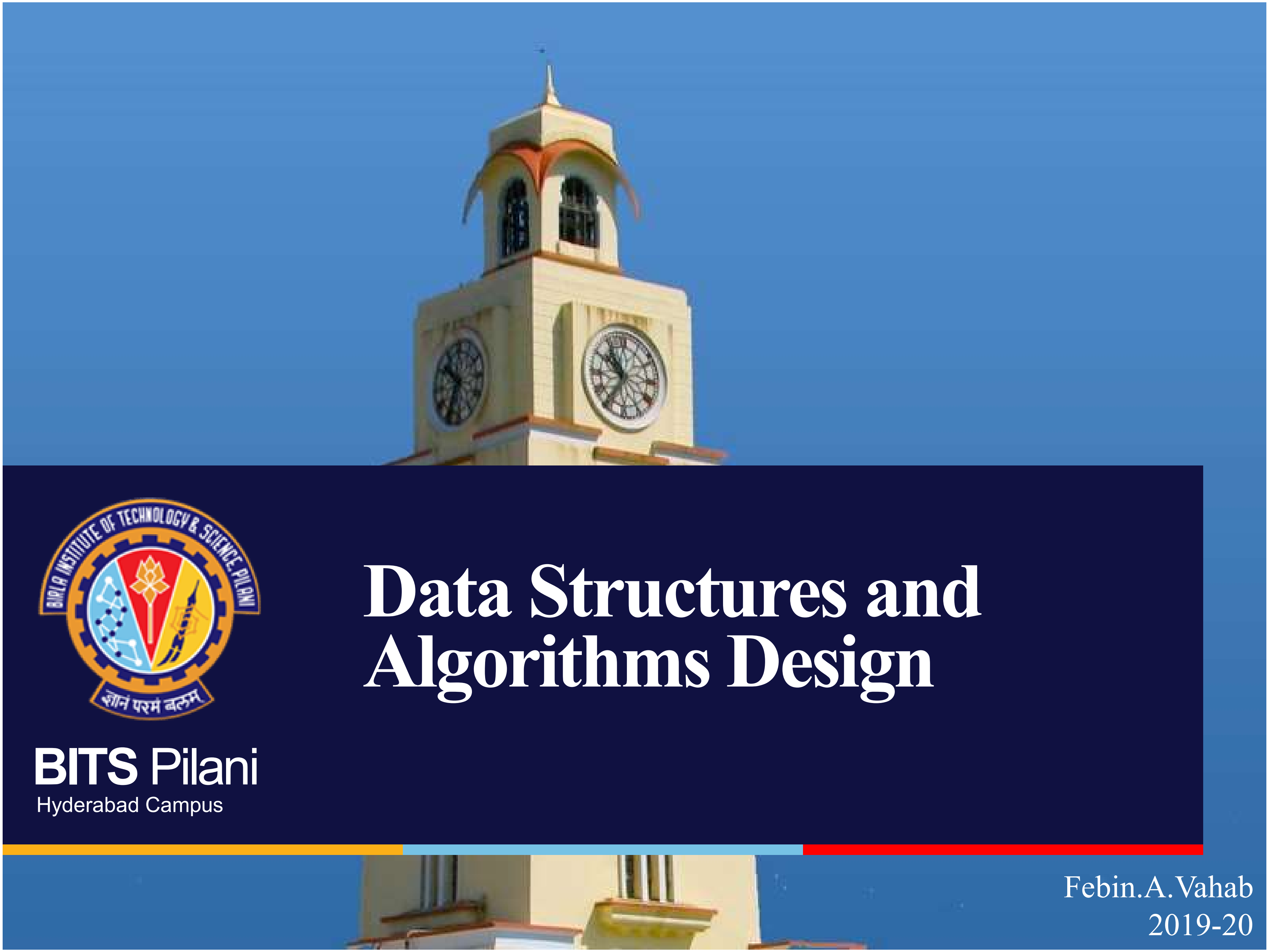


BITS Pilani
Hyderabad Campus

Data Structures and Algorithms Design

Febin.A.Vahab
2019-20

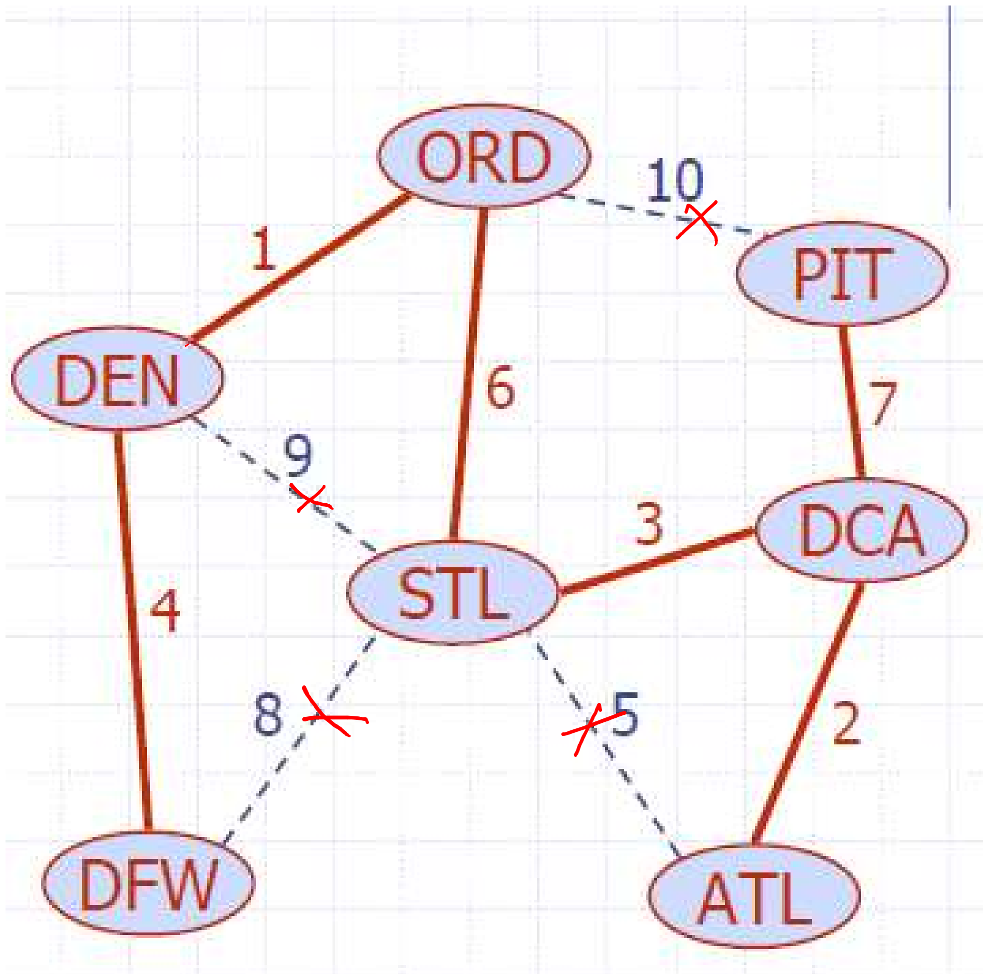
ONLINE SESSION 12 -PLAN

Sessions(#)	List of Topic Title	Text/Ref Book/external resource
12	**M5:Algorithm Design Techniques** Minimum Spanning Tree,	T1: 5.1, 7.1, 7.3

Minimum Spanning Tree

- Spanning subgraph
  - Subgraph of a graph G containing all the vertices of G
- Spanning tree
  - Spanning subgraph that is itself a tree
- Minimum spanning tree (MST)
  - Spanning tree of a weighted graph with minimum total edge weight
- Applications
  - Communications networks
  - Transportation networks

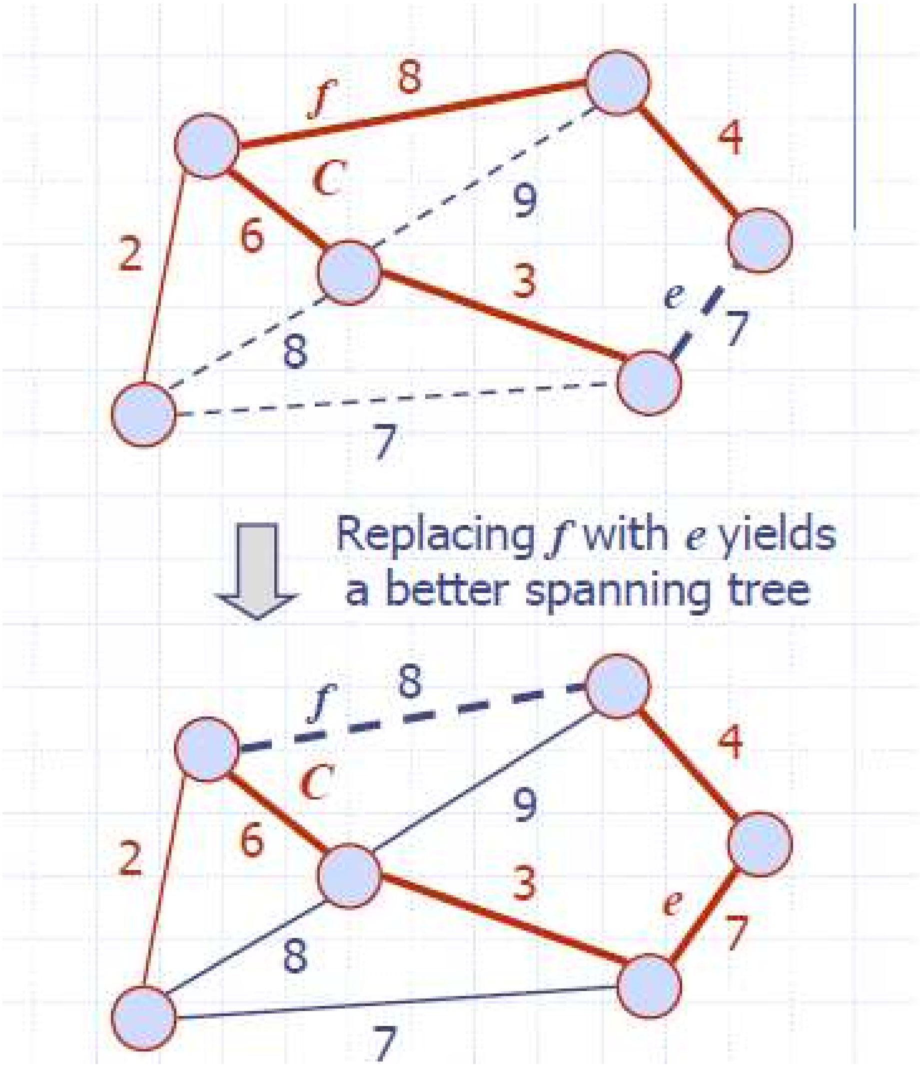
Minimum Spanning Tree



Minimum Spanning Tree

- 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*** and let ***C*** be the cycle formed by ***e*** with ***T***
  - For every edge ***f*** of ***C***, ***weight***(***f***) $\leq$ ***weight***(***e***)
- Proof:
- By contradiction
- If ***weight***(***f***) > ***weight***(***e***) we can get a spanning tree of smaller weight by replacing ***e*** with ***f***

Minimum Spanning Tree

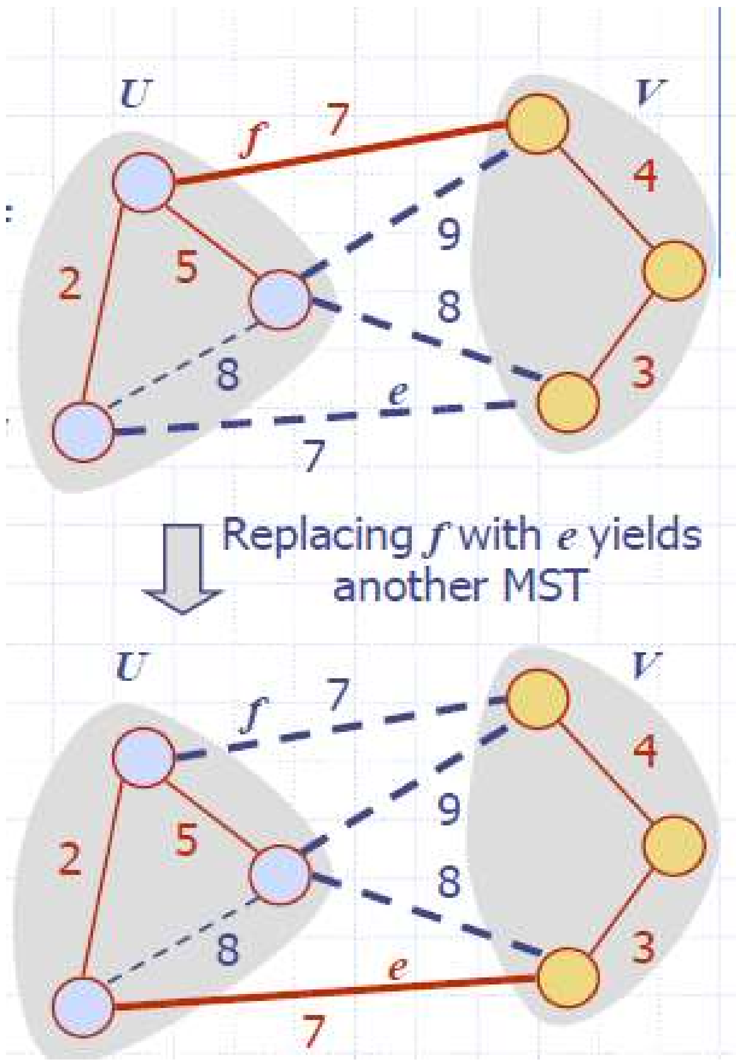


Minimum Spanning Tree

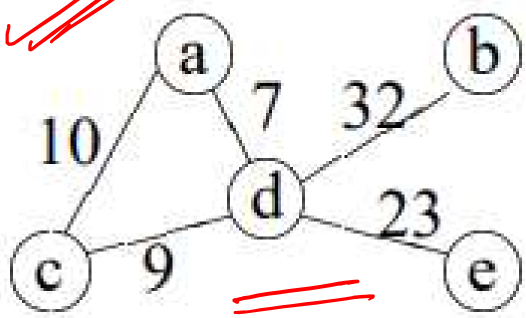
- Partition Property:
  - Consider a partition of the vertices of G into subsets U and V
  - Let e be an edge of minimum weight across the partition
  - There is a minimum spanning tree of G containing edge e

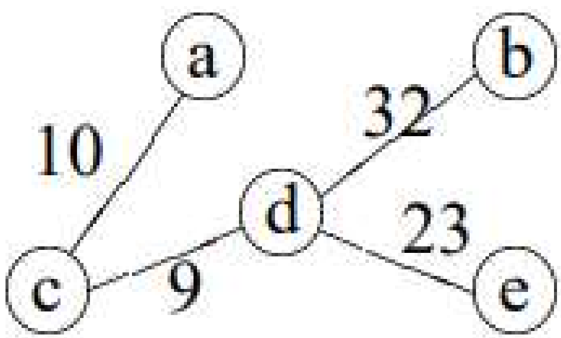
- Proof:
  - Let T be an MST of G
  - If T does not contain e, consider the cycle C formed by e with T and let f be an edge of C across the partition
  - By the cycle property, $weight(f) \leq weight(e)$
  - Thus, $weight(f) = weight(e)$
  - We obtain another MST by replacing f with e

Minimum Spanning Tree



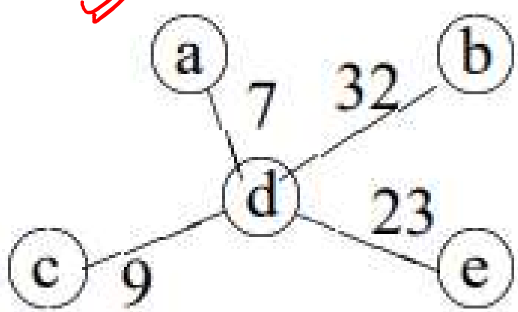
Minimum Spanning Tree
Prim's Algorithm





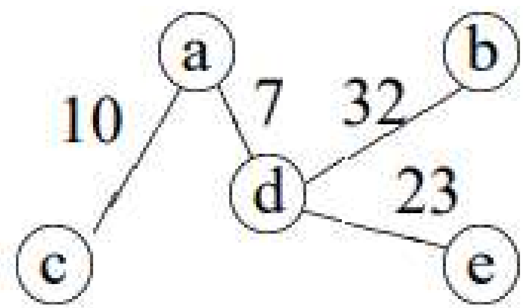
Tree 1. w=74

MST



Tree 2, w=71

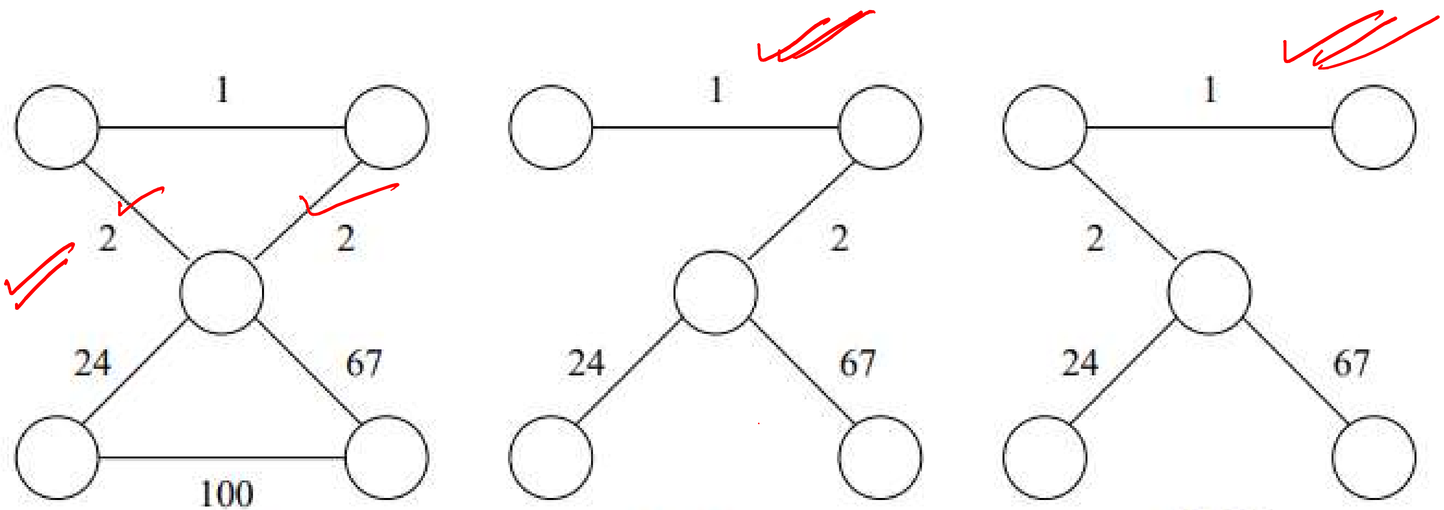
Spanning Tree



Tree 3, w=72

Minimum Spanning Tree
Prim's Algorithm

- The minimum spanning tree may not be unique. However, if the weights of all the edges are pairwise distinct, it is indeed unique

