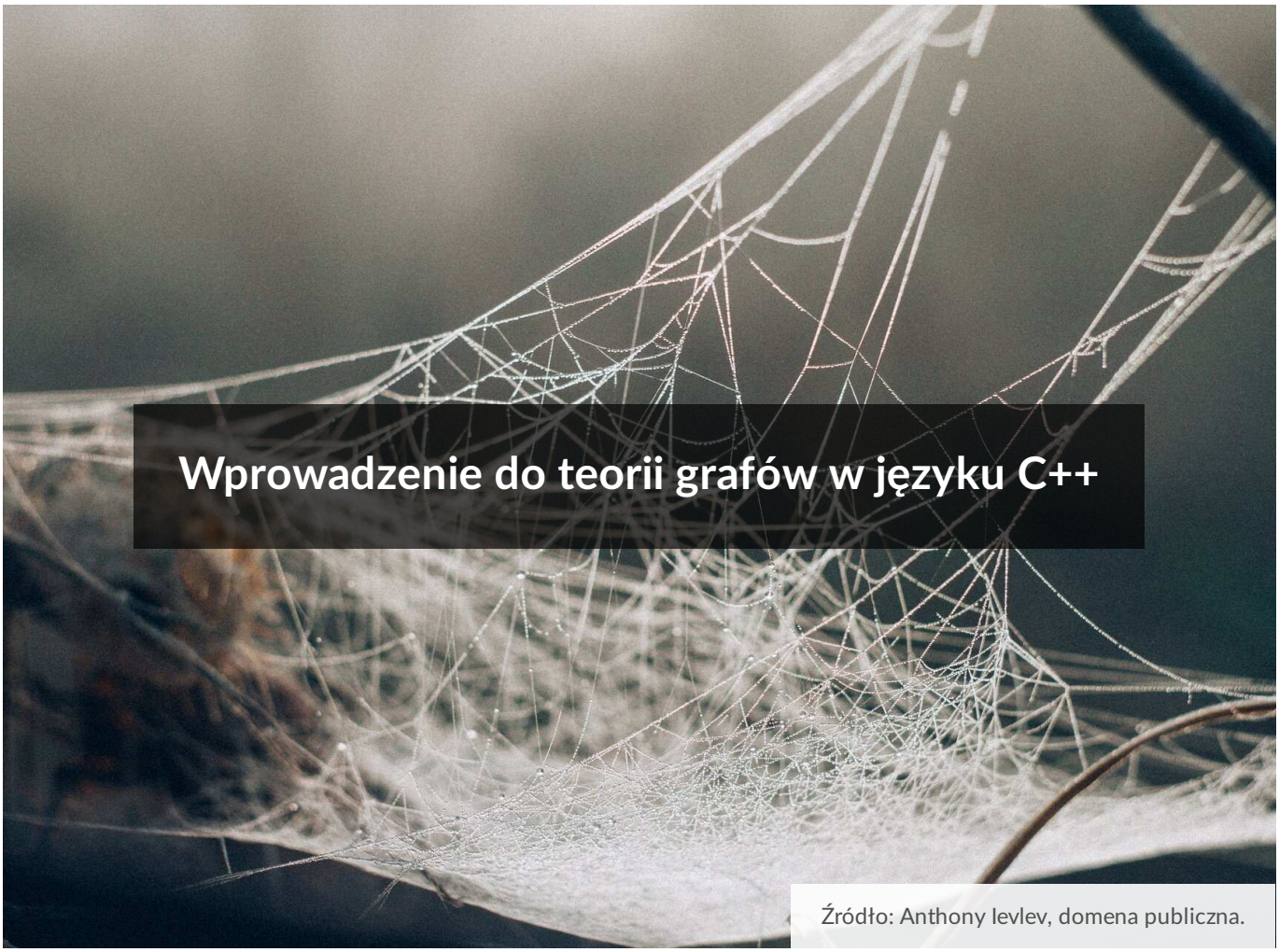
Wprowadzenie do teorii grafów w języku C++



Źródło: Anthony levlev, domena publiczna.

Grafy służą do modelowania różnych typów relacji i procesów w systemach fizycznych, biologicznych, społecznych i informacyjnych. Za pomocą wierzchołków oraz krawędzi przedstawiają wiele praktycznych sytuacji. Teoria grafów pozwala spojrzeć na problem z nieco innej perspektywy, opisać sytuację za pomocą połączeń lub powiązać relacje z pewnymi właściwościami.

W tym e-materiale przyjrzymy się zastosowaniu teorii grafów w języku C++. Więcej informacji o teorii grafów znajdziesz w e-materiale Wprowadzenie do teorii grafów.

Wyjaśnienie tego zagadnienia w kontekście pozostałych języków programowania znajdziesz w e-materiałach:

- Wprowadzenie do teorii grafów w języku Java,
- Wprowadzenie do teorii grafów w języku Python.

