



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ  
ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

**«МИРЭА - Российский технологический университет»**

**РТУ МИРЭА**

---

Отчет по выполнению практического задания №5

**Тема:** Основные алгоритмы работы с графами

**Дисциплина:** «Структуры и алгоритмы обработки данных»

Выполнил студент

Хан А.А.

группа

ИКБО-04-20

Москва 2021

## Оглавление

Тема .....	3
Цель .....	3
Персональный вариант и задание.....	3
Подход к решению .....	4
Работа пользователя с приложением.....	5
Код приложения .....	6
Тестирование .....	9
Выводы.....	12
Список использованной литературы .....	13

## Тема

Основные алгоритмы работы с графами.

## Цель

Получение практических навыков по выполнению операций над структурой данных граф.

## Персональный вариант и задание

### Вариант №13

Номер варианта	Представление графа в памяти	Задачи
13	Матрица смежности	Ввод с клавиатуры графа (применение операции вставки ребра в граф). Определить глубину графа. Составить программу реализации алгоритма построения остоного дерева методом поиска в ширину в неориентированном графе. Разработать доступный способ (форму) вывода результирующего дерева на экран монитора.

## Подход к решению

### Класс `graph`

Класс **graph** предназначен для хранения данных о неориентированном графе (количество вершин и матрица смежности).

#### Поля класса:

- `int n` – количество вершин в графе.
- `std::vector<std::vector<int>> M` – матрица смежности графа ( $M[i][j]$  – вес ребра между вершинами с номерами  $(i + 1)$  и  $(j + 1)$ )

#### Методы класса:

- `graph(int _n = 0)` – конструктор графа с `_n` вершинами (выделяет память под матрицу смежности  $_n \times _n$ ).
- `bool correct_ind(int a)` – проверка на корректность номера вершины `a` для данного графа.
- `void print_edge(int a, int b)` – вывод в консоль ребра между вершинами `a` и `b` с весом.
- `std::string add_edge(int a, int b, int w)` – добавление ребра между вершинами `a` и `b` с весом `w`, возвращает строку 'OK' или 'Error' (успешное и неуспешное добавление).
- `std::pair<int, int> find_distant_v(int v, std::vector<bool> &used)` – функция поиска самой дальней вершины относительно вершины `v` (алгоритмом DFS), возвращает пару {расстояние до самой дальней вершины, индекс самой дальней вершины}.
- `int find_h()` – функция поиска глубины (диаметра) графа. Функция ищет самую дальнюю вершину от нулевой (пусть это  $v_1$ ), далее ищет самую дальнюю вершину уже от  $v_1$  (пусть это  $v_2$ ). Тогда диаметр графа — это расстояние между вершинами  $v_1$  и  $v_2$ .
- `void get_min_ost()` – нахождение минимального остова для данного графа по алгоритму Прима. Алгоритм заключается в добавлении на каждой итерации к уже имеющемуся остову ребра с

минимальным весом из всех таких, что один конец ребра лежит в остове, а другой – вне остова. Используется оптимизация с `std::set` (для быстрого поиска ребра с минимальным весом, выходящего из каждой вершины), и асимптотика алгоритма становится

$O(m \log n)$ , где  $m$  – количество ребер в графе.

- `void print()` – функция, выводящая в консоль список ребер свесами и матрицу смежности.

## Работа пользователя с приложением

Организовано интерактивное взаимодействие с пользователем через консоль. При запуске программы появляется ‘help’, описывающий возможные корректные команды от пользователя, а также предложение ввести количество вершин в будущем графе:

```
add <a> <b> <w>    - add the edge between [a] and [b] with w weight
min_ost             - print min spanning tree of graph
h                   - return the height of graph
print               - print the graph

Enter the number of vertices:
```

После введения количества вершин в графе пользователь может вводить любые комбинации команд, описанных в ‘help’.

## Код приложения

```
#include <iostream>
#include <vector>
#include <string>
#include <set>

class graph {
private:
    int n;
    std::vector <std::vector <int>> M;

public:
    graph(int _n = 0) {
        n = _n;
        if (n > 0) {
            M.resize(n);
            for (int i = 0; i < n; ++i) {
                M[i].resize(n, 0);
            }
        }
    }

    bool correct_ind(int a) {
        return 0 <= a && a < n;
    }

    void print_edge(int a, int b) {
        if (!correct_ind(a) || !correct_ind(b) || a == b) {
            return;
        }
        std::cout << "[" << a + 1 << " ]---" << M[a][b] << "---[" << b + 1 << "]\n";
    }

    std::string add_edge(int a, int b, int w) {
        if (!correct_ind(a) || !correct_ind(b) || a == b) {
            return "Error";
        }

        M[a][b] = w;
        M[b][a] = w;

        return "OK";
    }

    std::pair <int, int> find_distant_v(int v, std::vector <bool> &used) {
        used[v] = 1;
        int ans_h = 0;
        int ans_v = v;

        for (int i = 0; i < n; ++i) {
            if (!used[i] && M[v][i]) {
                std::pair <int, int> cur = find_distant_v(i, used);
                if (cur.first + 1 > ans_h) {
                    ans_h = cur.first + 1;
                    ans_v = cur.second;
                }
            }
        }

        return {ans_h, ans_v};
    }

    int find_h() {
```

```

std::vector<bool> used(n, 0);
std::pair<int, int> cur1 = find_distant_v(0, used);

for (int i = 0; i < n; ++i) {
    used[i] = 0;
}

std::pair<int, int> cur2 = find_distant_v(cur1.second, used);

return cur2.first;
}

void get_min_ost() {
    std::vector<int> min_edge(n, -1), end_edge(n, -1);
    std::set<std::pair<int, int>> q;

    q.insert({0, 0});
    min_edge[0] = 0;

    std::set<std::pair<int, int>> ost;
    int ans = 0;

    while (q.size()) {
        int v = q.begin()->second;
        q.erase(q.begin());

        if (end_edge[v] != -1) {
            int a = std::min(v, end_edge[v]);
            int b = std::max(v, end_edge[v]);
            ost.insert({a, b});
        }

        for (int i = 0; i < n; ++i) {
            if (M[v][i] == 0) {
                continue;
            }
            if (min_edge[i] == -1 || M[v][i] < min_edge[i]) {
                q.erase({min_edge[i], i});
                min_edge[i] = M[v][i];
                end_edge[i] = v;
                q.insert({min_edge[i], i});
            }
        }
    }

    for (auto i : ost) {
        ans += M[i.first][i.second];
        print_edge(i.first, i.second);
    }

    std::cout << "cost = " << ans << "\n";
}

void print() {
    for (int i = 0; i < n; ++i) {
        for (int j = i; j < n; ++j) {
            if (M[i][j]) {
                print_edge(i, j);
            }
        }
    }

    std::cout << "\n";
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            std::cout << M[i][j] << " ";
        }
    }
}

