COP 3530: Computer Science III
Summer 2005

Graphs and Graph Algorithms – Part 5

Instructor : Mark Llewellyn
markl@cs.ucf.edu
CSB 242, 823-2790

http://www.cs.ucf.edu/courses/cop3530/summer05

School of Computer Science
University of Central Florida

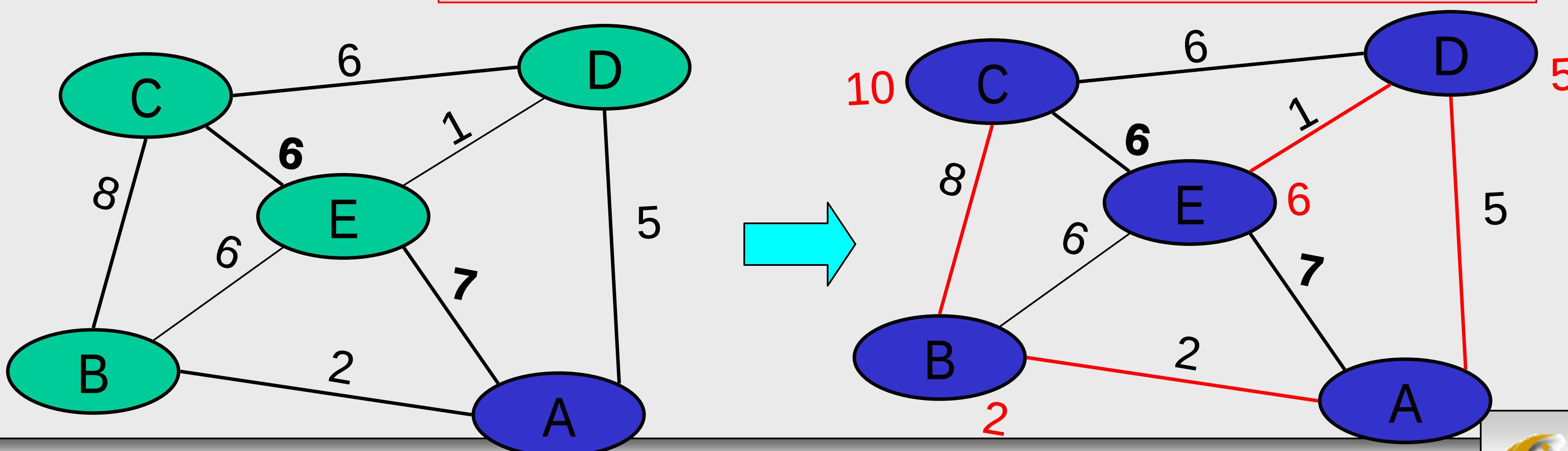
All-Pairs Shortest Paths

- Dijkstra's algorithm was a single source shortest path algorithm.
- Rather than defining one vertex as a starting vertex, suppose that we would like to find the distance between every pair of vertices in a weighted graph G. In other words, the shortest path from every vertex to every other vertex in the graph.
- One option would be to run Dijkstra's algorithm in a loop considering each vertex once as the starting point. A better option would be to use the Floyd-Warshall dynamic programming algorithm (typically referred to as Floyd's algorithm).

- **Dijkstra's Algorithm – Single Source Shortest Path**
  - **Label Path length to each vertex as** ∞ (or 0 for the start vertex)
  - **Loop while there is a vertex**
    - **Remove up the vertex with minimum path length**
    - **Check and if required update the path length of its adjacent neighbors**
  - **end loop**

Update rule:

Let 'a' be the vertex removed and 'b' be its adjacent vertex
Let 'e' be the edge connecting a to b.
if (a.pathLength + e.weight < b.pathLength)
b.pathLength $\leftarrow$ a.PathLength + e.weight
b.parent $\leftarrow$ a



- **DAG based Algorithm - Single Source Shortest Path**

  May have negative weight

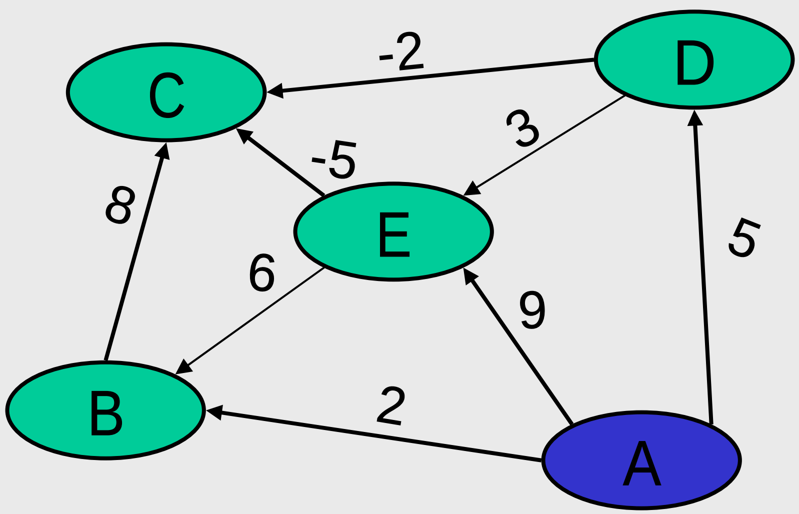
  – **Label Path length to each vertex as** ∞ (or 0 for the start vertex)

  – **Compute Topological Ordering**

  – **loop for i ← 1 to n-1**

    - **Check and if required update the path length of adjacent neighbors of v_i**

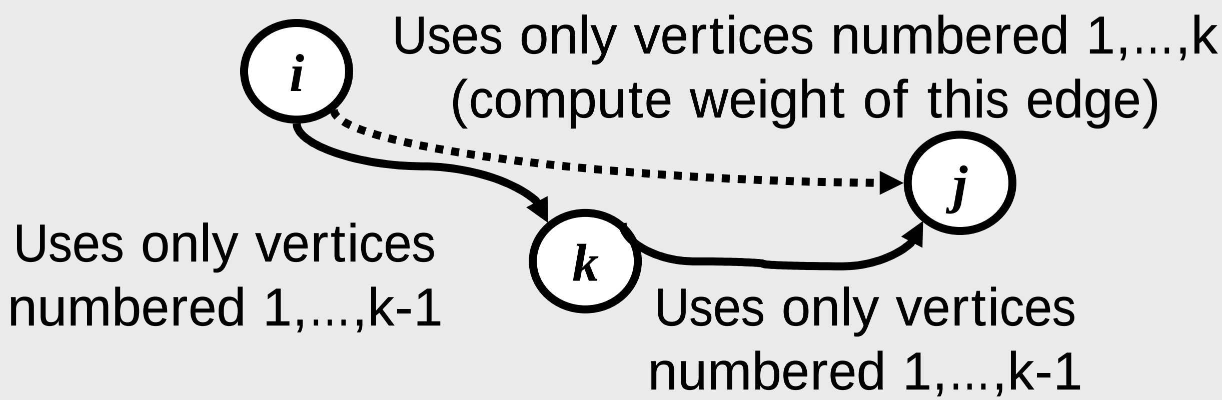
  – **end loop**



Path Length from A

To		*i*=1	*i*=2	*i*=3	*i*=4
B	∞				
C	∞				
D	∞				
E	∞				

All-Pairs Shortest Paths



$$D^0_{i,j} = \begin{cases} 0 & \text{if } i = j \\ weight\left(edge(v_i, v_j)\right) & \text{if } (v_i, v_j) \text{ is an edge} \\ \infty & \text{otherwise} \end{cases}$$

$$D^k_{i,j} = \min\left\{D^{k-1}_{i,j}, \quad D^{k-1}_{i,k} + D^{k-1}_{k,j}\right\}$$

All-Pairs Shortest Paths

Dynamic Programming approach

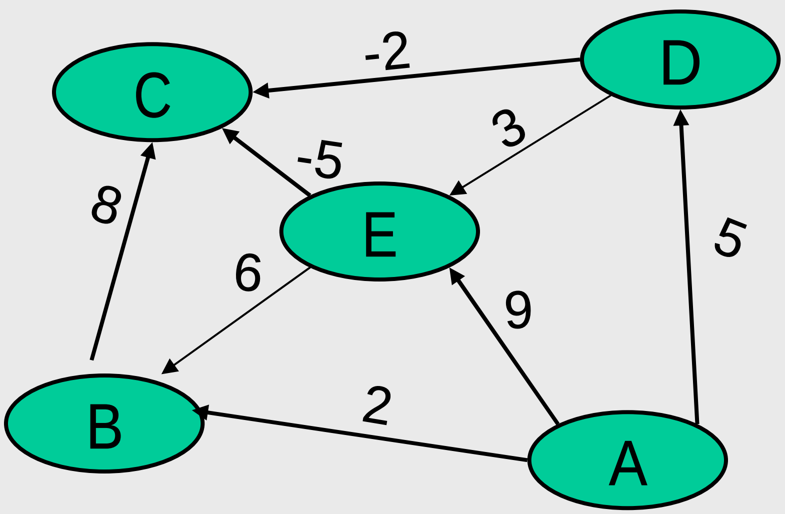
$$\boxed{d_{i,j}^{k} = \min\left\{ d_{i,j}^{k-1}, \quad d_{i,k}^{k-1} + d_{k,j}^{k-1} \right\}}$$

- **Initialize a n×n table with 0 or ∞ or edge length.**
- **Check and Update the table bottom up in nested loops**

for *k*,

for *i*,

for *j*



i↓ d j→	v_1	v_2	v_3	v_4	v_5
v_1					
v_2					
v_3					
v_4					
v_5					

Minimum Spanning Tree

Spanning subgraph

- Subgraph of a graph ***G*** containing all the vertices of ***G***

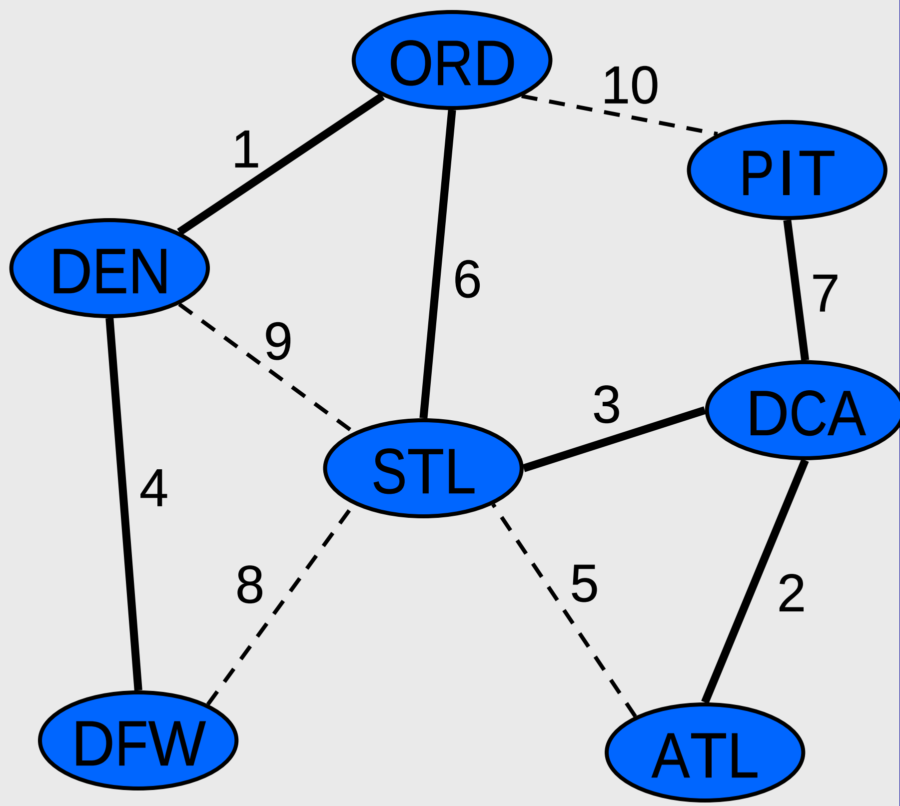
Spanning tree

- Spanning subgraph that is itself a (free) tree

Minimum spanning tree (MST)

- Spanning tree of a weighted graph with minimum total edge weight

- Applications
  - Communications networks
  - Transportation networks



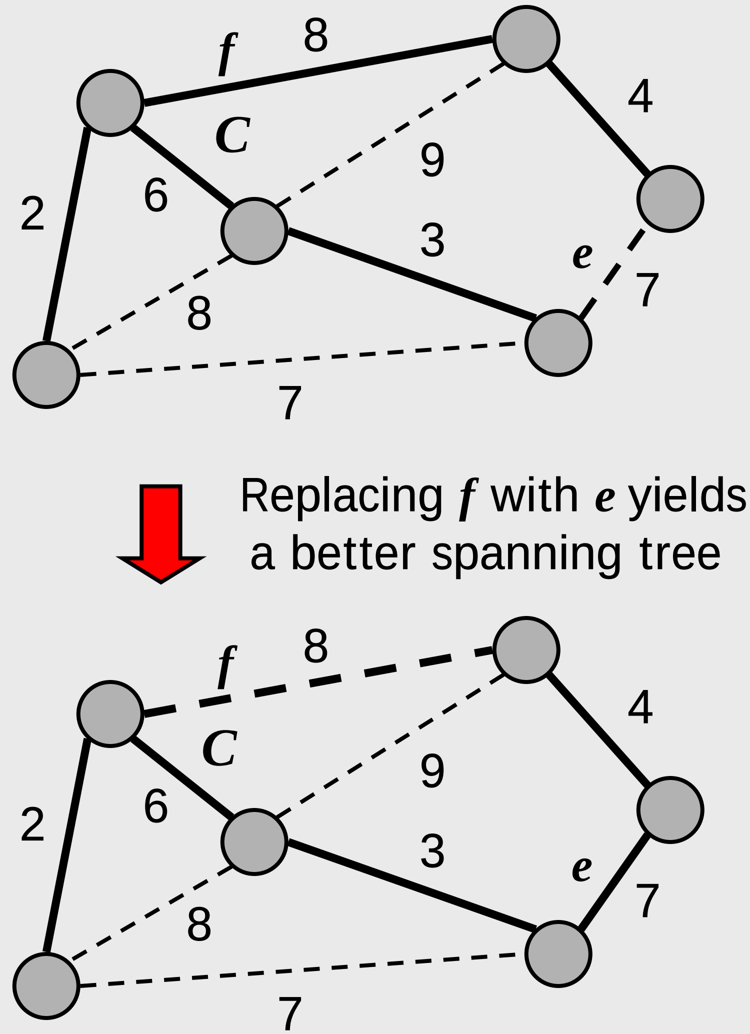
Cycle Property

Cycle Property:

- Let ***T*** be a minimum spanning tree of a weighted graph ***G***
- Let ***e*** be an edge of ***G*** that is not in ***T***
- Let ***C*** let be the cycle formed by ***e*** with ***T***
- For every edge ***f*** of ***C***, ***weight***(***f***) ≤ ***weight***(***e***)