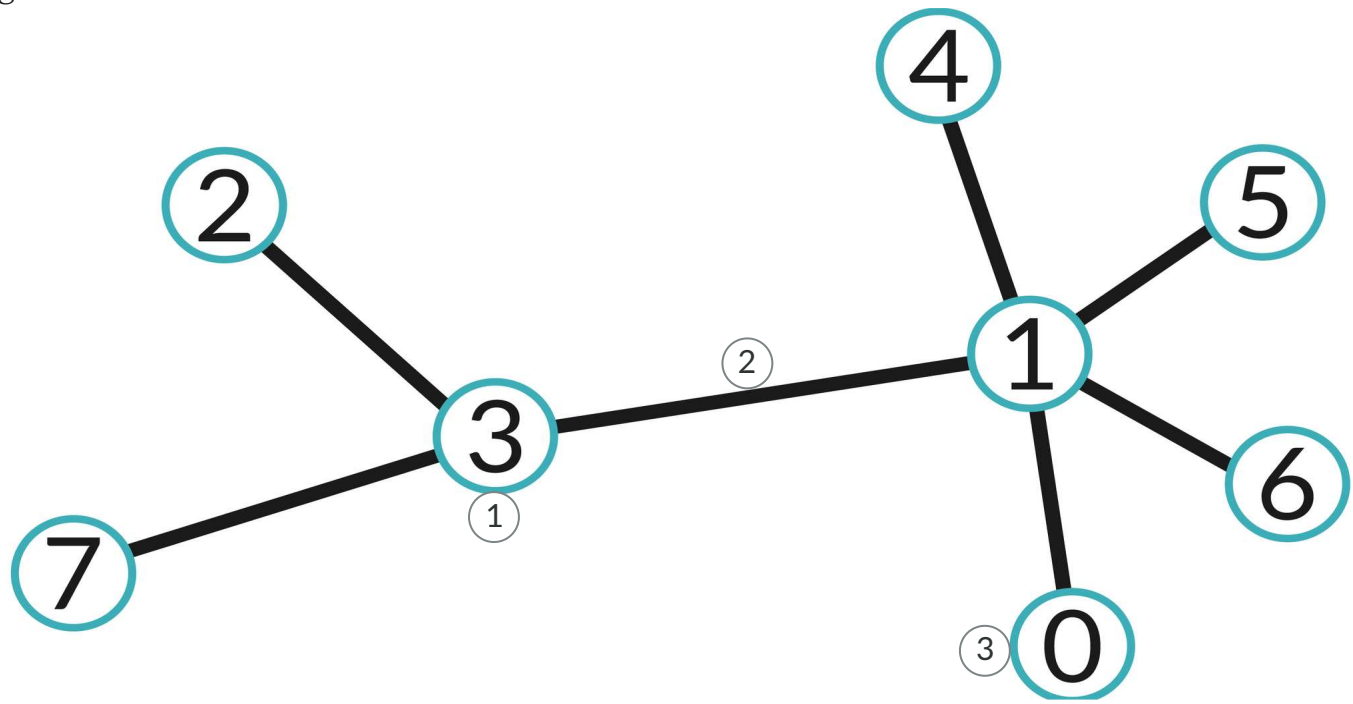
Twoje cele

- Zapoznasz się z przykładową implementacją klasy grafu w języku C++.
- Zaimplementujesz klasę grafu, zawierającą wszystkie potrzebne dane na temat wierzchołków, krawędzi i incydencji.
- Rozwiążesz problemy związane z sąsiedztwem wierzchołków grafów.

Przeczytaj

Struktura grafu

Graf jest strukturą składającą się z obiektów (wierzchołków) oraz z powiązań między nimi (krawędzi). Zazwyczaj przedstawiany jest w formie diagramu jako zbiór punktów połączonych odcinkami, gdzie punkty przedstawiają wierzchołki, a odcinki – krawędzie grafu.



1

Wierzchołek oznaczony jako „3”

2

Krawędź łącząca wierzchołki „3” i „1”

3

Sąsiad wierzchołka „1”

Wierzchołki i krawędzie

Podczas implementacji potrzebna jest możliwość rozróżnienia wierzchołków. Osiągnąć to można, na przykład oznaczając poszczególne wierzchołki kolejnymi liczbami naturalnymi, zaczynając od 0. Przechowywać je będziemy w tablicy statycznej typu `int[]`.

```
int nazwa_tablicy[]
```

W przypadku krawędzi wykorzystamy szablon struktury `pair`, który umożliwia przechowywanie dwóch obiektów jako pojedynczej jednostki. Dla naszej implementacji typem krawędzi będzie `pair<int, int>`. Przypiszmy zatem zmiennej `krawedz` wartość opisującą krawędź, która łączy wierzchołki 4 i 6:

> **Ważne!**
>
> Klasa `pair` pozwala traktować dwa obiekty jako pojedynczy obiekt. Obiekty mogą być różnego typu.
>
> ```
> pair<first, second> nazwa_pary
> ```

```
pair<int, int> krawedz = {4, 6};
```

Typ `pair` zawiera dwa pola składowe: `first` i `second`, przechowujące odpowiednio pierwszy i drugi obiekt.

```
pair<int, int> krawedz = {4, 6};
cout << krawedz.first << ", " << krawedz.second << endl;
// wypisze w konsoli:
// 4, 6
```

Zbiór krawędzi może być pusty albo składać się z jednej lub więcej krawędzi – pary wierzchołków. Dlatego niezbędna będzie tablica statyczna przechowująca obiekty owej pary. Oto jak możemy zadeklarować tę tablicę:

```
pair<int, int> krawedzie[3] = {{4, 6}, {5, 7}, {1, 2}};
```

