

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**"МИРЭА - Российский технологический университет"**

**РТУ МИРЭА**

Институт информационных технологий (ИТ)

Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ)

**ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №6**

**по дисциплине**

**«Структуры и алгоритмы обработки данных»**

Тема. Основные алгоритмы работы с графами. Применение графа в решении практических задач

|  |  |  |
| --- | --- | --- |
| Выполнил студент группы ИКБО-04-22 |  | Кликушин В.И. |
| Принял старший преподаватель |  | Скворцова Л.А. |

Москва 2023

ОГЛАВЛЕНИЕ

[1 ЗАДАНИЕ №1 3](#_Toc152858513)

[1.1 УСЛОВИЕ ЗАДАЧИ И ЗАДАНИЕ ВАРИАНТА 3](#_Toc152858514)

[1.2 ОБОБЩЕННОЕ ОПИСАНИЕ ОСНОВНЫХ АЛГОРИТМОВ 3](#_Toc152858515)

[1.3 ОПИСАНИЕ ПРОЦЕССА РАЗРАБОТКИ 3](#_Toc152858516)

[1.3.1 Подход к решению 3](#_Toc152858517)

[1.3.2 Алгоритм вывода графа 4](#_Toc152858518)

[1.3.3 Алгоритм ввода графа с клавиатуры 4](#_Toc152858519)

[1.3.4 Алгоритм создания неориентированного графа 4](#_Toc152858520)

[1.3.5 Алгоритм подсчета глубины графа от заданной вершины 4](#_Toc152858521)

[1.3.6 Построение остовного дерева 4](#_Toc152858522)

[1.4 КОД ПРИЛОЖЕНИЯ 5](#_Toc152858523)

[1.4.1 Код файла Graph.h 5](#_Toc152858524)

[1.4.2 Код файла Graph.cpp 6](#_Toc152858525)

[1.4.3 Код файла main.cpp 11](#_Toc152858526)

[1.5 ТЕСТИРОВАНИЕ 14](#_Toc152858527)

[2 ЗАДАНИЕ №2 20](#_Toc152858528)

[2.1 УСЛОВИЕ ЗАДАЧИ 20](#_Toc152858529)

[2.2 ОПИСАНИЕ ПОДХОДА К РЕШЕНИЮ 20](#_Toc152858530)

[2.3 КОД ПРОГРАММЫ 21](#_Toc152858531)

[2.4 ТЕСТИРОВАНИЕ 22](#_Toc152858532)

[3 ВЫВОДЫ 25](#_Toc152858533)

[4 ИСПОЛЬЗУЕМЫЕ ИСТОЧНИКИ 26](#_Toc152858534)

# 1 ЗАДАНИЕ №1

## 1.1 УСЛОВИЕ ЗАДАЧИ И ЗАДАНИЕ ВАРИАНТА

Представление графа в памяти: Матрица смежности.

Задачи:

1) Определить глубину графа.

2) Составить программу реализации алгоритма построения остовного дерева методом поиска в ширину в неориентированном графе. Разработать доступный способ (форму) вывода результирующего дерева на экран монитора.

## 1.2 ОБОБЩЕННОЕ ОПИСАНИЕ ОСНОВНЫХ АЛГОРИТМОВ

Глубина графа вычисляется путем нахождения максимальной из глубин от каждой вершины. Используется обход графа в глубину, где считается количество ребёр при максимально длинном простом пути от данной вершины. (алгоритм может также поддерживать работу, где будет считаться «стоимость» обхода – веса рёбер). Остовное дерево строится при помощи очереди и обхода графа в ширину.

## 1.3 ОПИСАНИЕ ПРОЦЕССА РАЗРАБОТКИ

### 1.3.1 Подход к решению

Граф представляется матрицей смежности. Были созданы две структуры, отвечающие за ребро и сам граф соответственно. В структуре ребра поля – вершина-исток, вершина-сток, информационная часть ребра. В структуре графа – количество вершин, количество рёбер, матрица смежности, два конструктора для инициализации графа.

struct Edge //Структура ребра

{

// v и w это номера вершин ребра

int v; // исток

int w; // сток

int info;

};

Edge EDGE(int v, int w, Edge e);

//Структура представления графа на матрице смежности

struct Graph

{

int V; // Количество вершин

int E; // Количество ребер

int\*\* Matr; // Матрица смежности

Graph(int V) : V(V), E(0) // Изменить конструктор для установки значения V

{

Matr = new int\* [V];

