

Graph Algorithms

15-121

Introduction to Data Structures

Ananda Gunawardena

8/2/2011

In this lecture..

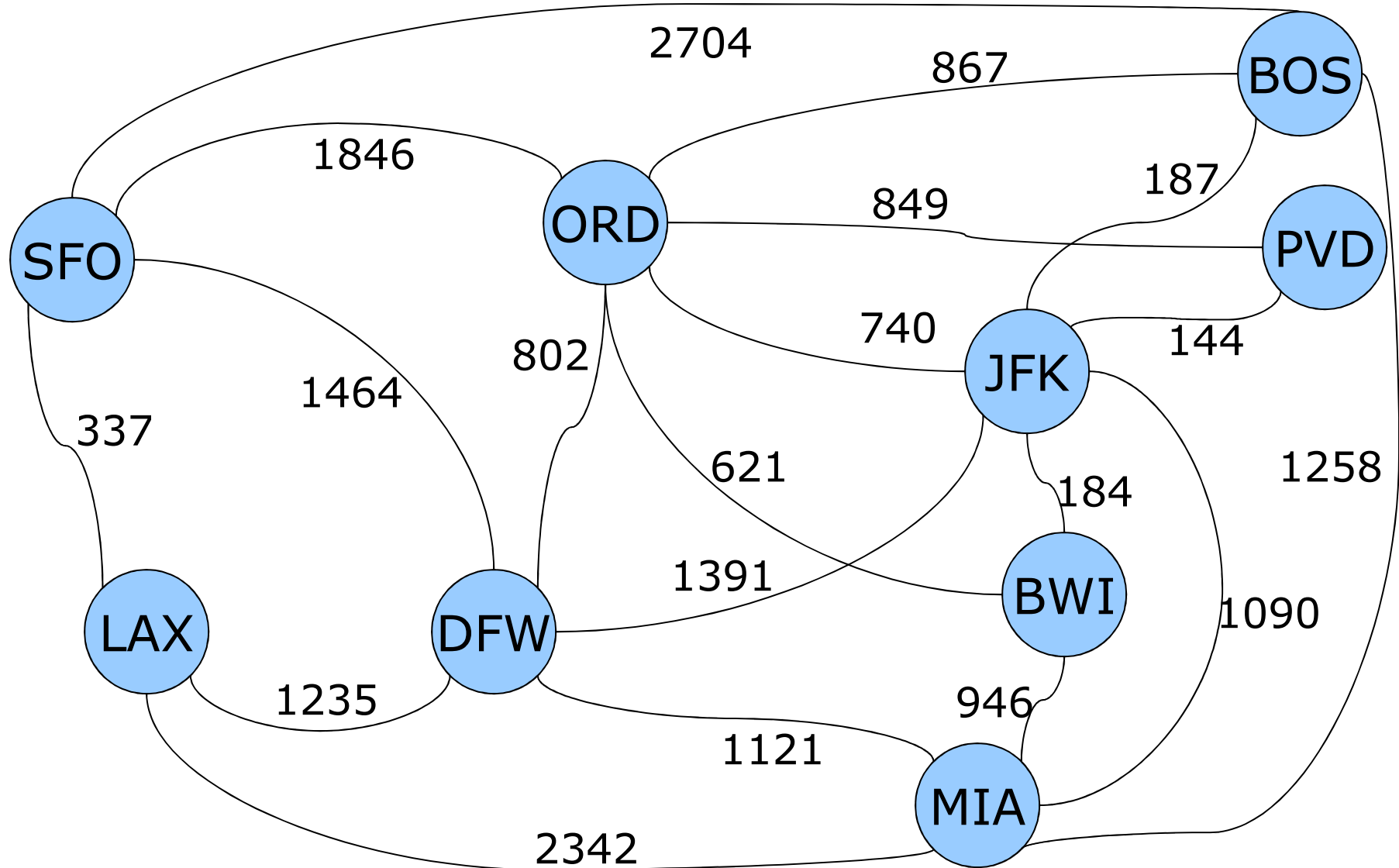


- Main idea is finding the Shortest Path between two points in a Graph
- We will look at
 - Graphs with non negative cost edges
 - Dijkstra's Algorithm



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...

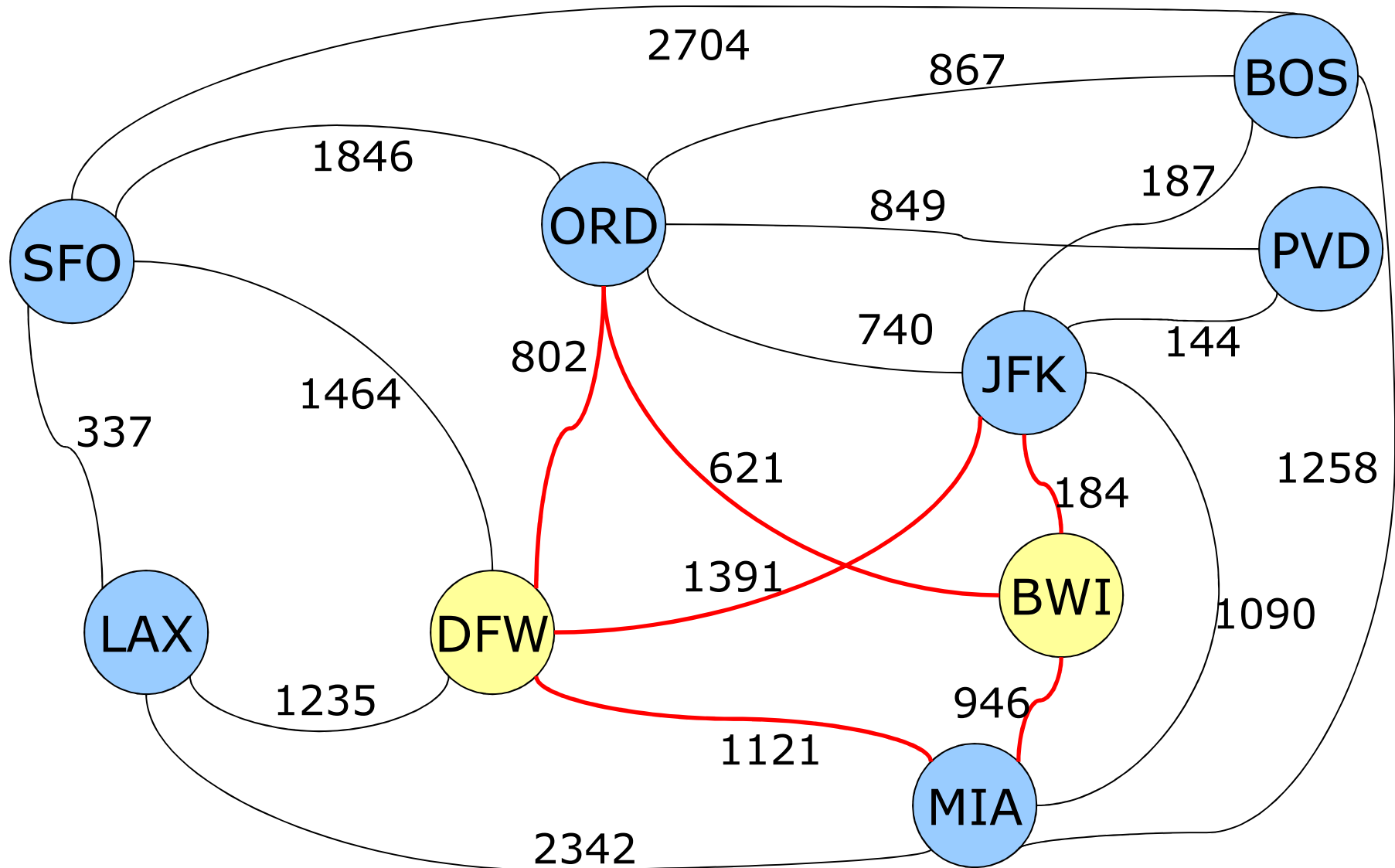
*Shortest Path Algorithm
for
Non-negative weights
(Dijkstra's Algorithm)*

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.

