Министерство образования и науки РФ
Пермский национальный исследовательский политехнический университет
Электротехнический факультет
Кафедра информационных технологий и автоматизированных систем

Дискретная математика и математическая логика

Лабораторная работа № 6

Тема: «Построение остовного дерева по алгоритму Краскала»

Выполнил: студент группы
ИВТ-23-1Б
Долганова Диана Евгеньевна
Проверил: ст. пр.
Рустамханова Г. И.

г. Пермь – 2025

Оглавление

Цель работы

Разработать программу, которая будет строить остовное дерево по алгоритму Краскала.

Задачи работы

Построение остовного дерева Алгоритм смотреть по таблице (Прима или Краскала):

1. На вход подается матрица расстояний размером 10× 10
2. На выходе вес минимального остовного дерева.
3. Само дерево можно представить:
    a. Перечислением ребер
    b. Матрица смежности, исключив лишние ребра из основной матрицы смежности.
4. Остов должен быть единой компонентой связности.

Этапы выполнения

Программа была разработана на языке C++ в программе среде Microsoft Visual Studio 2022.

1. Структура ребра графа

Структура Edge представляет ребро графа. u и v — это вершины, которые соединяет ребро, weight — вес ребра. Конструктор инициализирует поля структуры (рис. 1).

```
struct Edge {
    int u;
    int v;
    int weight;
    Edge(int u, int v, int weight) : u(u), v(v), weight(weight) {}
};
```

Рисунок 1 - Структура Edge

2. Сортировка ребер

Функция compareEdges (рис. 2) используется для сортировки рёбер по возрастанию веса. Она передается в sort для упорядочивания рёбер.

```
bool compareEdges(const Edge& a, const Edge& b) {
    return a.weight < b.weight;
}
```

Рисунок 2 - Функция compareEdges

3. Работа с непересекающимися множествами

Класс DSU (рис. 3) реализует структуру данных для работы с непересекающимися множествами (Disjoint Set Union):

- parent — массив, где parent[i] хранит родителя вершины i.
- rank — массив для оптимизации объединения множеств (ранг вершины).

- find — находит корень множества, к которому принадлежит вершина x, с использованием эвристики сжатия пути.
- unite — объединяет множества, содержащие вершины x и y, с использованием ранговой эвристики.

```cpp
class DSU {
private:
    vector<int> parent;
    vector<int> rank;
public:
    DSU(int n) {
        parent.resize(n);
        rank.resize(n, 0);
        for (int i = 0; i < n; ++i) {
            parent[i] = i;
        }
    }

    int find(int x) {
        if (parent[x] != x) {
            parent[x] = find(parent[x]);
        }
        return parent[x];
    }

    void unite(int x, int y) {
        int rootX = find(x);
        int rootY = find(y);
        if (rootX != rootY) {
            if (rank[rootX] < rank[rootY]) {
                parent[rootX] = rootY;
            }
            else {
                parent[rootY] = rootX;
                if (rank[rootX] == rank[rootY]) {
                    rank[rootX]++;
                }
            }
        }
    }
};
```

Рисунок 3 - Класс DSU

4. Чтение матрицы из файла

Функция readMatrixFromFile (рис. 4) читает матрицу из файла и выводит ошибку, если файл невозможно прочитать.

```cpp
vector<vector<int>> readMatrixFromFile(const string& filename) {
    ifstream file(filename);
    if (!file.is_open()) {
        cerr << "Ошибка открытия файла!" << endl;
        exit(1);
    }

    vector<vector<int>> matrix(10, vector<int>(10));
    for (int i = 0; i < 10; ++i) {
        for (int j = 0; j < 10; ++j) {
            file >> matrix[i][j];
        }
    }

    file.close();
    return matrix;
}
```

Рисунок 4 - Функция readMatrixFromFile

5. Главная функция

5.1 Настройка локали и чтение файла

Добавляем строчку для корректного отображения русского языка и читаем файл с матрицей (рис. 5).

```cpp
int main() {
    setlocale(LC_ALL, "RU");
    string filename = "g21.txt"; // Имя файла с матрицей
    vector<vector<int>> matrix = readMatrixFromFile(filename);
```

