Naiwne wyszukiwanie wzorca w tekście

Wzorzec to spójny podciąg (podtekst), który występuje w danym ciągu znaków.

Naiwne wyszukiwanie wzorca jest najprostszą metodą sprawdzania czy w przeszukiwanym tekście można znaleźć określone wyrażenie. Jego prosta idea jest wystarczająca, aby wykonać zadanie, ale w przypadku większych danych algorytm ten może okazać się mało efektywny.

Problem znalezienia jednego tekstu w drugim to problem, z którym mamy do czynienia praktycznie na co dzień, być może nawet nie zdając sobie z tego sprawy.
Gdy jesteśmy na jakiejś stronie internetowej, albo mamy otwarty dokument tekstowy i wciskamy znany skrót CTRL+F, to wtedy właśnie wykonujemy przeszukiwanie tekstu w celu znalezienia wystąpień jakiegoś zadanego ciągu znaków.

Podstawowe pojęcia dotyczące przetwarzania tekstów

Alfabet - skończony zbiór symboli.

Łańcuchy - skończone ciągi symboli alfabetu.

s [*i*] – oznacza *i*-ty znak łańcucha *s*. Indeksy w łańcuchach rozpoczynają się od 0 (C++).

|*s*| – oznacza długość łańcucha (string length), 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. Na przykład, jeśli *s* = "ALA MA BOCIANA", to *s* [4 : 9] = "MA BO". Taki fragment łańcucha *s* będziemy nazywali oknem łańcucha.

Długość podłańcucha wyliczamy ze wzoru:
| *s* [*i* : *j*] | = *j* - *i*

Podłańcuch *s* [*i* : *i*] jest łańcuchem pustym – posiada długość 0, co wynika bezpośrednio z podanego powyżej wzoru.

Problem Wyszukiwania Wzorca – WW (ang. pattern matching) to jeden z podstawowych problemów tekstowych, który intensywnie badali wybitni informatycy. Rozwiązaniem jest wskazanie w ciągu ***s*** wszystkich pozycji ***i*** takich, że zachodzi równość:

$s[i : i + |p|] = p$

Oznacza to, iż wzorzec ***p*** jest fragmentem łańcucha ***s*** występującym na pozycji ***i***-tej.

Algorytm N – **naiwny** – ustawia okno o długości wzorca *p* na pierwszej pozycji w łańcuchu *s*. Następnie sprawdza, czy zawartość tego okna jest równa wzorcowi *p*. Jeśli **tak**, pozycja okna jest zwracana jako wynik, po czym okno przesuwa się o jedną pozycję w prawo i cała procedura powtarza się. Algorytm kończymy, gdy okno wyjdzie poza koniec łańcucha.

Okno wzorca słowa ABBA

```
#include<iostream>
using namespace std;

int main()
{
	string wzorzec, tekst;
	cin>> wzorzec >> tekst ;

	int m = wzorzec.size(),  n = tekst.size();

	for (int i = 0; i <= n-m; i++)
	{

		bool pasuje = true;
		for (int j = 0; j<m; j++)
			if (wzorzec[j] != tekst[i + j])
				pasuje = false;
		if (pasuje)
			cout << i << endl;

	}
	}
```

Zastosowanie zmiennej logicznej **bool**

```
#include <iostream>
using namespace std;

bool wzorzec(string a, string b) {
   int i, j;

   for (i = 0; i < b.length() - a.length(); i++) {
      j = 0;
      while (j < a.length()) {
         if (a[j] == b[i + j]) {
            j++;
         } else {
            break;
         }
      }
      if (j == a.length()) {
         return true;
      }
   }
   return false;
}
int main() {
   string a, b;

   a = "kot";
   b = "alamakota";

   if (wzorzec(a, b)) {
      cout << a << jest podtekstem" << b << endl;
   } else {
      cout << a << " nie jest podtekstem " << b << endl;
   }
   return 0;
}
```

Program sprawdza, czy w tekście znajduje się łańcuch tekstowy (podtekst, wzorzec).

Metoda **length()** zwraca długość łańcucha, czyli liczbę znaków. Zapis **a.length()** zwraca liczbę znaków zapamiętanych w zmiennej **a typu string .**

Zmienna **a** jest wzorcem do wyszukania, zmienna **b** tekstem , w którym szukany jest wzorzec.

```
#include<bits/stdc++.h>

using namespace std;

int szukaj(string wzorzec, string tekst)
{
	for(int i=0;i <= tekst.size() - wzorzec.size();i++) //po tekście
	{
		int c = 0;
		for(int j=0;j<wzorzec.size();j++)
		{
			if(wzorzec[j] != tekst[i + c])
				break;
			if(j == wzorzec.size() - 1)
				return i+1;
			++c;
		}
	}
	return -1;
}
int main(){
	string wzorzec, tekst;
	cout<<"Podaj tekst: ";
	cin>>tekst;
	cout<<"Podaj wzorzec: ";
	cin>>wzorzec;
	int pos = szukaj(wzorzec, tekst);

	if(pos == -1)
		cout<<"Wzorca nie znaleziono";
	else
		cout<<"Wzorzec znaleziono na "<<pos<<" pozycji";

	return 0;
}
```

Wyszukiwanie wzorca w tekście

Rozwiązanie z wykorzystaniem typu string oraz funkcji.

