# SOLVING PRIZE-COLLECTING TRAVELING SALESMAN PROBLEM USING REINFORCEMENT LEARNING

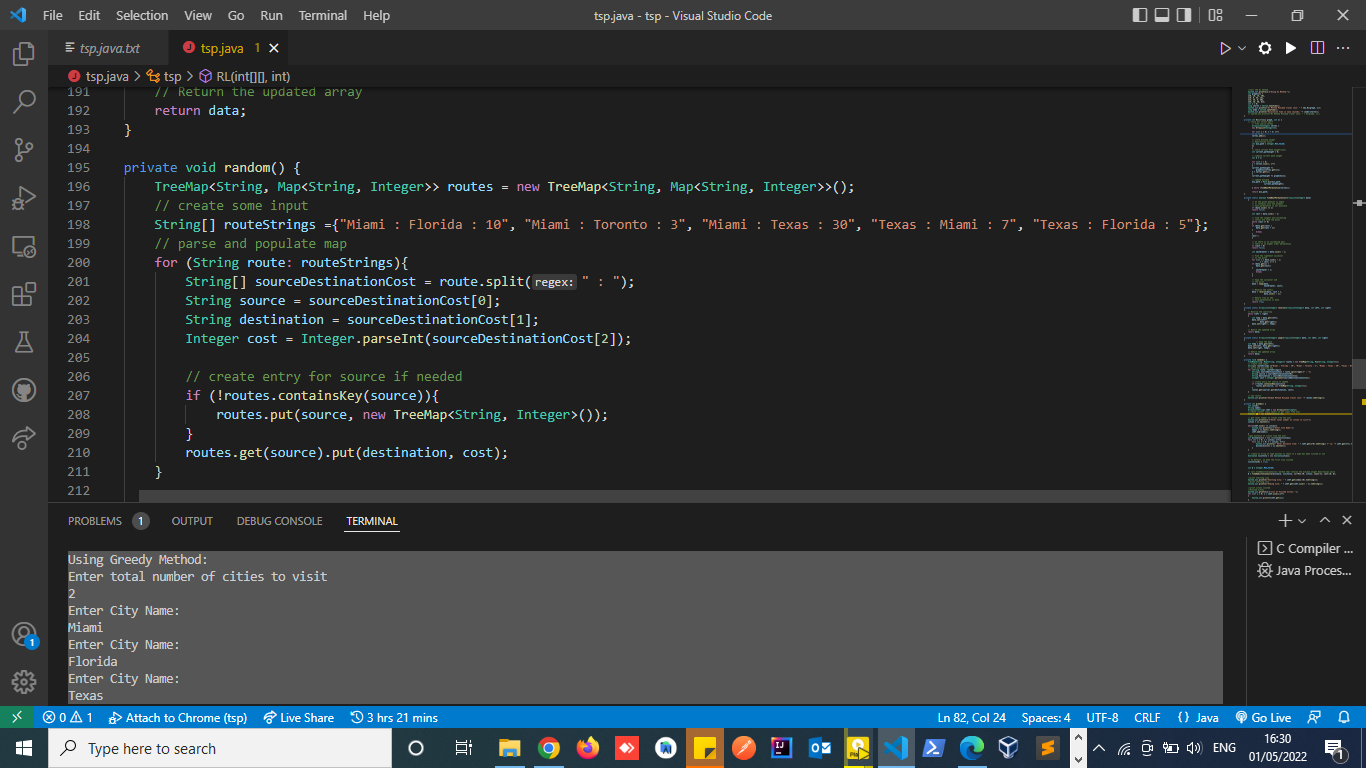
**How to compile and run the program**

1. Extract the zip file to a specific folder say Desktop
2. Open CMD in administrator mode
3. In cmd Navigate to the folder i.e cd C:\Users\{User Name}\Desktop\tsp
4. Run javac tsp.java – Once this runs,
5. Run java tsp
6. Input the required info,

