Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №7

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Обход графа в глубину»

**Выполнил:**

Студент группы 23ВВВ2

Стрельцов А.П.

Федоров Б.М.

**Приняли:**

Юрова О. В.

Митрохин М. А.

Пенза 2024

**Цель работы:** проработать операции обхода графа в глубину, выполнить задания и усвоить материал.

**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.

**Задание 2**

**3.\*** Реализуйте процедуру обхода в глубину для графа, представленного

списками смежности.

**ЛИСТИНГ**

**laba7 .cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

// Структура для представления списка смежности

typedef struct Node {

int vertex;

struct Node\* next;

} Node;

// Функция для генерации матрицы смежности неориентированного графа

int\*\* generateAdjacencyMatrix(int n) {

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

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

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

}

srand(time(NULL));

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

for (int j = i + 1; j < n; j++) {

adjMatrix[i][j] = rand() % 2;

adjMatrix[j][i] = adjMatrix[i][j];

}

adjMatrix[i][i] = 0; // чтобы не было петель

}

return adjMatrix;

}

// Функция для вывода матрицы смежности

void printAdjacencyMatrix(int n, int\*\* adjMatrix) {

printf("Матрица смежности:\n");

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

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

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

}

printf("\n");

}

}

// Функция для обхода графа в глубину (рекурсивная реализация)

void depthFirstSearch(int n, int\*\* adjMatrix, int\* visited, int startVertex) {

visited[startVertex] = 1;

printf("%d ", startVertex);

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

if (adjMatrix[startVertex][i] && !visited[i]) {

depthFirstSearch(n, adjMatrix, visited, i);

}

}

}

// Функция для обхода графа в глубину (нерекурсивная реализация)

void depthFirstSearchIterative(int n, int\*\* adjMatrix, int startVertex) {

int\* visited = (int\*)calloc(n, sizeof(int));

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

int top = -1;

visited[startVertex] = 1;

printf("%d ", startVertex);

stack[++top] = startVertex;

while (top != -1) {

int currentVertex = stack[top];

int found = 0;

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

if (adjMatrix[currentVertex][i] && !visited[i]) {

visited[i] = 1;

printf("%d ", i);

stack[++top] = i;

found = 1;

break;

}

}

if (!found) {

top--;

}

}

free(visited);

free(stack);

}

// Функция для обхода графа в глубину (список смежности)

void depthFirstSearchAdjacencyList(Node\*\* adjList, int n, int startVertex) {

int\* visited = (int\*)calloc(n, sizeof(int)); // Массив для отслеживания посещенных вершин

int\* stack = (int\*)malloc(n \* sizeof(int)); // Стек для хранения текущего пути

int top = -1; // Индекс вершины в стеке

// Начало обхода с стартовой вершины

visited[startVertex] = 1;

printf("%d ", startVertex);

stack[++top] = startVertex;

while (top != -1) {

int currentVertex = stack[top]; // Текущая вершина

int found = 0; // Флаг для проверки, найдена ли не посещенная вершина

Node\* currentNode = adjList[currentVertex];

while (currentNode != NULL) {

if (!visited[currentNode->vertex]) {

visited[currentNode->vertex] = 1;

printf("%d ", currentNode->vertex);

stack[++top] = currentNode->vertex;

found = 1;

break; // Переход к следующему уровню

}

currentNode = currentNode->next;

}

if (!found) {

top--; // Удаление вершины из стека, если все смежные вершины посещены

}

}

free(visited); // Освобождение памяти

free(stack); // Освобождение памяти

}

// Функция для сортировки списка смежности по возрастанию индексов вершин

void sortAdjacencyList(Node\*\* adjList, int n) {

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

// Используем пузырьковую сортировку для сортировки списка смежности

Node\* current = adjList[i];

while (current != NULL) {

Node\* next = current->next;

while (next != NULL) {

if (current->vertex > next->vertex) {

// Меняем местами вершины

int temp = current->vertex;

current->vertex = next->vertex;

next->vertex = temp;

