Applied Algorithms

CSCI-B505 / INFO-I500

Lecture 18.

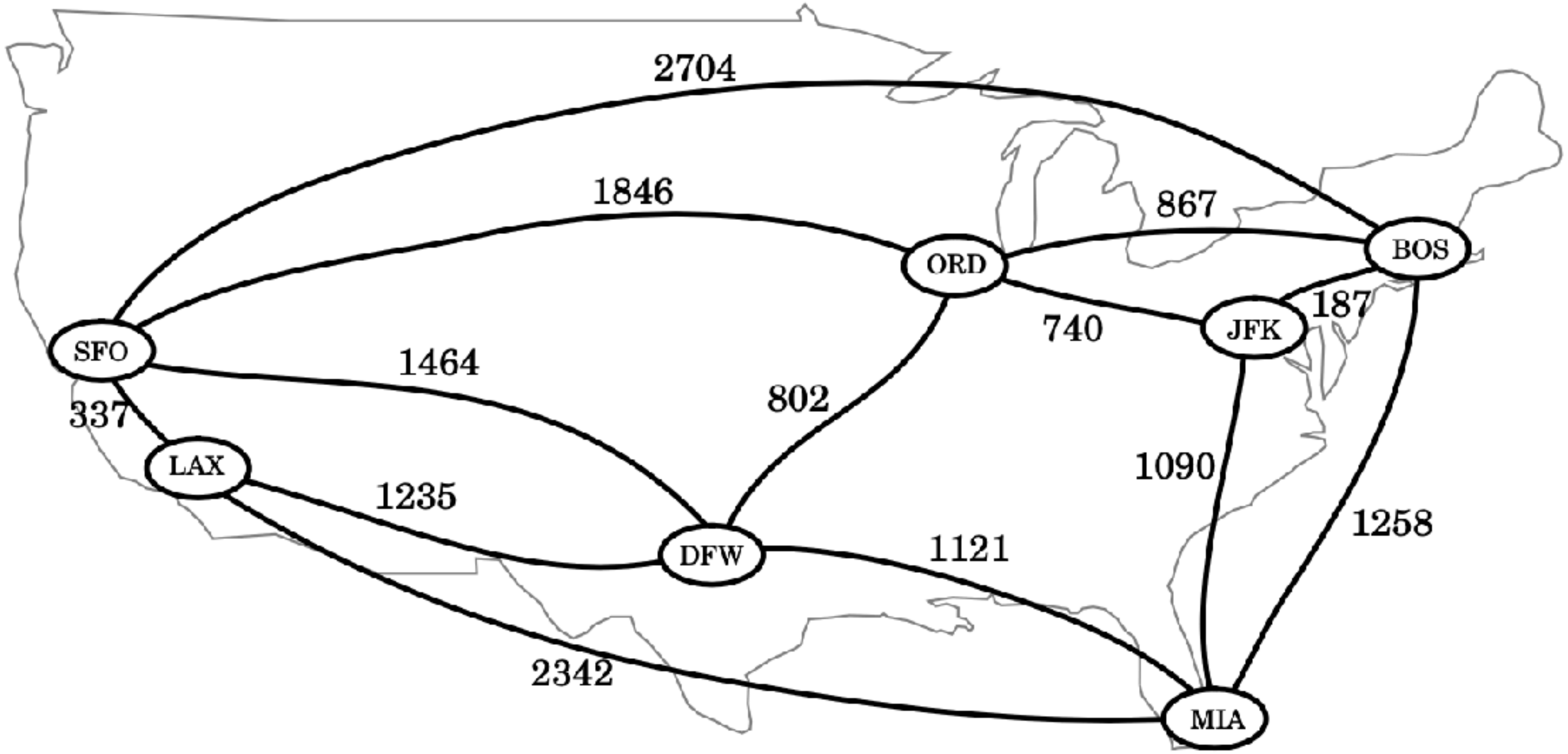
Shortest Path and Minimum Spanning Tree

M. Oguzhan Kulekci

- Shortest Path - Dijkstra’s algorithm
- Minimum spanning tree - Prim’s and Kruskal’s algorithms
- Graph coloring

Shortest Path in a Weighted Graph

- What is the shortest path from an initial vertex s to a target vertex t ?



Shortest Path in a Weighted Graph

- Assume a path from vertex v_1 to v_k is $P = \{(v_1, v_2), (v_2, v_3), \ldots\ldots, (v_{k-1}, v_k)\}$

What should be the P that minimizes $w(P) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$?

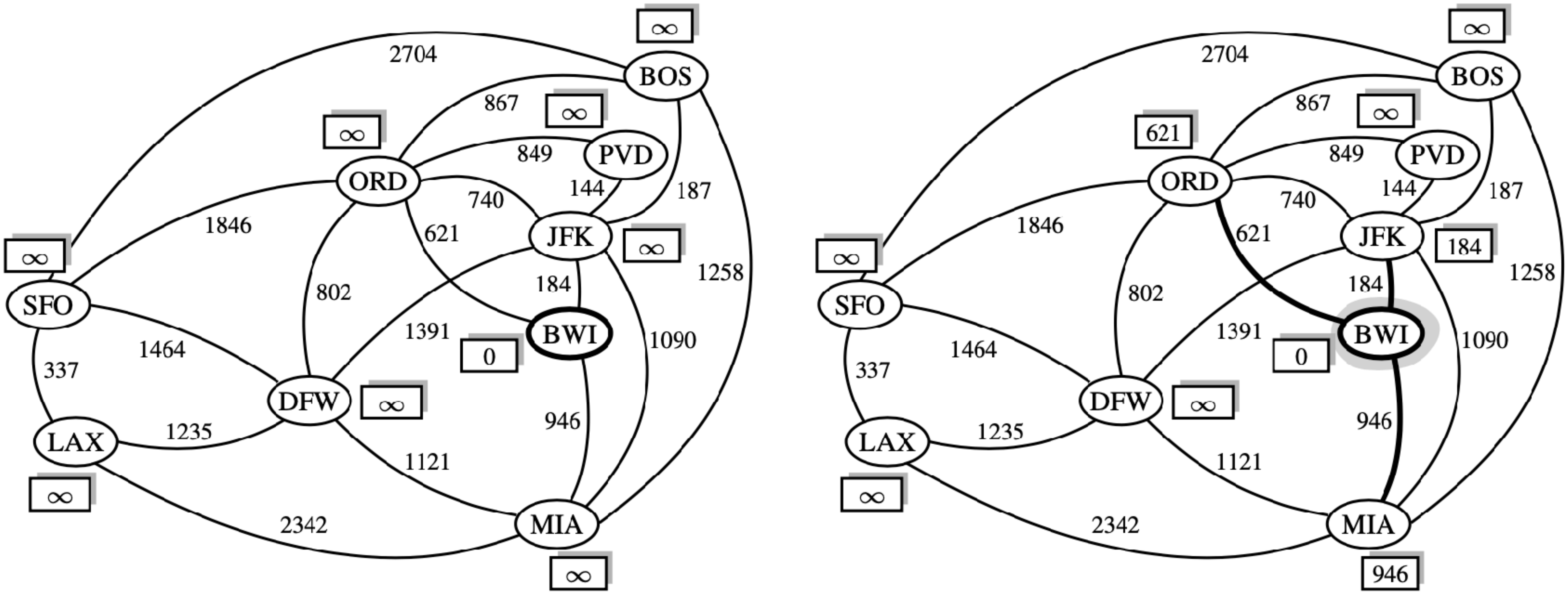
- $w(v_i, v_{i+1})$ is the **non-negative** weight from vertex v_i to v_{i+1}.
- Dijkstra's shortest path computes **single-source** shortest paths, from an initial vertex to all other vertices
- It is a **greedy algorithm** that provides the **optimal** solution.

Shortest Path in a Weighted Graph

- $D[u]$ denotes the minimum distance so far detected from vertex s to the vertex u
- Initially all $D[u]$ are set to infinity, but $D[s] = 0$.
- At each step of iteration, the smallest $D[u]$ is selected and all the $D[v]$ such that there is an edge from u to v are updated according to **edge relaxation** rule

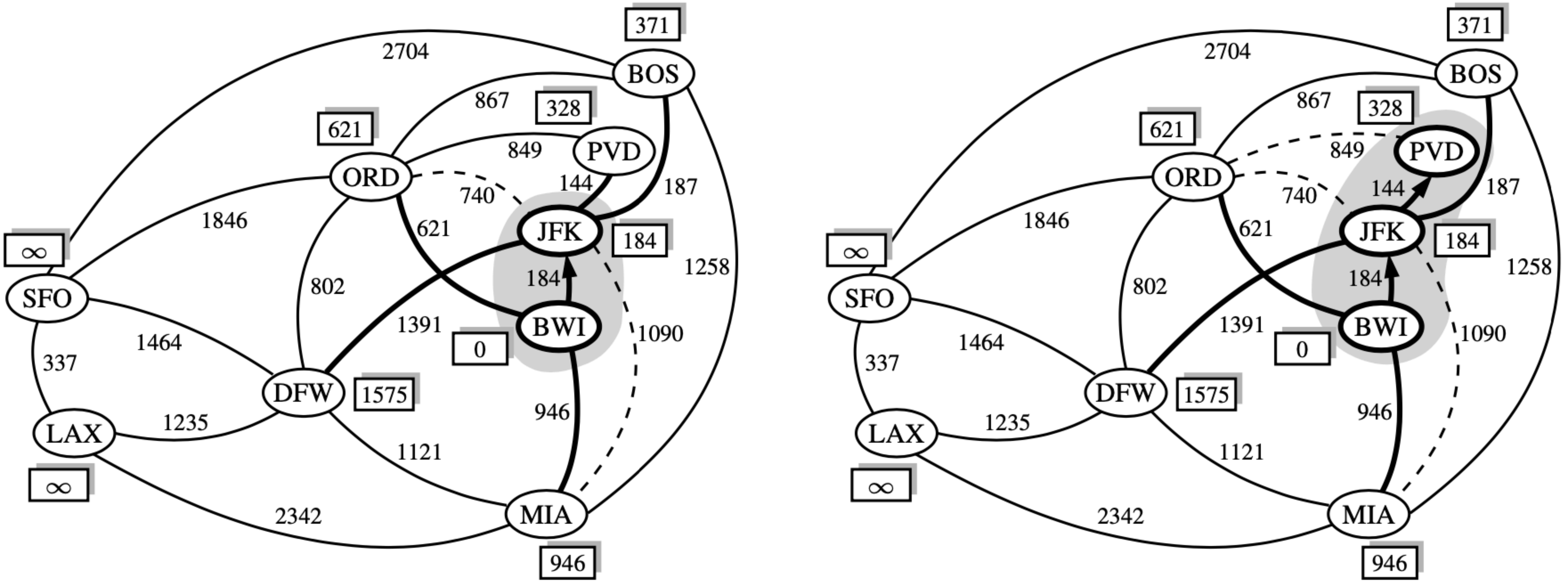
  If $D[u] + w(u, v) < D[v]$,then $D[v] = D[u] + w(u, v)$
- After the updates, again the vertex with the smallest $D[]$ value is selected and the procedure is repeated until all vertices appear in the selected vertices.

Shortest Path in a Weighted Graph



(a) (b)

Shortest Path in a Weighted Graph

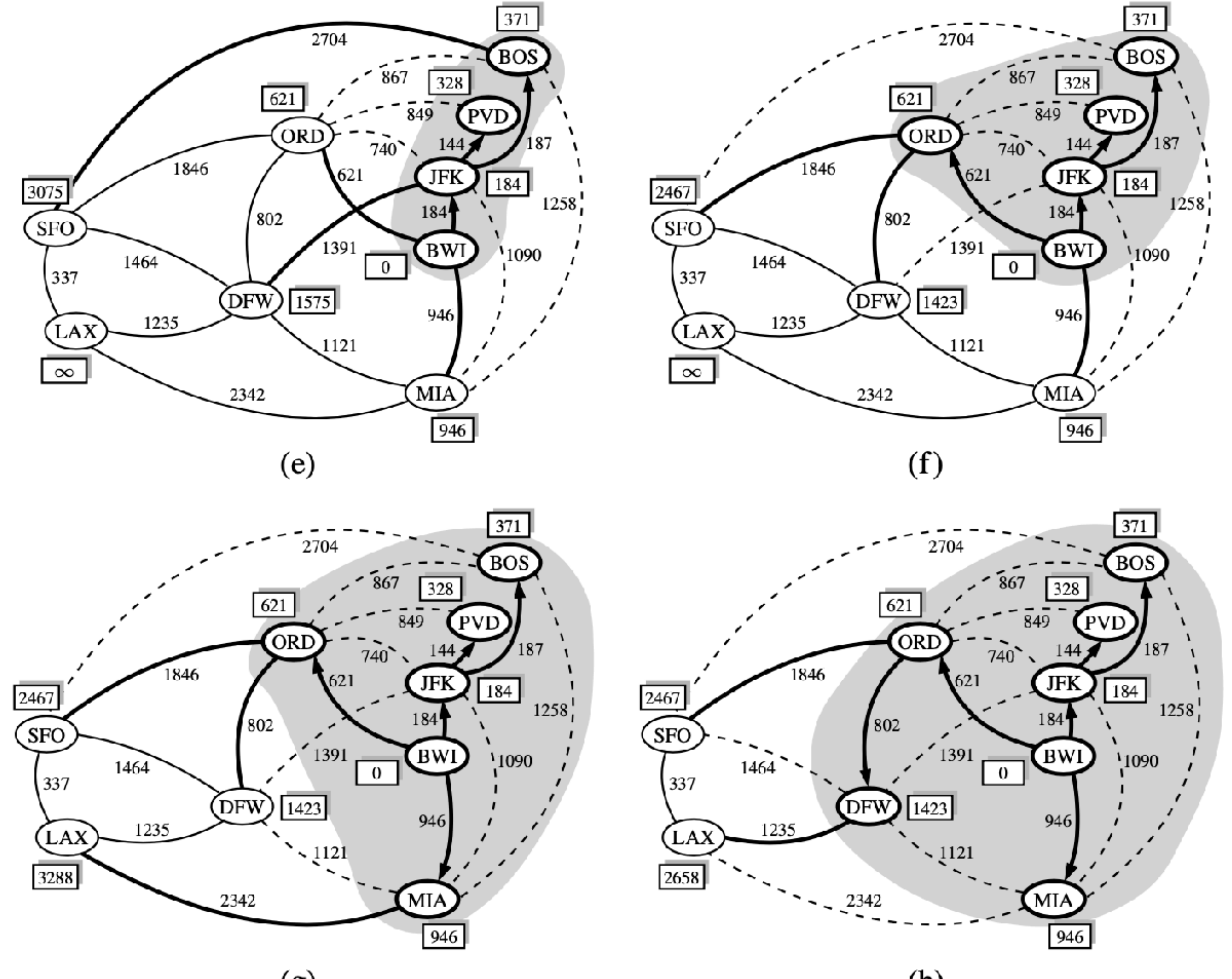


(c)

