Fundamentals of Data Structure

Mahesh Shirole

VJTI, Mumbai-19

Slides are prepared from

1. Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014

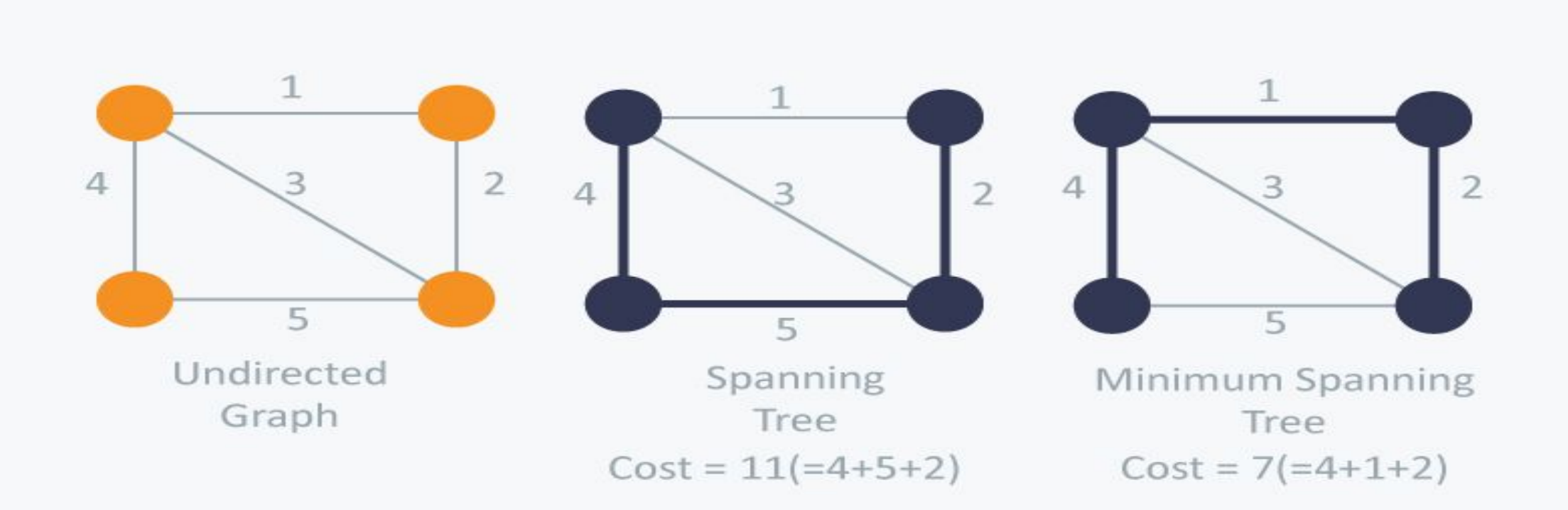
2.Data Structures and Algorithms in Java, by Robert Lafore, Second Edition, Sams Publishing

Minimum Spanning Trees

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

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

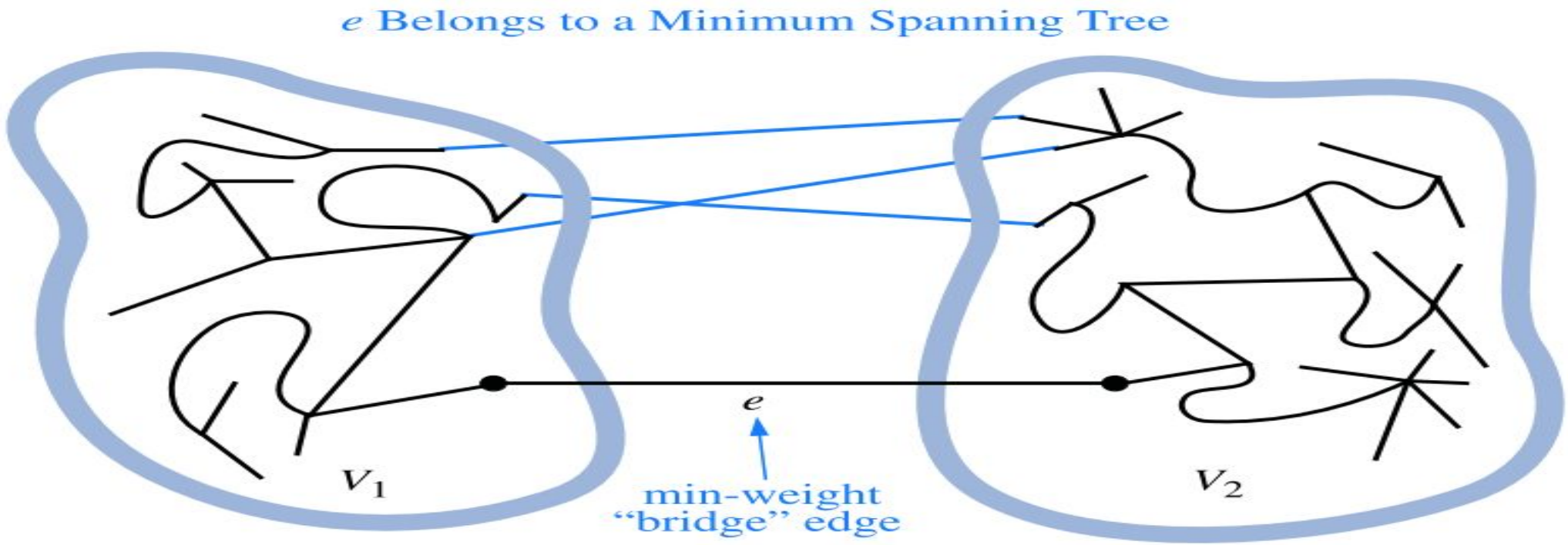
- A tree, such as this, that contains every vertex of a connected graph ***G*** is said to be a spanning tree, and the problem of computing a spanning tree ***T*** with smallest total weight is known as the minimum spanning tree (or MST) problem



A Crucial Fact about Minimum Spanning Trees

- The two MST algorithms we discuss are based on the greedy method, which in this case depends crucially on the following fact

Proposition 14.25: *Let G be a weighted connected graph, and let V_1 and V_2 be a partition of the vertices of G into two disjoint nonempty sets. Furthermore, let e be an edge in G with minimum weight from among those with one endpoint in V_1 and the other in V_2. There is a minimum spanning tree T that has e as one of its edges.*



Prim-Jarn´ık Algorithm

- In the Prim-Jarn´ık algorithm, we grow a minimum spanning tree from a single cluster starting from some “root” vertex ***s***
- We will begin with some vertex ***s***, defining the initial “cloud” of vertices ***C***
- Then, in each iteration, we choose a minimum-weight edge ***e= (u,v)***, connecting a vertex ***u*** in the cloud ***C*** to a vertex ***v*** outside of ***C***
- The vertex ***v*** is then brought into the cloud ***C*** and the process is repeated until a spanning tree is formed

Prim-Jarník Algorithm

Algorithm PrimJarnik(G):

