Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Российский химико-технологический университет имени Д.И. Менделеева»

Факультет цифровых технологий и химического инжиниринга

Кафедра информационных компьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

№ 5 ПО КУРСУ

«АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ»:

СТУДЕНТ группы КС-33 Костяева К.С.

Москва
2024

ОГЛАВЛЕНИЕ

ТЕОРИЯ

Граф — это структура данных, состоящая из вершин (узлов) и рёбер (связей).

Графы бывают:

- **Ориентированные** (рёбра имеют направление) и **неориентированные** (связь в обе стороны).
- **Взвешенные** (у рёбер есть "стоимость", например, расстояние) и **невзвешенные**.
- **Разреженные** (мало рёбер) и **плотные** (почти все вершины соединены).

Минимальное остовное дерево (MST - Minimum Spanning Tree)

Минимальное остовное дерево (МОД) – это подграф связного неориентированного графа, который:

1. **Связывает все вершины** графа.
2. **Не содержит циклов** (является деревом).
3. **Имеет минимальную возможную сумму весов рёбер**.

То есть, среди всех возможных остовных деревьев выбирается то, у которого сумма весов рёбер минимальна.

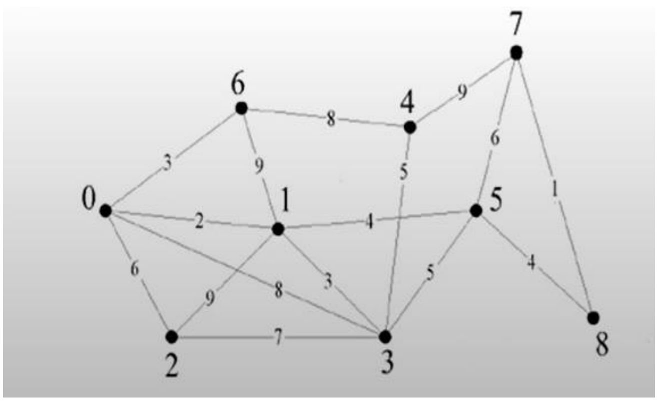
Алгоритм Краскала для поиска минимального остовного дерева

Алгоритм Краскала – это один из алгоритмов для нахождения минимального остовного дерева в графе.

Идея алгоритма

Отсортировать все рёбра по возрастанию их весов.
Добавлять рёбра одно за другим в остовное дерево, если они не образуют цикла.
Повторять процесс, пока в дереве не будет (n-1) рёбер, где n – количество вершин.

Вес	Ребро	Множество вершин	Вес дерева
		{0} {1} {2} {3} {4} {5} {6} {7} {8}	
1	7 – 8	{0} {1} {2} {3} {4} {5} {6} {7, 8}	1
2	0 – 1	{0, 1} {2} {3} {4} {5} {6} {7, 8}	3
3	0 – 6	{0, 1, 6} {2} {3} {4} {5} {7, 8}	6
3	1 – 3	{0, 1, 3, 6} {2} {4} {5} {7, 8}	9
4	1 – 5	{0, 1, 3, 5, 6} {2} {4} {7, 8}	13
4	5 – 8	{0, 1, 3, 5, 6, 7, 8} {2} {4}	17
5	3 – 4	{0, 1, 3, 4, 5, 6, 7, 8} {2}	22
5	3 – 5	образует цикл	-
6	0 – 2	{0, 1, 2, 3, 4, 5, 6, 7, 8}	28
6	5 – 7	конец алгоритма	-
7	2 – 3	-	-
8	0 – 3	-	-
8	4 – 6	-	-
9	1 – 2	-	-
9	1 – 6	-	-
9	4 - 7	-	-



ЗАДАНИЕ

вариант 4

---

1. Создайте взвешенный граф, состоящий из [10, 20, 50, 100] вершин.
   - Каждая вершина графа связана со случайным количеством вершин, минимум с [3, 4, 10, 20].
   - Веса ребер задаются случайным значением от 1 до 20.
   - Каждая вершина графа должна быть доступна, т.е. до каждой вершины графа должен обязательно существовать путь до каждой вершины, не обязательно прямой.

