ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 8

«Графы»

Выполнил работу

Смирнов Александр

Академическая группа №J3111

Принято

Ментор Вершинин Владислав Константинович

Санкт-Петербург

2024

1. Введение

Цель работы: попрактиковаться в решении задач на графы, реализовать алгоритмы BFS, DFS, Дейкстра и А\*

Задачи:

1. Разобраться в условии, изучить предоставленный граф
2. Написать парсер графа из текстового файла
3. Реализовать алгоритмы
4. Реализовать алгоритм поиска ближайшей ноды
5. Провести анализ полученных результатов и оформить отчёт
6. Теоретическая подготовка

Использовал 4 алгоритма из условия задачи. Для этого применял встроенные типы данных, такие как словари, bool, double, pair и и т.д., а так же кастомные структуры для хранения графа, вершин и ребер.

1. Реализация

Я начал с создания структур для хранения графа. Код представлен на рис.1

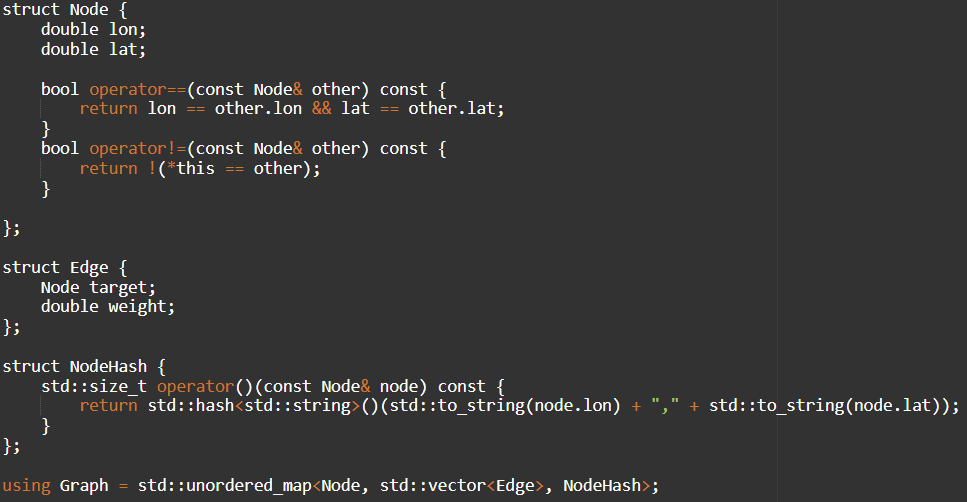


Рисунок 1. Код с инициализацией структуры для хранения графа

Структура вершины хранит две координаты широты и долготы, а также бинарные операторы для сравнения вершин на признак равенства/неравенства. Структура ребра хранит ноду, в которую ведет это ребро, а также вес ребра. Структура NodeHash позволяет переводить вершину, заданную парой чисел с плавающей точкой, в строку, как она хранится в текстовом файле графа. Тогда граф представляет из себя словарь, где ключи – вершины, а элементы – вектора ребер.

Затем я написал функцию, которая по заданным координатам находит ближайшую вершину, чтобы можно было вводить произвольные координаты, а не только те, что есть в графе. Код представлен на рис. 2.

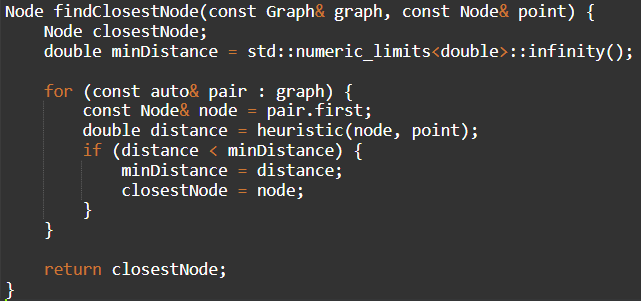


Рисунок 2. Функция нахождения ближайшей ноды

Затем я написал парсер графа из текстового файла. Функция парсера представлена на рис.3

