Министерство науки и высшего образования Российской Федерации

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 группы 22ВВВ3:

Куракин Н.Н.

Майоров Н.А.

Матюшин К. М.

Приняли:

Юрова О.В.

Акифьев И.В.

Пенза 2023

**Название**

Обход графа в глубину.

**Цель работы**

Изучить и реализовать процедуру обхода графа, представленного в виде матрицы смежности или списков смежности, 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.

**Программа***Задание 1:*

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <locale.h>

#include <malloc.h>

#include <stdlib.h>

#include <time.h>

struct node {

int numNode;

struct node\* nextNode;

};

void DFS1(int\* graph, int vertexes, int\* visited, int vertex) {

visited[vertex] = 1;

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

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

if (\*(graph + vertex \* vertexes + i) && !visited[i]) {

DFS1(graph, vertexes, visited, i);

}

}

}

void DepthFirstSearch1(int\* graph, int vertexes) {

if (!graph) return;

int\* visited = (int\*)malloc(sizeof(int) \* vertexes);

if (!visited) return;

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

visited[i] = 0;

}

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

if (!visited[i]) {

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

if (\*(graph + i \* vertexes + j) && !visited[j])

break;

if (j == vertexes - 1) {

if (i == 0)

printf("1 ");

return;

}

}

DFS1(graph, vertexes, visited, i);

}

}

free(visited);

}

void DFS2(struct node\* graph, int vertexes, int\* visited, int vertex, struct node\* head) {

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

visited[vertex] = 1;

struct node\* temp = graph;

while (temp->nextNode) {

temp = temp->nextNode;

if (!visited[(temp->numNode) - 1])

DFS2(head + (temp->numNode) - 1, vertexes, visited, (temp->numNode) - 1, head);

}

}

void DepthFirstSearch2(struct node\* graph, int vertexes) {

if (!graph) return;

int\* visited = (int\*)malloc(sizeof(int) \* vertexes);

if (!visited) return;

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

visited[i] = 0;

}

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

if (!visited[i]) {

if (graph->nextNode == NULL || (graph->nextNode != NULL && visited[graph->nextNode->numNode - 1])) {

if (i == 0)

printf("1 ");

return;

}

DFS2(graph + i, vertexes, visited, i, graph);

}

}

free(visited);

}

int\* CreateGraph(int vertexes) {

int\* Mtrx = (int\*)malloc(sizeof(int) \* vertexes \* vertexes);

if (!Mtrx) return 0;

for (int i = 0; i != vertexes; i++) {

\*(Mtrx + i \* vertexes + i) = 0;

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

\*(Mtrx + i \* vertexes + j) = rand() % 2;

\*(Mtrx + j \* vertexes + i) = \*(Mtrx + i \* vertexes + j);

}

}

return Mtrx;

}

int ModifyGraph(struct node\* list, int\* graph, int vertexes) {

if (!list || !graph) return 0;

for (int i = 0; i != vertexes; i++) {

struct node\* current = list + i;

current->numNode = i + 1;

for (int j = 0; j != vertexes; j++) {

if (\*(graph + i \* vertexes + j)) {

current->nextNode = (struct node\*)malloc(sizeof(struct node));

current = current->nextNode;

current->numNode = j + 1;

}

}

current->nextNode = NULL;

}

return 1;

}

void ShowGraph(int\* graph, int vertexes) {

if (!graph) return;

for (int i = 0; i != vertexes; i++) {

for (int j = 0; j != vertexes; j++) {

printf("%i ", \*(graph + i \* vertexes + j));

}

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

printf("\n");

}

}

void ShowList(struct node\* list, int vertexes) {

if (!list) return;

printf("\n");

for (int i = 0; i != vertexes; i++) {

struct node\* current = list + i;

//printf("|%i\t|->", current->numNode);

while (current) {

printf("|%i|\t->", current->numNode);

current = current->nextNode;

}

printf("\n");

