Министерство высшего образования и науки Российской Федерации

Пензенский Государственный Университет

Кафедра «Вычислительная техника»

Выполнили:

студенты группы 20ВВ2

Аверочкин П.С.

Зиновьев Я.М.

Принял:

к. т. н. доцент Юрова О.В.

д. т. н., профессор Митрохин М.А.

Пенза 2020

**ОТЧЕТ**

по лабораторной работе №4

по курсу «Л и ОА в ИС»

на тему «Обход графа в глубину»

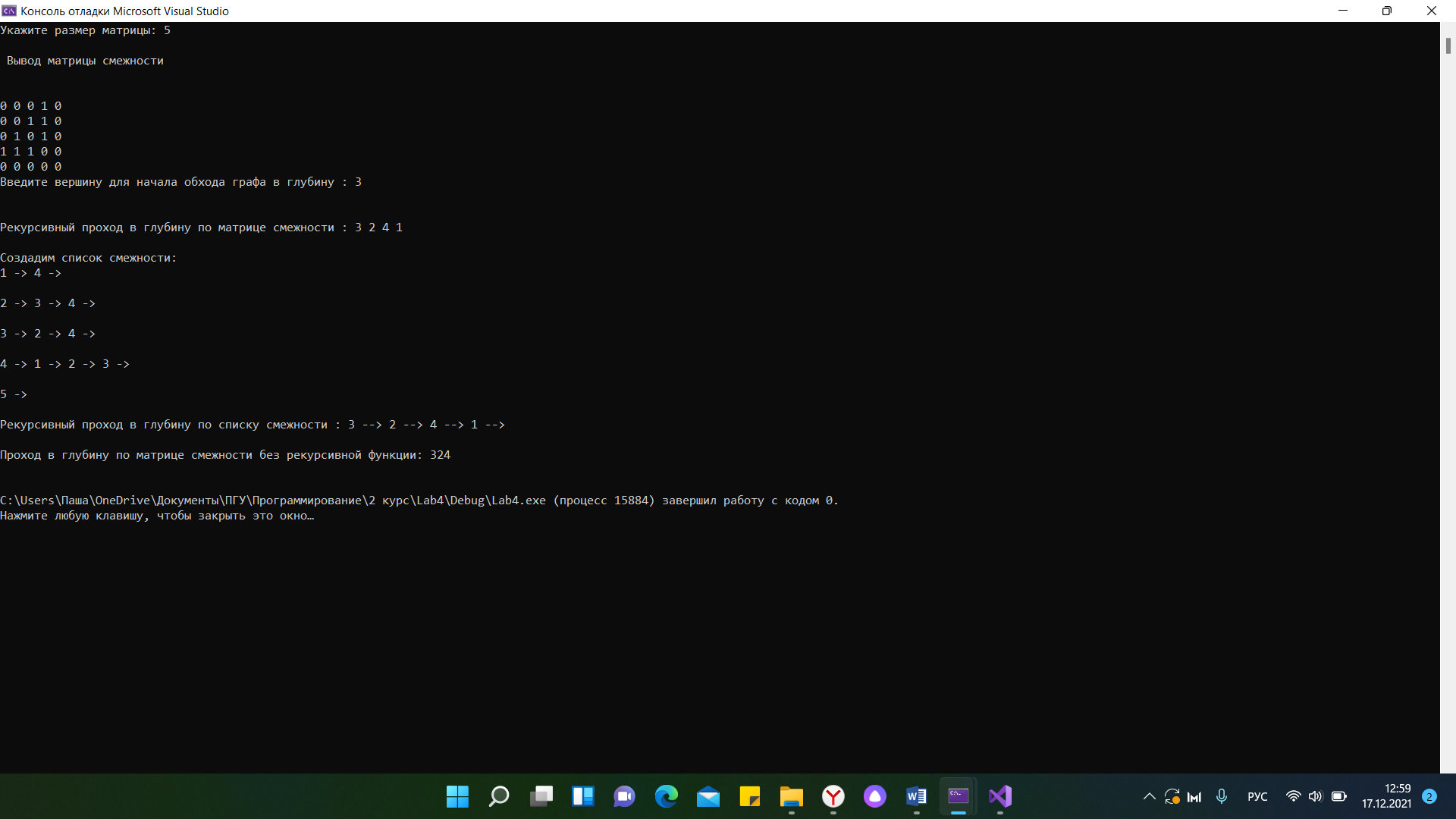
Ход работы:

Программа задает динамический двумерный массив, размерность которого задается с клавиатуры пользователем. Он в свою очередь описывает собой граф. Каждая единичка в массиве обозначает ребро между двумя смежными вершинами. С помощью функции rand() программа генерирует значения массива, состоящие из 0 или 1. Обязательным условием является наличие нулей по диагонали, что означает невозможность вершин быть смежными с самими с собой, т. е. отсутствие петель. Массив выводится на экран. Функция DFS является основной рабочей частью программы. Она совершает сам обход и счет пройденных вершин в графе.

Суть обхода: DFS работает рекурсивно, т. е. ссылается сама на себя по новым координатам. При обходе определенной вершины она ставит 1 в дополнительном массиве, отмечая тем самым, что она побывала здесь и дальнейший обход этой точки невозможен. В результате выводится номера вершин по очередности обхода программой.