Proof:

- By contradiction
- If ***weight***(***f***) > ***weight***(***e***) we can get a spanning tree of smaller weight by replacing ***e*** with ***f***



Partition Property

Partition Property:

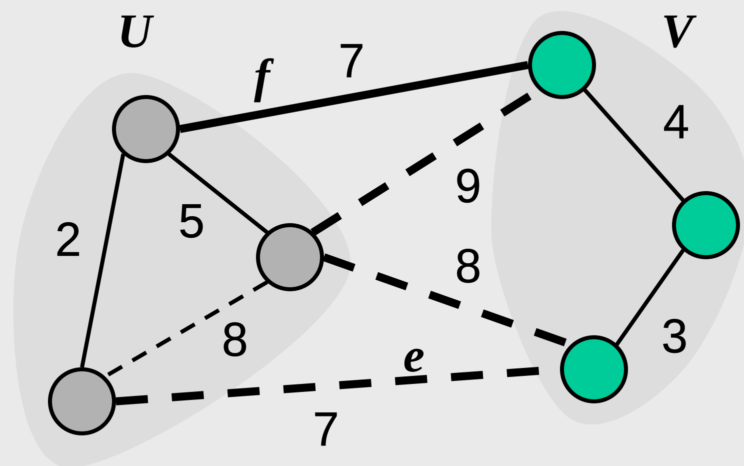
- Consider a partition of the vertices of G into subsets U and V
- Let e be an edge of minimum weight across the partition
- There is a minimum spanning tree of G containing edge e

Proof:

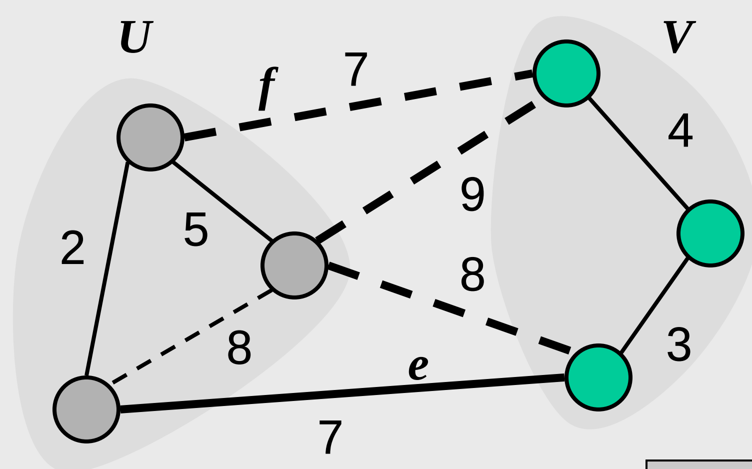
- Let T be an MST of G
- If T does not contain e, consider the cycle C formed by e with T and let f be an edge of C across the partition
- By the cycle property,

  $$\boldsymbol{weight}(f) \leq \boldsymbol{weight}(e)$$

- Thus, $\boldsymbol{weight}(f) = \boldsymbol{weight}(e)$
- We obtain another MST by replacing f with e

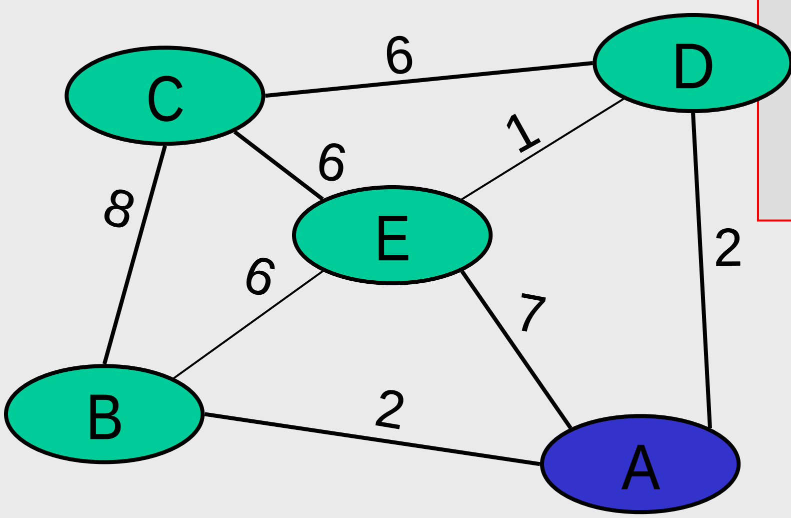


Replacing f with e yields another MST



Minimum Spanning Tree

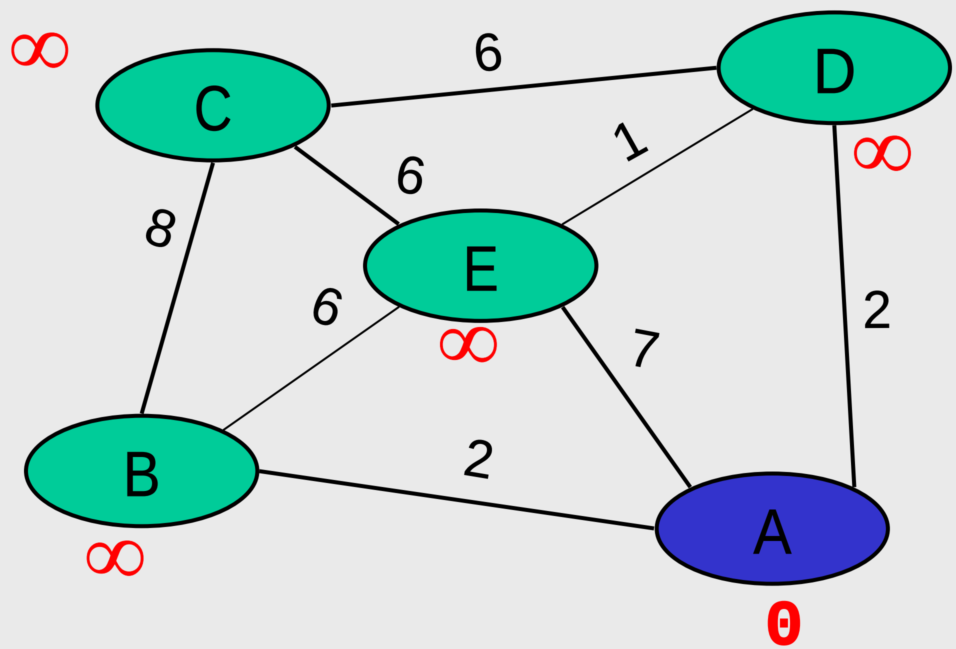
- **Prim's Algorithm (Prim-Jarnik Algorithm)**
  - **Label cost of each vertex as** ∞ (or 0 for the start vertex)
  - **Loop while there is a vertex**
    - **Remove a vertex that will extend the tree with minimum additional cost**
    - **Check and if required update the path length of its adjacent neighbors (Update rule different from Dijkstra's algorithm)**
  - **end loop**



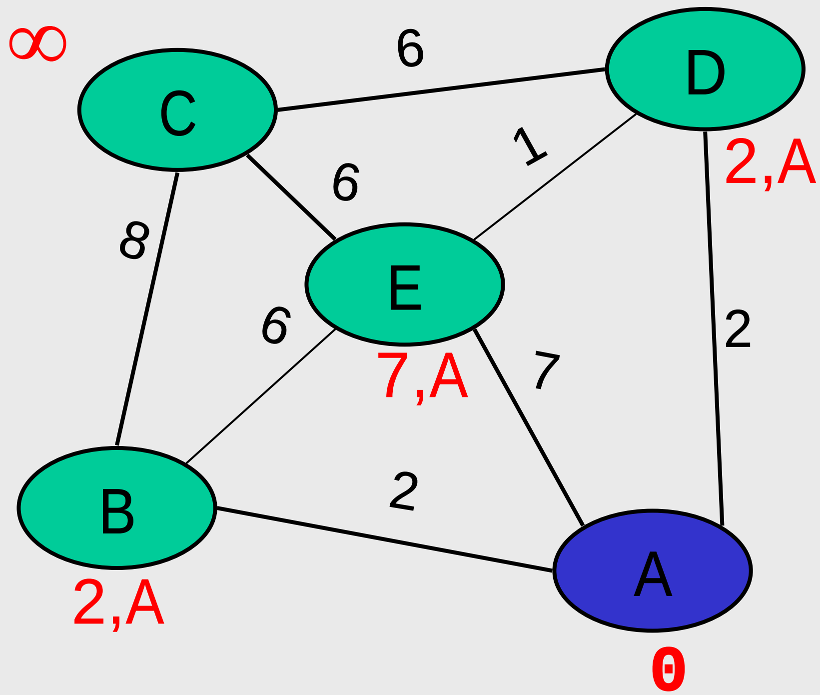
Update rule:

Let 'a' be the vertex removed and 'b' be its adjacent vertex
Let 'e' be the edge connecting a to b.
if (e.weight < b.cost)
    b.cost $\leftarrow$ e.weight
    b.parent $\leftarrow$ a