   > (Можно использовать генератор из предыдущей лабораторной работы.)

2. Выведите получившийся граф в виде матрицы смежности. Пример вывода данных:

	1	2	3	4	5
1		10	30	50	10
2					
3					10
4		40	20		
5	10		10	30	

3. Для каждого графа требуется провести серию из 5 - 10 тестов, в зависимости от времени затраченного на выполнение одного теста., необходимо:
   - **Вариант 1**. Найти кратчайшие пути между всеми вершинами графа и их длину с помощью алгоритма Дейкстры.
   - **Вариант 2**. Построить минимальное остовное дерево взвешенного связного неориентированного графа с помощью алгоритма Прима.
   - **Вариант 3**. Найти кратчайшие пути между всеми вершинами графа и их длину с помощью алгоритма Флойда — Уоршелла.
   - **Вариант 4**. Построить минимальное остовное дерево взвешенного связного неориентированного графа с помощью алгоритма Краскала.
4. В рамках каждого теста, необходимо замерить потребовавшееся время на выполнение задания из пункта 3 для каждого набора вершин. По окончанию всех тестов необходимо построить график используя полученные замеры времени, где на ось абсцисс (X) нанести N – количество вершин, а на ось ординат(Y) - значения затраченного времени.

ПРАКТИЧЕСКАЯ ЧАСТЬ

Код:

язык программирования C++

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <random>
#include <chrono>
#include <locale>  // Для поддержки русских символов
#include <set>     // Для работы с set

using namespace std;
using namespace std::chrono;

struct Edge {
    int u, v, weight;
    bool operator<(const Edge& other) const {
        return weight < other.weight;
    }
};

// Класс графа
class Graph {
private:
    int V; // Количество вершин
    vector<Edge> edges; // Список рёбер

public:
    Graph(int vertices) : V(vertices) {}

    void addEdge(int u, int v, int weight) {
        edges.push_back({ u, v, weight });
    }

    void generateRandomGraph(int minEdges, int maxWeight) {
        random_device rd;
        mt19937 gen(rd()); //Генерация случайных весов
        uniform_int_distribution<int> weightDist(1, maxWeight);

        // Генерация связности графа: делаем минимально связным
       // Каждая вершина i соединяется хотя бы с одной из предыдущих
        for (int i = 1; i < V; ++i) {
            int parent = rand() % i;
            int weight = weightDist(gen);
            addEdge(parent, i, weight);
        }

        // Гарантируем минимум 3 рёбер для каждой вершины
        for (int i = 0; i < V; ++i) {
            int numEdges = minEdges + rand() % (V - minEdges);   // Случайное количество
рёбер
            set<int> addedNeighbors;  // Чтобы избежать повторных рёбер