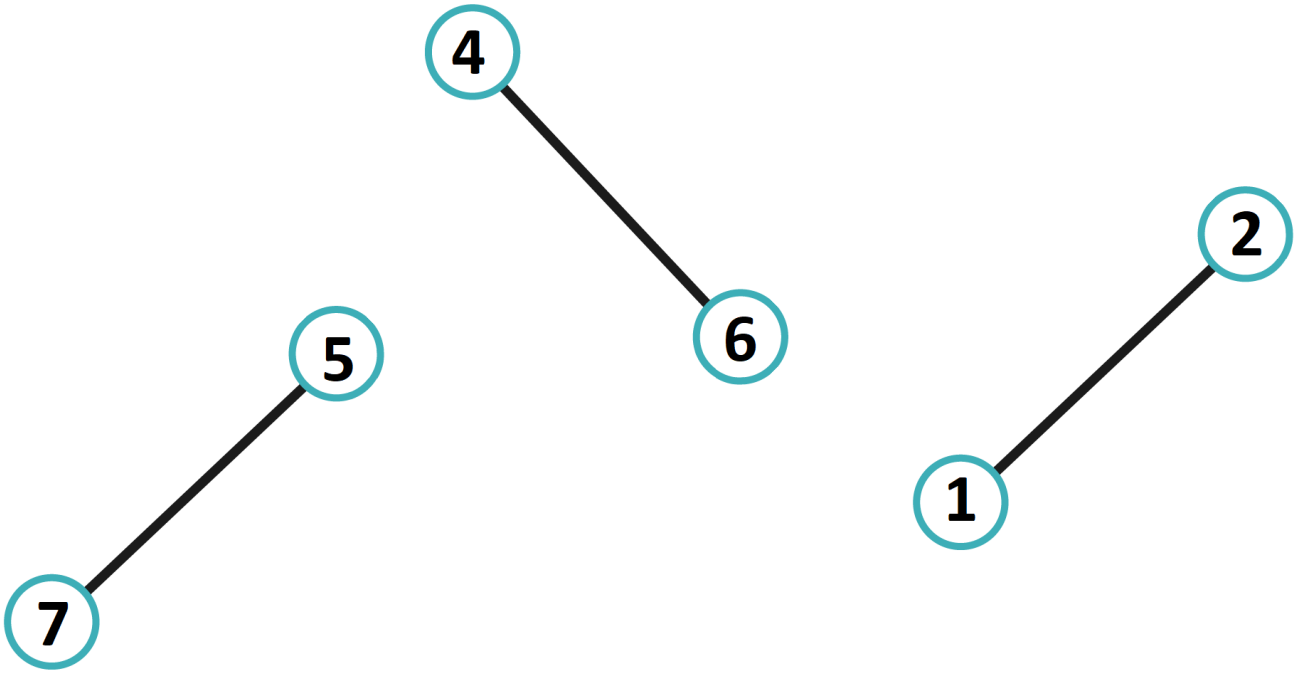
Na podstawie tablicy krawędzi może przedstawić również tablicę przechowującą numery wierzchołków:

```
int wierzcholki[] = {4, 6, 5, 7, 1, 2};
```

Graf opisany za pomocą poniższych tablic:

```
pair<int, int> krawedzie[3] = {{4, 6}, {5, 7}, {1, 2}};
int wierzcholki[] = {4, 6, 5, 7, 1, 2};
```

wyglądałby następująco:



> **Polecenie 1**
>
> Jak nazywamy graf, którego nie wszystkie wierzchołki połączone są ze sobą krawędziami?

Przykład

W jaki sposób moglibyśmy za pomocą kodu zaprezentować graf zamieszczony na początku tej sekcji?

Zacznijmy od wierzchołków.

```
int wierzcholki[] = {7, 2, 3, 1, 4, 5, 6, 0};
```

Mamy do czynienia z grafem spójnym, więc jego wierzchołki połączone są krawędziami.

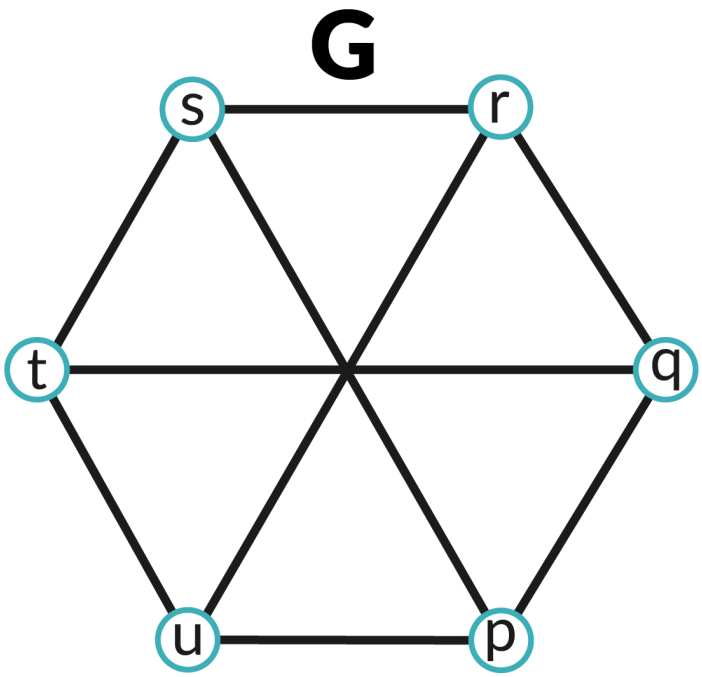
```
pair<int, int> krawedzie[7] = {{7, 3}, {2, 3}, {3, 1}, {4, 1}, {5
```

Wierzchołki w zaprezentowanym grafie deklarujemy zatem za pomocą poniższych tablic:

```
int wierzcholki[] = {7, 2, 3, 1, 4, 5, 6, 0};
pair<int, int> krawedzie[7] = {{7, 3}, {2, 3}, {3, 1}, {4, 1}, {5
```

Ćwiczenie 1

Dany jest graf G.



Opisz go za pomocą klasy `Graf` zawierającej listę krawędzi oraz listę wierzchołków.

Słownik

graf

struktura składająca się z niepustego zbioru wierzchołków oraz ze zbioru krawędzi; dla grafu G zbiór wierzchołków oznaczamy $V(G)$ (ang. *verticies* – wierzchołki), a zbiór krawędzi – $E(G)$ (ang. *edges* – krawędzie); liczbę elementów zbioru $V(G)$ oznaczamy literą n, zaś liczbę elementów zbioru $E(G)$ – m

graf spójny

graf, w którym każda para wierzchołków połączona jest drogą

izomorfizm

właściwość grafów polegająca na tym, że liczba krawędzi łączących dane dwa wierzchołki jednego grafu jest równa liczbie krawędzi łączących odpowiadające im wierzchołki drugiego grafu; jeśli graf G jest izomorficzny z grafem H, to gdy jakieś dwa wierzchołki są połączone krawędzią w jednym z grafów, odpowiadające im wierzchołki w drugim grafie również łączy krawędź; izomorfizm grafów zachowuje takie własności jak: liczba wierzchołków, liczba krawędzi, stopnie wierzchołków, czy spójność grafu

krawędź

nieuporządkowana para (niekoniecznie różnych) elementów zbioru wierzchołków, tj. takich, które są ze sobą połączone; w reprezentacji graficznej jest to linia łącząca te wierzchołki; krawędź może łączyć ze sobą jeden wierzchołek (wtedy nazywana jest pętlą)