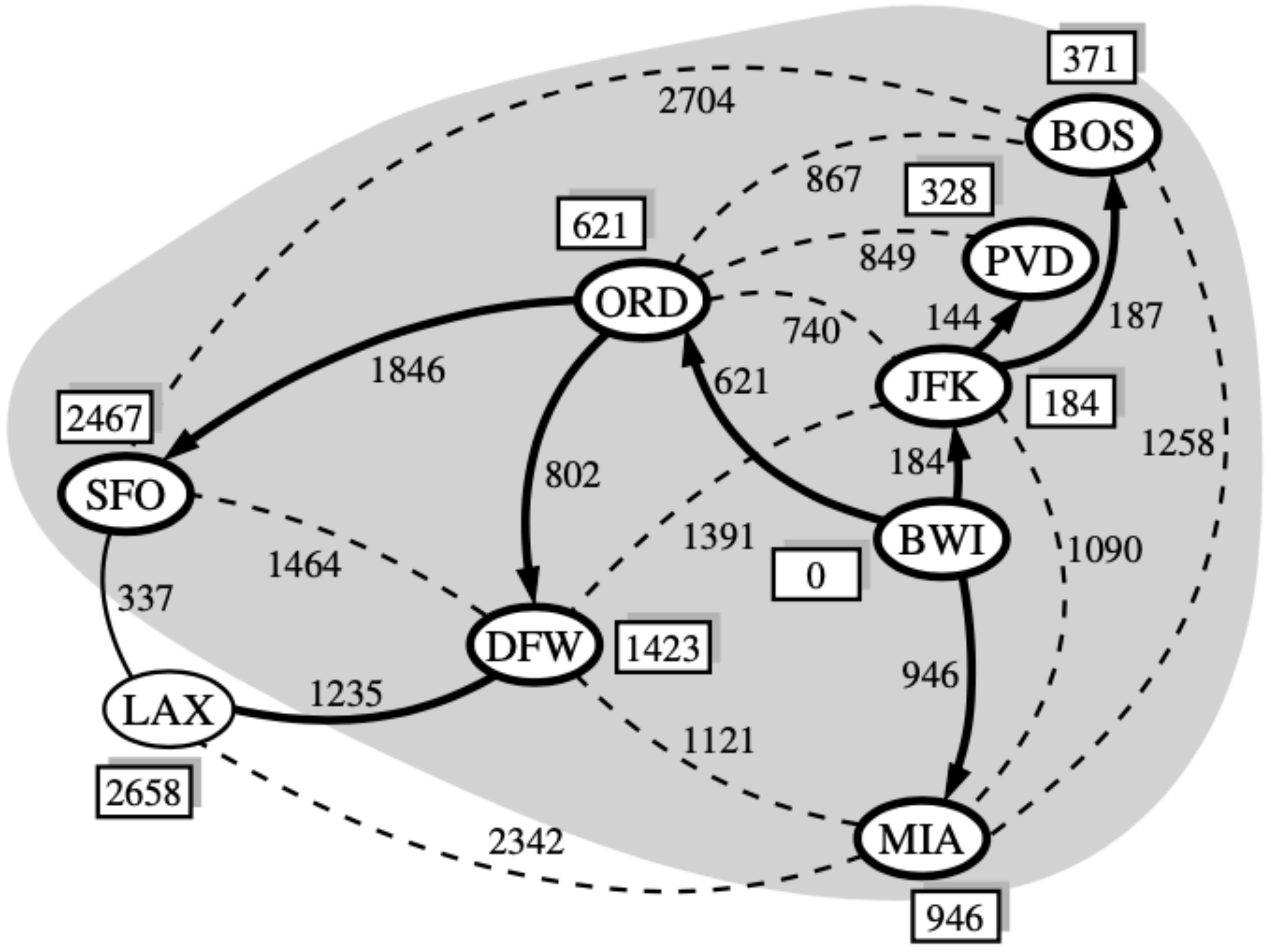
(d)

Would it be possible to have a shorter path from BWI to JFK ? Why?

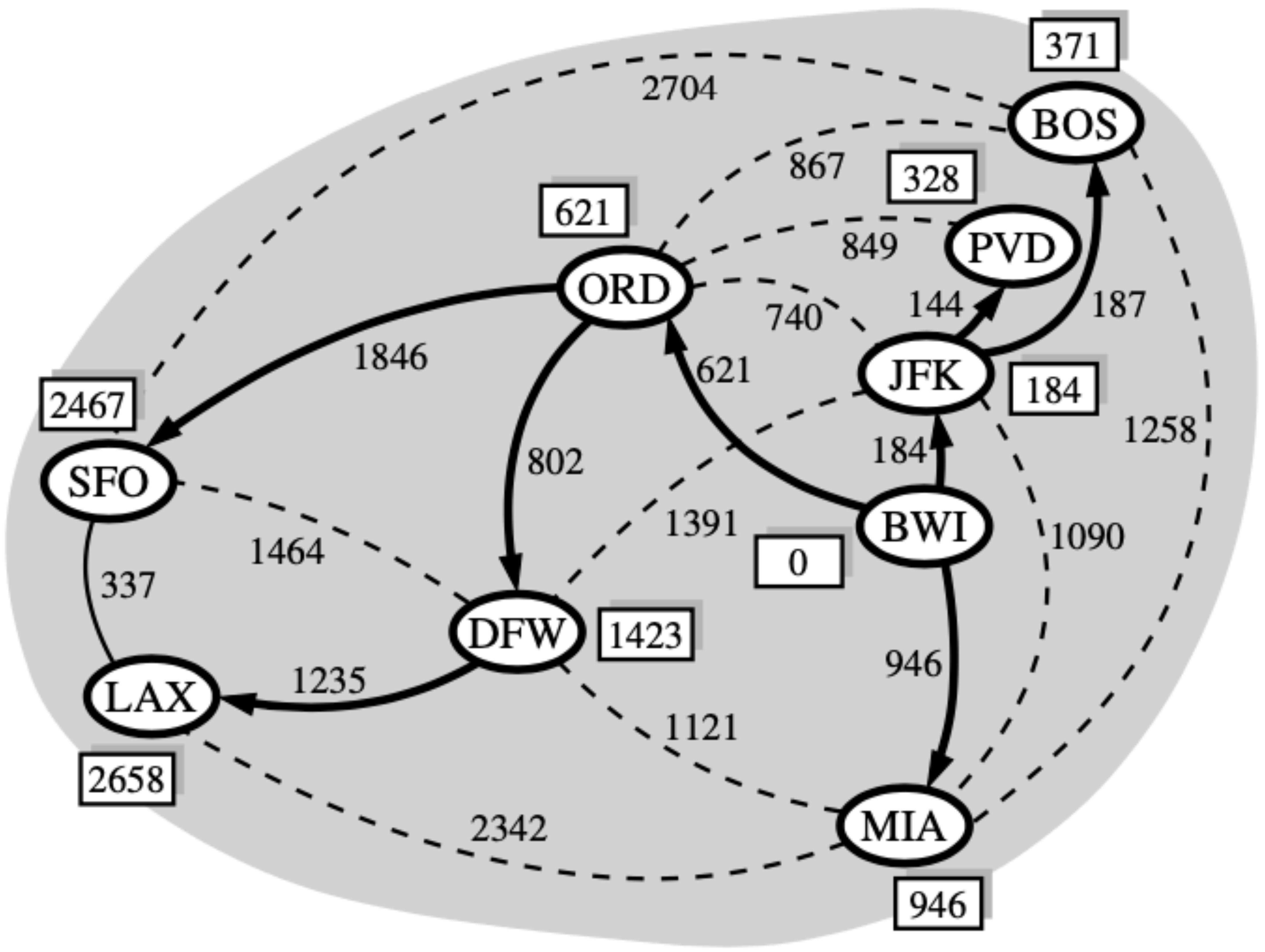
Shortest Path in a Weighted Graph



Shortest Path in a Weighted Graph



(i)



(j)

Shortest Path in a Weighted Graph

$O((n+m)\log n)$

Algorithm ShortestPath(G, s):

Input: A weighted graph G with nonnegative edge weights, and a distinguished vertex s of G.

Output: The length of a shortest path from s to v for each vertex v of G.

```
Initialize D[s] = 0 and D[v] = ∞ for each vertex v ≠ s.
Let a priority queue Q contain all the vertices of G using the D labels as keys.
while Q is not empty do
    {pull a new vertex u into the cloud}
    u = value returned by Q.remove_min()
    for each vertex v adjacent to u such that v is in Q do
        {perform the relaxation procedure on edge (u, v)}
        if D[u] + w(u, v) < D[v] then
            D[v] = D[u] + w(u, v)
            Change to D[v] the key of vertex v in Q.
return the label D[v] of each vertex v
```

All n vertices visited

$O(\log n)$-time to remove min from the heap

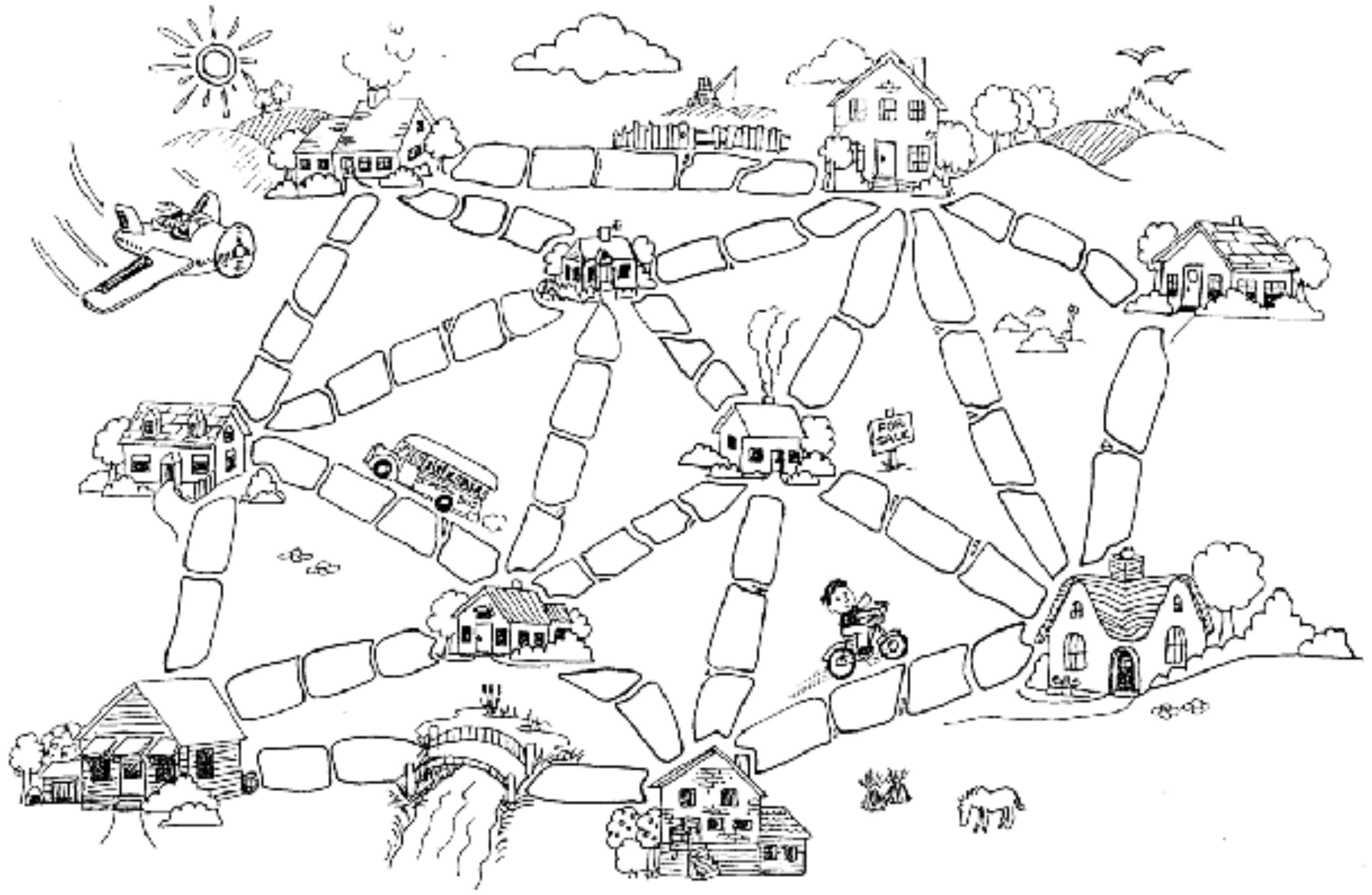
Traverses over all m edges in total

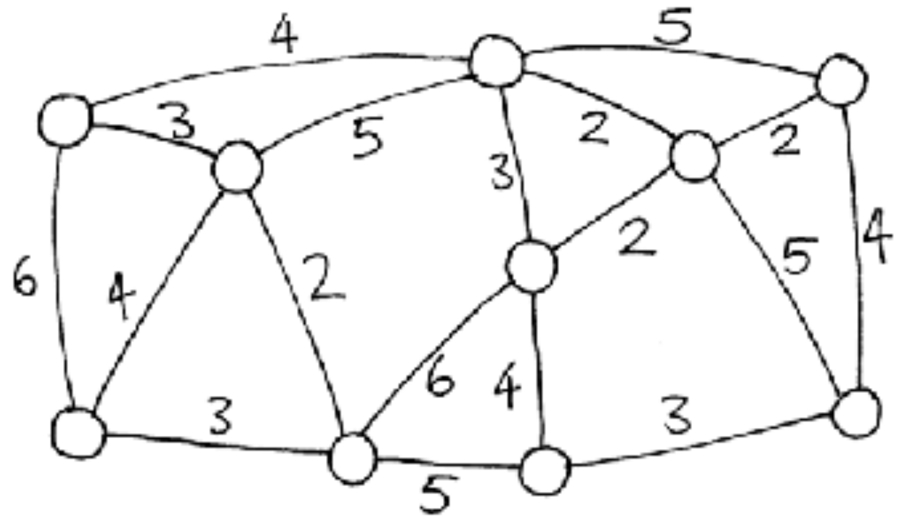
$O(\log n)$-time to update the node in the heap

