Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра інформаційних систем та мереж

### ЗВІТ

про виконання лабораторної роботи №8

на тему: «Графи. Обхід графу. Пошук.»

з предмету «Алгоритми та структури даних»

Виконала:

ст. гр. СА-32

Малинка А.А.

Прийняв:

доцент

Щербак С.С.

Львів-2017

**Мета роботи:** набуття навичок програмування графів

Лабораторне завдання: **(варіант 19)**

Розробити програму пошуку вершини в графі за її значенням.

Забезпечити зберігання та зчитування графа у вигляді матриць суміжності та  
інцидентності.

Блок-схема реалізації пошуку вершини у графі за її значенням

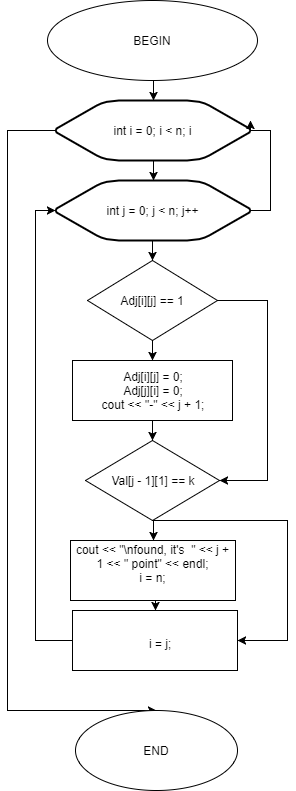


Рис 1. Блок-схема пошуку вершини у графі за її значенням

Згідно до реалізованого алгоритму, у матриці (інцидентності або суміжності) реалізується обхід графа відповідно до можливих переходів. Алгоритми дещо відмінні, та реалізація аналогічна, лише для матриці суміжності – рухаємось вздовж стрічок, а для матриці інцидентності – вздовж рядків. Знайшовши можливий перехід переходимо на цю вершину та порівнюємо її значення з вказаним параметром. Якщо значення вершини і параметра збігаємось, успішно завершуємо, якщо ні, рухаємось далі по стовбцях і рядках ( переходах між вершинами) поки не знайдемо потрібне значення. Виводимо номер вершини.

Код програми ( мовою с++):

///findtop.cpp

#include "findtop.h"

int \*\*Graph:: CreateArray(size\_t row, size\_t culloms)

{

int \*\* Arr = new int \*[row];

for (size\_t i = 0; i < culloms; ++i) {

Arr[i] = new int[culloms];

}

return Arr;

}

void Graph::AddArray(int \*\*Arr, size\_t row, size\_t culloms)

{

cout << endl;

for (size\_t i = 0; i < row; ++i) {

for (size\_t j = 0; j < culloms; ++j) {

cout << "Input array: " << i << " " << j << " = ";

if (!(cin >> Arr[i][j]))

throw 2;

}

}

}

void Graph::ShowArray(int\*\*Arr, size\_t row, size\_t culloms)

{

for (size\_t i = 0; i < row; ++i)

{

for (size\_t j = 0; j < culloms; ++j)

{

cout << "\t" << Arr[i][j] << ' ';

}

cout << endl;

}

}

int \*Graph:: CreateValueArray(size\_t row)

{

int \*Arr = new int[row];

return Arr;

}

void Graph:: FillValueArray(int \*Arr, size\_t row)

{

for (size\_t i = 0; i < row; ++i) {

cout << "Input array: ";

if (!(cin >> Arr[i]))

throw 2;

}

for (size\_t i = 0; i < row; ++i)

{

cout << "\t" << Arr[i] << ' ';

}

cout << endl;

}

int Graph::IncidencyFind(int\*\*Arr, int \*Val, size\_t row, size\_t culloms, int value)

{

for (int i = 0; i < row; i) {

for (int j = 0; j < culloms; j++) {

if (Arr[i][j] == 1) {

cout << "-" << j + 1;

Arr[i][j] = 0;

for (int l = 0; l < row; l++) {

if (Arr[l][j] == 1) {

Arr[l][j] = 0;

if (Val[l - 1] == value)

{

//cout << "\nfound, it's " << l << " point" << endl;

return l;

i = row;

j = culloms;

break;

}

i = l;

break;

}

}

}

}

}

}

int Graph::AdjacencyFind(int\*\*Arr, int\*Val, size\_t row, size\_t culloms, int value)

