Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №7

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Обход графа в глубину »

Выполнили:

студенты группы 21ВВ2

Сагателов А.К  
Митрошин А.Д

Приняли:

Митрохин М. А.

Юрова О. В.

Пенза 2022

**Название**

Обход графа в глубину.

**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 обхода в глубину, реализованную в соответствии с приведенным выше описанием.

**3.**\* Реализуйте процедуру обхода в глубину для графа, представленного списками смежности.

**Задание 2\***

1. Для матричной формы представления графов выполните преобразование рекурсивной реализации обхода графа к не рекурсивной.

**Листинг**

#include "stdafx.h"

#include <iostream>

#include <list>

#include <stack>

#include <time.h>

using namespace std;

struct node

{

int vertex;

struct node\* next;

};

struct Graph

{

int numVertices;

struct node\*\* adjLists;

};

struct node\* createNode(int v)

{

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

newNode->vertex = v;

newNode->next = NULL;

return newNode;

}

struct Graph\* createGraph(int vertices)

{

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

graph->numVertices = vertices;

graph->adjLists = (node\*\*)malloc(vertices \* sizeof(struct node\*));

int i;

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

graph->adjLists[i] = NULL;

return graph;

}

void addEdge(struct Graph\* graph, int src, int dest)

{

// добавление вершины в список элемента

struct node\* newNode = createNode(dest);

newNode->next = graph->adjLists[src];

graph->adjLists[src] = newNode;

}

void printGraph(struct Graph\* graph)

{

for (int v = 0; v < graph->numVertices; v++)

{

struct node\* temp = graph->adjLists[v];

printf("\nVertex %d: ", v);

while (temp)

{

printf("%d ", temp->vertex);

temp = temp->next;

}

}

}

void toString(int\*\* Matrix, int n) {

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

cout << i << " : ";

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

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

}

cout << endl;

}

}

int\*\* Matrix;

bool\* visited;

bool\* v;

bool\* visit;

int numVertices=7;

void DFS\_M(int start) //обход рекурсией

{

visited[start] = true;

cout << start << " ";

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

if ((Matrix[start][i] != 0) && (!visited[i]))

DFS\_M(i);

}

void DFS\_S(struct Graph\* graph, int start) //обход без рекурсии

{

struct node\* adjList = graph->adjLists[start];

struct node\* temp = adjList;

visit[start] = 1;

printf("%d ", start);

while (temp != NULL) { //если вершина != 0, переходим дальше

int connectedVertex = temp->vertex;

if (!visit[connectedVertex]) {

DFS\_S(graph, connectedVertex);

}

temp = temp->next;

}

}

void DFS(int start) { //обход для списка смежности

stack<int> S;

int t;

S.push(start);

v[start]++;

while (!S.empty())

{

t = S.top();

S.pop();

cout << t << " ";

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

if (Matrix[t][i] && !v[i])

{

S.push(i);

v[i]++;

}

}

}

int main()

{

setlocale(0, "");

srand(time(0));

struct Graph\* graph = createGraph(numVertices);

Matrix = (int\*\*)malloc(sizeof(int\*) \* numVertices);

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

Matrix[i] = (int\*)malloc(sizeof(int\*) \* numVertices);

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

{

Matrix[i][j]=0; // создание матрицы смежности

}

}

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

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

if (0 < rand() % 2)

Matrix[i][j] = Matrix[j][i] = 1, addEdge(graph, i, j), addEdge(graph, j, i); // связи вершин графа

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

visit = (bool\*)malloc(numVertices \* sizeof(bool));

v = (bool\*)malloc(numVertices \* sizeof(bool));

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

visited[i] = false, visit[i] = false, v[i] = false;

printf\_s("Matrix: \n");

toString(Matrix, numVertices);

printf\_s("\nList: ");

printGraph(graph);

int start;

printf\_s("\n\nEnter strating vertex: "); scanf\_s("%d", &start);

printf\_s("\nPoradok obhoda: \n");

DFS\_M(start);

printf\_s("\nPoradok obhoda: \n");

DFS\_S(graph,start);

printf\_s("\nPoradok obhoda: \n");

DFS(start);

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

free(Matrix[i]);

free(Matrix);

free(visited);

free(visit);

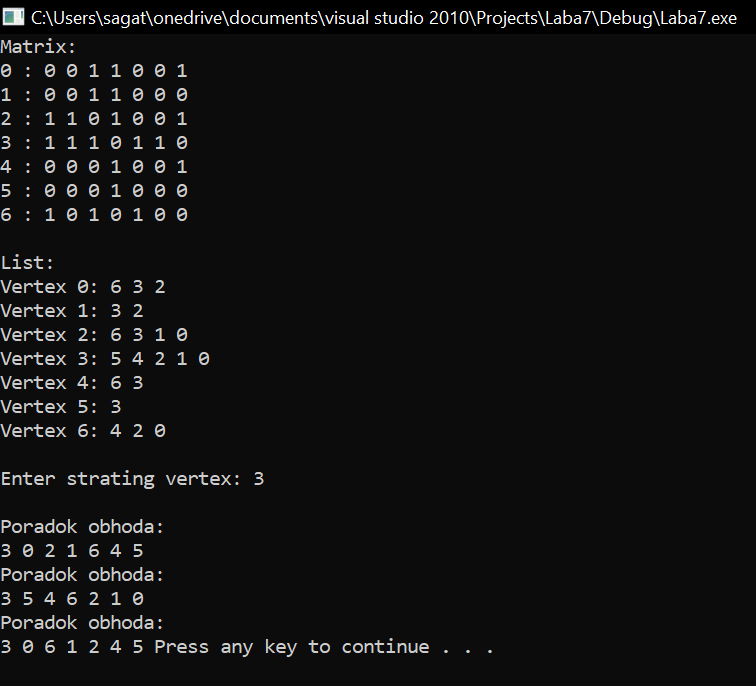
free(v);

system ("pause");

return 0;

}

**Результат работы программы**



### Выводы

В данной лабораторной работе мы научились осуществлять процедуру обхода в глубину. Реализовывать процедуру обхода в глубину для графа, представленного списками смежности. Для матричной формы представления графов выполнили преобразование рекурсивной реализации обхода графа к не рекурсивной.