Input: An undirected, weighted, connected graph G with n vertices and m edges

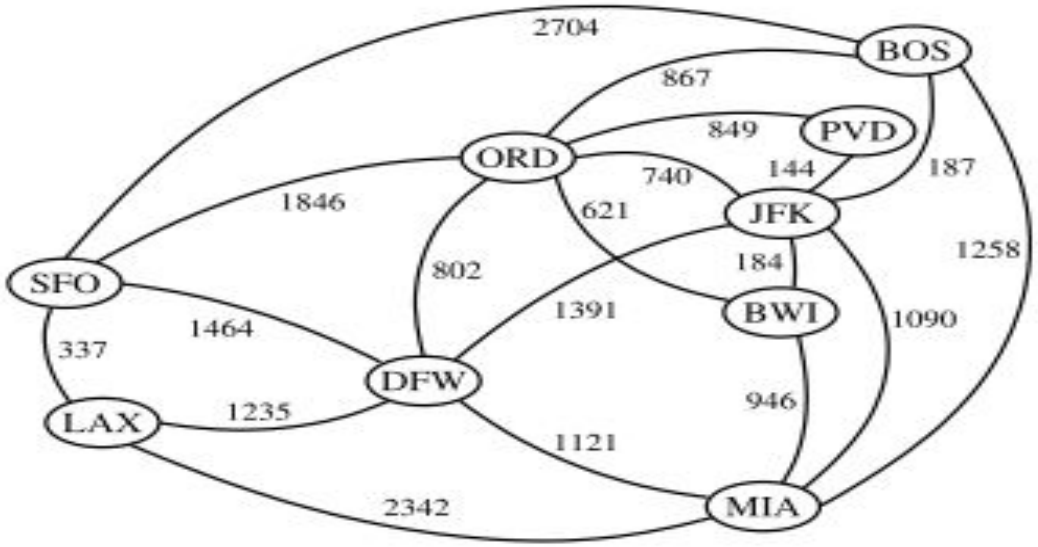
Output: A minimum spanning tree T for G

Pick any vertex s of G
$D[s] = 0$
for each vertex $v \neq s$ **do**
  $D[v] = \infty$
Initialize $T = \emptyset$.
Initialize a priority queue Q with an entry $(D[v], v)$ for each vertex v.
For each vertex v, maintain connect(v) as the edge achieving $D[v]$ (if any).
while Q is not empty **do**
  Let u be the value of the entry returned by Q.removeMin().
  Connect vertex u to T using edge connect(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**
      $D[v] = w(u, v)$
      connect(v) $= e'$.
      Change the key of vertex v in Q to $D[v]$.
return the tree T

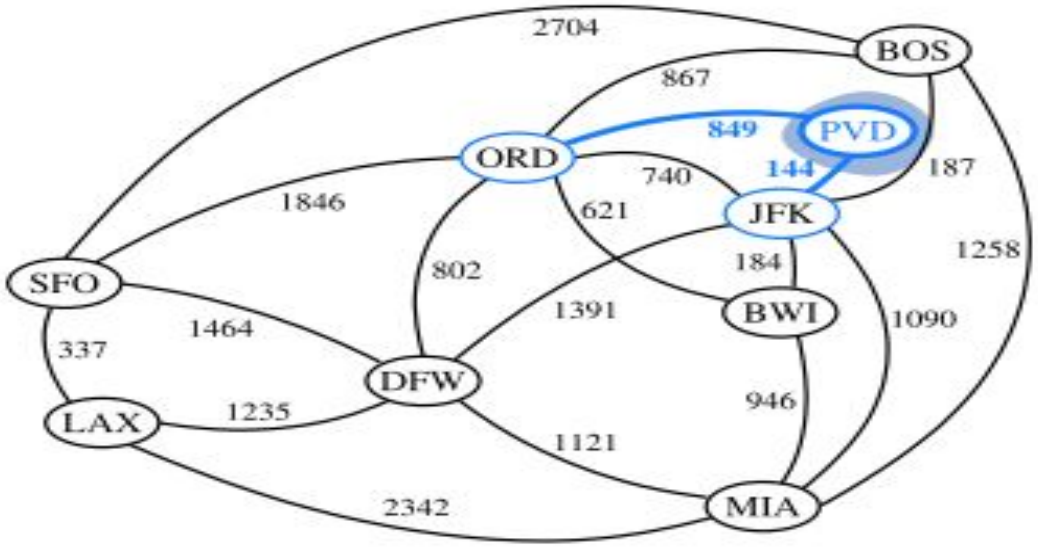
Analyzing the Prim-Jarn´ık Algorithm

- We initially perform ***n*** insertions into ***Q***, later perform n extract-min operations, and may update a total of m priorities as part of the algorithm
- Those steps are the primary contributions to the overall running time
- With a heap-based priority queue, each operation runs in ***O(log n)*** time, and the overall time for the algorithm is ***O((n+m) log n)***, which is ***O(m log n)*** for a connected graph
- Alternatively, we can achieve $O(n^2)$ running time by using an unsorted list as a priority queue

