МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Вятский государственный университет»**

(ВятГУ)

**ОТЧЕТ**

**ПО УЧЕБНОЙ ПРАКТИКЕ №4**

Подготовил студент группы ФИб-3301-51-00

Ощепков Д. О.

Киров, 2024 г

Содержание

[Постановка задачи 3](#_Toc165296914)

[Введение 5](#_Toc165296915)

[1 Основные алгоритмы поиска путей в графе 5](#_Toc165296916)

[1.1 Алгоритм Дейкстры (Dijkstra's Algorithm) 5](#_Toc165296917)

[1.2 Алгоритм Беллмана-Форда (Bellman-Ford Algorithm) 5](#_Toc165296918)

[1.3 Алгоритм A (A-star Algorithm) 6](#_Toc165296919)

[1.4 Алгоритм поиска в ширину (BFS - Breadth-First Search) 6](#_Toc165296920)

[2 Хранение графа в памяти компьютера 7](#_Toc165296921)

[2.1 Списки смежности (Adjacency Lists) 7](#_Toc165296922)

[2.2 Матрица смежности (Adjacency Matrix) 8](#_Toc165296923)

[2.3 Список рёбер (Edge List) 8](#_Toc165296924)

[3 Обзор использованных технологий 8](#_Toc165296925)

[3.1 Google Benchmark 9](#_Toc165296926)

[3.2 Google Test 9](#_Toc165296927)

[3.3 Visual C++ Package Manager 10](#_Toc165296928)

[3.4 Visual Studio 2022 11](#_Toc165296929)

[4 Реализация алгоритмов и структур данных 13](#_Toc165296930)

[4.1 Отношения классов 13](#_Toc165296931)

[4.2 Генерация тестовых графов 14](#_Toc165296932)

[4.3 Создание тестов 14](#_Toc165296933)

[5 Результаты бенчмарков 14](#_Toc165296934)

[Заключение 18](#_Toc165296935)

[Приложения 18](#_Toc165296936)

[Приложение А. Листинг программы 18](#_Toc165296937)

[Приложение B. Генерация графов 29](#_Toc165296938)

[Приложение C. Бенчмарки 32](#_Toc165296939)

[Приложение D. Тесты 35](#_Toc165296940)

# Постановка задачи

**Дано**: неориентированный, невзвешенный граф G с количеством вершин V [100…30000].

**Цель**:

1. разработка алгоритма, позволяющих последовательно находить в G набор кратчайших путей между заданными парами вершин;
2. определить оптимальное представление графа для разрабатываемого алгоритма. Критерий оптимальности - соотношение скорости работы к требуемым ресурсам.

**Задачи:**

1. Реализовать не менее двух способов представления G в памяти ПЭВМ
2. Разработать алгоритм, реализующий поиск набора кратчайших путей в G
3. Реализовать алгоритм на языке С/С++ с учётом различных представлений G.
4. Провести тестирование реализованных алгоритма на разных входных данных с замером времени работы алгоритма и потреблением памяти
5. Свести полученную статистику в мини-отчёт.

**Ограничения:**

1. Вершины входящие в какой-либо путь не могут быть использованы при построении других путей. Но конечные точки путей могут совпадать;
2. Количество путей - не более 10;

**Входные и выходные данные:**

1. Должны быть представлены в текстовых файлах;
2. Найденный путь необходимо вывести в виде последовательности номеров/названий вершин;
3. Если пути между вершинами нет, то необходимо вывести «Пути нет».

**Результат практики:**

1. Алгоритмы, представленные в лексическом виде на русском языке или графическом (блок схема);
2. Реализованные алгоритмы на языке программирования,
3. Мини-отчёт.

# Введение

В мире компьютерных наук и приложений важное место занимает задача поиска кратчайших путей в графах.

Поиск кратчайших путей в графе заключается в нахождении пути между двумя вершинами, который имеет наименьшую суммарную стоимость или длину.

Эта задача имеет множество практических приложений, включая построение оптимальных маршрутов для навигационных систем, оптимизацию транспортных и логистических сетей, а также в области обработки изображений, анализа данных и машинного обучения.

# Основные алгоритмы поиска путей в графе

## Алгоритм Дейкстры (Dijkstra's Algorithm)

* + Описание: Алгоритм Дейкстры является классическим методом для поиска кратчайшего пути от одной вершины к любой другой в графе с неотрицательными весами рёбер. Он работает путём пошагового расширения кратчайших путей от начальной вершины ко всем остальным.
  + Преимущества: Простота реализации, эффективность на графах с небольшим количеством вершин и положительными весами рёбер.
  + Недостатки: Не работает на графах с отрицательными весами рёбер и не является оптимальным для больших и плотных графов.

## Алгоритм Беллмана-Форда (Bellman-Ford Algorithm)

* + Описание: Алгоритм Беллмана-Форда используется для поиска кратчайших путей в графе даже с отрицательными весами рёбер. Он обновляет оценки расстояний до вершин итеративно, проверяя наличие возможных улучшений.
  + Преимущества: Работает на графах с отрицательными весами рёбер, способен обнаруживать отрицательные циклы.
  + Недостатки: Является более ресурсоемким из-за необходимости выполнения множества итераций.

## Алгоритм A (A-star Algorithm)

* + Описание: Алгоритм A\* является эвристическим методом, который комбинирует информацию о стоимости прохождения от начальной вершины до текущей с оценкой стоимости до конечной вершины. Он использует эвристику для принятия решения о выборе следующей вершины для раскрытия.
  + Преимущества: Эффективен на графах с большим количеством вершин и эвристическими оценками, позволяет эффективно находить кратчайшие пути в сложных графах.
  + Недостатки: Не гарантирует нахождение оптимального пути в некоторых случаях, также требует выбора подходящей эвристики для достижения хороших результатов.

## Алгоритм поиска в ширину (BFS - Breadth-First Search)

* + Описание: BFS является одним из простейших алгоритмов поиска пути в графе. Он начинает с определенной начальной вершины и исследует все ближайшие (смежные) вершины перед тем, как двигаться дальше. Он исследует граф уровень за уровнем, начиная с вершины, затем переходит к вершинам, которые находятся на расстоянии одного ребра от начальной, затем к вершинам, находящимся на расстоянии двух рёбер, и так далее.
  + Преимущества: Простота реализации, гарантия нахождения кратчайшего пути на невзвешенных графах без циклов.
  + Недостатки: Неэффективен на графах с большим количеством вершин и рёбер, не обрабатывает взвешенные графы, не является оптимальным для нахождения кратчайших путей во всех ситуациях.

Каждый из этих алгоритмов имеет свои уникальные характеристики и области применения, и выбор конкретного метода зависит от требований задачи и характеристик графа.

В контексте поставленной задачи подходят лишь BFS и алгоритм Дейкстры (в нашем случае длина каждого ребра будет равна 1). A\* - эвристический алгоритм, а Алгоритм Беллмана-Форда выполняет лишние вычисления (он находит все пути от исходной вершины до остальных).

В задаче, поставленной на практике, требуется находить пути последовательно, поэтому алгоритмы находящие пути скопом, а также алгоритмы минимизирующую среднюю длину не рассматривались.

# Хранение графа в памяти компьютера

Существует несколько основных способов хранения графа, каждый из которых имеет свои преимущества и недостатки. Вот некоторые из них:

## Списки смежности (Adjacency Lists)

* + Описание: Каждая вершина графа хранит список смежных с ней вершин (т.е. вершин, с которыми она соединена ребрами). Этот способ хранения часто используется для разреженных графов, где количество рёбер меньше количества вершин.
  + Преимущества: Экономичное использование памяти для разреженных графов, эффективность при обходе всех смежных вершин для данной вершины.
  + Недостатки: Неэффективен для поиска смежных вершин с известным идентификатором, требует больше времени на поиск рёбер между двумя вершинами.

## Матрица смежности (Adjacency Matrix)

* + Описание: Граф представляется в виде квадратной матрицы, где значение элемента (i, j) указывает на наличие ребра между вершинами i и j. Если ребро присутствует, то значение соответствующего элемента может быть весом ребра или просто флагом.
  + Преимущества: Эффективен для поиска прямого наличия ребра между двумя вершинами, хранит информацию обо всех рёбрах в графе.
  + Недостатки: Требует больше памяти, особенно для больших и разреженных графов, неэффективен при работе с разреженными графами.

## Список рёбер (Edge List)

* + Описание: Граф представляется в виде списка рёбер, где каждое ребро представлено парой вершин, которые оно соединяет, и возможно, весом или другими атрибутами ребра.
  + Преимущества: Экономит память по сравнению с матрицей смежности, подходит для разреженных графов, легко представляет информацию о рёбрах.
  + Недостатки: Медленный доступ к смежным вершинам, особенно в больших графах, требует дополнительного времени для поиска конкретного ребра.

Каждый из этих методов подходит для определенных типов графов и задач, и выбор конкретного способа зависит от конкретных требований и характеристик графа.

Список ребер не подходит не для какого их рассмотренных алгоритмов, поэтому не будем его реализовывать.

# Обзор использованных технологий

Кроме стандартной библиотеки С++17 были использованы некоторые сторонние библиотеки.

## Google Benchmark

Обладает следующими преимуществами:

1. Надежность: Google Benchmark разработан и поддерживается инженерами Google, что гарантирует его надежность и актуальность. Он широко используется в индустрии и имеет множество пользователей, что свидетельствует о его стабильности.
2. Простота использования: Google Benchmark предоставляет простой и интуитивно понятный интерфейс для написания бенчмарков. Он позволяет легко измерять производительность кода, создавать тесты и анализировать результаты.
3. Большие возможности: Библиотека обладает множеством возможностей для настройки и управления бенчмарками, таких как управление числом повторений, измерение времени выполнения с высокой точностью и анализ результатов.
4. Кроссплатформенность: Google Benchmark поддерживает работу на различных операционных системах, что делает его удобным инструментом для бенчмарков на различных платформах.
5. Активное сообщество: Существует активное сообщество пользователей Google Benchmark, где можно получить поддержку, задать вопросы и обменяться опытом в использовании библиотеки.

## Google Test

Использование Google Test (GTest) для написания и запуска тестов имеет ряд преимуществ, вот некоторые из них:

1. Простота написания тестов: GTest предоставляет простой и интуитивно понятный синтаксис для написания тестов, что делает процесс тестирования быстрым и эффективным.
2. Мощные возможности проверки: Библиотека предоставляет широкий набор возможностей для проверки функций и классов, включая различные типы утверждений (assertions), сравнение значений, проверку исключений и многое другое.
3. Гибкость в организации тестов: GTest поддерживает организацию тестов в тестовые кейсы и тестовые наборы, что позволяет легко управлять и запускать большое количество тестов.
4. Интеграция с CI/CD системами: GTest легко интегрируется с системами непрерывной интеграции и непрерывной доставки (CI/CD), такими как Jenkins, Travis CI, GitLab CI и другими, что позволяет автоматизировать процесс тестирования.
5. Поддержка множества платформ: GTest поддерживает различные операционные системы и компиляторы, что делает его универсальным инструментом для написания и запуска тестов на различных платформах.
6. Активное сообщество: Библиотека имеет активное сообщество пользователей и разработчиков, что обеспечивает поддержку, обновления и расширение функциональности в соответствии с требованиями пользователей.