}

}

int main() {

setlocale(LC\_ALL, "RU");

srand(time(NULL));

int vertexNum = 0;

struct node\* listGraph = 0;

printf("Введите кол-во вершин для графа: ");

(void)scanf("%i", &vertexNum);

if (!vertexNum) return 1;

int\* smej = CreateGraph(vertexNum);

ShowGraph(smej, vertexNum);

DepthFirstSearch1(smej, vertexNum);

listGraph = (struct node\*)malloc(sizeof(struct node) \* vertexNum);

if (!ModifyGraph(listGraph, smej, vertexNum)) return 1;

ShowList(listGraph, vertexNum);

DepthFirstSearch2(listGraph, vertexNum);

free(smej);

return 0;

}

*Задание 2:*

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <locale.h>

#include <malloc.h>

#include <stdlib.h>

#include <time.h>

int\* CreateGraph(int vertexes) {

int\* Mtrx = (int\*)malloc(sizeof(int) \* vertexes \* vertexes);

if (!Mtrx) return 0;

for (int i = 0; i != vertexes; i++) {

\*(Mtrx + i \* vertexes + i) = 0;

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

\*(Mtrx + i \* vertexes + j) = rand() % 2;

\*(Mtrx + j \* vertexes + i) = \*(Mtrx + i \* vertexes + j);

}

}

return Mtrx;

}

void ShowGraph(int\* graph, int vertexes) {

if (!graph) return;

for (int i = 0; i != vertexes; i++) {

for (int j = 0; j != vertexes; j++) {

printf("%i ", \*(graph + i \* vertexes + j));

}

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

printf("\n");

}

}

void DepthSearch(int\* graph, int vertexes) {

int\* pureStack = (int\*)malloc(sizeof(int) \* 1000), \* visited = (int\*)malloc(sizeof(int) \* vertexes), temp = -1;

if (!pureStack || !visited) return;

for (int i = 0; i != vertexes; i++) {

\*(pureStack + i) = -1;

\*(visited + i) = 0;

}

for (int i = 0; i != vertexes; i++) {

if (!visited[i]) {

visited[i] = 1;

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

}

for (int j = vertexes - 1; j >= 0; j--) {

if (\*(graph + i \* vertexes + j) && !visited[j]) {

temp++; pureStack[temp] = j;

}

}

if (temp == -1) {

return;

}

if (pureStack[0] != -1) {

i = pureStack[temp] - 1;

pureStack[temp] = -1; temp--;

}

}

}

int main() {

setlocale(LC\_ALL, "RU");

srand(time(NULL));

int vertexNum = 10;

struct node\* listGraph = 0;

printf("Введите кол-во вершин для графа: ");

(void)scanf("%i", &vertexNum);

if (!vertexNum) return 1;

int\* smej = CreateGraph(vertexNum);

ShowGraph(smej, vertexNum);

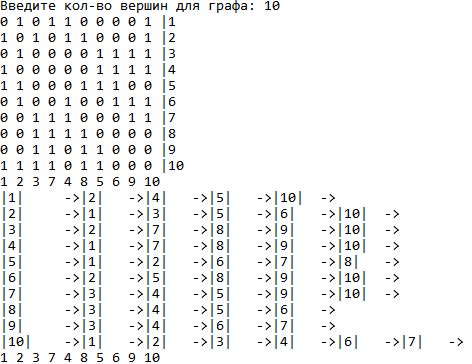
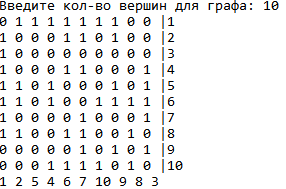
DepthSearch(smej, vertexNum);

free(smej);

return 0;

}

**Результаты выполнения программы**

****Задание 1  
 Задание 2

**Вывод:** в ходе выполнения лабораторной работы были получены навыки реализации процедуры обхода графа, представленного в виде матрицы смежности или списков смежности, в глубину.