Рисунок 5 - Локаль и чтение файла

5.2 Создание списка ребер

Создается список ребер графа. Нулевые значение в матрице игнорируются (отсутствие ребер) (рис. 6).

```cpp
vector<Edge> edges;
for (int i = 0; i < 10; ++i) {
    for (int j = i + 1; j < 10; ++j) {
        if (matrix[i][j] != 0) { // Игнорируем нулевые рёбра
            edges.emplace_back(i, j, matrix[i][j]);
        }
    }
}
```

Рисунок 6 - Создание списка ребер

5.3 Сортировка ребер

Ребра сортируются по возрастанию веса (рис. 7).

```cpp
sort(edges.begin(), edges.end(), compareEdges);
```

Рисунок 7 - Сортировка ребер

5.4 Алгоритм Краскала

Используем алгоритм Краскала:

1. Создается объект DSU для работы с множествами.
2. Проходим по отсортированным рёбрам и добавляем их в минимальное остовное дерево (MST), если они не создают цикл.
3. Если количество рёбер в MST достигает 9 (для графа с 10 вершинами), алгоритм завершается.
4. Если MST содержит меньше 9 рёбер, граф несвязный, и программа завершается с ошибкой (рис. 8).

```cpp
DSU dsu(10);
vector<Edge> mst;
int totalWeight = 0;

for (const Edge& e : edges) {
    if (dsu.find(e.u) != dsu.find(e.v)) {
        mst.push_back(e);
        totalWeight += e.weight;
        dsu.unite(e.u, e.v);
        if (mst.size() == 9) break;
    }
}

if (mst.size() != 9) {
    cerr << "Граф несвязный. Построение минимального остовного дерева невозможно." << endl;
    return 1;
}
```