В качестве альтернативы была рассмотрена библиотека CppUTest, но для меня избыточна и Google Test, и CppUTest. Поэтому я выбрал более распространенную Google Test.

## Visual C++ Package Manager

Vcpkg (Visual C++ Package Manager) - это инструмент для управления библиотеками и зависимостями в проектах на языке C++. Вот несколько причин, почему я рассмотрел использование vcpkg:

1. Простота установки и использования: Vcpkg обеспечивает простой способ установки библиотек и их интеграции в проекты. Он предоставляет простую командную строку для установки и управления зависимостями.
2. Широкий выбор библиотек: Vcpkg предлагает широкий выбор библиотек и фреймворков, которые могут быть установлены и использованы в проектах. Это включает в себя популярные библиотеки, такие как Boost, OpenCV, TensorFlow и многие другие.
3. Поддержка кросс-платформенной разработки: Vcpkg поддерживает работу на различных операционных системах, включая Windows, Linux и macOS. Это делает его удобным инструментом для кросс-платформенной разработки.
4. Интеграция с Visual Studio: Vcpkg интегрируется с Visual Studio, что позволяет легко добавлять библиотеки в проекты, управлять зависимостями и обновлять их через интерфейс Visual Studio.
5. Активное сообщество и поддержка: Vcpkg имеет активное сообщество пользователей и разработчиков, что обеспечивает поддержку, обновления и расширение функциональности в соответствии с требованиями пользователей.

В качестве альтернативы был рассмотрен Conan 2.0, но по-настоящему его потенциал можно раскрыть только на больших проектах, поэтому был использован более простой и отзывчивый Vcpkg.

## Visual Studio 2022

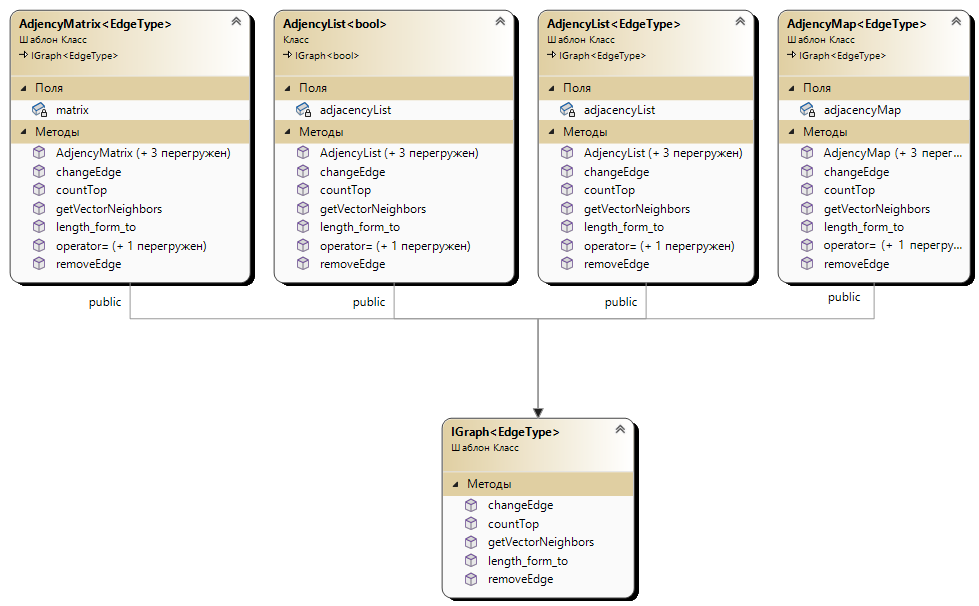
Вот некоторые из преимуществ Visual Studio:

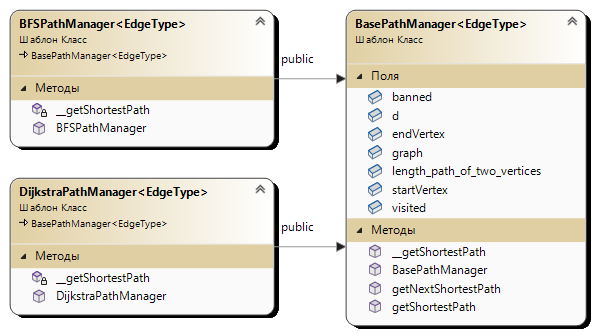
1. Мощный редактор кода: Visual Studio предоставляет мощный и удобный редактор кода с подсветкой синтаксиса, автозаполнением, функцией быстрой навигации и другими инструментами для повышения производительности разработчика.
2. Интеграция с различными языками и технологиями: Visual Studio поддерживает широкий спектр языков программирования, включая C++, C#, Visual Basic, F#, Python и многие другие. Он также интегрируется с различными технологиями и платформами, включая .NET Framework, .NET Core, ASP.NET, Azure, Xamarin и т. д.
3. Отладка: Visual Studio предоставляет мощные инструменты для отладки кода, включая возможность установки точек останова, просмотра переменных и стека вызовов, а также анализа производительности и памяти.
4. Широкий выбор инструментов и расширений: Visual Studio поставляется с широким набором инструментов для разработки, включая средства для работы с версионным контролем, создания интерфейсов пользователя, тестирования и т. д. Он также поддерживает множество расширений и плагинов от сторонних разработчиков.
5. Интеграция с другими продуктами и сервисами Microsoft: Visual Studio интегрируется с другими продуктами и сервисами Microsoft, такими как Azure DevOps, GitHub, Microsoft 365 и другими, что обеспечивает удобство разработки и совместной работы в команде.
6. Поддержка сообщества и обучение: Visual Studio имеет активное сообщество пользователей, обширную документацию, руководства, видеоуроки и онлайн-ресурсы, что делает его доступным и легко изучаемым для новичков и опытных разработчиков.

Кроме того, VS 2022 предоставляет свою систему сборки с очень удобным управлением в небольших проектах.

# Реализация алгоритмов и структур данных

## Отношения классов





Было принято решение делать статическую библиотеку в виду ее простоты.

Шаблон был реализован для того, чтобы сравнить затраты памяти и времени при хранении длины ребра в разных типах данных (bool и int)

Так же была реализована специализация AdjencyList<bool> для того, чтобы эффективней хранить невзвешенный граф при данном представлении длины ребра.

Реализация находится в приложении A.

## Генерация тестовых графов

Тестовые графы были сгенерированы с помощью библиотеки networkx из python.

Виды графов:

* Случайных граф, с количеством коэффициентом связи 25, 50 процентов
* Полносвязные небольшие графы размером 50, 100
* Граф сетка размерами 16x16, 32x32, 64x64

Код генерации находится в приложении B.

## Создание тестов

Листинг тестов находится в приложении D.

# Результаты бенчмарков

В дальнейшем используются срезы строк таблиц из приложения С. Таблицы имеют шапку:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Algorithm** | **Type** | **Graph** | **real\_time**  **(ms)** | **cpu\_time**  **(ms)** | **iterations** |

Обозначения колонке Graph:

* graph\_xxxx\_- кол-во вершин графа
* random – граф случайный. В случайных графах (с вероятностью, что вершины связаны) расстоянии между вершинами составляет 1-4 вершины.
* xxpersent\_prob – вероятность, что 2 вершины связаны
* grid\_graph\_xx – граф сетка (не больше 4-х соседов) xx на xx размера. Во всех тестах начальная и конечная вершина брались так, чтобы кратчайший путь был наибольшим
* graph\_50, graph\_100 – полносвязные графы размера 50 и 100 соответственно

Аргументация выводов была выполнения по таблице с bool хранением графов, так как в дальнейшем выяснилось, что такое хранение эффективнее.

Для чистоты экспериментов графы, исходные и конечные вершины фиксировались в коде бенчмарков.

Количество путей, найденных один за одним, равно 5 в каждом бенчмарке. Это сделано для того, чтобы можно было корректно сравнивать с графом сеткой, у которого максимум 4 пути между заданными вершинами (вместо 5-ого будет выдано значение, означающее, что путь не существует).

* При относительной близости вершин лучше всего использовать алгоритм bfs. Для иллюстрации этого несколько строк из бенчмарков:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Dijkstra | AdjMatrix | graph\_8192\_random\_50persent\_prob | 1943,54 | 1312,5 | 1 |
| **BFS** | **AdjMatrix** | **graph\_8192\_random\_50persent\_prob** | **2,26** | **1,38** | **1120** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Dijkstra | AdjList | graph\_2048\_random\_25persent\_prob | 357,06 | 195,31 | 4 |
| **BFS** | **AdjList** | **graph\_2048\_random\_25persent\_prob** | **0,21** | **0,12** | **7965** |

Это связано с тем, что алгоритм останавливается, когда рассматривает вершину назначения.

* При большой дальности вершин, как в случае в графе-стеке, лучше использовать алгоритм Дейкстры

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Dijkstra** | **AdjMatrix** | **grid\_graph\_64** | **86,7** | **52,08** | **15** |
| BFS | AdjMap | grid\_graph\_16 | 5580,81 | 3203,12 | 1 |

Сравнения на разных графах, потому что bfs не завершался за адекватное время на графах большого размера.

* При малом количестве ребер, лучшим образом показывает себя список смежности в качестве хранения графа

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Dijkstra | AdjMatrix | grid\_graph\_64 | 86,7 | 52,08 | 15 |
| **Dijkstra** | **AdjList** | **grid\_graph\_64** | **12,58** | **8,3** | **64** |
| Dijkstra | AdjMap | grid\_graph\_64 | 23,23 | 15,35 | 56 |

* Хранение смежных вершин в std::map (красно-черное дерево в основе) частично решает проблему медленного поиска связи между двумя вершинами в алгоритме дейкстры

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Dijkstra** | **AdjMap** | **graph\_2048\_random\_25persent\_prob** | **219,5** | **132,81** | **6** |
| Dijkstra | AdjList | graph\_2048\_random\_25persent\_prob | 357,06 | 195,31 | 4 |

* На малых графах эффективно хранить матрицу в матрице смежности. Специализация std::vector<bool> позволяет эффективно хранить данные в матрице.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Dijkstra | AdjList | graph\_50 | 0,15 | 0,09 | 11200 |
| **Dijkstra** | **AdjMatrix** | **graph\_50** | **0,07** | **0,05** | **11200** |
| Dijkstra | AdjMap | graph\_50 | 0,15 | 0,08 | 8960 |

* BFS показывает неприемлемые результаты в больших графах, где вершины далеко друг от друга (тесты считались очень долго, и было принято решения убрать их)
* В среднем хранения ребра в bool чуть быстрее, но занимает гораздо меньше памяти, поэтому нет смысла использовать int в качестве хранения ребер невзвешенных графах

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Dijkstra** | **AdjMatrix** | **graph\_8192\_random\_25persent\_prob** | **800,96** | **453,12** | **2** |
| Dijkstra | AdjMatrix | graph\_8192\_random\_25persent\_prob | 823,11 | 500 | 1 |

Результаты бенчмарков представлены в приложении C.

