МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего

образования Национальный исследовательский технологический университет «МИСИС»

Институт компьютерных наук

Кафедра автоматизированнх систем управления

**Курсовая работа**

**«Задача построения кратчайшего связывающего дерева. Алгоритм Краскала»**

дисциплина «Алгоритмы дискретной математики»

|  |  |
| --- | --- |
| Выполнил: | Проверил: |
| Хайдукова М. И. (ФИО студента) | Гласов Александр (преподаватель) |
| БИВТ-23-7  (№ группы) |
| .2025  (дата сдачи) |
|  |  |

**ВВЕДЕНИЕ**

**Актуальность темы исследования.** Задача построения кратчайшего связывающего дерева (минимального остовного дерева) является одной из фундаментальных в теории графов и находит широкое применение в различных областях, таких как проектирование компьютерных и телекоммуникационных сетей, электрических цепей, транспортных систем, а также при решении задач кластеризации и маршрутизации.

Алгоритм Краскала, являющийся одним из классических жадных алгоритмов, предлагает простой и эффективный метод построения минимального остовного дерева. Благодаря своей модульности и удобству реализации, алгоритм широко применяется как в академических исследованиях, так и в практических инженерных задачах.

Современные технологии требуют обработки всё больших объёмов данных и работы со сложными графовыми структурами. В этой связи изучение и применение алгоритма Краскала сохраняет свою актуальность. Кроме того, на фоне стремительного роста интереса к распределённым системам и интеллектуальному анализу данных, вопросы оптимизации сетей и структуры соединений становятся особенно важными.

**Цель** **работы**: реализовать алгоритм Краскала для построения минимального остовного дерева, изучить его принципы работы и визуализировать процесс формирования дерева на основе заданных графов.

**Объектом исследования** являются неориентированные взвешенные графы, для которых решается задача построения минимального остовного дерева.

**Предметом исследования** являются алгоритмы построения минимального остовного дерева, в частности алгоритм Краскала и его применение к взвешенным графам.

**Задачи:**

1. Изучить теоретические основы задачи построения минимального остовного дерева
2. Ознакомиться с принципами работы алгоритма Краскала и проанализировать его вычислительную сложность
3. Разработать программную реализацию алгоритма Краскала
4. Реализовать визуализацию пошагового построения остовного дерева
5. Протестировать работу алгоритма на различных графах и проанализировать результаты

**Актуальность работы:**

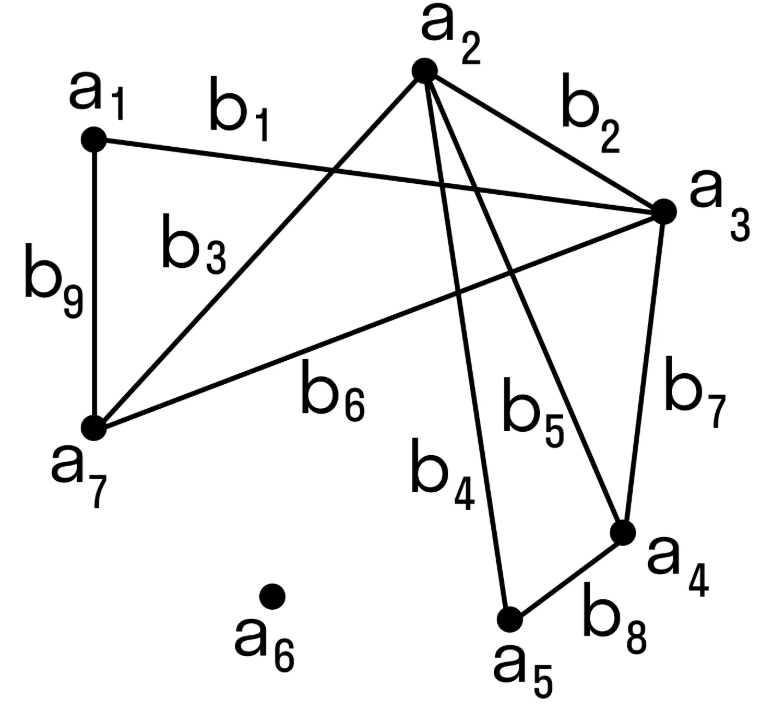
Задача построения минимального остовного дерева имеет широкое применение в таких областях, как проектирование сетей, логистика, телекоммуникации и информационные технологии. Актуальность исследования обусловлена потребностью в оптимизации структуры связей в различных системах, а также важностью визуализации алгоритмических процессов для лучшего понимания и обучения.

**Глава 1. Теоретические основы задачи построения минимального остовного дерева**

**1.1. Основные понятия теории графов**

Графы: Математический объект, состоящий из множества вершин и множества ребер (дуг), соединяющих некоторые пары вершин. Используются для моделирования связей и отношений.

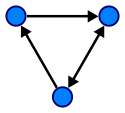
Для чего строят графы: чтобы отобразить отношения на множествах. По сути, графы помогают визуально представить всяческие сложные взаимодействия: аэропорты и рейсы между ними, разные отделы в компании, молекулы в веществе.

  
*Рис. 1. Пример графа*

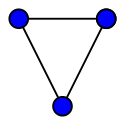
В данном случае точки — это вершины графа, а связки — рёбра графа.

Графом называется система объектов произвольной природы (вершин) и связок (ребер), соединяющих некоторые пары этих объектов.

Рассмотрим ориентированность. Ориентированный граф - граф, где множество ребер представляет собой набор **упорядоченных** пар, то есть ребро (u, v) означает, что из вершины u можно напрямую попасть в вершину v, но при этом ничего не говорит о возможности попасть из вершины v в вершину u.

