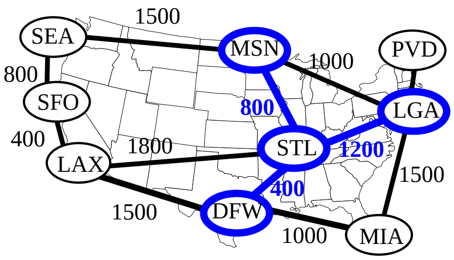
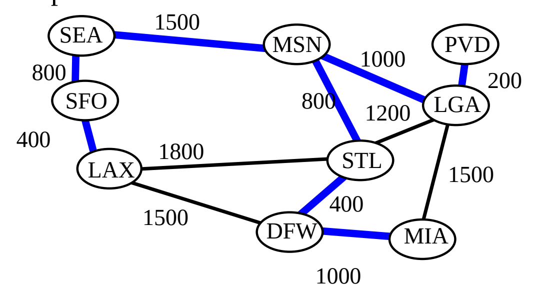
Minimum Spanning Tree

- Prim-Jarnik algorithm
- Kruskal algorithm

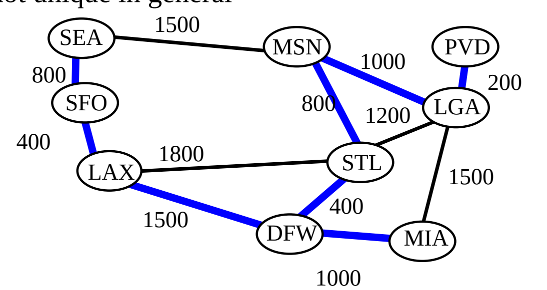


Minimum Spanning Tree

- spanning tree of minimum total weight
- e.g., connect all the computers in a building with the least amount of cable
- example

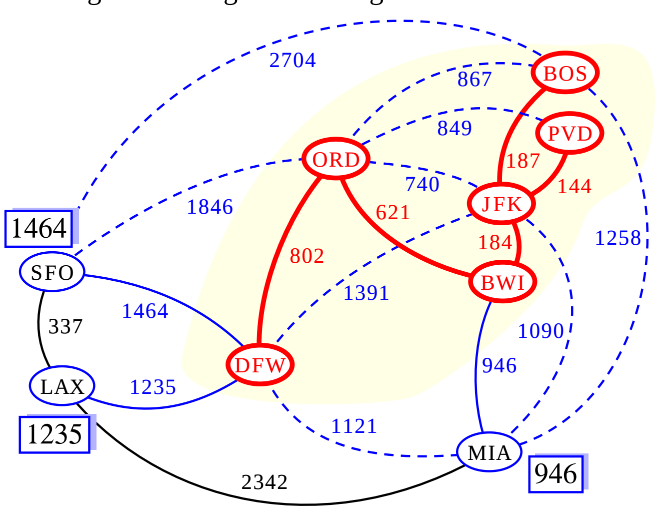


- not unique in general

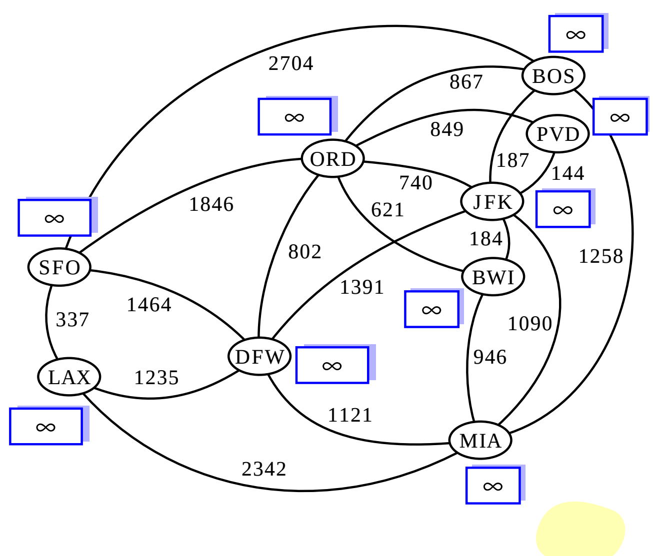


Prim-Jarnik Algorithm

- similar to Dijkstra's algorithm
- grows the tree T one vertex at a time
- cloud covering the portion of T already computed
- labels D[v] associated with vertex v
- if v is not in the cloud, then D[v] is the minimum weight of an edge connecting v to the tree



