**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ФКТИ**

Курсовая РАБОТА

**по дисциплине «Алгоритмы и Структуры Данных»**

Тема: Алгоритм Краскала

|  |  |  |
| --- | --- | --- |
| Студент гр. 3376 |  | Михайлов Н.Ф. |
| Преподаватель |  | Молдовян Д.Н. |

Санкт-Петербург

2024

**ЗАДАНИЕ**

**на курсовую работу**

|  |
| --- |
| Студент Михайлов Н.Ф. |
| Группа 3376 |
| Тема работы: Поиск минимального отстовного дерева на основе алгоритма Краскала |
| Исходные данные:  Реализовать алгоритм поиска минимального остова на основе алгоритма Краскала, а также знать, как работают обходы, сортировки, разные методы хранения графов и системы непересекающихся множеств. |

**Аннотация**

Курсовая работа посвящена анализу методов реализации графов и работы с ними, а также анализу и реализации алгоритма Краскала для поиска минимального остова в графе.

**содержание**

[ВВЕДЕНИЕ 5](#_Toc185599243)

[1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ 6](#_Toc185599244)

[1.1 Понятие графов и их представление 6](#_Toc185599245)

[1.2 Алгоритм Краскала (Крускала) 7](#_Toc185599246)

[2. ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ 9](#_Toc185599247)

[2.1. Разработка алгоритмов и структур данных 9](#_Toc185599248)

[2.2. Тестирование программы 14](#_Toc185599249)

[ЗАКЛЮЧЕНИЕ 18](#_Toc185599250)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 19](#_Toc185599251)

ВВЕДЕНИЕ

В ходе данной работы будут изучены и реализованы следующие алгоритмы и структуры данных:

1. Сортировка графа
2. Обходы графа в длину и ширину
3. Алгоритм Краскала
4. Методы хранения графа такие как:
   1. Список смежности
   2. Матрица инцидентности
   3. Матрица смежности
5. Граф на основе матрицы смежности
6. Система непересекающихся множеств

Таким образом, курсовая работа направлена на изучение как алгоритма Краскала, так и на изучение и реализацию графов как таковых.

1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ

* 1. Понятие графов и их представление

Графы являются одной из основополагающих структур данных в дискретной математике и компьютерных науках. Формально граф определяется как пара , где — конечное множество вершин (или узлов), а — множество рёбер (или дуг), соединяющих пары вершин. Графы используются для моделирования различных процессов и объектов, включая социальные сети, транспортные системы, компьютерные сети и многие другие. В зависимости от структуры и свойств графов можно выделить несколько классификаций:

**Ориентированные и неориентированные графы:** В ориентированных графах рёбра имеют направление (например, от вершины A к вершине B), в то время как в неориентированных графах рёбра не имеют направления и представляют симметричные отношения.

**Взвешенные и невзвешенные графы:** Взвешенные графы имеют значения (веса) на рёбрах, которые могут представлять стоимость, расстояние или другие метрики. В невзвешенных графах рёбра не имеют дополнительных атрибутов.

**Связные и несвязные графы:** Граф считается связным, если существует путь между любой парой вершин. Несвязные графы содержат вершины, между которыми нет путей.

**Циклические и ациклические графы:** Циклические графы содержат циклы (замкнутые пути), тогда как ациклические графы-cycleless графы не содержат циклов. В частности, ориентированные ациклические графы (Directed Acyclic Graphs, DAG) часто используются в задачах, связанных с зависимостями.

#### Представление графов:

Существуют различные способы представления графов, каждый из которых имеет свои преимущества и недостатки в зависимости от контекста их применения:

**Матрица смежности:** это квадратная матрица размера , где — число вершин графа. Элемент в матрице равен 1 (или весу рёбра), если существует рёбер между вершинами и , в противном случае — 0.