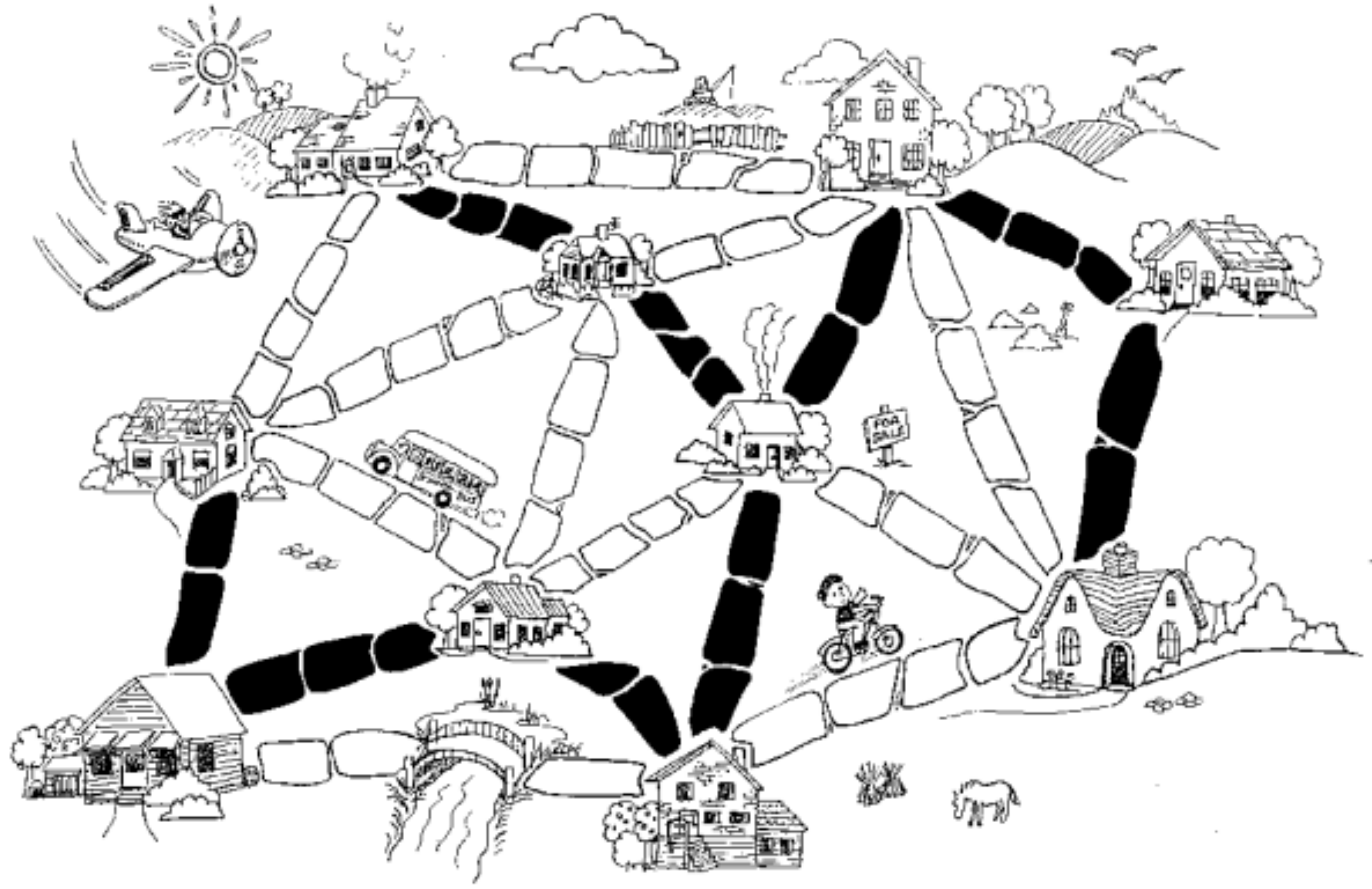
Minimum Spanning Trees

Given an undirected, **weighted graph G**, we are interested in finding a tree T that that contains all the vertices in G and minimizes the sum

$$w(T) = \sum_{\forall u,v \in T} w(u,v)$$





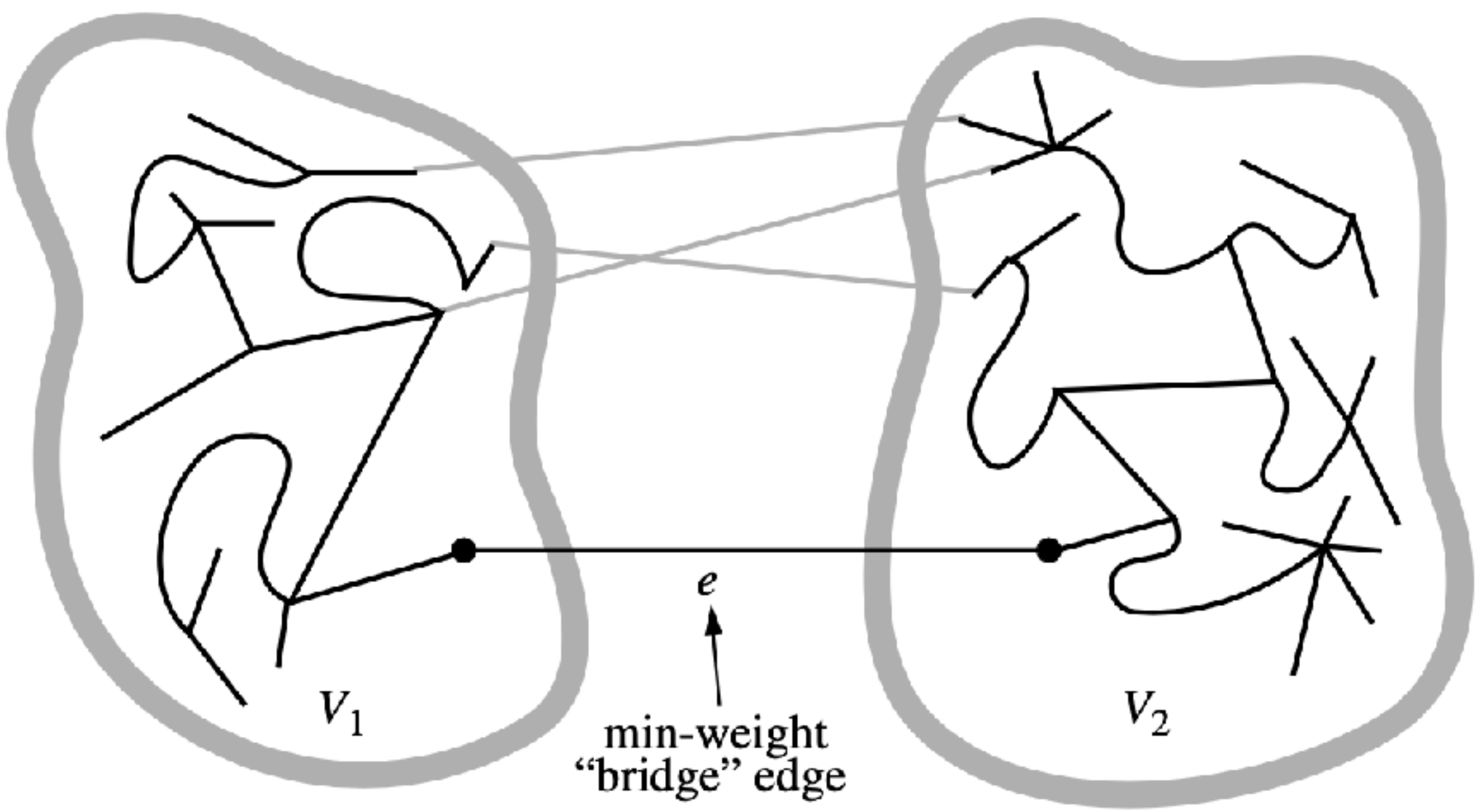


Remember that the BFS or DFS traversals also produce the spanning trees, but not minimizing the cost on a weighted graph.

Minimum Spanning Trees

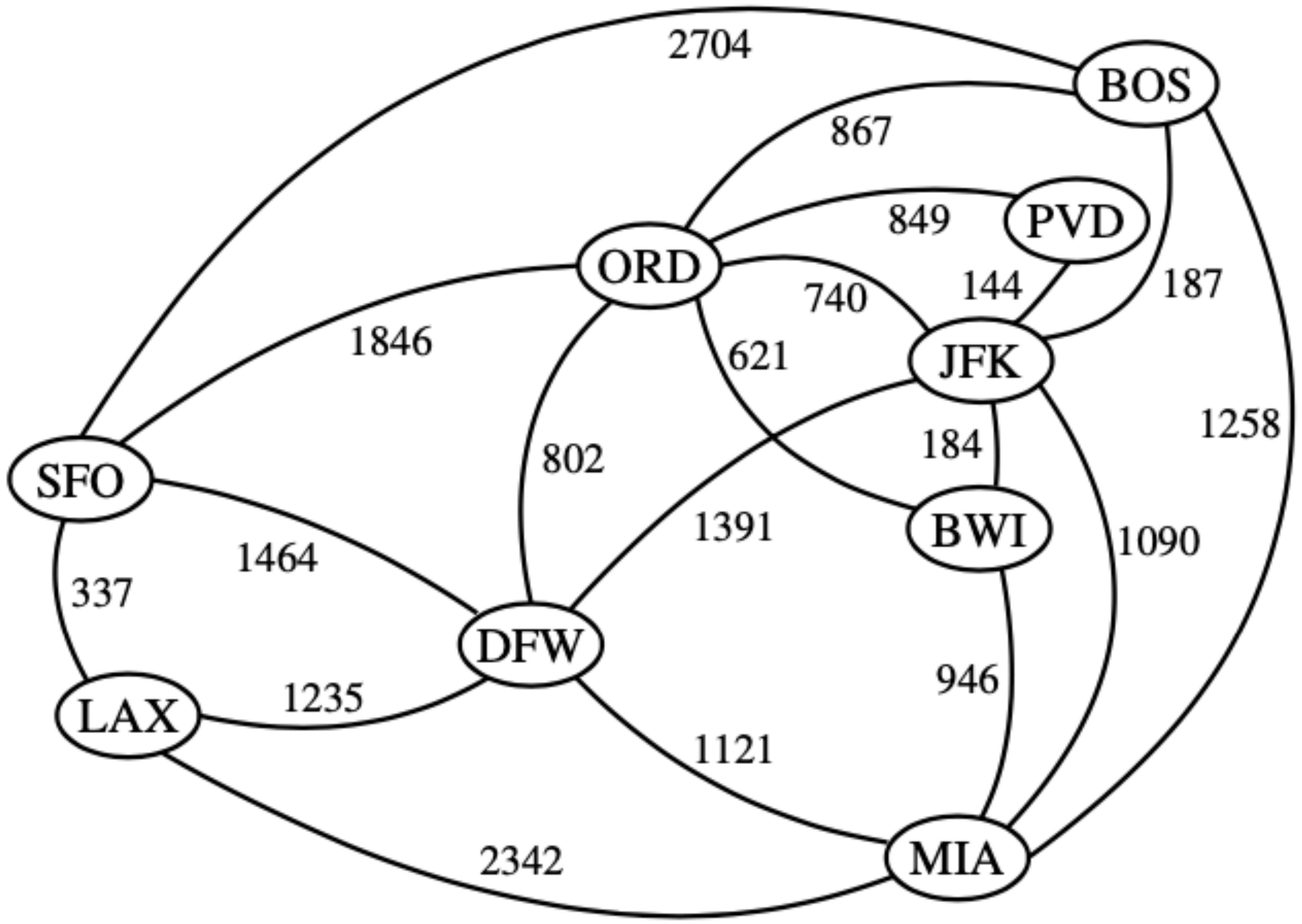
- Assume the vertices are split into two disjoint sets as shown.
- The observation is that the edge with the minimum weight between the vertices of first and second partitions should be in the minimum spanning tree.

e Belongs to a Minimum Spanning Tree

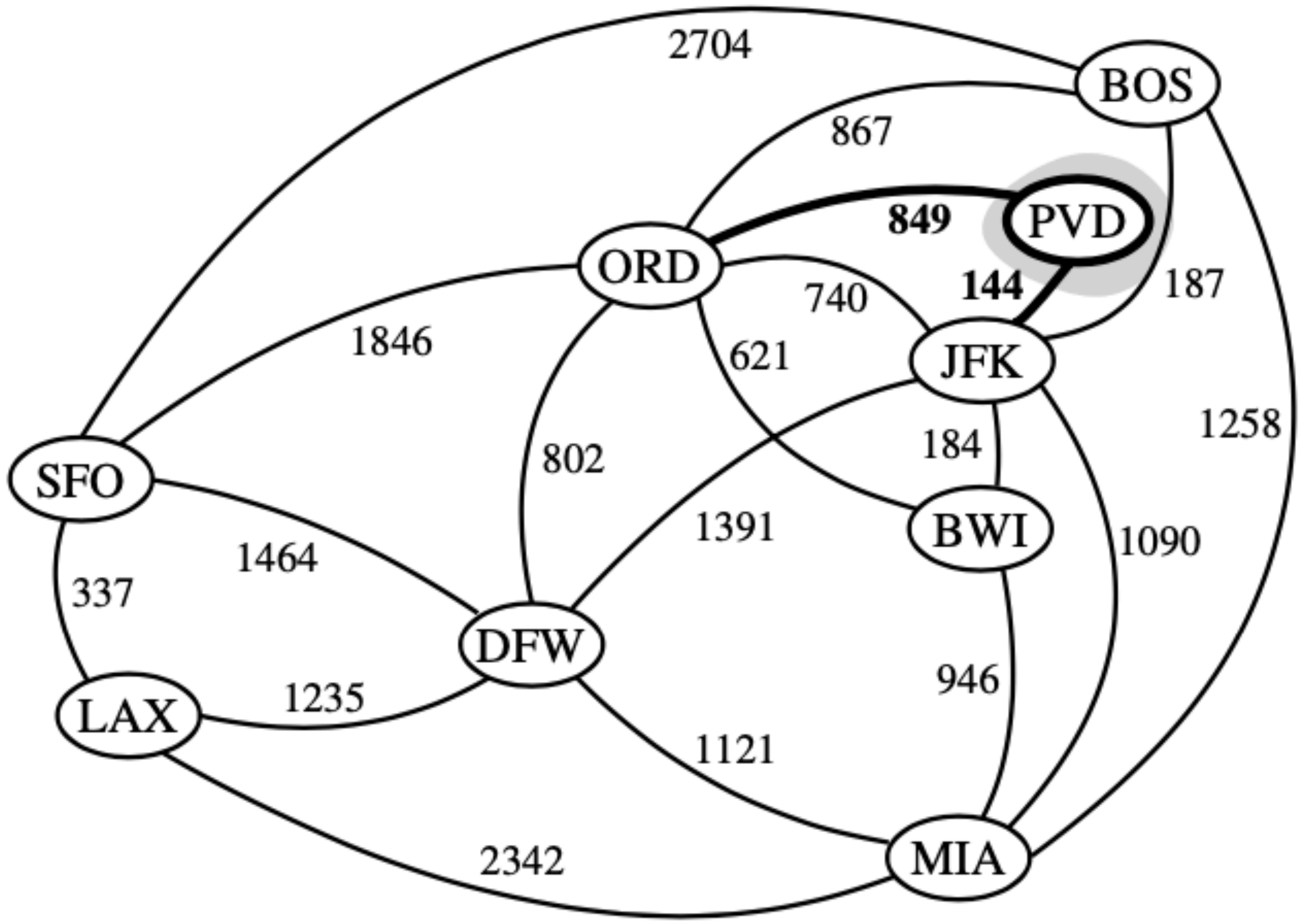


- Assume e is not in the minimum spanning tree.
- Then when it is added to the MST, the MST will contain a cycle.
- This is because the end points of the e are already on MST, and thus, adding e creates a cycle.
- Since e has the minimal weight, all other edges on the cycle are greater than it.
- So, if we remove one of the other edges in this MST, and add e we will have a new MST with a smaller weight !

