|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | |
| МИНОБРНАУКИ РОССИИ | | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"МИРЭА - Российский технологический университет"**  **РТУ МИРЭА** | | | |
| **Институт искусственного интеллекта** | | |  |
| **Кафедра высшей математики** | | |  |

|  |  |
| --- | --- |
| **КУРСОВАЯ РАБОТА** | |
| **по дисциплине** | |
| **«**Алгоритмы и теория сложности**»** | |
| **Тема курсовой работы**  **«Минимальные остовные деревья. Алгоритм Крускала»** | |
| Студент группы КМБО- |  |
| Руководитель курсовой работы | *Драгилева И.П.* |
|  |  |
|  |  |

|  |  |  |
| --- | --- | --- |
| Работа представлена к защите | «\_\_»\_\_\_\_\_\_\_2024 г. | *(подпись студента)* |
|  |  |  |
| «Допущен(ы) к защите» | «\_\_»\_\_\_\_\_\_\_2024 г. | *(подпись руководителя)* |

Москва – 2024

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"МИРЭА - Российский технологический университет"**  **РТУ МИРЭА** | | | | **Институт искусственного интеллекта** | | | | **Кафедра высшей математики** | | | | | | |  |
|  | | **Утверждаю** | | |
|  | | Заведующий  кафедрой\_\_\_\_\_\_\_\_\_\_\_\_\_Шатина А.В. | | |
|  | | «\_2\_» \_декабря\_\_2024г. | | |
| **ЗАДАНИЕ** | | | | |
| **на выполнение курсовой работы** | | | | |
| **по** **дисциплине** «Алгоритмы и теория сложности» | | | | |
|  | | | | |
| Студент Группа *КМБО-.* | | | | |
|  | | | | |
| 1. **Тема: «Минимальные остовные деревья. Алгоритм Крускала»** | | | | |
| **2. Исходные данные:** тестовые примеры и наборы данных для испытаний | | | | |
| **3**. **Перечень вопросов, подлежащих обработке, и обязательного графического материала:**  1) Алгоритм(ы) решения задачи на языке высокого уровня.  2) Получение и сравнение количественных результатов при различных исходных данных.  3) Асимптотические оценки временной сложности. | | | | |
|  | | | | |
| **4. Срок представления к защите курсовой работы:** **до** «20» декабря 2024 г. | | | | |
|  | | | | |
| Задание на курсовую  работу выдал | «1» октября 2024г. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | (*\_\_\_\_\_\_\_\_\_\_\_\_*) | |
| Задание на курсовую  работу получил | «\_\_\_»\_\_\_\_\_\_2024г. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | (\_\_\_\_\_\_\_\_\_\_\_\_) | |

**Оглавление**

[**Введение** 4](#_Toc183877805)

[**Постановка задачи** 5](#_Toc183877806)

[**Описание алгоритмов** 6](#_Toc183877807)

[**Доказательство корректности** 12](#_Toc183877808)

[**Асимптотическая оценка временной сложности алгоритмов** 14](#_Toc183877809)

[**Результаты запуска программы** 16](#_Toc183877810)

[**Заключение** 16](#_Toc183877811)

[**Список литературы** 18](#_Toc183877812)

[**Приложения** 19](#_Toc183877813)

## **Введение**

***Минимальное остовное дерево (Minimum Spanning Tree, MST)*** — это остовное дерево связного взвешенного графа, которое включает все его вершины и имеет минимальную возможную сумму весов рёбер.

Построение MST находит широкое применение в задачах оптимизации, таких как *прокладка коммуникационных сетей*, *проектирование инфраструктур* и *минимизация затрат в системах распределения ресурсов*.

В данной работе рассматриваются два классических алгоритма для нахождения MST: ***алгоритм Крускала*** и ***алгоритм Прима***.

* ***Алгоритм Крускала*:** строит MST, начиная с множества всех рёбер, которые упорядочиваются по весу. Алгоритм последовательно добавляет в MST рёбра с минимальным весом, избегая образования циклов. Для оптимизации используется структура данных Union-Find, обеспечивающая эффективное объединение компонент связности и проверку принадлежности вершины компоненте.
* ***Алгоритм Прима*:** начинает построение MST с произвольной вершины, добавляя рёбра минимального веса, которые соединяют уже построенное дерево с непосещёнными вершинами.

***Цель работы*** — изучить реализацию и особенности указанных алгоритмов, сравнить их производительность и продемонстрировать их применение для решения задачи построения минимального остовного дерева. Рассматриваются различные подходы к реализации, включая наивные методы и оптимизации с использованием эффективных структур данных.

## **Постановка задачи**

***Дано:***связный неориентированный граф и вещественная стоимость для каждого ребра

***Найти****:* остовное дерево графа с минимально возможной суммой реберных стоимостей.

***Замечание****: стоимость* ребер может быть отрицательной и не обязательно различна.

***Необходимо реализовать*** *два алгоритма нахождения минимального остовного дерева*:

* Алгоритм Прима;
* Алгоритм Крускала.