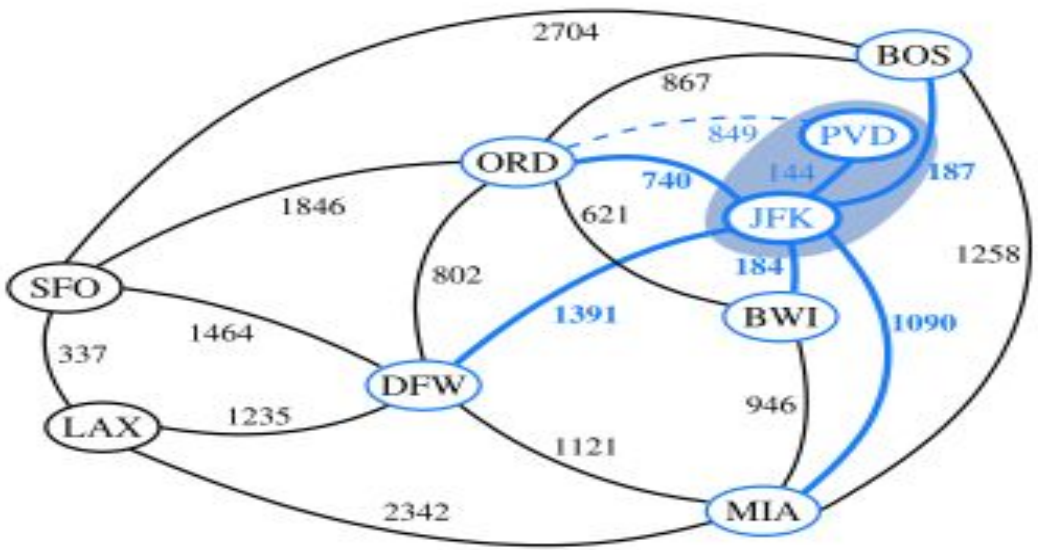
Illustrating the Prim-Jarn´ık Algorithm



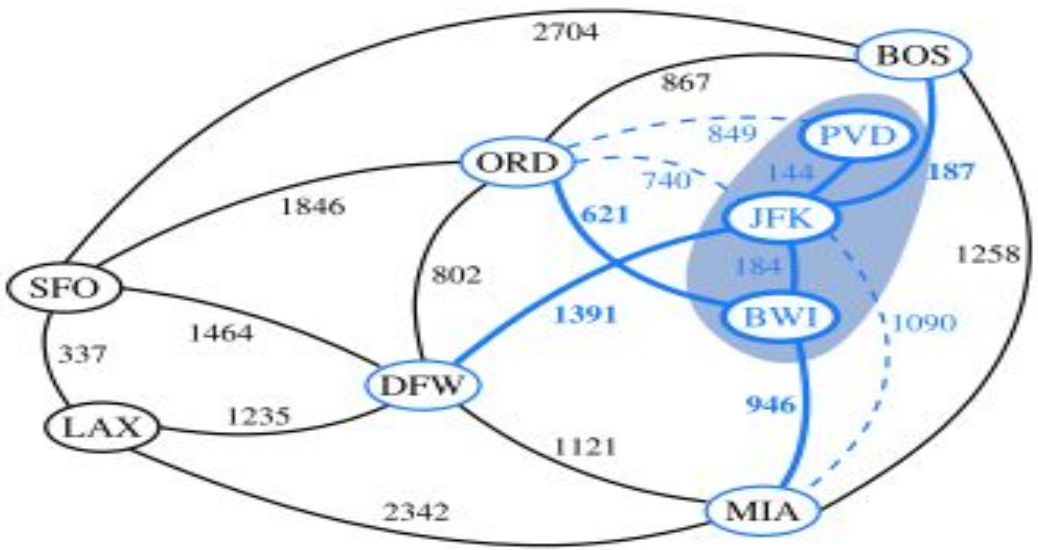
(a)



(b)

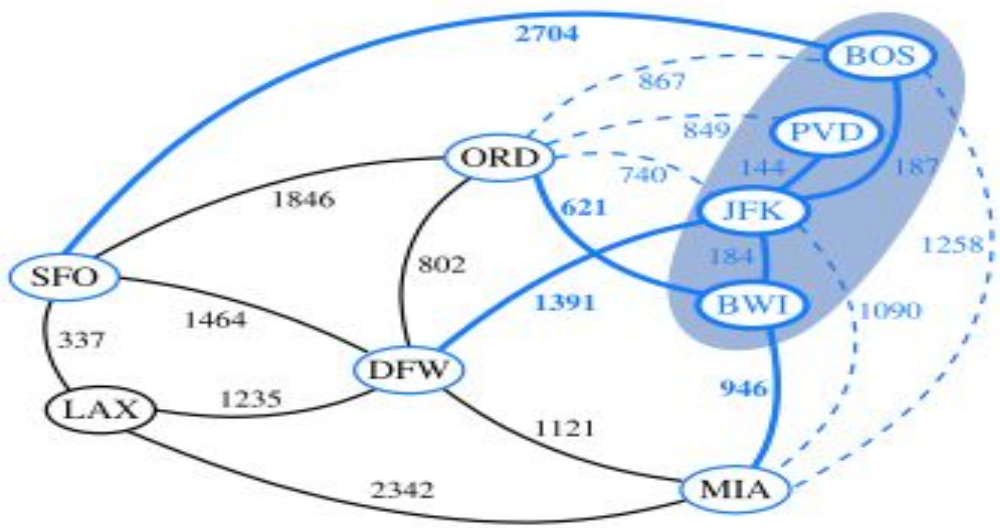


(c)

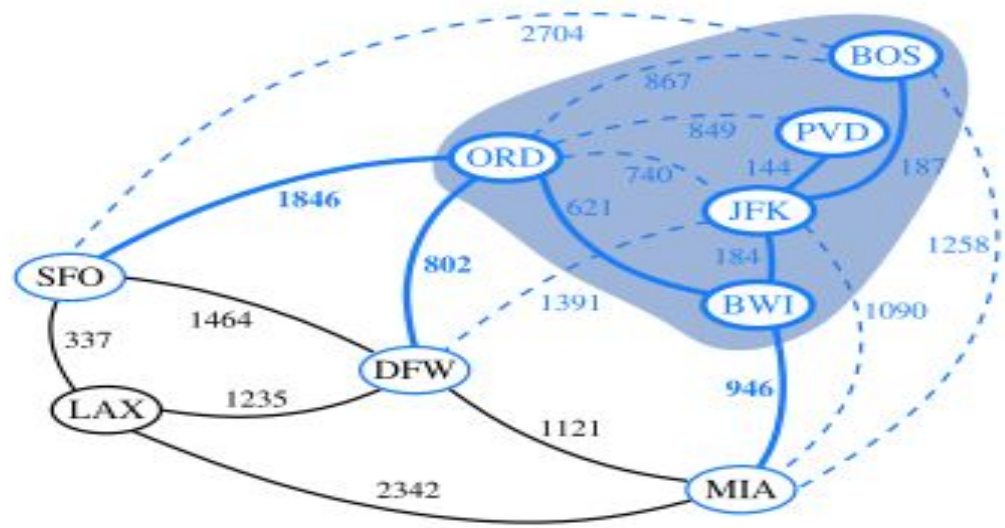


(d)

