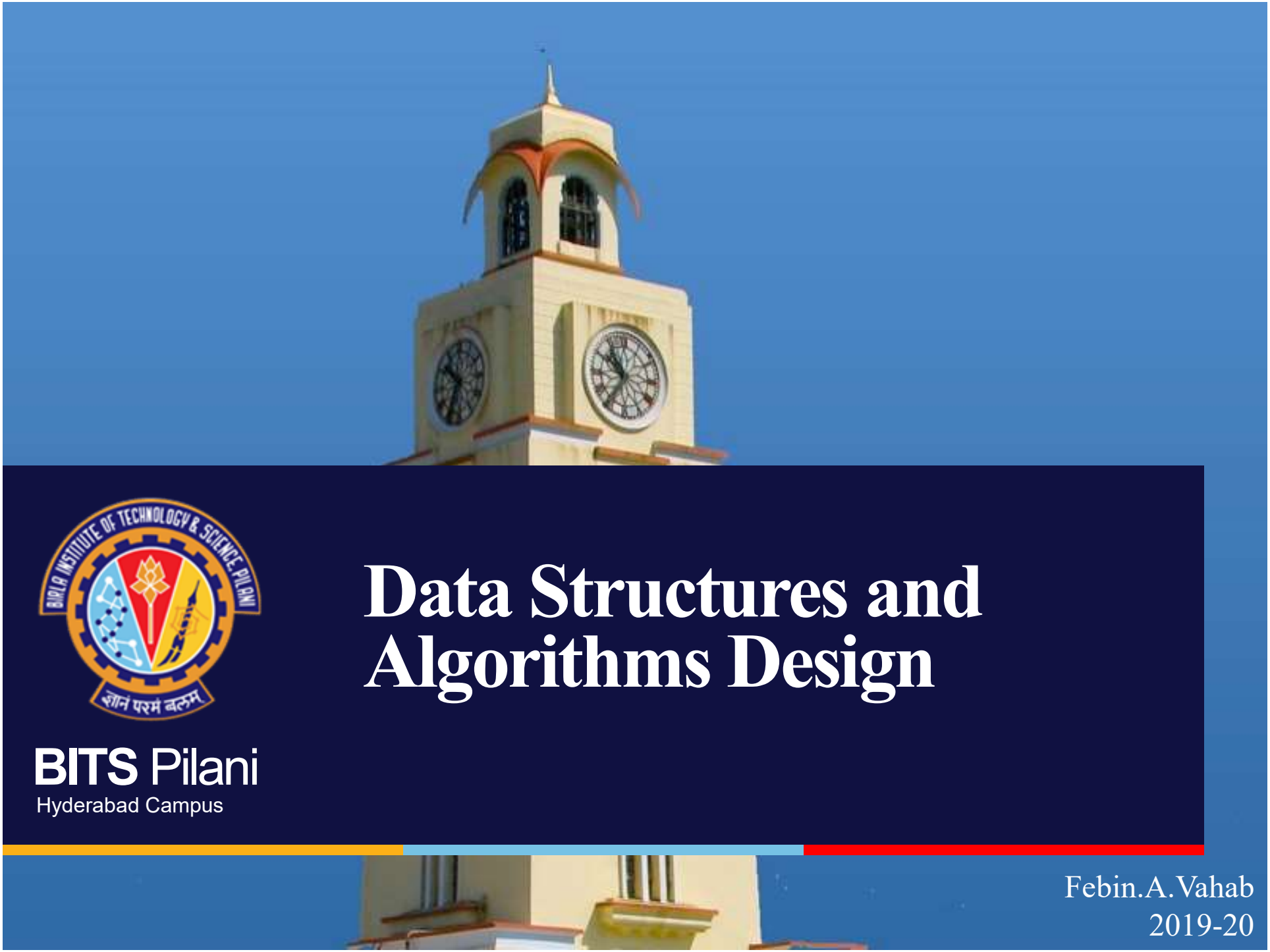


Data Structures and Algorithms Design

BITS Pilani
Hyderabad Campus

Febin.A.Vahab
2019-20

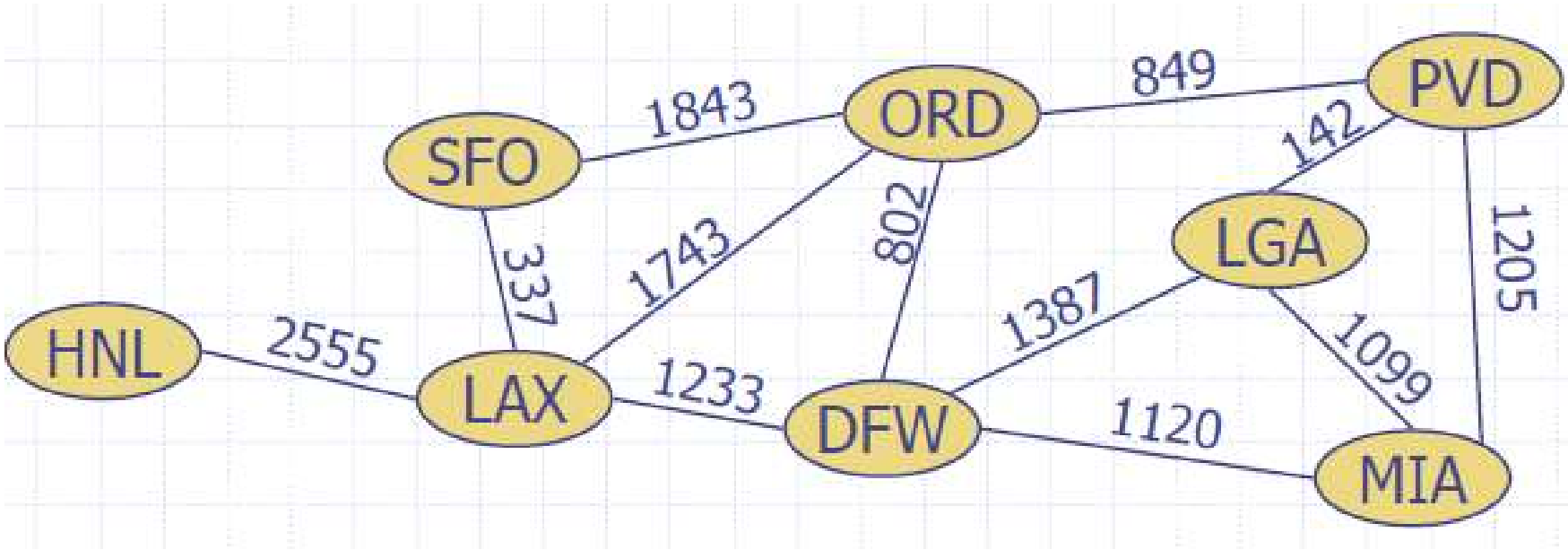
ONLINE SESSION 11 -PLAN

Sessions(#)	List of Topic Title	Text/Ref Book/external resource
11	**M5:Algorithm Design Techniques** Greedy Method - Task Scheduling Problem Minimum Spanning Tree, Shortest Path Problem - Dijkstra's Algorithm	T1: 5.1, 7.1, 7.3

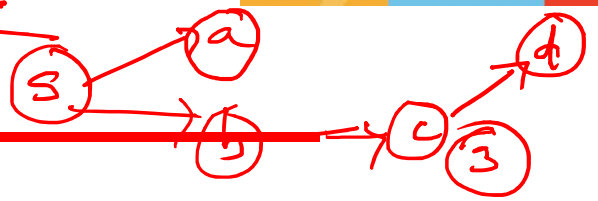
Weighted Graphs

- In a weighted graph, each edge has an associated numerical value, called the weight of the edge
- Edge weights may represent, distances, costs, etc.
- Example: A flight route graph,the weight of an edge represents the distance in miles between the endpoint airports.

Weighted Graphs



Dijkstra's Algorithm-Single source shortest path algorithm



- The ***distance*** of a vertex ***v*** from a vertex ***s*** is the length of a shortest path between ***s*** and ***v***
- Dijkstra's algorithm computes the ***distances*** of all the vertices from a given start vertex ***s***
- Assumptions:
  - The graph is connected
  - The edge weights are nonnegative

Dijkstra's Algorithm

- We grow a "**cloud**" of vertices,beginning with ***s*** and eventually covering all the vertices
- We store with each vertex ***v*** a label ***v.d*** representing the distance of ***v*** from ***s*** in the subgraph consisting of the cloud and its adjacent vertices

Dijkstra's Algorithm

- At each step
  - We add to the cloud the vertex ***u*** outside the cloud with the smallest distance label, ***u.d***- [Greedy choice]
  - We update the labels of the vertices adjacent to ***u***

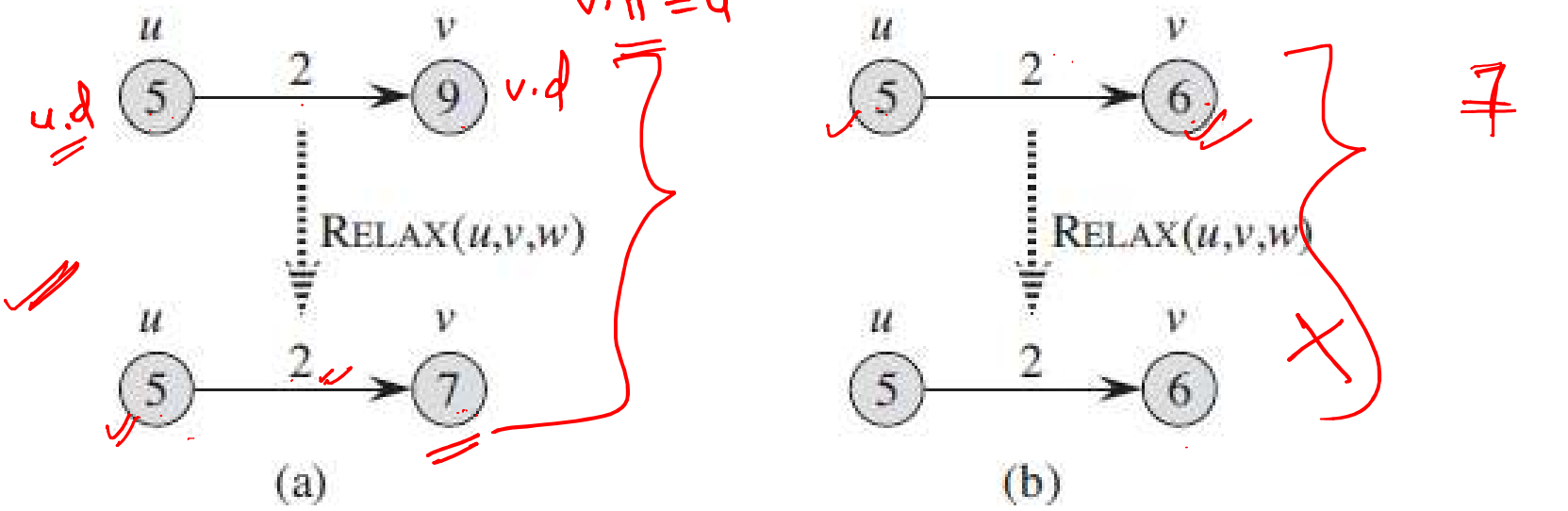
Dijkstra's Algorithm

Edge Relaxation


```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

predecessor

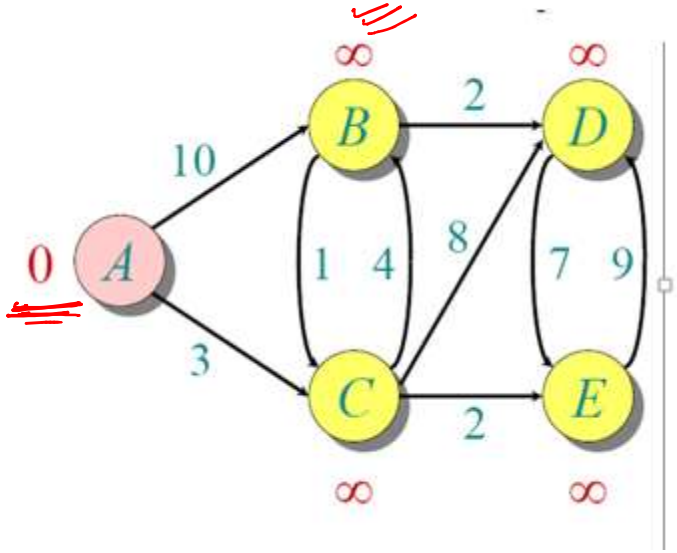
Dijkstra's Algorithm



- Relaxing an edge (u,v)with weight =2. The shortest-path estimate of each vertex appears within the vertex.
  a) Because v.*d* > u.*d* + w(u,v) prior to relaxation, the value of v,*d* decreases.
  b) Here, v.d < u.*d* + w(u,v) before relaxing the edge, and so the relaxation step leaves v.*d* unchanged.

Dijkstra's Algorithm-{Cormen}

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ G.V
2      v.d = ∞
3      v.π = NIL
4  s.d = 0
```



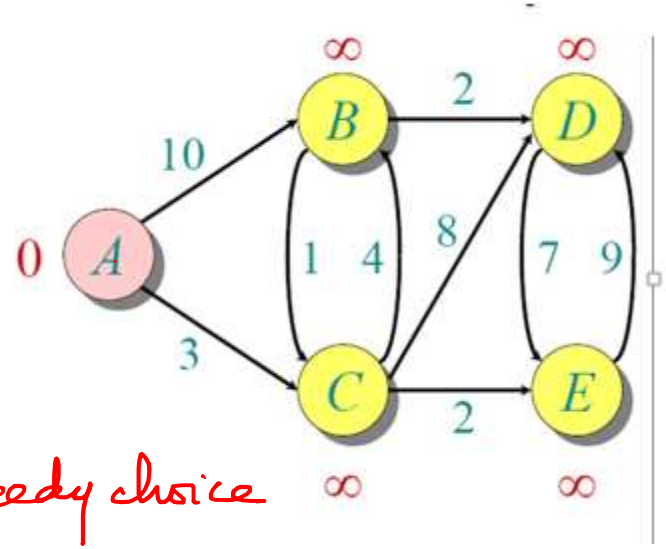

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

A.π = Nil
B.π = Nil
C.π = Nil
D.π = Nil
E.π = Nil

Dijkstra's Algorithm-{Cormen}

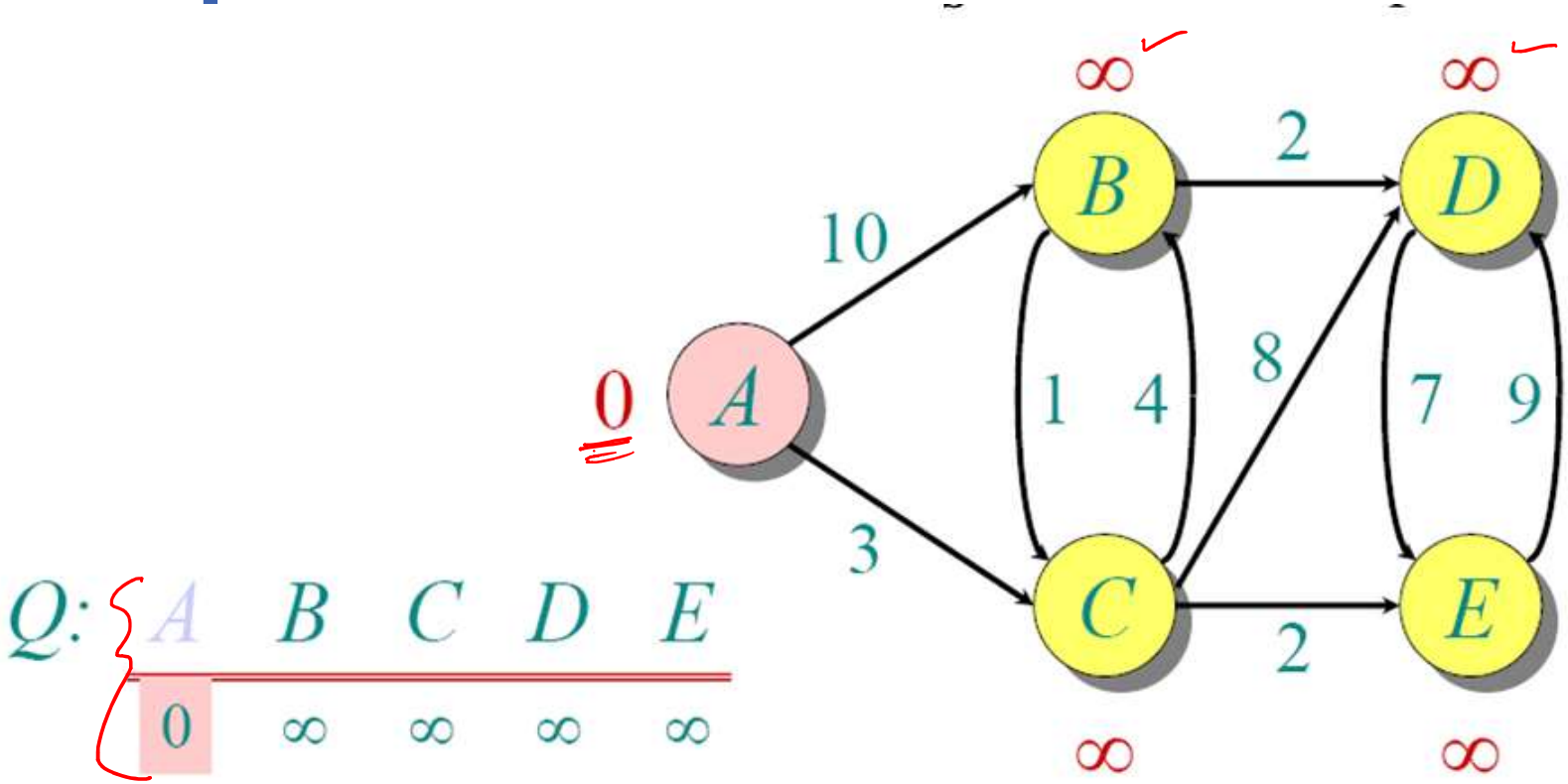
```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G,
2  S = ∅
3  Q = G.V
4  while Q ≠ ∅
5      u = EXTRACT-MIN(Q)
6      S = S ∪ {u}
7      for each vertex v ∈ G.Adj[u]
8          RELAX(u, v, w)
```

