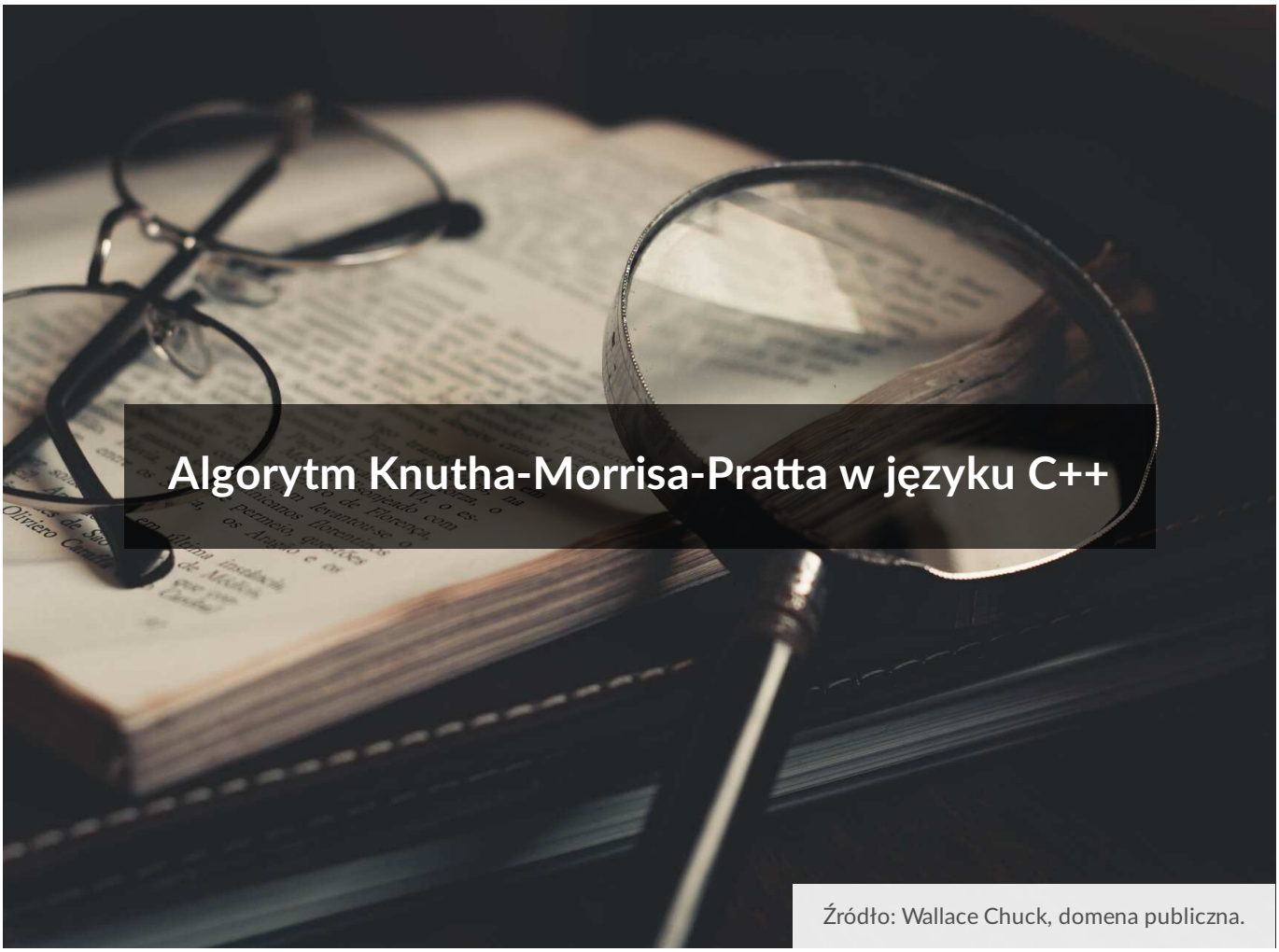
Algorytm Knutha-Morrisa-Pratta w języku C++



Źródło: Wallace Chuck, domena publiczna.

W e-materiale Algorytm Knutha-Morrisa-Pratta poznaliśmy algorytm służący do wyszukiwania wzorca w tekście. W tym e-materiale zdobyte wcześniej teoretyczne informacje wykorzystamy w praktyce. Prześledzimy w nim krok po kroku implementację omawianego algorytmu w języku C++, a także rozwiążemy zadania praktyczne.

