Algorytmy i złożoność obliczeniowa

Laboratorium 13. Algorytmy tekstowe.

1. Algorytmy tekstowe

Algorytmy tekstowe mają decydujące znaczenie przy wyszukiwaniu informacji typu tekstowego, ten typ informacji jest szczególnie popularny w informatyce, np. w edytorach tekstowych i wyszukiwarkach internetowych. Tekst jest ciągiem symboli. Przyjmujemy, że jest on zadany tablicą $x[1,\ldots,n]$, elementami której są symbole ze skończonego zbioru *A* (zwanego alfabetem). Liczba $n = |x|$ jest długością (rozmiarem) tekstu. Typowe reprezentacje tekstów to reprezentacja listowa i reprezentacja tablicowa. Podstawowym problemem dotyczącym tekstów jest problem wyszukiwania wzorca. Problem ten polega na znalezieniu wszystkich wystąpień tekstu *x*, zwanego wzorcem w tekście *y*. Przyjmujemy, że $|x| = m$, $|y| = n$, $n \geq m$.

Wyszukiwanie wzorca było wszechstronnie badane ze względu na duże zastosowanie praktyczne. Poniżej zostaną zaprezentowane podstawowe algorytmy dla problemu wyszukiwania wzorca takie jak : algorytm N (algorytm „naiwny"), Algorytm KMP (Knutha-Morrisa-Pratta), Algorytm GS' (wersja algorytmu Galila-Seiferasa dla pewnej klasy wzorców)- więcej algorytmów zostało omówione na wykładzie http://www.ipipan.waw.pl/~penczek .

2. Problem wyszukiwania wzorca

Niech x, będzie wzorcem, a y tekstem, w którym szukamy wzorca. Dla słowa z przyjmijmy $z[i..j] = z[i]z[i+1]\ldots z[j]$, gdzie $i \leq j$. Mówimy, że x występuje w y na pozycji i , gdy $y[i..i+m-1] = x$ - inaczej mówiąc, gdy począwszy od i-tej pozycji, wzorzec „pasuje" do tekstu. Z tego powodu problem wyszukiwania wzorca jest również nazywany problemem dopasowywania wzorca.

Przykład

-alfabet: {a, b}

-tekst: abbbaabaabba

-wzorzec: aaba

wystąpienia:

1 abbbaabaabaa

2 abbbaabaaba

2.1. Algorytm N(„naiwny")

Schemat najbardziej bezpośredniego algorytmu, zwanego „naiwnym", wygląda następująco:

```
begin
    i := 1;
    while i <= n - m + 1 do
    begin
        if x[1..m] = y[i..i + m -1] then write(i);
        i := i +1; (przesunięcie =1)
    end;
end;
```

Pełny algorytm otrzymamy, rozpisując instrukcję sprawdzenia równości tekstów.

```
begin
    i := 1;
while i <= n - m + 1 do
    begin
        j := 0; while x[j+1] = y[i+j] do  j := j + 1;
        if j = m then write(i); i := i +1; {przesunięcie =1}
    end;
end;
```

Przykład

teskt=bbabbbaabb, n=10

wzorzec=aab, m=3

-i=1 nie nastąpi wejście do zagnieżdżonej pętli while, nie jest również spełniony warunek if.
- i=2 nie nastąpi wejście do zagnieżdżonej pętli while, nie jest również spełniony warunek if.
- i=3 nastąpi wejście do zagnieżdżonej pętli while, ale dla j=1 nastąpi wyjście, po tym wyjściu warunek if będzie niespełniony,
- i=4..6 nie nastąpi wejście do zagnieżdżonej pętli while, nie jest również spełniony warunekif.
- i=7 nastąpi wejście do zagnieżdżonej pętli while, która wykonywana będzie aż do j=3, po tym wyjściu warunek if jest spełniony tzn. znaleziono dopasowanie.
- i=8 nie nastąpi wejście do zagnieżdżonej pętli while, nie jest również spełniony warunek if.
- i=9 nie jest spełniony główny warunek, algorytm zostaje zakończony.