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

{

Matr[i] = new int[V];

for (int j = 0; j < V; j++)

Matr[i][j] = 0;

}

}

Graph() {V = 0; }

};

### 1.3.2 Алгоритм вывода графа

Граф выводится в виде списка вершин в следующем виде:

1: [2(3),4(7)]. Первое число – вершина исток, в квадратных скобках список смежных вершин – сама вершина и вес ребра. Для вывода в таком виде необходимо пройтись по матрице смежности и вывести соответствующие вершины, если элемент вершины матрицы смежности не равен нулю.

### 1.3.3 Алгоритм ввода графа с клавиатуры

Вводится количество вершин графа, затем в бесконечном цикле пользователь может выбрать между завершением ввода и продолжением. Вводится вершина исток, вершина сток, вес ребра.

### 1.3.4 Алгоритм создания неориентированного графа

Создание графа производится путем добавления ребра. Обновляется количество вершин в графе и матрица смежности.

### 1.3.5 Алгоритм подсчета глубины графа от заданной вершины

От рассматриваемой вершины происходит переход к одной из смежных непосещённых вершин, и она становится рассматриваемой. Процесс повторяется до момента, пока есть смежные непосещённые вершины, после чего происходит откат к предыдущей вершине. Считается количество рёбер при обходе (стоимость обхода).

### 1.3.6 Построение остовного дерева

Строится путем обхода в ширину.

## 1.4 КОД ПРИЛОЖЕНИЯ

### 1.4.1 Код файла Graph.h

#pragma once

#include <iostream>

#include <queue>

#include<set>

using namespace std;

struct Edge //Структура ребра

{

// v и w это номера вершин ребра

int v; // исток

int w; // сток

int info;

};

Edge EDGE(int v, int w, Edge e);

//Структура представления графа на матрице смежности

struct Graph

{

int V; // Количество вершин

int E; // Количество ребер

int\*\* Matr; // Матрица смежности

Graph(int V) : V(V), E(0) // Изменить конструктор для установки значения V

{

Matr = new int\* [V];

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

{

Matr[i] = new int[V];

for (int j = 0; j < V; j++)

Matr[i][j] = 0;

}

}

Graph() {V = 0; }

};

void GRAPHInit(int V, Graph& G);

void GRAPHInsert(Graph& G, Edge e);

bool printGraphMatrix(Graph G);

bool printGraph(Graph G);

int dfs(Graph G, int v, int\* visited);

int maindfs(Graph G, int v);

void buildSpanningTree(Graph& G, Graph& T, int start);

void printSpanningTree(Graph T);

void colorGraph(Graph& G);

### 1.4.2 Код файла Graph.cpp

#include "Graph.h"

Edge EDGE(int v, int w, Edge e)

{

e.v = v;

e.w = w;

return e;

}

void GRAPHInit(int V, Graph& G)

{

G.V = V;//инициализация значения V

G.E = 0;//инициализация значения Е

//Создание двумерного массива (матрицы) размером V\*V

G.Matr = new int\* [V];

for (int v = 0; v < V; v++)

G.Matr[v] = new int[V];

//заполнение элементов матрицы значением 0

for (int v = 0; v < V; v++)

for (int w = 0; w < V; w++)

{

G.Matr[v][w] = 0;

}

}

//Предусловие: на вход поступает граф G и данные вставляемого ребра e/

//Постусловие: вставка ребра в матрицу смежности неориентированного графа

void GRAPHInsert(Graph& G, Edge e)

{

int v = e.v-1, w = e.w-1;

if (G.Matr[v][w] == 0)

{

G.E++;//подсчет количества вставленных ребер

G.Matr[v][w] = e.info;

G.Matr[w][v] = e.info;

}

}

bool printGraphMatrix(Graph G)

{

if (G.V == 0)

return 1;

for (int i = 0; i < G.V; i++)

{

for (int j = 0; j < G.V; j++)

{

cout << G.Matr[i][j] << ' ';

}

cout << '\n';

}

return 0;

}

bool printGraph(Graph G)

{

if (G.V == 0)

return 1;

bool flag = 0;

for (int i = 0; i < G.V; i++)