greedy choice



Q:	A	B	C	D	E
	0	∞	∞	∞	∞

Example



A ⇒ AB ⇒ if $B.d > A.d + w(A,B)$

$B.d = A.d + w(A,B)$

$B.\pi = A$

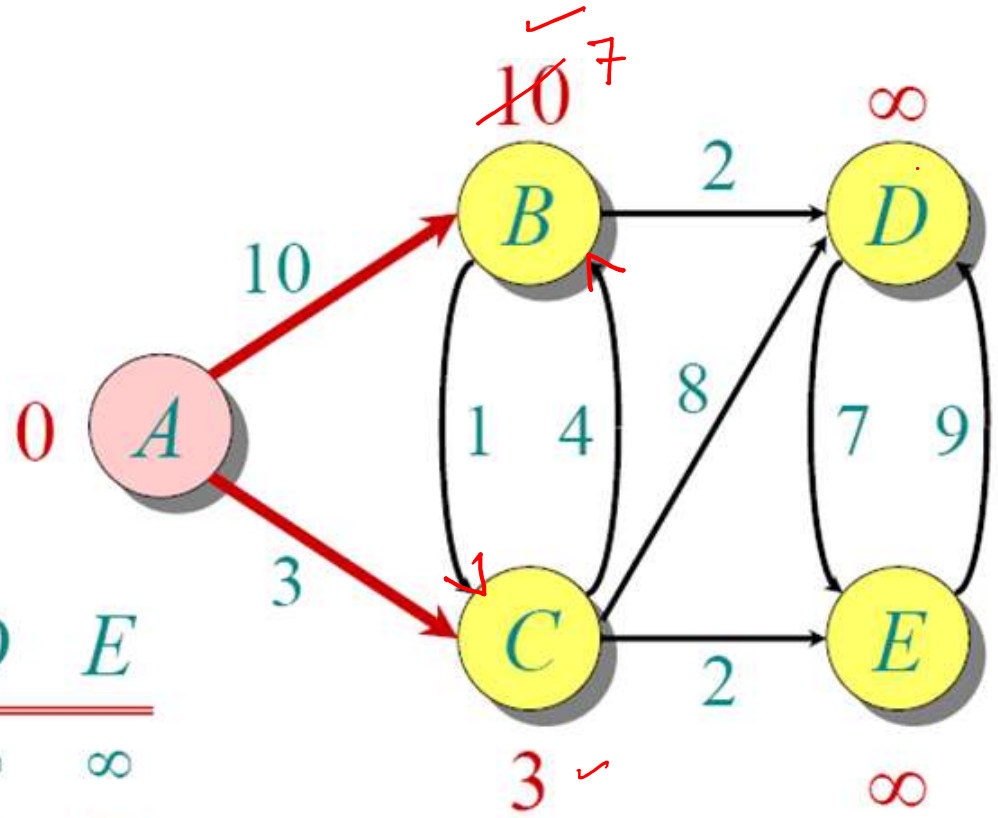
AC ⇒ $C.d = 3$, $C.\pi = A$

if $\infty > 0+10$

$B.d = 10$

$B.\pi = A$

Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
d		10	3	∞	∞

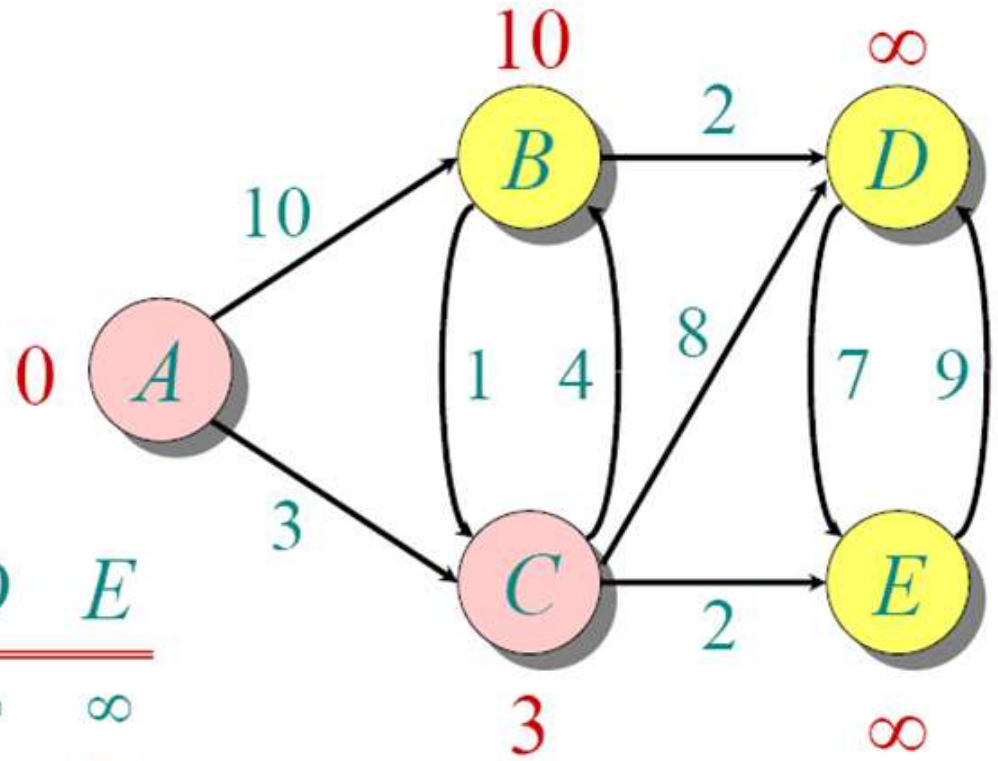
$S: \{ A \}$

C ⇒ $CE = E.d = 5,\ E.\pi = C$

$CB = B.d = 7,\ B.\pi = C$

$CD = D.d = 11,\ D.\pi = C$

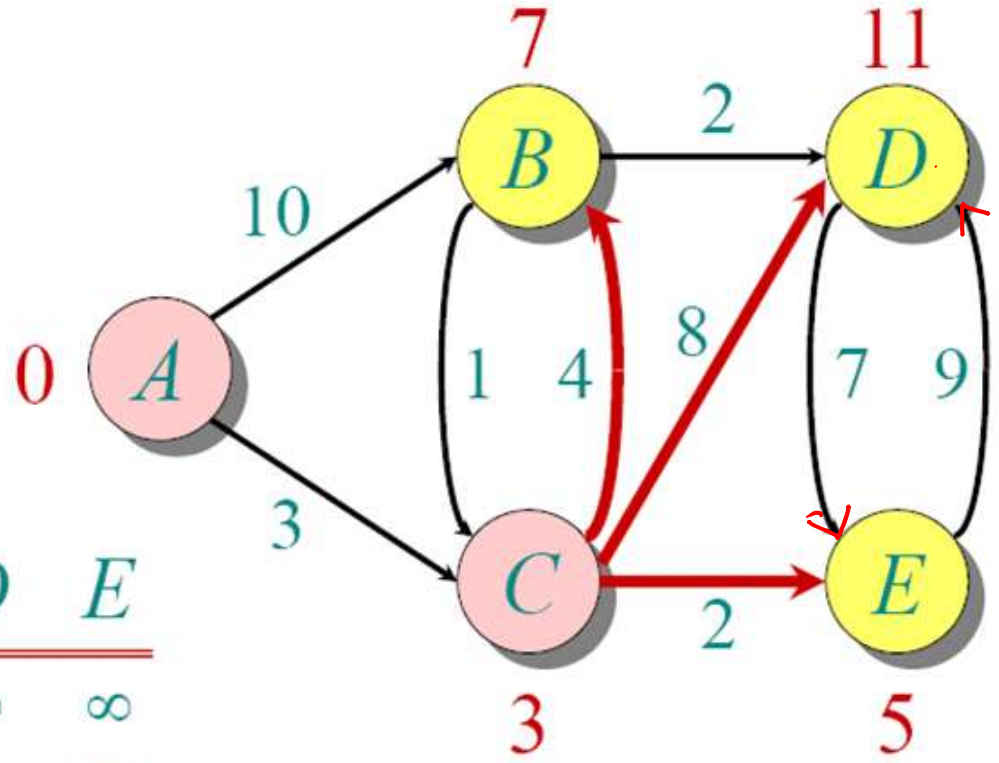
Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A, C }

Example

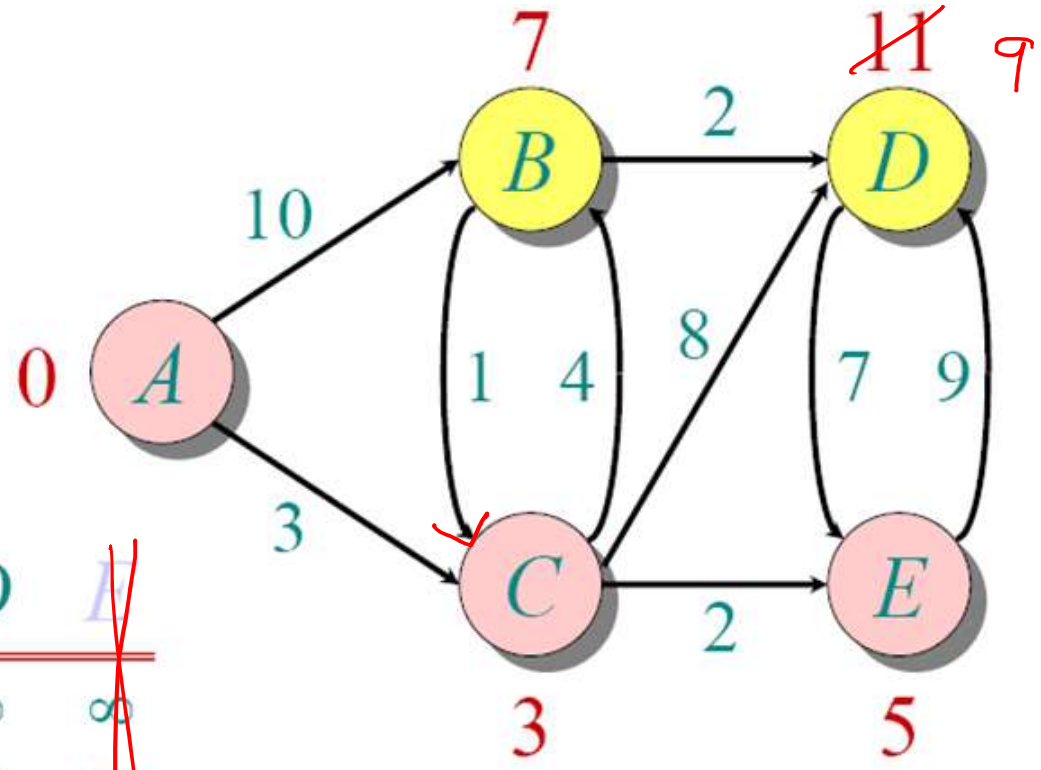


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

$S: \{ A, C \}$

E ⇒ ED = No relation

Example

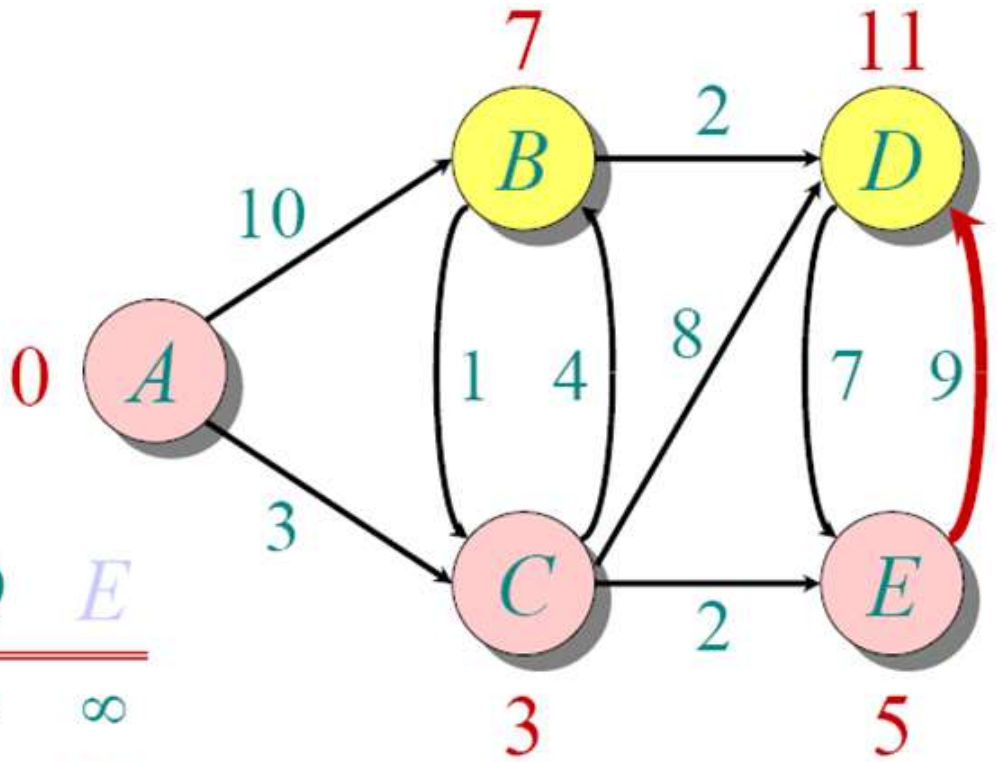


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

$S: \{ A, C, E \}$

B ⇒ BC = No Relaxation
⇒ BD = D.d = 9, D.π = B

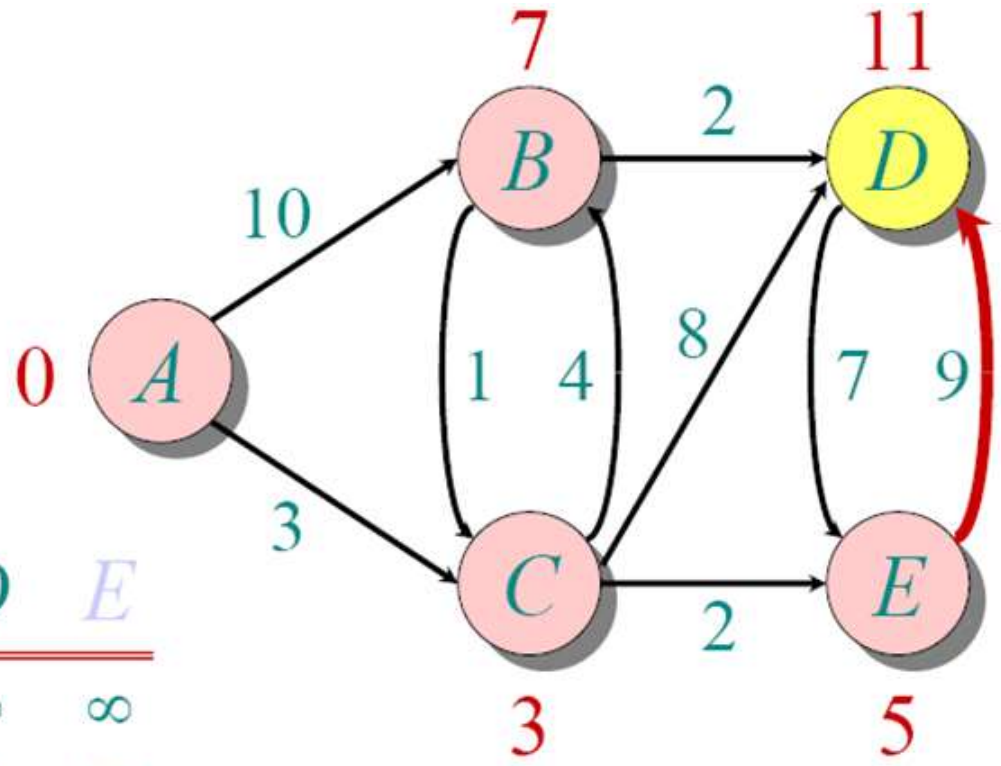
Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { A, C, E }

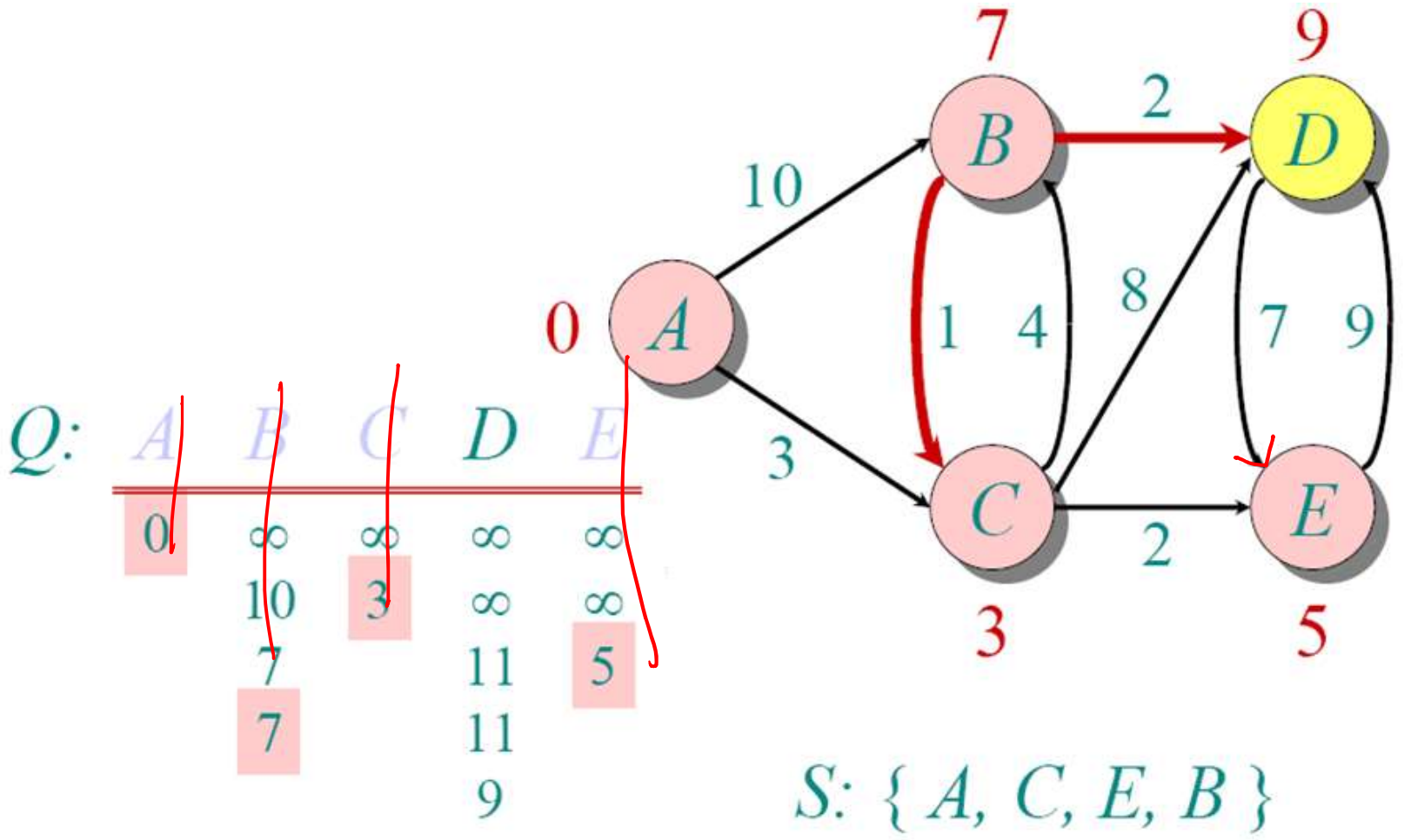
Example



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { A, C, E, B }

Example



Q:

A	B	C	D	E
0	∞	∞	∞	∞
	10	3	∞	∞
	7		11	5
	7		11	
			9	

D=DE ⇒ No relaxation

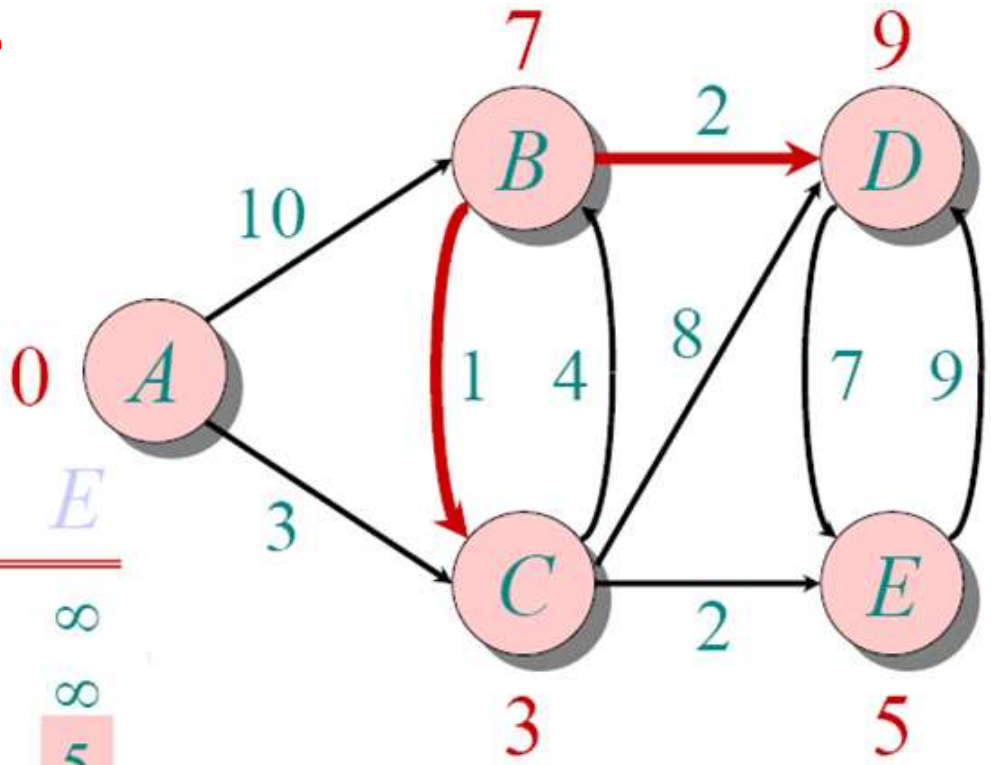
Example

A→B = A→C→B

A→C = A→C

A→D = A→C→B→D

A→E = A→C→E



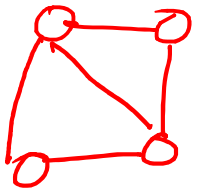
Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

S: { A, C, E, B, D }

Dijkstra's Algorithm-Analysis

- Dijkstra's 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|)**

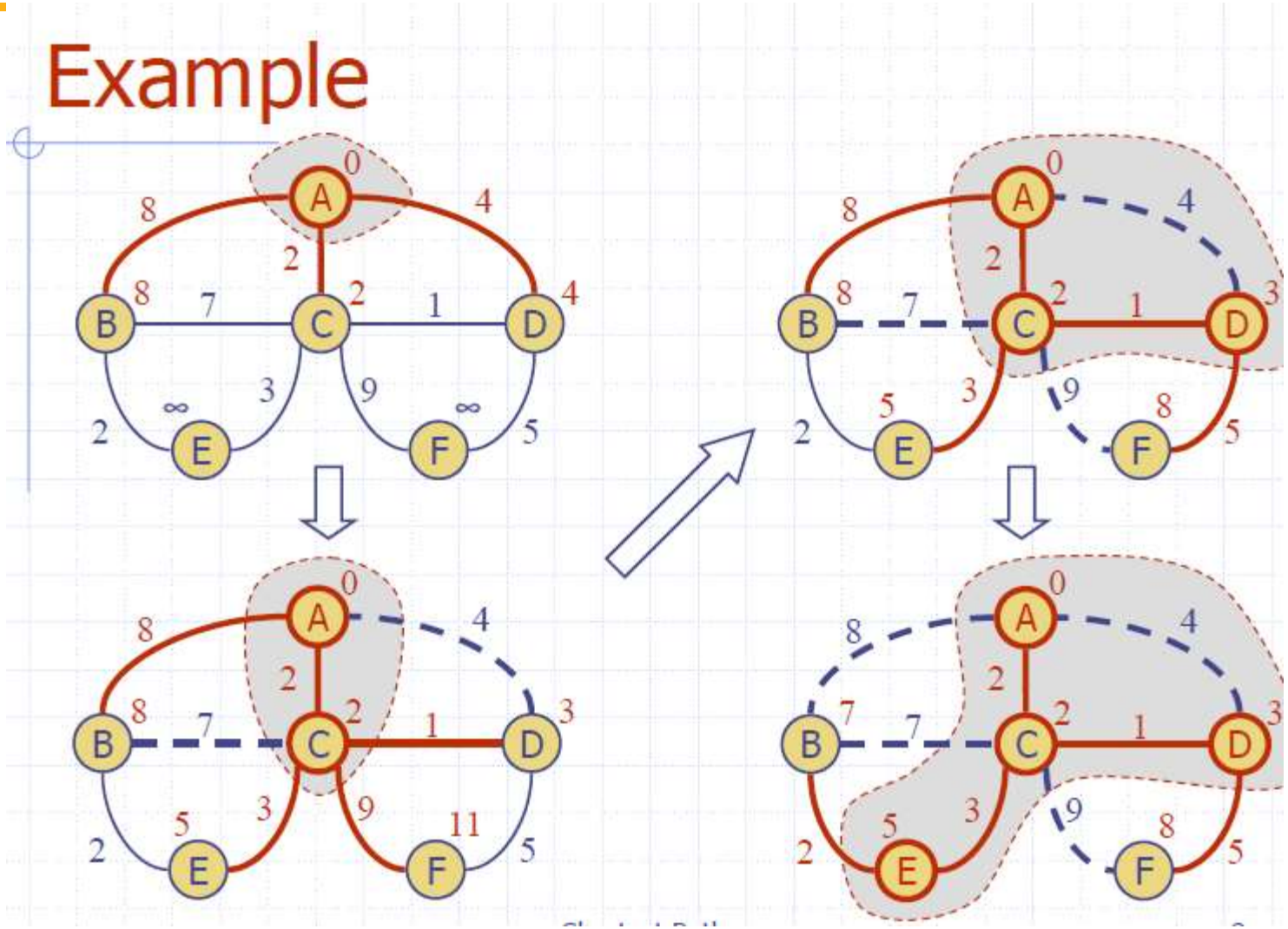
```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G
2  S = ∅
3  Q = G.V
4  while Q ≠ ∅
5      u = EXTRACT-MIN(Q)
6      S = S ∪ {u}
7      for each vertex v ∈ G.Adj[u]
8          RELAX(u, v, w)
```



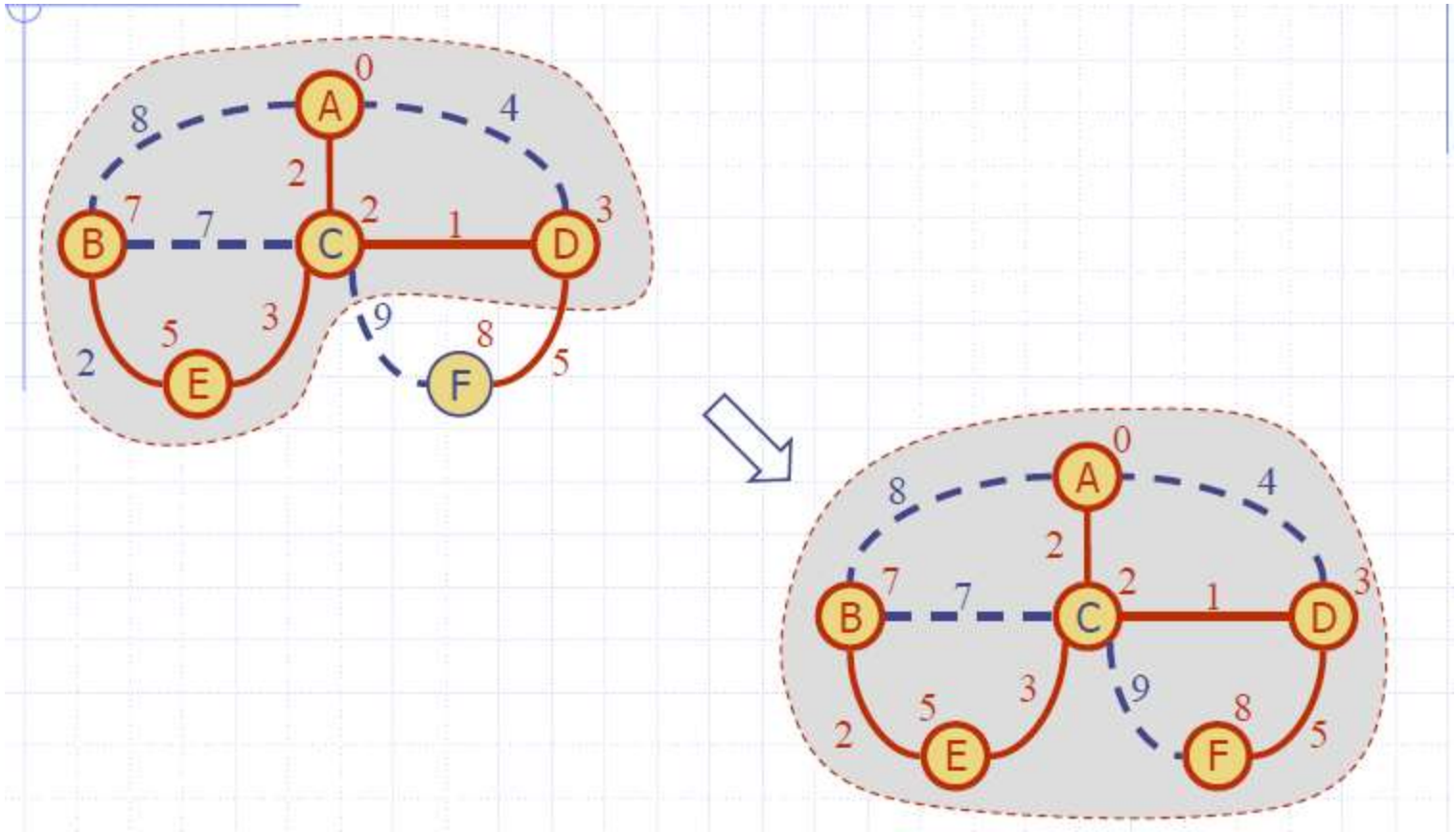
Dijkstra's Algorithm-Analysis

- The simplest implementation of the Dijkstra's algorithm stores vertices in an ordinary linked list or array
- Good for dense graphs (many edges)
  - |V| vertices and |E| edges
  - Initialization O(|V|)
  - While loop O(|V|)
  - Find and remove min distance vertices O(|V|)
  - Potentially |E| updates
  - Update costs O(1)
  - Total time $O(|V^2| + |E|) = O(|V^2|)$

Dijkstra's Algorithm



Dijkstra's Algorithm

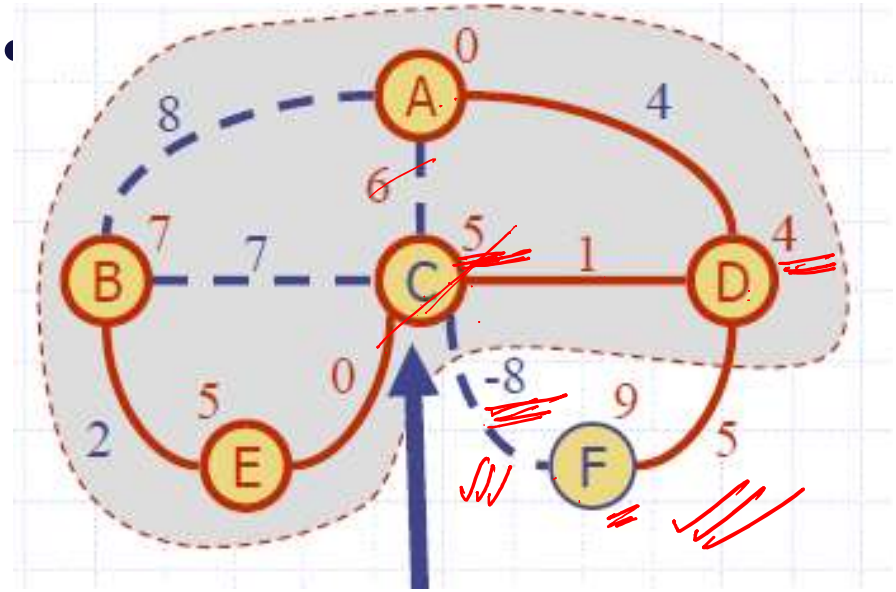


Dijkstra's Algorithm-Why It Doesn't Work for Negative-Weight Edges

- Dijkstra relies on one "simple" fact: if all weights are non-negative, adding an edge can never make a path shorter. That's why picking the shortest candidate edge (local optimality) always ends up being correct (global optimality).

Dijkstra's Algorithm-Why It Doesn't Work for Negative-Weight Edges

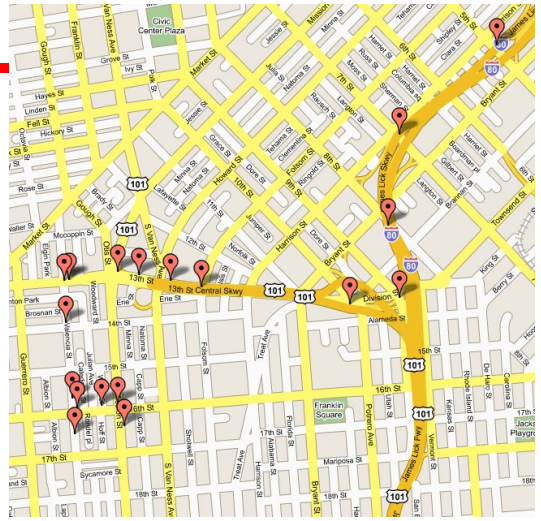
- If a node with a negative incident edge were to be added late to the cloud, it could mess up distances for vertices already in the cloud.



»C's true distance is 1, but it is already in the cloud with d(C)=5!

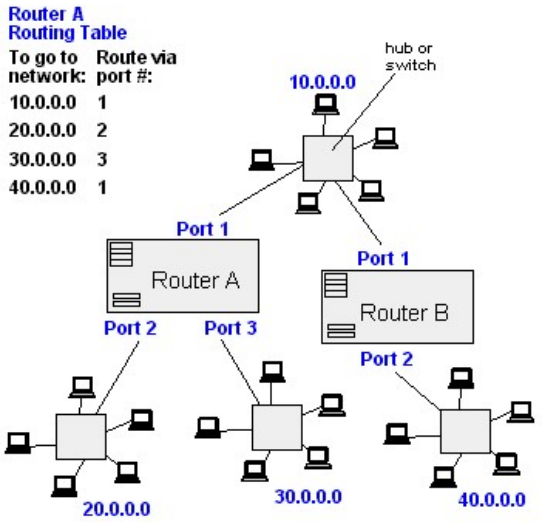
Dijkstra's Applications

- Network routing protocols - Open Shortest Path First
- It is used in geographical Maps.
- A* Algorithm in graph traversals.(AI)



From Computer Desktop Encyclopedia
© 1998 The Computer Language Co. Inc.

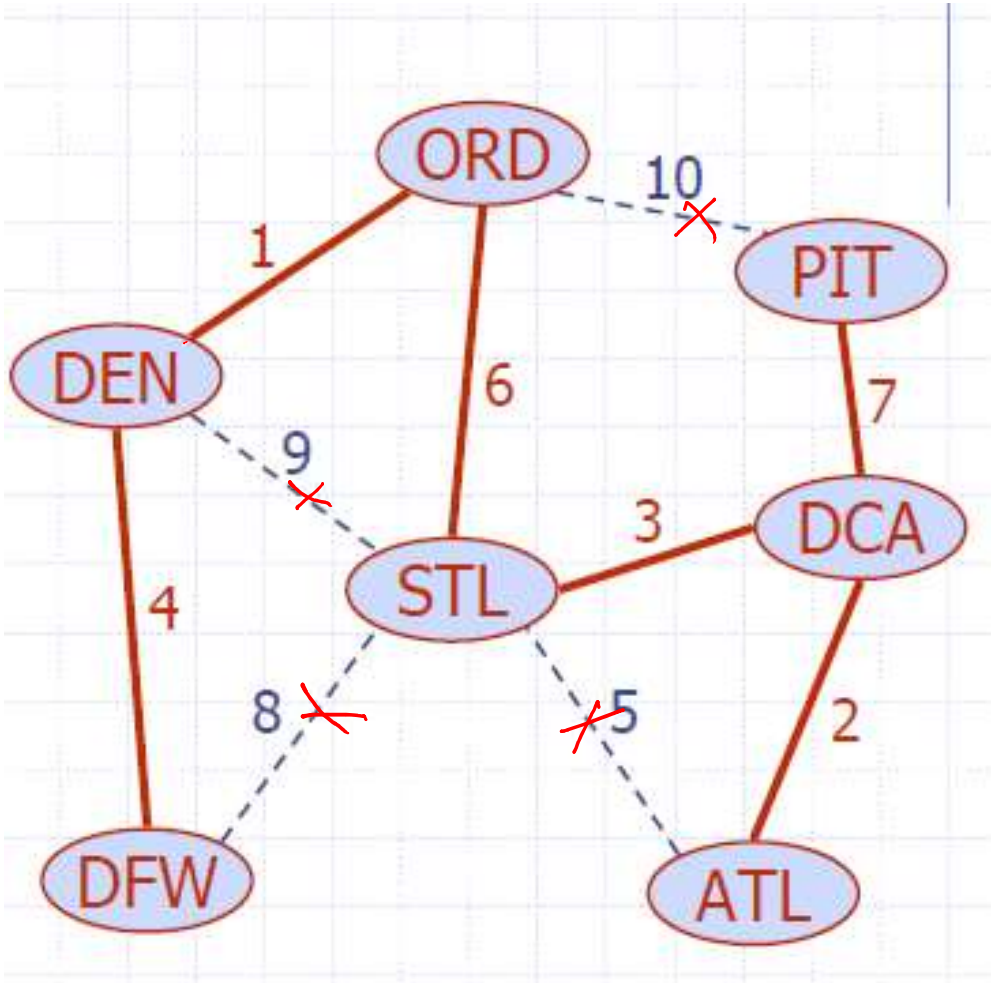
- Applied example
- https://prezi.com/po0cka9lrrqd/applied-example-of-dijkstras-algorithm/



