**Документация**

**Техническое задание**

Введение

Имеется карта минского метрополитена:



Требования

Известно: время движения поезда между соседними станциями, расписание (время начала маршрута). Считаем, что на каждой линии всегда по одному поезду в каждом направлении, как только поезд доходит до конечной станции, он разворачивается и идет обратно.

Нужно написать приложение на Visual Studio (платформу можно выбрать любую), желательно, с использованием объектно-ориентированного подхода, которое выдавало бы самый быстрый маршрут : из любой точки А в точку Б в момент времени С (А, Б, С задаются пользователем).

Для упрощения и сокращения времени на конфигурирование – количество станций можно сильно сократить, линии оставить 3 – важен алгоритм и принцип.

Решение прислать в виде исполняемого файла (выложить в облако) и исходных кодов.   
Алгоритм прислать отдельно в Word с описанием.

Срок

3-5 дней

**Алгоритм**

Базовый алгоритм

Используется алгоритм Дейкстры, ссылку привожу.

https://ru.wikipedia.org/wiki/Алгоритм\_Дейкстры

В статье по ссылке лучше всего смотреть Пример в Неформальном объяснении. Здесь не описываю своими словами, потому что в статье описано очень хорошо и с иллюстрациями, нет смысла копировать.

Поправки к базовому алгоритму

Задача проекта отличается от задачи, решаемой этим алгоритмом, следующим:

1. Алгоритм не предусматривает, что при движении из одной точки в другую возможно придется сначала дождаться транспорта (расписание поездов), причем время ожидания будет зависеть от направления.
2. Алгоритм направлен на нахождение кратчайшего (а значит и самого быстрого, учитывая, что расписания движения нет) пути из заданной точки во все остальные точки сети дорог. А в текущей задаче нужно не до всех, а только до заданной точки назначения.

Я изменила алгоритм так:

1. Время, которое надо ждать в точке A перед тем, как проехать из нее в точку B (соседнюю), я прибавляю к расстоянию из точки А в точку B. При этом, чтобы действовать в одной и той же размерности, расстояние я измеряю в минутах, за которые поезд проедет из A в B. Считаю, что скорость поезда в каждой ветке я знаю заранее. Время, которое надо ждать в точке A, перед тем, как проехать из нее в точку B,считаю ненулевым и высчитываю в двух случаях:
   1. если A – перекресток, и B находится в другой ветке – не в той, по которой я двигалась прежде, чем попасть в А;
   2. если А – точка отправки;

Нахожу это время по таким константным заранее заданным параметрам:

1. за какое время поезд делает круг в той линии метро, где находится AB,
2. на каком расстоянии (напоминаю, оно в минутах) А находится от первой точки линии,
3. во сколько начинает двигаться поезд в этой линии – считаю для всех одинаково, в 0:00.
4. Базовый алгоритм подразумевает прекращение движения, когда все точки пройдены (всем точкам присвоены веса, которые суть есть расстояния этих точек до исходной). Мой алгоритм, подразумевает прекращение движения уже тогда, когда веса всех рассматриваемых в данный момент точек больше, чем вес точки назначения. Объясняю так: если у точки вес больше, чем вес точки назначения, значит даже если из нее мы и придем в точку назначения, то с еще большим весом, а значит новый маршрут для точки назначения мы при этом не утвердим.

**Объяснение дополнительное:** если точка отправки – корень дерева, то базовый алгоритм пройдет все уровни дерева – до листьев, а мой алгоритм пройдет до того уровня дерева, на котором все узлы имеют вес больше, чем точка назначения. В отличие от дерева, здесь нет строгого направления, и листья могут соединяться (неориентированный граф), но это не влияет.

Дальнейшее применение и модификация алгоритма

Поскольку между пунктом отправки и пунктом назначения узлов может быть очень много, задача может занять много времени. Это можно будет решить так.