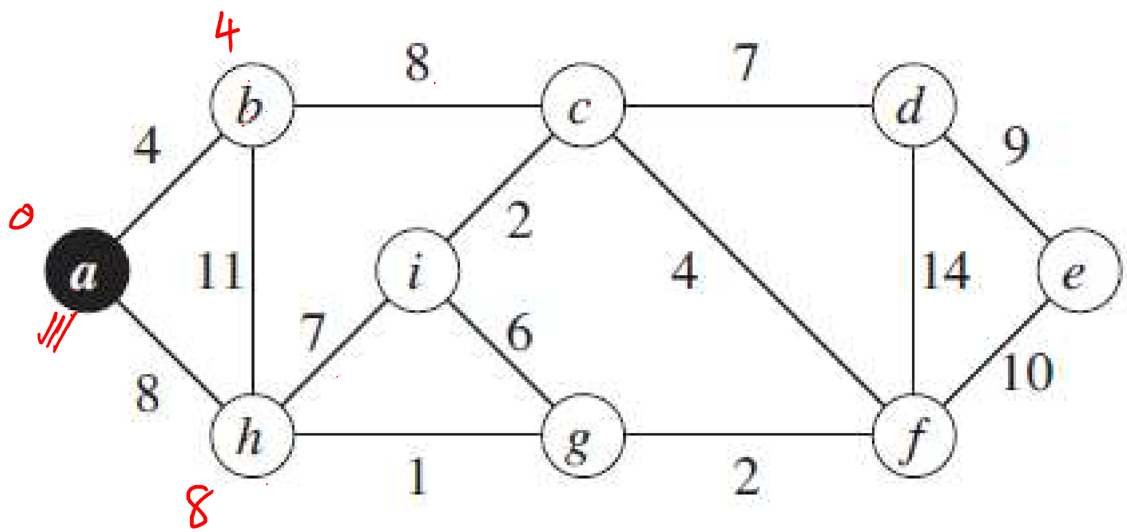


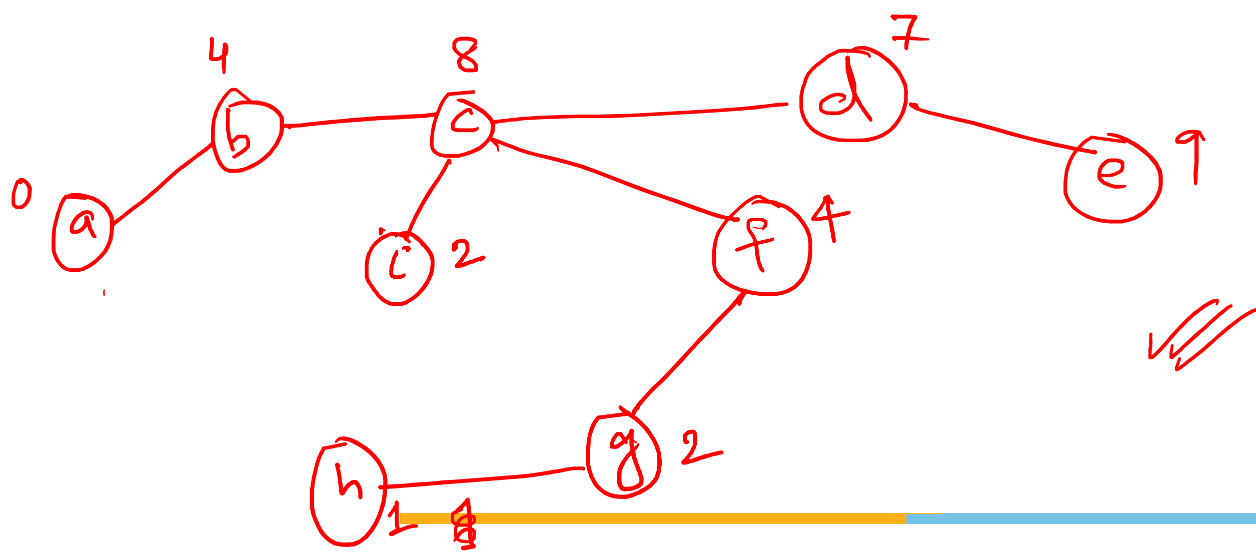
weighted graph MST1 MST2

Minimum Spanning Tree
Prim's Algorithm

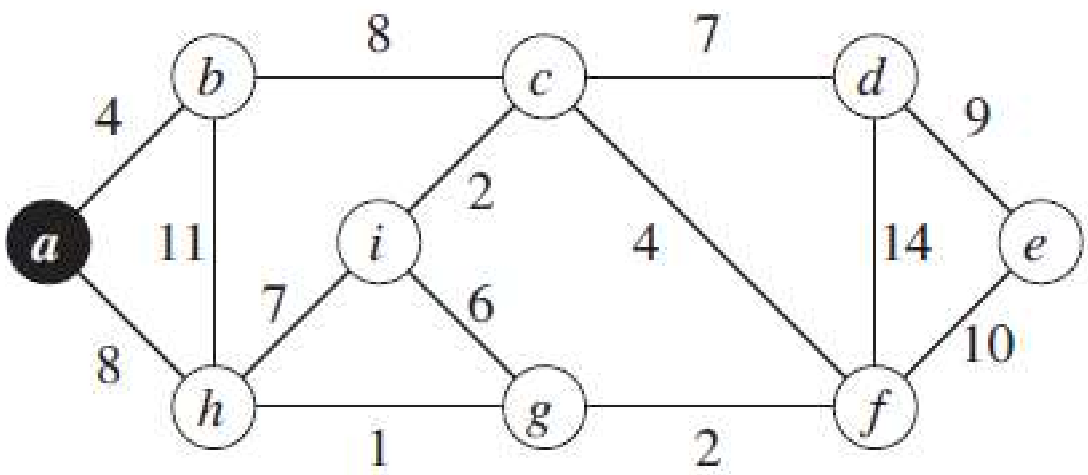
- Prim's algorithm is a greedy algorithm that finds a minimum spanning tree for a weighted undirected graph.
- This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized.
- At each step:
  - We add to the cloud the vertex ***u*** outside the cloud with the smallest distance label (greedy choice)
  - We update the labels of the vertices adjacent to ***u***

