Report

Input:

The program take a csv file as an input which contains information about the flight's arrival time, departure time and also the capacity of passenger of every flight. No need to pass the capacity externally.

The file structure of this file:

Range Index: 477 entries
Data Column: 5
Departure: 477 not-null object
Arrival: 477 not-null object
Dept_time: 477 not-null object
Arr_time: 477 not-null object
Capacity: 477 not-null int64
dtypes: int64(1), object(4)

Every Column in the file is a string object except the capacity column which is an integer.
Departure column contains the information of 'departure-airport'.
Arrival column contains the information of 'arrival-airport'.
Dept_time column contains the information of departure time from the 'departure-airport'.
Arr_time column contains the information of arrival time at the 'arrival-airport'.
Capacity column contains the information of passenger carrying capacity of particular flight.

Pseudocode:

Required: MultiDiGraph()
Return: Maximum Passenger capacity of the entire network.

Initialization: nodes = airports; edges = tuple('airport1', 'airport2', {'dept_time': '', 'arr_time': '', 'capacity': 0}); min_capacity = INT.MAX, total_capacity = 0

1: Start with the initial node(airport) 'LAX' and append it in visited.
2: Use Recursion with DepthFirstSearch (DFS) for all the flights from 'LAX'.
3: nodes.push('LAX'), visited_edges = [], visited_nodes = []

```
For neighbour in 'LAX'.neighbour():
    If neighbour not in visited:
        travel_time = 24 – duration of the first flight from 'LAX'
        visited_nodes.push('LAX')
```

getMinCapacity(neighbour, travel_time, visited_nodes)

4: getMinCapacity(neighbour, travel_time, visited_nodes)

If travel time is greater or equal to 24:

Reset value of minimum capacity

visited.append(edge)

Return minimum capacity found so far

- While traversing to get the path to 'JFK', consider checking time constraint of that particular flight, the departure time should be greater than or equal to the arrival time of the incoming flight. Also consider travel time should not be greater than 24.
- Keep track of the minimum capacity of the path.
- Now, while going back to the initial node 'LAX', subtract this minimum capacity from each edge's capacity in that path. Now we get the residual graph.
- If the residual capacity of any edge becomes 0, mark that edge as visited. That means, it can not be traversed again in future.
- At the end, update the total capacity = total capacity + min_capacity, each time algorithm traverses along a single path and gets to 'JFK'.
- **Return** Updated total capacity.

5: end getMinCapacity function.
6: Update total capacity = total capacity + capacity returned by getMinCapacity function
7: end for loop of neighbours.
8: return total capacity.

Time Complexity:

As the given algorithm uses DFS, the worst case time complexity would depend on the branching factor **b,** which is the total number of flights leaving each airport.

Let's say 'b' flights leave from 'LAX'. At each level, there can be 'b' flights from the next connecting airport. Hence for next level the worst case would be b^2. (b raise to 2)

Let's say there are 'd' cities. So, when algorithm finally reaches to 'JFK', it would have traversed:

$b + b^2 + \ldots\ldots + b^d = O(b^d)$

So the time complexity would be **b^d.**

Output:

```
('LAX', 'DEN', {'arr': '10:00', 'capacity': 126, 'dept': '07:00'})
3
('DEN', 'IAD', {'arr': '21:00', 'capacity': 150, 'dept': '16:00'})
14
('LAX', 'DEN', {'arr': '10:00', 'capacity': 126, 'dept': '07:00'})
3
('DEN', 'IAD', {'arr': '21:00', 'capacity': 180, 'dept': '16:00'})
14
('LAX', 'DEN', {'arr': '10:00', 'capacity': 126, 'dept': '07:00'})
3
('DEN', 'IAD', {'arr': '23:00', 'capacity': 150, 'dept': '17:00'})
16
('LAX', 'DEN', {'arr': '10:00', 'capacity': 126, 'dept': '07:00'})
3
('DEN', 'IAD', {'arr': '23:00', 'capacity': 180, 'dept': '18:00'})
16
('LAX', 'DEN', {'arr': '10:00', 'capacity': 126, 'dept': '07:00'})
3
('DEN', 'IAD', {'arr': '05:00', 'capacity': 126, 'dept': '24:00'})
22
('LAX', 'DEN', {'arr': '10:00', 'capacity': 126, 'dept': '07:00'})
3
('LAX', 'DEN', {'arr': '12:00', 'capacity': 128, 'dept': '08:00'})
4
('LAX', 'DEN', {'arr': '12:00', 'capacity': 128, 'dept': '09:00'})
3
('LAX', 'DEN', {'arr': '12:00', 'capacity': 165, 'dept': '09:00'})
3
('LAX', 'DEN', {'arr': '13:00', 'capacity': 165, 'dept': '09:00'})
4
('LAX', 'DEN', {'arr': '14:00', 'capacity': 78, 'dept': '10:00'})
4
('LAX', 'DEN', {'arr': '14:00', 'capacity': 128, 'dept': '11:00'})
3
('LAX', 'DEN', {'arr': '17:00', 'capacity': 126, 'dept': '13:00'})
4
('LAX', 'DEN', {'arr': '18:00', 'capacity': 128, 'dept': '14:00'})
4
Maximum Capacity:  6703
```