Graph Algorithms

15-121

Introduction to Data Structures

Ananda Gunawardena

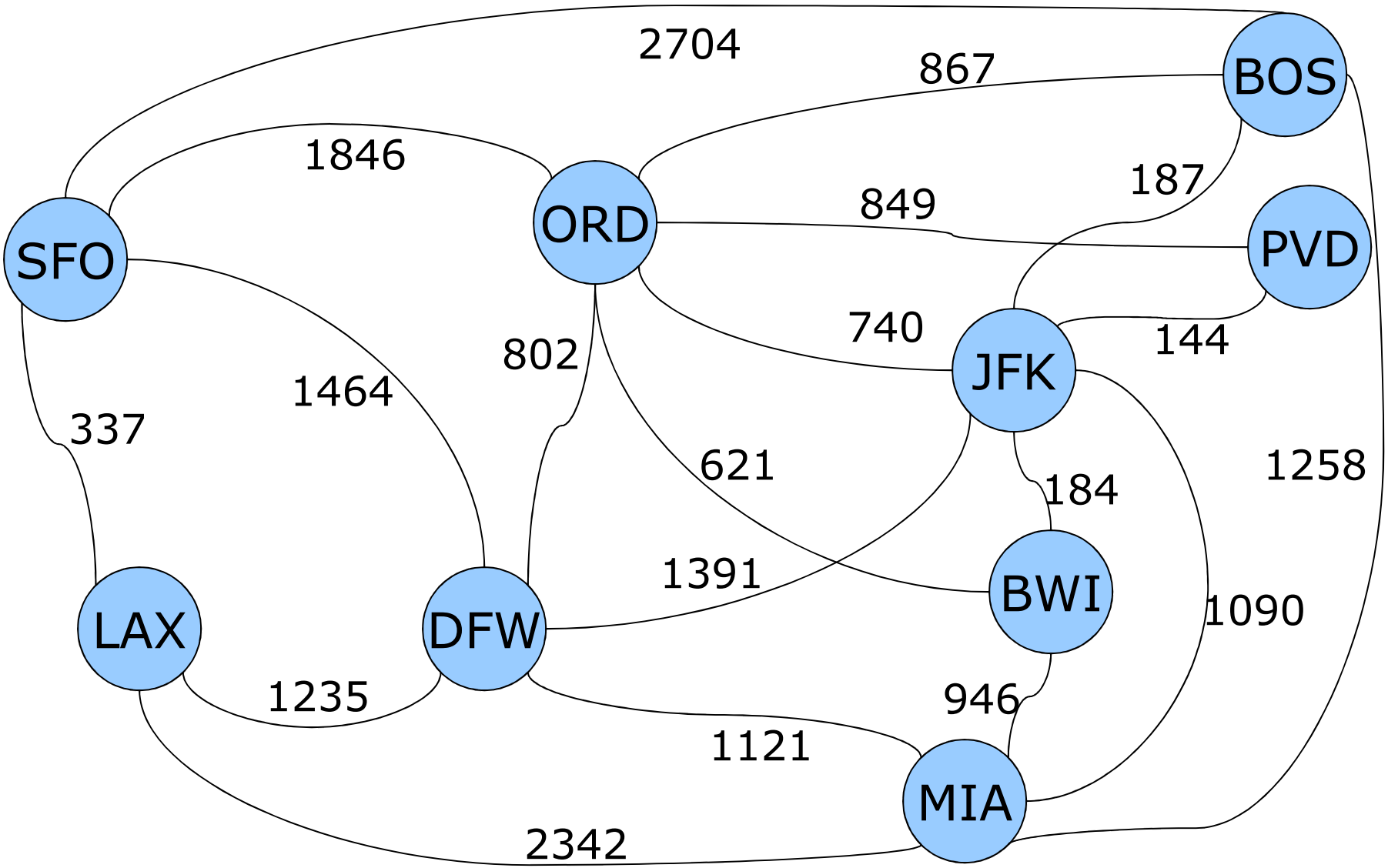
In this lecture..

- Main idea is finding the Shortest Path between two points in a Graph

- We will look at
  - Graphs with non negative cost edges
    - Dijkstra's Algorithm

Shortest Paths

Airline routes



How to find the shortest Path?

The naïve solution is O(n!)

Greedy Algorithms

Greedy Algorithms

- In a *greedy algorithm*, during each phase, a decision is made that appears to be optimal, without regard for future consequences.
- This "take what you can get now" strategy is the source of the name for this class of algorithms.
- When a problem can be solved with a greedy algorithm, we are usually quite happy
- Greedy algorithms often match our intuition and make for relatively painless coding.

Greedy Algorithms

- 4 ingredients needed
  - Optimization problem
    - Maximization or minimization
  - Can only proceed in stages
    - No direct solution available
  - Greedy Choice Property
    - A locally optimal solution (greedy) will lead to a globally optimal solution
  - Optimal Substructure
    - An optimal solution to the problem contains, within it the optimal solution to the sub problem

Examples

- Find the minimum number of coins necessary to change 63 cents
  - Assume we have 25-cent, 10-cent, 5-cent, 1-cent coins
- Dijkstra's algorithm for shortest paths
  - Next…

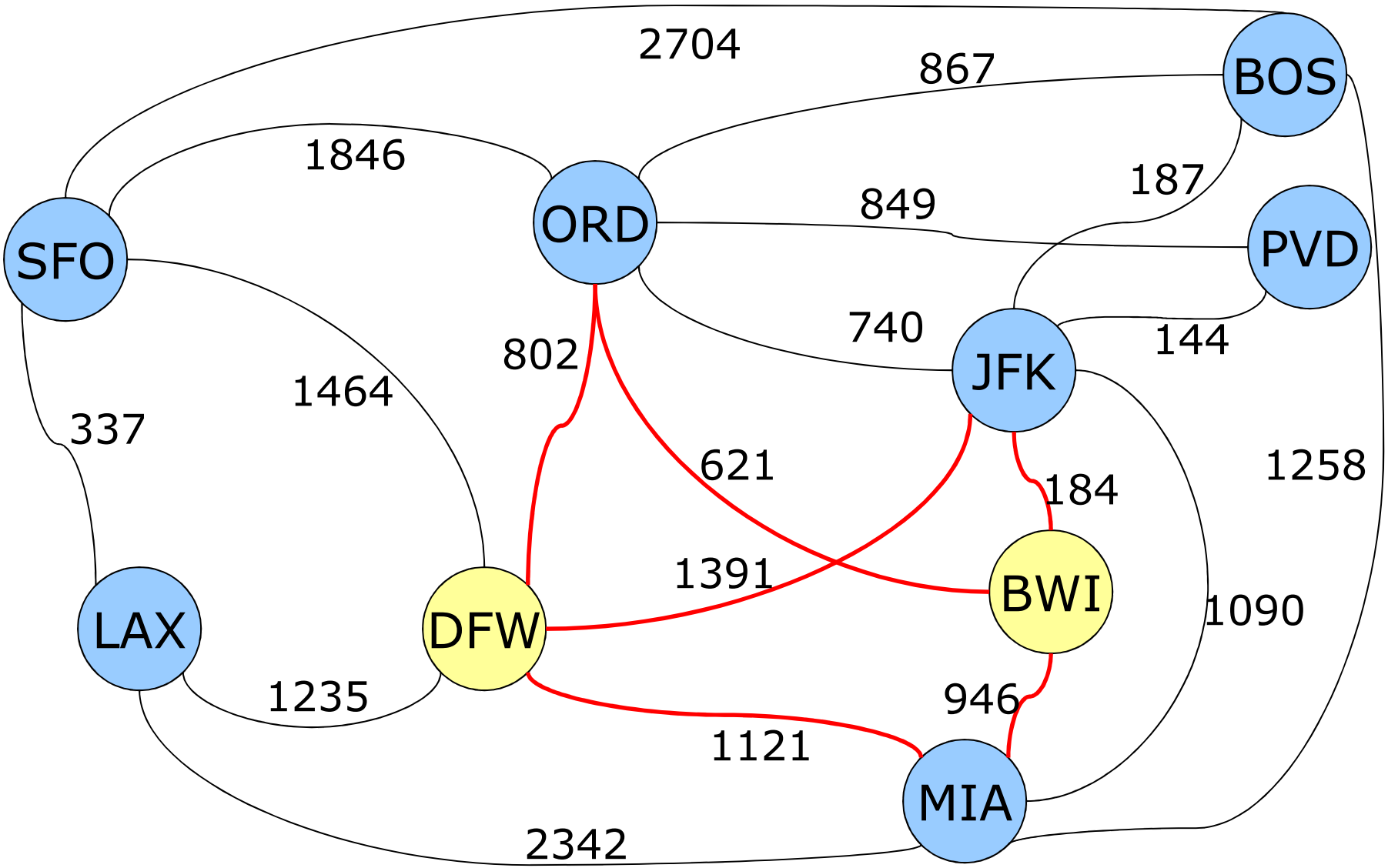
Shortest Path Algorithm for Non-negative weights