Implementację tego algorytmu w pozostałych językach programowania znajdziesz w e-materiałach:

- Algorytm Knutha-Morrisa-Pratta w języku Java,
- Algorytm Knutha-Morrisa-Pratta w języku Python.

Więcej zadań? Przejdź do e-materiału Algorytm Knutha-Morrisa-Pratta – zadania maturalne.

> **Twoje cele**
>
> - Scharakteryzujesz działanie algorytmu KMP.
> - Zaimplementujesz algorytm KMP w języku C++.
> - Rozwiążesz przykładowe zadania programistyczne z wykorzystaniem algorytmu KMP.

Film samouczek

Problem 1

Zaimplementuj algorytm KMP i wykorzystaj go do wyszukiwania wzorca w tekście. Swoje rozwiązanie przetestuj dla napisu `"MEEMNEMME"` i wzorca `"ME"`.

Specyfikacja:

Dane:

- `wzorzec` – ciąg znaków przechowujący szukany tekst
- `napis` – ciąg znaków przechowujący przeszukiwany tekst

Wynik:

Program na wyjście standardowe wypisuje oddzielone spacjami wartości indeksów w tablicy `napis`, od których rozpoczynają się kolejne wystąpienia wzorca podanego w tablicy `wzorzec`.

Przykłady działania programu:

Dane:

- `napis = "AGHAAGAAHAA";`
- `wzorzec = "HAA";`

Wynik:

```
2 8
```

Dane:

- `napis = "ADAADAADAADA";`
- `wzorzec = "DAA";`

Wynik:

```
1 4 7
```

```
#include <iostream>

using namespace std;

int main () {

    // Tutaj dodaj kod. Żeby coś wypisać, użyj polecenia: cout
}
```

```
1
```

Polecenie 1

Dodaj do swojego programu komentarze tak, żeby był zrozumiały dla osoby, która nie potrafi programować.

Polecenie 2

Porównaj swoje rozwiązanie z filmem.

Film dostępny pod adresem /preview/resource/RgPg2hxVhBAds

Film przedstawia etapy pisania programu w języku C++wykorzystującego algorytm KMP do wyszukiwania wzorca w tekście.

Przeczytaj

Podsumowanie informacji o algorytmie KMP w języku C++

W omawianym przykładzie będziemy posługiwać się następującymi zmiennymi:

- `tekst` – ciąg znaków przechowujący przeszukiwany tekst,
- `znajdz` – ciąg znaków przechowujący wyszukiwany tekst,
- `dopasowania` – tablica częściowych dopasowań o rozmiarze równym liczbie znaków zapisanych w zmiennej `znajdz`.

Funkcja tworząca tablicę częściowych dopasowań

Krok 1

Zapiszmy nagłówek naszej funkcji:

`void` – funkcja nic nie zwraca, tylko wypełnia tablicę;

`string znajdz` – łańcuch znaków, na którym będziemy operować;

`int dopasowania[]` – tablica dopasowań.

```
1 void stworzTablice(string znajdz, int dopasowania[])
```

Krok 2

Deklarujemy zmienną pomocniczą `a`.

```
1 void stworzTablice(string znajdz, int dopasowania[])
2 {
3     int a = 0;
4 }
```

Krok 3

W tablicy dopasowań wartość komórki zapisanej pod indeksem zero ustawiamy na 0.

```
1 void stworzTablice(string znajdz, int dopasowania[])
```

```
{
    int a = 0;

    dopasowania[0] = 0;
}
```

Krok 4

Deklarujemy zmienną pomocniczą `i`, a następnie ustawiamy jej wartość na 1.

```
void stworzTablice(string znajdz, int dopasowania[])
{
    int a = 0;

    dopasowania[0] = 0;
    int i = 1;
}
```

Krok 5

Tworzymy pętlę, która będzie się wykonywała, dopóki zmienna `i` nie osiągnie wartości równej liczbie znaków w łańcuchu `znajdz`. Do pobrania długości tekstu użyj funkcji `size()`.

```
void stworzTablice(string znajdz, int dopasowania[])
{
    int a = 0;

    dopasowania[0] = 0;
    int i = 1;

    while(i < znajdz.size())
    {

    }
}
```

