**flightData.keys():**

'Num', 'Orig', 'Dest', 'DepTime', 'ArrTime', 'PrevFlightBySameAircraft', 'PrevFlights', 'NextFlights', 'FlyingDate', 'AircraftUsed', 'FlightType', 'RemCabinCapacityGivenItin'

**itinData.keys():**

'Num', 'InOrOut', 'UnitCost', 'PaxCount', 'NumOfLegs', 'LegFlightNum', 'LegFlyDate', 'LegCabinClass', 'SourceAirport', 'ItinStartTime', 'ItinStartDate', 'SinkAirport', 'ItinEndTime', 'ItinEndDate'

**rotationData.keys():**

'FlightNum', 'DepDate', 'Aircraft'

| **Function** | **Arguments** | **Description** |
| --- | --- | --- |
| **CabinCapacity** | flightData  aircraftData  itinData | Remaining capacity of each flight when a given itin is implemented:  - if you pass **data\_flights**, it should output full capacities  - if you pass **data\_recovflights**, it should output remaining capacities |
| **PrevNextFlightList** | flightData  rotationData (for aircraft)  flightTypeData (D/C/I)  turnaroundTime (35 mins etc.) | This function creates a subdictionary of feasible previous and next flights for every flight f. That is, if we pick flight f in our path, what are the possible previous and next flights that can go in the path. |
| **NextReachableAirport** | givenAirport  givenDateTime  flightData  recovByTime (end time of recovery period) | From givenAirport (JFK) and givenDateTime (8 AM), here is a dict with keys as the possible  airports that can be reached (BOS,LAX) and the value for each possible destination is a list of corresponding flights from givenAirport to that possible airport |
| **CreatingGraphGivenAnItinerary** | itinData  itinIndex  flightData  allAirports  disrupStartTimev  recovByTime  MaxLegNumIncrease (2 legs etc) | Recursive function that builds the feasible DAG graph given an itinerary  Lots of pruning happens inside. |
| **GraphBuilder** | givenAirport  givenDateTime  flightData  sinkAirport  recovByTime  visited | Recursive function that builds a graph given source + time and sink  Uses **CreateGraphGivenAnItinerary** |
|  |  |  |

The 3 output DAGs (Directed Acyclic Graphs) of **CreateGraphGivenAnItinerary**:

**airportDAG** (keys: itineraries)

for key 0 (itinerary 0), keyvalue is a dict itself. In this nested dict, 'BIQ': ['CDG', 'ORY'] means the arcs BIQ -> CDG and BIQ -> ORY are in the DAG

Sample output:

Key: 0, values: {'BIQ': ['CDG', 'ORY'], 'CDG': ['ORY']}

Key:1, values: {'BIQ': ['CDG', 'ORY'], 'CDG': ['ORY'], 'ORY': ['AVN']}

Key 2, values: {'BIQ': ['CDG', 'ORY'], 'CDG': ['ORY']}

Key 3, values: {'BIQ': ['CDG', 'ORY'], 'CDG': ['ORY'], 'ORY': ['AVN']}

Key 4, values: {'ORY': ['CDG', 'LIG', 'MRS', 'TLS', 'NCE', 'BOD', 'SXB', 'MLH', 'BES', 'LYS', 'CFE', 'PUF', 'BIQ', 'MPL'], 'MRS': ['CDG'], 'CDG': ['BIQ'], 'BES': ['CDG'], 'LYS': ['CDG'], 'TLS': ['CDG'], 'SXB': ['CDG'], 'NCE': ['CDG'], 'BOD': ['CDG'], 'MLH': ['CDG'], 'PUF': ['CDG'], 'CFE': ['CDG'], 'MPL': ['CDG'], 'LIG': ['CDG']}

**flightDAG (**keys: itineraries)

For key 0 (itinerary 0), keyvalue is a dict itself. In this nested dict,

nested key is a tuple ('BIQ', 'CDG') is an arc of the DAG served by flights 4498 and 4502

Sample output:

Key 0:

values:

('BIQ', 'CDG'): ['4498', '4502'], ('BIQ', 'ORY'): ['4344', '4348', '4352', '4354', '4358'],

('CDG', 'ORY'): ['25', '26', '27', '28', '29', '30', '31', '32', '33', '34', '35', '36', '37', '38', '39', '40', '41', '42', '43', '44', '45', '46', '47', '48', '49', '50', '51', '52', '53', '54', '55', '56', '57', '58', '59', '60', '61', '62', '63', '64', '65', '66', '67', '68', '69', '70', '71', '72']

Key 1:..

**allPathsDAG** (key:itineraries)

key j is itinerary j. keyvalue is a nested dictionary for e.g. keyvalue for itin 0 looks like {0:['BIQ','CDG','ORY'],1:['BIQ','CDG','NTE'],2...}

# list of all paths from BIQ to ORY

Sample Output:

Key 0: values: {0: ['BIQ', 'CDG', 'ORY'], 1: ['BIQ', 'ORY']}

Key: 1, values: {0: ['BIQ', 'CDG', 'ORY', 'AVN'], 1: ['BIQ', 'ORY', 'AVN']}

Key :2, values: {0: ['BIQ', 'CDG', 'ORY'], 1: ['BIQ', 'ORY']}

Key: 3, values: {0: ['BIQ', 'CDG', 'ORY', 'AVN'], 1: ['BIQ', 'ORY', 'AVN']}

**Even for the small dataset, there are itineraries with 130 possible paths between source to sink**