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»

**Выполнил студент группы 19ВВ1:**

Артемов К.А.

**Приняли:**

д.т.н. профессор Митрохин М. А.

к.т.н. Юрова О.В.

Пенза 2020.

**Цель работы:** изучить обход в глубину, научится реализовывать его как через рекурсию, так и без неё, научится применять обход в глубину на списках.

**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. }

**Стек** – это структура данных, в которой элементы поддерживают принцип LIFO (“Last in – first out”): пос

**PUSH** – кладёт элемент на вершину стека

**POP** – снимает элемент с вершины стека, перемещая вершину к следующему элементу

ледним зашёл – первым вышел. Или первым зашёл – последним вышел.

**Структура** — это объединение нескольких объектов, возможно, различного типа под одним именем, которое является типом структуры. В качестве объектов могут выступать переменные, массивы, указатели и другие структуры.  
  
Структуры позволяют трактовать группу связанных между собой объектов не как множество отдельных элементов, а как единое целое. Структура представляет собой сложный тип данных, составленный из простых типов.

**Односвязный список** – структура данных, в которой каждый элемент (узел) хранит информацию, а также ссылку на следующий элемент. Последний элемент списка ссылается на NULL.

**Код программы**

#define \_CRT\_SECURE\_NO\_WARNINGS

#define SIZE 5

#include <time.h>

#include <locale.h>

#include <iostream>

void rand\_Zap(int\* Matrix, int n) {

srand(time(NULL));

printf("G\n");

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

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

if (i == j) {

Matrix[i \* n + j] = 0;

}

if (i < j) {

Matrix[i \* n + j] = rand() % 2;

Matrix[j \* n + i] = Matrix[i \* n + j];

}

}

}

void print\_G(int\* Matrix, int n) {

printf(" ");

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

printf("%3d", i + 1);

printf("\n\n");

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

printf("%2d", i + 1);

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

printf("%3d", Matrix[i \* n + j]);

}

printf("\n");

}

printf("\n");

}

void DFS(int v,int\* G, int n, int\* num) {

num[v] = 1;

printf("%d\t",v+1);

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

if (G[v \* n + i] == 1 && num[i] == 0)

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

}

}

struct stack {

int elem[SIZE];

int top;

};

void init(struct stack\* stk) {

stk->top = 0;

}

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

if (stk->top < SIZE) {

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

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;

}

}

void DFSonStack(struct stack\* stk, int v, int\* G, int n, int\* num) {

int i=0;

num[v] = 1;

printf("%d\t", v + 1);

push(stk, v);

push(stk, v);

while (stk->top != 0 ) {

if (i == n) { v = pop(stk); }

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

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

num[i] = 1;

printf("%d\t", i + 1);

if(stk->top != SIZE)

push(stk, i);

v = i;

break;

}

}

}

}

typedef struct Node {

int value;

struct Node\* next;

}Spisok;

Spisok\* create\_V(int name)

{

// Выделение памяти под корень списка

Spisok\* tmp = (Spisok\*)malloc(sizeof(Spisok));

// Присваивание имя вершине

tmp->value = name;

// Присваивание указателю на следующий элемент значения NULL

tmp->next = NULL;

return(tmp);

}

void add\_element\_end(int data, Spisok\* head)

{

// Выделение памяти под корень списка

Spisok\* tmp = (Spisok\*)malloc(sizeof(Spisok));

// 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

Spisok\* 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(Spisok\* head) {

Spisok\* v = head;

if (v != NULL) {

printf("%d: ", v->value + 1);

v = v->next;

}

while (v != NULL)

{

// Вывод значения узла

printf("%d ", v->value + 1);

// Сдвиг указателя к следующему узлу

v = v->next;

}

}

void graf\_iz\_matrix(int(&Matrix)[SIZE][SIZE],Spisok\* G\_S[SIZE]) {

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

Node\* tmp = create\_V(i);

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

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

add\_element\_end(j, tmp);

}

G\_S[i] = tmp;

}

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

print(G\_S[i]);

printf("\n");

}

printf("\n");

}

void DFSsp(Spisok\* (&P)[SIZE], int ver,int\* num) {

num[ver] = 1;

printf("%d\t", ver + 1);

Spisok\* v = P[ver];

v = v->next;

while (v != NULL) {

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

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

}

v = v->next;

}

}

int main()

{

setlocale(LC\_ALL, "Rus");

Spisok\* G\_S1[SIZE];

struct stack\* stk;

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

init(stk);

int G[SIZE][SIZE], num[SIZE], v,i=0;

rand\_Zap(&G[0][0], SIZE);

print\_G(&G[0][0], SIZE);

/\*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\*/

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

num[i] = 0;

}

printf(" введите вершину для начала обхода\n");

scanf("%d", &v);

while ((v < 1) || (v > SIZE)) {

printf("Ошибка!!! введите занаво\n");

scanf("%d", &v);

}

printf("\n");

v--;

printf(" Результат с рекурсией\n");

DFS(v, &G[0][0], SIZE, &num[0]);

printf("\n\n");

/\*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\*/

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

num[i] = 0;

}

printf(" введите вершину для начала обхода\n");

scanf("%d", &v);

while ((v < 1) || (v > SIZE)) {

printf("Ошибка!!! введите занаво\n");

scanf("%d", &v);

}

printf("\n");

v--;

printf(" Результат без рекурсии\n");

DFSonStack(stk, v, &G[0][0], SIZE, &num[0]);

printf("\n\n");

/\*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\*/

printf("\n представление графа в виде списка\n");

graf\_iz\_matrix(G, G\_S1);

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

num[i] = 0;

}

printf(" введите вершину для начала обхода\n");

scanf("%d", &v);

while ((v < 1) || (v > SIZE)) {

printf("Ошибка!!! введите занаво\n");

scanf("%d", &v);

}

printf("\n");

v--;

printf(" Результат обхода списков\n");

DFSsp(G\_S1,v, &num[0]);

}

**Результат работы кода**

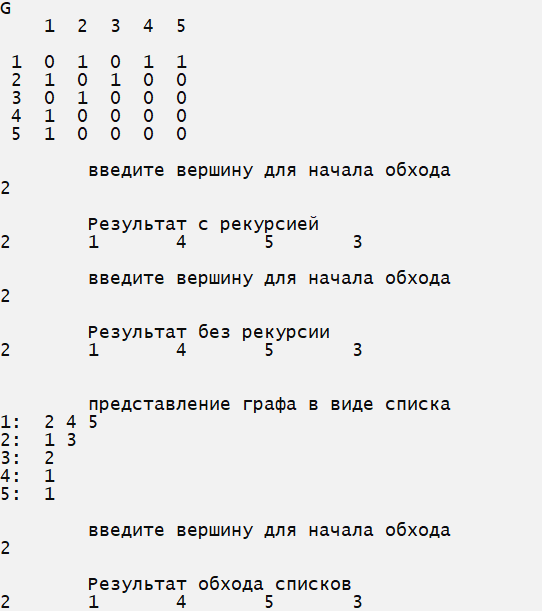


Рис.1 Результат работы программы

**Вывод:** изучил обход в глубину, научился его реализовывать, вспомнил организацию стека и создание односвязных списков.