            // Строим рёбра, избегая самосоединений
            for (int j = 0; j < numEdges; ++j) {
                int neighbor = rand() % V;
                if (neighbor != i && addedNeighbors.find(neighbor) ==
addedNeighbors.end()) {
                    int weight = weightDist(gen);
                    addEdge(i, neighbor, weight);
                    addedNeighbors.insert(neighbor);
                }
            }
        }
    }
```

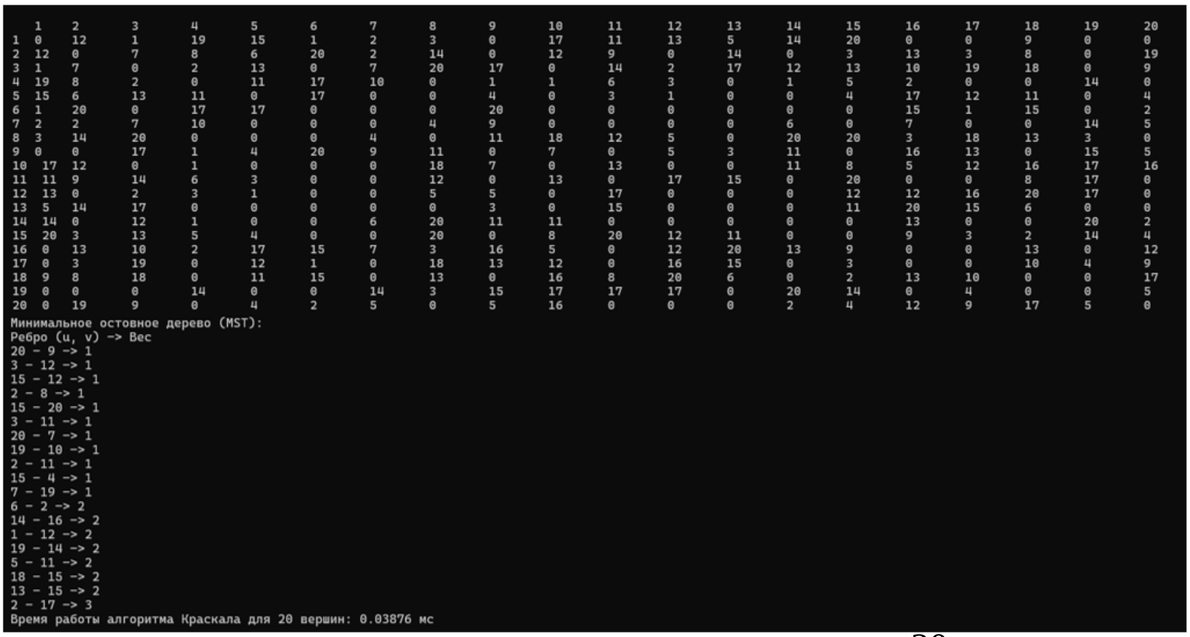
```
    vector<Edge> kruskalMST() {
        vector<Edge> mst;
        sort(edges.begin(), edges.end()); //Сортируем рёбра по весу

        vector<int> parent(V); //Поиск и объединение множеств
        for (int i = 0; i < V; i++)
            parent[i] = i;

        auto find = [&](int v) { //Функция find (поиск корня множества)
            while (v != parent[v]) v = parent[v];
            return v;
            };

        auto unite = [&](int a, int b) { //Функция unite (объединение двух множеств)
            parent[find(a)] = find(b);
            };

        for (const Edge& edge : edges) { //Обход всех рёбер и построение остовного дерева
            if (find(edge.u) != find(edge.v)) { // принадлежат разным компонентам
                mst.push_back(edge);
                unite(edge.u, edge.v);
            }
        }
        return mst;
    }

    void printAdjacencyMatrix() {
        vector<vector<int>> matrix(V, vector<int>(V, 0));

        // Заполняем матрицу весами рёбер
        for (const Edge& edge : edges) {
            matrix[edge.u][edge.v] = edge.weight;
            matrix[edge.v][edge.u] = edge.weight; // Граф неориентированный
        }

        // Выводим таблицу
        cout << "\n   "; // Отступ перед заголовком
        for (int i = 0; i < V; i++) {
            cout << i + 1 << "\t"; // Выводим номера столбцов
        }
        cout << "\n";

        // Перебор строк и столбцов для вывода
        for (int i = 0; i < V; i++) {
            cout << i + 1 << "  "; // Выводим номера строк
            for (int j = 0; j < V; j++) {
                // Добавляем условие для корректного вывода чисел
                if (matrix[i][j] == 0)
                    cout << "0\t";
                else
                    cout << matrix[i][j] << "\t";
            }
            cout << "\n";
        }
    }

    double measureKruskalTime() { //замер времени работы алгоритма
        auto start = high_resolution_clock::now();
        kruskalMST();
        auto stop = high_resolution_clock::now();
        return duration<double, milli>(stop - start).count();
    }
};

void printMST(const vector<Edge>& mst) {
    cout << "\nМинимальное остовное дерево (MST):\n";
    cout << "Ребро (u, v) -> Вес\n";
```