Krok 6

Sprawdzamy, czy znak zapisany pod indeksem `i` jest taki sam, jak znak zapisany pod indeksem `a`, czyli czy możemy rozszerzyć zakres długości prefiksu i sufiksu (pamiętajmy, że zgodnie z założeniem algorytmu prefiks zaczynający się od indeksu 0 i kończący się na indeksie $a-1$ jest identyczny z sufiksem o długości a, kończącym się na pozycji $i-1$).

```
void stworzTablice(string znajdz, int dopasowania[])
{
    int a = 0;

    dopasowania[0] = 0;
    int i = 1;

    while(i < znajdz.size())
    {
        if(znajdz[i] == znajdz[a])
        {

        }
    }
}
```

Krok 7

Jeśli tak jest, zwiększamy wartość zmiennej `a` o jeden, ustawiamy komórkę w tablicy `dopasowania` o indeksie `i` na wartość zmiennej `a`, następnie zwiększamy wartość zmiennej `i` o 1.

```
void stworzTablice(string znajdz, int dopasowania[])
{
    int a = 0;

    dopasowania[0] = 0;
    int i = 1;

    while(i < znajdz.size())
    {
        if(znajdz[i] == znajdz[a])
        {
            a++;
            dopasowania[i] = a;
```

```
            i++;
        }
    }
}
```

Krok 8

Jeśli tak nie jest, sprawdzamy, czy `a` jest różne od zera.

```
void stworzTablice(string znajdz, int dopasowania[])
{
    int a = 0;

    dopasowania[0] = 0;
    int i = 1;

    while(i < znajdz.size())
    {
        if(znajdz[i] == znajdz[a])
        {
            a++;
            dopasowania[i] = a;
            i++;
        }
        else
        {
            if(a != 0)

        }
     }
}
```

Krok 9

Jeśli tak, przypisujemy zmiennej `a` wartość zmiennej zapisanej w tablicy `dopasowania` pod indeksem `a - 1`.

```
void stworzTablice(string znajdz, int dopasowania[])
{
    int a = 0;
```

```
4
5       dopasowania[0] = 0;
6       int i = 1;
7
8       while(i < znajdz.size())
9       {
10          if(znajdz[i] == znajdz[a])
11          {
12              a++;
13              dopasowania[i] = a;
14              i++;
15          }
16          else
17          {
18              if(a != 0)
19                  a = dopasowania[a-1];
20          }
21       }
22  }
```

Krok 10

W przeciwnym razie przypisujemy zmiennej zapisanej pod indeksem `i` w tablicy `dopasowania` wartość zero, a następnie zwiększamy wartość zmiennej `i` o 1.

```
1  void stworzTablice(string znajdz, int dopasowania[])
2  {
3       int a = 0;
4
5       dopasowania[0] = 0;
6       int i = 1;
7
8       while(i < znajdz.size())
9       {
10          if(znajdz[i] == znajdz[a])
11          {
12              a++;
13              dopasowania[i] = a;
14              i++;
15          }
16          else
```

```
        {
            if(a != 0)
                a = dopasowania[a-1];
            else
            {
                dopasowania[i] = 0;
                i++;
            }
    }
}
```

Wyszukiwanie wzorca w tekście

Napisz właściwą funkcję algorytmu, która będzie wykorzystywała zapisaną funkcję tworzenia tablicy częściowych dopasowań.

Krok 1

Zapiszmy nagłówek funkcji.

```
void KnuthMorrisPratt(string znajdz, string tekst)
{

}
```

`void` – funkcja nic nie zwraca, tylko wypełnia tablicę;

`string znajdz` – łańcuch znaków, który przechowuje wyszukiwany wzorzec;

`string tekst` – łańcuch znaków przechowujący tekst, w którym szukamy wzorca.

Krok 2

Deklarujemy tablicę o liczbie komórek równej liczbie znaków we wzorcu. Będzie to tablica częściowych dopasowań.

```
void KnuthMorrisPratt(string znajdz, string tekst)
{
    int dopasowania[znajdz.size()];
}
```

Krok 3

Wywołujemy funkcję `stworzTablice()`, którą wcześniej zapisaliśmy, podając jako jej parametry łańcuch znaków `znajdz` oraz tablicę `dopasowania`.

```
void KnuthMorrisPratt(string znajdz, string tekst)
{
    int dopasowania[znajdz.size()];
    stworzTablice(znajdz, dopasowania);
}
```

