**Министерство образования и науки Республики Казахстан**

**Astana IT University**

**НАУЧНЫЙ ПРОЕКТ**

на тему:

**«Применение алгоритмов построения минимального остовного дерева (МОД)»**

Выполнили: Серикбай А. А, Оралбай М.

Группа: MCS-2301

Проверила: ассоц. проф. Молдахметова З. Н.

Астана

2025

**СОДЕРЖАНИЕ**

**ВВЕДЕНИЕ**

**ГЛАВА1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ МИНИМАЛЬНОГО ОСТОВНОГО ДЕРЕВА**

[1.1. Основные понятия теории графов](#_Toc483955736)

[1.2. Остовное дерево. Задача построения остовного дерева минимального](#_Toc483955737) веса

[1.3. А](#_Toc483955736)лгоритм Прима

[1.4. А](#_Toc483955737)лгоритм Крускала

**ГЛАВА2. РЕАЛИЗАЦИЯ И АНАЛИЗ АЛГОРИТМОВ**

2.1. **Инструменты и технологии**

2.2. **Реализация кода**

2.3. Тестирование алгоритмов

2.4. **Сравнительный анализ**

**ЗАКЛЮЧЕНИЕ**

**СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ**

**Введение**

## **Актуальность темы исследования:**

## Актуальность и практическая значимость предлагаемой работы заключается в том, что в большинстве областей возникает необходимость поисков эффективных способов планирования сложных процессов и проектов, которые в итоге можно реализовать с помощью теорий графов. С помощью теории графов можно минимизировать затраты в ходе производства, а далее принимать оптимальные управленческие решения. И в теории графов часто возникает задача поиска минимального остовного дерева (МОД). И сложно определить по получившемуся графу какой алгоритм решит эту задачу оптимально.

## Примеры таких задач:

## - **Инфраструктура:** Оптимизация прокладки дорог, трубопроводов, кабелей и других сетей.

## - **Компьютерные сети:** Минимизация затрат на подключение серверов и маршрутизаторов.

## - **Энергетика:** Построение минимальных сетей энергоснабжения.

## - **Генетика и биоинформатика:** Анализ филогенетических деревьев.

## **Объект исследования:**

Выбор оптимального алгоритма для поиска минимального остовного дерева для определенного графа.

## **Предмет исследования:**

Алгоритмы Прима и Крускала для поиска минимального остовного дерева.

**Цель:**

Изучить алгоритмы Прима и Крускала для поиска минимального остовного дерева и исследовать их эффективность на различных графах.

**Задачи работы:**

1. Проанализировать основные теоретические аспекты задачи минимального остовного дерева;

2. Реализовать алгоритмы Крускала и Прима на языке Python;

3. Разработать алгоритм для генерации тестовых графов и их визуализации;

4. Сделать сбор данных;

5. Из полученной информации сравнить эффективность алгоритмов;

## **Методы исследования:**

-Метод сравнения.

-**Экспериментальный метод**.

-Сравнительный анализ временной сложности алгоритмов.

-**Математическое моделирование**.

## **Теоретические основы работы:**

Работа базируется на теории графов, алгоритмах прима и крускала для поиска минимального остовного дерева

## **Практическая значимость:**

Результаты работы могут быть использованы в прикладных задачах для минимизации затрат. Примером такой задачи может быть транспортное планирование где необходимо найти такой путь чтобы общая длина пути и стоимость была минимальной. Такая же задача возникает в проектирование компьютерных сетей, а именно в протоколе STP, который устраняет петли в сети выбирая при этом лучшие петли по скорости соеденения.

## **Структура работы:**

-Работа состоит из введения, двух глав, заключения, списка литературы и приложения.

-В введении мы написали характеристики работы которые определяют тему нашей работы и ее актуальность, объект и предмет исследования, цели и задачи, которые необходимо рассмотреть, методы исследования, теоретические основы и практическую значимость нашей работы.

