Seminár z algoritmizácie a programovania 1

Martin Bobák
Ústav informatiky
Slovenská akadémia vied

Obsah prednášky

1. Vyhľadávanie v reťazcoch

Spätná väzba:
https://forms.gle/iKbuLdF6xDtNSEDP8

Vyhľadávanie v reťazcoch

Vstup/úloha:

- hľadať zadaný vzor (reťazec) v reťazci znakov

Výstup:

- index prvého znaku v reťazci, od ktorého sa postupnosť znakov v reťazci zhoduje s postupnosťou znakov vo vzore

Konvencia:

- **n** = dĺžka reťazca
- **m** = dĺžka (hľadaného) vzoru – tiež reťazec
- **c** = počet znakov abecedy

Motivácia

Zrejmé aplikácie:

- vyhľadávanie v textovom dokumente, na Internete
- grep
- databázové systémy

Pokročilé aplikácie:

- bioinformatika (identifikácia génov, mutácií)
- bezpečnosť (monitorovanie, detekcia spamu)
- kontrola pravopisu (v textových editoroch), plagiátorstvo

Brute force algoritmus

Od začiatku postupne prejdeme celý reťazec až po jeho koniec a porovnávame vzor znak po znaku so znakmi v reťazci.

Tomuto prístupu sa hovorí aj brute force – skúsime všetky možnosti (v tomto prípade možné pozície) výskytu vzoru v reťazci. Nevyužívame žiadnu dodatočnú informáciu, pozíciu vzoru hľadáme hrubou silou.

Časová zložitosť: **O(nm)** resp. **O(n²)**

Pamäťová zložitosť: **O(1)**

Brute force algoritmus

```
int najdi_sekv(char* vzor, int m, char* retazec, int n)
{
      for (int j=0; j < n - m + 1; j++){
            for (int i=0; i < m; i++) {
                  if(retazec[j+i] != vzor[i])
                        break;
            }
            if (i==m) {
                  return j;
            }
      }
      return -1;
}
```

Knuth-Morris-Prathov algoritmus

Triviálny algoritmus má najväčší problém v situácii, keď po dlhej zhode so vzorom príde znak, ktorý sa nezhoduje so vzorom

- nie je potrebné porovnávať ešte raz všetky znaky.
- koľko znakov môžeme preskočiť? -> dĺžka opakujúcej sa podpostupnosti -> najdlhší prefix, ktorý je zároveň aj sufix

```
retazec = "AAAAAAAAAAAAAAAAAAAAAAAAAAAB"
   vzor = "AAAB"
```

Aby sme vedeli koľko znakov môžeme preskočiť vzor si predspracujeme.

KMP – predspracovanie

- pomocné pole **lps** (veľkosť m)
- pre každú pozíciu v hľadanom vzore identifikujeme dĺžku najdlhšieho prefixu, ktorý je zároveň aj sufix (=lps[i])
  - neberieme do úvahy celý reťazec, preto vždy platí, že lps[0] = 0

```
vzor = "ABABACD"
 lsp = [0,0,1,2,?]

prefix = {A, AB, ABA, ABAB}
 sufix = {A, BA, ABA, BABA}
```

ABA je najdlhší prefix, ktorý je zárovaň aj sufixom, preto lsp[4] = 3

KMP – predspracovanie

```
void computeLPSArray(char* vzor, int m, int* lps) {
    int len = 0, i = 1;
    lps[0] = 0; // lps[0] je vzdy 0
    while(i < m) {
        if (vzor[i] == vzor[len]) {
            len++;
            lps[i] = len;
            i++;
        } else{ // (vzor[i] != vzor[len])
            if (len != 0) {
                len = lps[len - 1];
            } else { // if (len == 0)
                lps[i] = 0;
                i++; }}}}
```

KMP – algoritmus

Po vytvorení pola lps, spracujeme vstupy rovnako ako v predchádzajúcom algoritme – od začiatku postupne prejdeme celý reťazec až po jeho koniec a zarovnávame znaky vzoru so znakmi v reťazci.

Rozdiel je teda tom, že pri nezhode preskakujeme porovnania, ktoré sme overili už predtým.