{

cout << "Steps: 1";

for (int i = 0; i < row; i) {

for (int j = 0; j < row; j++) {

if (Arr[i][j] == 1) {

Arr[i][j] = 0;

Arr[j][i] = 0;

cout << "-" << j + 1;

if (Val[j - 1] == value)

{

return j;

i = row;

break;

}

i = j;

break;

}

}

}

}

void Graph::TestIncidency() {

const int n = 7, m = 11, k = 4;

int Val[n][1] = { 5, 6, 7, 4, 2, 1, 9 };

int In[n][m] =

{

{ 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0 },

{ 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0 },

{ 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0 },

{ 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0 },

{ 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1 },

{ 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1 } };

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

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

{

cout << In[i][j] << ' ';

}

cout << endl;

}

cout << endl;

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

cout << Val[i - 1][1] << ' ';

}

cout << "way: 0";

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

for (int j = 0; j < m; j++) {

if (In[i][j] == 1) {

cout << "-" << j + 1;

In[i][j] = 0;

for (int l = 0; l < n; l++) {

if (In[l][j] == 1) {

In[l][j] = 0;

if (Val[l - 1][1] == k)

{

cout << "\nfound, it's " << l + 1 << " point" << endl;

i = n;

j = m;

break;

}

i = l;

break;

}

}

}

}

}

}

void Graph::TestAdjacency()

{

const int n = 7, m = 11, k = 4;// число вершин

int s = 0; // стартовая вершина (вершины везде нумеруются с нуля)

// чтение графа

int Adj[n][n] =

{ { 0, 1, 1, 1, 1, 0, 0 },

{ 1, 0, 0, 0, 1, 1, 0 },

{ 1, 0, 0, 1, 0, 0, 1 },

{ 1, 0, 1, 0, 1, 1, 1 },

{ 1, 1, 0, 1, 0, 0, 1 },

{ 0, 1, 0, 1, 0, 0, 1 },

{ 0, 0, 1, 1, 0, 1, 0 } };

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

{

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

{

cout << Adj[i][j] << ' ';

}

cout << endl;

}

cout << endl;

int Val[n][1] = { 5, 6, 7, 4, 2, 1, 9 };

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

{

cout << Val[i - 1][1] << ' ';

}

cout << "\nSteps: 1";

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

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

if (Adj[i][j] == 1) {

Adj[i][j] = 0;

Adj[j][i] = 0;

cout << "-" << j + 1;

if (Val[j - 1][1] == k)

{

cout << "\nfound, it's " << j + 1 << " point" << endl;

i = n;

break;

}

i = j;

break;

}

}

}

cout << endl;

}

/// findtop.h

#pragma once

#include<iostream>

#include <vector>

#include <queue>

using namespace std;

class Graph

{

public:

int \*\*CreateArray(size\_t row, size\_t culloms);

void AddArray(int \*\*Arr, size\_t row, size\_t culloms);

void ShowArray(int\*\*Arr, size\_t row, size\_t culloms);

int \*CreateValueArray(size\_t row);

void FillValueArray(int \*Arr, size\_t row);

int IncidencyFind(int\*\*Arr, int \*Val, size\_t row, size\_t culloms, int value);

int AdjacencyFind(int\*\*Arr, int\*Val, size\_t row, size\_t culloms, int value);

void TestIncidency();

void TestAdjacency();

};

//main.cpp

#include "findtop.h"

void main()

{

int Choice = 0;

Graph graph;

size\_t row, culloms;

int k = 0;

int n = 1;

cout << "(1)-Add insidence matrix, (2)-add adjacency matrix, (3)-test insidence, (4)-test adjacency\n";

cin >> Choice;

try {

switch (Choice) {

case 1: {

cout << "Let's add an insidence matrix for graph \nInput number of tops ";

cin >> row;

cout << "Input number of ways"; cin >> culloms;

cout << "\nInput values for graph tops\n ";

int \*Val1 = graph.CreateValueArray(row);

graph.FillValueArray(Val1, row);

int \*\* Arr1 = graph.CreateArray(row, culloms);

graph.AddArray(Arr1, row, culloms);

graph.ShowArray(Arr1, row, culloms);

cout << "Value of top to find "; cin >> k;

cout << "\nFound, it's " << graph.IncidencyFind(Arr1, Val1, row, culloms, k) << " point" << endl;

break;

}

case 2:

{

cout << "Let's add an adjacency matrix for graph /n Input number of tops ";

cin >> row;

culloms = row;

cout << "Input values for graph tops\n ";

int \*Val2 = graph.CreateValueArray(row);

graph.FillValueArray(Val2, row);

int \*\*Arr2 = graph.CreateArray(row, culloms);

graph.AddArray(Arr2, row, culloms);

graph.ShowArray(Arr2, row, culloms);

cout << "Value of top to find "; cin >> k;

cout << "\nFound, it's " << graph.AdjacencyFind(Arr2, Val2, row, culloms, k) << " point" << endl;

break;

}

case 3:

{

cout << "Here is the test insidence matrix for graph\n";

graph.TestIncidency();

break;

}

case 4:

{

cout << "Here is the test adjacency matrix for graph\n";

graph.TestAdjacency();

break;

}

}

}

catch (int error)

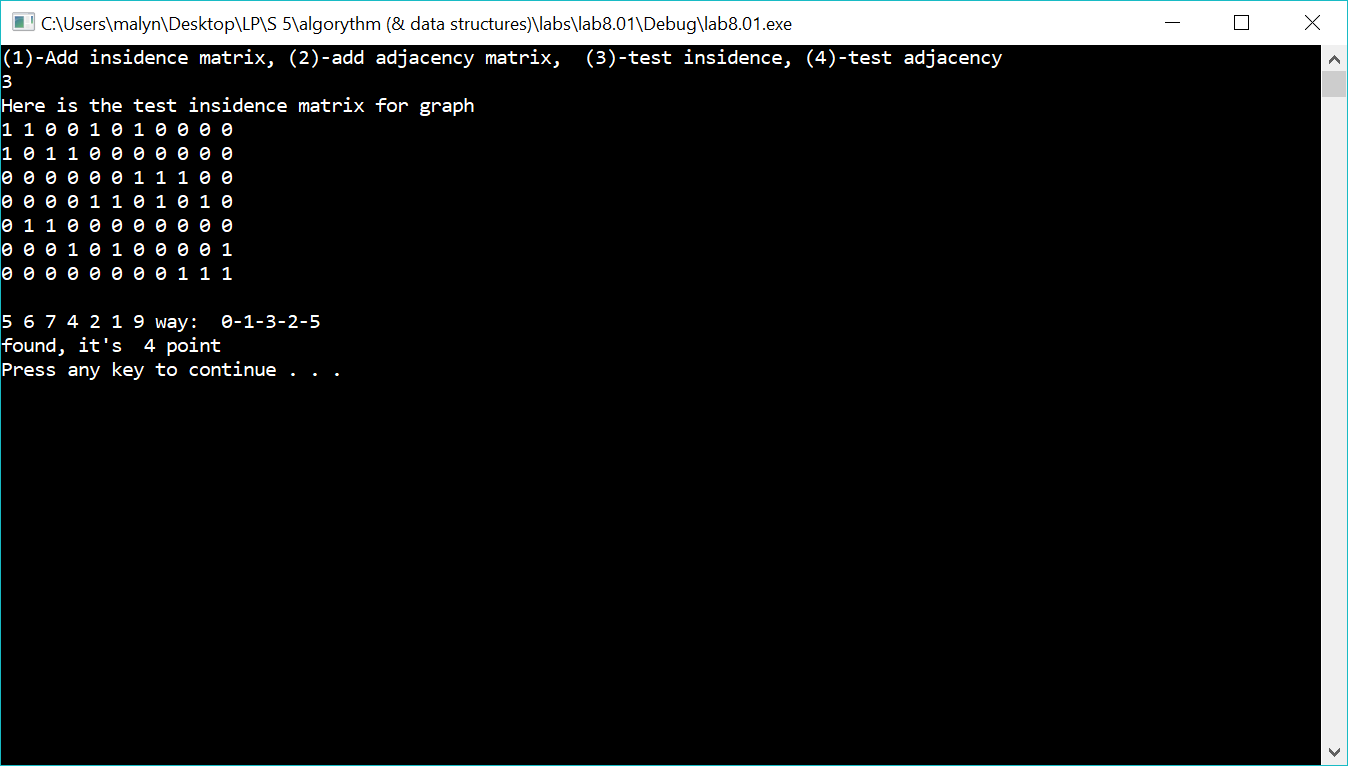
{

cout << "Illegal data for matrix\n";

}

system("pause");

}



Висновок

В результаті виконання цієї лабораторної роботи, я навчилася реалізовувати здійснювати обхід графа, зберігати і читати матриці інцидентності та суміжності, та шукати вершину за її значенням.