-В первой главе рассмотрена теоритическая часть поиска минимального остовного дерева

-Во второй главе произведен анализ алгоритмов поиска минимального остовного дерева и их реализация.

-В заключении сделаны выводы на основе полученной информации.

-В списке литературы приведены источники информации и ссылки на них.

-В приложении представлены графики, диаграмы и прочие рисунки использованные в исследовании.

**ГЛАВА1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ МИНИМАЛЬНОГО ОСТОВНОГО ДЕРЕВА**

* 1. **Основные понятия теории графов**

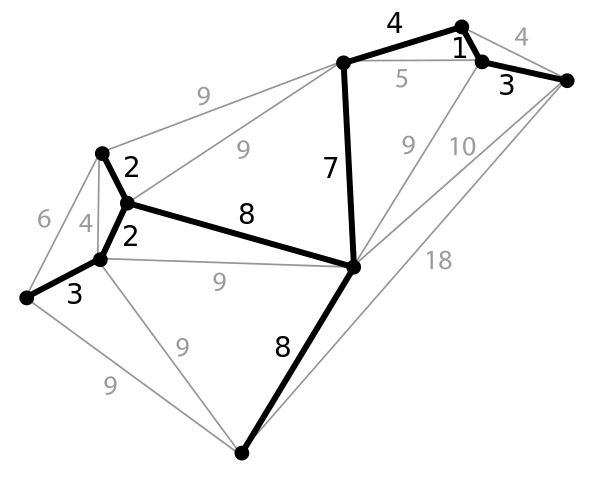
Теория графов — это раздел дискретной математики, изучающий структуры, состоящие из вершин и рёбер, соединяющих их. Граф определяется как G = (V, E), где V — множество вершин, а E — множество рёбер, соединяющих вершины. Графы бывают ориентированными и неориентированными, взвешенными и невзвешенными, связными и несвязными.

* 1. **Задача построения минимального остовного дерева**

Остовное дерево (spanning tree) — это подграф, содержащий все вершины исходного графа, но не содержащий циклов и имеющий минимальное возможное количество рёбер. Минимальное остовное дерево (МОД) — это остовное дерево с минимальной суммой весов рёбер. Построение МОД — это важная задача в теории графов, находящая применение в сетевом планировании, маршрутизации и инфраструктурных проектах.

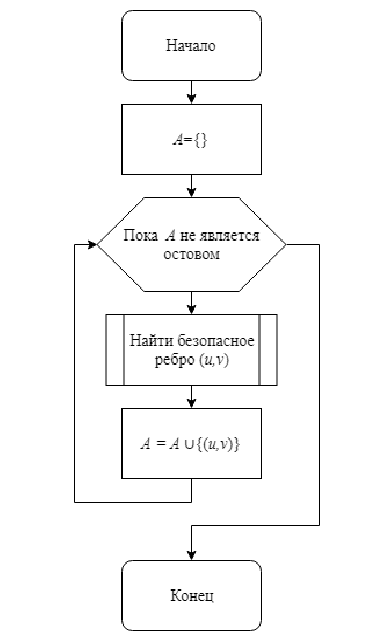
Например представим какой то город где есть различные районы соедененные дорогами. Каждая дорога имеет свою протяженность, и по ней могут ездить машины. Для эффективной работы транспортной системы города требуется чтобы не было замкнутых маршрутов(циклов), которые могут привести к избыточному пробегу машин. Наша задача состоит в том чтобы каждый район был доступен для машин, и суммарная протяженность дорог по которым ездят машины была минимальной. Такую задачу можно описать как задачу поиска минимального остовного дерева, где узлы — это районы города, а ребра — это дороги между районами с весом, равным протяженности дороги.