***Оценить время выполнения алгоритма*** и ***привести асимптотическую оценку*** временной сложности разработанного решения, обосновав её.

***Проверить решение задачи*** для двух наборах данных:

* Тестовый пример (кол-во вершин: 6, кол-во ребер: 10)
* Усложненный набор данных (кол-во вершин: 500, кол-во ребер: 2184)

## **Описание алгоритмов**

* 1. ***Алгоритм Прима.***

Наивный алгоритм Прима строит минимальное остовное дерево (MST) поэтапно, начиная с произвольной вершины и добавляя рёбра минимальной стоимости, которые соединяют уже построенную часть дерева с остальной частью графа.

***Описание работы алгоритма:***

1. *Инициализация:* выбираем произвольную начальную вершину (например, вершину 0). Отмечаем её как посещённую. Инициализируем список рёбер минимального остовного дерева (MST).
2. *Выбор минимального ребра:* на каждом шаге перебираем все рёбра графа. Находим минимальное ребро, которое соединяет уже посещённую вершину с непосещённой.
3. *Добавление ребра в дерево:* добавляем выбранное ребро в MST. Отмечаем новую вершину как посещённую.

***Остановка.***Алгоритм завершает работу, когда все вершины графа включены в MST (количество рёбер в MST становится равно , где — число вершин).

|  |
| --- |
| ***Prim***  ***Вход:*** связный неориентированный граф представленный в виде списков смежности, и стоимость , для каждого ребра .  ***Выход:*** ребра минимального остовного дерева графа . |
|  |

***Псевдокод 1.*** Псевдокод алгоритма Прима для нахождения минимального остовного дерева.

Рассмотрим на тестовом примере выполнение алгоритма Прима для нахождения минимального остовного дерева.

На таблице 1 можно увидеть представление тестового примера нашей задачи.

|  |  |
| --- | --- |
| Формат данных тестового файла | Данные тестового файла |
| [число\_вершин] [число\_ребер]  [одна\_конечная\_точка\_ребра\_1] [другая\_конечная\_точка\_ребра\_1] [стоимость\_ребра\_1]  [одна\_конечная\_точка\_ребра\_2] [другая\_конечная\_точка\_ребра\_2] [стоимость\_ребра\_2]  … | 6 10  1 2 6  1 4 5  1 5 4  2 4 1  2 5 2  2 3 5  2 6 3  3 6 4  4 5 2  5 6 4 |

***Таблица 1.*** Представление тестового файла.

Итак, приступим к выполнению ***алгоритма Прима*** на тестовом примере.

1. **Инициализация**:

Начинаем с вершины 1.

] (только вершина 1 посещена).

.

1. **Итерация 1**:

Рёбра из 1:

Минимальное ребро:

Добавляем в , обновляем: ,

.

1. **Итерация 2**:

Рёбра:

Минимальное ребро:

Добавляем в , обновляем: ,

1. **Итерация 3**:

Рёбра:

Минимальное ребро:

Добавляем в *MST*, обновляем:

1. **Итерация 4**:

Рёбра:

Минимальное ребро:

Добавляем в *MST*, обновляем: ,

.

1. **Итерация 5**:

Рёбра:

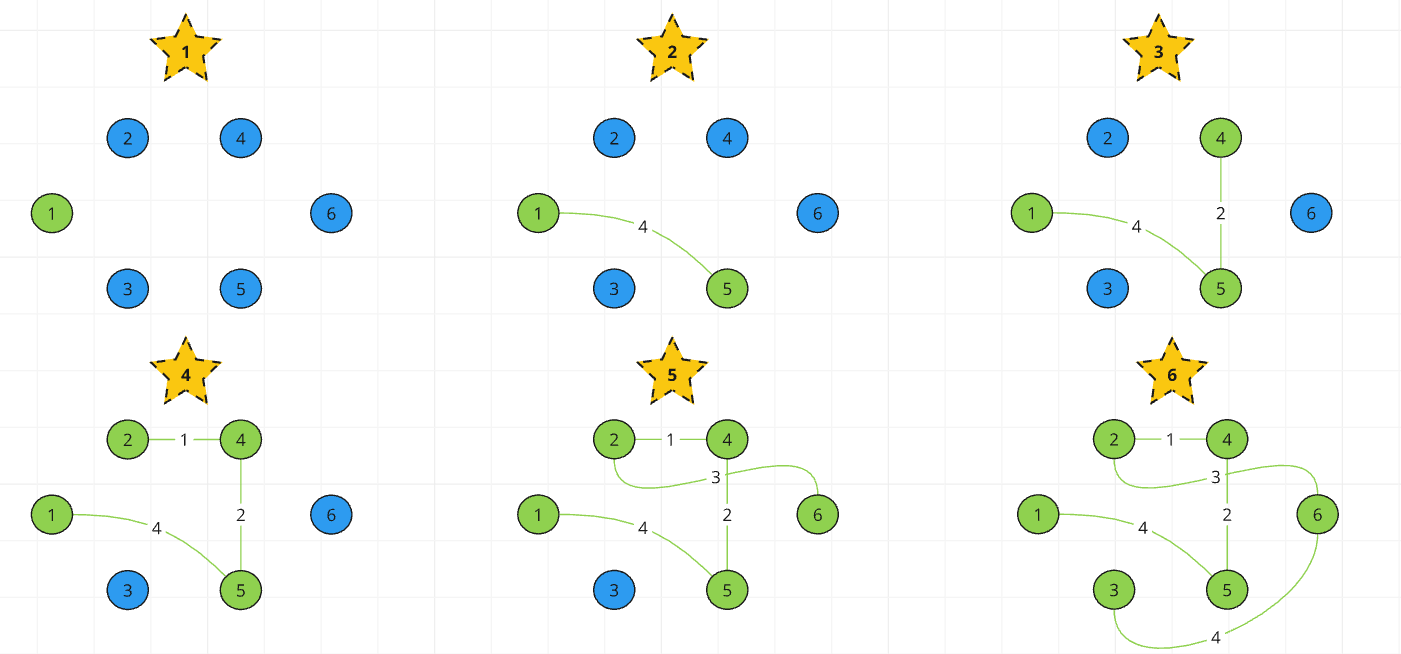
Минимальное ребро:

Добавляем в *MST*, обновляем:

Все вершины посещены.

**Результат (Прим):**

На рисунке 1 показано, как поэтапно строится минимальное остовное дерево на тестовом наборе данных с помощью алгоритма Прима.



***Рисунок 1.*** Поэтапное построение минимального остовного дерева для тестового набора данных с помощью алгоритма Прима.

* 1. ***Алгоритм Крускала (наивная реализация).***

Наивный подход к реализации алгоритма Крускала заключается в управлении компонентами связности вручную, без использования оптимизированной структуры данных (например, *Union-Find*). Мы будем хранить текущие компоненты как множества, обновляя их при добавлении рёбер в остовное дерево.

***Описание алгоритма Крускала (наивный подход):***

1. *Инициализация:* сортируем рёбра графа по их весу. Каждая вершина изначально образует свою собственную компоненту связности.
2. *Добавление рёбер:* перебираем рёбра в порядке возрастания их весов. Если конечные вершины ребра принадлежат разным компонентам, добавляем это ребро в остовное дерево.
3. *Обновление компонент связности:* при добавлении ребра объединяем две компоненты связности, содержащие его вершины.

***Остановка:*** Алгоритм завершает работу, когда в остовное дерево включено рёбер ( — число вершин).

|  |
| --- |
| ***Kruskal***  ***Вход:*** связный неориентированный граф представленный в виде списков смежности, и стоимость , для каждого ребра .  ***Выход:*** ребра минимального остовного дерева графа . |
|  |

***Псевдокод 2.*** Псевдокод алгоритма Крускала (наивная реализация) нахождения минимального остовного дерева.

Теперь посмотрим, чем отличается ***алгоритм Крускала*** от ***алгоритма Прима*** нахождения минимального остовного дерева на тестовом примере:

1. **Сортировка рёбер**:
2. **Инициализация**:
3. **Итерация 1**:

Выбираем (4,2,1), вершины в разных компонентах.  
Добавляем (4,2,1) в MST, обновляем: mst\_cost=1.

1. **Итерация 2**:

Выбираем (5,4,2), вершины в разных компонентах.  
Добавляем (5,4,2) в MST, обновляем: mst\_cost=3.

1. **Итерация 3**:

Выбираем (5,2,2), вершины в одной компоненте.  
Пропускаем.

1. **Итерация 4**:

Выбираем (2,6,3), вершины в разных компонентах.  
Добавляем (2,6,3) в MST, обновляем: mst\_cost=6.

1. **Итерация 5**:

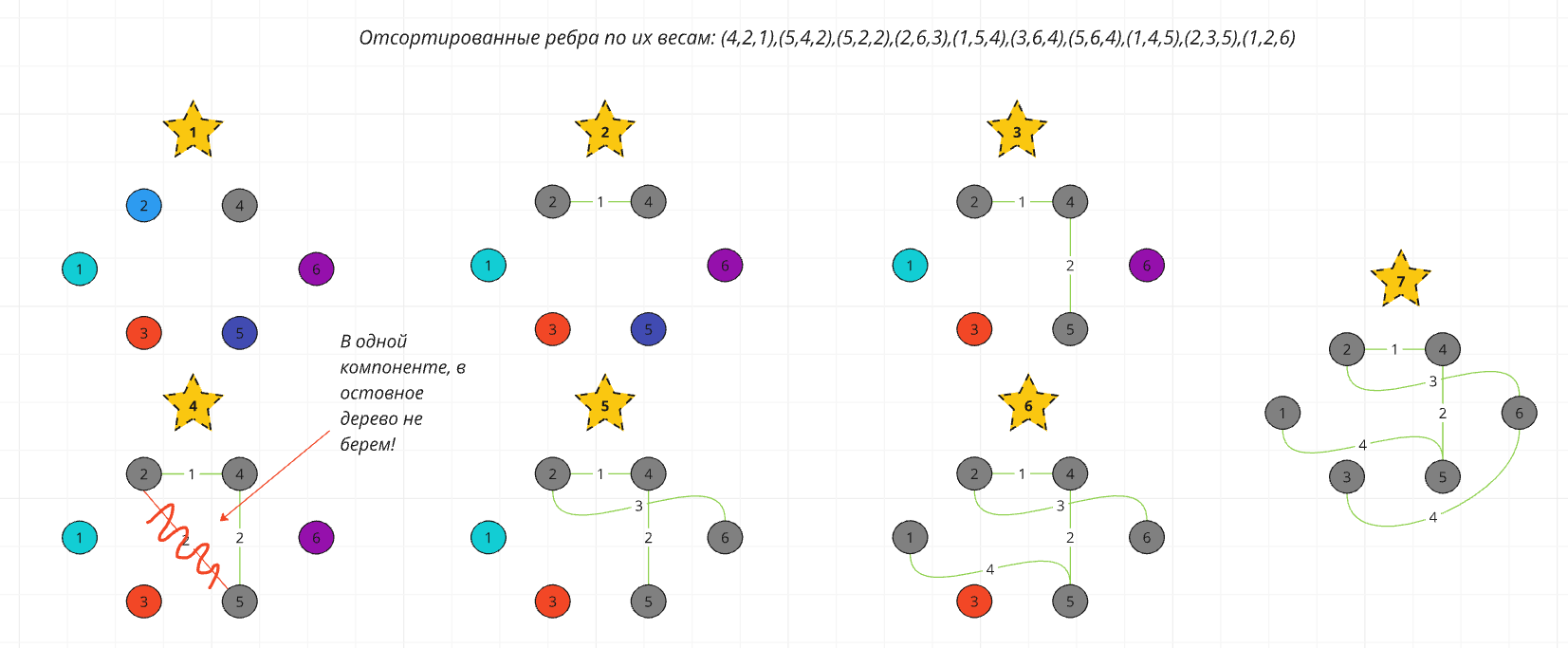
Выбираем (1,5,4), вершины в разных компонентах.  
Добавляем (1,5,4) в MST, обновляем: mst\_cost=10.

1. **Итерация 6**:

Выбираем (3,6,4), вершины в разных компонентах.  
Добавляем (3,6,4) в MST, обновляем: mst\_cost=14.

**Результат (Крускал):**

На рисунке 2 показано, как поэтапно строится минимальное остовное дерево на тестовом наборе данных с помощью алгоритма Крускала.



***Рисунок 2.*** Поэтапное построение минимального остовного дерева для тестового набора данных с помощью алгоритма Крускала.

* 1. ***Алгоритм Крускала (на основе структуры Union\_Find).***

***Алгоритм Крускала*** с использованием структуры данных *Union-Find* (или *Disjoint Set Union, DSU*) является более эффективным подходом для построения минимального остовного дерева (*MST*). Эта структура позволяет быстро объединять компоненты связности и проверять, принадлежат ли две вершины одной компоненте.

***Основная идея***:

1. Использовать *Union-Find* для управления компонентами связности.
2. Сортировать рёбра графа по весу.
3. Перебирать рёбра в порядке возрастания веса и добавлять их в остовное дерево, если вершины, которые они соединяют, принадлежат разным компонентам.

***Описание методов структуры Union\_Find:***

1. ***Метод Find.*** Определяет корневого представителя множества, к которому принадлежит элемент node. Используется техника сжатия пути (path compression), чтобы оптимизировать будущие запросы.

* Если не является своим родителем (т.е. ), выполняется рекурсивный вызов для
* После получения корневого представителя обновляется для всех промежуточных узлов (сжатие пути).

1. ***Метод Union.*** Объединяет два множества, содержащих элементы и . Используется техника сжатия рангов *(union by rank*), чтобы минимизировать высоту деревьев.

* Определяются корневые представители для и с помощью .
* Если представители одинаковые, элементы уже в одном множестве, объединение не требуется.
* В противном случае корень с меньшим рангом подключается к корню с большим рангом (если ранги равны, любой из корней может стать родителем другого, но ранг этого корня увеличивается на 1).

***Описание алгоритма Крускала на основе структуры Union\_Find:***

1. Инициализация: создаём объект Union-Find для отслеживания компонент связности. Сортируем рёбра по весу.
2. Добавление рёбер: перебираем рёбра в порядке возрастания их весов. Если вершины ребра принадлежат разным компонентам (проверка с помощью Union-Find), добавляем ребро в MST и объединяем компоненты.

***Остановка:*** Алгоритм завершает работу, когда в MST включено рёбер ( — число вершин).

|  |
| --- |
| ***Kruskal (на основе структуры Union\_Find)***  ***Вход:*** связный неориентированный граф представленный в виде списков смежности, и стоимость , для каждого ребра .  ***Выход:*** ребра минимального остовного дерева графа . |
| // структура данных Union-Find отсортировать ребра Е по стоимости      *)*  *//* в T нет v-w-пути, поэтому можно добавить    *//* обновить из-за слияния компонент |

***Псевдокод 3.*** Псевдокод алгоритма Крускала (на основе структуры Union\_Find) нахождения минимального остовного дерева.