* 1. Алгоритм Краскала (Крускала)

Алгоритм Крускала — это эффективный метод нахождения минимального остовного дерева (МОД или MST) в неориентированном взвешенном графе. Он был предложен в 1956 году болгарским математиком Вацлавом Крускалом. Алгоритм работает на основе жадного принципа, который последовательно добавляет рёбра в остовное дерево, выбирая при этом наименьшие рёбра, не образующие циклов. Алгоритм Крускала можно описать следующими шагами:

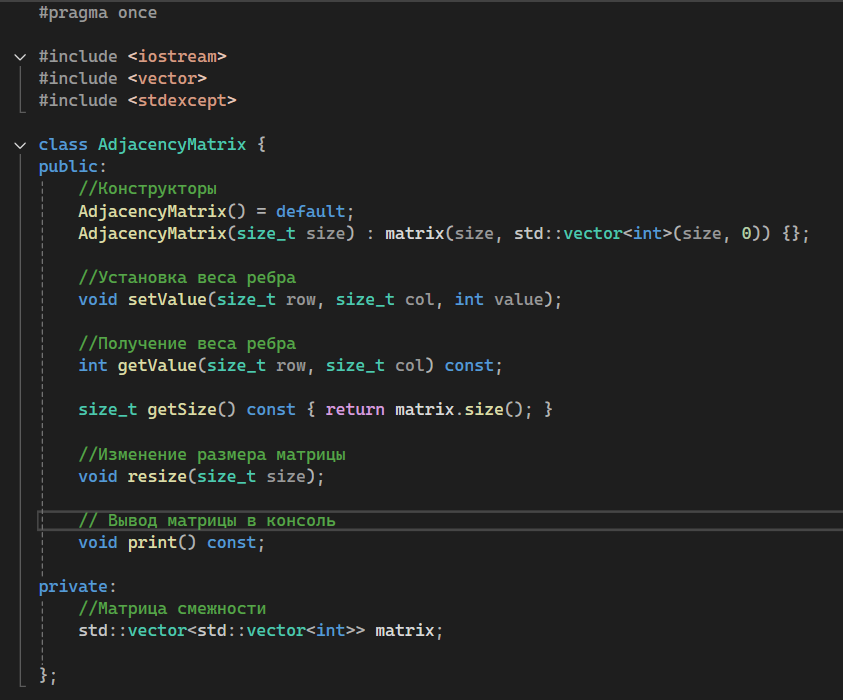
1. **Инициализация**: создаём пустое множество для хранения рёбер минимального остовного дерева. Формируем список всех рёбер графа, отсортированных по возрастанию веса.
2. **Выбор рёбер**: поочерёдно рассматриваем рёбра в отсортированном порядке. Для каждого рёбра проверяем, образует ли оно цикл с уже добавленными рёбрами. Это можно сделать с помощью структуры данных, поддерживающей информацию о компонентах связности (например, с помощью системы непересекающихся множеств Union-Find).
3. **Добавление рёбер**: если текущее ребро не образует цикл, добавляем его в остовное дерево. Процесс повторяется, пока не будет рёбер, где — общее количество вершин в графе.
4. **Завершение**: Алгоритм завершает свою работу, когда добавлено рёбер в остовное дерево, что гарантирует, что все вершины связаны.

2. ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ

2.1. Разработка алгоритмов и структур данных

Для работы алгоритма Краскала реализуем граф с помощью матрицы смежности (так требует задание).

