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 № 4

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.

В рамках лабораторной работы необходимо было реализовать генератор случайных графов, который позволяет создавать графы с заданными параметрами, такими как количество вершин, количество ребер, направленность графа, а также ограничения на количество входящих и выходящих ребер. Сгенерированные графы должны быть представлены в виде матрицы смежности, матрицы инцидентности, списка смежности и списка ребер.

После генерации графов требовалось провести анализ эффективности двух алгоритмов поиска пути в графе: поиска в ширину (BFS) и поиска в глубину (DFS). Для этого необходимо было:

1. Сгенерировать 10 графов с возрастающим количеством вершин и ребер.
2. Для каждого графа случайным образом выбрать начальную и конечную вершины.
3. Замерить время выполнения BFS и DFS для поиска пути между выбранными вершинами.
4. Результаты замеров записать в файл и проанализировать эффективность алгоритмов.

# Описание метода/модели.

**Графы**

Граф — это структура данных, состоящая из вершин (узлов) и ребер (связей между вершинами). Графы могут быть:

* **Ориентированными**: ребра имеют направление (дуги).
* **Неориентированными**: ребра не имеют направления.

Для представления графов используются следующие структуры:

1. **Матрица смежности**: квадратная матрица, где элемент [i][j] равен 1, если существует ребро из вершины i в вершину j, и 0 в противном случае.
2. **Матрица инцидентности**: матрица, где строки соответствуют вершинам, а столбцы — ребрам. Элемент [i][j] равен 1, если вершина i инцидентна ребру j, и 0 в противном случае.
3. **Список смежности**: массив списков, где каждый элемент массива соответствует вершине, а список содержит вершины, смежные с ней.
4. **Список ребер**: список пар вершин, представляющих ребра графа.

**Алгоритмы поиска**

1. **Поиск в ширину (BFS)**:
   * Использует очередь для обхода графа.
   * Находит кратчайший путь в невзвешенном графе.
   * Сложность: O(V + E), где V — количество вершин, E — количество ребер.
   * Потребляет память порядка O(V) для хранения очереди и меток посещенных вершин.
2. **Поиск в глубину (DFS)**:
   * Использует стек (рекурсию) для обхода графа.
   * Не гарантирует нахождение кратчайшего пути.
   * Сложность: O(V + E).
   * Потребляет память порядка O(V) для хранения стека и меток посещенных вершин.

**Преимущества и недостатки**

* **BFS**:
  + Преимущества: находит кратчайший путь, хорошо работает на разреженных графах.
  + Недостатки: требует больше памяти для хранения очереди.
* **DFS**:
  + Преимущества: требует меньше памяти, хорошо работает на графах с длинными путями.
  + Недостатки: не гарантирует кратчайший путь, может зацикливаться на графах с циклами.

# Выполнение задачи.

**Язык программирования и организация программы**

Для реализации задачи был выбран язык C++. Программа состоит из двух основных классов:

1. **Graph**:
   * Реализует структуру графа.
   * Содержит методы для добавления ребер, заполнения матриц, вывода данных и выполнения BFS/DFS.
2. **RandomGeneratorGraph**:
   * Генерирует случайные графы с заданными параметрами.

**Тестирование**

1. Было сгенерировано 10 графов с количеством вершин от 5 до 20 и количеством ребер от 10 до 50.
2. Для каждого графа случайным образом выбирались начальная и конечная вершины.
3. Для каждого графа выполнялись BFS и DFS, замерялось время выполнения.
4. Результаты записывались в файл results.csv.

**Результаты**

* Время выполнения BFS и DFS увеличивалось с ростом количества вершин и ребер.
* BFS показал более стабильное время выполнения на графах с большим количеством ребер.
* DFS оказался быстрее на графах с небольшим количеством ребер, но его время выполнения сильно варьировалось в зависимости от структуры графа.

Изображение выглядит как линия, диаграмма, График

Контент, сгенерированный ИИ, может содержать ошибки.

# Заключение.

В ходе выполнения лабораторной работы были реализованы генератор случайных графов и алгоритмы BFS и DFS. Проведенные тесты показали, что:

1. **BFS** эффективен для поиска кратчайшего пути в невзвешенных графах, особенно на графах с большим количеством ребер.
2. **DFS** может быть быстрее на графах с небольшим количеством ребер, но не гарантирует нахождение кратчайшего пути.
3. Время выполнения обоих алгоритмов зависит от структуры графа: BFS более стабилен, а DFS может быть непредсказуемым на графах с циклами.

Реализация задачи оказалась достаточно простой благодаря использованию стандартных структур данных C++ (векторы, очереди, стеки). Однако для более сложных графов (например, взвешенных или с большим количеством вершин) потребуется оптимизация алгоритмов и использование более эффективных структур данных.

В целом, лабораторная работа позволила лучше понять особенности работы с графами и применение алгоритмов поиска на практике.