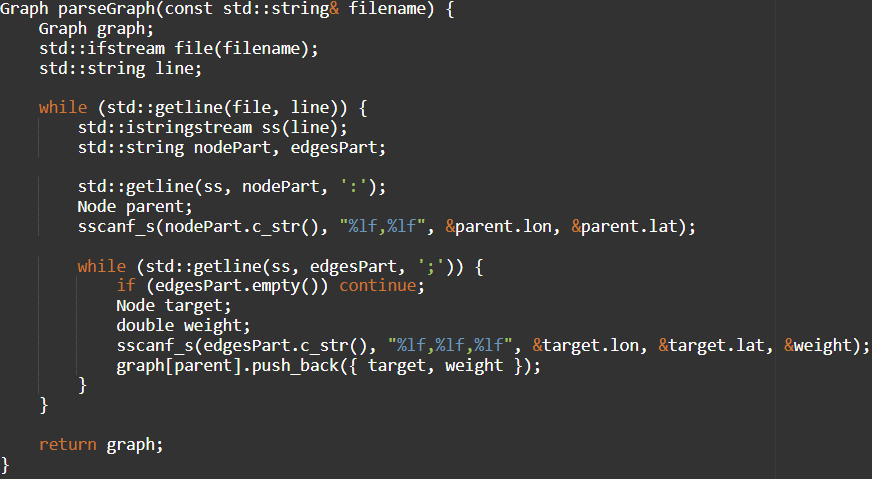


Рисунок 3. Функция парсинга графа

Далее я начал реализовывать алгоритмы для поиска кратчайшего пути.

Я начал с алгоритма A\*. Код алгоритма представлен на рис. 3.

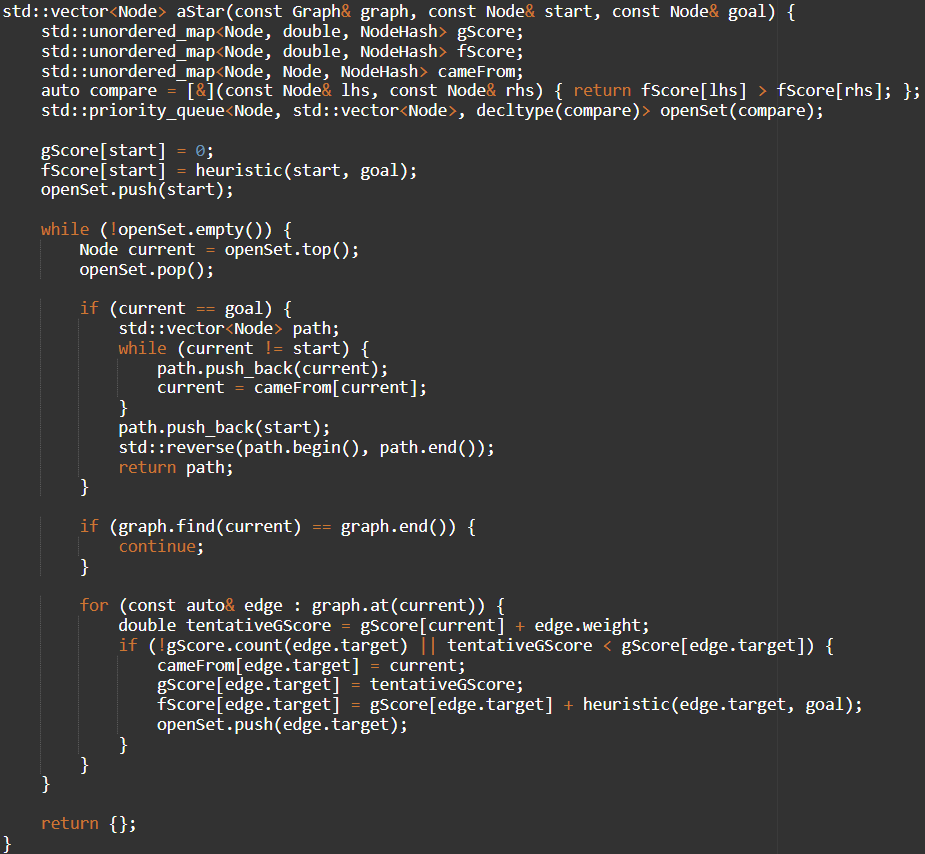


Рисунок 3. Алгоритм А\*

Алгоритм A\* находит оптимальный вариант благодаря вычислению суммарной стоимости всех путей между начальной и конечной точкой. Он пошагово просматривает все пути, ведущие от начальной вершины в конечную, пока не найдёт минимальный.