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

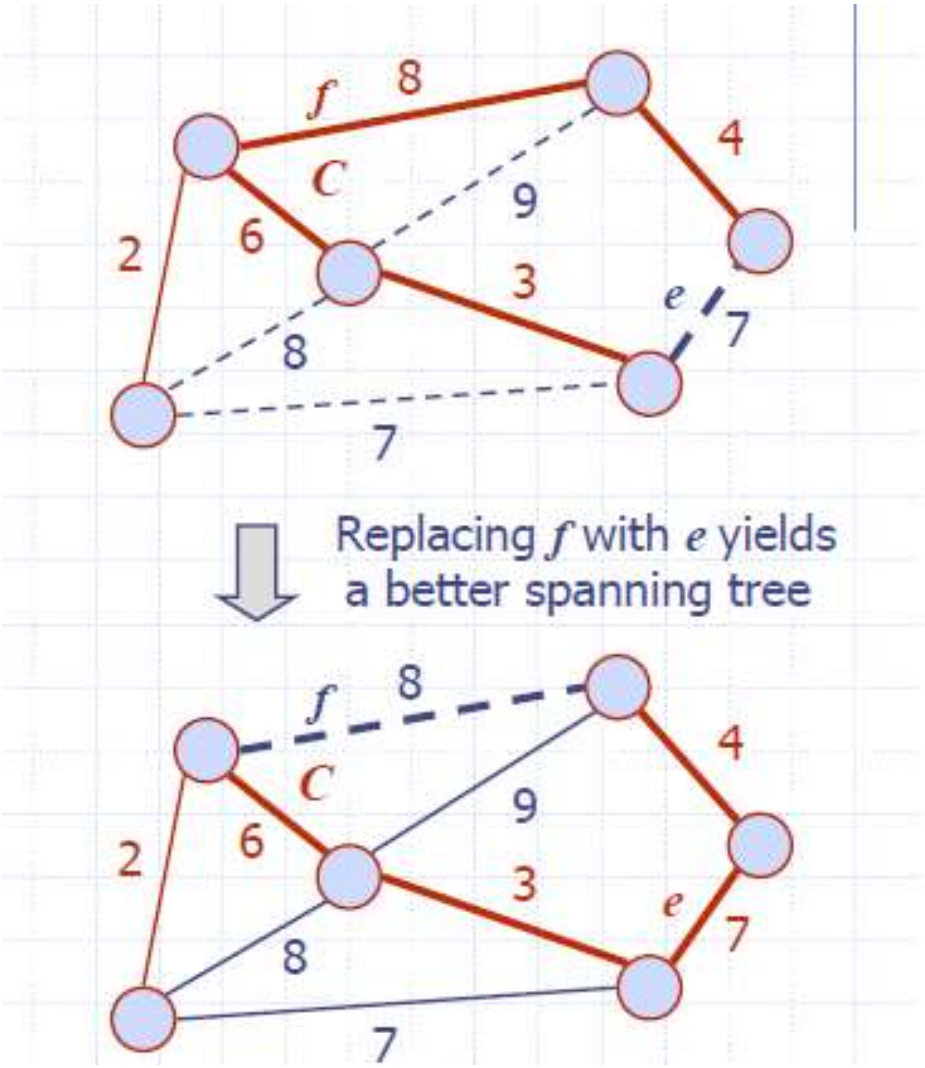
Minimum Spanning Tree



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

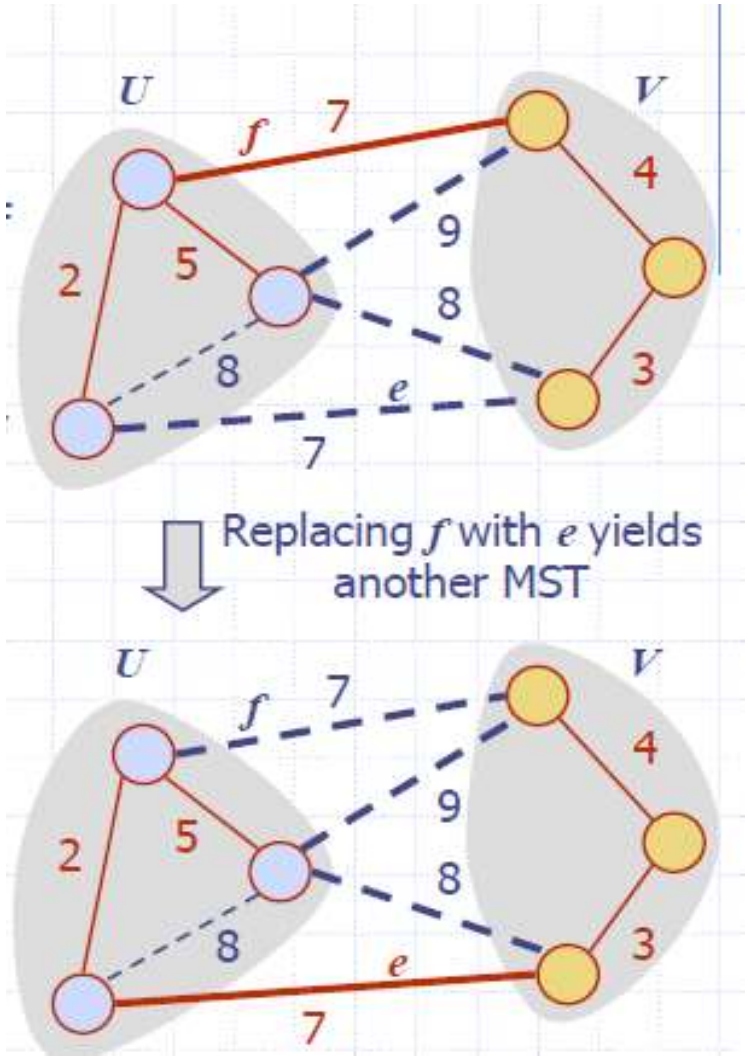


Minimum Spanning Tree

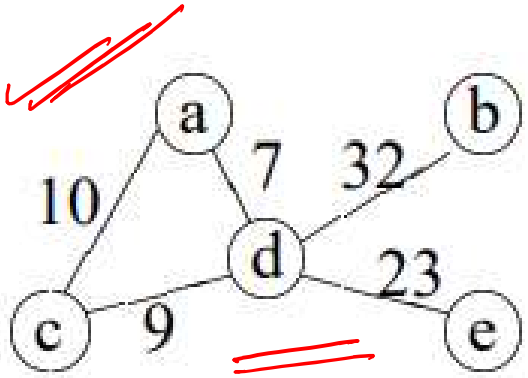
- Partition Property:
  - Consider a partition of the vertices of G into subsets U and V
  - Let e be an edge of minimum weight across the partition
  - There is a minimum spanning tree of G containing edge e

- 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

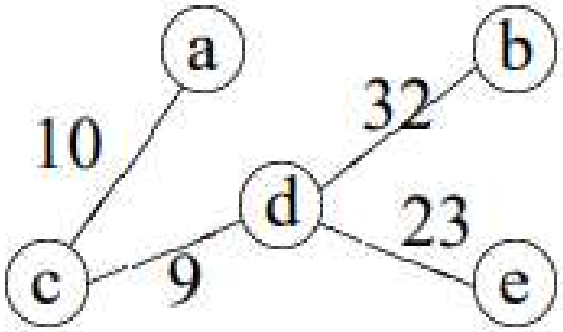


Minimum Spanning Tree
Prim's Algorithm

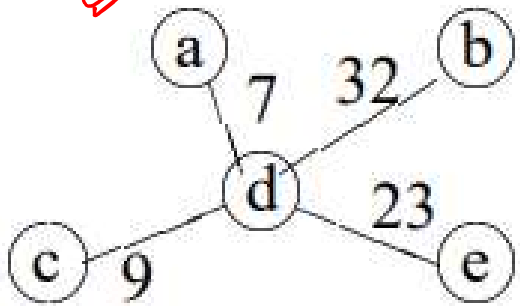


Spanning Tree

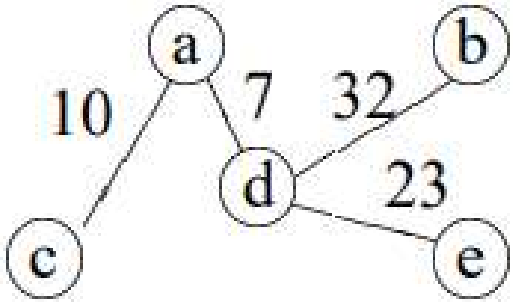
MST



Tree 1. w=74



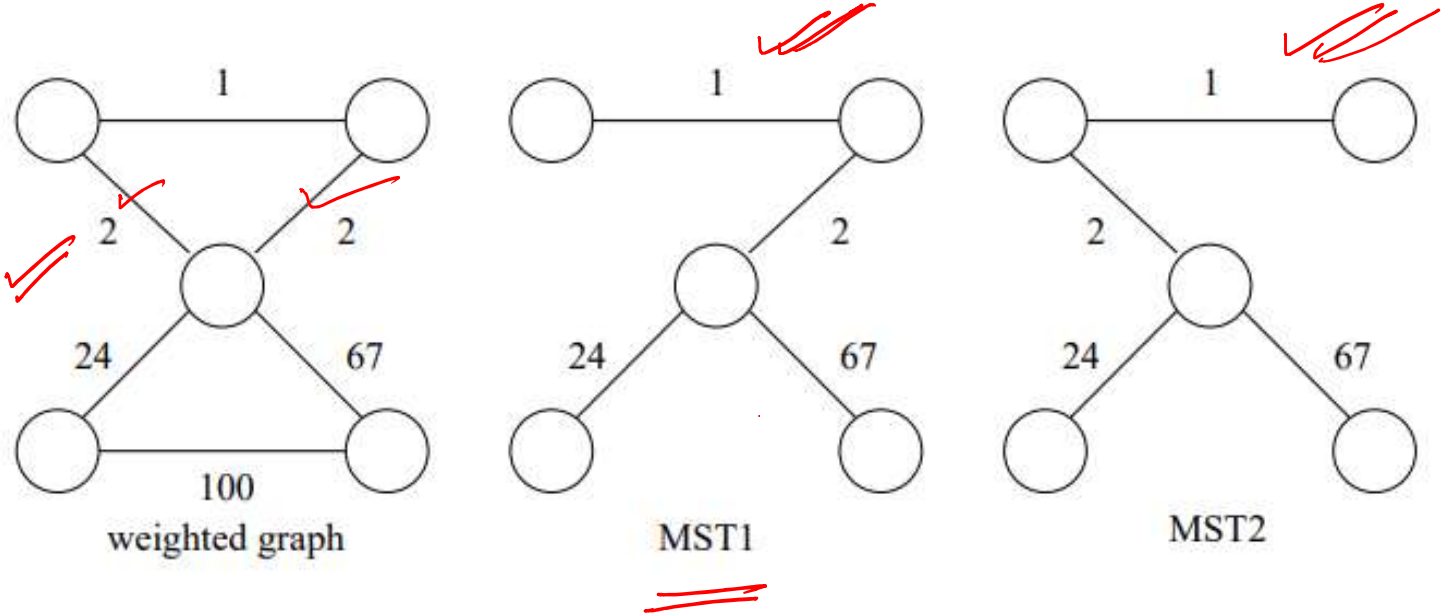
Tree 2, w=71



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

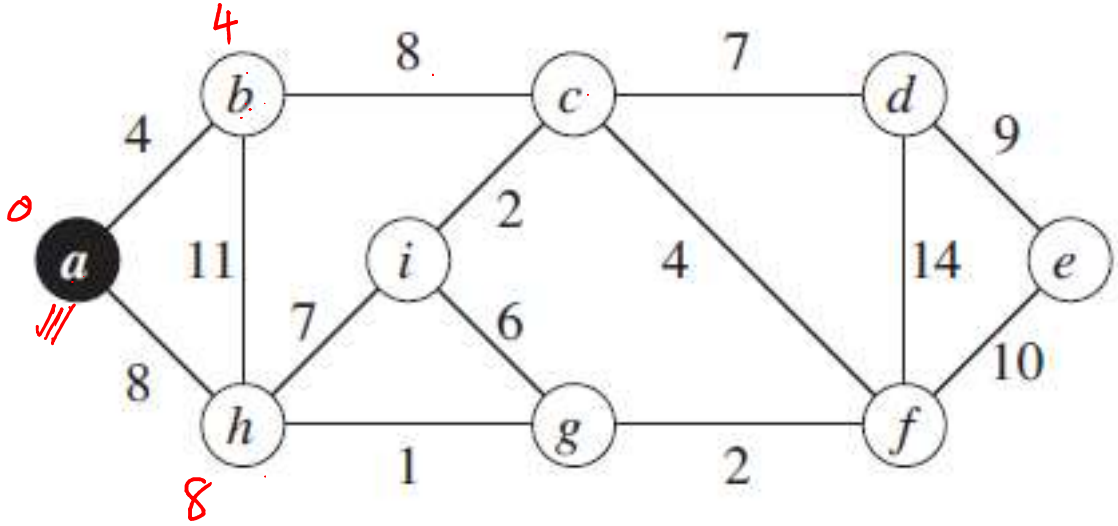


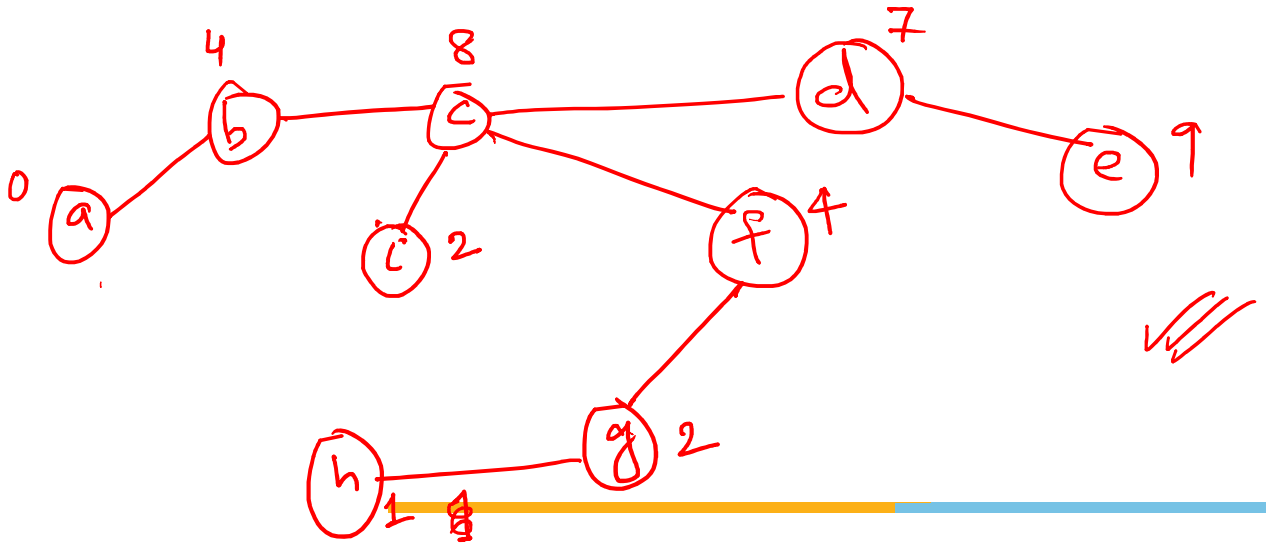
weighted graph MST1 MST2

Minimum Spanning Tree
Prim's Algorithm

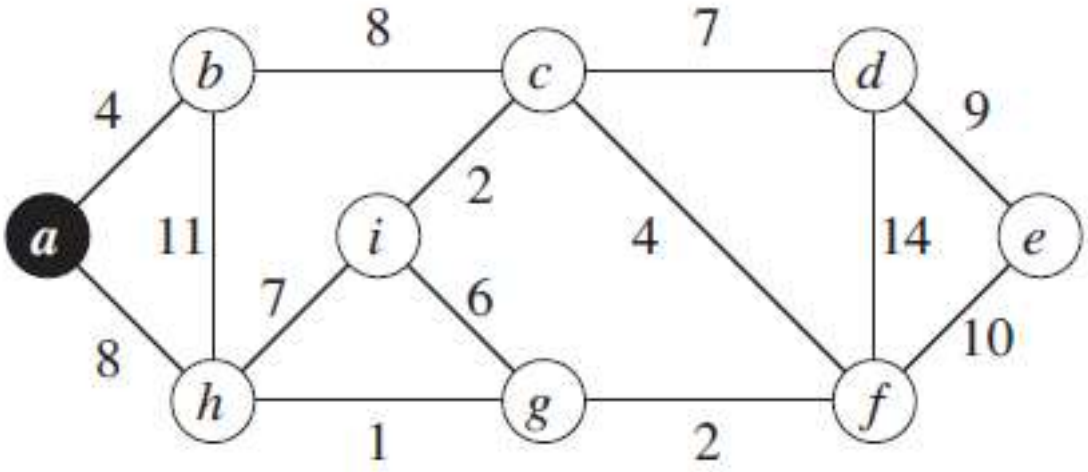
- Prim's algorithm is a greedy algorithm that finds a minimum spanning tree for a weighted undirected graph.
- This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized.
- At each step:
  - We add to the cloud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

