**Protokoll**

1.Verwendeter Algorithmus:

Wir verwenden den Dijkstra-Algorithmus, der ein gängiger Algorithmus zur Lösung des Problems des kürzesten Pfades in einem Graphen ist. Dieser Algorithmus beginnt an einem Startknoten und arbeitet sich schrittweise durch den Graphen, indem er immer den Knoten mit den geringsten Gesamtkosten (vom Startknoten aus) auswählt und zu einem bereits besuchten Pfad hinzufügt. Sobald der Zielknoten erreicht wird, gibt der Algorithmus den gefundenen Pfad zurück.

Wir haben diesen Algorithmus leicht modifiziert, um eine zusätzliche "Strafe" für das Wechseln von Linien einzuführen. Das U-Bahn-Netzproblem stellt eine Situation dar, in der das Wechseln von Linien oft mit zusätzlichem Aufwand und Zeit verbunden ist. Daher ist es sinnvoll, Linienwechsel zu minimieren, und der Algorithmus berücksichtigt dies, indem er die Kosten für das Wechseln von Linien zu den Gesamtkosten hinzufügt.

2.Aufwand des Algorithmus (O-Notation):

Die Zeitkomplexität des Dijkstra-Algorithmus ist O((V+E) log V), wobei V die Anzahl der Knoten (Stationen) und E die Anzahl der Kanten (Verbindungen zwischen Stationen) im Graphen ist. Diese Komplexität ergibt sich aus zwei Hauptkomponenten:

1. O(|V|log|V|): Jeder der |V| Knoten kann maximal einmal in die Prioritätswarteschlange eingefügt und entfernt werden. Da die Prioritätswarteschlange als Binär-Heap implementiert ist, kostet das Einfügen und Entfernen eines Elements O(log|V|). Daher ergibt die Behandlung aller Knoten insgesamt O(|V|log|V|).
2. O(|E|log|V|): Für jede der |E| Kanten kann die Prioritätswarteschlange aktualisiert werden, wenn die Kante verarbeitet wird. Da eine Aktualisierung in der Prioritätswarteschlange ebenfalls O(log|V|) kostet, resultiert die Behandlung aller Kanten in O(|E|log|V|).

Die Summe dieser beiden Komponenten ergibt die Gesamtzeitkomplexität O((V+E) log V).

Die Raumkomplexität ist O(V), da wir Informationen für jeden Knoten in der Prioritätswarteschlange und in der Menge der besuchten Knoten speichern müssen.

Die Effizienz des Algorithmus hängt stark von der Struktur des U-Bahn-Netzes ab. Bei einem dicht verbundenen Netz könnte der Algorithmus ineffizient sein, während er bei einem weniger dicht verbundenen Netz effizienter sein könnte.

**Quellen:**

Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2009). Introduction to Algorithms (Third Edition).

Sedgewick, Robert; Wayne, Kevin. "Algorithms, Part II" - Coursera Kurs. Verfügbar auf: <https://www.coursera.org/learn/algorithms-part2>

"Graph Data Structures" - GeeksforGeeks. Verfügbar auf: <https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/>

3.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.

Nützliche Seiten:

https://www.guru99.com/python-regular-expressions-complete-tutorial.html

https://www.w3schools.com/python/ref\_string\_strip.asp

<https://www.geeksforgeeks.org/defaultdict-in-python/>

<https://docs.python.org/3/library/heapq.html>

https://docs.python.org/3/library/time.html