(Dijkstra's Algorithm)

Weighted shortest path

- Now suppose we want to minimize the total mileage.
- Breadth-first search does not work!
  - Minimum number of hops does not mean minimum distance.
  - Consider, for example, BWI-to-DFW:

Three 2-hop routes to DFW



Dijkstra's Algorithm

Intuition behind Dijkstra's alg.

- For our airline-mileage problem, we can start by guessing that every city is ∞ miles away.
  - Mark each city with this guess.
- Find all cities one hop away from BWI, and check whether the mileage is less than what is currently marked for that city.
  - If so, then revise the guess.
- Continue for 2 hops, 3 hops, etc.

Dijkstra's: Greedy algorithm

- Assume that every city is infinitely far away.
  - I.e., every city is ∞ miles away from BWI (except BWI, which is 0 miles away).
  - Now perform something similar to breadth-first search, and *optimistically guess that we have found the best path to each city as we encounter it*.
  - If we later discover we are wrong and find a better path to a particular city, then update the distance to that city.

Dijkstra's algorithm

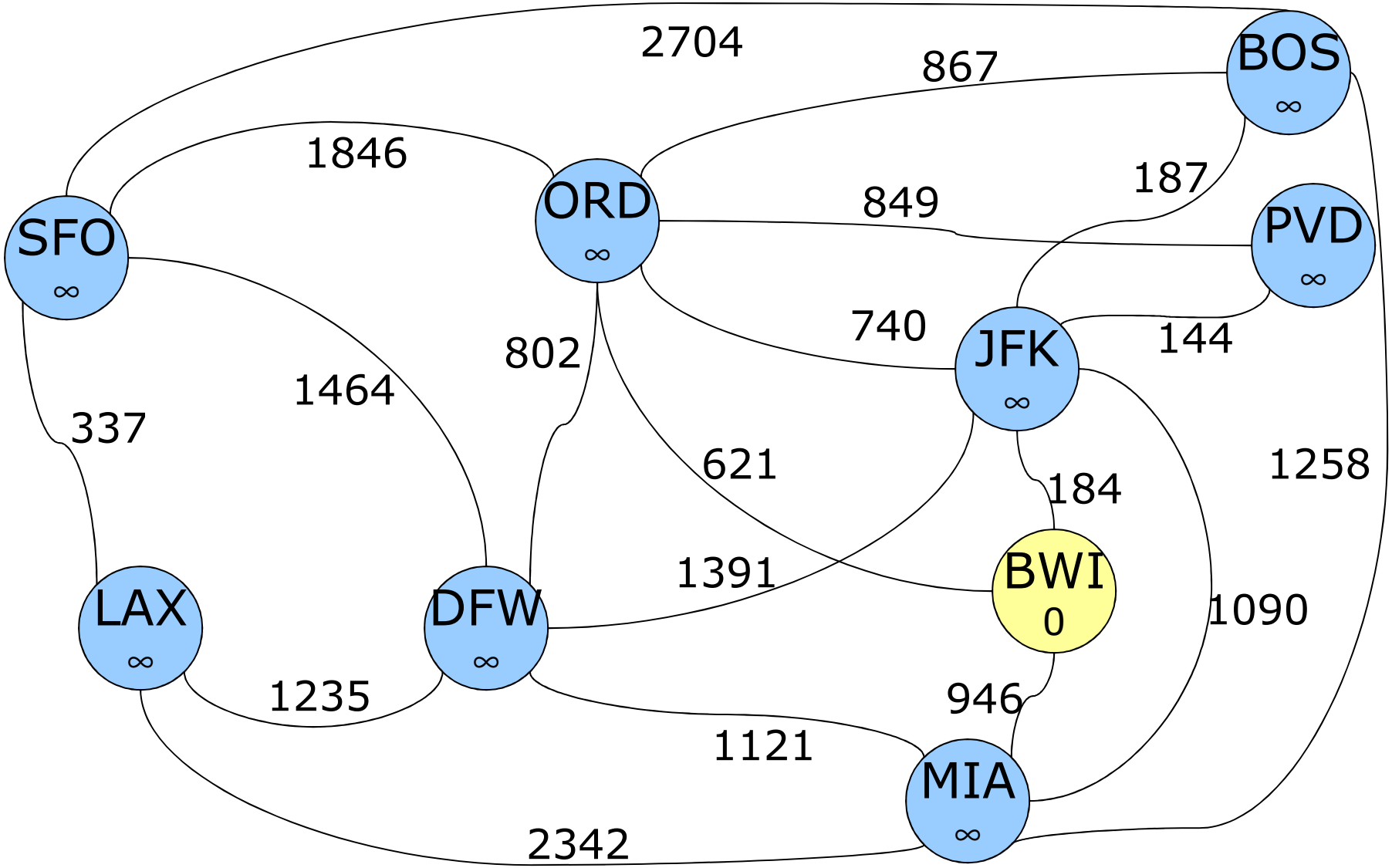
- Algorithm initialization:
  - Label each node with the distance ∞, except start node, which is labeled with distance 0.
    - D[v] is the distance label for v.
  - Put all nodes into a priority queue Q, using the distances as labels.

