Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №4

# по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Обход графа в глубину»

**Выполнили:**

студенты группы 20ВВ3

Пантюшов Егор

Шмелёв Данила

**Приняли:**

Юрова О.В.

Митрохин М.А.

Пенза 2021

# Название

Обход графа в глубину

**Цель работы** – изучение алгоритма обхода графа в глубину.

# Методические указания

Обход графа – одна из наиболее распространенных операций с

графами. Задачей обхода является прохождение всех вершин в графе.

Обходы применяются для поиска информации, хранящейся в узлах графа,

нахождения связей между вершинами или группами вершин и т.д.

# Лабораторное задание

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.

**Псевдокод**

Вход: G – матрица смежности графа.

Выход: номера вершин в порядке их прохождения на экране.

Алгоритм ПОГ

1.1. для всех i положим NUM[i] = False пометим как &quot;не посещенную&quot;;

1.2. ПОКА существует &quot;новая&quot; вершина v

1.3. ВЫПОЛНЯТЬ DFS (v).

Алгоритм DFS(v):

2.1. пометить v как &quot;посещенную&quot; NUM[v] = True;

2.2. вывести на экран v;

2.3. ДЛЯ i = 1 ДО size\_G ВЫПОЛНЯТЬ

2.4. ЕСЛИ G(v,i) = = 1И NUM[i] = = False

2.5. ТО

2.6. {

2.7. DFS(i);

2.8. }

# Листинг

#define \_CRT\_NONSTDC\_NO\_WARNINGS

#define \_CRT\_SECURE\_NO\_WARNINGS

#define HEADER ("Лабораторная работа №4\nВыполнили: Шмелёв Д. и Пантюшов Е.\n")

#include <stdio.h>

#include <locale.h>

#include <stdlib.h>

#include <time.h>

#include <stdbool.h>

struct node

{

int vertex;

struct node\* next;

};

struct Graph

{

int numVertices;

struct node\*\* adjList;

};

struct node\* createNode(int v)

{

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

newNode->vertex = v;

newNode->next = NULL;

return newNode;

}

struct Graph\* createGraph(int vertices)

{

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

graph->numVertices = vertices;

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

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

graph->adjList[i] = NULL;

return graph;

}

void addEdge(struct Graph\* graph, int i, int j)

{

struct node\* newNode = createNode(j);

if (graph->adjList[i] == false)

{

graph->adjList[i] = newNode;

newNode = NULL;

}

struct node\* temp = graph->adjList[i];

while (temp->next)

{

temp = temp->next;

}

temp->next = newNode;

newNode = createNode(i);

if (graph->adjList[j] == false)

{

graph->adjList[j] = newNode;

newNode = NULL;

}

temp = graph->adjList[j];

while (temp->next)

{

temp = temp->next;

}

temp->next = newNode;

}

void printGraph(struct Graph\* graph)

{

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

{

struct node\* temp = graph->adjList[i];

printf("%d-я вершина: ", i);

while (temp)

{

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

temp = temp->next;

}

printf("\n");

}

}

struct stack

{

int\* elem;

int top;

};

void init(struct stack\* stk, int n)

{

stk->elem = (int\*)malloc(n \* sizeof(int));

stk->top = 0;

}

void push(struct stack\* stk, int f, int n)

{

if (stk->top < n)

{

stk->elem[stk->top] = f;

stk->top++;

}

else

printf("Стек полон, количество элементов: %d !\n", stk->top);

}

int pop(struct stack\* stk)

{

int elem;

if ((stk->top) > 0)

{

stk->top--;

elem = stk->elem[stk->top];

return elem;

}

else

{

printf("Стек пуст!\n");

return 0;

}

}

int isempty(struct stack\* stk)

{

if (stk->top == 0)

return 1;

else

return 0;

}

int stkTop(struct stack\* stk)

{

if ((stk->top) > 0)

{

return stk->elem[stk->top - 1];

}

else {

printf("Стек пуст!\n");

return 0;

}

}

void dfs(int\*\* G, int \*NUM, int v, int n)

{

NUM[v] = true;

printf(" %d ", v);

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

{

if ((G[v][i] == 1) && (NUM[i] == false))

{

dfs(G, NUM, i, n);

}

}

}

void dfs\_list(struct Graph\* graph, int\* NUM, int v, int n)

{

struct node\* list = graph->adjList[v];

NUM[v] = true;

printf(" %d ", v);

while (list)

{

if (NUM[list->vertex] == false)

{

dfs\_list(graph, NUM, list->vertex, n);

}

list = list->next;

}

}

void task\_1\_1(int\*\* G, int n)

{

printf("\nЗадание 1.\n\nПункт 1.\nМатрица смежности:\n");

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

{

G[i] = (int\*)malloc(n \* sizeof(int));

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

{

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

}

}

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

{

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

{

if (i == j)

{

G[i][j] = 0;

}

else

{

G[i][j] = G[j][i];

}

printf(" %d ", G[i][j]);

}

printf("\n");

}

}

void task\_1\_2(int\*\* G, int n)

{

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

int v;

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

{

NUM[i] = false;