## **Доказательство корректности**

* 1. ***Алгоритм Прима.***

Для доказательства алгоритма Прима рассмотрим и докажем лемму о безопасном ребре.

***Лемма о безопасном ребре.*** Рассмотрим связный неориентированный взвешенный граф  с весовой функцией . Пусть  — подграф некоторого минимального остовного дерева ,  — разрез , такой, что ни одно ребро из  не пересекает разрез, а  — ребро минимального веса среди всех ребер, пересекающих разрез . Тогда ребро  является безопасным для .

***Доказательство леммы о безопасном ребре:***

Достроим  до некоторого минимального остовного дерева, обозначим его .

Если ребро , то лемма доказана, поэтому рассмотрим случай, когда ребро.

Рассмотрим путь в  от вершины до вершины . Так как эти вершины принадлежат разным долям разреза, то хотя бы одно ребро пути пересекает разрез, назовем его . По условию леммы . Заменим ребро  в  на ребро . Полученное дерево также является минимальным остовным деревом графа , поскольку все вершины *G* по-прежнему связаны и вес дерева не увеличился.

*Следовательно можно дополнить до минимального остовного дерева в графе G, то есть* ***ребро e — безопасное****.*

В алгоритме Прима по поддерживаемым инвариантам после извлечения вершины  из  ребро является ребром минимального веса, пересекающим разрез . Значит, по ***лемме о безопасном ребре.*** оно безопасно.

***Алгоритм построения MST, добавляющий безопасные ребра, причём делающий это ровно  раз, корректен.***

* 1. ***Алгоритм Крускала (наивная реализация).***

Корректность алгоритма Крускала также следует из леммы о безопасном ребре:

1. На каждом шаге алгоритма Крускала мы выбираем ребро с минимальным весом, которое не образует цикл. Это ребро пересекает разрез между уже добавленными вершинами и остальными вершинами графа.
2. Согласно ***лемме о безопасном ребре***, любое ребро, минимальное среди рёбер, пересекающих разрез между выбранными и невыбранными вершинами, является безопасным.
3. Алгоритм добавляет безопасные рёбра в остовное дерево, ***гарантируя, что на каждом шаге не нарушается минимальность и связность дерева.***
4. Алгоритм завершает свою работу, когда добавлено рёбер, что соответствует числу рёбер в минимальном остовном дереве.

***Поскольку алгоритм всегда добавляет безопасные рёбра, которые минимизируют сумму весов рёбер на каждом шаге, в итоге алгоритм Крускала формирует минимальное остовное дерево.***

* 1. ***Алгоритм Крускала (на основе структуры Union\_Find).***

Для доказательства корректности алгоритма Крускала с использованием структуры данных *Union-Find*, будем рассматривать, как алгоритм выбирает рёбра для минимального остовного дерева (*MST*) и как гарантируется, что этот процесс всегда ведет к корректному решению.

1. ***Механизм выбора рёбер:***

* Алгоритм Крускала выбирает рёбра в порядке их веса, начиная с минимального. На каждом шаге алгоритм выбирает ребро с минимальным весом, которое не образует цикла в остовном дереве.
* *Проверка на цикличность осуществляется с помощью операции find структуры Union-Find*: если две вершины, соединяемые текущим ребром, уже находятся в одной компоненте, то добавление этого ребра приведет к циклу, и оно не добавляется в остовное дерево.

1. ***Обоснование минимальности:***

* Алгоритм всегда выбирает ребро с минимальным весом среди доступных рёбер, что является ключом к построению минимального остовного дерева.
* Если бы алгоритм выбрал ребро с большим весом, то оно нарушало бы минимальность остовного дерева, так как всегда существует ребро с меньшим весом, которое соединяет те же вершины.

1. ***Использование Union-Find для избегания циклов:***

* При добавлении нового ребра алгоритм проверяет с помощью и принадлежат ли вершины 𝑢 и 𝑣 одной компоненте. Если да — ребро не добавляется, так как это приведет к циклу.
* Если вершины принадлежат разным компонентам, то они объединяются в одну компоненту с помощью операции .

1. ***Теорема о корректности с Union-Find:***

* Если граф связен, то алгоритм Крускала, с использованием Union-Find, всегда находит минимальное остовное дерево. Это связано с тем, что каждый раз выбираем минимальное доступное ребро, которое не создает цикл (проверка через Union-Find), и продолжаем строить дерево до тех пор, пока не соединятся все вершины.