lista krawędzi

struktura przechowująca wszystkie krawędzie występujące w grafie; choć nazywana jest listą, w konkretnych językach programowania może być realizowana np. za pomocą tablicy

stopień wierzchołka

liczba krawędzi incydentnych z danym wierzchołkiem (wychodzących z danego wierzchołka)

szablon

konstrukcja, która generuje klasę lub funkcję podczas kompilacji na podstawie argumentów dostarczanych przez użytkownika dla parametrów szablonu; umożliwia definiowanie operacji klasy lub funkcji i pozwala programiście określić, na jakich konkretnie typach operacje te powinny być wykonywane

wierzchołek

inaczej: punkt, węzeł; element niepustego zbioru, który wraz ze zbiorem krawędzi tworzy graf

klasa generyczna

klasa w języku Java pozwalająca stworzyć ogólną, uniwersalną klasę, która będzie wykonywała działania z różnymi typami danych, umożliwiając ponowne użycie kodu bez konieczności redefiniowania klasy dla każdego typu

para nieuporządkowana

zbiór złożony jedynie z dwóch **różnych** elementów: $\{a, b\}$

ścieżka

skończony ciąg krawędzi, w którym wszystkie krawędzie są różne

droga

ścieżka, w której wierzchołki są różne (z wyjątkiem ewentualnej równości wierzchołków pierwszego i ostatniego – mamy wtedy do czynienia ze szczególnym rodzajem drogi, drogą zamkniętą, tzw. cyklem)

Film samouczek

Polecenie 1

Zapoznaj się z prezentacją przedstawiającą to, w jaki sposób w języku C++ tworzona jest klasa grafu. Zastanów się, jakiego typu problemy programistyczne (ale nie tylko) można rozwiązywać za pomocą algorytmów grafowych.

Klasa grafu

1

Mamy już za sobą deklarowanie tablic krawędzi i wierzchołków. W następnym kroku definiujemy klasę `Graf` przechowującą wszystkie potrzebne pola i metody.

2

Klasa zawiera cztery pola:

- `int liczbaWierzcholkow` – dodatnia liczba całkowita przechowująca liczbę wierzchołków grafu;
- `int liczbaKrawedzi` – nieujemna liczba całkowita przechowująca liczbę krawędzi grafu;
- `int wierzcholki[MAX_N]` – tablica przechowująca numery wierzchołków typu `int`;
- `pair<int, int> krawedzie[MAX_M]` – tablica przechowująca pary numerów wierzchołków połączonych krawędzią.

3

W kodzie parametry `MAX_N` i `MAX_M` definiujemy za pomocą dyrektywy `#define` jako stałe określające odpowiednio maksymalną liczbę wierzchołków oraz maksymalną liczbę krawędzi.

4

Zapisujemy także konstruktor `Graf::Graf(int n)` przyjmujący jako argument liczbę wierzchołków. Jego zadaniem jest inicjalizacja wszystkich pól klasy `Graf`.

5

Wierzchołkom przypisujemy kolejne liczby, zaczynając od `0`, a kończąc na `liczbaWierzcholkow - 1`. Gdybyśmy chcieli odwołać się do pola `Graf::liczbaWierzcholkow`, dokonamy tego za pomocą kropki:

6

W tym momencie dodawanie wierzchołków lub krawędzi może zostać wykonane wyłącznie poprzez przypisanie pierwszemu wolnemu polu nowej wartości. Na przykład, aby dodać krawędź, która połączy wierzchołki 4 i 5, należy zapisać następujące instrukcje:

7

Jak poradzić sobie w sytuacji, gdy jedna z tablic krawędzi lub wierzchołków jest zapełniona, dowiemy się na przykładzie dodawania wierzchołków.

8

Gdy chcemy dodać do grafu wierzchołek o numerze `wierzcholek`, najpierw sprawdzamy, czy pole `liczbaWierzcholkow` jest mniejsze od `MAX\_N`. Następnie upewniamy się, czy nie istnieje jeszcze wierzchołek o danym numerze i dopiero wtedy możemy przypisać elementowi tablicy `wierzcholki` o indeksie `liczbaWierzcholkow` dany numer wierzchołka.

9

W przypadku gdy konieczne jest dodanie do grafu dużej liczby wierzchołków, można skorzystać z metody `void Graf::dodajWierzcholek(int wierzcholek)` zdefiniowanej poniżej.

10

Ważną cechą funkcji `dodajWierzcholek` jest działanie w przypadku zapełnienia tablicy lub istnienia identycznego wierzchołka. Instrukcja `return` powoduje natychmiastowe opuszczenie funkcji, w wyniku czego nie zostaje wykonana operacja dodania wierzchołka. Gdyby

jednak nie wykonano tej instrukcji, funkcja przeszłaby do przypisania numeru wierzchołka `wierzcholek` w tablicy `wierzcholki` o indeksie `liczbaWierzcholkow`.

11

Opierając się na tym, jak zdefiniowana jest metoda `dodajWierzcholek`, możemy zastosować to samo podejście przy implementacji metody `dodajKrawedz` dodającej krawędzie do grafu:

12

Nasz program wygląda następująco:

13

Zapiszmy funkcję `main`. Chcemy dodać do naszego grafu trzy krawędzie:

oraz dwa wierzchołki: 2 oraz 10.