Minimum Spanning Trees - Prim’s Algorithm

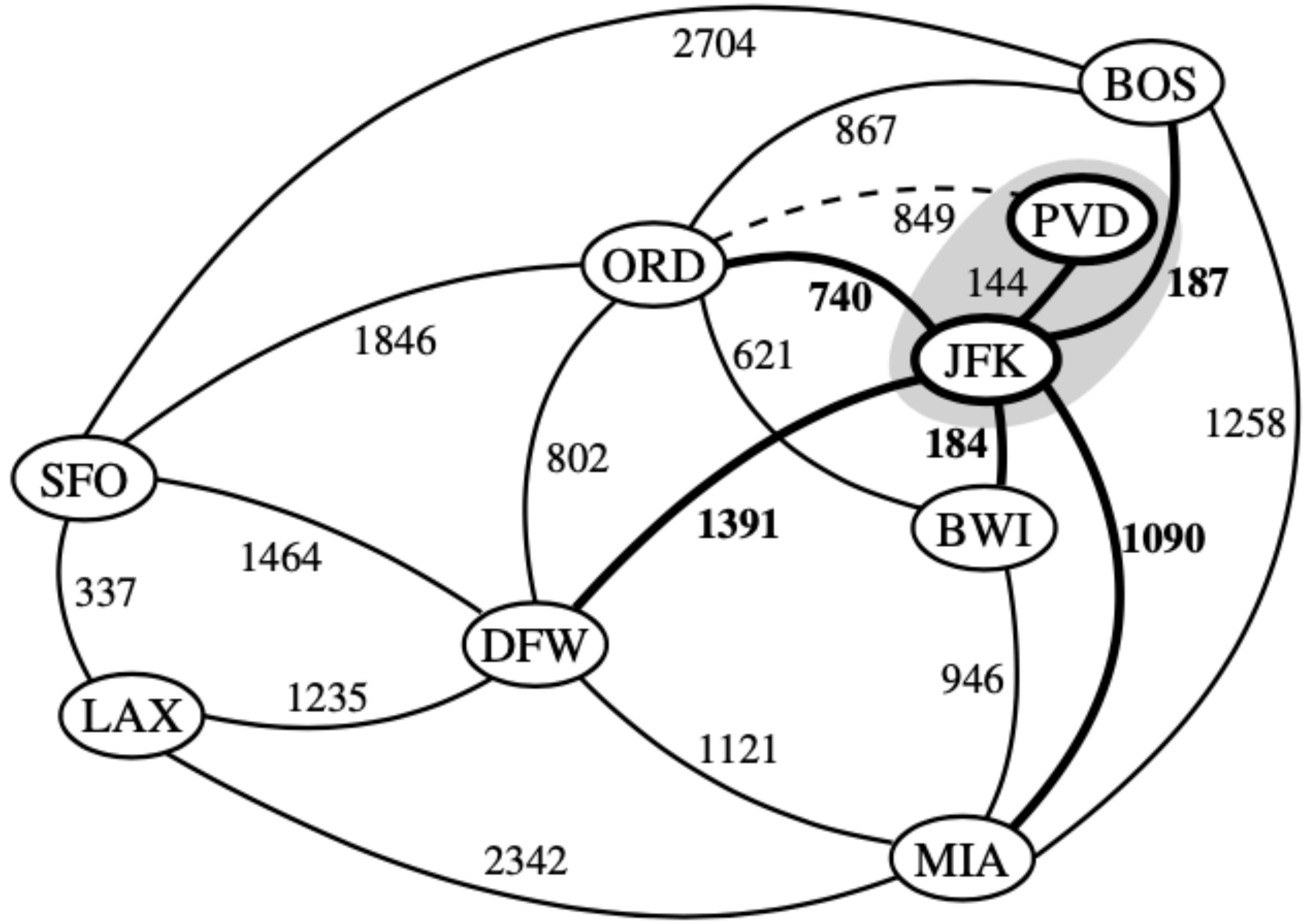


(a)

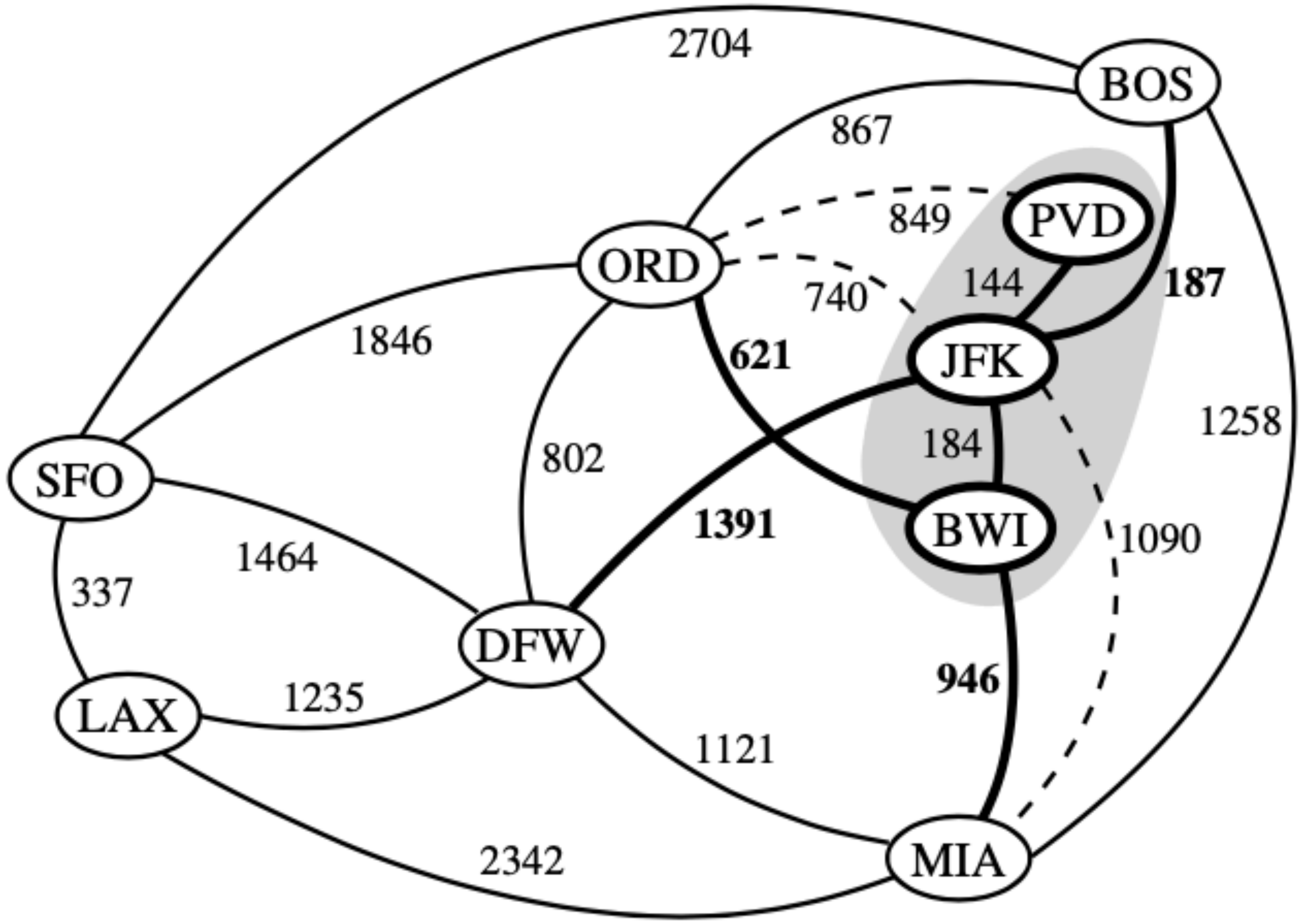


(b)

Minimum Spanning Trees - Prim’s Algorithm

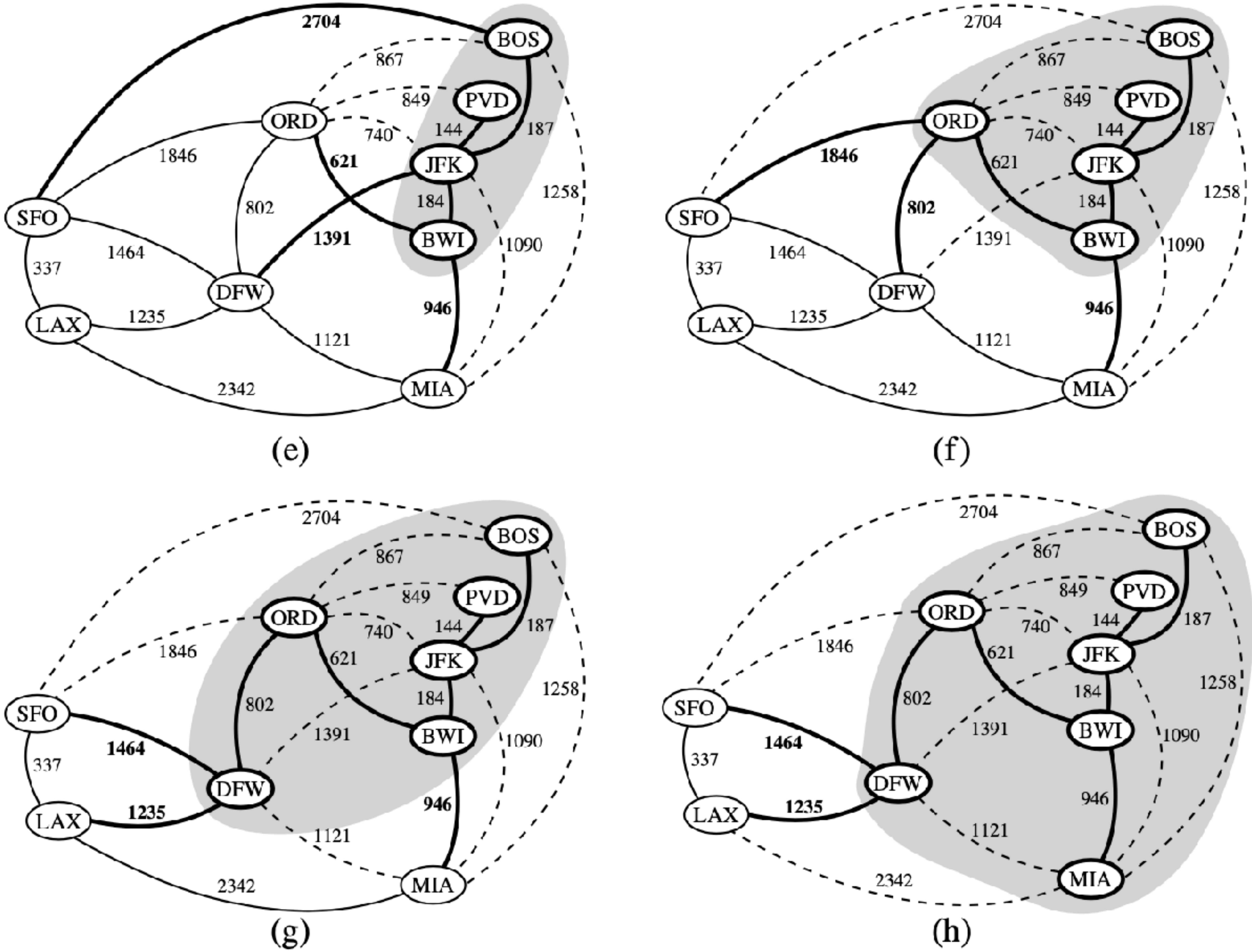


(c)



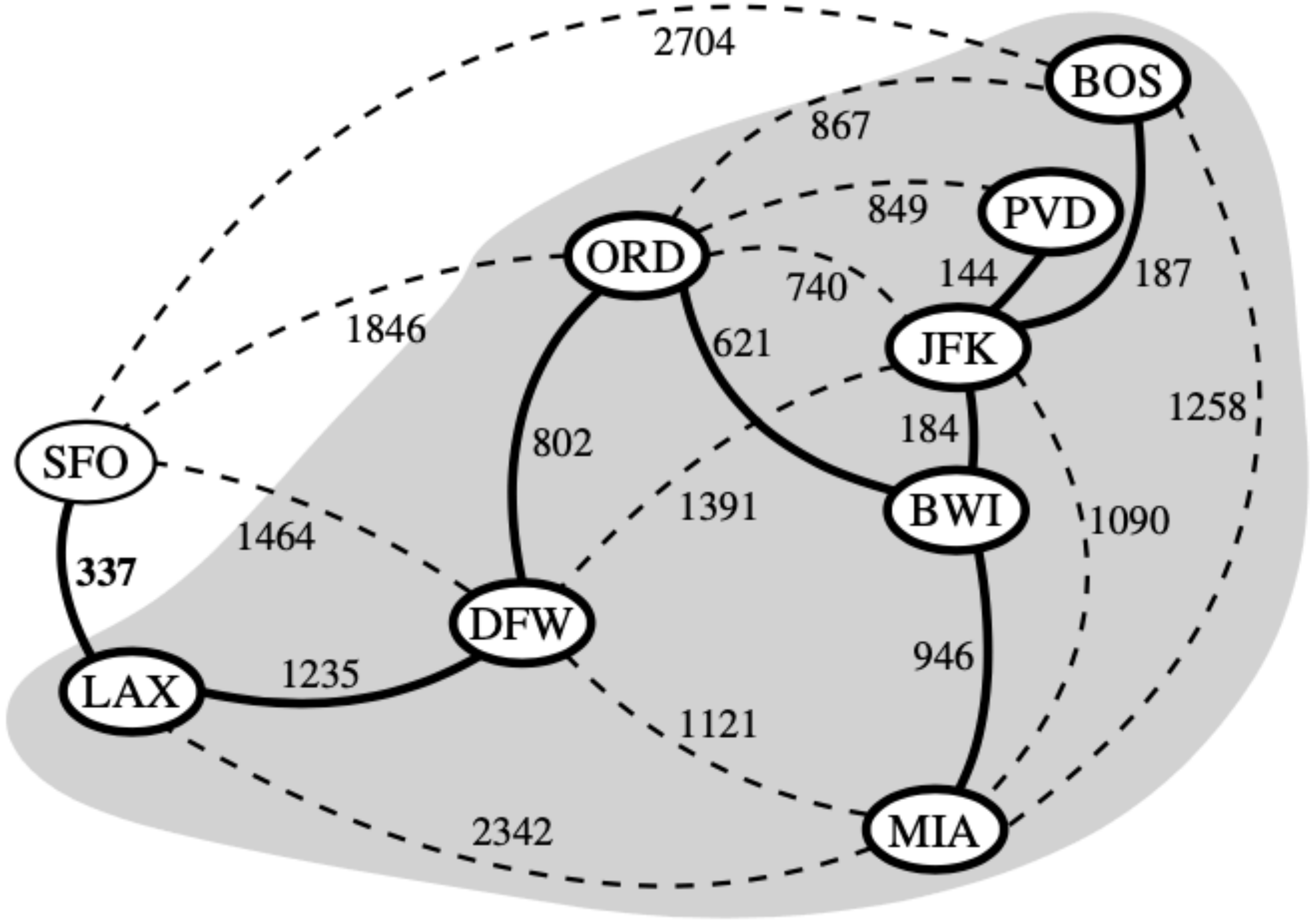
(d)

