ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № VIII

«Графы. Medium+ (BFS,DFS, Dijkstra, А\*)»

Выполнил работу

Никитин Данил

Академическая группа №J3110

Принято

Должность, звание Фамилия Имя преподавателя

Санкт-Петербург

2024

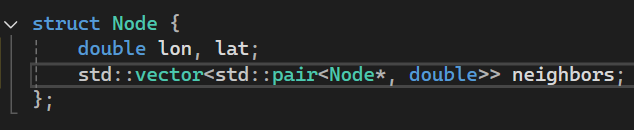
**Структура отчёта:**

1. Введение

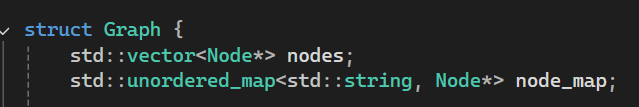
Цель: Приобрести навыки по работе с графами и алгоритмами на них.

Задачи:

1. Парсинг строк датасета
2. Создание структуры графа
3. Реализация алгоритмов BFS, DFS, Dijkstra, A\*
4. Реализация вывода кратчайшего пути найденного каждым из алгосов
5. Подсчет реального времени – эксперимент
6. Теоретическая временная и пространственная сложность
7. Теоретическая подготовка
8. Алгоритмы BFS, DFS, Dijkstra, A\*
9. Работа с файлами в С++
10. Реализации структуры графа и его вершин
11. Структуры данных : Стэк ,очередь, хэш-таблицы(unordered\_map), пара(pair).
12. Реализация
13. Структура вершин хранит в себе координаты вершины и вектор пар на смежные с ней вершины и веса ребер инцидентных ей и смежной.

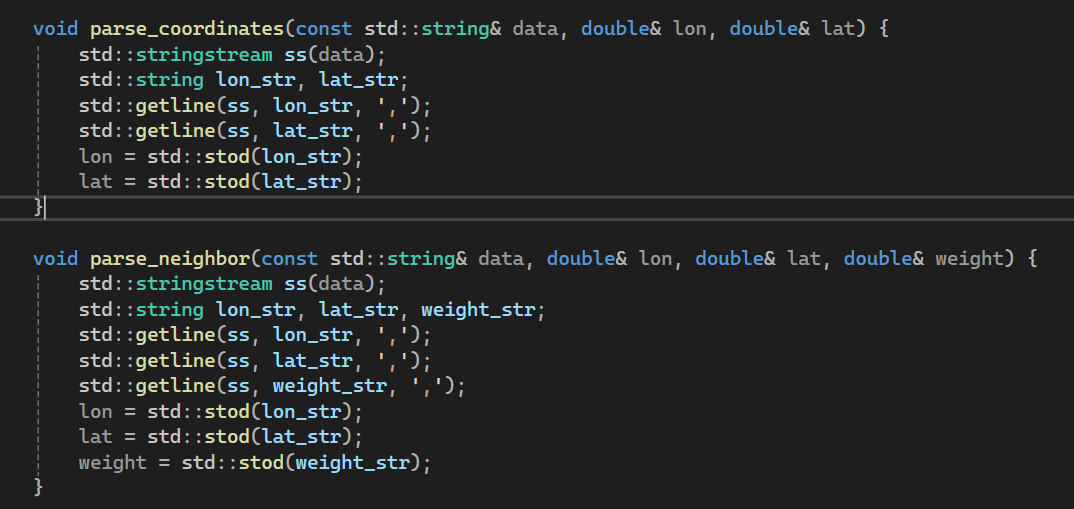


2) Структура графа хранит вектор ссылок на все вершины и хэш-таблицу ключи которой есть строка широта&долгота, а значения ссылка на вершину.

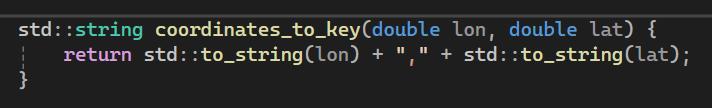


1. Парсинг строк по запятым , вытаскиваем долготу ,широту и веса.