14

Funkcja `main`:

15

Program w całości:

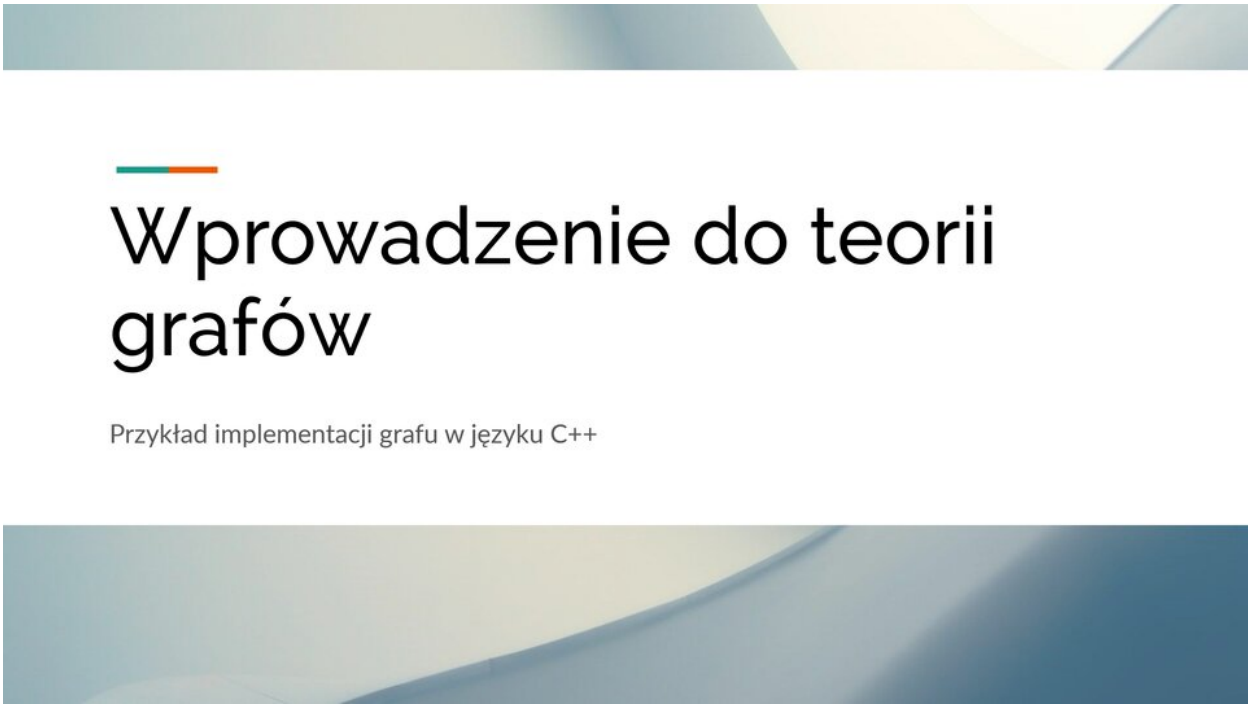
16 Wynik działania programu:

Dla zainteresowanych

Dalsza część tej sekcji poświęcona jest zaawansowanym sposobom reprezentacji grafów. Więcej informacji na ich temat znajdziesz w e-materiale Sposoby reprezentacji grafów. W filmie wykorzystano zaawansowane struktury danych, które omawiane są w e-materiałach Dynamiczne struktury danych oraz Dynamiczne struktury danych w języku C++.

Polecenie 2

Zapoznaj się z filmem przedstawiającym przykład implementacji grafu z wagami w języku C++. W filmie wykorzystano zaawansowane struktury danych – wektory (kontenery typu `vector`).



Film dostępny pod adresem /preview/resource/RneRnin7YBUOD

Film nawiązujący do treści materiału: Wprowadzenie do teorii grafów. Przykład implementacji grafu w języku C++.

Plik o rozmiarze 1.41 KB w języku polskim

Problem 1

Napisz program implementujący klasę grafu w języku C++. Klasa ta powinna zawierać zbiór wierzchołków grafu oraz krawędzie, łączące poszczególne wierzchołki. Każdy wierzchołek powinien mieć swój numer, aby można było je rozróżnić. W implementacji wykorzystaj klasę `vector` – strukturę umożliwiającą dynamiczne dodawanie elementów, która została przedstawiona w filmie.

Sprawdź się

Pokaż ćwiczenia:

Ćwiczenie 1

Do pojęć z zakresu teorii grafów przyporządkuj elementy, które można za ich pomocą przedstawić.

Wierzchołek

Krawędź

Graf

wiązanie chemiczne

miasto jako punkt na mapie

relacja znajomości

cząsteczka

miasto jako sieć ulic

schemat elektroniczny

osoba

mapa połączeń drogowych

połączenie drogowe

Ćwiczenie 2

Wskaż, z ilu wierzchołków (reprezentowanych liczbami naturalnymi) składa się graf spójny o podanej liście krawędzi:

```
krawedzie= [[1, 0], [1, 2], [1, 3], [2, 3], [3, 4], [5, 1], [5, 3],
```

Graf spójny to graf, w którym dowolne dwa wierzchołki są połączone drogą.

- ◯ 6
- ◯ 8
- ◯ 5
- ◯ 7

Ćwiczenie 3

Połącz każde pojęcie z zakresu teorii grafów z odnoszącym się do niego zapisem w języku C++.

krawędź	`pair<int, int>`
zbiór krawędzi	`int[]`
wierzchołek	`pair<int, int>[]`
zbiór wierzchołków	`int`

Ćwiczenie 4

Korzystając z podanych elementów, uzupełnij zapis funkcji `czyRowne()` sprawdzającej, czy dwie krawędzie typu `pair<int, int>` są równe, tzn. czy łączą te same wierzchołki. Funkcja zwróci odpowiednio wartość `true` lub `false`.

```
[          ] czyRowne(pair<int, int> krawedz1, pair<[          ]>
krawedz2) {
    if (krawedz1.first == krawedz2.second) {
        return krawedz1.[          ][          ][          ].[          ]
;
    } else {
        return [          ] == krawedz2;
    }
}
```

`int` `krawedz2` `krawedz1` `bool` `!=` `==` `first` `second` `float, float` `int, int`

Ćwiczenie 5

Dopasuj odpowiednie typy lub klasy do właściwej grupy, która może zostać przedstawiona za ich pomocą w języku C++. Pamiętaj, że typy reprezentujące krawędzie muszą przedstawiać relację pomiędzy dwoma obiektami.

wierzchołki

krawędzie

```
pair<struktura1, struktura1>
```

```
class klasa1 {
public:
    string nazwa;
    klasa1(string nazwa = ""):
        nazwa(nazwa) {}
};
```

```
int
```

```
class klasa2 {
public:
    int a, b;
    klasa2(int a, int b):
        a(a), b(b) {}
};
```

```
float
```

```
struct struktura2 {
    float a;
    float b;
};
```

```
struct struktura1 {
    string name;
};
```

```
pair<klasa1, klasa1>
```

Ćwiczenie 6

Napisz funkcję sprawdzającą, czy w grafie reprezentowanym przez listę krawędzi istnieje krawędź łącząca dane wierzchołki. Funkcja powinna zwracać indeks krawędzi w liście krawędzi lub -1, jeżeli taka krawędź nie istnieje. Krawędzie numerujemy od zera.

