FACULTY OF ENGINEERING
COMPUTER ENGINEERING DEPARTMENT
Tınaztepe Yerleşkesi, Buca-Kaynaklar, Dokuz Eylül Üniversitesi, İZMİR, TÜRKİYE
04.01.2024
CME 2201 Homework Report: Journey Planner for Paris Metro
Prepared by: Kaan YILMAZ – İrem TEKİN

1- **Introduction**: The Paris Metro stands as one of the busiest and extensive metro systems globally, connecting millions of passengers daily across its vast network of stations and lines. As of May 2022, it comprises 16 lines with a total of 308 stations, making it an intricate transportation infrastructure. To facilitate efficient journey planning for commuters, a specialized electronic search engine, known as a Journey Planner, is essential.This assignment delves into the development of an algorithm designed to find the optimal journey between two specified metro stations within the Paris Metro system. The algorithm considers two optimization criteria: minimizing the number of stops and minimizing the overall travel time. The former involves equal edge weights in the graph, while the latter utilizes time intervals between consecutive stops. The representation of each metro station as a node and each line as a directed edge forms a transportation graph, creating a structured framework for journey planning.

2- **Project Description**: The Paris Metro Journey Planner focuses on providing commuters with optimal routes within the metro system, considering key criteria such as minimizing stops or travel time. The algorithm employs a directed graph representation of the metro system, incorporating walk-distance edges to ensure seamless transfers between stations. To enhance the user experience, the algorithm limits transfers to direct routes or those involving up to two transfers. This restriction aims to simplify journeys and streamline the planning process. Additionally, the system offers efficient suggestions by presenting users with multiple alternative paths, granting them flexibility and the ability to choose the route that best suits their preferences and constraints.

2.1.1- getBreathFirstTraversal: This algorithm, known as Breadth-First Traversal, is designed to explore and discover the most efficient path between a specified starting point and a destination within a graph or tree structure. It initiates by resetting the nodes to an unvisited state, then marks the starting node as visited. A queue is employed to keep track of the order of visitation, with the starting node enqueued. The algorithm continues to explore neighboring nodes in a broad manner, enqueuing unvisited neighbors, until either the destination node is found or there are no more nodes to visit. The resulting queue encapsulates the order in which the nodes were visited, providing insight into the optimal path. This approach aligns with the principles of Breadth-First Search, ensuring a systematic and comprehensive exploration of the available nodes.

2.1.2- getDepthFirstTraversal: This code implements a Depth-First Traversal algorithm to explore a graph or tree from a specified origin node. Using a stack to manage the traversal order, the algorithm systematically visits nodes in a depth-first manner, marking them as visited. It proceeds by popping nodes from the stack, exploring neighbors, and enqueuing the node's label. This process continues until all reachable nodes are visited, resulting in a queue that reflects the traversal order. The algorithm is efficient for scenarios where a comprehensive exploration of paths is required.

2.2.1- getShortesPath: This code implements an iterative Shortest Path algorithm to find the most efficient route between two specified stations in a graph. Using a breadth-first search approach, the algorithm iteratively explores the network, marking visited stations and calculating costs until the destination is reached. It reconstructs the shortest path by backtracking through predecessors. The output includes the total station count and a printed representation of the shortest path. This iterative process ensures a comprehensive search for optimal routes in dynamic scenarios.

2.2.2 getCheapestPath: This code implements a Cheapest Path algorithm to find the most cost-effective route between two specified stations in a graph. Using a priority queue for optimal cost consideration, the algorithm iteratively explores the network, marking visited stations and calculating cumulative costs until the destination is reached. The resulting path and total cost are then printed. This algorithm is effective for scenarios where minimizing cost is a priority.

3- **Algorithms and Strategies:** In the realm of complex networks, the utilization of effective algorithms and strategies becomes paramount for optimizing pathfinding. The title "Algorithms and Strategies: Optimizing Pathfinding in Complex Networks" encapsulates a multifaceted exploration into the methodologies employed to navigate intricate structures efficiently. In this context, algorithms serve as the guiding principles, and strategies act as the tactical approaches, both working in tandem to address the challenges posed by networks characterized by interconnectivity and complexity. Whether seeking the shortest, most cost-effective, or diverse paths, the development and implementation of sophisticated algorithms and strategies become indispensable tools for achieving optimal navigation within these intricate systems. This exploration delves into the intricacies of pathfinding, shedding light on the innovative approaches that propel us toward efficient traversal and decision-making in the dynamic landscape of interconnected networks.

4- **Conclusion:** In summary, the presented algorithms for journey planning in the Paris Metro system showcase the significance of algorithmic efficiency in optimizing routes. The Breadth-First Traversal and Depth-First Traversal algorithms systematically explore the metro network, providing users with comprehensive and flexible journey suggestions. The Shortest Path algorithm, utilizing breadth-first search iteratively, reveals the most efficient routes by considering the minimum number of stops. Lastly, the Cheapest Path algorithm employs a priority queue to prioritize cost-effective routes, contributing to a diverse set of journey planning tools.These algorithms collectively demonstrate the power of algorithmic strategies in enhancing user experience and efficiency within complex transportation networks. As technology advances, these foundational algorithms pave the way for continued innovation and the development of more sophisticated journey planning solutions.

04.01.2024 Kaan Yılmaz – İrem Tekin
2020510072 2021510062