INPUT:

- **Node list** : This is the list of all possible flights in the NAS
    - Format: (str(Airport_from, Dep_hour, Airport_to, Arr_hour, Airline_code))
- **Edge list** : This is the list of all possible nodes (flights) which can be reached from the 'To Airport' of the node as per the constraints of departure time.
- **Edge weight list** : This is the list of passenger capacity of the flight which can be taken from the current flight.

PSEUDOCODE:

```
function breadth_first_search(node_data_list, edge_list, weight_list, source, sink, residual_capacity):
    visited=[[] for x in range(len(node_data_list))] //    # Mark all the vertices as not visited
    for ind,node in enumerate(node_data_list):
        visited[ind].append([False]*len(edge_list[ind]))


    queue <- []
    queue <- source //start augmenting paths from source
    capacity <- 0
    while queue:
        node <- queue.pop(0)
        if node != sink:
            node_index <- node_data_list.index(node)
            for to_in_edge in edge_list[node_index]:
                edge_node_index <- (edge_list[node_index]).index(to_in_edge)
                current_node <- (visited[node_index])[0]
                if current_node[edge_node_index] equals False:
                    queue.append(to_in_edge)
                    current_node[edge_node_index] <- True
                    capacity <- int(weight_list[node_index][edge_node_index]) // add path capacity to array

    return capacity, visited




def find_network_capacity(node_data_list, edge_list, weight_list, source, sink):
    total_capacity <- 0
    residual_capacity <- []
    while True:
        capacity_of_path, path <- breadth_first_search(node_data_list, edge_list, weight_list, source, sink, residual_ca
pacity)
        if capacity_of_path equals 0:
```

```
        break
    total_capacity <-  capacity_of_path // add capacity_of_path to total_capacity
    v <- sink
return total_capacity
```

TIME COMPLEXITY:

The time complexity of the algorithm is O(V*(E^2)) for a graph G (V, E). Since we are doing a Breadth First Search to find the augmenting path, every time we get the shortest possible path where atleast one edge in E which will be saturated by the flow, and the maximum length of an augmenting path is equal to the number of vertices of V.

OUTPUT OF THE CODE FOR THE DATASET:

```
229566
```