Swoje rozwiązanie przetestuj dla trzech par wierzchołków:

```
1,3
5,4
2,4
```

Specyfikacja problemu:

Dane:

- `krawedzie` – tablica krawędzi w grafie składająca się z par liczb całkowitych; indeksowana od zera

Wynik:

Program na wyjście standardowe wypisuje indeks znalezionej krawędzi lub -1, jeśli krawędź nie istnieje.

Przykładowe wyjście:

```
1
5
-1
```

Twoje zadania

1. Zdefiniowanie funkcji `znajdzKrawedz()` sprawdzającej, czy w grafie reprezentowanym przez listę krawędzi istnieje taka krawędź, która łączy wierzchołki p i q. Funkcja ma zwrócić indeks znalezionej krawędzi lub −1, jeśli taka krawędź nie istnieje.

```cpp
#include <iostream>
#define m 6

using namespace std;

pair<int, int> krawedzie[m] = {{1, 2}, {1, 3}, {1, 4}, {2, 3}, {3, 4}, {4, 5}};

int znajdzKrawedz(int p, int q) {
    // Zdefiniuj kod tutaj
    return -1;
}

int main() {
```

```

```

Ćwiczenie 7

Napisz funkcję sprawdzającą, czy w grafie reprezentowanym przez listę krawędzi istnieje przekazana jako parametr droga. Działanie programu przetestuj dla następujących danych:

```
pair<int, int> krawedzie[m] = {{1, 2}, {3, 1}, {1, 4}, {2, 4}, {
int droga1[] = {1, 2, 4, 3}
int droga2[] = {1, 2, 3, 4}
```

Specyfikacja problemu:

Dane:

- `krawedzie` – tablica krawędzi w grafie składająca się z par liczb całkowitych
- `droga1` – tablica liczb całkowitych
- `droga2` – tablica liczb całkowitych

Wynik:

Program na wyjście standardowe wypisuje 1, jeśli droga istnieje, lub 0, jeśli nie istnieje.

Twoje zadania

1. Zdefiniowanie funkcji `znajdzDroge()` sprawdzającej, czy w grafie reprezentowanym przez listę krawędzi istnieje dana droga w postaci tablicy `droga` indeksów wierzchołków. Funkcja ma zwrócić odpowiednio `true` lub `false`. Zastosuj wcześniej zaimplementowaną funkcję `znajdzKrawedz()`.

```
#include <iostream>
#define m 5

using namespace std;

pair<int, int> krawedzie[m] = {{1, 2}, {3, 1}, {1, 4}, {2, 4},
{3, 4}};
```

```
7
8  bool znajdzDroge(int droga[], int d) {
9      // Tutaj zmodyfikuj kod
10     return true;
11 }
12
```

```
1
```

Ćwiczenie 8

Napisz funkcję, która wypełni tablicę `stopnie` wartościami stopni wierzchołków. Graf reprezentowany jest przez listę krawędzi. Każdy wierzchołek może pojawić się tylko raz. Działanie programu przetestuj dla następujących danych:

```
pair<int, int> krawedzie[m] = {{0, 6}, {1, 2}, {1, 3}, {2, 6}, {
int wierzcholki[n] = {0, 1, 2, 3, 4, 5, 6}
int stopnie[n] = {0}
```

Specyfikacja problemu:

Dane:

- `krawedzie` – tablica krawędzi w grafie składająca się z par liczb całkowitych
- `wierzcholki` – tablica wierzchołków w grafie; tablica liczb całkowitych

Wynik:

Program wypisuje stopnie kolejnych wierzchołków.