Задача в общем виде выглядит так. Дан связанный, взвешенный,  
неориентированный граф G = (V, E), где V — множество вершин, а E — множество рёбер, соединяющих вершины. Для каждого ребра задан определенный вес задающий стоимость соедениние двух вершин. Задача состоит в нахождении такого подграфа который соединяет все вершины и общий вес которой минимален(см Рисунок 1)



**Рисунок 1 – Минимальное остовное дерево связанного графа**

**Искомый остов получается из подграфа А. Граф А строится постепенно и  
изначально является пустым. На каждом шаге алгоритм добавляет одно новое ребро и так пока А не станет остовом. Подграф А графа G называется *безопасным* еслионо является подграфом какого-то минимального остого.**



**Рисунок 2 – Блок-схема алгоритма построения минимального остовного  
дерева в общем виде**

* 1. **Алгоритм Прима**

Алгоритм Прима является жадным методом поиска минимального остовного дерева(поскольку на каждом шаге он добавляет ребро минимального веса). Он строит остовное дерево, начиная с произвольной вершины, и на каждом шаге добавляет к дереву ближайшую вершину, соединенную минимальным по весу ребром.

Основные шаги алгоритма:

1. Выбираем произвольную начальную вершину.
2. Инициализируем множество А остовного дерева (изначально пустое).
3. Добавляем минимальное ребро, соединяющее вершину с остальной частью графа к множеству.
4. Повторяем процесс, пока все V-1 ребер не будут включены в множество.

Приведем пример работы алгоритма. На рисунке 3 изображен граф G

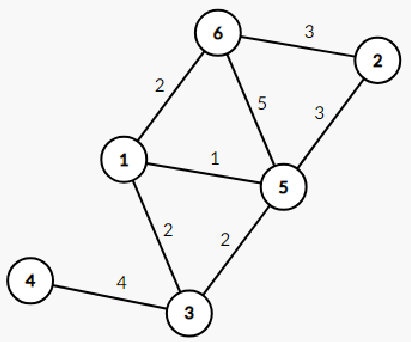


Рисунок 3 – Граф на котором будет запущен алгоритм Прима

Выберем произвольную вершину r = 6. Просмотрим все ребра исходящие из r среди таких минимальный вес имеет ребро (6, 1). Добавим его в остов  
(рисунок 4).

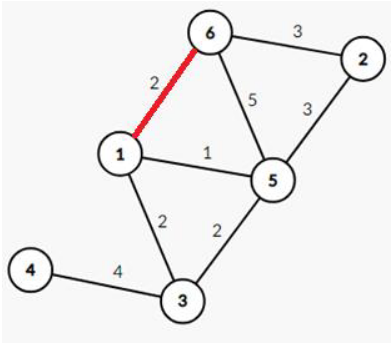


Рисунок 4 – Подграф А после первого шага алгоритма Прима

Теперь у нас есть две вершины 1 и 6. Рассмотрим все инцидентный им  
ребра. Среди них наименьший вес имеет ребро (1, 5). Добавим его в остов  
(рисунок 5)

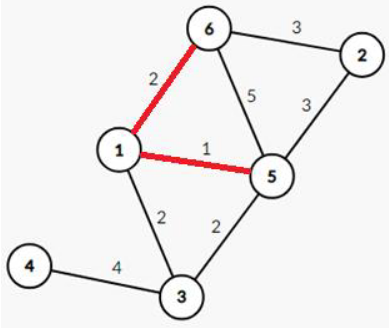


Рисунок 5 – Подграф А после второго шага алгоритма Прима

Аналогично продолжим на каждой итерации выбирать безопасное ребро  
минимального веса. В итоге получим остов, изображенный на рисунке 6.

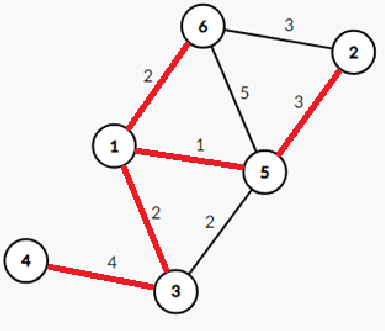


Рисунок 6 – Финальный подграф А(минимальный остов), полученный алгоритма Прима

Если это будет разреженный граф асимптотика поиска минимального ребра займет O(E\*log2(V) операции. Иначе если это будет полный граф это займет O(V^2) операции.