# Заключение

В процессе работы были рассмотрены основные алгоритмы, такие как алгоритм Дейкстры и алгоритм обхода в ширину (BFS). Алгоритмы имеют сильные и слабые стороны, исходя из этого нужно грамотно выбирать метод для поставленной задачи.

В ходе работы также были рассмотрены различные способы представления графа в памяти компьютера. Каждый из этих способов имеет свои особенности и подходит для определенных типов задач и характеристик графа.

# Приложения

## Приложение А. Листинг программы

using namespace std;

template<typename EdgeType>

BasePathManager<EdgeType>::BasePathManager(shared\_ptr<IGraph<EdgeType>> graph, int startVertex, int endVertex) :

graph(graph), startVertex(startVertex), endVertex(endVertex),

d(graph->countTop(), numeric\_limits<int>::max()),

visited(graph->countTop(), false),

banned(graph->countTop(), false) {}

template<typename EdgeType>

pair<int, vector<int>> DijkstraPathManager<EdgeType>::\_\_getShortestPath(int startVertex, int endVertex) {

const int SIZE = this->graph->countTop();

/// Тип для хранения расстояний и вершин

typedef std::pair<int, int> Pair; // <расстояние, вершина>

/// Создаем структуру для сравнения пар

struct ComparePair {

/// \brief Оператор сравнения пар.

///

/// \param a Первая пара.

/// \param b Вторая пара.

/// \return True, если расстояние первой пары больше расстояния второй пары, иначе false.

bool operator()(Pair const& a, Pair const& b) {

return a.first > b.first; // сравниваем по расстоянию

}

};

/// Инициализация кучи

std::priority\_queue<Pair, std::vector<Pair>, ComparePair> pq;

/// Заменяем инициализацию массива d на вставку пар в кучу

pq.push(std::make\_pair(0, startVertex)); // начальное расстояние до стартовой вершины равно 0

this->d.assign(this->graph->countTop(), numeric\_limits<int>::max());

this->d[startVertex] = 0;

/// Итерации алгоритма Дейкстры

int dist, u;

do {

if (pq.empty()) {

return { -1, {} };

}

Pair top = pq.top(); pq.pop();

dist = top.first;

u = top.second;

/// Проверка, посещали ли мы эту вершину

if (this->visited[u]) continue;

this->visited[u] = true;

/// Перебор соседей вершины u

for (int i : this->graph->getVectorNeighbors(u)) {

int v = i;

int weight = this->graph->length\_form\_to(u, i);

/// Обновление расстояния до вершины v

if (dist + weight < this->d[v]) {

this->d[v] = dist + weight;

pq.push(std::make\_pair(this->d[v], v)); // вставляем в кучу с новым расстоянием

}

}

} while (u != endVertex && dist != numeric\_limits<int>::max());

vector<int> shortestPath;

shortestPath.push\_back(endVertex);

int indexPrevVertex = 0;

int weight = this->d[endVertex];

int currentVertex = endVertex;

while (currentVertex != startVertex) {

for (int i = 0; i < SIZE; ++i) {

if (this->graph->length\_form\_to(i, currentVertex) > 0) {

int temp = weight - this->graph->length\_form\_to(i, currentVertex);

if (temp == this->d[i]) {

weight = temp;

currentVertex = i;

shortestPath.push\_back(i);

}

}

}

}

reverse(shortestPath.begin(), shortestPath.end());

return make\_pair(this->d[endVertex], shortestPath);

}

template<typename EdgeType>

DijkstraPathManager<EdgeType>::DijkstraPathManager(std::shared\_ptr<IGraph<EdgeType>> graph, int startVertex, int endVertex) :

BasePathManager<EdgeType>(graph, startVertex, endVertex){}

template<typename EdgeType>

pair<int, vector<int>> BasePathManager<EdgeType>::getShortestPath() {

visited.assign(graph->countTop(), false);

banned.assign(graph->countTop(), false);

length\_path\_of\_two\_vertices.second.clear();

auto [length, path] = \_\_getShortestPath(startVertex, endVertex);

for (auto bannedVertex : path) {

banned[bannedVertex] = true;

}

return { length, path };

}

template<typename EdgeType>

pair<int, vector<int>> BasePathManager<EdgeType>::getNextShortestPath() {

/// Начальные вершины должны быть в любом случае

visited = banned;

visited[startVertex] = false;

visited[endVertex] = false;

/// Обработка особого случая, когда путь состоити только из начальной и конечной вершины

if (!length\_path\_of\_two\_vertices.second.empty()) {

graph->removeEdge(length\_path\_of\_two\_vertices.second[0], length\_path\_of\_two\_vertices.second[1]);

}

auto [length, path] = \_\_getShortestPath(startVertex, endVertex);

if (path.size() == 2) {

length\_path\_of\_two\_vertices = { length, path };

}

/// Обработка особого случая, когда путь состоити только из начальной и конечной вершины

if (!length\_path\_of\_two\_vertices.second.empty()) {

graph->changeEdge(length\_path\_of\_two\_vertices.second[0], length\_path\_of\_two\_vertices.second[1], length\_path\_of\_two\_vertices.first);

}

for (auto bannedVertex : path) {

banned[bannedVertex] = true;

}

return { length, path };

}

template<typename EdgeType>

std::pair<int, std::vector<int>> BFSPathManager<EdgeType>::\_\_getShortestPath(int startVertex, int endVertex)

{

queue<int> q;

q.push(startVertex);

vector<int> prev(this->graph->countTop(), -1); // Вектор для хранения предыдущих вершин

bool is\_find = false;

while (!is\_find) {

if (q.empty()) {

return { -1, {} };

}

int current = q.front();

q.pop();

this->visited[current] = true;

if (current == endVertex) break;

for (int neighbor : this->graph->getVectorNeighbors(current)) {

if (!this->visited[neighbor]) {

q.push(neighbor);

prev[neighbor] = current; // Запоминаем предыдущую вершину

if (neighbor == endVertex) {

is\_find = true;

break;

}

}

}

}

vector<int> path;

int current = endVertex;

while (current != -1) { // Пока есть предыдущая вершина

path.push\_back(current);

current = prev[current];

}

reverse(path.begin(), path.end()); // Переворачиваем путь

return { path.size() - 1, path };

}

template<typename EdgeType>

BFSPathManager<EdgeType>::BFSPathManager(std::shared\_ptr<IGraph<EdgeType>> graph, int startVertex, int endVertex) :

BasePathManager<EdgeType>(graph, startVertex, endVertex) {};

template<typename EdgeType>

AdjencyMatrix<EdgeType>::AdjencyMatrix(std::string path)

{

std::ifstream file(path);

if (!file.is\_open()) {

throw std::runtime\_error("Failed to open file " + path);

}

int countTop, countEdges;

if (!(file >> countTop >> countEdges)) {

throw std::runtime\_error("Failed to read counts from file");

}

this->matrix = std::vector<std::vector<EdgeType>>(countTop, std::vector<EdgeType>(countTop, 0));

for (int i = 0; i < countEdges; ++i) {

int from, to;

double length;

if (!(file >> from >> to >> length)) {

throw std::runtime\_error("Failed to read edge from file");

}

if (from < 0 || from >= countTop || to < 0 || to >= countTop) {

throw std::runtime\_error("Invalid edge indices in file");

}

matrix[from][to] = static\_cast<EdgeType>(length);

}

file.close();

}

template<typename EdgeType>

AdjencyMatrix<EdgeType>::AdjencyMatrix(const AdjencyMatrix& other) : matrix(other.matrix) {}

template<typename EdgeType>

AdjencyMatrix<EdgeType>::AdjencyMatrix(AdjencyMatrix<EdgeType>&& other) noexcept

: matrix(std::move(other.matrix)){

other.~AdjencyMatrix();

}

template<typename EdgeType>

AdjencyMatrix<EdgeType>& AdjencyMatrix<EdgeType>::operator=(const AdjencyMatrix<EdgeType>& other) {

if (this != &other) {

AdjencyMatrix temp(other);

swap(temp.matrix, matrix);

}

return \*this;

}

template<typename EdgeType>

AdjencyMatrix<EdgeType>& AdjencyMatrix<EdgeType>::operator=(AdjencyMatrix<EdgeType>&& other) noexcept {

if (this != &other) {

AdjencyMatrix temp(std::move(other));

swap(temp.matrix, matrix);

}

return \*this;

}

template<typename EdgeType>

AdjencyMatrix<EdgeType>::AdjencyMatrix(int countTop) : matrix(vector<vector<EdgeType>>(countTop, vector<EdgeType>(countTop, 0))) {}

template<typename EdgeType>

EdgeType AdjencyMatrix<EdgeType>::length\_form\_to(int from, int to) const

{

if (from < 0 || from >= matrix.size()) {

throw std::runtime\_error("Vertex " + std::to\_string(from) + " is not exist");

}

if (to < 0 || to >= matrix.size()) {

throw std::runtime\_error("Vertex " + std::to\_string(to) + " is not exist");

}

return matrix[from][to];

}

template<typename EdgeType>

int AdjencyMatrix<EdgeType>::countTop() const

{

return matrix.size();

}

template<typename EdgeType>

void AdjencyMatrix<EdgeType>::changeEdge(int from, int to, EdgeType value)

{

if (from < 0 || from >= matrix.size()) {

throw std::runtime\_error("Vertex " + std::to\_string(from) + " is not exist");

}

if (to < 0 || to >= matrix.size()) {

throw std::runtime\_error("Vertex " + std::to\_string(to) + " is not exist");

}

matrix[from][to] = value;

}

template<typename EdgeType>

std::vector<int> AdjencyMatrix<EdgeType>::getVectorNeighbors(int vert)

{

std::vector<int> result;

result.reserve(400);

for (size\_t i = 0; i < matrix[vert].size(); ++i) {

if (matrix[vert][i] > 0) {

result.push\_back(i);

}

}

return result;

}

template<typename EdgeType>

AdjencyList<EdgeType>::AdjencyList(int vertices) : adjacencyList(vertices) {}

template<typename EdgeType>

int AdjencyList<EdgeType>::countTop() const {

return adjacencyList.size();

}

template<typename EdgeType>

EdgeType AdjencyList<EdgeType>::length\_form\_to(int from, int to) const {

if (from < 0 || from >= adjacencyList.size()) {

throw std::runtime\_error("Vertex " + std::to\_string(from) + " is not exist");

}

if (to < 0 || to >= adjacencyList.size()) {

throw std::runtime\_error("Vertex " + std::to\_string(to) + " is not exist");

}

for (const auto& neighbor : adjacencyList[from]) {

if (neighbor.first == to) {

return neighbor.second; // Возвращаем вес ребра

}

}

return 0; // Если между вершинами нет ребра

}

template<typename EdgeType>

void AdjencyList<EdgeType>::changeEdge(int from, int to, EdgeType value) {

if (from < 0 || from >= adjacencyList.size()) {

throw std::runtime\_error("Vertex " + std::to\_string(from) + " is not exist");