На каждом шаге алгоритм выбирает вершину с наименьшей суммой стоимости пути от начальной вершины до текущей и эвристической оценки расстояния от текущей вершины до цели. Чем меньше эта сумма, тем раньше открывается вершина, так как через неё предположительно можно быстрее достичь цели.

Таким образом, алгоритм A\* сначала просматривает те маршруты, которые благодаря имеющейся информации (эвристической функции) в данный момент являются наилучшими.

Эффективность алгоритма A\* в значительной степени зависит от используемой эвристики, выбор которой может существенно повлиять на производительность и результативность алгоритма. В нашем случае лучше всего подходит эвристика евклидова расстояния. Код функции для вычисления эвристики представлен на рис. 4.

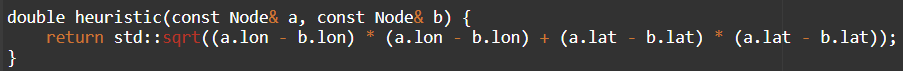


Рисунок 4. Функция вычисления эвристики

Следующим алгоритмом был обход в ширину. Код представлен на рис. 5

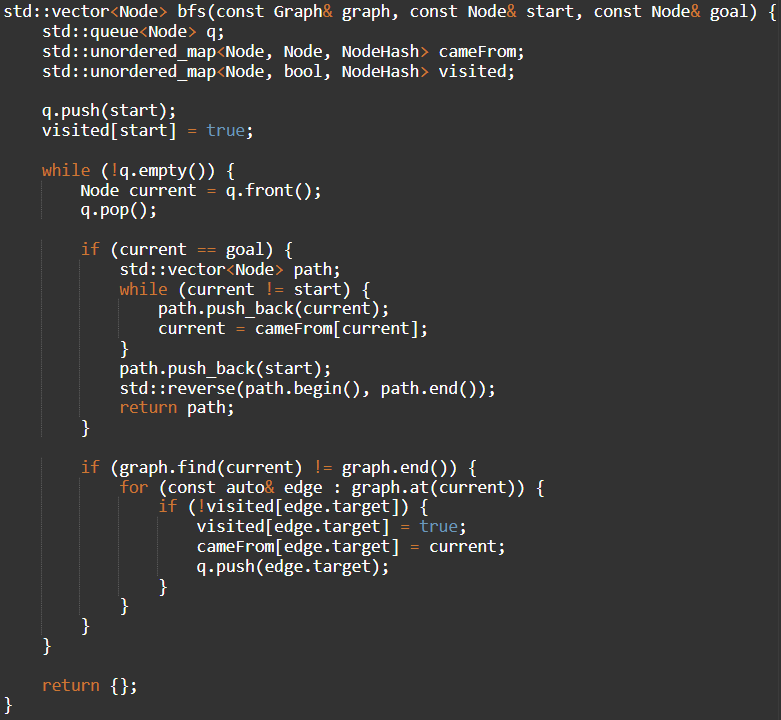


Рисунок 5. Код функции обхода в ширину

Обход в ширину работает по принципу посещения всех вершин графа по одному разу, но в другом порядке: по увеличению расстояния до начальной вершины.

Сначала исследуются все вершины, смежные с начальной (с которой начинается обход). Эти вершины находятся на расстоянии 1 от начальной. Затем исследуются все вершины на расстоянии 2 от начальной, затем все на расстоянии 3 и так далее. При этом для каждой вершины сразу находятся длина кратчайшего маршрута от начальной вершины.