Twoje zadania

1. Zdefiniowanie funkcji `stopnieWierzcholkow()` zapisującej w tablicy `stopnie` kolejnych wierzchołków.

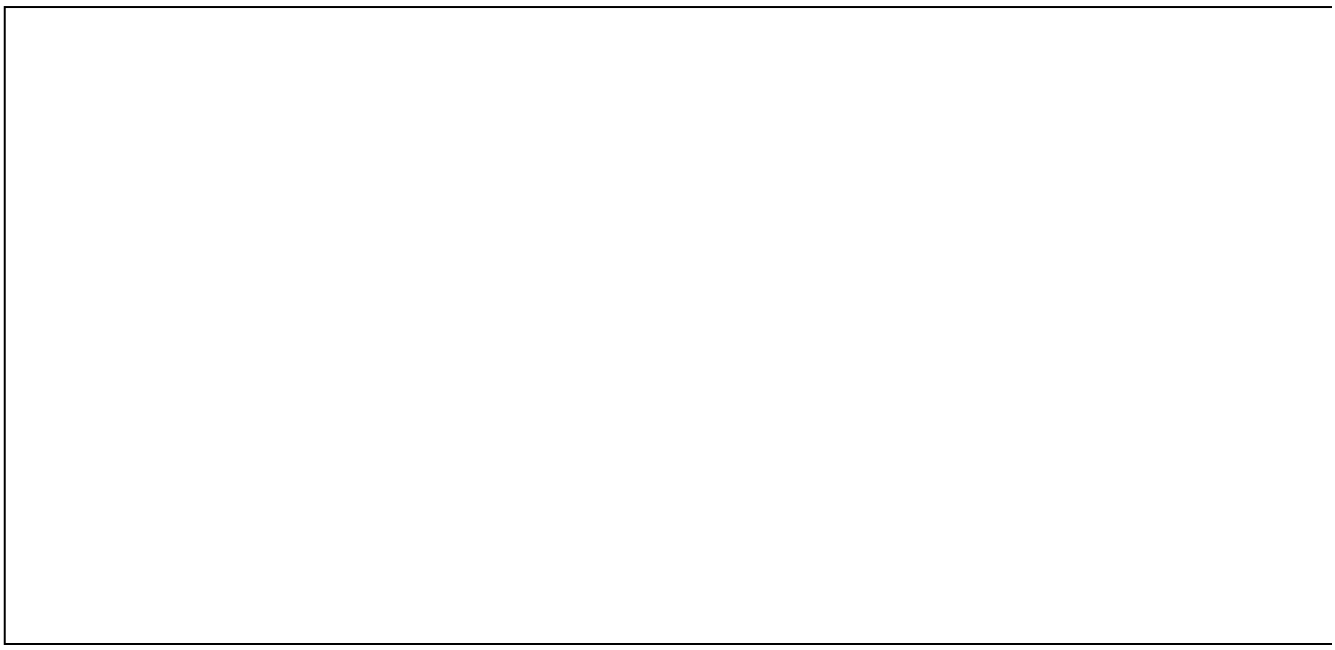
```
#include <iostream>
#include <algorithm>
#define m 8
#define n 7

using namespace std;

pair<int, int> krawedzie[m] = {{0, 6}, {1, 2}, {1, 3}, {2, 6},
{3, 4}, {5, 1}, {5, 3}, {5, 6}};
int wierzcholki[n] = {0, 1, 2, 3, 4, 5, 6};
int stopnie[n] = {0};
```

```
11
12 void stopnieWierzcholkow() {
13     // Uzupełnij kod tutaj
```

```
1
```



Ćwiczenie 9

Napisz funkcję, która zapisze w tablicy `sasiedzi` wszystkie sąsiednie wierzchołki dla danego wierzchołka grafu. Każdy sąsiad może pojawić się tylko raz. Indeksy sąsiadów mają być posortowane rosnąco. Dodatkowo funkcja powinna zwrócić stopień wierzchołka podanego jako parametr. Działanie programu przetestuj dla wierzchołka o indeksie 1.

Specyfikacja problemu:

Dane:

- `krawedzie` – tablica krawędzi w grafie składająca się z par liczb całkowitych

Wynik:

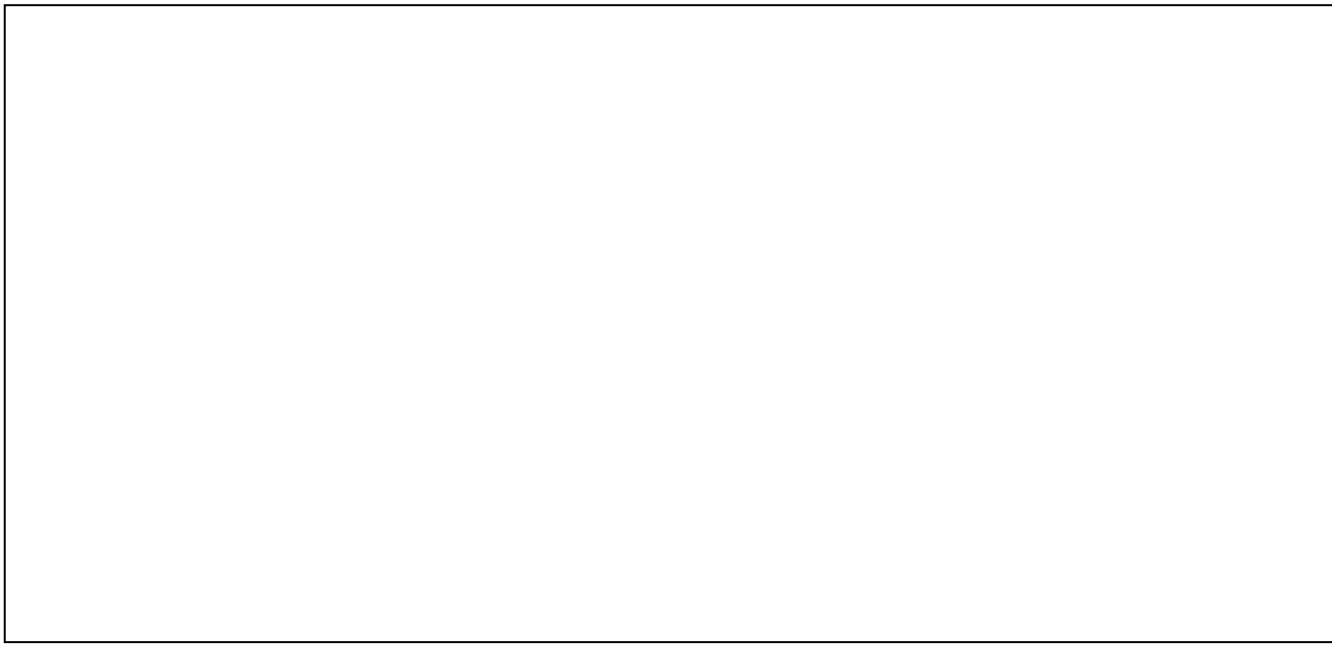
Program na wyjście standardowe wypisuje indeksy sąsiednich wierzchołków, oddzielone spacjami.

