|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт Информационных технологий | |
|  | |
| Кафедра инструментального и прикладного программного обеспечения | |
|  | |
|  | |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 7.2** | |
| **по дисциплине** | |
| **«**Графы: создание, алгоритмы обхода**»**  **Тема: «Нелинейные структуры»** | |
|  | |
| Выполнил студент группы ИКБО-10-23 | Харитонов А.Н. |
| Принял преподаватель | Макеева О.В. |

|  |  |  |
| --- | --- | --- |
| Лабораторная работа выполнена | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_ \_ \_\_202\_\_ г. | *(подпись руководителя)* |

Москва 2024

# **Цель работы**

Составить программу создания графа и реализовать процедуры для работы с графом.

# **Ход работы**

Для реализации графа были написаны классы Edge и Graph.

*struct* Edge {

*int* u, v, weight;

*bool* operator<(Edge const& *other*) { //для сравнивания

        return weight < *other*.weight;

    }

};

*class* Graph {

*private:*

*int* V; // количество вершин

    vector<Edge> edges; // список рёбер

Для реализации остального были написаны функции

*void* addEdge(*int* *u*, *int* *v*, *int* *weight*) {

        edges.push\_back({*u*, *v*, *weight*});

    }

    // Поиск вершины в множествах

*int* findSet(*int* *v*, vector<*int*>& *parent*) {

        if (*v* == *parent*[*v*]) return *v*;

        return *parent*[*v*] = findSet(*parent*[*v*], *parent*);

    }

    // добавляем вершину между 2 множествами без цикла (пересекаем)

*void* unionSets(*int* *a*, *int* *b*, vector<*int*>& *parent*, vector<*int*>& *rank*) {

*a* = findSet(*a*, *parent*);

*b* = findSet(*b*, *parent*);

        if (*a* != *b*) {

            if (*rank*[*a*] < *rank*[*b*])

                swap(*a*, *b*);

*parent*[*b*] = *a*;

            if (*rank*[*a*] == *rank*[*b*])

*rank*[*a*]++;

        }

    }

    //Основная идея - шаг за шагом накидывать ребра с мин весом, пока не получится так, что все вершины соединены

*void* kruskalMST() {

        sort(edges.begin(), edges.end()); // Сортируем рёбра по возрастанию веса

        vector<*int*> parent(V);

        vector<*int*> rank(V, 0);

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

            parent[i] = i;

        }

        vector<Edge> result;

*int* cost = 0;

        for (Edge e : edges) {

            if (findSet(e.u, parent) != findSet(e.v, parent)) {

                cost += e.weight;

                result.push\_back(e);

                unionSets(e.u, e.v, parent, rank);

            }

        }

        cout << "Minimal ost tree: " << endl;

        for (Edge e : result) {

            cout << e.u + 1 << " - " << e.v + 1 << " : " << e.weight << endl;

        }

        cout << "Whole: " << cost << endl;

    }

AddEdge – добавляет ребро

findSet – ищет вершину в множествах

unionSet – добавляет пересечение множествам

kruskalMST – основной алгоритм

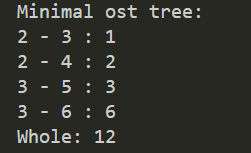


Рисунок 1. Тестирование программы

1. **Вывод**

В ходе работы я составил программу создания графа и реализовал процедуры для работы с графом