Graphen Protokoll

Lorenz Rentenberger, Samuel Hammerschmidt

Juni 2024

Kantenerstellung

Die Methode `add_edge(self, line, from_station, to_station, cost)` erstellt eine Kante zwischen zwei Stationen. Als erstes, überprüft die Methode, ob die Stationen bereits im Graph enthalten sind. Ist dies nicht der Fall, wird eine neue Station erstellt. Danach wird die Verbindung zwischen den beiden Stationen generiert, die Kosten der Verbindung werden als Gewicht der Kante gesetzt. Die Kante wird dann in die Adjazenzliste der beiden Stationen eingefügt.

Dijkstra-Algorithmus

`dijkstra(self, start, end)` berechnet den kürzesten Weg zwischen zwei Stationen, die vom Benutzer aus einer Textdatei gewählt werden. Zuerst initialisiert die Methode die kürzesten Distanzen als `infinity` für alle Stationen im Graphen und die vorherigen Stationen `previous_stations` werden auf `None` gesetzt. Die Distanz von der Startstation wird auf 0 gesetzt. Danach erstellt die Methode ein Set von nicht-besuchten Stationen, die zunächst alle Stationen des Graphen beinhaltet. Als nächstes betritt die Methode eine Schleife, die so lange läuft, solange noch nicht alle Stationen besucht wurden. In jeder Iteration wird die kürzeste Distanz zur nächsten, unbesuchten Station gewählt - diese Station wird dann aus dem `unvisited` Set entfernt. Sollte die aktuelle Station die Endstation sein, wird der Pfad rekonstruiert, indem von der Zielstation aus rückwärts durch die vorherigen Stationen iteriert wird. Die Gesamtkosten bzw. Fahrzeit des Pfades werden berechnet und zurückgegeben. Wenn die aktuelle Station nicht die Endstation ist, aktualisiert die Methode die kürzesten Distanzen und die vorherigen Stationen für alle Stationen, die von der aktuellen Station erreicht werden können. Sollte jede Station besucht werden, bevor die Endstation erreicht wird, wird `None` und `infinity` zurückgegeben.

Print-Funktion

Die Methode `print_shortest_path(path, distance, start, end)` gibt den kürzesten Pfad zwischen zwei Stationen in der Konsole aus. Die Argumente `path` (Liste der durchlaufenen Stationen) und `distance` (Gesamtfahrzeit /-kosten) werden vom Dijkstra-Algorithmus zurückgegeben. Bei jedem Durchlauf wird die aktuelle Station der `path` Liste ausgegeben. Danach wird die Linie der aktuellen Station gespeichert. Wenn sich die Linie an der nächsten Station ändert, wird der Schritt zum Umsteigen ausgegeben. Am Ende wird die Gesamtfahrzeit bzw. die Gesamtkosten des Pfades ausgegeben.

Aufwandsabschätzung

Operation	**Best Case**	**Worst Case**	**Average Case**
Kürzester Pfad (dijkstra)	$O(VlogV + E)$	$O((V^2)logV)$	$O((V + E)logV)$
Pfad ausdrucken (print_shortest_path)	$O(n)$	$O(n)$	$O(n)$

Tabelle 1: Aufwandsabschätzung für verschiedene Operationen

Experiment

In diesem Experiment haben wir die durchschnittliche Zeit gemessen, die der Dijkstra-Algorithmus benötigt, um den kürzesten Weg zwischen zwei Stationen in verschiedenen Graphen mit verschienden Längen zu finden. Die Ergebnisse sind in Tabelle 2 dargestellt. Die angegebenen Zeiten sind Durchschnittswerte von 10.000 Durchläufen.

Um einen besseren Vergleich zu erhalten haben wir die Anzahl der Stationen variiert. Die Strecke ist immer dieselbe.
Wiener Strecke: $Ottakring \rightarrow KagranerPlatz$
Londoner Strecke: $Bank \rightarrow HighBarnet$

Graph	**Länge 3**	**Länge 7**	**Länge 10**	**Länge 15**	**Länge 20**
Wien (450 Stationen)	0.071 ms	0.106 ms	0.158 ms	0.356 ms	0.547 ms
Wien (90 Stationen, U-Bahn)	0.015 ms	0.018 ms	0.027 ms	0.061 ms	0.086 ms
London (204 Stationen)	0.036 ms	0.096 ms	0.143 ms	0.202 ms	0.229 ms

Tabelle 2: Experiment