Twoje zadania

1. Zdefiniowanie funkcji `listaSasiadow()`, która zwróci stopień wierzchołka o indeksie `wierzcholek`. Indeksy sąsiadów muszą znaleźć się w tablicy globalnej `sasiedzi` posortowane rosnąco. Stopień wierzchołka powinien znaleźć się w zmiennej `stopienWierzcholka` wewnątrz funkcji `listaSasiadow()`.

```
1  #include <iostream>
2  #include <algorithm>
3  #define m 6
4  #define n 6
5
6  using namespace std;
7
8  pair<int, int> krawedzie[m] = {{1, 2}, {1, 3}, {2, 3}, {5, 1},
   {5, 4}, {5, 6}};
9  int sasiedzi[n] = {0};
10
11 int listaSasiadow(int wierzcholek) {
12     int stopienWierzcholka = 0;
13
```

```
1
```



Ćwiczenie 10

Napisz funkcję, która wypisze listę wierzchołków grafu wraz ze wszystkimi ich sąsiadami. Działanie programu przetestuj dla następujących danych:

```
pair<int, int> krawedzie[m] = {{1, 2}, {1, 3}, {2, 3}, {3, 4}, {
int wierzcholki[n] = {1, 2, 3, 4, 5, 6}
```

Specyfikacja problemu:

Dane:

- `krawedzie` – tablica krawędzi w grafie składająca się z par liczb całkowitych
- `wierzcholki` – tablica wierzchołków w grafie; tablica liczb całkowitych

Wynik:

Program na wyjście standardowe wypisuje w kolejnych wierszach indeksy wierzchołków oraz po dwukropku indeksy sąsiadów danego wierzchołka.

Twoje zadania

1. Zdefiniowanie funkcji `wypiszGraf()`, która wypisze listę wszystkich wierzchołków wraz z listą sąsiadów każdego wierzchołka grafu reprezentowanego przez listę krawędzi. Zastosuj gotową funkcję `listaSasiadow()`. Indeksy sąsiadów, jak i kolejność wierzchołków, dla których listy wypisujemy, muszą być posortowane rosnąco.

```
#include <iostream>
#include <algorithm>
#define m 7
#define n 6

using namespace std;

pair<int, int> krawedzie[m] = {{1, 2}, {1, 3}, {2, 3}, {3, 4},
{5, 1}, {5, 3}, {5, 6}};
```

```
 9 int wierzcholki[n] = {1, 2, 3, 4, 5, 6};
10 int sasiedzi[n] = {0};
11
12 void wypiszGraf() {
13     int stopienWierzcholka = 0;
```

```
1
```

Dla nauczyciela

Autorka: Paulina Wierzbińska
Przedmiot: Informatyka

Temat: Wprowadzenie do teorii grafów w języku C++

Grupa docelowa:

Szkoła ponadpodstawowa, 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:

1) w zależności od problemu rozwiązuje go, stosując metodę wstępującą lub zstępującą;

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:

1) projektuje i tworzy rozbudowane programy w procesie rozwiązywania problemów, wykorzystuje w programach dobrane do algorytmów struktury danych, w tym struktury dynamiczne i korzysta z dostępnych bibliotek dla tych struktur;

2) stosuje zasady programowania strukturalnego i obiektowego w rozwiązywaniu problemów;

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:

j) grafy (do przedstawiania abstrakcyjnego modelu sytuacji problemowych).

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):

- Zapoznasz się z przykładową implementacją klasy grafu w języku C++.
- Zaimplementujesz klasę grafu, zawierającą wszystkie potrzebne dane na temat wierzchołków, krawędzi i incydencji.
- Rozwiążesz problemy związane z sąsiedztwem wierzchołków grafów.

Strategie nauczania:

- konstruktywizm;
- konektywizm;
- behawioryzm.

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. Uczniowie przygotowują prezentacje dotyczące problemów grafowych.
2. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Wprowadzenie do teorii grafów w języku C++". Uczniowie zapoznają się z treściami w sekcji „Przeczytaj".

Faza wstępna:

1. Uczniowie odpowiadają na pytanie o to, w jakich sytuacjach może mieć zastosowanie graf.
2. Przedstawienie tematu zajęć oraz wspólne z uczniami ustalenie kryteriów sukcesu.
3. **Rozpoznanie wiedzy uczniów.** Uczniowie tworzą pytania dotyczące tematu zajęć, na które odpowiedzą w trakcie lekcji.

Faza realizacyjna:

1. **Praca z tekstem.** Nauczyciel ocenia, na podstawie informacji na platformie, stan przygotowania uczniów do zajęć. Jeżeli jest ono niewystarczające, prosi wybraną osobę o przedstawienie najważniejszych informacji z sekcji „Przeczytaj".
2. **Praca z multimedium.** Nauczyciel wyświetla zawartość sekcji „Film samouczek". Uczniowie indywidualnie zapoznają się z treścią znajdującej się tam prezentacji multimedialnej. W zależności od poziomu grupy, nauczyciel może klasie zaproponować zapoznanie się również z filmem.
3. **Ćwiczenie umiejętności.** Uczniowie wykonują ćwiczenia nr 1–5 z sekcji „Sprawdź się". Nauczyciel sprawdza poprawność wykonanych zadań, omawiając je wraz z uczniami.
4. Nauczyciel dzieli uczniów na grupy czteroosobowe. Uczniowie wykonują ćwiczenia nr 6–8 z sekcji „Sprawdź się", a następnie porównują swoje odpowiedzi z inną grupą.

Faza podsumowująca:

1. Nauczyciel wyświetla na tablicy temat lekcji i cele zawarte w sekcji „Wprowadzenie". W kontekście ich realizacji podsumowuje przebieg zajęć, a także wskazuje mocne i słabe strony pracy uczniów.
2. Na koniec zajęć nauczyciel prosi uczniów o rozwinięcie zdania: „Na dzisiejszych zajęciach nauczyłam/łem się jak…".

Praca domowa:

1. Uczniowie wykonują ćwiczenia 9–10 z sekcji „Sprawdź się".

Materiały pomocnicze:

- Oficjalna dokumentacja techniczna dla języka C++.
- 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.
- Robin J. Wilson, *Wprowadzenie do teorii grafów*, PWN, Warszawa 2007.

Wskazówki metodyczne:

- „Film samouczek" wprowadza nowe wiadomości i jest przeznaczony dla osób zainteresowanych teorią grafów – dotyczy zaawansowanych sposobów reprezentacji grafów. Można go wykorzystać w ramach kółkach zainteresowań jako podstawę do napisania programu, który pobierałby od ucznia informacje na temat dwóch grafów, a następnie weryfikował, czy są to grafy izomorficzne.