Krok 4

Deklarujemy zmienne `i` oraz `j` i przypisujemy im wartości równe 0.

```
void KnuthMorrisPratt(string znajdz, string tekst)
{
    int dopasowania[znajdz.size()];
    stworzTablice(znajdz, dopasowania);
    int i = 0;
    int j = 0;
}
```

Krok 5

Tworzymy pętlę wykonującą się dopóki wartość zmiennej `i` będzie mniejsza od liczby znaków w przeszukiwanym tekście.

```
void KnuthMorrisPratt(string znajdz, string tekst)
{
    int dopasowania[znajdz.size()];
    stworzTablice(znajdz, dopasowania);
    int i = 0;
    int j = 0;
    while (i < tekst.size())
    {

    }
}
```

Krok 6

Sprawdzamy, czy znak zapisany w łańcuchu `znajdz` w komórce o indeksie `j` jest taki sam, jak znak zapisany w łańcuchu `tekst` w komórce o indeksie `i`.

```
void KnuthMorrisPratt(string znajdz, string tekst)
{
    int dopasowania[znajdz.size()];
    stworzTablice(znajdz, dopasowania);
    int i = 0;
    int j = 0;
    while (i < tekst.size())
    {
        if (znajdz[j] == tekst[i])
        {

        }
    }
}
```

Krok 7

Jeśli tak jest, inkrementujemy zmienną `i` oraz zmienną `j` .

```
void KnuthMorrisPratt(string znajdz, string tekst)
{
    int dopasowania[znajdz.size()];
    stworzTablice(znajdz, dopasowania);
    int i = 0;
    int j = 0;
    while (i < tekst.size())
    {
        if (znajdz[j] == tekst[i])
        {
            i++;
            j++;
        }
    }
}
```

Krok 8

Sprawdzamy, czy zmienna `j` jest równa liczbie znaków w łańcuchu `znajdz`.

```
void KnuthMorrisPratt(string znajdz, string tekst)
{
    int dopasowania[znajdz.size()];
    stworzTablice(znajdz, dopasowania);
    int i = 0;
    int j = 0;
    while (i < tekst.size())
    {
        if (znajdz[j] == tekst[i])
        {
            i++;
            j++;
        }

        if (j == znajdz.size())
        {

        }
    }
}
```

Krok 9

Jeśli tak jest, to wyświetlamy odpowiedni komunikat o odnalezieniu wzorca na pozycji `i - j` oraz ustawiamy wartość zmiennej `j` na wartość równą zmiennej zapisanej w tablicy `dopasowania` pod indeksem `j - 1`.

```
void KnuthMorrisPratt(string znajdz, string tekst)
{
    int dopasowania[znajdz.size()];
    stworzTablice(znajdz, dopasowania);
    int i = 0;
    int j = 0;
    while (i < tekst.size())
    {
        if (znajdz[j] == tekst[i])
        {
```

```
            i++;
            j++;
        }

        if (j == znajdz.size())
        {
            cout << "Znaleziono wzor na pozycji: " << i - j << en
            j = dopasowania[j - 1];
        }
    }
}
```

Krok 10

Jeśli warunek w poprzednim kroku nie jest spełniony, sprawdzamy, czy zmienna `i` jest mniejsza od liczby znaków w przeszukiwanym tekście i jednocześnie, czy znak zapisany w łańcuchu `znajdz` jest różny od znaku zapisanego w łańcuchu `tekst` pod indeksem `i`.

```
void KnuthMorrisPratt(string znajdz, string tekst)
{
    int dopasowania[znajdz.size()];
    stworzTablice(znajdz, dopasowania);
    int i = 0;
    int j = 0;
    while (i < tekst.size())
    {
        if (znajdz[j] == tekst[i])
        {
            i++;
            j++;
        }

        if (j == znajdz.size())
        {
            cout << "Znaleziono wzor na pozycji: " << i - j << en
            j = dopasowania[j - 1];
        }
        else if (i < tekst.size() && znajdz[j] != tekst[i])
        {

        }
```

```
    }
}
```

Krok 11

Jeśli tak, sprawdzamy, czy zmienna `j` jest różna od zera.

```
void KnuthMorrisPratt(string znajdz, string tekst)
{
    int dopasowania[znajdz.size()];
    stworzTablice(znajdz, dopasowania);
    int i = 0;
    int j = 0;
    while (i < tekst.size())
    {
        if (znajdz[j] == tekst[i])
        {
            i++;
            j++;
        }

        if (j == znajdz.size())
        {
            cout << "Znaleziono wzor na pozycji: " << i - j << en
            j = dopasowania[j - 1];
        }
        else if (i < tekst.size() && znajdz[j] != tekst[i])
        {
            if (j != 0)
        }
    }
}
```

