Assignment 1: AIFA (AI61005)

Group Members:

18EE10025: Yash Kulkarni
18EE10043: Rhitvik Sinha
18EE30021: Pratyush Jaiswal
18EE30029: Nuruddin Jiruwala

Problem chosen for Assignment:

(3) Electric Vehicles

Definition of Problem

We have been provided with a set of constraints / information for each EV, as defined in the problem sheet. Given that information, we came up with an algorithm to route all the vehicles from their respective sources to destinations such that the total time taken for all EVs to finish their movements is minimized.

Implementation Logic

- To record all the information provided, a `class car` was declared, where we `initiate` an object with all of the information provided. Helper functions were declared for the class, namely, `shortest_path` and `update_parameters`.
- Each **event** involves an EV entering / exiting a node. This consumes the battery power of the EV.
- So far, we haven't considered the delays that could potentially occur due to the constraint: "Assume that each city has a single charging station which can charge one EV at a time." *To include that into our considerations,* we define a `schedule` function that decides which car to charge in case of a **competition**.
- We loop through all the **events**, starting at initial time, and resolve the **competition** between EVs for charging points (which cause waits), using this `schedule` function.

Some Terms Used:

- Event

  An event is the arrival (or *departure*) of an EV to (or *from*) a node / city. All events are declared globally and stored in a list named: `global_time_list`. The information contained in an event:
  - the EV (and it's properties)
  - it's arrival / departure information
  - the node at which the EV arrives / departs
  - the time of occurrence of the event.
- Competition

  A competition signifies the time overlap of two or more **events** at a certain node. It is named so, because the constraint on charging stations at a node (*one at a time*) causes EVs to compete against each other for recharge. This competition is resolved using the `schedule` function *(mentioned above, explained below)*.

Explaining the Code

- To record information about the EVs from the user, the following snippet was used:

```
src={}
dest={}
battery_status={}
charging_rate={}
discharging_rate={}
Max_battery={}
avg_speed={}
K = int(input("Number of Cars: "))

for i in range(K):
    src[i]=int(input(f"source node of {i+1}th car: "))-1
    dest[i]=int(input(f"destination node of {i+1}th car: "))-1
    battery_status[i]=float(input(f"initial battery status of {i+1}th car:
"))
    charging_rate[i]=float(input(f"charging rate of {i+1}th car: "))
    discharging_rate[i]=float(input(f"discharging rate of {i+1}th car: "))
    Max_battery[i]=float(input(f"Max battery capacity of {i+1}th car: "))
    avg_speed[i]=float(input(f"average speed of {i+1}th car: "))
```

- Shortest path for every vehicle is obtained by calling the helper function **`shortest_path`**. It is an implementation of **Dijkstra's Algorithm**. The **`shortest_path`** function also calculates and updates the battery status of the EV.

- The following snippet is used to initialize all car objects and store their shortest path information. (Unrelated to minimum time)

```
all_cars = {}
for i in range(K): # K is number of cars, entered by user earlier
    obj = car(i, src[i], dest[i], battery_status[i], charging_rate[i],
              discharging_rate[i], Max_battery[i], avg_speed[i], adj)
    obj.shortest_path()
    all_cars[i] = obj
```

- We now populate the aforementioned **`global_time_list`** with all events, and sort it by time of occurence. Each item in **`global_time_list`** is a list that looks like: `[car object, “arrival” or “departure”, node at which event occurs, time of event]`.

- We go through the while loop working through the **`global_time_list`** items chronologically, resolving **competition** using the **Minimax Algorithm**.