Priority Queue (PQ) table:

	A	B	C	D	E	F
	0	∞	∞	∞	∞	∞

$$D[A] = 0$$
$$D[B] = \infty$$

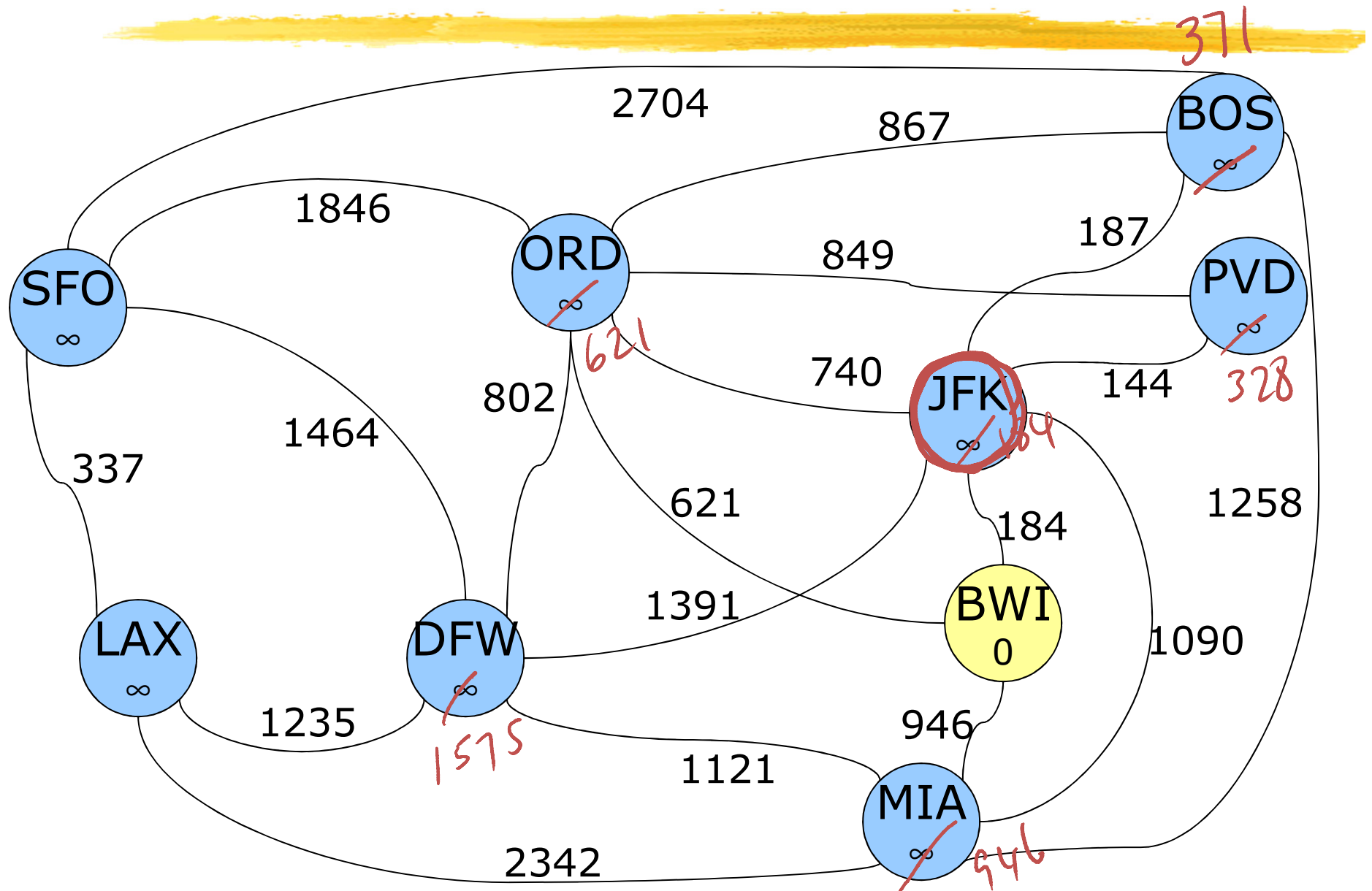
Dijkstra's algorithm, cont'd

- While Q is not empty do:
 - $u = Q.\text{removeMin}$
 - for each node z one hop away from u do:
 - if $D[u]$ + $\text{miles}(u,z)$ < $D[z]$ then
 - $D[z] = D[u] + \text{miles}(u,z)$
 - 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).