Example



Pseudo Code

Algorithm PrimJarnik(G):
  Input: A weighted graph G.
  Output: A minimum spanning tree T for G.
pick any vertex v of G
{grow the tree starting with vertex v}
T $\leftarrow \{v\}$
  $D[u] \leftarrow 0$
  $E[u] \leftarrow \emptyset$
for each vertex $u \neq v$ **do**
  $D[u] \leftarrow +\infty$
let Q be a priority queue that contains all the
    vertices using the D labels as keys
while $Q \neq \emptyset$ **do**
    {pull u into the cloud C}
    $u \leftarrow Q$.removeMinElement()
    add vertex u and edge $(u,E[u])$ to T
    for each vertex z adjacent to u **do**
      if z is in Q
        {perform the relaxation operation on edge (u, z) }
        if weight$(u, z) < D[z]$ **then**
          $D[z] \leftarrow$ weight(u, z)
          $E[z] \leftarrow (u, z)$
          change the key of z in Q to $D[z]$
  return tree T

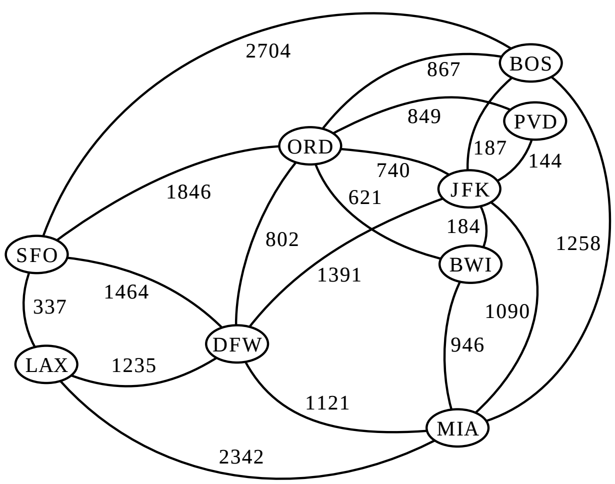
Running Time

T $\leftarrow \{v\}$
  $D[u] \leftarrow 0$
  $E[u] \leftarrow \emptyset$
for each vertex $u \neq v$ **do**
  $D[u] \leftarrow +\infty$
let Q be a priority queue that contains all the
    vertices using the D labels as keys
while $Q \neq \emptyset$ **do**
    $u \leftarrow Q$.removeMinElement()
    add vertex u and edge $(u,E[u])$ to T
    for each vertex z adjacent to u **do**
      if z is in Q
        if weight$(u, z) < D[z]$ **then**
          $D[z] \leftarrow$ weight(u, z)
          $E[z] \leftarrow (u, z)$
          change the key of z in Q to $D[z]$
return tree T

O((n+m) log n)

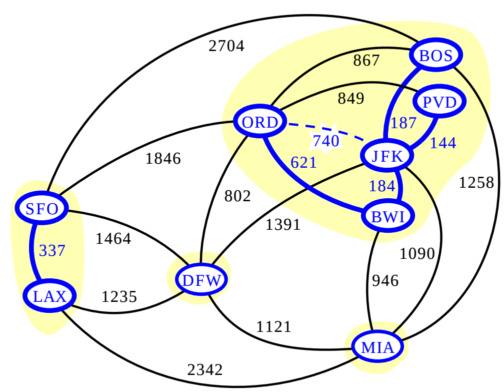
Kruskal Algorithm

- add the edges one at a time, by increasing weight
- accept an edge if it does not create a cycle



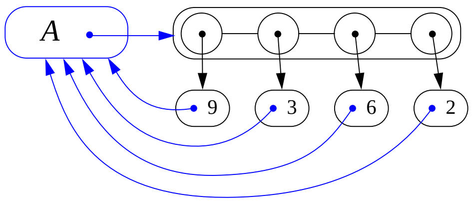
Data Structure for Kruskal Algortihm

- the algorithm maintains a forest of trees
- an edge is accepted it if connects vertices of distinct trees
- we need a data structure that maintains a partition, i.e.,a collection of disjoint sets, with the following operations
  - **find**(u): return the set storing u
  - **union**(u,v): replace the sets storing u and v with their union



Representation of a Partition

- each set is stored in a sequence
- each element has a reference back to the set



- operation **find**(u) takes O(1) time
- in operation **union**(u,v), we move the elements of the smaller set to the sequence of the larger set and update their references
- the time for operation **union**(u,v) is $\min(n_u, n_v)$, where n_u and n_v are the sizes of the sets storing u and v
- whenever an element is processed, it goes into a set of size at least double
- hence, each element is processed at most log n times

Pseudo Code

Algorithm Kruskal(G):
  Input: A weighted graph G.
  Output: A minimum spanning tree T for G.

let P be a partition of the vertices of G, where each vertex forms a separate set

let Q be a priority queue storing the edges of G and their weights

$T \leftarrow \emptyset$
while $Q \neq \emptyset$ **do**
  $(u,v) \leftarrow Q$.removeMinElement()
  if P.**find**(u) $\neq$ P.**find**(u) **then**
    add edge (u,v) to T
    P.**union**(u,v)

return T

Running time: O((n+m) log n)