**1.4. Алгоритм Крускала**  
Алгоритм Крускала — еще один жадный алгоритм(поскольку на каждом шаге он добавляет ребро минимального веса), который сортирует все рёбра по весу и последовательно добавляет их в остовное дерево, если они не создают цикла. Основные шаги алгоритма:

1. Сортируем рёбра графа по возрастанию веса.
2. Инициализируем множество А остовного дерева (изначально пустое).
3. Последовательно добавляем рёбра, если они не образуют цикла.
4. Повторяем процесс, пока все V-1 ребер не будут включены в множество.

Приведем пример работы алгоритма. На рисунке 7 изображен граф G

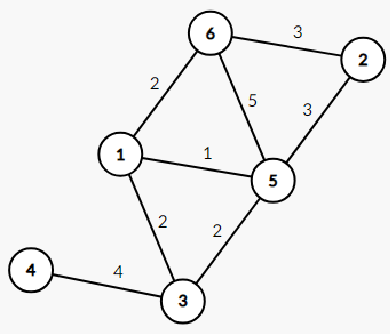
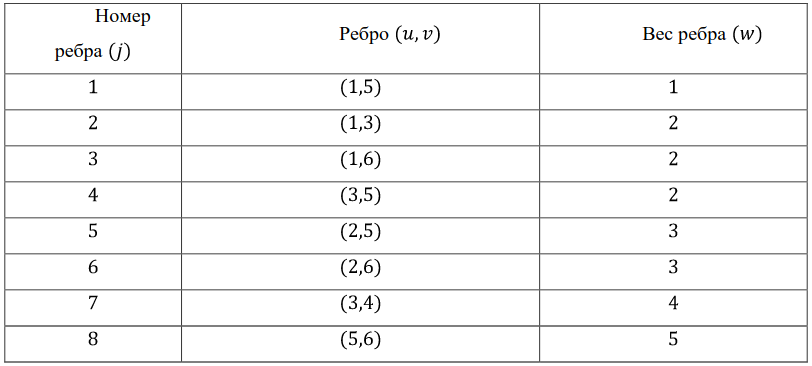


Рисунок 7 – Граф на котором будет запущен алгоритм Крускала

Выпишем все ребра в порядке возрастания их весов. Результат приведен в  
таблице 1.

Таблица 1 – Ребра графа в порядке возрастания весов

Для начала берем ребро с минимальным весом(1, 5) и добавляем в наш остов (рисунок 8).

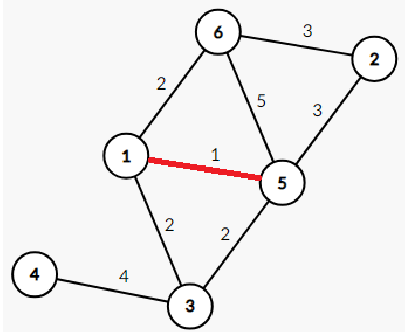


Рисунок 8 – Дерево после первого шага алгоритма Крускала

Такой же логикой берем ребра (1, 3),(1, 6) и добавляем в наш остов (рисунок 9).

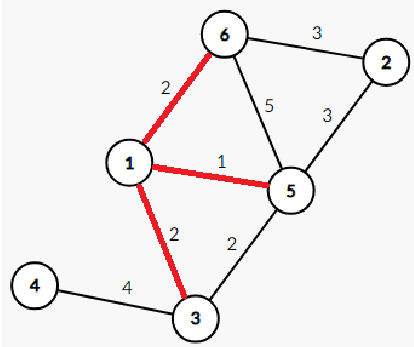


Рисунок 9 – Дерево после третьего шага алгоритма Крускала

Пропускаем ребро (3, 5) (так как тогда образуется цикл из ребер 1, 2, 4) и проверяем ребро (2, 5). Текущее ребро не создает противоречий, значит мы включаем его в остов (рисунок 10)