Časová zložitosť: **O(n + m)** resp. **O(n)**

Pamäťová zložitosť: **O(m)**

```
int KMPSearch(char* vzor, int m, char* retazec, int n){
    //Predprocesing -> pole lps
    int lps[m];
    computeLPSArray(vzor, m, lps);

    int i = 0; // index pre retazec[]
    int j = 0; // index pre vzor[]
    while (i < n) {
        if (pat[j] == txt[i]) {
            j++;
            i++;
        }
        if (j == m) // nasli sme cely vzor
            return (i - j);

        // nezhoda po j zhodach
        if (i < n && pat[j] != txt[i]) {
            if (j != 0)
                j = lps[j - 1];
            else
                i = i + 1;}}}
```

Ukážka

"AAAAABAAABA"
"AAAA"
lps[] = {0, 1, 2, 3}

i = 0, j = 0
"**A**AAAABAAABA"
"**A**AAA"

i = 1, j = 1
"A**A**AAABAAABA"
"A**A**AA"

i = 2, j = 2
"AA**A**AABAAABA"
"AA**A**A"

i = 3, j = 3
"AAA**A**ABAAABA"
"AAA**A**"

i = 4, j = 4
Celý vzor je nájdený
j = lps[j-1] = lps[3] = 3

i = 4, j = 3
"AAAA**A**BAAABA"
"AAA**A**"

i = 5, j = 4
Celý vzor je nájdený
j = lps[j-1] = lps[3] = 3

i = 5, j = 3
"AAAAA**B**AAABA"
"AAA**A**"

i = 5, j = 2
"AAAAA**B**AAABA"
"AA**A**A"

i = 5, j = 1
"AAAAA**B**AAABA"
"A**A**AA"

i = 5, j = 0
"AAAAA**B**AAABA"
"**A**AAA"

i = 6, j = 0
"AAAAAB**A**AABA"
"**A**AAA"

i = 7, j = 1
"AAAAABA**A**ABA"
"A**A**AA"

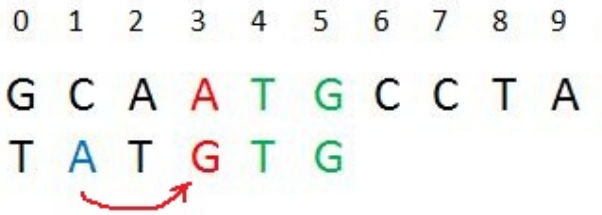
...

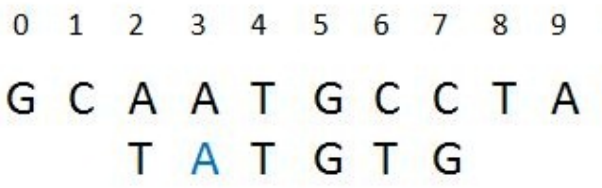
Boyer-Mooreov algoritmus

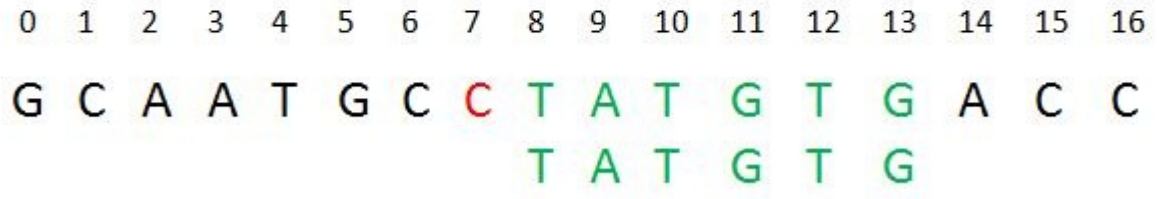
- znaky vzoru sa porovnávajú so znakmi reťazca sprava
- vzor posúvame na základe dvoch pravidiel:
  - pravidlo zlého znaku
  - pravidlo dobrého sufixu

Pravidlo zlého znaku

- zlý znak = znak, ktorý spôsobí nezhodu
- posúvame vzor, kým:
  - nenastane zhoda
  - prešli sme celý vzor