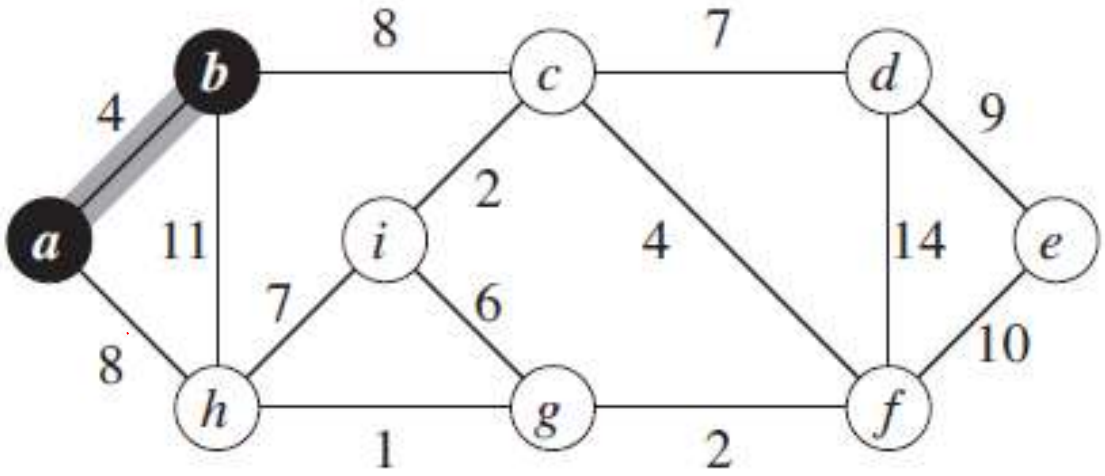
Example



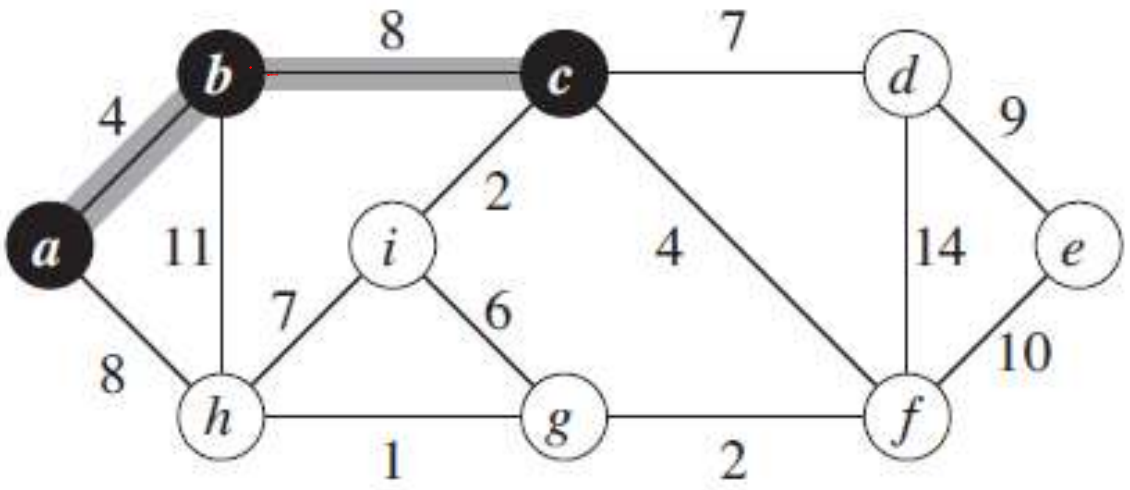


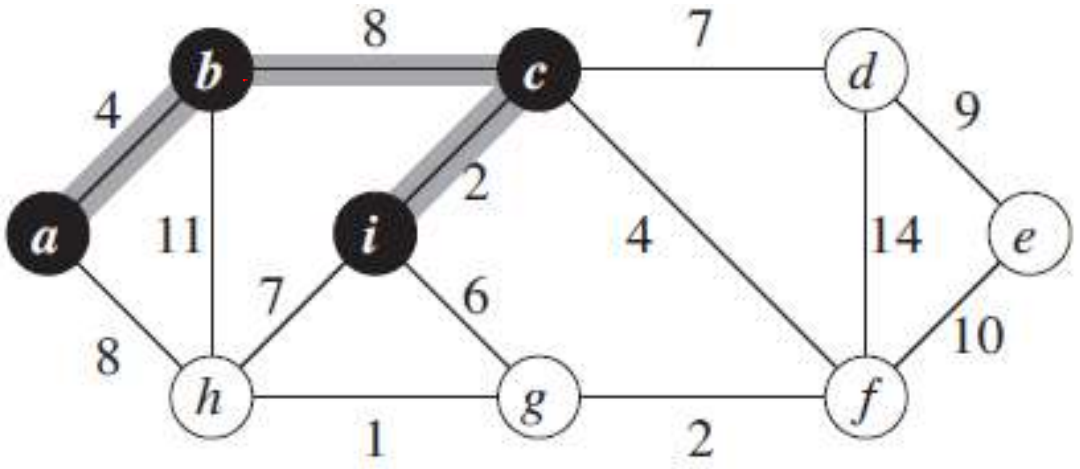
Example


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

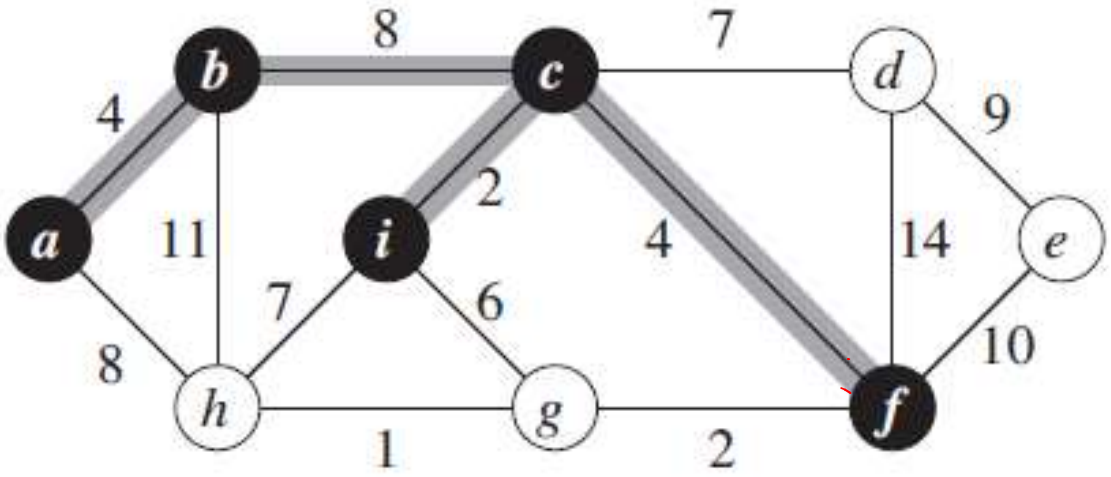

a
b
c
d
e
f
g
h
i
4
8
8
7
9
11
7
2
4
14
6
10
8
1
2

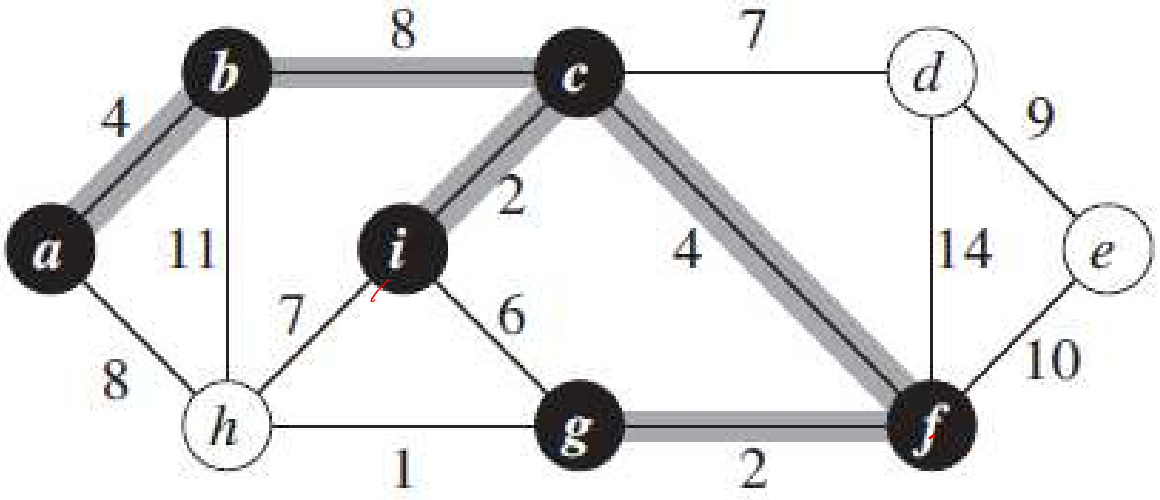
Example


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

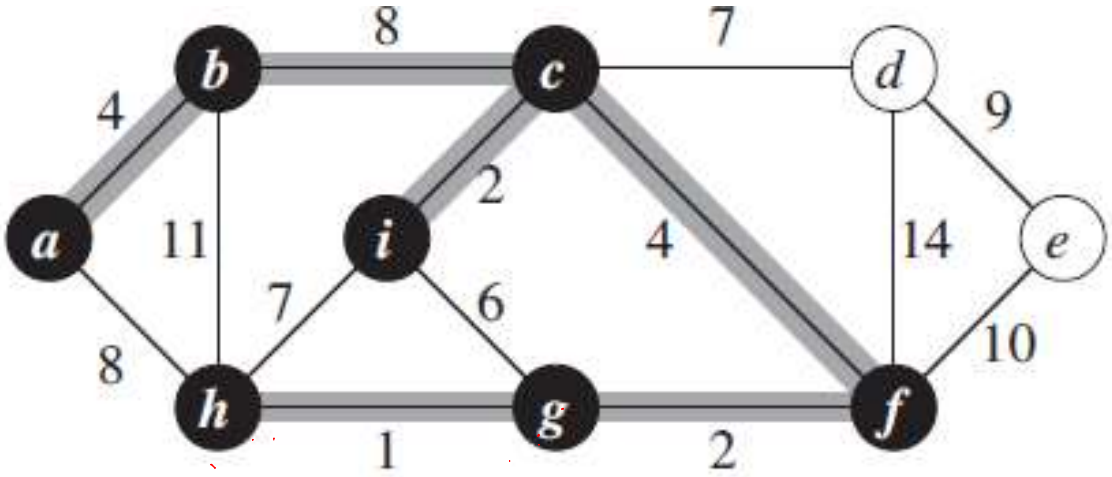

a
b
c
d
e
f
g
h
i
4
8
7
11
8
2
4
14
9
7
6
10
1
2

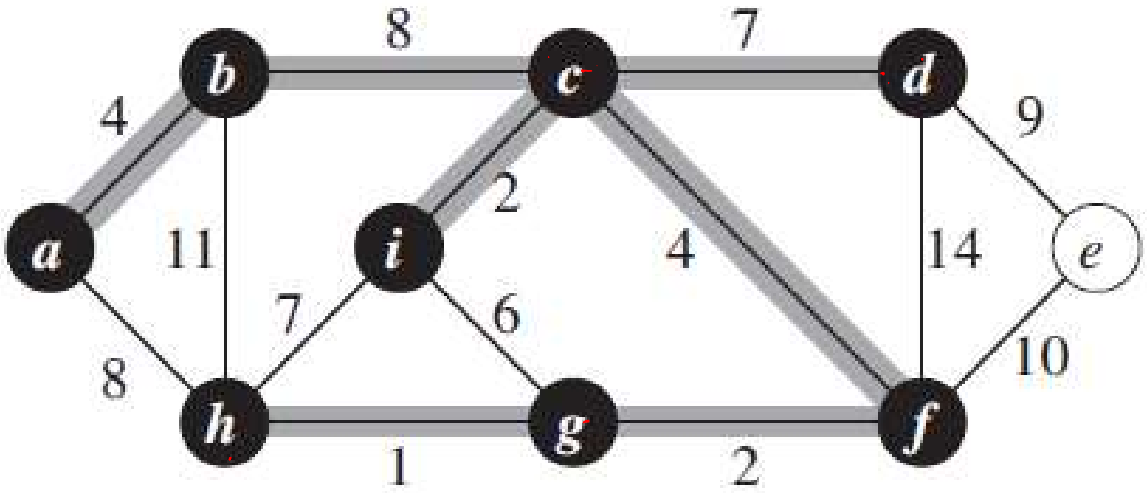
Example


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

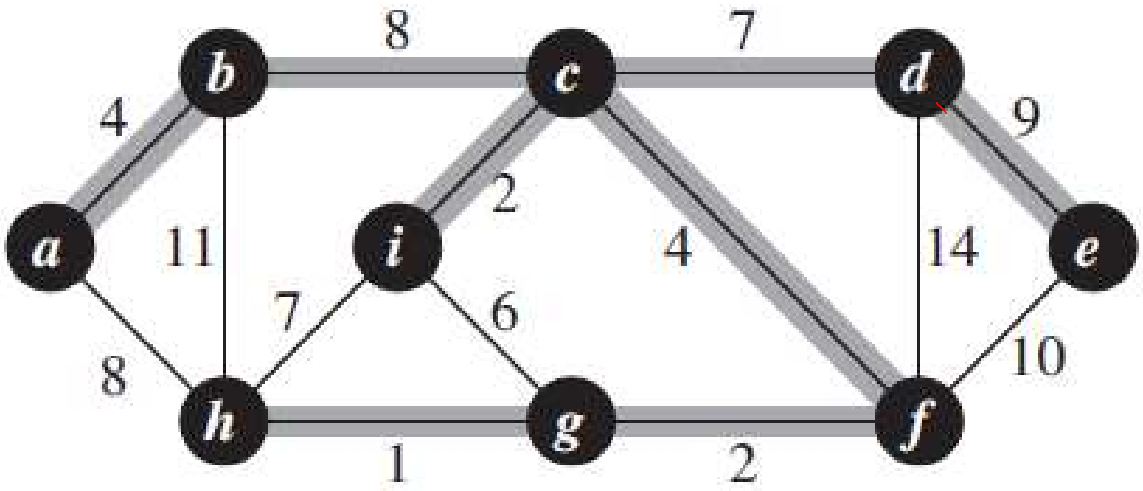

a
b
c
d
e
f
g
h
i
4
8
7
9
11
2
4
14
7
6
10
8
1
2

Example


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


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

Example



Minimum Spanning Tree
Prim's Algorithm

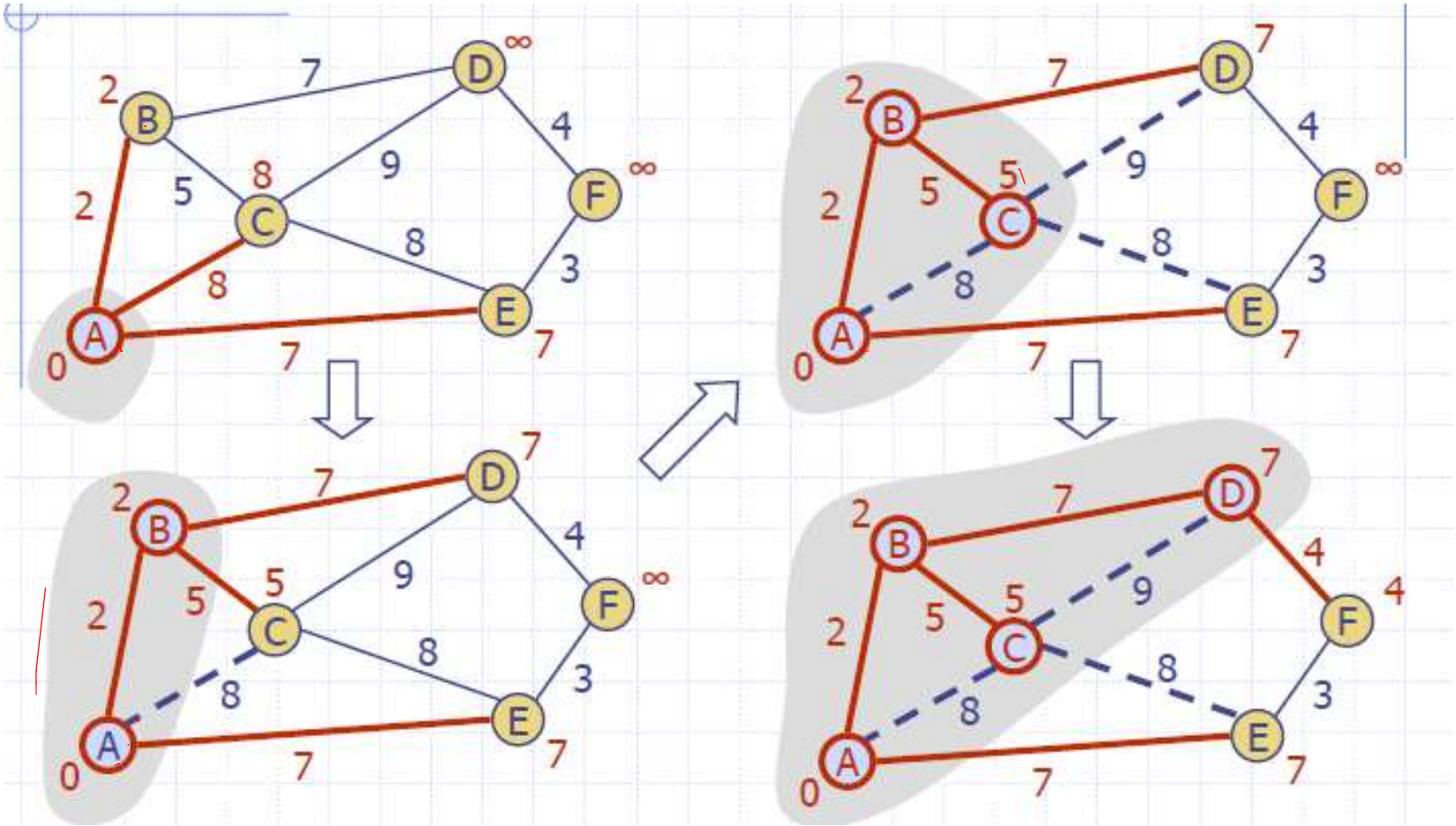
innovate achieve lead

```
MST-PRIM(G, w, r)
 1  for each u ∈ G.V
 2      u.key = ∞
 3      u.π = NIL
 4  r.key = 0
 5  Q = G.V
 6  while Q ≠ ∅
 7      u = EXTRACT-MIN(Q)
 8      for each v ∈ G.Adj[u]
 9          if v ∈ Q and w(u, v) < v.key
10              v.π = u
11              v.key = w(u, v)
```

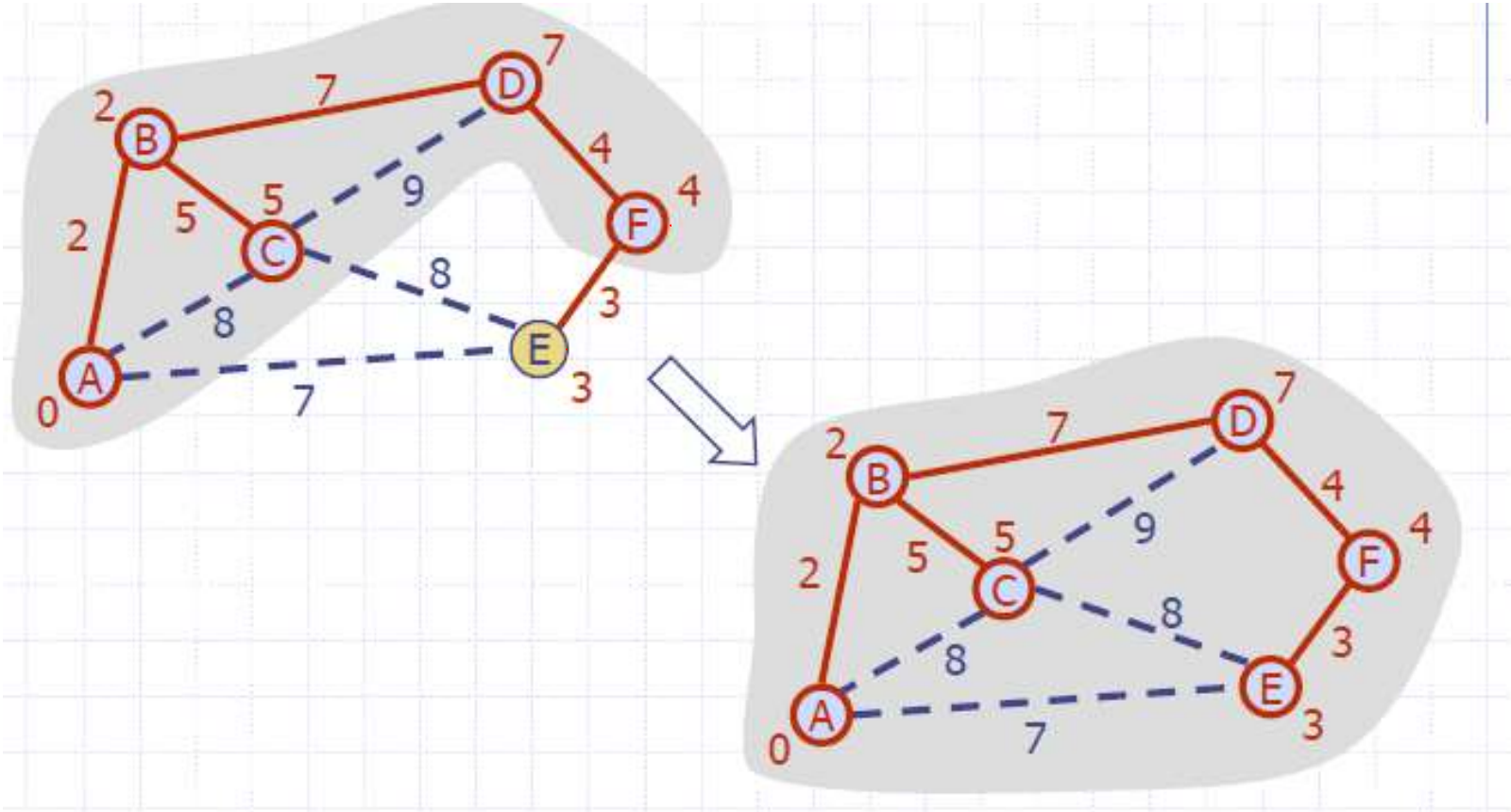
Prims's Algorithm-Analysis

- 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



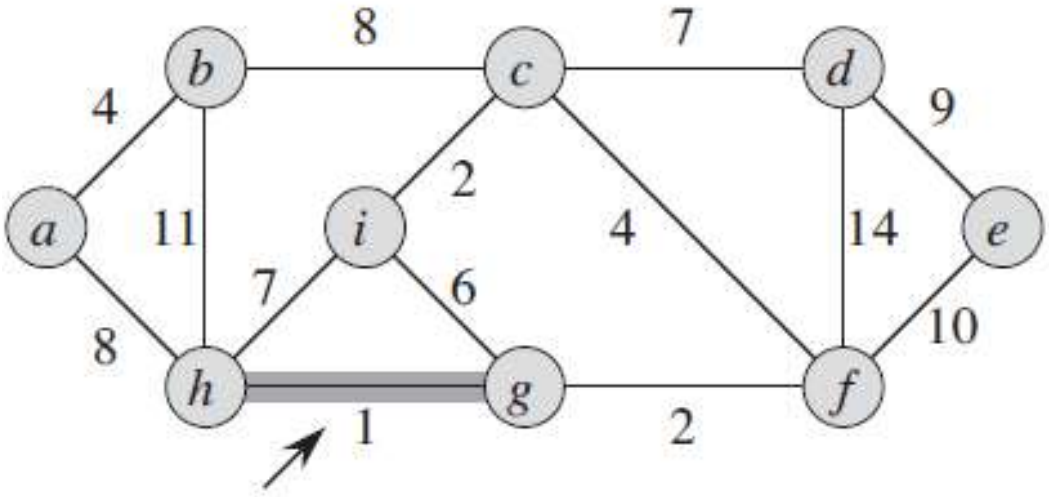
Minimum Spanning Tree
Prim's Algorithm



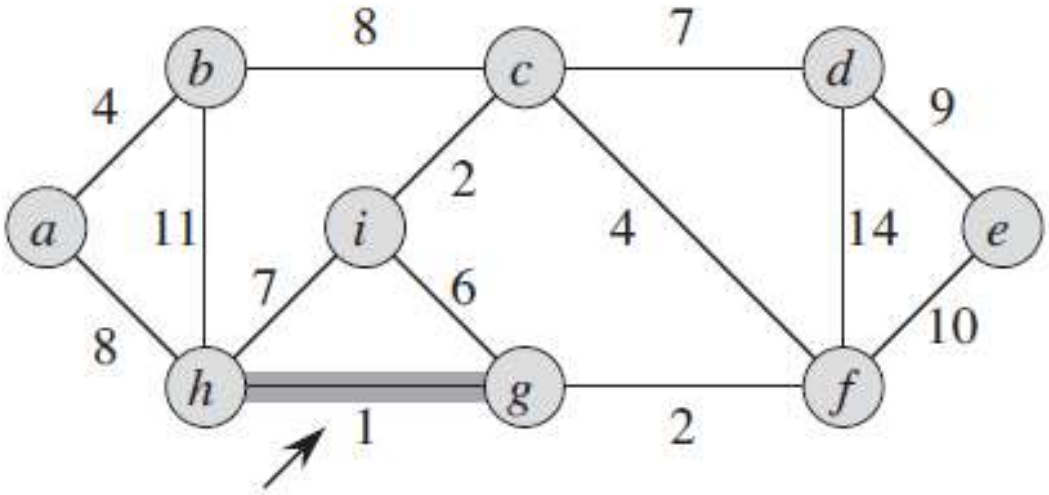
Minimum Spanning Tree
Kruskal's Algorithm

