# Protokoll GRAPHEN

Matthias Kemmer, Li Wen Wang

Das Programm wurde in Python implementiert. Die CSV-Daten der Wiener Verkehrslinien werden eingelesen und als Graph gespeichert. Die Stationen bilden dabei die Konten des Graph und die Verbindungen und Kosten bilden die gewichtete Kanten.

## Beschreibung des Algorithmus (Dijkstra)

Der Algorithmus sucht den kürzesten Pfad zwischen zwei Stationen. Der implementierte Algorithmus basiert auf einer modifizierten Version von Dijkstra's Algorithmus und verwendet eine **Prioritätswarteschlange** (**Min-Heap**), um die Station mit der geringsten Entfernung effizient zu finden.

Im ersten Schritt findet eine **Initialisierung der Distanzen** zu allen anderen Knoten im Graphen mit unendlich statt. Der Startknoten zu sich selbst hat Distanz 0. Der Algorithmus verwendet auch Hilfsstrukturen, wie eine Vorgängerliste und eine Liste für die Anzahl der Linienwechsel.

Der Algorithmus durchläuft eine Schleife, wo schrittweise immer der **Knoten mit der geringsten Distanz** gesucht und als besucht markiert. Für jeden Nachbarn dieses Knotens werden die Distanz und die Anzahl der Linienwechsel aktualisiert und überprüft, ob ein kürzerer Weg gefunden wird. Die Aktualisierung der Distanz und der Linienwechselkosten erfolgt durch Hinzufügen der Kosten des aktuellen Knotens und der Kantenkosten zwischen den Knoten. Wenn die Aktualisierung erfolgt, wird der Nachbarknoten in die Prioritätswarteschlange eingefügt.

Der Algorithmus endet, wenn entweder alle Knoten besucht wurden oder der Zielknoten erreicht wurde. Anschließend wird der **kürzeste Pfad durch Rückverfolgung der Vorgängerliste** von Zielknoten zu Startknoten erstellt.

Die Implementierung verwendet auch eine **Zeitmessung**, um die Laufzeit des Algorithmus zu erfassen. Die Messung wurde in den Funktionen find_path und find_path1 umgesetzt, wobei find_path1 die optimierte Version ist, die die Prioritätswarteschlange verwendet (Experimente bezüglich der Zeiteffizienz).

## Laufzeitanalyse

V = Stationen/Knoten
E = Verbindungen/Kanten

**Time-Complexity (ohne heapq):**
$O((V+E)\cdot V)$
lineare Suche: $O(V)$

**Time-Complexity (mit heapq):**
$O((V+E)\cdot \log(V))$
heapq-Operationen(push&pop): $O(\log(V))$