}

if (to < 0 || to >= adjacencyList.size()) {

throw std::runtime\_error("Vertex " + std::to\_string(to) + " is not exist");

}

for (auto& neighbor : adjacencyList[from]) {

if (neighbor.first == to) {

neighbor.second = value;

return;

}

}

adjacencyList[from].push\_back(std::make\_pair(to, value));

}

template<typename EdgeType>

std::vector<int> AdjencyList<EdgeType>::getVectorNeighbors(int vert) {

std::vector<int> neighbors;

neighbors.reserve(400);

for (const auto& neighbor : adjacencyList[vert]) {

neighbors.push\_back(neighbor.first);

}

return neighbors;

}

template<typename EdgeType>

AdjencyList<EdgeType>::AdjencyList(std::string path) {

std::ifstream file(path);

if (!file.is\_open()) {

throw std::runtime\_error("Failed to open file " + path);

}

int countTop, countEdges;

if (!(file >> countTop >> countEdges)) {

throw std::runtime\_error("Failed to read counts from file");

}

this->adjacencyList = std::vector<std::vector<std::pair<int, EdgeType>>>(countTop);

for (int i = 0; i < countEdges; ++i) {

int from, to, length;

if (!(file >> from >> to >> length)) {

throw std::runtime\_error("Failed to read edge from file");

}

if (from < 0 || from >= countTop || to < 0 || to >= countTop) {

throw std::runtime\_error("Invalid edge indices in file");

}

this->adjacencyList[from].push\_back({ to, length });

}

file.close();

}

template<typename EdgeType>

AdjencyList<EdgeType>::AdjencyList(const AdjencyList<EdgeType>& other) : adjacencyList(other.adjacencyList) {}

template<typename EdgeType>

AdjencyList<EdgeType>::AdjencyList(AdjencyList<EdgeType>&& other) noexcept : adjacencyList(std::move(other.adjacencyList)) {}

template<typename EdgeType>

AdjencyList<EdgeType>& AdjencyList<EdgeType>::operator=(const AdjencyList<EdgeType>& other) {

if (this != &other) {

AdjencyList temp(other);

std::swap(adjacencyList, temp.adjacencyList);

}

return \*this;

}

template<typename EdgeType>

AdjencyList<EdgeType>& AdjencyList<EdgeType>::operator=(AdjencyList<EdgeType>&& other) noexcept {

if (this != &other) {

adjacencyList = std::move(other.adjacencyList);

}

return \*this;

}

template<typename EdgeType>

void AdjencyList<EdgeType>::removeEdge(int from, int to) {

if (from < 0 || from >= adjacencyList.size()) {

throw std::runtime\_error("Vertex " + std::to\_string(from) + " is not exist");

}

if (to < 0 || to >= adjacencyList.size()) {

throw std::runtime\_error("Vertex " + std::to\_string(to) + " is not exist");

}

for (auto neighbor\_iter = adjacencyList[from].begin();

neighbor\_iter != adjacencyList[from].end(); neighbor\_iter++) {

if ((\*neighbor\_iter).first == to) {

adjacencyList[from].erase(neighbor\_iter);

return;

}

}

}

template<typename EdgeType>

void AdjencyMatrix<EdgeType>::removeEdge(int from, int to) {

if (from < 0 || from >= matrix.size()) {

throw std::runtime\_error("Vertex " + std::to\_string(from) + " is not exist");

}

if (to < 0 || to >= matrix.size()) {

throw std::runtime\_error("Vertex " + std::to\_string(to) + " is not exist");

}

matrix[from][to] = 0;

}

template<typename EdgeType>

AdjencyMap<EdgeType>::AdjencyMap(int vertices) : adjacencyMap(vertices){}

template<typename EdgeType>

int AdjencyMap<EdgeType>::countTop() const

{

return this->adjacencyMap.size();

}

template<typename EdgeType>

EdgeType AdjencyMap<EdgeType>::length\_form\_to(int from, int to) const

{

if (from < 0 || from >= adjacencyMap.size()) {

throw std::runtime\_error("Vertex " + std::to\_string(from) + " is not exist");

}

if (to < 0 || to >= adjacencyMap.size()) {

throw std::runtime\_error("Vertex " + std::to\_string(to) + " is not exist");

}

auto it = this->adjacencyMap[from].find(to);

if (it != this->adjacencyMap[from].end()) {

return it->second;

}

return 0; // Если между вершинами нет ребра

}

template<typename EdgeType>

void AdjencyMap<EdgeType>::changeEdge(int from, int to, EdgeType value)

{

if (from < 0 || from >= adjacencyMap.size()) {

throw std::runtime\_error("Vertex " + std::to\_string(from) + " is not exist");

}

if (to < 0 || to >= adjacencyMap.size()) {

throw std::runtime\_error("Vertex " + std::to\_string(to) + " is not exist");

}

this->adjacencyMap[from].insert({ to, value });

}

template<typename EdgeType>

std::vector<int> AdjencyMap<EdgeType>::getVectorNeighbors(int vert)

{

// Создание вектора для хранения ключей

std::vector<int> keys;

keys.reserve(this->adjacencyMap.size());

// Получение всех ключей из std::map и добавление их в вектор

for (const auto& pair : this->adjacencyMap[vert]) {

keys.push\_back(pair.first);

}

return keys;

}

template<typename EdgeType>

AdjencyMap<EdgeType>::AdjencyMap(std::string path)

{

std::ifstream file(path);

if (!file.is\_open()) {

throw std::runtime\_error("Failed to open file " + path);

}

int countTop, countEdges;

if (!(file >> countTop >> countEdges)) {

throw std::runtime\_error("Failed to read counts from file");

}

this->adjacencyMap = std::vector<std::map<int, EdgeType>>(countTop);

for (int i = 0; i < countEdges; ++i) {

int from, to, length;

if (!(file >> from >> to >> length)) {

throw std::runtime\_error("Failed to read edge from file");

}

if (from < 0 || from >= countTop || to < 0 || to >= countTop) {

throw std::runtime\_error("Invalid edge indices in file");

}

this->adjacencyMap[from][to] = length;

}

file.close();

}

template<typename EdgeType>

AdjencyMap<EdgeType>::AdjencyMap(const AdjencyMap<EdgeType>& other) : adjacencyMap(other.adjacencyMap) {}

template<typename EdgeType>

AdjencyMap<EdgeType>::AdjencyMap(AdjencyMap<EdgeType>&& other) noexcept : adjacencyMap(std::move(other.adjacencyMap)) {}

template<typename EdgeType>

AdjencyMap<EdgeType>& AdjencyMap<EdgeType>::operator=(const AdjencyMap<EdgeType>& other)

{

if (this != &other) {

AdjencyList temp(other);

std::swap(adjacencyMap, temp.adjacencyMap);

}

return \*this;

}

template<typename EdgeType>

AdjencyMap<EdgeType>& AdjencyMap<EdgeType>::operator=(AdjencyMap<EdgeType>&& other) noexcept

{

if (this != &other) {

adjacencyMap = std::move(other.adjacencyMap);

}

return \*this;

}

template<typename EdgeType>

void AdjencyMap<EdgeType>::removeEdge(int start, int end)

{

if (start < 0 || start >= adjacencyMap.size()) {

throw std::runtime\_error("Vertex " + std::to\_string(start) + " is not exist");

}

if (end < 0 || end >= adjacencyMap.size()) {

throw std::runtime\_error("Vertex " + std::to\_string(end) + " is not exist");

}

auto it = this->adjacencyMap[start].find(end);

if (it != this->adjacencyMap[start].end()) {

this->adjacencyMap[start].erase(it);

}

}

bool AdjencyList<bool>::length\_form\_to(int from, int to) const {

if (from < 0 || from >= adjacencyList.size()) {

throw std::runtime\_error("Vertex " + std::to\_string(from) + " is not exist");

}

if (to < 0 || to >= adjacencyList.size()) {

throw std::runtime\_error("Vertex " + std::to\_string(to) + " is not exist");

}

auto it = std::find(adjacencyList[from].begin(), adjacencyList[from].end(), to);

return it != adjacencyList[from].end(); // Если между вершинами нет ребра

}

void AdjencyList<bool>::changeEdge(int from, int to, bool value)

{

if (from < 0 || from >= adjacencyList.size()) {

throw std::runtime\_error("Vertex " + std::to\_string(from) + " is not exist");

}

if (to < 0 || to >= adjacencyList.size()) {

throw std::runtime\_error("Vertex " + std::to\_string(to) + " is not exist");

}

auto it = std::find(adjacencyList[from].begin(), adjacencyList[from].end(), to);

if (it != adjacencyList[from].end() && !value) {

adjacencyList[from].erase(it);

}

if (it == adjacencyList[from].end() && value) {

adjacencyList[from].push\_back(value);

}

}

std::vector<int> AdjencyList<bool>::getVectorNeighbors(int vert) {

return std::vector(adjacencyList[vert]);

}

AdjencyList<bool>::AdjencyList(std::string path) {

std::ifstream file(path);

if (!file.is\_open()) {

throw std::runtime\_error("Failed to open file " + path);

}

int countTop, countEdges;

if (!(file >> countTop >> countEdges)) {

throw std::runtime\_error("Failed to read counts from file");

}

this->adjacencyList = std::vector<std::vector<int>>(countTop);

for (int i = 0; i < countEdges; ++i) {

int from, to, length;

if (!(file >> from >> to >> length)) {

throw std::runtime\_error("Failed to read edge from file");

}

if (from < 0 || from >= countTop || to < 0 || to >= countTop) {

throw std::runtime\_error("Invalid edge indices in file");

}

this->adjacencyList[from].push\_back(to);

}

file.close();

}

void AdjencyList<bool>::removeEdge(int from, int to) {

if (from < 0 || from >= adjacencyList.size()) {

throw std::runtime\_error("Vertex " + std::to\_string(from) + " is not exist");

}

if (to < 0 || to >= adjacencyList.size()) {

throw std::runtime\_error("Vertex " + std::to\_string(to) + " is not exist");

}

auto it = std::find(adjacencyList[from].begin(), adjacencyList[from].end(), to);

if (it != adjacencyList[from].end()) {

adjacencyList[from].erase(it);

}

}

AdjencyList<bool>::AdjencyList(int vertices) : adjacencyList(vertices) {}

int AdjencyList<bool>::countTop() const {

return adjacencyList.size();

}

AdjencyList<bool>::AdjencyList(const AdjencyList<bool>& other) :

adjacencyList(other.adjacencyList) {}

AdjencyList<bool>::AdjencyList(AdjencyList<bool>&& other) noexcept :

adjacencyList(std::move(other.adjacencyList)) {}

AdjencyList<bool>& AdjencyList<bool>::operator=(const AdjencyList<bool>& other) {

if (this != &other) {

AdjencyList temp(other);

std::swap(adjacencyList, temp.adjacencyList);

}

return \*this;