1. Нецелесообразно обходить абсолютно все. Например, это не карта метро, а карта автомобильных дорог. Тогда можно присваивать узлам разный ранг, и сначала находить ближайшие к точке отправке и точке назначения узлы первого (высшего) ранга, и путь до них. Затем находить кратчайший путь, между этими узлами учитывая только узлы первого ранга.
2. Некоторые наиболее важные и часто используемые пути можно хранить в БД. Причем сохранять во время выполнения программы для какого-то пользователя. Сортировать при этом надо так, чтобы сначала шли наиболее часто используемые.

**Архитектура программы**

Файловая структура

Весь исходный код находится в одном файле **route\_task.cpp**. Это сделано из соображений, что задача небольшая, я сэкономлю время на переключении между файлами, и постороннему в коде будет быстрее разобраться. Хотя на практике положено отдельный файл под каждый класс.

Классы и контейнеры

1. Класс **Станция (Station)** содержит ее имя, код, ссылки на смежные станции **Ветви (\_branches)**, все Пункты расписания, в которых она участвует.
2. Если станция – перекресток, будет больше одного пункта – по одному для каждой ветки. **Пункты (\_schedule\_item)** организованы в map. Пункт содержит только время, через которое станцию пройдет поезд в данной ветке за первый рейс в прямом направлении. Время в обратном направлении можно рассчитать, зная длительность одного круга поезда (задаю заранее). Время, ближайшее к моменту времени, в котором находится пассажир – тоже можно рассчитать.
3. Рассчитывает это время, а также хранит длительности цикла класс **Расписание (Schedule)**.
4. Совокупность Станций (они в паре со своими кодами хранятся в объекте map, чтобы автоматизировать и сделать эффективной сортировку), а также класс Расписания объединяет класс-контейнер **Поисковик (Researcher)**. В нем же реализована и вся логика поиска (алгоритм).
5. Все действия по поиску пути происходят в функциях search(), которая вызывается рекурсивно для каждого узла, и findPointDistance() внутри нее, которая вызывается для каждой смежной ветки (\_**branches**) узла. Для каждого узла, который мы обошли функцией findPointDistance(), мы сохраняем маршрут (**current\_rout**), а также его длину (вес – **current\_distance**). После окончания рассчета выводим маршрут того узла, который есть пункт назначения.
6. Некоторые константы, структуры и функции являются глобальными, однако ограничены **пространством имен (route\_task)**.

