

Łańcuchy znaków

Obiekt klasy string vs tablica znaków

<code>#include <iostream></code>	<code>#include <cstdio></code>
<pre>char t[2000]; cin >> t; //lub jeśli z białymi znakami cin.get(t,2000); cout << t << endl;</pre>	<pre>char t[2000]; scanf("%s",t); //lub jeśli z białymi znakami fgets(t, 2000, stdin); printf("%s\n",t);</pre>

Podstawowe pojęcia

Alfabet - skończony zbiór symboli

Łańcuchy / teksty - skończone ciągi symboli alfabetu

Pusty łańcuch - łańcuch nie zawierający ani jednego znaku.

$s[i]$ – oznacza i -ty znak łańcucha s .

$|s|$ – oznacza długość łańcucha, czyli liczbę przechowywanych w nim aktualnie znaków. Łańcuch pusty ma długość 0.

$s[i : j]$ – oznacza fragment łańcucha (ang substring) zawierający kolejne znaki $s[i]$ $s[i + 1]$ $s[i + 2]$... $s[j - 1]$. Znak $s[j]$ nie należy do tej sekwencji.

Długość podłańcucha wyliczamy ze wzoru:

$$|s[i : j]| = j - i$$

Podłańcuch $s[i : i]$ jest łańcuchem pustym – posiada długość 0

$s = t$ – równość dwóch łańcuchów oznacza, że są one tej samej długości oraz posiadają identyczne znaki na tych samych pozycjach.

Prefiks łańcucha s jest łańcuchem zbudowanym z k początkowych znaków:

$$\text{Pref}(s) = s[0 : k]$$

Sufiks łańcucha s jest łańcuch zbudowany z k końcowych znaków:

$$\text{Suff}(s) = s[n - k : n]$$

Mówimy, że prefiks lub sufiks jest właściwy, jeśli nie obejmuje całego łańcucha s .

Jeśli istnieje prefiks s , który jest równy sufiksowi s , to mówimy, iż tworzą one tzw. prefikso-sufiks (ang. border)

Jeśli dany łańcuch s posiada prefikso-sufiks o długości k , to okresem nazywamy taką liczbę całkowitą d , że zachodzi warunek:

$$s[0 : k] = s[d : n]$$

Sposób działania algorytmu:

Tablica P - przechowuje długości maksymalnych prefikso-sufiksów dla kolejnych prefiksów łańcucha s. Indeks w tablicy P oznacza długość prefiksu łańcucha s, natomiast element o tym indeksie ma wartość długości maksymalnego prefikso-sufiksu w danym prefiksie. Na przykład, jeśli $P[6] = 2$, to prefiks 6-cio znakowy łańcucha s posiada prefikso-sufiks maksymalny o długości dwóch znaków.

1. Początkowo w zerowym elemencie tablicy umieszczamy wartownika (minus jeden)

A B A A B A C A B A A B A B

Tablica
P -1

2. Prefiks jednoznakowy A posiada prefikso-sufiks pusty, ponieważ prefikso-sufiks musi się składać z prefiksu i sufiksu właściwego.

A B A A B A C A B A A B A B

Tablica
P -1 0

3. Prefiks dwuznakowy AB posiada prefikso-sufiks pusty

A B A A B A C A B A A B A B

Tablica
P -1 0 0

4. Prefiks trzyznakowy ABA posiada prefikso-sufiks o długości jeden

A B A A B A C A B A A B A B

Tablica
P -1 0 0 1

5. Sprawdzamy czy da się rozszerzyć poprzedni prefikso-sufiks o jeszcze jeden znak (porównując żółte znaki)

A B A A B A C A B A A B A B

Tablica
P -1 0 0 1 1

Postępujemy w dalszym ciągu tak samo, a gdy napotkamy na brak możliwości rozszerzenia prefikso-sufiksu, próbujemy sprawdzić czy da się rozszerzyć zredukowany prefikso- sufiks (poprzedniego sufiksu)

Widać to na końcowym etapie działania tego algorytmu:

A B A A B A C A B A A B A B

Tablica P	-1	0	0	1	1	2	3	0	1	2	3	4	5	6
-----------	----	---	---	---	---	---	---	---	---	---	---	---	---	---

Prefikso-sufiksu ABAABA nie da się rozszerzyć o jeden znak, należy więc sprawdzić czy prefikso-sufiks tego prefiksu (to jest ABA) jest rozszerzalny

A B A A B A C A B A A B A B

Tablica P	-1	0	0	1	1	2	3	0	1	2	3	4	5	6
														^

Długość prefiksu wynosi 6 (jest ona zapisana w tablicy P).

A B A A B A C A B A A B A B

Tablica P	-1	0	0	1	1	2	3	0	1	2	3	4	5	6
														^

Interesuje nas więc szósty element tablicy P. P[6] wynosi 3. Oznacza to długość maksymalnego prefikso-sufiksu o długości 6 (tym prefikso-sufiksem jest ABA). Lecz ABA także nie jest rozszerzalna (patrz żółte litery). Sprawdzamy więc czy rozszerzalny jest prefikso-sufiks tego trzyznakowego prefiksu (jego długość wynosi jeden – jest zapisana w tablicy w P[3])

A B A A B A C A B A A B A B

Tablica P	-1	0	0	1	1	2	3	0	1	2	3	4	5	6
-----------	----	---	---	---	---	---	---	---	---	---	---	---	---	---

Ten prefiks jest rozszerzalny. P[14] wyniesie więc dwa.