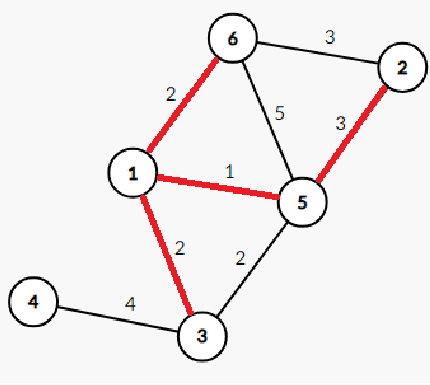


Рисунок 10 – Дерево после четвертого шага алгоритма Крускала

Ребро (2, 6) нам не подходит так как будет образован цикл из ребер 1, 3, 5, 6. Ребро (3, 4) мы включаем в минимальный остов (рисунок 11).

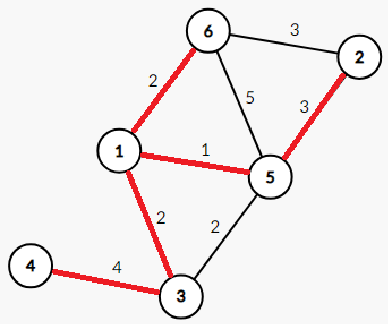


Рисунок 11 – Остов на на пятом шаге алгоритма Крускала

Так как мы включили n-1 ребро, алгоритм можно завершить. Минимальный остов найден.

Асимптотика алгоритма Крускала сильно зависит от используемой  
сортировки и при использовании быстрой сортировки она равна O(E\*log2(E)) операциям.

**ГЛАВА 2. РЕАЛИЗАЦИЯ И АНАЛИЗ АЛГОРИТМОВ**

**2.1. Инструменты и технологии**

Для реализации алгоритмов минимального остовного дерева использовался язык программирования Python и библиотека NetworkX,  
которая предоставляет удобные инструменты для работы с графами. Также использовались matplotlib для визуализации графов и numpy   
для статистической обработки результатов.

**Используемые библиотеки:**

time – для измерения времени выполнения алгоритмов.

random – для генерации случайных рёбер графа.

networkx – для работы с графами и реализации алгоритмов Прима и Краскала.

matplotlib.pyplot – для построения графов и визуализации результатов.

numpy – для вычисления среднего времени выполнения алгоритмов.

**2.2. Реализация кода**

**Входные данные:**

Программа запрашивает у пользователя несколько параметров для генерации случайного взвешенного неориентированного графа и последующего сравнения алгоритмов Прима и Крускала для построения минимального остовного дерева (MST). Входные параметры включают:

1. **Количество вершин (num\_vertices)** – целое число, задающее число узлов в графе.
2. **Минимальный вес ребра (edge\_min)** – вещественное число, задающее нижнюю границу для случайно генерируемых весов рёбер.
3. **Максимальный вес ребра (edge\_max)** – вещественное число, задающее верхнюю границу для весов рёбер.
4. **Диапазон весов (edge\_range)** – кортеж, формируемый на основе edge\_min и edge\_max.
5. **Вероятность рёбра (edge\_probability)** – вещественное число от 0 до 1, определяющее вероятность существования ребра между двумя вершинами.
6. **Количество запусков (runs)** – целое число, задающее, сколько раз программа должна сгенерировать случайный граф и выполнить алгоритмы.

**Алгоритм работы программы:**

1. Запрашивает у пользователя входные параметры.
2. Генерирует случайный граф на основе параметров:
   * Перебирает все возможные рёбра между вершинами.
   * Добавляет ребро с заданной вероятностью edge\_probability и случайным весом из edge\_range.
3. Запускает алгоритмы Прима и Крускала столько раз, сколько запросил пользователь, измеряя время их выполнения.
4. Вычисляет среднее время работы обоих алгоритмов.
5. Определяет, какой алгоритм быстрее в среднем.
6. Записывает результаты в файл mst\_results.txt.
7. Отображает графики, показывающие время выполнения каждого алгоритма на разных запусках.

