10. prednáška (03. 05. 2012)

Vyhľadávanie v textoch

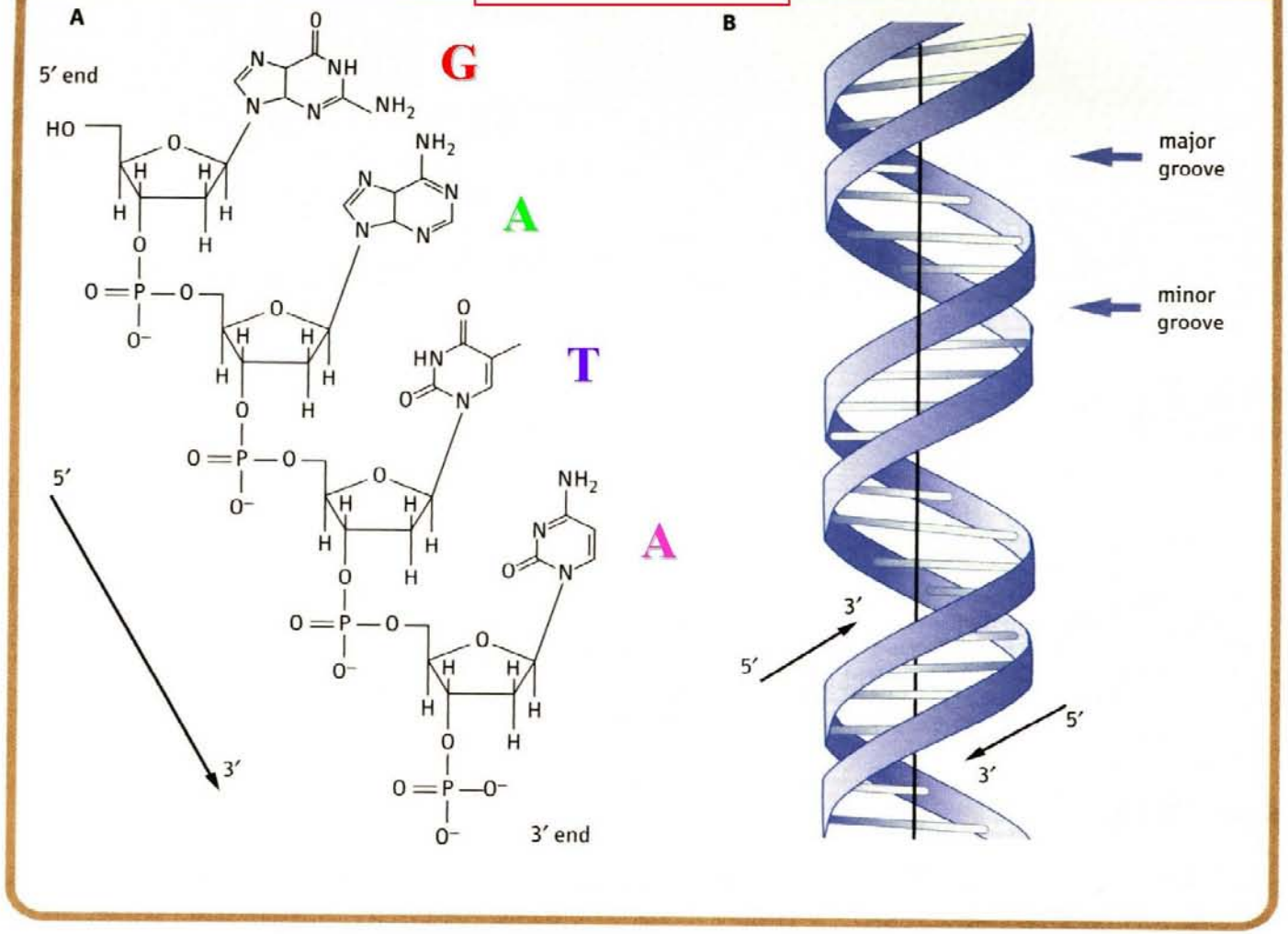
Vyhľadávanie v textoch

Úloha **nájsť v texte všetky výskyty zadaných textových vzoriek** (slov, alebo ich častí) patrí v praxi medzi najrozšírenejšie.

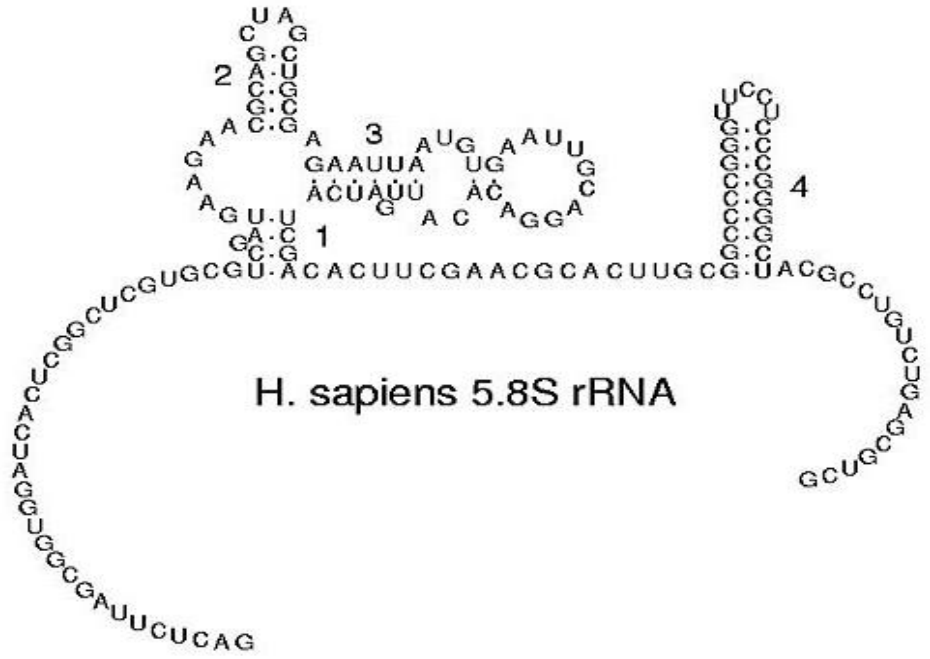
Kde všade sa stretneme s vyhľadávaním:

- textové editory,
- rešeršné systémy,
- ale i taký internet (google, zoznam, …),
- utility typu grep v Unixe,
- kontrola plagiátorstva

DNA



Vyhľadávanie génov



GACUCUUAGCGGUGGAUCACUCGGUCGUGCGUCAU...