Zwrócenie pozycji wystąpienia pierwszego znaku wzorca w tekście lub -1, jeśli wzorca nie znaleziono

Algorytmy szybkie

Wyszukiwanie wzorca algorytmem :

Morrisa-Pratta

Knutha-Morrisa-Pratta

Boyera-Moore'a

Karpa-Rabina

Algorytm Boyera - Moora

Algorytm **Boyera-Moore'a** (w skrócie algorytm BM) rozpoczyna porównywanie od ostatniego znaku wzorca, czyli odwrotnie niż algorytm naiwny. 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.

Wyszukajmy wzorzec ABCAB w tekście ACBADBABCABD.

Lp.	Wyszukiwanie wzorca	Opis
1.	**A C B A D** B A B C A B D **A B C A B**	Okno wzorca umieszczamy na początku przeszukiwanego tekstu.
2.	**A C B A D** B A B C A B D **A B C A B**	Porównanie rozpoczynamy od ostatniego znaku wzorca. Znaki są różne. Dodatkowo znak D z tekstu nie występuje we wzorcu.
3.	A C B A D **B A B C A** B D → → → → → **A B C A B**	Dlatego okno wzorca możemy od razu przesunąć o 5 pozycji (z tylu znaków składa się cały wzorzec).
4.	A C B A D **B A B C A** B D **A B C A B**	Znów porównujemy ostatni znak wzorca ze znakiem tekstu. Jest niezgodność. Lecz w tym przypadku litera A tekstu występuje we wzorcu.
5.	A C B A D B **A B C A B** D → **A B C A B**	Okno wzorca przesuwamy tak, aby litera A z tekstu i ostatnia litera A ze wzorca zrównały się pozycjami.
6.	**A C B A D B A B C A B** D **A B C A B**	Teraz porównanie daje zgodność wszystkich znaków wzorca – zadanie zostało wykonane.

Algorytm Boyera - Moora

Cz.1

```
#include <iostream>
#include <string>
#include <cstdlib>
#include <time.h>
#include<stdlib.h>

using namespace std;

const int N  = 80;  // długość łańcucha s
const int M  =  5;  // długość wzorca p
const int zp = 65;  // kod pierwszego znaku alfabetu
const int zk = 66;  // kod ostatniego znaku alfabetu

int main()
{
string s,p;
int Last[zk - zp + 1],i,j,pp;

system("color 5");

srand((unsigned)time(NULL));

// generujemy łańcuch s

s = "";
for(i = 0; i < N; i++)
s += zp + rand() % (zk - zp + 1);

// generujemy wzorzec

p = "";
for(i = 0; i < M; i++)
p += zp + rand() % (zk - zp + 1);
```

Cz.2

```
// wypisujemy wzorzec

cout << p << endl;

// wypisujemy łańcuch

cout << s;

// dla wzorca obliczamy tablicę Last[]

for(i = 0; i <= zk - zp; i++)
Last[i] = -1;
for(i = 0; i < M; i++) Last[p[i] - zp] = i;
```

Cz.3

```
// szukamy pozycji wzorca w łańcuchu

pp = i = 0;
while(i <= N - M)
{
j = M - 1;
while((j > -1) && (p[j] == s[i + j])) j--;
if(j == -1)
{
while(pp < i)
{
cout << " "; pp++;
}
cout << "^"; pp++;
i++;
}
else i += max(1,j - Last[s[i + j] - zp]);
}
cout << endl << endl;

system("pause");
return 0;
}
```

Palindromy

Palindrom - łańcuch znakowy , który czyta się tak samo w obu kierunkach.

Przykład:
ABBCBBA – jest palindromem
ABBCABA – nie jest palindromem

Palindromy pojawiają się w genetyce (łańcuchy DNA, RNA), w tekstach, muzyce, matematyce, geometrii, fizyce itd. Stąd duże zainteresowanie informatyków w efektywnych algorytmach ich znajdowania. W badaniach genetycznych często szuka się tzw. przybliżonych , czyli nie pasujących do dokładnego. Takie palindromy występują w łańcuchach DNA, w których wystąpiły różnego rodzaju błędy genetyczne.

Wikipedia

Przykładowe wyrazy będące palindromami: Abba, Ada, Aga, Ala, anilina, Anna, bób kajak, kok PKP pop, potop radar, rotor, sedes sos zakaz, zaraz, zez.

A to idiota. A to kanapa pana Kota. A to kawa kota. Atak kata. Ej, żyrafa ryż je. I co idioci? I kobyłkom mokły boki. Może jeż łże jeżom. Wół utył i ma miły tułów. Zakopane na pokaz.

Palindromy

```
#include<iostream>
using namespace std;
#include<string>

int main()
{
int i,j;
string wyraz;

cout << "Podaj wyraz, ktory chcesz sprawdzic: ";
cin >> wyraz;
// i - pokazuje na kolejne litery wyrazu począwszy od początku, j - zaczyna od końca
// funkcja length() wywołana na obiekcie string zwraca jego długość
for (i = 0, j = wyraz.length()-1; i < j; i++, j--)
{
if (wyraz[i] != wyraz[j])
break;
}
if (i < j)
cout << "Podany wyraz nie jest palindromem" << endl;
else
cout << "Podany wyraz jest palindromem" << endl;

return 0;
}
```