Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

**МЕХАНІКО-ТЕХНОЛОГІЧНИЙ ФАКУЛЬТЕТ**

Кафедра програмування та захисту інформації

**Звіт**

з виконаної лабораторної роботи № 2

дисципліни “ Дискретна математика ”

на тему

«Подання графа для обробки за допомогою комп’ютера»

Виконав :

студент академічної групи КІ-15

Аннаєв А. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Перевірив :

Викладач

Гермак В.С.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кропивницкий- 2017

**Лабораторная работа №2**

**Тема**: Представление графа для обработки с помощью компьютера

**Цель**: Изучить способы представления графов как информационных

**ЗАДАНИЕ**

Выбрать согласно вашего номера в списке группы соответствующий граф и выполнить следующее:

1) вручную подать граф при всех способах;

2) запрограммировать представление произвольного конечного графа.

1

2

5

3

4

6

1

4

3

2

8

6

7

9

5

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** |
| **1** | **0** | **1** | **0** | **1** | **1** | **0** |
| **2** | **1** | **0** | **1** | **0** | **1** | **0** |
| **3** | **0** | **1** | **0** | **0** | **1** | **1** |
| **4** | **1** | **0** | **0** | **0** | **1** | **0** |
| **5** | **1** | **1** | **1** | **1** | 0 | **1** |
| **6** | **0** | **0** | **1** | **0** | **1** | **0** |

1. Матрица смежности

**Количество вершин : 6**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **1** | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| **2** | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| **3** | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| **4** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| **5** | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| **6** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |

1. Матрица инцидентности

**Количество вершин : 6**

**Количество ребер : 9**

1. Список ребер

**Количество вершин : 6**

**Количество ребер : 9**

**[ 1] : [ 1-2 ] = {1,2} or {2,1}**

**[ 2] : [ 1-4 ] = {1,4} or {4,1}**

**[ 3] : [ 1-5 ] = {1,5} or {5,1}**

**[ 4] : [ 2-3 ] = {2,3} or {3,2}**

**[ 5] : [ 2-5 ] = {2,5} or {5,2}**

**[ 6] : [ 3-5 ] = {3,5} or {5,3}**

**[ 7] : [ 3-6 ] = {3,6} or {6,3}**

**[ 8] : [ 4-5 ] = {4,5} or {5,4}**

**[ 9] : [ 5-6 ] = {5,6} or {6,5}**

1. Список смежности

**Количество вершин : 6**

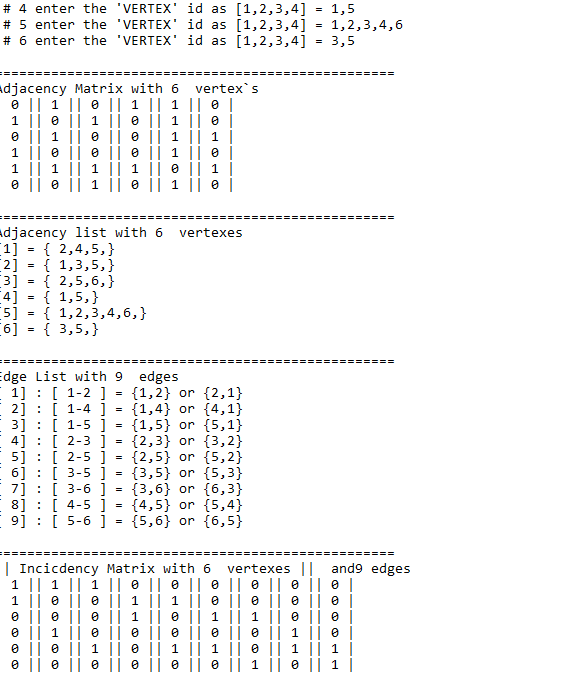
**[1] = { 2,4,5}**

**[2] = { 1,3,5}**

**[3] = { 2,5,6}**

**[4] = { 1,5}**

**[5] = { 1,2,3,4,6}**

**[6] = { 3,5}**

#include <iostream>

#include <display\_graph.h>

*using* *namespace* std;

void Separator(int length){

cout << endl;

*for* (int i=0; i< length; i++){

cout << "=";

}

cout << endl;

}

int main()

{

graph object(6);

Separator(50);

object.AdjacencyMatrix();

Separator(50);

object.AdjacencyList();

Separator(50);

object.EdgeList();

Separator(50);

object.IncidenceMatrix();

*return* 0;

#ifndef DISPLAY\_GRAPH\_H

#define DISPLAY\_GRAPH\_H

#include <vector>

#include <string>

*using* *namespace* std;

*struct* vertex{

int id;

vector<int> list; *//* *список* *смежных* *вершин*

vertex(int id):id(id){}

};

*//* *абстрактный* *класс* *ребра*

*struct* edge{

int id;

int beg;

int end;

edge(int id):id(id){}

};

*class* graph{

int vertex\_count;

int edge\_count;

*public*:

vector<vertex> list\_vertex;

vector<edge> list\_edge;

graph(int vertex\_count);

void Algorithm();

void setVertexList(int i, vertex \*ver);

void ParseString(string str, vertex \*ver);

void CreateGraph(int vertex\_count);

bool check\_edge(edge \*temp);