Krok 12

Jeśli tak, ustawiamy wartość zmiennej `j` na wartość równą wartości zapisanej w tablicy `dopasowania`, pod indeksem `j - 1`.

```
void KnuthMorrisPratt(string znajdz, string tekst)
{
```

```
    int dopasowania[znajdz.size()];
    stworzTablice(znajdz, dopasowania);
    int i = 0;
    int j = 0;
    while (i < tekst.size())
    {
        if (znajdz[j] == tekst[i])
        {
            i++;
            j++;
        }

        if (j == znajdz.size())
        {
            cout << "Znaleziono wzor na pozycji: " << i - j << en
            j = dopasowania[j - 1];
        }
        else if (i < tekst.size() && znajdz[j] != tekst[i])
        {
            if (j != 0)
                j = dopasowania[j - 1];
        }
    }
}
```

Krok 13

W przeciwnym razie inkrementujemy zmienną `i`.

```
void KnuthMorrisPratt(string znajdz, string tekst)
{
    int dopasowania[znajdz.size()];
    stworzTablice(znajdz, dopasowania);
    int i = 0;
    int j = 0;
    while (i < tekst.size())
    {
        if (znajdz[j] == tekst[i])
        {
            i++;
            j++;
```

```
        }

        if (j == znajdz.size())
        {
            cout << "Znaleziono wzor na pozycji: " << i - j << en
            j = dopasowania[j - 1];
        }
        else if (i < tekst.size() && znajdz[j] != tekst[i])
        {
            if (j != 0)
                j = dopasowania[j - 1];
            else
                i++;
        }
    }
}
```

Całkowita implementacja w języku C++

Cały algorytm KMP w postaci obu funkcji prezentuje się następująco:

```
#include <iostream>
#include <string>

using namespace std;

void stworzTablice(string znajdz, int dopasowania[])
{
   int a = 0;

   dopasowania[0] = 0;
   unsigned int i = 1;

   while(i < znajdz.size())
   {
       if(znajdz[i] == znajdz[a])
       {
           a++;
           dopasowania[i] = a;
           i++;
       }
```

```
        else
        {
            if(a != 0)
                a = dopasowania[a-1];
            else
            {
                dopasowania[i] = 0;
                i++;
            }
        }
    }
}


void KnuthMorrisPratt(string znajdz, string tekst)
{
    int dopasowania[znajdz.size()];
    stworzTablice(znajdz, dopasowania);

    unsigned int i = 0;
    unsigned int j = 0;

    while(i < tekst.size())
    {
        if(znajdz[j] == tekst[i])
        {
            i++;
            j++;
        }

        if(j == znajdz.size())
        {
            cout<<"Znaleziono wzor na pozycji: "<<i-j<<endl;
            j = dopasowania[j-1];
        }
        else if(i < tekst.size() && znajdz[j] != tekst[i])
        {
            if(j != 0)
                j = dopasowania[j-1];
            else
                i++;
        }
```

```
63      }
64  }
65
66  int main() {
67      KnuthMorrisPratt("HAA", "AGHAAGAAHAA");
68      KnuthMorrisPratt("DAA", "ADAADAADAADA");
69      KnuthMorrisPratt("kot", "ala ma kota kot ma ale");
70
71      return 0;
72  }
```

Słownik

> **funkcja `size()`**
>
> funkcja w miejscu wywołania zwraca liczbę znaków w łańcuchu znaków (`string`); przykładowe wywołanie: `tekst.size()`
>
> **wzorzec**
>
> ciąg znaków, którego wystąpień poszukujemy w tekście

Sprawdź się

Pokaż ćwiczenia: 

Ćwiczenie 1

Napisz program, który obliczy i wyświetli tablicę prefikso-sufiksów dla podanego wzorca. Swój algorytm przetestuj dla wzorca `"AOAGDNMAOAGUIO"`.