```
    for (const Edge& edge : mst) {
        cout << edge.u + 1 << " - " << edge.v + 1 << " -> " << edge.weight << "\n";
    }
}

// Главная функция
int main() {
    setlocale(LC_ALL, ""); // Установка русской локали

    vector<int> sizes = { 10, 20, 50, 100 }; //Определяем размеры графов
    vector<double> times; // Вектор для хранения времени

    for (int size : sizes) {
        Graph g(size);
        g.generateRandomGraph(3, 20);
        g.printAdjacencyMatrix();

        double avgTime = 0;
        int tests = 5;
        for (int i = 0; i < tests; i++) {
            avgTime += g.measureKruskalTime();
        }
        avgTime /= tests;
        times.push_back(avgTime);

        // Вычисляем MST и выводим его
        vector<Edge> mst = g.kruskalMST();
        printMST(mst);

        cout << "Время работы алгоритма Краскала для " << size << " вершин: " << avgTime
<< " мс\n";
    }

    cout << "\nДанные для построения графика:\n";
    cout << "N (кол-во вершин) -> Время (мс)\n";
    for (size_t i = 0; i < sizes.size(); i++) {
        cout << sizes[i] << " -> " << times[i] << " мс\n";
    }

    return 0;
}
```

Результат работы программы:

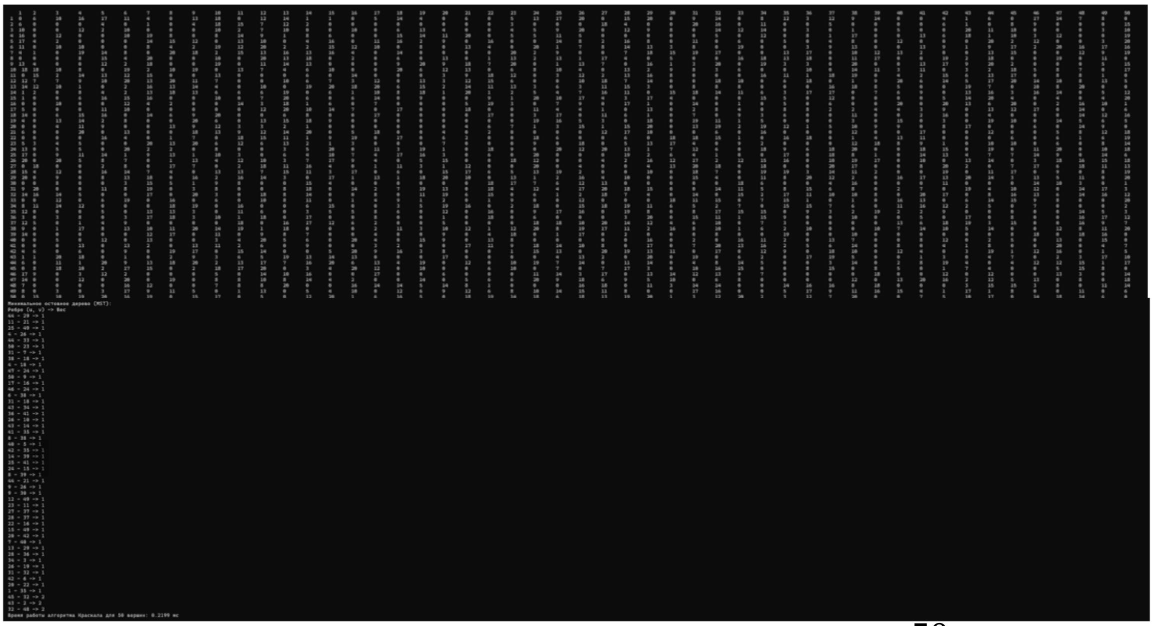
	1	2	3	4	5	6	7	8	9	10
1	0	15	3	0	6	1	0	3	18	0
2	15	0	15	5	15	0	12	0	0	11
3	3	15	0	16	0	2	2	8	3	15
4	0	5	16	0	12	1	0	0	0	11
5	6	15	0	12	0	20	12	18	7	10
6	1	0	2	1	20	0	0	6	9	5
7	0	12	2	0	12	0	0	5	1	14
8	3	0	8	0	18	6	5	0	19	17
9	18	0	3	0	7	9	1	19	0	3
10	0	11	15	11	10	5	14	17	3	0

матрица смежности и минимальное остовное дерево для 10 вершин



