priebežne úloh (vrátane zadaní)
  - minimálne 5 bodov z priebežného testu
- **Záverečná skúška** (max. 35 bodov, treba získať min. 15)

Čo bude na skúške…

- Odhady zložitosti (10%)
- Základný prehľad a vlastnosti (20%)
  - Algoritmy usporadúvania
  - Binárne vyhľadávanie
  - Vyhľadávacie stromy (binárne, písmenkové, ...)
  - Hashovanie
- Podrobne (60%)
  - Grafové algoritmy
  - Dynamické programovanie
  - Výpočtová geometria
  - Algoritmy s reťazcami
- Prehľad (10%)
  - NP, redukcie

STU FIIT

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Dátové štruktúry a algoritmy

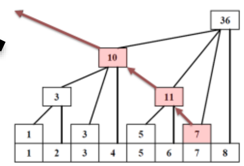
Quo vadis?

12. 12. 2017

zimný semester 2017/2018

Kam ďalej? (1)

Nové typy vyhľadávacích stromov

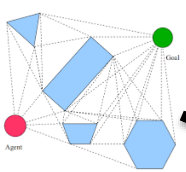


Spracovanie dlhých reťazcov

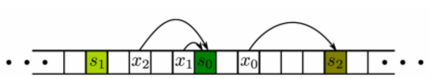
```
aa-bb-ccaabb
|x || | | |
ababbbc-a-b-
```

Algoritmy a dátové štruktúry

Spracovanie veľkých grafov, nové algoritmy



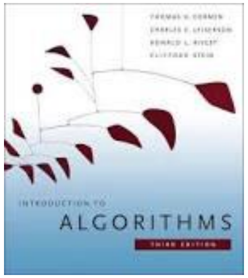
Plánovanie pohybu (robota), spracovanie obrazu



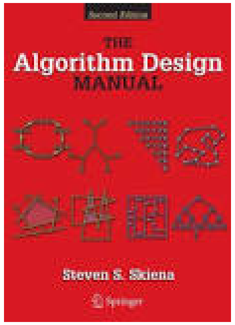
Nové spôsoby riešenia kolízií, spočítavania prvkov

Kam ďalej? (2)

- Algoritmy
  - **Introduction to Algorithms**
    Book by Charles E. Leiserson, Clifford Stein, Ronald Rivest, and Thomas H. Cormen