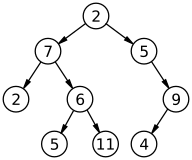
  
*Рис. 2. Ориентированный граф*

Неориентированный граф - граф, где множество вершин представляет собой неупорядоченный набор пар, то есть наличие ребра (u, v) означает, что из вершины u можно попасть в вершину v и наоборот.

  
*Рис. 3. Неориентированный граф*

Связный граф — граф, содержащий ровно одну компоненту связности. Это означает, что между любой парой вершин этого графа существует как минимум один путь.

Дерево - связный граф, не содержащий циклов.

  
*Рис. 4. Пример дерева*

Рассмотрим **способы** **представления графов.**

Матрица смежности - двумерный массив, где на месте [i][j] стоит 0 или 1. 1 в случае, если существует ребро из вершины i в вершину j, 0 - если не существует.

Список смежности - массив, в каждой ячейке которого хранится список/множество вершин, для которых существует ребро в них из исходной.

**1.2. Постановка задачи построения остовного дерева**

Рассмотрим следующую задачу. Дан неориентированный взвешенный связный граф. Необходимо выбрать в нем подграф, являющийся деревом, содержащий все вершины графа, имеющий минимальную сумму всех ребер.

Интерпретировать эту задачу можно так: есть некоторое количество городов, между которыми можно проложить дороги различной стоимости (протяженности). Необходимо выбрать набор дорог, соединяющий все города, минимальной суммарной стоимости. Эта задача называется задачей построения минимального остовного дерева или каркаса. При этом у графа может быть несколько различных минимальных остовных деревьев.

Для решения этой задачи есть два известных алгоритма: алгоритм Прима и Краскала. У этих алгоритмов общий подход — алгоритм добавляет некоторые ребра графа по одному так, что в любой момент выбранные ребра составляют часть некоторого минимального остовного дерева, то есть всегда можно уже выбранные ребра завершить до минимального остовного дерева. Алгоритм продолжается пока не будет выбрано ребро, образующее дерево (при этом нужно следить, чтобы выбранные ребра в каждый момент не образовывали циклов). Но принцип выбора очередного ребра в каждом графе отличается.

**1.3. Сравнительный обзор алгоритмов (Краскал, Прим, Борувка и др.)**

Существует несколько классических алгоритмов для решения задачи построения минимального остовного дерева, среди которых наиболее известны алгоритмы Краскала, Прима и Борувки. Каждый из них имеет свои особенности, преимущества и области применения.

Алгоритм Краскала основан на жадном подходе и заключается в сортировке всех рёбер графа по возрастанию веса и последовательном добавлении рёбер в остовное дерево при условии, что их включение не создаёт циклов. Такой метод хорошо подходит для разреженных графов и легко реализуется с помощью структур данных, таких как объединение и поиск (Union-Find). Алгоритм обладает относительно низкой сложностью и удобен для понимания и визуализации.

Алгоритм Прима, напротив, начинает построение остовного дерева с произвольной вершины и на каждом шаге добавляет к текущему дереву ребро с минимальным весом, которое соединяет уже построенную часть с новой вершиной. Такой подход хорошо работает для плотных графов, так как требует постоянного обновления множества рёбер, выходящих из уже включённых вершин.

Алгоритм Борувки (также известный как алгоритм Sollin) строит минимальное остовное дерево путем одновременного выбора минимального ребра для каждой компоненты графа на каждом шаге, после чего компоненты объединяются. Этот алгоритм хорошо масштабируется и может эффективно работать в параллельных вычислениях.

Выбор конкретного алгоритма зависит от структуры графа, его размера и требований к производительности. Алгоритм Краскала часто предпочитают за простоту реализации и эффективность на разреженных графах, алгоритм Прима — для плотных графов, а алгоритм Борувки — для задач с возможностью параллелизации.

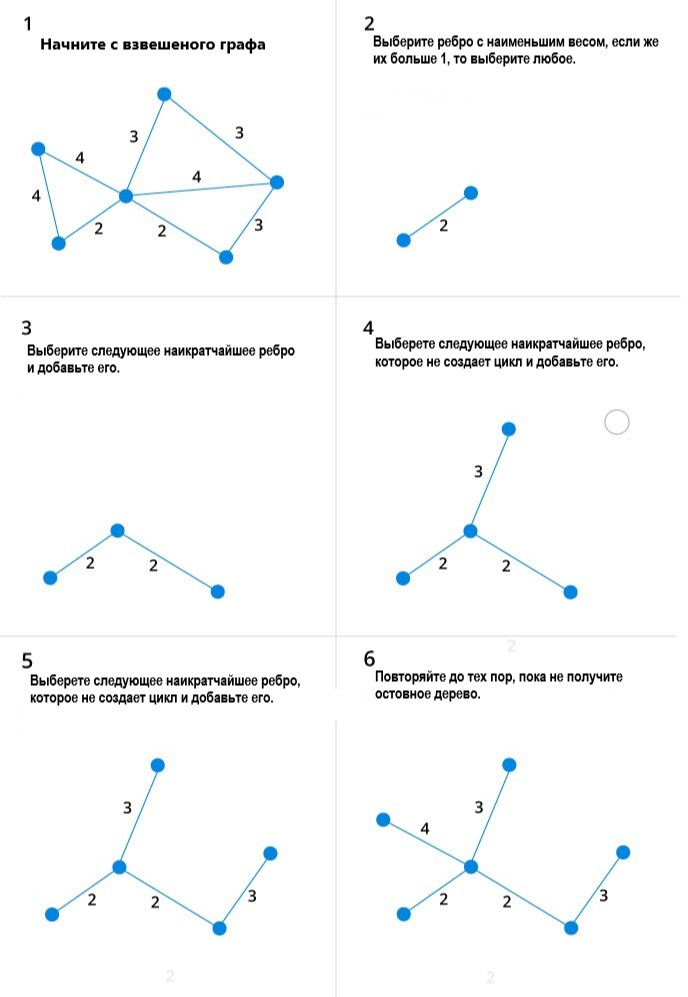
По ресурсам алгоритм Краскала работает за O(E log E) за счёт сортировки рёбер и операций объединения, при этом требует памяти для хранения рёбер и структур Union-Find. Он эффективен для разреженных графов.