**2.3. Тестирование алгоритмов**

Для оценки эффективности алгоритмов выполнялось сравнение их времени выполнения на одинаковых входных данных.

Параметры запуска и их результаты:

Vertices=100, Edge Range=(1.0, 10.0), Runs=10000, Edge Probability=0.2

Prim's Algorithm Time: 0.000914991 seconds

Kruskal's Algorithm Time: 0.001368660 seconds

----------------------------------------

Run Parameters: Vertices=100, Edge Range=(1.0, 10.0), Runs=10000, Edge Probability=1.0

Prim's Algorithm Time: 0.003510142 seconds

Kruskal's Algorithm Time: 0.007393810 seconds

----------------------------------------

Run Parameters: Vertices=100, Edge Range=(1.0, 10.0), Runs=100, Edge Probability=0.2

Prim's Algorithm Time: 0.001040226 seconds

Kruskal's Algorithm Time: 0.001414157 seconds

----------------------------------------

Run Parameters: Vertices=100, Edge Range=(1.0, 10.0), Runs=100, Edge Probability=1.0

Prim's Algorithm Time: 0.003564128 seconds

Kruskal's Algorithm Time: 0.006800484 seconds

----------------------------------------

Run Parameters: Vertices=100, Edge Range=(1.0, 10.0), Runs=10, Edge Probability=1.0

Prim's Algorithm Time: 0.003481890 seconds

Kruskal's Algorithm Time: 0.005959110 seconds

----------------------------------------

Run Parameters: Vertices=100, Edge Range=(1.0, 10.0), Runs=10, Edge Probability=0.2

Prim's Algorithm Time: 0.001268250 seconds

Kruskal's Algorithm Time: 0.001977470 seconds

----------------------------------------

Run Parameters: Vertices=10, Edge Range=(1.0, 10.0), Runs=100, Edge Probability=0.2

Prim's Algorithm Time: 0.000047553 seconds

Kruskal's Algorithm Time: 0.000064860 seconds

----------------------------------------

Run Parameters: Vertices=10, Edge Range=(1.0, 10.0), Runs=100, Edge Probability=1.0

Prim's Algorithm Time: 0.000071498 seconds

Kruskal's Algorithm Time: 0.000104386 seconds

----------------------------------------

Run Parameters: Vertices=200, Edge Range=(1.0, 1000.0), Runs=1000, Edge Probability=0.05

Prim's Algorithm Time: 0.001262719 seconds

Kruskal's Algorithm Time: 0.001707723 seconds

----------------------------------------

Run Parameters: Vertices=1000, Edge Range=(1.0, 1000.0), Runs=1000, Edge Probability=0.001

Prim's Algorithm Time: 0.002008828 seconds

Kruskal's Algorithm Time: 0.002628301 seconds

----------------------------------------

Run Parameters: Vertices=10, Edge Range=(0.0, 10.0), Runs=100, Edge Probability=1e-07

Prim's Algorithm Time: 0.000044970 seconds

Kruskal's Algorithm Time: 0.000041362 seconds

----------------------------------------

Run Parameters: Vertices=10, Edge Range=(0.0, 10.0), Runs=10000, Edge Probability=1e-08

Prim's Algorithm Time: 0.000010659 seconds

Kruskal's Algorithm Time: 0.000019290 seconds

----------------------------------------

Run Parameters: Vertices=10, Edge Range=(0.0, 10.0), Runs=1, Edge Probability=1e-05

Prim's Algorithm Time: 0.002188650 seconds

Kruskal's Algorithm Time: 0.000789269 seconds

----------------------------------------

Run Parameters: Vertices=10, Edge Range=(1.0, 10.0), Runs=10, Edge Probability=1.0

Prim's Algorithm Time: 0.000130740 seconds

Kruskal's Algorithm Time: 0.000202430 seconds

----------------------------------------

Run Parameters: Vertices=100, Edge Range=(1.0, 1000.0), Runs=10000, Edge Probability=0.1