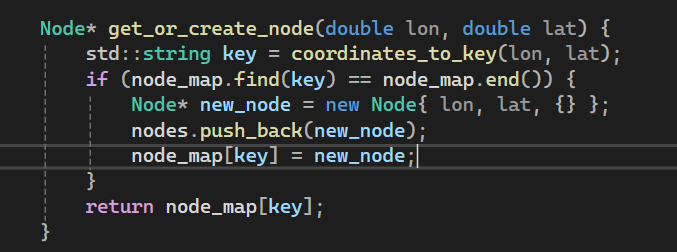
Функции две для вершины и для ее соседей ,т.к. в основной функции мы делим входящую строку по символам «;» и «:».



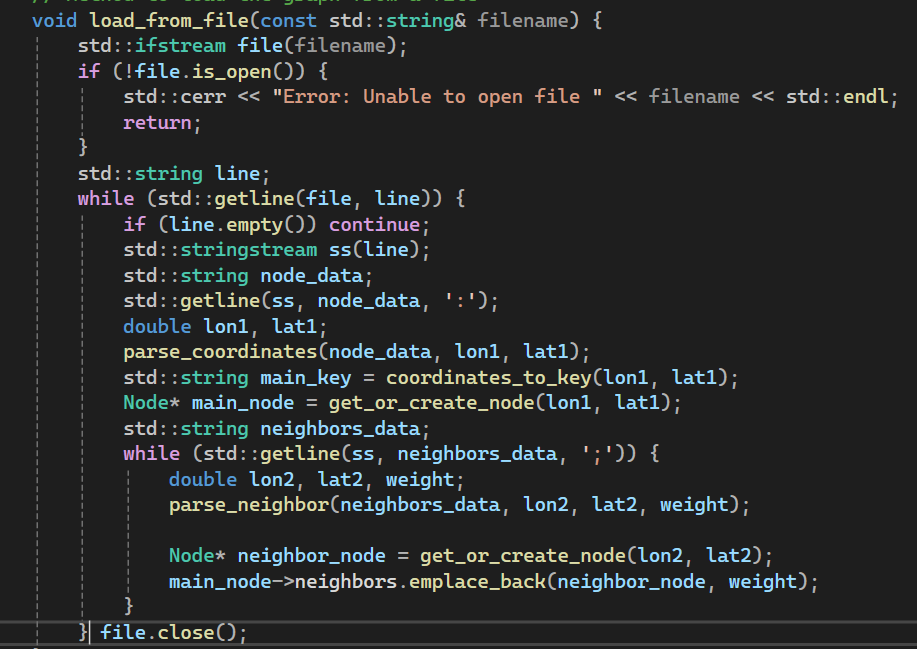
Функция по созданию строчных ключей для нашей хэш-таблицы из структуры графа.



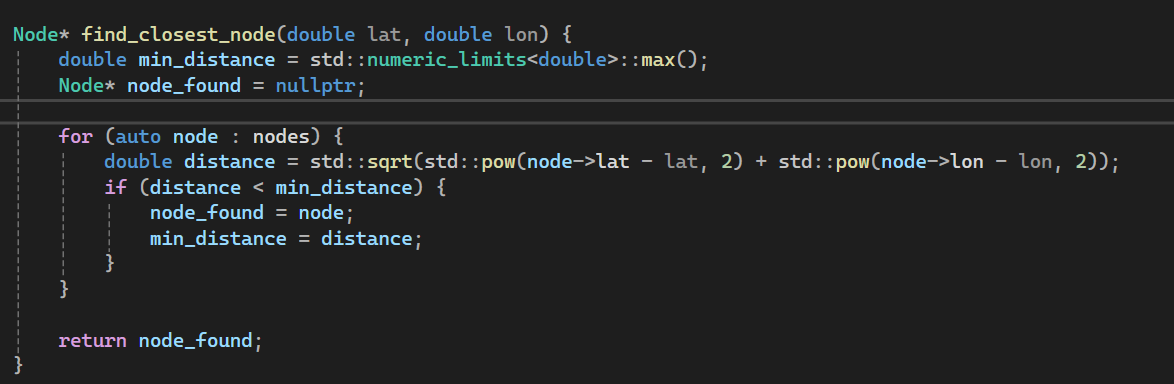
Функция для добавления вершин в структуру граф



Функция для извлечения данных из файла в граф: Читаем файл, парсим строку до начала данных о соседей ,создаем узел или если он существует возвращаем указатель на него, то же самое делаем с соседями.