Алгоритм Прима при использовании приоритетной очереди имеет сложность около O(E log V) и требует больше памяти для хранения списков смежности и очереди, что выгодно для плотных графов.

Алгоритм Борувки тоже работает за O(E log V) и хорошо подходит для параллельной обработки, с умеренными требованиями к памяти.

**Глава 2. Алгоритм Краскала**

**2.1. Описание алгоритма**

  
*Рис. 5. Пример алгоритма Краскала*

В алгоритме Краскала весь единый список ребер упорядочивается по не убыванию весов ребра. Далее ребра перебираются от ребер с меньшим весом к большему, и очередное ребро добавляется к каркасу, если оно не образовывает цикла с ранее выбранными ребрами. В частности, первым всегда выбирается одно из ребер минимального веса в графе.

Для проверки того, что выбранные ребра не образовывает цикл, будем представлять граф, как объединение нескольких компонент связности. В самом начале, когда ни одно ребро графа не выбрано, каждая вершина является отдельной компонентой связности. По мере добавления новых ребер компоненты связности будут объединяться, пока не получится одна общая компонента связности. Пронумеруем все компоненты связности и для каждой вершины будем хранить номер ее компоненты связности, таким образом, в самом начале для каждой вершины номер ее компоненты связности будет равен номеру самой вершины, а в конце у всех вершин будут одинаковые номера компоненты связности, которой они принадлежал.

При рассмотрении очередного ребра посмотрим номера компонент связности, соответствующих концам этого ребра. Если эти номера совпадают, то ребро соединяет две вершины, уже лежащие в одной компоненте связности, поэтому добавление этого ребра образовывает цикл. Если же ребро соединяет две разные компоненты связности, например, с номерами a и b, то ребро добавляется к части основного дерева, а эти две компоненты связности объединяются вместе. Для этого можно, например, всем вершинам, которые раньше находились в компоненте b изменить номер компоненты на a.

Сложность такого алгоритма O(MlogM + N2), так как алгоритм сначала сортирует все ребра, затем просматривает ребра по одному, и при этом N - 1 раз производится объединение двух компонент связности, что требует просмотра всего списка вершин.

**2.2. Пошаговый разбор работы алгоритма**

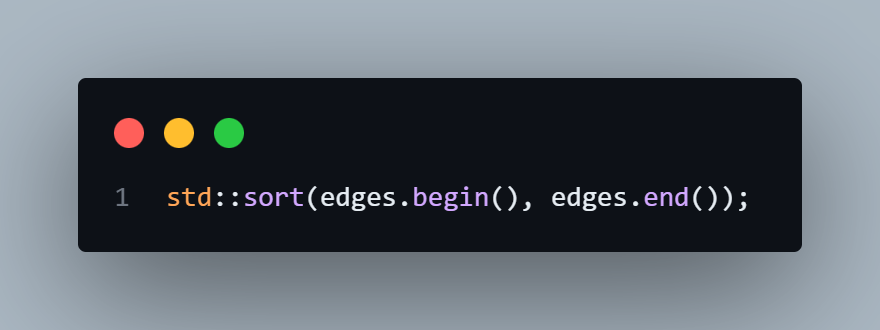
*Рис. 6. Реализация алгоритма на C++*



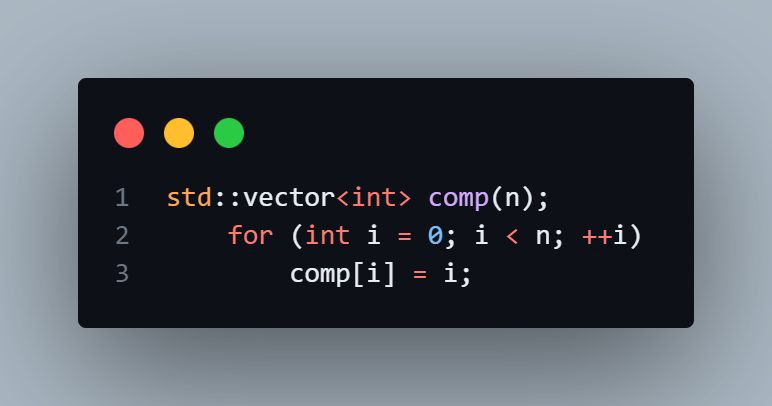
Алгоритм реализует построение минимального остовного дерева (МОД) с помощью жадного подхода. Входными данными являются количество вершин n и количество рёбер m, после чего считываются сами рёбра. Каждое ребро представлено в формате: вес, начальная и конечная вершины.

