**Indian Railway Connection Solver:**

In this project, the goal is to find the best train connections between any two stations using different cost functions, based on a dataset of Indian railway schedules.

I have chosen to treat this as a graph search problem, where each station and train connection forms a node and edge, respectively.

**Step 1: Defining a Function to Load Data:**

I have created a function that takes the filename (either mini-schedule.csv or schedule.csv) and loaded the data accordingly. This way, I can dynamically switch between datasets based on the Schedule column in the problems.csv.

**Step 2: Data Preprocessing:**

I have cleaned and prepared the data for analysis. This includes:

• Stripping quotes and extra spaces from string fields.

• Converting Arrival time and Departure time to datetime objects to facilitate time calculations.

Next, I have built the graph representation of the railway network.

**Step 3: Building the Graph:**

I have used a graph data structure to model the railway network, where:

• Nodes represent stations.

• Edges represent connections between consecutive stops of a train, annotated with relevant details such as travel time, distance, and potentially costs for various cost functions.

For the initial setup:

1. Creating the graph – I have used networkx, a Python library for constructing and manipulating complex networks.

2. Populating the graph - I have added nodes for each station and edges between stations that are consecutive stops on any train, using data from the mini-schedule or schedule.

So, I have created a graph representation of the train network using the mini-schedule/schedule data. Each edge contains information about the train number, departure time, arrival time, travel time, and the distance between stations. I have made this choice to take advantage of the strengths of graph-based search algorithms.

**Step 4: Implementing Search Algorithms:**

Next, I have implemented search algorithms to find optimal routes between stations based on different cost functions;

I have started by implementing Dijkstra’s algorithm, which is suitable for finding the shortest path in weighted graphs I have adapted it to work with various cost functions.

Dijkstra's Algorithm: Dijkstra's algorithm was chosen to find the shortest paths because it efficiently handles weighted graphs and guarantees the shortest path for graphs with non-negative weights. This is essential for this problem where edge weights represent various costs (stops, travel time, price, arrival time). Dijkstra's algorithm was especially suitable for cost functions based on distance or travel time.

**Step 5: Defining Cost Functions:**

I have adapted the cost function (number of stops, travel time, price, and arrival time) as specified in problems.csv.

I have implemented and switched between these required dynamically recalculating edge weights in the graph based on the selected cost function.

Each cost function required specific handling in the algorithm to ensure the correct metric was minimized.

**Step 6: Handling Time Across Days:**

I have handled train changes. I have ensured there is enough time for train changes by considering a minimum change time.

Train schedules often span multiple days. Thus, the solution needed to correctly account for arrival and departure times that cross midnight. This was managed by adding a day to times that logically followed an earlier time. A significant design challenge was handling cases where train schedules spanned over midnight, requiring an "add a day" logic to accurately calculate arrival times and potential overnight stays.

**Step 7. Path Formatting:**

I have formed the connecting tracks to clearly show the sequence of the adopted train routes. This involved merging consecutive segments of the same train and separating different trains with semicolons.

**Step 8: Solving Problems:**

• For each problem, I have determined the cost function based on the problem's cost function name.

• I have used Dijkstra's algorithm to find the optimal path and its cost.

• I have stored the solutions, including the problem number, connection details, and cost.

**Step 9: Saving Solutions:**

• I have created a DataFrame solutions\_df with the results and save it to a CSV file named solutions.csv.