Aufgabe 3: Abbiegen

Richard Wohlbold
Team-ID: 00487

29. Dezember 2019

Inhaltsverzeichnis

1 Lösungsidee

Für die Beantwortung von Bilals Frage muss das Straßennetz zunächst als Datenstruktur repräsentiert werden. Daraufhin arbeite ich mit einer modifizierten Version des Dijkstra-Shortest-Path-Algorithmus, um einen Weg zu finden, dessen Länge innerhalb der gegebenen Grenzen liegt und auf dem so wenig wie möglich abgebogen werden muss.

1.1 Darstellung des Straßennetzes

Zwei Abbildungen einfügen

pseudocode

weights for t edges

Straßennetze werden allgemein als gewichtete Graphen, d.h. Mengen von Knoten und Kanten zwischen diesen, repräsentiert. Normalerweise entsprechen dabei Knoten den Kreuzungen und Kanten den Straßen zwischen den Kreuzungen, ein sog. *node-based-graph*. Dabei hat jede Kante ein Gewicht, d.h. einen ihr zugewiesenen numerischen Wert, der meistens der Länge der entsprechenden Straße entspricht.

Wenn jedoch auch das Abbiegen miteinbezogen werden soll, wird die Darstellung des Straßennetzes schwieriger. Oft soll beispielsweise auf einer Route weniger abgebogen werden, weshalb Abbiegevorgängen Gewichte zugeordnet werden, sogenannte *turn costs* [1]. Bei diesen Gewichten stößt die traditionelle Repräsentation der Straßennetze an ihre Grenzen, sodass bei *turn costs* der Ansatz eines *edge-based-graph*s gewählt wird. Dabei wird jede Straße als Knoten repräsentiert. Für jede Möglichkeit, von einer Straße auf eine andere zu gelangen, wird eine Kante zwischen den zwei entsprechenden Knoten hinzugefügt. Falls

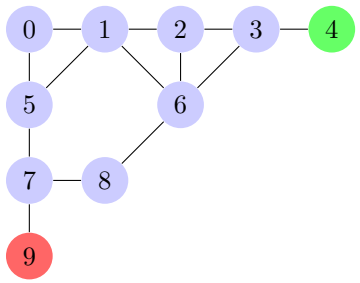

Abbildung 1: Beispiel aus der Aufgabe, als *node-based-graph* repräsentiert.
Knoten 9 ist dabei der Ausgangsknoten und Knoten 4 der Zielknoten

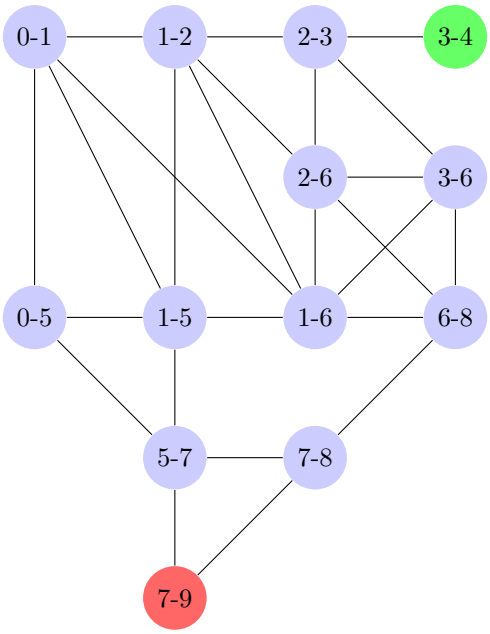

Abbildung 2: Beispiel aus der Aufgabe, als *edge-based-graph* repräsentiert, hier ohne Gewicht

zwischen den Straßen abgebogen werden muss, kann dies entweder als Gewicht der Kante dargestellt werden oder ein einfaches Wahr oder Falsch für jede Kante gespeichert werden. Die Länge jeder Straße muss für einen *edge-based-graph* für jeden Knoten statt für jede Kante gespeichert werden.

1.2 Finden des optimalen Weges

Das Finden des optimalen Weges löse ich mithilfe des Dijkstra-Shortest-Path-Algorithmus. Dieser benutzt eine PriorityQueue, um Kreuzungen geordnet nach ihrer Distanz zum Ausgangspunkt zu „besuchen“. Wird eine Kreuzung "besucht", werden alle Kreuzungen, die von dieser Kreuzung über eine Straße erreicht werden können, der PriorityQueue mit ihrer Distanz vom Ursprung hinzugefügt. Wird die Zielkreuzung gefunden, terminiert der Algorithmus und der kürzeste Weg kann rekonstruiert werden.

Der erste Schritt meines Programms ist es, den Dijkstra-Shortest-Path-Algorithmus auf den in Abschnitt 1.1 beschriebenen *edge-based-graph* anzuwenden. Dadurch wird die kürzeste Distanz vom Start zum Ziel ermittelt. Zusätzlich kann jeweils die Anzahl an Abbiegevorgängen neben der Distanz in die PriorityQueue hinzugefügt werden, damit auch die maximale Anzahl an Abbiegevorgängen n ermittelt werden kann.

Im Folgenden soll nun versucht werden, den kürzesten Weg mit einer gewissen Anzahl an Abbiegevorgängen m zu finden. Anfangs ist $m = n - 1$, falls ein solcher Weg existiert und Bilals Toleranz nicht überschreitet, wird m um jeweils 1 verringert, bis eine der Bedingungen verletzt ist. Dadurch wird der Weg mit der niedrigsten Anzahl an Abbiegevorgängen, der Bilals Anforderungen genügt, gefunden.

Wie auch anfangs verfolgt die modifizierte Version des Dijkstra-Algorithmus die Anzahl der Abbiegevorgänge, die benötigt werden, um auf eine Straße (zu einem Knoten) zu gelangen. Anders als zuvor fügt der Algorithmus der PriorityQueue keine Straßen hinzu, für die $> m$ Mal abgebogen werden muss. Zusätzlich wird es erlaubt, Knoten öfter zu besuchen, sofern bei den neuen Besuchen die Zahl der Abbiegevorgänge auf der Route geringer ist, als vorher.

Somit werden nur Routen gefunden, die maximal m Abbiegevorgänge haben. Auch werden keine Routen ignoriert, die zwar etwas länger sind, aber trotzdem die entsprechende Anzahl an Abbiegevorgängen aufweisen könnten.

2 Umsetzung

Ich habe die Lösungsidee in Python 3 umgesetzt. Umsetzung

3 Beispiele

Beispiele

4 Quellcode (ausschnittsweise)

Quellcode (ausschnitts-weise)

remove

Liste der noch zu erledigenden Punkte

Literatur

[1] Robert Geisberger und Christian Vetter. „Efficient Routing in Road Networks with Turn Costs“. In: *International Symposium on Experimental Algorithms*. 2011. URL: `https://algo2.iti.kit.edu/download/urn_ch.pdf`.