***Алгоритм завершится после добавления рёбер, что соответствует количеству рёбер в минимальном остовном дереве. Все рёбра, которые соединяют вершины, будут выбраны на каждом шаге, при этом ни одно не будет создавать цикл, благодаря использованию Union-Find.***

## **Асимптотическая оценка временной сложности алгоритмов**

* 1. ***Алгоритм Прима.***

***Наивный алгоритм Прима*** выполняет следующие операции:

1. На каждом шаге выбирается минимальное ребро, соединяющее уже добавленную вершину с не добавленной, что требует времени, поскольку нужно пройти по всем рёбрам. Также для каждой новой добавленной вершины нужно обновить информацию о минимальных рёбрах, что также требует времени.
2. Алгоритм выполняет шагов, где — количество вершин в графе (ведь в конце нужно добавить все вершины в дерево).

Таким образом, общая асимптотическая сложность наивного алгоритма Прима:

где:

* — количество вершин,
* — количество рёбер.
  1. ***Алгоритм Крускала (наивная реализация).***

***Наивный алгоритм Крускала*** выполняет следующие операции:

1. Для сортировки всех рёбер по весу используется стандартный алгоритм сортировки, например, быстрая сортировка или сортировка слиянием, которые имеют асимптотику где — количество рёбер в графе.
2. Для каждого ребра требуется времени для проверки, образует ли оно цикл. Всего будет обработано рёбер, поэтому проверка всех рёбер занимает

Общая асимптотическая сложность наивного алгоритма Крускала:

Так как и — это разные порядки величин, нужно учесть, что для больших графов будет доминировать, если значительно больше . Поэтому итоговая сложность алгоритма будет:

где:

* — количество вершин,
* — количество рёбер.
  1. ***Алгоритм Крускала (на основе структуры Union\_Find).***

Алгоритм Крускала на основе структуры *Union-Find* выполняет следующие операции:

1. Сортировка E займет , как и в случае с наивным алгоритмом Крускала, так как используются все те же алгоритмы сортировки.
2. Работа с СНМ займет где — обратная функция Аккермана, которая не превосходит 4 во всех практических приложениях и которую можно принять за константу.

Таким образом, алгоритм работает за

где:

* — количество вершин,
* — количество рёбер.

## **Результаты запуска программы**

Приступим к тестированию наших алгоритмов на предложенных наборах данных.

Проверять работу наших алгоритмов необходимо на двух наборах данных:

* ***Тестовый пример*** (кол-во вершин: 6, кол-во ребер: 10)
* ***Усложненный набор данных*** (кол-во вершин: 500, кол-во ребер: 2184)

Напомню, что каждый файл описывает экземпляр задачи о нахождении минимального остовного дерева в связном неориентированном графе и имеет следующий формат:

[число\_вершин] [число\_ребер]

[одна\_конечная\_точка\_ребра\_1] [другая\_конечная\_точка\_ребра\_1] [стоимость\_ребра\_1]

[одна\_конечная\_точка\_ребра\_2] [другая\_конечная\_точка\_ребра\_2] [стоимость\_ребра\_2]

…

В таблице 2 приведено сравнение трех алгоритмов нахождения минимального остовного дерева в графе (алгоритма Прима, наивного и оптимизированного алгоритма Крускала) по времени выполнения и вычисленной суммой весов минимального остовного дерева на обоих наборах данных.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Алгоритм | Время выполнения на тестовом наборе данных | Время выполнения на усложненном наборе данных | Результат на тестовом наборе данных (сумма весов минимального остовного дерева) | Результат на усложненном наборе данных  (сумма весов минимального остовного дерева) |
| Алгоритм Прима | < 0.00001 с. | 0.3373689651489258 секунды | 14 | -3612829 |
| Алгоритм Крускала (наивная реализация) | < 0.00001 с. | 0.01189565658569336 секунды | 14 | -3612829 |
| Алгоритм Крускала (на основе структуры Union-Find) | < 0.00001 с. | 0.0015425682067871094 секунды | 14 | -3612829 |

**Таблица 2.** Сравнение алгоритмов нахождения минимального остовного дерева в графе (алгоритма Прима, наивного и оптимизированного алгоритма Крускала).

## **Заключение**

В ходе курсовой работы были рассмотрены следующие ***алгоритмы нахождения минимального остовного дерева*** в связном неориентированном графе:

* Алгоритм Прима (сложность ,
* Наивный алгоритм Крускала (сложность *,*
* Оптимизированный алгоритм Крускала на основе структуры Union-Find (сложность )