Так же по матрице смежности строятся списки смежности (структура) и вместо обхода по массиву, аналог функции DFS обходит структуру.

Обход графа без рекурсии осуществляется за счет очереди.



Результат работы программы:

Рис 1 – Результат работы программы

Листинг:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <conio.h>

#include <iostream>

#include <locale.h>

#include <stack>

using namespace std;

typedef struct Node {

int vertex;

struct Node\* next;

} Node;

void add\_Node\_to\_beginning(Node\*\* head, int data) //функция добавления узла в начало

{

Node\* tmp\_node = (Node\*)malloc(sizeof(Node)); // создаем новый узел

tmp\_node->vertex = data;

tmp\_node->next = (\*head); //присваиваем указателю tmp адрес след. узла

(\*head) = tmp\_node; //Присваиваем указателю head адрес tmp

} //после выхода из функции tmp уничтожается

Node\* findLastHead(Node\* head) { //поиск адреса последнего элемента

if (head == NULL) {

return NULL;

}

while (head->next) {

head = head->next;

}

return head;

}

void add\_Node\_to\_end(Node\* head, int data) { //функция добавления нового узла в конец

Node\* last = findLastHead(head); //получаем указатель на последний элемент списка

Node\* tmp\_node = (Node\*)malloc(sizeof(Node)); // создаем новый узел

tmp\_node->vertex = data;

tmp\_node->next = NULL;

last->next = tmp\_node; //записываем в последний элемент списка указатель на новый узел

}

void createLinkedList(int\*\* a, Node\*\* head, int n) //создаем связный список

{

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

{

add\_Node\_to\_beginning(&head[i], i); //передаем адрес вершины и ее номер

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

{

if (a[i][j] == 1)

{

add\_Node\_to\_end(head[i], j); //добавляем к вершине связные с ней вершины

}

}

}

}

void printLinkedList(const Node\* head) {

cout << endl;

while (head) {

cout << head->vertex + 1 << " -> ";

head = head->next;

}

cout << endl;

}

void DFS(int\*\* a, bool\* visited, int n, int v) //поиск в глубину, v - текущая вершина, n - кол-во вершин, a - массив

{

cout << v + 1 << " ";

visited[v] = true; //помещаем текущюю вершину

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

{

if (a[v][i] == 1 && visited[i] == false) //проверяем смежные с ней вершины

{

DFS(a, visited, n, i); //вызываем рекурсию

}

}

}

void DFSlist(Node\*\* head, bool\* visited, int v)

{

cout << v + 1 << " --> ";

visited[v] = true;

Node\* tmp\_node = head[v];

while (tmp\_node) //идем по текущей вершине

{

if (visited[tmp\_node->vertex] == false)

DFSlist(head, visited, tmp\_node->vertex); //если вершина не посещена то передаем ее адрес и номер и идем дальше по ней

tmp\_node = tmp\_node->next;

}

}

void DFS\_no\_rec(int\*\* a, bool\* visited, int n, int v) { //обход графа без рекурсии

stack <int> stack;

stack.push(v);

while (!stack.empty())

{

if (visited[stack.top()] == false)

{

visited[stack.top()] = true;

cout << stack.top() + 1;

v = stack.top();

stack.pop();

for (int i = n - 1; i > 0; i--)

{

if (a[v][i] == 1 && visited[i] == false)

{

stack.push(i);

}

}

}

else

{

stack.pop();

}

}

}

void main()

{

setlocale(LC\_ALL, "RUS");

srand(time(NULL));

stack <int> stack;

int\*\* a, N, num; //N-кол-во вершин, num-вершина для ввода

cout << "Укажите размер матрицы: ";

cin >> N;

Node\*\* head = (Node\*\*)malloc(N \* sizeof(Node\*));

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

head[i] = NULL;

}

bool\* visited = (bool\*)malloc(N \* sizeof(bool));

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

{

visited[i] = false;

}

a = (int\*\*)malloc(N \* sizeof(int\*));

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

{

a[i] = (int\*)malloc(N \* sizeof(int));

}

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

{

a[i][i] = 0;

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

{

a[i][j] = rand() % 2;

a[j][i] = a[i][j];

}

}

cout << "\n Вывод матрицы смежности\n ";

cout << endl;

cout << "\t";

for (int i = 0; i < N; i++) { // вывод матриц смежности

cout << endl;

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

cout << a[i][j] << " ";

}

}

cout << "\nВведите вершину для начала обхода графа в глубину : ";

cin >> num;

num--;

cout << "\n\nРекурсивный проход в глубину по матрице смежности : ";

DFS(a, visited, N, num);

cout << endl;

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

visited[i] = false;

createLinkedList(a, head, N);

cout << "\nСоздадим список смежности: ";

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

printLinkedList(head[i]);

cout << "\nРекурсивный проход в глубину по списку смежности : ";

DFSlist(head, visited, num);

cout << endl;

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

visited[i] = false;

}

cout << "\nПроход в глубину по матрице смежности без рекурсивной функции: ";

DFS\_no\_rec(a, visited, N, num);

cout << endl << endl;

\_getch;

}

Вывод:

Мы научились реализовывать алгоритм обхода графа в глубину, по матрице смежности и спискам смежности и нерекурсивный обход.