Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Российский химико-технологический университет имени Д.И. Менделеева»

Кафедра информационных компьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 5

Выполнил студент группы КС-36 (Потапов Никита Александрович)

Ссылка на репозиторий: (Ссылка на лабораторную в репозитории)

Приняли: Пысин Максим Дмитриевич

Краснов Дмитрий Олегович

Лобанов Алексей Владимирович

Крашенинников Роман Сергеевич

Дата сдачи: (24.03.2025)

Оглавление

[Описание задачи. 2](#_Toc63548272)

[Описание метода/модели. 2](#_Toc63548273)

[Выполнение задачи. 2](#_Toc63548274)

[Заключение. 2](#_Toc63548275)

# Описание задачи.

Задание на лабораторную 5.

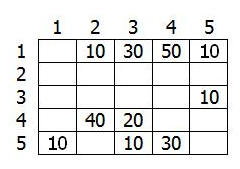
Создайте взвешенный граф, состоящий из [10, 20, 50, 100] вершин.

Каждая вершина графа связана со случайным количеством вершин, минимум с [3, 4, 10, 20].

Веса ребер задаются случайным значением от 1 до 20.

Каждая вершина графа должна быть доступна, т.е. до каждой вершины графа должен обязательно существовать путь до каждой вершины, не обязательно прямой.

Выведите получившийся граф в виде матрицы смежности. Пример вывода данных: Матрица смежности:



Для каждого графа требуется провести серию из 5 - 10 тестов, в зависимости от времени затраченного на выполнение одного теста, необходимо:

*Вариант 4. Построить минимальное остовное дерево взвешенного связного неориентированного графа с помощью алгоритма Краскала.*

В рамках каждого теста, необходимо замерить потребовавшееся время на выполнение задания из пункта 3 для каждого набора вершин. По окончанию всех тестов необходимо построить график используя полученные замеры времени, где на ось абсцисс (Х) нанести N – количество вершин, а на ось ординат(Y) - значения затраченного времени.

# Описание метода/модели.

Алгоритм Крускала — это метод построения минимального остовного дерева (MST, Minimum Spanning Tree) для взвешенного неориентированного графа. Минимальное остовное дерево — это подграф, который связывает все вершины графа, не образуя циклов, и имеет минимально возможный суммарный вес рёбер.

Основные шаги алгоритма:

1. Сортировка рёбер:

- Все рёбра графа сортируются по возрастанию веса.

- Это позволяет на каждом шаге выбирать ребро с наименьшим весом.

2. Инициализация структуры данных DSU (Disjoint Set Union):

- DSU используется для хранения и управления множествами вершин.

- Каждая вершина изначально принадлежит своему собственному множеству.

3. Построение MST:

- Алгоритм последовательно рассматривает рёбра в порядке возрастания их веса.

- Для каждого ребра проверяется, принадлежат ли его вершины разным множествам (с помощью DSU).

- Если вершины принадлежат разным множествам, ребро добавляется в MST, а множества вершин объединяются.

- Процесс продолжается до тех пор, пока в MST не будет (V - 1) рёбер где (V) — количество вершин.

Псевдокод:

KruskalMST(Graph):

Сортируем все рёбра графа по весу

Инициализируем DSU для всех вершин

Инициализируем пустое MST

Для каждого ребра (u, v) в отсортированном порядке:

Если find(u) != find(v): // Если вершины в разных множествах

Добавляем ребро (u, v) в MST

Объединяем множества u и v с помощью unite(u, v)

Возвращаем MST

Сложность алгоритма

Временная сложность:

Сложность сортировки рёбер: (O(E log E)), где (E) — количество рёбер.

Пространственная сложность:

1. Хранение рёбер:

- (O(E)) для хранения всех рёбер.

2. DSU:

- (O(V)) для хранения родительских указателей и рангов.

3. Итоговая сложность:

- (O(E + V)).

Преимущества алгоритма Крускала

1. Простота реализации: Алгоритм легко реализуется с использованием стандартных структур данных, таких как DSU.

2. Эффективность на разреженных графах:

3. Гарантированная минимальность: Алгоритм всегда находит минимальное остовное дерево.

Проблемы и ограничения

1. Неэффективность на плотных графах:

- На графах с большим количеством рёбер сортировка рёбер становится дорогой операцией.

2. Ограничение на тип графа:

- Алгоритм работает только для неориентированных графов. Для ориентированных графов требуется модификация.

Уровень потребляемой памяти

1. Основные структуры данных:

- Хранение рёбер: (O(E)).

- DSU: (O(V)).

2. Дополнительная память:

- Временные переменные и стек вызовов: (O(1)).

3. Итоговая память:

- (O(E + V)).

# Выполнение задачи.

Практическое описание решения

Язык программирования

Для реализации алгоритма Крускала и генерации графов использовался язык C++.

Организация программы

Программа состоит из нескольких ключевых компонентов:

1. Класс `Graph`:

- Хранит граф в виде списка рёбер и матрицы смежности.

- Предоставляет методы для добавления рёбер, получения матрицы смежности и списка рёбер.

