WYDZIAŁ
INFORMATYKI
I TELEKOMUNIKACJI

Grafy

Veronika Tronchuk

I rok Informatyki

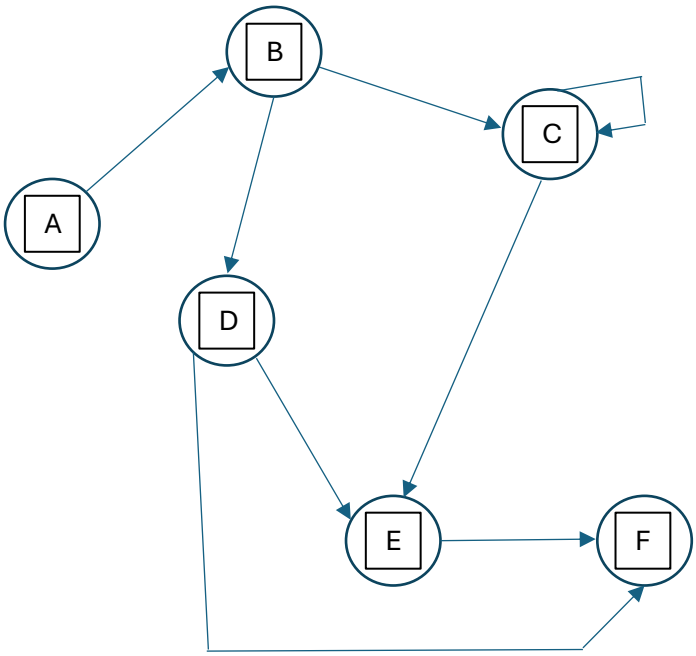
Nr albumu: 30019

Zadanie 1

Każdy kawałek lądu jest połączony nieparzystą liczbą mostów. Aby było to możliwe, wszystkie wierzchołki muszą mieć parzysty stopień lub maksymalnie dwa wierzchołki mogą mieć nieparzysty. W Królewcu wszystkie mają nieparzysty stopień, więc nie da się przejść wszystkich mostów jeden raz.

Zadanie 2

Graf na Obrazku 2 jest planarny, ponieważ można go narysować tak, aby krawędzie się nie przecinały. Poniżej przedstawiono przykład jego planarnej formy.



Zadanie 3

```
GraphMatrix:
0 1 0
1 0 1
0 1 0

GraphList:
0: 1
1: 0 2
2: 1
```

```
#include <vector>
```

dołącza bibliotekę do używania wektorów.

```
using namespace std;
```

pozwala nie pisać std:: przed vector czy cout.

```
class GraphMatrix {
private:
    vector<vector<int>> graphMatrix;
    int verticesNumber;
```

Deklaracja klasy GraphMatrix.
Zmienne prywatne: graphMatrix — macierz sąsiedztwa, verticesNumber — liczba wierzchołków.

```
public:
    GraphMatrix(int vertices) {
        verticesNumber = vertices;
        graphMatrix.resize(vertices, vector<int>(vertices, 0));
    }
```

Metody publiczne.
Konstruktor: przyjmuje liczbę wierzchołków, zapisuje ją i tworzy macierz NxN wypełnioną zerami.

```
void addEdge(int i, int j) {
    graphMatrix[i][j] = 1;
    graphMatrix[j][i] = 1;
}
```

Metoda addEdge ustawia 1 w macierzy dla połączenia między wierzchołkami i i j w obu kierunkach (graf nieskierowany).

```
void print() {
    for (int i = 0; i < verticesNumber; ++i) {
        for (int j = 0; j < verticesNumber; ++j) {
            cout << graphMatrix[i][j] << " ";
        }
        cout << "\n";
    }
}
```

Metoda print przechodzi przez całą macierz i wypisuje wartości (0 lub 1) dla każdego wierzchołka. Każdy wiersz to wiersz macierzy.

```
class GraphList {
private:
    vector<vector<int>> graphList;
```

Deklaracja klasy GraphList.
Zmienna prywatna: graphList — lista sąsiedztwa (tablica list sąsiadów).

```
public:
    GraphList(int vertices) {
        graphList.resize(vertices);
    }
```

Publiczny konstruktor: przyjmuje liczbę wierzchołków i tworzy pustą listę dla każdego wierzchołka.

Metoda addEdge dodaje wierzchołek v do listy u i odwrotnie — czyli łączy wierzchołki dwukierunkowo

```
void print() {
    for (int i = 0; i < graphList.size(); ++i) {
        cout << i << ": ";
        for (int neighbor : graphList[i]) {
            cout << neighbor << " ";
        }
        cout << "\n";
    }
}
```

Metoda print przechodzi przez wszystkie wierzchołki, wypisuje ich numer, a potem wszystkich sąsiadów tego wierzchołka

```
GraphMatrix gm(3);
```

Tworzy graf z 3 wierzchołkami (macierz).

```
gm.addEdge(0, 1);
```

Dodaje krawędź między wierzchołkami 0 i 1

```
gm.addEdge(1, 2);
```

Dodaje krawędź między wierzchołkami 1 i 2.

```
gm.print();
```

Wypisuje macierz sąsiedztwa.

```
GraphList gl(3);
```

Tworzy graf z 3 wierzchołkami (lista).

```
gl.addEdge(0, 1);
```

Dodaje krawędź między wierzchołkami 0 i 1.

```
gl.addEdge(1, 2);
```

Dodaje krawędź między wierzchołkami 1 i 2.

```
gl.print();
```

Wypisuje listę sąsiedztwa.