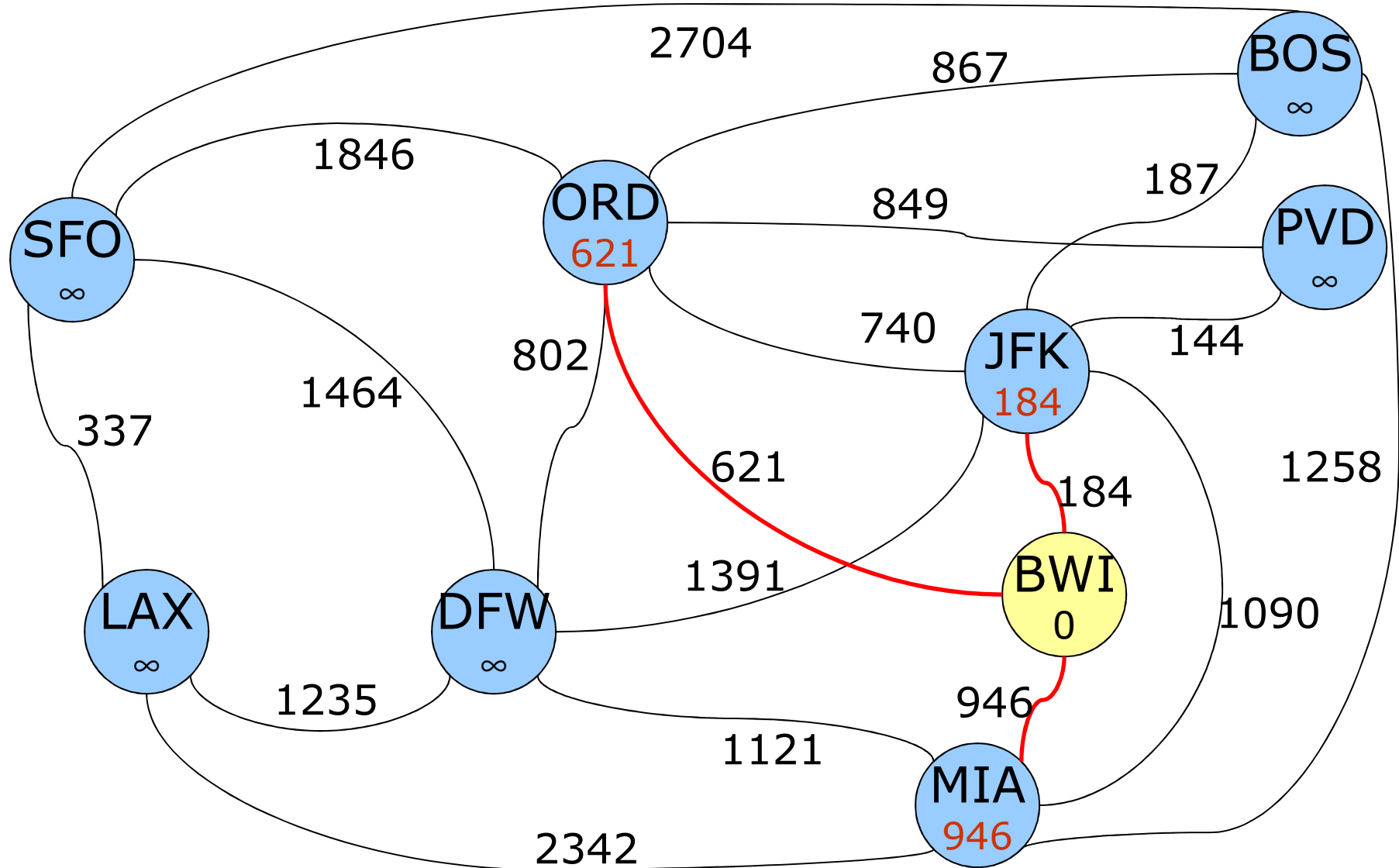


An Example

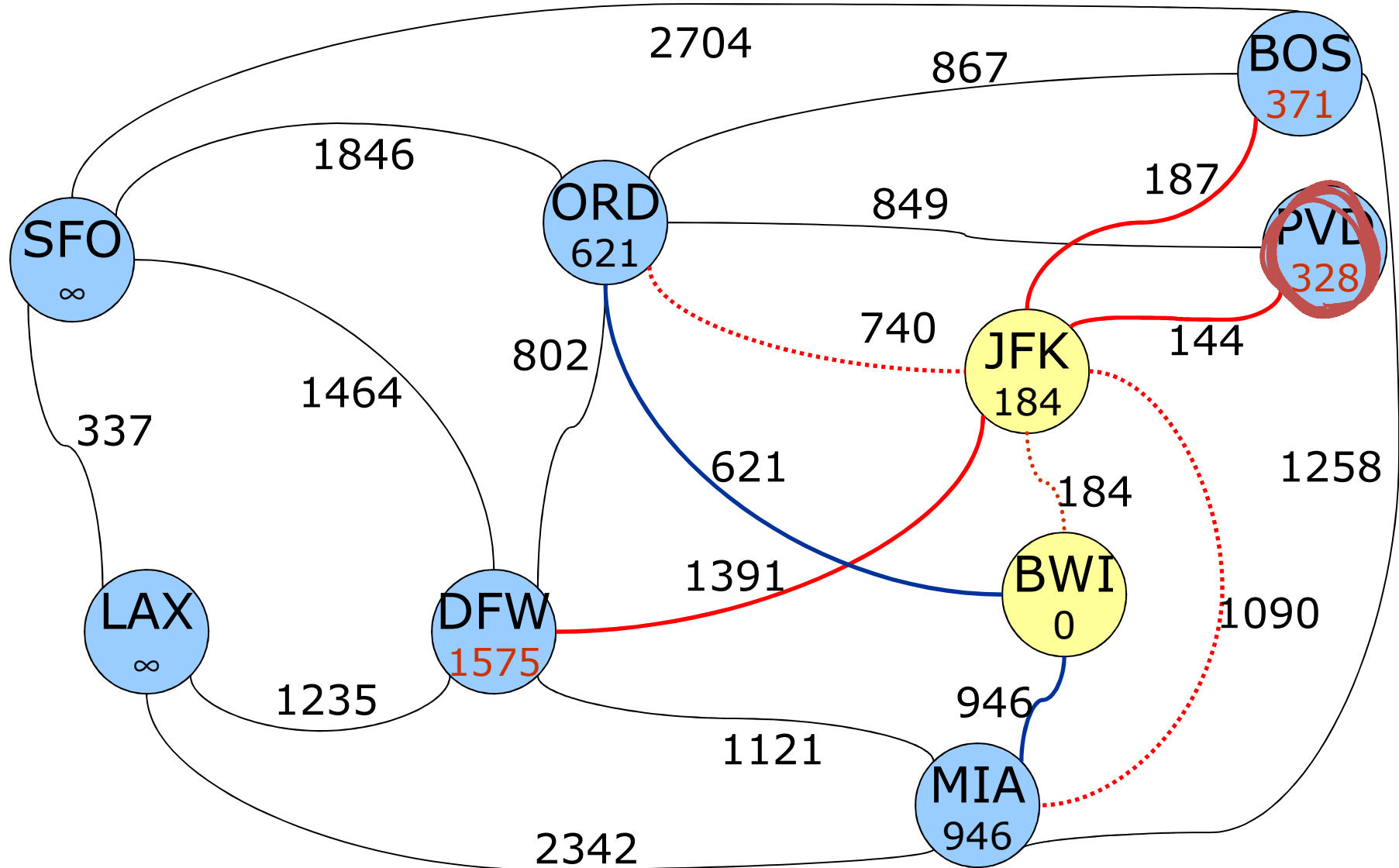
Shortest mileage from BWI



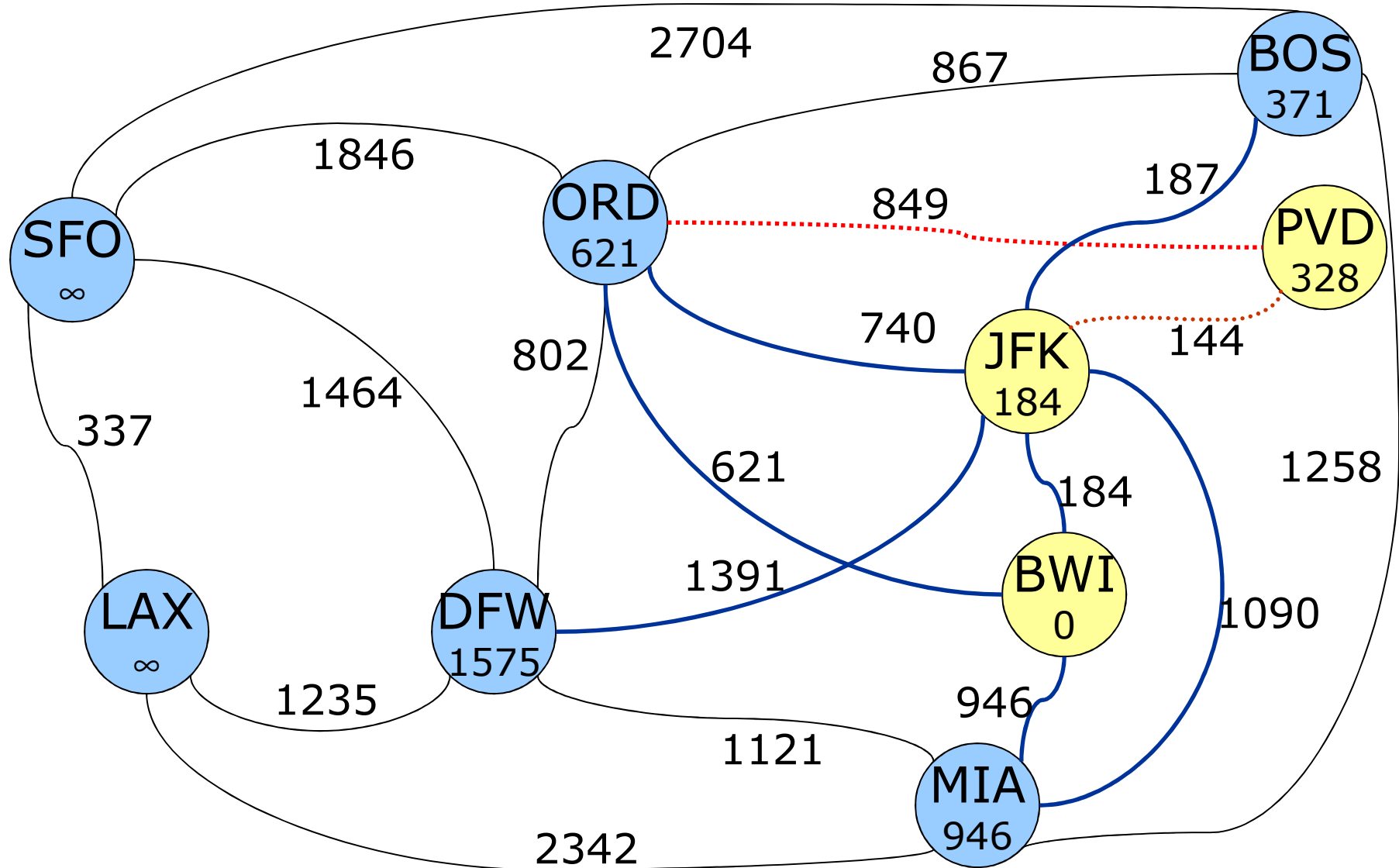
Shortest mileage from BWI



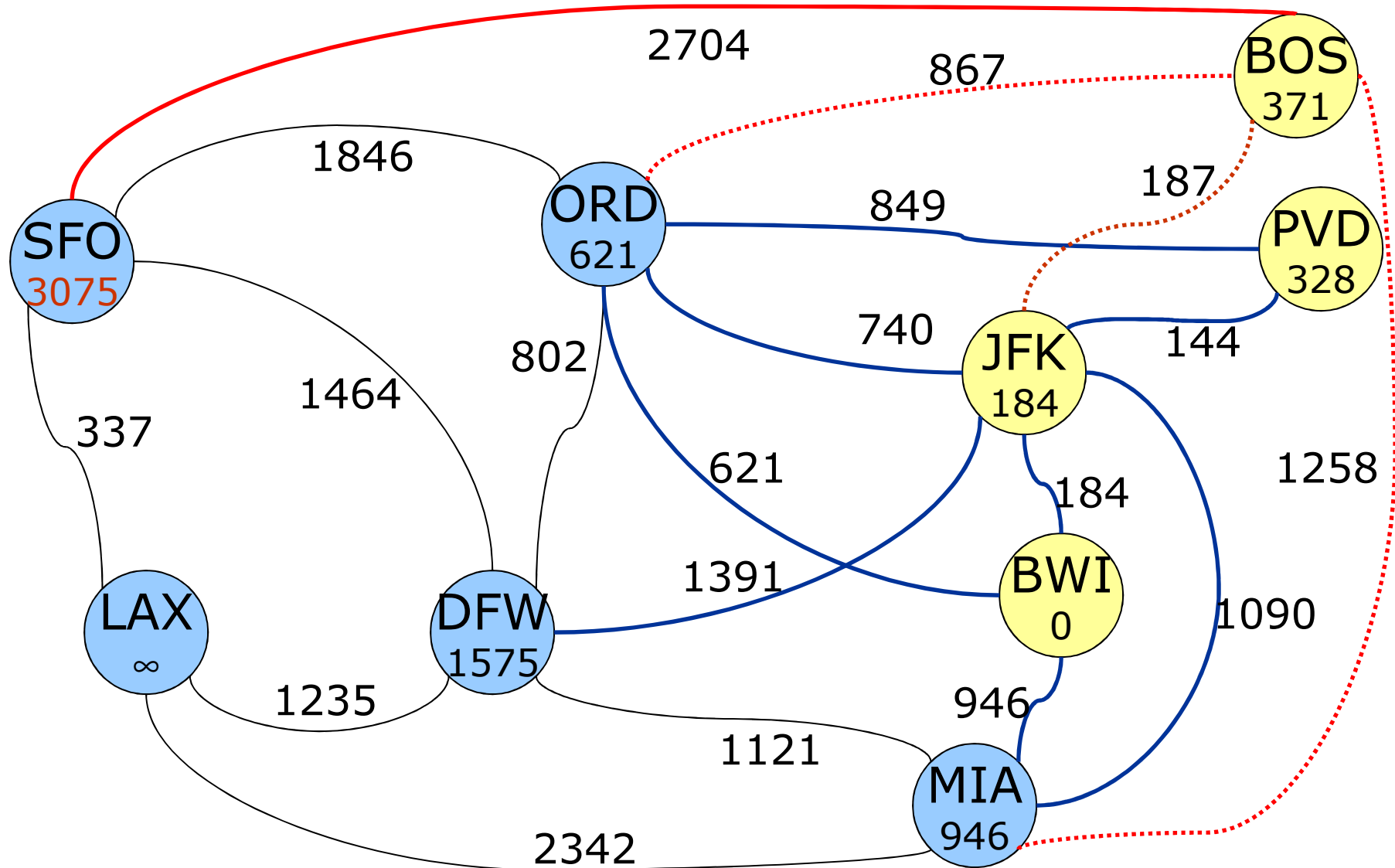
Shortest mileage from BWI



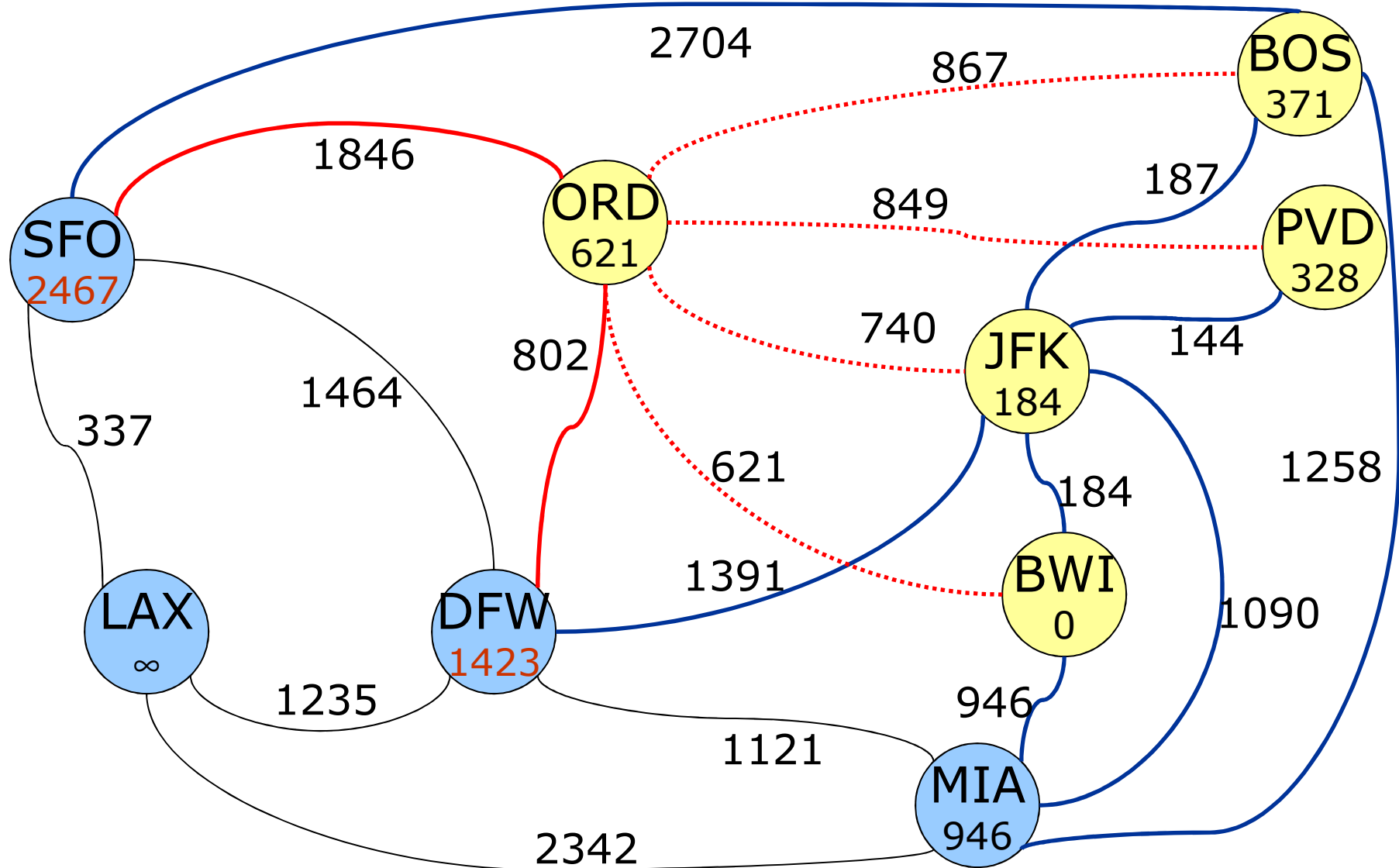