Example

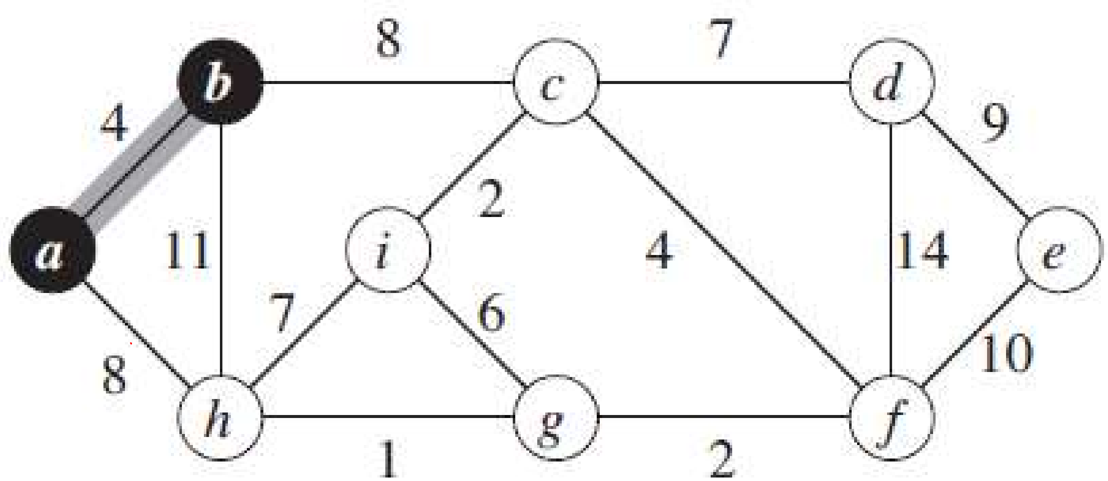




Example

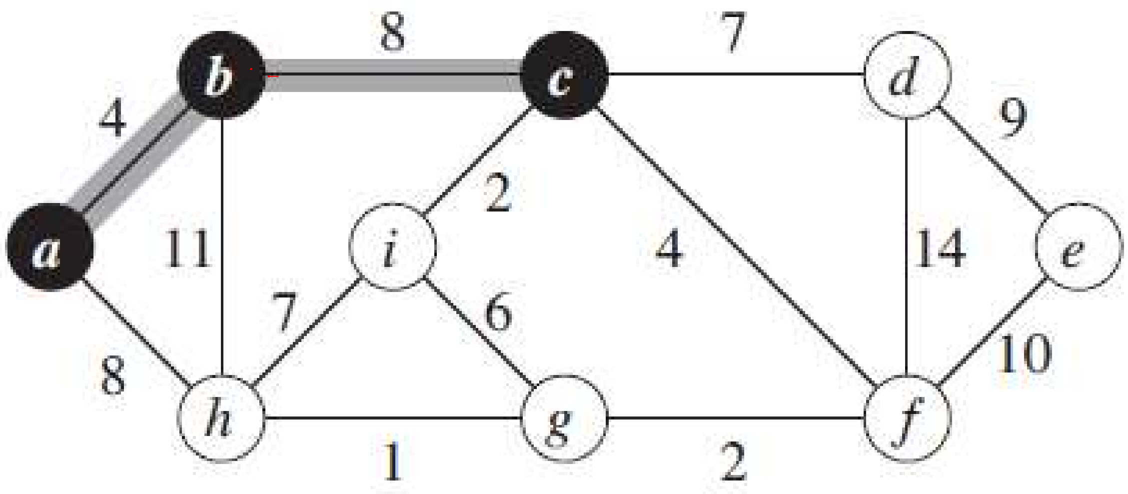


a
b
c
d
e
f
g
h
i
4
8
7
9
11
2
4
14
7
6
10
8
1
2

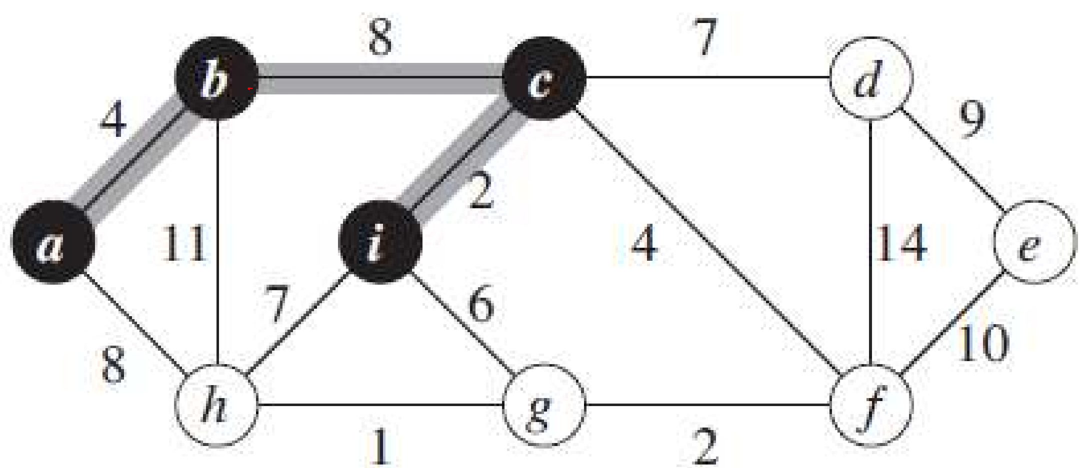


a
b
c
d
e
f
g
h
i
4
8
7
9
11
2
4
14
7
6
10
8
1
2

Example

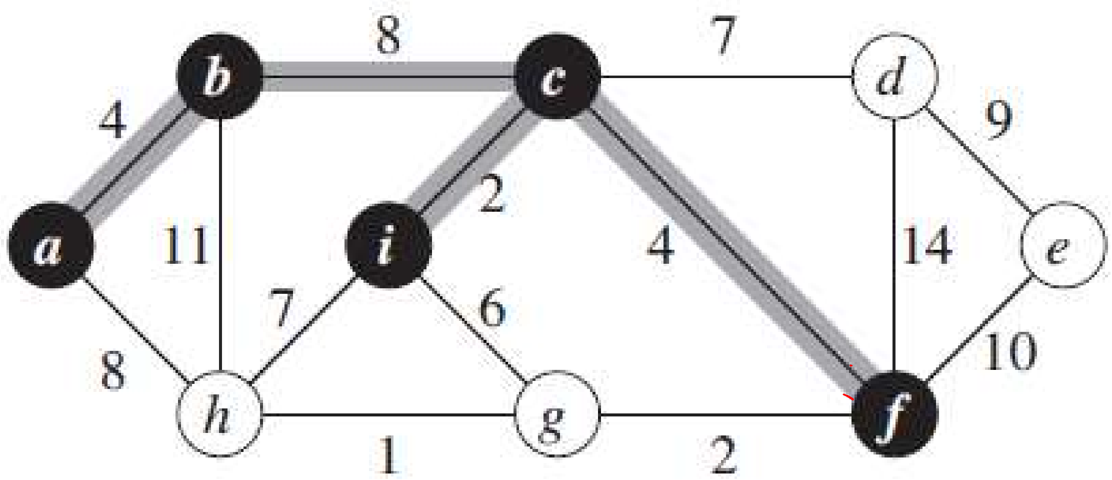


a
b
c
d
e
f
g
h
i
4
8
7
9
11
2
4
14
7
6
10
8
1
2

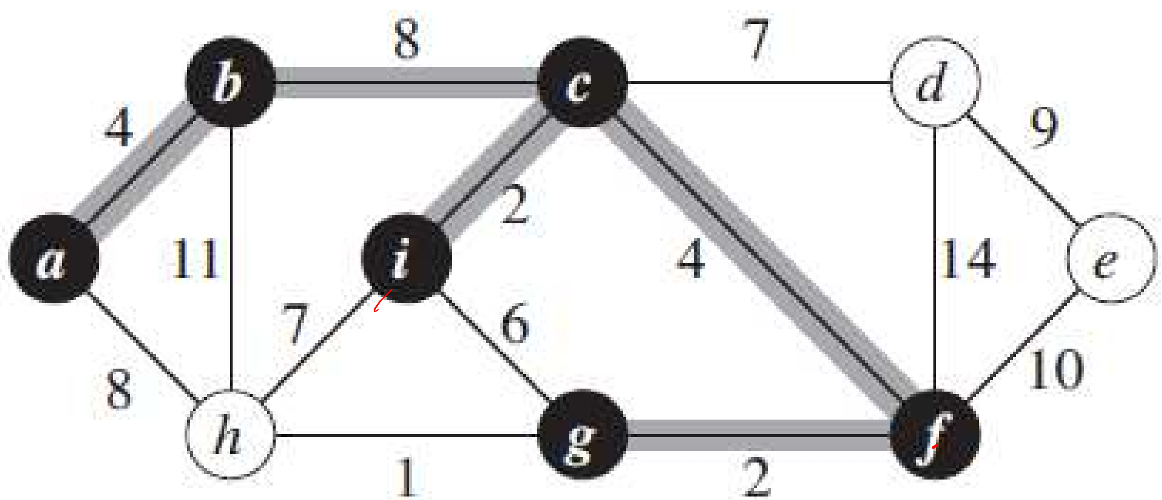


a
b
c
d
e
f
g
h
i
4
8
7
9
11
2
4
14
7
6
10
8
1
2

Example

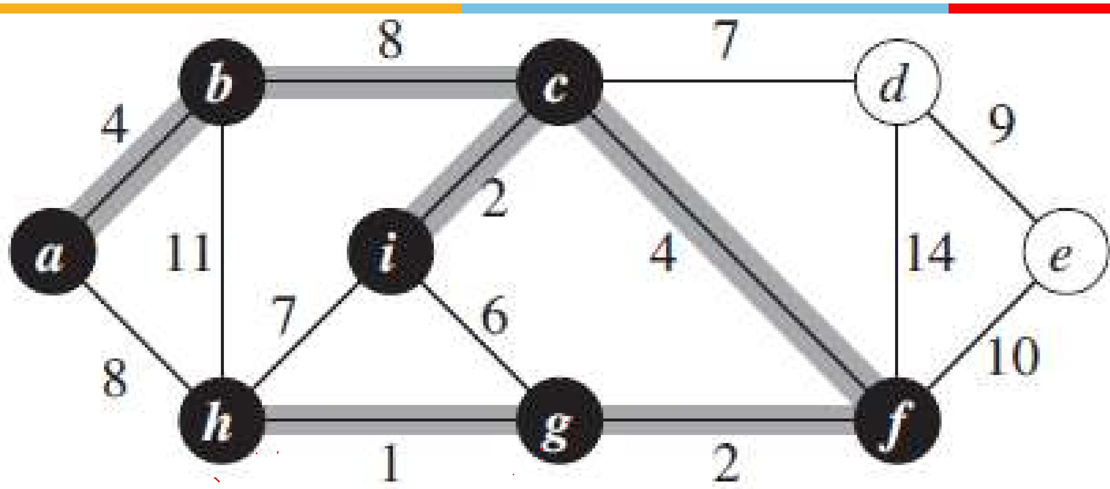


8
7
b
c
d
4
9
2
a
11
i
4
14
e
7
6
10
8
h
g
f
1
2

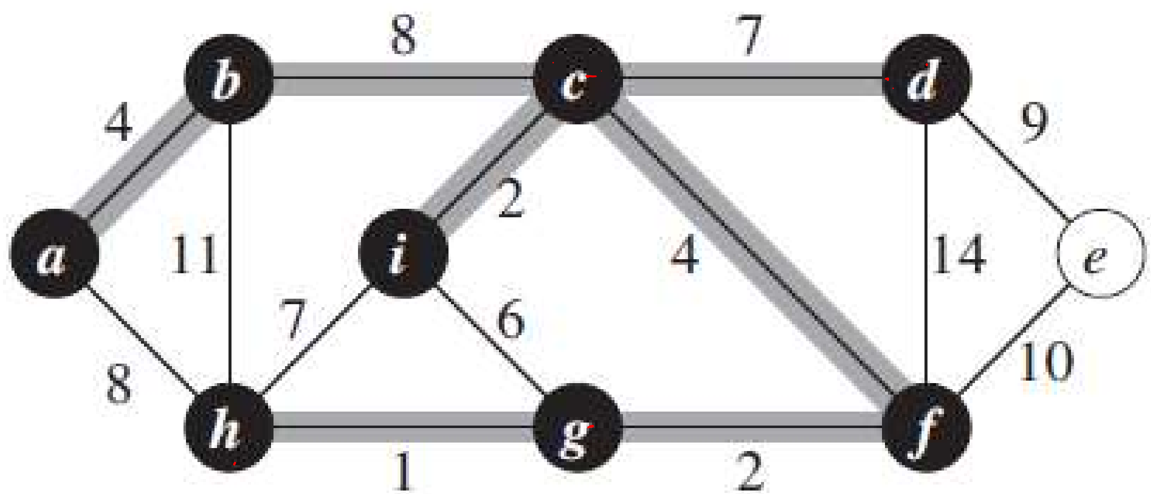


8
7
b
c
d
4
9
2
a
11
i
4
14
e
7
6
10
8
h
g
f
1
2

Example

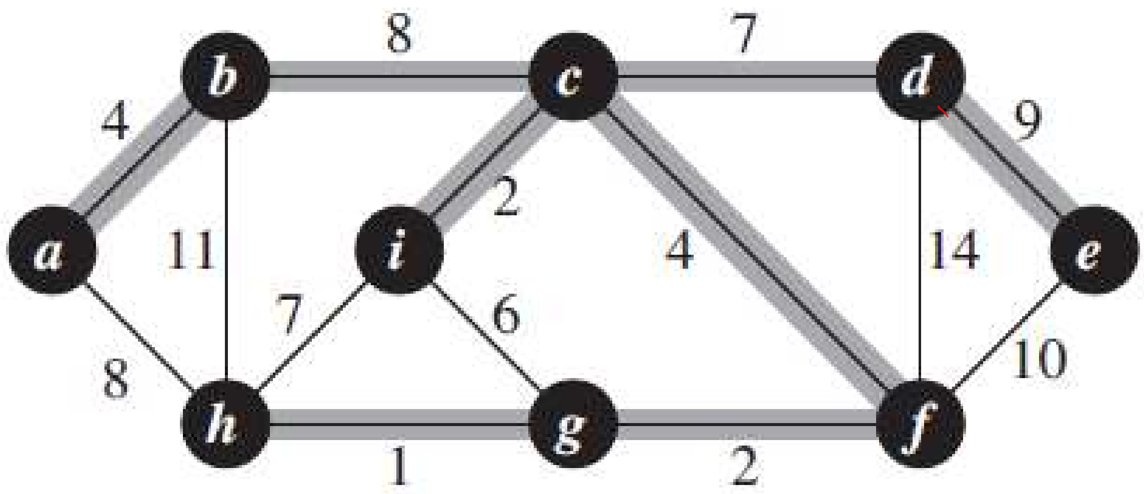


a
b
c
d
e
f
g
h
i
4
8
7
9
11
2
4
14
7
6
10
8
1
2



a
b
c
d
e
f
g
h
i
4
8
7
9
11
2
4
14
7
6
10
8
1
2

Example



Minimum Spanning Tree
Prim's Algorithm

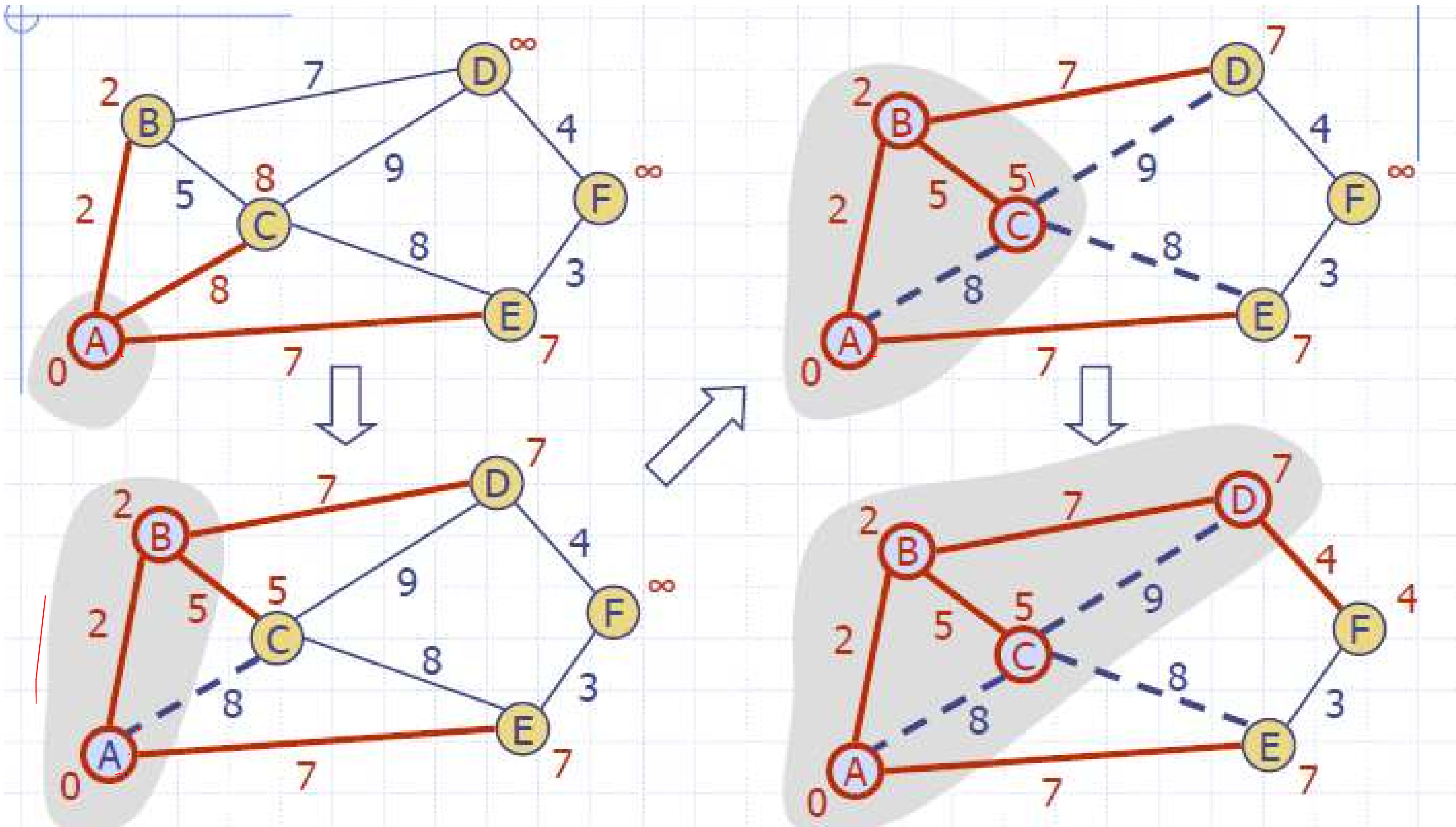
```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    for each v ∈ G.Adj[u]
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

∈ V log V

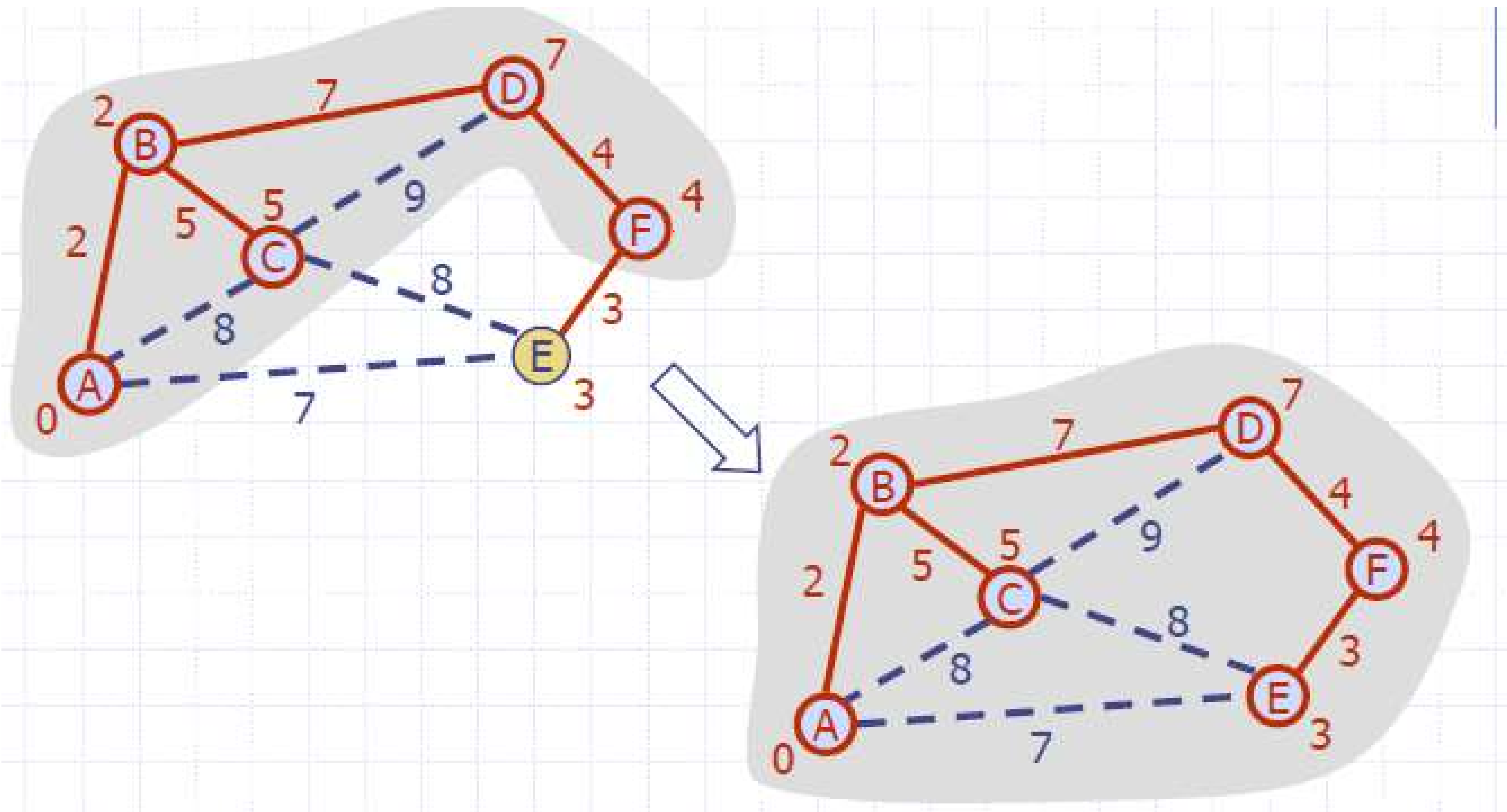
Prims's Algorithm-Analysis

- Similar to Dijkstra's,Prim's alogorithm can be implemented more efficiently by priority queue
  - Initialization O(|V|) using O(|V|) buildHeap
  - While loop O(|V|)
  - Find and remove min distance vertices = O(log |V|) using O(log |V|) deleteMin
  - Taken together that part of the loop and the calls to ExtractMin take O(VlogV) time
  - Potentially |E| updates:The for loop is executed once for each edge in the graph (E times), and within the for loop ,the update costs O(log |V|) using decreaseKey
  - Total time O(|V|log|V| + |E|log|V|) = O ((V+E)log V)