Hľadáme úsek: AUCACU

Vyhľadávanie v textoch

صادف العيد لهذه السنة الاحد الماضي , كان يوما جميلا وطقسا رائعا , حيث استيقظنا مبكرا حوالي الساعة السادسة فجراً, ارتدينا ملابس جديدة , ذهب زوجي عبدالواحد الى المسجد حيث أقيمت صلاة العيد وكان المسجد مليئا بالناس وكانوا فرحين جدا وسعداء بهذه المناسبة ,عاد زوجي عبدالواحد من المسجد بعد انتهاء صلاة العيد الى المنزل وتبادلنا تهانئ العيد , ذهب عبدالواحد برفقة أصدقائه الى خارج المدينة وذهبت انا بصحبة ابني فرج الى صديقاتي وتجمعنا في بيت صديقةٍ لنا ... تناولنا وجبة الغداء مع بعض وجلسنا نتناول الحديث مع بعض, وأقمنا صلاة الظهر و صلاة العصر مع بعض , و تناولنا الشاي والحلويات , وبعد عودة زوجي وأصدقائه الى المدينة عدت انا وابني فرج الى المنزل وبعد لحظات زارني أخي عثمان وتناولنا وجبة العشاء مع بعض والشاي والحلويات والفواكه , وقبل مغادرته أقمنا صلاة المغرب وصلاة العشاء مع بعض .

Vyhľadávanie v textoch

TRANSLITEROVANÝ TEXT

SADF ALEED LEHADEH ALSANA YWM ALAHD ALMADE,KANA YWMN JAMEELN WA TAKSN RAEAN,HAITO ESTIKDNA MOBKKERN HWALE ALSAA ALSADESA FJRAN,ERTDINA MLABES JADEDA,WA MN TMMA ZHBA ZWJE ABDOLWAHD ELA ALMSJD HAITO OKEMT SALAT ALEED WA KANA ALMSJD MALEAN BENNAS WA KANA ALNNAS FAREHEEN JEDDAN WA SOADAA BEHADEH ALMNASBA,ADA ZWJE MNA ALMSJD BAAD SALAT ALEED ELA ALMNZL WA TBADLNA THANE ALEED, ZHBA ABDOLWAHD BEREFKT ASDEKAEH ELA KAREJ ALMDENA WA ZHBTO ANA BEREFKT EBNE FRJ ELA SADEKATE WA TAJMMAANA FE BAIT SADEKATEN LNA ESMHA DENA, TNAWLNA WJBT ALKDAA MAA BAAD WA JLSNA NTNAWL ASHAAI WA ALHELWEIAT ,WA AKMNA SALAT ALDHUHR WA SALAT ALASR MA BAAD ,KANA YWMN JAMEELN WA RAEAN,WA BAAD AWDT ZWJE WA ASDEKAEH ELA ALMDENA OTTO ANA WA EBNE ELA ALBAIT WA BAAD LHDAAT ZARNE AKEE OTMAN WA TNAWLNA WJBT ALASHA WA KDALEK ASHAAI WA ALHELWEIAT,WA KBL MOKHADRTEH AKMNA SALAT ALMGHRB WA ALESHAA MA BAAD .

Vyhľadávanie v textoch

- vyhľadávanie v bežnom textovom dokumente - nie je to problém, stačí nám jednoduchý „naivný algoritmus“,
- ale v prípade veľkých rešeršných systémov by naivný algoritmus pracoval veľmi pomaly, preto je dôležité hľadať efektívnejšie riešenia

Vyhľadávanie v textoch

Obsah

- Naivný algoritmus
- Knuth Morris Prattov algoritmus
- Boyer-Mooreov algoritmus
- Rabin-Karpov algoritmus

Vyhľadávanie v textoch

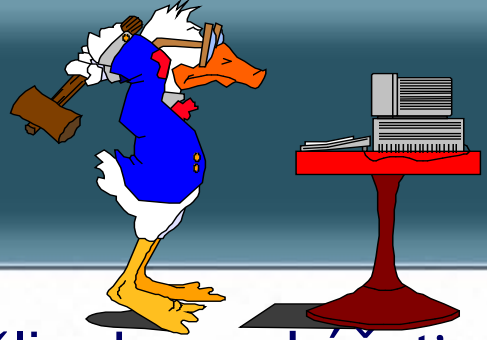
- Považujme text za dlhý reťazec znakov
- Hľadané slovo je krátky reťazec znakov
- Základný problém – nájsť slovo v texte.
- Je možné hľadať niektorý výskyt, ale tiež všetky výskyty.

```
les   Do lesa vedie cesta cez prales Koleso
```

História algoritmov

- Brute force algoritmus
  - známy veľmi dávno
  - preskúmavaný viackrát, stále zostáva východiskom
- Knuth & Pratt, 1970
  - nezávisle od nich toto vylepšenie našiel Morris
  - spoločne publikovali 1976 ako “Knuth-Morris-Pratt (KMP)”
- Boyer & Moore našli lepší algoritmus ešte pred 1976
  - nezávisle od nich toto vylepšenie našiel aj Gosper
- Karp & Rabin našli ďalšie vylepšenie v 1980

Brute force



- Najobvyklejší algoritmus je založený na myšlienke: pokúšať sa nájsť slovo od každej pozície v texte a porovnávať všetky znaky slova so znakmi v texte :

```
for  (i = 0;  i<=n-m; i++) { //dokument t dĺžky n
   for  (j = 0; j<= m-1; j++) {//slovo p dĺžky m
      porovnaj p[j] s t[i+j]
      if sa nerovná, exit vnútorného cyklu
      }
      //V prípade, že prebehne celý vnútorný cyklus,
      //vzorový reťazec sa v cieľovom vyskytuje.
      }
```

- Zložitosť je v najhoršom prípade O(m*n) a v najlepšom O(n).

Vylepšenie vyhľadávania v texte

- Existuje rýchlejší algoritmus, v ktorom sa porovnávajú posledné znaky najprv:

```
Do the first then do the other one
the
```

porovnaj ‘e’ s‘ ‘, nerovný, tak sa posuň o 3 miesta

```
Do the first then do the other one
   the
```

Je možný posun len 2 miesta

Vylepšenie vyhľadávania v texte (2)

Pozorovanie:

- V každom prípade, keď daný znak v dokumente nie je ani jedným zo znakov hľadaného slova, je možný posun o *n* znakov. V niektorých prípadoch je to menej.