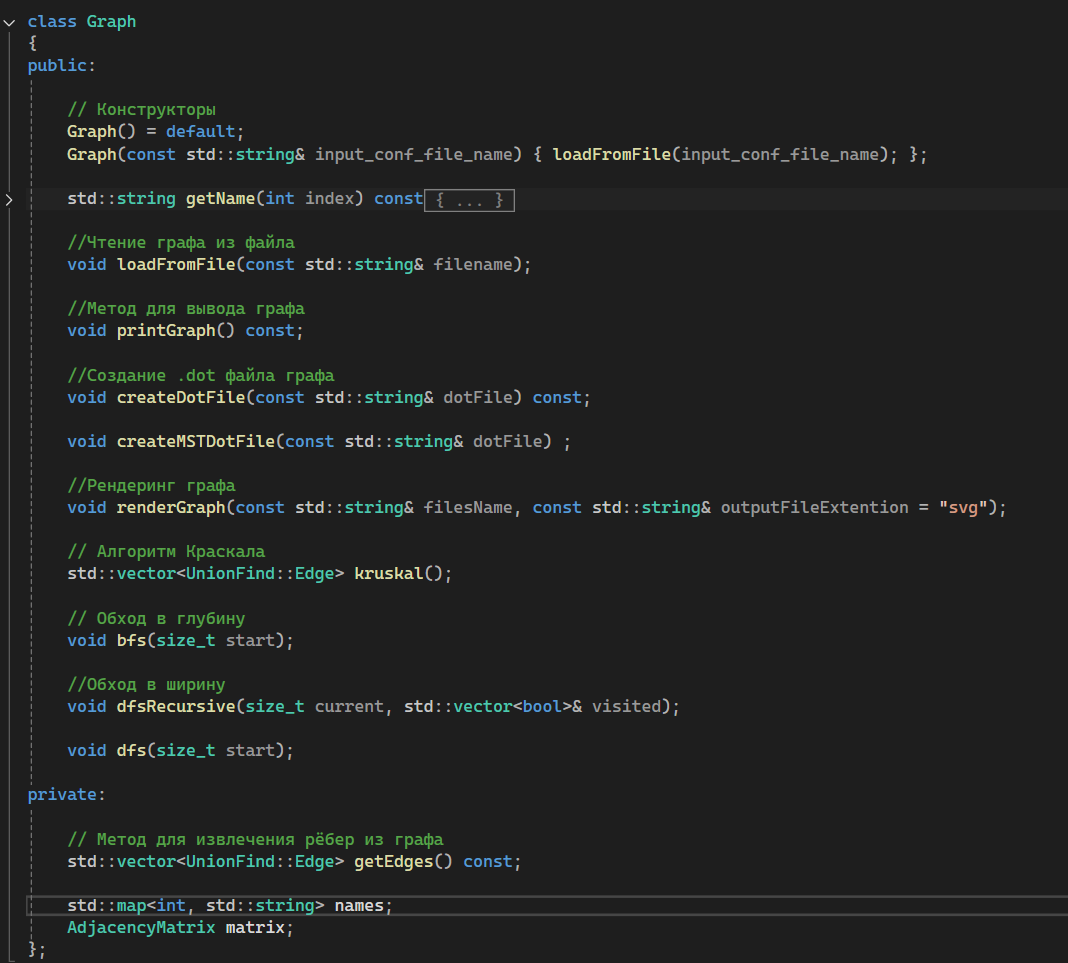
Для удобства напишем класс AdjacencyMatrix, на котором в дальнем будет работать наш граф.

****

*Рис. 1. Описание класса AdjacencyMatrix*

std::vector<std::vector<int>> matrix — матрица, которая хранит веса путей, а всё остальное — методы для удобной работы с ней (фактическую их реализацию я опустил, т.к. считаю что она сейчас не особо нужна для понимания).

