Vyhľadávanie reťazcov

Nasledujúce príklady prezentujú možnosti vyhľadávania reťazcov v texte (takto boli prezentované v dobe ich vzniku). Uvedené techniky sa ale využívajú v rôznych iných situáciách. Znak je len reprezentácia bajtu (prípadne viacerých bajtov) a teda je možné napr. vyhľadávať charakteristickú sekvenciu binárneho kódu počítačového vírusu v súboroch, hľadať či nejaký kód nebol využitý v inej aplikácií a pod.

Majme operáciu *searchstring (P, T), kde P* je vzor a *T* je text v ktorom má byť tento vzor nájdený. Nech táto operácia nájde *k* také, že *p = substring (T, k, |P|),* teda *k* je pozícia začiatku vzoru v texte.
Chceme teda v zadanom texte vyhľadať výskyt zadaného podreťazca. Najjednoduchší je priamy algoritmus.

Priamy algoritmus

```
pre všetky k od 0 po |T| – |P| urob
        i = 0
        while i < |P| and P[i] = T[k + i] do
                //porovnáme všetky znaky vzoru
               inc (i)
        ak i = |P| tak return k //našli sme zhodu, ktorá začína pozíciou k
return –1 // zhoda sa nenašla
```

P[i] – prístup k i - temu znaku reťazca P.

```
0  1  2 3  4 5  6 7  8 9 10
A B C E F G A B C D E    ..... – text T

0  1 2  3
A B C D
  A B C D
    A B C D
      A B C D
        .....
              A B C D – hľadaný podreťazec P
```

Zložitosť: O(|*P*| * |*T*|)

Akceptovateľné iba pre:
- **kratšie texty**
- **krátke vzory**

Algoritmus teda „posúva“ hľadaný podreťazec a po každom posunutí sa postupne skontrolujú všetky znaky, ak sa nenájde zhoda dôjde k ďalšiemu posunu o jeden znak.

Knuth – Morris - Pratt algoritmus

Nevýhodou použitia priameho algoritmu je, že sa posun uskutoční vždy len o jediný znak.

Príklad (nadväzuje na predchádzajúcu situáciu):
P[3] nesúhlasí s T[3] => vzor sa posunie o 1 znak
My však vieme, že na pozícií T[3] sa nachádza znak E. Keďže v hľadanom reťazci P sa znak E nenachádza môžeme konštatovať, že sa môže posun uskutočniť nie o jeden, ale o štyri miesta. Celý hľadaný reťazec sa posunie až za znak E.

Hlavná idea: Pri nesúhlase posuň o maximálny počet znakov, ktorý nevedie k preskočeniu hľadaného reťazca.

Nie vždy možný je možný taký veľký posun ako v predchádzajúcom príklade:

```
0 1 2 3 4 5 6
X Y X Y X Y c   .... T

0 1 2 3  4
X Y X Y  Z ..............  P
    X Y X Y Z
```

Po nesúhlase P[4] s T[4] nemožno presunúť o päť pozícií ale iba o dve.

Dĺžka posuvu je závislá na:

- **pozícii nesúhlasných znakov**
- **hodnote nesúhlasného znaku**

Napr. ak T[6] = X => posun o 2 znaky
ak T[6] = E => posun o 5 znakov
ak T[6] = Z => úspešný koniec – nájdená zhoda

Môžeme definovať funkciu *skip* (*P, i, c*), kde *P* je hľadaný reťazec, i je pozícia v hľadanom reťazci, kde bola nájdená nezhoda a *c* je znak, ktorý bol nájdený na porovnávanej pozícií v prehľadávanom texte, ktorá vráti hodnotu posunu.

Základ algoritmu: Ak je nesúhlas medzi *P*[i] a *c*, tak posunieme vzor o hodnotu *d = skip* (*P, i, c*) doprava.
Je dosť časovo náročné počítať hodnotu funkcie pri každej nezhode, preto sa ešte pred začatím vyhľadávania vytvorí tabuľka návratových hodnôt funkcie pre hľadaný reťazec. Prípadná nezhoda sa teda ošetrí v konštantnom čase.

Majme hľadaný vzor ABCD:

c ϵ S 0 1 2 3 <--- i

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**

Pri porovnávaní znaku vzoru s indexom 3 (indexujeme od 0, teda na tretej pozícií je „D“) sme na príslušnej pozícii v porovnávanom reťazci našli znak „B“. Podľa tabuľky sa posunieme o 4 znaky doprava.

skip(ABCD, 3, B) = 4

Pri porovnávaní znaku vzoru s indexom 2 (indexujeme od 0, teda na druhej pozícií je „C“) sme na príslušnej pozícii v porovnávanom

reťazci našli znak „Z“. Podľa tabuľky sa posunieme o 3 znaky doprava.

skip(ABCD, 2, Z) = 3
Majme hľadaný vzor XYXYZ:

c ϵ S 0 1 2 3 4 <--- i

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

Samotný algoritmus hľadania je potom len modifikovaný priamy algoritmus, pričom sa posun neuskutočňuje o jedinú pozíciu, ale vždy o počet pozícií daný funkciou skip.

Asymptotický čas hľadania reťazca: lineárny
Hľadanie: O(|*T*|)
Vybudovanie tabuľky: O(|*P*|)

Boyer - Moore algoritmus

Podobný Knuth – Morris - Pratt algoritmu, je vo všeobecnosti považovaný za rýchlejší.

Karp - Rabin algoritmus

Princíp: využíva „odtlačok prsta“ reťazca (fingerprint), ktorého výpočet je podobný hešovacej funkcii
Je rýchly, ale náročnejší na dobrú implementáciu.
Používaný pre enormne veľké texty.

Základná idea:
Nech *H* je funkcia, ktorá pre každý reťazec *w* prinesie malé číslo *H(w)* nazývané *odtlačok prsta, fingerprint, signatúra* reťazca *w*. Nedá sa vylúčiť, že dva reťazce budú mať rovnaký odtlačok (kolízia) -

snaha, aby to bolo s čo najmenšou pravdepodobnosťou (analógia s hešovaním).

Ak máme vzor *P* a text *T* a nech

f_i = *H*(substring(*T, i , |P|*)), to znamená, že f_i je odtlačok |*P*| znakov textu *T* počnúc od *i*.

Algoritmus:

- **Vypočítaj a porovnaj *H(P)* a f_0**
- **Ak *H(p)* = f_0 , tak to <u>môže</u> byť zhoda**
  - **porovnaj vzor *P* znak po znaku s prvými |*P*| znakmi textu *T***
- **Ak *H(p)* =/= f_0 tak <u>určite</u> nie je zhoda**
- **Vypočítaj a porovnaj *H(p)* a f_1 atď.**

Nároky na *H*:

- **rýchly výpočet**
- **minimalizácia kolízií**

Odporúčané algoritmy sú založené na výpočte f_i pomocou malej modifikácie f_{i-1}.
Algoritmus sa využíva napríklad v systémoch hľadajúcich podobnosť textov. Pre každé slovo sa vypočíta hodnota *H*.
Rýchlym porovnávaním je možné stanoviť mieru zhody textov.