Следующим алгоритмом был обход в глубину. Код представлен на рис. 6

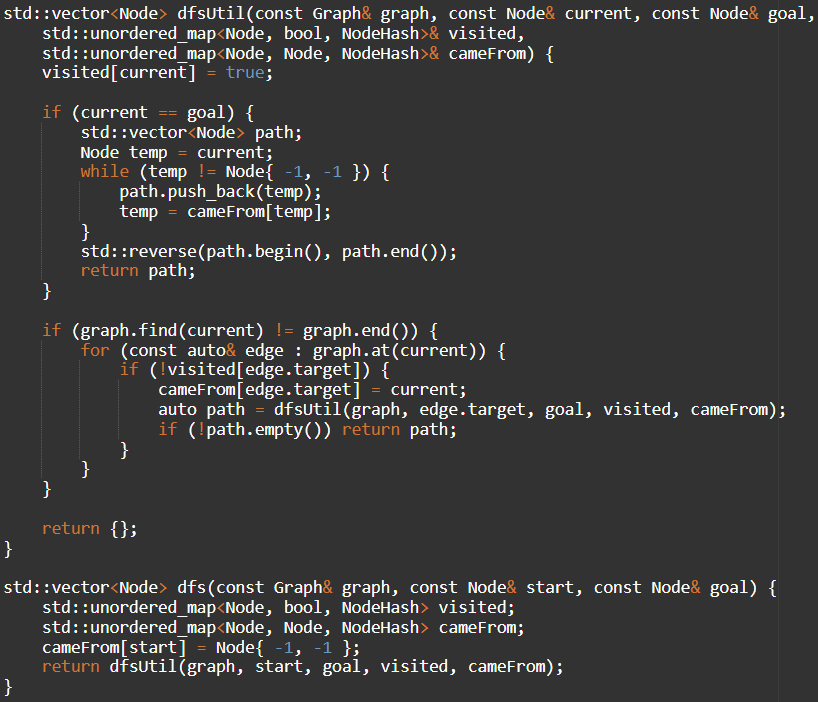


Рисунок 6. Код функции обхода в глубину

При этом я использовал рекурсивную функцию dfsUtil. Рекурсивный алгоритм обхода в глубину работает следующим образом:

Все вершины графа отмечаются как непосещённые. Выбирается первая вершина и помечается как посещённая.

Для последней помеченной как посещённой вершины выбирается смежная вершина, которая первая помечена как непосещённая, и ей присваивается значение посещённой. Если таких вершин нет, то берётся предыдущая помеченная вершина.

Повторяется шаг 2 до тех пор, пока все вершины не будут помечены как посещённые. Таким образом, для обхода всего графа нужно переместиться в соседнюю вершину, после чего повторить для этой вершины алгоритм обхода.