Definícia problému, terminológia

- Nech *p* je vzorový reťazec (pattern string)
- Nech *t* je cieľový reťazec (target string)
- Nech k je index znaku v cieľovom reťazci, ktorý "leží nad" prvým znakom vzorového reťazca
- **Úloha:**
- Pre dané dva reťazce, *p* a *t*, v abecede Σ, určiť, či sa *p* vyskytuje ako podreťazec v *t*
- Teda, určiť či existuje index *k* taký, že platí *p=Substring(t, k, |p|).*

Priame vyhľadávanie v reťazci

Pseudokód:

function *SimpleStringSearch*(string p,t): **integer;**

// Nájde prvý výskyt *p* v *t (ak existuje)*;
// vráti jeho pozíciu alebo -1, ak *p* nie je podreťazcom *t*

```
for k = 0 to Length(t) – Length(p) do {
        i = 0;
        while i < Length(p) a zároveň p[i] = t[k+i] do
                i = i+1;
        if i == Length(p) then return k
}
return -1
```

Aplikácia *SimpleStringSearch*

t[0]	*t[1]*	*t[2]*	*t[3]*	*t[4]*	*t[5]*	*t[6]*	*t[7]*	*t[8]*	*t[9]*	*t10]*
A	B	C	E	F	G	A	B	C	D	E

p[0]	*p[1]*	*p[2]*	*p[3]*
A	B	C	D

Y **Y** **Y** **N**

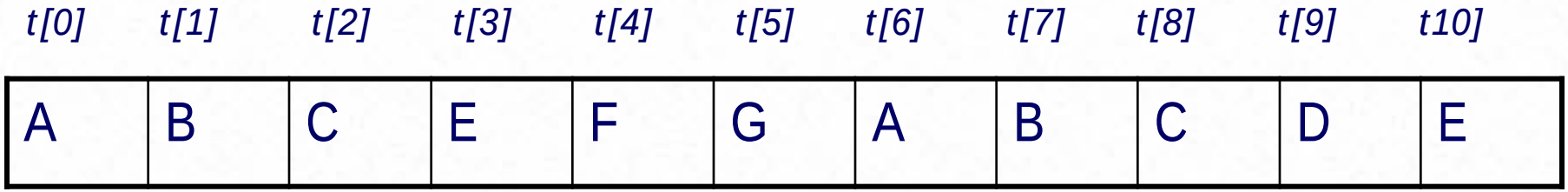
Aplikácia *SimpleStringSearch*

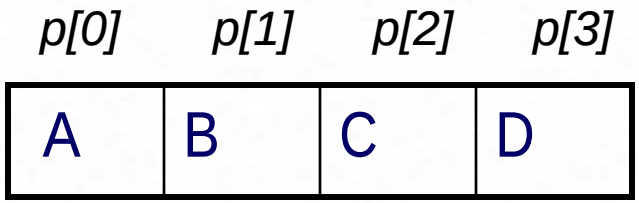
t[0]	*t[1]*	*t[2]*	*t[3]*	*t[4]*	*t[5]*	*t[6]*	*t[7]*	*t[8]*	*t[9]*	*t10]*
A	B	C	E	F	G	A	B	C	D	E

p[0]	*p[1]*	*p[2]*	*p[3]*
A	B	C	D

N

Aplikácia *SimpleStringSearch*





N

Aplikácia *SimpleStringSearch*

t[0]	*t[1]*	*t[2]*	*t[3]*	*t[4]*	*t[5]*	*t[6]*	*t[7]*	*t[8]*	*t[9]*	*t10]*
A	B	C	E	F	G	A	B	C	D	E

p[0]	*p[1]*	*p[2]*	*p[3]*
A	B	C	D

N

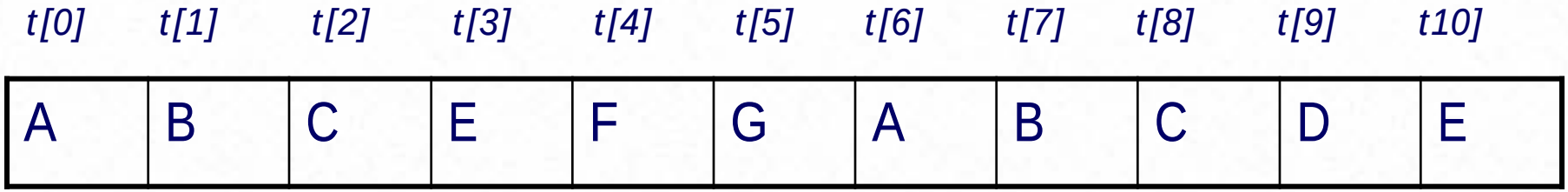
Aplikácia *SimpleStringSearch*

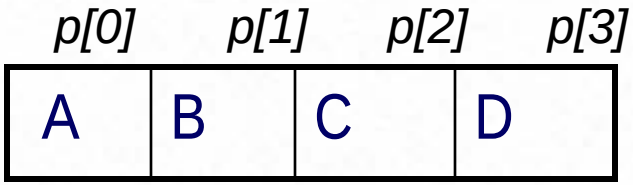
t[0]	*t[1]*	*t[2]*	*t[3]*	*t[4]*	*t[5]*	*t[6]*	*t[7]*	*t[8]*	*t[9]*	*t10]*
A	B	C	E	F	G	A	B	C	D	E

