Algorytm Manachera

Oznaczmy:

w	niepusty łańcuch znaków.
p_P	palindrom parzysty postaci $p_P = ww^R$
p_N	palindrom nieparzysty postaci $p_N = wXw^R$
r_p	promień palindromu - długość podsłowa w
i_s	Środek palindromu – pozycja pierwszego znaku za słowem w

Algorytm Manachera wyznacza maksymalne palindromy, których środki występują na kolejnych pozycjach znakowych przeszukiwanego łańcucha *s*. Dzięki takiemu podejściu redukujemy złożoność obliczeniową fazy przeszukiwania łańcucha s. Mając maksymalny palindrom możemy bez problemów wyznaczyć zawarte w nim podpalindromy. Wykorzystujemy tutaj własność symetrii palindromu:

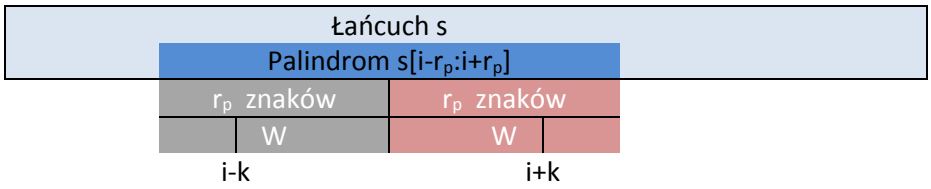
Dla danego łańcucha *s* algorytm Manachera tworzy tablicę dwuwymiarową R:

R [0, ...] – promienie palindromów parzystych
R [1, ...] – promienie palindromów nieparzystych

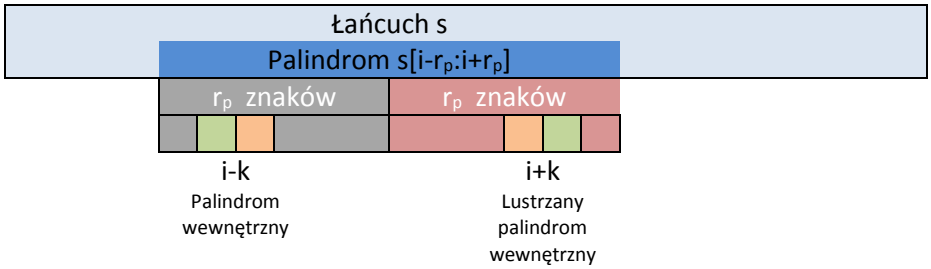
Indeksy tych tablic określają kolejne pozycje znakowe w łańcuchu *s*, natomiast wartości zawierają maksymalne promienie palindromów o środkach na danej pozycji.

Używając w odpowiedni sposób tablicy R oraz własności symetrii palindromu, algorytm Manachera wykorzystuje sprytnie informację o wcześniej wyznaczonych promieniach palindromów maksymalnych do wyszukiwania następnych palindromów. Otóż po wyznaczeniu promienia r_p palindromu na pozycji *i*-tej w łańcuchu *s*, sprawdzane są promienie palindromów na kolejnych pozycjach poprzedzających pozycję *i*-tą w obszarze podsłowa *w* – tutaj algorytm wymaga dwóch wersji – osobnej dla palindromów parzystych i osobnej dla nieparzystych. Zasada jest identyczna dla obu wersji. Rozważmy zatem możliwe przypadki (dla palindromu parzystego):

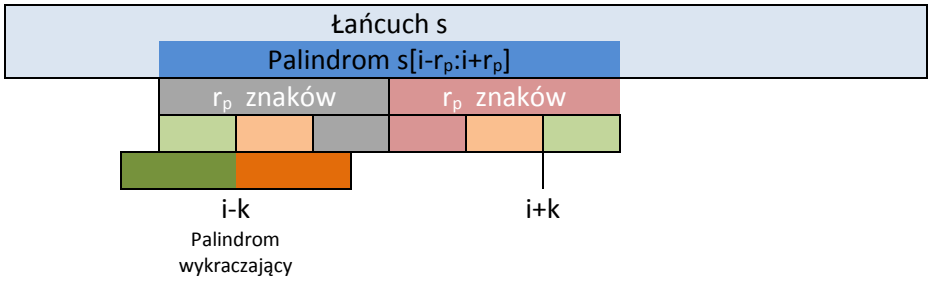
a) Pozycja i-k, dla k=1,2,..,r_p **nie jest środkiem** żadnego palindromu, skoro tak, to przez symetrię możemy stwierdzić, że na pozycji lustrzanej i+k również nie będzie żadnego palindromu. Pozycja i+k może zostać pominięta przy dalszym wyszukiwaniu palindromów.