Minimum Spanning Trees - Prim’s Algorithm

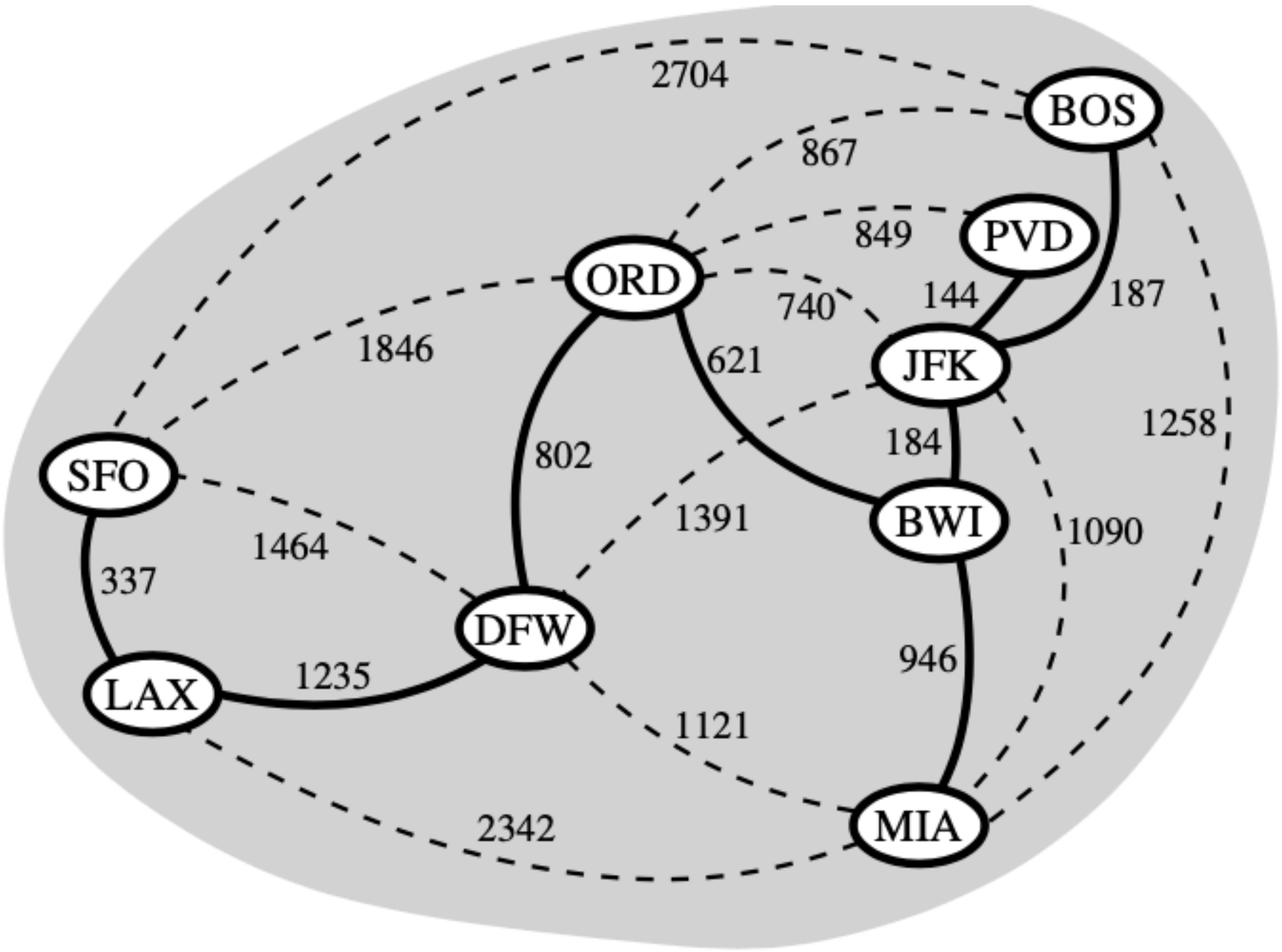


(e) (f) (g) (h)

Minimum Spanning Trees - Prim's Algorithm



(i)



(j)

Minimum Spanning Trees - Prim's Algorithm

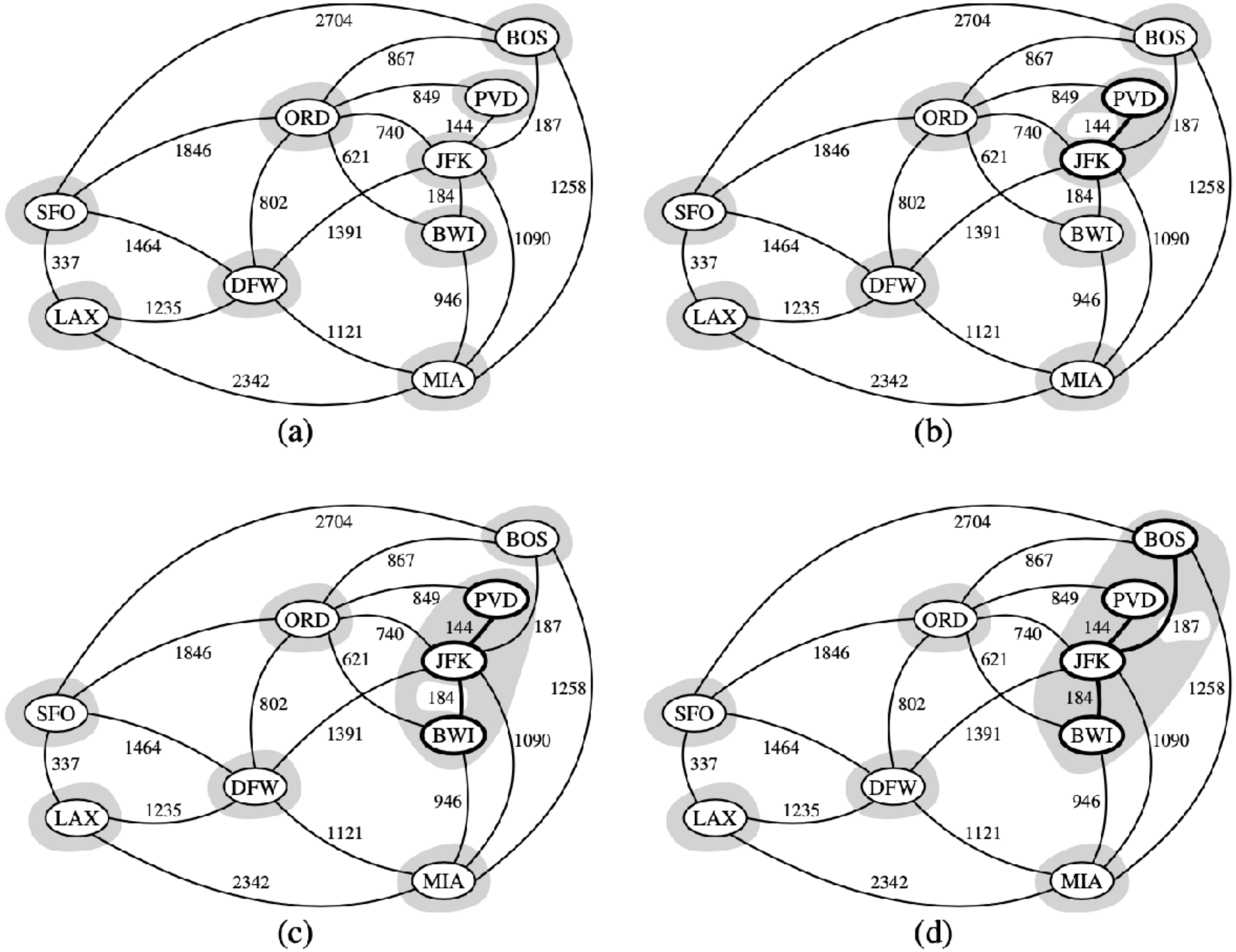
Algorithm PrimJarnik(G):
  Input: An undirected, weighted, connected graph G with n vertices and m edges
  Output: A minimum spanning tree T for G

```
Pick any vertex s of G
D[s] = 0
for each vertex v ≠ s do
   D[v] = ∞
Initialize T = ∅.
Initialize a priority queue Q with an entry (D[v], (v, None)) for each vertex v,
where D[v] is the key in the priority queue, and (v, None) is the associated value.
while Q is not empty do
   (u, e) = value returned by Q.remove_min()
   Connect vertex u to T using edge e.
   for each edge e' = (u, v) such that v is in Q do
      {check if edge (u, v) better connects v to T}
      if w(u, v) < D[v] then
         D[v] = w(u, v)
         Change the key of vertex v in Q to D[v].
         Change the value of vertex v in Q to (v, e').
return the tree T
```

- n insertion into heap
- n remove-min from heap
- At most m updates in the heap

With a heap data structure and an adjacency list to represent the graph $O((n+m)\log n)$ time

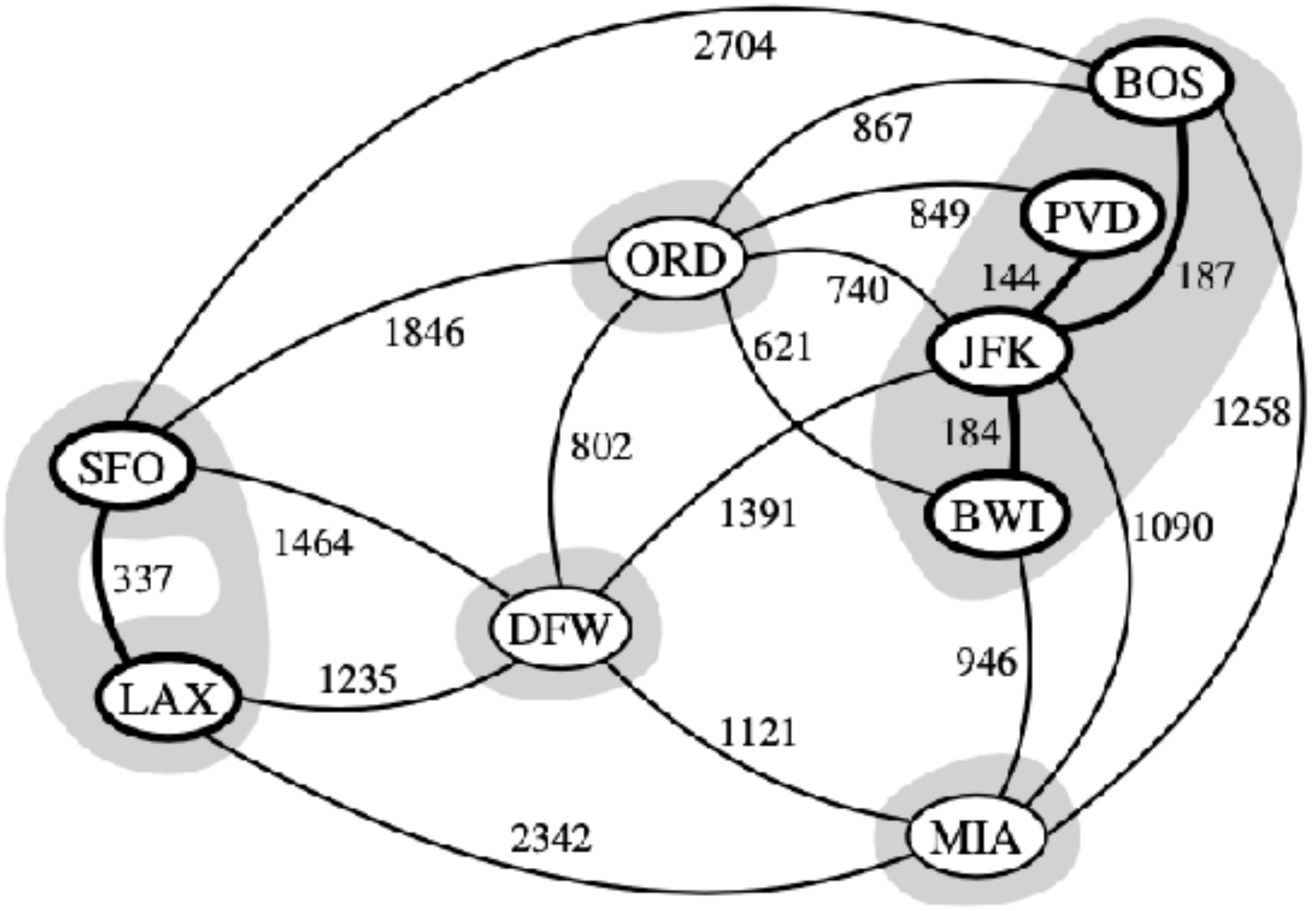
Minimum Spanning Trees - Kruskal's Algorithm



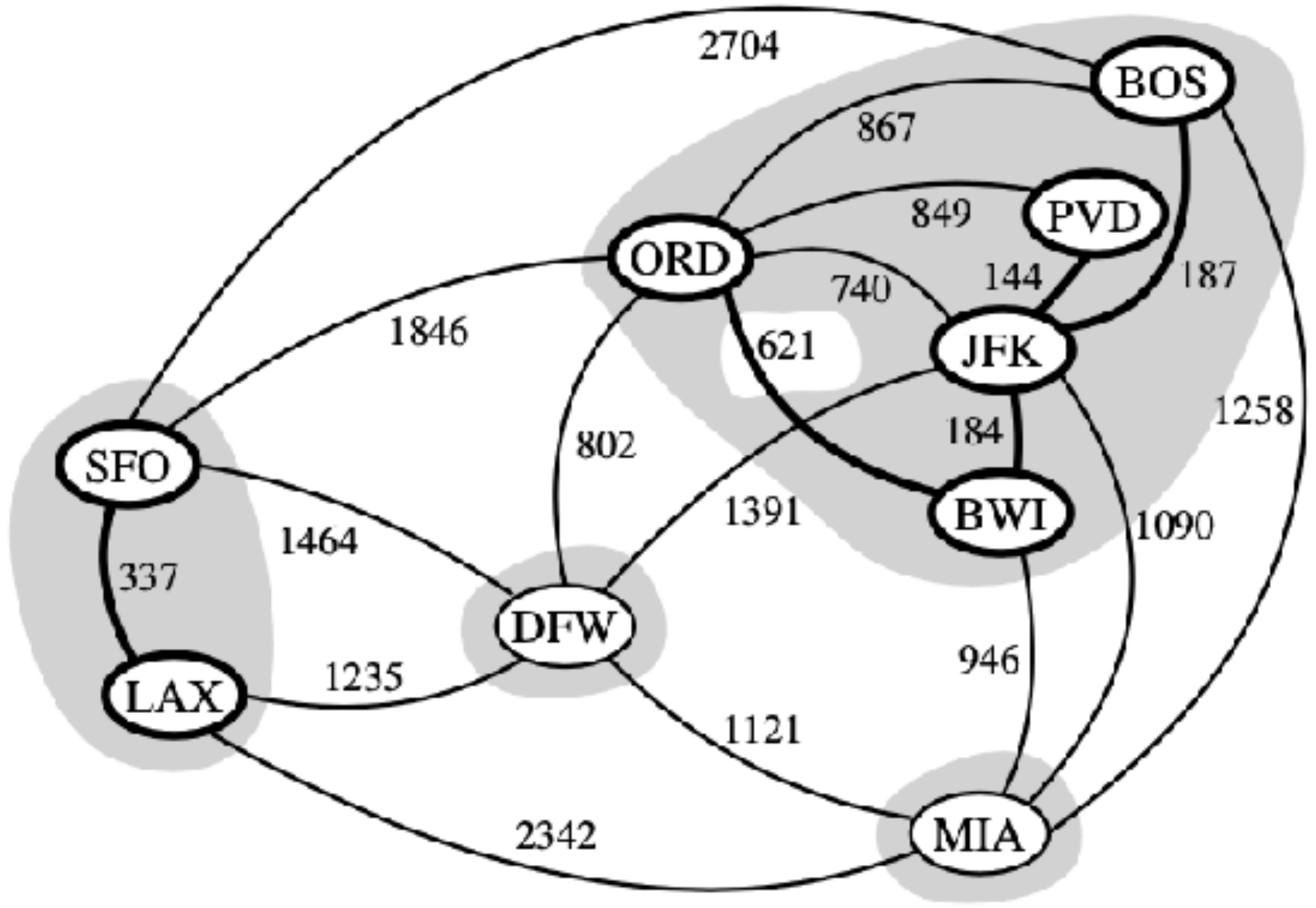
(a) (b) (c) (d)

