CSE551 : Programming Assignment

December 9, 2022
Monalisa Dokania (1225344640)

1. **Input provided** :- The input file is given as a .TXT file, which contains rows of entries, each row containing the following columns, delimited by space-
    a. Source airport code,
    b. Destination airport code,
    c. Arrival time,
    d. Departure time,
    e. Capacity of the flight
2. **Language used** :- Java, and Python for converting from .txt to .csv
3. **Algorithm employed** :- Ford Fulkerson algorithm
4. **Strategy used** :-
    a. **Preparing input file** - The input is given as a .txt file. In order to use this as an input to our algorithm, this is converted into a csv file using Python to_csv() function, as written in the txt_to_csv_conversion.py file.
    b. **Java classes Used** : Two classes were used for this project. The 'Flight' class contains the variables with a unique id, the Source city, Destination city, Departure time, Arrival time, and capacity of the flight, for all the given flights for different airlines given in the input. The second class is the 'NASMaxFlow' class which is the main class used for implementing the algorithm.
    c. **Methodology** -
        i. The NASMaxFlow class first creates all the nodes in the graph, as a 2D array, with 194 values (8 intermediate airports * 24 hours + 1 Source airport + 1 Target airport). So the 0th node is LAX, the 1st-24th nodes are SFO with different hours in 24 hours, and so on, till the 193rd node JFK.
        ii. Then the program uses the getData() method to read the previously converted input csv file (data.csv), and creates instances of the Flights class, representing the entire dataset provided. Thus the number of instances is equal to the number of rows provided in the input file.
        iii. Next, we check if there are any direct flights from LAX to JFK, that is, from node 0 to node 193. So, for each flight journey, if the arrival and departure time lie within the range given in the problem (6:00AM - 5:59AM), in case the flight is a direct one, we add its capacity to the net flow. If it is not a direct flight, we update the nodes matrix with the capacity value. Also separately, if the airports of 2 nodes are the same (for example- PHX 7:00AM to PHX 9:00AM) we just assign Integer.MAX_VALUE to them.
        iv. Now that we have set every node and the edges, our graph is ready. As done in class, we will set the initial flows to zero and keep updating the flow values till we find an augmented path. So now we call the Ford Fulkerson algorithm to calculate the maximum flow.

v. We use Breadth First Search in order to traverse through the graph, which helps us to find an augmented path. It returns true if a path exists.

vi. We iterate in a loop till a path exists (found by BFS), and in each case we keep updating the flow along the edges. This is done by first finding out the bottleneck capacity in the path and then updating the entire path by increasing the flow and correspondingly decreasing the residual capacity due to the flow. This is done by increasing the forward flow edges and decreasing the edge back flow in the graph. After covering the current augmented path, we add the flow obtained from this path to the net flow.

vii. We keep repeating the step vi. till there is no path remaining, and then return the net flow, which is our final output.

5. **Output** :- For the given input provided (flights.txt), consisting of 10 airports and 722 rows, output = **8199**
6. **Instructions to run the code** :-
   a. Copy the files flights.txt, txt_to_csv_conversion.py, NASMaxFlow.java, Flight.java in the same directory. (In order to test with any other input, the name and format should be same as flights.txt)
   b. Open the terminal and cd to the above directory. Then install pandas library using the command 'pip3 install pandas'
   c. Run the python file using the command 'python3 txt_to_csv_conversion.py'. This will create a file named 'data.csv'
   d. Next, compile and run the java code, using commands 'javac NASMaxFlow.java', and 'java NASMaxFlow.java'.
7. **References** :-

   a. https://www.programiz.com/dsa/ford-fulkerson-algorithm
   b. https://www.geeksforgeeks.org/ford-fulkerson-algorithm-for-maximum-flow-problem/