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 (Меркушов Даниил Игоревич)

Ссылка на репозиторий: (Ссылка на лабораторную в репозитории)

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 реализовать генератор случайных графов, генератор должен содержать следующие параметры:

Максимальное/Минимальное количество генерируемых вершин

Максимальное/Минимальное количество генерируемых ребер

Максимальное количество ребер связанных с одной вершины

Генерируется ли направленный граф

Максимальное количество входящих и выходящих ребер

Сгенерированный граф должен быть описан в рамках одного класса (этот класс не должен заниматься генерацией), и должен обладать обязательно следующими методами:

Выдача матрицы смежности

Выдача матрицы инцидентности

Выдача список смежности

Выдача списка ребер

В качестве проверки работоспособности, требуется сгенерировать 10 графов с возрастающим количеством вершин и ребер (количество выбирать в зависимости от сложности расчета для вашего отдельно взятого ПК). На каждом из сгенерированных графов требуется выполнить поиск кратчайшего пути или подтвердить его отсутствие из точки А в точку Б, выбирающиеся случайным образом заранее, поиском в ширину и поиском в глубину, замерев время требуемое на выполнение операции. Результаты замеров наложить на график и проанализировать эффективность применения обоих методов к этой задаче.

# 

# Описание метода/модели.

Язык: Golang, Python

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

Направленными (ориентированными): Ребра имеют направление, то есть ребро из вершины

A в вершину B не означает наличие ребра из B в A.

Ненаправленными (неориентированными): Ребра не имеют направления, и связь между вершинами двусторонняя.

Графы могут быть представлены в виде:

Матрицы смежности: Двумерный массив, где элементы matrix[i][j] указывает на наличие или отсутствие ребра между вершинами.

Списка смежности: Массив списков, где каждый элемент массива соответствует вершине, а список содержит вершины, смежные с ней.

Списка ребер: Список пар вершин, представляющих ребра графа.

Алгоритмы поиска пути

В данной работе реализованы два алгоритма поиска пути в графе:

* Поиск в ширину (BFS, Breadth-First Search).
* Поиск в глубину (DFS, Depth-First Search).

Поиск в ширину (BFS)

Описание:

BFS исследует граф "слой за слоем". Алгоритм начинает с начальной вершины, посещает все вершины на расстоянии 1, затем на расстоянии 2 и так далее. Для реализации используется очередь (FIFO).

Шаги алгоритма:

* Добавить начальную вершину в очередь и пометить её как посещённую.
* Пока очередь не пуста:
* Извлечь вершину из очереди.
* Для каждой смежной вершины, если она не посещена, добавить её в очередь и пометить как посещённую.
* Если целевая вершина найдена, завершить алгоритм.

Сложность:

Временная сложность: O(V+E), где V — количество вершин, E — количество ребер.

Пространственная сложность: O(V) для хранения очереди и меток посещения.

Преимущества:

Находит кратчайший путь в невзвешенном графе.

Гарантирует нахождение пути, если он существует.

Недостатки:

Требует больше памяти, чем DFS, так как хранит все вершины текущего уровня.

**Поиск в глубину (DFS)**

Описание:

DFS исследует граф, двигаясь "вглубь" до тех пор, пока это возможно. Алгоритм использует стек для реализации.

Шаги алгоритма:

* Добавить начальную вершину в стек и пометить её как посещённую.
* Пока стек не пуст:
* Извлечь вершину из стека.
* Для каждой смежной вершины, если она не посещена, добавить её в стек и пометить как посещённую.
* Если целевая вершина найдена, завершить алгоритм.

Сложность:

Временная сложность: O(V+E), где V — количество вершин, E — количество ребер.

Пространственная сложность: O(V) для хранения стека и меток посещения.

Преимущества:

Требует меньше памяти, чем BFS, так как хранит только текущий путь.

Подходит для задач, где нужно исследовать все возможные пути (например, поиск циклов).

Недостатки:

Не гарантирует нахождение кратчайшего пути.

Может зациклиться, если граф содержит циклы, и не используются метки посещения.

# 

# Выполнение задачи.

**Структура Graph**

Хранит данные графа: направленность (directed), количество вершин (vertices), список рёбер (edges) и список смежности (adjacencyList).