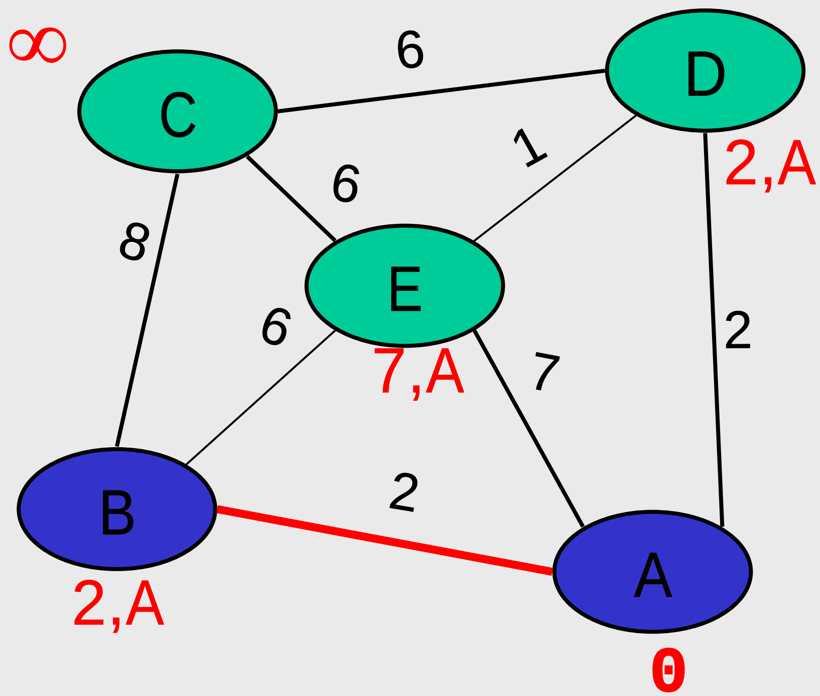
MST: Prim-Jarnik's Algorithm



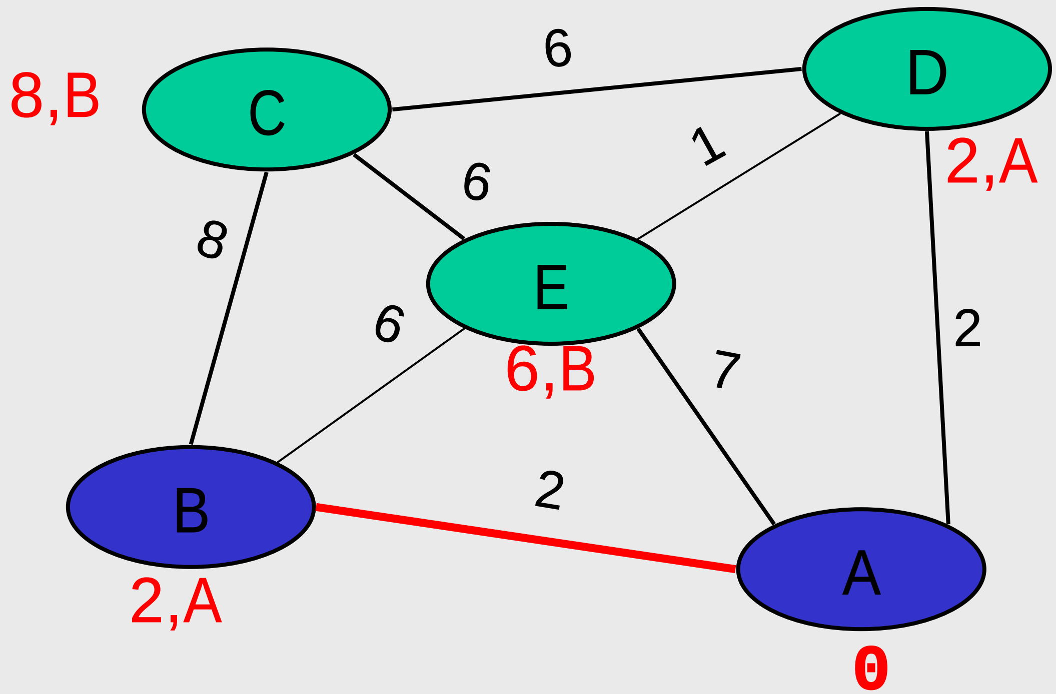
MST: Prim-Jarnik's Algorithm



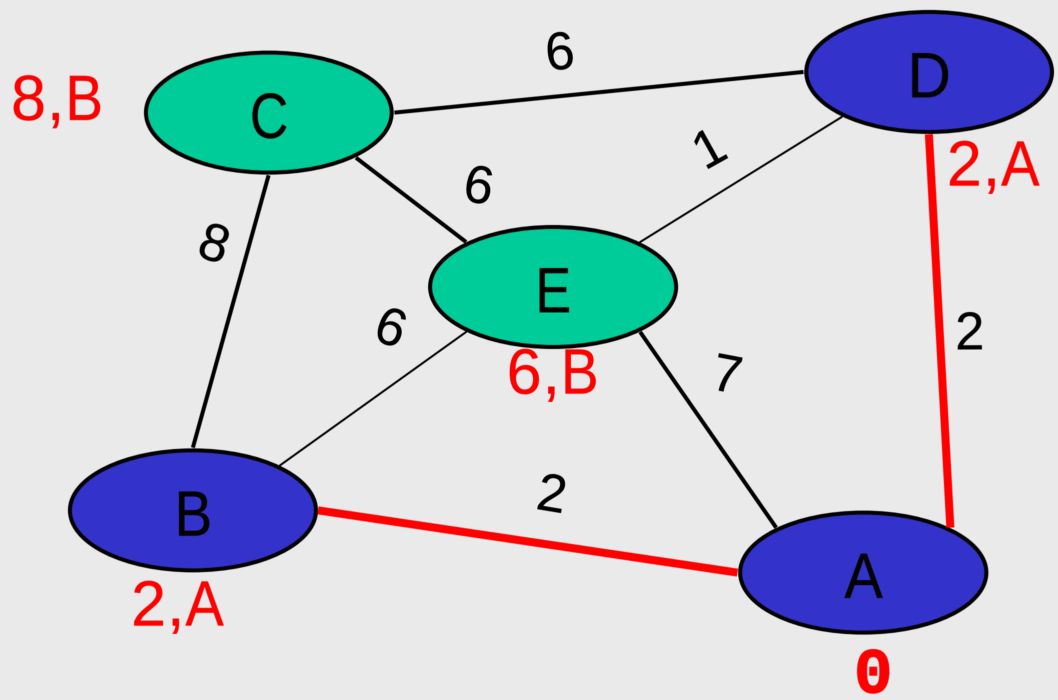
MST: Prim-Jarnik's Algorithm



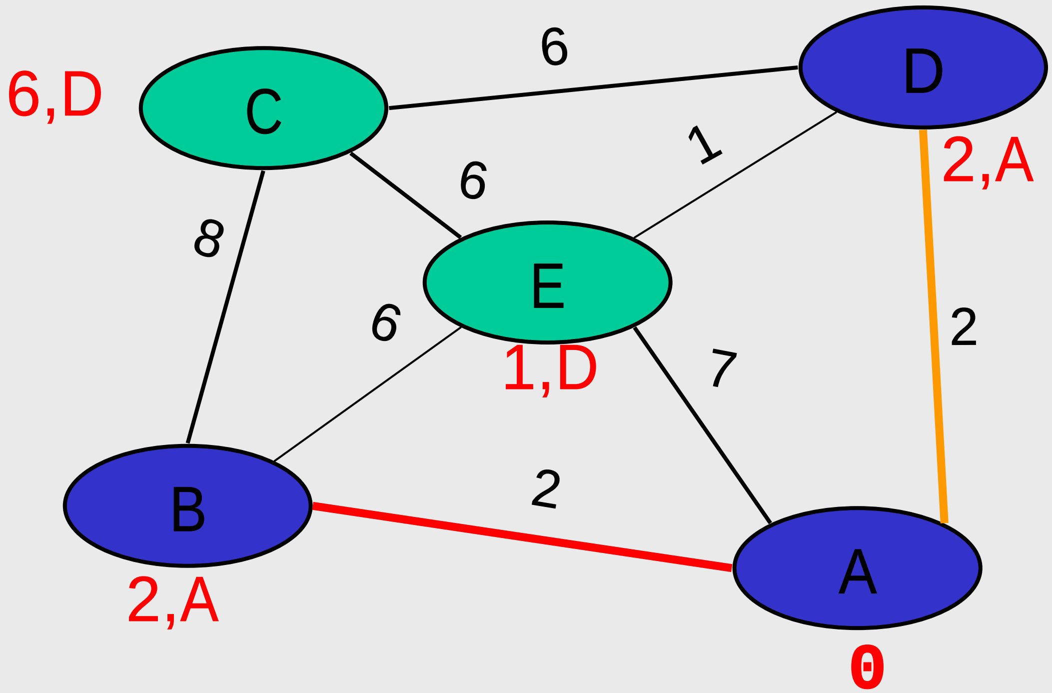
MST: Prim-Jarnik's Algorithm



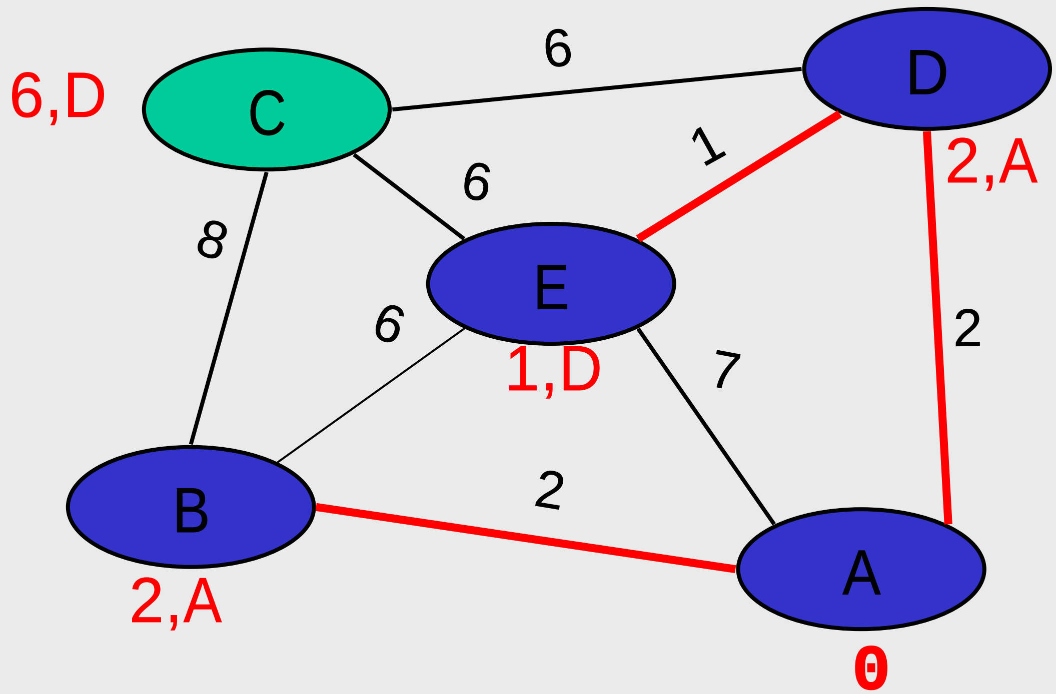
MST: Prim-Jarnik's Algorithm



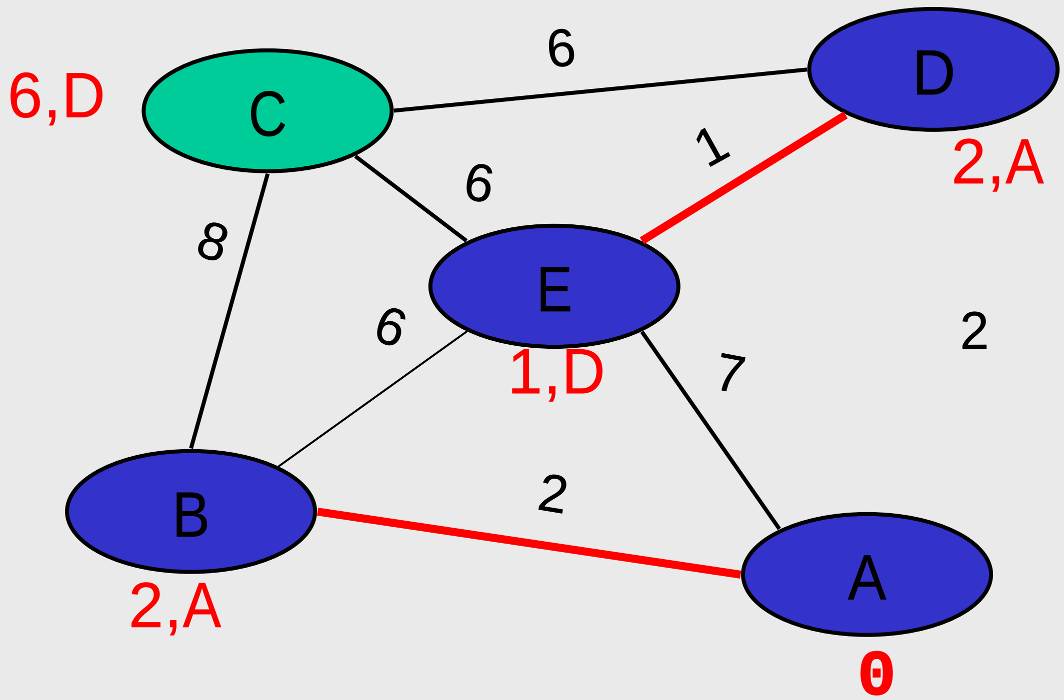
MST: Prim-Jarnik's Algorithm



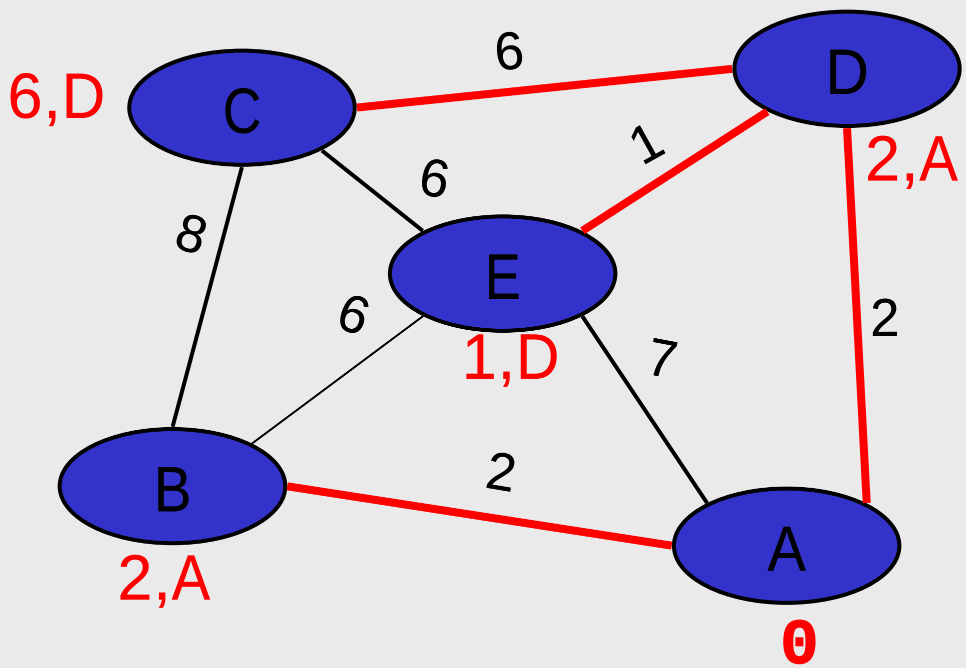
MST: Prim-Jarnik's Algorithm



MST: Prim-Jarnik's Algorithm



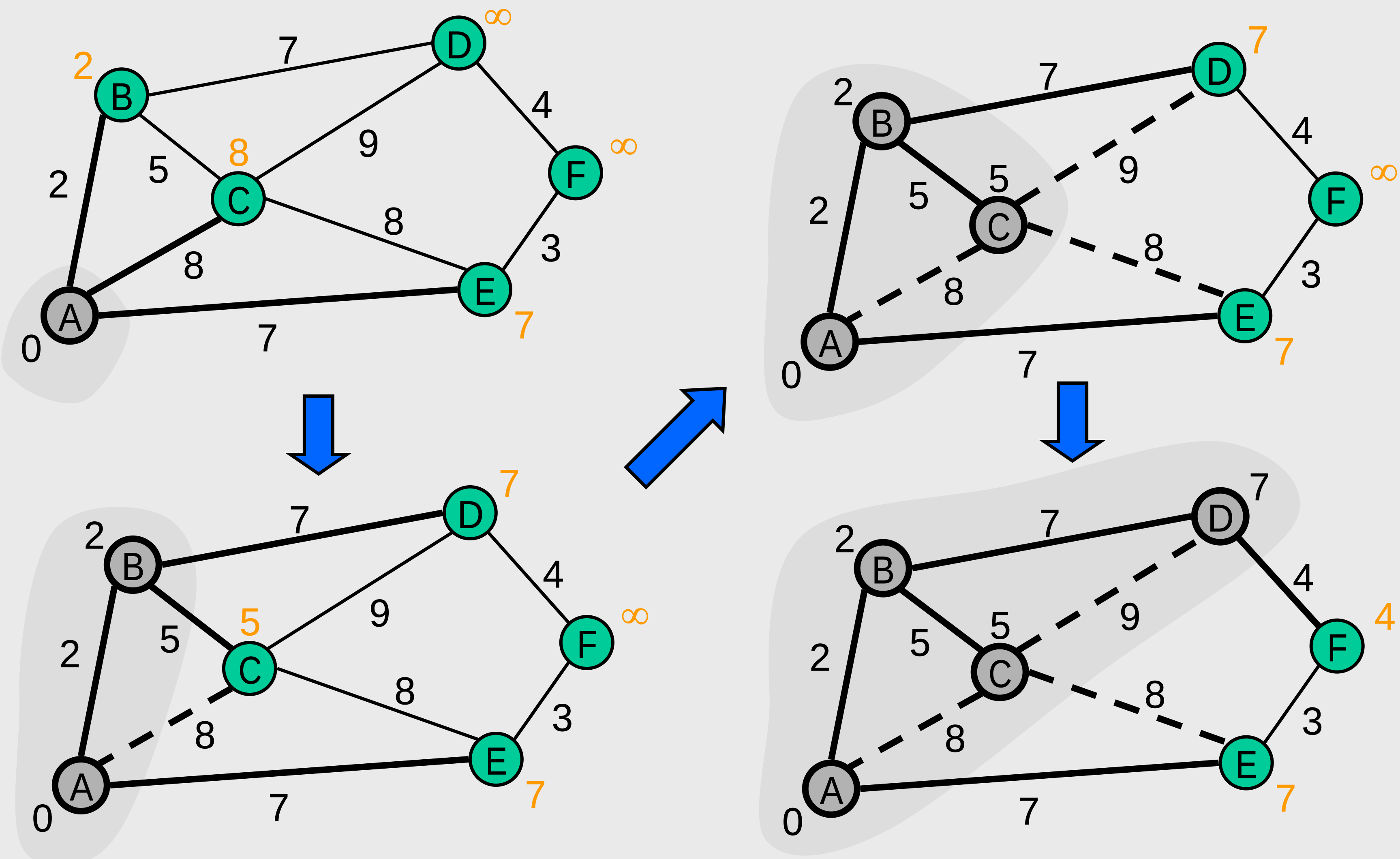
MST: Prim-Jarnik's Algorithm



Prim-Jarnik Algorithm

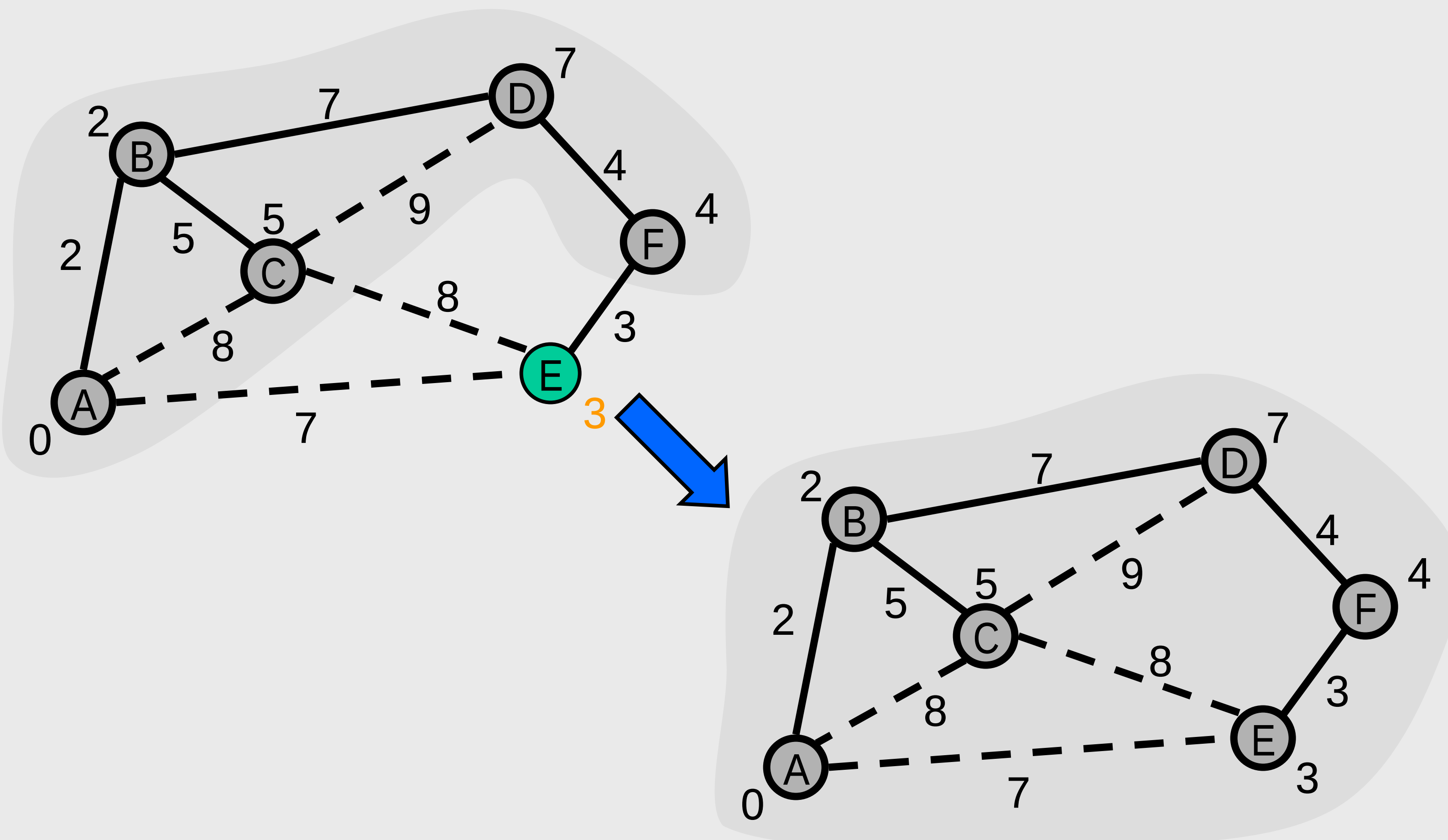
```
Algorithm MST (G)
   Q ← new priority queue
   Let s be any vertex
   for all  v ∈ G.vertices()                 O(n)
      if v = s
         v.cost ← 0
      else v.cost ← ∞
         v. parent ← null
      Q.enQueue(v.cost, v)                    O(n log n)
   while  ¬Q.isEmpty()