p[0]	*p[1]*	*p[2]*	*p[3]*
A	B	C	D

N

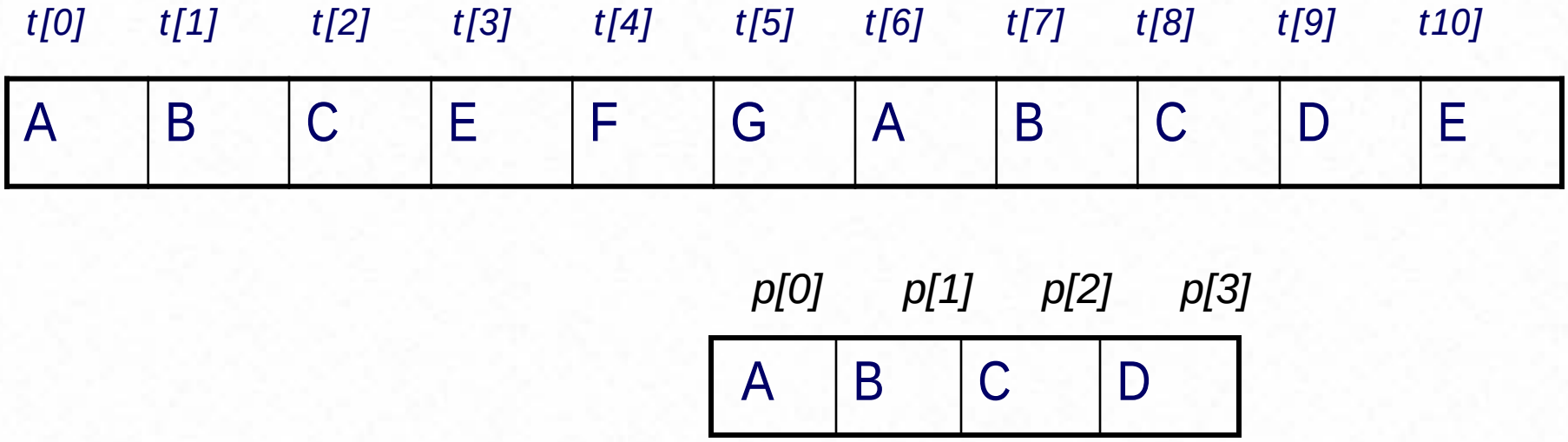
Aplikácia *SimpleStringSearch*





N

Aplikácia *SimpleStringSearch*



Aplikácia *SimpleStringSearch*

t[0]	t[1]	t[2]	t[3]	t[4]	t[5]	t[6]	t[7]	t[8]	t[9]	t10]
A	B	C	E	F	G	A	B	C	D	E

p[0]	p[1]	p[2]	p[3]
A	B	C	D

Y Y Y Y

Zložitosť *SimpleStringSearch*

- Najhorší prípad:
  - Vzorový reťazec sa vždy zhodne s textom okrem posledného znaku
  - Príklad: hľadať XXXXY v cieľovom reťazci XXXXXXXXXXXXXXXXXXX
  - Vonkajší cyklus je vykonávaný pre každý znak v cieľovom reťazci
  - Vnútorný cyklus sa vykoná jedenkrát pre každý znak vo vzorovom reťazci
  - $\Theta(|p| * |t|)$
- Je to v poriadku, keď sú reťazce krátke, ale existuje lepší algoritmus

Knuth-Morris-Pratt

- Kľúčová idea:
  - ak vzor sa v znaku nezhodne s cieľom, tak posunúť vzor napravo o toľko znakov, koľko je najviac možné so zachovaním už zistenej zhody

Knuth-Morris-Pratt

t[0]	t[1]	t[2]	t[3]	t[4]	t[5]	t[6]	t[7]	t[8]	t[9]	t10]
X	Y	X	Y	X	Y	c				

p[0]	p[1]	p[2]	p[3]	p[4]
X	Y	X	Y	Z

Y Y Y Y N

X	Y	X	Y	Z

Y Y Y Y ?

Knuth-Morris-Pratt

- Správne posunutie vzoru závisí od oboch indexov nezhody a od znakov nezhody
- Ak *c* == X : posun o 2 znaky napravo
- Ak *c* == E : posun o 5 znakov napravo
- Ak *c* == Z : zhoda nájdená; algoritmus končí

Knuth-Morris-Pratt

- **Cieľ:** určiť d, počet znakov, o ktoré je možný posun napravo; nájsť najmenšie d také, že:
  - p[0] = t[k+d]
  - p[1] = t[k+d+1]
  - p[2] = t[k+d+2]
  - ...
  - p[i-d] = t[k+i]

Knuth-Morris-Pratt

- **Poznámka:** pomocou vzoru samotného a znaku, ktorý spôsobil nezhodu
- Hodnota d závisí len od:
  - Vzorového reťazca
  - Hodnoty i
  - Od znaku v textovom reťazci, ktorý sa nezhoduje - znak c (v položke t[k+i])

Knuth-Morris-Pratt

- Môžeme definovať funkciu *KMPskip(p, i, c)*, ktorá bude počítať správne d
  - Vráti najmenší index d, $0 \le d \le |p|$, taký že p[i-d] == c a p[j] == p[j+d] pre každé $0 \le j \le$ i-d-1
  - Vráti i+1, ak žiadne také d neexistuje
- Vypočítať všetky hodnoty *KMPskip* pre vzor p a uložiť ich v dvojrozmernom poli *KMPskiparray*
- Urobiť cyklus pre každú nezhodu

Knuth-Morris-Pratt

- Pre vzorový reťazec ABCD :

c =	p= A	B	C	D
A	0	1	2	3
B	1	0	3	4
C	1	2	0	4
D	1	2	3	0
iné	1	2	3	4

Vráti najmenší index d, $0 \le d \le |p|$, taký že
p[i-d] == c a
p[j] == p[j+d] pre každé
$0 \le j \le i-d-1$

c==A, i=0, p[i-0]==c==A

Teda d=0

p[j] == p[j+d] pre všetky j,

j vlastne neexistuje

Knuth-Morris-Pratt

- Pre vzorový reťazec ABCD :

c =	p= A	B	C	D
A	0	1	2	3
B	1	0	3	4
C	1	2	0	4
D	1	2	3	0
iné	1	2	3	4

Vráti najmenší index d, $0 \le d \le |p|$, taký že
p[i-d] == c a
p[j] == p[j+d] pre každé
$0 \le j \le i-d-1$

c==A, i=1, p[i-1]==c==A

Teda d=1

p[j] == p[j+d] pre všetky j,

j vlastne neexistuje

Knuth-Morris-Pratt

- Pre vzorový reťazec ABCD :

c =	p= A	B	C	D
A	0	1	2	3
B	1	0	3	4
C	1	2	0	4
D	1	2	3	0
iné	1	2	3	4

Vráti najmenší index d, $0 \le d \le |p|$, taký že
p[i-d] == c a
p[j] == p[j+d] pre každé
$0 \le j \le i-d-1$

c==A, i=2, p[i-2]==c==A

Teda d=2

p[j] == p[j+d] pre všetky j,

j vlastne neexistuje

Knuth-Morris-Pratt

- Pre vzorový reťazec ABCD :

c =	p= A	B	C	D
A	0	1	2	3
B	1	0	3	4
C	1	2	0	4
D	1	2	3	0
iné	1	2	3	4

Vráti najmenší index d, $0 \le d < |p|$, taký že
p[i-d] == c a
p[j] == p[j+d] pre každé
$0 \le j \le i-d-1$

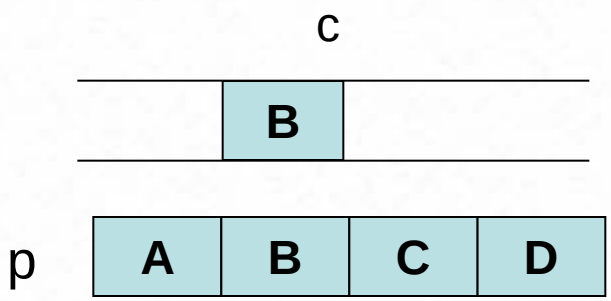
c==A, i=3, p[i-3]==c==A

Teda d=3

p[j] == p[j+d] pre všetky j,

j vlastne neexistuje

Knuth-Morris-Pratt



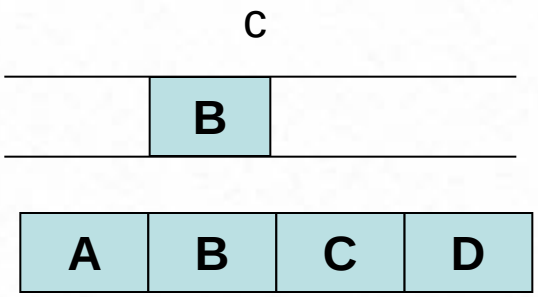
Ak je porovnávaný prvý prvok vzorového reťazca s prvkom v c, je potrebný posun o 1 znak.