*//* *представление* *графа*

void AdjacencyMatrix();

void AdjacencyList();

void IncidenceMatrix();

void EdgeList();

};

#endif *//* *DISPLAY\_GRAPH\_H*

#include "display\_graph.h"

#include <vector>

#include <string>

#include<iostream>

*using* *namespace* std;

*//* *GRAPH*

*//===================================================================*

*//* *конструктор* *по* *умолчанию*

graph::graph(int vertex\_count){

*this*->vertex\_count = vertex\_count;

CreateGraph(vertex\_count);

}

*//=====================================================================================*

*//* *создание* *графа*

void graph::CreateGraph(int vertex\_count){

cout << "\n\tFilling vertex value " << endl;

*for*(int i=0; i< vertex\_count; i++){

vertex temp(i);

setVertexList(i,&temp);

list\_vertex.push\_back(temp);

}

Algorithm();

}

*//=====================================================================================*

*//* *заполнения* *списка* *смежных* *вершин*

void graph::setVertexList(int i, vertex \*ver){

string temp;

cout << " # " << i+1 << " enter the \'VERTEX\' id as [1,2,3,4] = ";

cin >> temp;

ParseString(temp,ver);

}

*//=====================================================================================*

*//* *парсинг* *строки*

void graph::ParseString(string str, vertex \*ver){

string temp;

*for*(int i = 0; i < int (str.length()); i++){

*if*(!isdigit(str[i])){

ver->list.push\_back(atoi(temp.c\_str()));

temp.clear();

}

*else*{

temp += str[i];

}

}

ver->list.push\_back(atoi(temp.c\_str()));

}

*//=====================================================================================*

*//* *алгоритм* *преобразования* *полученых* *данных* *о* *вершинах* *в* *список* *ребер*

void graph::Algorithm(){

int count = 0;

*for*(int i=0; i< vertex\_count; i++)

*for*(int j=0; j < int(list\_vertex[i].list.size()); j++){

edge temp(count);

temp.beg = i+1;

temp.end = list\_vertex[i].list[j];

*if*(check\_edge(&temp)){

list\_edge.push\_back(temp);

count++;

}

}

edge\_count = count;

}

*//=====================================================================================*

*//* *проверка* *списка* *ребер*

bool graph:: check\_edge(edge \*temp){

*for*(int i=0; i < int(list\_edge.size()); i++){

*if*(((temp->beg == list\_edge[i].beg)&&(temp->end == list\_edge[i].end))||

((temp->beg == list\_edge[i].end)&&(temp->end == list\_edge[i].beg)))

*return* *false*;

}

*return* *true*;

}

*//=====================================================================================*

*//* *Adjacency* *Matrix*

void graph::AdjacencyMatrix(){

cout << "Adjacency Matrix with " << vertex\_count << " vertex`s" << endl;

bool flag =*false*;

*for*(int i=0; i < vertex\_count; i++){

*for*(int j=1; j <= vertex\_count; j++){

*for*(int k=0; k < int(list\_vertex[i].list.size()); k++){

*if*(list\_vertex[i].list[k]== j)

flag = *true*;

}

*if*(flag){

cout << "| " << "1" << " |";

flag =*false*;

} *else* {

cout << "| " << "0"<< " |";

}

}

cout << endl;

}

}

*//=====================================================================================*

*//* *Incidency* *Matrix*

void graph::IncidenceMatrix(){

cout << "|| Incicdency Matrix with " << vertex\_count << " vertexes || and"

<< edge\_count <<" edges" << endl;

*for*(int i=0; i < vertex\_count; i++){

*for*(int j=0 ; j < edge\_count; j++){

*if*(list\_edge[j].beg == i+1 ||list\_edge[j].end == i+1){

cout << "| " << "1" << " |";

}

*else* {

cout << "| " << "0"<< " |";

}

}

cout << endl;

}

}

*//=====================================================================================*

*//* *Adjacency* *List*

void graph::AdjacencyList(){

cout << "Adjacency list with " << vertex\_count << " vertexes" << endl;

*for*(int i=0; i < vertex\_count; i++){

cout << "["<< i+1 <<"] = { ";

*for*(int k=0; k < int(list\_vertex[i].list.size()); k++){

cout << list\_vertex[i].list[k]<<",";

}

cout <<"}"<< endl;

}

}

*//=====================================================================================*

*//* *Edge* *list*

void graph::EdgeList(){

cout << "Edge List with " << edge\_count << " edges" << endl;

*for*(int j=0 ; j < edge\_count; j++){

cout << "[ " << list\_edge[j].id+1 << "] : "

<< "[ " << list\_edge[j].beg << "-"

<< list\_edge[j].end << " ] = "

<< "{" << list\_edge[j].beg << ","

<< list\_edge[j].end << "}"

<< " or "

<< "{" << list\_edge[j].end << ","

<< list\_edge[j].beg << "}" << endl;

}

}

*//=====================================================================================*