      v ← Q .removeMin()
       v.pathKnown ← true
      for all  e ∈ G.incidentEdges(v)         O((n+m) log n)
            w ← opposite(v,e)
            if ¬w.pathKnown
               if  weight(e) < w.cost
                  w.cost ← weight(e)
                  w. parent ← v
                  update key of w in Q
```

Example



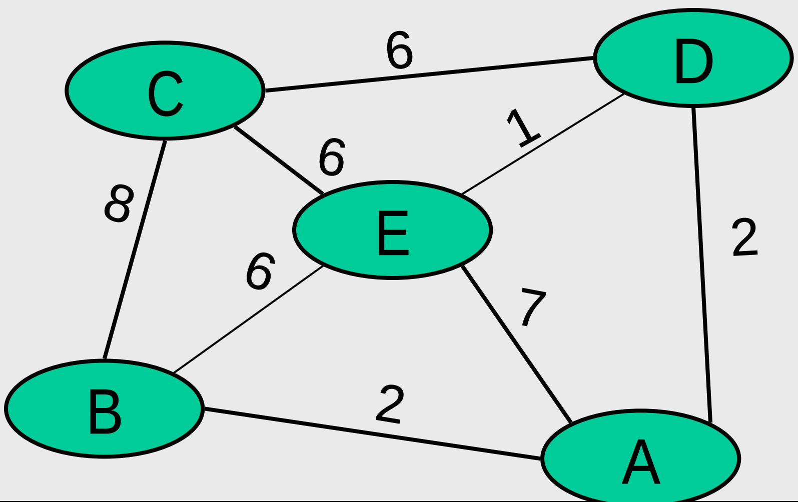
A B C D E F
0 2 8 ∞ 7 ∞
7 4 9 5 2 8 8 3 7
A B C D E F
0 2 5 7 7 ∞
7 4 9 5 2 8 8 3 7
A B C D E F
0 2 5 7 7 ∞
7 4 9 5 2 8 8 3 7
A B C D E F
0 2 5 7 7 4
7 4 9 5 2 8 8 3 7

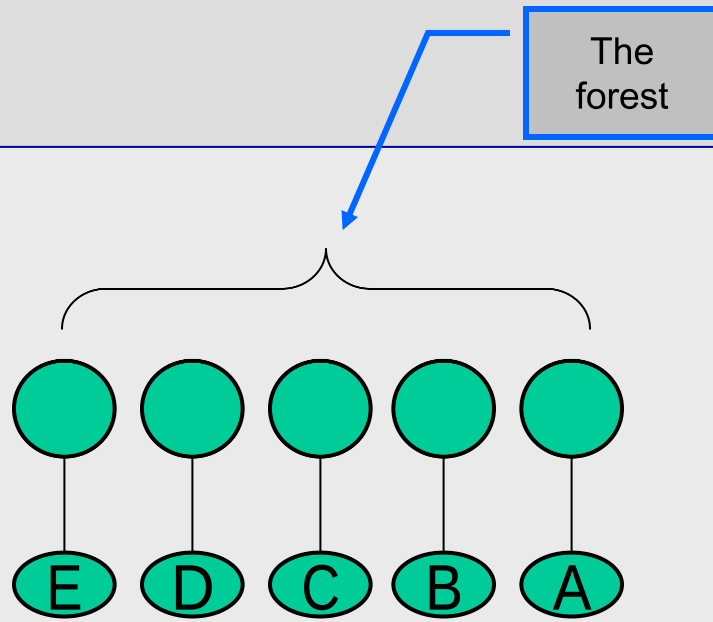
Example (cont.)



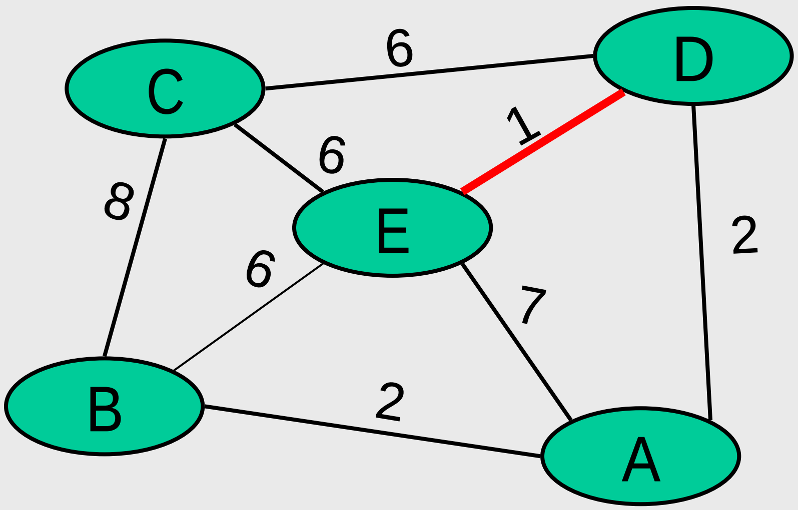
Minimum Spanning Tree

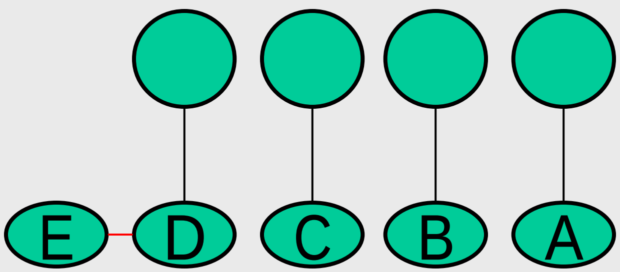
- **Kruskal's Algorithm**
  - **Create a forest of n trees**
  - **Loop while (there is > 1 tree in the forest)**
    - **Remove an edge with minimum weight**
    - **Accept the edge only if it connects 2 trees from the forest in to one.**
  - **end loop**