Рисунок 8 - алгоритм Краскала

5.5 Вывод результатов

Выводим ребра, сумму весов ребер, матрицу смежности (рис. 9).

```
    // Вывод суммы весов рёбер
    cout << "Сумма весов рёбер минимального остовного дерева: " << totalWeight << endl;

    // Вывод рёбер остовного дерева
    cout << "\nРёбра остовного дерева:" << endl;
    for (const Edge& e : mst) {
        cout << e.u << " - " << e.v << " (вес " << e.weight << ")" << endl;
    }

    // Создание и вывод матрицы смежности остовного дерева
    vector<vector<int>> adjacencyMatrix(10, vector<int>(10, 0));
    for (const Edge& e : mst) {
        adjacencyMatrix[e.u][e.v] = e.weight;
        adjacencyMatrix[e.v][e.u] = e.weight;
    }

    cout << "\nМатрица смежности остовного дерева:" << endl;
    for (const auto& row : adjacencyMatrix) {
        for (int val : row) {
            cout << val << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Рисунок 9 - Вывод результатов

Тестирование программы

Для проверки программы на корректность добавим файл с матрицей и запустим программу (рис. 10 - 11).

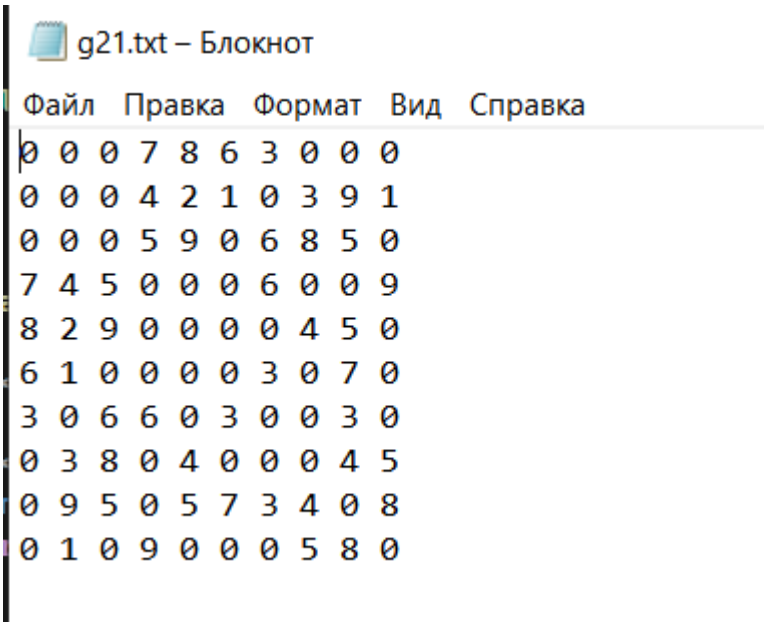

Рисунок 10 - Первая матрица

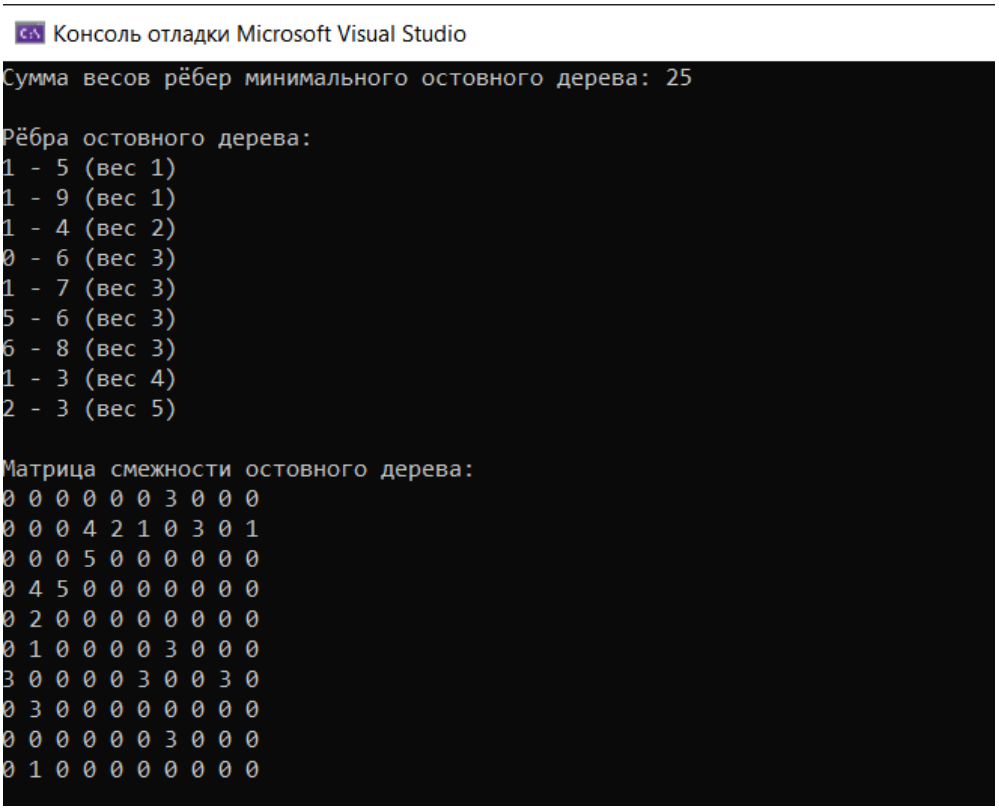

Рисунок 11 - Результат

Заключение

Была разработана программа для построения остовного дерева по алгоритму Краскала.

Список используемой литературы

1. Алгоритм Краскала, Прима для нахождения минимального остовного дерева / Хабр URL: https://habr.com/ru/articles/569444/ (дата обращения: 18.03.2025).
2. Построение минимального остовного дерева • Информатика, Теория графов • Фоксфорд Учебник URL: foxford.ru/wiki/informatika/postroenie-minimalnogo-ostovnogo-dereva (дата обращения: 18.03.2025).