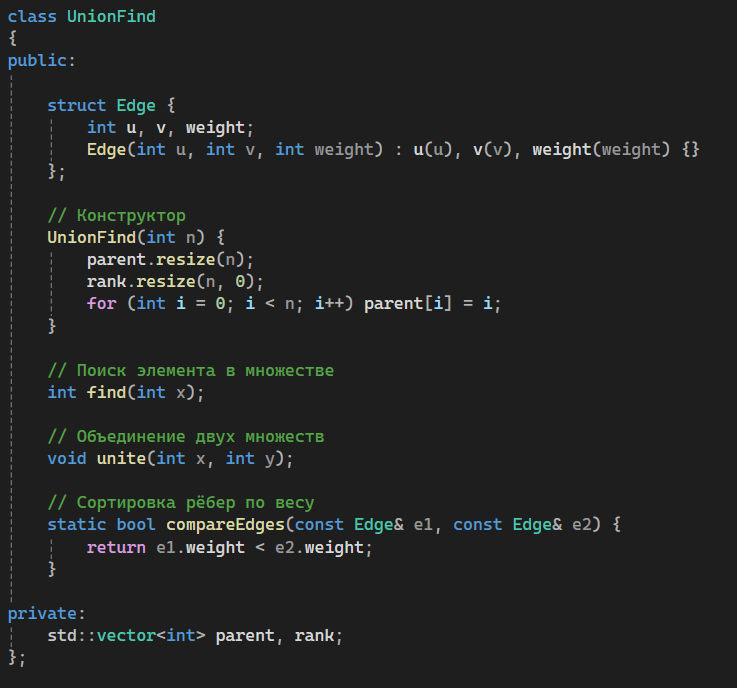
Далее реализуем сам граф.

*Рис. 2. Описание класса Graph*

В классе Graph у нас есть матрица смежности, std::map для удобного хранения имён узлов, а так не множество полезных методов, таких как:

1. Чтение графа из файла, как требует задание.
2. Создание .dot файлов и их последующий рендеринг для красивого отображение графа.
3. Сам алгоритм Краскала.
4. Обходы в ширину и глубину.
5. Метод для получения рёбер графа (используется в алгоритме Краскала).

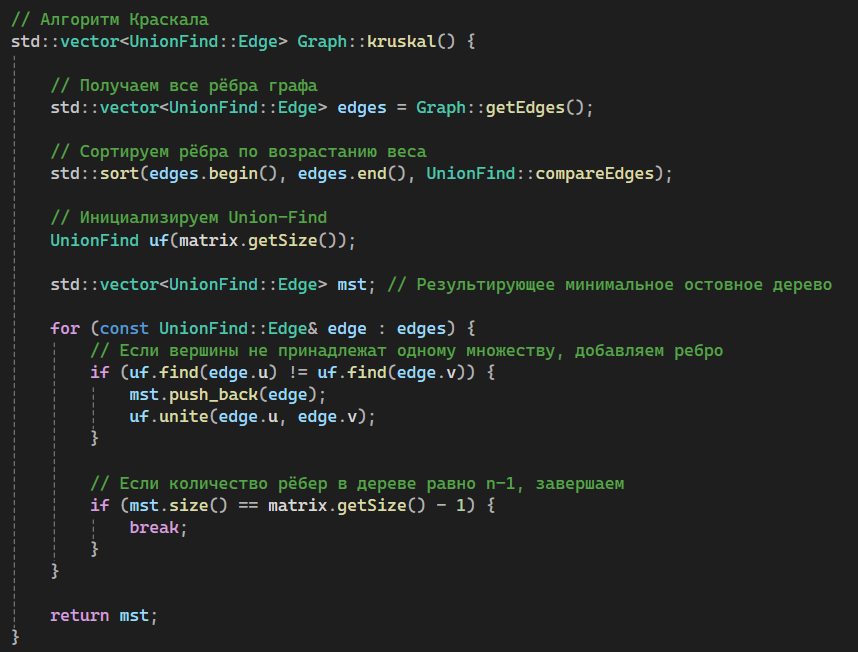
Так же для реализации алгоритма поиска минимального остовного дерева нам понадобится такая структура как Union-Find. Реализуем и её.