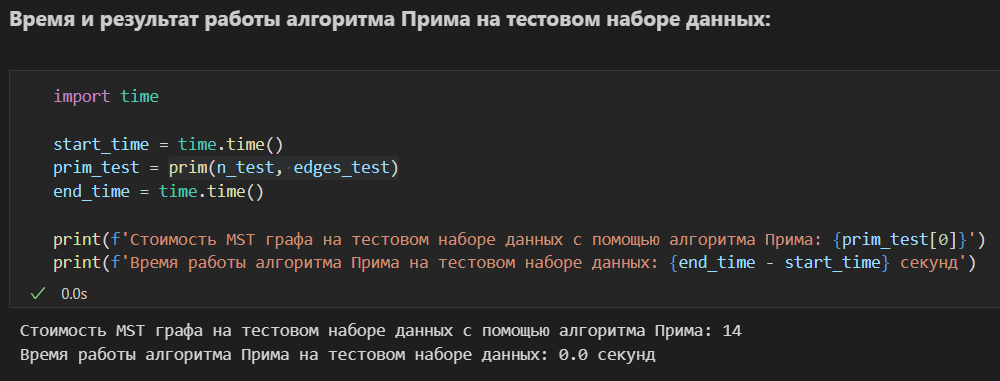
Для всех алгоритмов была доказана корректность, а также они были протестированы на тестовом и усложненном наборе данных, где лучшим по времени выполнения оказался ***оптимизированный алгоритм Крускала на основе структуры Union-Find*,** выполнив задачу на усложненном наборе данных за 0.0015425682067871094 секунды и вычислив результат равный -3612829.

***Данный алгоритм превзошел наивный алгоритм Крускала в 10 раз, а наивный алгоритм Прима в более чем 300 раз.***

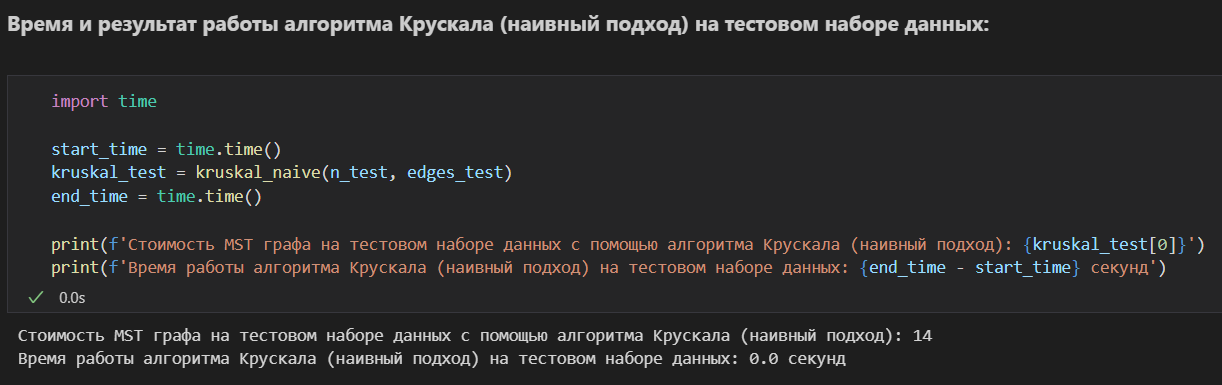
## **Список литературы**

1. Харари Ф. Теория графов. /пер. с англ. — изд. 2-е — М.: Едиториал УРСС, 2003. — 296 с. — ISBN 5-354-00301-6
2. Бхаргава А. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих. - 3-e изд. - СПб: Питер, 2024. - 288 с.
3. Рафгарден Т. Совершенный алгоритм. Основы. - 2-e изд. - СПб: Питер, 2023. - 256 с.
4. Скиена С.С. Алгоритмы. Руководство по разработке. - 2-е изд. - М.: БХБ, 2018. - 720 с.
5. Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн — Алгоритмы: построение и анализ, 2-е издание. Пер. с англ. — М.:Издательский дом "Вильямс", 2010. — 1296 с.: ил. — Парал. тит. англ. — ISBN 978-5-8459-0857-5.