**Как читать код:** изучить все объекты в том порядке, в котором я о них рассказала выше.

**Тестирование**

**Тестовый комплект 1**

|  |  |  |  |
| --- | --- | --- | --- |
| **Система**: OC Windows 10 pro, Microsoft Visual Studio Community 2019, версия 16.0.3  **Предусловие**: запустить приложение из каталога x64/Debug, действовать по инструкциям командной строки.  **Доп. комментарий:** критерий правильности ответа: из всех путей выбран тот, в котором минимальное количество узлов, если цикл движения поездов одинаков. Если по какой-либо из ветвей цикл движения поезда гораздо больше (скажем, в 3 раза), то в большинстве случаев будет выбран тот, который включает быстродействующие ветви (если выбор есть).  для удобства: открыть одновременно на экране консоль приложения, картинку и тестовые кейсы. | | | |
| Позитивные тесты | | | |
| **Тест-кейс 0** – Тривиальные условия – время отправки 0 (минут с момента 0:00, т.е. соответствует 0:00), маршрут в пределах одной ветви, не проходит через пересечения | | | |
| **Действия** | **Ожидаемый результат** | **Рез.** | **Приоритет** |
| П. отправки: 110, назначения: 110 | «Вы уже приехали, попробуете другой маршрут?» | + | 1 |
| П. отправки: 109, назначения: 110 | Кратчайший путь: 109->110 | + | 1 |
| П. отправки: 112, назначения: 110 | Кратчайший путь: 112->111->110 | + | 1 |
| П. отправки: 219, назначения: 222 | Кратчайший путь: 219->220->221->222 | + | 1 |
| **Тест-кейс 1** – время отправки 0, маршрут в пределах одной ветви, проходит через пересечения | | | |
| П. отправки: 114, назначения: 117 | Кратчайший путь: 114 -> 115 -> 216 -> 117 | + | 1 |
| П. отправки: 112, назначения: 115 | Кратчайший путь: 112 -> 113 -> 114 -> 115 | + | 1 |
| П. отправки: 115, назначения: 215 | Кратчайший путь: 115 -> 216 -> 215 | + | 1 |
| П. отправки: 217, назначения: 222 | Кратчайший путь: 217 –> 218 -> 219 -> 220 -> 221 -> 222 | ?  (Не успела проверить) | 1 |
| **Тест-кейс 2** – время отправки 0, проходит через пересечения, меняет ветвь | | | |
| П. отправки: 113, назначения: 313 | Кратчайший путь: 113 -> 114 -> 115 -> 313 | + | 1 |
| П. отправки: 313, назначения: 213 | Кратчайший путь: 313 -> 115 -> 315 -> 218 -> 217 -> 216 -> 215 -> 214  (не самый короткий на карте, потому что для самого короткого надо пересесть на голубую линию, а по ней поезд только недавно прошел в момент пересадки) | + | 1 |
| П. отправки: 114, назначения: 215 | Кратчайший путь: 114 –> 115 -> 216 -> 215 -> 214 | ? | 1 |
| **Тест-кейс 3** – время отправки 1000, проходит через пересечения, меняет ветвь | | | |
| П. отправки: 113, назначения: 313 | Кратчайший путь: 113 -> 114 -> 115 -> 313 | ? | 1 |
| П. отправки: 313, назначения: 213 | Кратчайший путь: 313 -> 115 -> 216 -> 215 -> 214 -> 213 | ? | 1 |
| П. отправки: 114, назначения: 215 | Кратчайший путь: 114 –> 115 -> 216 -> 215 -> 214 | ? | 1 |
| **Тест-кейс 4** – в голубой ветви цикл движения поезда в 4 раза дольше(160000), чем в зеленой и красной (40000) (можно сделать только **в коде, строка 417**) и красной. Маршрут выбран так, чтобы был выбор – проехать одну остановку по голубой или несколько по красной и(или) зеленой | | | |
| время отправки 40000  П. отправки: 217, назначения: 115 | 217 -> 218 -> 315 -> 115  (выбран длинный путь по красной и зеленой линии вместо короткого по синей линии, т.к. долгая пересадка) | + | 2 |
| время отправки 40000  П. отправки: 216, назначения: 313 | 216 -> 217 -> 218 -> 315 -> 115 -> 313  (выбран длинный путь по красной и зеленой линии вместо короткого, т.к. долгая пересадка) | + | 2 |
| время отправки 40000  П. отправки: 114, назначения: 117 | 114 -> 115 -> 216 -> 117  (выбран короткий путь по синей линии, а не длинный по красной и зеленой, т.к. пересадка не нужна – стартуем с синей) |  | 2 |
| время отправки 40000  П. отправки: 117, назначения: 218 | 117 -> 216 -> 217 -> 218  (выбран короткий путь по красной и зеленой, т.к. пересадка не требует много времени, и она понадобится в любом случае) |  | 2 |
| Негативные тесты | | | |
| **Тест-кейс 4 –** проверка поведения системы при некорректном вводе с клавиатуры | | | |
| По каждому из параметров попытаться ввести нечисловое значение. По остальным при этом вводить числовое | Система не позволяет вводить дальше, выбрасывая сообщение об ошибке и предлагая ввести значение еще раз. | 4 | 4 |
| Попытаться ввести несуществующие коды станций | Сообщение об ошибке, предложено ввести все элементы заново |  | 4 |
| **Тест-кейс 5 – …** | | | |
|  |  |  |  |

**Инструкция пользователя**

См. предусловие к тестовому комплекту 1.