*Рис. 2. Описание класса* *UnionFind*

В классе UnionFind есть:

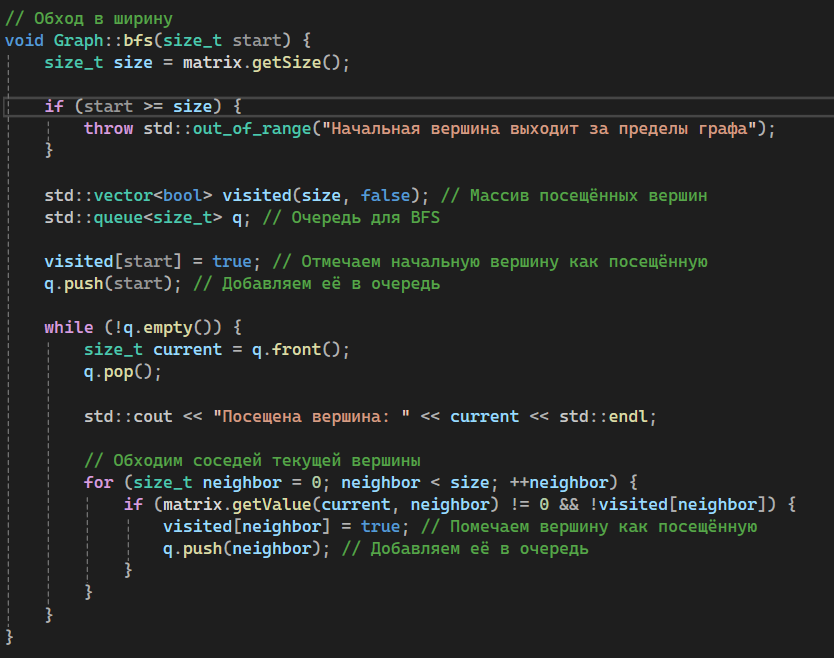
1. Структура, описывающая ребро.
2. Конструктор, создающий множество определённой длины.
3. Методы поиска элемента и объединения множеств.
4. Метод для сравнения рёбер по весу, чтобы позже использовать в std::cort().

Основные классы, которые нужно было реализовать для выполнения задания мы рассмотрели. Теперь рассмотрим работу алгоритма Краскала и ещё некоторые методы:

1. Алгоритм Краскала

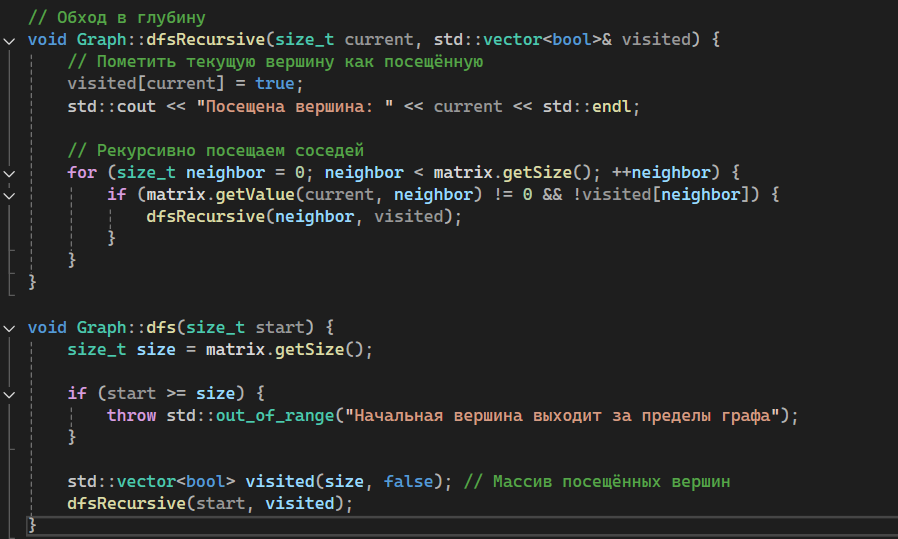
*Рис. 3. Реализация алгоритма Краскала*

* 1. Получаем вектор всех рёбер графа.
  2. Сортируем рёбра по возрастанию веса.
  3. Создаёт объект типа UnionFind размера матрицы смежности графа.
  4. Создаём итоговый вектор из рёбер mst.
  5. Проходим по всем рёбрам и сравниваем на принадлежность одному множеству. Если они не принадлежат, то добавляем ребро.
  6. Завершаем работу алгоритма, когда рёбер осталось .