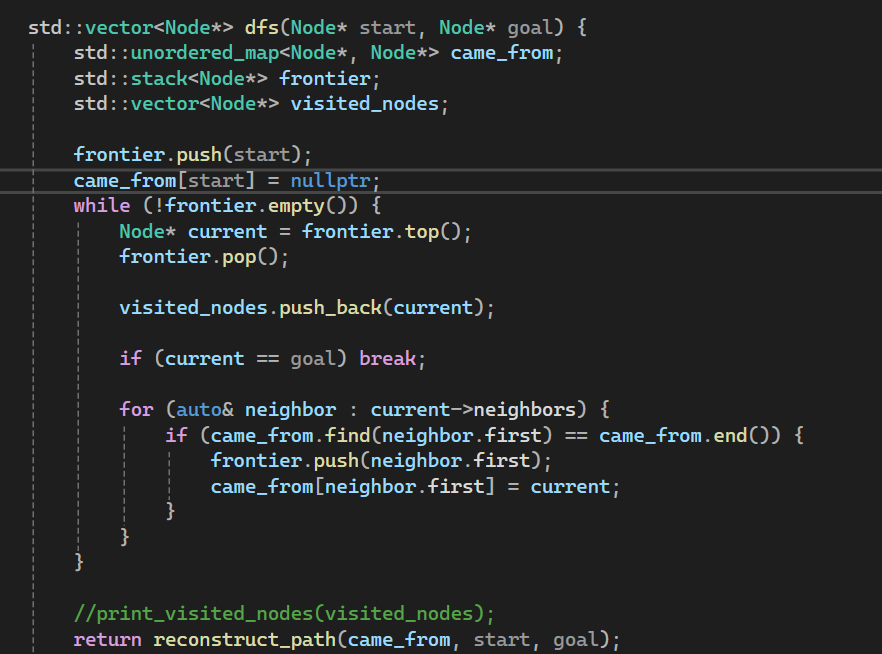
1. Функция для поиска ближайших существующих в графе узлов к месту. (Доработанная версия по данному в задании)