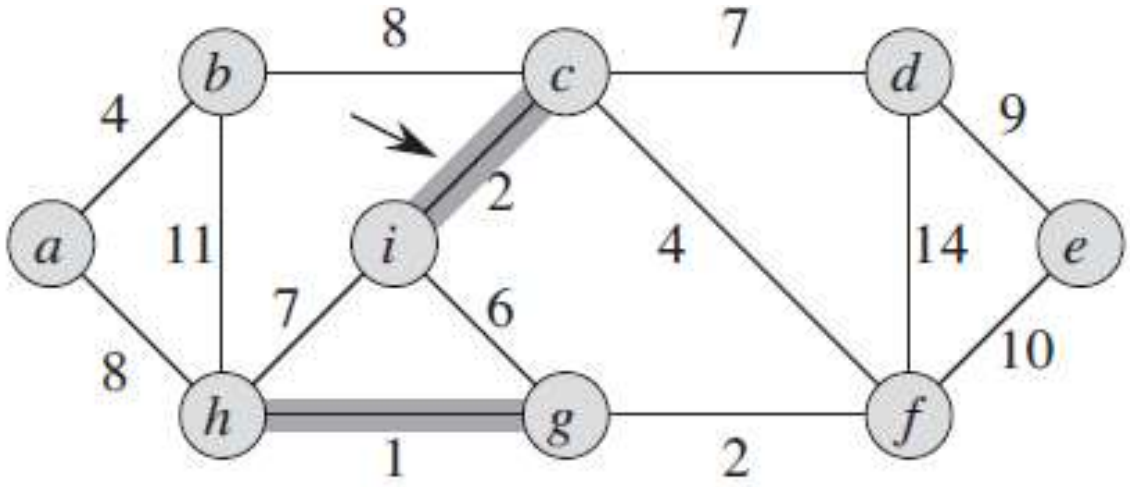
- It builds the MST in forest.
- Initially, each vertex is in its own tree in forest.
- Then, algorithm consider each edge in turn, order by increasing weight.
- If an edge (u, v) connects two different trees, then (u, v) is added to the set of edges of the MST, and two trees connected by an edge (u,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

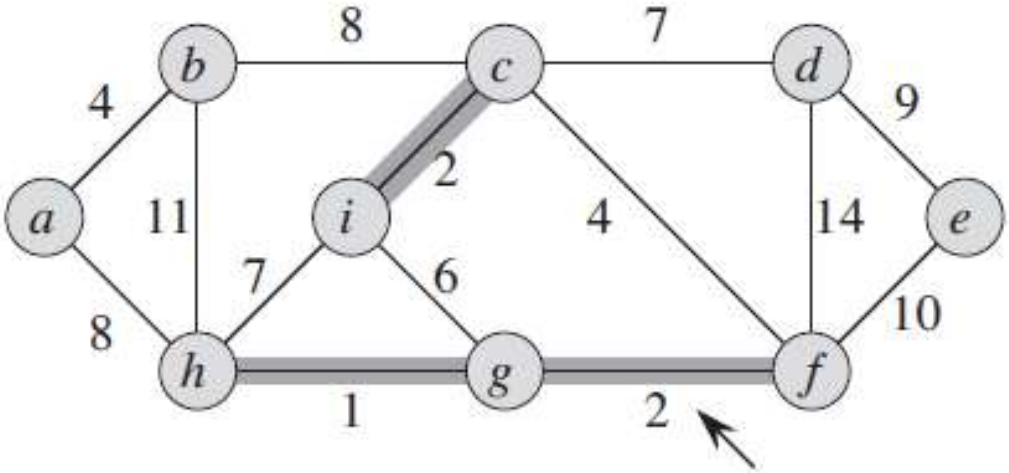


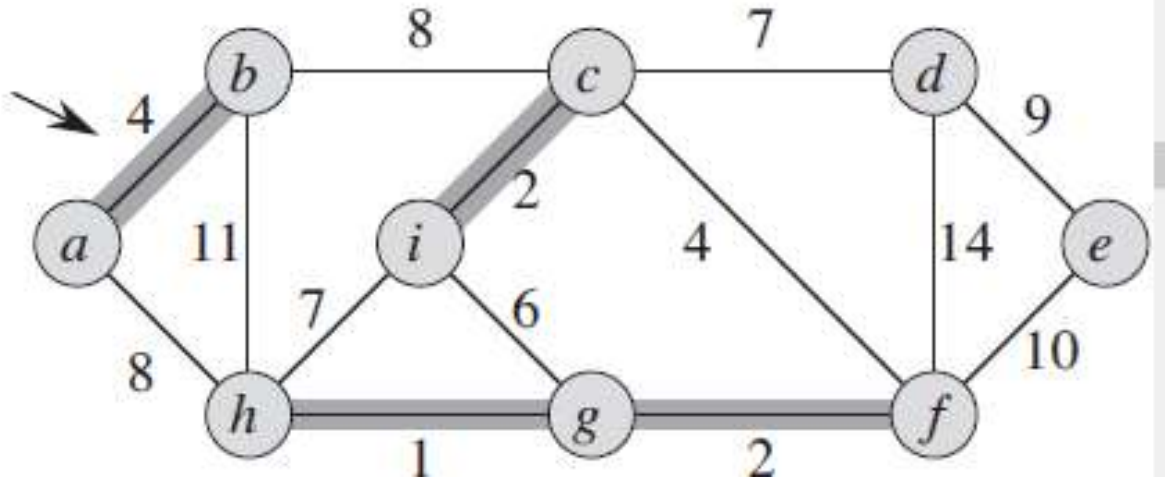
Minimum Spanning Tree
Kruskal's Algorithm-Example



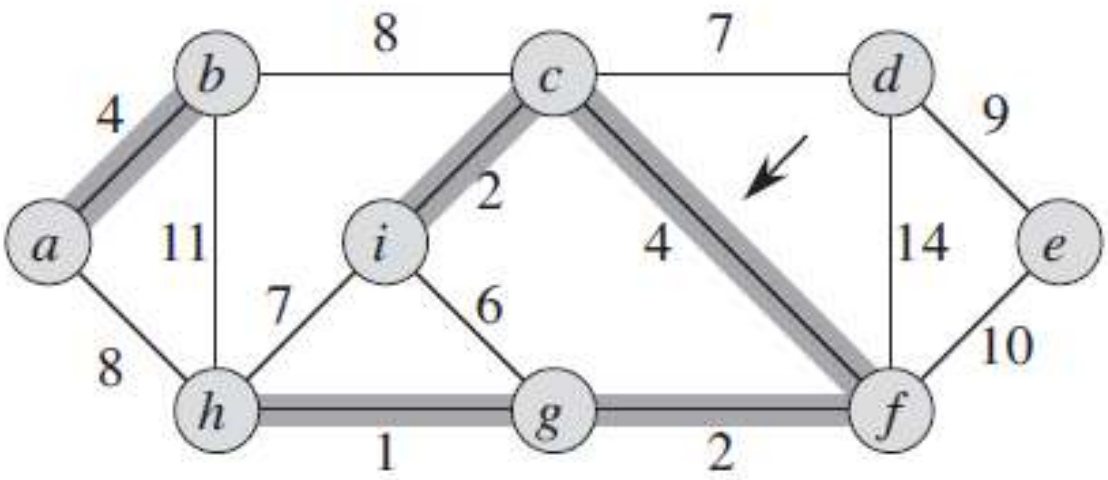


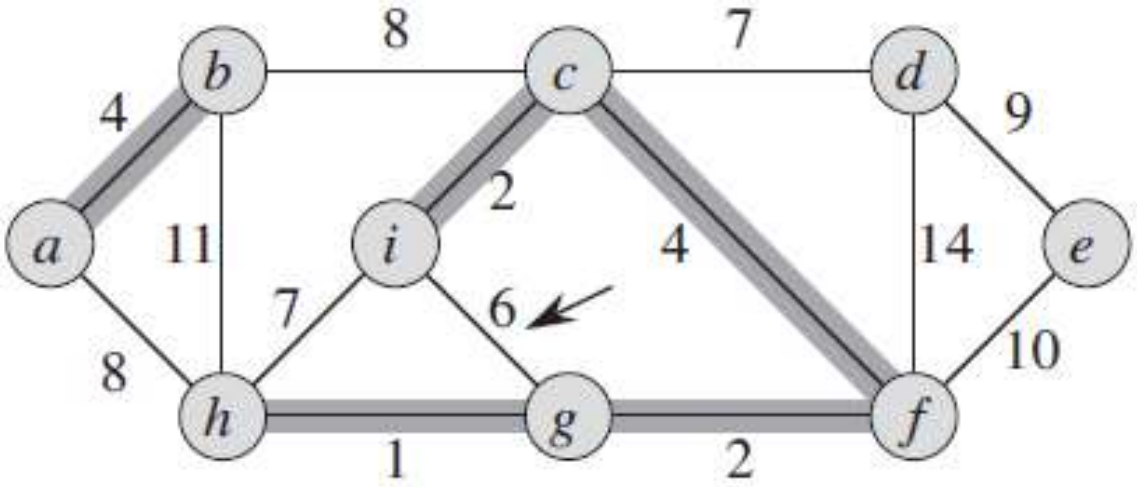
Minimum Spanning Tree
Kruskal's Algorithm-Example



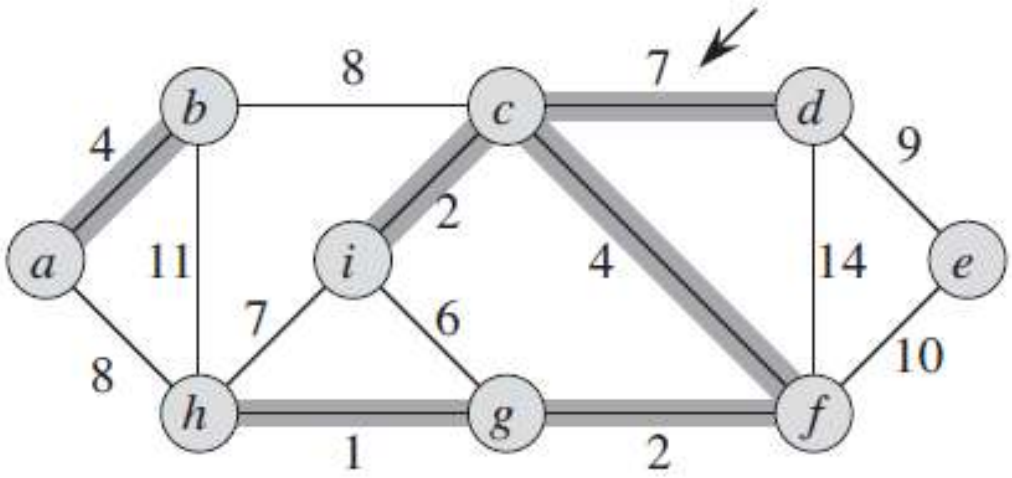


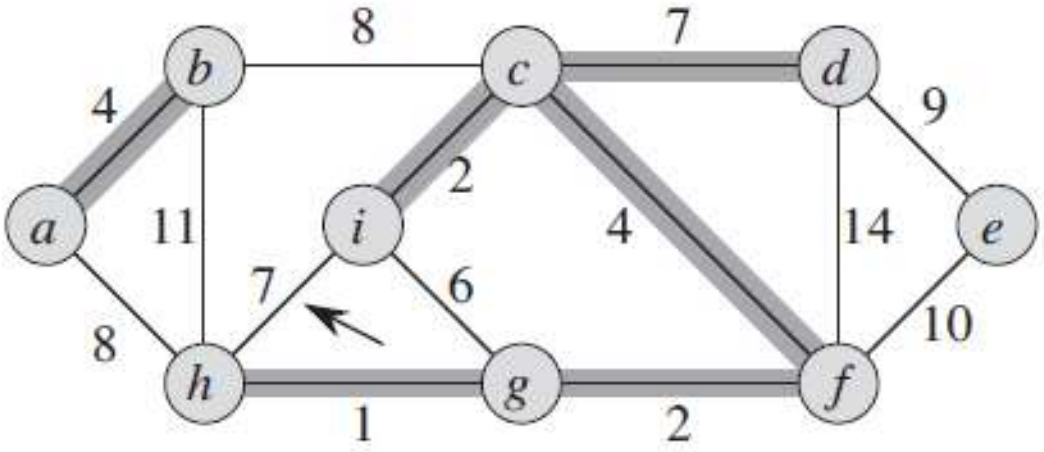
Minimum Spanning Tree Kruskal's Algorithm-Example