  - Since graph is connected, number of edges should be atleast n-1 is m>=n-1 ***ie.*** |V| = O(|E|) assuming a connected graph
  - Hence **the total time=O(|E| log |V|)**

Minimum Spanning Tree
Prim's Algorithm-Example



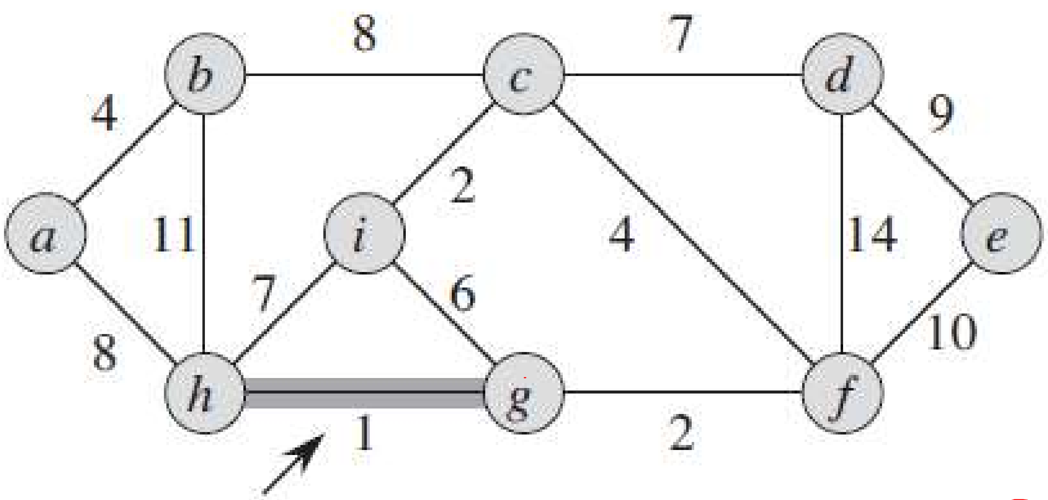
Minimum Spanning Tree
Prim's Algorithm



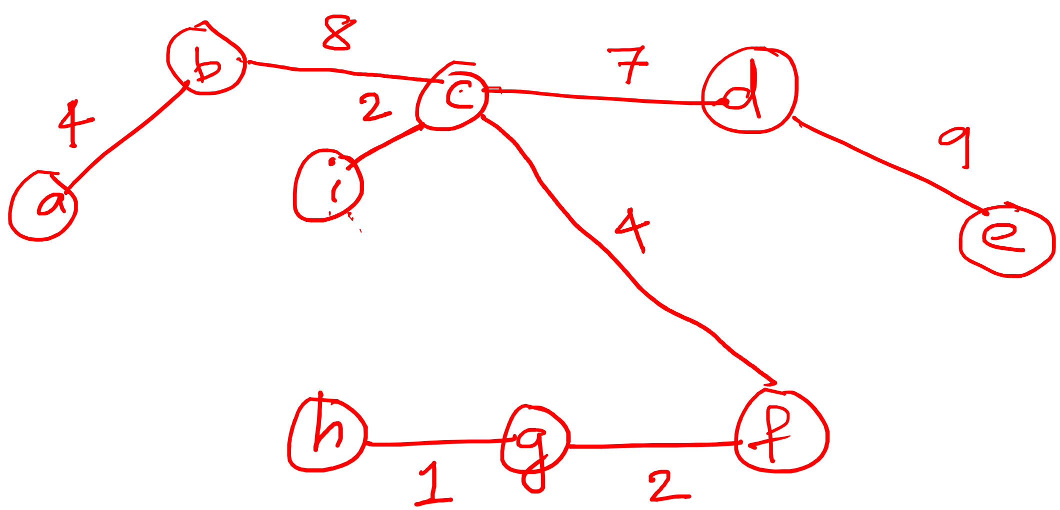
Minimum Spanning Tree
Kruskal's Algorithm

- It builds the MST in forest.
- Initially, each vertex is in its own tree in forest.
- Then, algorithm consider each edge in turn, order by increasing weight.
- If an edge (u, v) connects two different trees, then (u, v) is added to the set of edges of the MST, and two trees connected by an edge (u, v) are merged into a single tree
- If an edge (u, v) connects two vertices in the same tree, then edge (u, v) is discarded.

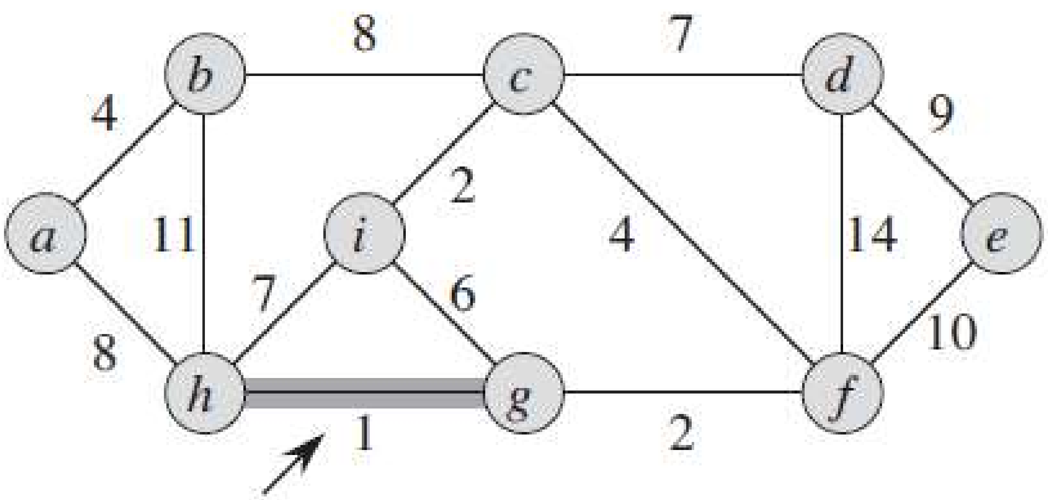
Minimum Spanning Tree
Kruskal's Algorithm-Example



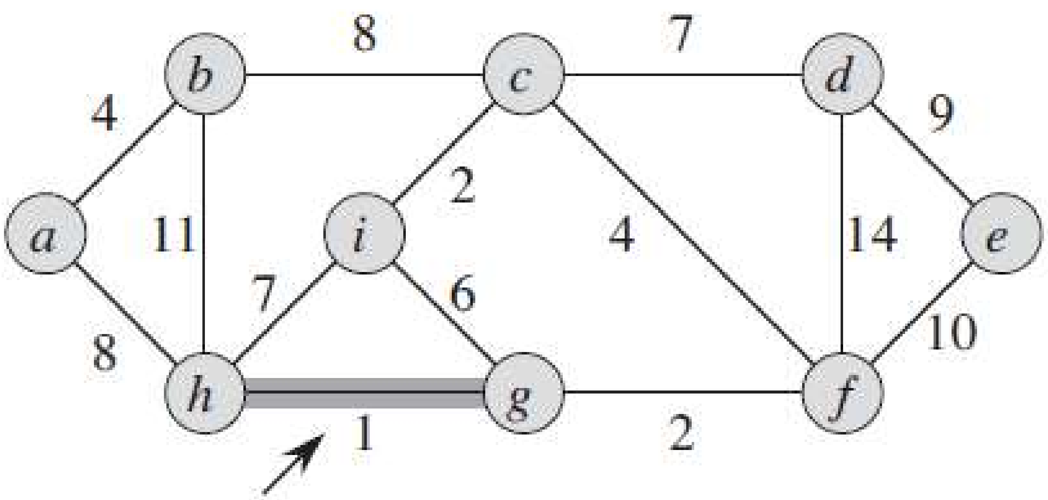
increasing order of weights

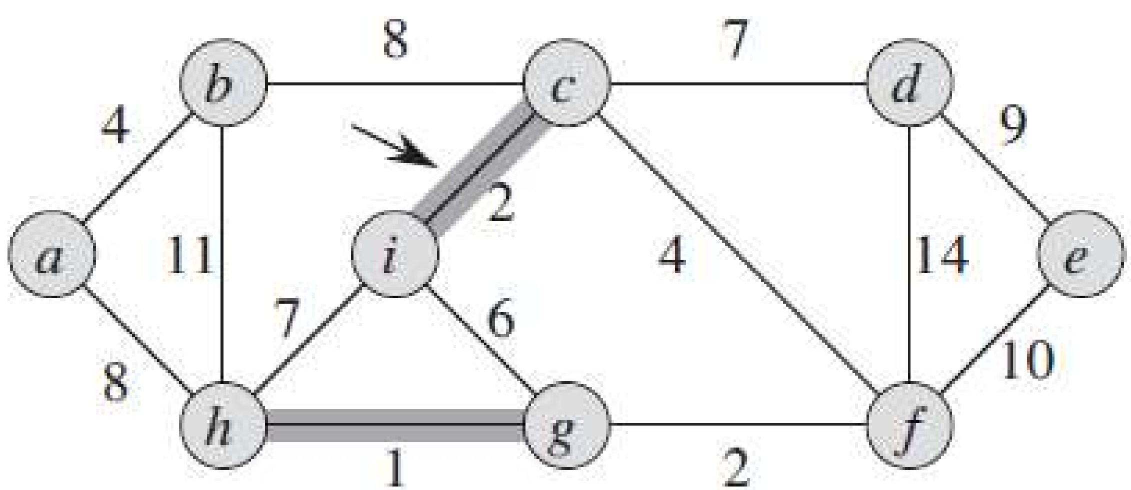


Minimum Spanning Tree
Kruskal's Algorithm-Example

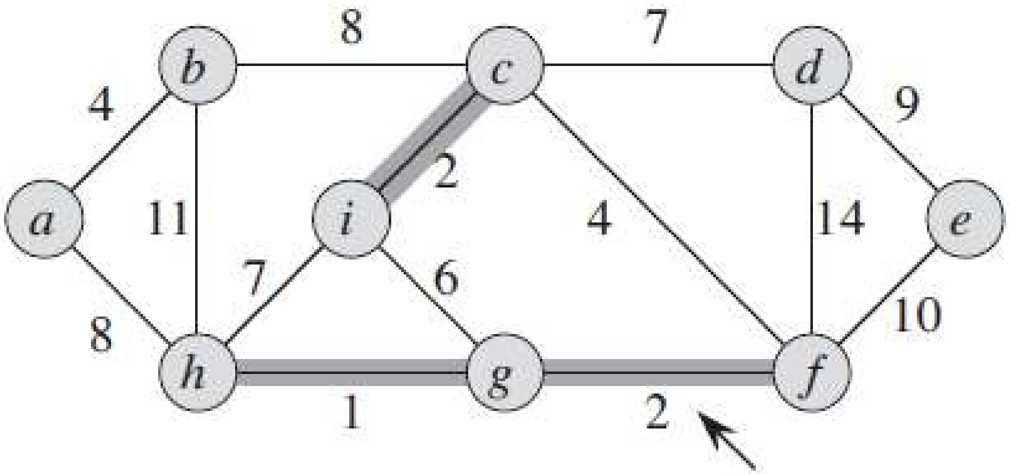


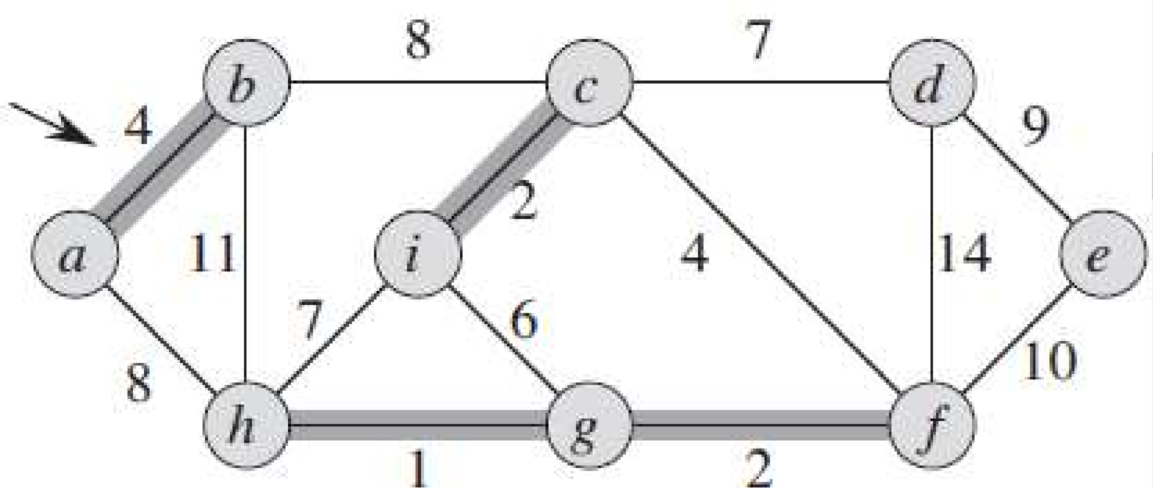
Minimum Spanning Tree
Kruskal's Algorithm-Example



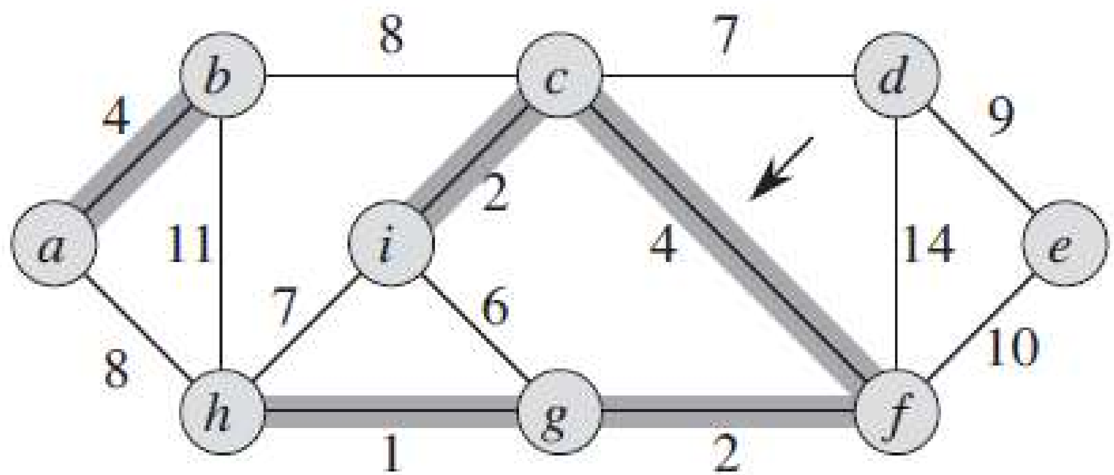


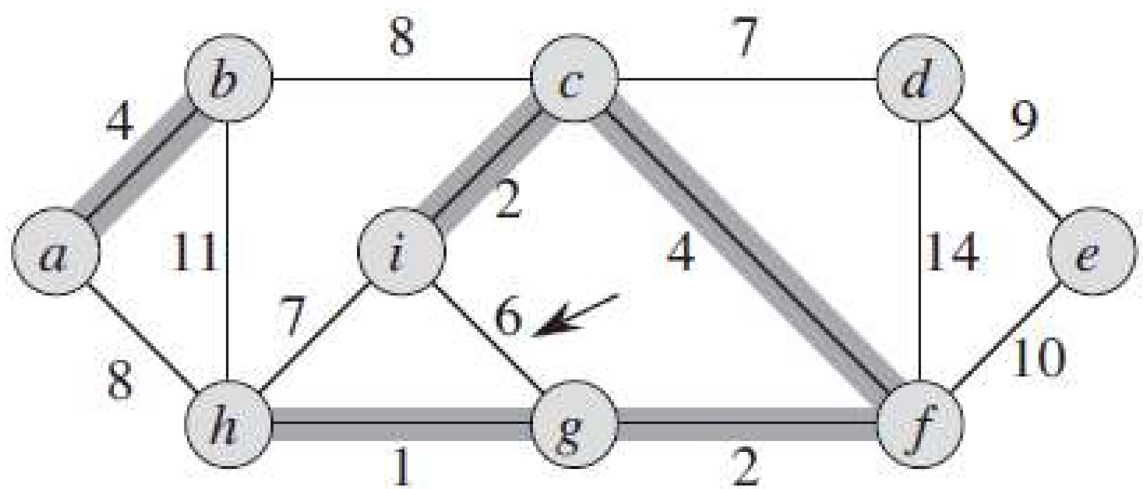
Minimum Spanning Tree
Kruskal's Algorithm-Example



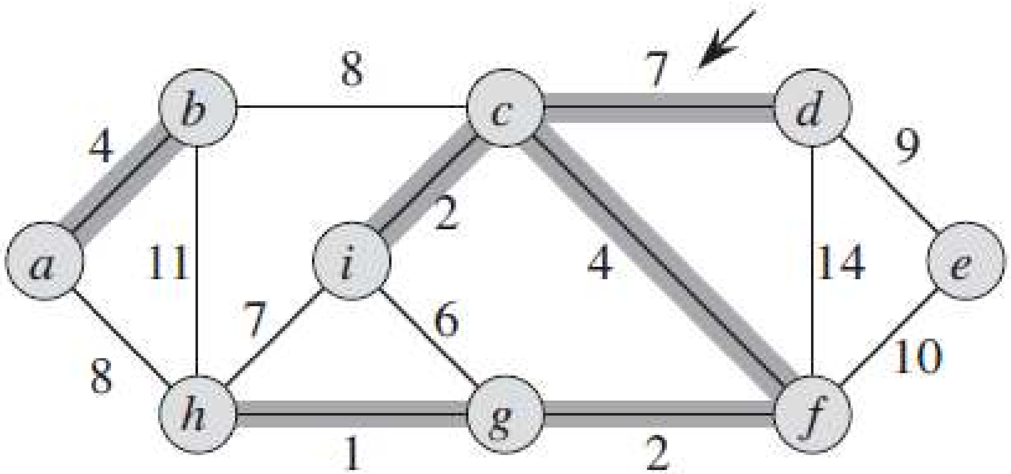


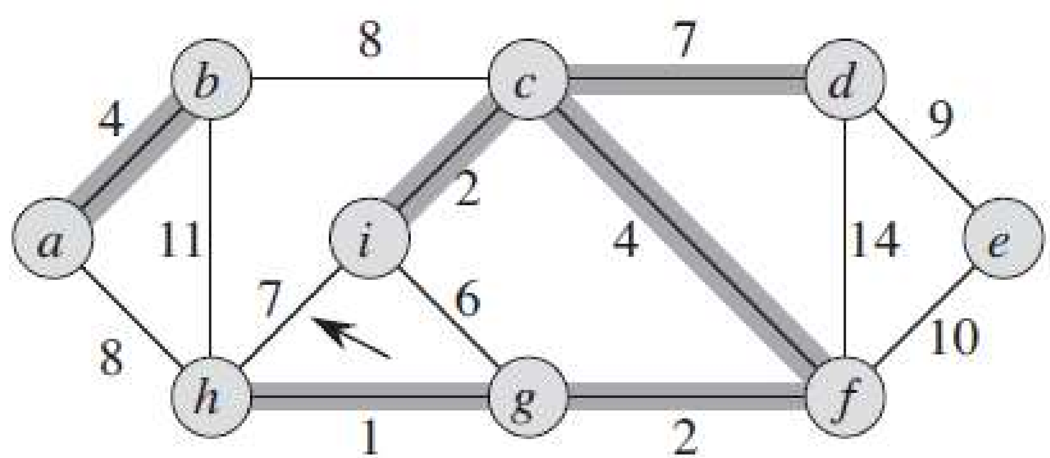
Minimum Spanning Tree
Kruskal's Algorithm-Example



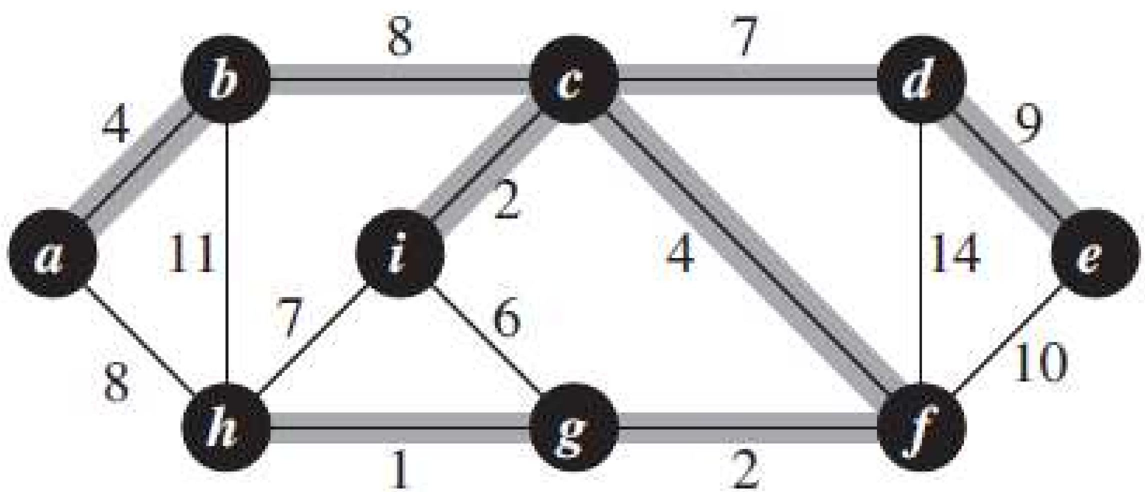


Minimum Spanning Tree
Kruskal's Algorithm-Example





Minimum Spanning Tree
Kruskal's Algorithm-Example



Minimum Spanning Tree
Kruskal's Algorithm