}

AdjencyList<bool>& AdjencyList<bool>::operator=(AdjencyList<bool>&& other) noexcept {

if (this != &other) {

adjacencyList = std::move(other.adjacencyList);

}

return \*this;

}

## Приложение B. Генерация графов

# %%

import networkx as nx

# %%

# Генерация случайного графа

def generate\_random\_graph(num\_nodes, probability):

    return nx.erdos\_renyi\_graph(num\_nodes, probability, directed=True)

# Генерация графа с заданным числом узлов и процентом соединений

def generate\_connected\_graph(num\_nodes, percent\_edges):

    num\_edges = int((num\_nodes \* (num\_nodes - 1) / 2) \* (percent\_edges / 100))

    return nx.gnm\_random\_graph(num\_nodes, num\_edges, directed=True)

def generate\_grid\_graph(rows, cols):

    G = nx.grid\_2d\_graph(rows, cols)

    return nx.DiGraph(G)

# Генерация графа в виде цикла

def generate\_cycle\_graph(num\_nodes):

    return nx.cycle\_graph(num\_nodes, create\_using=nx.DiGraph)

def generate\_random\_tree(num\_nodes):

    # Генерируем случайное дерево

    G = nx.random\_tree(num\_nodes)

    # Преобразуем дерево в направленный граф

    directed\_G = nx.DiGraph()

    directed\_G.add\_nodes\_from(G.nodes())

    directed\_G.add\_edges\_from(G.edges())

    return directed\_G

def write\_graph\_to\_file(graph, filename):

    with open(filename, 'w') as file:

        num\_nodes = graph.number\_of\_nodes()

        num\_edges = graph.number\_of\_edges()

        file.write(f"{num\_nodes} {num\_edges}\n")  # Записываем количество вершин и рёбер

        for edge in graph.edges():

            file.write(f"{edge[0]} {edge[1]} 1\n")

# %%

graph8192\_random\_25persent\_prob = generate\_random\_graph(8192, 0.25)

write\_graph\_to\_file(graph=graph8192\_random\_25persent\_prob, filename="graph8192\_random\_25persent\_prob.txt")

graph8192\_random\_50persent\_prob = generate\_random\_graph(8192, 0.50)

write\_graph\_to\_file(graph=graph8192\_random\_50persent\_prob, filename="graph8192\_random\_50persent\_prob.txt")

graph8192\_random\_100persent\_prob = generate\_random\_graph(8192, 1.00)

write\_graph\_to\_file(graph=graph8192\_random\_100persent\_prob, filename="graph8192\_random\_100persent\_prob.txt")

del graph8192\_random\_50persent\_prob,graph8192\_random\_25persent\_prob,graph8192\_random\_100persent\_prob

# %%

graph4096\_random\_25persent\_prob = generate\_random\_graph(4096, 0.25)

write\_graph\_to\_file(graph=graph4096\_random\_25persent\_prob, filename="graph4096\_random\_25persent\_prob.txt")

graph4096\_random\_50persent\_prob = generate\_random\_graph(4096, 0.50)

write\_graph\_to\_file(graph=graph4096\_random\_50persent\_prob, filename="graph4096\_random\_50persent\_prob.txt")

graph4096\_random\_100persent\_prob = generate\_random\_graph(4096, 1.00)

write\_graph\_to\_file(graph=graph4096\_random\_100persent\_prob, filename="graph4096\_random\_100persent\_prob.txt")

del graph4096\_random\_25persent\_prob, graph4096\_random\_50persent\_prob, graph4096\_random\_100persent\_prob

# %%

graph2048\_random\_25persent\_prob = generate\_random\_graph(2048, 0.25)

write\_graph\_to\_file(graph=graph2048\_random\_25persent\_prob, filename="graph2048\_random\_25persent\_prob.txt")

graph2048\_random\_50persent\_prob = generate\_random\_graph(2048, 0.50)

write\_graph\_to\_file(graph=graph2048\_random\_50persent\_prob, filename="graph2048\_random\_50persent\_prob.txt")

graph2048\_random\_100persent\_prob = generate\_random\_graph(2048, 1.00)

write\_graph\_to\_file(graph=graph2048\_random\_100persent\_prob, filename="graph2048\_random\_100persent\_prob.txt")

del graph2048\_random\_25persent\_prob, graph2048\_random\_50persent\_prob, graph2048\_random\_100persent\_prob

# %%

def write\_graph\_grid\_to\_file(graph, filename, num\_edge):

    with open(filename, 'w') as file:

        num\_nodes = graph.number\_of\_nodes()

        num\_edges = graph.number\_of\_edges()

        file.write(f"{num\_nodes} {num\_edges}\n")  # Записываем количество вершин и рёбер

        for edge in graph.edges():

            file.write(f"{edge[0][0] \* num\_edge + edge[0][1]} {edge[1][0] \* num\_edge + edge[1][1]} 1\n")  # Учитываем направление ребра

# %%

grid\_graph16 = generate\_grid\_graph(16, 16)

write\_graph\_grid\_to\_file(graph=grid\_graph16, filename="grid\_graph16.txt", num\_edge=16)

grid\_graph32 = generate\_grid\_graph(32, 32)

write\_graph\_grid\_to\_file(graph=grid\_graph32, filename="grid\_graph32.txt", num\_edge=32)

grid\_graph64 = generate\_grid\_graph(64, 64)

write\_graph\_grid\_to\_file(graph=grid\_graph64, filename="grid\_graph64.txt", num\_edge=64)

del grid\_graph16, grid\_graph32, grid\_graph64

# %%

tree\_graph8192 = generate\_random\_tree(8192)

write\_graph\_to\_file(graph=tree\_graph8192, filename="tree\_graph8192.txt")

tree\_graph2048= generate\_random\_tree(2048)

write\_graph\_to\_file(graph=tree\_graph2048, filename="tree\_graph2048.txt")

tree\_graph4096 = generate\_random\_tree(4096)

write\_graph\_to\_file(graph=tree\_graph4096, filename="grid\_graph4096.txt")

del tree\_graph8192, tree\_graph2048, tree\_graph4096

# %%

graph100 = generate\_random\_graph(100, 1)

write\_graph\_to\_file(graph=graph100, filename="graph100.txt")

# %%

graph50 = generate\_random\_graph(50, 1)

write\_graph\_to\_file(graph=graph50, filename="graph50.txt")

## Приложение C. Бенчмарки

//int\_srorage

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Algorithm** | **Type** | **Graph** | **real\_time**  **(ms)** | **cpu\_time**  **(ms)** | **iterations** |
| Dijkstra | AdjMatrix | graph\_2048\_random\_25persent\_prob | 67,12 | 39,38 | 25 |
| Dijkstra | AdjMatrix | graph\_4096\_random\_25persent\_prob | 360,09 | 138,39 | 7 |
| Dijkstra | AdjMatrix | graph\_8192\_random\_25persent\_prob | 823,11 | 500 | 1 |
| Dijkstra | AdjMatrix | graph\_2048\_random\_50persent\_prob | 137,78 | 104,17 | 6 |
| Dijkstra | AdjMatrix | graph\_4096\_random\_50persent\_prob | 433,06 | 269,53 | 4 |
| Dijkstra | AdjMatrix | graph\_8192\_random\_50persent\_prob | 1991,91 | 812,5 | 1 |
| Dijkstra | AdjMatrix | grid\_graph\_16 | 0,51 | 0,29 | 2489 |
| Dijkstra | AdjMatrix | grid\_graph\_32 | 7,31 | 3,79 | 231 |
| Dijkstra | AdjMatrix | grid\_graph\_64 | 108,9 | 48,08 | 13 |
| Dijkstra | AdjMatrix | graph\_100 | 0,2 | 0,11 | 7467 |
| Dijkstra | AdjMatrix | graph\_50 | 0,07 | 0,04 | 32000 |
| Dijkstra | AdjList | graph\_2048\_random\_25persent\_prob | 358,69 | 208,33 | 3 |
| Dijkstra | AdjList | graph\_4096\_random\_25persent\_prob | 3235,57 | 1671,88 | 1 |
| Dijkstra | AdjList | graph\_8192\_random\_25persent\_prob | 14446,53 | 8625 | 1 |
| Dijkstra | AdjList | graph\_2048\_random\_50persent\_prob | 1604,7 | 843,75 | 1 |
| Dijkstra | AdjList | graph\_4096\_random\_50persent\_prob | 9111,33 | 4484,38 | 1 |
| Dijkstra | AdjList | graph\_8192\_random\_50persent\_prob | 82515,92 | 35093,75 | 1 |
| Dijkstra | AdjList | grid\_graph\_16 | 0,29 | 0,09 | 10000 |
| Dijkstra | AdjList | grid\_graph\_32 | 1,72 | 0,87 | 896 |
| Dijkstra | AdjList | grid\_graph\_64 | 12,58 | 9,03 | 90 |
| Dijkstra | AdjList | graph\_100 | 0,99 | 0,4 | 1659 |
| Dijkstra | AdjList | graph\_50 | 0,14 | 0,06 | 10000 |
| BFS | AdjMatrix | graph\_2048\_random\_25persent\_prob | 0,35 | 0,16 | 6292 |
| BFS | AdjMatrix | graph\_4096\_random\_25persent\_prob | 0,99 | 0,34 | 1792 |
| BFS | AdjMatrix | graph\_8192\_random\_25persent\_prob | 1,01 | 0,5 | 1384 |
| BFS | AdjMatrix | graph\_2048\_random\_50persent\_prob | 0,55 | 0,29 | 2987 |
| BFS | AdjMatrix | graph\_4096\_random\_50persent\_prob | 0,75 | 0,31 | 2133 |
| BFS | AdjMatrix | graph\_8192\_random\_50persent\_prob | 2,35 | 0,98 | 896 |
| BFS | AdjMatrix | grid\_graph\_16 | 18370,68 | 8859,38 | 1 |
| BFS | AdjMatrix | graph\_100 | 0,02 | 0,01 | 112000 |
| BFS | AdjMatrix | graph\_50 | 0,01 | 0,01 | 100000 |
| BFS | AdjList | graph\_2048\_random\_25persent\_prob | 0,21 | 0,08 | 8960 |
| BFS | AdjList | graph\_4096\_random\_25persent\_prob | 0,63 | 0,22 | 2635 |
| BFS | AdjList | graph\_8192\_random\_25persent\_prob | 0,59 | 0,27 | 3200 |
| BFS | AdjList | graph\_2048\_random\_50persent\_prob | 0,39 | 0,18 | 3446 |
| BFS | AdjList | graph\_4096\_random\_50persent\_prob | 0,47 | 0,3 | 2635 |
| BFS | AdjList | graph\_8192\_random\_50persent\_prob | 1,41 | 0,71 | 896 |
| BFS | AdjList | grid\_graph\_16 | 5299,47 | 3359,38 | 1 |
| BFS | AdjList | graph\_100 | 0,02 | 0,01 | 49778 |
| BFS | AdjList | graph\_50 | 0,01 | 0,01 | 100000 |
| BFS | AdjMap | graph\_2048\_random\_25persent\_prob | 0,27 | 0,14 | 4978 |
| BFS | AdjMap | graph\_4096\_random\_25persent\_prob | 0,73 | 0,45 | 1730 |
| BFS | AdjMap | graph\_8192\_random\_25persent\_prob | 0,85 | 0,5 | 1673 |
| BFS | AdjMap | graph\_2048\_random\_50persent\_prob | 0,46 | 0,26 | 3200 |
| BFS | AdjMap | graph\_4096\_random\_50persent\_prob | 0,61 | 0,39 | 2489 |
| BFS | AdjMap | graph\_8192\_random\_50persent\_prob | 2,16 | 1,17 | 896 |
| BFS | AdjMap | grid\_graph\_16 | 5681,16 | 2921,88 | 1 |
| BFS | AdjMap | graph\_100 | 0,02 | 0,01 | 49778 |
| BFS | AdjMap | graph\_50 | 0,01 | 0,01 | 100000 |
| Dijkstra | AdjMap | graph\_100 | 0,56 | 0,23 | 2133 |
| Dijkstra | AdjMap | graph\_50 | 0,15 | 0,08 | 8960 |
| Dijkstra | AdjMap | graph\_2048\_random\_25persent\_prob | 218,84 | 78,12 | 7 |
| Dijkstra | AdjMap | graph\_4096\_random\_25persent\_prob | 1163,13 | 500 | 1 |
| Dijkstra | AdjMap | graph\_8192\_random\_25persent\_prob | 3104,19 | 1578,12 | 1 |
| Dijkstra | AdjMap | graph\_2048\_random\_50persent\_prob | 570,41 | 307,29 | 3 |
| Dijkstra | AdjMap | graph\_4096\_random\_50persent\_prob | 1896,84 | 609,38 | 1 |
| Dijkstra | AdjMap | graph\_8192\_random\_50persent\_prob | 9206,97 | 4796,88 | 1 |
| Dijkstra | AdjMap | grid\_graph\_16 | 0,33 | 0,16 | 3446 |
| Dijkstra | AdjMap | grid\_graph\_32 | 2,37 | 1,3 | 1000 |
| Dijkstra | AdjMap | grid\_graph\_64 | 24,72 | 10,18 | 112 |