{

flag = 0;

cout << " " << i + 1 << ": [";

for (int j = 0; j < G.V; j++)

{

if (G.Matr[i][j] != 0)

{

if (not flag)

{

cout << j + 1 << "(" << G.Matr[i][j] << ")";

flag = 1;

}

else

{

cout << ", " << j+1 << "(" << G.Matr[i][j] << ")";

}

}

}

cout << ']'<<'\n';

}

return 0;

}

int dfs(Graph G, int v, int\* visited)

{

int maxDepth = 0;

visited[v] = 1;

for (int w = 0; w < G.V; w++)

{

if (G.Matr[v][w] != 0 && visited[w] == 0)

{

int depth = dfs(G, w, visited) + G.Matr[v][w];

if (depth > maxDepth)

maxDepth = depth;

}

}

visited[v] = 0;

return maxDepth;

}

int maindfs(Graph G, int v)

{

int\* visited = new int[G.V];

for (int v1 = 0; v1 < G.V; v1++)

visited[v1] = 0;

int depth = dfs(G, v, visited);

delete[] visited;

return depth;

}

void buildSpanningTree(Graph& G, Graph& T, int start)

{

T.V = G.V; // Количество вершин в остовном дереве равно количеству вершин в графе

T.E = 0; // Количество ребер в остовном дереве инициализируем нулем

T.Matr = new int\* [T.V]; // Создаем матрицу смежности для остовного дерева

for (int i = 0; i < T.V; i++)

{

T.Matr[i] = new int[T.V];

for (int j = 0; j < T.V; j++)

T.Matr[i][j] = 0;

}

std::queue<int> queue; // Очередь для обхода в ширину

std::vector<bool> visited(T.V, false); // Вектор для отслеживания посещенных вершин

visited[start] = true; // Помечаем стартовую вершину как посещенную

queue.push(start); // Добавляем стартовую вершину в очередь

while (!queue.empty())

{

int v = queue.front(); // Берем вершину из начала очереди

queue.pop(); // Удаляем вершину из очереди

for (int w = 0; w < T.V; w++)

{

if (G.Matr[v][w] != 0 && !visited[w]) // Если есть ребро из текущей вершины в смежную непосещенную вершину

{

T.Matr[v][w] = G.Matr[v][w]; // Добавляем ребро в остовное дерево

T.Matr[w][v] = G.Matr[v][w];

T.E++; // Увеличиваем количество ребер в остовном дереве

visited[w] = true; // Помечаем смежную вершину как посещенную

queue.push(w); // Добавляем смежную вершину в очередь

}

}

}

}

void printSpanningTree(Graph T)

{

for (int i = 0; i < T.V; i++)

{

for (int j = 0; j < T.V; j++)

{

if (T.Matr[i][j] != 0)

{

cout << i + 1 << " -> " << j + 1 << " (" << T.Matr[i][j] << ")" << '\n';

}

}

}

}

void colorGraph(Graph& G) {

int n = G.V; // Количество вершин

int\*\* A = G.Matr; // Матрица смежности

set<int> V; // Множество вершин

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

V.insert(i);

}

int k = 1; // Цвет

set<int> Sk; // Независимое множество покрашенных вершин

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

if (V.find(i) == V.end()) { // Вершина уже покрашена, переходим к следующей

continue;

}

Sk.insert(i); // Красим текущую вершину

V.erase(i);

while (true) {

int j = i + 1;

while (j < n && (A[i][j] == 1 || V.find(j) == V.end())) {

j++;

}

if (j == n) {

break;

}

Sk.insert(j); // Красим соседнюю вершину

V.erase(j);

// Обновление строки i матрицы A

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

A[i][l] |= A[j][l];

}

}

// Вывод информации о текущей цветовой группе

cout << "Color " << k << ": ";

for (int v : Sk) {

cout << v+1 << " ";

}

cout << endl;

k++;

}

}

### 1.4.3 Код файла main.cpp

#include "Graph.h"

int main()

{

system("chcp 1251");

Graph graph,T;

int num,V,deep,maxDeep=0;

bool res;

while (true)