}

printf("\nПункт 2.\nВведите точку входа: ");

scanf("%d", &v);

printf("Результат работы алгоритма обхода в глубину(матрица): ");

dfs(G, NUM, v, n);

free(NUM);

}

void task\_1\_3(struct Graph\* graph, int n, int\*\* G)

{

struct node\* node = graph->adjList;

int v;

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

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

{

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

{

if ((G[i][j] == 1) && (i < j))

{

addEdge(graph, i, j);

}

}

}

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

{

NUM[i] = false;

}

printf("\n\nПункт 3.\nСписок смежности: \n");

printGraph(graph);

printf("Введите точку входа: ");

scanf("%d", &v);

printf("Результат работы алгоритма обхода в глубину(список): ");

dfs\_list(graph, NUM, v, n);

printf("\n\n");

free(NUM);

}

void task\_2\_1(int\*\* G, int n)

{

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

struct stack\* Stack = (struct stack\*)malloc(sizeof(struct stack));

printf("Задание 2.\n\nПункт 1.\n\nВведите точку входа: ");

int v;

scanf("%d", &v);

printf("Результат работы алгоритма обхода в глубину(матрица)(без рекурсии): ");

init(Stack, n);

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

{

NUM[i] = false;

}

push(Stack, v, n);

while (isempty(Stack) == 0)

{

int v = stkTop(Stack);

pop(Stack);

if (NUM[v] == true)

{

continue;

}

NUM[v] = true;

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

{

if ((G[v][j] == 1) && (NUM[j] != true))

{

push(Stack, j, n);

NUM[j] = true;

}

}

printf(" %d ", v);

}

printf("\n");

free(NUM);

}

int main()

{

setlocale(LC\_ALL, "RUS");

srand(time(NULL));

printf(HEADER);

printf("Введите размерность матрицы: ");

int n;

scanf("%d", &n);

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

struct Graph\* graph = createGraph(n);

task\_1\_1(G, n);

task\_1\_2(G, n);

task\_1\_3(graph, n, G);

task\_2\_1(G, n);

free(G);

system("PAUSE");

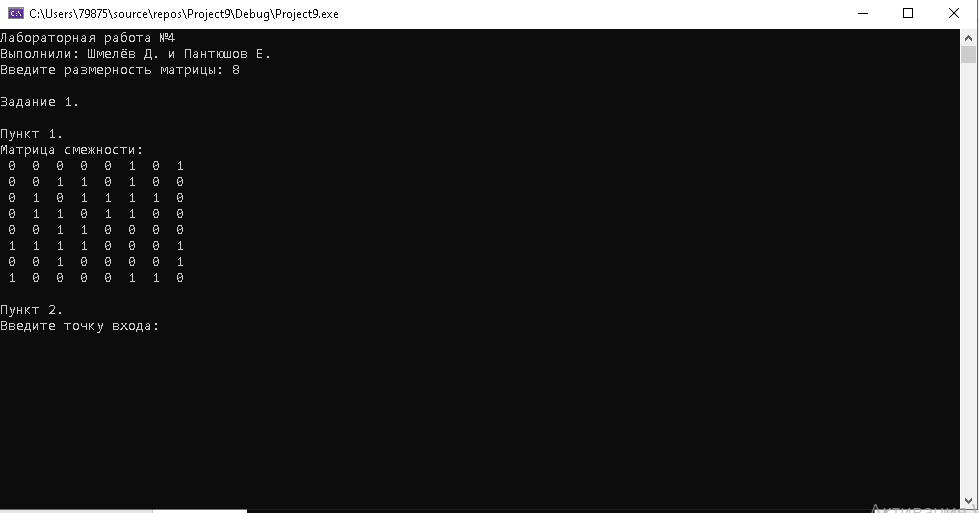
return 0;

}

# Результат работы программы

**Пункт 1.**

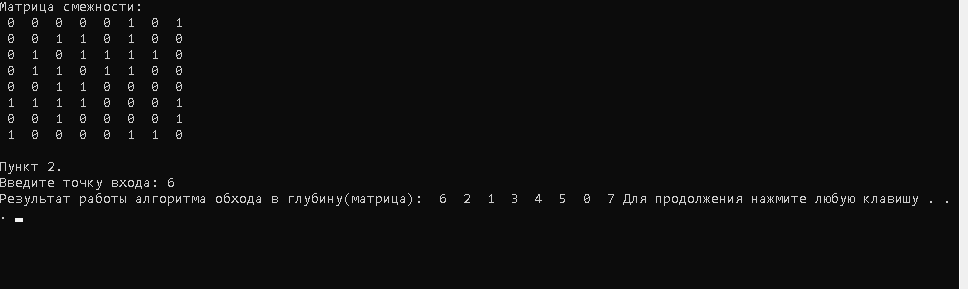
Результаты работы программы показаны на рисунке 1.



**Рисунок 1 – Результат работы программы**

**Пункт 2.**

Результаты работы программы показаны на рисунке 2.



**Рисунок 2 – Результат работы программы**

# Вывод

В ходе выполнения лабораторной работы была разработана программа, в который был реализован алгоритм обхода графа в глубину.