Minimum Spanning Tree
Kruskal's Algorithm-Example





Minimum Spanning Tree
Kruskal's Algorithm

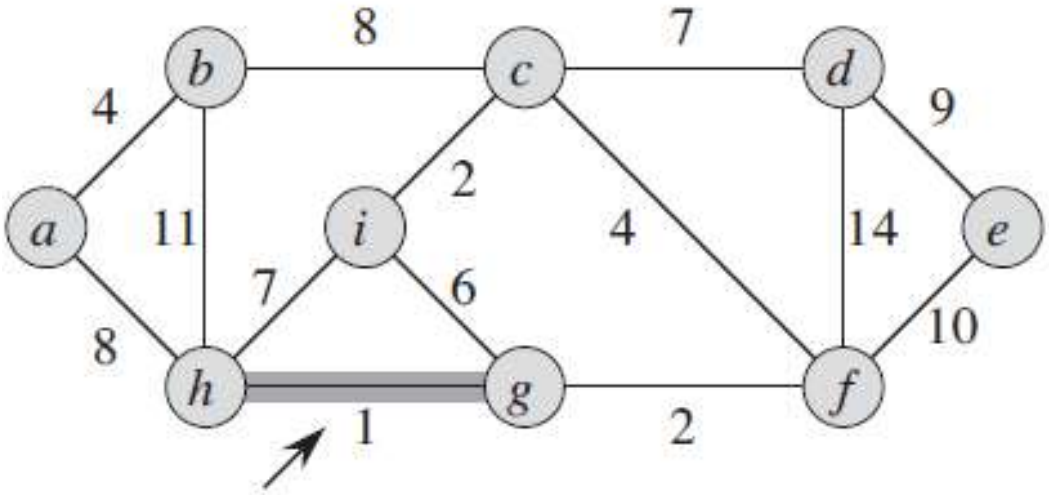
innovate achieve lead

```
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
- Contd…

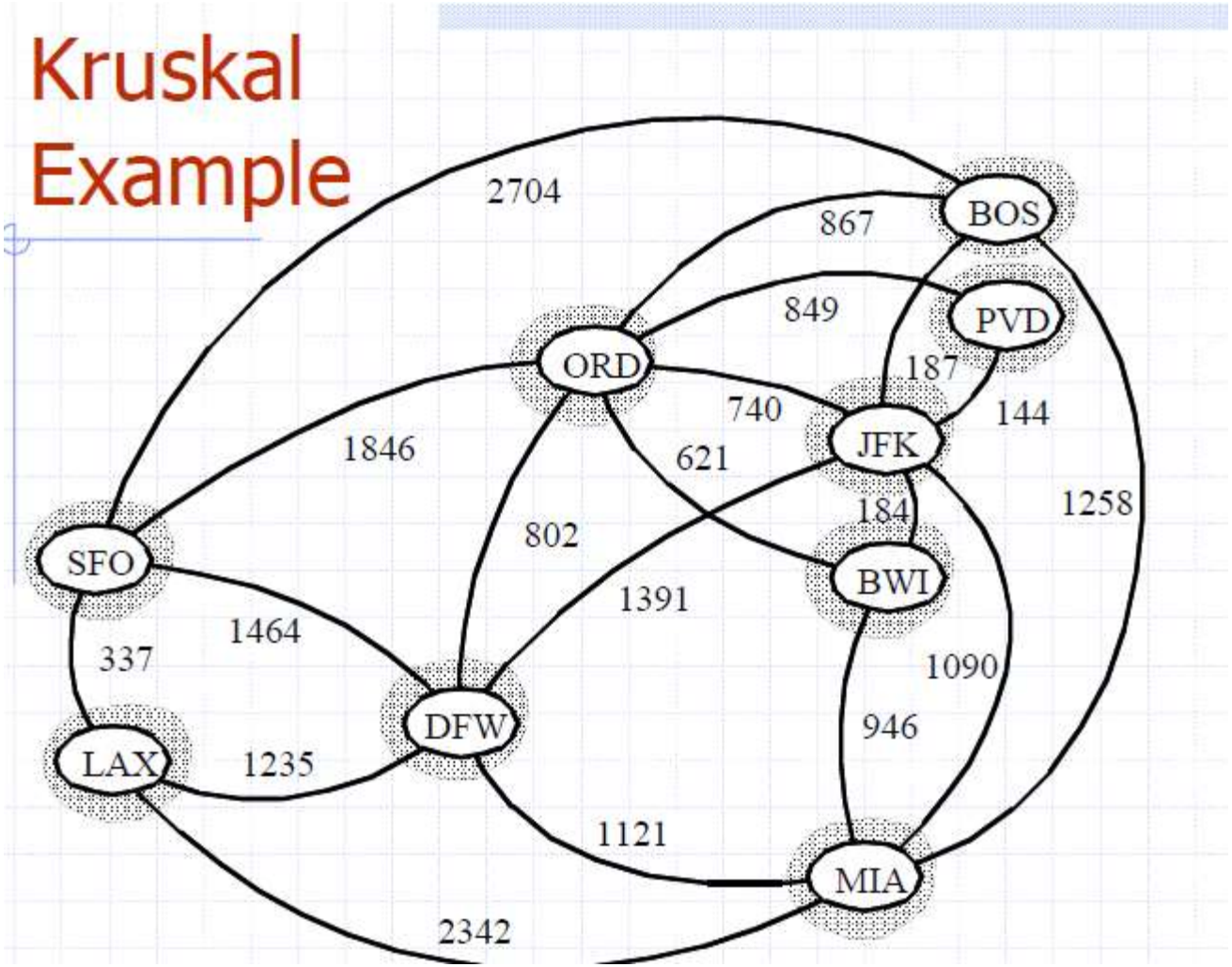
Minimum Spanning Tree
Kruskal's Algorithm-Analysis



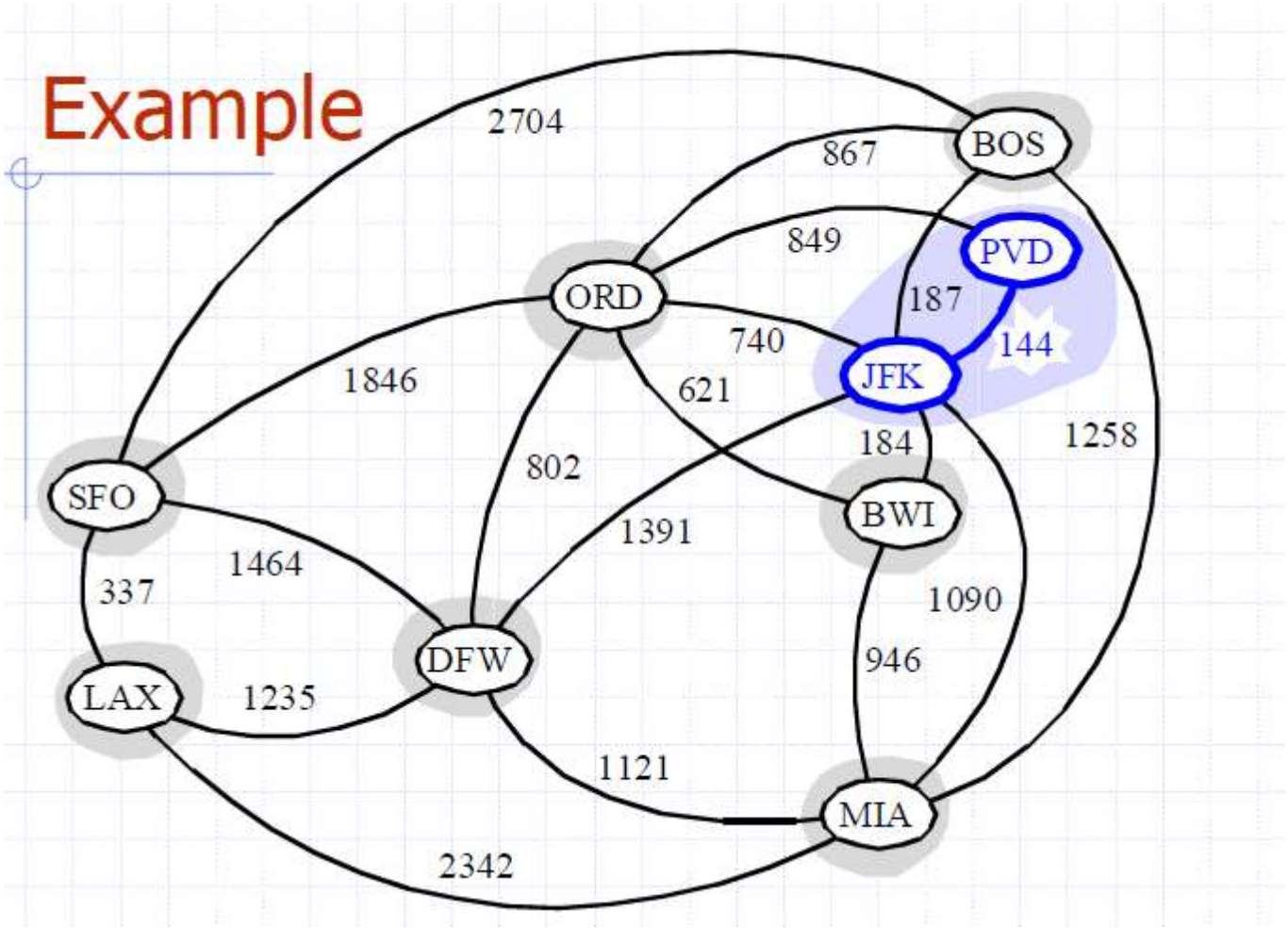
Minimum Spanning Tree Kruskal's Algorithm

- Here final tree will have n-1 edges
  - So n union operations need to be performed-O(n log n) [Cost of find included]
  - Since graph is connected, number of edges should be atleast n-1 is 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.