Dijkstra's algorithm, cont'd

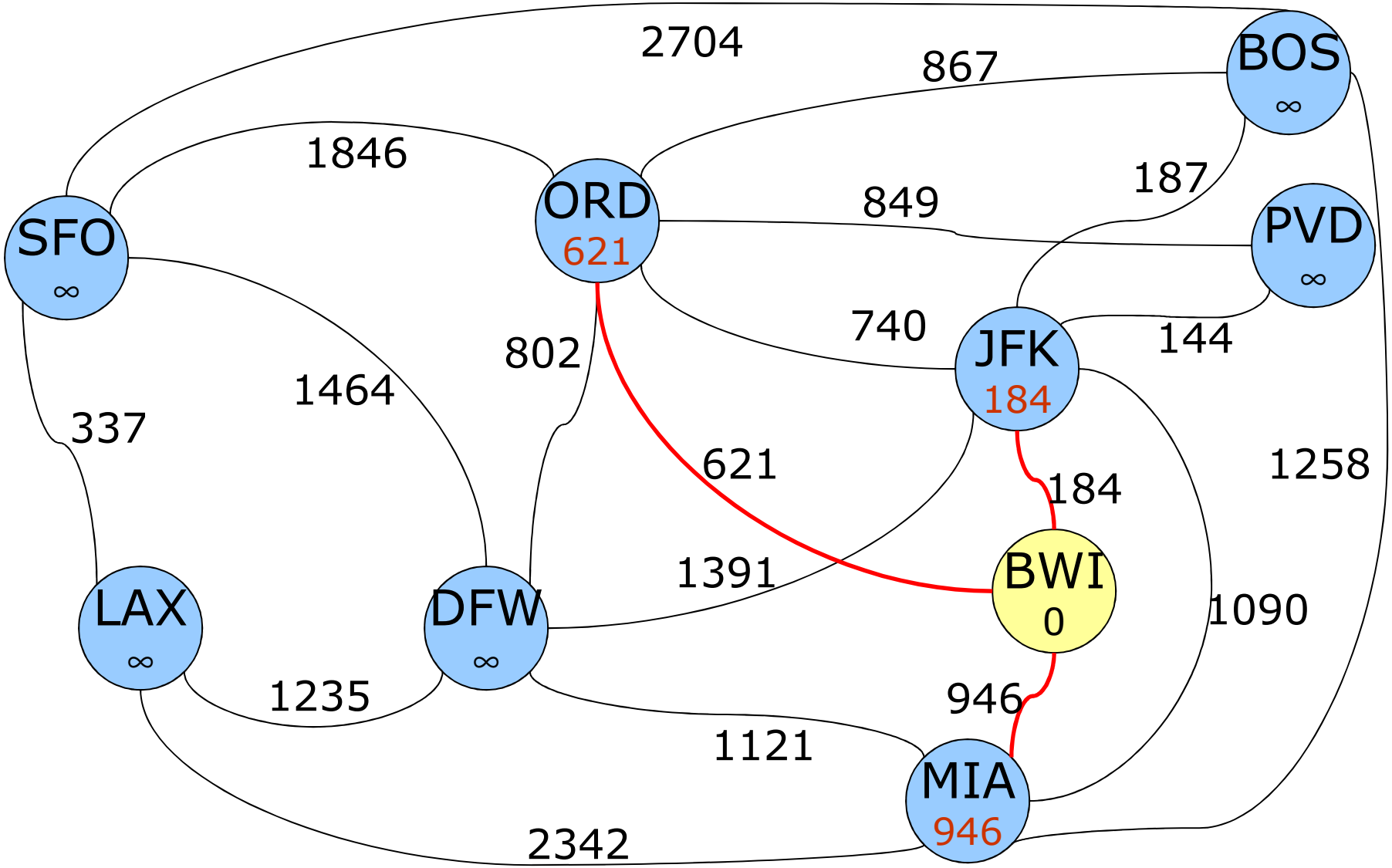
- While Q is not empty do:
  - u = Q.removeMin
  - for each node z one hop away from u do:
    - if D[u] + miles(u,z) < D[z] then
      - D[z] = D[u] + miles(u,z)
      - change key of z in Q to D[z]
- Note use of priority queue(Heap) allows "finished" nodes to be found quickly (in $O(\log |V|)$ time).