//bool\_storage

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Algorithm** | **Type** | **Graph** | **real\_time** | **cpu\_time** | **iterations** |
| Dijkstra | AdjMatrix | graph\_2048\_random\_25persent\_prob | 67,92 | 19,23 | 26 |
| Dijkstra | AdjMatrix | graph\_4096\_random\_25persent\_prob | 329,62 | 187,5 | 4 |
| Dijkstra | AdjMatrix | graph\_8192\_random\_25persent\_prob | 800,96 | 453,12 | 2 |
| Dijkstra | AdjMatrix | graph\_2048\_random\_50persent\_prob | 136,58 | 82,59 | 7 |
| Dijkstra | AdjMatrix | graph\_4096\_random\_50persent\_prob | 427,85 | 234,38 | 3 |
| Dijkstra | AdjMatrix | graph\_8192\_random\_50persent\_prob | 1943,54 | 1312,5 | 1 |
| Dijkstra | AdjMatrix | grid\_graph\_16 | 0,48 | 0,27 | 2133 |
| Dijkstra | AdjMatrix | grid\_graph\_32 | 5,75 | 3,32 | 193 |
| Dijkstra | AdjMatrix | grid\_graph\_64 | 86,7 | 52,08 | 15 |
| Dijkstra | AdjMatrix | graph\_100 | 0,18 | 0,11 | 5600 |
| Dijkstra | AdjMatrix | graph\_50 | 0,07 | 0,05 | 11200 |
| Dijkstra | AdjList | graph\_2048\_random\_25persent\_prob | 357,06 | 195,31 | 4 |
| Dijkstra | AdjList | graph\_4096\_random\_25persent\_prob | 3169,21 | 2062,5 | 1 |
| Dijkstra | AdjList | graph\_8192\_random\_25persent\_prob | 14308,58 | 8968,75 | 1 |
| Dijkstra | AdjList | graph\_2048\_random\_50persent\_prob | 1564,84 | 890,62 | 1 |
| Dijkstra | AdjList | graph\_4096\_random\_50persent\_prob | 9010,77 | 5828,12 | 1 |
| Dijkstra | AdjList | graph\_8192\_random\_50persent\_prob | 81793,29 | 47953,12 | 1 |
| Dijkstra | AdjList | grid\_graph\_16 | 0,27 | 0,16 | 4073 |
| Dijkstra | AdjList | grid\_graph\_32 | 1,74 | 1,06 | 1120 |
| Dijkstra | AdjList | grid\_graph\_64 | 12,58 | 8,3 | 64 |
| Dijkstra | AdjList | graph\_100 | 0,97 | 0,62 | 1000 |
| Dijkstra | AdjList | graph\_50 | 0,15 | 0,09 | 11200 |
| BFS | AdjMatrix | graph\_2048\_random\_25persent\_prob | 0,34 | 0,2 | 3200 |
| BFS | AdjMatrix | graph\_4096\_random\_25persent\_prob | 0,97 | 0,64 | 1116 |
| BFS | AdjMatrix | graph\_8192\_random\_25persent\_prob | 1 | 0,57 | 1120 |
| BFS | AdjMatrix | graph\_2048\_random\_50persent\_prob | 0,56 | 0,33 | 2133 |
| BFS | AdjMatrix | graph\_4096\_random\_50persent\_prob | 0,75 | 0,44 | 1792 |
| BFS | AdjMatrix | graph\_8192\_random\_50persent\_prob | 2,26 | 1,38 | 1120 |
| BFS | AdjMatrix | grid\_graph\_16 | 20022,03 | 12000 | 1 |
| BFS | AdjMatrix | graph\_100 | 0,02 | 0,01 | 56000 |
| BFS | AdjMatrix | graph\_50 | 0,01 | 0,01 | 100000 |
| BFS | AdjList | graph\_2048\_random\_25persent\_prob | 0,21 | 0,12 | 7965 |
| BFS | AdjList | graph\_4096\_random\_25persent\_prob | 0,61 | 0,41 | 2358 |
| BFS | AdjList | graph\_8192\_random\_25persent\_prob | 0,6 | 0,37 | 2036 |
| BFS | AdjList | graph\_2048\_random\_50persent\_prob | 0,38 | 0,25 | 3733 |
| BFS | AdjList | graph\_4096\_random\_50persent\_prob | 0,47 | 0,28 | 2358 |
| BFS | AdjList | graph\_8192\_random\_50persent\_prob | 1,46 | 0,95 | 640 |
| BFS | AdjList | grid\_graph\_16 | 5362,75 | 3000 | 1 |
| BFS | AdjList | graph\_100 | 0,02 | 0,01 | 74667 |
| BFS | AdjList | graph\_50 | 0,01 | 0,01 | 100000 |
| BFS | AdjMap | graph\_2048\_random\_25persent\_prob | 0,25 | 0,16 | 6432 |
| BFS | AdjMap | graph\_4096\_random\_25persent\_prob | 0,73 | 0,48 | 1867 |
| BFS | AdjMap | graph\_8192\_random\_25persent\_prob | 0,81 | 0,59 | 1000 |
| BFS | AdjMap | graph\_2048\_random\_50persent\_prob | 0,46 | 0,31 | 2987 |
| BFS | AdjMap | graph\_4096\_random\_50persent\_prob | 0,62 | 0,43 | 2635 |
| BFS | AdjMap | graph\_8192\_random\_50persent\_prob | 2,51 | 1,13 | 747 |
| BFS | AdjMap | grid\_graph\_16 | 5580,81 | 3203,12 | 1 |
| BFS | AdjMap | graph\_100 | 0,02 | 0,01 | 77195 |
| BFS | AdjMap | graph\_50 | 0,01 | 0 | 100000 |
| Dijkstra | AdjMap | graph\_100 | 0,58 | 0,36 | 2635 |
| Dijkstra | AdjMap | graph\_50 | 0,16 | 0,08 | 10000 |
| Dijkstra | AdjMap | graph\_2048\_random\_25persent\_prob | 219,5 | 132,81 | 6 |
| Dijkstra | AdjMap | graph\_4096\_random\_25persent\_prob | 1152,88 | 687,5 | 1 |
| Dijkstra | AdjMap | graph\_8192\_random\_25persent\_prob | 3093,92 | 1875 | 1 |
| Dijkstra | AdjMap | graph\_2048\_random\_50persent\_prob | 580,32 | 312,5 | 2 |
| Dijkstra | AdjMap | graph\_4096\_random\_50persent\_prob | 1869,35 | 968,75 | 1 |
| Dijkstra | AdjMap | graph\_8192\_random\_50persent\_prob | 9274,94 | 5093,75 | 1 |
| Dijkstra | AdjMap | grid\_graph\_16 | 0,32 | 0,18 | 3446 |
| Dijkstra | AdjMap | grid\_graph\_32 | 2,4 | 1,33 | 448 |
| Dijkstra | AdjMap | grid\_graph\_64 | 23,23 | 15,35 | 56 |

## Приложение D. Тесты

// int storage

#include <Benchmark/benchmark.h>

#include <memory>

#include <iostream>

#include <string>

#include "DijkstraPathManager.h"

#include "DijkstraPathManager.cpp"

#include "Graph.h"

#include "Graph.cpp"

using namespace std;

#define STRINGIFY(x) #x

#define EXPAND(x) STRINGIFY(x)

string projectDir;

string getProjectDir() {

if (!projectDir.empty())

return projectDir;

string s = EXPAND(UNITTESTPRJ);

s.erase(0, 1); // erase the first quote

s.erase(s.size() - 2); // erase the last quote and the dot

projectDir = s;

return projectDir;

}

string solutionDir;

string getSolutionDir() {

if (!solutionDir.empty())

return solutionDir;

string s = EXPAND(UNITTESTPRJ);

s.erase(0, 1); // erase the first quote

s.erase(s.size() - 2); // erase the last quote and the dot

// Находим позицию последнего символа '\'

size\_t found = s.find\_last\_of('\\');

// Если найден символ '\', обрезаем строку по этой позиции

if (found != std::string::npos) {

s = s.substr(0, found);

}

solutionDir = s;

return solutionDir;

}

static void BM\_Dijkstra\_AdjMatrix(benchmark::State& state, const std::string& graphFileName, int from, int to) {

AdjencyMatrix<int> matrix(getSolutionDir() + "\\" + graphFileName);

DijkstraPathManager<int> manager(std::make\_shared<AdjencyMatrix<int>>(matrix), from, to);

auto [length, path] = manager.getShortestPath();

for (auto \_ : state) {

for (int i = 0; i < 4; i++) {

auto res = manager.getNextShortestPath();

benchmark::DoNotOptimize(res);

if (res.first == -1) {

std::cout << "Path is not exist!\n";

}

}

benchmark::DoNotOptimize(manager.getShortestPath());

