Пензенский государственный университет  
Кафедра «Вычислительной техники»

**Отчет**по лабораторной работе №4  
по дисциплине: «Логика и основы алгоритмизации в инженерных задачах»  
на тему: «Обход графа в глубину»

**Выполнили студенты группы 19ВВ1:**

Тебнев Р.  
Балалаев А.

**Приняли:**

д.т.н. профессор Митрохин М. А.

Юрова О.В.

Пенза 2020

**Цель работы:** изучить обход в глубину, научится реализовывать его через рекурсию.

**Теория**

Большое число задач, связанных с графами требует перебора вершин графа,

т.е. просмотра каждой вершины в точности один раз (задачи поиска).

**Обход в глубину** осуществляется с некоторой вершины вдоль ребер графа, до

попадания в лист. После этого нужно возвращаться назад вдоль пройденного пути, пока

не будет обнаружена вершина, у которой есть еще не посещенная смежная вершина, и

затем двигаться в направлении не посещённой вершины. Эти действия повторяются до

возврата в начальную вершину после посещения всех остальных.

Т.о. основная идея поиска в глубину – сначала полностью исследовать одну ветку вглубь

и только потом переходить к другим веткам.

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. }

**Код программы**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <time.h>

#include <locale.h>

#define N 5

void newMat(int\* mas) {

srand(time(0));

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

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

if (i == j) {

mas[N \* i + j] = 0;

}

if (i < j) {

mas[N \* i + j] = rand() % 2;

mas[j \* N + i] = mas[N \* i + j];

}

}

}

}

void printMat(int\* mas) {

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

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

printf("%d ", mas[N \* i + j]);

}

printf("\n");

}

}

void DFS(int s, int\* G, int\* num) {

num[s] = 1;

printf("%d ", s);

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

if (G[N \* s + i] == 1 && num[i] == 0) {

DFS(i, G, &num[0]);

}

}

}

struct stack {

int elem[N];

int top;

};

void init(struct stack\* stk) {

stk->top = 0;

}

void push(struct stack\* stk, int k) {

if (stk->top < N) {

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

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!");

return 0;

}

}

void DFSstack(struct stack\* stk, int s, int\* G, int\* num) {

int i = 0;

num[s] = 1;

printf("%d ", s);

push(stk, s);

push(stk, s);

while (stk->top != 0) {

if (i == N) s = pop(stk);

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

if (G[N \* s + i] == 1 && num[i] == 0) {

num[i] = 1;

printf("%d ", i);

if (stk->top != N)

push(stk, i);

s = i;

break;

}

}

}

}

typedef struct Spisok {

int value;

struct Spisok\* next;

}smej;

smej\* create(int num) {

// Выделение памяти под корень списка

smej\* tmp = new smej;

// Присваивание имя вершине

tmp->value = num;

// Присваивание указателю на следующий элемент значения NULL

tmp->next = NULL;

return(tmp);

}

void addElem(int data, smej\* head) {

// Выделение памяти под корень списка

smej\* tmp = new smej;

// Присваивание значения узлу

tmp->value = data;

// Присваивание указателю на следующий элемент значения NULL

tmp->next = NULL;

// Присваивание новому указателю указателя head.

// Присваивание выполняется для того, чтобы не потерять указатель на «голову» списка

smej\* p = head;

// Сдвиг указателя p в самый конец первоначального списка

while (p->next != NULL)

p = p->next;

// Присваивание указателю p -> next значения указателя tmp (созданный новый узел)

p->next = tmp;

}

void print(smej\* head) {

smej\* s = head;

if (s != NULL) { //номер вершины

printf("%d: ", s->value);

s = s->next;

}

while (s != NULL) //номера узлов

{

// Вывод значения узла

printf("%d ", s->value);

// Сдвиг указателя к следующему узлу

s = s->next;

}

}

void spisokSmej(int(&Matrix)[N][N], smej\* G\_S[N]) {

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

Spisok\* tmp = create(i);

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

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

addElem(j, tmp);

}

G\_S[i] = tmp;

}

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

print(G\_S[i]);

printf("\n");

}

printf("\n");

}

void DFSsp(smej\* (&P)[N], int ver, int\* num) {

num[ver] = 1;

printf("%d ", ver);

smej\* s = P[ver];

s = s->next;

while (s != NULL) {

if (num[s->value] == 0) {

DFSsp(P, s->value, &num[0]);

}

s = s->next;

}

}

//void SpisokSmej(spisok\*\* create, int name, int(&Matrix)[N][N], spisok\* G\_S[N], int data) {

// spisok\* head;

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

// (\*create) = new spisok;

// (\*create)->value = name;

// (\*create)->next = NULL;

//

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

// if (Matrix[i][j] == 1) {

// (\*create) = new spisok;

// (\*create)->value = data;

// (\*create)->next = NULL;

//

// spisok\* p = head;

// while (p->next != NULL) {

// p = p->next;

// p->next = \*create;

// }

// }

// }

// G\_S[i] = \*create;

// }

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

// print(G\_S[i]);

// printf("\n");

// }

// printf("\n");

//}

int main() {

int G[N][N], num[N], s;

setlocale(LC\_ALL, "rus");

struct stack\* stk;

stk = new stack;

Spisok\* G\_S[N];

newMat(&G[0][0]);

printMat(&G[0][0]);

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

num[i] = 0;

}

printf("\nВведите номер вершины для начала обхода: ");

scanf("%d", &s);

printf("Результат с рекурсией: ");

DFS(s, &G[0][0], &num[0]);

printf("\n\n");

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

num[i] = 0;

}

printf("Представление графа в виде списка смежности: \n");

spisokSmej(G, G\_S);

printf("Введите номер вершины для начала обхода: ");

scanf("%d", &s);

DFSsp(G\_S, s, &num[0]);

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

num[i] = 0;

}

init(stk);

printf("\n\nВведите номер вершины для начала обхода: ");

scanf("%d", &s);

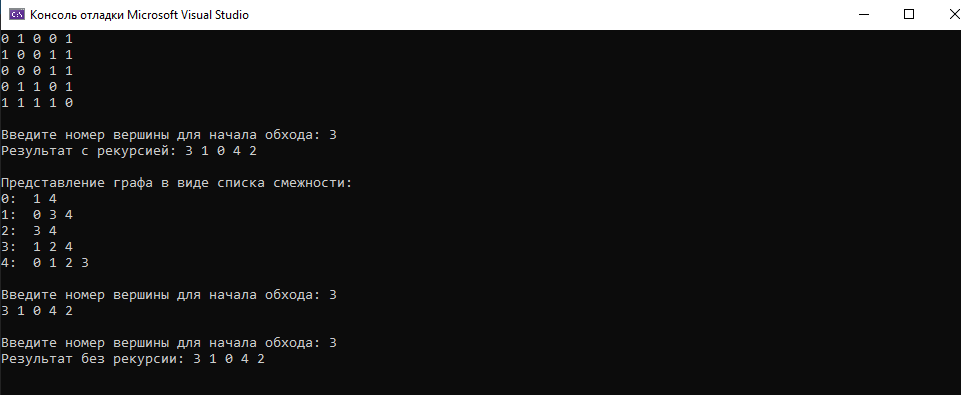
printf("Результат без рекурсии: ");

DFSstack(stk, s, &G[0][0], &num[0]);

printf("\n\n");

}

**Результат работы программы**



**Вывод:** изучили обход графа в глубину; реализовали его рекурсивно и нерекурсивно; научились представлять граф в виде списка смежности и реализовывать его обход рукурсивно.