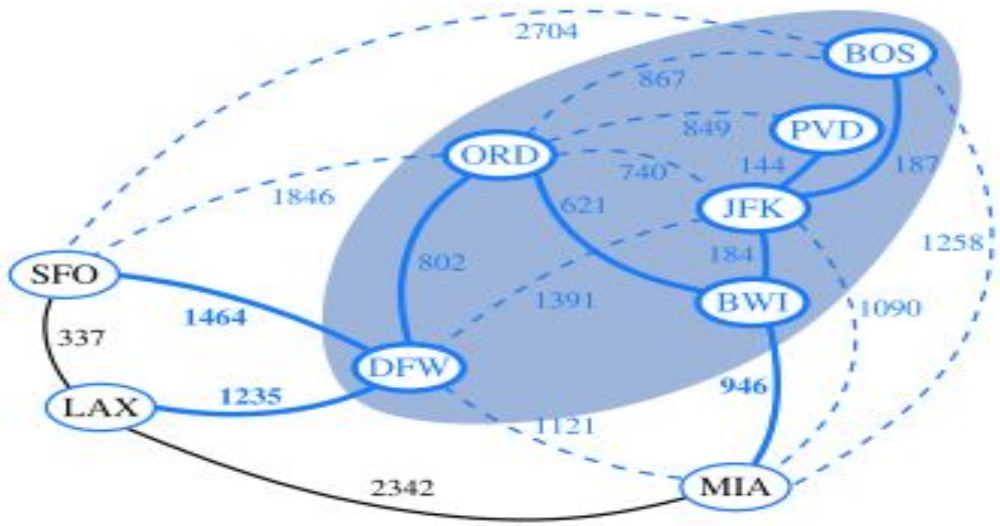
Illustrating the Prim-Jarn´ık Algorithm



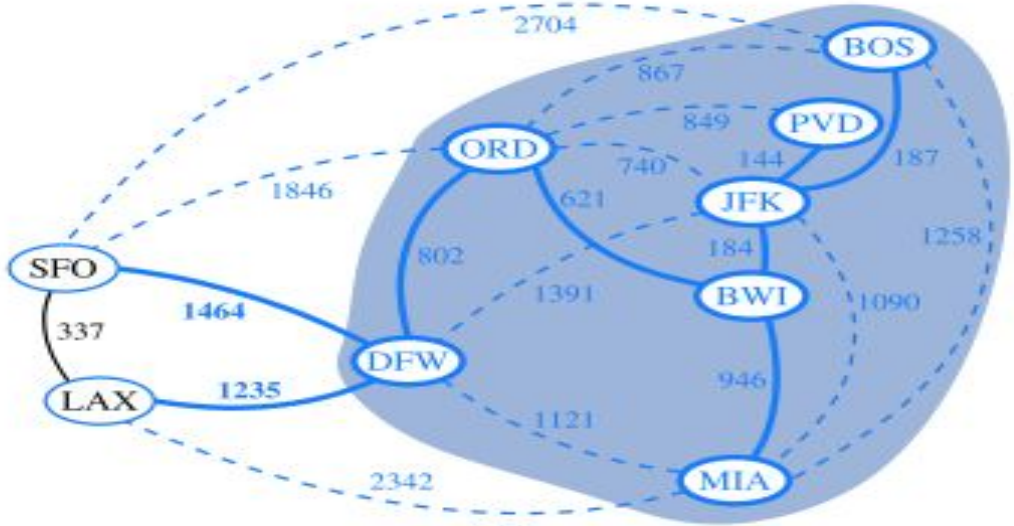
(e)



(f)

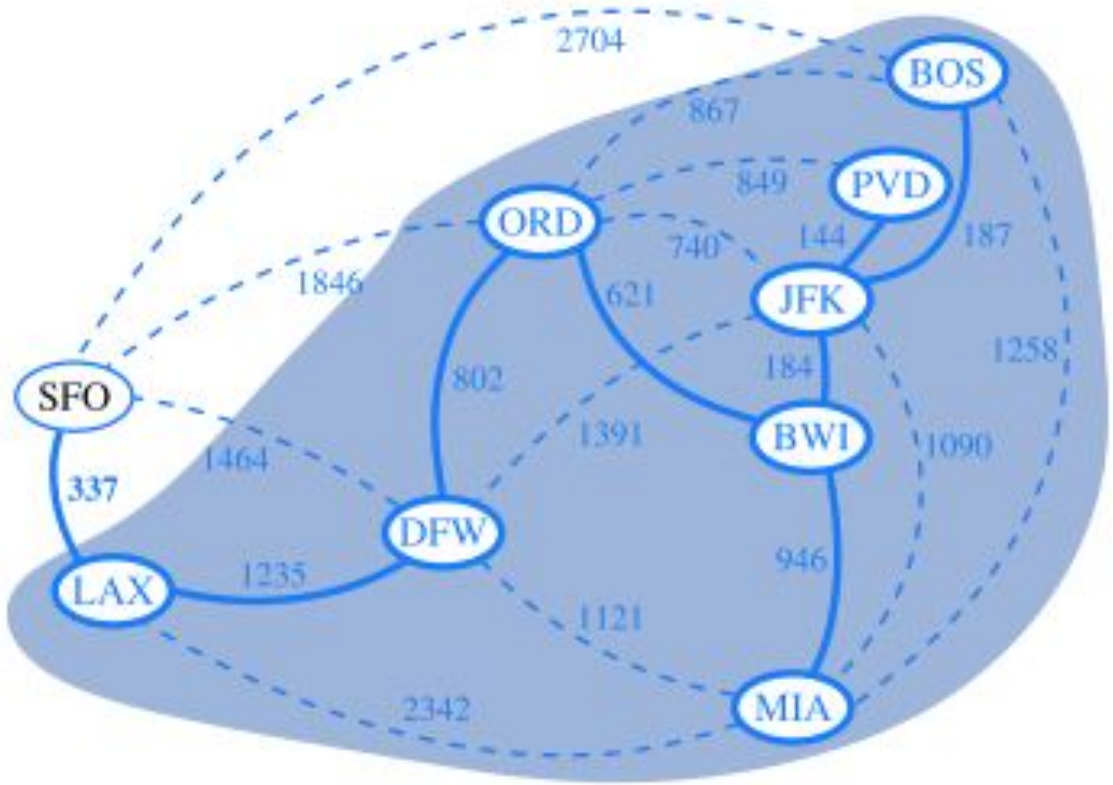


(g)

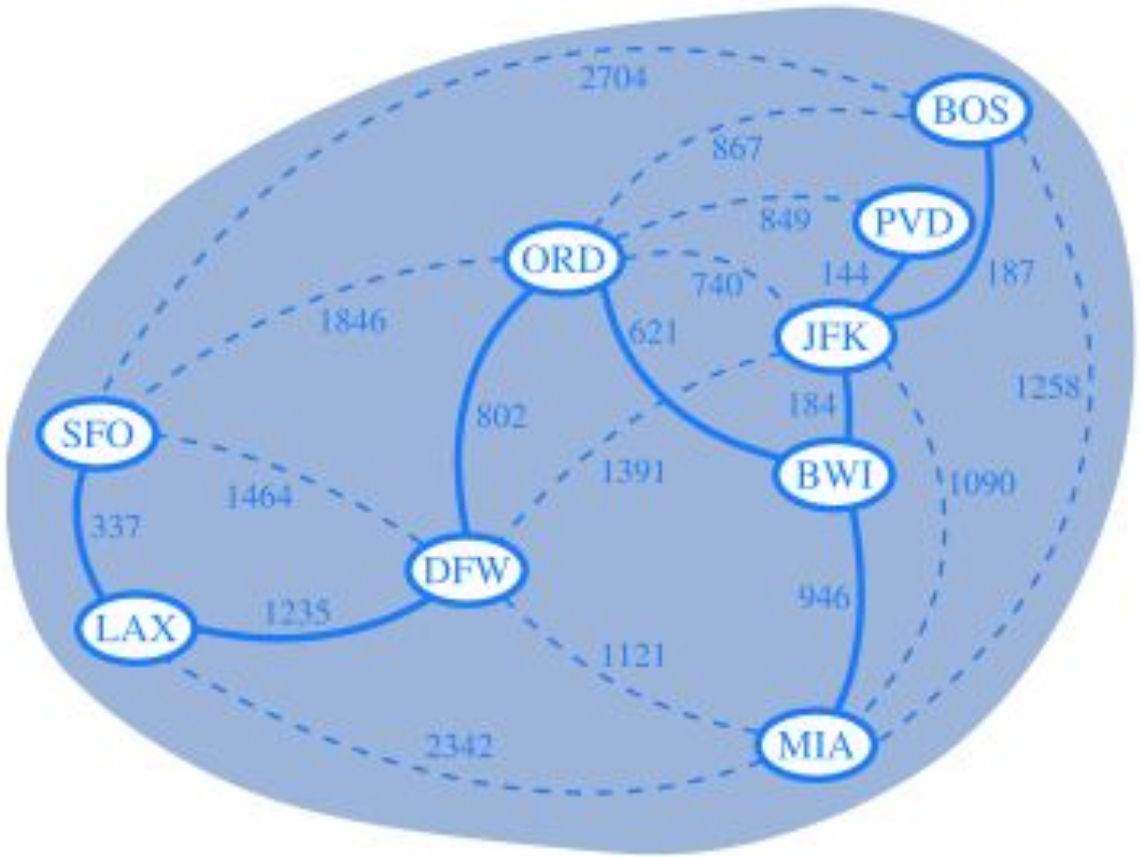


(h)