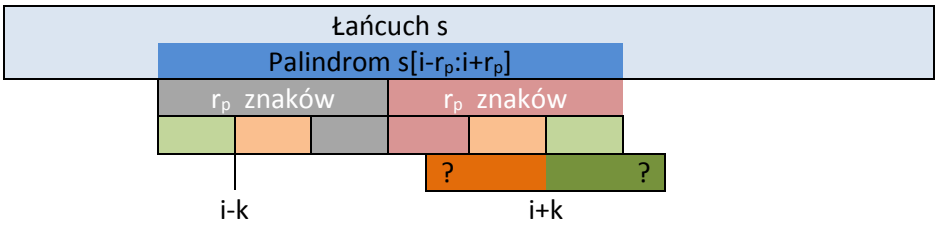
b) Na pozycji i - k jest palindrom o promieniu r < r_p - k (w całości zawiera się wewnątrz rozważanego palindromu i nie styka się z jego brzegiem). Poprzez symetrię wnioskujemy, iż na pozycji i + k również musi występować taki sam palindrom, którego już dalej nie da się rozszerzyć. Pozycji i + k nie musimy już dalej sprawdzać.



c) Na pozycji i - k jest palindrom o promieniu r > r_p - k (wykracza z lewej strony poza obszar rozważanego palindromu). Na pozycji i + k znajduje się palindrom o promieniu r = r_p - k i palindromu tego nie da się już rozszerzyć. Wyjaśnienie tego faktu jest bardzo proste – gdyby palindrom na pozycji i + k posiadał większy promień niż wyliczone r, to również z uwagi na symetrię przeglądany palindrom posiadałby promień większy od r_p, a przecież jest to palindrom maksymalny. Pozycję i+k również możemy pominąć.



d) na pozycji i - k występuje palindrom o promieniu r = r_p - k. Taki sam palindrom musi być na pozycji i + k, jednakże w tym przypadku palindrom ten może być rozszerzalny. Pozycję i + k musimy zatem sprawdzić na obecność palindromu o promieniu większym od r.



Wyszukiwanie wzorca w tekście

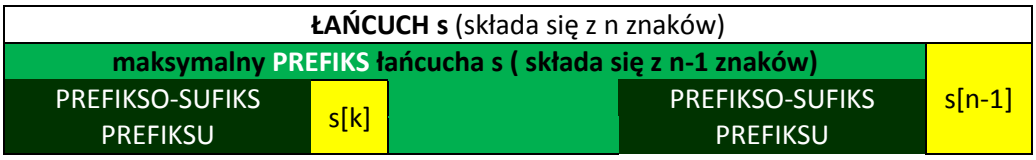
Jednym z podstawowych problemów związanych z przetwarzaniem tekstów jest poszukiwanie wzorca w tekście. Rozwiązaniem jest podanie w ciągu s wszystkich pozycji i takich, że zachodzi równość:

s [i : i + | p |] = p, czyli że wzorzec p jest fragmentem łańcucha s występującym na pozycji i-tej.

Rozwiązanie 1. Algorytm naiwny ustawia okno o długości wzorca na pierwszej pozycji w łańcuchu s. Następnie sprawdza, czy zawartość tego okna jest równa wzorcowi. Jeśli tak, pozycja okna jest zwracana jako wynik, po czym okno przesuwa się o jedną pozycję w prawo i cała procedura powtarza się. Klasa pesymistycznej złożoności obliczeniowej algorytmu N jest równa O (n × m), gdzie n oznacza liczbę znaków tekstu, a m liczbę znaków wzorca.

Rozwiązanie 2. Oparte na maksymalnym prefikso-sufiksie. Algorytm dynamiczny wyznaczania najdłuższego prefikso-sufiksu w czasie liniowym opiera się na dwóch poniższych własnościach:

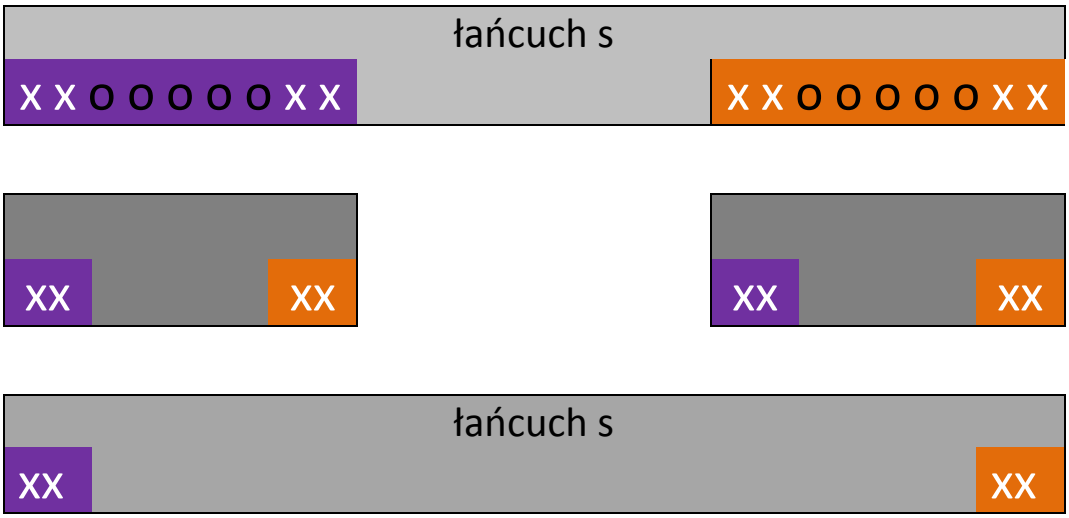
I. Rozszerzalność prefikso-sufiksu



Prefikso-sufiks prefiksu jest rozszerzalny do prefikso-sufiksu całego łańcucha s, jeśli znak s[k] (czyli znak leżący tuż za prefiksem prefiksu) jest równy znakowi s[n-1] (czyli znakowi leżącemu tuż za sufiksem prefiksu). Wtedy długość prefikso-sufiksu zwiększa się do k + 1 i staje się on prefikso-sufiksem łańcucha s.

II. Redukcja prefikso-sufiksu

Jeśli łańcuch tekstowy s posiada prefikso-sufiks, a ten prefikso-sufiks również posiada swój własny prefikso-sufiks, to jest on także prefikso-sufiksem łańcucha s



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)

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

3. Prefiks dwuznakowy AB posiada prefikso-sufiks pusty

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

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

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.

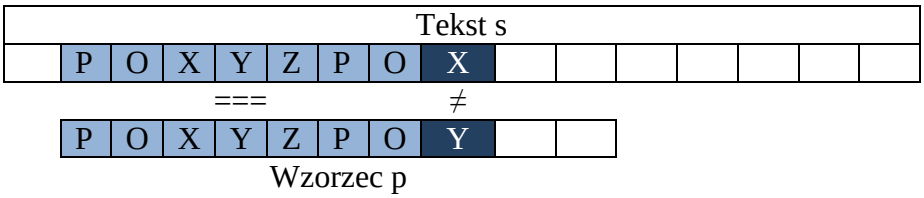
		A	B	A	A	B	A	C	A	B	A	A	B	A	B
Tablica P	-1	0	0	1	1	2	3	0	1	2	3	4	5	6	2

Kod programu:

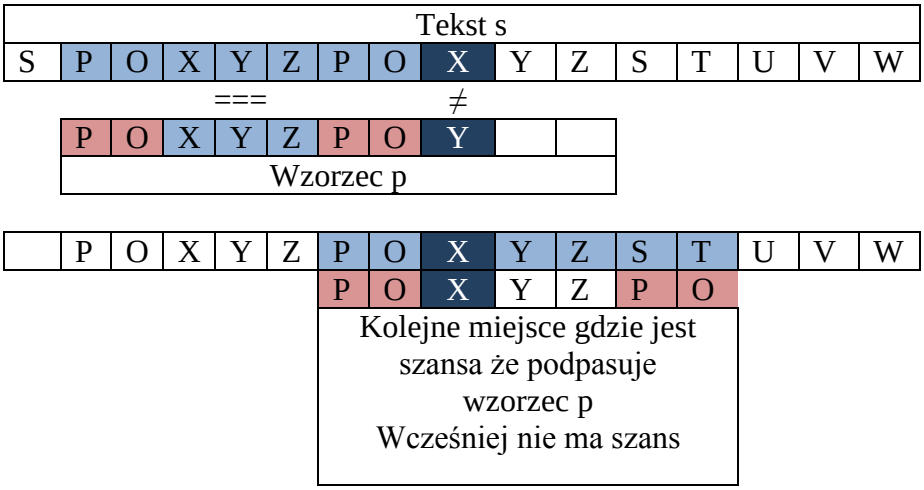
```cpp
int pi[1000000];
string a;
int main()
{
    pi[0] = -1;
    cin>>a;
    a = "#"+a;
    int dl=a.size();
    for(int i=1;i<=dl;i++)
    {
        int x = pi[i-1];
        while(x!=-1)
        {
            if(a[x+1] == a[i])
            {
                pi[i] = x+1;
                break;
   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