Specyfikacja:

Dane:

- `tekst` – wzorzec dla którego należy znaleźć tablicę prefikso-sufiksów; ciąg znaków

Wynik:

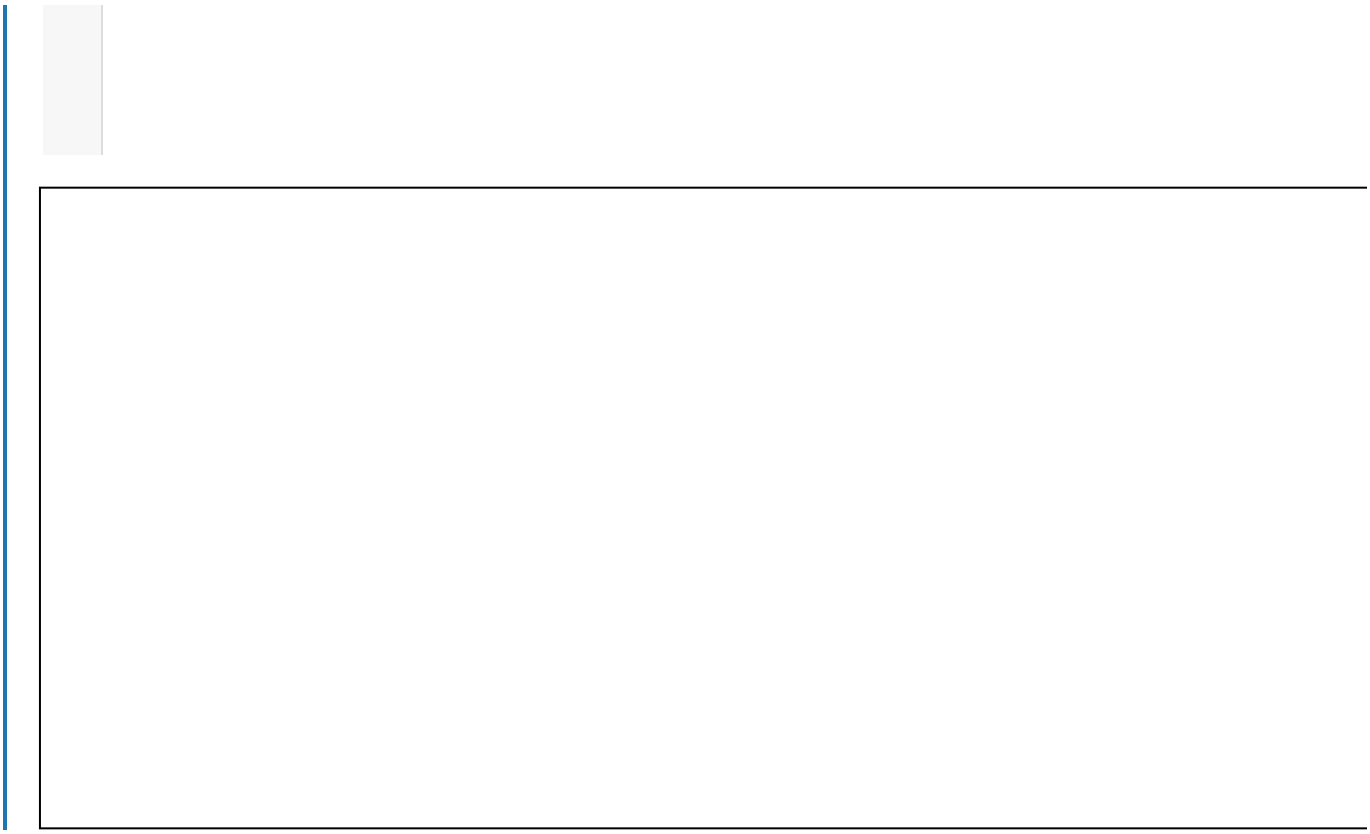
Program wyświetla na wyjściu standardowym zawartość `tablicaPrefiksoSufiksow` - tablicy prefikso-sufiksów dla podanego wzorca.

Twoje zadania

1. Program powinien wyświetlić kolejne komórki tablicy prefikso-sufiksów oddzielone spacjami.

```
#include <iostream>

using namespace std;

void generujTablicePrefiksoSufiksow(string wzorzec, int
tablicaPrefiksoSufiksow[]) {
    int j = 0;
    int i = 1;

    tablicaPrefiksoSufiksow[0] = 0;
    while (i < wzorzec.size()) {
        // Tu uzupełnij kod
    }
}
```

```

```



Ćwiczenie 2

Napisz program, który wyświetli wszystkie pozycje, na których został znaleziony podany wzorzec w tekście. Użyj algorytmu KMP. Swoje rozwiązanie przetestuj dla napisu `"AOAGDNMAOAGUIO"` i wzorca `"AOAG"`.