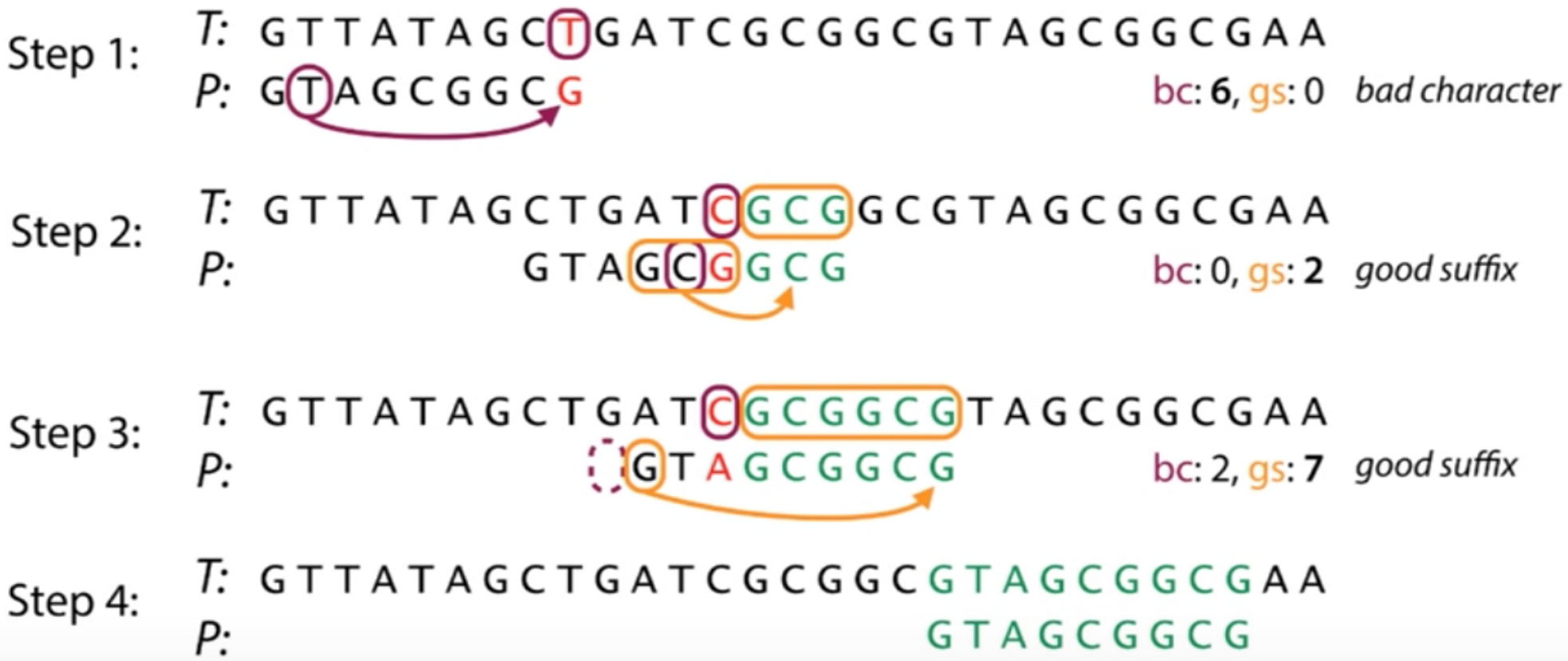




Pravidlo dobrého sufixu

- snažíme sa zachovať časť vzoru, ktorá bola úspešne porovnaná

Boyer-Mooreov algoritmus



Námety na semestrálnu prácu

- palindrómy
- hľadanie viacerých vzoriek (Aho-Corasick)
- Rabin-Karpov algoritmus
- Lempel–Ziv–Welch kompresia
- Vyhľadávanie regulárnych výrazov (Thompson)
- lexikografické stromy (trie), sufixové stromy (Ukkonen)
- Sufixové polia
- Editačná vzdialenosť (Hirschberg)
- Najdlhšia spoločná podpostupnosť (Hunt-Szymanski)

Zdroje

Brute force algoritmus

http://www2.fiit.stuba.sk/~pospichal/soltis/kapitola5.htm

Knuth-Morris-Prathov algoritmus

https://www.geeksforgeeks.org/kmp-algorithm-for-pattern-searching/

Bover-Mooreov algoritmus

1. https://www.youtube.com/watch?v=4Xyhb72LCX4
2. https://www.youtube.com/watch?v=Wj606N0IAsw

Ďakujem vám za pozornosť!

Spätná väzba:
https://forms.gle/iKbuLdF6xDtNSEDP8