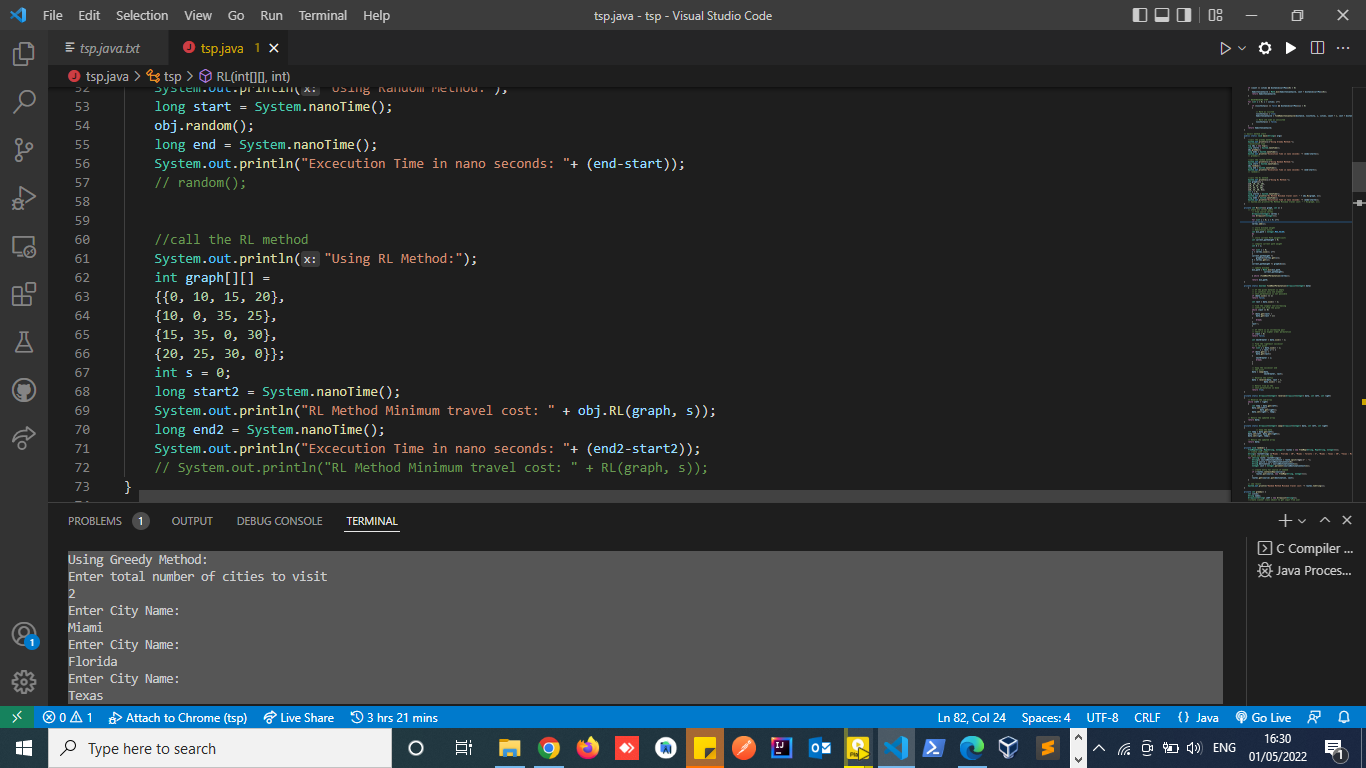
**How to modify the Action Selection Rule to incorporate the prizes**

Each method uses distinct data sets to calculate travel costs between towns. On running a user is required to enter city names and distances.

For the RL and Random methods, these values can be edited in the specific functions i.e