Prim's Algorithm Time: 0.000691713 seconds

Kruskal's Algorithm Time: 0.000936161 seconds

----------------------------------------

Run Parameters: Vertices=100, Edge Range=(1.0, 10.0), Runs=1000, Edge Probability=0.01

Prim's Algorithm Time: 0.000196323 seconds

Kruskal's Algorithm Time: 0.000273965 seconds

----------------------------------------

Run Parameters: Vertices=100, Edge Range=(1.0, 1000.0), Runs=10000, Edge Probability=0.0

Prim's Algorithm Time: 0.000005987 seconds

Kruskal's Algorithm Time: 0.000010504 seconds

----------------------------------------

Run Parameters: Vertices=1000, Edge Range=(1.0, 10.0), Runs=1000, Edge Probability=0.01

Prim's Algorithm Time: 0.010418063 seconds

Kruskal's Algorithm Time: 0.013833279 seconds

----------------------------------------

Run Parameters: Vertices=1000, Edge Range=(1.0, 1000.0), Runs=1000, Edge Probability=1e-12

Prim's Algorithm Time: 0.000029257 seconds

Kruskal's Algorithm Time: 0.000026415 seconds

----------------------------------------

Run Parameters: Vertices=1000, Edge Range=(1.0, 1000.0), Runs=1000, Edge Probability=0.0

Prim's Algorithm Time: 0.000028101 seconds

Kruskal's Algorithm Time: 0.000024425 seconds

----------------------------------------

Run Parameters: Vertices=1000, Edge Range=(1.0, 10.0), Runs=1000, Edge Probability=1e-05

Prim's Algorithm Time: 0.000077912 seconds

Kruskal's Algorithm Time: 0.000070695 seconds

----------------------------------------

Run Parameters: Vertices=100, Edge Range=(1.0, 10.0), Runs=1000, Edge Probability=0.0001

Prim's Algorithm Time: 0.000012463 seconds

Kruskal's Algorithm Time: 0.000016529 seconds

----------------------------------------

Run Parameters: Vertices=10, Edge Range=(1.0, 100.0), Runs=1000, Edge Probability=0.0

Prim's Algorithm Time: 0.000006503 seconds

Kruskal's Algorithm Time: 0.000010011 seconds

----------------------------------------

Run Parameters: Vertices=1000, Edge Range=(1.0, 1000.0), Runs=10, Edge Probability=1.0

Prim's Algorithm Time: 1.123664750 seconds

Kruskal's Algorithm Time: 1.303239230 seconds

**2.4. Сравнительный анализ**

1. Алгоритм Прима – лучший для плотных графов

Если граф плотный (то есть содержит много рёбер), алгоритм Прима работает эффективнее, потому что он растёт от одной вершины, постепенно добавляя ближайшие минимальные рёбра.

Когда использовать:

-Если у вас граф задан матрицей смежности (особенно когда |E| ≈ |V|²).

-Если требуется начать построение остовного дерева с конкретной вершины.

-Если важна эффективность при обработке больших графов – при использовании кучи Фибоначчи он работает быстрее, чем Крускал.

-Если задача связана с непрерывным ростом сети (например, проводка электричества, кабелей интернета, строительство дорог).

Примерные приложения:

-Построение инфраструктурных сетей (электросети, газопроводы, оптоволоконные сети).

-Генерация лабиринтов и игровых карт (игровой ИИ, построение карт).

-Оптимальные маршруты робототехники (сканирование помещений).

Недостатки:

-Менее удобен для разреженных графов.

-Не так хорош, если граф представлен списком рёбер.

2. Алгоритм Крускала – лучший для разреженных графов

Если граф разреженный (то есть рёбер мало по сравнению с количеством вершин), то алгоритм Крускала работает лучше, так как он сортирует рёбра и объединяет компоненты с помощью DSU (Disjoint Set Union – система непересекающихся множеств).