1. Обход в ширину (bfs):

*Рис. 4. Реализация обхода в ширину*

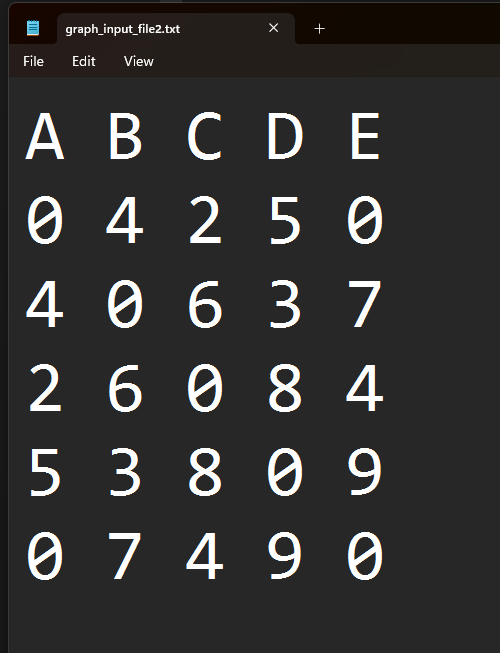
* 1. Получаем размер матрицы смежности.
  2. Создаём массив посещённых вершин.
  3. Создаём очередь.
  4. Берём первую вершину, добавляем её в очередь и отмечаем её как посещённую.
  5. Далее до тех пор, пока очередь не будет пуста выполняем следующие действия:
     1. Выбираем первую вершину в очереди.
     2. Проходим всех соседей этой вершины слева направо
     3. Отмечаем их посещёнными и добавляем в массив посещённых вершин.

1.  Обход в глубину(dfs):

*Рис. 5. Реализация обхода в глубину*

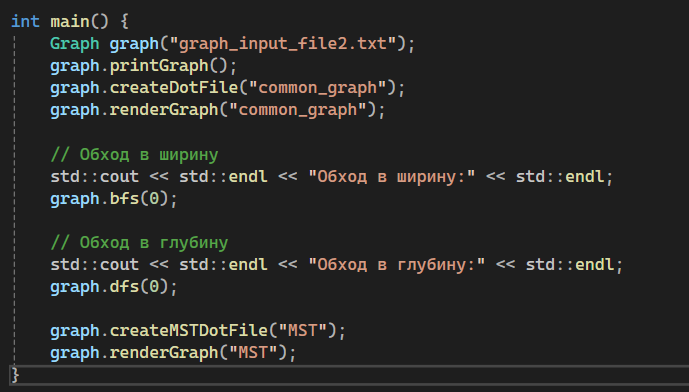
* 1. Получаем размер матрицы смежности.
  2. Создаём вектор посещённых вершин.
  3. Вызываем рекурсивную функцию обхода в глубину:
     1. Записываем вершину в массив посещённых.
     2. Рекурсивно посещаем всех соседей в глубину.