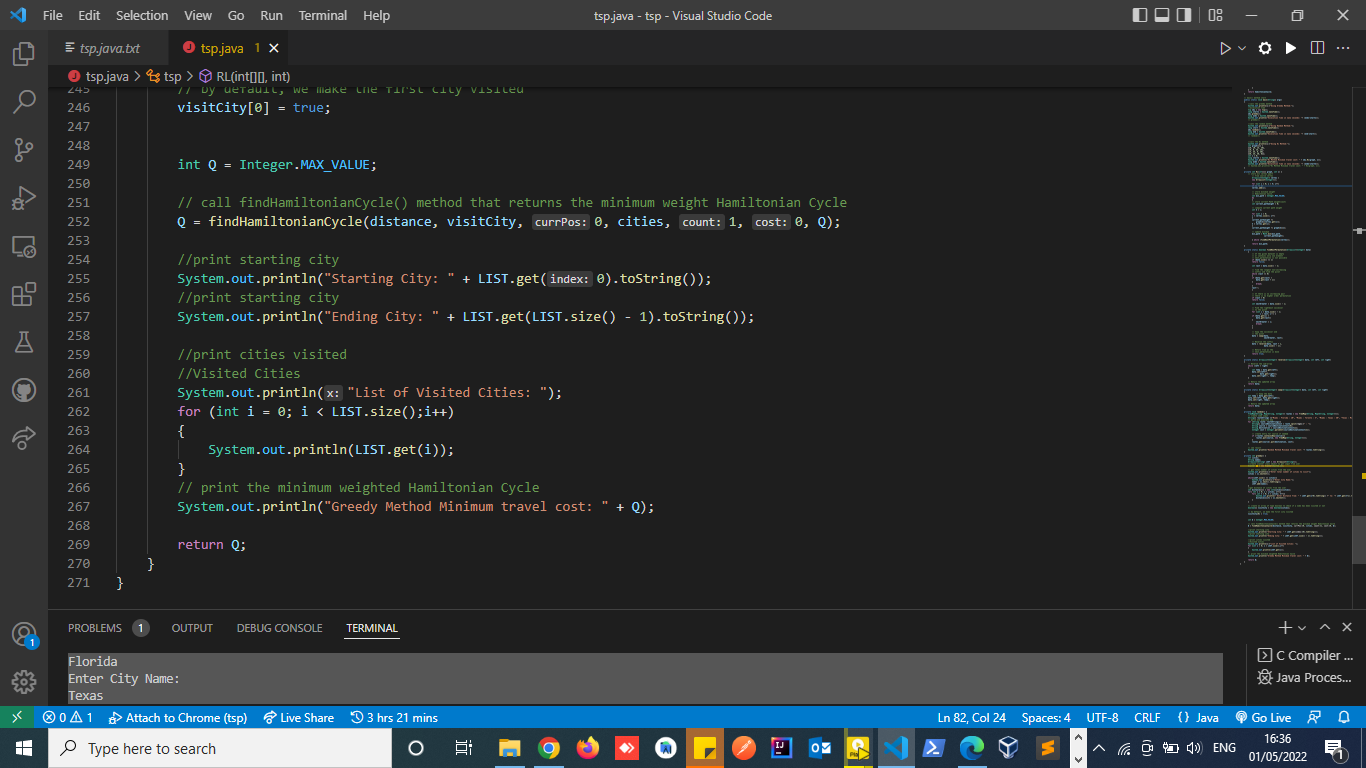


The RL method uses a graph to do cost calculations, this can be edited from this method.



**How to modify the Algo. 1 to incorporate the prizes**

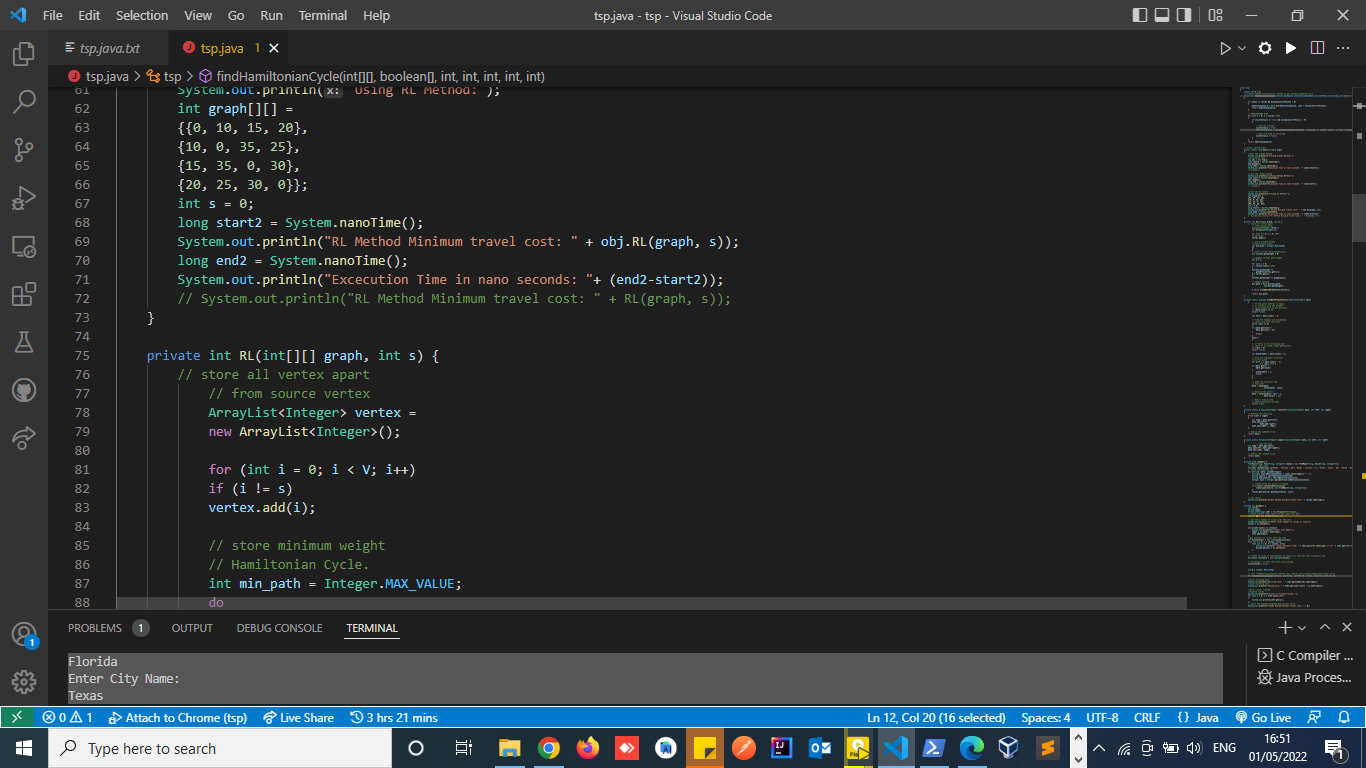
Algorithm one which is greedy(); method in this case; uses user inputs to calculate the cost of traversing several cities. Prizes can be modified in this method. Note that this method uses the Hamiltonian Cycle to calculate weighted cost between nodes.