Specyfikacja:

Dane:

- `napis` – tekst do przeszukania; ciąg znaków
- `wzorzec` – wzorzec, który należy odnaleźć; ciąg znaków

Wynik:

Program wypisuje na standardowe wyjście wszystkie indeksy w tablicy `napis` od których rozpoczyna się wzorzec z tablicy `wzorzec`, oddzielone spacjami.

Twoje zadania

1. Program powinien wyświetlić oddzielone spacjami pozycje w tekście zapisanym w łańcuchu `napis`, na których został znaleziony wzorzec zapisany w łańcuchu `wzorzec`.

```
#include <iostream>

using namespace std;

void generujTablicePrefiksoSufiksow(string wzorzec, int tablicaPrefiksoSufiksow[]) {
    // Tu uzupełnij kod
}

void wyszukajKMP(string wzorzec, string napis) {
    int tablicaPrefiksoSufiksow[wzorzec.size()] = {};
    generujTablicePrefiksoSufiksow(wzorzec, tablicaPrefiksoSufiksow);
    // Tu uzupełnij kod
```

```
1
```

Ćwiczenie 3

Napisz program, który sprawdzi, czy słowa - elementy tablicy `slowa` - występują w podanym ciągu znaków `tekst`. Dla każdego słowa wypisz w osobnej linii TAK, jeśli występuje w tekście choć raz, lub NIE w przeciwnym wypadku.

Specyfikacja:

Dane:

- `tekst` – tekst do przeszukania; ciąg znaków
- `slowa` – tablica wzorców, dla których należy określić czy występują w tekście co najmniej raz czy też nie; tablica ciągów znaków

Wynik:

Dla każdego elementu tablicy `slowa`, program wypisuje na standardowe wyjście TAK, jeżeli występuje ono w tekście w zmienniej `tekst` co najmniej raz lub NIE, w przeciwnym razie.

Twoje zadania

1. Program powinien wypisać na standardowe wyjście słowo TAK, jeśli element tablicy `slowa` występuje w tekście, i NIE w przeciwnym wypadku.

```
#include <iostream>

using namespace std;

void generujTablicePrefiksoSufiksow(string wzorzec, int
tablicaPrefiksoSufiksow[]) {
    // Tu uzupełnij kod
}

int wyszukajKMP(string wzorzec, string napis) {
    int tablicaPrefiksoSufiksow[wzorzec.size()] = {};
    generujTablicePrefiksoSufiksow(wzorzec,
tablicaPrefiksoSufiksow);
```

```
12      // Tu uzupełnij kod
```

```
1
```

Dla nauczyciela

Autor: zespół autorski Contentplus.pl sp. z o.o.
Przedmiot: Informatyka

Temat: Algorytm Knutha-Morrisa-Pratta w języku C++

Grupa docelowa:

Liceum ogólnokształcące i technikum, liceum ogólnokształcące, technikum, zakres rozszerzony

Podstawa programowa:

Cele kształcenia – wymagania ogólne

I. Rozumienie, analizowanie i rozwiązywanie problemów na bazie logicznego i abstrakcyjnego myślenia, myślenia algorytmicznego i sposobów reprezentowania informacji.

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera oraz innych urządzeń cyfrowych: układanie i programowanie algorytmów, organizowanie, wyszukiwanie i udostępnianie informacji, posługiwanie się aplikacjami komputerowymi.

Treści nauczania – wymagania szczegółowe

I. Rozumienie, analizowanie i rozwiązywanie problemów.

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

2) do realizacji rozwiązania problemu dobiera odpowiednią metodę lub technikę algorytmiczną i struktury danych;

3) objaśnia dobrany algorytm, uzasadnia poprawność rozwiązania na wybranych przykładach danych i ocenia jego efektywność;

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera i innych urządzeń cyfrowych.

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

3) sprawnie posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

I + II. Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

3) objaśnia, a także porównuje podstawowe metody i techniki algorytmiczne oraz struktury danych, wykorzystując przy tym przykłady problemów i algorytmów, w szczególności:

g) metodę haszowania (wyszukiwanie wzorca w tekście),

Kształtowane kompetencje kluczowe:

- kompetencje cyfrowe;
- kompetencje osobiste, społeczne i w zakresie umiejętności uczenia się;
- kompetencje matematyczne oraz kompetencje w zakresie nauk przyrodniczych, technologii i inżynierii.

