Пензенский государственный университет Кафедра «Вычислительная техника»

**Отчет**о выполнении лабораторной работы №4.2 по дисциплине “Логика и основа алгоритмизации в инженерных задачах” на тему**:**

**Обход графа в глубину**

Выполнили студенты гр. 19ВВ4:   
Хлыстов А.Ю.  
Привалов А.Э.

Проверили:

Юрова О.В.  
Митрохин М. А.

Пенза, 2020 г.

**Название**Обход графа в глубину

**Цель работы**Реализация алгоритма обхода графа в глубину.

**Лабораторное задание**1. Сгенерируйте (используя генератор случайных чисел) две матрицу

смежности для неориентированного графа G. Выведите сгенерированные

матрицы на экран.

2. Реализуйте процедуру обхода в глубину для графа, представленного списками смежности.

**Код программы**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <conio.h>

#include <iostream>

#include <locale.h>

#include <list>

#include <iterator>

using namespace std;

struct graph {

int\*\* matrix;

int size;

}m;

typedef struct Node {

int num;

Node\* next;

Node\* back;

}Node;

typedef struct HeadCells {

Node\* start = NULL;

int empty;

int num;

}HeadCells;

Node\* it; ///глобальный указатель

int\* NUM;

HeadCells\*\* headers;

int null\_exist(int\* NUM, int size) {

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

if (NUM[i] != 1) {

return i;

}

}

return -1;

}

int\* generate\_list(int size, int value) {

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

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

NUM[i] = value;

}

return NUM;

}

void DFS(int v) {

NUM[v] = 1;

Node\* tmp = headers[v]->start;

printf("%d ", v);

while (tmp != NULL) {

if (NUM[tmp->num] == 0) {

DFS(tmp->num);

printf("\n");

}

tmp = tmp->next;

}

}

int\*\* createMatrix(int size) {

int\*\* matrix = (int\*\*)malloc(size \* sizeof(int\*));

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

matrix[i] = (int\*)calloc(size, sizeof(int));

}

return matrix;

}

void fillMatrix(graph m) {

int value;

for (int i = 0; i < m.size; i++) {

for (int j = i + 1; j < m.size; j++) {

value = rand() % 10;

if (value < 3)

value = 0;

else

value = 1;

m.matrix[i][j] = value;

m.matrix[j][i] = m.matrix[i][j];

}

}

}

void outputMatrix(graph m) {

printf\_s(" ");

for (int i = 0; i < m.size; i++) {

printf\_s("%d ", i);

}

printf\_s("\n");

for (int i = 0; i < m.size; i++) {

printf\_s("%d ", i);

for (int j = 0; j < m.size; j++) {

printf\_s("%d ", m.matrix[i][j]);

}

printf\_s("\n");

}

}

HeadCells\*\* createHList(int size) {

HeadCells\*\* Headers = (HeadCells\*\*)malloc(size \* sizeof(HeadCells\*));

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

Headers[i] = (HeadCells\*)malloc(sizeof(HeadCells));

Headers[i]->empty = 1;

}

return Headers;

}

Node\* createNode() {

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

N->back = NULL;

N->next = NULL;

return N;

}

void output\_alist(HeadCells\*\* headers, int size) {

int i = 0;

HeadCells\* work\_vertex;

Node\* tmp;

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

work\_vertex = headers[i];

tmp = work\_vertex->start;

printf\_s("%d. ", i);

while (tmp != NULL){

printf\_s("%d ", tmp->num);

tmp = tmp->next;

}

printf\_s("\n");

}

}

void push\_back(HeadCells\* list, int num) {

Node\* tmp;

if (list->empty == 1) {

it->num = num;

it->next = NULL;

list->start = it;

list->empty = 0;

}

else {

tmp = createNode();

tmp->num = num;

tmp->next = NULL;

it->next = tmp;

tmp->back = it;

it = tmp;

}

}

HeadCells\*\* matrix\_to\_list(graph m) {

HeadCells\*\* headers = createHList(m.size);

int\* line;

HeadCells\* workVertex;

for (int i = 0; i < m.size; i++) {

line = m.matrix[i];

workVertex = headers[i];

if (workVertex->empty == 1) {

it = createNode();

}

for (int j =0; j < m.size; j++) {

if (line[j] == 1) {

push\_back(workVertex, j);

}

else {

continue;

}

}

}

return headers;

}

int main() {

setlocale(LC\_ALL, "Rus");

m.size = 7;

int v = 0;

m.matrix = createMatrix(m.size);

fillMatrix(m);

outputMatrix(m);

printf("\n");

headers = matrix\_to\_list(m);

output\_alist(headers, m.size);

NUM = generate\_list(m.size, 0);

printf("\n");

printf("Рекурсивная реализация DFS:\n");

while (true) {

if ((v = null\_exist(NUM, m.size)) != -1) {

DFS(v);

printf("\n");

}

break;

}

return 0;

}

**Результаты работы программы**



**Рисунок 1 – результат выполнения задания**

**Выводы**В процессе выполнения лабораторной работы был реализован алгоритм обхода массива в глубину.