Пензенский государственный университет

Кафедра «Вычислительная техника»

ОТЧЕТ

По лабораторной работе №7

по дисциплине: "Логика и основы алгоритмизации в инженерных задачах"

## на тему:"Обход графа в глубину"

Выполнили студенты гр. 22ВВП2:

Корнилов В.М.

Самофалова А.В.

Горбунов Д.А.

Приняли:

Акифьев И.В.

Юрова О.В.

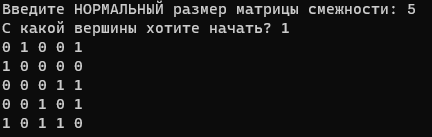
Пенза,2023

**Цель: Обойти граф в глубину.**

**Задание 1:**

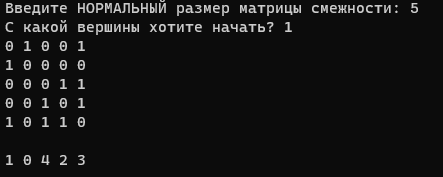
1. Сгенерировать (используя генератор случайных чисел) матрицу

смежности для неориентированного графа G. Вывести матрицу на экран.



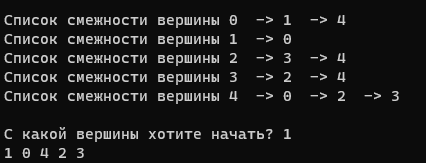
1. Для сгенерированного графа осуществить процедуру обхода в

глубину.



3.\* Реализовать процедуру обхода в глубину для графа, представленного

списками смежности.



**Задание 2**\*

1. Для матричной формы представления графов выполнить

преобразование рекурсивной реализации обхода графа к не рекурсивной.



**Вывод:** научились реализовывать обход в графа в матричном виде и графа, представленного списками смежности в глубину, обход графа в глубину без использования рекурсивного вызова функции.

**Приложение А (Листинг)**

**Файл 7.с**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

#include <malloc.h>

#include <stdbool.h>

typedef struct Node {

int ver;

struct Node\* next;

} Node;

int n, v;

int\* vis;

int\* vis1;

int\*\* A;

int\* stack;

bool\* vis2;

void DFS(int\*\* A, int\* vis, int n, int v) {

vis[v] = 1;

printf("%d ", v);

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

if (A[v][i] == 1 && vis[i] == 0)

DFS(A, vis, n, i);

}

}

void DFS2(Node\*\* List, int\* vis, int ver) {

vis[ver] = 1;

printf("%d ", ver);

struct Node\* temp = List[ver];

while (temp != NULL) {

int cv = temp->ver;

if (!vis[cv]) {

DFS2(List, vis, cv);

}

temp = temp->next;

}

}

void DFSnotRecursive(int\*\* A, int n, int v, bool\* vis2, int\* stack) {

int top = -1;

int g;

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

vis2[i] = false;

}

stack[++top] = v;

vis2[v] = true;

while (top != -1) {

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

if (A[g][i] == 1 && vis2[i] == false) {

g = stack[top--];

printf("%d ", g);

vis2[i] = true;

stack[++top] = i;

break;

}

}

}

printf("\n");

}

void addver(Node\*\* List, int x, int y) {

// Создание нового узла

Node\* newNode = (Node\*)malloc(sizeof(Node));

newNode->ver = y;

newNode->next = NULL;

// Проверка, существует ли уже ребро в списке

if (List[x] == NULL) {

List[x] = newNode;

}

else {

Node\* temp = List[x];

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

}

}

// Функция отображения списков смежности

void vivod(Node\*\* List, int n) {

int i;

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

Node\* temp = List[i];

printf("Список смежности вершины %d ", i);

while (temp) {

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

temp = temp->next;

}

printf("\n");

}

}

int main() {

setlocale(LC\_ALL, "RUSSIAN");

printf("Введите НОРМАЛЬНЫЙ размер матрицы смежности: ");

scanf("%d", &n);

printf("С какой вершины хотите начать? ");

scanf("%d", &v);

int count = 0;

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

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

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

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

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

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

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

A[j][i] = A[i][j];

if (i == j) A[i][j] = 0;

if (A[i][j] == 1) count++;

}

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

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

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

}

printf("\n");

}

printf("\n");

DFS(A, vis, n, v);

// Создание списков смежности

Node\*\* list1 = (Node\*\*)malloc(n \* sizeof(Node\*));

int i, j;

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

list1[i] = NULL;

}

// Заполнение списков смежности

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

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

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

addver(list1, i, j);

}

}

}

// Отображение списков смежности

printf("\n\n");

vivod(list1, n);

vis1 = (int\*)calloc(n, sizeof(int));

printf("\n");

printf("С какой вершины хотите начать? ");

scanf("%d", &v);

DFS2(list1, vis1, v);

printf("\n");

printf("С какой вершины начинать не рекурсивный обход? ");

scanf("%d", &v);

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

vis2 = (int\*)malloc(sizeof(bool)\*n);

DFSnotRecursive(A, n, v, vis2, stack);

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

free(A[i]);

free(A);

free(vis);

free(vis1);

free(stack);

free(vis2);

return 0;

}