{

cout << "-----------------------------------------------------------------------------------------------------------" << endl;

cout << " Меню\n";

cout << " 1. Заполнить граф с клавиатуры\n";

cout << " 2. Вывести матрицу смежности графа на экран\n";

cout << " 3. Вывести граф в виде списка смежных вершин\n";

cout << " 4. Определить глубину графа\n";

cout << " 5. Построить остовное дерево методом поиска в ширину. Вывести дерево на экран\n";

cout << " 6. Раскрасить граф\n";

cout << " 7. Завершить выполнение программы\n";

cin >> num;

switch (num)

{

case 1:

{

cout << " Введите количество вершин в графе: ";

cin >> V;

if (V == 0)

cout << " Задан пустой граф\n";

else

{

GRAPHInit(V, graph);

while (true)

{

Edge e;

cout << " Введите номера вершин ребра (от 1 до " << graph.V << "):\n";

cout << " Вершина-исток: ";

cin >> e.v;

cout << " Вершина-сток: ";

cin >> e.w;

cout << " Введите вес вершины: ";

cin >> e.info;

if (e.w > graph.V or e.w < 1 or e.v<1 or e.v>graph.V)

cout << " Неккоректые номера вершин, повторите ввод\n";

else if (e.w == e.v)

cout << " Граф не должен иметь рёбра, исходящие из вершины и входящие в неё\n";

else

GRAPHInsert(graph, e);

cout << " Продолжить? 1 - да, 0 - нет\n";

cin >> num;

if (num == 0)

break;

}

}

break;

}

case 2:

{

cout << " Матрица смежности графа:\n";

res = printGraphMatrix(graph);

if (res)

cout << " Пустой граф\n";

break;

}

case 3:

{

cout << " Граф в виде списка смежных вершин (в скобках указан вес вершины):\n";

res = printGraph(graph);

if (res)

cout << " Пустой граф\n";

break;

}

case 4:

{

if (graph.V == 0)

cout << " Пустой граф\n";

else

{

maxDeep = 0;

for (int i = 0; i < graph.V; i++)

{

deep = maindfs(graph, i);

cout << " Глубина графа от вершины " << i+1 << ": " << deep << endl;

maxDeep = max(deep, maxDeep);

}

cout << " Глубина графа: " << maxDeep << endl;

}

break;

}

case 5:

if (graph.V == 0)

cout << " Пустой граф\n";

else

{

buildSpanningTree(graph, T, 1); // Строим остовное дерево методом поиска в ширину

printSpanningTree(T);

}

break;

case 6:

if (graph.V == 0)

cout << " Пустой граф\n";

else

{

colorGraph(graph);

}

break;

case 7:

return 0;

default:

{

cout << " Нет такого пункта\n";

}

}

}

}

## 1.5 ТЕСТИРОВАНИЕ

Для проведения тестирования функционала приложения был выбран следующий граф:

Изображение выглядит как диаграмма, линия, рисунок, зарисовка

Автоматически созданное описание

Рисунок 1 – Граф для тестирования

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 2 – Создание и вывод предложенного графа

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 3 – Вывод матрицы смежности для предложенного графа

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

Рисунок 4 – Подсчет глубины графа для предложенного графа

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 5 – Построение и вывод остовного дерева предложенного графа

Подготовим случайный граф для дополнительного тестирования функций.

Изображение выглядит как линия, диаграмма, круг

Автоматически созданное описание

Рисунок 6 – Случайный граф

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 7 – Построение и вывод случайного графа

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

Рисунок 8 – Глубина случайного графа

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 9 – Остовное дерево случайного графа

# 2 ЗАДАНИЕ №2

## 2.1 УСЛОВИЕ ЗАДАЧИ

Раскрасить граф так что любые две смежные вершины были раскрашены в разные цвета, а число использованных цветов было минимально возможным.

## 2.2 ОПИСАНИЕ ПОДХОДА К РЕШЕНИЮ

Предложенный алгоритм работает на основе битовых операций над матрицей смежности. A — квадратная матрица размера n×n, в которой значение элемента aij равно числу рёбер из i-й вершины графа в j-ю вершину, причем aij ∈ {0, 1} (между двумя вершинами может быть только одно ребро). Будем считать, что любая вершина смежна сама с собой, то есть на главной диагонали матрицы стоят 1. Над матрицей A проводятся следующие операции:

Начиная с первой строки матрицы A (i=1), производится поиск по строке первой несмежной неокрашенной вершины. Несмежной вершиной j окажется та, для которой в столбце j i-ой строки будет стоять 0.

