Daily Coding Problem Blog

Daily Coding Problem #41

Problem

This problem was asked by Facebook.

Given an unordered list of flights taken by someone, each represented as (origin, destination) pairs, and a starting airport, compute the person's itinerary. If no such itinerary exists, return null. If there are multiple possible itineraries, return the lexicographically smallest one. All flights must be used in the itinerary.

For example, given the list of flights [('SFO', 'HKO'), ('YYZ', 'SFO'), ('YUL', 'YYZ'), ('HKO', 'ORD')] and starting airport 'YUL', you should return the list ['YUL', 'YYZ', 'SFO', 'HKO', 'ORD'].

Given the list of flights [('SFO', 'COM'), ('COM', 'YYZ')] and starting airport 'COM', you should return null.

Given the list of flights [('A', 'B'), ('A', 'C'), ('B', 'C'), ('C', 'A')] and starting airport 'A', you should return the list ['A', 'B', 'C', 'A', 'C'] even though ['A', 'C', 'A', 'B', 'C'] is also a valid itinerary. However, the first one is lexicographically smaller.

Solution

This problem is similar to the N queens problem a few days ago: we have a desired final state (all the flights are used up), we can construct partial itineraries and reject them, and at each step we have potentially multiple avenues to explore. That suggests that backtracking is again a very likely candidate for solving our problem.

In particular, we can do the following:

- Keep a list of itinerary candidates
- Keep a current itinerary initialized with our starting airport
- Then, recursively:
    - Iterate over all the flights that start from the last airport in our itinerary
    - For each flight, temporarily add the destination to our itinerary and remove it from the flight list. Then call ourselves recursively with the new itinerary and flight list.
    - Concatenate all the results to our list of itinerary candidates.
- Sort our itinerary candidates and pick the lexicographically smallest one.

To speed this up, we'll store all the flights into a dictionary with the origin as a key and a list of flight destinations from that origin as the value. Then we can look up our options in O(1) time instead of O(N) time.

```
from collections import defaultdict

def get_itinerary(flights, start):
    # Store all the flights into a dictionary key:origin -> val:list of destinations
    flight_map = defaultdict(list)
    for origin, destination in flights:
        flight_map[origin] += [destination]

    def visit(flight_map, total_flights, current_itinerary):
        # If our itinerary uses up all the flights, we're done here.
        if len(current_itinerary) == total_flights + 1:
            return [current_itinerary[:]]

        last_stop = current_itinerary[-1]
        # If we haven't used all the flights yet but we have no way
```

```
    # of getting out of this airport, then we're stuck. Backtrack out.
    if not flight_map[last_stop]:
        return []

    # Otherwise, let's try all the options out of the current stop recursively.
    # We temporarily take them out of the mapping once we use them.
    potential_itineraries = []
    for i, flight in enumerate(flight_map[last_stop]):
        flight_map[last_stop].pop(i)
        current_itinerary.append(flight)
        potential_itineraries.extend(visit(flight_map, total_flights, current_itinerary))
        flight_map[last_stop].insert(i, flight)
        current_itinerary.pop()
    return potential_itineraries

valid_itineraries = visit(flight_map, len(flights), [start])
if valid_itineraries:
    return sorted(valid_itineraries)[0]
```