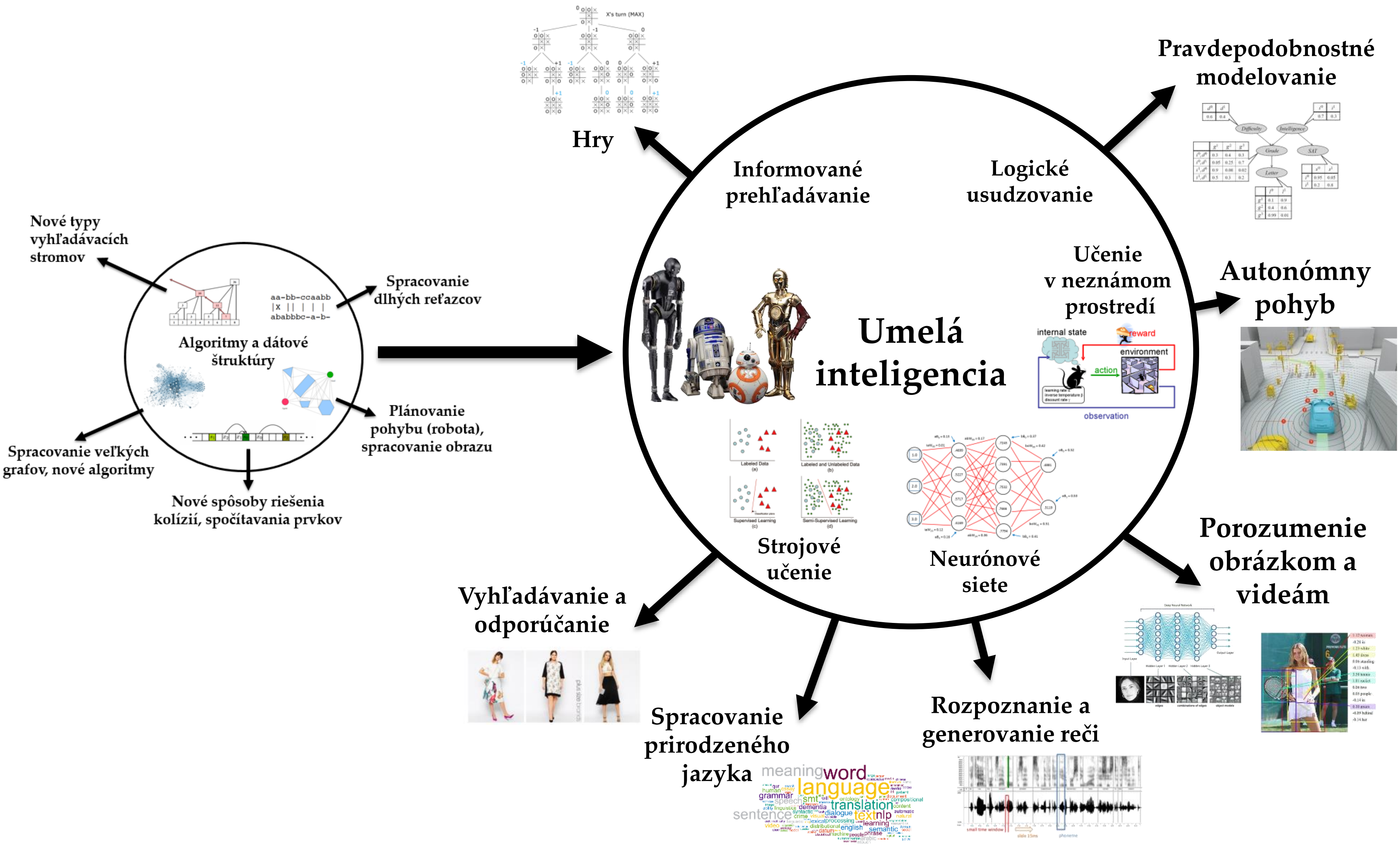


  - **The Algorithm Design Manual**
    Book by Steven Skiena



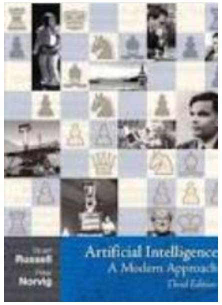
- Programátorské súťaže
  - TopCoder, ACM ICPC, Spoj, ...
- Nadväzujúce predmety na FIIT:
  - **Analýza a zložitosť algoritmov**
    doc. Mária Lucká
  - **Tvorba efektívnych algoritmov a programov**
    prof. Rastislav Kráľovič

Kam ďalej? (3)

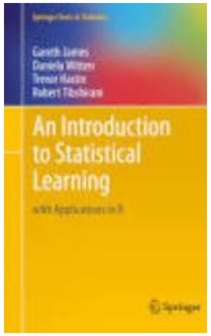

Hry
Pravdepodobnostné modelovanie
Informované prehľadávanie
Logické usudzovanie
Učenie v neznámom prostredí
Umelá inteligencia
Autonómny pohyb
internal state
reward
environment
action
observation
Strojové učenie
Neurónové siete
Porozumenie obrázkom a videám
Vyhľadávanie a odporúčanie
Spracovanie prirodzeného jazyka
Rozpoznanie a generovanie reči
Nové typy vyhľadávacích stromov
Spracovanie dlhých reťazcov
aa-bb-ccaabb
ababbbc-a-b-
Algoritmy a dátové štruktúry
Plánovanie pohybu (robota), spracovanie obrazu
Spracovanie veľkých grafov, nové algoritmy
Nové spôsoby riešenia kolízií, spočítavania prvkov
X's turn (MAX)
Labeled Data
Labeled and Unlabeled Data
Supervised Learning
Semi-Supervised Learning
meaning
word
language
grammar
speech
translation
sentence
text
nlp
dialogue

Kam ďalej? (4)

- **Artificial Intelligence: A Modern Approach**
  Book by Peter Norvig, Stuart J. Russel



- **An Introduction to Statistical Learning**
  Book by Daniela Witten, Gareth James, Robert Tibshirani, Trevor Hastie



- Data Science súťaže
  - Kaggle, ...

- Nadväzujúci predmet na FIIT:
  Umelá inteligencia
  Dr. Peter Lacko

Dovidenia nabudúce... (end credits)

- Prednášky:
  Jozef Tvarožek
- Cvičiaci:
  Róbert Cuprík
  Michal Farkaš
  Tomáš Farkaš
  Ivan Kapustík
  Samuel Pecár
  Jakub Ševcech
  Petra Vrablecová

Koniec.