Далее, согласно алгоритму, все соседи вершины j добавляются к соседям вершины i (происходит стягивание вершин). В результате получается обновленная i-ая строка матрицы A, в которой 0 стоят только в столбцах с номерами вершин, которые несмежны ни с i-ой, ни с j-ой вершинами. Математически это можно заменить операцией дизъюнкции булевых векторов, представляющих собой строки i и j.

Номера строк i и j матрицы A формируют первую цветовую группу.

Затем производим ту же операцию поиска несмежной вершины или, можно сказать, первого 0 в обновленной строке с номером i и таким образом, возможно, добавляем новые вершины в первую цветовую группу.

Набор в группу заканчивается в двух случаях: во-первых, когда в i-ой строке накапливаются все 1 – это означает, что все несмежные друг с другом и с вершиной i вершины уже попали в группу с одним цветом; во-вторых, когда все вершины графа G уже покрашены, то есть любая вершина находится в какой-либо цветовой группе.

Если набор в группу закончился по первой причине, то алгоритм продолжает работу со следующей неокрашенной строки по порядку после i-ой. Производится набор строк в новую цветовую группу согласно вышеописанной последовательности действий. Алгоритм полностью останавливается в случае, если всем вершинам графа G назначены цвета, другими словами, все строки матрицы A распределены по цветовым группам.

## 2.3 КОД ПРОГРАММЫ

void colorGraph(Graph& G) {

int n = G.V; // Количество вершин

int\*\* A = G.Matr; // Матрица смежности

set<int> V; // Множество вершин

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

V.insert(i);

}

int k = 1; // Цвет

set<int> Sk; // Независимое множество покрашенных вершин

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

if (V.find(i) == V.end()) { // Вершина уже покрашена, переходим к следующей

continue;

}

Sk.insert(i); // Красим текущую вершину

V.erase(i);

while (true) {

int j = i + 1;

while (j < n && (A[i][j] == 1 || V.find(j) == V.end())) {

j++;

}

if (j == n) {

break;

}

Sk.insert(j); // Красим соседнюю вершину

V.erase(j);

// Обновление строки i матрицы A

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

A[i][l] |= A[j][l];

}

}

// Вывод информации о текущей цветовой группе

cout << "Color " << k << ": ";

for (int v : Sk) {

cout << v+1 << " ";

}

cout << endl;

k++;

Sk.clear();

}

}

## 2.4 ТЕСТИРОВАНИЕ

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 10 – Тестирование на случайном графе из задания 1

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 11 – Тестирование на графе из 3 вершин, каждая из которых является смежной с двумя другими

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 12 – Тестирование на выбранном графе

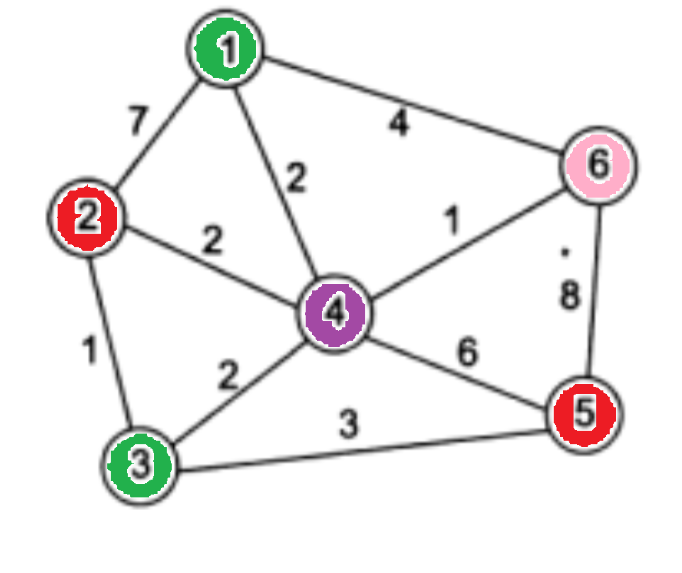


Рисунок 13 – Раскраска выбранного графа

# 3 ВЫВОДЫ

В ходе выполнения практической работы были получены навыки работы с такой структурой данных как граф.

# 4 ИСПОЛЬЗУЕМЫЕ ИСТОЧНИКИ

1. Учебно-методическое пособие СиАОД (часть 2)
2. Приложение к практическим работам – СДО (online-edu.mirea.ru)