- Prim builds MST as a single tree
- Kruskal starts with n trees, merge them until you are left with a single tree
- Both use the same observation we discussed before

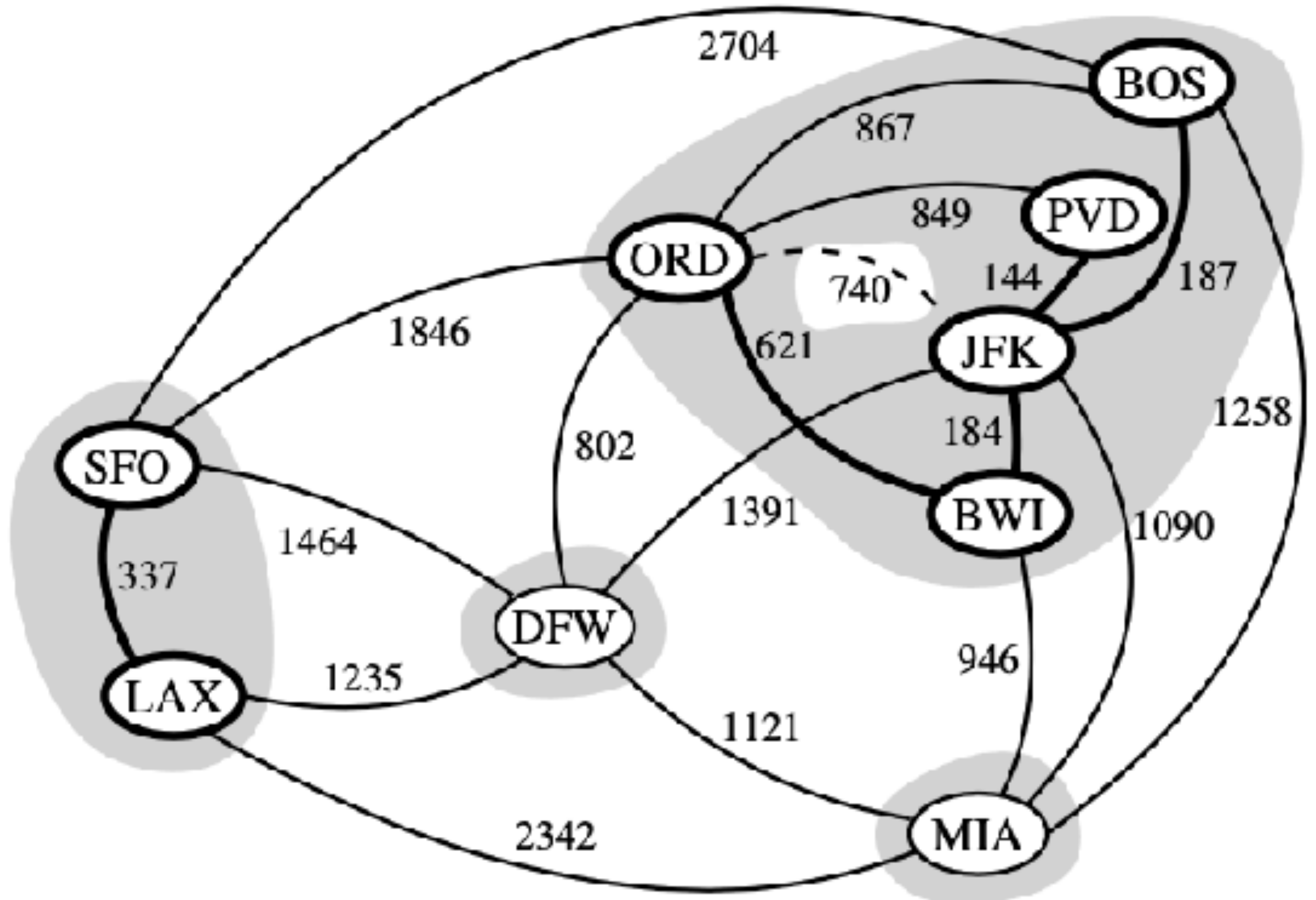
Minimum Spanning Trees - Kruskal’s Algorithm



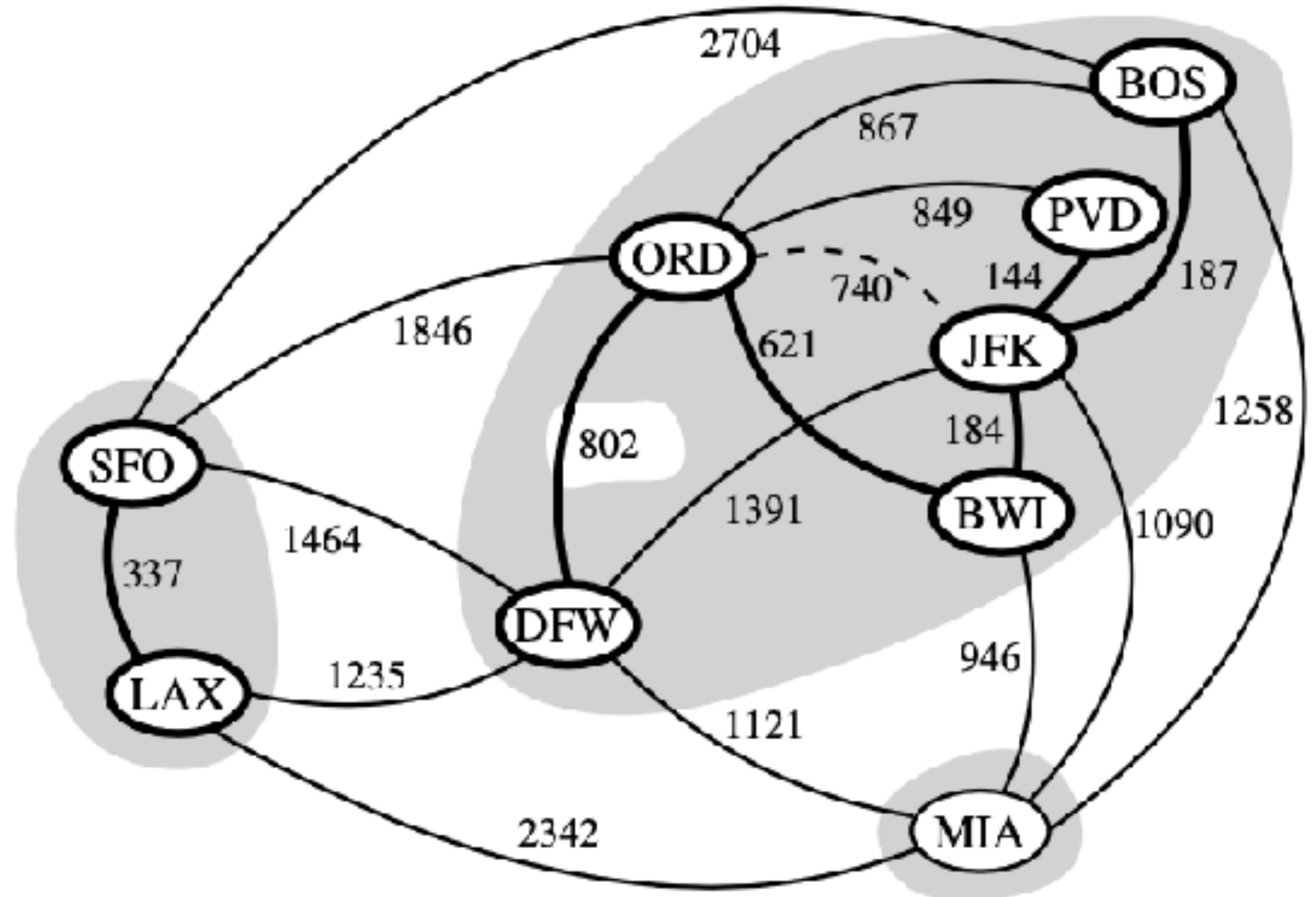
(e)



(f)

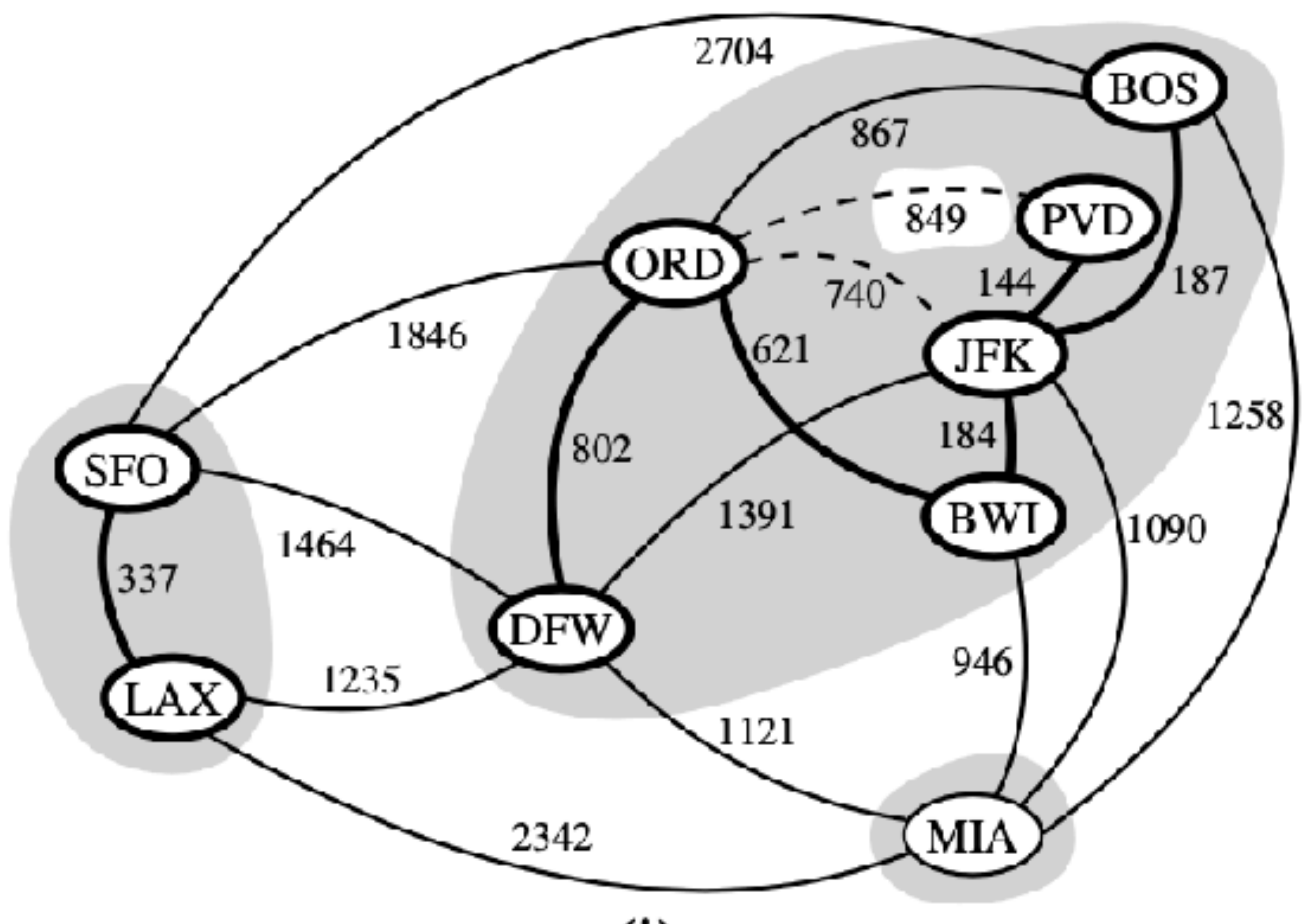


(g)

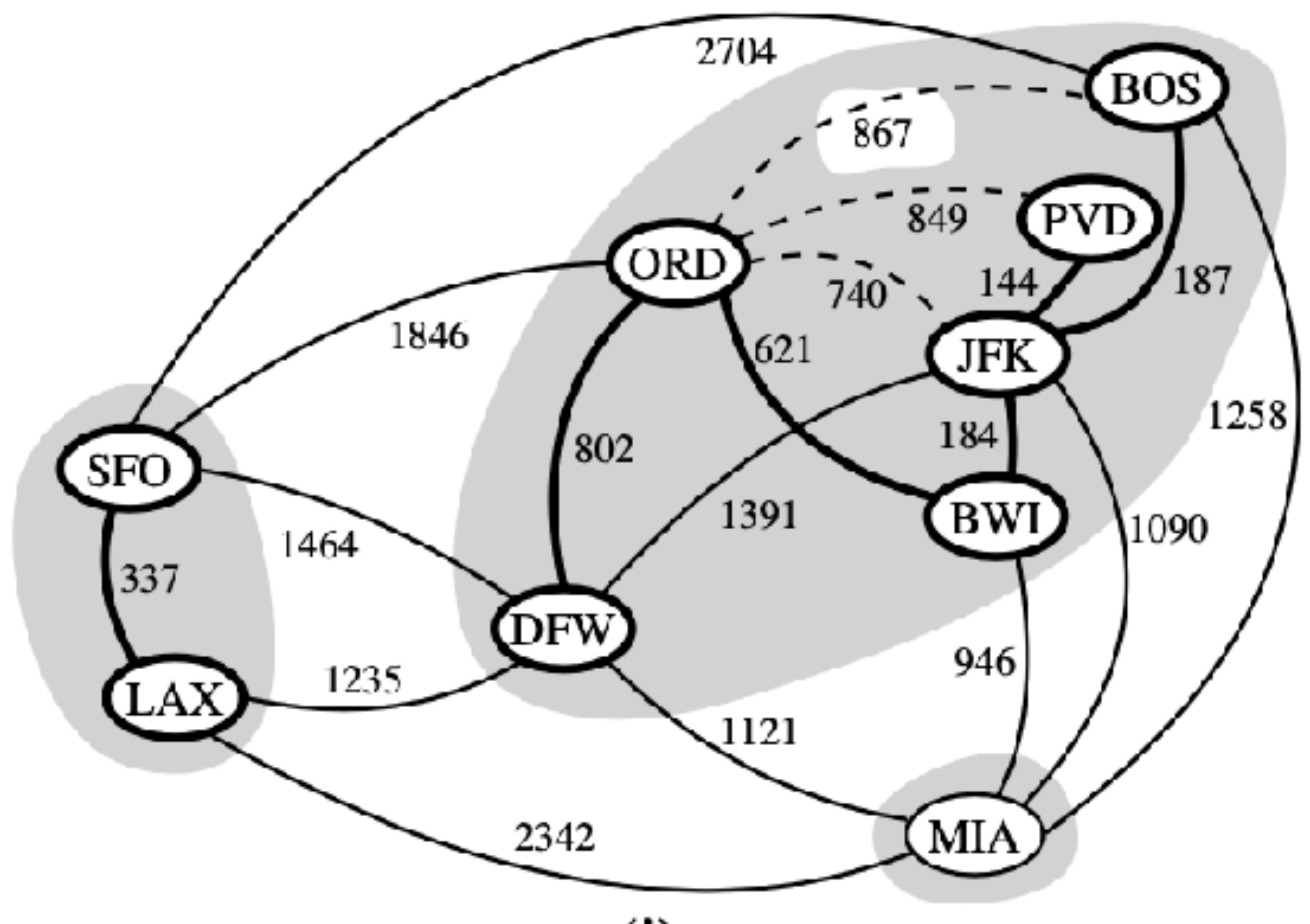


(h)