$P[0] = A \neq B$, teda $d=1$

Ak je porovnávaný druhý prvok vzorového reťazca s prvkom v c, je potrebný posun o 0 znakov.

$P[1]=B$, teda $d=0$

Knuth-Morris-Pratt



Ak je porovnávaný tretí prvok vzorového reťazca s prvkom v c, je potrebný posun o 3 znaky.

P[2]=C≠B, je možný posun o d=3

Ak je porovnávaný štvrtý prvok vzorového reťazca s prvkom v c, je potrebný posun o 4 znaky.

P[3]=D≠B, je možný posun o d=4

Toto je zaznamenané v druhom riadku matice.

Knuth-Morris-Pratt

- Pre vzorový reťazec XYXYZ:

	X	Y	X	Y	Z
X	0	1	0	3	2
Y	1	0	3	0	5
Z	1	2	3	4	0
iné	1	2	3	4	5

Knuth-Morris-Pratt

```
Function KMPSearch(string p, t): integer
{Nájsť p v t; vráti jeho pozíciu alebo  -1 ak p nie je podreťazcom v t}
KMPskiparray = Vypočítať KMPskiparray(p)
K = 0;
i = 0;
While k < Length(t) – Length(p) do {
    if i == Length(p) then return k
    d = KMPskiparray[I, t[k+i]];
    k = k + d;
    i = I + 1 –d;
}
Return -1;
```

Knuth-Morris-Pratt

KMPskiparray = *Vypočítať KMPskiparray(p)*

- Tento príkaz nám nevysvetľuje, ako možno pole *KMPskiparray* vypočítať
- Autori tento výpočet zrealizovali pomocou jednorozmerného poľa – priložený program pracuje s týmto jednorozmerným poľom, argumenty je treba zadať ako ${string_prompt}.

Boyer-Moore Algoritmus

- Podobný KMP v tomto:
  - Vzorový reťazec je porovnávaný s cieľovým
  - Pri nezhode je posun doprava čo najďalej
- Rozdielny od KMP v tomto:
  - Porovnáva vzory sprava doľava namiesto zľava doprava
- Spôsobí to rozdiel?
  - Áno!! – oveľa rýchlejší na veľkých textových reťazcoch; mnoho znakov v cieľovom reťazci nemusí byť odskúšaných

Boyer-Moore príklad

t[0]	t[1]	t[2]	t[3]	t[4]	t[5]	t[6]	t[7]	t[8]	t[9]	t10]
A	B	C	E	F	G	A	B	C	D	E

p[0]	p[1]	p[2]	p[3]
A	B	C	D

N

Neexistuje žiadne E vo vzore : teda vzor sa nemôže zhodovať so žiadnym znakom ležiacim pod t[3]. Takže je možný posun o 4 znaky napravo.

Boyer-Moore príklad

t[0]	t[1]	t[2]	t[3]	t[4]	t[5]	t[6]	t[7]	t[8]	t[9]	t10]
A	B	C	E	F	G	A	B	C	D	E

p[0]	p[1]	p[2]	p[3]
A	B	C	D

N

Znovu, žiadna zhoda. Ale B je vo vzore. Takže je možný posun o 2 znaky napravo.