Shortest mileage from BWI



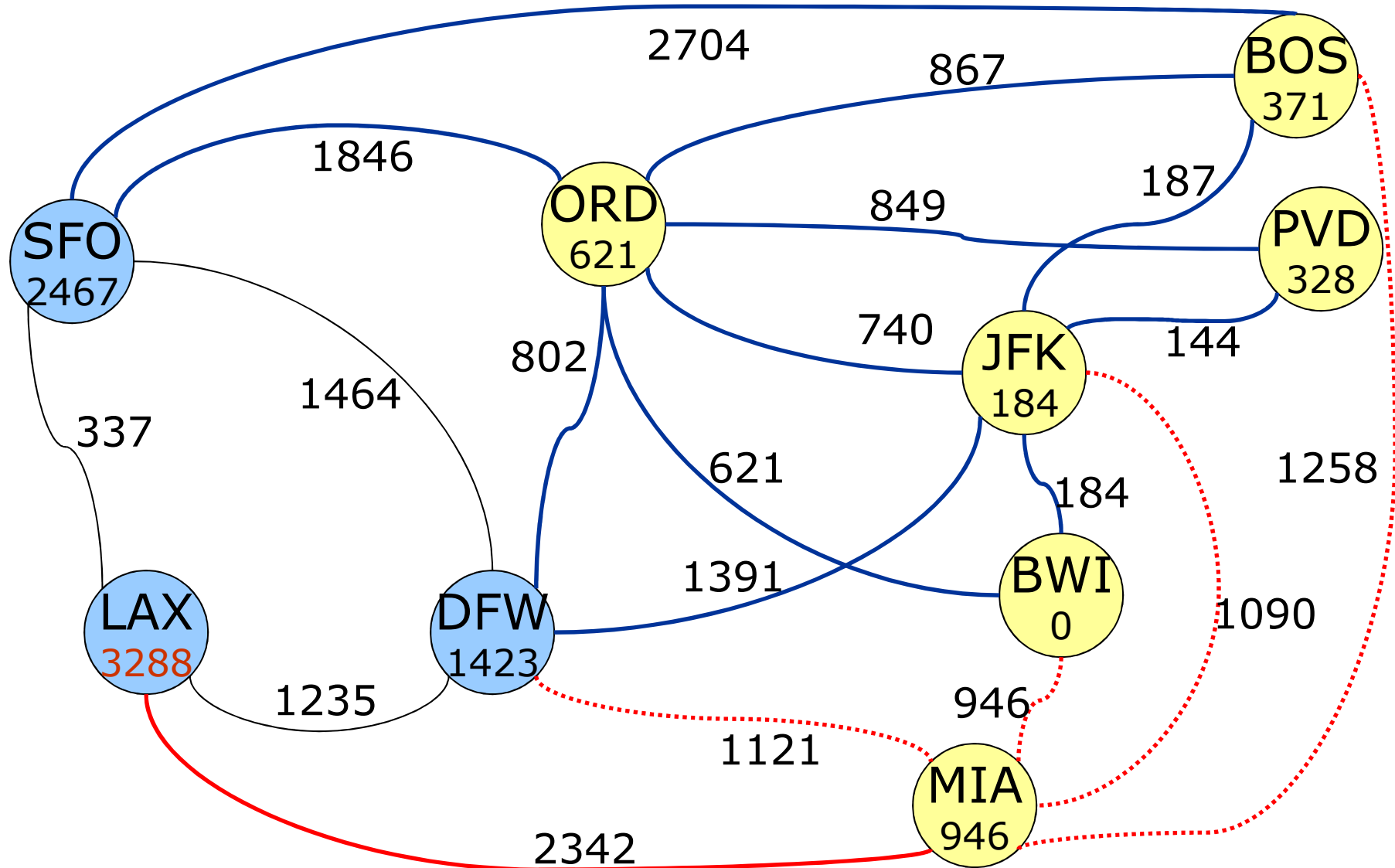
Shortest mileage from BWI



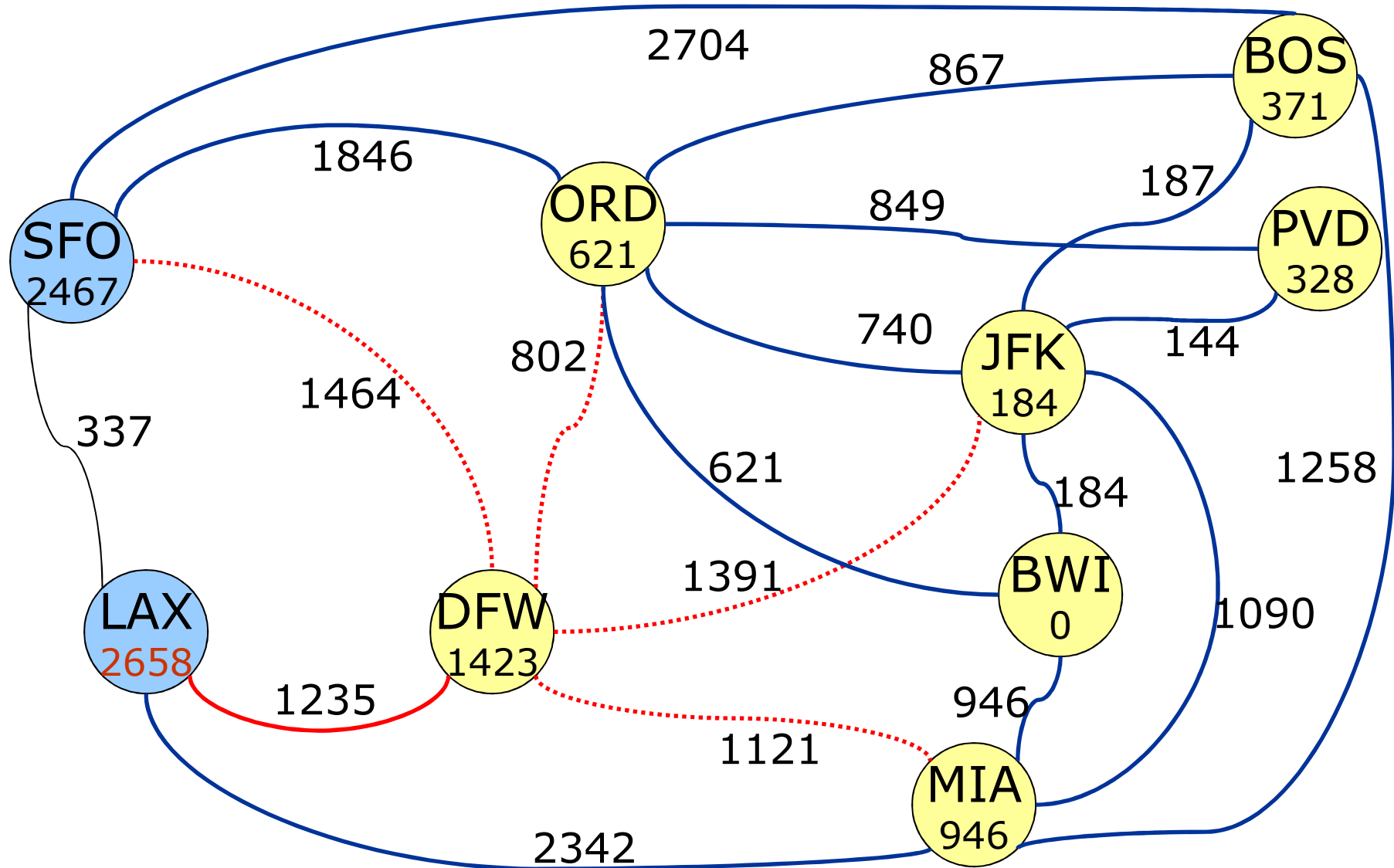
Shortest mileage from BWI



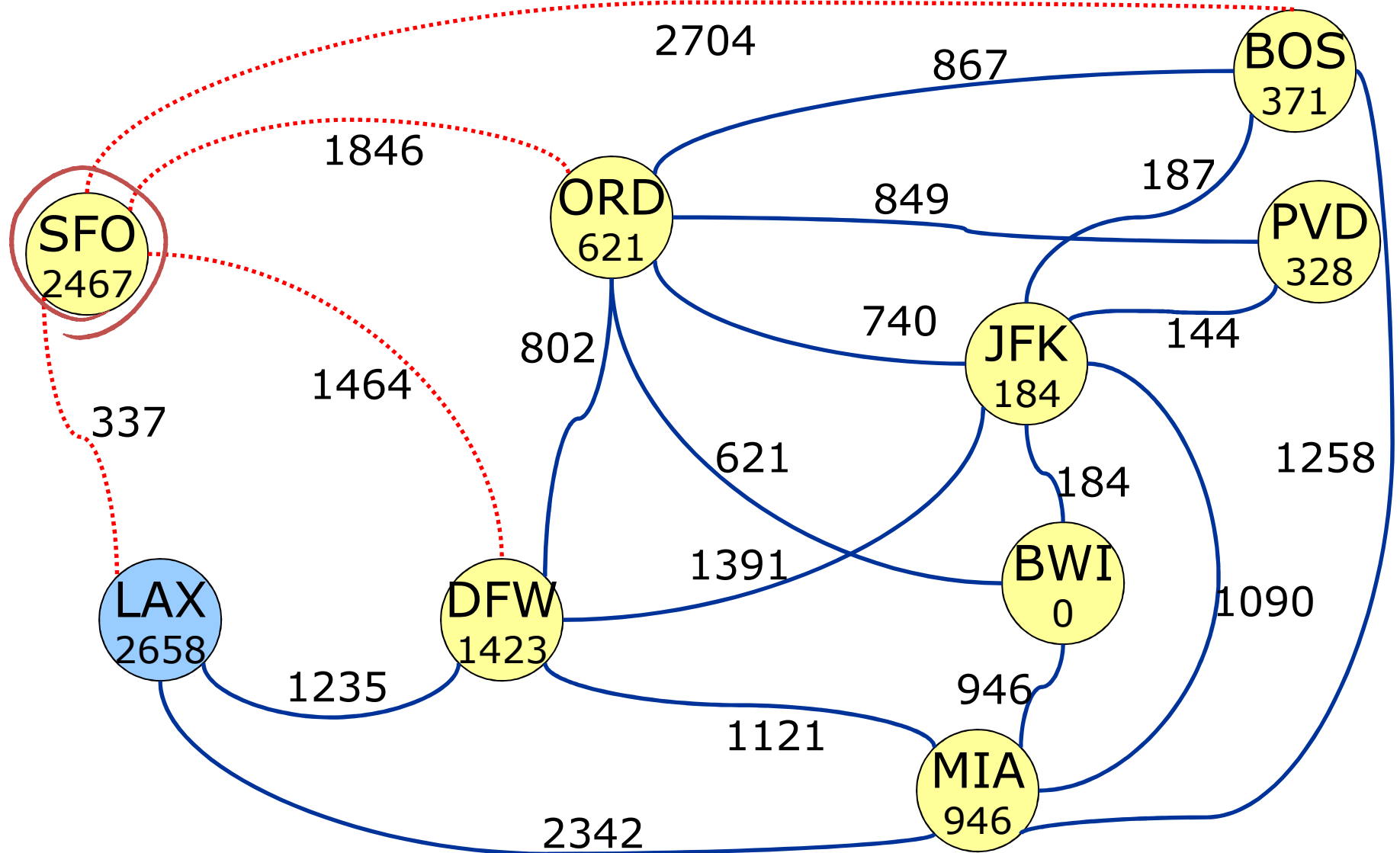
Shortest mileage from BWI



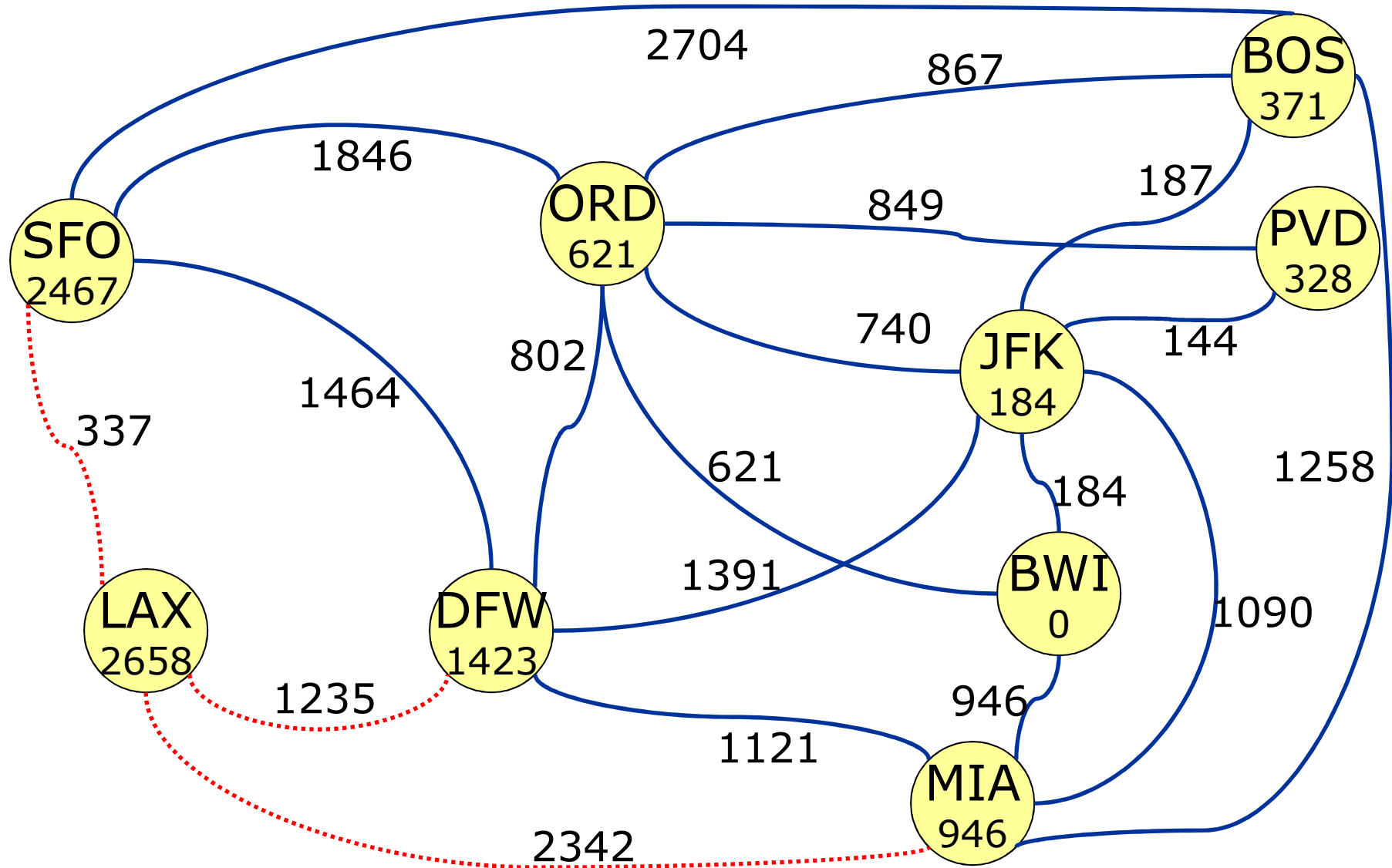
Shortest mileage from BWI