2.2. Тестирование программы

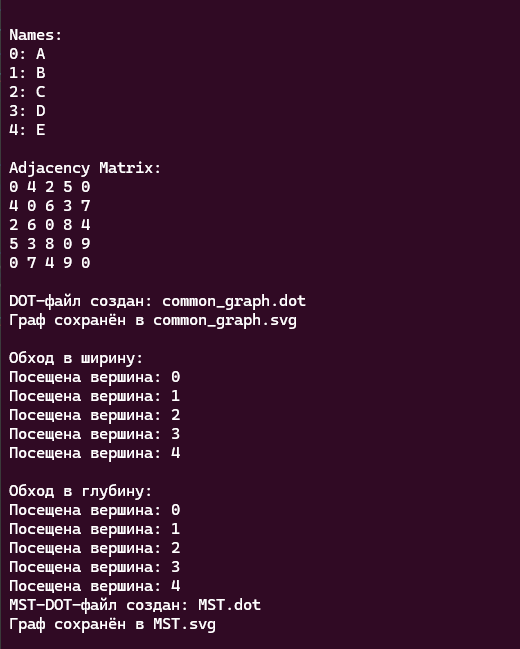
На рисунке 6 приведено содержимое файла graph\_input\_file2.txt.

*Рис. 6. Матрица смежности для графа*

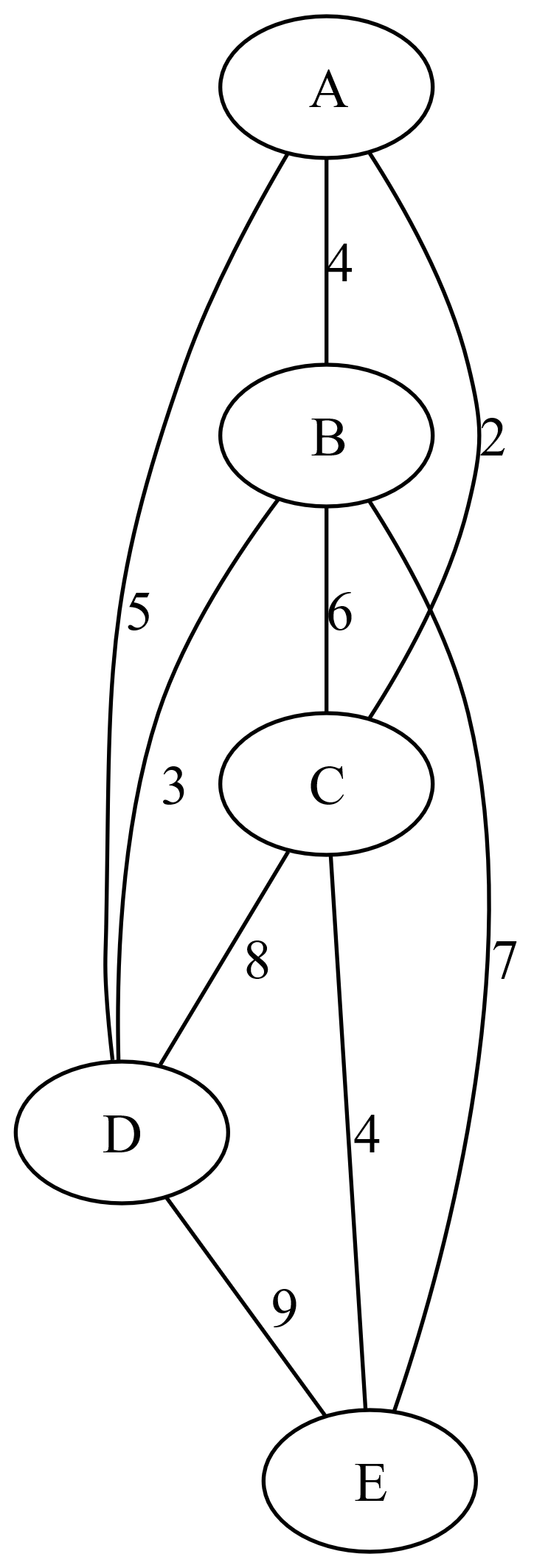
Запустим программу, прочитаем файл, выведем матрицу смежности в консоль для проверки, отрендерим граф в виде векторной картинки common\_graph.svg, проделаем два обхода и отрендерим минимальное остовное дерево в виде векторной картинки MST.svg.