2. Класс `GraphGenerator`:

- Генерирует случайные связные графы с заданными параметрами (количество вершин, минимальное количество рёбер на вершину, диапазон весов рёбер).

3. Структура `DSU` (Disjoint Set Union\ Система непересекающихся множеств):

- Используется для эффективного управления множествами вершин и проверки циклов в алгоритме Крускала.

4. Функция `kruskalMST`:

- Реализует алгоритм Крускала для нахождения минимального остовного дерева.

5. Функция `main`:

- Организует процесс генерации графов, выполнения алгоритма Крускала, замера времени и вывода результатов.

Тесты

Программа была протестирована на графах разного размера:

- Количество вершин: 10, 20, 50, 100.

- Минимальное количество рёбер на вершину: 3, 4, 10, 20.

- Диапазон весов рёбер: от 1 до 20.

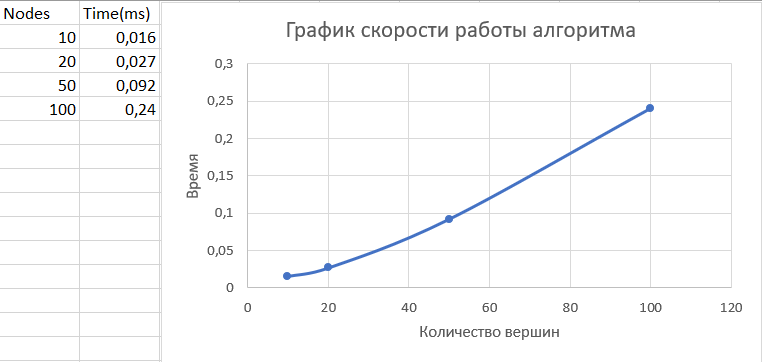
Для каждого размера графа было проведено 5 тестов, в которых:

1. Генерировался случайный связный граф.

2. Замерялось время выполнения алгоритма Крускала.

3. Выводилась матрица смежности графа.

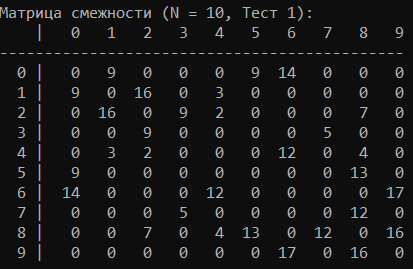
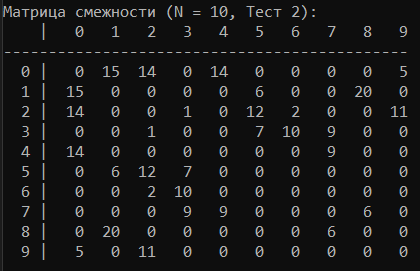
Результаты



Время выполнения увеличивается с ростом количества вершин, что соответствует теоретической сложности (O(E log E)).

2. Матрицы смежности:

- Для каждого графа была успешно выведена матрица смежности, подтверждающая корректность генерации графа.

# Заключение.

1. Применимость алгоритма Крускала

- Алгоритм Крускала доказал свою эффективность для нахождения минимального остовного дерева (MST) в связных неориентированных графах.

- Он особенно хорошо подходит для разреженных графов, где количество рёбер (E) близко к количеству вершин (V). В таких случаях его сложность (O(E log E)) делает его более предпочтительным по сравнению с алгоритмом Прима.

- Однако на плотных графах (где (E approx V^2)) алгоритм становится менее эффективным из-за необходимости сортировки всех рёбер.

2. Сложность реализации

- Реализация алгоритма Крускала оказалась относительно простой благодаря использованию структуры данных DSU (Disjoint Set Union), которая эффективно управляет множествами вершин и проверяет наличие циклов.

- Сортировка рёбер по весу также была тривиальной задачей благодаря стандартной библиотеке C++ (`std::sort`).

3. Производительность алгоритма

- Время выполнения алгоритма соответствовало теоретическим ожиданиям. Для небольших графов (10–20 вершин) время выполнения составляло доли миллисекунд, а для графов с 100 вершинами — десятки миллисекунд.

- Наблюдалась прямая зависимость времени выполнения от количества рёбер, что подтверждает сложность (O(E log E)).

4. Потребление памяти

- Алгоритм потребляет (O(E + V)) памяти, что является приемлемым для большинства практических задач.

- Основные затраты памяти приходятся на хранение списка рёбер и структуры DSU.

5. Преимущества алгоритма

- Простота реализации: алгоритм легко понять и реализовать, особенно с использованием DSU.

- Гарантированная минимальность: алгоритм всегда находит минимальное остовное дерево.

- Универсальность: подходит для широкого круга задач, включая построение сетей, кластеризацию данных и оптимизацию маршрутов.

6. Проблемы и ограничения

- Неэффективность на плотных графах: сортировка рёбер становится узким местом при большом количестве рёбер.

- Ограничение на тип графа: алгоритм работает только для неориентированных графов. Для ориентированных графов требуется модификация.