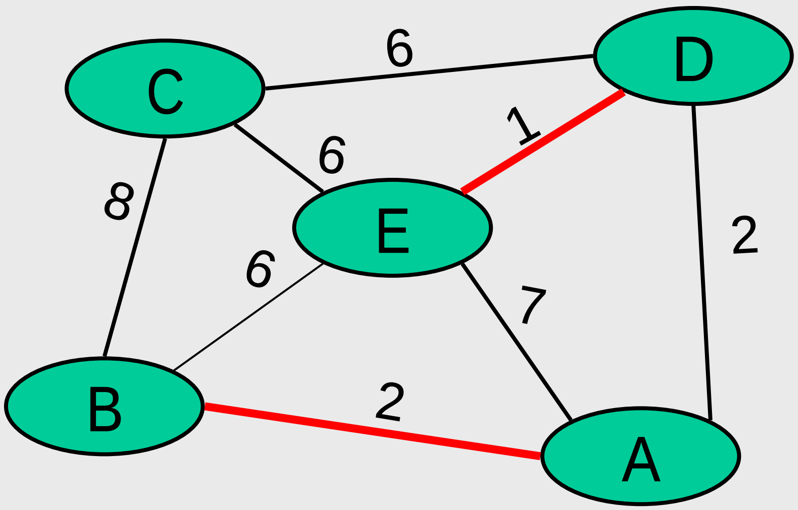


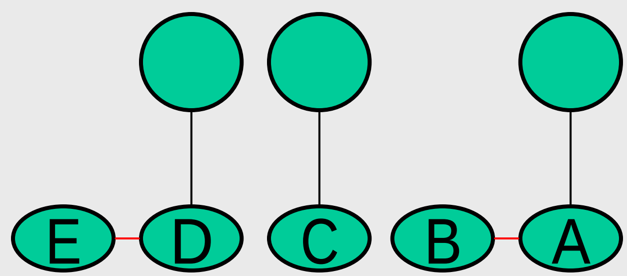
MST: Kruskal's Algorithm



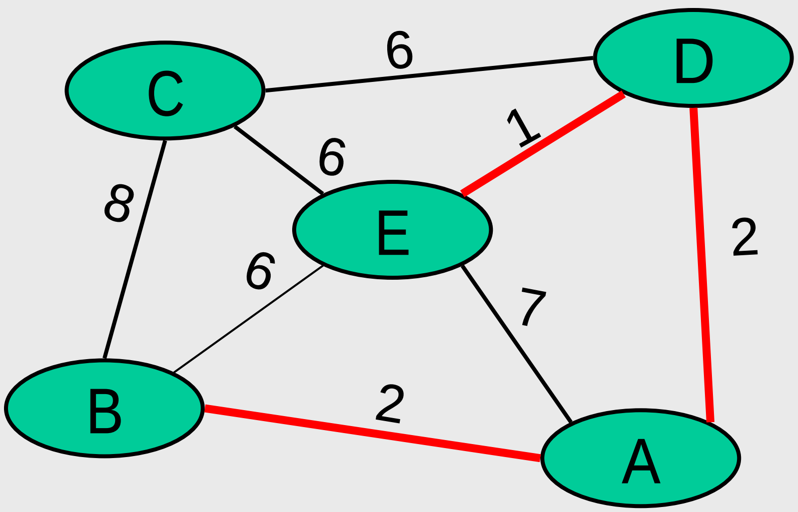


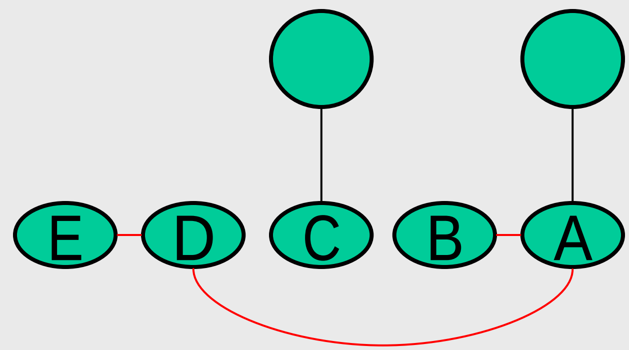
MST: Kruskal's Algorithm



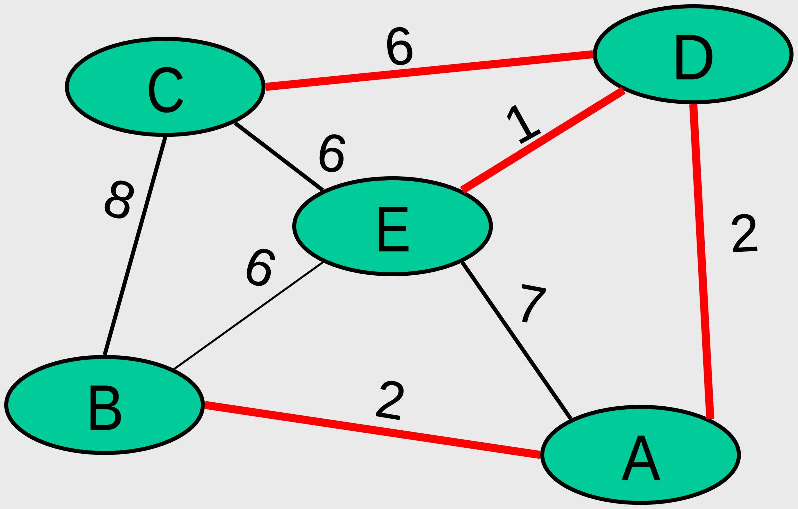


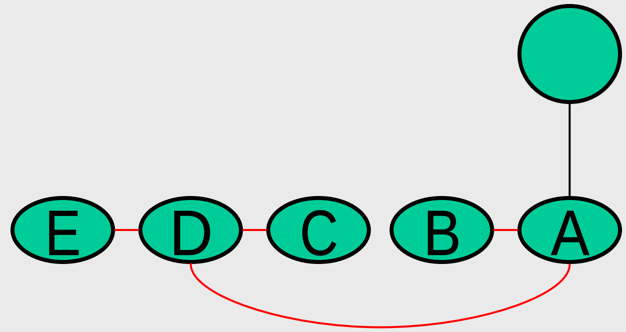
MST: Kruskal's Algorithm

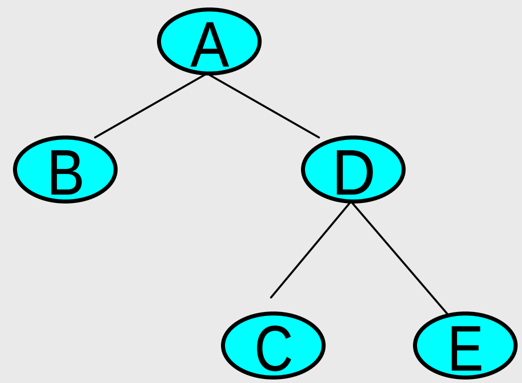




MST: Kruskal's Algorithm







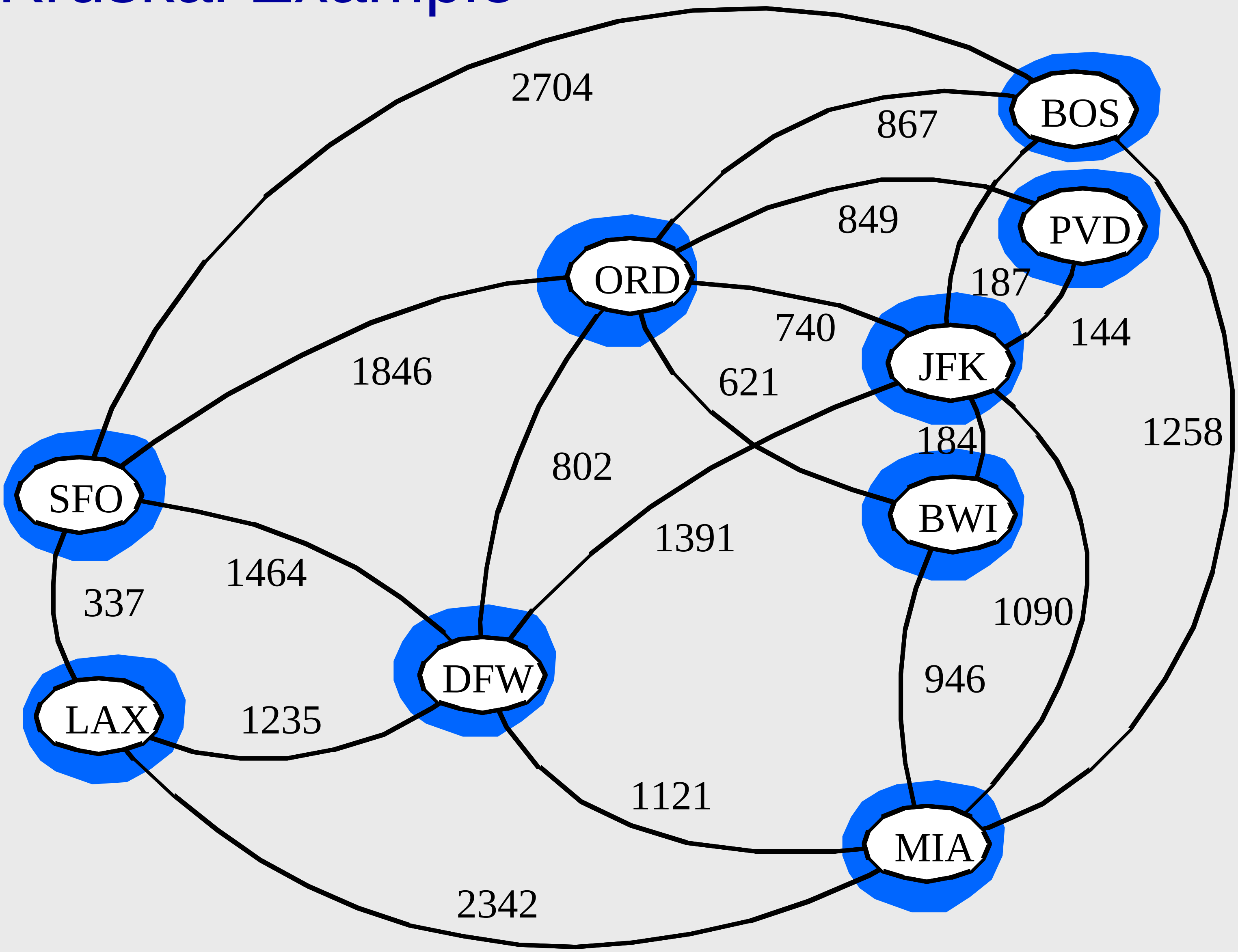
Kruskal's Algorithm

Algorithm ***KruskalMST*(*G*)**
  let ***Q*** be a priority queue.
  Insert all edges into ***Q*** using their
    weights as the key
  Create a forest of n trees
    where each vertex is a tree
  numberOfTrees ← n
  while numberOfTrees > 1**do**
    edge ***e*** ← ***Q.removeMin()***
    Let ***u*, *v*** be the endpoints of *e*
    if ***Tree(v)*** ≠ ***Tree(u)*** **then**
      Combine ***Tree(v)*** and ***Tree(u)***
        using edge ***e***
      decrement numberOfTrees
  return ***T***

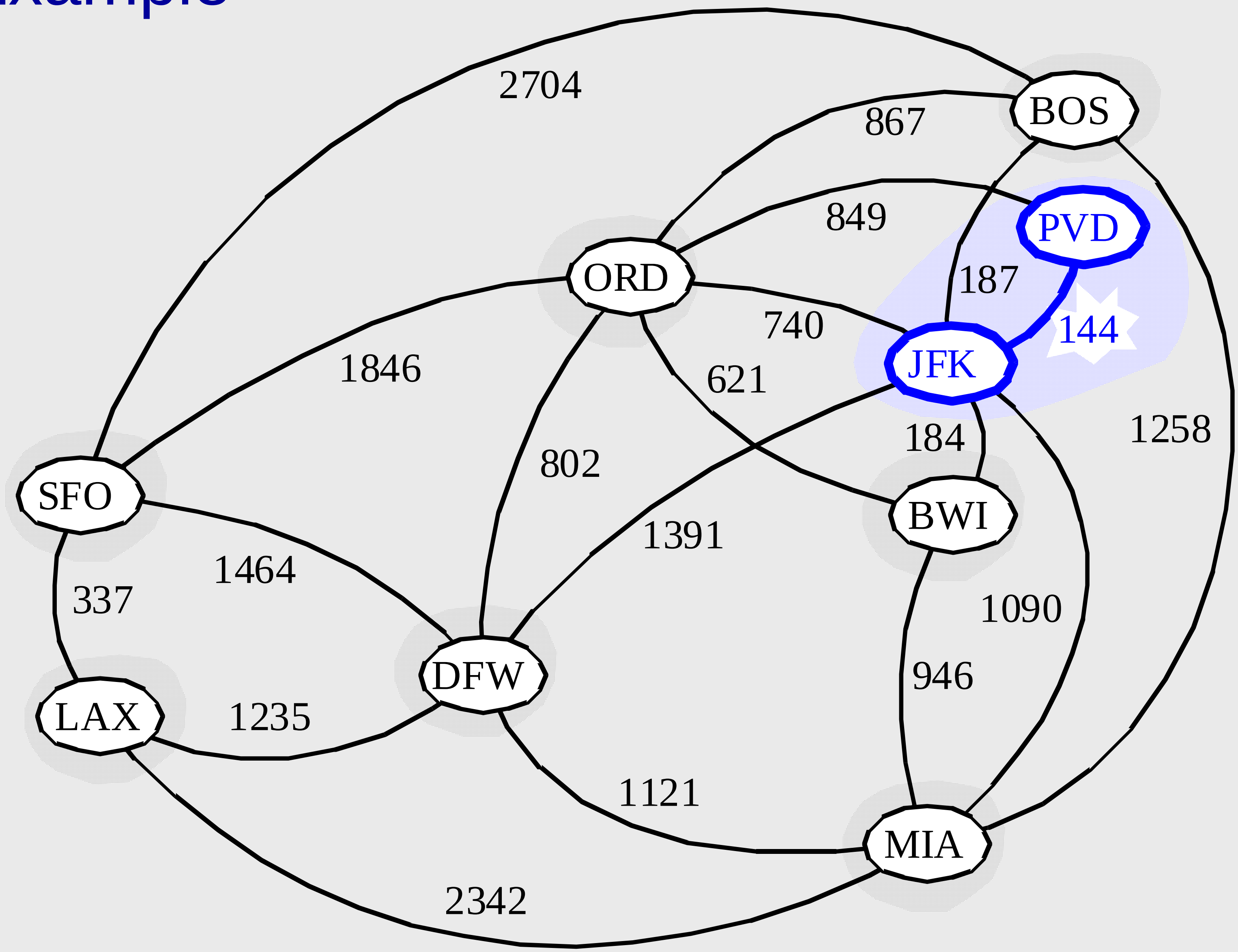
O(m log m)

O(m log m)

Kruskal Example

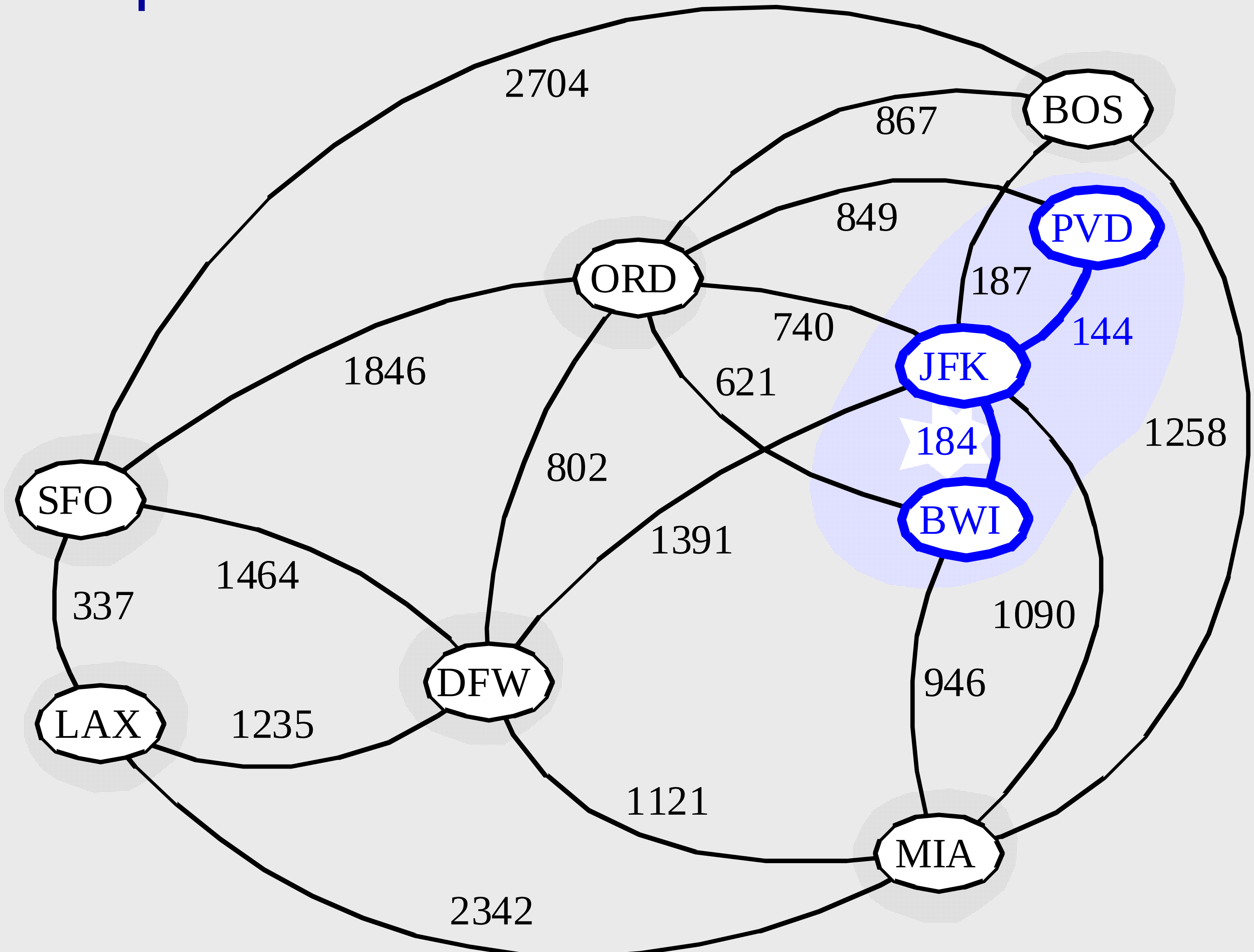


Example



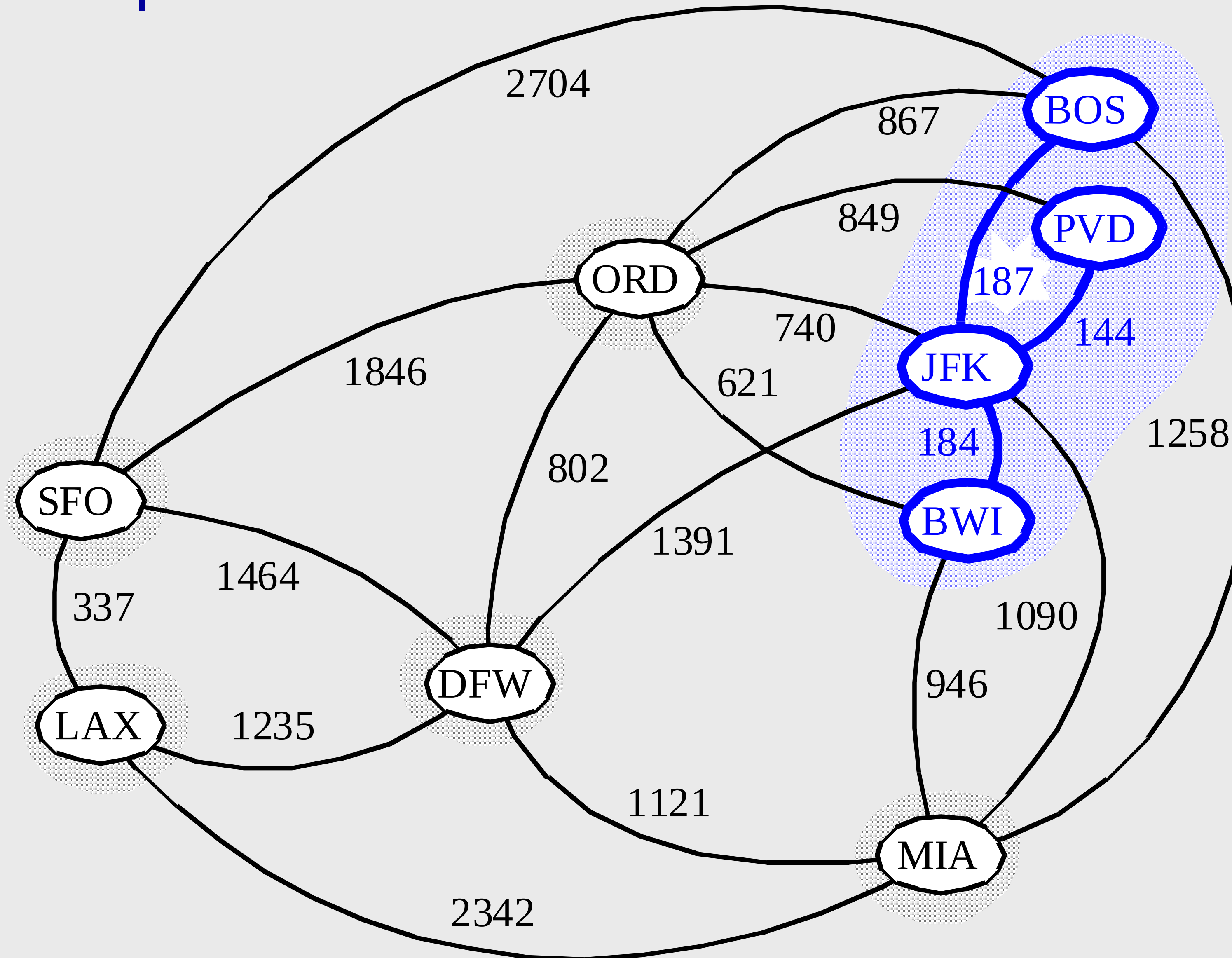
BOS
PVD
ORD
JFK
SFO
BWI
DFW
LAX
MIA
2704
867
849
187
740
144
1846
621
1258
184
802
1391
1464
337
1090
946
1235
1121
2342