A B A A B A C A B A A B A B

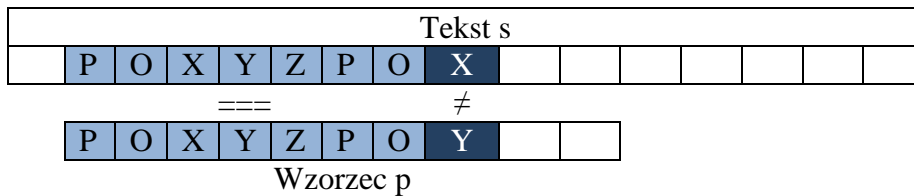
Tablica P	-1	0	0	1	1	2	3	0	1	2	3	4	5	6	2
-----------	----	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Kod programu:

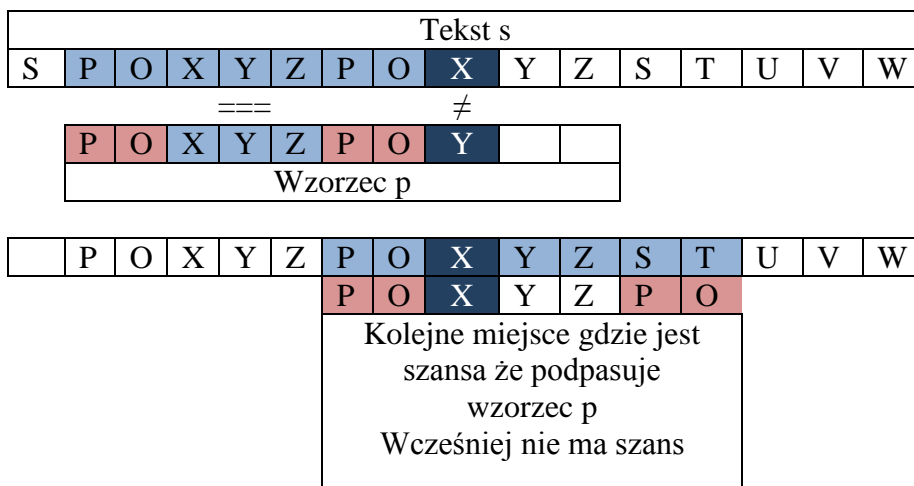
```
int pi[1000000];
string a;
int main()
{
    pi[0] = -1;
    cin>>a;
    a = "#"+a;
    int dl=a.size();
    for(int i=1;i<=dl;i++)
    {
        int x = pi[i-1];
        while(x!=-1)
        {
            if(a[x+1] == a[i])
            {
                pi[i] = x+1;
                break;
            }
            x = pi[x];
        }
    }
    for(int i=0;i<dl;i++)
        cout<<pi[i]<<" ";
    return 0;
}
```

Jak wykorzystamy ten algorytm do poszukiwania wzorca w tekście?

Załóżmy, iż poszukujemy pierwszego wystąpienia wzorca p w danym łańcuchu tekstowym s . W trakcie porównywania kolejnych znaków łańcucha ze znakami wzorca natrafiliśmy na sytuację, w której pewien prefiks wzorca p , pasuje do prefiksu okna w łańcuchu s , jednakże znak s oznaczony na rysunku symbolem X różni się od znaku oznaczonego symbolem Y, który znajduje się we wzorcu tuż za pasującym prefiksem:



Algorytm naiwny w takiej sytuacji przesuwając okno wzorca o jedną pozycję w prawo względem przeszukiwanego tekstu i rozpoczyna od początku porównywanie znaków wzorca p ze znakami okna nie korzystając zupełnie z faktu zgodności części znaków. Tymczasem wykorzystując fakt, że w naszym pasującym prefiksie istnieje jakiś najdłuższy prefikso-sufiks, możemy pominąć pewne porównania znaków bez żadnej szkody dla wyniku poszukiwań. Następne miejsce gdzie możemy podłożyć okno wzorca zaczyna się w naszym tekście tam, gdzie zaczyna się ten maksymalny prefikso-sufiks:



Dla każdego prefiksu wzorca szerokość maksymalnego prefikso-sufiksu można wyznaczyć przed rozpoczęciem wyszukiwania – do tego właśnie celu wykorzystamy algorytm *omówiony wcześniej*. Dla danej długości prefiksu b możemy odczytać szerokość maksymalnego prefikso-sufiksu tego prefiksu. W naszym przypadku będzie to:

$$bb = \Pi [b]$$

Teraz porównujemy znak wzorca $p [bb]$ (oznaczony na rysunku symbolem C) ze znakiem $s [i]$ (symbol A). Jeśli wciąż mamy niezgodność, to procedurę powtarzamy, aż do wyczerpania się prefikso-sufiksów – w takim przypadku okno wzorca oraz indeks i przesuwamy o jedną pozycję w prawo.

Jeśli otrzymamy zgodność, to pasujący prefiks zwiększa swoją długość o 1 znak. Przesuwamy również indeks i o 1, ponieważ znak na tej pozycji został już całkowicie wykorzystany przez algorytm MP.

```

// dla wzorca obliczamy tablicę PI [ ]

PI [ 0 ] = b = -1;
for( i = 1; i <= M; i++ )
{
    while( ( b > -1 ) && ( p [ b ] != p [ i - 1 ] ) ) b = PI [ b ];
    PI [ i ] = ++b;
}

// wypisujemy wzorzec

cout << p << endl;

// wypisujemy łańcuch s

cout << s;

// poszukujemy pozycji wzorca w łańcuchu

pp = b = 0;
for( i = 0; i < N; i++ )
{
    while( ( b > -1 ) && ( p [ b ] != s [ i ] ) ) b = PI [ b ];
    if( ++b == M )
    {
        while( pp < i - b + 1 )
        {
            cout << " "; pp++;
        }
        cout << "^"; pp++;
        b = PI [ b ];
    }
}
cout << endl;

```