Illustrating the Prim-Jarn´ık Algorithm



(i)



(j)

Kruskal's Algorithm

- The Prim-Jarn´ık algorithm builds the MST by growing a single tree until it spans the graph, Kruskal's algorithm maintains many smaller trees in a forest, repeatedly merging pairs of trees until a single tree spans the graph

- Initially, each vertex is in its own cluster
- The algorithm then considers each edge in turn, ordered by increasing weight
- If an edge ***e*** connects vertices in two different clusters, then ***e*** is added to the set of edges of the minimum spanning tree, and the two trees are merged with the addition of ***e***
- If, on the other hand, ***e*** connects two vertices in the same cluster, then ***e*** is discarded
- Once the algorithm has added enough edges to form a spanning tree, it terminates and outputs this tree as the minimum spanning tree

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 an MST}

while T has fewer than $n - 1$ edges **do**

$(u, v) =$ value returned by Q.removeMin()

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

The Running Time of Kruskal's Algorithm

- The ordering of edges by weight can be implemented in ***O(m log m)***, either by use of a sorting algorithm or a priority queue ***Q***

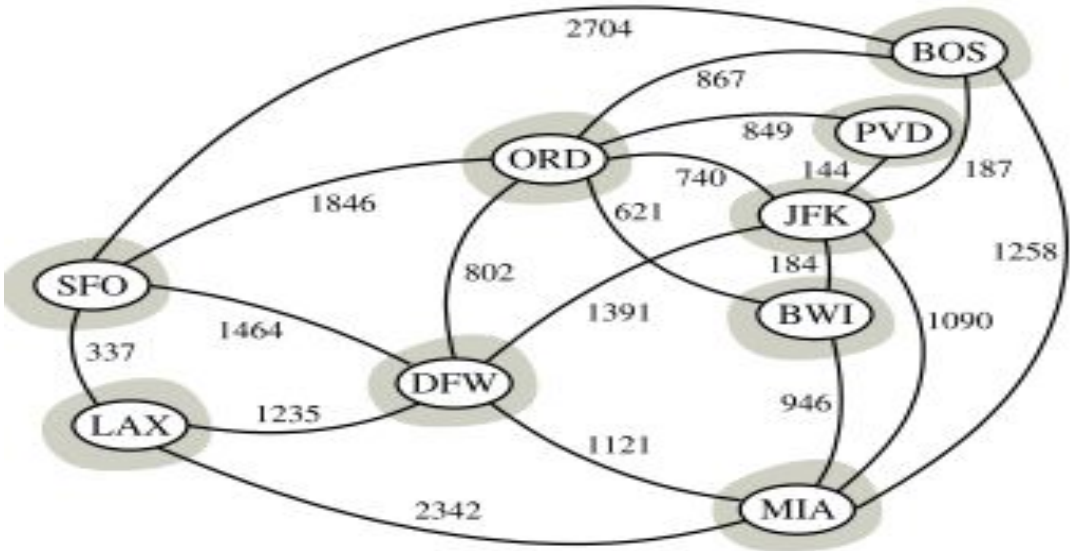
- If that queue is implemented with a heap, we can initialize ***Q*** in ***O(m log m)*** time by repeated insertions

- The subsequent calls to removeMin each run in ***O(log m)*** time, since the queue has size ***O(m)***

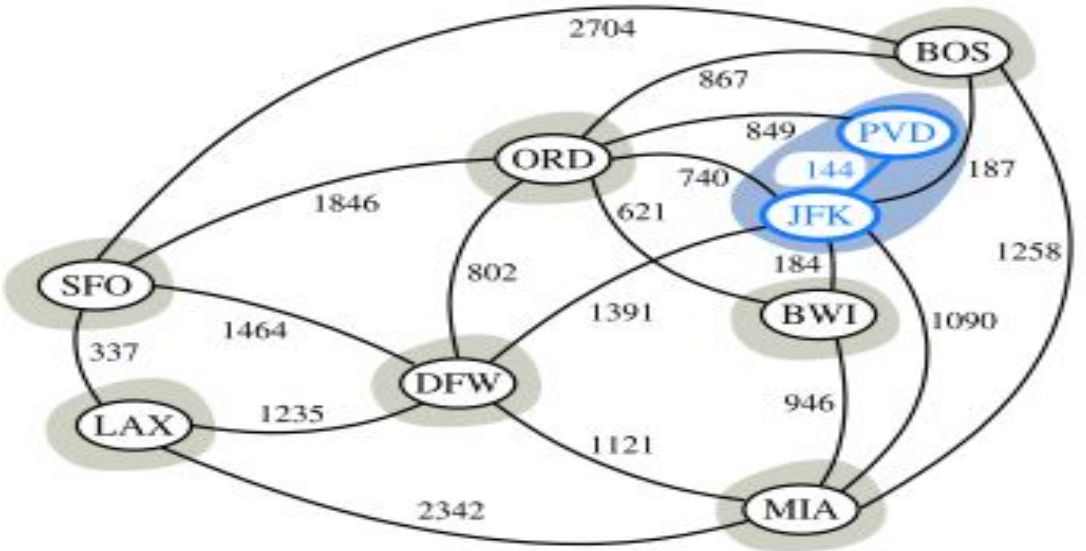
- We note that since m is $O(n^2)$ for a simple graph, ***O(log m)*** is the same as ***O(log n)***