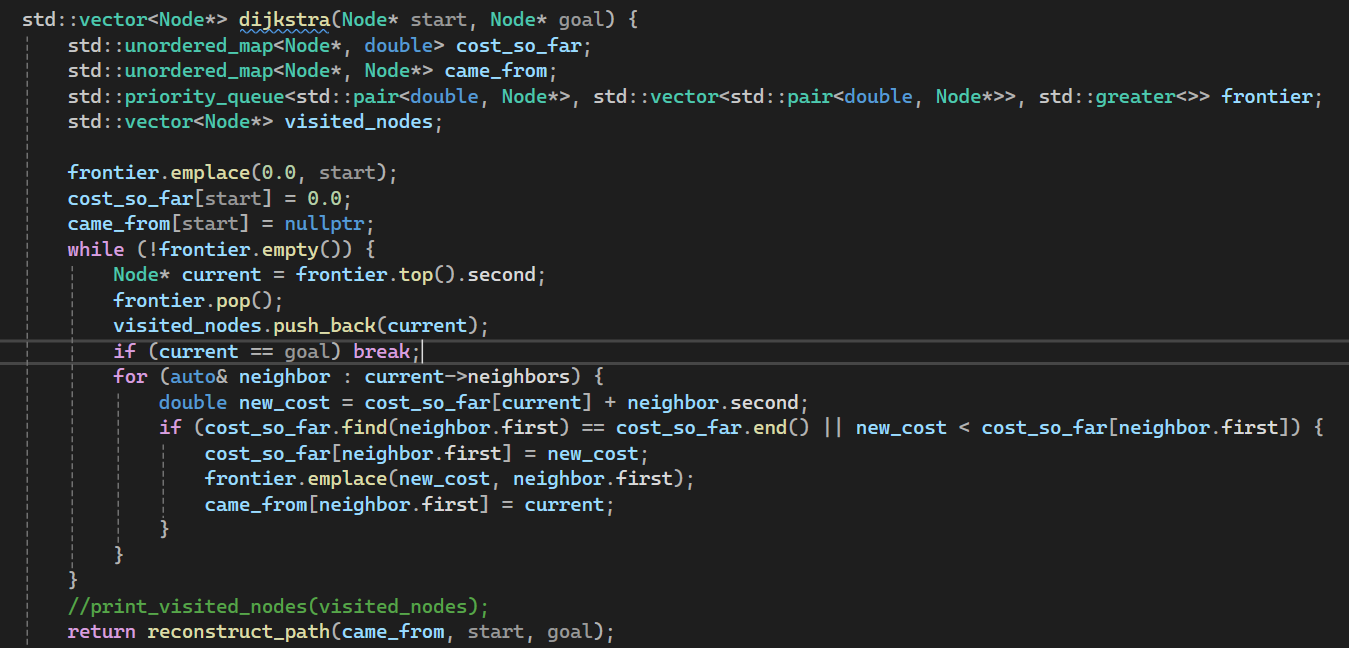
1. Алгоритм поиска в ширину основан на очереди(FIFO), мы добавляем вершину start а смежные ей добавляем в очередь , так делаем обход в ширину по слоям. Кратчайший путь этим способом не учитывает вес ребер ,он ищет именно самый короткий по количеству ребер путь. Также мы добавили запоминание пути по которому идет алгоритм ,его можно вывести с помощью вспомогательной функции (будет представлена в практическом разделе), запоминание маршрута добавлено во все 4 алгоритма.



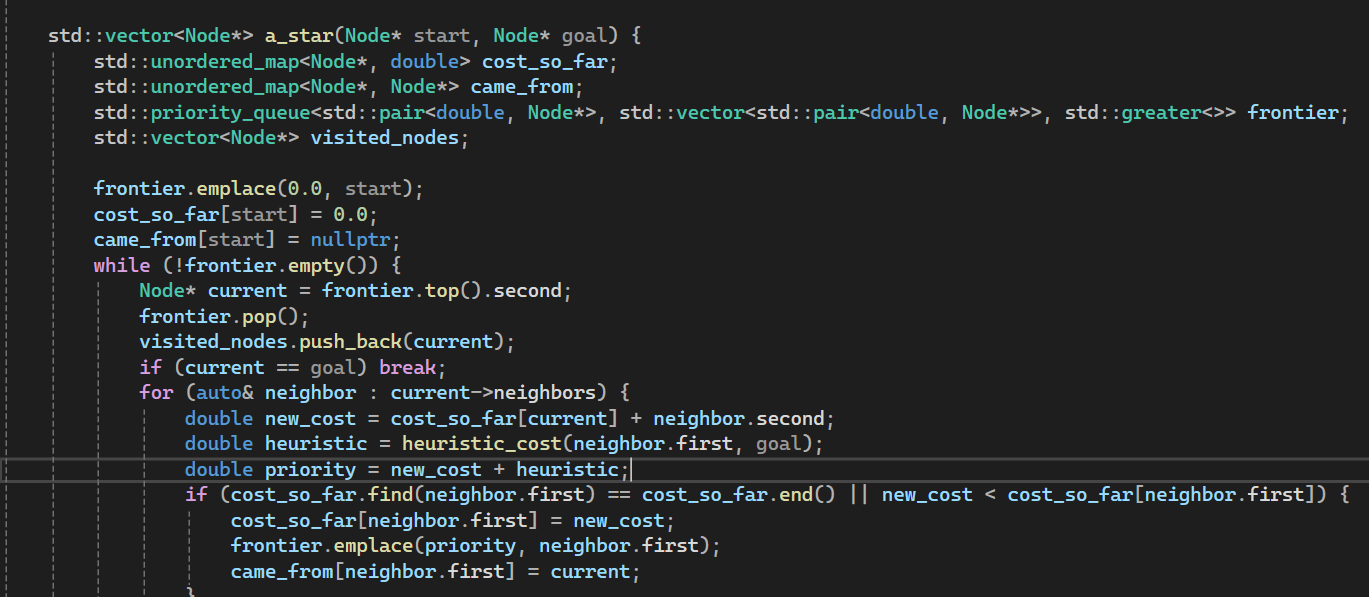
1. Алгоритм поиска в глубину основан на стэке (LIFO) , в отличие от очереди он един для всей программы и обновляется каждой вершиной , а не строится заново для каждой вершины.

