- **<u>Code:</u>**
  Please refer to the following files from the Zip Folder:
  1. **MaxCapacityAirspace.java** (Contains the main function)
  2. **FlightCityDetail.java** (Consists of Getter Setter methods)

- **<u>Dataset:</u>**
  Please refer to the "**dataset.csv**" for dataset

- **<u>Input to the Algorithm:</u>**
  The input to the algorithm is a **weighted adjacency matrix**.
  1. First, each city is represented as 24 nodes. These nodes are used to capture the temporal aspect. Each node stands for one of the 24 hours (rounded to nearest hour) used in a 24-hour format.
     Mapping of a city is done using a Hashmap data-structure wherein the source node "LAX" is considered 1 and destination node "JFK" is considered 10 and other intermediate cities accordingly.
     (This is done to ease the implementation of constructing an adjacency matrix)
  2. Consider a flight between two cities City1 and City2, leaving from City1 at time t1 and reaching at City2 at time t2. The capacities of all such flights are added and the corresponding weight is assigned to datapoint in weighted adjacency matrix at [(City1 , t1) , (City2 , t2)]
     (Initially all the values of the weighted adjacency matrix are zero)
  3. Additionally, I have considered a source node and a sink node in constructing the above matrix. This is done so that Ford-Fulkerson can be applied which takes into consideration only one source node and a single destination node.
  4. All the 24 nodes representing a city are connected to capture the essence of waiting time and layover flights. The weights are assigned a very high value so that they don't serve as the bottleneck to the algorithm.
  5. Considering point 3, I have considered 24 weighted edges (very high value) from source to first city **LAX** and 24 weighted edges (very high value) from last city **JFK** to sink.

- **<u>Pseudo Code:</u>**

Procedure breadthFirstSearch (adjMatrixGraph, source, sink, parent)

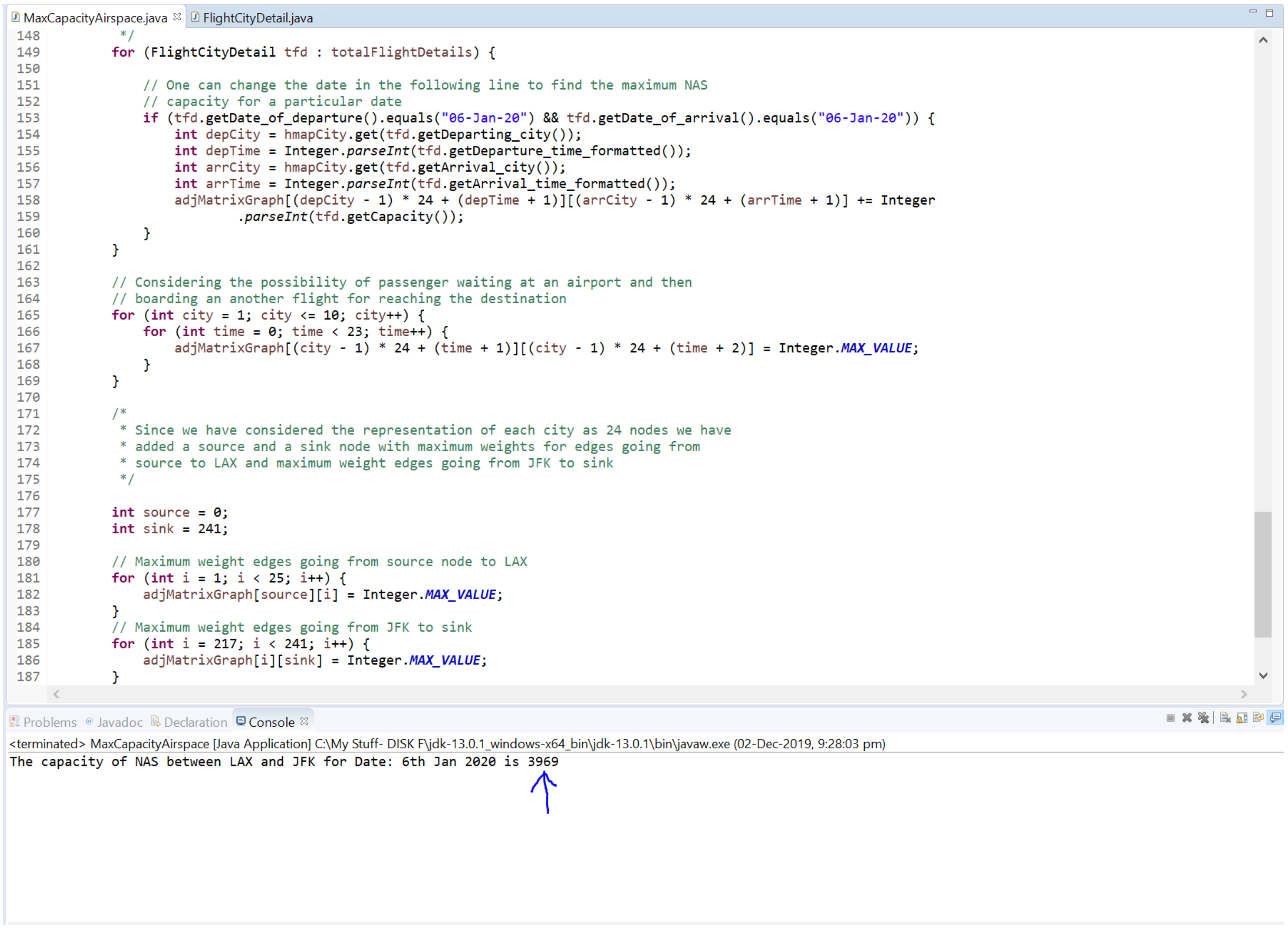
```
    for all nodes in adjMatrixGraph
            Mark visited[nodes]=false
    end for

    Mark the source node as visited
    Initialize the parent of sink node to -1
    Initialize a queue datastructure

    While queue is not empty:
        pop a node u
        for each node v adjacent to u do
            if not visited[v] then
                visited[v] ← true
                parent[v] ← u
                insert v to queue
            end if
        end for
    end while

    if sink is visited then return true else return false
end procedure
```

```
Procedure fordFulkerson (flightgraph, source, sink)
    start with initial maxflow as zero
    While there is an augmenting path from source to sink:
        Add the flow obtained from this to maxflow
    end While
    return maxflow
end procedure
```

```
Procedure main
    read dataset from csv file

    Construct an adjacency matrix as follows:
        1. Consider a flight between two cities City1 and City2,
        leaving from City1 at time t1 and reaching at City2 at time t2.
        The capacities of all such flights are added and the corresponding
        weight is assigned to datapoint in weighted adjacency matrix at [ (City1 , t1) , (City2 , t2) ]

        2. Add directed weighted edges from source to city LAX and from JFK to sink as well.

    Call the fordFulkerson method to find the maximum capacity of NAS
end procedure
```