  
*Рис. 7. Ввод данных и хранение рёбер*

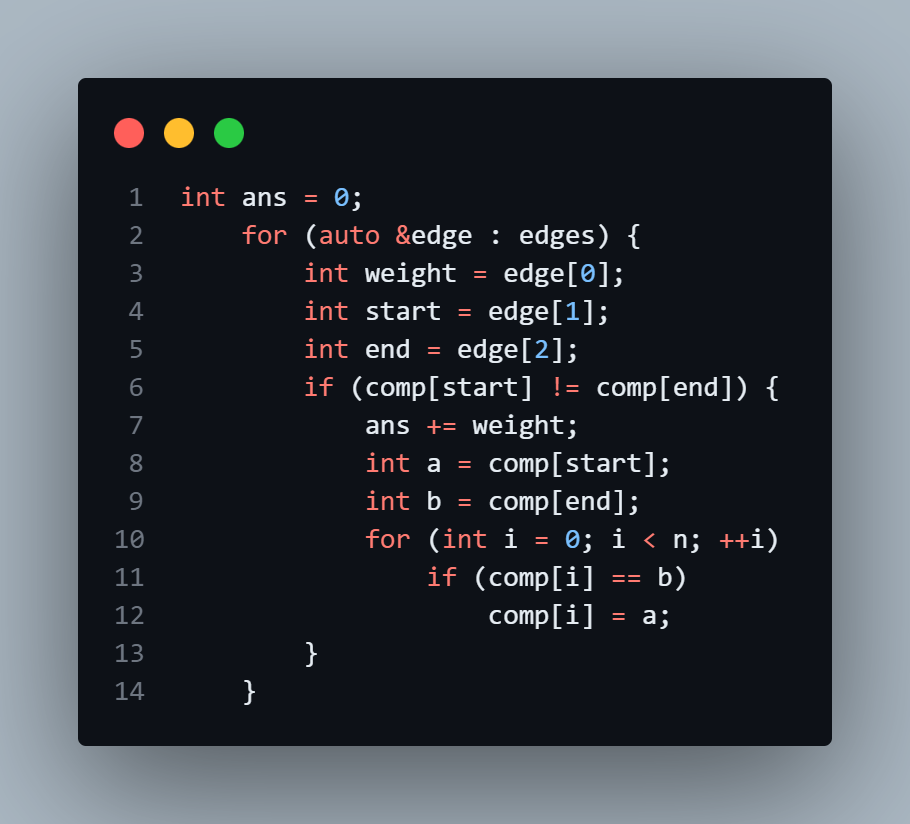
Рёбра сохраняются в векторе edges, где каждое ребро — это вектор из трёх элементов: вес, вершина A, вершина B. Индексация вершин уменьшается на 1 для перехода к нумерации с нуля.

  
*Рис. 8. Сортировка рёбер по возрастанию веса*

Все рёбра сортируются по весу. Это ключевая особенность алгоритма Краскала — сначала обрабатываются наилегчайшие рёбра.

  
*Рис. 9. Инициализация компонент связности*

Создаётся массив comp, где для каждой вершины задаётся номер её компоненты связности (изначально каждая вершина — отдельная компонента).

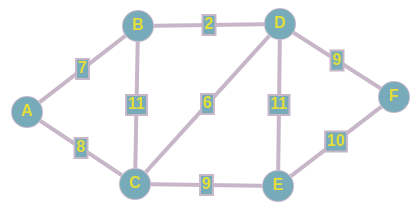
  
*Рис. 10. Обработка рёбер и построение остовного дерева*

Алгоритм проходит по рёбрам в порядке возрастания веса. Если вершины start и end принадлежат разным компонентам (не соединены), ребро добавляется в остовное дерево, а компоненты объединяются: все вершины, принадлежавшие компоненте b, приравниваются к компоненте a.

После завершения цикла переменная ans содержит суммарный вес минимального остовного дерева — ответ на задачу.

В данной реализации используется простая структура для хранения компонент связности (массив без дерева или рангов), что делает код наглядным, но менее эффективным на больших графах. Более оптимальной была бы реализация через структуру *Disjoint Set Union* (DSU, «система непересекающихся множеств») с эвристиками сжатия пути и объединения по рангу.

**2.3. Разбор конкретного примера**

*Рис. 11. Исходный граф*

Из представленного сверху графа, выпишем все его ребра в отсортированном порядке:

1) D <--> B; w = 2

2) D <--> C; w = 6

3) A <--> B; w = 7

4) A <--> C; w = 8

5) C <--> E; w = 9

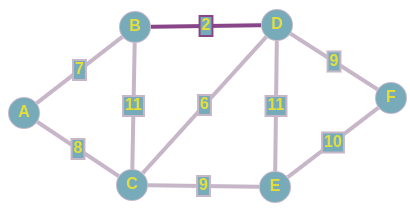
6) D <--> F; w = 9

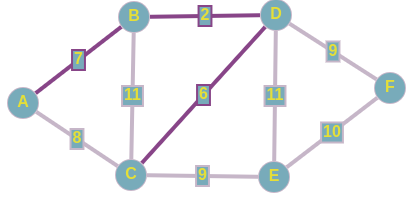
7) F <--> E; w = 10

8) B <--> C; w = 11

9) D <--> E; w = 11

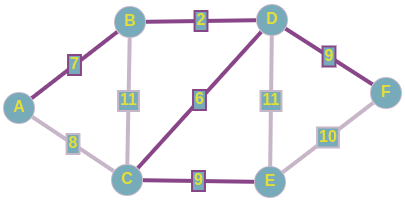
И начнем по списку добавлять эти ребра в наш остов:

  
*Рис. 12. Подграф после добавления 1-го ребра*

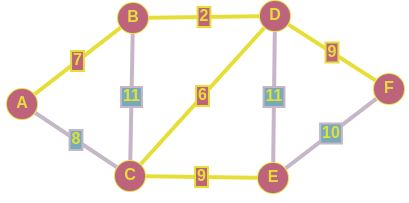
  
*Рис. 13. Подграф после добавления 2-го и 3-го рёбер*

При добавлении в наше остовное дерево ребра A <--> C, как вы можете заметить, образовывается цикл, поэтому мы просто пропускаем данное ребро.

По итогу у нас образовывается следующий подграф, и как вы заметили, мы соединили все вершины ребрами с минимально-возможными весами, а значит, нашли минимальное остовное дерево для нашего исходного графа.

  
*Рис. 14. Минимальный остов*

И из-за того, что при равенстве весов рёбер мы можем выбрать любое из них, конечные подграфы, являющиеся минимальными остовными деревьями, могут различаться с точностью до некоторых рёбер.