```

```

        std::cout << "\n";
    }
}

};

int main() {
    std::cout << "add <a> <b> <w>      - add the edge between [a] and [b] with w weight\n";
    std::cout << "min_ost              - print min spanning tree of graph\n";
    std::cout << "h                  - return the height of graph\n";
    std::cout << "print              - print the graph\n";
    std::cout << "\n";

    std::cout << "Enter the number of vertices: ";
    int n;
    std::cin >> n;

    std::cout << "_____ \n";

    graph G(n);

    while (1) {
        std::string op;
        std::cin >> op;

        std::cout << "\n";

        if (op == "add") {
            int a, b, w;
            std::cin >> a >> b >> w;
            std::cout << G.add_edge(a - 1, b - 1, w) << "\n";
        } else if (op == "print") {
            G.print();
        } else if (op == "h") {
            std::cout << "h = " << G.find_h() << "\n";
        } else if (op == "min_ost") {
            G.get_min_ost();
        } else {
            std::cout << "Error\n";
        }

        std::cout << "_____ \n";
    }

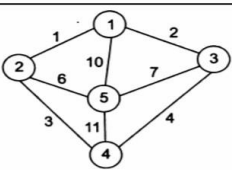
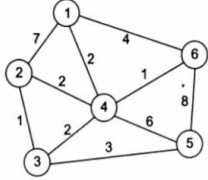
    return 0;
}

```



# Тестирование

Тестирование функции *min\_ost()*

№	Граф
1	
2	

## Тест 1:

```
Enter the number of vertices: 5
add 1 2 1
OK
add 1 3 2
OK
add 1 5 10
OK
add 2 5 6
OK
add 3 5 7
OK
add 5 4 11
OK
add 2 4 3
OK
add 4 3 4
OK
```

```
min_ost
[2]---1---[1]
[3]---2---[1]
[4]---3---[2]
[5]---6---[2]
cost = 12
```

Ввод

Вывод

## Тест 2:

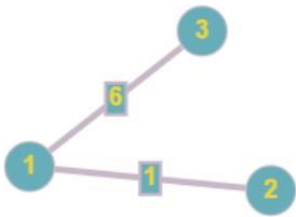
```
Enter the number of vertices: 6
add 1 2 7
OK
add 1 4 2
OK
add 1 6 4
OK
add 2 4 2
OK
add 4 6 1
OK
add 2 3 1
OK
add 3 4 2
OK
add 3 5 3
OK
add 4 5 6
OK
add 5 6 8
OK
```

```
min_ost
[1]---2---[4]
[2]---1---[3]
[2]---2---[4]
[3]---3---[5]
[4]---1---[6]
cost = 9
```

Ввод

Вывод

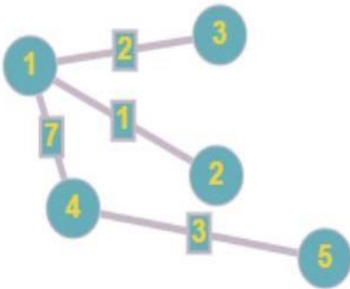
## Тестирование функции *print()*



```

Enter the number of vertices: 3
add 1 2 1
OK
add 1 3 6
OK
print
[1]---1---[2]
[1]---6---[3]
0 1 6
1 0 0
6 0 0
  
```

## Тестирование функции *find\_h()*



```

Enter the number of vertices: 5
add 1 2 1
OK
add 1 3 2
OK
add 1 4 7
OK
add 4 5 3
OK
h
h = 3
  
```

## **Выводы**

В результате проделанной работы, я получила практические навыки по выполнению операций над структурой данных граф. А также изучила основные алгоритмы работы с графами. На практике научилась определять глубину графа. Составлять программу реализации алгоритма построения остовного дерева методом поиска в ширину в неориентированном графе.

## **Список использованной литературы**

1. Лекционный материал по структурам и алгоритмам обработки данных Сартакова М. В. (дата обращения 01.11.2021)
2. Дополнительный материал к практическим работам по структурам и алгоритмам обработки данных Сорокина А. В. (дата обращения 01.11.2021)