An Example

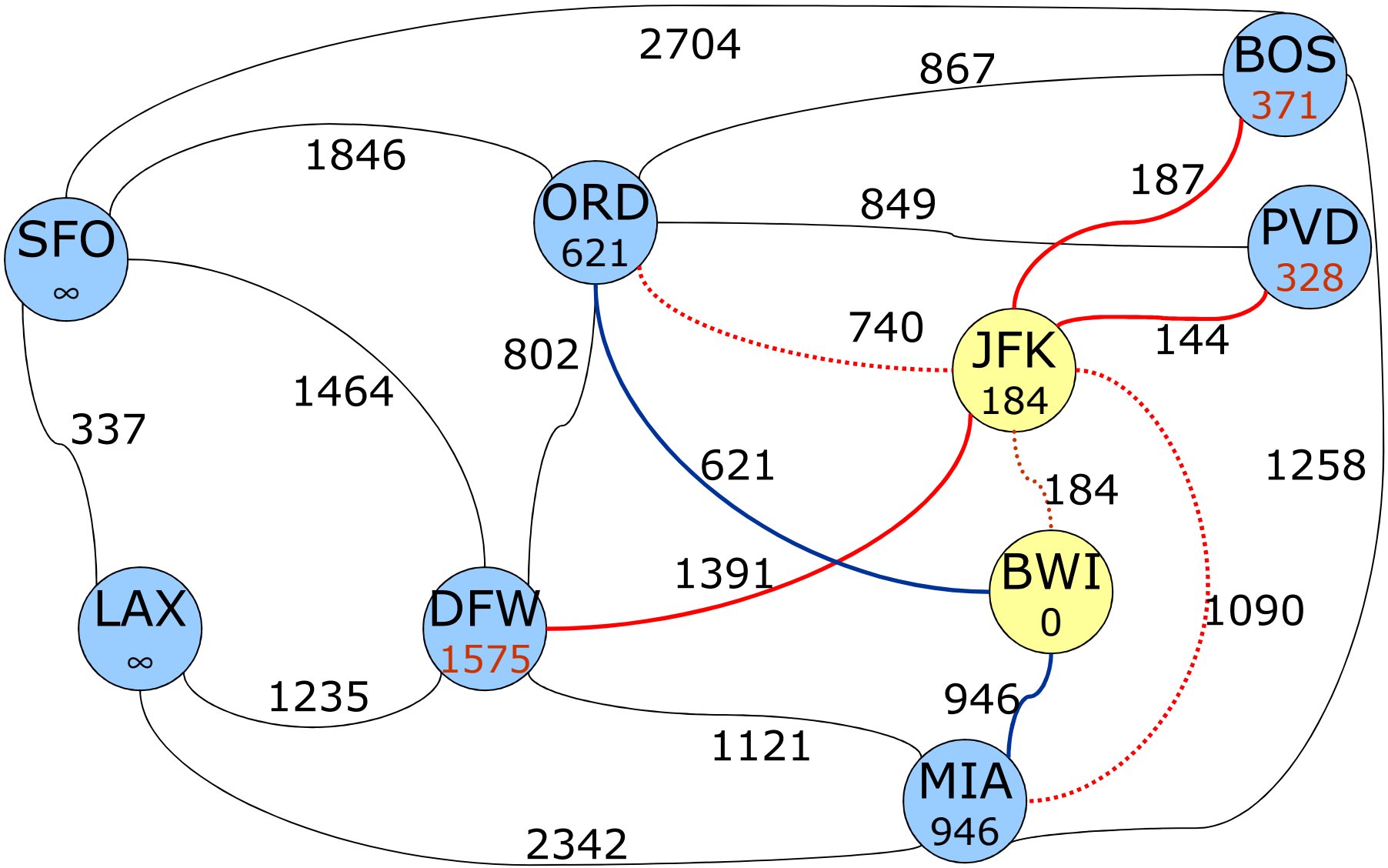
Shortest mileage from BWI



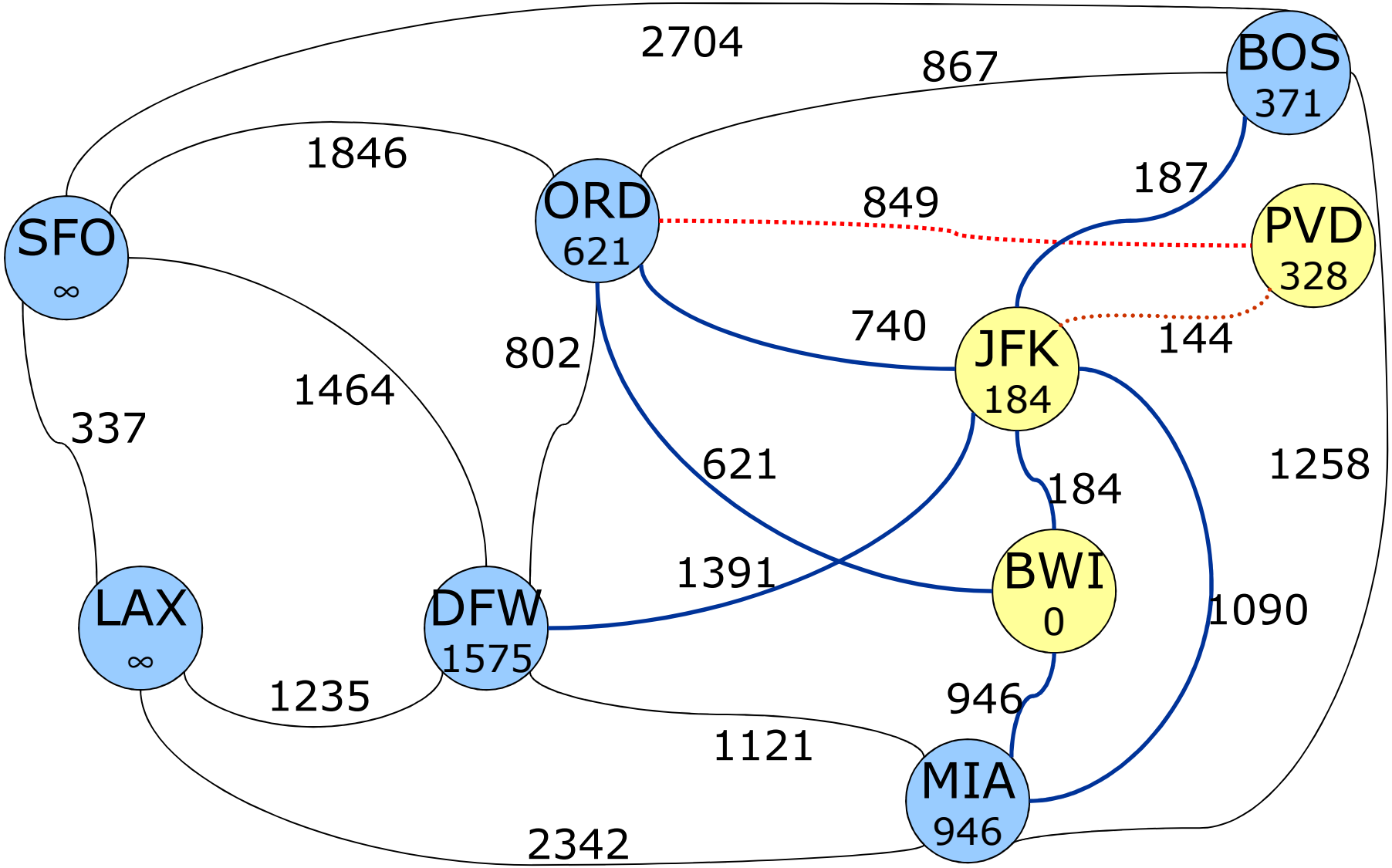
Shortest mileage from BWI



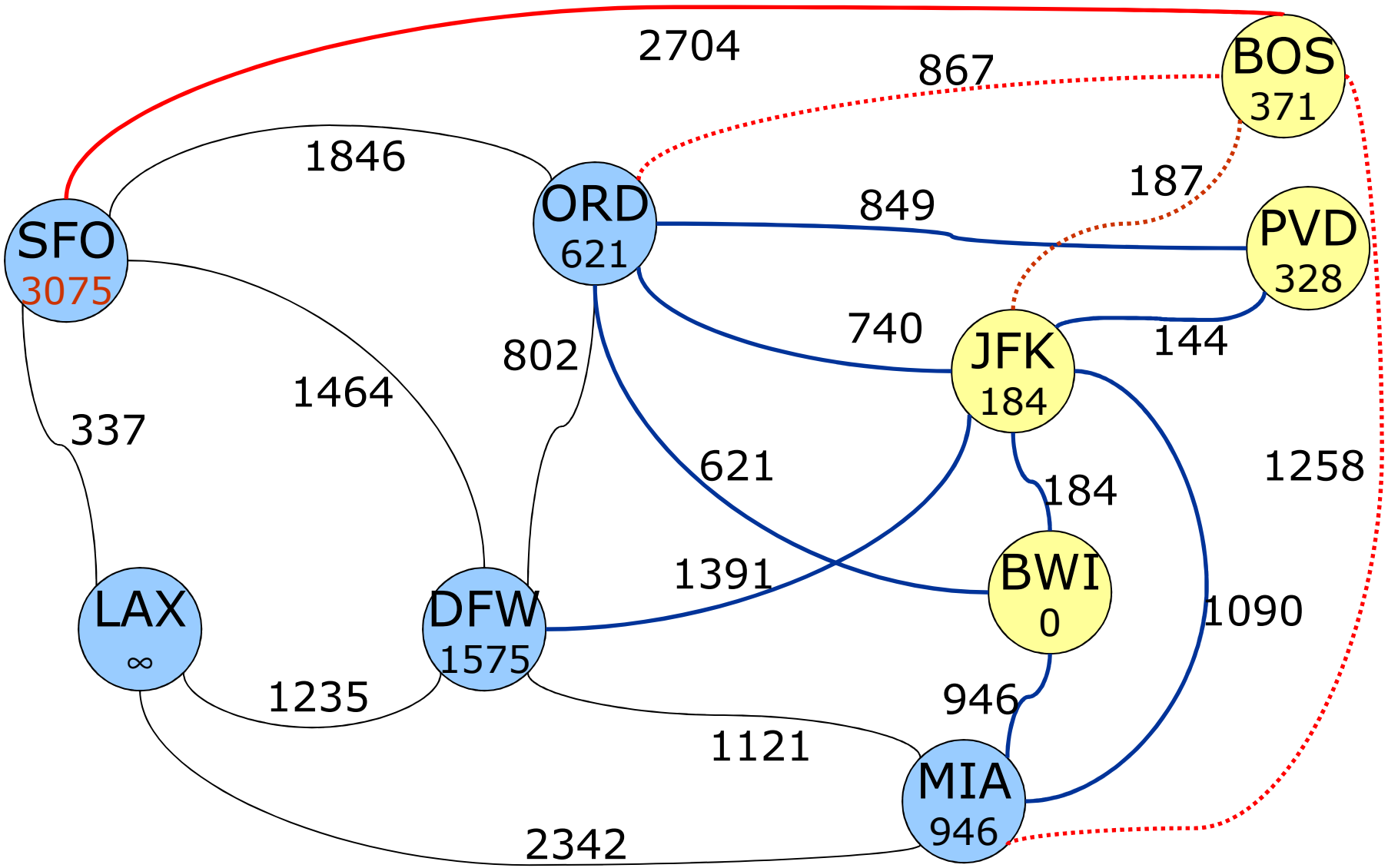
Shortest mileage from BWI



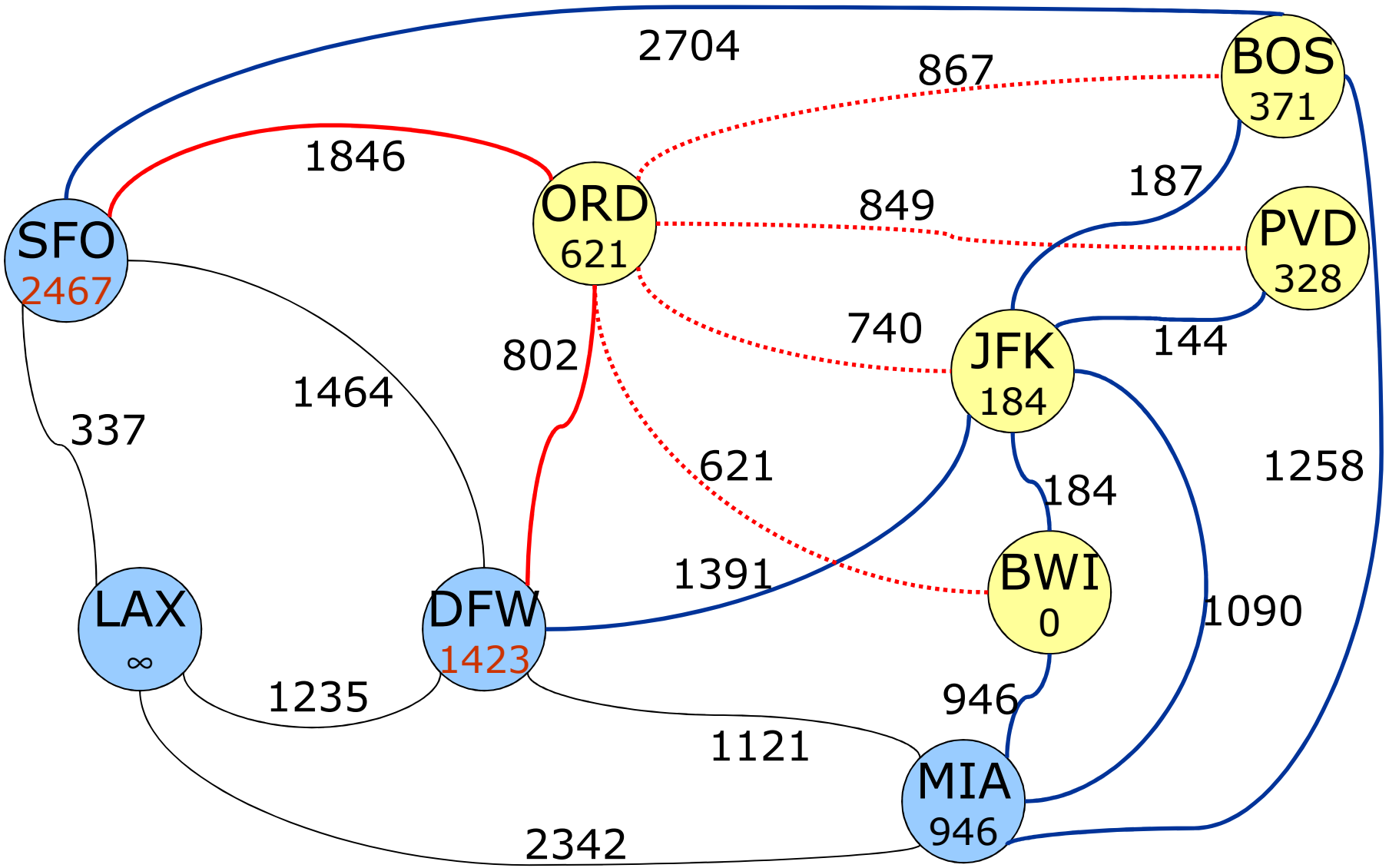
Shortest mileage from BWI



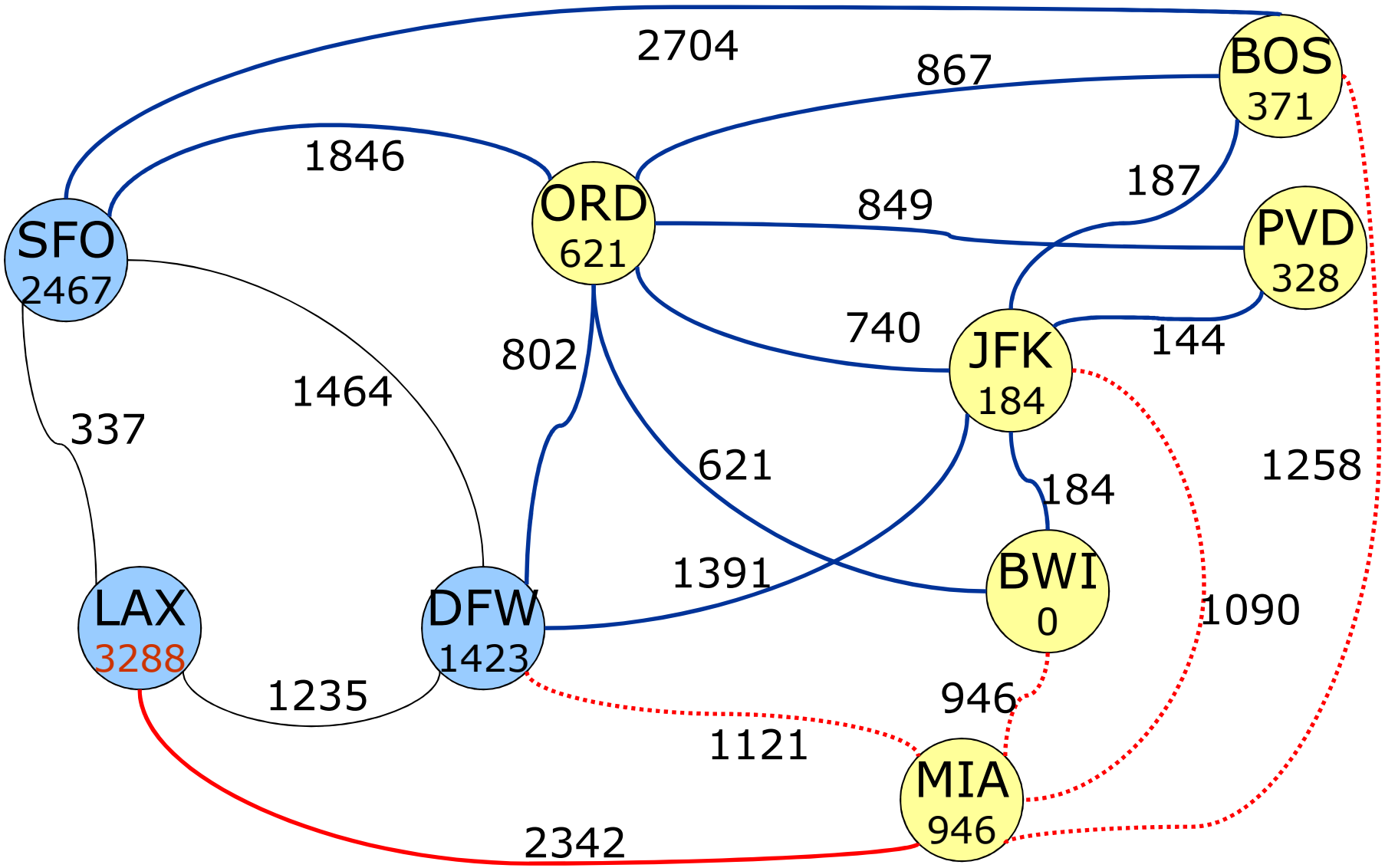
Shortest mileage from BWI



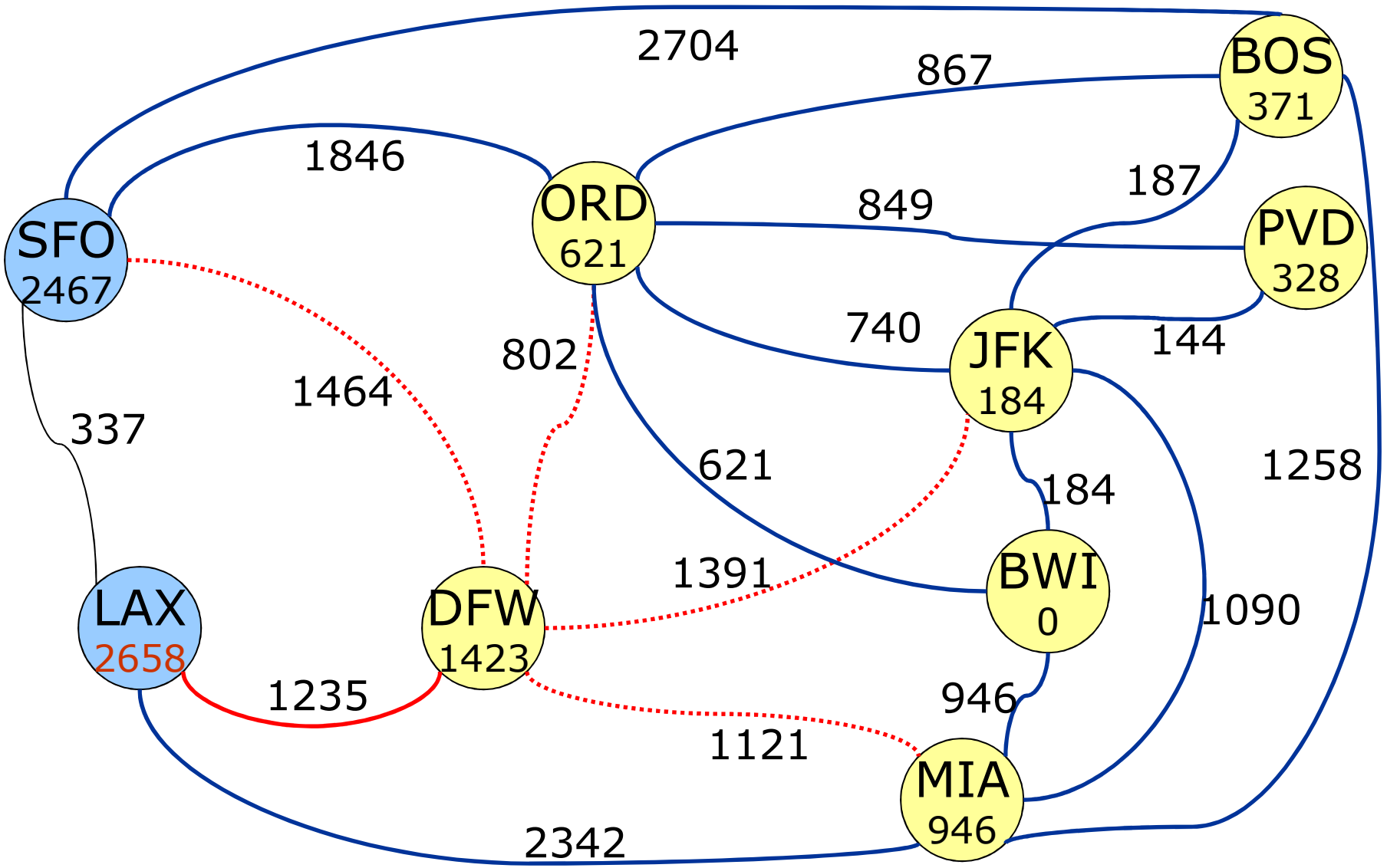