Cele operacyjne (językiem ucznia):

- Scharakteryzujesz działanie algorytmu KMP.
- Zaimplementujesz algorytm KMP w języku C++.
- Rozwiążesz przykładowe zadania programistyczne z wykorzystaniem algorytmu KMP.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych;
- ćwiczenia praktyczne.

Formy pracy:

- praca indywidualna;
- praca w parach;
- praca w grupach;
- praca całego zespołu klasowego.

Środki dydaktyczne:

- komputery z głośnikami, słuchawkami i dostępem do internetu;
- zasoby multimedialne zawarte w e-materiale;
- tablica interaktywna/tablica, pisak/kreda;

- oprogramowanie dla języka C++, w tym kompilator GCC/G++ 4.5 (lub nowszej wersji) i Code::Blocks 16.01 (lub nowszej wersji), Orwell Dev-C++ 5.11 (lub nowszej wersji) lub Microsoft Visual Studio.

Przebieg lekcji

Przed lekcją:

1. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Algorytm Knutha-Morrisa-Pratta w języku C++". Uczniowie mają zapoznać się z treściami w sekcji „Przeczytaj".

Faza wstępna:

1. Nauczyciel wyświetla temat i cele zajęć. Prosi uczniów, by na podstawie wiadomości zdobytych przed lekcją zaproponowali kryteria sukcesu.
2. **Rozpoznanie wiedzy uczniów.** Uczniowie tworzą pytania dotyczące tematu zajęć, na które odpowiedzą w trakcie lekcji.

Faza realizacyjna:

1. **Praca z tekstem.** Jeżeli przygotowanie uczniów do lekcji jest niewystarczające, nauczyciel prosi o indywidualne zapoznanie się z treścią zawartą w sekcji „Przeczytaj". Każdy uczestnik zajęć podczas cichego czytania wynotowuje najważniejsze kwestie poruszane w tekście.
2. **Praca z multimedium** Nauczyciel wyświetla zawartość sekcji „Film samouczek". Uczniowie zapoznają się z treścią Problemu 1 i opracowują rozwiązanie. Porównują w parach swoje odpowiedzi. Następnie weryfikują je, porównując z rozwiązaniem przedstawionym w filmie.
3. **Ćwiczenie umiejętności.** Uczniowie, pracując w parach, wykonują ćwiczenie nr 1 z sekcji „Sprawdź się". Nauczyciel sprawdza poprawność pisanych kodów, porównuje je i omawia wraz z uczniami. Wskazuje najbardziej efektywne rozwiązanie.
4. Liga zadaniowa – uczniowie pracując w parach, wykonują ćwiczenie nr 2 z sekcji „Sprawdź się", a następnie dzielą się swoimi wynikami przez porównywanie napisanego kodu z inną grupą, która również zakończyła zadanie.

Faza podsumowująca:

1. Nauczyciel ponownie wyświetla na tablicy temat i cele lekcji zawarte w sekcji „Wprowadzenie". W kontekście ich realizacji następuje omówienie ewentualnych problemów z rozwiązaniem ćwiczeń z sekcji „Sprawdź się".
2. Wybrany uczeń podsumowuje zajęcia z programowania w C++, zwracając uwagę na nabyte umiejętności.

Praca domowa:

1. Uczniowie wykonują ćwiczenie 3 z sekcji „Sprawdź się”.
2. Uczniowie wykonują polecenie: Poszukaj przykładów wyszukiwania w tekście wzorca.

Materiały pomocnicze:

- Oficjalna dokumentacja techniczna dla języka C++.
- Oficjalna dokumentacja techniczna dla kompilatora GCC/G++ 4.5 (lub nowszej wersji).
- Oficjalna dokumentacja techniczna dla kompilatora GCC/G++ 4.5 (lub nowszej wersji).
- Oficjalna dokumentacja techniczna dla oprogramowania Code::Blocks 16.01 (lub nowszej wersji), Orwell Dev-C++ 5.11 (lub nowszej wersji) lub Microsoft Visual Studio.

Wskazówki metodyczne:

- Uczniowie mogą wykorzystać multimedium w sekcji „Film samouczek” do przygotowania się do lekcji powtórkowej.