Когда использовать:

-Если граф задан списком рёбер (особенно когда |E| ≈ |V|).

-Если требуется построить минимальный остовной лес (например, для нескольких несвязанных компонент).

-Если важно обрабатывать динамические изменения в графе – можно легко удалять/добавлять рёбра.

-Если ребра уже отсортированы или их сортировка не занимает много времени.

Примерные приложения:

-Логистика и маршруты транспорта (авиасообщения, железнодорожные пути).

-Социальные сети (определение минимальных связей между группами людей).

-Кластеризация данных в машинном обучении (группировка схожих объектов).

Недостатки:

-Работает медленнее на плотных графах.

-Хуже работает с матрицей смежности.

-Не строит дерево, начиная с определённой вершины.

**ЗАКЛЮЧЕНИЕ**

В данной работе проведено исследование алгоритмов Прима и Крускала для нахождения минимального остовного дерева (MST) в случайных взвешенных неориентированных графах. Были рассмотрены их теоретические основы, реализованы программные версии на языке Python с использованием библиотеки NetworkX и проведены экспериментальные замеры времени выполнения на различных наборах входных данных.

Анализ полученных результатов показал, что эффективность работы алгоритмов зависит от структуры входного графа. В частности, алгоритм Прима продемонстрировал преимущество на графах с высокой плотностью рёбер, что объясняется его жадной стратегией добавления вершин. В то же время алгоритм Крускала, основывающийся на сортировке рёбер, оказался более эффективным для графов с малой плотностью рёбер.

Экспериментальные данные подтвердили, что при большом количестве рёбер преимущество Прима становится более заметным, в то время как Крускал быстрее справляется с разреженными графами. Таким образом, выбор алгоритма зависит от конкретных условий задачи: если граф представлен в виде списка рёбер, целесообразнее использовать алгоритм Крускала, тогда как для представления в виде матрицы смежности или списков смежности более эффективным будет алгоритм Прима.

Возможными направлениями дальнейших исследований могут стать:

* Оптимизация реализации алгоритмов с использованием структур данных, таких как Фибоначчиевы кучи, для ускорения работы алгоритма Прима.
* Анализ влияния различных моделей случайных графов (например, графов Эрдоша-Реньи или Барбаши-Альберта) на эффективность алгоритмов.
* Расширение исследования на параллельные и распределённые версии алгоритмов для работы с большими графами.

Таким образом, проведённая работа не только подтвердила известные теоретические результаты, но и позволила на практике сравнить алгоритмы, выявив их сильные и слабые стороны, что может быть полезно при выборе метода для конкретных прикладных задач.

**СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ**

Кормен Т. Х. и др. Часть VI. Алгоритмы для работы с графами // Алгоритмы: построение и анализ = Introduction to Algorithms. — 2-е изд. — М.: Вильямс, 2006. – 1296 c.(<http://e-maxx.ru/bookz/files/cormen.pdf>)

Лутц М. Программирование на Python, том I, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 992 с.

Лутц М. Программирование на Python, том II, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 992 с.

<https://cyberleninka.ru/article/n/postroenie-minimalnogo-ostovnogo-dereva-algoritmom-boruvki-programmnaya-realizatsiya/viewer>

<https://cyberleninka.ru/article/n/nahozhdenie-ostovnogo-dereva-minimalnogo-vesa-s-primeneniem-algoritma-kraskala-i-algoritma-prima/viewer>

[https://dspace.tltsu.ru/bitstream/123456789/9734/1/%D0%A1%D0%B0%D0%B1%D0%B8%D1%80%D0%BE%D0%B2%20%D0%94.%D0%91.\_%D0%9F%D0%9C%D0%98%D0%B1-1501.pdf](https://dspace.tltsu.ru/bitstream/123456789/9734/1/Сабиров%20Д.Б._ПМИб-1501.pdf)

<https://brestprog.by/topics/mst/>

<https://algorithmica.org/ru/mst>