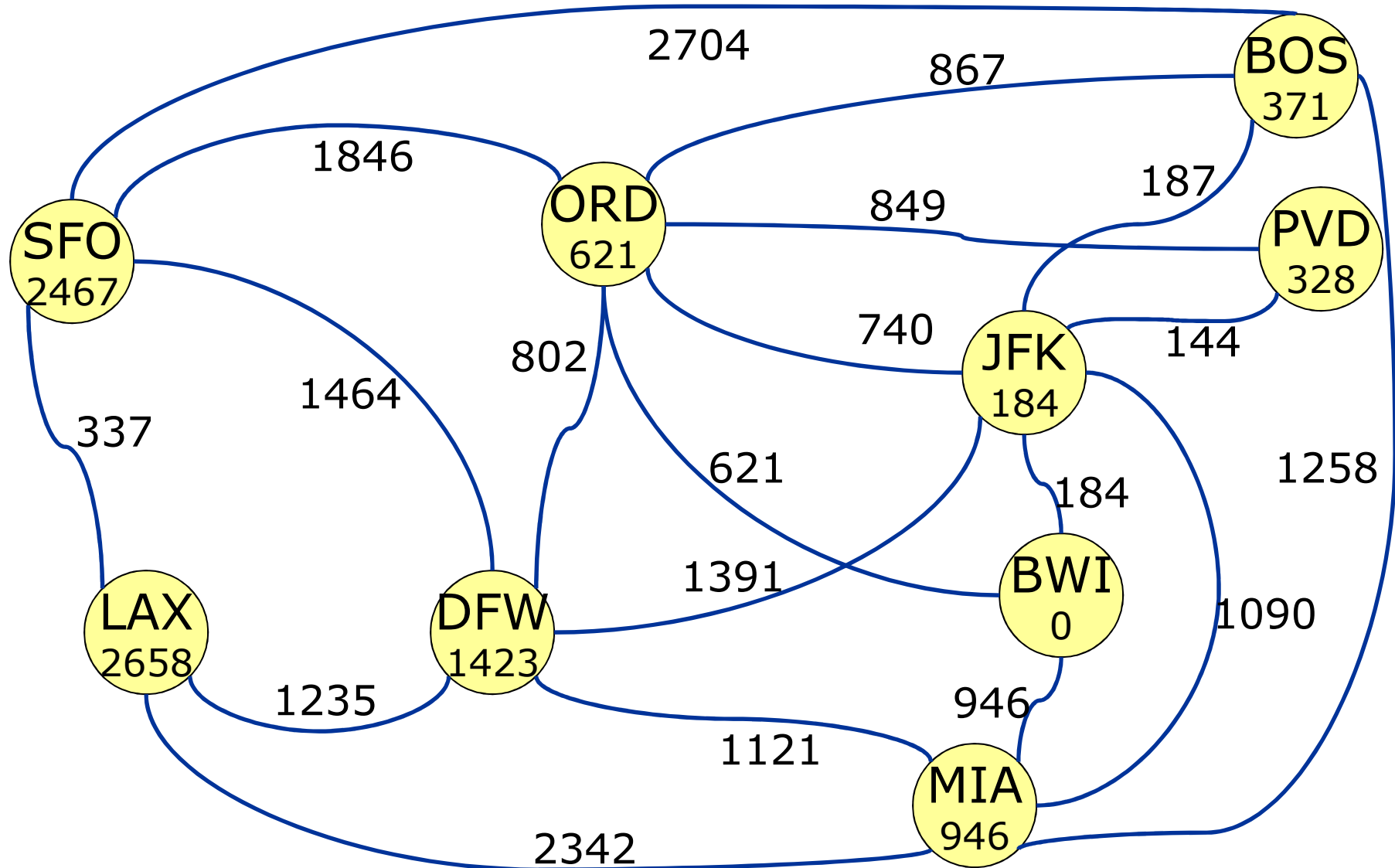
Shortest mileage from BWI



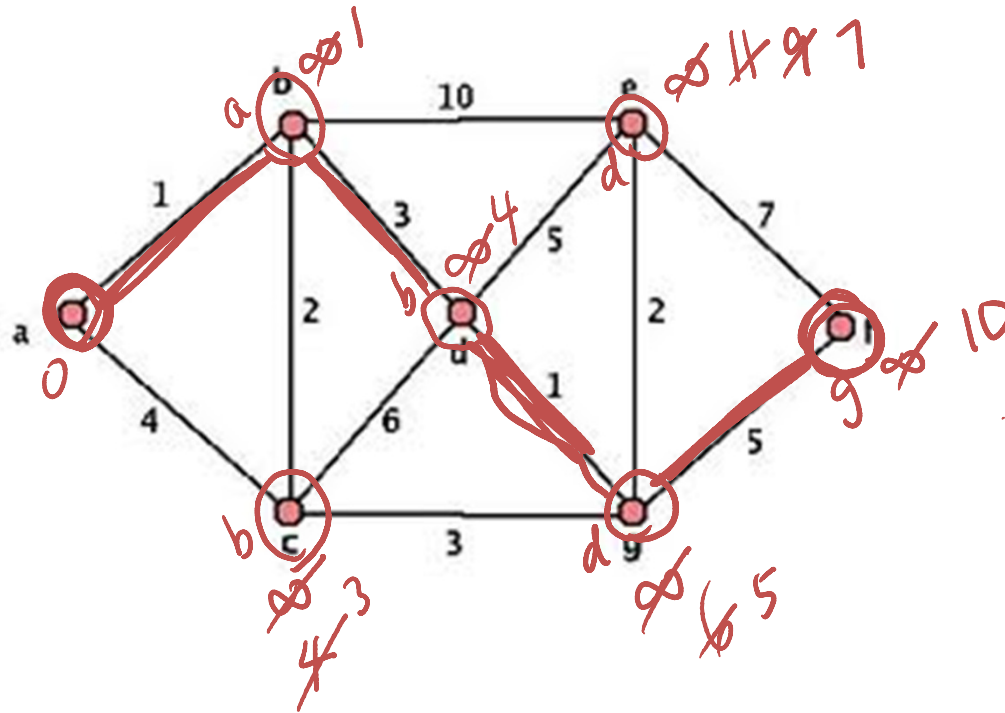
Shortest mileage from BWI



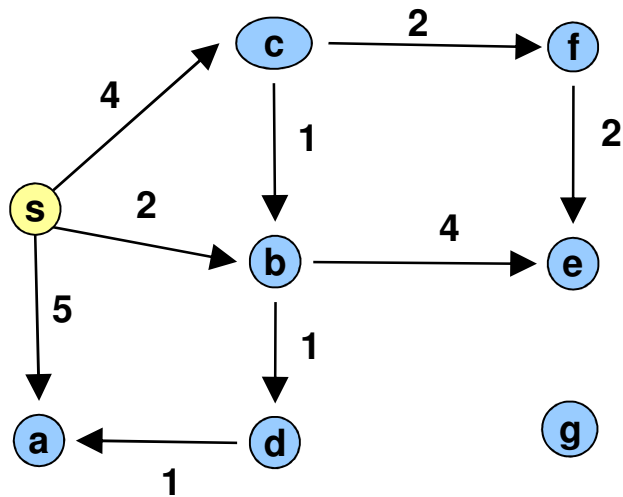
Shortest mileage from BWI



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: $O(n)$

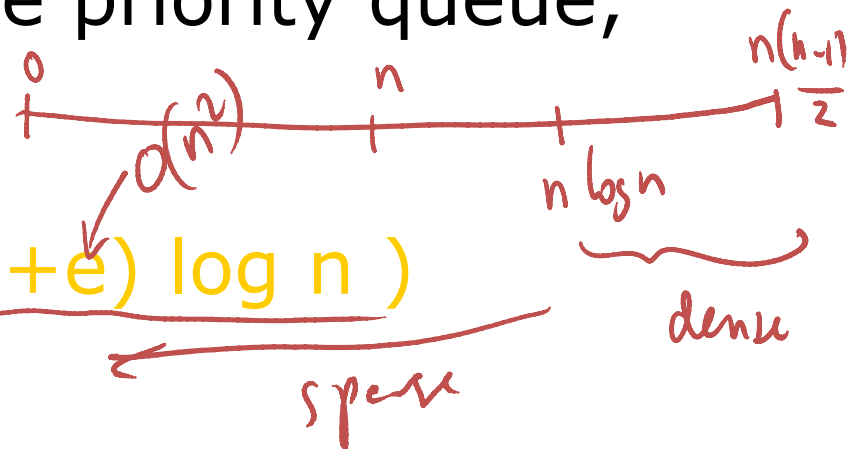