```
Algorithm KruskalMST(G)
    for each vertex V in G do
        define a Cloud(v) of ← {v}
    let Q be a priority queue.
    Insert all edges into Q using their
    weights as the key
    T ← ∅
    while T has fewer than n-1 edges do
        edge e = T.removeMin()
        Let u, v be the endpoints of e
        if Cloud(v) ≠ Cloud(u) then
            Add edge e to T
            Merge Cloud(v) and Cloud(u)
    return T
```

Minimum Spanning Tree
Kruskal's Algorithm

- A priority queue stores the edges-
  - **Key: weight**
  - **Element: edge**
- We can implement the priority queue Q using a heap.
- Thus, we can initialize Q in **O(E) time using bottom-up heap construction**
- In addition, at each iteration of the **while** loop, we can remove a minimum-weight edge in O(log E) time.
- Taken together that part of the loop and the calls to removeMin take O(ElogE) time

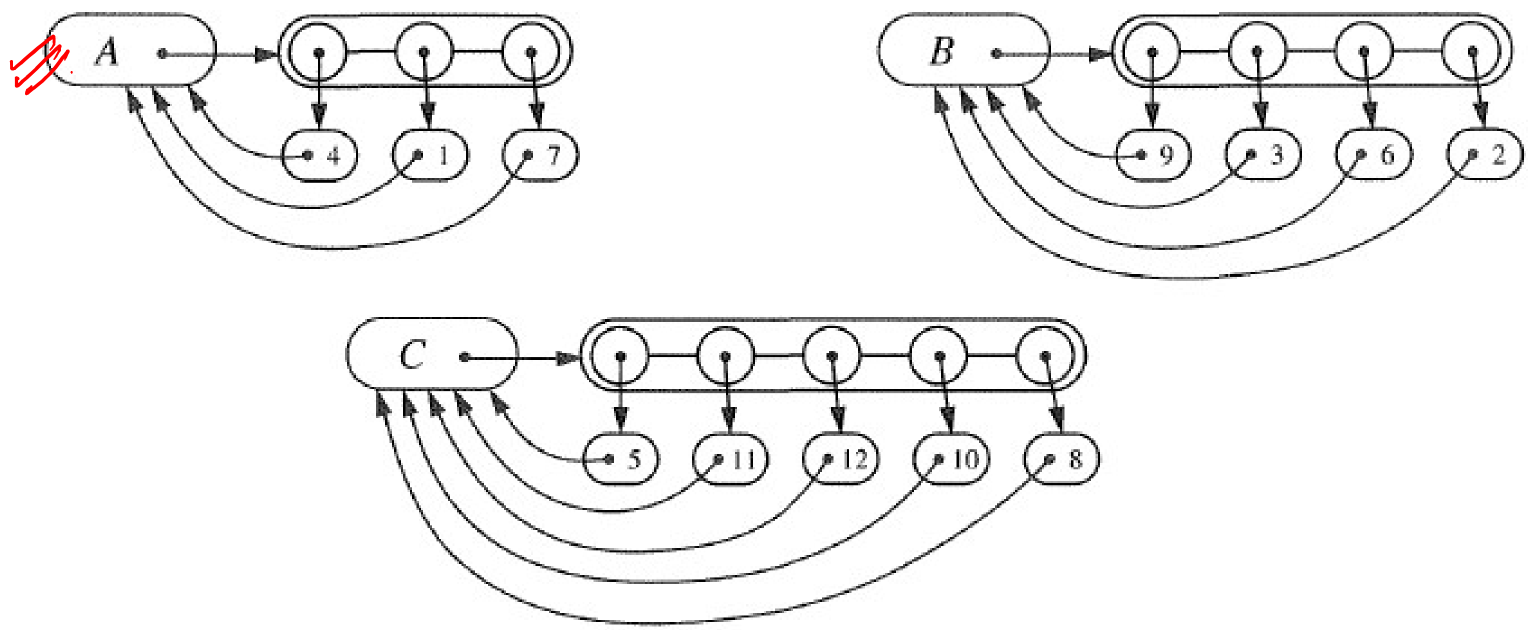
```
Algorithm KruskalMST(G)
  for each vertex V in G do
    define a Cloud(v) of ← {v}
  let Q be a priority queue.
  Insert all edges into Q using their
  weights as the key
  T ← ∅
  while T has fewer than n-1 edges do      m logn
    edge e = T.removeMin()
    Let u, v be the endpoints of e
    if Cloud(v) ≠ Cloud(u) then
      Add edge e to T
      Merge Cloud(v) and Cloud(u)
  return T
```

Minimum Spanning Tree
Kruskal's Algorithm

- Each set is stored in a sequence
- The sequence for a set A stores locator nodes as its elements. Each locator node has a reference to its element e and a reference to the sequence storing e.
- Element () of the locator ADT takes O(1) time,
  - operation find(u) takes O(1) time, and returns the set of which u is a member.
  - 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(which is O(n) in the worst case)

Minimum Spanning Tree
Kruskal's Algorithm

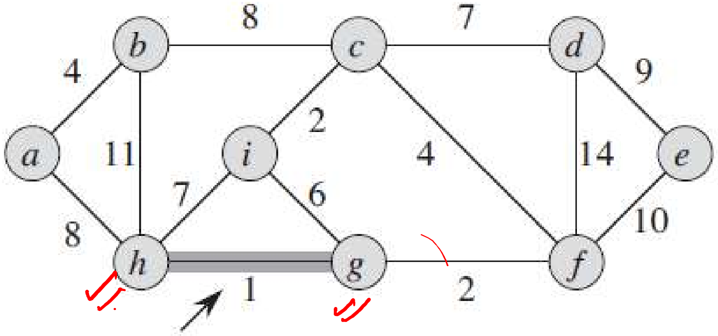
a b c d e f



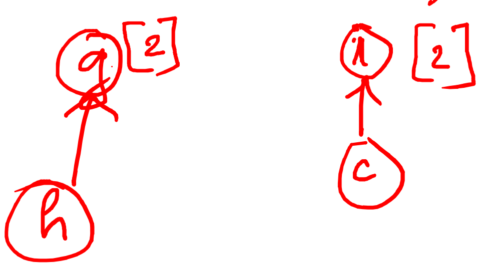
Minimum Spanning Tree
Kruskal's Algorithm-Example

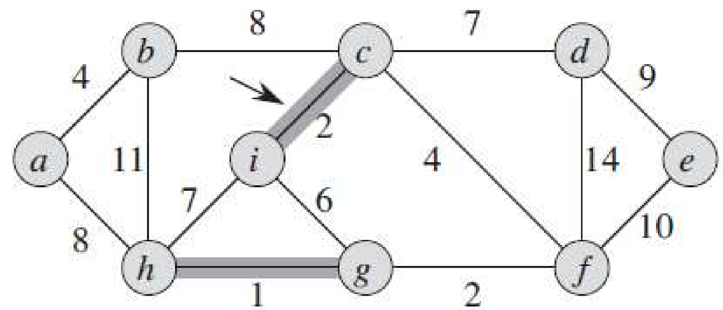
a:a	b:b	c:c	d:d	e:e	f:f	g:g	h:h	i:i

find = O(1)



a:a	b:b	c:c	d:d	e:e	f:f	g:g	h:g	i:i

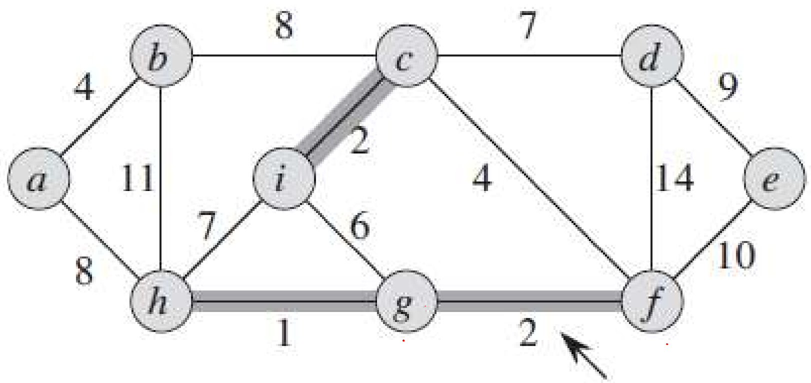




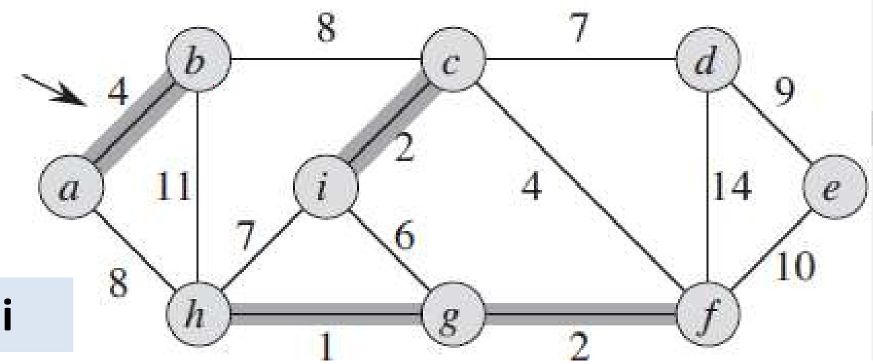
a:a	b:b	c:i	d:d	e:e	f:f	g:g	h:g	i:i

Minimum Spanning Tree
Kruskal’s Algorithm-Example

a:a	b:b	c:i	d:d	e:e	f:f	g:g	h:g	i:i



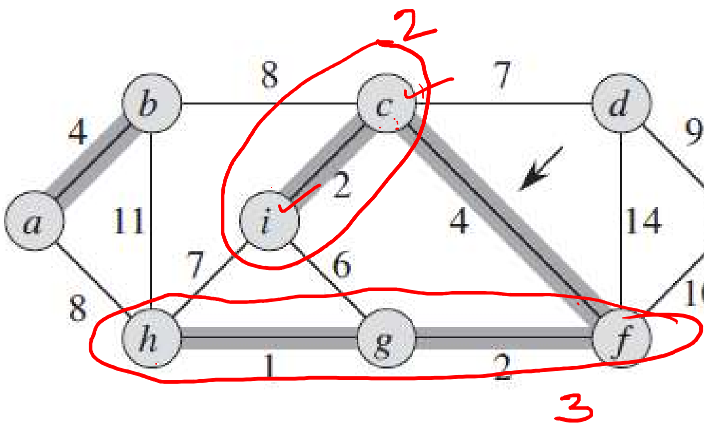
a:a	b:b	c:i	d:d	e:e	f:g	g:g	h:g	i:i



a:b	b:b	c:i	d:d	e:e	f:g	g:g	h:g	i:i

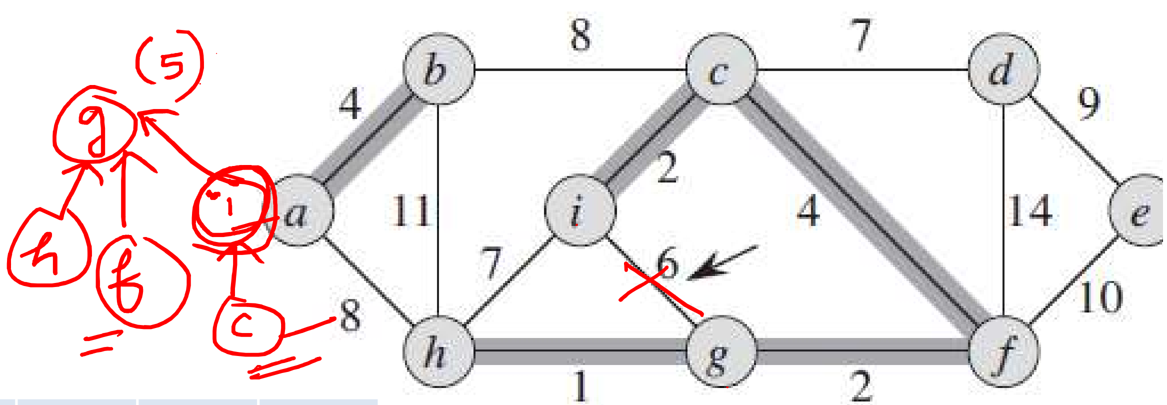
Minimum Spanning Tree
Kruskal's Algorithm-Example

a:b	b:b	c:i	d:d	e:e	f:g	g:g	h:g	i:i



$O(\min(c_v), c_u))$

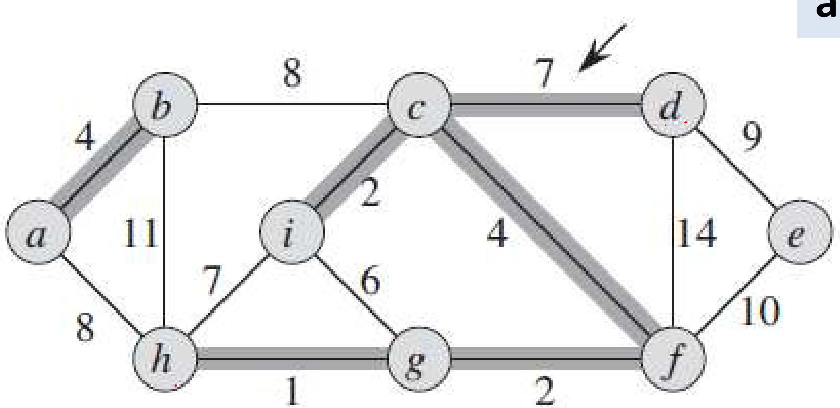
a:b	b:b	c:g	d:d	e:e	f:g	g:g	h:g	i:g



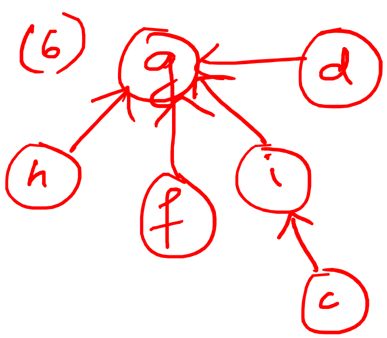
a:b	b:b	c:g	d:d	e:e	f:g	g:g	h:g	i:g

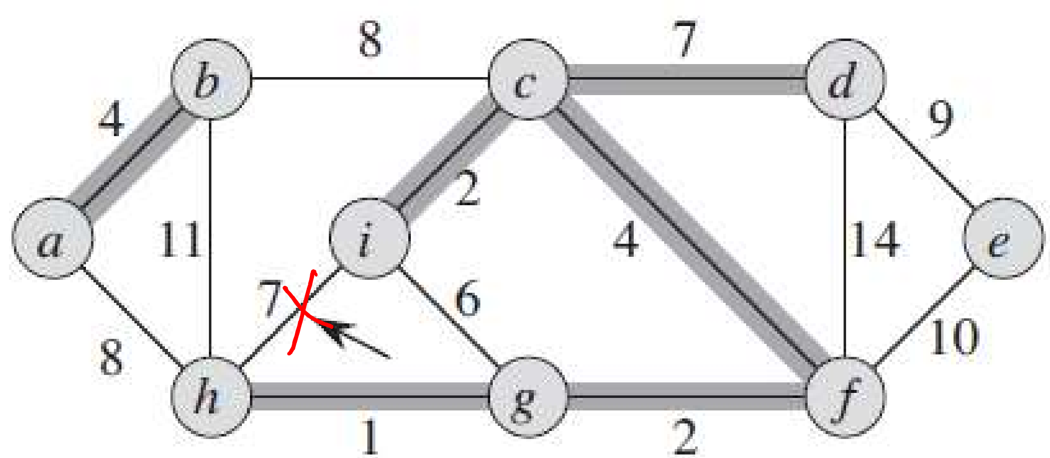
Minimum Spanning Tree
Kruskal's Algorithm-Example

a:b	b:b	c:g	d:d	e:e	f:g	g:g	h:g	i:g



a:b	b:b	c:g	d:g	e:e	f:g	g:g	h:g	i:g

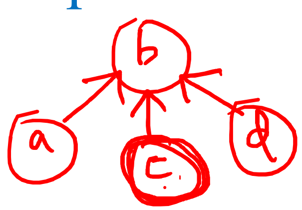


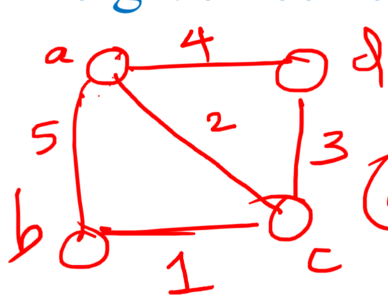


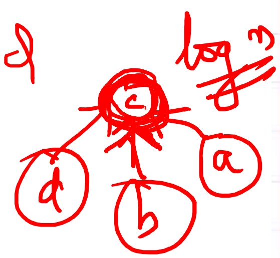
Minimum Spanning Tree
Kruskal's Algorithm

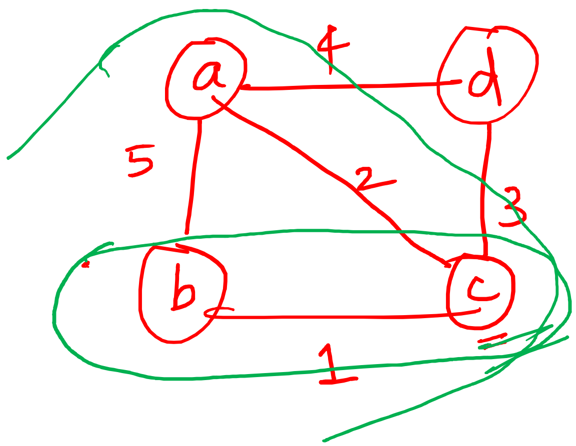
- Performing a series of n union , and find operations, using the sequence-based implementation, starting from an initially empty partition takes O(n log n) time
- The important observation is that each time we move a locator from one set to another, the size of the new set at least doubles. Thus, each locator is moved from one set to another at most **log n** times
- There are **n** different elements referenced in the given series of operations.