Example



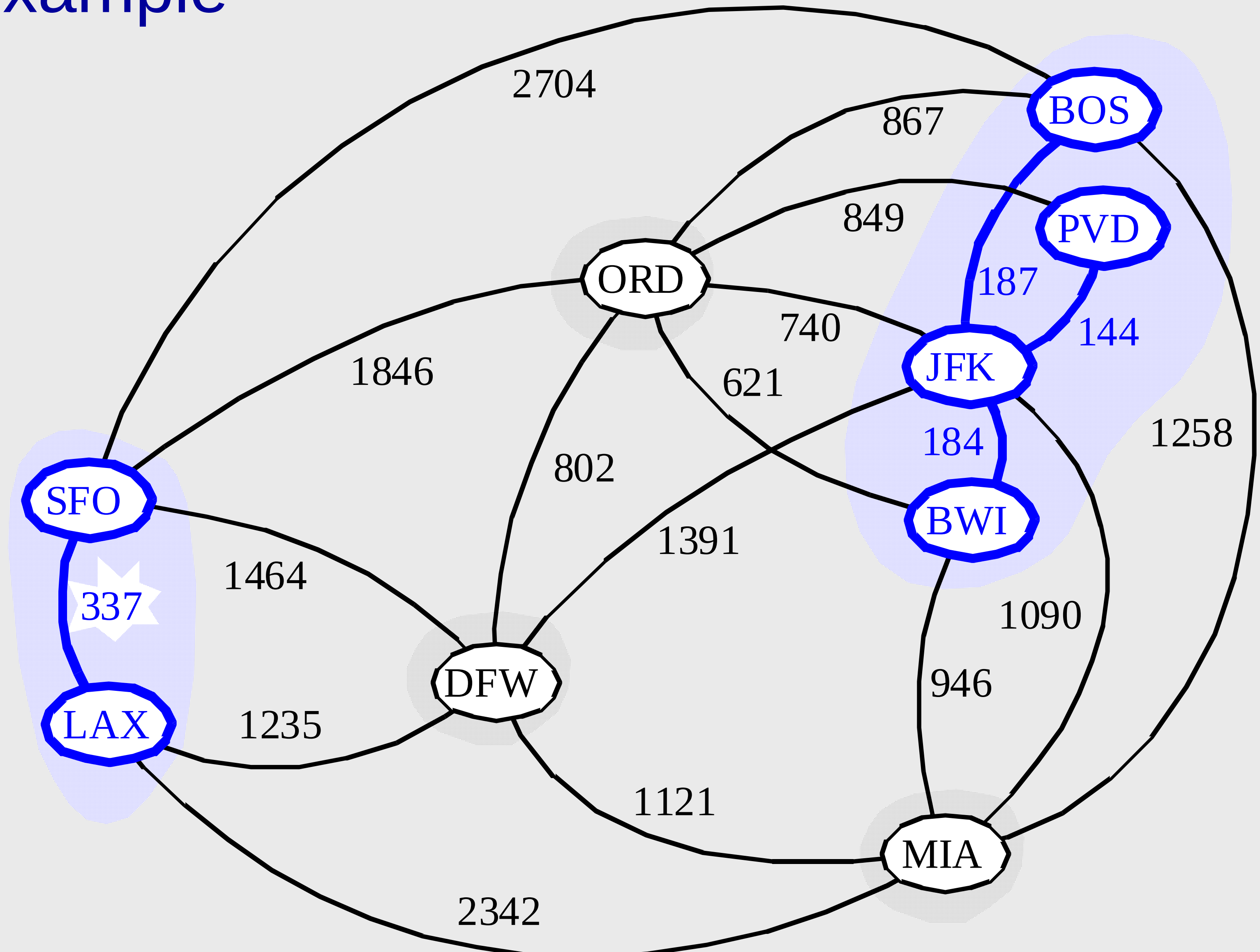
BOS
PVD
ORD
JFK
SFO
BWI
DFW
LAX
MIA
2704
867
849
187
144
740
621
1846
184
1258
802
1391
1464
337
1090
946
1235
1121
2342