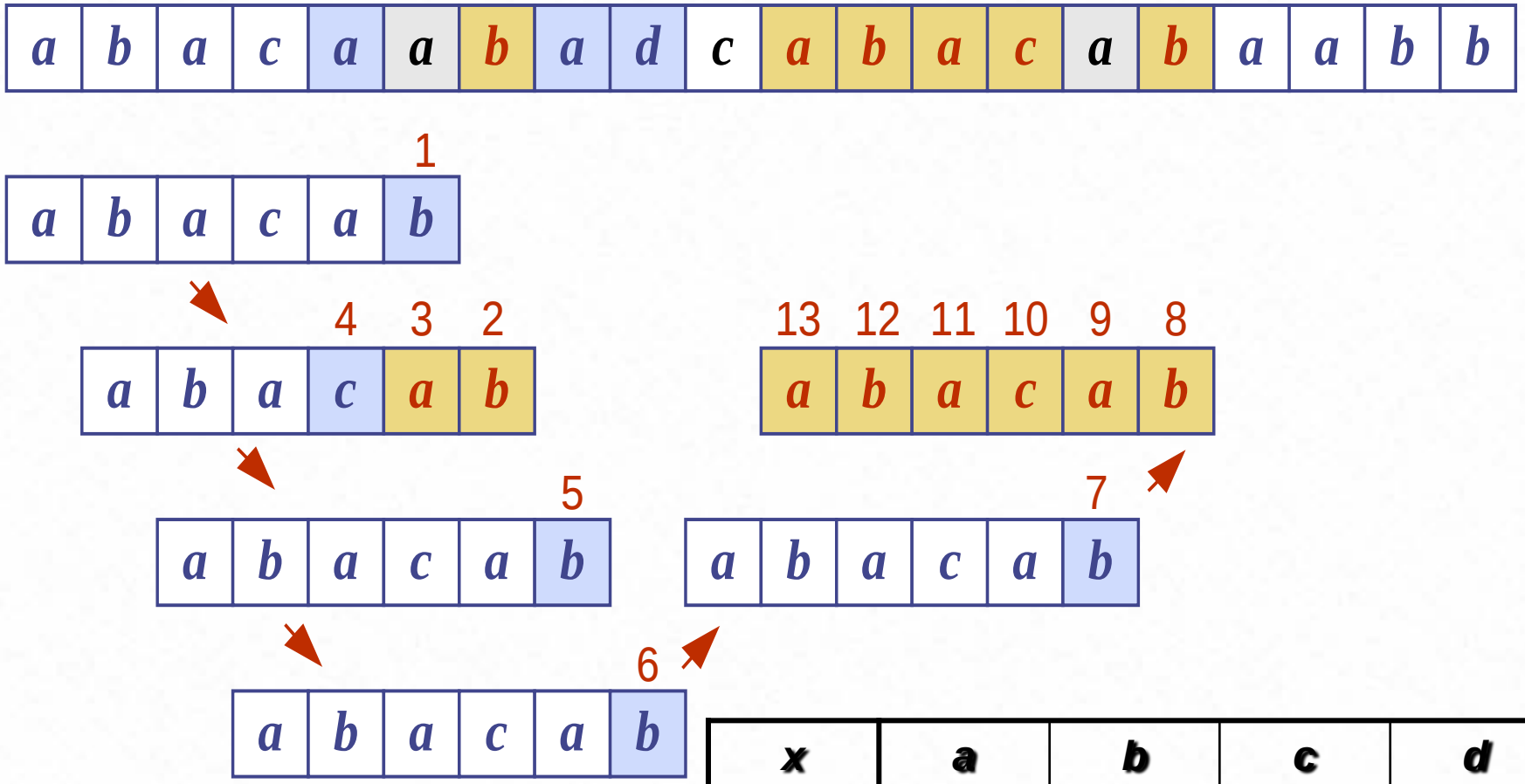
Boyer-Moore príklad

t[0]	t[1]	t[2]	t[3]	t[4]	t[5]	t[6]	t[7]	t[8]	t[9]	t10]
A	B	C	E	F	G	A	B	C	D	E

p[0]	p[1]	p[2]	p[3]
A	B	C	D

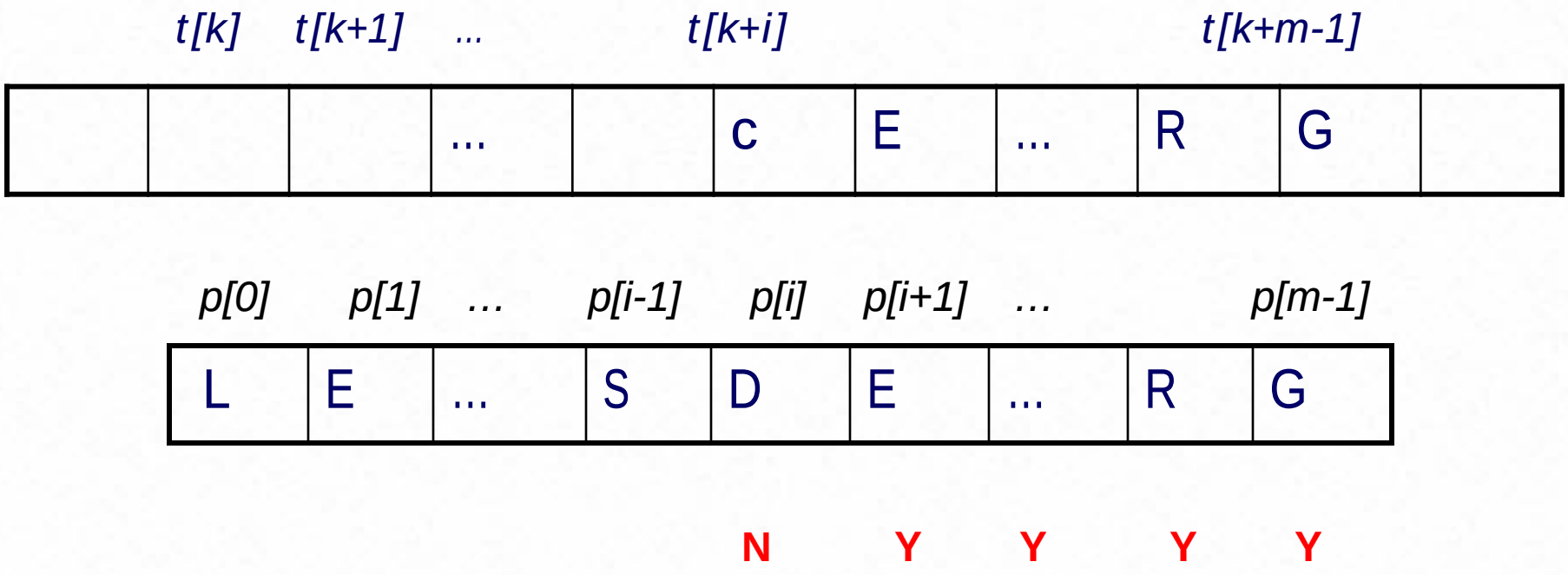
Y Y Y Y

Boyer-Moore príklad



x	a	b	c	d
	4	5	3	–1

Boyer-Moore : iný príklad



Problém: určiť d, počet znakov, o ktoré môže byť vzor posunutý napravo

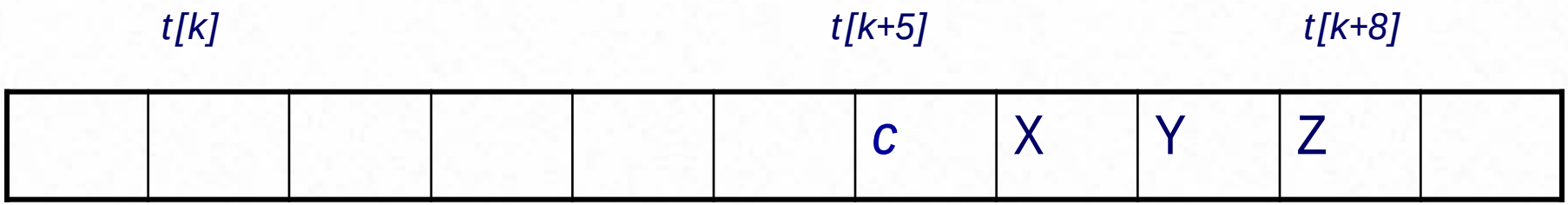
d by malo byť najmenšie také, že t[k+m-1]= p[m-1-d], t[k+m-2] = p[m-2-d], … t[k+i] = p[i-d]

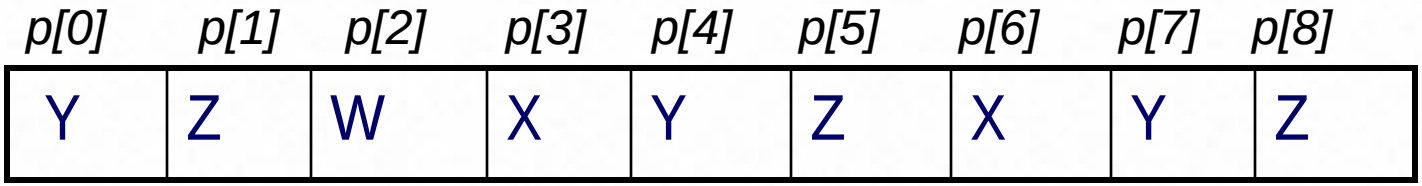
Boyer-Moore Algoritmus

Vieme:

- **d by malo byť najmenšie také, že:**
  - t[k+m-1] = p[m-1-d]
  - t[k+m-2] = p[m-2-d]
  - t[k+i] = p[i-d]
- **Pripomíname:**
  - k = štartovací index v cieľovom reťazci
  - m = dĺžka vzorového reťazca
  - i = index nezhody
- **Problém: príkaz je vyhovujúci pre d<= i**
  - Potrebujeme zabezpečiť, aby sme neprekročili ľavú stranu vzoru.