```
Algorithm KruskalMST(G)
  for each vertex V in G do
    define a Cloud(v) of ← {v}
  let Q be a priority queue.
  Insert all edges into Q using their
  weights as the key
  T ← ∅
  while T has fewer than n-1 edges do
    edge e = T.removeMin()
    Let u, v be the endpoints of e
    if Cloud(v) ≠ Cloud(u) then
      Add edge e to T
      Merge Cloud(v) and Cloud(u)
  return T
```



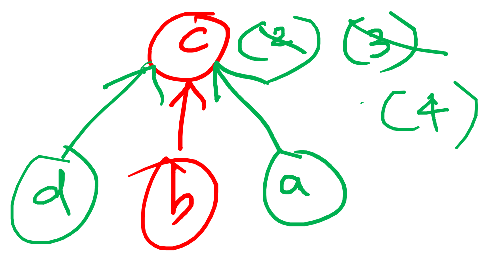


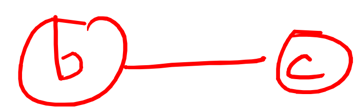




a:a	b:b	c:c	d:d
a:a	b:c	c:c	d:d
a:c	b:c	c:c	d:d
a:c	b:c	c:c	d:c

b-c = 1
a-c = 2
d-c = 3
a-d = 4
a-b = 5





Minimum Spanning Tree
Kruskal's Algorithm

- Here final tree will have n-1 edges
  - So n union and find operations need to be performed- O(n log n)
  - Since graph is connected, number of edges should be atleast n-1 ie m>=n-1
  - So cost of union operations=O(m log m)

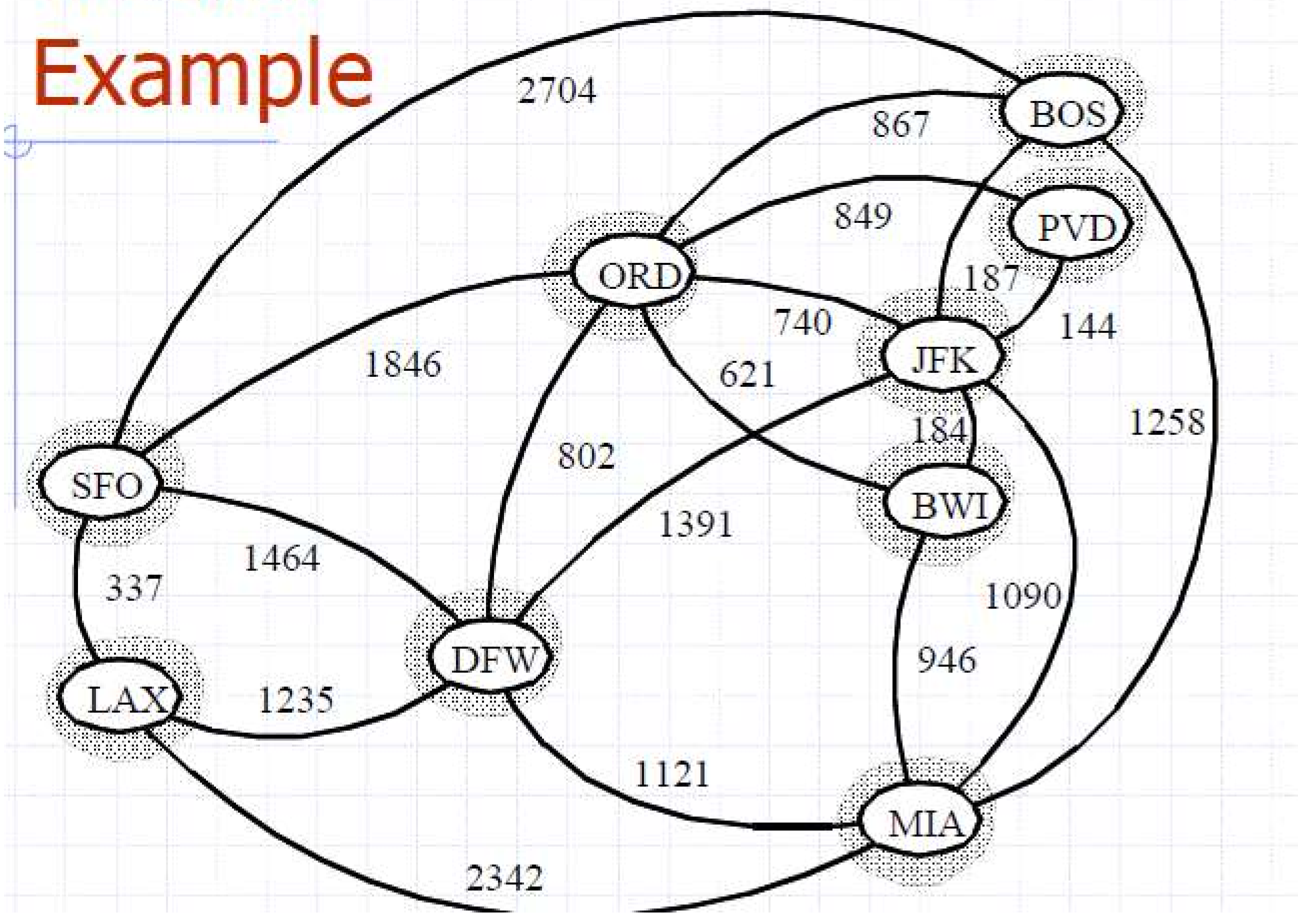
    =O(E log E)
- Thus, the total time spent performing priority queue operations is no more than **O(E log E)**.
  - The number of edges in a simple graph m =n(n–1)/2.
  - **m is atmost n^2** ie **log m =2log n ie lg |E| = O(lg |V|)**
- Thus, the total time spent performing priority queue operations is O(E log V). [Same as Prim's]

Algorithm *KruskalMST*(*G*)
  for each vertex *V* in *G* **do**
    define a *Cloud(v)* of ← {*v*}
  let *Q* be a priority queue.
  Insert all edges into *Q* using their weights as the key
  T ← ∅
  while *T* has fewer than *n*-1 edges **d**
    edge *e* = *T.removeMin()*
    Let *u*, *v* be the endpoints of *e*
    if *Cloud(v)* ≠ *Cloud(u)* **then**
      Add edge *e* to *T*
      Merge *Cloud(v)* and *Cloud(u)*
  return *T*

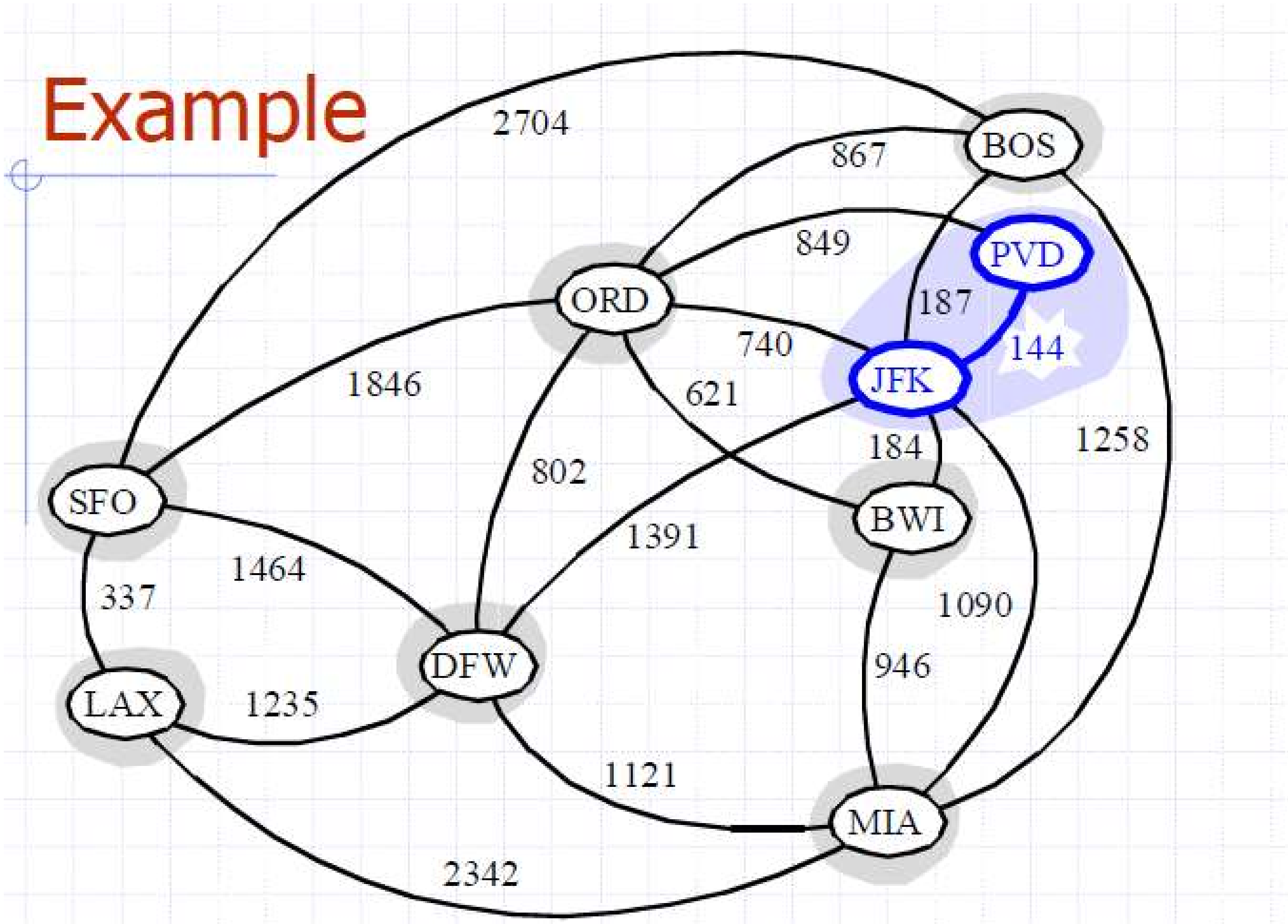
O(n log m)

Minimum Spanning Tree

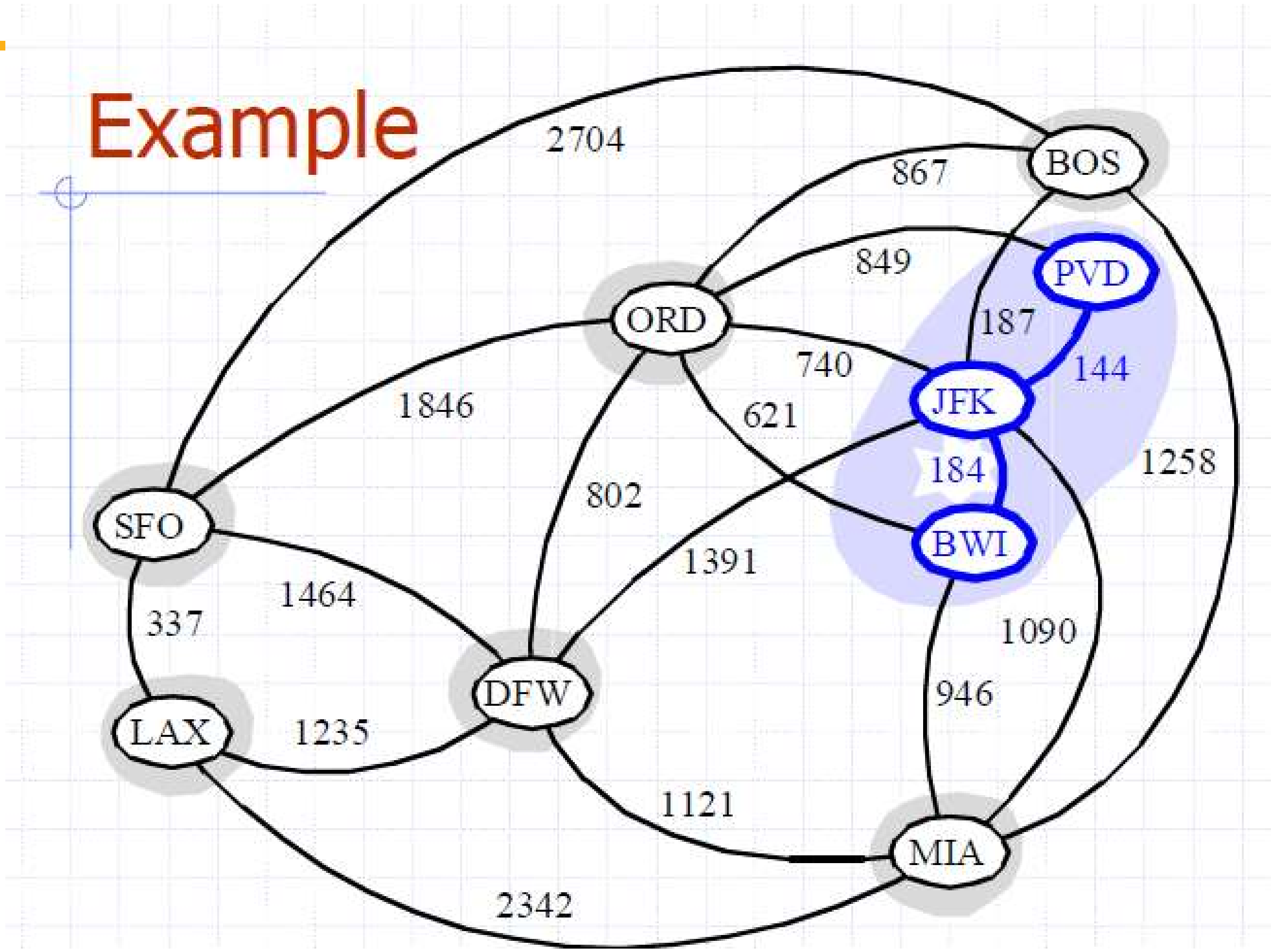
Kruskal Example



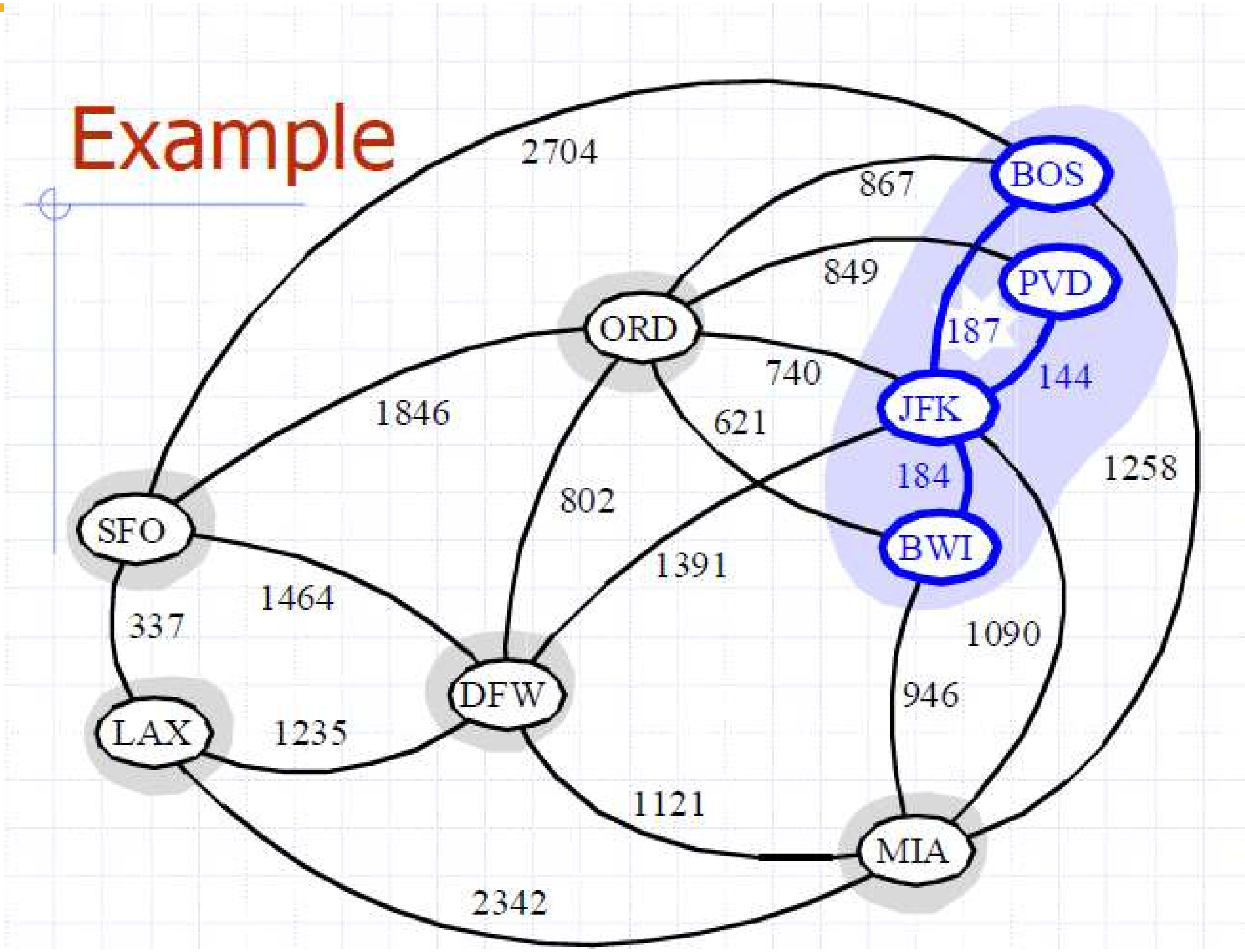
Minimum Spanning Tree



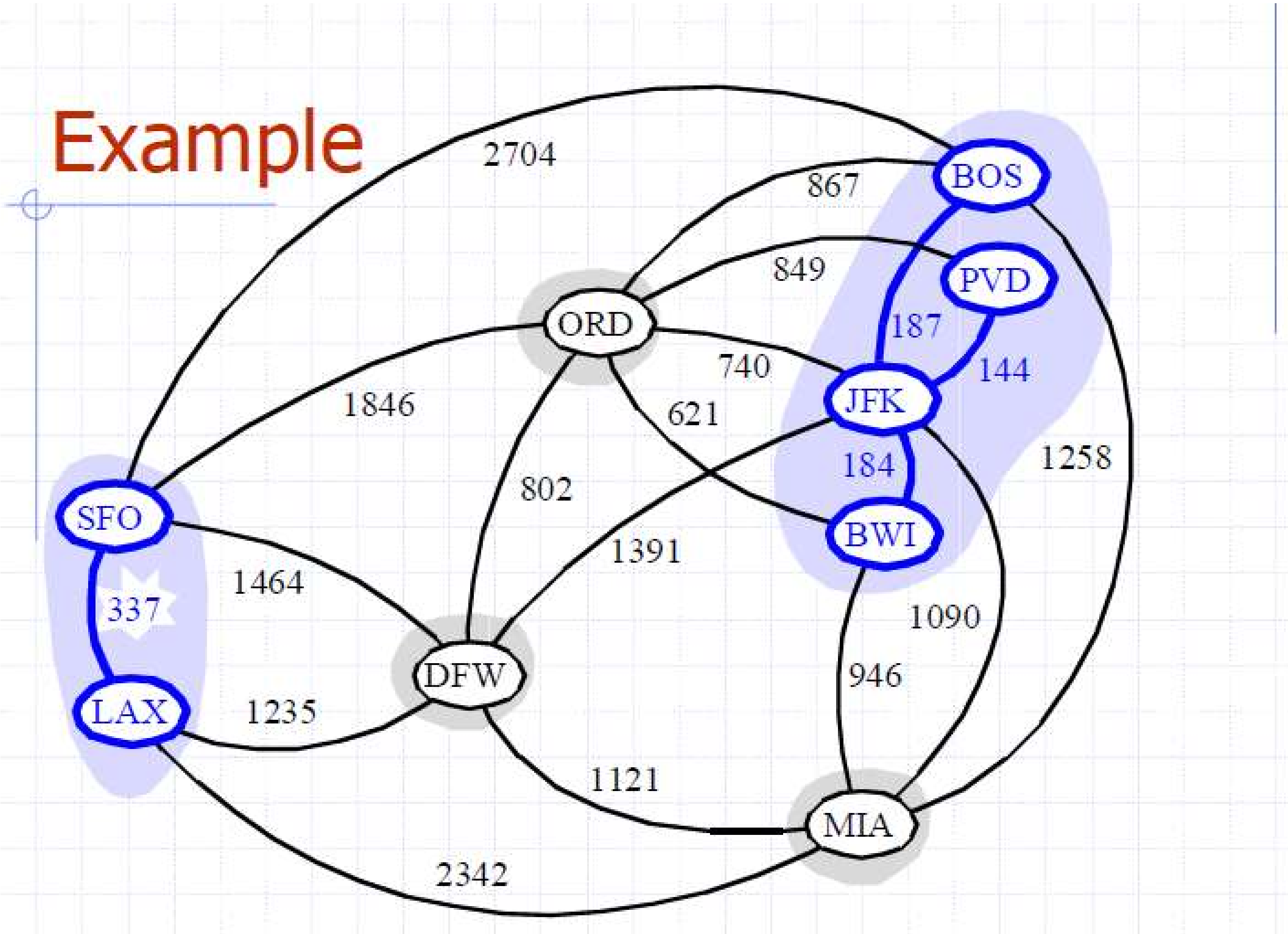
Minimum Spanning Tree



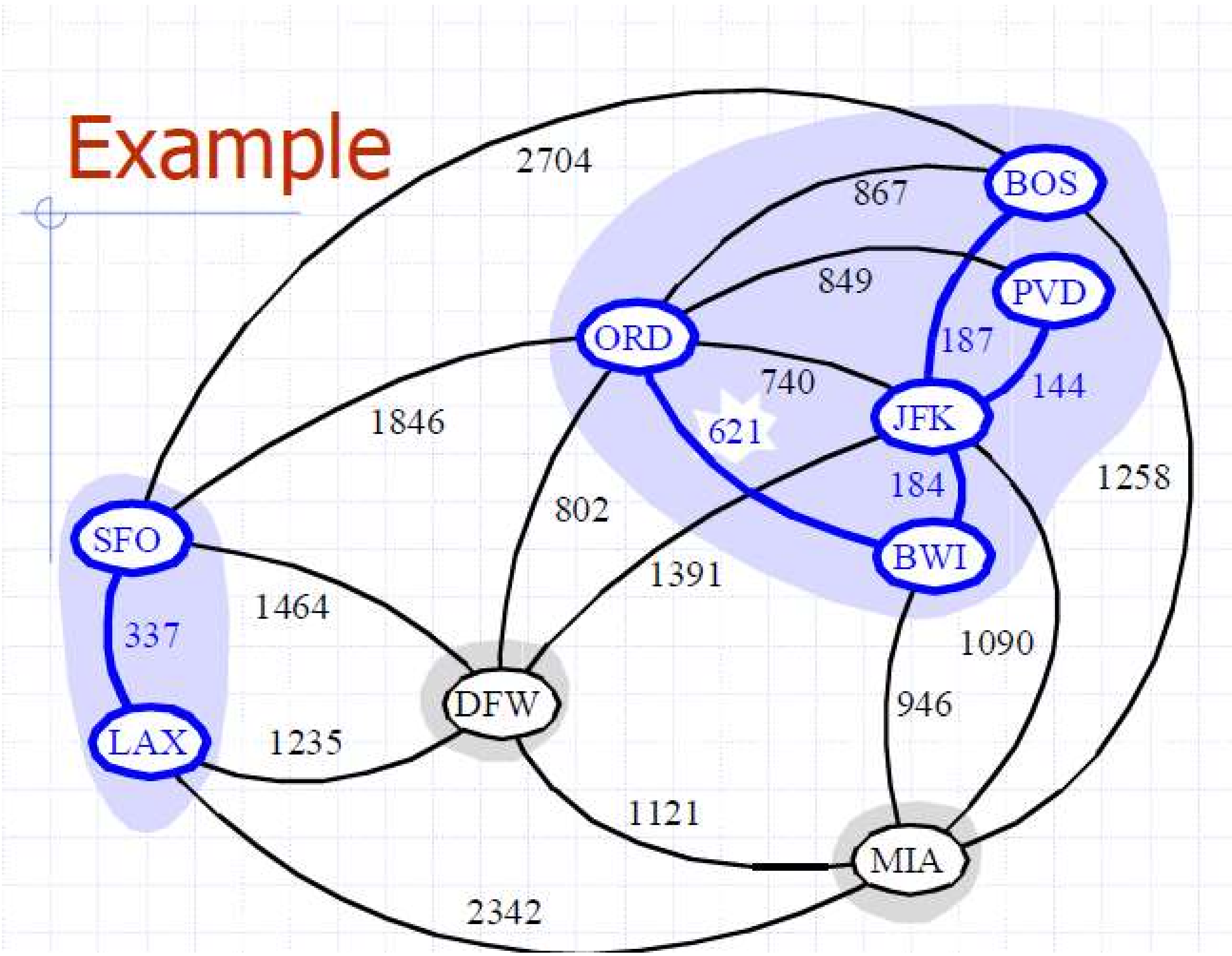
Minimum Spanning Tree



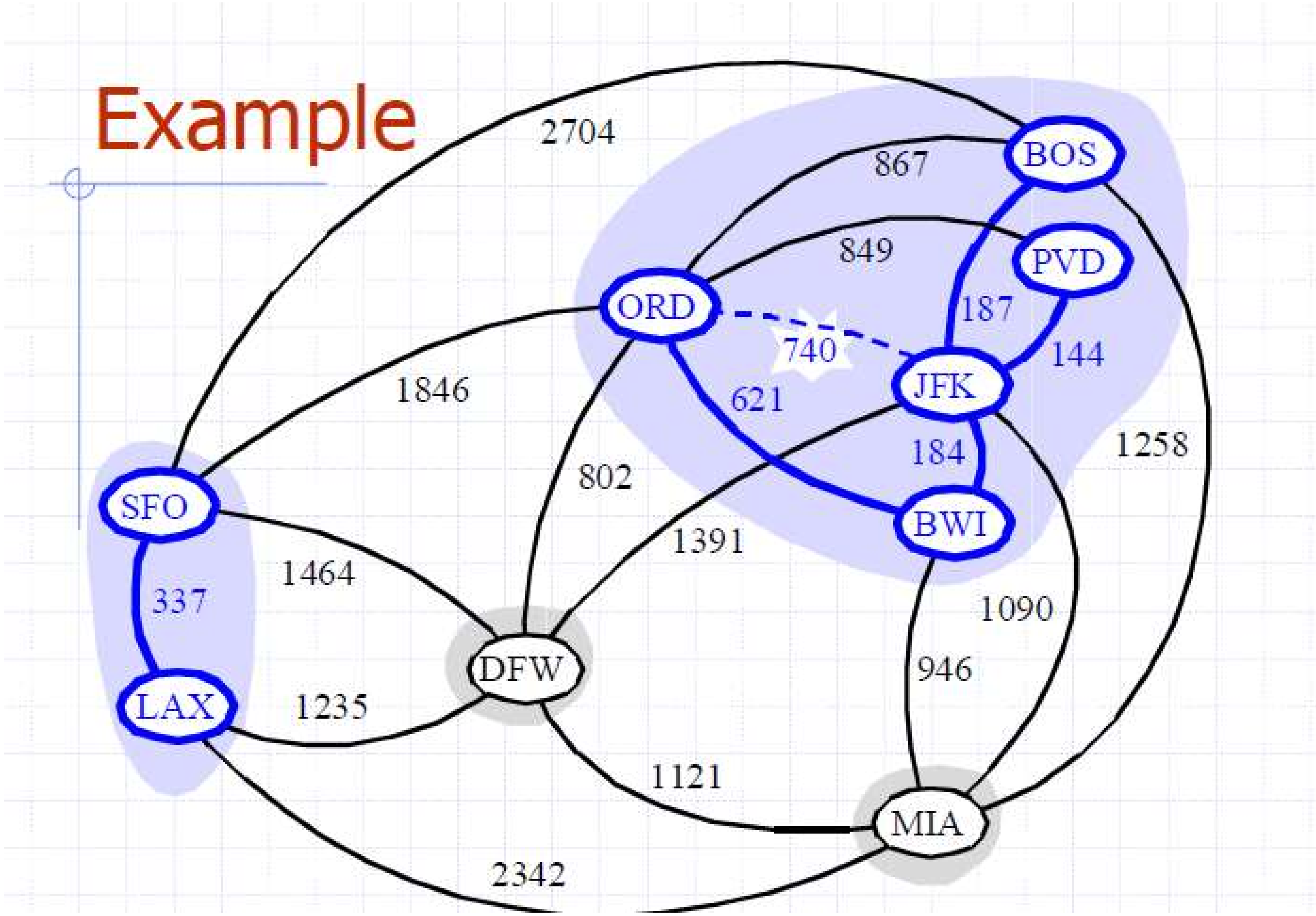