n vertices

Visitation loop: n calls

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

$n = |V|$
 $e = |E|$

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?

Compression

Alphabet = $\{A, B, C, D, E\}$

Fixed length encoding

A A B B C D E
↓ ↓ ↓ ↓
000 000 001 010

100 Chars

File size = 100 bytes

File size = $\frac{300}{8}$ bytes

A = 000

B = 001

C = 010

D = 011

E = 100

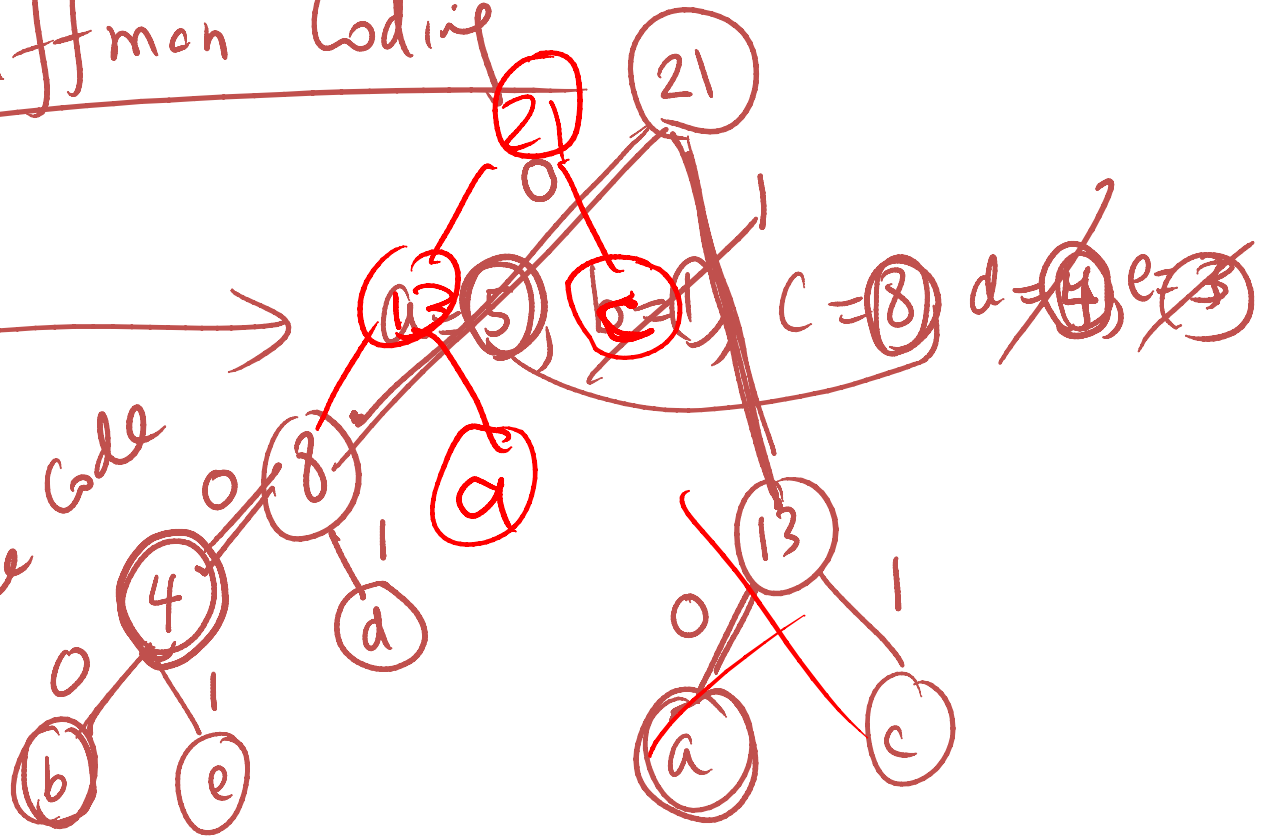
$abecce = \underbrace{10}_{a} \underbrace{00}_{b} \underbrace{00}_{e} \underbrace{00}_{e} \underbrace{01}_{e} 11111 \rightarrow ab$

Huffman Coding

