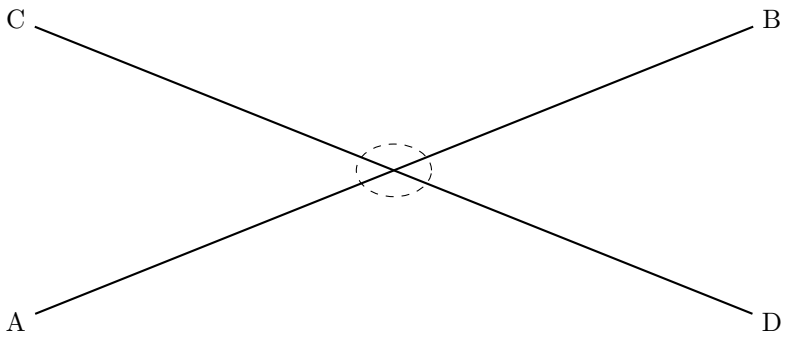
1 Lösungsidee

Die Lösungsidee besteht darin, den Dijkstra Algorithmus derart umzugestalten, dass er den Weg mit der geringsten Zahl an Abbiegungen statt der geringsten Länge sucht. Zuerst wird der kürzeste Weg berechnet, woraufhin dessen Länge und die Anzahl der darin enthaltenen Abbiegungen bestimmt wird. Daraufhin beginnt die Ermittlung des endgültigen Weges.

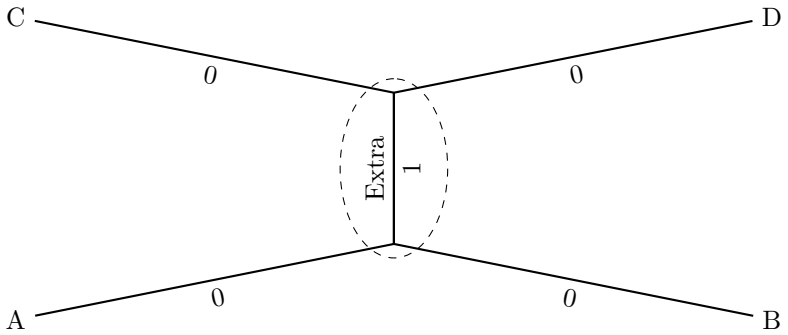
Grundlegend verläuft dieser Prozess wie der normale Dijkstra Algorithmus, jedoch mit ein paar kleinen Unterschieden: Der erste Unterschied besteht darin, dass der Algorithmus alle Wege mit einer Länge größer der Länge des kürzesten Wegs plus die erlaubte Extrastrecke (also +15% oder +30%) eliminiert. Der Hauptunterschied besteht jedoch darin, dass Dijkstra hierbei nicht nach Länge, sondern nach Anzahl an Abbiegungen optimiert.

1.1 Modifizierter Dijkstra im Detail

Um Dijkstra im Hinblick auf die minimale Anzahl an Abbiegungen umzudefinieren, könnte die zugrunde liegende Kostenfunktion anstelle der Länge des Graphen die Existenz von Abbiegungen betrachten, entweder mit einer '1' für eine Abbiegung oder einer '0' für eine Gerade. Dabei fällt auf, dass zur Berechnung der Kosten für eine Kante auch die im Weg vorangehende Kante betrachtet werden muss. Dies bedeutet aber, dass Dijkstra nicht in seiner ursprünglichen Form verwendet werden kann. Es könnte ansonsten der Fall auftreten, dass Knoten des Graphen von einem anderen Weg aus erneut besucht würden, was die Kosten der Kanten verändern würde, die mit dem Knoten verbunden sind.



Um dieses Problem zu umgehen, wird der Graph verändert: Zwar werden Kreuzungen innerhalb des Graphen immer noch als Knoten dargestellt, jedoch gibt es für jede Kreuzung mehrere Knoten, nämlich jeweils einen für jede geradlinige Straße, die in der Kreuzung vorhanden ist. Alle Knoten einer Kreuzung sind dabei verbunden durch Kanten mit Kosten von 1.



Die Priority-Queue des Dijkstra-Algorithmus ist hierbei primär nach Anzahl an Abbiegungen und sekundär nach Länge sortiert.

Der endgültige Algorithmus geht dann wie folgt vor:

1. Initialisiere eine leere Priority-Queue von Wegen.

2. Füge den Startpunkt als Weg ohne Länge und Abbiegungen hinzu.
3. Nehme den obersten Weg aus der Priority-Queue (mit den wenigsten Abbiegungen und kürzester Länge).
4. Ermittle alle Wege, die aus diesem entspringen und noch nicht vorher betrachtet wurden.
5. Speichere alle Wege zu einer Kreuzung (inklusive Richtung, in der diese befahren wird), die besser als die bisherigen Wege sind und füge diese der Priority-Queue hinzu. Dabei werden eingefügte Elemente primär nach Abbiegung und sekundär nach Weglänge sortiert werden.
6. Wiederhole 3-5, bis das Ziel an der Spitze der Priority-Queue ist.

2 Umsetzung

Der Algorithmus wurde in C# 8.0 mit .NET Core 3.1 implementiert. Es wurde als Library OptimizedPriorityQueue[1] verwendet. Diese implementiert eine Priority Queue, die in Dijkstra verwendet wird. Der Code ist in zwei Projekte geteilt; `Afg3Abbiegen.GUI` und `Afg3Abbiegen`.

`Afg3Abbiegen.GUI` kümmert sich um das User-Interface und ruft `Afg3Abbiegen` auf, das den eigentlichen Algorithmus enthält. `Afg3Abbiegen` definiert einige Typen:

Vector2Int stellt einen zweidimensionalen Vektor mit Integerkomponenten dar.

DirectedVector2Int stellt die Kombination einer `Vector2Int` Position und einer genormten Richtung dar.

Street definiert eine Straße zwischen zwei `Vector2Int` Endpunkten.

MapParser ist zuständig für das Einlesen der Beispieldateien.

EnumerableExtensions definiert eine Erweiterungsmethode für `IEnumerable<Vector2Int>`, die die Anzahl an Abbiegung zählt.

Map stellt eine Ansammlung aus Straßen mit Start und Ende dar und enthält den Hauptalgorithmus.

`Map` definiert dabei zwei Hauptmethoden `ShortestPath` und `BilalsPath`. `ShortestPath` ermittlet via Dijkstra den kürzesten Weg zwischen Start und Ende. `BilalsPath` ermittelt via dem obigen Algorithmus den in der Aufgabenstellung beschriebenen Weg.

Zur Verwendung des Programms gibt es ein GUI. Zuerst muss mit “Load Map” eine der Beispieldateien geladen werden, danach kann unter “Factor by which the path may be longer” der Faktor, um den Bilals Weg länger als der kürzeste Weg sein darf eingegeben werden, also 1.15 für +15% oder 1.30 für +30%.

3 Beispiele

TBD

4 Code

TBD

[1] https://github.com/BlueRaja/High-Speed-Priority-Queue-for-C-Sharp