Minimum Spanning Tree



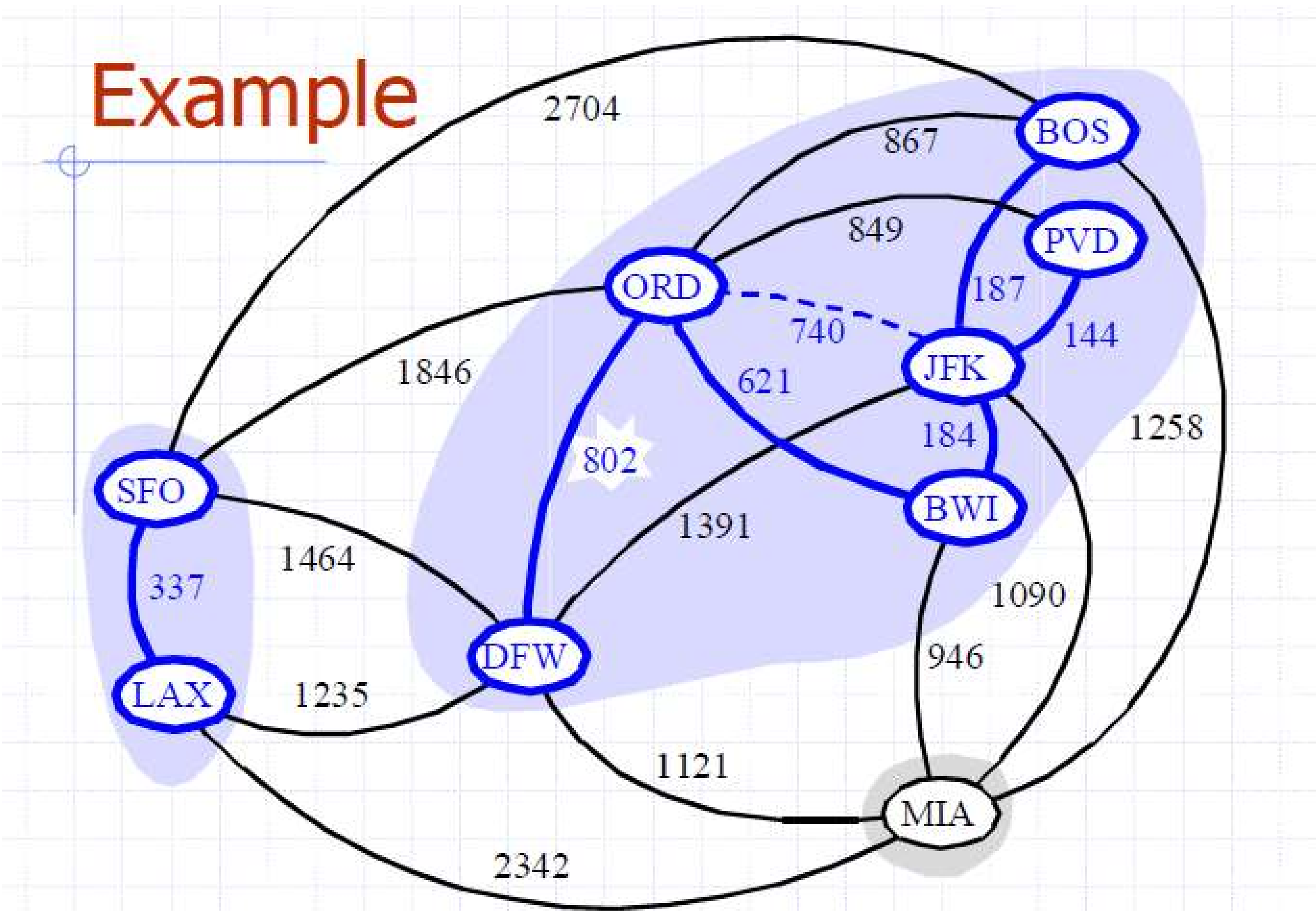
Minimum Spanning Tree



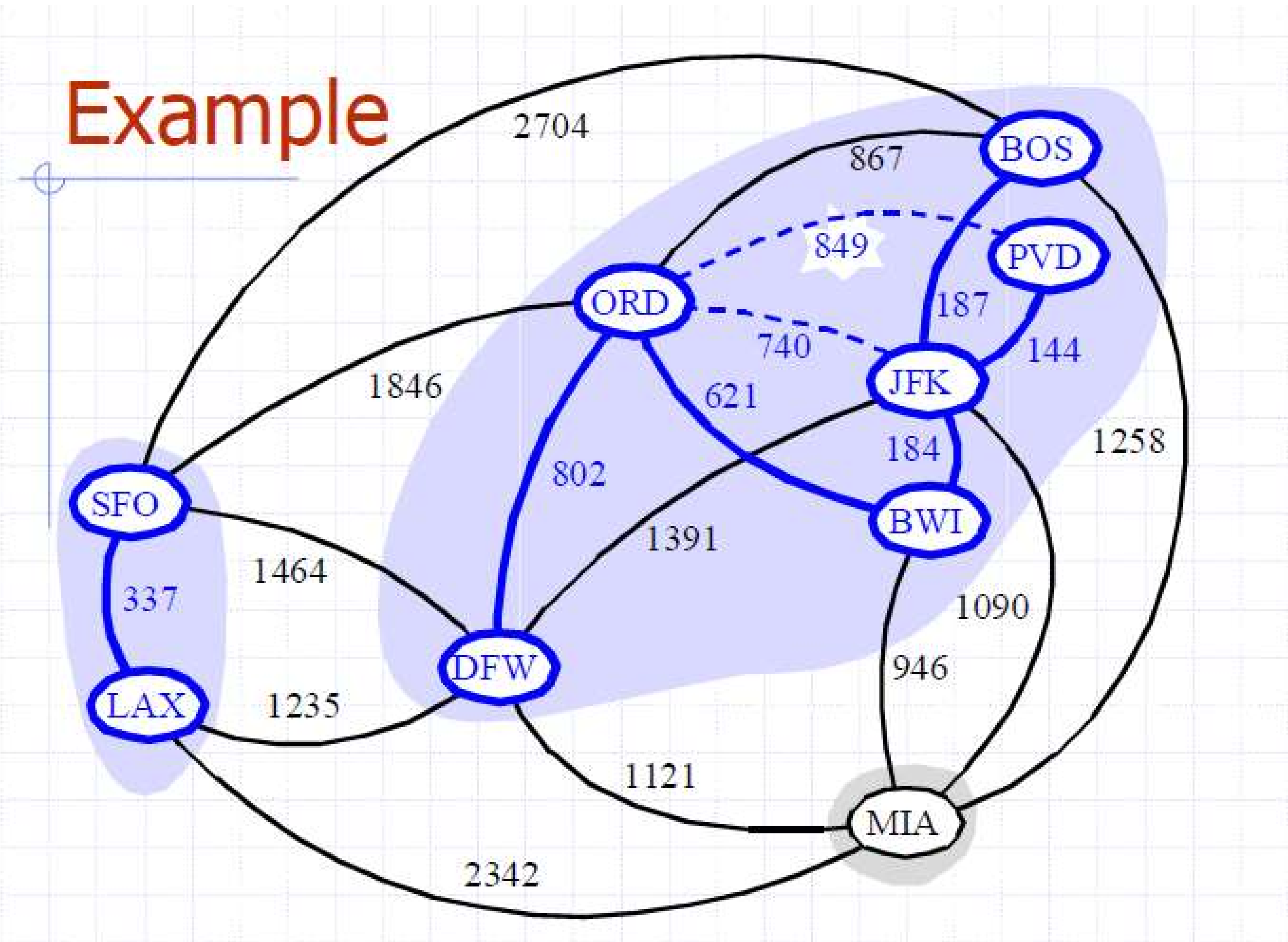
Minimum Spanning Tree



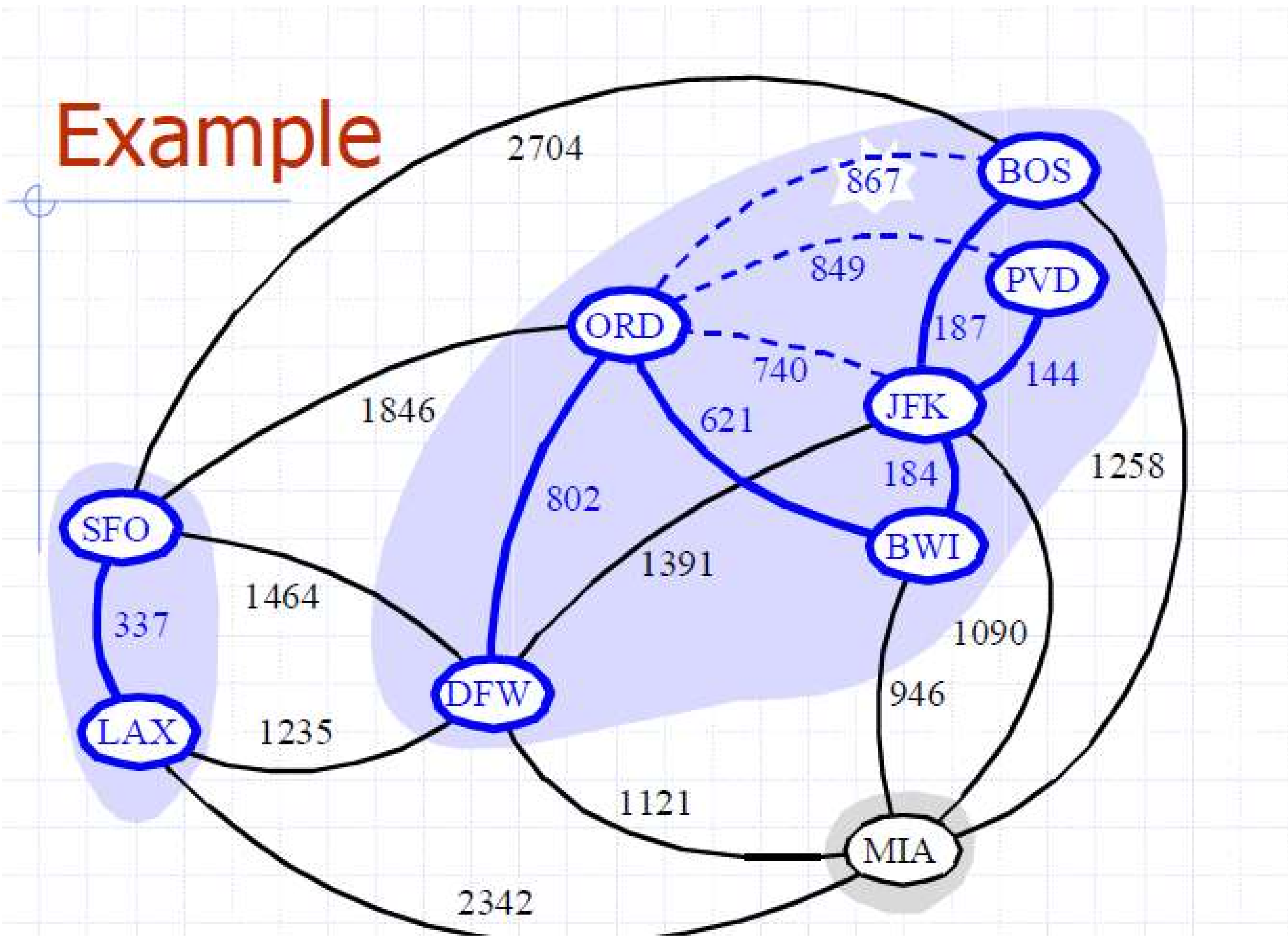
Minimum Spanning Tree



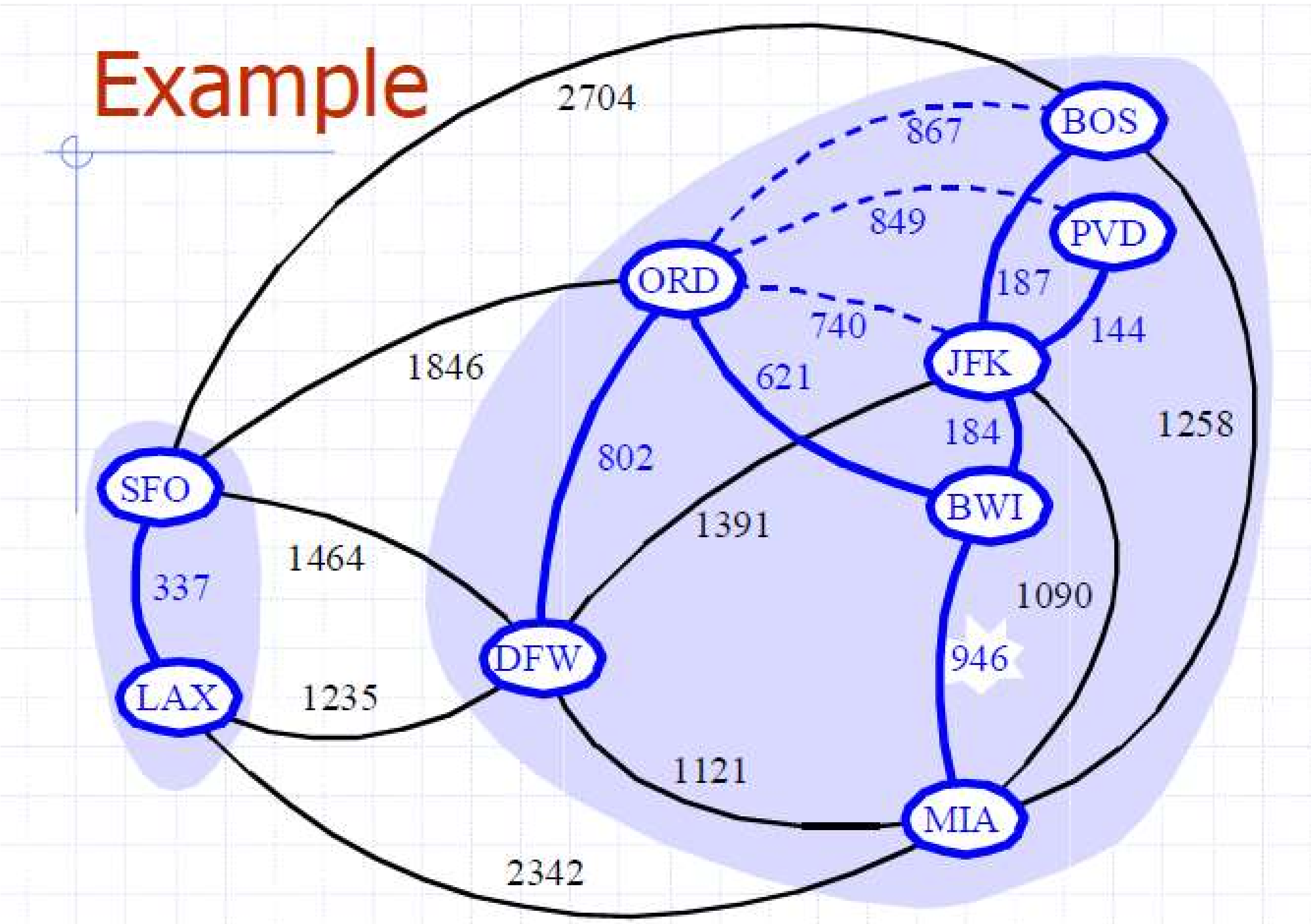
Minimum Spanning Tree



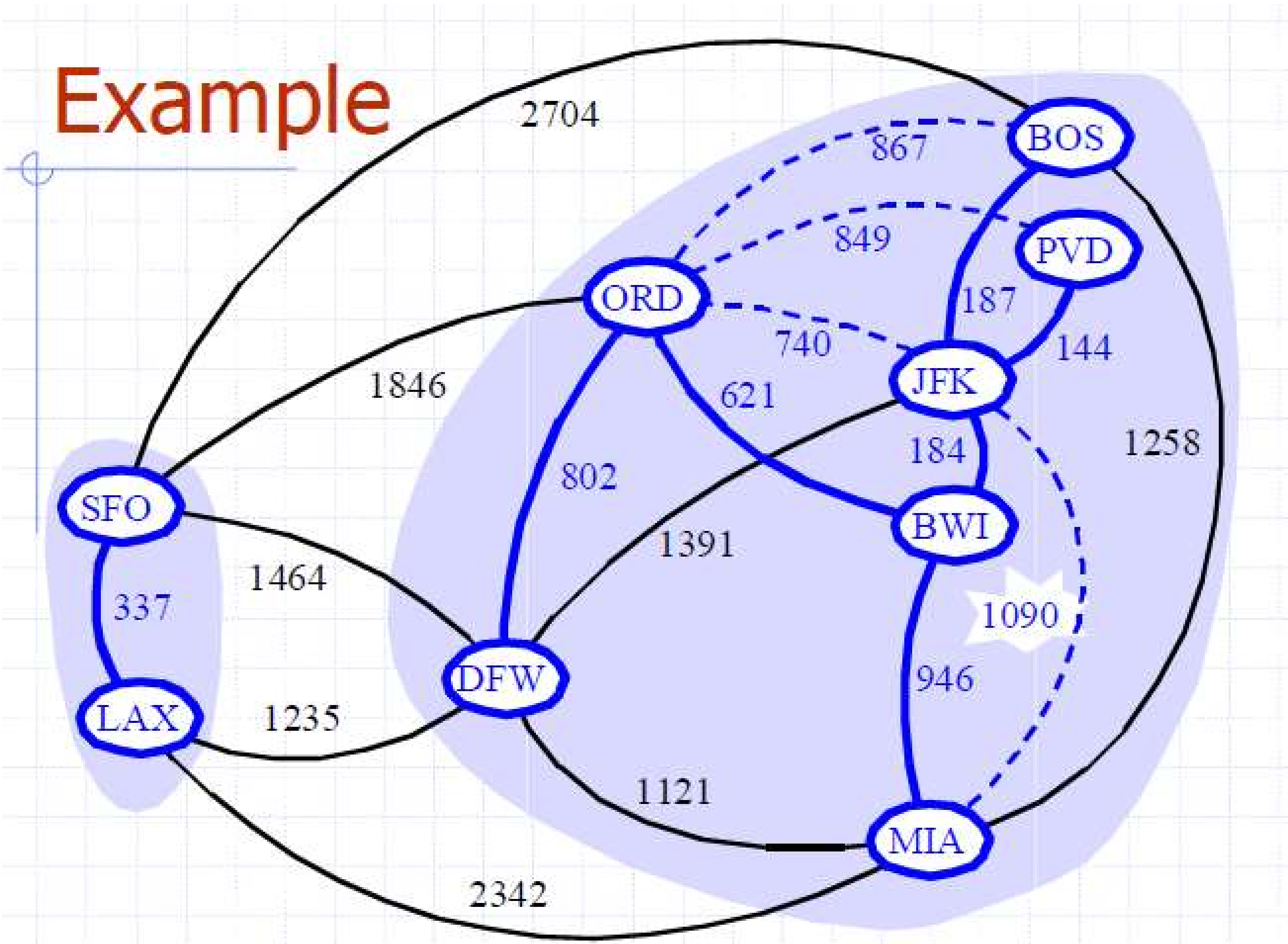
Minimum Spanning Tree



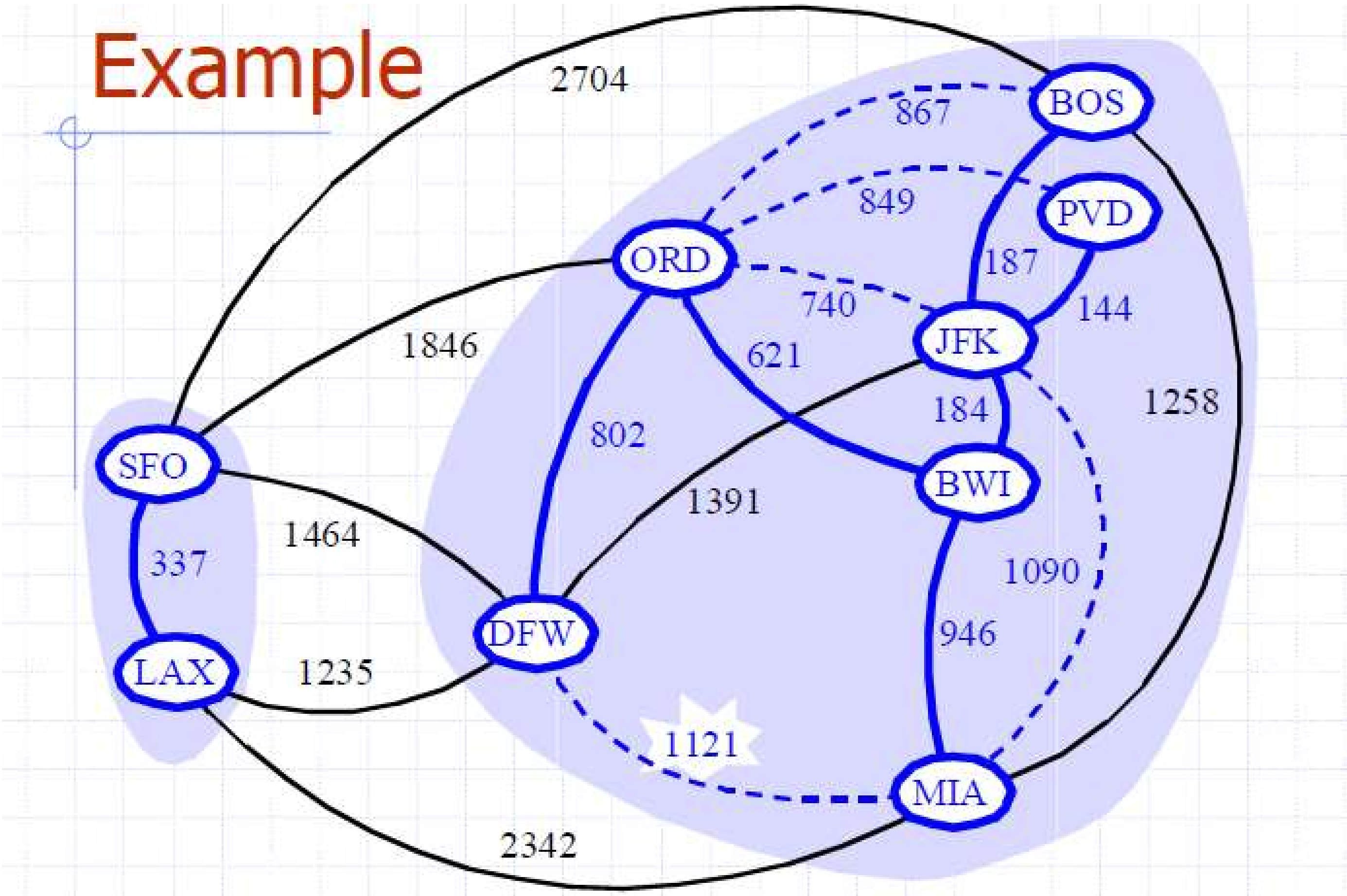
Minimum Spanning Tree



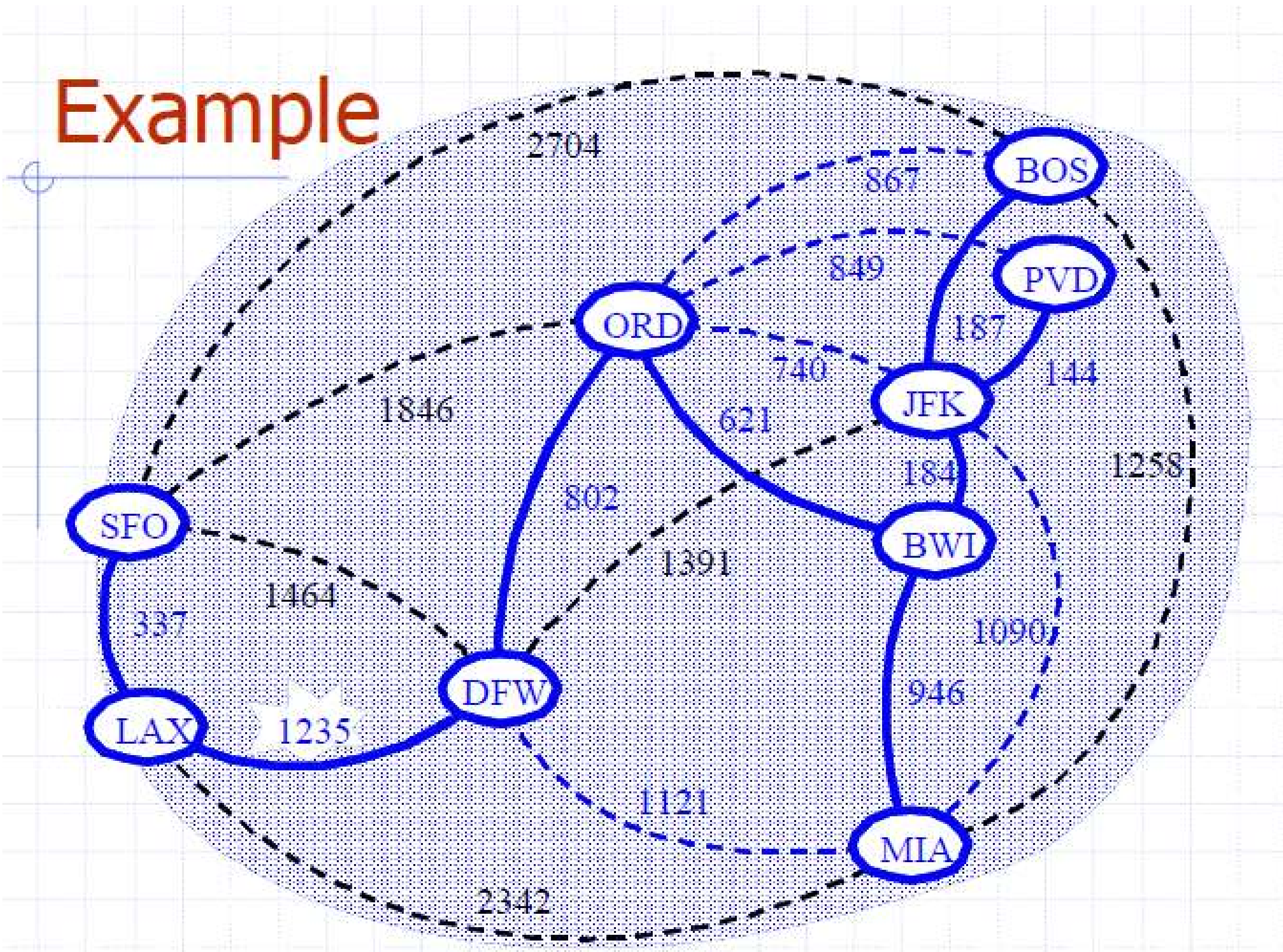
Minimum Spanning Tree



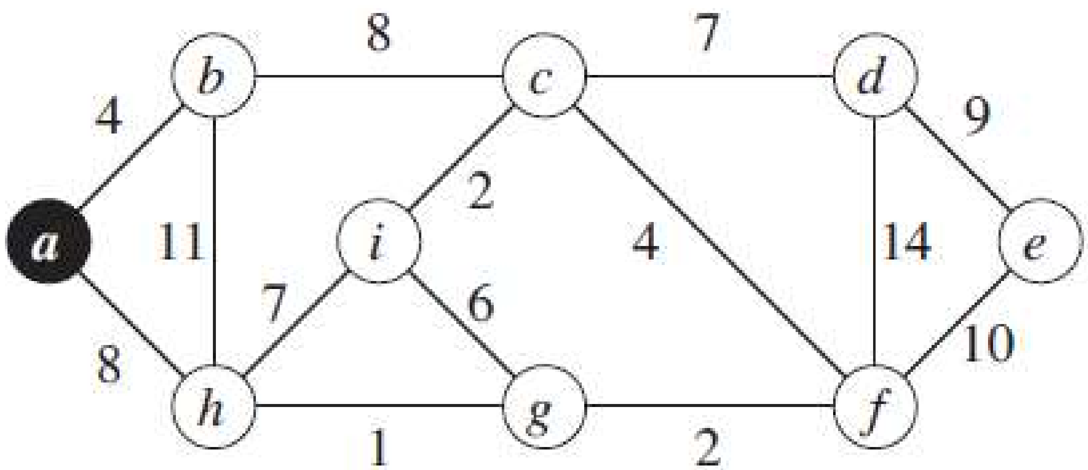
Minimum Spanning Tree



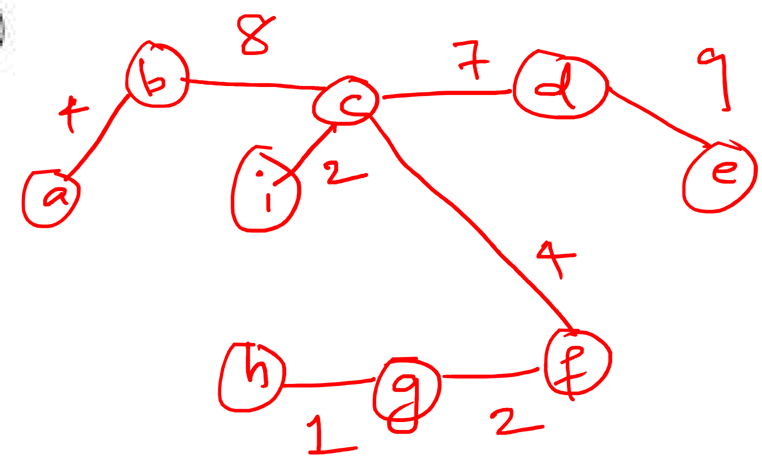
Minimum Spanning Tree



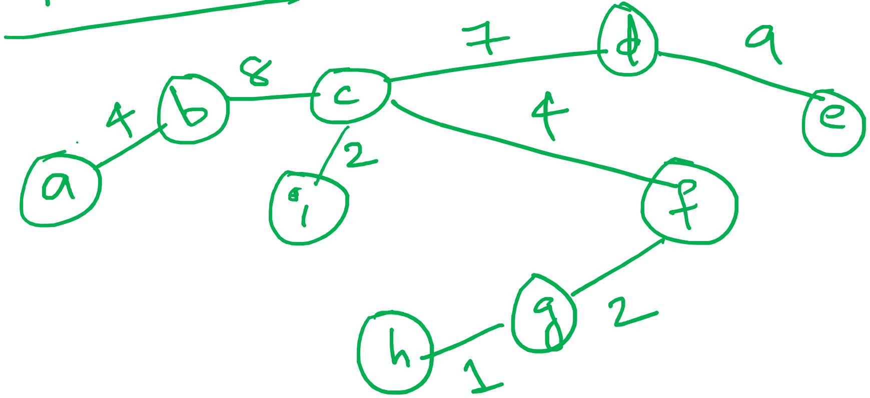
Example1-Prims' & Kruskal's



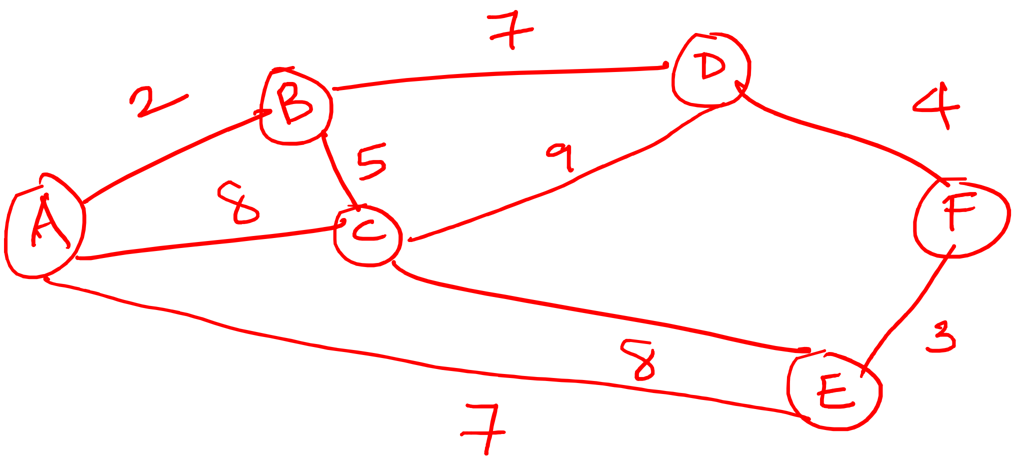
Kruskal's



Prim's



Example2-Prims' & Kruskal's



PRIMS'S Vs KRUSKAL'S ALGORITHM

- **KRUSKAL's**
- Select the shortest edge in a network
- Select the next shortest edge which does not create a cycle
- Repeat step 2 until all vertices have been connected
- Kruskal's begins with forest and merge into tree.
- **PRIM's**
- Select any vertex
- Select the shortest edge connected to that vertex
- Select the shortest edge connected to any vertex already connected
- Prim's always stays as a tree.

PRIMS'S Vs KRUSKAL'S ALGORITHM

- **KRUSKAL's**
- Can be used when graph is sparse(less edges)
- **PRIM's**
- Can be used if the graph has more edges.

MST-Applications

- **Network design.**

– *telephone, electrical, hydraulic, TV cable, computer, road* The standard application is to a problem like phone network design. You have a business with several offices; you want to lease phone lines to connect them up with each other; and the phone company charges different amounts of money to connect different pairs of cities. You want a set of lines that connects all your offices with a minimum total cost. It should be a spanning tree, since if a network isn't a tree you can always remove some edges and save money.

MST-Applications

- **Cluster analysis**