**How to initialize the Q table and R table with consideration of the prizes.**

Q table is a simple lookup table where we calculate the maximum expected future rewards for action at each state. Basically, this table will guide us to the best action at each state. There will be four numbers of actions at each non-edge tile.

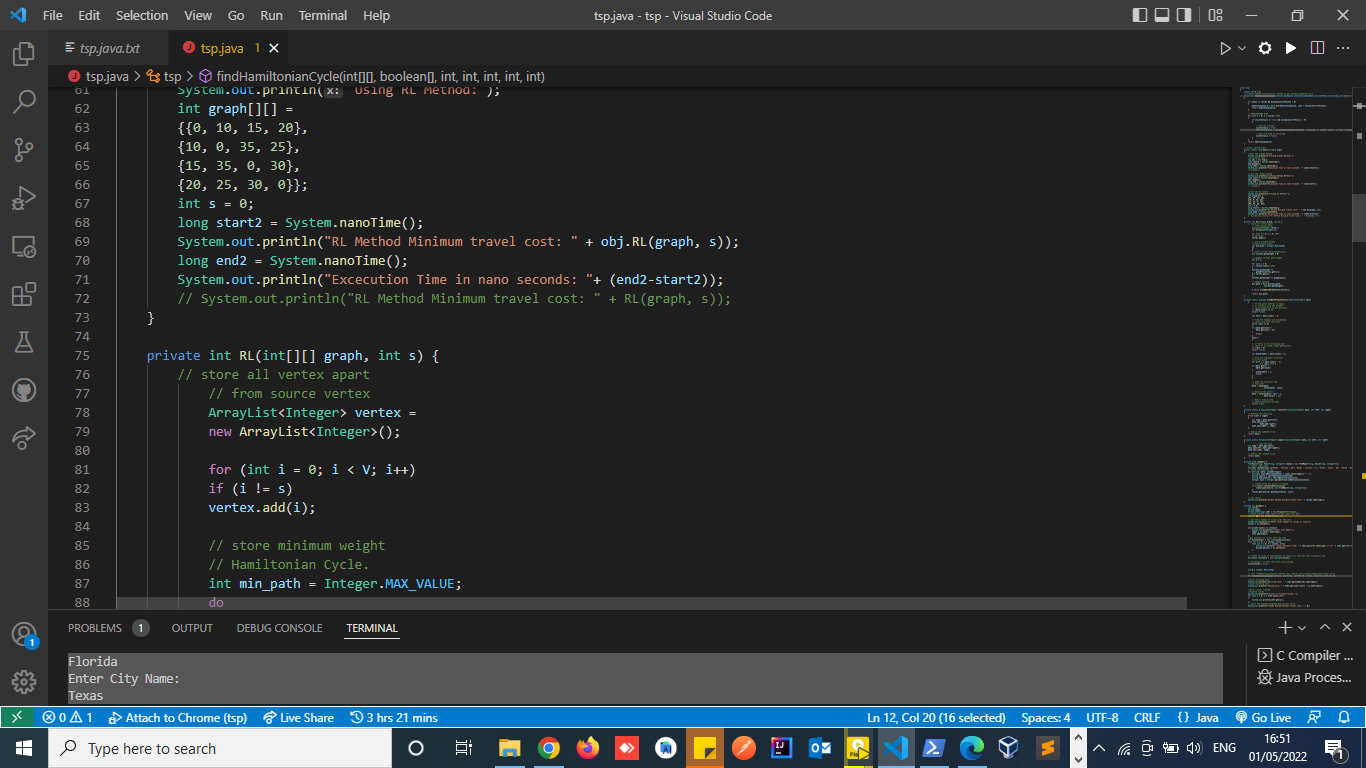
On my RL method the Q and R table are initialized as a graph array and prizes i.e cost implemented on each array value.