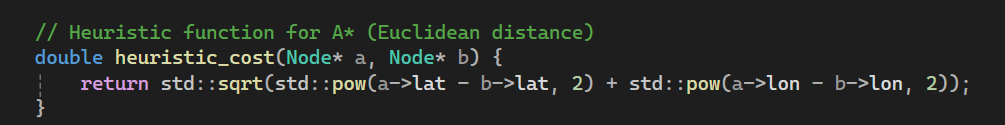


1. Алгоритм Дейкстры, является жадным алгоритмом ,реализуется с помощью приоритетной очереди, на каждом шаге мы выбираем инцедентное ребро с наименьшим весом.



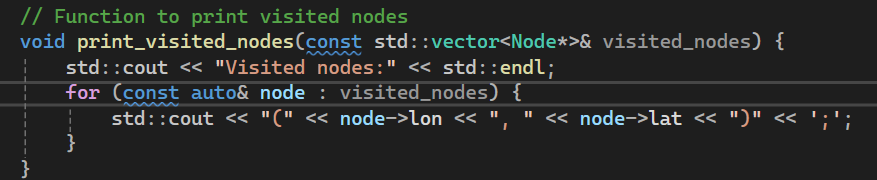
1. Алгоритм А\* - есть улучшенная версия алгоритма Дейкстры ,за счет добавленной эвристики мы боремся с ситуацией , когда Дейкстра из-за жадности ,может долго спускаться по маршруту ,который находится за «мостом» той вершины в которую нам надо, фактически просто может обойти не нужную нам компоненту графа. Мы берем стандартную эвристику евклидова расстояния от текущей вершины к цели , теперь мы выбираем не самое легкое ребро , а самое легкое из вес ребра + эвристика.



