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 Акулинин Андрей Игоревич

Ссылка на репозиторий: <https://github.com/MUCTR-IKT-CPP/AkulininAI_36/tree/main/algorithms>

Приняли: Пысин Максим Дмитриевич

Краснов Дмитрий Олегович

Лобанов Алексей Владимирович

Крашенинников Роман Сергеевич

Дата сдачи: 17.03.2025

Оглавление

[Описание задачи. 2](#_Toc63548272)

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

[Выполнение задачи. 2](#_Toc63548274)

[Заключение. 2](#_Toc63548275)

# Описание задачи.

Задача заключалась в создании взвешенного графа с заданным количеством вершин (10, 20, 50, 100), где каждая вершина связана с минимальным количеством других вершин (3, 4, 10, 20 соответственно). Веса ребер задавались случайным образом в диапазоне от 1 до 20. Основное требование — граф должен быть связным, то есть между любыми двумя вершинами должен существовать путь. После создания графа необходимо было построить минимальное остовное дерево (MST) с использованием алгоритма Прима и измерить время выполнения этого алгоритма для каждого графа. На основе полученных данных требовалось построить график зависимости времени выполнения от количества вершин.

# Описание метода/модели.

Для решения задачи использовались следующие структуры данных и алгоритмы:

1. **Граф**:
   * Граф представлен в виде матрицы смежности, где каждая ячейка [i][j] содержит вес ребра между вершинами i и j. Если ребра нет, значение равно 0.
   * Матрица смежности удобна для работы с плотными графами, но требует O(V2) памяти, где V — количество вершин.
2. **Алгоритм Прима**:
   * Алгоритм Прима используется для построения минимального остовного дерева (MST) в связном взвешенном неориентированном графе.
   * Основная идея: на каждом шаге добавляется ребро с минимальным весом, соединяющее уже добавленные вершины с оставшимися.
   * В данной реализации используется линейный поиск для нахождения вершины с минимальным ключом, что делает сложность алгоритма O(V2). Это подходит для небольших графов, но для больших графов (например, 100 вершин) эффективнее использовать приоритетную очередь, что снизит сложность до O(logV), где E — количество ребер.
3. **Генерация графа**:
   * Граф генерируется случайным образом, но с соблюдением условия связности. Для этого сначала создается связный граф, а затем добавляются дополнительные ребра, чтобы каждая вершина имела минимальное количество связей.
4. **Тестирование**:
   * Для каждого графа проводится серия тестов (5-10 запусков), чтобы измерить среднее время выполнения алгоритма Прима.
   * Результаты записываются в файл для последующего анализа и построения графика

Изображение выглядит как линия, График, диаграмма, текст

Контент, сгенерированный ИИ, может содержать ошибки.

# Выполнение задачи.

1. **Язык программирования и организация программы**:
   * Программа написана на языке C++.
   * Код разделен на классы:
     + Graph: отвечает за хранение графа (матрица смежности) и реализацию алгоритма Прима.
     + GraphGenerator: отвечает за генерацию случайного связного графа с заданными параметрами.
     + Основная логика тестирования и вывода результатов находится в функции main.
2. **Тесты**:
   * Для каждого набора вершин (10, 20, 50, 100) и минимального количества ребер (3, 4, 10, 20) генерируется граф.
   * Для каждого графа выполняется 10 тестов, в которых измеряется время выполнения алгоритма Прима.
   * Результаты (среднее время выполнения) записываются в файл results.csv.
3. **Результаты**:
   * Время выполнения алгоритма Прима увеличивается с ростом количества вершин. Например:
     + Для 10 вершин: ~50 микросекунд.
     + Для 100 вершин: ~5000 микросекунд.
   * Полученные данные используются для построения графика зависимости времени выполнения от количества вершин.

# Заключение.

1. **Применимость алгоритма**:
   * Алгоритм Прима хорошо подходит для нахождения минимального остовного дерева в небольших и средних графах. Однако для больших графов (например, 1000 вершин) его реализация с линейным поиском будет неэффективной. В таких случаях рекомендуется использовать приоритетную очередь.
2. **Сложность реализации**:
   * Реализация алгоритма Прима на основе матрицы смежности проста и понятна, но требует O(V2) памяти. Для более сложных графов (разреженных) лучше использовать списки смежности.
3. **Рекомендации**:
   * Для улучшения производительности можно оптимизировать алгоритм Прима, используя приоритетную очередь.
   * Для анализа больших графов можно рассмотреть другие алгоритмы, такие как Крускала, который может быть более эффективным для разреженных графов.

В целом, задача была успешно выполнена, а результаты подтвердили теоретические ожидания относительно сложности алгоритма Прима.