Министерство образования Республики Беларусь

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра экономической информатики

**Лабораторная работа №16**

**“Графы. Решение практических задач с использованием графов** **”**

**Вариант 1**

|  |  |
| --- | --- |
| Выполнил: | Студент группы 172302  Гец Герман Альбертович |
| Проверил: | Ассистент  Кафедры ЭИ  Полоско Е.И. |

Минск 2018

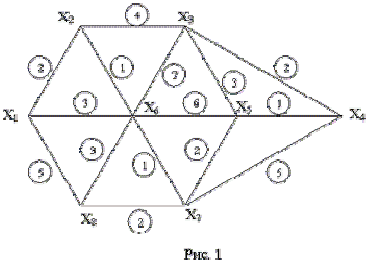
**Цель:** изучить основные принципы работы с графами и научиться писать программы с их использованием.

**Задачи:** написать код программы с использованием графов.

**Индивидуальное задание**

1. Построить граф, используя язык С, согласно данной схеме на рис.1.
2. По запросу пользователя должны удаляться:

* все рёбра с номером 3,
* вершина №6,
* вывести на экран общий вид получившегося в результате графа

1. Разработать алгоритм построения матрицы смежности для данного графа

#include <cstdlib>

#include <iostream>

#include <conio.h>

#include <stdio.h>

#include<iomanip>

#include <stdlib.h>

#include <windows.h>

using namespace std;

int \*\*Adjacency\_matrix; //Матрица смежности

int \*\*I; // Матрица инцидентности

int M; // Количество рёбер в матрице инцидентности

struct AdjListNode // Структура для представления списка смежности узла

{

int dest;

struct AdjListNode\* next;

};

struct AdjList // Структура для представления списка смежности

{

struct AdjListNode\* head; // указатель на головной узел списка

};

struct Graph // Структура для представления графа. Граф представляет собой массив из списков смежности.

{

int V; // Размер массива будет V (число вершин в графе)

struct AdjList\* array;

};

struct AdjListNode\* newAdjListNode(int dest) // вспомогательная функция для создания узла нового списка смежности

{

struct AdjListNode\* newNode = (struct AdjListNode\*)malloc(sizeof(struct AdjListNode));

newNode->dest = dest;

newNode->next = NULL;

return newNode;

}

int del\_node=99;

void delAdjListNode(struct Graph\* graph, int dest) // вспомогательная функция для удаления узла из графа

{

int n = graph->V;

for (int i = dest+1; i < n ; i++)

{

graph->array[i - 1].head = graph->array[i].head;

}

graph->V--;

int counter=0;

for (int i = 0; i < n; i++)

{

struct AdjListNode\* pCrawl = graph->array[i].head;

struct AdjListNode\* p = graph->array[i].head;

while (pCrawl)

{

if (pCrawl->dest == dest)

{

if (pCrawl->next)

{

p = pCrawl->next;

graph->array[i].head = p;

}

}

if (pCrawl->next)

{

if (pCrawl->next->dest == dest)

{

if (pCrawl->next->next)

{

p = pCrawl->next;

for (int i = 0; i < counter; i++)

{

if (graph->array[i].head->next)

graph->array[i].head->next;

}

/\*if(graph->array[i].head->next)\*/

graph->array[i].head->next = p->next;

}

}

}

pCrawl = pCrawl->next;

counter++;

}

}

del\_node = dest;

}

struct Graph\* createGraph(int V) // Служебная функция, которая создает граф с V вершинами

{

struct Graph\* graph = (struct Graph\*)malloc(sizeof(struct Graph));

graph->V = V;

// Создание массива списков смежности. Размер массива будет V

graph->array = (struct AdjList\*)malloc(V \* sizeof(struct AdjList));

// Инициализация каждого списка смежности как пустого, сделав начало = NULL

int i;

for (i = 0; i < V; ++i)

graph->array[i].head = NULL;

return graph;

}

void addEdge(struct Graph\* graph, int src, int dest) //Добавляет ребро неориентированного графа

{

// Добавляет ребро из src в построение. Новый узел добавляется в список смежности

// Список src.Узел добавляется в начале

struct AdjListNode\* newNode = newAdjListNode(dest);

newNode->next = graph->array[src].head;

graph->array[src].head = newNode;

// Поскольку граф неориентированный, также добавим ребро от Dest в SRC

newNode = newAdjListNode(src);

newNode->next = graph->array[dest].head;

graph->array[dest].head = newNode;