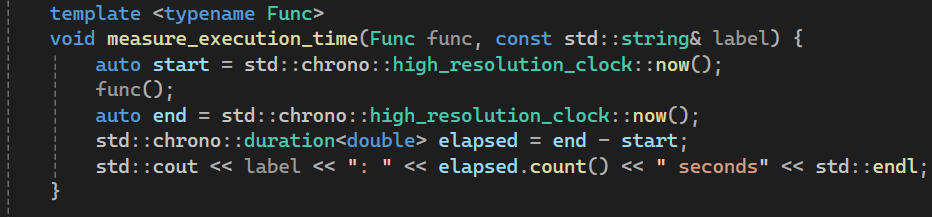


1. Экспериментальная часть

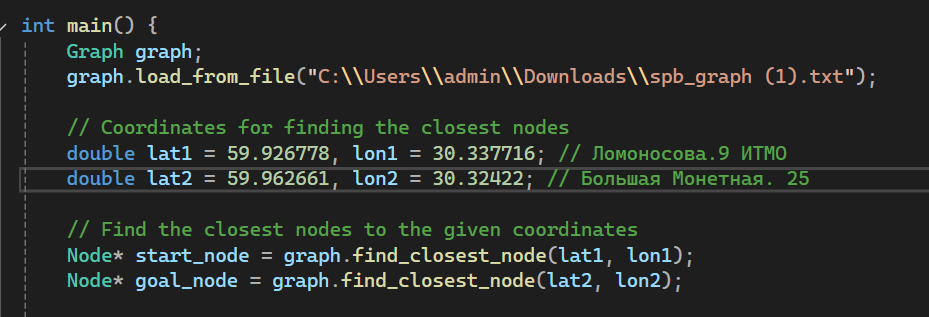
Функция для подсчета количества ребер и выводу найденного пути. В файле программы она представлена в закомментированном виде ,потому что она слишком увеличивает время выполнения алгоритмов.