- Therefore, the running time due to the ordering of edges is ***O(m log n)***

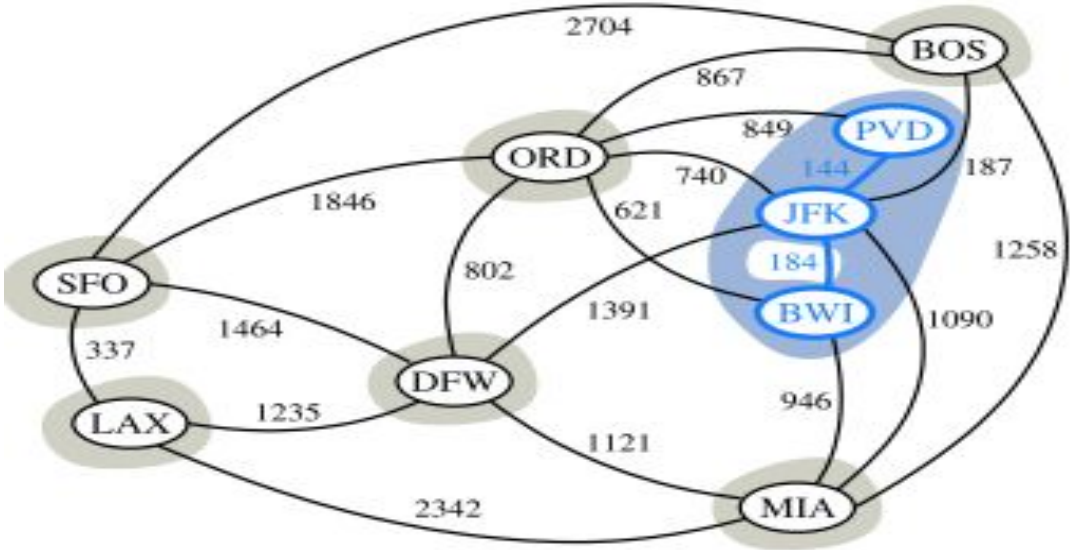
Illustrating the Kruskal's Algorithm



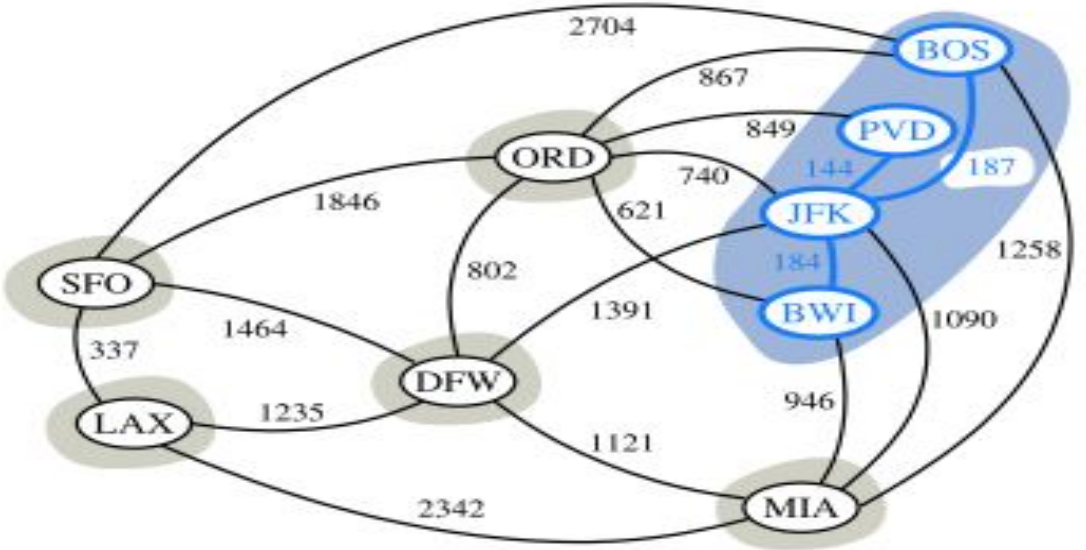
(a)



(b)

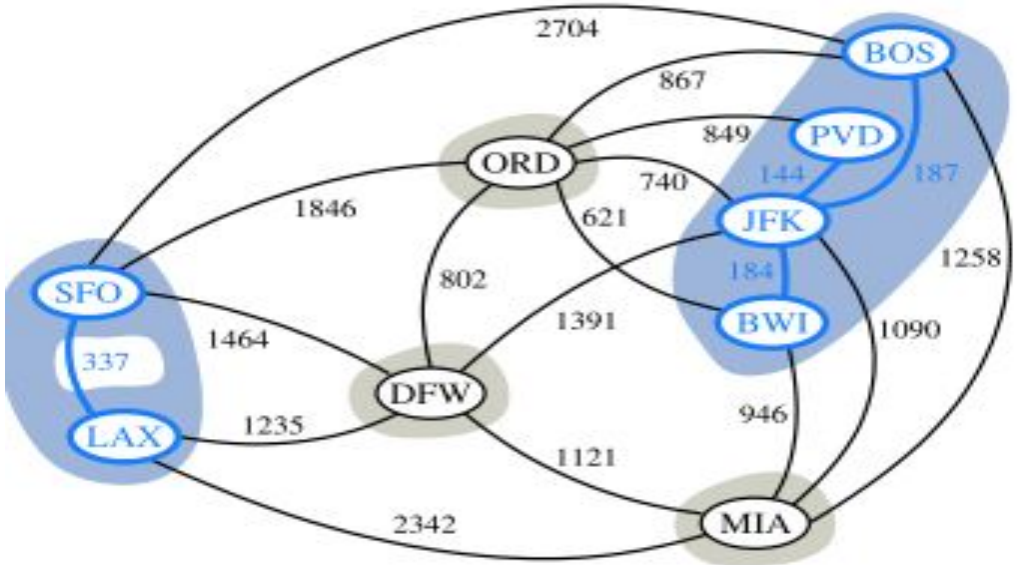


(c)

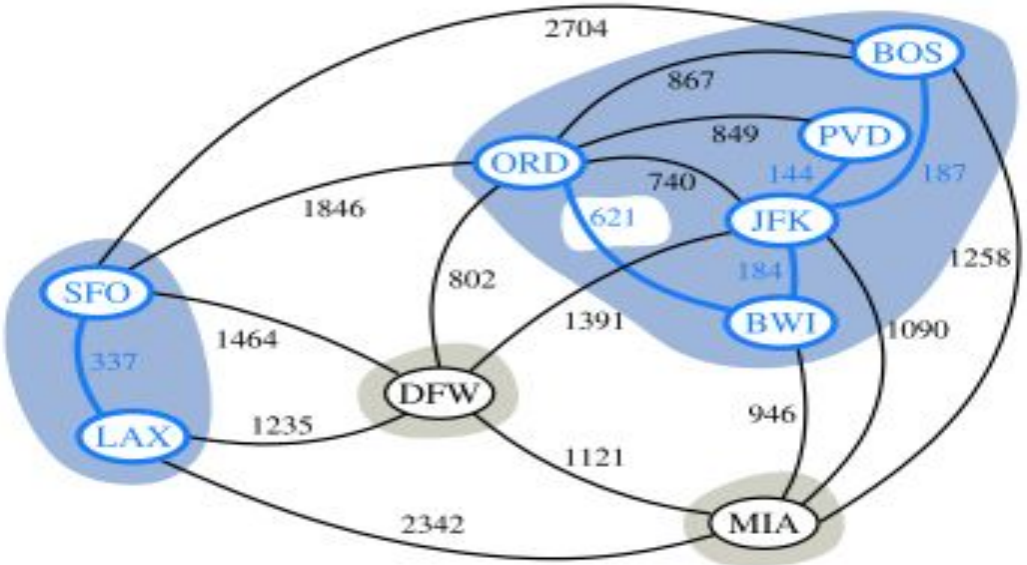


(d)