}

}

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMatrix, graph\_2048\_random\_25persent\_prob, "../Graphs/graph2048\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMatrix, graph\_4096\_random\_25persent\_prob, "../Graphs/graph4096\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMatrix, graph\_8192\_random\_25persent\_prob, "../Graphs/graph8192\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMatrix, graph\_2048\_random\_50persent\_prob, "../Graphs/graph2048\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMatrix, graph\_4096\_random\_50persent\_prob, "../Graphs/graph4096\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMatrix, graph\_8192\_random\_50persent\_prob, "../Graphs/graph8192\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMatrix, grid\_graph\_16, "../Graphs/grid\_graph16.txt", 17, 254)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMatrix, grid\_graph\_32, "../Graphs/grid\_graph32.txt", 33, 1022)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMatrix, grid\_graph\_64, "../Graphs/grid\_graph64.txt", 160, 4094)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMatrix, graph\_100, "../Graphs/graph100.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMatrix, graph\_50, "../Graphs/graph50.txt", 0, 30)->Unit(benchmark::kMillisecond);

static void BM\_Dijkstra\_AdjList(benchmark::State& state, const std::string& graphFileName, int from, int to) {

AdjencyList<int> matrix(getSolutionDir() + "\\" + graphFileName);

DijkstraPathManager<int> manager(make\_shared<AdjencyList<int>>(matrix), from, to);

auto [length, path] = manager.getShortestPath();

for (auto \_ : state) {

for (int i = 0; i < 4; i++) {

auto res = manager.getNextShortestPath();

benchmark::DoNotOptimize(res);

if (res.first == -1) {

std::cout << "Path is not exist!\n";

}

}

benchmark::DoNotOptimize(manager.getShortestPath());

}

}

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjList, graph\_2048\_random\_25persent\_prob, "../Graphs/graph2048\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjList, graph\_4096\_random\_25persent\_prob, "../Graphs/graph4096\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjList, graph\_8192\_random\_25persent\_prob, "../Graphs/graph8192\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjList, graph\_2048\_random\_50persent\_prob, "../Graphs/graph2048\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjList, graph\_4096\_random\_50persent\_prob, "../Graphs/graph4096\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjList, graph\_8192\_random\_50persent\_prob, "../Graphs/graph8192\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjList, grid\_graph\_16, "../Graphs/grid\_graph16.txt", 17, 254)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjList, grid\_graph\_32, "../Graphs/grid\_graph32.txt", 33, 1022)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjList, grid\_graph\_64, "../Graphs/grid\_graph64.txt", 160, 4094)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjList, graph\_100, "../Graphs/graph100.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjList, graph\_50, "../Graphs/graph50.txt", 0, 30)->Unit(benchmark::kMillisecond);

static void BM\_BFS\_AdjMatrix(benchmark::State& state, const std::string& graphFileName, int from, int to) {

AdjencyMatrix<int> matrix(getSolutionDir() + "\\" + graphFileName);

BFSPathManager<int> manager(std::make\_shared<AdjencyMatrix<int>>(matrix), from, to);

auto [length, path] = manager.getShortestPath();

for (auto \_ : state) {

for (int i = 0; i < 4; i++) {

auto res = manager.getNextShortestPath();

benchmark::DoNotOptimize(res);

if (res.first == -1) {

std::cout << "Path is not exist!\n";

}

}

benchmark::DoNotOptimize(manager.getShortestPath());

}

}

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMatrix, graph\_2048\_random\_25persent\_prob, "../Graphs/graph2048\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMatrix, graph\_4096\_random\_25persent\_prob, "../Graphs/graph4096\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMatrix, graph\_8192\_random\_25persent\_prob, "../Graphs/graph8192\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMatrix, graph\_2048\_random\_50persent\_prob, "../Graphs/graph2048\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMatrix, graph\_4096\_random\_50persent\_prob, "../Graphs/graph4096\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMatrix, graph\_8192\_random\_50persent\_prob, "../Graphs/graph8192\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMatrix, grid\_graph\_16, "../Graphs/grid\_graph16.txt", 17, 254)->Unit(benchmark::kMillisecond);

//BENCHMARK\_CAPTURE(BM\_BFS\_AdjMatrix, grid\_graph\_32, "../Graphs/grid\_graph32.txt", 33, 1022)->Unit(benchmark::kMillisecond);

//BENCHMARK\_CAPTURE(BM\_BFS\_AdjMatrix, grid\_graph\_64, "../Graphs/grid\_graph64.txt", 160, 4094)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMatrix, graph\_100, "../Graphs/graph100.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMatrix, graph\_50, "../Graphs/graph50.txt", 0, 30)->Unit(benchmark::kMillisecond);

static void BM\_BFS\_AdjList(benchmark::State& state, const std::string& graphFileName, int from, int to) {

AdjencyList<int> matrix(getSolutionDir() + "\\" + graphFileName);

BFSPathManager<int> manager(make\_shared<AdjencyList<int>>(matrix), from, to);

auto [length, path] = manager.getShortestPath();

for (auto \_ : state) {

for (int i = 0; i < 4; i++) {

auto res = manager.getNextShortestPath();

benchmark::DoNotOptimize(res);

if (res.first == -1) {

std::cout << "Path is not exist!\n";

}

}

benchmark::DoNotOptimize(manager.getShortestPath());

}

}

BENCHMARK\_CAPTURE(BM\_BFS\_AdjList, graph\_2048\_random\_25persent\_prob, "../Graphs/graph2048\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjList, graph\_4096\_random\_25persent\_prob, "../Graphs/graph4096\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjList, graph\_8192\_random\_25persent\_prob, "../Graphs/graph8192\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjList, graph\_2048\_random\_50persent\_prob, "../Graphs/graph2048\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjList, graph\_4096\_random\_50persent\_prob, "../Graphs/graph4096\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjList, graph\_8192\_random\_50persent\_prob, "../Graphs/graph8192\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjList, grid\_graph\_16, "../Graphs/grid\_graph16.txt", 17, 254)->Unit(benchmark::kMillisecond);

//BENCHMARK\_CAPTURE(BM\_BFS\_AdjList, grid\_graph\_32, "../Graphs/grid\_graph32.txt", 33, 1022)->Unit(benchmark::kMillisecond);

//BENCHMARK\_CAPTURE(BM\_BFS\_AdjList, grid\_graph\_64, "../Graphs/grid\_graph64.txt", 160, 4094)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjList, graph\_100, "../Graphs/graph100.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjList, graph\_50, "../Graphs/graph50.txt", 0, 30)->Unit(benchmark::kMillisecond);

static void BM\_BFS\_AdjMap(benchmark::State& state, const std::string& graphFileName, int from, int to) {

AdjencyMap<int> matrix(getSolutionDir() + "\\" + graphFileName);

BFSPathManager<int> manager(std::make\_shared<AdjencyMap<int>>(matrix), from, to);

auto [length, path] = manager.getShortestPath();

for (auto \_ : state) {

for (int i = 0; i < 4; i++) {

auto res = manager.getNextShortestPath();

benchmark::DoNotOptimize(res);

if (res.first == -1) {

std::cout << "Path is not exist!\n";

}

}

benchmark::DoNotOptimize(manager.getShortestPath());

}

}

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMap, graph\_2048\_random\_25persent\_prob, "../Graphs/graph2048\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMap, graph\_4096\_random\_25persent\_prob, "../Graphs/graph4096\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMap, graph\_8192\_random\_25persent\_prob, "../Graphs/graph8192\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMap, graph\_2048\_random\_50persent\_prob, "../Graphs/graph2048\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMap, graph\_4096\_random\_50persent\_prob, "../Graphs/graph4096\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMap, graph\_8192\_random\_50persent\_prob, "../Graphs/graph8192\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMap, grid\_graph\_16, "../Graphs/grid\_graph16.txt", 17, 254)->Unit(benchmark::kMillisecond);

//BENCHMARK\_CAPTURE(BM\_BFS\_AdjMap, grid\_graph\_32, "../Graphs/grid\_graph32.txt", 33, 1022)->Unit(benchmark::kMillisecond);

//BENCHMARK\_CAPTURE(BM\_BFS\_AdjMap, grid\_graph\_64, "../Graphs/grid\_graph64.txt", 160, 4094)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMap, graph\_100, "../Graphs/graph100.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMap, graph\_50, "../Graphs/graph50.txt", 0, 30)->Unit(benchmark::kMillisecond);

static void BM\_Dijkstra\_AdjMap(benchmark::State& state, const std::string& graphFileName, int from, int to) {

AdjencyMap<int> matrix(getSolutionDir() + "\\" + graphFileName);

DijkstraPathManager<int> manager(std::make\_shared<AdjencyMap<int>>(matrix), from, to);

auto [length, path] = manager.getShortestPath();

for (auto \_ : state) {

for (int i = 0; i < 4; i++) {

auto res = manager.getNextShortestPath();

benchmark::DoNotOptimize(res);

if (res.first == -1) {

std::cout << "Path is not exist!\n";

}

}

benchmark::DoNotOptimize(manager.getShortestPath());

}

}

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMap, graph\_100, "../Graphs/graph100.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMap, graph\_50, "../Graphs/graph50.txt", 0, 30)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMap, graph\_2048\_random\_25persent\_prob, "../Graphs/graph2048\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMap, graph\_4096\_random\_25persent\_prob, "../Graphs/graph4096\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMap, graph\_8192\_random\_25persent\_prob, "../Graphs/graph8192\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMap, graph\_2048\_random\_50persent\_prob, "../Graphs/graph2048\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMap, graph\_4096\_random\_50persent\_prob, "../Graphs/graph4096\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMap, graph\_8192\_random\_50persent\_prob, "../Graphs/graph8192\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMap, grid\_graph\_16, "../Graphs/grid\_graph16.txt", 17, 254)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMap, grid\_graph\_32, "../Graphs/grid\_graph32.txt", 33, 1022)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMap, grid\_graph\_64, "../Graphs/grid\_graph64.txt", 160, 4094)->Unit(benchmark::kMillisecond);

BENCHMARK\_MAIN();

//bool storage

#include <Benchmark/benchmark.h>

#include <memory>

#include <iostream>

#include <string>

#include "DijkstraPathManager.h"

#include "DijkstraPathManager.cpp"

#include "Graph.h"

#include "Graph.cpp"

using namespace std;

#define STRINGIFY(x) #x

#define EXPAND(x) STRINGIFY(x)

string projectDir;

string getProjectDir() {

if (!projectDir.empty())

return projectDir;

string s = EXPAND(UNITTESTPRJ);

s.erase(0, 1); // erase the first quote

s.erase(s.size() - 2); // erase the last quote and the dot

projectDir = s;

return projectDir;

}

string solutionDir;

string getSolutionDir() {

if (!solutionDir.empty())

return solutionDir;

string s = EXPAND(UNITTESTPRJ);

s.erase(0, 1); // erase the first quote

s.erase(s.size() - 2); // erase the last quote and the dot

// Находим позицию последнего символа '\'

size\_t found = s.find\_last\_of('\\');

// Если найден символ '\', обрезаем строку по этой позиции

if (found != std::string::npos) {

s = s.substr(0, found);