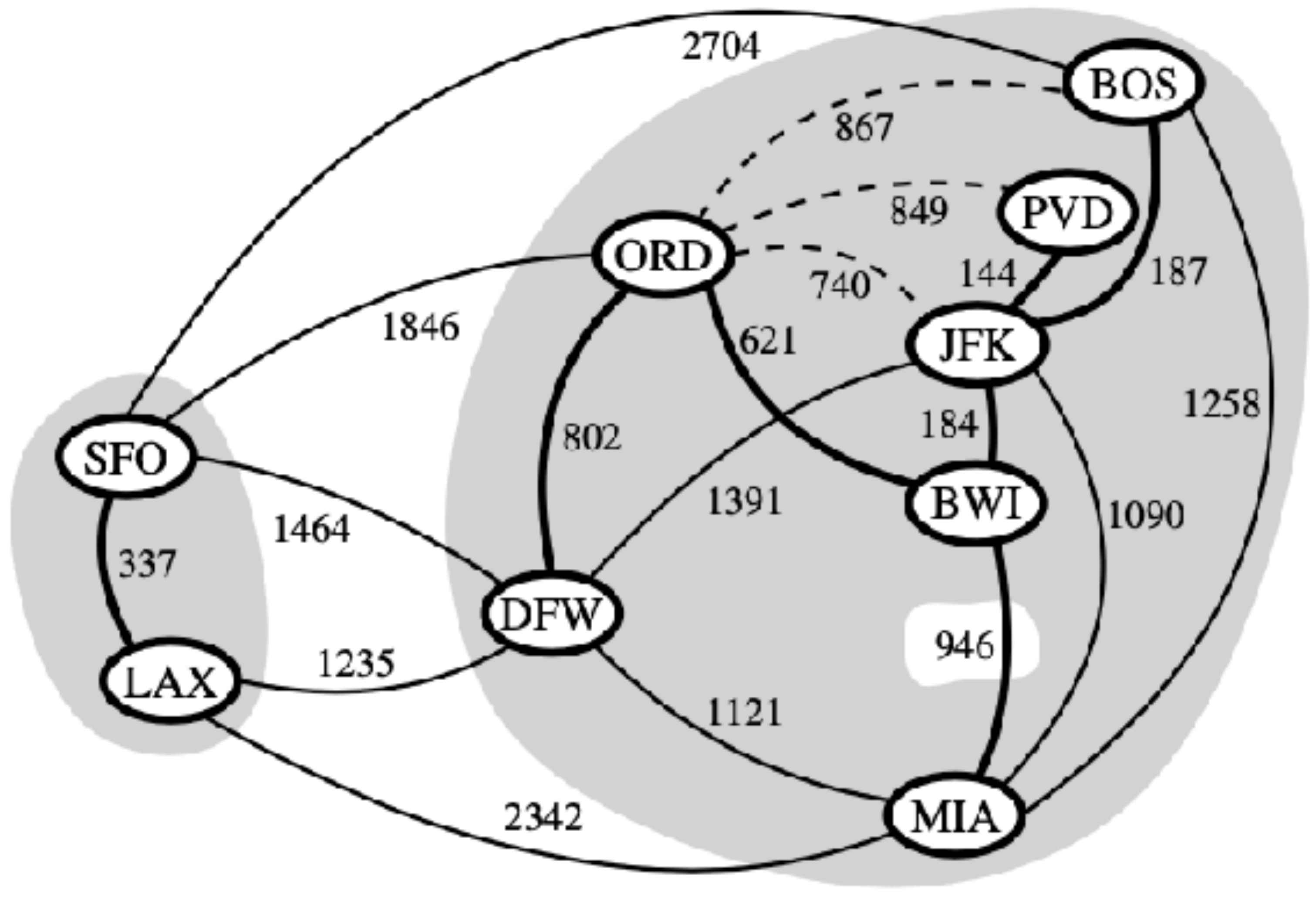
Minimum Spanning Trees - Kruskal's Algorithm



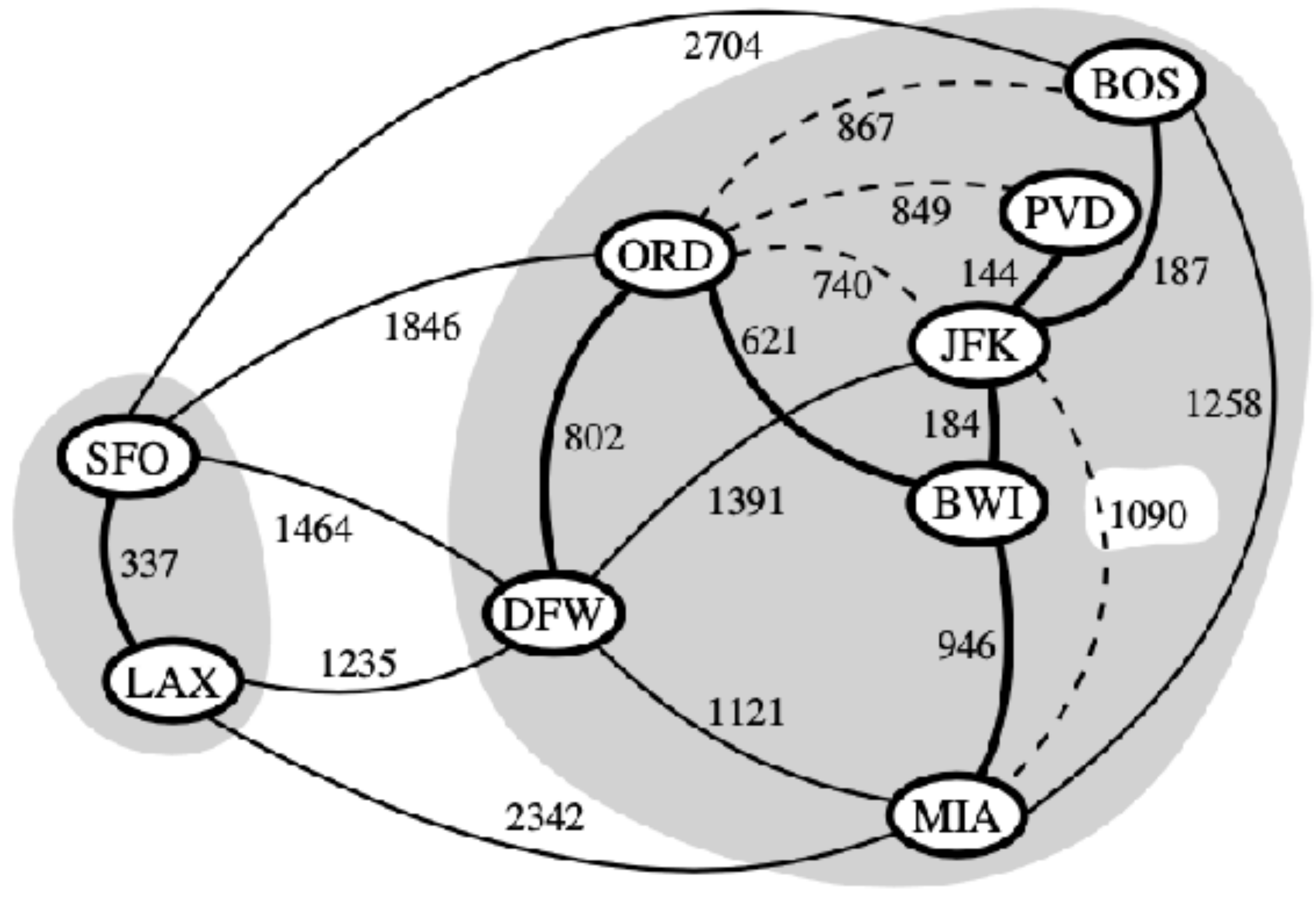
(i)



(j)



(k)



(l)

Minimum Spanning Trees - Kruskal's Algorithm

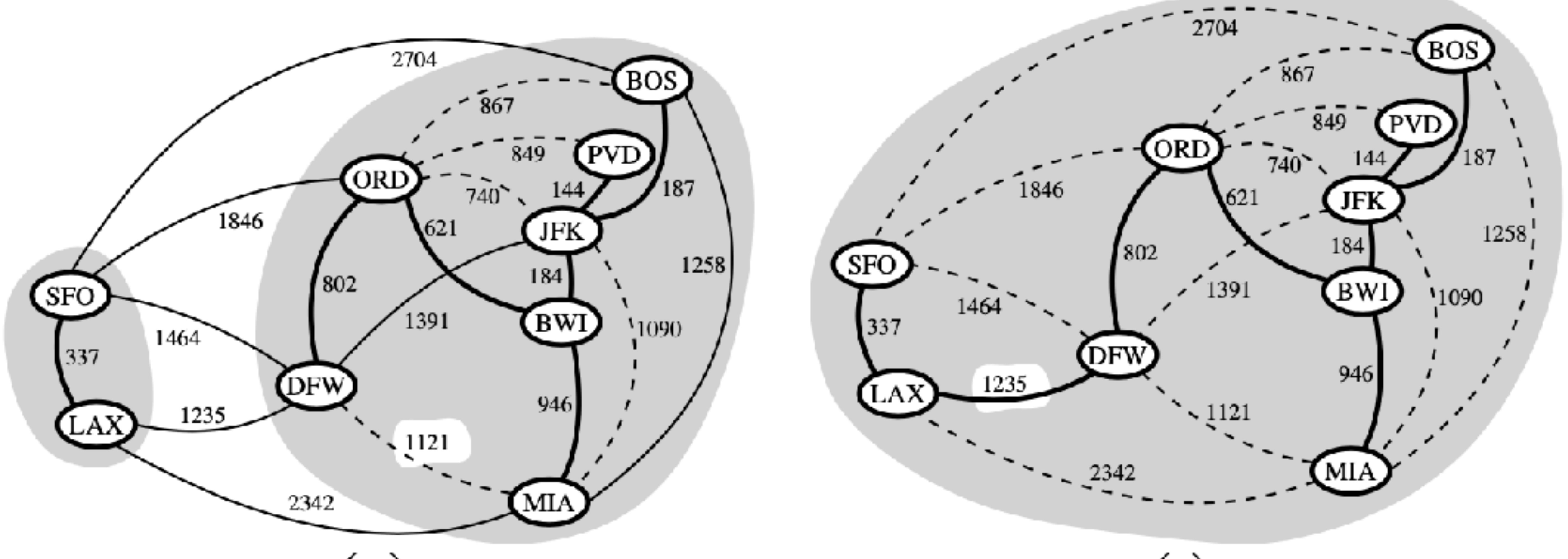
Algorithm Kruskal(G):

Input: A simple connected weighted graph G with n vertices and m edges
Output: A minimum spanning tree T for G

for each vertex v in G **do**
  Define an elementary cluster $C(v) = \{v\}$.
Initialize a priority queue Q to contain all edges in G, using the weights as keys.
$T = \emptyset$ $\{T$ will ultimately contain the edges of the MST$\}$
while T has fewer than $n-1$ edges **do**
  $(u,v) =$ value returned by Q.remove_min()
  Let $C(u)$ be the cluster containing u, and let $C(v)$ be the cluster containing v.
  if $C(u) \neq C(v)$ **then**
    Add edge (u,v) to T.
    Merge $C(u)$ and $C(v)$ into one cluster.
return tree T

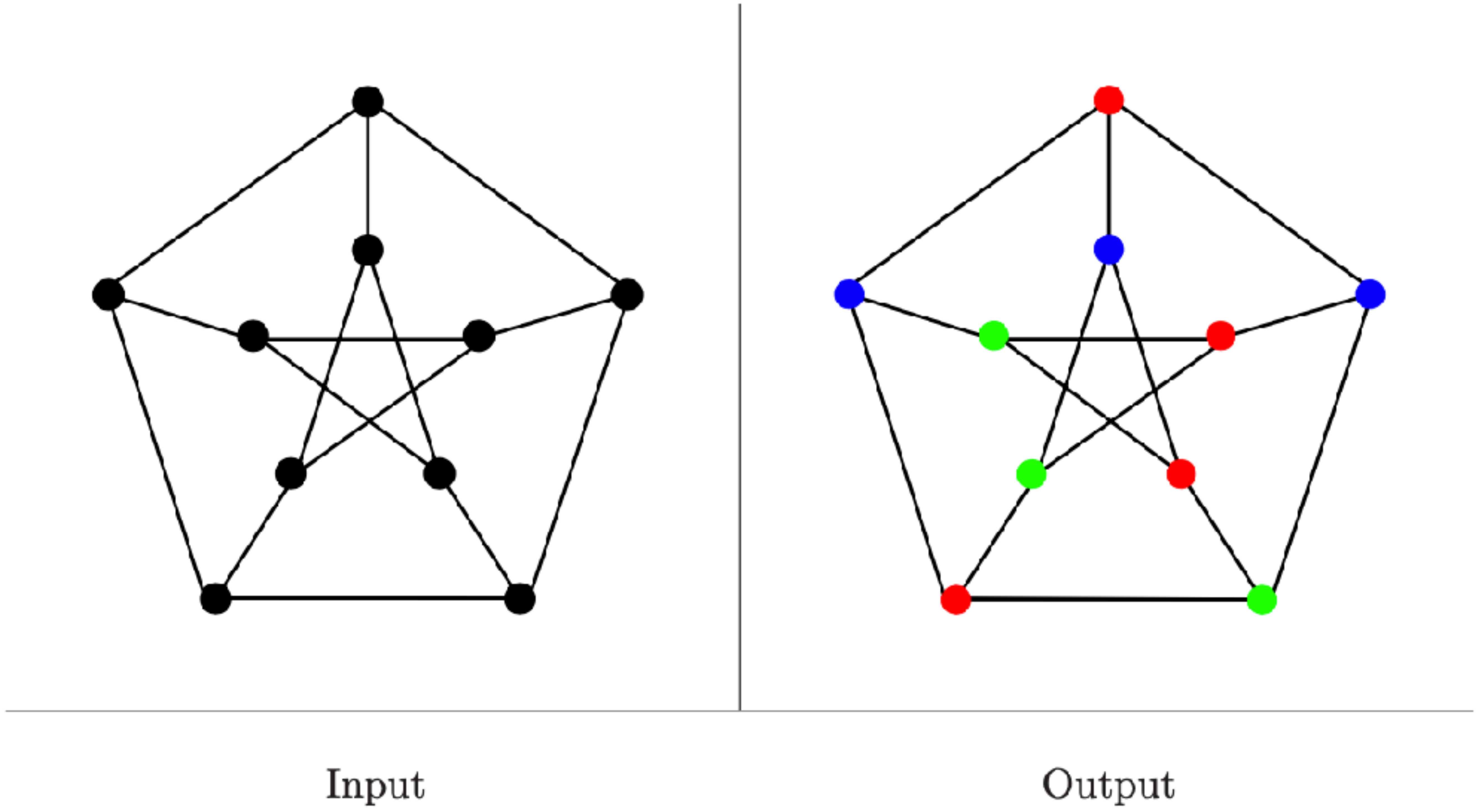
With a more sophisticated data structure the running time of the Kruskal becomes $O(m \log n)$.
See Goodrich's related chapter for details.

