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

Баулин Александр

Культиясов Ярослав

Духнов Олег

**Приняли:**

Юрова О.В.

Митрохин М.А.

Пенза 2021

# Название

Обход графа в глубину

**Цель работы**–изучение функций, предназначенных для обхода 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\*

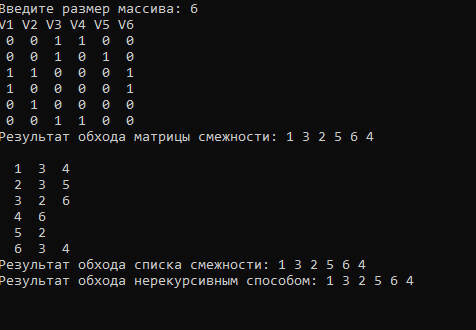
1. Для матричной формы представления графов выполните

преобразование рекурсивной реализации обхода графа к не рекурсивной.

**Теоретические сведения**

Одним из способов обхода графов является поиск в глубину. Идея такого обхода состоит в том, чтобы начав обход из какой-либо вершины всегда переходить по первой встречающейся в процессе обхода связи в следующую вершину, пока существует такая возможность.

# Результат работы программы



**Рисунок 1 – Результат работы программы**

**Листинг**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

#include <stdbool.h>

#include <conio.h>

#include <time.h>

#include <stack>

using namespace std;

typedef struct Node {

int data;

struct Node\* next;

}Node;

Node\* init(int num)

{

Node\* list;

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

list->data = num;

list->next = NULL;

return list;

}

Node\* add(Node\* list, int num)

{

Node\* p, \* temp;

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

p = list->next;

list->next = temp;

temp->data = num;

temp->next = p;

return temp;

}

int\*\* create\_arr(int n)

{

int s = 1;

int \*\*Arr;

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

srand(time(NULL));

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

{

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

Arr[i][i] = 0;

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

{

if (rand() % 100 > 60)

{

Arr[i][j] = 0;

}

else {

Arr[i][j] = 1;

}

}

s++;

}

s = 1;

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

{

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

Arr[j][i] = Arr[i][j];

s++;

}

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

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

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

{

printf("\n");

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

{

printf("%2d ", Arr[i][j]);

}

}

return Arr;

}

void DFS(int\*\* Arr, int n, bool\* Arr2, int i)

{

Arr2[i] = true;

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

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

{

if (Arr[i][j] == 1 && Arr2[j] == false)

DFS(Arr, n, Arr2, j);

}

}

void Free(int\*\* Arr, bool\* Arr2, int n, Node\*\* sp1)

{

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

free(Arr[i]);

free(Arr);

free(Arr2);

}

void fun\_init(Node\*\* sp1, int n)

{

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

sp1[i] = init(i);

}

void spisok(int n, int\*\* Arr, Node\* list, int i)

{

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

{

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

add(list, j);

}

}

void fun\_spisok(Node\*\* sp1, int n, int\*\* Arr)

{

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

spisok(n, Arr, sp1[i], i);

}

void print\_spisok(Node\*\* sp1, int n)

{

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

{

Node\* adr = sp1[i];

printf("\n");

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

{

if (sp1[i] != NULL)

{

printf("%\*d", 3, sp1[i]->data + 1);

sp1[i] = sp1[i]->next;

}

}

sp1[i] = adr;

}

}

void DFS\_spisok(Node\*\* sp1, bool\* Arr2, int n, int i)

{

int j;

Arr2[i] = true;

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

while (sp1[i]->next != NULL)

{

sp1[i] = sp1[i]->next;

j = sp1[i]->data;

if (Arr2[j] == false)

DFS\_spisok(sp1, Arr2, n, j);

}

}

void stk(int n, int i, int\*\* Arr)

{

stack<int> stk;

int\* nodes, x;

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

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

nodes[i] = 0;

//printf("Введите вершину: ");

//scanf("%d", &x);

stk.push(0);

printf("\nРезультат обхода нерекурсивным способом: ");

while (!stk.empty())

{

int node = stk.top();

stk.pop();

if (nodes[node] == 2)

continue;

nodes[node] = 2;

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

{

if (Arr[node][j] == 1 && nodes[j] != 2)

{

stk.push(j);

nodes[j] = 1;

}

}

printf("%d ", node + 1);

}

}

int main()

{

setlocale(LC\_ALL, "Ru");

int n, \*\*t;

bool\* Arr2;

printf("Введите размер массива: ");

scanf("%d", &n);

Node\*\* sp1 = (Node\*\*)malloc(n \* sizeof(Node\*));

Arr2 = (bool\*)malloc(n \* sizeof(bool));

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

Arr2[i] = false;

t = create\_arr(n);

printf("\nРезультат обхода матрицы смежности: ");

DFS(t, n, Arr2, 0);

fun\_init(sp1, n);

fun\_spisok(sp1, n, t);

printf("\n");

print\_spisok(sp1, n);

printf("\nРезультат обхода списка смежности: ");

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

Arr2[i] = false;

DFS\_spisok(sp1, Arr2, n, 0);

stk(n, 0, t);

Free(t, Arr2, n, sp1);

\_getch();

}