```
   1   2    3    4    5    6    7    8    9    10   11   12   13   14   15   16   17   18   19   20
1  0   12   1    19   15   1    2    3    0    17   11   13   5    14   20   0    0    9    0    0
2  12  0    7    8    6    20   2    14   0    12   9    0    14   0    3    13   3    8    0    19
3  1   7    0    2    13   0    7    20   17   0    14   2    17   12   13   10   19   18   0    9
4  19  8    2    0    11   17   10   0    1    1    6    3    0    1    5    2    0    0    14   0
5  15  6    13   11   0    17   0    0    4    0    3    1    0    0    4    17   12   11   0    4
6  1   20   0    17   17   0    0    0    20   0    0    0    0    0    0    15   1    15   0    2
7  2   2    7    10   0    0    0    4    9    0    0    0    0    6    0    7    0    0    14   5
8  3   14   20   0    0    0    4    0    11   18   12   5    0    20   20   3    18   13   3    0
9  0   0    17   1    4    20   9    11   0    7    0    5    3    11   0    16   13   0    15   5
10  17  12   0    1    0    0    0    18   7    0    13   0    0    11   8    5    12   16   17   16
11  11  9    14   6    3    0    0    12   0    13   0    17   15   0    20   0    0    8    17   0
12  13  0    2    3    1    0    0    5    5    0    17   0    0    0    12   12   16   20   17   0
13  5   14   17   0    0    0    0    0    3    0    15   0    0    0    11   20   15   6    0    0
14  14  0    12   1    0    0    6    20   11   11   0    0    0    0    0    13   0    0    20   2
15  20  3    13   5    4    0    0    20   0    8    20   12   11   0    0    9    3    2    14   4
16  0   13   10   2    17   15   7    3    16   5    0    12   20   13   9    0    0    13   0    12
17  0   3    19   0    12   1    0    18   13   12   0    16   15   0    3    0    0    10   4    9
18  9   8    18   0    11   15   0    13   0    16   8    20   6    0    2    13   10   0    0    17