2.2. Algorytm KMP

W algorytmie N początek wzorca przesuwamy zawsze o jeden, podczas gdy możliwe jest czasami dużo większe przesunięcie. Taką wartość przesunięcia można obliczyć, korzystając z informacji o maksymalnej wartości j, obliczonej w poprzednim kroku algorytmu.P[j] = max { 0 <= k < j | x[1..k] jest sufiksem x[1..j]} Przyjmujemy P[0] = P[1] = 0. Jeśli tablica *P* jest już policzona, to problem wyszukiwania wzorca można efektywnie rozwiązać za pomocą algorytmu KMP, którego struktura jest podobna do struktury algorytmu N. Wartość przesunięcia początku wzorca w *y* jest określona teraz wzorem *przesunięcie(j) = max(1,j – P[j]).*

```
begin
    i := 1; j := 0;
while i <= n - m + 1 do
    {x[1..P[j]] = x[j-P[j]+1..j] = y[i..i+P[j]-1]}
    begin
        j := P[j];
   while x[j+1] = y[i+j] do  j := j + 1;
        if j = m then write(i);
        i := i + przesunięcie(j); {przesunięcie >= 1}
    end
end;
```

Pozostaje jeszcze problem obliczenia tablicy P. Dla każdego $j \geq 2$ obliczamy $P[j]$, korzystając z następującej obserwacji: niech $i \geq 1$ będzie minimalne, takie że $x[P^{(i)}[j-1]+1] = x[j]$. Jeśli takie i istnieje, to $P[j] = P^{(i)}[j-1]+1$; w przeciwnym razie P[j] = 0.

```
begin
   P[0] := P[1] := 0;
   t :=0;
for j := 2 to m do
   begin {obliczamy wartość P[j]}
       {t = P[j - 1]}
       while (t > 0) and (x[t + 1] ≠ x[j]) do t := P[t];
       if x[t + 1] = x[j] then t := t + 1;
       P[j] := t;
   end;
end;
```

Przykład wypełniania tablicy P, wzorzec=**abbabcabb,** obliczmy teraz dla takiego wzorca tablicę *P*

Przyjmujemy *P[0]=0* oraz *P[1]=0*, następnie:

P[2]=0, P[3]=0,

P[4]=1, gdyż a (początek wzorca) jest sufiksem abb**a** (wzorzec[1..4])

P[5]=2, gdyż ab (początek wzorca) jest sufiksem abb**ab** (wzorzec[1..5])

P[6]=0,

P[7]=1, gdyż a (początek wzorca) jest sufiksem abbabc**a** (wzorzec[1..7])

P[8]=2, gdyż ab (początek wzorca)jest sufiksem abbabc**ab** (wzorzec[1..8])

P[9]=3, gdyż abb (początek wzorca) jest sufiksem abbabc**abb** (wzorzec[1..9])

2.3. Algorytm GS' (wersja algorytmu Galila-Seiferasa dla pewnej klasy wzorców)

Jednym z ciekawszych algorytmów wyszukiwania wzorca jest algorytm GS, który umożliwia rozwiązanie problemu w czasie liniowym i jednocześnie w pamięci. Mówimy, że wzorzec jest łatwy gdy żadne słowo niepuste postaci vvv nie jest jego prefiksem (na przykład x = abababba nie jest łatwy a wzorzec x = abbabbbab jest łatwy). W przypadku łatwych wzorców można skonstruować algorytm podobny do KMP, w którym tablicę P zastępuje się przez pewne przybliżone oszacowanie:

P[j] < 2j/3, przesunięcie(j) >= j/3.

```
{Algorytm GS': wersja algorytmu Galila-Seiferasa
dla łatwych wzorców}
begin
    i := 1;
while i <= n - m + 1 do
   begin
        j := 0;
while x[j+1] = y[i+j +1] do  j := j + 1;
        if j = m then write(i);
        i := i + max(1, [j/3]); {przesunięcie >= 1}
    end;
end;
```

Algorytm ten nie będzie działał prawidłowo dla wzorców, które nie są łatwe.

