Meklēšana II

1. Bojera-Mūra algoritms

Mums ir teksts T[0]...T[n-1] un apakšvirkne P[0]...P[m-1], kas jāatrod tekstā T. Gan iepriekšējā lekcijā apskatītie algoritmi (naivais un Knuta-Morisa-Prata algoritms), gan Bojera-Mūra algoritms to dara, salīdzinot P[0]...P[m-1] ar tāda paša garuma teksta apakšvirkni (piemēram, T[i]...T[i-m-1]).

Naivais un KMP algoritmi to dara, sākot ar pirmajiem simboliem (P[0] un T[i], tad sakritības gadījumā P[1] un T[i+1], utt.). Gadījumā, ja atšķiras pirmais simbols (P[0]≠T[i]), tad salīdzināmo apakšvirkni pabīda vienu simbolu tālāk (T[i]...T[i-m-1] vietā mēģina T[i+1]...T[i-m]).

Bojera-Mūra algoritms sāk salīdzināšanu no pēdējā simbola (P[m-1] un T[i+m-1]). Šīs pieejas priekšrocība ir tāda, ka nesakritības gadījumā salīdzināmo virkni var būt iespējams pabīdīt uz priekšu par vairāk nekā 1 simbolu. (Piemēram, ja T[i+m-1] ir simbols, kas nesakrīt ne ar vienu no P[0]...P[m-1] simboliem, tad mēs varam pārbīdīt virkni par m simboliem uz priekšu.)

Tāpēc naivajam algoritmam un KMP vienmēr nepieciešamas n operācijas, bet Bojera-Mūra algoritmam dažos gadījumos var pietikt ar aptuveni n/m operācijām. (Sliktākajos gadījumos gan Bojera-Mūra algoritms nav labāks par KMP.)

Algoritma apraksts:

```
n = garums (T);
m = garums (P);
s = 0;
while (s ✹ n-m) {
        j = m;
        while ((j>0) and (P[j-1] = T[s+j-1])) do
                j = j-1;
        if (j = 0) then {
                print („Apakšvirkne atrasta sākot ar burtu Nr.”, s);
                s = s+ ✚ [0];
        } else
                s = s + max(✚ [j], j-1-🍃[T[s+j-1]]);
}
```

Algoritmā tiek izmantoti divi masīvi (✚ un 🍃) kas apraksta, cik lielas pārbīdes iespējamas. Šie masīvi un to konstruēšana aplūkoti nākošajā nodaļā.

2. Pārbīžu masīvi

Līdzīgi kā KMP algoritmā, mums ir divi pārbīžu masīvi: sliktā simbola tabula 🍃 un labā sufiksa tabula ✚.

Sliktā simbola tabulu indeksē ar simboliem, kas var būt sastopami P. Priekš katra simbola x tajā tiek ierakstīts lielākais i, kur P[i]=x. Ja x nav sastopams vārdā, tad 🍃[x]=-1. Piemēram, ja P=abcab, tad tabula izskatās šādi:

x	🍃[x]
a	3
b	4
c	2
*	-1

Ar * šeit apzīmēts jebkurš cits simbols.

<u>Pseidokods priekš tabulas izveidošanas:</u>
Katram simbolam a: 🍃[a]=-1;
For j=0 to m-1 do 🍃[P[j]]=j.

Labā prefiksa tabulu ✚[i] indeksē ar skaitļiem i no 0 līdz m. Šīs tabulas semantika ir šāda: ja P[i]...P[m-1] sakrīt ar T[k+i]...T[k+m-1], bet P[i-1] ᅪ T[k+i-1], tad ✚[i] ir mazākā pārbīde j, kuru ir vērts mēģināt.

Ja i=m, tad ✚[i]=1.

Ja i✹m, tad ✚[i] ir vienāds ar mazāko j>0, kam piemīt viena no šīm divām īpašībām:

- i ᅎ j un P[i]...P[m-1] sakrīt ar P[i-j]...P[m-j-1];
- i ✹ j un P[j]...P[m-1] sakrīt ar P[0]...P[m-j-1].

Ja tāds j neeksistē, tad ✚[i] = m;

Piemēram, vārdam T = abcab šī tabula izskatīsies šādi:

j	✚[j]
5	1
4	3
3	3
2	3
1	3
0	3

Saturiski, tas nozīmē, ka ja nav sakritis pēdējais simbols, tad nākošā iespēja, kas jāmēģina ir T[i+1]...T[i+5], bet, ja nav sakritis kāds no iepriekšējiem simboliem, tad varam uzreiz pāriet uz T[i+3]...T[i+7]. (Šajā konkrētajā gadījumā to mums garantē simbols P[4]=T[i+4]=b, jo b vārdā P ir tikai P[4] un P[1].)

Tabulu var izrēķināt vairākos veidos:

1. **Naivais algoritms.** Katram iespējamajam i var pārbaudīt visus iespējamos j, līdz atrasts mazākais j, kas der. Šāds algoritms der maziem vārda garumiem m (tādiem kā mājas darbos sastopamie), bet būs lēns lielākiem m, jo tā darbības laiks var būt $O(m^3)$.
2. **Labāks algoritms.** Tabulu var izrēķināt laikā O(m), ar šādu algoritmu.

Pseidokods priekš tabulas izveidošanas:

```
👁 = compute-prefix-function (P);
P’ = reverse (P);
👁’ = compute-prefix-function (P’);
for j = 0 to m do
        ✚[j] = m - 👁[m];
for l = 1 to m do
{
        j = m - 👁’[l];
        ✚[j] = max (✚[j], l -👁’[l]);
}
```

Šajā algoritmā, „compute-prefix-function” ir funkcija, kas saņem virkni P, izrēķina šai virknei atbilstošo Knuta-Morisa Prata algoritma (lekcija „Meklēšana I”) prefiksu masīvu k[j] un atgriež šo masīvu.

3. Piemērs

Apskatīsim T = daababcabaab un P = abcab. Tabulas priekš šī P mēs jau esam uzkonstruējuši. Meklēšanu var vizualizēt šādi:

```
      a d a a b a b c a b a a b
1.       x a b
2.               x
3.           a b c a b
4.                   x a b
```

Katra rindiņa attēlo vienu mēģinājumu atrast sakrītošu apakšvirkni un x apzīmē burtu, kurā simbols no P nav sakritis ar atbilstošo simbolu no T.

Pirmajā mēģinājumā sakrīt P[4] un P[3], bet ne P[2]. Tad mēs pavirzāmies 3 simbolus uz priekšu (saskaņā ar „labā sufiksa tabulu”) un skatāmies vai P[0]...P[4] sakrīt ar T[3]...T[7]. Nesakritība ir jau P[4] un tad saskaņā ar „sliktā simbola tabulu” pavirzāmies pa 2 simboliem, lai atrastais c sakristo ar pirmo iespējamo c, kas ir apakšvirknē P (pirmo – no beigām).

Trešajā mēģinājumā mums sakrīt visa apakšvirkne. Ja nepieciešams atrast visas vietas tekstā T, kur ir apakšvirkne P, tad saskaņā ar ✚[0] pārvietojamies 3 simbolus uz priekšu un mēģinām vēl. (Šis mēģinājums ir neveiksmīgs un tad mēs esam sasnieguši teksta beigas.)