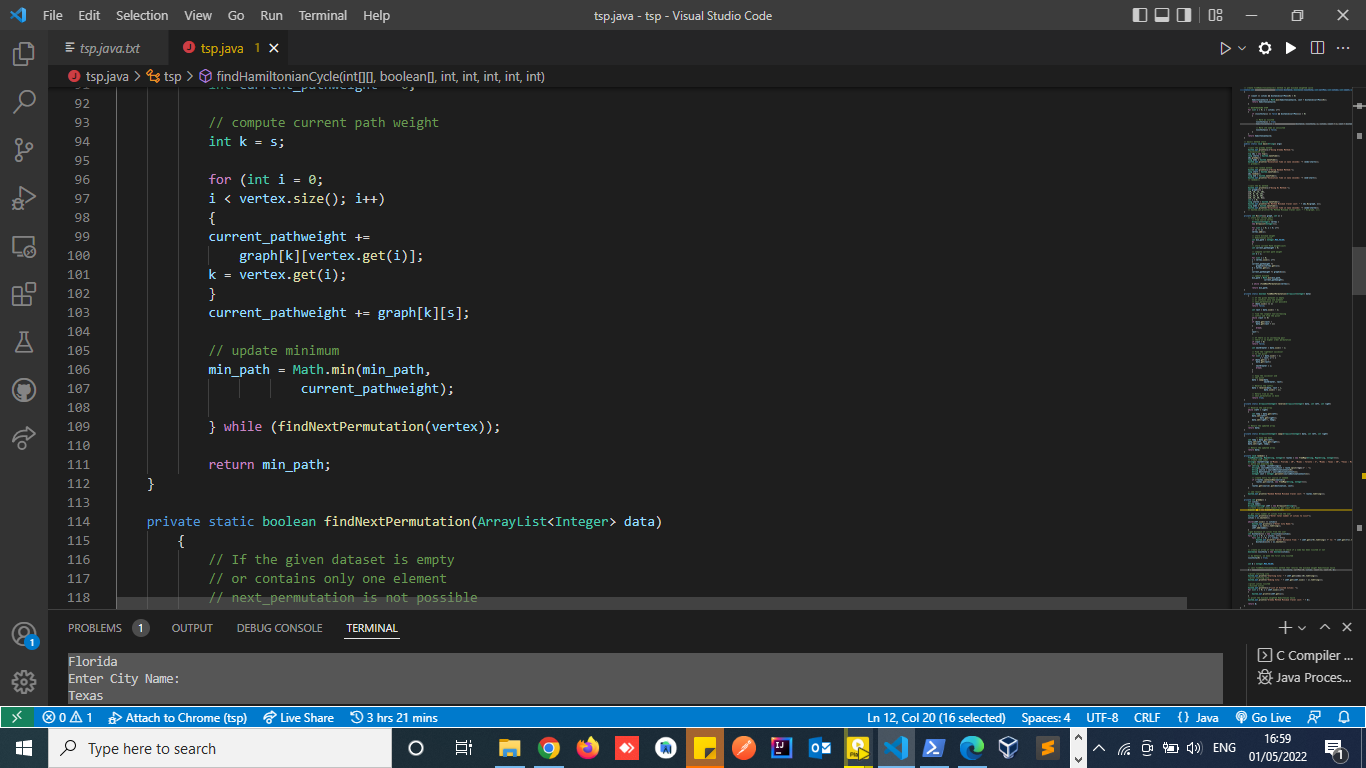
**The plots**

Java Plot is a term which is mainly use for plotting coordinates on a cartesian plane. A plot is a graphical technique for representing a data set, usually as a graph showing the relationship between two or more variables.