Zadanie

```
Wybierz typ grafu:
1 - Macierz sasiedztwa (GraphMatrix)
2 - Lista sasiedztwa (GraphList)
1
Podaj liczbe wierzcholkow: 3
Podaj nazwy wierzcholkow:
1
2
3
Podaj krawedzie. Wpisz stop aby zakonczyc:
1
2
3
stop
Nieprawidlowe nazwy!
stop
Macierz sasiedztwa:
   1 2 3
1 0 1 0
2 1 0 0
3 0 0 0
```

```
Wybierz typ grafu:
1 - Macierz sasiedztwa (GraphMatrix)
2 - Lista sasiedztwa (GraphList)
2
Podaj liczbe wierzcholkow: 2
Podaj nazwy wierzcholkow:
1
2
Podaj krawedzie. Wpisz stop aby zakonczyc:
1
2
stop
Lista sasiedztwa:
1: 2
2: 1
```

```
vector<string> vertexNames;
```

Dodano do klas GraphMatrix i GraphList. To tablica do przechowywania nazw wierzchołków, które użytkownik wpisuje z klawiatury.

```
void setVertexName(int index, string name) {
    vertexNames[index] = name;
}
```

Nowa metoda w każdej klasie. Umożliwia zapisanie nazwy danego wierzchołka według jego numeru.

```
cout << "    ";
for (int i = 0; i < verticesNumber; ++i) {
    cout << vertexNames[i] << " ";
}
cout << "\n";
for (int i = 0; i < verticesNumber; ++i) {
    cout << vertexNames[i] << " ";
    for (int j = 0; j < verticesNumber; ++j) {
        cout << graphMatrix[i][j] << " ";
    }
    cout << "\n";
}
```

Zamiast samych numerów teraz u góry i z boku wypisywane są nazwy wierzchołków.

```
cout << vertexNames[i] << ": ";
for (int neighbor : graphList[i]) {
    cout << vertexNames[neighbor] << " ";
}
cout << "\n";
```

Zamiast indeksów teraz pokazuje nazwę wierzchołka i nazwy sąsiadów.

```
for (int i = 0; i < n; ++i) {
    gm.setVertexName(i, names[i]);
}
```

Dodano, aby przekazać podane przez użytkownika nazwy do obiektu grafu od razu po wczytaniu.

```
cout << "Wybierz typ grafu:\n";
cout << "1 - Macierz sasiedztwa (GraphMatrix)\n";
cout << "2 - Lista sasiedztwa (GraphList)\n";
cin >> choice;
```

Pyta użytkownika, w jakiej formie ma być graf: macierz czy lista.

```
cout << "Podaj liczbe wierzcholkow: ";
cin >> n;
```

Pyta, ile wierzchołków ma mieć graf.

```
vector<string> names(n);
cout << "Podaj nazwy wierzcholkow:\n";
for (int i = 0; i < n; ++i) {
    cin >> names[i];
}
```

Zapisuje podane przez użytkownika nazwy wierzchołków w wektorze names.23:01

```
for (int i = 0; i < n; ++i) {
    gm.setVertexName(i, names[i]);
```

Przekazuje nazwy wierzchołków do obiektu grafu.

```
cout << "Podaj krawedzie. Wpisz stop aby zakonczyc:\n";
while (true) {
    string uName, vName;
    cin >> uName;
    if (uName == "stop") break;
    cin >> vName;

    int u = -1, v = -1;
    for (int i = 0; i < n; ++i) {
        if (names[i] == uName) u = i;
        if (names[i] == vName) v = i;
    }

    if (u != -1 && v != -1) {
        gm.addEdge(u, v);
    }
    else {
        cout << "Nieprawidlowe nazwy!\n";
    }
}
```

Pozwala wpisać krawędzie według nazw wierzchołków; wyszukuje indeksy po nazwach; dodaje krawędź.

Wnioski

Zrealizowane zadanie pozwoliło na praktyczne zastosowanie dwóch sposobów reprezentacji grafu oraz na stworzenie interfejsu, który umożliwia użytkownikowi wybór typu grafu, wprowadzanie liczby wierzchołków, ich nazw oraz połączeń między nimi. W trakcie pracy największym wyzwaniem okazało się poprawne powiązanie wszystkich etapów wprowadzania danych, wyszukiwanie wierzchołków po nazwach oraz zapewnienie spójności struktury grafu przy dynamicznym dodawaniu krawędzi. Zauważono także drobne niedoskonałości, takie jak brak dodatkowej kontroli na duplikaty nazw wierzchołków, brak blokady przed wielokrotnym dodaniem tej samej krawędzi oraz możliwość poprawy formatowania macierzy i listy sąsiedztwa, aby wynik był jeszcze bardziej czytelny23:07