CSIS 2420 Summer 2016
Steven Benjamin
Final Code

The problem of finding connecting flights between any two cities in an adjacency matrix is easily solved using Dijkstra's shortest path algorithm. Dijkstra's algorithm is similar to Prim's in that it repeatedly adds the shortest edge that connects a new node beginning at some starting node. In Prim's, the goal is to create a minimum spanning tree connecting all nodes, so the starting node is arbitrary and doesn't affect the outcome. Dijkstra's algorithm is well suited to solving the connecting flight problem because it allows us to choose any starting and ending edge, and find the shortest path between them.

While Princeton provides us with two classes that would be useful in solving this problem, Dijkstra.java and Graph.java, I will show a simple solution that does not rely on the Graph class but instead implements Dijkstra on the provided flight adjecency matrix.

First, two N by N matrices are created: one containing the adjacencies(connecting flights), where AM[i][j] represents the distance between the city represented at row index i and column index j, and the the other represents the 'cost', to use Dijkstra's terminology, between the cities. A row and column with the same index value represent the same city e.g. AM[1][1] is Seattle, AM[2][2] is San Francisco, but AM[1][2] denotes the travel distance between Seattle and San Francisco. If there is no connecting flight, AM[i][j] will equal zero. D[i][j] will hold the connecting distance(cost) where there are connections in AM and will be set to infinity where there are no connections. The table of cities and travel distances will be displayed before reading in the source and destination.

Number of Cities N;
Map Index to City;
Adjacency Matrix AM [N][N];
Distance Matrix D[N][N];
Visited[N];
Distance[N];
Path[];
Minimum Distance;
Starting City;
Ending City;

Read in a list of cities, from a file or standard input , starting with city i;
Map(i, city)and increment i;

For each city row in AM [][],
read in the travel distances at each destination city column and set them correspondingly in AM[][];

For each city row in AM [i][],
For each city column in AM [][j],
Print out the the city name keyed to i in the Map and print out each city j where the distance is not 0;

Read in the Starting and Ending City;

For each position in AM,
If the distance at that position = 0,
then set the corresponding position in D to infinity;

Next, set the index that corresponds to the starting city to true in the visited matrix. The other indexes will be set to true as the corresponding nodes are visited, preventing nodes from being revisited. Here, the starting city is Seattle(index 1). The distance to each node from the starting node is tracked in Distance[N]. After initializing Distance with the the distances of the cities reachable from the starting city, the distance to the starting position is set to 0. Because the algorithm hasn't begun visiting nodes, the Minimum Distance is infinity.

Visited[1] = true;
For each city up to N, set Distance[i]=D[Starting City][i]
Distance[1]=0;
Minimum Distance = infinity;

While there are unvisited cities, mark the next unvisited city as visited and if it's distance is less than the current minimum distance, use it to set the new minimum distance and add it to Path[]. Once the path includes the destination, the search can stop.

While the total visited is less than N,
For each city i up to N, excluding the Starting City,
If Distance[i] is less than the Minimum Distance,
Then Minimum Distance = Distance[i]
And the Next City = i ;

For each city i up to N, excluding the Starting City,
If Minimum Distance + D[Next City][i] is less than Distance[i]
Then Distance[i] = Minimum Distance + D[Next City][i]
Path[i]=Next City;

Increment the total visited;

When the algorithm is finished checking all of the possible cities and travel distances, either print out the path, the list of connecting flights from the starting city to the destination, or print that no connection was found.

If Path[] does not include Ending City, print out "Sorry!"

Else, print out Path[]

References:

Dijkstra's algorithm - From Wikipedia, the free encyclopedia
https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

Dijkstra's Algorithm - Melissa Yan
http://math.mit.edu/~rothvoss/18.304.3PM/Presentations/1-Melissa.pdf

4.4 Shortest Paths - Algorithms, 4th edition
http://algs4.cs.princeton.edu/44sp/

Dijkstra.java - Princeton.edu
ftp://ftp.cs.princeton.edu/pub/cs226/map/Dijkstra.java