  
*Рис. 15. Провели проверку*

**Глава 3. Реализация и визуализация алгоритма Краскала**

**2.1. Выбор среды программирования и инструментов**

Для реализации и визуализации алгоритма Краскала была выбрана среда программирования Python версии 3.8 и выше. Основными причинами выбора Python являются наличие специализированных библиотек для работы с графами и визуализацией: библиотека networkx предоставляет удобный интерфейс для создания и обработки графовых структур, а matplotlib позволяет быстро и гибко отображать графы с возможностью динамического изменения их внешнего вида.

Кроме того, простой и лаконичный синтаксис языка упрощает реализацию алгоритма и делает её более наглядной, что особенно важно при демонстрации. Python также обладает кроссплатформенностью и широкой поддержкой — он установлен практически на всех современных операционных системах, а необходимые библиотеки доступны через стандартный пакетный менеджер pip. Для удобства разработки и отладки использовалась интегрированная среда разработки VS Code. Сочетание Python 3, библиотек networkx и matplotlib, а также привычных инструментов разработки обеспечили удобную, наглядную и достаточно быструю платформу для создания, отладки и демонстрации алгоритма Краскала.

**2.2. Структура программы и ключевые модули**

*Рис. 16. Реализация визуала Краскала*

Реализация алгоритма Краскала в работе построена на языке Python. Основными используемыми библиотеками стали networkx (для работы с графами) и matplotlib.pyplot (для визуализации). Программа организована в виде одной основной функции kruskal\_visualization(graph), которая принимает на вход граф в виде словаря с двумя ключами: 'nodes' — список вершин и 'edges' — список рёбер в формате (u, v, вес).

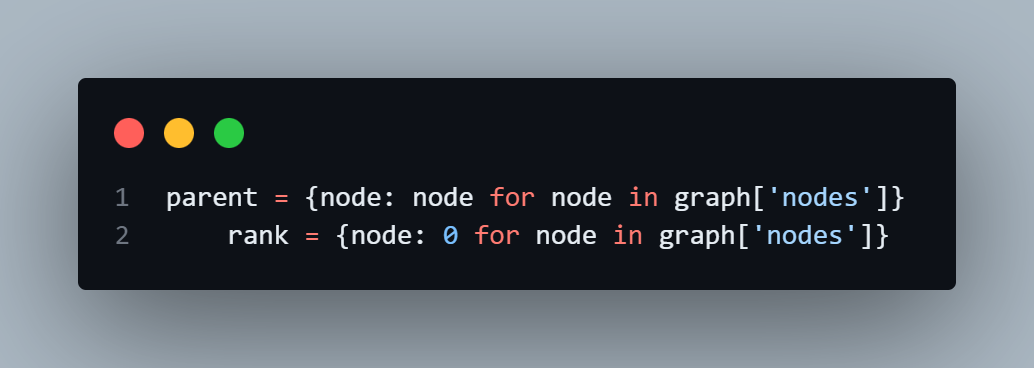
В начале функции происходит построение объекта графа G средствами библиотеки networkx. Для этого используется цикл:

  
*Рис. 17. Построение графа*

Здесь каждое ребро добавляется в граф с указанием веса. Далее осуществляется предварительная обработка рёбер, они сортируются по возрастанию веса с помощью лямбда-функции:

  
*Рис. 18. Сортировка*

После этого инициализируются две ключевые структуры данных: parent и rank, которые используются для отслеживания компонент связности и реализации структуры «система непересекающихся множеств» (Union-Find):

  
*Рис. 19. Отслеживание компонент связности*

Определены две вложенные функции: find(u) — реализует поиск корня компоненты вершины u, с применением сжатия пути, и union(u, v) — объединяет две компоненты, если они ещё не соединены. Эти функции являются ключевыми при реализации Краскала, так как они предотвращают образование циклов в строящемся остовном дереве.

**2.3. Интерфейс и визуализация**

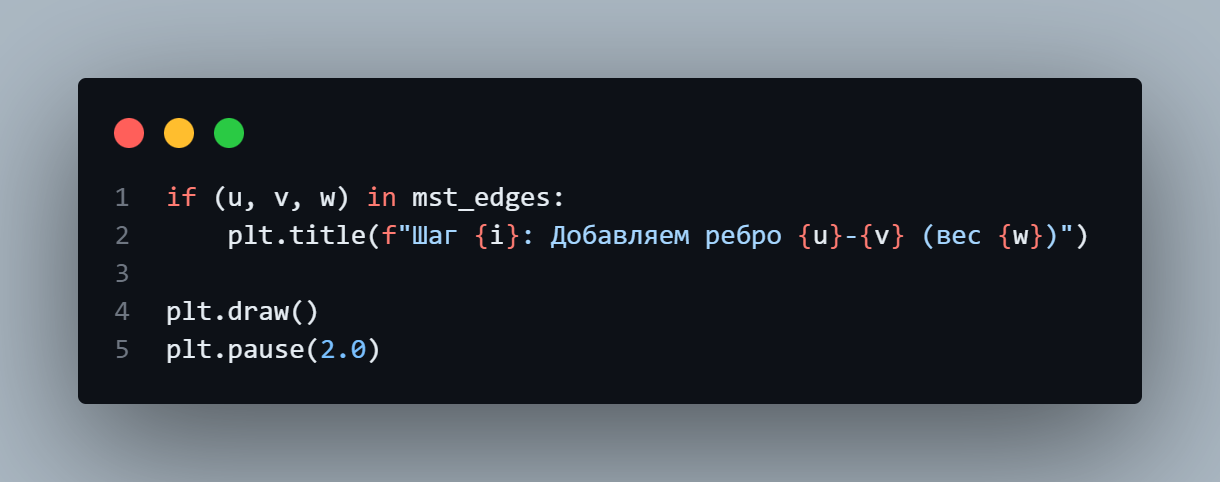
Одной из отличительных особенностей программы является её пошаговая визуализация, реализованная с использованием библиотеки matplotlib. Расположение вершин в графе фиксируется при помощи функции nx.spring\_layout(G), которая обеспечивает удобное и понятное представление графа на плоскости:

  
*Рис. 20. Фиксация расположения вершин*

Перед началом основного цикла вызывается plt.figure(figsize=(10, 6)), после чего на каждом шаге алгоритма происходит очистка экрана (plt.clf()), отрисовка всех рёбер серым цветом, отрисовка добавленных в остов рёбер красным, а также отображение вершин и их подписей:

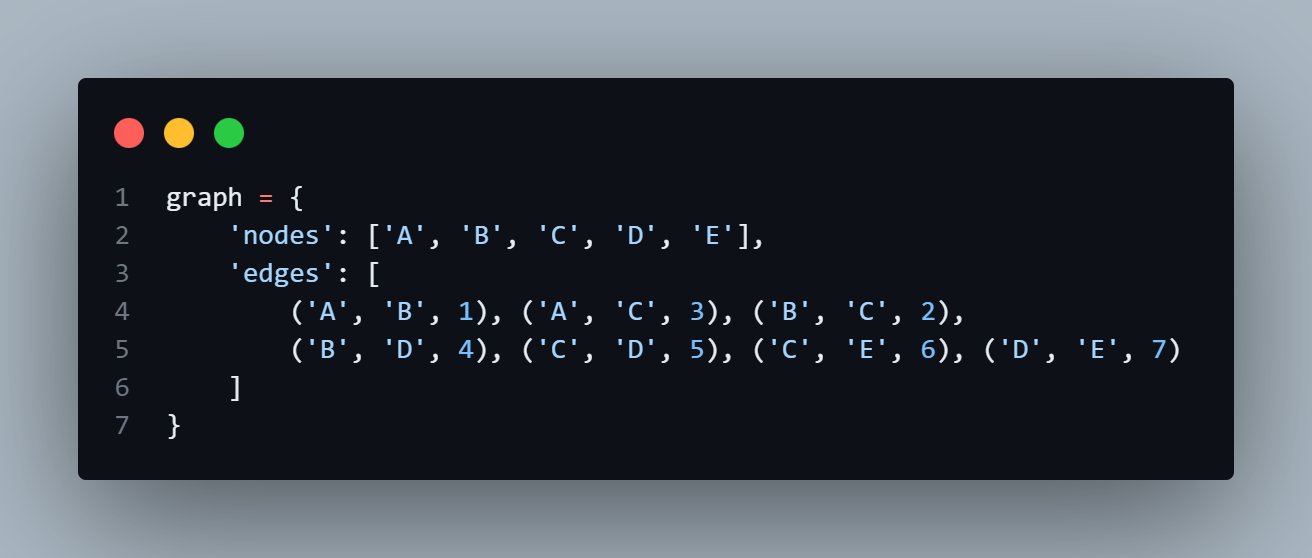
  
*Рис. 21. Отрисовка*

Дополнительно отображаются веса рёбер с помощью функции nx.draw\_networkx\_edge\_labels. На каждом шаге графическое окно обновляется заголовком с текущим действием (например, "Шаг 3: Добавляем ребро A-B (вес 1)") и делается пауза 2 секунды:

  
*Рис. 22. Обновление и пауза*

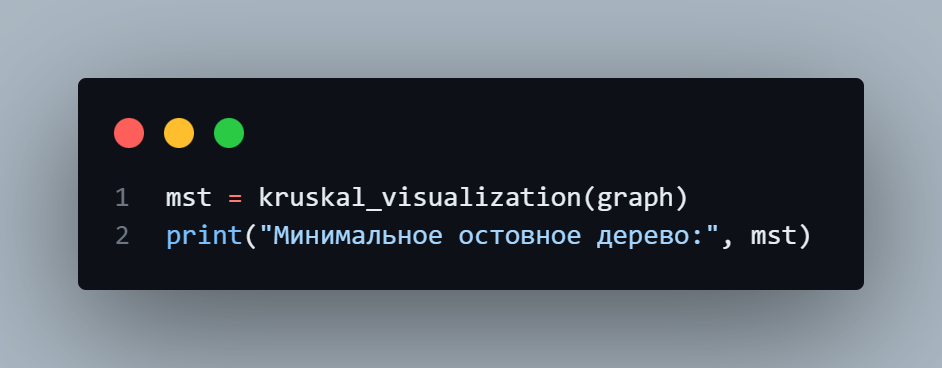
**2.4. Тестирование на примерах графов**

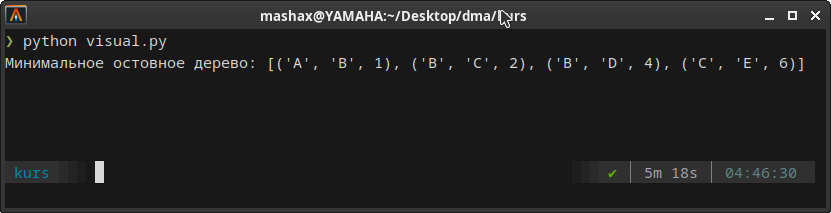
Для тестирования алгоритма был использован заранее подготовленный граф с вершинами A, B, C, D, E и семью рёбрами с различными весами:

  
*Рис. 23. Тестируем на заготовленном графе*

После запуска функции kruskal\_visualization(graph) происходит построение минимального остовного дерева. Алгоритм корректно исключает рёбра, образующие цикл (например, ребро ('C', 'D', 5) при уже добавленных ('B', 'C') и ('B', 'D')), и добавляет только те рёбра, которые соединяют новые компоненты.