In this case the RL(); method accepts a graph and in input,

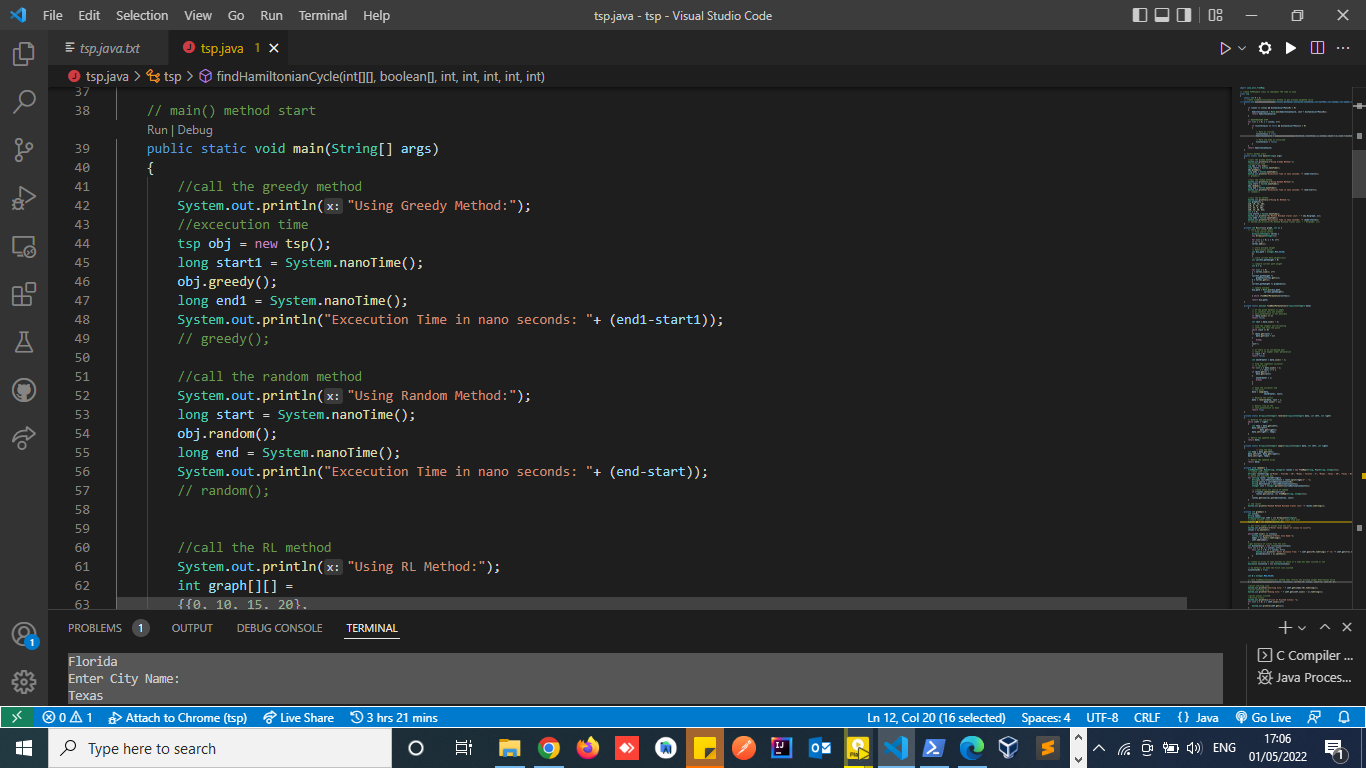


The method iterates through the graph to find the next permutation and returns the minimum path to the next node. To fix m=10, the number of iterations are limited to the length of user inputs i.e number of cities to visit.