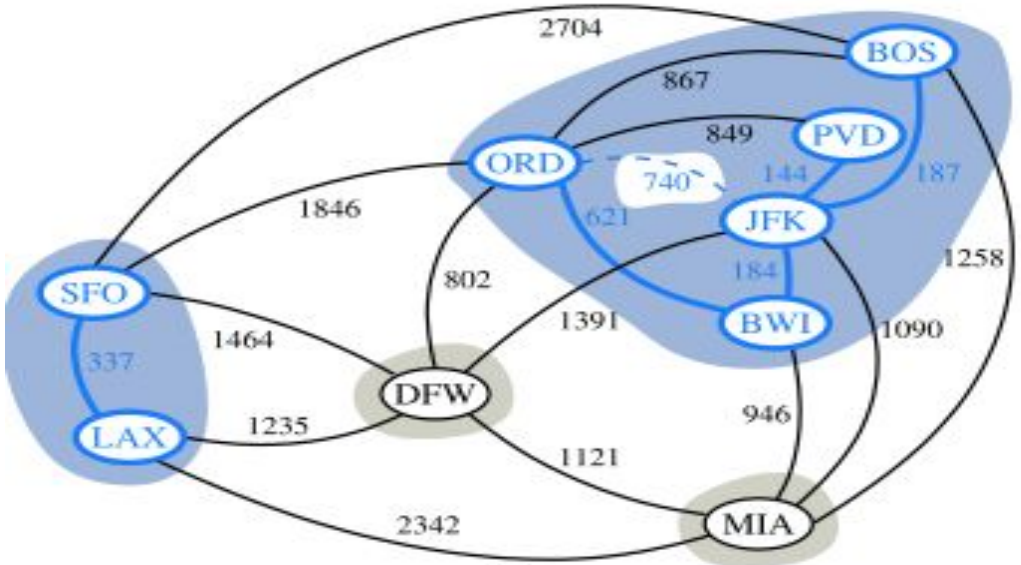
Illustrating the Kruskal's Algorithm



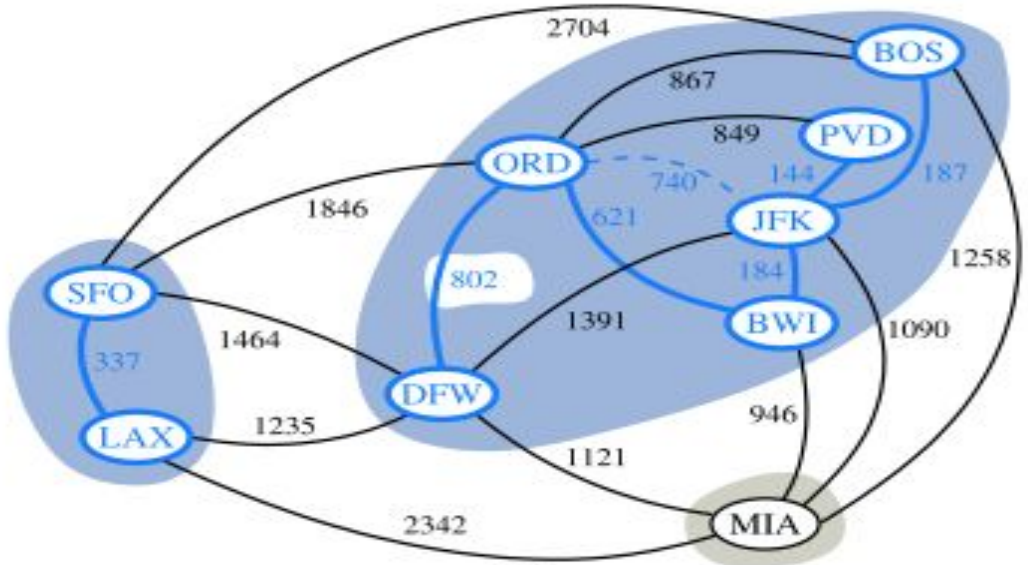
(e)



(f)

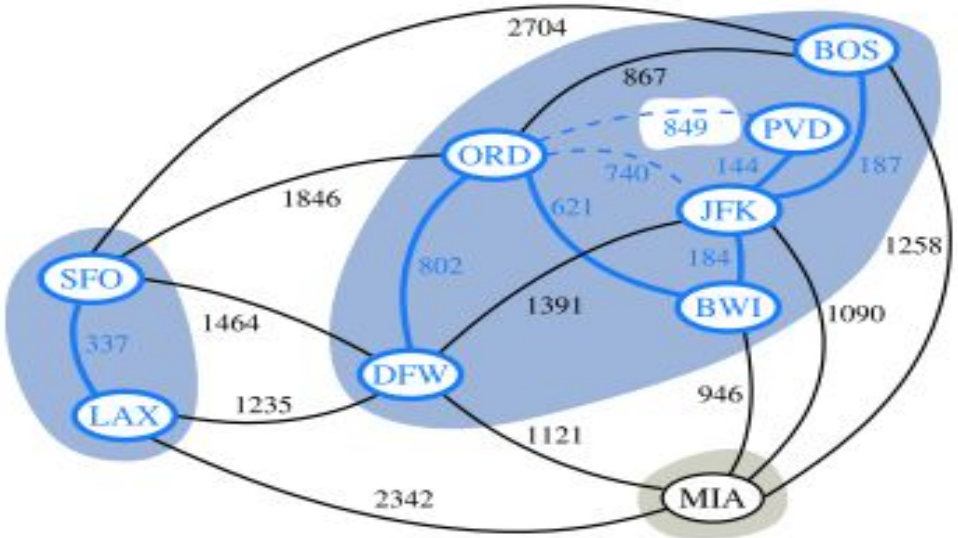


(g)

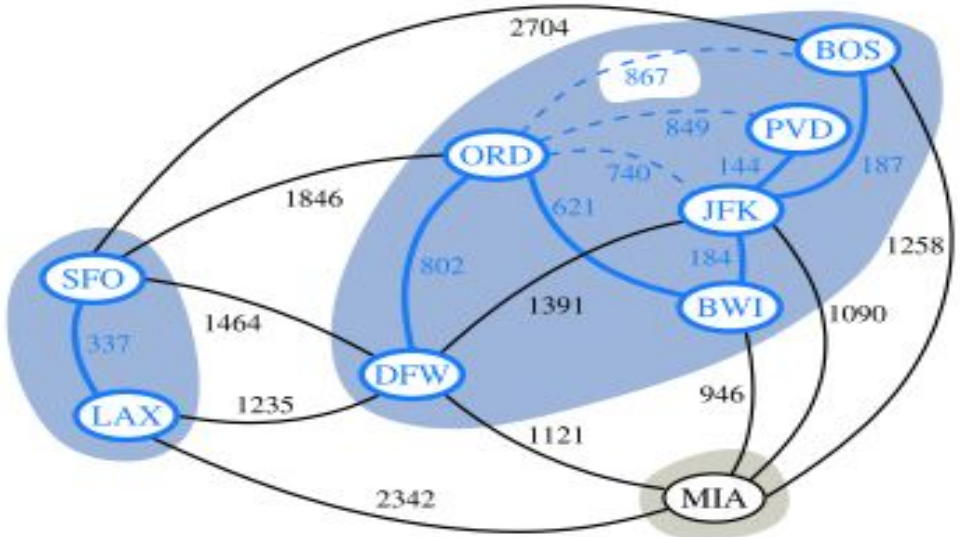


(h)