  k clustering problem can be viewed as finding an MST and deleting the k-1 most expensive edges.
- **Image registration and segmentation**
  - **Image segmentation** is the process of portioning image to components and its purpose is to decompose an image to significant and convenient regions and also extract a specific object from image: Image segmentation strives to partition a digital image into regions of pixels with similar properties, e.g. homogeneity.
  - **Image registration** is the process of transforming different sets of data into one coordinate system. Data may be multiple photographs, data from different sensors, times, depths, or viewpoints. Registration is necessary in order to be able to compare or integrate the data obtained from these different measurements.

MST-Applications

- **Taxonomy**:
  - Taxonomy is the practice and science of classification
- **Feature extraction**
  - In machine learning, pattern recognition and in image processing, feature extraction starts from an initial set of measured data and builds derived values (features) intended to be informative and non-redundant, facilitating the subsequent learning and generalization steps, and in some cases leading to better human interpretations
  - When the input data to an algorithm is too large to be processed and it is suspected to be redundant (e.g. the same measurement in both feet and meters, or the repetitiveness of images presented as pixels), then it can be transformed into a reduced set of features (also named a feature vector). Determining a subset of the initial features is called feature selection.

MST-Applications

- Regionalization of socio-geographic areas, the grouping of areas into homogeneous, contiguous regions.
- Comparing ecotoxicology data.:
  - Ecotoxicology is the study of the effects of toxic chemicals on biological organisms, especially at the population, community, ecosystem, and biosphere levels.
- Topological observability in power systems.
- Measuring homogeneity of two-dimensional materials. [Link]

MST-Problem -HW

MST QN