}

void create\_Adjacency\_matrix(struct Graph\* graph)

{

int n = graph->V;

Adjacency\_matrix = new int\* [n];

for (int i = 0; i < n; i++)

Adjacency\_matrix[i] = new int[n];

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

Adjacency\_matrix[i][j] = 0;

}

}

for (int i = 0; i < n; i++)

{

struct AdjListNode\* pCrawl = graph->array[i].head;

while (pCrawl)

{

Adjacency\_matrix[i][pCrawl->dest] = 1;

pCrawl = pCrawl->next;

}

}

}

void print\_Adjacency\_matrix(struct Graph\* graph)

{

int n = graph->V;

cout << "Матрица смежности:" << endl << endl;

for (int i = 1; i < n; i++)

{

for (int j = 1; j < n; j++)

{

cout << setw(2) << Adjacency\_matrix[i][j];

}

cout << endl;

}

cout << endl;

}

void printGraph(struct Graph\* graph)

{

int v;

int counter=0;

cout << "+---------+---------------------------+" << endl;

cout << "| Вершина | Смежные ей другие вершины |" << endl;

for (v = 1; v < graph->V; v++)

{

counter = 0;

cout << "+---------+---------------------------+" << endl;

struct AdjListNode\* pCrawl = graph->array[v].head;

if (v >= del\_node)

{

cout << "|" << setw(5) << v+1 << setw(5) << "|";

}

else

cout << "|" << setw(5) << v << setw(5) << "|";

while (pCrawl)

{

cout << setw(3) << pCrawl->dest;

counter++;

pCrawl = pCrawl->next;

}

cout <<setw(28 - counter\*3) << "|" << endl;

}

cout << "+---------+---------------------------+" << endl;

cout << endl;

}

void create\_Incid\_matrix(struct Graph\* graph)

{

M = 0;

int n = graph->V;

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

M += Adjacency\_matrix[i][j];

}

}

M /= 2; // вычисляем количество ребер

I = new int\* [n]; // Матрица инцидентности

for (int i = 0; i < n; i++)

I[i] = new int[M];

for (int i = 0; i < n; i++) // Заполняем нулями

{

for (int j = 0; j < M; j++)

{

I[i][j] = 0;

}

}

int C = 0; // Номер текущего ребра

for (int i = 0; i < n; i++) // Заполняем матрицу инцидентности

for (int j = i + 1; j < n; j++)

if (Adjacency\_matrix[i][j])

{

I[i][C] = 1;

I[j][C] = 1;

C++;

}

}

void print\_Incid\_matrix(struct Graph\* graph)

{

int n = graph->V;

cout << "Матрица инцидентности:" << endl << endl;

for (int i = 1; i < n; i++)

{

for (int j = 0; j < M; j++)

{

cout << setw(2) << I[i][j];

}

cout << endl;

}

}

void del\_Edge(int number)

{

int V = 8 + 1;

struct Graph\* graph = createGraph(V);

addEdge(graph, 1, 2);

addEdge(graph, 1, 8);

addEdge(graph, 2, 3);

addEdge(graph, 2, 6);

addEdge(graph, 3, 4);

addEdge(graph, 3, 6);

addEdge(graph, 4, 5);

addEdge(graph, 4, 7);

addEdge(graph, 5, 6);

addEdge(graph, 5, 7);

addEdge(graph, 6, 7);

addEdge(graph, 7, 8);

create\_Adjacency\_matrix(graph); // создание матрицы смежности

create\_Incid\_matrix(graph); // создание матрицы инцидентности

delAdjListNode(graph, 6); // удалить вершину 6

printGraph(graph); // представление списка смежности вышеуказанного графа

print\_Adjacency\_matrix(graph); // вывод матрицы смежности

print\_Incid\_matrix(graph);

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

cout << "Создаём граф согласно заданию 1: |" << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|" << endl;