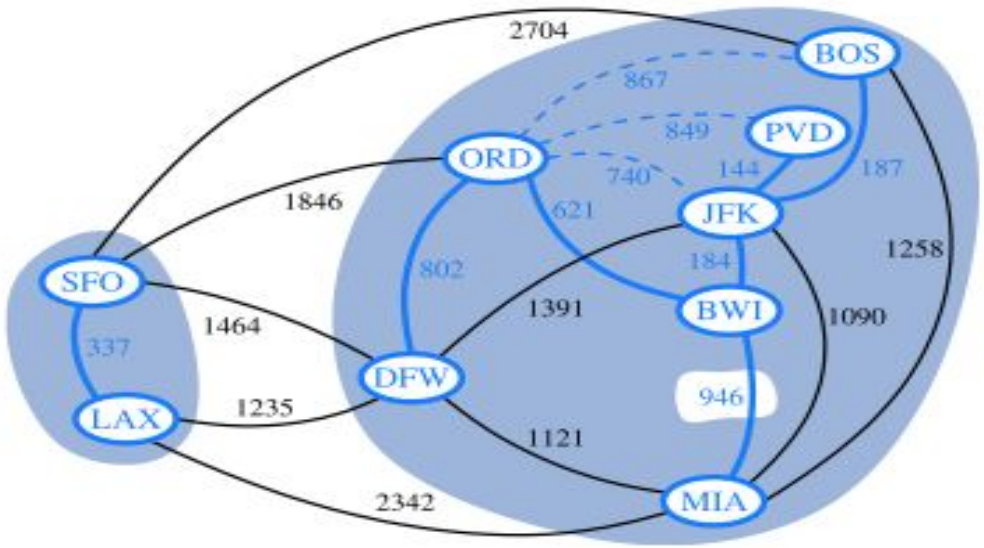
Illustrating the Kruskal's Algorithm



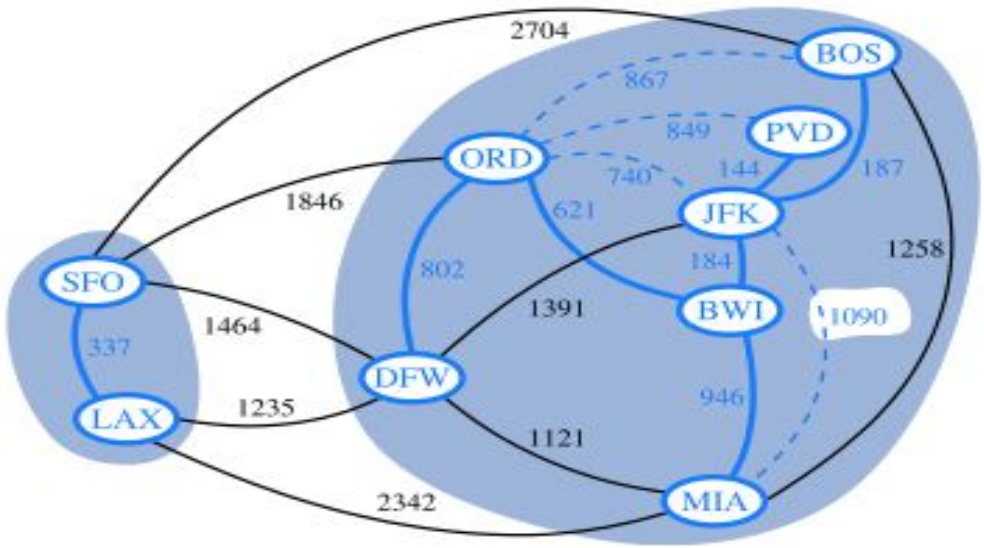
(i)



(j)

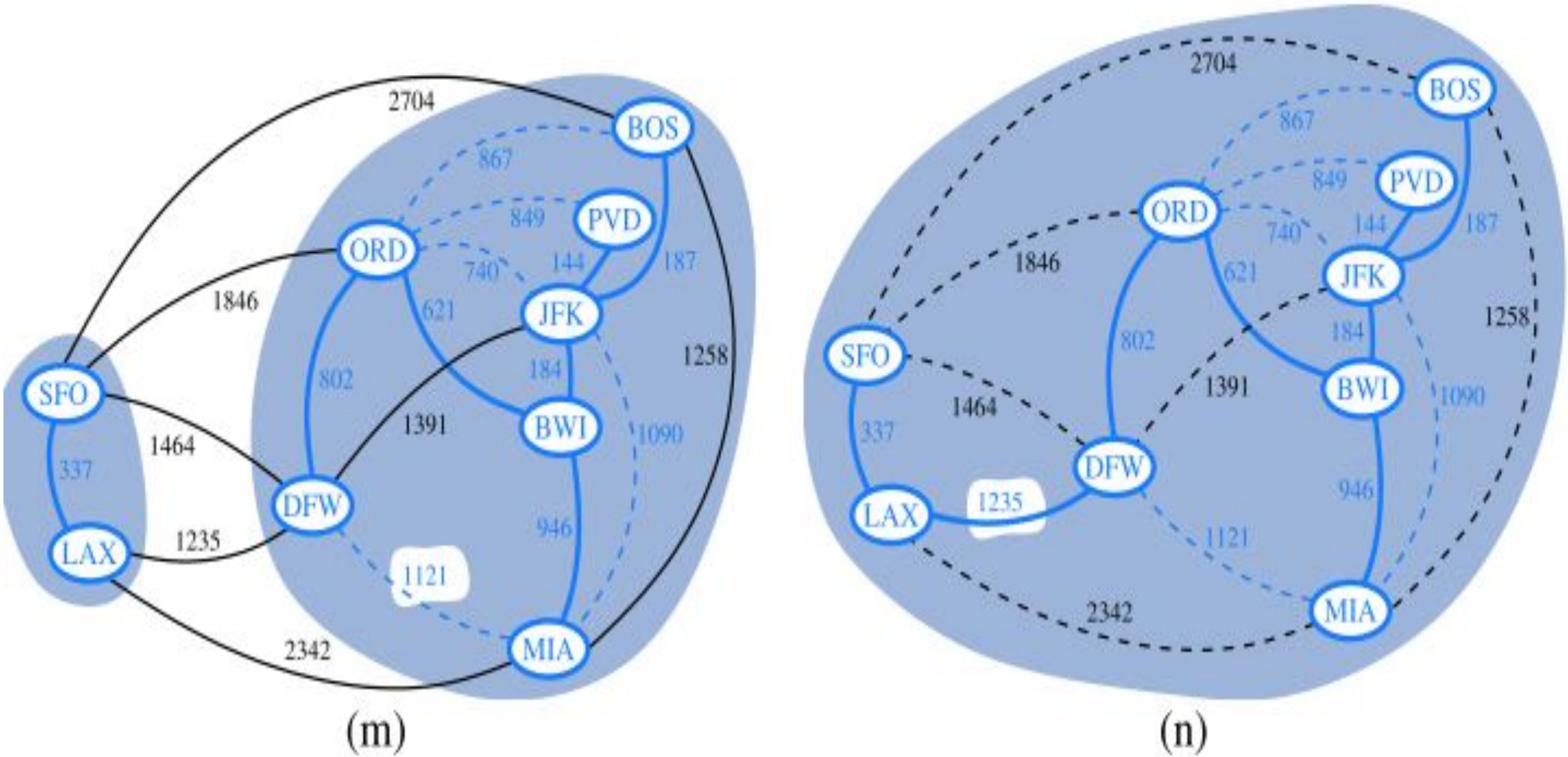


(k)



(l)

Illustrating the Kruskal's Algorithm



Lab Assignment - 8

- Implement Graph using any one representation
- Implement BFS and DFS algorithm
- Implement Prims and Kruskal's algorithm to find minimum spanning tree

- Use graph of all capital cities of all states in India as nodes and distance between them as weight of the edge
- Use website https://www.distancefromto.net/distance-from-bhopal-in-to-mumbai-in to find distance between any two capitals