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.

**ЛИСТИНГ**

**Lab7 – 1,2.cpp**

﻿#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdlib.h>

#include <stdio.h>

#include <locale.h>

int\*\* createG(int size) {

int\*\* G = NULL;

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

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

G[i] = (int\*)malloc(size \* sizeof(int));

}

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

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

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

if (i == j) {

G[i][j] = 0;

}

G[j][i] = G[i][j];

}

}

return G;

}

void printG(int\*\* G, int size) {

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

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

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

}

printf("\n");

}

return;

}

void dfs(int s, int\*\* G, int size, int\* vis) {

vis[s] = 1;

printf("%d", s);

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

if (G[s][i] == 1 && vis[i] == 0) {

dfs(i, G, size, vis);

}

}

}

int main()

{

setlocale(LC\_ALL, "Rus");

int size = 5;

int\*\* G1 = createG(size);

printf("Введите размер матрицы \n");

scanf("%d", &size);

printf("Матрица 1\n");

printG(G1, size);

int\* vis = NULL;

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

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

vis[i] = 0;

}

printf("Список посещенных вершин: ");

dfs(0, G1, size, vis);

return 0;

}

**Результат работы программы**

На рисунке 1 показана реализация задания №1 – 1,2.

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

Автоматически созданное описание

**Рисунок 1 – Результат работы программы №1**

**Задание 1**

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

списками смежности.

**ЛИСТИНГ**

**Lab7 1-3.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

// Структура для узла списка смежности

typedef struct AdjListNode {

int dest;

struct AdjListNode\* next;

} AdjListNode;

// Структура для списка смежности

typedef struct AdjList {

AdjListNode\* head;

} AdjList;

// Структура для графа

typedef struct Graph {

int V;

AdjList\* array;

} Graph;

// Функция для создания нового узла списка смежности

AdjListNode\* newAdjListNode(int dest) {

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

newNode->dest = dest;

newNode->next = NULL;

return newNode;

}

// Функция для создания графа с V вершинами

Graph\* createGraph(int V) {

Graph\* graph = (Graph\*)malloc(sizeof(Graph));

graph->V = V;

graph->array = (AdjList\*)malloc(V \* sizeof(AdjList));

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

graph->array[i].head = NULL;

}

return graph;

}

// Функция для добавления ребра в неориентированный граф

void addEdge(Graph\* graph, int src, int dest) {

// Добавляем ребро от src до dest

AdjListNode\* newNode = newAdjListNode(dest);

newNode->next = graph->array[src].head;

graph->array[src].head = newNode;

// Поскольку граф неориентированный, добавляем ребро от dest до src также

newNode = newAdjListNode(src);

newNode->next = graph->array[dest].head;

graph->array[dest].head = newNode;

}

// Функция для генерации случайной матрицы смежности для неориентированного графа

int\*\* createRandomAdjMatrix(int size) {

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

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

G[i] = (int\*)malloc(size \* sizeof(int));

}

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

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

if (i == j) {

G[i][j] = 0;

}

else {

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

G[j][i] = G[i][j]; // Обеспечиваем симметрию для неориентированного графа

}

}

}

return G;

}

// Функция для вывода матрицы смежности

void printMatrix(int\*\* G, int size) {

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

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

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

}

printf("\n");

}

}

// Рекурсивная вспомогательная функция DFS для представления в виде списка смежности

void DFSUtil(int v, int visited[], Graph\* graph) {

visited[v] = 1;

printf("%d ", v);

AdjListNode\* temp = graph->array[v].head;

while (temp) {

int adjVertex = temp->dest;

if (!visited[adjVertex]) {

DFSUtil(adjVertex, visited, graph);

}

temp = temp->next;

}

}

// Выполнение обхода в глубину с использованием списка смежности

void DFS(Graph\* graph, int startVertex) {

int\* visited = (int\*)malloc(graph->V \* sizeof(int));

for (int i = 0; i < graph->V; i++) {

visited[i] = 0;

}

DFSUtil(startVertex, visited, graph);

free(visited);

}

int main() {

setlocale(LC\_ALL, "Rus");

srand((unsigned)time(NULL));

int size;

printf("Введите размер матрицы: ");

scanf("%d", &size);

// Генерация и вывод матрицы смежности

int\*\* G = createRandomAdjMatrix(size);

printf("Сгенерированная матрица смежности:\n");

printMatrix(G, size);

// Создание графа из матрицы смежности

Graph\* graph = createGraph(size);

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

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

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

addEdge(graph, i, j);

}

}

}

// Выполнение обхода в глубину начиная с вершины 0

printf("Результат обхода в глубину начиная с вершины 0:\n");

DFS(graph, 0);

// Освобождение выделенной памяти

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

free(G[i]);

}

free(G);

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

AdjListNode\* current = graph->array[i].head;

while (current) {

AdjListNode\* temp = current;

current = current->next;

free(temp);

}

}

free(graph->array);

free(graph);

return 0;

}

**Результат работы программы**

На рисунке 2 показана реализация задания №1 – 3.

**Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание**

**Рисунок 2 – Результат работы программы №2**

**Задание 2\***

**1.** Для матричной формы представления графов выполните

преобразование рекурсивной реализации обхода графа к не рекурсивной.

**ЛИСТИНГ**

**Lab7 2-1.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

// Функция для генерации случайной матрицы смежности для неориентированного графа

int\*\* createRandomAdjMatrix(int size) {

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

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

G[i] = (int\*)malloc(size \* sizeof(int));

}

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

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

if (i == j) {

G[i][j] = 0; // Нет петель

}

else {

G[i][j] = rand() % 2; // Случайное ребро (0 или 1)

G[j][i] = G[i][j]; // Обеспечиваем симметрию для неориентированного графа

}

}

}

return G;

}

// Функция для вывода матрицы смежности

void printMatrix(int\*\* G, int size) {

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

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

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

}

printf("\n");

}

}

// Непосредственный обход в глубину (DFS) с использованием стека

void dfsNonRecursive(int start, int\*\* G, int size) {

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

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

visited[i] = 0; // Изначально все вершины не посещены

}

int\* stack = (int\*)malloc(size \* sizeof(int)); // Стек для хранения вершин

int top = -1;

stack[++top] = start; // Начинаем с заданной начальной вершины

while (top != -1) {

int v = stack[top--]; // Извлекаем вершину из стека

if (!visited[v]) {

printf("%d ", v); // Посещаем вершину

visited[v] = 1; // Отмечаем как посещенную

}

// Добавляем непосещенные соседние вершины в стек

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

if (G[v][i] == 1 && !visited[i]) {

stack[++top] = i; // Помещаем в стек

}

}

}

free(stack);

free(visited);

}

int main() {

setlocale(LC\_ALL, "Rus");

srand((unsigned)time(NULL));

int size;

printf("Введите размер матрицы: ");

scanf("%d", &size);

// Генерация и вывод матрицы смежности

int\*\* G = createRandomAdjMatrix(size);

printf("Сгенерированная матрица смежности:\n");

printMatrix(G, size);

// Выполнение обхода в глубину начиная с вершины 0

printf("Результат обхода в глубину начиная с вершины 0:\n");

dfsNonRecursive(0, G, size);

// Освобождение выделенной памяти

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

free(G[i]);

}

free(G);

return 0;

}

**Результат работы программы**

На рисунке 3 показана реализация задания №2.

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

Автоматически созданное описание

**Рисунок 3 – Результат работы программы №3**

**Вывод**

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