// создать граф, согласно заданной выше фигуре

int V = 8 + 1;

struct Graph\* graph = createGraph(V);

addEdge(graph, 1, 2);

addEdge(graph, 1, 6);

addEdge(graph, 1, 8);

addEdge(graph, 2, 3);

addEdge(graph, 2, 6);

addEdge(graph, 3, 4);

addEdge(graph, 3, 5);

addEdge(graph, 3, 6);

addEdge(graph, 4, 5);

addEdge(graph, 4, 7);

addEdge(graph, 5, 6);

addEdge(graph, 5, 7);

addEdge(graph, 6, 7);

addEdge(graph, 6, 8);

addEdge(graph, 7, 8);

create\_Adjacency\_matrix(graph); // создание матрицы смежности

create\_Incid\_matrix(graph); // создание матрицы инцидентности

printGraph(graph); // представление списка смежности вышеуказанного графа

print\_Adjacency\_matrix(graph); // вывод матрицы смежности

print\_Incid\_matrix(graph); // вывод матрицы инцидентности

cout << endl << ",\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_," << endl;

cout << "| Удаляем вершину 6: |" << endl;

cout << "|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|" << endl;

delAdjListNode(graph, 6); // удалить вершину 6

create\_Adjacency\_matrix(graph); // создание матрицы смежности

create\_Incid\_matrix(graph); // создание матрицы инцидентности

printGraph(graph); // представление списка смежности вышеуказанного графа

print\_Adjacency\_matrix(graph); // вывод матрицы смежности

print\_Incid\_matrix(graph); // вывод матрицы инцидентности

cout << endl << ",\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_," << endl;

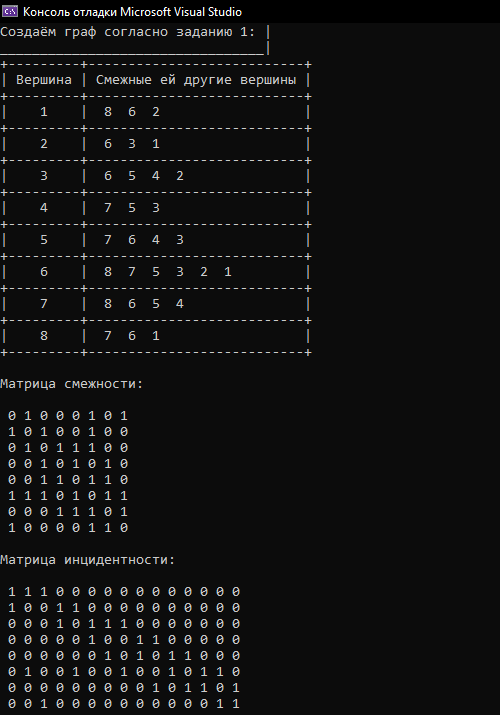
cout << "| Удаляем ребро 3: |" << endl;

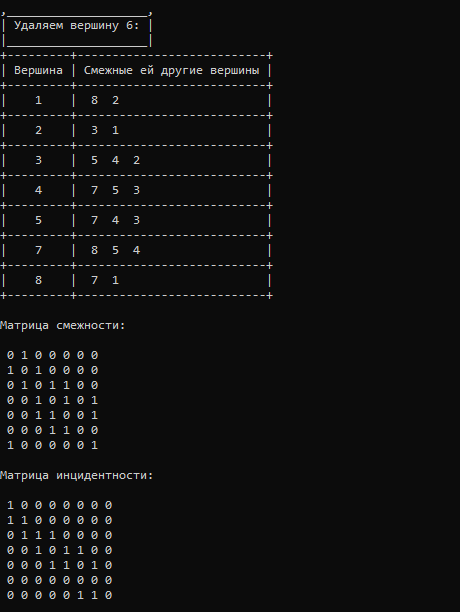
cout << "|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|" << endl;

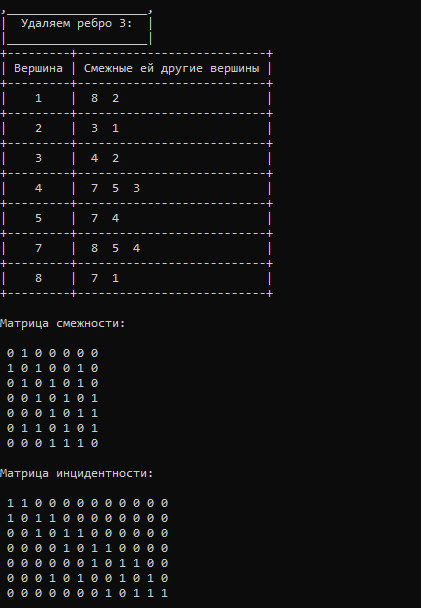
del\_Edge( 3); // удалить ребро 3

return 0;

}







**Вывод:** Я изучил основные принципы работы с графами и научился писать элементарные программы с их использованием.