Последним алгоритмом был алгоритм Дейкстры. Код алгоритма представлен на рис. 6

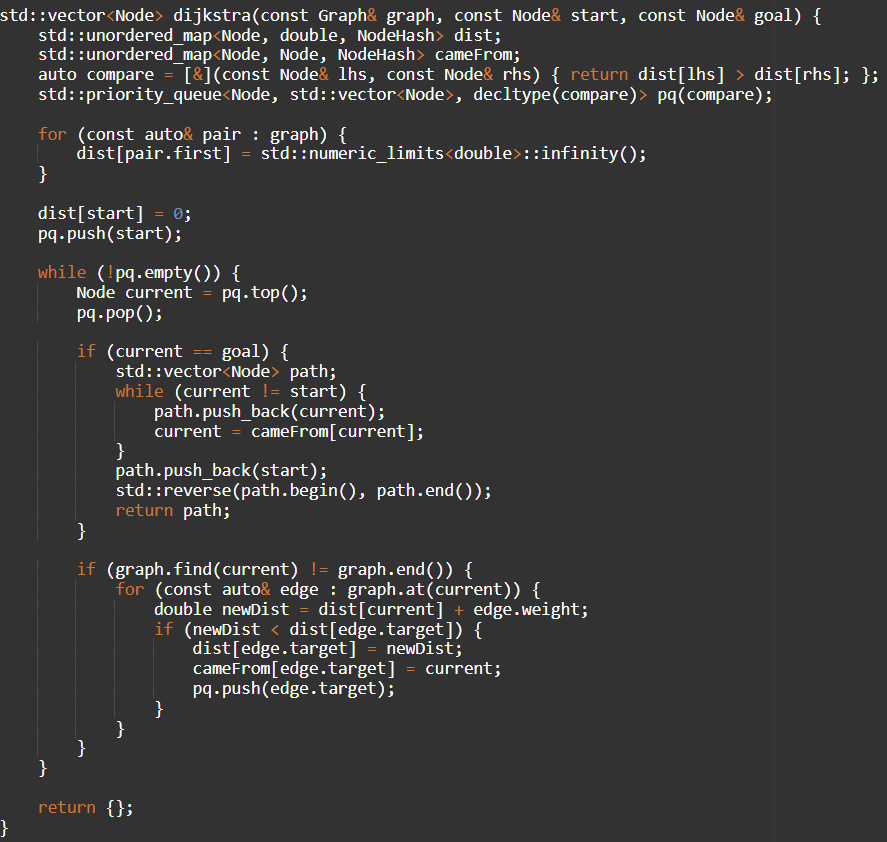


Рисунок 6. Алгоритм Дейкстры

Алгоритм Дейкстры работает пошагово. Сначала выбирается точка, от которой будут отсчитываться пути. Затем алгоритм поочерёдно ищет самые короткие маршруты из исходной точки в другие. Вершины, где он уже побывал, отмечает посещёнными. Алгоритм использует посещённые вершины, когда рассчитывает пути для непосещённых.

Идея алгоритма в том, что на каждом шаге помечается определённым образом выбранная вершина, а далее просматриваются все последующие (ещё не отмеченные) вершины графа и вычисляется длина пути до каждой из них от начальной точки. Помеченная на каждом этапе новая вершина (та, до которой путь оказался кратчайшим) становится определяющей для следующего шага.

Когда непосещённые вершины заканчиваются, алгоритм прекращает работу. Результат его действия — список кратчайших маршрутов до каждой точки из исходной.

1. Экспериментальная часть

Чтобы понять, какой алгоритм является наилучшим, проведем сравнительный анализ. Сравнение преимуществ, недостатков и времени работы алгоритмов представлено в таблице 1.

| **Алгоритм** | **Краткость пути** | **Сложность** | **Преимущества** | **Недостатки** |
| --- | --- | --- | --- | --- |
| **A\*** | Гарантированно кратчайший путь при хорошей эвристике | O(E \* log(V)) | Быстрое нахождение кратчайшего пути с эвристикой | Зависит от качества эвристики |
| **Дейкстра** | Гарантирует кратчайший путь | O((V + E) \* log(V)) | Находит оптимальный путь без эвристики | Медленный, если эвристика отсутствует |
| **BFS** | Только для невзвешенных графов | O(V + E) | Простой и эффективный для невзвешенных графов | Не работает с графами с весами рёбер |
| **DFS** | Не гарантирует кратчайший путь | O(V + E) | Прост в реализации | Не находит кратчайший путь, плох для длинных путей |

Таблица 1. Сравнительная таблица алгоритмов

Можно сделать следующие выводы:

* A\* и Дейкстра — наилучшие для поиска кратчайшего пути в графах с весами рёбер.
* BFS идеально подходит для графов с одинаковыми или отсутствующими весами рёбер.
* DFS полезен в задачах, где не требуется кратчайший путь, например, для обхода или поиска всех возможных путей.

Таким образом, в контексте именно нашей задачи, наилучшим вариантом является алгоритм А\*, так как эвристика евклидова расстояния в теории достаточно эффективна в нашей задаче. Проверим эту гипотезу, запустив алгоритмы на предоставленном графе. Я запустил поиск пути от дома до корпуса ИТМО, как требовалось в условии задачи, результат представлен на рис. 7

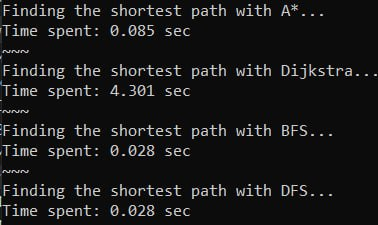


Рисунок 7. Сравнение времени работы алгоритмов на реальном графе.

Здесь видим, что BFS и DFS работают значительно быстрее. При этом А\* заметно быстрее Дейкстры, что говорит о том, что мы подобрали удачную эвристику.

1. Заключение

В ходе выполнения работы мною были реализованы алгоритмы для решения задачи нахождения кратчайшего пути между двумя локациями в городе, представленном в виде связного графа. Цель работы была достигнута путём тестирования на разных входных параметрах. Полученные результаты также совпадают с теоретическими оценками сложности алгоритмов.

В качестве дальнейших исследований можно рассмотреть варианты оптимизации алгоритмов, улучшения эвристики и экспериментов на графах большего размера.