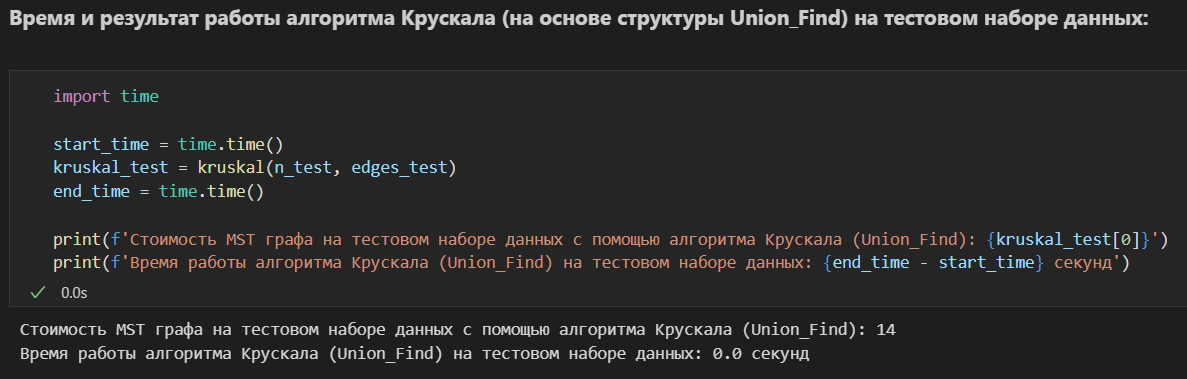
## **Приложения**



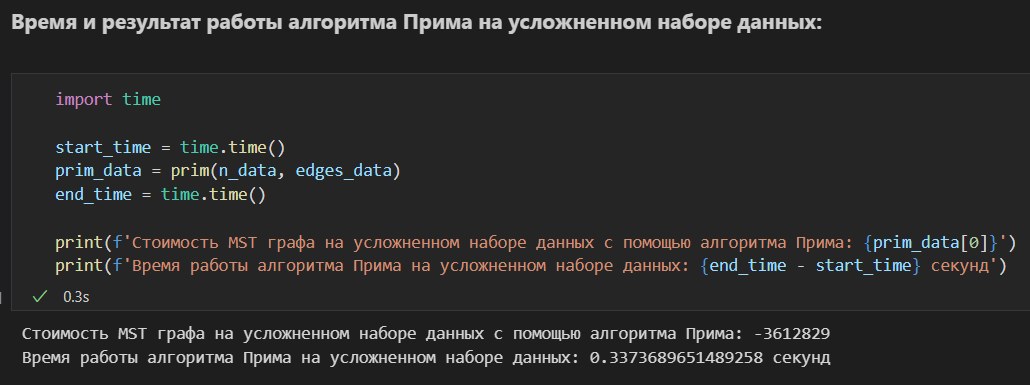
***Рисунок 3.*** Результат выполнения алгоритма Прима на тестовых данных.



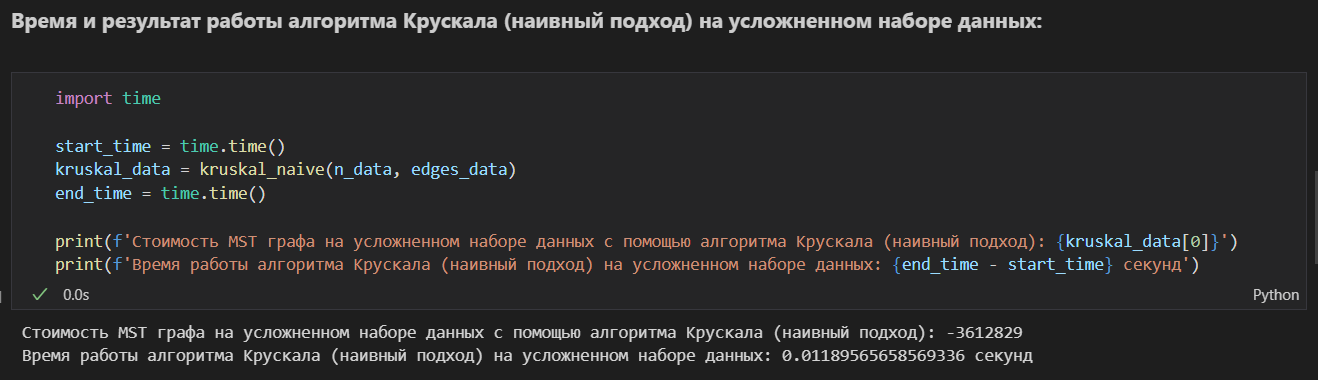
***Рисунок 4.*** Результат выполнения наивного алгоритма Крускала на тестовых данных.



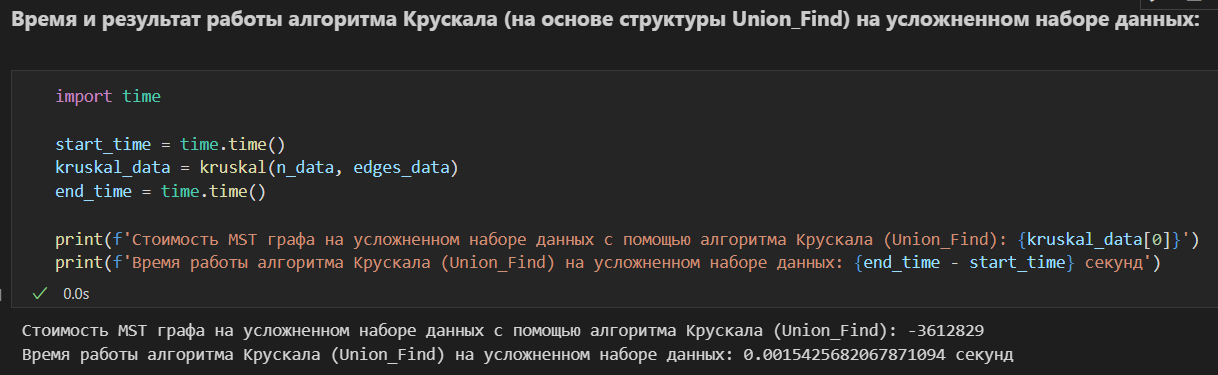
***Рисунок 5.*** Результат выполнения оптимизированного алгоритма Крускала на тестовых данных.



***Рисунок 6.*** Результат выполнения алгоритма Прима на усложненных данных.



***Рисунок 7.*** Результат выполнения наивного алгоритма Крускала на усложненных данных.



***Рисунок 8.*** Результат выполнения оптимизированного алгоритма Крускала на усложненных данных.