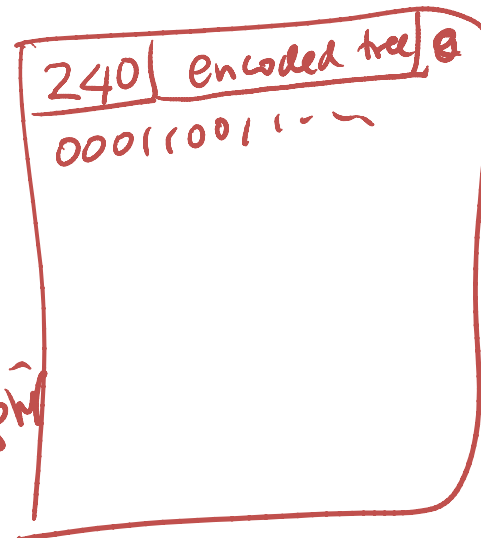
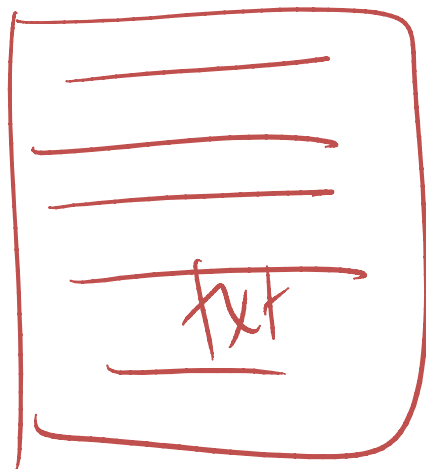
ABACAD
 ✓

$b = 000$
 $e = 001$
 $a = 10$
 $c = 11$

Prefix-free Code

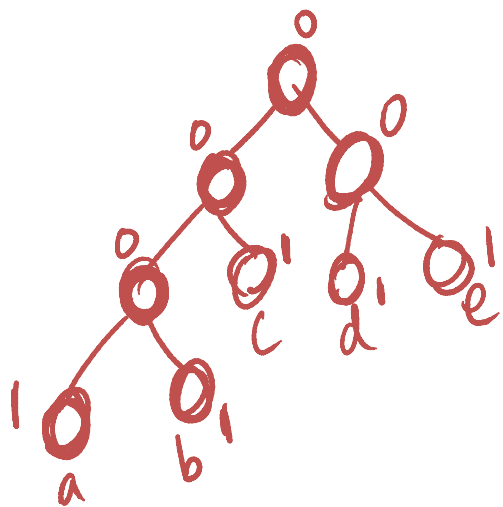


no code can be
 a prefix of another Full tree



Root left Right

pre order
in order



Pre order
traversal

zip