}

solutionDir = s;

return solutionDir;

}

static void BM\_Dijkstra\_AdjMatrix(benchmark::State& state, const std::string& graphFileName, int from, int to) {

AdjencyMatrix<bool> matrix(getSolutionDir() + "\\" + graphFileName);

DijkstraPathManager<bool> manager(std::make\_shared<AdjencyMatrix<bool>>(matrix), from, to);

auto [length, path] = manager.getShortestPath();

for (auto \_ : state) {

for (int i = 0; i < 4; i++) {

auto res = manager.getNextShortestPath();

benchmark::DoNotOptimize(res);

if (res.first == -1) {

std::cout << "Path is not exist!\n";

}

}

benchmark::DoNotOptimize(manager.getShortestPath());

}

}

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMatrix, graph\_2048\_random\_25persent\_prob, "../Graphs/graph2048\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMatrix, graph\_4096\_random\_25persent\_prob, "../Graphs/graph4096\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMatrix, graph\_8192\_random\_25persent\_prob, "../Graphs/graph8192\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMatrix, graph\_2048\_random\_50persent\_prob, "../Graphs/graph2048\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMatrix, graph\_4096\_random\_50persent\_prob, "../Graphs/graph4096\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMatrix, graph\_8192\_random\_50persent\_prob, "../Graphs/graph8192\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMatrix, grid\_graph\_16, "../Graphs/grid\_graph16.txt", 17, 254)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMatrix, grid\_graph\_32, "../Graphs/grid\_graph32.txt", 33, 1022)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMatrix, grid\_graph\_64, "../Graphs/grid\_graph64.txt", 160, 4094)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMatrix, graph\_100, "../Graphs/graph100.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMatrix, graph\_50, "../Graphs/graph50.txt", 0, 30)->Unit(benchmark::kMillisecond);

static void BM\_Dijkstra\_AdjList(benchmark::State& state, const std::string& graphFileName, int from, int to) {

AdjencyList<bool> matrix(getSolutionDir() + "\\" + graphFileName);

DijkstraPathManager<bool> manager(make\_shared<AdjencyList<bool>>(matrix), from, to);

auto [length, path] = manager.getShortestPath();

for (auto \_ : state) {

for (int i = 0; i < 4; i++) {

auto res = manager.getNextShortestPath();

benchmark::DoNotOptimize(res);

if (res.first == -1) {

std::cout << "Path is not exist!\n";

}

}

benchmark::DoNotOptimize(manager.getShortestPath());

}

}

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjList, graph\_2048\_random\_25persent\_prob, "../Graphs/graph2048\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjList, graph\_4096\_random\_25persent\_prob, "../Graphs/graph4096\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjList, graph\_8192\_random\_25persent\_prob, "../Graphs/graph8192\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjList, graph\_2048\_random\_50persent\_prob, "../Graphs/graph2048\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjList, graph\_4096\_random\_50persent\_prob, "../Graphs/graph4096\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjList, graph\_8192\_random\_50persent\_prob, "../Graphs/graph8192\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjList, grid\_graph\_16, "../Graphs/grid\_graph16.txt", 17, 254)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjList, grid\_graph\_32, "../Graphs/grid\_graph32.txt", 33, 1022)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjList, grid\_graph\_64, "../Graphs/grid\_graph64.txt", 160, 4094)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjList, graph\_100, "../Graphs/graph100.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjList, graph\_50, "../Graphs/graph50.txt", 0, 30)->Unit(benchmark::kMillisecond);

static void BM\_BFS\_AdjMatrix(benchmark::State& state, const std::string& graphFileName, int from, int to) {

AdjencyMatrix<bool> matrix(getSolutionDir() + "\\" + graphFileName);

BFSPathManager<bool> manager(std::make\_shared<AdjencyMatrix<bool>>(matrix), from, to);

auto [length, path] = manager.getShortestPath();

for (auto \_ : state) {

for (int i = 0; i < 4; i++) {

auto res = manager.getNextShortestPath();

benchmark::DoNotOptimize(res);

if (res.first == -1) {

std::cout << "Path is not exist!\n";

}

}

benchmark::DoNotOptimize(manager.getShortestPath());

}

}

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMatrix, graph\_2048\_random\_25persent\_prob, "../Graphs/graph2048\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMatrix, graph\_4096\_random\_25persent\_prob, "../Graphs/graph4096\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMatrix, graph\_8192\_random\_25persent\_prob, "../Graphs/graph8192\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMatrix, graph\_2048\_random\_50persent\_prob, "../Graphs/graph2048\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMatrix, graph\_4096\_random\_50persent\_prob, "../Graphs/graph4096\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMatrix, graph\_8192\_random\_50persent\_prob, "../Graphs/graph8192\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMatrix, grid\_graph\_16, "../Graphs/grid\_graph16.txt", 17, 254)->Unit(benchmark::kMillisecond);

//BENCHMARK\_CAPTURE(BM\_BFS\_AdjMatrix, grid\_graph\_32, "../Graphs/grid\_graph32.txt", 33, 1022)->Unit(benchmark::kMillisecond);

//BENCHMARK\_CAPTURE(BM\_BFS\_AdjMatrix, grid\_graph\_64, "../Graphs/grid\_graph64.txt", 160, 4094)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMatrix, graph\_100, "../Graphs/graph100.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMatrix, graph\_50, "../Graphs/graph50.txt", 0, 30)->Unit(benchmark::kMillisecond);

static void BM\_BFS\_AdjList(benchmark::State& state, const std::string& graphFileName, int from, int to) {

AdjencyList<bool> matrix(getSolutionDir() + "\\" + graphFileName);

BFSPathManager<bool> manager(make\_shared<AdjencyList<bool>>(matrix), from, to);

auto [length, path] = manager.getShortestPath();

for (auto \_ : state) {

for (int i = 0; i < 4; i++) {

auto res = manager.getNextShortestPath();

benchmark::DoNotOptimize(res);

if (res.first == -1) {

std::cout << "Path is not exist!\n";

}

}

benchmark::DoNotOptimize(manager.getShortestPath());

}

}

BENCHMARK\_CAPTURE(BM\_BFS\_AdjList, graph\_2048\_random\_25persent\_prob, "../Graphs/graph2048\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjList, graph\_4096\_random\_25persent\_prob, "../Graphs/graph4096\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjList, graph\_8192\_random\_25persent\_prob, "../Graphs/graph8192\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjList, graph\_2048\_random\_50persent\_prob, "../Graphs/graph2048\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjList, graph\_4096\_random\_50persent\_prob, "../Graphs/graph4096\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjList, graph\_8192\_random\_50persent\_prob, "../Graphs/graph8192\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjList, grid\_graph\_16, "../Graphs/grid\_graph16.txt", 17, 254)->Unit(benchmark::kMillisecond);

//BENCHMARK\_CAPTURE(BM\_BFS\_AdjList, grid\_graph\_32, "../Graphs/grid\_graph32.txt", 33, 1022)->Unit(benchmark::kMillisecond);

//BENCHMARK\_CAPTURE(BM\_BFS\_AdjList, grid\_graph\_64, "../Graphs/grid\_graph64.txt", 160, 4094)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjList, graph\_100, "../Graphs/graph100.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjList, graph\_50, "../Graphs/graph50.txt", 0, 30)->Unit(benchmark::kMillisecond);

static void BM\_BFS\_AdjMap(benchmark::State& state, const std::string& graphFileName, int from, int to) {

AdjencyMap<bool> matrix(getSolutionDir() + "\\" + graphFileName);

BFSPathManager<bool> manager(std::make\_shared<AdjencyMap<bool>>(matrix), from, to);

auto [length, path] = manager.getShortestPath();

for (auto \_ : state) {

for (int i = 0; i < 4; i++) {

auto res = manager.getNextShortestPath();

benchmark::DoNotOptimize(res);

if (res.first == -1) {

std::cout << "Path is not exist!\n";

}

}

benchmark::DoNotOptimize(manager.getShortestPath());

}

}

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMap, graph\_2048\_random\_25persent\_prob, "../Graphs/graph2048\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMap, graph\_4096\_random\_25persent\_prob, "../Graphs/graph4096\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMap, graph\_8192\_random\_25persent\_prob, "../Graphs/graph8192\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMap, graph\_2048\_random\_50persent\_prob, "../Graphs/graph2048\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMap, graph\_4096\_random\_50persent\_prob, "../Graphs/graph4096\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMap, graph\_8192\_random\_50persent\_prob, "../Graphs/graph8192\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMap, grid\_graph\_16, "../Graphs/grid\_graph16.txt", 17, 254)->Unit(benchmark::kMillisecond);

//BENCHMARK\_CAPTURE(BM\_BFS\_AdjMap, grid\_graph\_32, "../Graphs/grid\_graph32.txt", 33, 1022)->Unit(benchmark::kMillisecond);

//BENCHMARK\_CAPTURE(BM\_BFS\_AdjMap, grid\_graph\_64, "../Graphs/grid\_graph64.txt", 160, 4094)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMap, graph\_100, "../Graphs/graph100.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_BFS\_AdjMap, graph\_50, "../Graphs/graph50.txt", 0, 30)->Unit(benchmark::kMillisecond);

static void BM\_Dijkstra\_AdjMap(benchmark::State& state, const std::string& graphFileName, int from, int to) {

AdjencyMap<bool> matrix(getSolutionDir() + "\\" + graphFileName);

DijkstraPathManager<bool> manager(std::make\_shared<AdjencyMap<bool>>(matrix), from, to);

auto [length, path] = manager.getShortestPath();

for (auto \_ : state) {

for (int i = 0; i < 4; i++) {

auto res = manager.getNextShortestPath();

benchmark::DoNotOptimize(res);

if (res.first == -1) {

std::cout << "Path is not exist!\n";

}

}

benchmark::DoNotOptimize(manager.getShortestPath());

}

}

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMap, graph\_100, "../Graphs/graph100.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMap, graph\_50, "../Graphs/graph50.txt", 0, 30)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMap, graph\_2048\_random\_25persent\_prob, "../Graphs/graph2048\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMap, graph\_4096\_random\_25persent\_prob, "../Graphs/graph4096\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMap, graph\_8192\_random\_25persent\_prob, "../Graphs/graph8192\_random\_25persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMap, graph\_2048\_random\_50persent\_prob, "../Graphs/graph2048\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMap, graph\_4096\_random\_50persent\_prob, "../Graphs/graph4096\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMap, graph\_8192\_random\_50persent\_prob, "../Graphs/graph8192\_random\_50persent\_prob.txt", 0, 89)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMap, grid\_graph\_16, "../Graphs/grid\_graph16.txt", 17, 254)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMap, grid\_graph\_32, "../Graphs/grid\_graph32.txt", 33, 1022)->Unit(benchmark::kMillisecond);

BENCHMARK\_CAPTURE(BM\_Dijkstra\_AdjMap, grid\_graph\_64, "../Graphs/grid\_graph64.txt", 160, 4094)->Unit(benchmark::kMillisecond);

BENCHMARK\_MAIN();