Boyer-Moore : iný príklad





N Y Y Y

Ak c == W, tak d by mohlo byť 3

Ak c == R, tak d by mohlo byť 7

BMPSkip

- Nech $m = |p|$
- Pre ľubovoľný znak c a ľubovoľné i také, že $0 \leq i < m$, definujeme *BMPSkip(p, i, c)* takto:
  - Časť vzoru je možné posunúť napravo, keď znaky i+1 až m-1 vzoru sa zhodujú s odpovedajúcimi znakmi v cieľovom reťazci, ale p[i] sa nezhoduje so znakom c.
- Potom BMPSkip(p, i, c) by mala vrátiť najmenšie d také, že:
  - $p[j] = p[j-d]$ pre všetky j také, že $max(i+1,d) \leq j \leq m-1$, a
  - $p[i-d] = c$ ak $d \leq i$

Boyer-Moore

- Pre vzor ABCD:

A nezhodný znak v cieľovom reťazci je tento -→

	A	B	C	D
A	-	4	4	3
B	4	-	4	2
C	4	4	-	1
D	4	4	4	-
iné	4	4	4	4

<- ak pozícia vo vzore je tento znak

Potom je možné vynechať toľko …

Boyer-Moore

- Pre vzor XYXYZ:

A nezhodný znak v cieľovom reťazci je tento -→	X	Y	X	Y	Z
X	-	5	-	5	2
Y	5	-	5	-	1
Z	5	5	5	5	-
iné	5	5	5	5	5

<- ak pozícia vo vzore je tento znak

Potom je možné vynechať toľko …

Poznámka:

- Hodnoty v Boyer-Moore poliach sú vo všeobecnosti väčšie než pri KMP; teda vzor sa môže posúvať rýchlejšie

BMSearch

```
Function BMSearch(string p, t): int
{Nájde p v t; vráti jeho pozíciu alebo -1 ak p sa v t nenachádza}
    BMSkiparray = Vypočítať BMSkipArray(p)
    k = 0
    while k <= Length(t) – Length(p) do {
        i = Length(p) – 1
        while i >= 0 and p[i] = t[k+i] do
            I = i– 1;
        if i = -1 then return k
        k = k + BMSkiparray[i, t[k+i]]]
}
    return -1
```

BMSearch

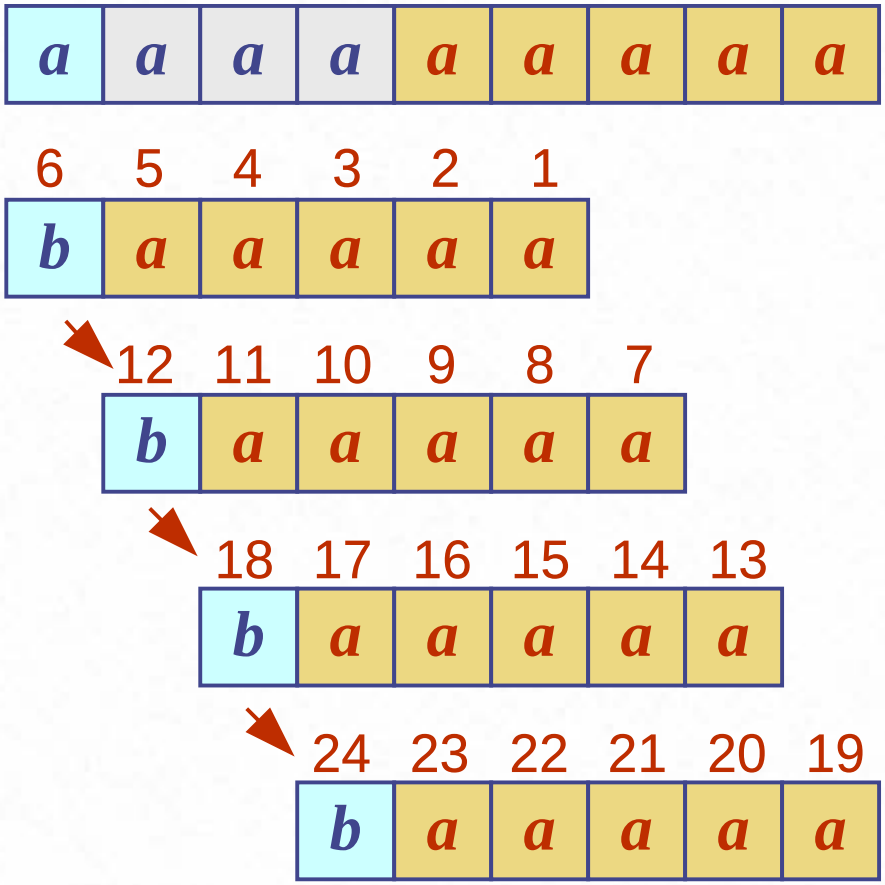
BMSkiparray = *Vypočítať BMSkipArray(p)*

- Tento príkaz nevysvetľuje, ako vypočítať dvojrozmerné pole BMSkiparray
- Pokúste sa hľadať nejaké súvislosti, ktoré by viedli k tomu, aby nebolo nutné počítať toto pole.

Zložitosť BM algoritmu

- Boyer-Moore algoritmus pracuje v čase $O(nm + s)$
- Príklad najhoršieho prípadu:
  - $T = aaa \ldots a$
  - $P = baaa$
- Najhorší prípad sa môže vyskytnúť v obrázkoch alebo v DNA sekvenciách. V obyčajných textoch je málo pravdepodobný.
- Boyer-Moore algoritmus je významne rýchlejší ako brute force algoritmus na anglických textoch.