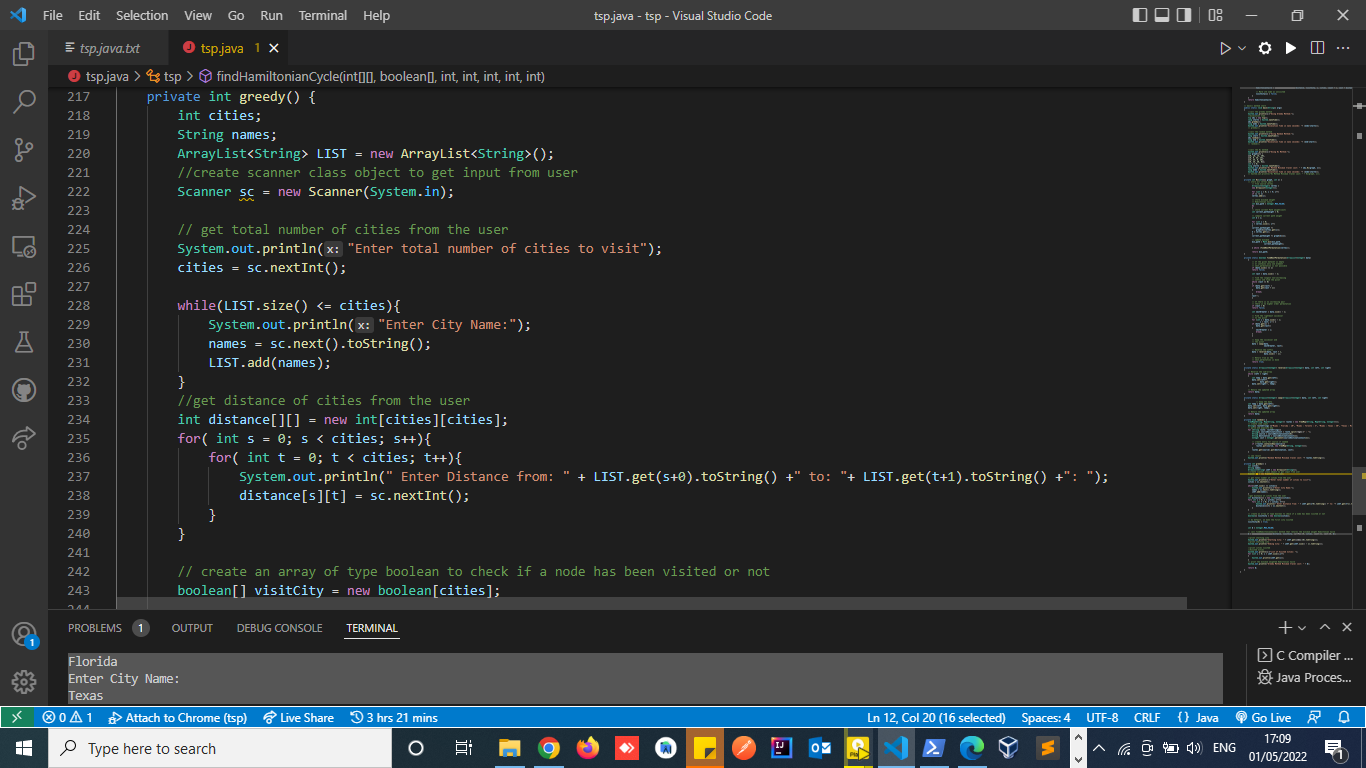
For the Execution time, the main class that is tsp() is initialized as an object. Each method that is greedy(), random() and RL() are attached to the object.

I recorded each method call start time and end time, My execution time is calculated by getting the difference in start and end time of every method call.



The number of iterations (Q =2500) are limited by the number of cities to visit.