Example

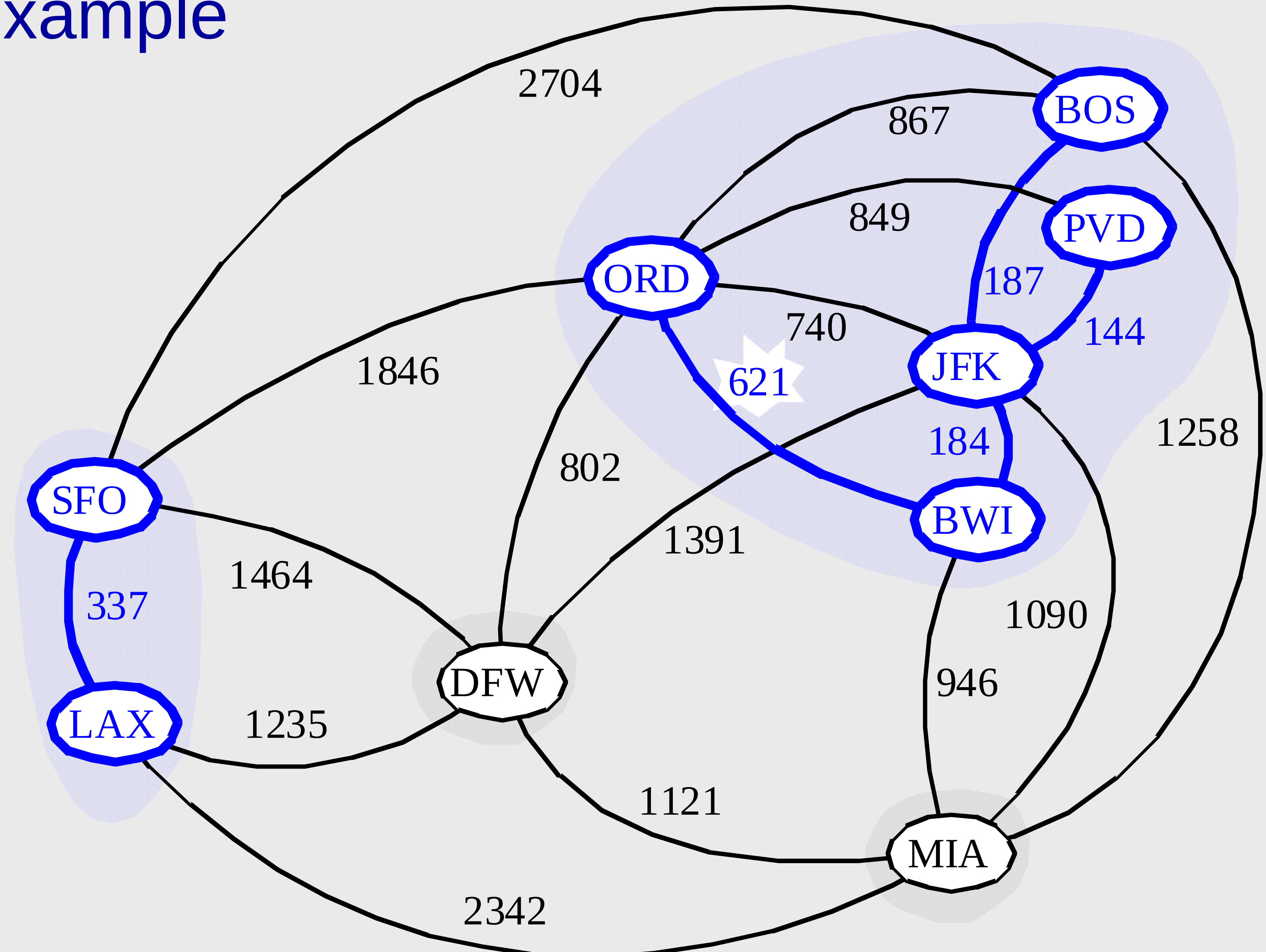


2704
867
BOS
849
PVD
ORD
187
144
740
JFK
1846
621
1258
184
802
SFO
BWI
1391
1464
337
1090
DFW
946
LAX
1235
1121
MIA
2342

Example

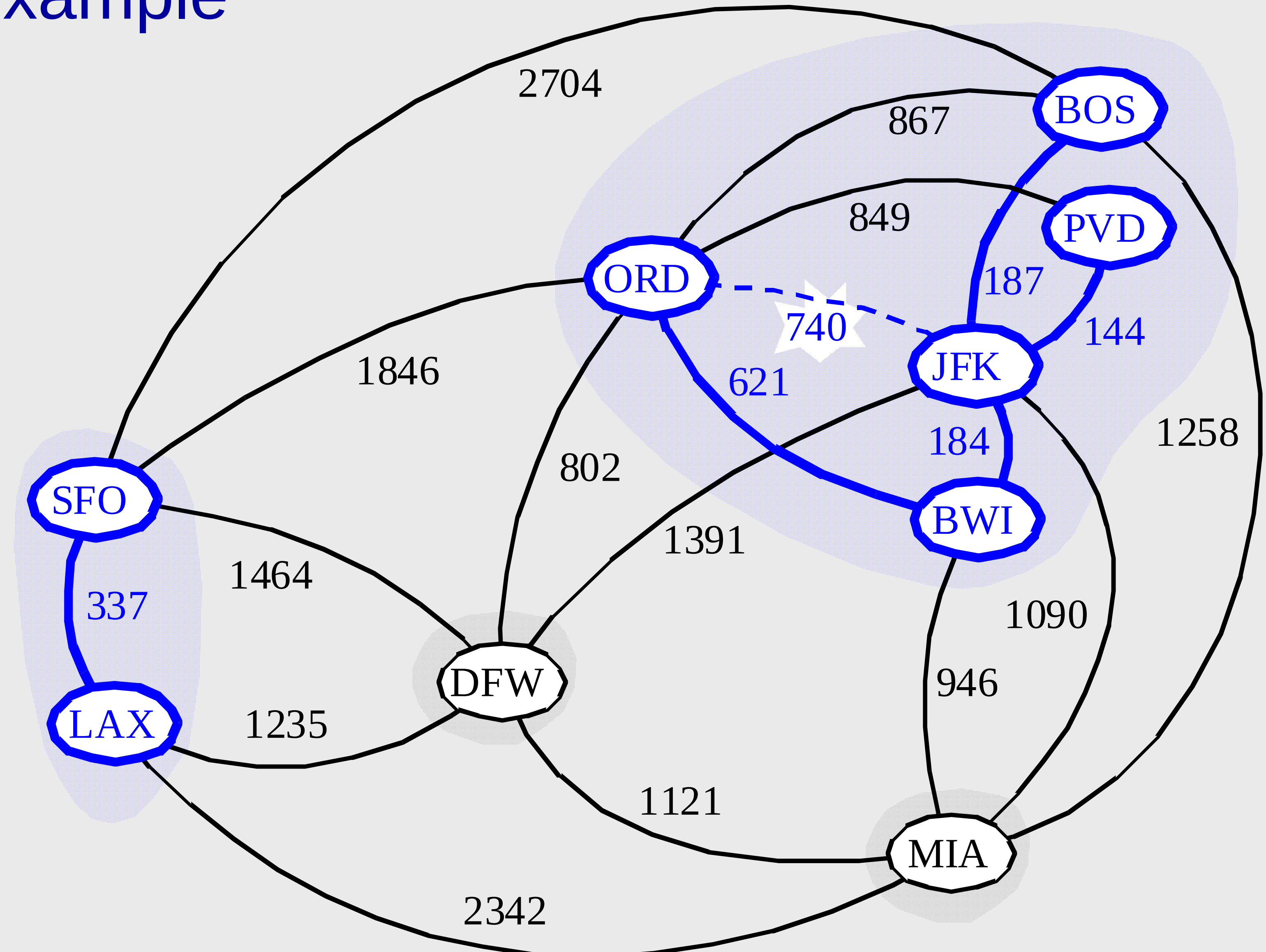

2704
BOS
867
849
PVD
ORD
187
740
144
JFK
1846
621
1258
184
802
SFO
BWI
1391
1464
337
1090
DFW
946
LAX
1235
1121
MIA
2342

Example

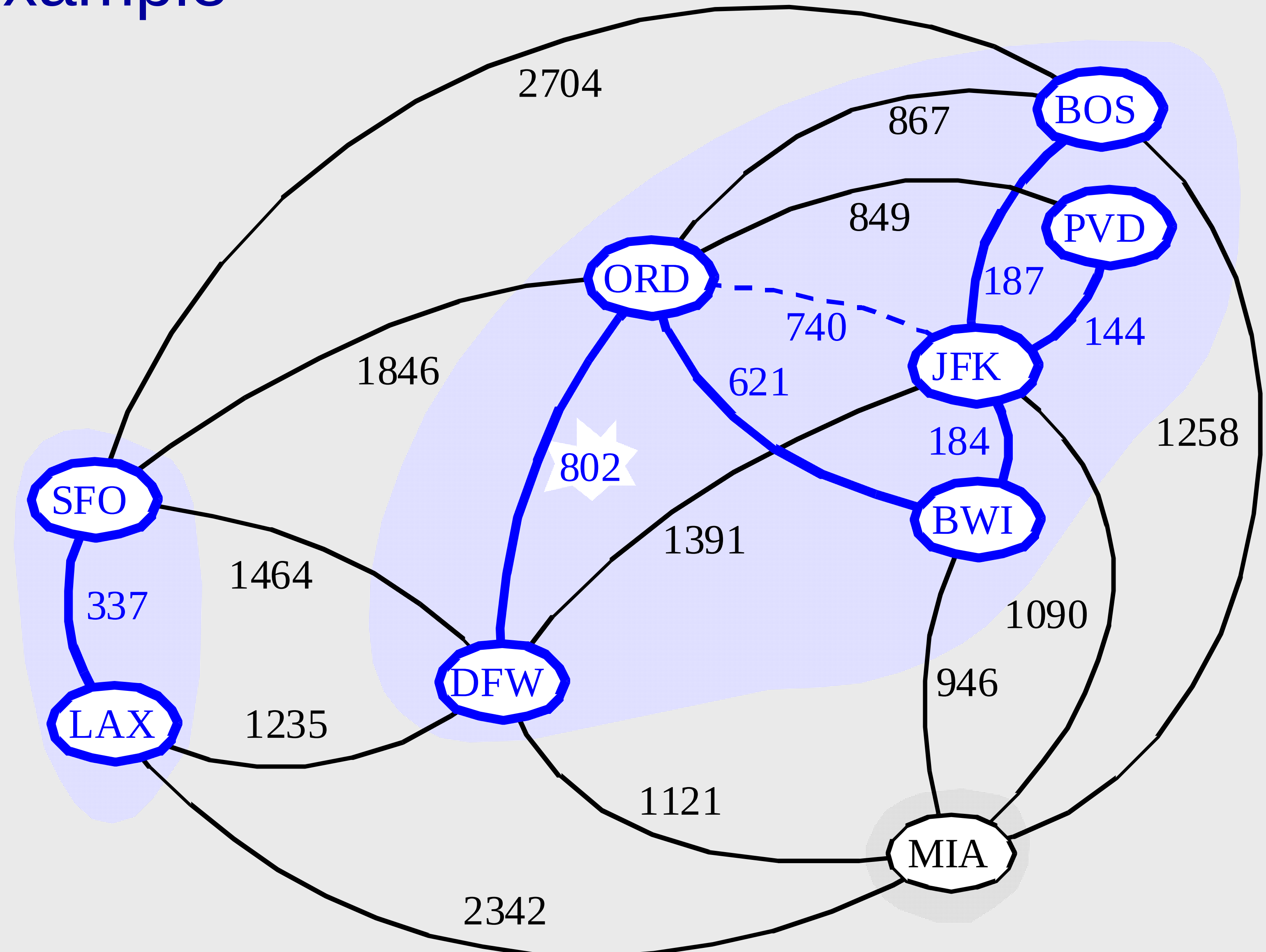


2704
BOS
867
849
PVD
ORD
187
144
740
621
JFK
1846
184
1258
802
SFO
BWI
1391
1464
337
1090
DFW
946
LAX
1235
1121
MIA
2342

Example

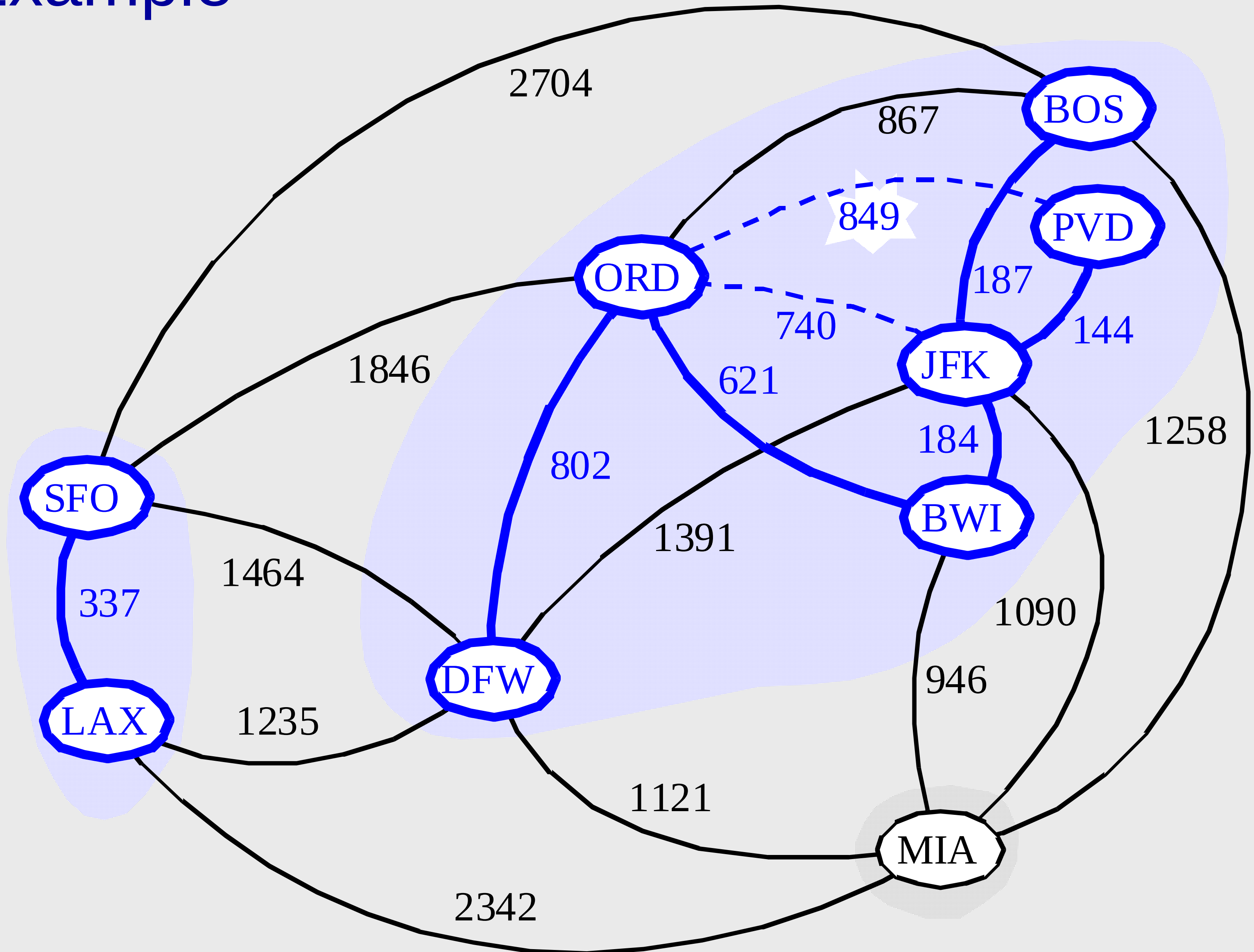

2704
867
BOS
849
PVD
ORD
187
740
144
JFK
1846
621
1258
184
SFO
802
BWI
1391
1464
337
1090
DFW
946
LAX
1235
1121
MIA
2342

Example



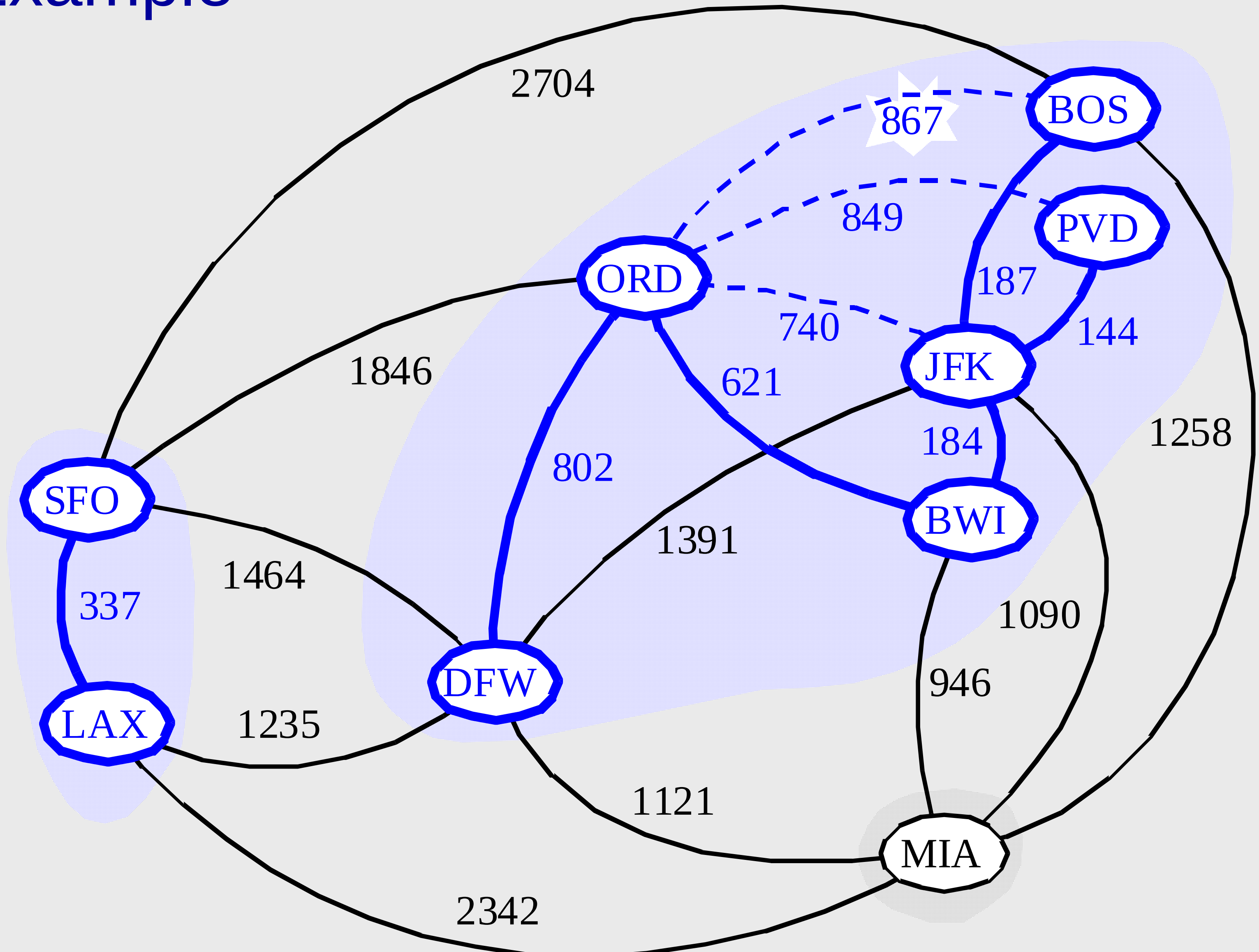
2704
867
BOS
849
PVD
ORD
187
740
144
1846
JFK
621
1258
184
802
SFO
BWI
1391
1464
337
1090
DFW
946
LAX
1235
1121
MIA
2342

Example



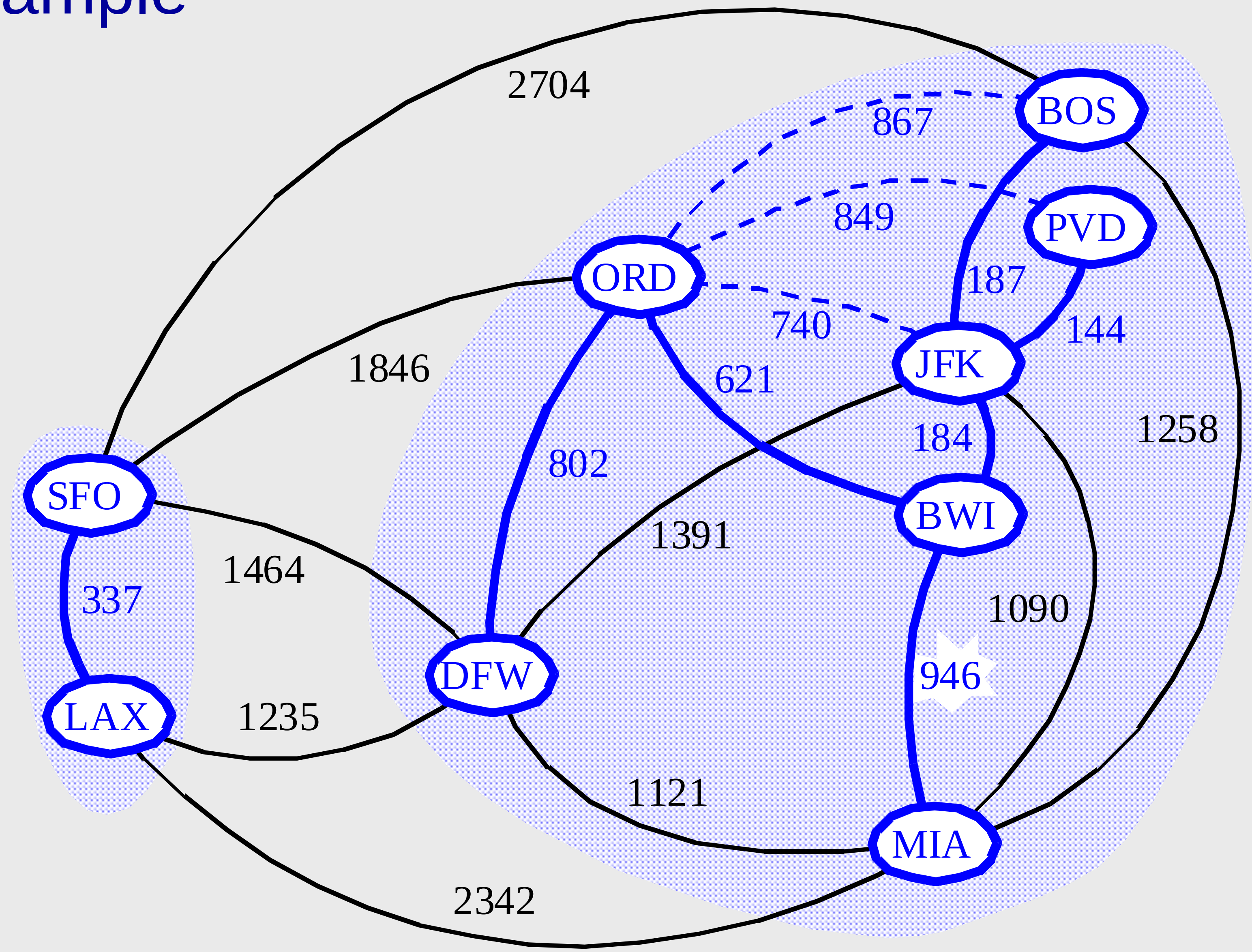
2704
BOS
867
849
PVD
ORD
187
740
144
1846
621
JFK
1258
184
SFO
802
BWI
1391
1464
337
1090
DFW
946
LAX
1235
1121
MIA
2342