BM algoritmus – najhorší prípad

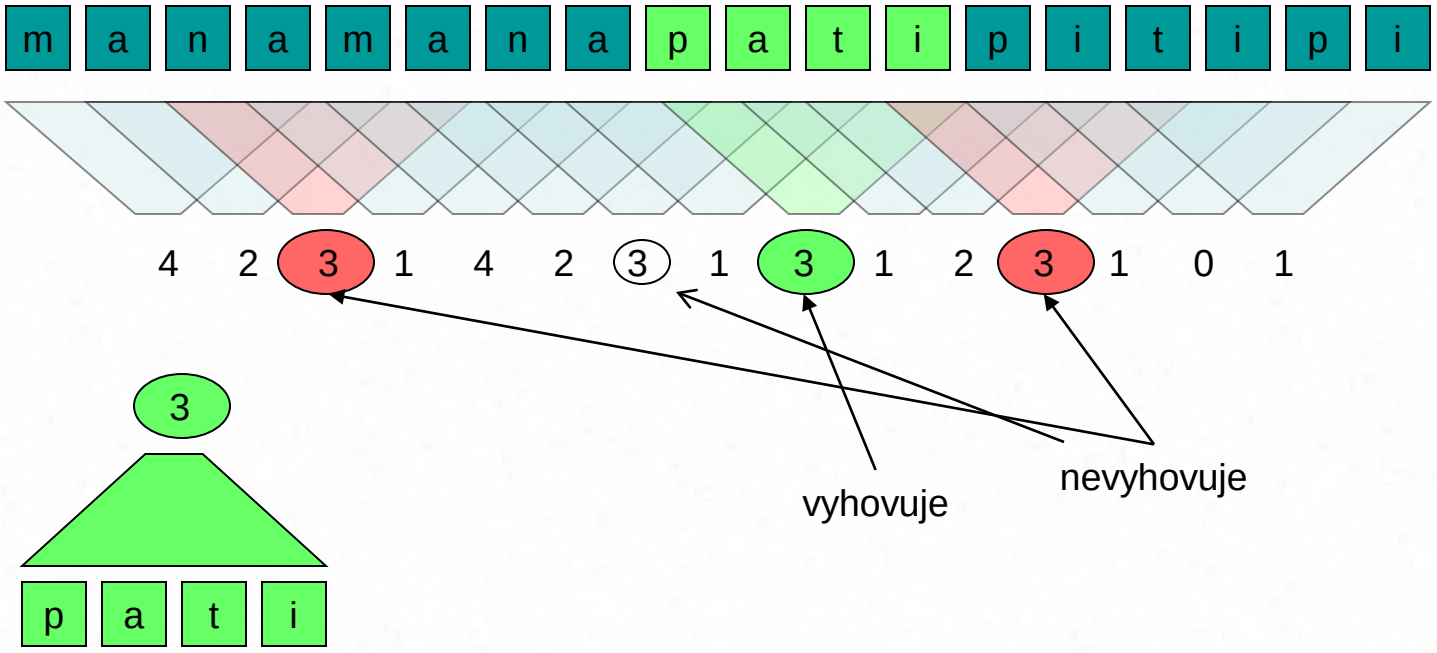


Idea Karp-Rabinovho algoritmu

- Karp & Rabin našli algoritmus, ktorý je:
  - Skoro taký rýchly ako Boyer-Moore
  - Dosť jednoduchý na pochopenie
  - Môže byť použitý na hľadanie v 2-dimenzionálnych cieľových obrázkoch
- Poďme naspäť k brute force idee, ale teraz použijeme na jediné číslo reprezentáciu slova, ktoré hľadáme, a jediné číslo pre bežnú časť dokumentu, v ktorom hľadáme, s ktorou práve porovnávame.

Rabin-Karp algoritmus

- Idea: Vypočítať
  - Kontrolný súčet pre vzor P a
  - Kontrolný súčet pre každý podreťazec T dĺžky m



Rabin - Karp algoritmus

- Výpočet kontrolných súčtov:
  - Vyberieme prvočíslo q, desiatková sústava
  - Nech d = $|\Sigma|$
- Example:
  - Nech d = 10, q = 13, Σ = { a, b, c, d, e, f, g, h, I, j }
  - Nech P=‘albf’= ‘0815’

$S_4(0815) = (0 \cdot 1000 + 8 \cdot 100 + 1 \cdot 10 + 5 \cdot 1) \mod 13 = 815 \mod 13 = 9$

T= 1708650854108151

Rabin- Karp algoritmus

Rabin-Karp-Matcher(T,P,d,q)

1. n ← length(T)
2. m ← length(P)
3. h ← d^{m-1} mod q
4. p ← 0
5. t_0 ← 0
6. for i ← 1 to m do
7. p ← (d p + P[i]) mod q
8. t_0 ← (d t_0 + T[i]) mod q

od

9. for s ← 0 to n-m do
10. if p = t_s then
11. if P[1..m] = T[s+1..s+m] then return "Pattern occurs with shift" s

fi

12. if s < n-m then
13. t_{s+1} ← (d(t_s-T[s+1]h) + T[s+m+1]) mod q

fi

od

Karp-Rabin Algoritmus

- Predpokladajme, že hľadáme 4-znakové slovo. Potom celé slovo (zakódované ako číslo) sa musí zhodovať so slovom **w** dĺžky 4 byty. Ak priebežné 4 byty dokumentu sú považované za jedno slovo **d**, jediné porovnanie môže vyjadriť, či sa rovnajú alebo nie. Posunúť pozdĺž dokumentu, v **d** pridať ďalší znak a odstrániť prvý znak.

Karp-Rabin Algoritmus

- Pre dlhšie slová použiť hashing. Znaky slova a dokumentu sú kombinované do jednoduchých hashových čísel **wh** a **dh**. Hašové číslo **dh** môže byť upravené pomocou vhodných matematických operácií v kóde pri pridaní ďalšieho znaku.

Rabin-Karp príklad

- Text „ahojrabinakarpakosamate“
- Hľadame vyraz „karp“
- Pre našu hashovaciu funkciu zvolime B=64 a P=113
- P – Prvočislo, B – Mocnina dvojky (sústava)
- karp =107,97,114,112 (hodnoty znakov v ASCII)
- Text v ASCII :
- 97,104,111,106,114,97,98,105,110,97,107,97,114,112, …
- Kod(karp) = (107*64^3 + 97*64^2 + 114*64^1 + 112*64^0)
- = 28 454 128

Rabin-Karp príklad

- H(karp) = kod(karp) mod 113 = 50

- T=ahojrabinakarpakosamate
- kod(ahoj) = 97*64^3+ 104*64^2 + 111* 64 + 106 =
- 25427968+425984+7104+106=25861162
- H(ahoj) =
- Kod(hojr) = (kod(ahoj)- 97*64^3)*64+106 = ...

Odkazy na animácie

- http://www.ics.uci.edu/~goodrich/dsa/11strings/demos/pattern/
- http://www-sr.informatik.uni-tuebingen.de/~buehler/BM/BM1.html
- http://www-igm.univ-mlv.fr/~lecroq/string/

Poďakovanie

Ďakujem za pozornosť