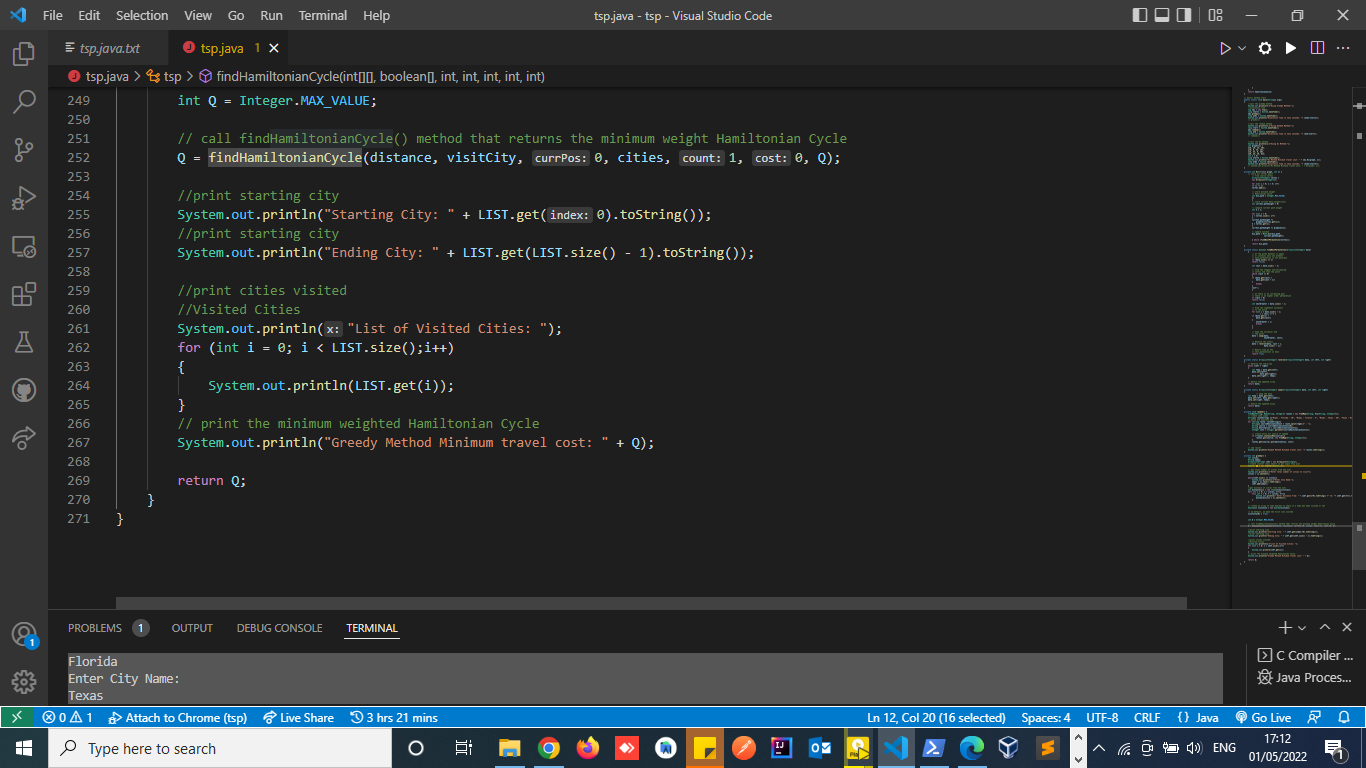
The more the number of cities the more the iterations will take longer, this also affects the execution time.



In my code the Q Value (Q Function): Usually denoted as Q(s,a) (sometimes with a π subscript, and sometimes as Q(s,a; θ) in Deep RL),

Q Value is a measure of the overall expected reward assuming the Agent is in state s and performs action a, and then continues playing until the end of the episode following some policy π.

This value is got by parsing distance, visitCity(or node), current node position, number of cities to visit, node position and the cost/price.



**Analysis of my plot**

The traveling salesman problem is a classic problem in combinatorial optimization. This problem is to find the **shortest path** that a salesman should take to traverse through a list of cities and return to the origin city. The list of cities and the distance between each pair are provided.

This is an **NP-hard** problem. In simple words, it means you cannot guarantee to find the shortest path within a reasonable time limit. This is not unique to TSP though. In real-world optimization problems, you frequently encounter problems for which you must find **sub-optimal** solutions instead of optimal ones.

The Reinforcement Learning or RL method used here guarantees to find the best answer to TSP. However, its time complexity exponentially increases with the number of cities. The time complexity with the RL method asymptotically equals N² × 2^N where N is the number of cities.

For a hint of how this time complexity increases, let me share my experiments. TSP with 10 cities can be solved by a RL method in almost 0.2 seconds using intel core i7. This number increases to almost 13 seconds (~60 times greater) with 15 cities. That is, the time complexity significantly increases even with a small increment in the number of cities.

To optimize the RL method, I could have used the memorization technique. However, the memorization technique with a large number of cities needs a 2^N × 2^N matrix that cannot be easily handled in memory.

**Conclusion**

There are many methods to solve traveling salesman problem. However, the above method is the most common one. I highly suggest the greedy method since it is implemented simply and executes fast. However, the simulated RL method is very powerful if you can properly tune it and you do not have a time constraint to find the final result.