Example



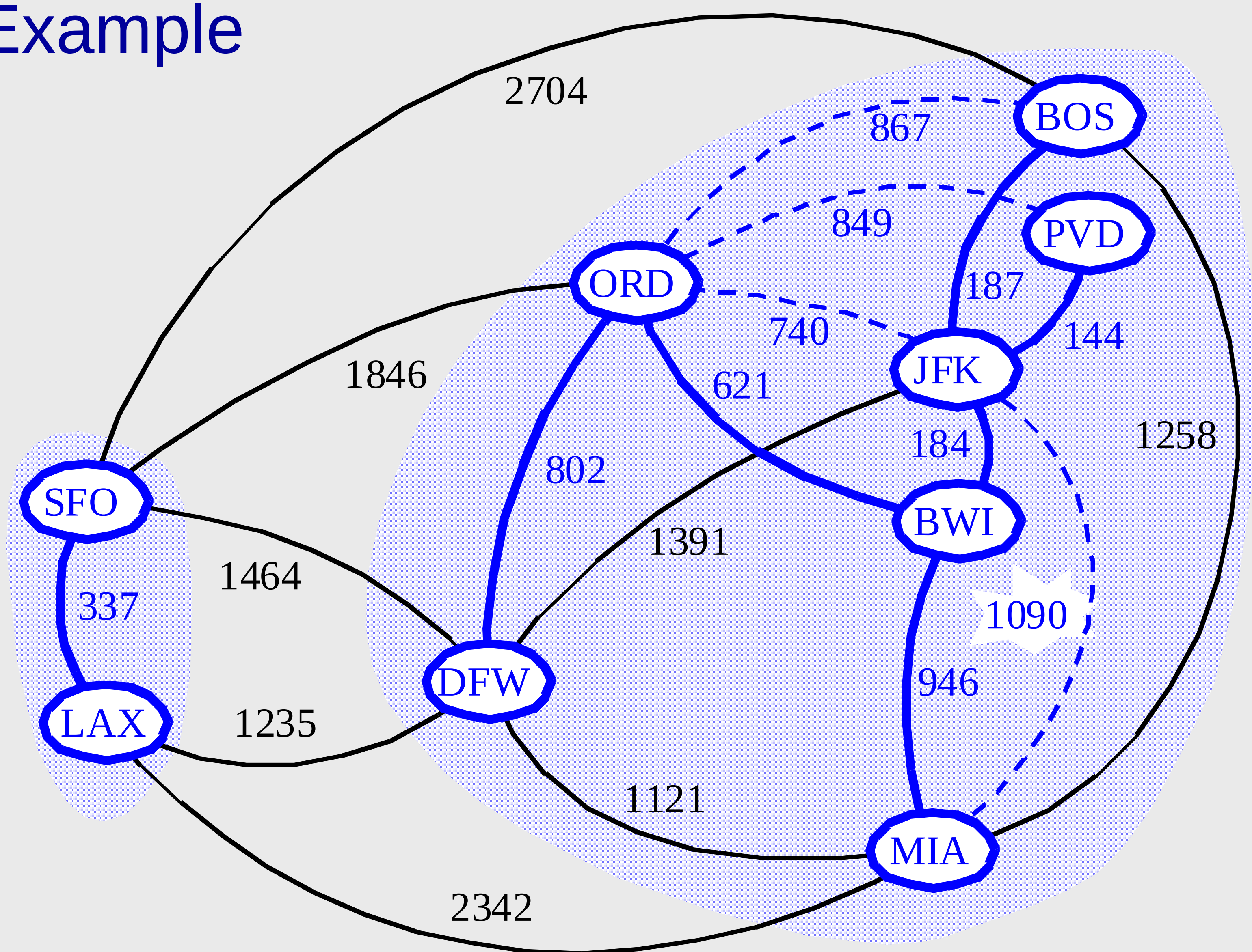
2704
867
BOS
849
PVD
ORD
187
740
144
1846
JFK
621
1258
184
802
SFO
BWI
1391
1464
337
1090
DFW
946
LAX
1235
1121
MIA
2342

Example

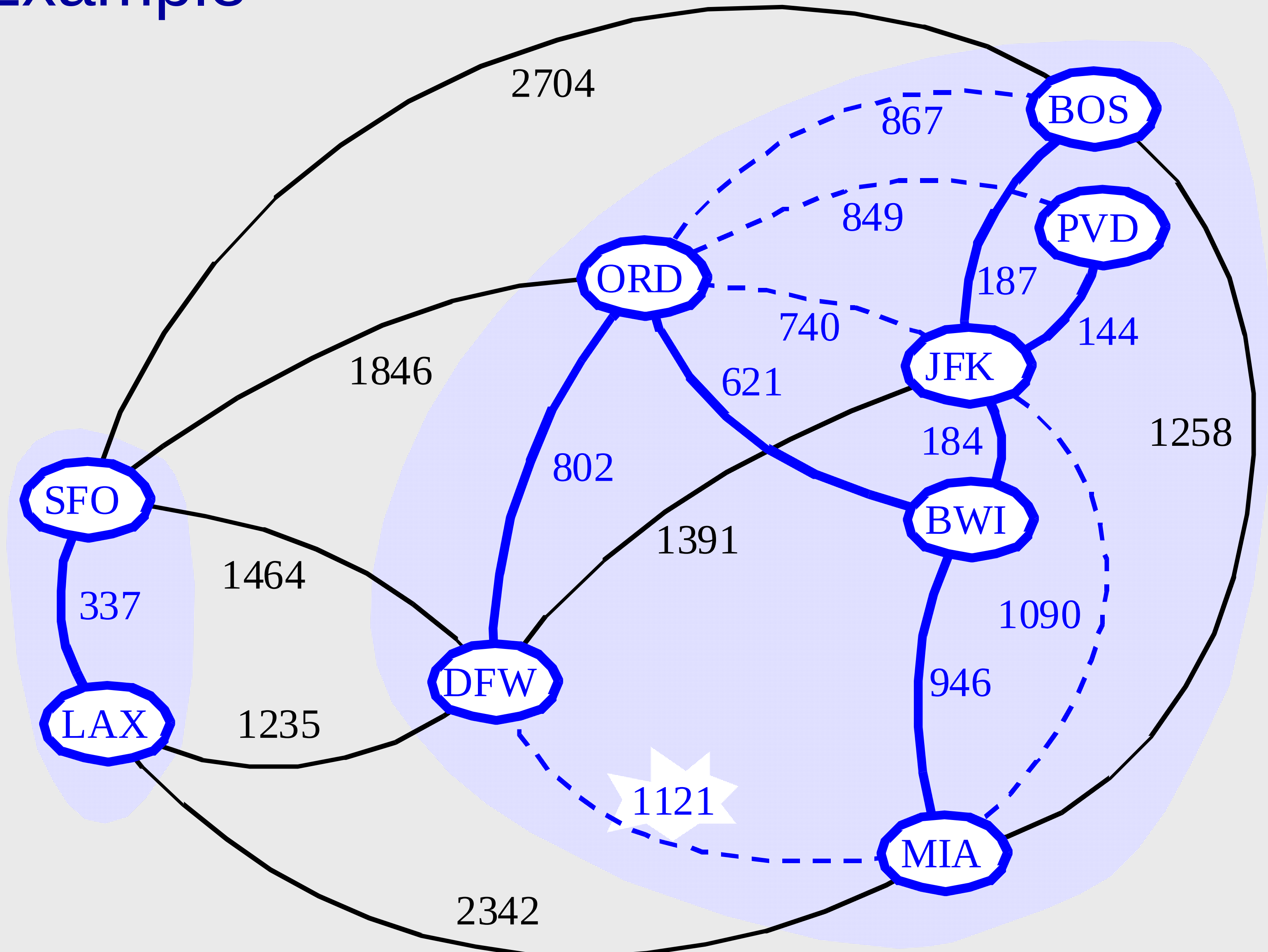


2704
867
BOS
849
PVD
ORD
187
740
144
1846
JFK
621
1258
184
SFO
802
BWI
1391
1464
337
1090
946
DFW
LAX
1235
1121
MIA
2342

Example

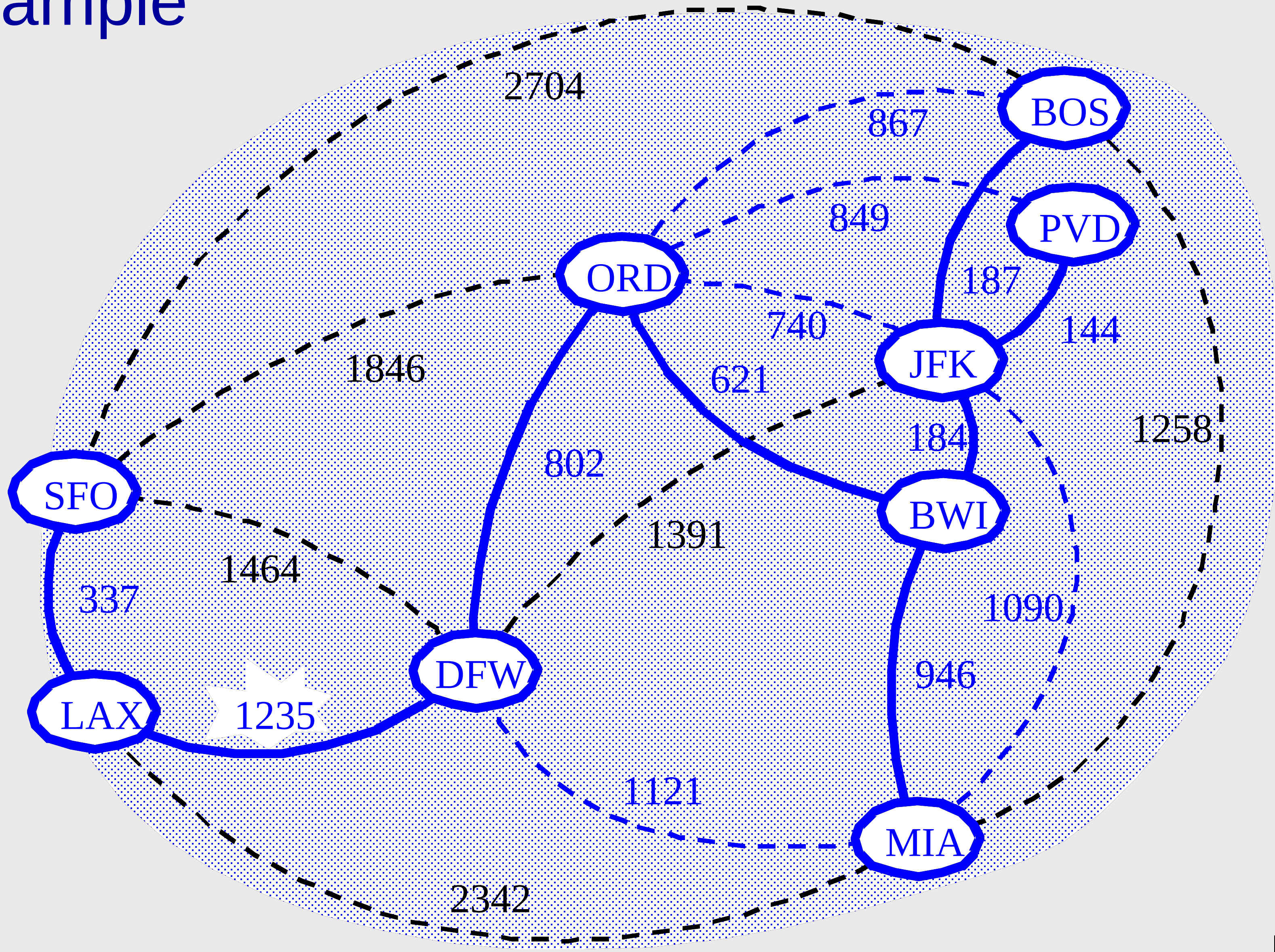

2704
BOS
867
PVD
849
ORD
187
740
144
1846
621
JFK
1258
184
802
SFO
BWI
1391
1464
1090
337
946
DFW
LAX
1235
1121
MIA
2342

Example



2704
BOS
867
849
PVD
ORD
187
740
144
1846
JFK
621
1258
184
802
SFO
BWI
1391
1464
337
1090
DFW
946
LAX
1235
1121
MIA
2342

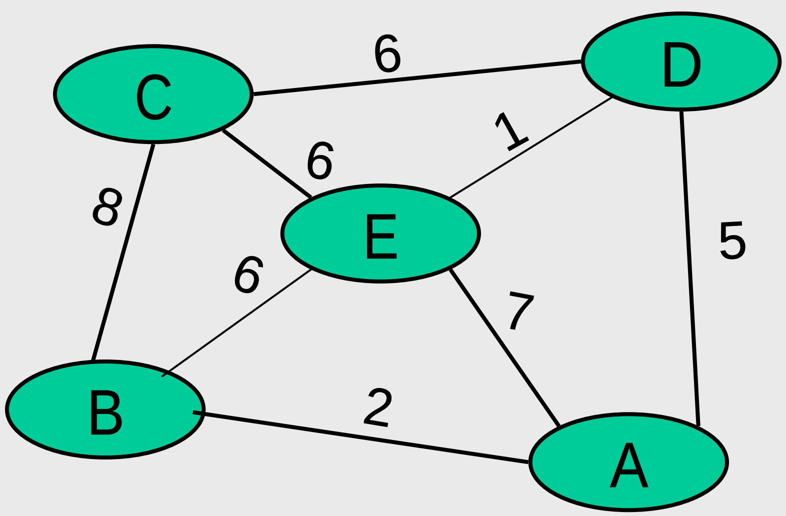
Example

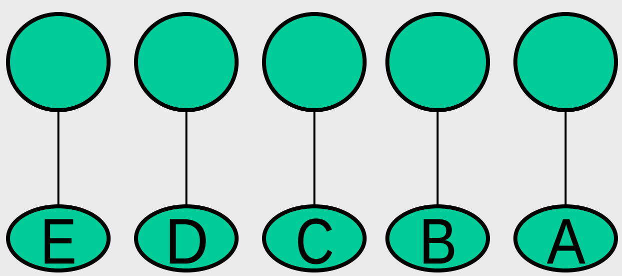


2704
BOS
867
849
PVD
ORD
187
740
144
JFK
1846
621
184
1258
SFO
802
BWI
1391
1464
337
1090
DFW
946
LAX
1235
1121
MIA
2342

Minimum Spanning Tree

- **Baruvka's Algorithm**
  - **Create a forest of n trees**
  - **Loop while (there is > 1 tree in the forest)**
    - **For each tree T_i in the forest**
      - **Find the smallest edge e = (u,v), in the edge list with u in T_i and v in $T_j \neq T_i$**
      - **connects 2 trees from the forest in to one.**
  - **end loop**





Baruvka's Algorithm

- Like Kruskal's Algorithm, Baruvka's algorithm grows many "clouds" at once.

Algorithm ***BaruvkaMST***(***G***)
  T ← ***V*** {just the vertices of ***G***}
 while ***T*** has fewer than ***n***-1 edges **do**
  for each connected component ***C*** in ***T*** **do**
   Let edge ***e*** be the smallest-weight edge from ***C*** to another component in ***T*.**
   if ***e*** is not already in ***T*** **then**
    Add edge ***e*** to ***T***
 return ***T***

- Each iteration of the while-loop halves the number of connected components in T.
  - The running time is O(m log n).