- **<u>Time Complexity of Algorithm:</u>**
  O(EV^3) where V: number of nodes (vertices) and E: number of edges

- **<u>Output of the code when executed on my dataset is as follows:</u>**


MaxCapacityAirspace.java | FlightCityDetail.java

```java
         */
        for (FlightCityDetail tfd : totalFlightDetails) {

            // One can change the date in the following line to find the maximum NAS
            // capacity for a particular date
            if (tfd.getDate_of_departure().equals("06-Jan-20") && tfd.getDate_of_arrival().equals("06-Jan-20")) {
                int depCity = hmapCity.get(tfd.getDeparting_city());
                int depTime = Integer.parseInt(tfd.getDeparture_time_formatted());
                int arrCity = hmapCity.get(tfd.getArrival_city());
                int arrTime = Integer.parseInt(tfd.getArrival_time_formatted());
                adjMatrixGraph[(depCity - 1) * 24 + (depTime + 1)][(arrCity - 1) * 24 + (arrTime + 1)] += Integer
                        .parseInt(tfd.getCapacity());
            }
        }

        // Considering the possibility of passenger waiting at an airport and then
        // boarding an another flight for reaching the destination
        for (int city = 1; city <= 10; city++) {
            for (int time = 0; time < 23; time++) {
                adjMatrixGraph[(city - 1) * 24 + (time + 1)][(city - 1) * 24 + (time + 2)] = Integer.MAX_VALUE;
            }
        }

        /*
         * Since we have considered the representation of each city as 24 nodes we have
         * added a source and a sink node with maximum weights for edges going from
         * source to LAX and maximum weight edges going from JFK to sink
         */

        int source = 0;
        int sink = 241;

        // Maximum weight edges going from source node to LAX
        for (int i = 1; i < 25; i++) {
            adjMatrixGraph[source][i] = Integer.MAX_VALUE;
        }
        // Maximum weight edges going from JFK to sink
        for (int i = 217; i < 241; i++) {
            adjMatrixGraph[i][sink] = Integer.MAX_VALUE;
        }
```

Problems | Javadoc | Declaration | Console

```
<terminated> MaxCapacityAirspace [Java Application] C:\My Stuff- DISK F\jdk-13.0.1_windows-x64_bin\jdk-13.0.1\bin\javaw.exe (02-Dec-2019, 9:28:03 pm)
The capacity of NAS between LAX and JFK for Date: 6th Jan 2020 is 3969
```

O/P: The capacity of NAS between LAX and JFK for Date: 6th Jan 2020 is 3969