}

next = next->next;

}

current = current->next;

}

}

}

int main() {

setlocale(LC\_ALL, "RUS");

int n;

printf("Введите количество вершин в графе: ");

scanf("%d", &n);

int\*\* adjMatrix = generateAdjacencyMatrix(n);

printAdjacencyMatrix(n, adjMatrix);

printf("\nОбход графа в глубину (рекурсивная реализация):\n");

int\* visited = (int\*)calloc(n, sizeof(int));

depthFirstSearch(n, adjMatrix, visited, 0);

free(visited);

printf("\n\nОбход графа в глубину (нерекурсивная реализация):\n");

depthFirstSearchIterative(n, adjMatrix, 0);

// Создание списка смежности

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

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

adjList[i] = NULL;

}

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

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

if (adjMatrix[i][j]) {

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

newNode->vertex = j;

newNode->next = adjList[i];

adjList[i] = newNode;

}

}

}

// Сортировка списка смежности

sortAdjacencyList(adjList, n);

printf("\n\nОбход графа в глубину (список смежности):\n");

depthFirstSearchAdjacencyList(adjList, n, 0);

// Освобождение памяти

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

free(adjMatrix[i]);

}

free(adjMatrix);

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

Node\* currentNode = adjList[i];

while (currentNode != NULL) {

Node\* temp = currentNode;

currentNode = currentNode->next;

free(temp);

}

}

free(adjList);

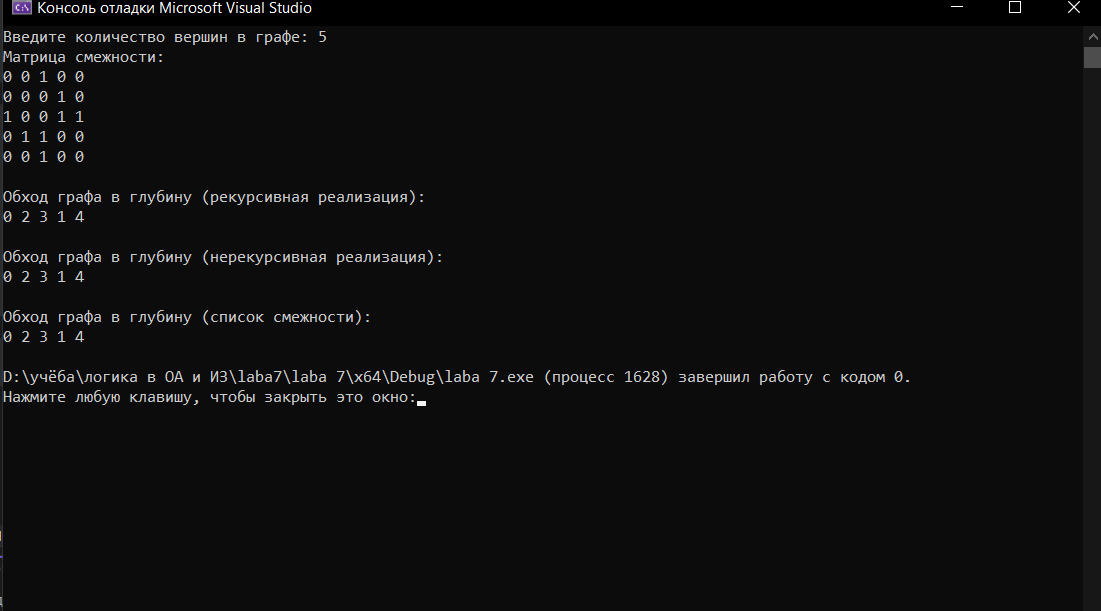
printf("\n");

return 0;

}

**Результат работы программы**

На рисунке 1 показана реализация задания №1 и 2.



**Рисунок 1 – Результат работы программы**

**Вывод**

Познакомились операциями обхода графа в глубину, изучили их работу и применили их в лабораторной работе.