WienerNoder

Projekt von Qi ZHU & Artjom MOISEJEV
12.06.2022

Vorwort: Die Aufgabenstellung besagt, dass der Umstieg zwischen Linien zusätzliche Kosten aufwirft. Da es sich dadurch um ein Netz aus Linien handelt und nicht aus Nodes, war es unlogisch, typische Node-Surch-Algorithmen zu verwenden. Der kürzeste Weg wird fast immer der sein, mit den wenigstens Umstiegen. Es ist daher irrelevant alle Umstiegsmöglichkeiten zu bewerten, wenn man die richtigen Linien-Kombinationen im Vorfeld errechnen kann. Trotzdem haben wir sehr viel Spaß an der Aufgabe gehabt und auch sehr viel gelernt!

Bedienung:

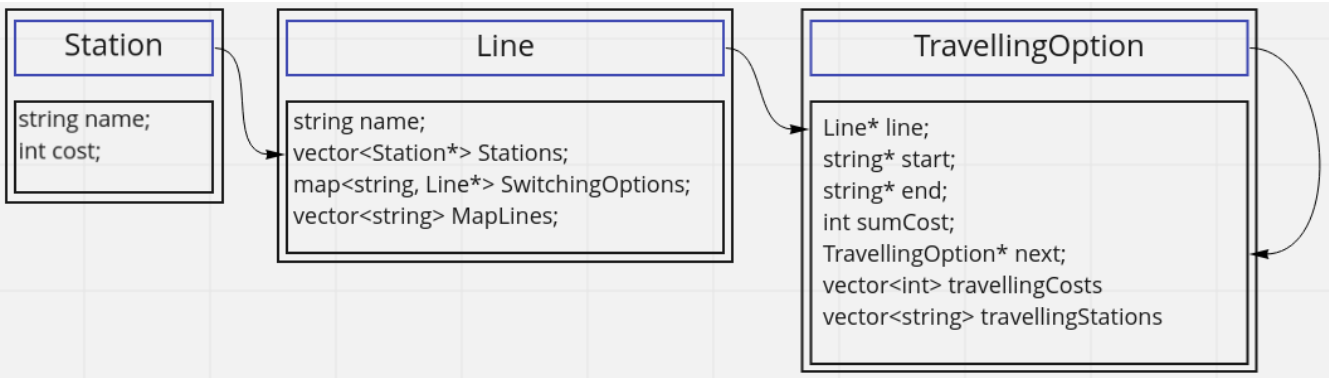
1. Drag & Drop data.txt (oder vor Start, City.hpp Zeile 22 ändern)
2. [StationsName1] / [StationsName2]

x. “q” zum Beenden

Aufbau:

Zuerst werden mehrere Datenstrukturen aufgebaut, aus den Daten der Text-Datei.

Structs:

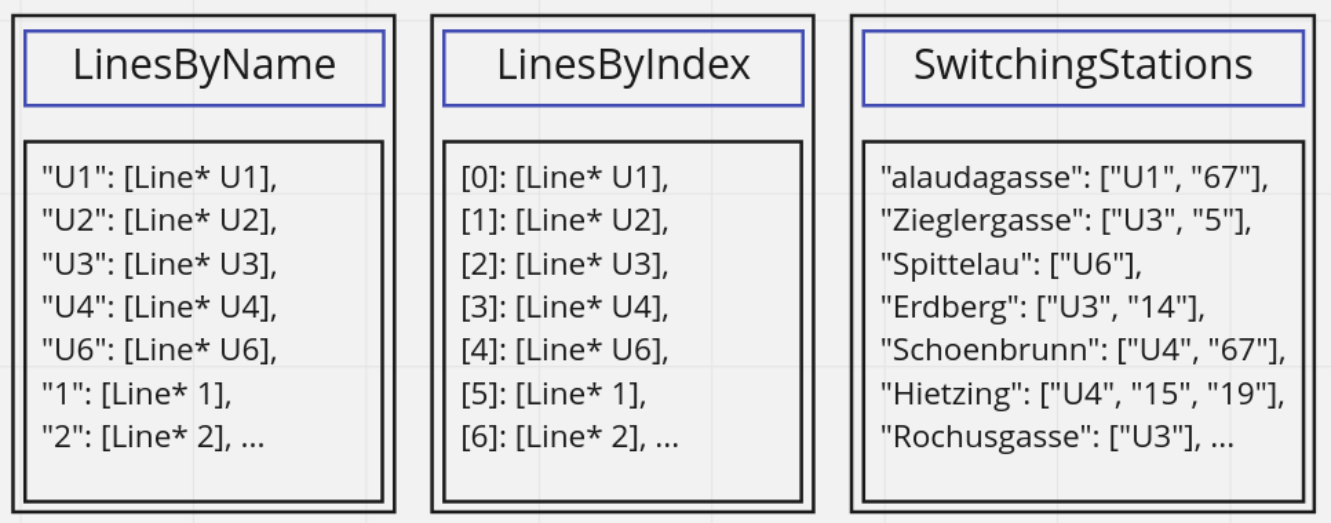


map<string, Line*> SwitchingOptions zeigt auf alle Linien, die sich mit dieser überschneiden. Alle Linien zusammen, bilden einen unendlichen Such-Baum, wobei jede Linie, von ihrer Perspektive aus, die Wurzel des Baumes ist. Jedes level an Tiefe, stellt einen Umstieg von einer, auf eine andere Linie dar.
Tiefe = 4 bedeutet 4x Umsteigen

unordered_map<string,Line*> LinesByName;

vector<Line*> LinesByIndex;

unordered_map<string, vector<string>> SwitchingStations;



unordered_map hat eine Zugriff von O(1)
vector-variante, um durch alle Linien zu iterieren.

Funktionsweise:

Nachdem Start- und End-Station eingegeben wurden, werden mithilfe der SwitchingStations-map die Linien auf den beiden Stationen abgerufen. O(1), da unordered_map

zB: Karlsplatz / Schwedenplatz
=> Karlsplatz: ["U1", "U4", "1", "2"]
=> Schwedenplatz: ["U1", "U2", "U4", "1"]

Dann wird erst gecheckt, ob es direkte Verbindungen gibt und welche:
(U1->U1, U4->U4, 1->1)

Bei direkten Verbindungen, bricht es die Suche ab und läuft nur noch die Linien ab, um die Namen und Kosten zu printen. => Ende

Bei keiner direkten Verbindung, beginnt es eine “recursive-multi-state-function”, die den unendlichen Suchbaum an Linien und deren Umstiegsmöglichkeiten durchsucht. Da die Tiefe der Breitensuchen gleichzeitig die Umstiegsanzahl ist, und da (wie im Vorwort erläutert) je weniger Umstiege, desto besser gilt, arbeitet sich die Funktion nur begrenzt in die Tiefe.

Um zu garantieren, dass ein weiterer Umstieg, als das Minimum, keine kürzere Strecke ergibt, durchläuft die rekursive Funktion immer eine zusätzliche Iteration, nachdem sie die ersten Ergebnisse gefunden hat.

Erst dann werden die Stationen selbst Schritt für Schritt aufgerufen um deren Namen und Kosten zu speichern. Vorher arbeitete das Programm nur mit den Linien.

Nachdem die Stationen abgelaufen und gespeichert wurden, werden die Ergebnisse sortiert und ausgegeben, angefangen bei dem Schnellsten.

Der Algorithmus hängt von der Anzahl der Verbindungen pro Linie und den benötigten Umstiegen.

Jeder Umstieg erhöht den Rechenaufwand ca. quadratisch. (wenn man davon ausgeht, dass jede Linie ca. gleich viele Umstiegsmöglichkeiten hat)

Gesamtaufwand:
mit direkter Verbindung: O(S) | S = Stationen auf der Linie
ohne direkte Verbindung: O(L^U) | L = Linienumstiegsmöglichkeiten, U = Anzahl der Umstiege