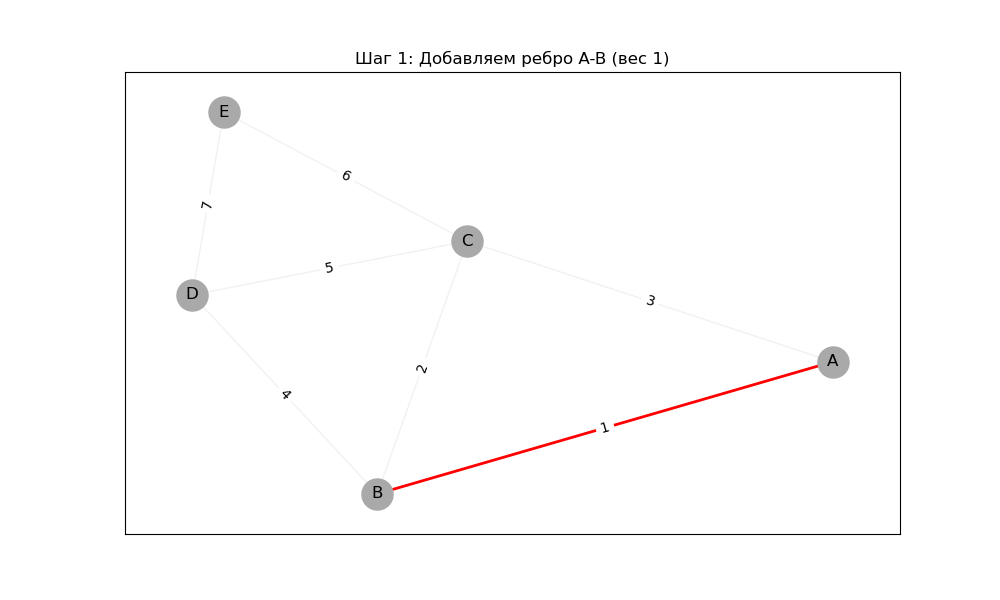
Результат выполнения сохраняется в переменную mst и выводится в консоль:

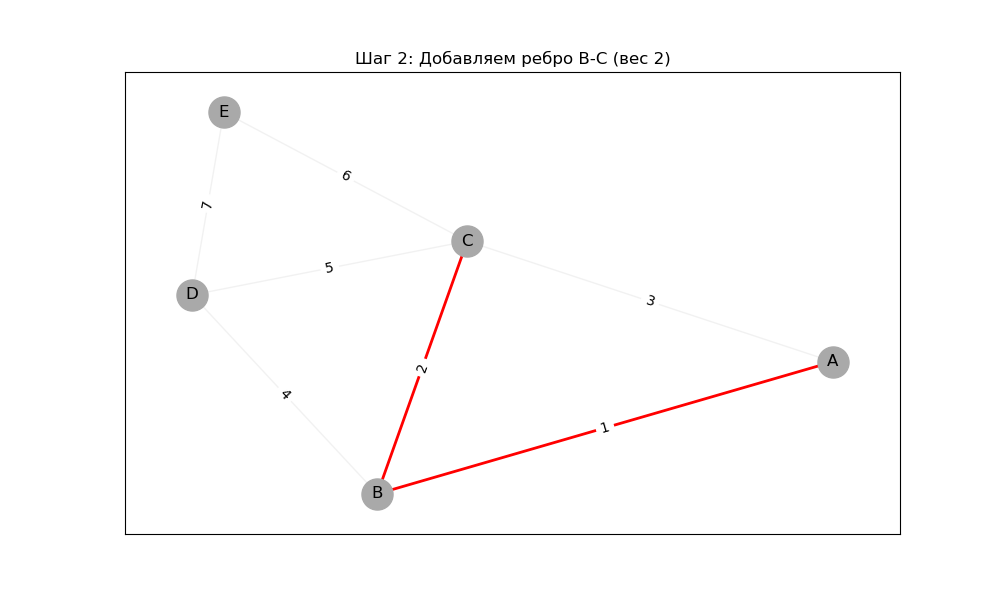
  
*Рис. 24. Окончание программы*

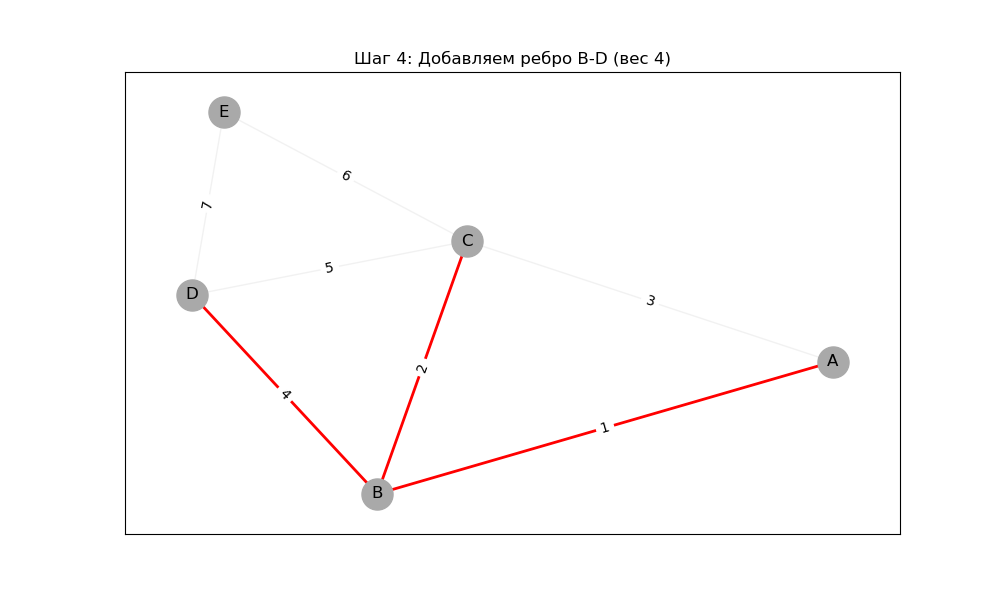
*  
Рис. 25. Вывод в консоль*

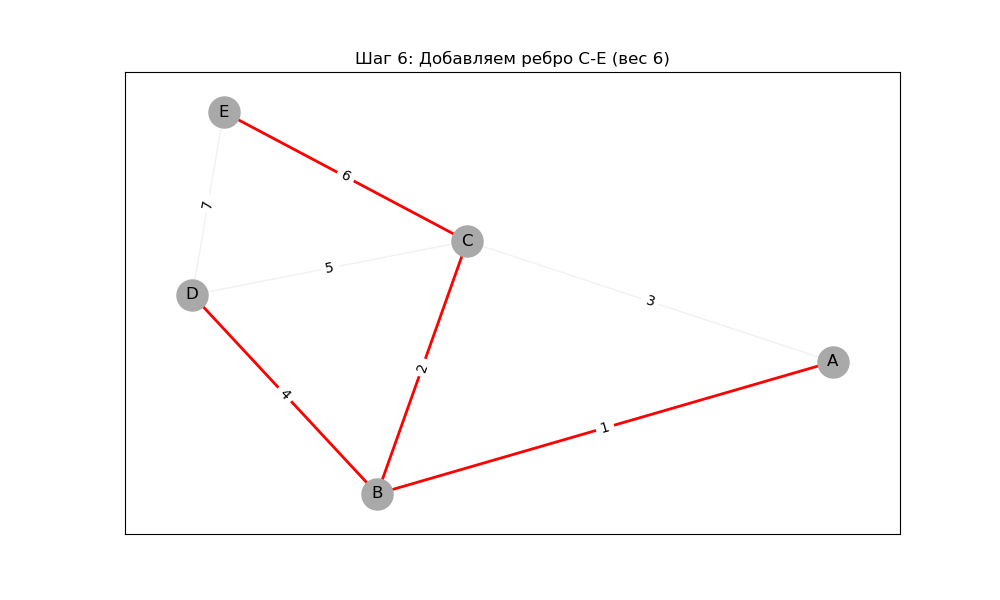
Суммарный вес дерева составил 13, что соответствует ожидаемому результату для данного графа. Графическая визуализация на экране также подтвердила корректность построения.

Теперь посмотрим на долгожданную визуализацию:

  
*Рис. 26. Первое фото: шаг 1*

  
*Рис. 27. Второе фото: шаг 2*

  
*Рис. 28. Третье фото: шаг 4*

  
*Рис. 29. Четвертое фото: шаг 6*

**2.5. Анализ полученных результатов**

Результаты, полученные в ходе тестирования, продемонстрировали, что реализация алгоритма Краскала выполняет свою задачу корректно. Алгоритм надёжно отбирает минимальные рёбра, соблюдая условие отсутствия циклов, и формирует остовное дерево, охватывающее все вершины.

Благодаря встроенной визуализации, удалось наглядно продемонстрировать механику работы алгоритма: от сортировки рёбер до объединения компонент. Такое представление особенно полезно для учебных целей, поскольку визуально объясняет принцип работы Union-Find и позволяет интуитивно понять, почему исключаются определённые рёбра.

Программа проста в расширении — её можно адаптировать под другие графовые алгоритмы или добавить пользовательский ввод. Благодаря модульности и использованию стандартных библиотек, её можно запускать в любой системе с установленным Python и необходимыми зависимостями.

**Заключение**

В данной курсовой работе была рассмотрена задача построения минимального остовного дерева (МОД) — одна из ключевых задач в теории графов, имеющая широкое практическое применение. Основное внимание было уделено алгоритму Краскала, как одному из наиболее известных и эффективно реализуемых алгоритмов для решения этой задачи.

В теоретической части были раскрыты основные понятия, связанные с графами, деревьями и остовами, а также дана формальная постановка задачи. Проведён сравнительный анализ классических алгоритмов (Краскала, Прима и Борувки), что позволило обосновать выбор алгоритма Краскала для дальнейшего исследования.

В практической части был подробно разобран принцип работы алгоритма Краскала, а также реализована программа, демонстрирующая пошаговое построение минимального остовного дерева. Для наглядности результаты работы алгоритма были визуализированы, что позволило лучше понять его механику и отследить логику построения остова на конкретных примерах.

Было проведено тестирование алгоритма на различных графах, включая ручной пример. Полученные результаты подтвердили корректность реализации и соответствие ожидаемому поведению алгоритма.

В ходе курсовой работы была изучена задача построения кратчайшего связывающего дерева с использованием алгоритма Краскала. Реализация показала, что алгоритм прост в понимании, эффективен и позволяет наглядно проследить процесс формирования минимального остовного дерева. Программа успешно обрабатывает входные графы и демонстрирует правильность работы алгоритма. Курсовая работа подтвердило, как теоретическую значимость, так и практическую применимость алгоритма Краскала при решении задач оптимизации в графах.

Все файлы можно посмотреть на GIT:

**Список литературы**

1. Онлайн школа Skysmart: https://skysmart.ru/articles/mathematic/osnovnye-ponyatiya-teorii-grafov
2. Алгоритм Краскала, Прима для нахождения минимального остовного дерева / Хабр: <https://habr.com/ru/articles/569444/>
3. Построение минимального остовного дерева foxford.ru: <https://foxford.ru/wiki/informatika/postroenie-minimalnogo-ostovnogo-dereva?utm_referrer=https%3A%2F%2Fyandex.ru%2F>
4. github.com hse-algorithms-and-data-structures: <https://github.com/an-sla/hse-algorithms-and-data-structures/blob/main/Lecture%2013_26.05.2022/theory.md>
5. Алгоритм Краскала, Прима для нахождения минимального остовного дерева. Хабр: https://habr.com/ru/articles/569444/