**Методы:**

**generateAdjacencyList():** преобразует список рёбер в список смежности.

**AdjacencyMatrix(), IncidenceMatrix():** создают матрицу смежности и инцидентности.

**EdgeList(), AdjacencyList():** возвращают список рёбер и список смежности.

**Генератор графов (Generator)**

Параметры: диапазоны вершин/рёбер, ограничения на степени (для направленных и ненаправленных графов).

**Метод Generate():**

Случайно выбирает количество вершин (V) и рёбер. Генерирует уникальные рёбра с учётом направленности и ограничений на степени. Исключает петли (рёбра вида a-a) и дубликаты.

**Алгоритмы BFS и DFS BFS**: использует очередь для поиска кратчайшего пути (в ненагруженном графе). DFS: использует стек для поиска любого пути. Оба алгоритма возвращают путь между вершинами или nil, если путь не найден.

**Основная функция (main)**

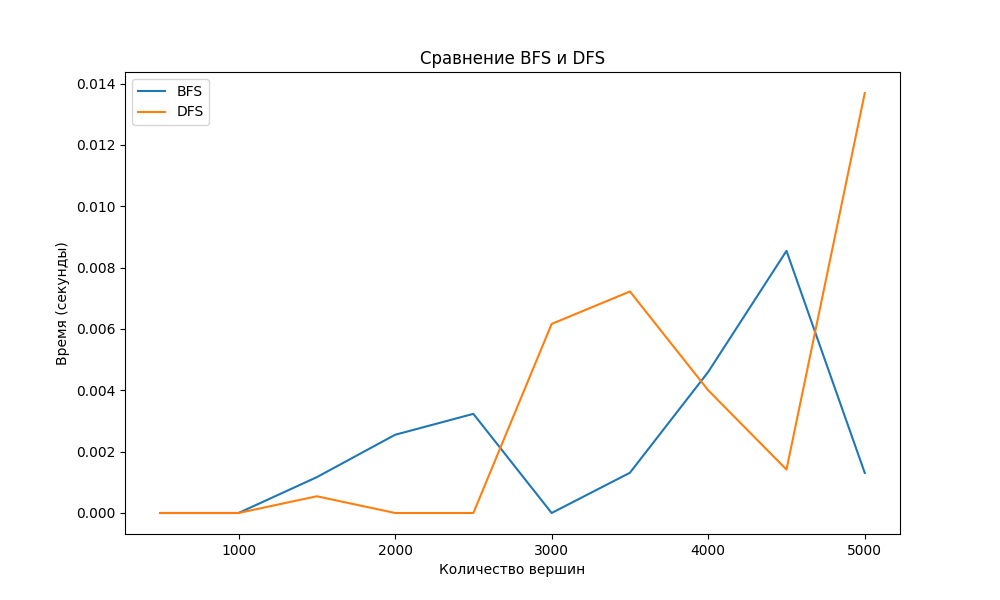
Генерирует 10 графов с увеличивающимся количеством вершин

Выбирает случайные стартовую (A) и конечную (B) вершины.

Замеряет время выполнения BFS и DFS.

Сохраняет результаты в CSV-файл (benchmark\_results.csv).

Для первых 10 графов выводит: Список рёбер. Список смежности (первые 10 вершин). Матрицу смежности (10x10).



# Заключение.

**Применимость графов**

Графы являются универсальной структурой данных, которая может быть использована для моделирования широкого спектра задач

В данной работе графы использовались для моделирования случайных сетей, что подтвердило их гибкость и применимость для задач поиска пути.

**Применимость алгоритмов BFS и DFS**

**BFS:**

Идеально подходит для поиска кратчайшего пути в невзвешенных графах.

Эффективен в задачах, где важно минимальное расстояние (например, маршрутизация в сетях).

Однако требует больше памяти, что может быть проблемой для очень больших графов.

**DFS:**

Подходит для задач, где нужно исследовать все возможные пути (например, поиск циклов, топологическая сортировка).

Требует меньше памяти, чем BFS, но не гарантирует нахождение кратчайшего пути.

Может зациклиться, если не используются метки посещения.

Вывод: Выбор между BFS и DFS зависит от задачи. Если нужен кратчайший путь, лучше использовать BFS. Если важно сэкономить память или исследовать все пути, предпочтительнее DFS.

**Замечания:**

Генерация графов с ограничениями потребовала больше времени, чем ожидалось.

Для очень больших графов (тысячи вершин) производительность BFS и DFS может быть недостаточной.