19  0   0    0    14   0    0    14   3    15   17   17   17   0    20   14   0    4    0    0    5
20  0   19   9    0    4    2    5    0    5    16   0    0    0    2    4    12   9    17   5    0
Минимальное остовное дерево (MST):
Ребро (u, v) -> Вес
20 - 9 -> 1
3 - 12 -> 1
15 - 12 -> 1
2 - 8 -> 1
15 - 20 -> 1
3 - 11 -> 1
20 - 7 -> 1
19 - 10 -> 1
2 - 11 -> 1
15 - 4 -> 1
7 - 19 -> 1
6 - 2 -> 2
14 - 16 -> 2
1 - 12 -> 2
19 - 14 -> 2
5 - 11 -> 2
18 - 15 -> 2
13 - 15 -> 2
2 - 17 -> 3
Время работы алгоритма Краскала для 20 вершин: 0.03876 мс
```

матрица смежности и минимальное остовное дерево для 20 вершин



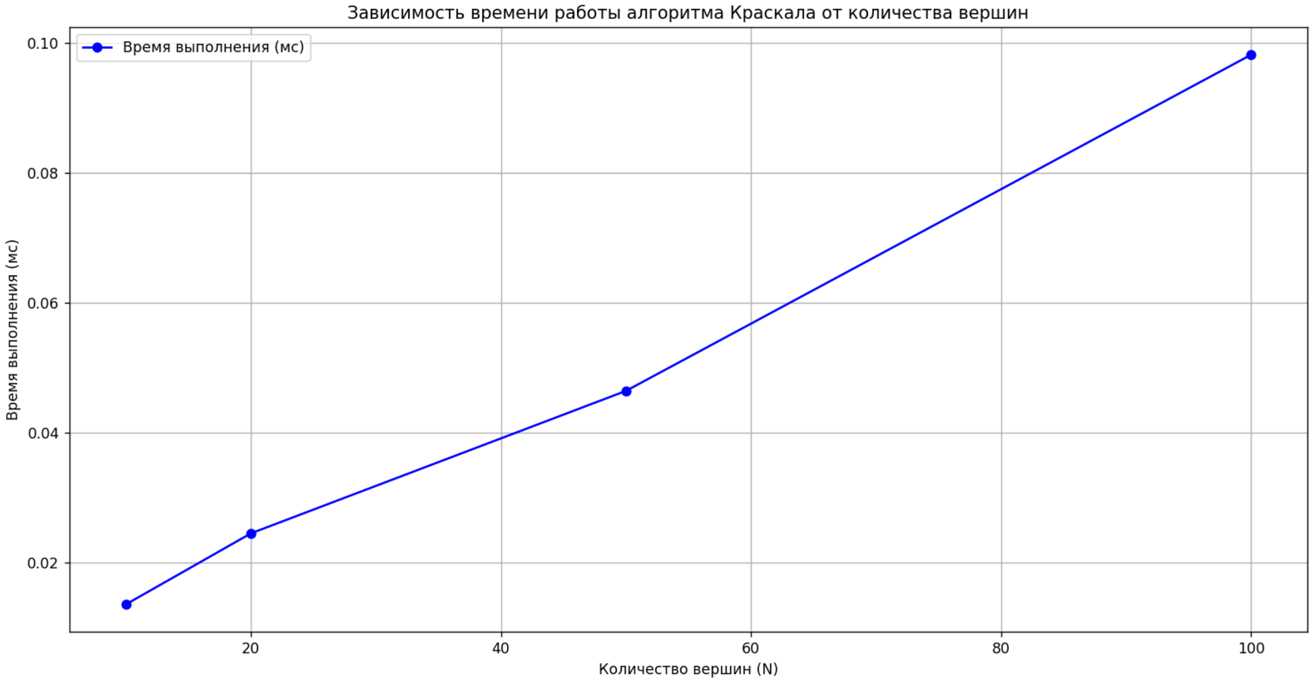
матрица смежности и минимальное остовное дерево для 50 вершин

матрица смежности и минимальное остовное дерево для 100 вершин

```
Данные для построения графика:
N (кол-во вершин) -> Время (мс)
10 -> 0.0209 мс
20 -> 0.04828 мс
50 -> 0.2746 мс
100 -> 1.07148 мс
```

Графики:

язык программирования Python



Ось X (абсцисс) → N (количество вершин)
Ось Y (ординат) → Время (мс)

ВЫВОД

- В рамках данной работы была реализована программа на C++, которая:
- Генерирует взвешенный, связный, неориентированный граф с заданным количеством вершин (10, 20, 50, 100).
- Создаёт матрицу смежности, гарантируя, что каждая вершина имеет минимум 3-20 связей и доступна из любой другой.
- Реализует алгоритм Краскала для нахождения минимального остовного дерева (MST).
- Замеряет время выполнения алгоритма Краскала и усредняет его по 5 тестам для каждого размера графа.
- Выводит матрицы смежности в виде таблицы.
- Получаем результаты на основе которых строим график зависимости времени работы алгоритма от количества вершин. Проанализировав полученные данные, можно сказать, что количества вершин влияет на скорость работы алгоритма. Чем больше количество вершин (N), тем больше времени требуется на выполнение алгоритма Краскала.