Przykład

Dla x = aaaaaaab i y = aaaaaaaab. Zostanie wypisane 0 wystąpień wzorca x w tekście y. Podczas gdy x występuje w y począwszy od pozycji i=2. Dla j = 7 otrzymamy przesunięcie równe 2 i następną badaną pozycją w y będzie i=i+2=3, a więc pozycja i = 2 zostanie pominięta.

2.4 Algorytm Boyera-Moore'a (BM), uproszczony

Algorytm Boyera-Moore'a rozpoczyna porównywanie od ostatniego znaku wzorca, czyli odwrotnie niż opisane poprzednio algorytmy. Dzięki temu jeśli ostatni znak wzorca nie zgadza się ze znakiem w przeszukiwanym tekście i dodatkowo wiemy, iż znak z przeszukiwanego tekstu nie występuje dalej we wzorcu, to okno wzorca możemy od razu przesunąć o tyle pozycji, ile znaków zawiera wzorzec. W przeciwnym razie wzorzec pozycjonujemy tak, aby zgrać pozycje znaku występującego jednocześnie w przeszukiwanym tekście i we wzorcu. Algorytmy naiwny i KMP nie pomijają w ten sposób znaków w przeszukiwanym tekście. Dzięki temu algorytm BM zwykle dużo szybciej znajduje rozwiązanie.

Przykład: Wyszukujemy wzorzec ABC w tekście ABDAABCA

A B D A A B C A A B C	Okno wzorca umieszczamy na początku tekstu. Rozpoczynamy porównanie od ostatniego znaku wzorca i stwierdzamy niezgodność. Ponadto znaku D nie ma we wzorcu.
A B D A A B C A A B C	Przesuwamy okno o całą długość wzorca. Znowu jest niezgodność, ale tym razem znak B występuje we wzorcu.
A B D A A B C A A B C	Przesuwamy okno tak, aby znak B z tekstu pokrywał się z ostatnim wystąpieniem znaku B we wzorcu
A B D A A B C A A B C	Znaleziono wzorzec!

Do wykonywania przesunięcia okna wzorca względem przeszukiwanego tekstu wykorzystamy tablicę Last. W tablicy tej indeksy poszczególnych elementów odpowiadają kodom znaków alfabetu. Elementy odpowiadają pozycji ostatniego wystąpienia danego znaku we wzorcu (licząc od 0). Jeśli litery nie ma we wzorcu, to reprezentujący ją element w tablicy Last ma wartość -1.

Na przykład, jeśli alfabet składa się z czterech znaków ABCD, to tablica Last będzie miała również 4 elementy. Niech A będzie kodowane przez 0, B przez 1, C przez 2, a D przez 3. Wtedy, dla wzorca ABC tablica Last wygląda następująco: [0,1,2,-1], czyli dla A mamy 0, dla B 1, dla C 2, i dla D mamy -1.

W uproszczonej wersji algorytmu BM stosujemy przesunięcie równe **`max(1, j - Last[X])`**, gdzie `j` to pozycja we wzorcu, na której wystąpiła niezgodność.

W podanym powyżej przykładzie na początku mamy przesunięcie o 3, ponieważ max(1, 2 – (-1)) = max(1,3) = 3, a w drugim przypadku przesuwamy okno o 1, ponieważ max(1, 2 – 2) = max(1,0) = 1.

Zadania

1. Napisz program, który sprawdzi czy podany wzorzec jest łatwy. **[3p]**
2. Dany jest tekst y oraz wzorzec x. Napisz program, który wypisze ile razy wzorzec wystąpił w tekście i na jakich pozycjach. Wykorzystaj algorytm:

   a) GS' **[1p]**

   b) KMP **[3p]**

   c) BM **[3p]**