Variations ??
- Maximum spanning tree
- Minimum product tree



(m) (n)

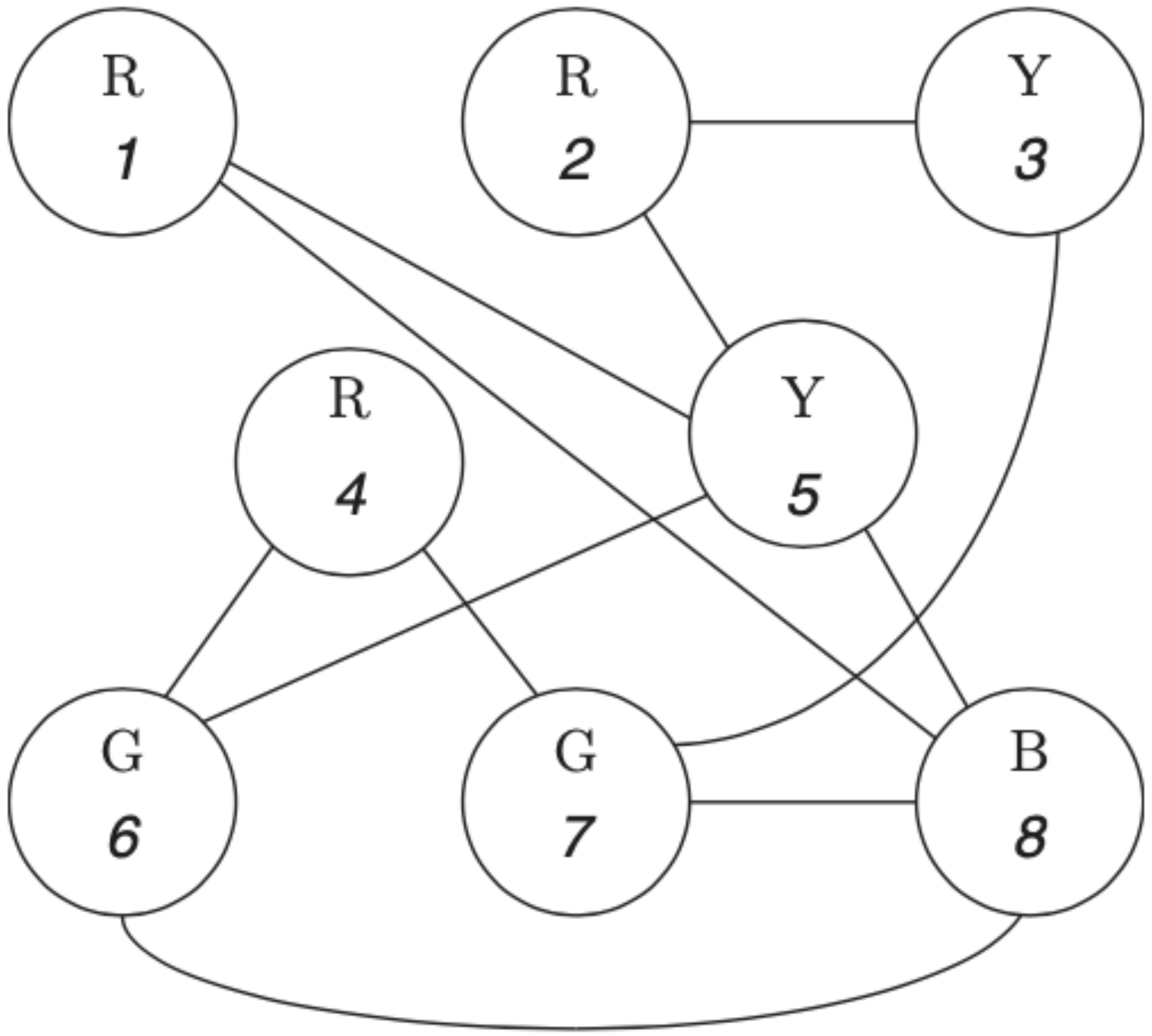
Graph Coloring



- Vertex coloring: What is the minimum number of colors such that no two adjacent vertices share the same?
- NP-complete, so we need some heuristics…

Graph Coloring

```
Greedy (G)
  for (v=1; v<=n; v++)
      for (c=1; c<=n; c++)
          if (G.checkColor(c, v)) {
              G.assignColor(c, v)
              break     // skip to next vertex
          }
  return G.colorCount()
```



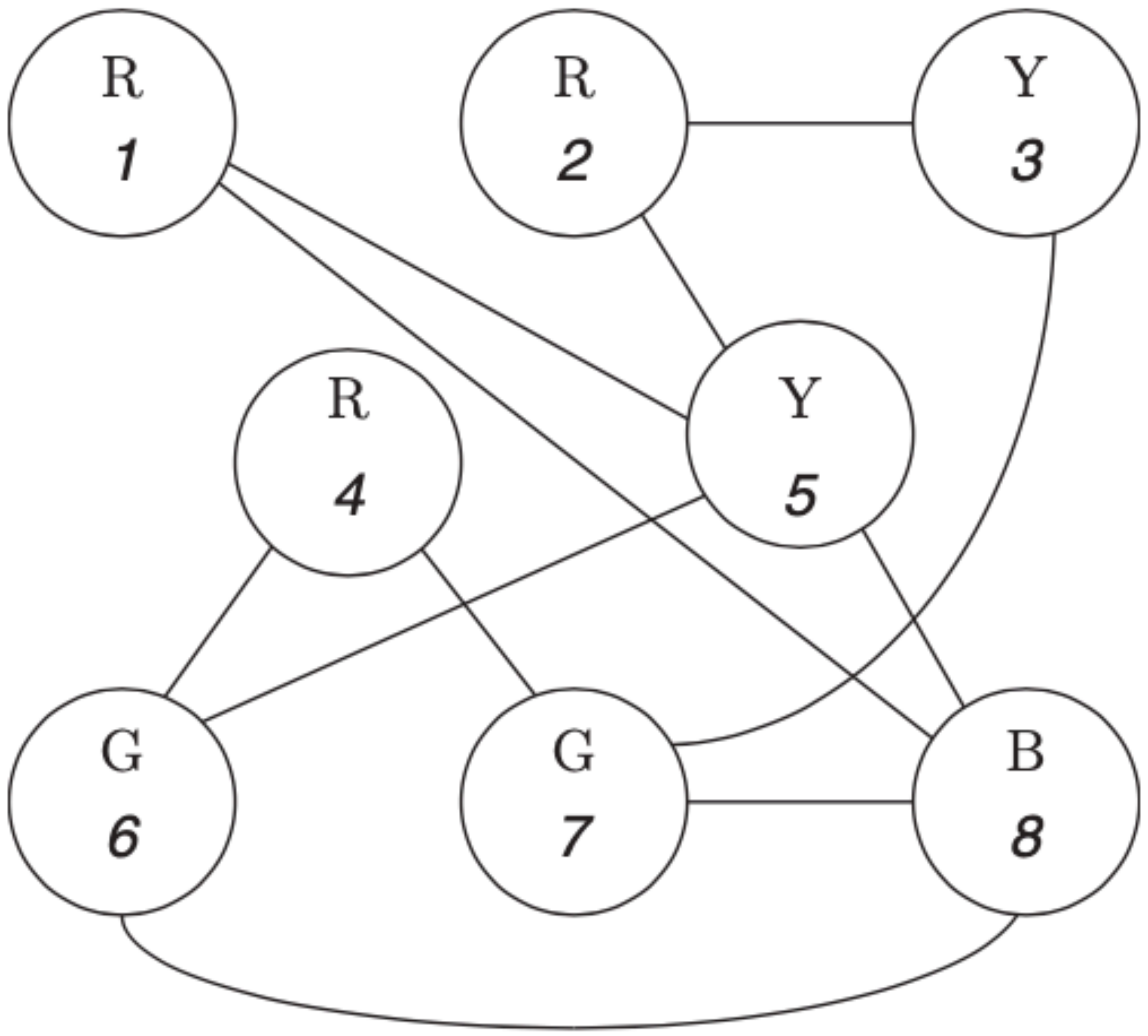
- checkColor(c,v): Is it valid to assign color c to vertex v

1	2	3	4	5	6	7	8
Red	Red	Yellow	Red	Yellow	Green	Green	Blue

- Apply Greedy() on the above graph by visiting vertexes 1,2,3,4,5,6,7,8

Graph Coloring

```
Random (G, I)
   bestCount = Infinity
   bestColoring = null
   for (i=1; i<=I; i++){
       G.unColor()          //remove colors
       G.randomVertexOrder()
       count = Greedy(G)
       if (count < bestCount) {
          bestCount = count
          bestColoring = G.saveColoring()
       }
   }
   report (bestColor, bestCount)
```



- Improving accuracy by visiting the vertices in different permutations
- 8,7,6,5,4,3,2,1 will yield a better one …

8	7	6	5	4	3	2	1
Red	Yellow	Yellow	Green	Red	Red	Yellow	Green

Wedding Seating Problem

- In a wedding N guests will be split among k tables with some constraints such that some of them cannot sit on the same table.
- What is the minimum number of tables or will X number of tables be enough to satisfy all such constraints?
- Many versions exist…
- NP-hard as it needs solving the graph coloring problem with N vertices, where an edge connects two persons that cannot sit on the same table.

Wedding Seating Problem

- Assume 50 people will attend and say we have 5 tables.
- Randomly assign 10 people to each table and check for the constraints. Too many possibilities to consider !
- We may relax the problem assuming that the following information is given
  - **The groups of people that should be seated together.**
  - **The people that should be seated at different tables**

Wedding Seating Problem

- On a binary matrix W_{NxN} , $W_{i,j} = 1$ if i and j should seat apart, otherwise $W_{i,j} = 0$.
- Create subsets $S = \{s_1, s_2, \ldots, s_k\}$that minimizes the objective function

$$\sum_{t=1}^{k} \sum_{\forall i,i \in s_t; i<j} W_{i,j}$$

- Now, assume also
  - no group (subset) is larger then N/k, and,
  - **it is more important to divide the people who have conflict than to minimize the tables.**

Wedding Seating Problem

- A solution approach with these assumptions:

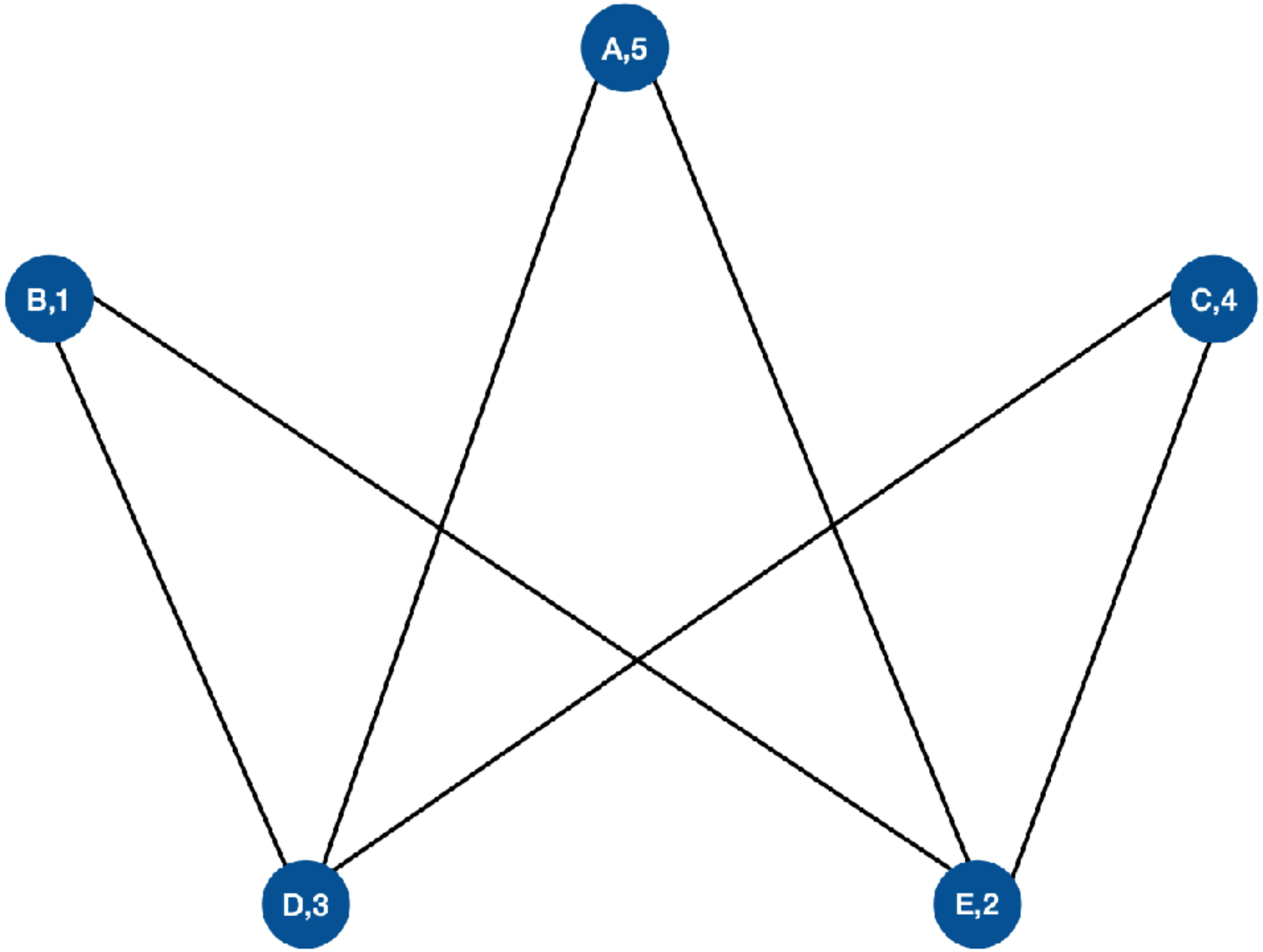
1. Assign each group to a vertex, where vertex weight is equal to the number of people in the group.
2. Two vertices are connected with an edge if there are conflicting guests between the groups.
3. We try to solve graph coloring of this graph (we can try the previous heuristics)

Group A	Ayşe	Buğra	Ceyda	Deniz	Emre
Group B	Ferman				
Group C	Gizem	Hıdır	Ilgın	Kayra	
Group D	Leyla	Mehmet	Nazlı		
Group E	Osman	Pelin			

A, B, C : RED
D, E : GREEN

Conflicts:

Ayşe-Leyla (Group A - Group D)
Buğra- Pelin (Group A - Group E)
Ferman-Mehmet (Group B - Group D)
Ferman- Pelin (Group B - Group E)
Gizem-Osman (Group C - Group E)
Ilgın-Nazlı (Group C - Group D)



Wedding Seating Problem

- Now we have different groups colored same:
- Since, tables have a fixed number of seats, we now need to partition groups with the same colors. We need to solve k-partitioning: given N integers would it be possible to split them into k subsets with equal sums? (NP-complete again, but we have some heuristics and pseudo-polinomial solutions with dynamic programming)
- For example, in our example, (A,5) (B,1) (C,4) have same color, meaning no conflict in between them. Set {5,1,4} can be partitioned into 2 equal parts as 5,4+1
- (D,3) (E,2) can also be combined to a table with 5 people.
- Therefore, three tables each with 5 people would solve the problem.

Reading assignment

- Skiena chapter 8.
- Goodrich et al. 14.6, 14.7