Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЁТ**

по лабораторной работе №6

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Поиск расстояний в графе»

Выполнили:

студенты группы 20ВВ2

Киреев Б.П.

Верховский М.В.

Лукин В.Д.

Приняли:

д.т.н. Митрохин М.А.

к.т.н. доцент Юрова О.В.

Пенза 2021

Название: Поиск расстояний в графе.

Цель работы: Научиться осуществлять поиск расстояний в графе.

Лабораторное задание:

**Задание 1:**

1. Сгенерируйте (используя генератор случайных чисел) матрицу

смежности для неориентированного графа G. Выведите матрицу на экран.

2. Для сгенерированного графа осуществите процедуру поиска

расстояний, реализованную в соответствии с приведенным выше описанием.

При реализации алгоритма в качестве очереди используйте класс queue из

стандартной библиотеки С++.

**Задание 1:**

Листинг:

Файл Lab6.cpp

#include <time.h>

#include <malloc.h>

#include <locale.h>

#include <iostream>

#include <queue>

#include <ctime>

#include <conio.h>

using namespace std;

int i, num = 0;

int\* vis = NULL, n, \*\* p = NULL;

int pogr = 0;

int Levo = 0, Control = 0;

//Вершина / След.элемент; Номер

struct SmegnElem {

SmegnElem\* Nextelem;

int num;

};

//Список смежности

struct SmegnList {

SmegnElem\* First;

int Colo;

} \*\*SpecMatrix = NULL, \* SpecMatrixRezerve = NULL;

//Создание списка смежности

SmegnList\* CreateList() {

SmegnList\* groups = (SmegnList\*)malloc(sizeof(SmegnList));

groups->First = NULL;

groups->Colo = 0;

return groups;

}

//Добавление элемента в список смежности

void AddSmegElem(SmegnList\* group, int Chis) {

SmegnElem\* newItem = (SmegnElem\*)malloc(sizeof(SmegnElem));

newItem->Nextelem = NULL;

newItem->num = Chis;

if (group->Colo == 0) { //При отсутствии элементов записываем как первый

group->First = newItem;

group->Colo++;

return;

}

SmegnElem\* last = group->First; //При наличии других элементов начинаем перебор

while (last->Nextelem != NULL) {

last = last->Nextelem;

}

last->Nextelem = newItem;

group->Colo++;

}

//Отрисовка списка смежности

void PrintSmegElem(SmegnList\* groups) {

if (groups->Colo == 0) { //При размере = 0, у нас нет данных

cout << "Список пуст" << endl;

return;

}

SmegnElem\* current = groups->First;

while (current != NULL) { //Перебор структуры до последнего элемента(с выводом)

int value = current->num;

cout << value << " ";

current = current->Nextelem;

}

cout << endl;

}

//Поиск уровней

void BFSD(int v, int\*\* p) {

queue <int> q;

q.push(v);

vis[v] = 0;

while (!q.empty()) {

v = q.front();

q.pop();

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

if (vis[i] == -1 && p[i][v] == 1) {

q.push(i);

vis[i] = vis[v] + 1;

}

}

}

cout << endl;

}

int main() {

setlocale(LC\_ALL, "Rus");

srand(time(0));

cout << "Введите размерность графа:";

cin >> n;

vis = (int\*)calloc(n, 3);

p = (int\*\*)calloc(n, 3);

SpecMatrix = (SmegnList\*\*)calloc(n, sizeof(SmegnList\*\*));

SpecMatrixRezerve = (SmegnList\*)calloc(n, sizeof(SmegnList\*));

SpecMatrixRezerve = CreateList();

cout.precision(3 \* n);

vis = (int\*)malloc(sizeof(int) \* n);

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

p[i] = (int\*)calloc(n, 3);

vis[i] = 0;

SpecMatrix[i] = CreateList(); //Создаём список смежности

AddSmegElem(SpecMatrix[i], i + 1); //Вставляем в начало каждого списка смежности номер отвечающей вершины

}

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

for (int m = 0 + pogr; m < n; m++) {

p[i][m] = 0 + rand() % 2;

p[m][i] = p[i][m];

p[i][i] = 0;

}

pogr++;

}

for (int i = 0; i < n; i++) { //Создание списков смежности из матрицы смежности

for (int m = 0; m < n; m++) { if (p[i][m] == 1) { AddSmegElem(SpecMatrix[i], m + 1); } }

}

cout << "\nМатрица 1:" << endl;

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

for (int m = 0; m < n; m++) { cout << p[i][m] << " "; }

cout << endl;

}

cout << "\nСписки смежности для Мартицы №1:" << endl; //Отрисовка списков смежности

for (int i = 0; i < n; i++) { PrintSmegElem(SpecMatrix[i]); }

cout << "Длина пути(По матрице / Очередь): ";

for (int i = 0; i < n; i++) { vis[i] = -1; } //Обнуление проходимых вершин

BFSD(0, p);

while (Control != n) {

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

if (vis[i] == Levo) {

cout << "Вершина " << i + 1 << " расположена на уровне " << Levo << endl;

Control++;

}

}

Levo++;

}

cout << endl;

return 0;

}

Пояснительный текст к программе:

Данная программа осуществляет поиск расстояний в графе. Работа алгоритма продолжается до тех пор, пока существуют непосещённые вершины.

Результаты работы программы:

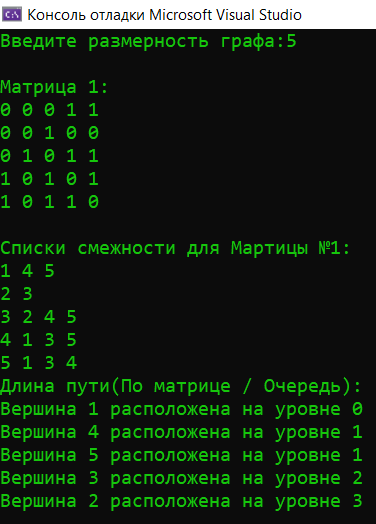


Рисунок №1 – Результат работы программы.

Вывод: При выполнении данной лабораторной работы мы смогли научиться осуществлять поиск расстояний в графе.