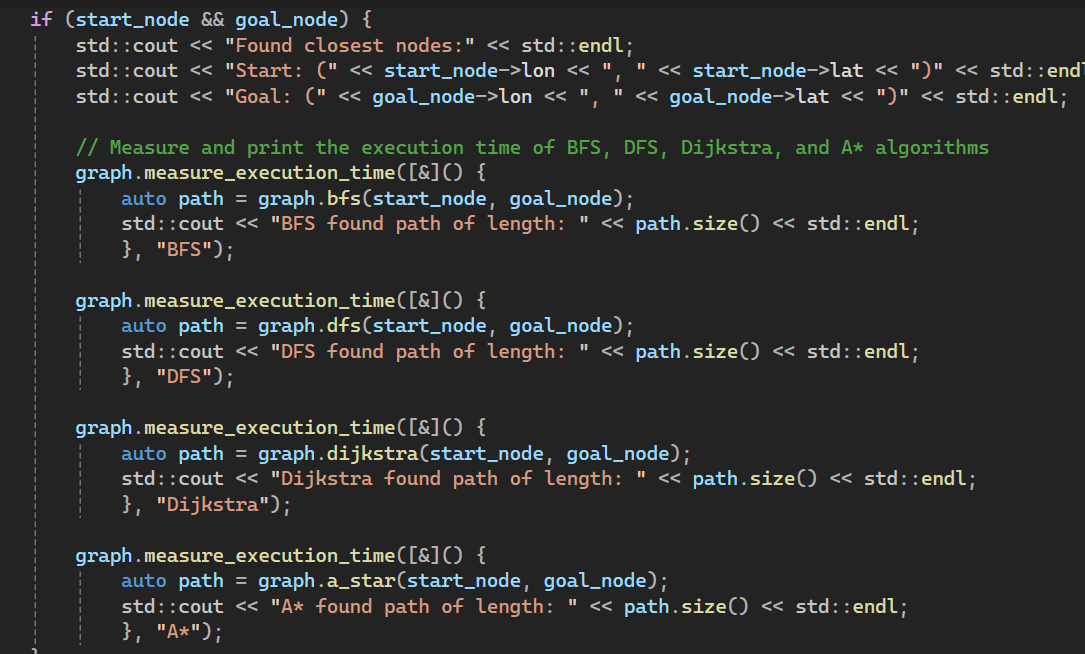


Функция для подсчета времени выполнения алгоритмов

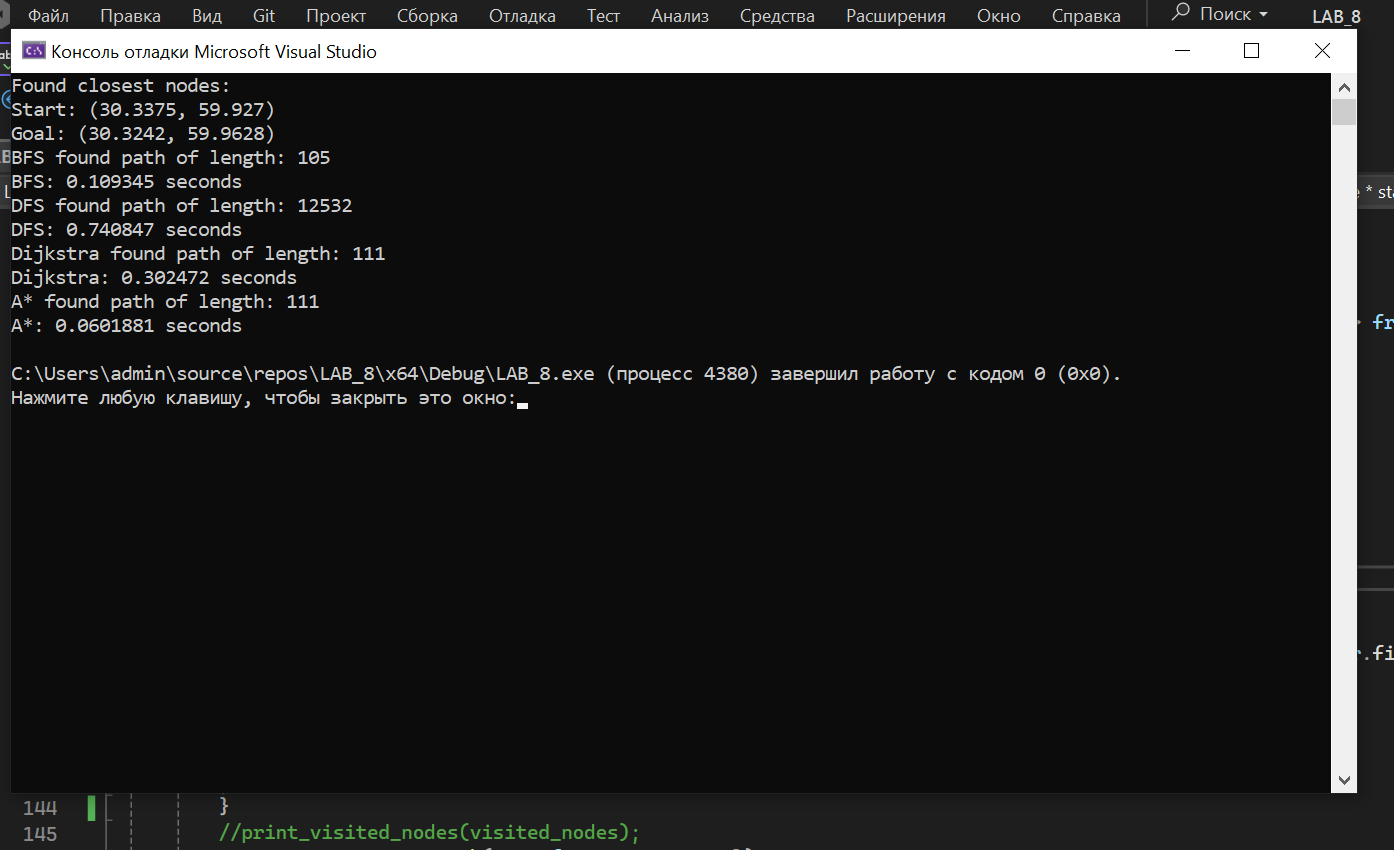


Основная функция вывода





Сначала мы находим в графе ближайшие к заданным вершины , а затем запускаем алгоритмы по очереди. Вывод представлен:



Самый лучший результат показал алгоритм А\*. В прочем подтверждая теоретические сведения.

1. Заключение

Анализируя консольный выход программы , мы можем сделать несколько выводов о структуре графа и эффективности работы алгоритмов. Заметим что кратчайший путь при поиске в глубину содержит в себе наибольшее количество ребер и занимает самое большое количество времени ,следовательно данные вершины находятся на одном ярусе (если провести мысленный эксперимент по подвешиванию графа за стартовую вершину). Также мы видим подтверждение теоретической оценки того ,что лучшим по времени является алгоритм А\* с правильно выбранной эвристикой, что интересно ,что по времени выполнения он обогнал поиск в ширину , хотя тот содержит меньшее количество ребер и не учитывает вес ребер, а учитывает только их количество и это могло сыграть роль.