Министерство науки и высшего 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”

Выполнили студенты группы 22ВВВ2:

Юртаев Д. Ю.

Шатин Д. Д.

Харитонов А. А.

Приняли:

Акифьев И. В.

Митрохин М. А.

Пенза 2023

**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.

**Задание №1.1.2**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <clocale>

#define MAX\_VERTICES 100

// Генерация случайной матрицы смежности

void generateRandomAdjacencyMatrix(int matrix[MAX\_VERTICES][MAX\_VERTICES], int numVertices, int numEdges) {

int i, j;

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

for (j = 0; j < numVertices; j++) {

matrix[i][j] = 0;

}

}

int edgeCount = 0;

while (edgeCount < numEdges) {

int vertex1 = rand() % numVertices;

int vertex2 = rand() % numVertices;

if (vertex1 != vertex2 && matrix[vertex1][vertex2] == 0) {

matrix[vertex1][vertex2] = 1;

matrix[vertex2][vertex1] = 1;

edgeCount++;

}

}

}

// Обход в глубину

void depthFirstSearch(int matrix[MAX\_VERTICES][MAX\_VERTICES], bool visited[MAX\_VERTICES], int currentVertex, int numVertices) {

printf("Посещен вершина: %d\n", currentVertex);

visited[currentVertex] = true;

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

if (matrix[currentVertex][i] == 1 && !visited[i]) {

depthFirstSearch(matrix, visited, i, numVertices);

}

}

}

int main() {

setlocale(LC\_ALL, "Russian");

int numVertices = 6; // Задайте желаемое количество вершин

int numEdges = 8; // Задайте желаемое количество ребер

int adjacencyMatrix[MAX\_VERTICES][MAX\_VERTICES];

generateRandomAdjacencyMatrix(adjacencyMatrix, numVertices, numEdges);

printf("Матрица смежности:\n");

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

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

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

}

printf("\n");

}

bool visited[MAX\_VERTICES];

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

visited[i] = false;

}

int startVertex;

printf("Введите стартовую вершину (от 0 до %d): ", numVertices - 1);

scanf("%d", &startVertex);

if (startVertex >= 0 && startVertex < numVertices) {

printf("Результат обхода в глубину, начиная с вершины %d:\n", startVertex);

depthFirstSearch(adjacencyMatrix, visited, startVertex, numVertices);

}

else {

printf("Некорректный ввод стартовой вершины.\n");

}

return 0;

}

**Задание №1.3**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <clocale>

#define MAX\_VERTICES 100

// Функция для добавления вершины в список смежности

void addEdge(int\* adjacencyLists[], int u, int v) {

// Добавить вершину v в список смежности вершины u

adjacencyLists[u][v] = 1;

// Для неориентированного графа

adjacencyLists[v][u] = 1;

}

// Функция для вывода списка смежности

void printAdjacencyList(int\* adjacencyLists[], int numVertices) {

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

printf("Список смежности для вершины %d: ", i);

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

if (adjacencyLists[i][j] == 1) {

printf("%d -> ", j);

}

}

printf("---- \n");

}

}

// Функция для обхода в глубину

void DFS(int\* adjacencyLists[], int numVertices, int vertex, int visited[]) {

visited[vertex] = 1;

printf("Посещенная вершина: %d\n", vertex);

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

if (adjacencyLists[vertex][i] == 1 && !visited[i]) {

DFS(adjacencyLists, numVertices, i, visited);

}

}

}

int main() {

setlocale(LC\_ALL, "Russian");

int numVertices = 6;

int\* adjacencyLists[MAX\_VERTICES];

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

adjacencyLists[i] = (int\*)malloc(numVertices \* sizeof(int));

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

adjacencyLists[i][j] = 0; // Инициализация нулями

}

}

addEdge(adjacencyLists, 0, 1);

addEdge(adjacencyLists, 0, 2);

addEdge(adjacencyLists, 1, 3);

addEdge(adjacencyLists, 2, 4);

addEdge(adjacencyLists, 3, 5);

int visited[MAX\_VERTICES] = { 0 };

printf("Список смежности:\n");

printAdjacencyList(adjacencyLists, numVertices);

int startVertex;

printf("Введите стартовую вершину (от 0 до %d): ", numVertices - 1);

scanf("%d", &startVertex);

if (startVertex >= 0 && startVertex < numVertices) {

printf("Результат обхода в глубину, начиная с вершины %d:\n", startVertex);

DFS(adjacencyLists, numVertices, startVertex, visited);

}

else {

printf("Некорректный ввод стартовой вершины.\n");

}

// Освобождение памяти

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

free(adjacencyLists[i]);

}

return 0;

}

**Задание №2**

#include <stdio.h>

#include <stdlib.h>

#include <clocale>

#define MAX\_VERTICES 100

// Функция для добавления вершины в стек

void push(int stack[], int\* top, int vertex) {

stack[++(\*top)] = vertex;

}

// Функция для извлечения вершины из стека

int pop(int stack[], int\* top) {

return stack[(\*top)--];

}

// Функция для обхода в глубину без использования рекурсии

void DFS(int adjacencyMatrix[][MAX\_VERTICES], int numVertices, int startVertex) {

int visited[MAX\_VERTICES] = { 0 };

int stack[MAX\_VERTICES];

int top = -1;

push(stack, &top, startVertex);

printf("Результат обхода в глубину:\n");

while (top != -1) {

int currentVertex = pop(stack, &top);

if (visited[currentVertex] == 1) {

continue;

}

printf("Посещена вершина: %d\n", currentVertex);

visited[currentVertex] = 1;

for (int i = numVertices - 1; i >= 0; i--) {

if (adjacencyMatrix[currentVertex][i] == 1 && visited[i] == 0) {

push(stack, &top, i);

}

}

}

}

int main() {

setlocale(LC\_ALL, "Russian");

int numVertices = 6;

int adjacencyMatrix[MAX\_VERTICES][MAX\_VERTICES] = { 0 };

adjacencyMatrix[0][1] = 1;

adjacencyMatrix[1][0] = 1;

adjacencyMatrix[0][2] = 1;

adjacencyMatrix[2][0] = 1;

adjacencyMatrix[1][3] = 1;

adjacencyMatrix[3][1] = 1;

adjacencyMatrix[2][4] = 1;

adjacencyMatrix[4][2] = 1;

adjacencyMatrix[3][5] = 1;

adjacencyMatrix[5][3] = 1;

int startVertex = 0;

DFS(adjacencyMatrix, numVertices, startVertex);

return 0;

}

**Результаты работы программ**

Изображение выглядит как текст, снимок экрана, программное обеспечение, компьютер

Автоматически созданное описание

**Рисунок 1 - Результат работы программы lab7.12**

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

**Рисунок 2 - Результат работы программы lab7.1.3**

Изображение выглядит как текст, снимок экрана, программное обеспечение, компьютер

Автоматически созданное описание

**Рисунок 3 - Результат работы программы lab7.2**

**Вывод**

В результате выполнения лабораторной работы были успешно реализованы алгоритмы построения матрицы смежности, усвоены и практически проверены навыки в алгоритме реализации проверки графа в глубину.