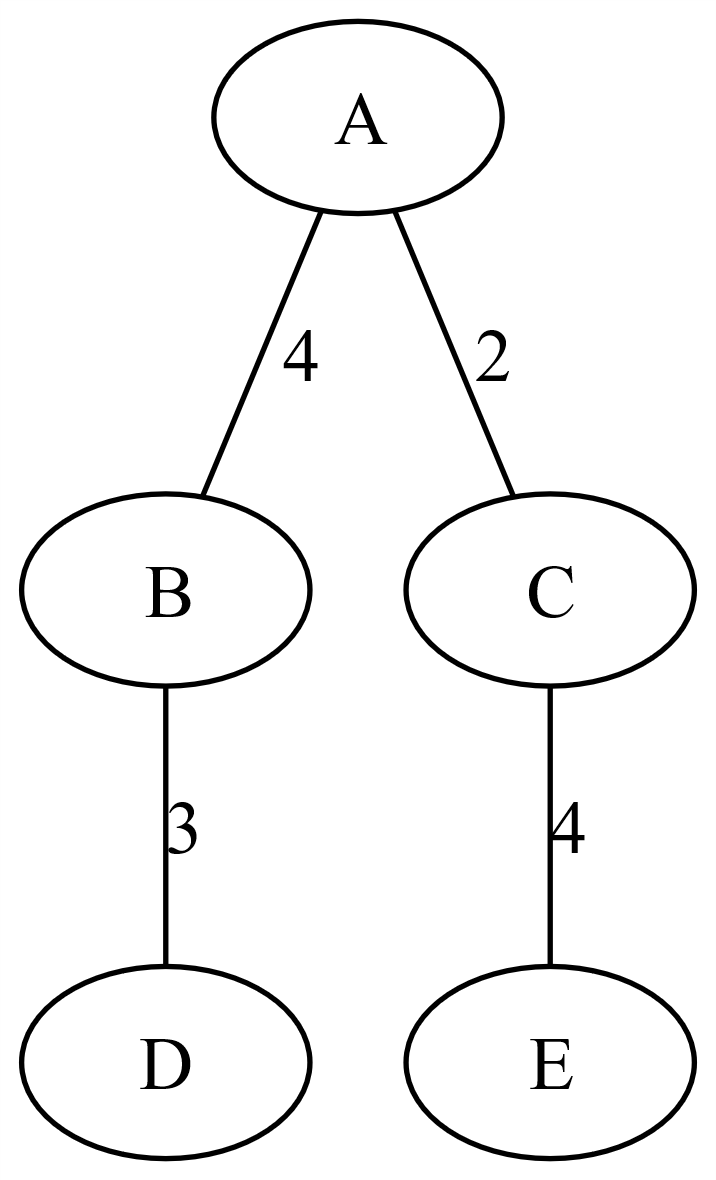
*Рис. 7. Главная функция программы*

Как видно из консольного вывода программы, матрица смежности и имена узлов прочитались верно. А также обходы отработали, как и ожидалось.

*Рис. 8. Консольные результаты работы программы*

Далее посмотрим, как выглядят отрендереные граф и минимальное остовное дерево.

*Рис.9. Отрендереный граф*

*Рис. 10. Минимальное остовное дерево*

Программа отработала верно, что граф, что минимальное остовное дерево прочитаны/найдены верно.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы изучение данного алгоритма позволило глубже понять жадные алгоритмы и их применение в практических задачах. По итогам курсовой работы были реализованы:

1. Все способы хранения графа:
   1. Матрица смежности.
   2. Матрица инцидентности.
   3. Список смежности.
2. Граф.
3. Система непересекающихся множеств, она же Union-Find.
4. Алгоритм Крускала.
5. Обходы графа в ширину и глубину.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Алгоритмы на графах: Дейкстры, Прима, Краскала // Сайт Максима Пелевина. URL: <https://markoutte.me/students/graphs/> (дата обращения: 016.12.2024).