**Beschreibung des verwendeten Algorithmus**

Der Algorithmus basiert auf einer modifizierten Version des Dijkstra-Algorithmus, wobei eine Vorrangwarteschlange (Heap) verwendet wird, um die Knoten mit den niedrigsten Kosten zu priorisieren. Der Graph wird aus einer Datei eingelesen und in einer dictionay-basierten Datenstruktur gespeichert. Die Funktion shortest\_path implementiert den modifizierten Dijkstra-Algorithmus, indem sie die Vorrangwarteschlange verwendet, um den aktuellen Knoten (Station) mit den geringsten Kosten zu finden und diesen Knoten zu erkunden. Um das Umsteigen zwischen Linien zu berücksichtigen, wird eine zusätzliche Kostenkomponente, line\_switch\_penalty, hinzugefügt, wenn ein Umstieg erforderlich ist. Der Algorithmus terminiert, sobald das Ziel erreicht ist oder keine weiteren Knoten zu erkunden sind.

**O-Notation:**

**O(|V|log|V|):**

Einfügen und Entfernen der Knoten aus der Prioritätswarteschlange. Im schlimmsten Fall werden alle Knoten zur Warteschlange hinzugefügt und aus der Warteschlange entfernt, was für jeden Knoten eine Zeit von O(log|V|) in Anspruch nimmt.

**O(|E|log|V|):**

Verarbeitung jeder Kante. Im schlimmsten Fall verarbeitet der Algorithmus alle Kanten. Für jede Kante aktualisiert der Algorithmus die Prioritätswarteschlange mit neuen Kosten. Die Aktualisierung der Prioritätswarteschlange kostet O(log|V|) pro Kante.

Daher ergibt sich aus der Kombination dieser Teile die insgesamt Zeitkomplexität von O(|V|log|V| + |E|log|V|).

**Experimente und Messungen zur Laufzeit:**

Um die Laufzeit des Algorithmus zu messen, können verschiedene Experimente mit unterschiedlichen Eingabedateien und U-Bahn-Netzwerken durchgeführt werden. Die Laufzeit kann mithilfe der Python time-Bibliothek gemessen werden, indem die Start- und Endzeit vor und nach der Ausführung des Algorithmus erfasst werden. Dabei sollte beachtet werden, dass die tatsächliche Laufzeit von verschiedenen Faktoren abhängt, wie z.B. der Größe des Netzwerks, der Anzahl der Umstiege und der Hardware des Computers, auf dem der Code ausgeführt wird.

**Quellen:**

Dijkstra's Algorithm:

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms. MIT press. Kapitel 24.3: Dijkstra's algorithm. (Link: <https://mitpress.mit.edu/books/introduction-algorithms-third-edition>)

Dijkstra's Algorithm auf Wikipedia: <https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm>

Priority Queues und Heaps:

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms. MIT press. Kapitel 6: Heapsort. (Link: <https://mitpress.mit.edu/books/introduction-algorithms-third-edition>)

Heap Data Structure auf Wikipedia: <https://en.wikipedia.org/wiki/Heap_(data_structure)>

Big-O Cheat Sheet

<https://static.packt-cdn.com/downloads/4874OS_Appendix_Big_O_Cheat_Sheet.pdf>

Laufzeitmessung und Performance-Analyse in Python:

Python time Library: <https://docs.python.org/3/library/time.html>

Profiling und Optimierung in Python: <https://realpython.com/python-timer/>