Shortest mileage from BWI



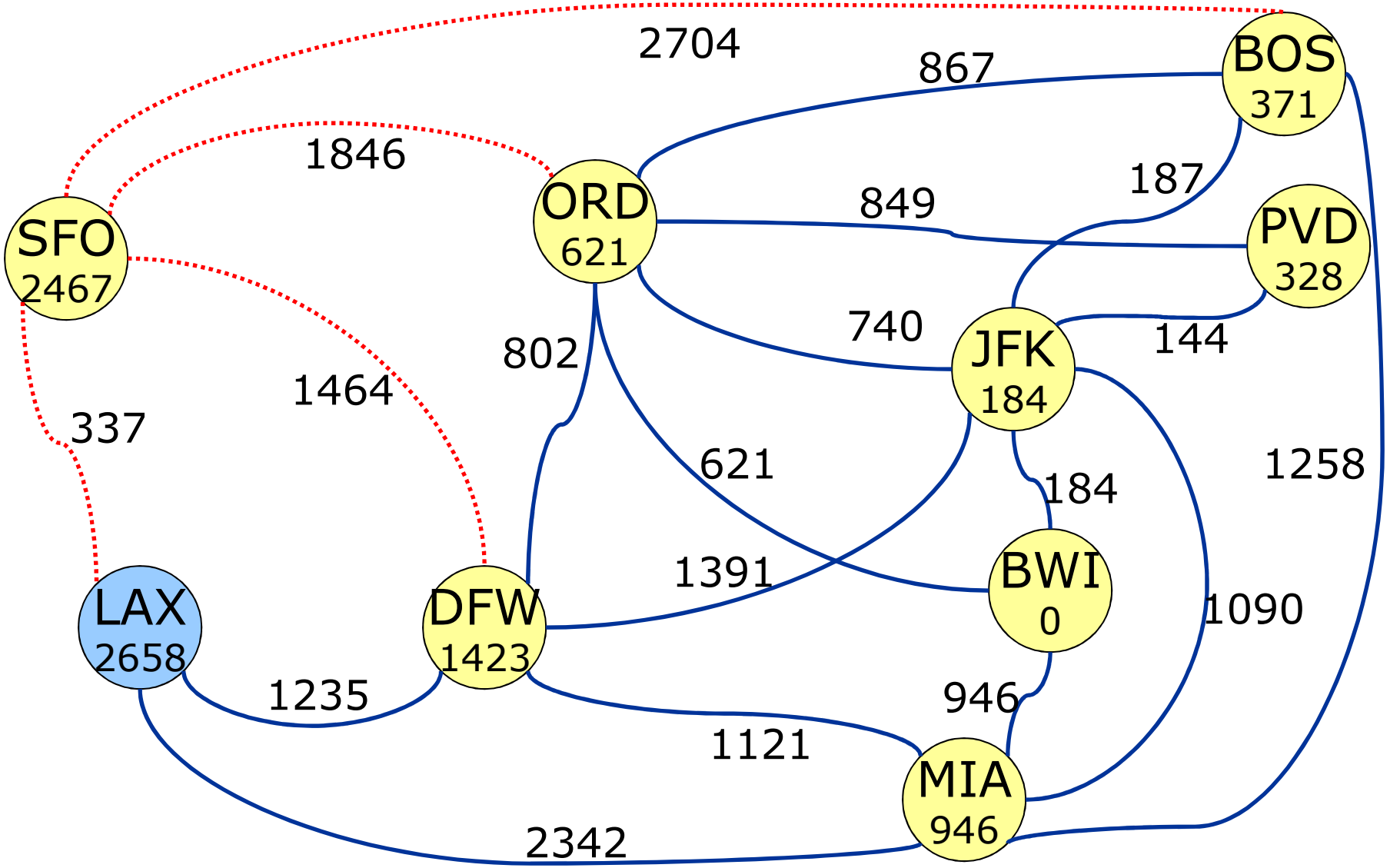
Shortest mileage from BWI



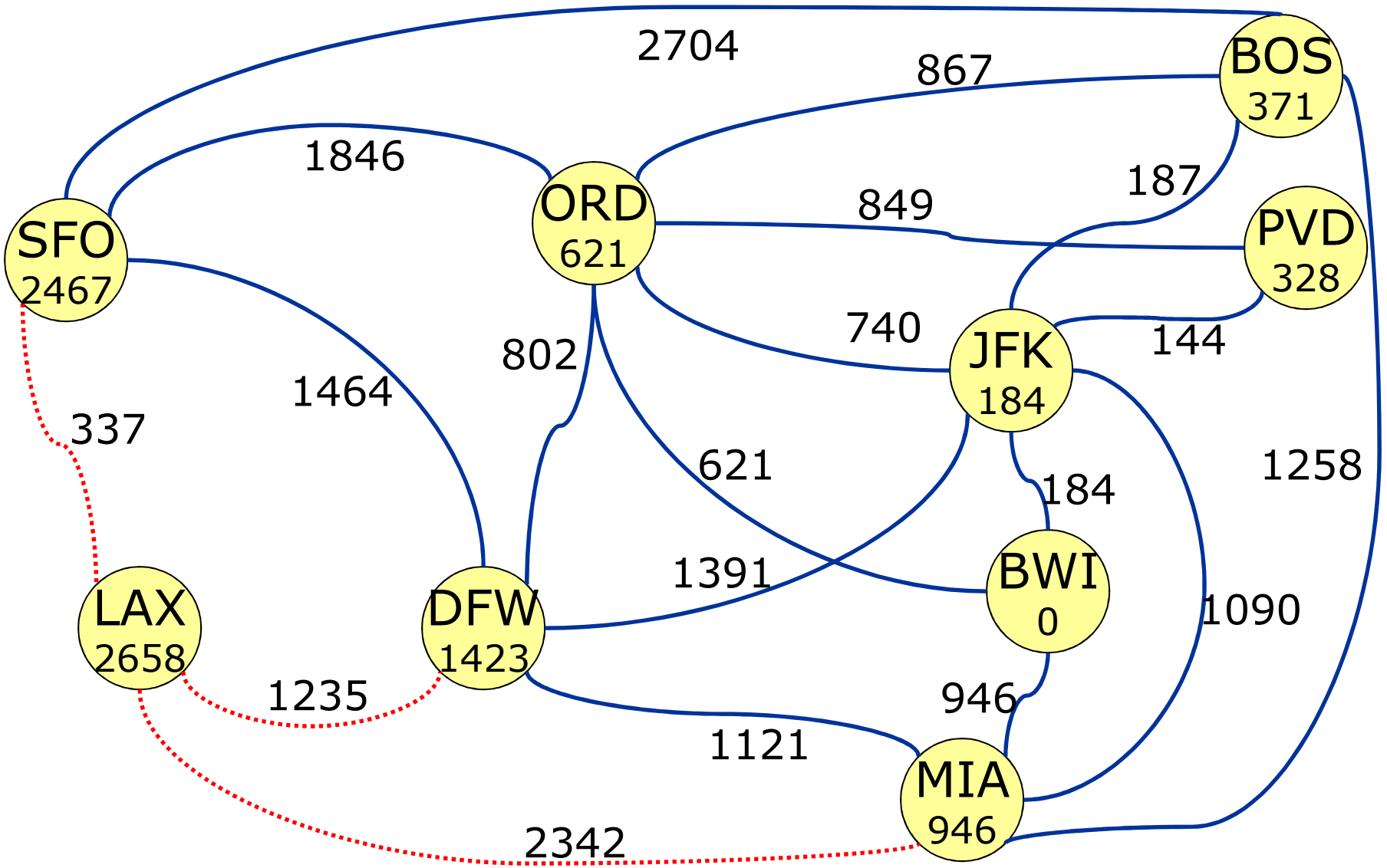
Shortest mileage from BWI



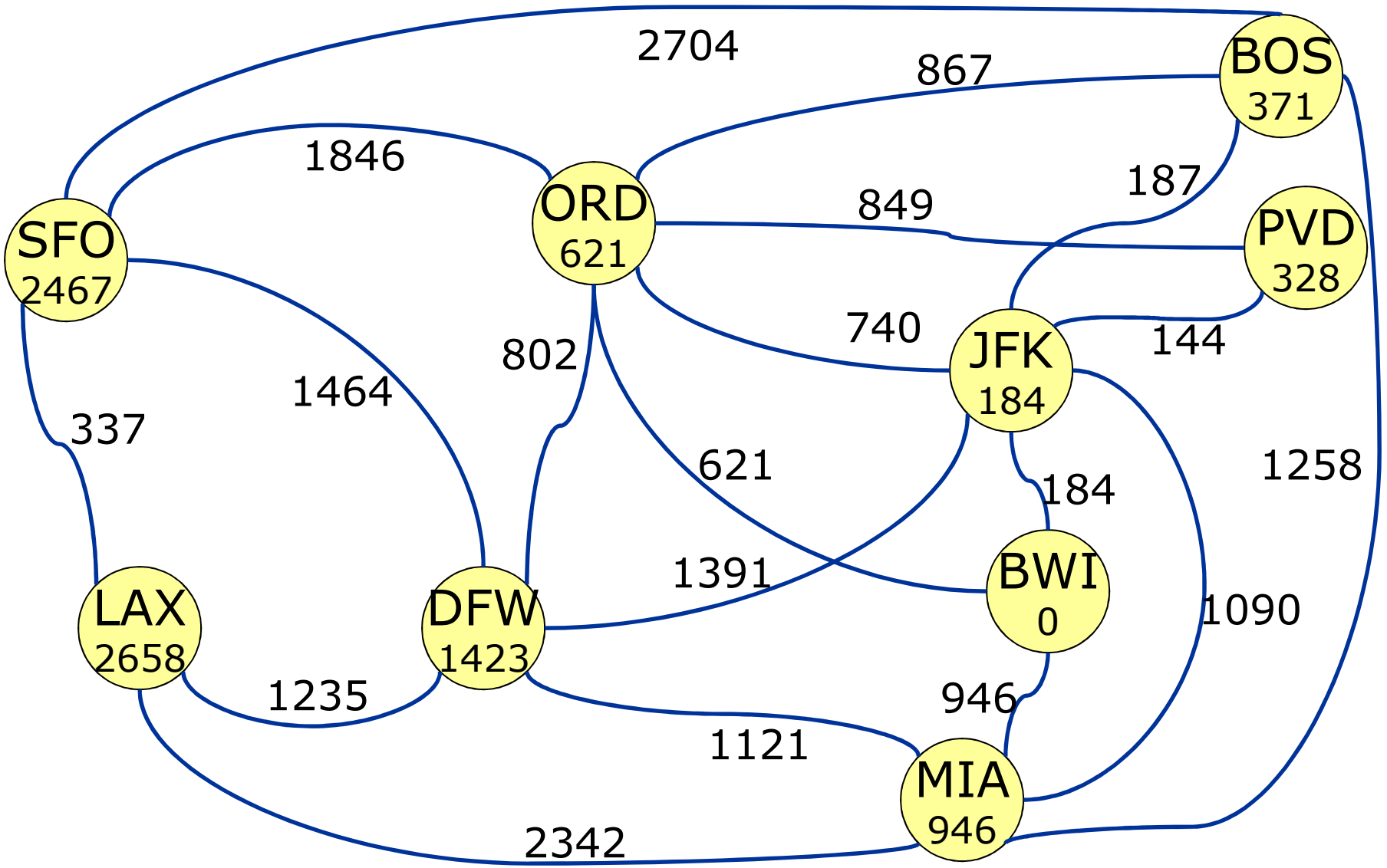
Shortest mileage from BWI



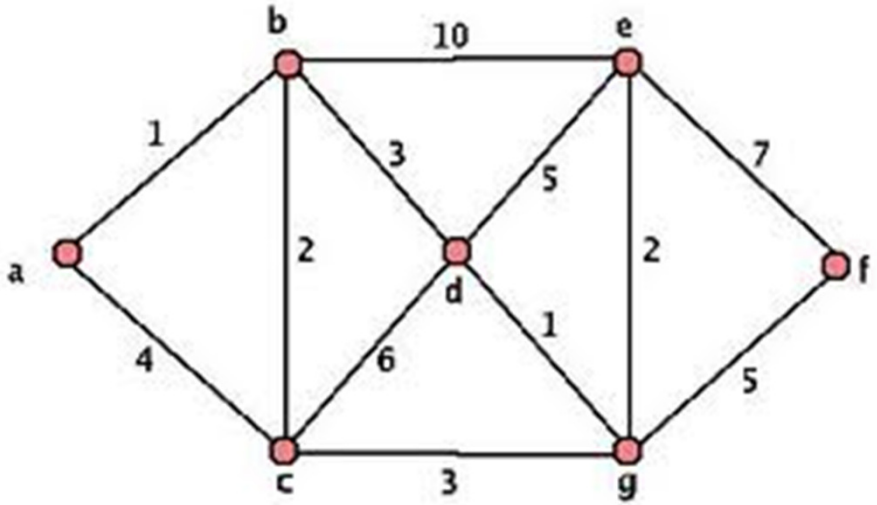
Shortest mileage from BWI



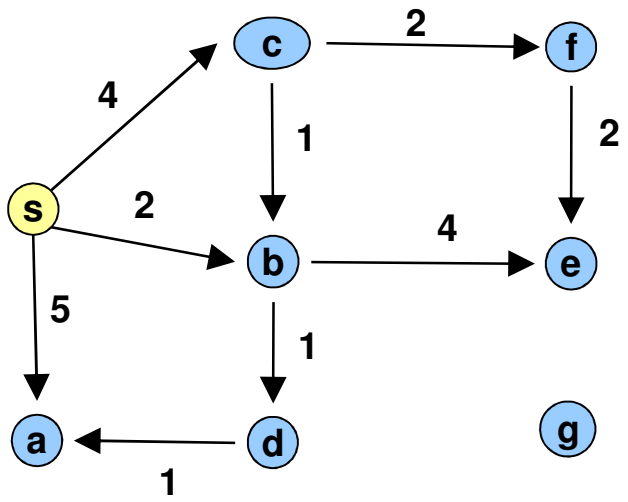
Shortest mileage from BWI



Classwork



Find the Shortest Paths from S



Dijkstra's Algorithm is greedy

1. Optimization problem
   - Of the many feasible solutions, finds the *minimum* or *maximum* solution.
2. Can only proceed in stages
   - no direct solution available
3. Greedy-choice property:

   A locally optimal (greedy) choice will lead to a globally optimal solution.
4. Optimal substructure:

   An optimal solution contains within it optimal solutions to subproblems

Features of Dijkstra's Algorithm

- "Visits" every vertex only once, when it becomes the vertex with minimal distance amongst those still in the priority queue
- Distances may be revised **multiple times**: current values represent 'best guess' based on our observations so far
- Once a vertex is finalized we are guaranteed to have found the shortest path to that vertex

Implementation

Heap is necessary to findMin

Initialization: O(n)

Visitation loop: n calls

- deleteMin(): O(log n)
- Each edge is considered only once during entire execution, for a total of e updates of the priority queue, each O(log n)

Overall cost: O((n+e) log n)

Question

- Dijkstra's only finds the length of the shortest path
- Is it possible to modify the Dijkstra's to actually find out the nodes in the shortest path?