  - ie **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]

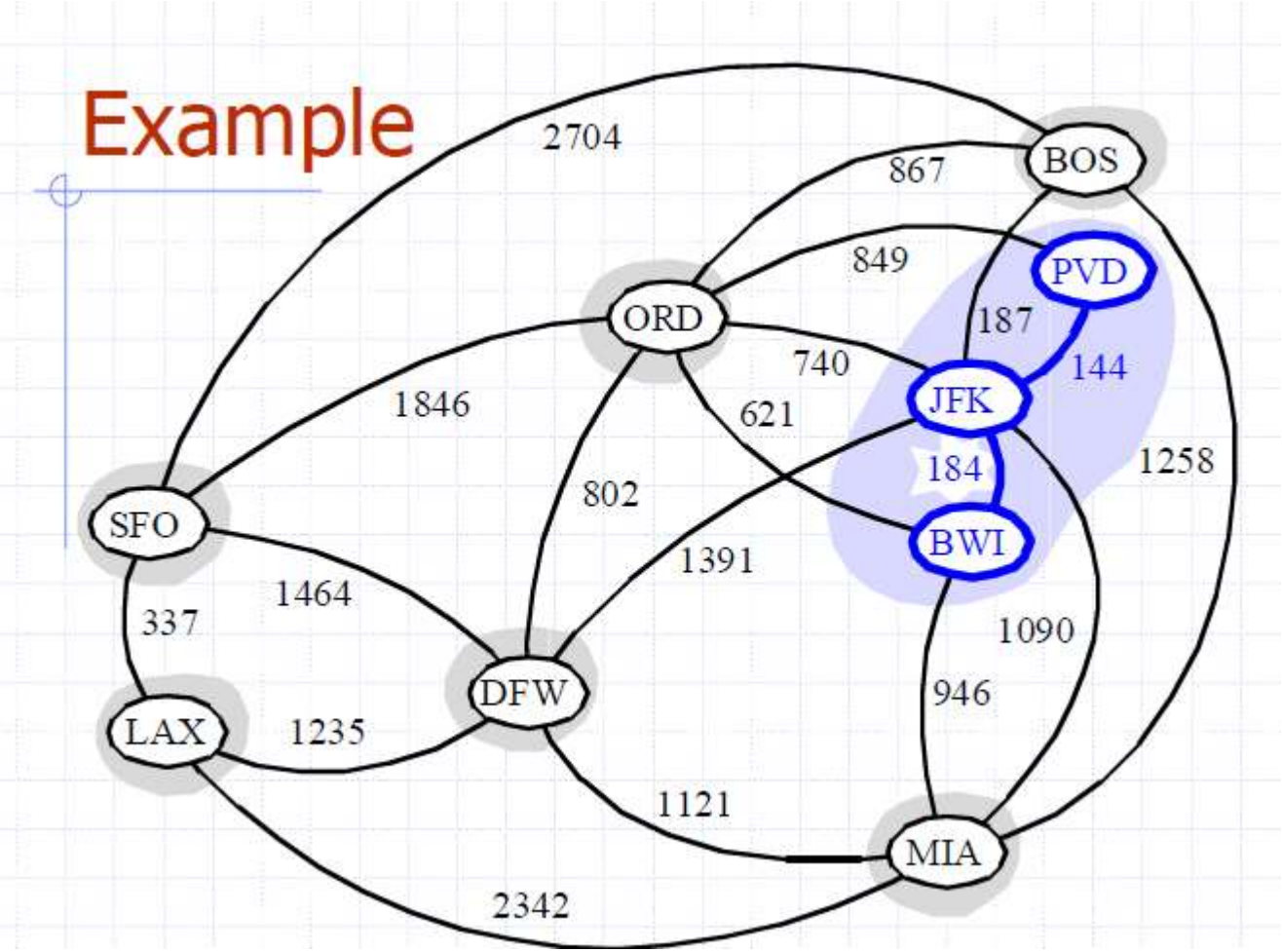
Minimum Spanning Tree



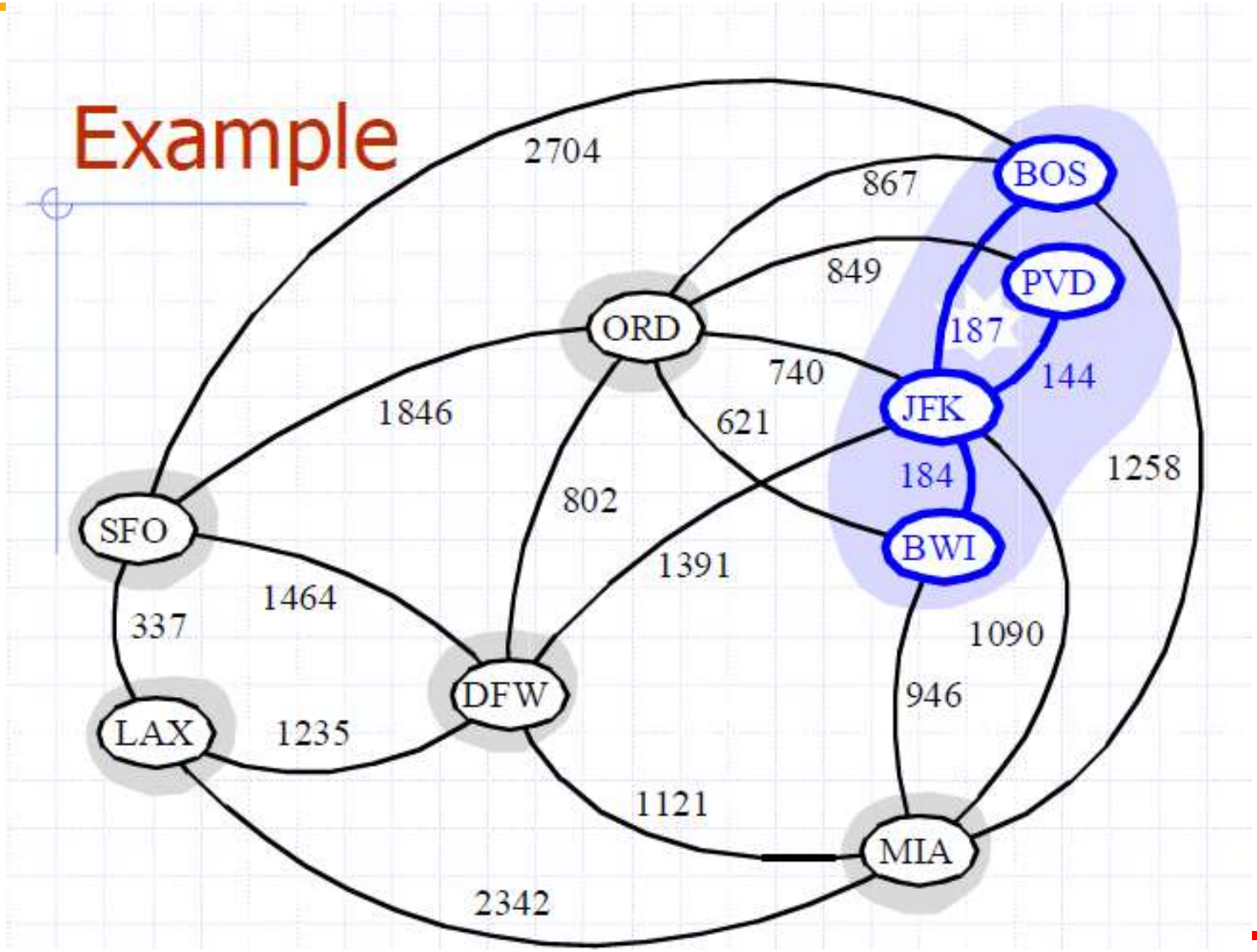
Minimum Spanning Tree



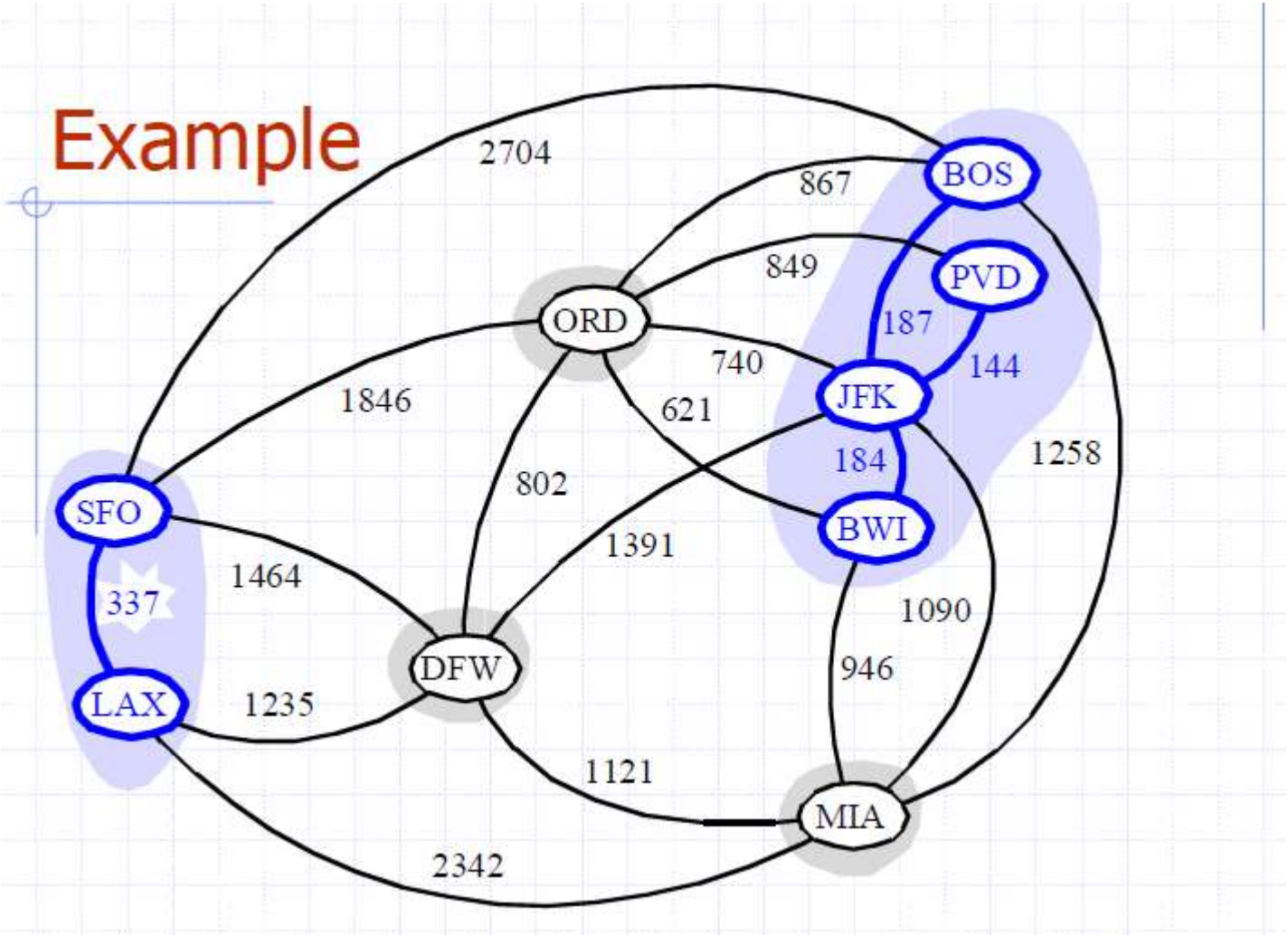
Minimum Spanning Tree



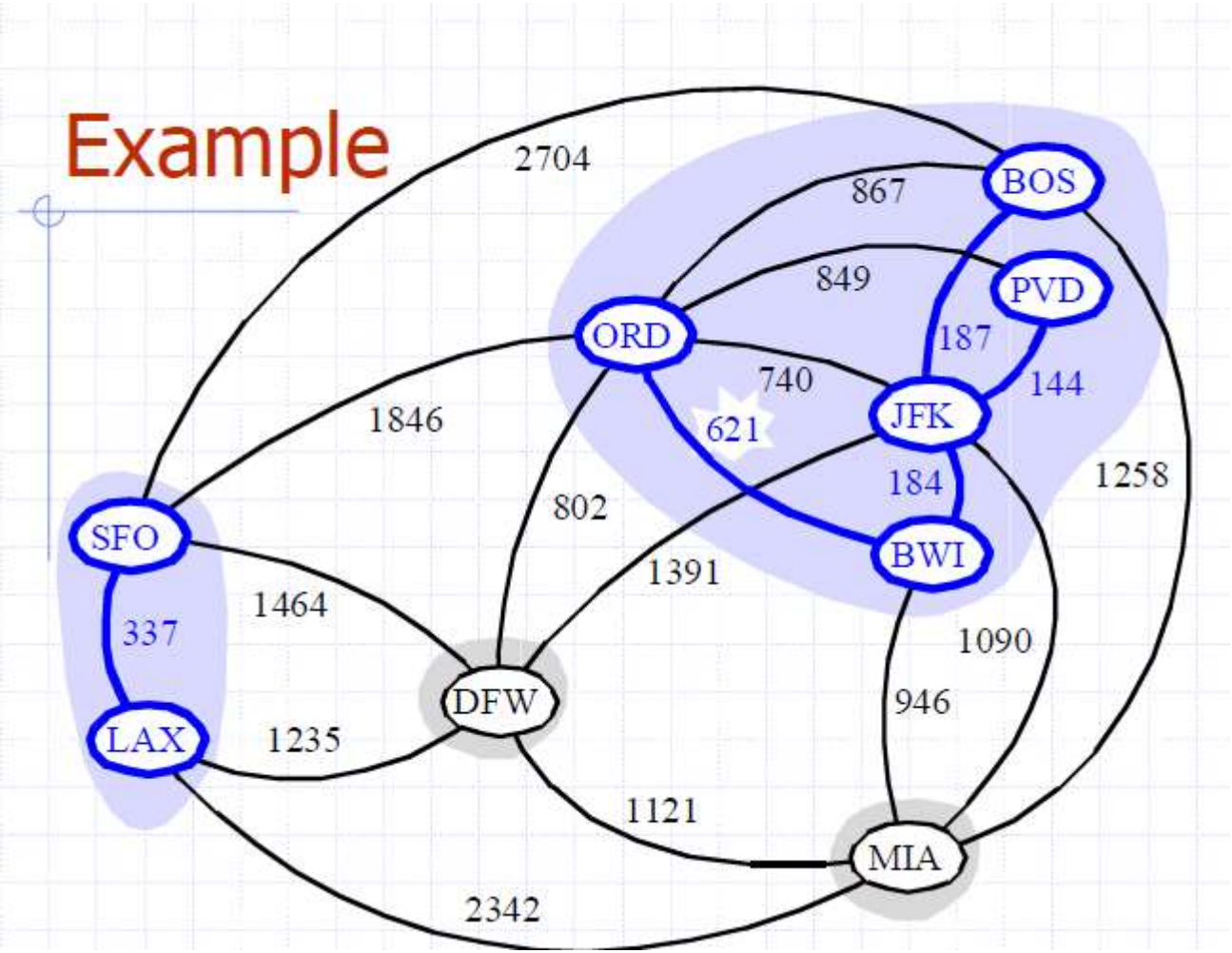
Minimum Spanning Tree



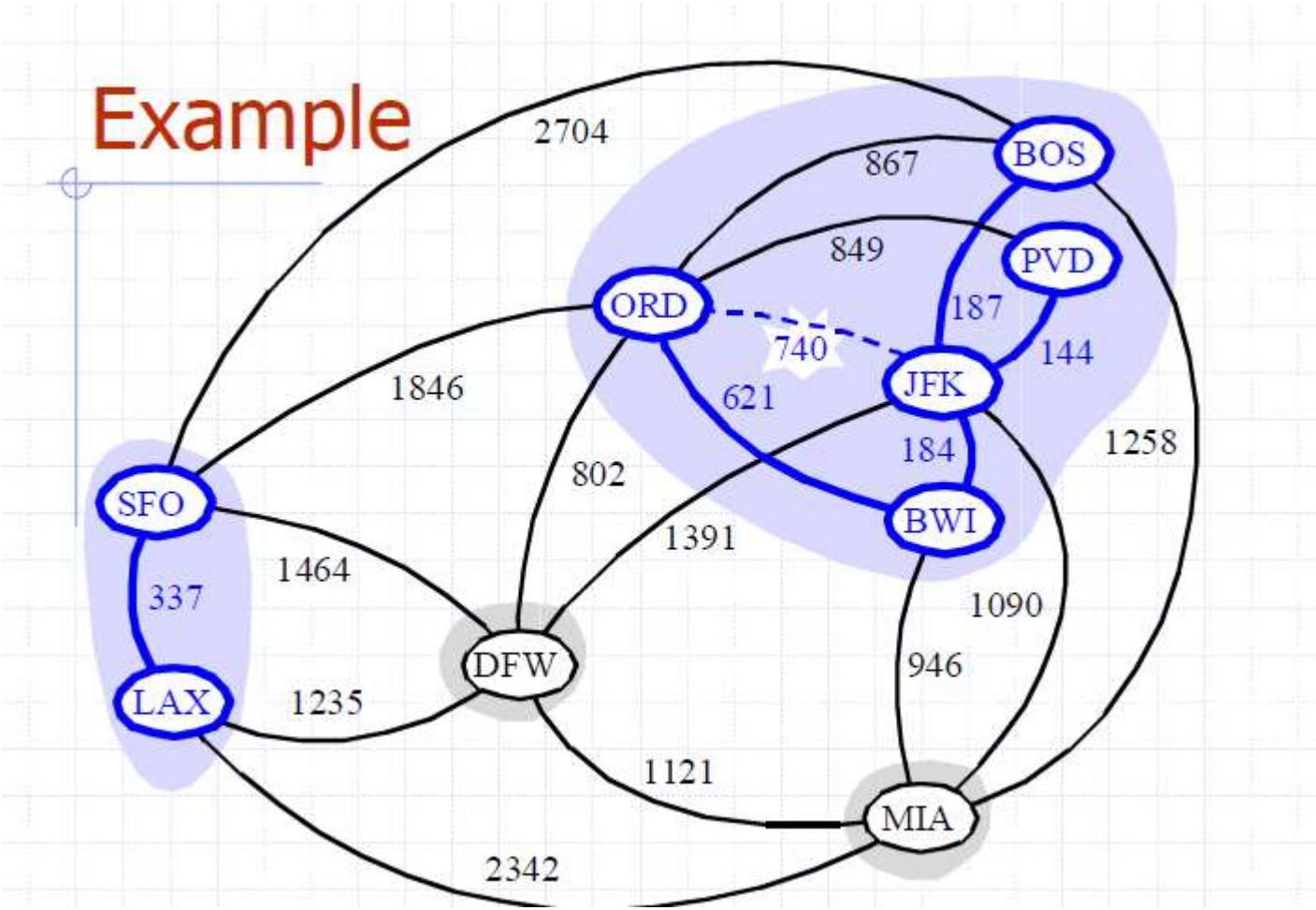
Minimum Spanning Tree



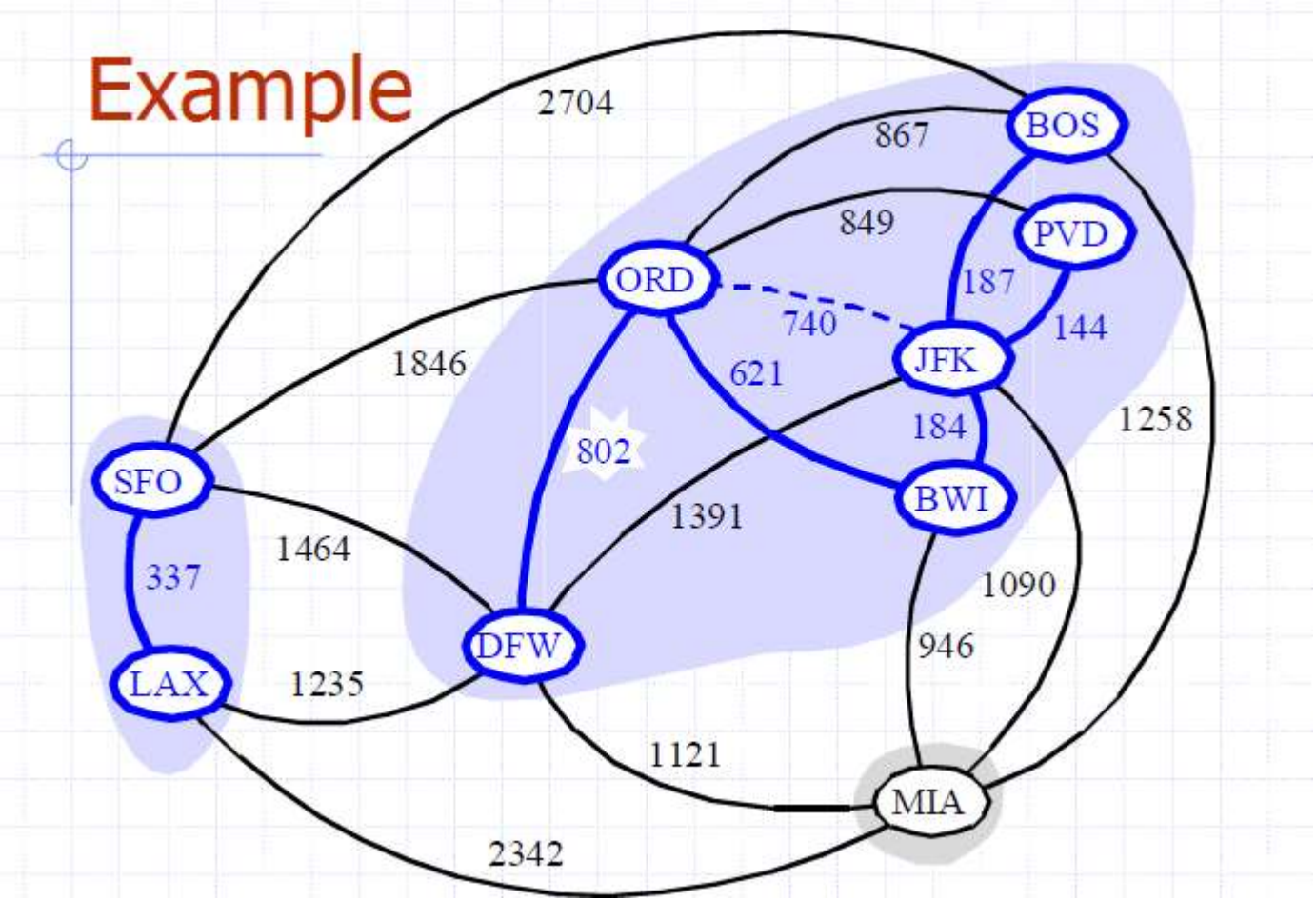
Minimum Spanning Tree



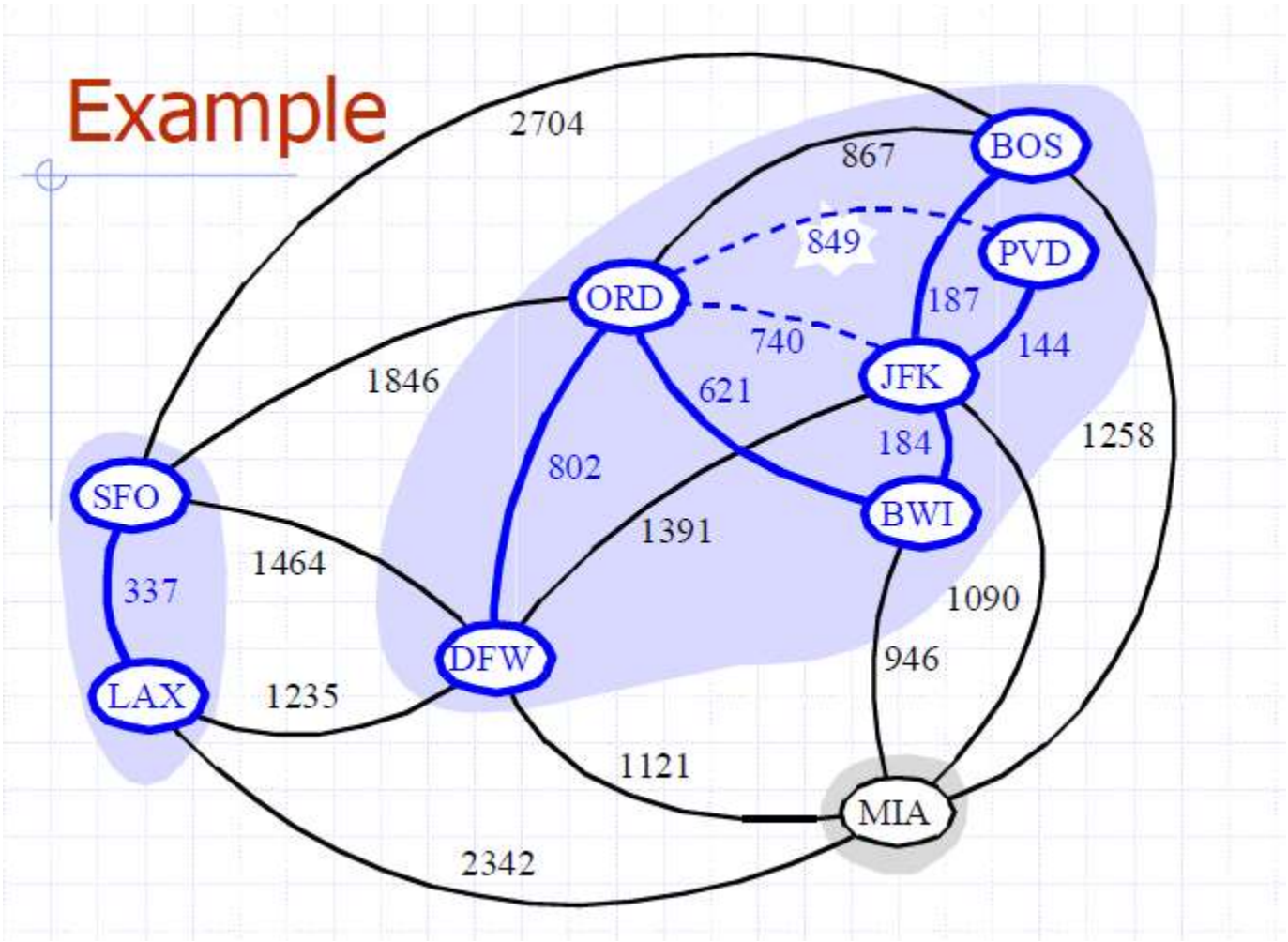
Minimum Spanning Tree



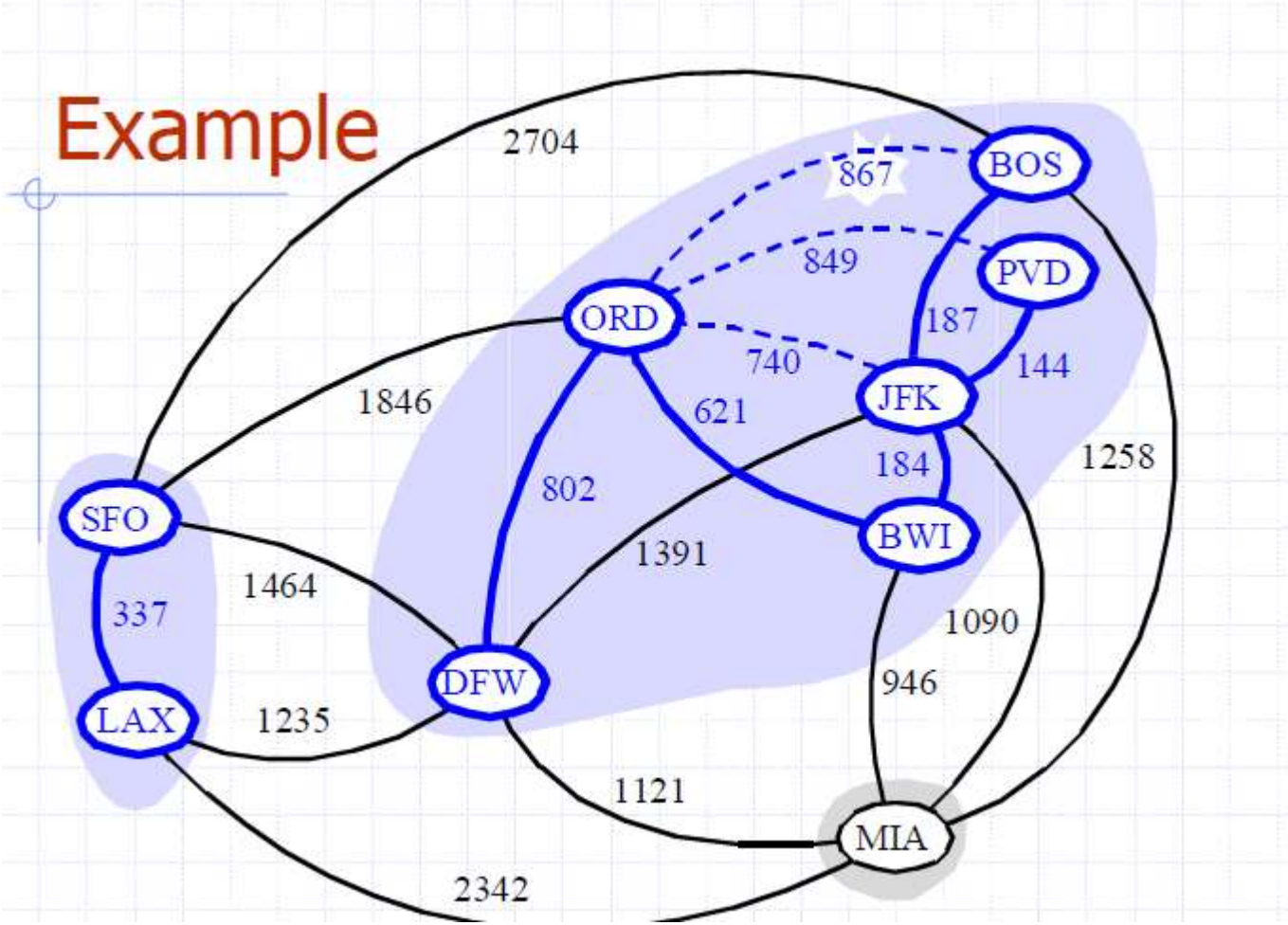
Minimum Spanning Tree



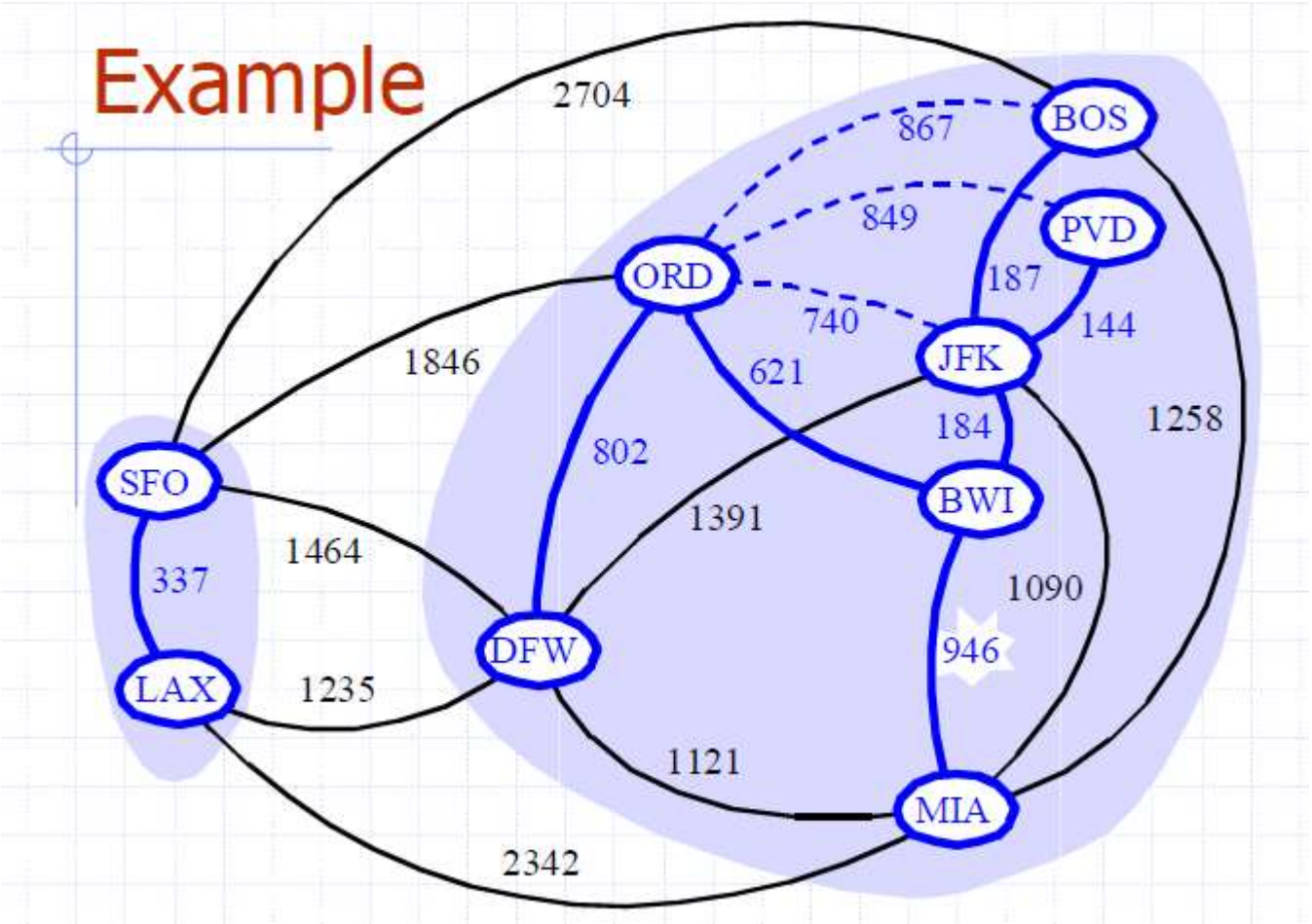
Minimum Spanning Tree



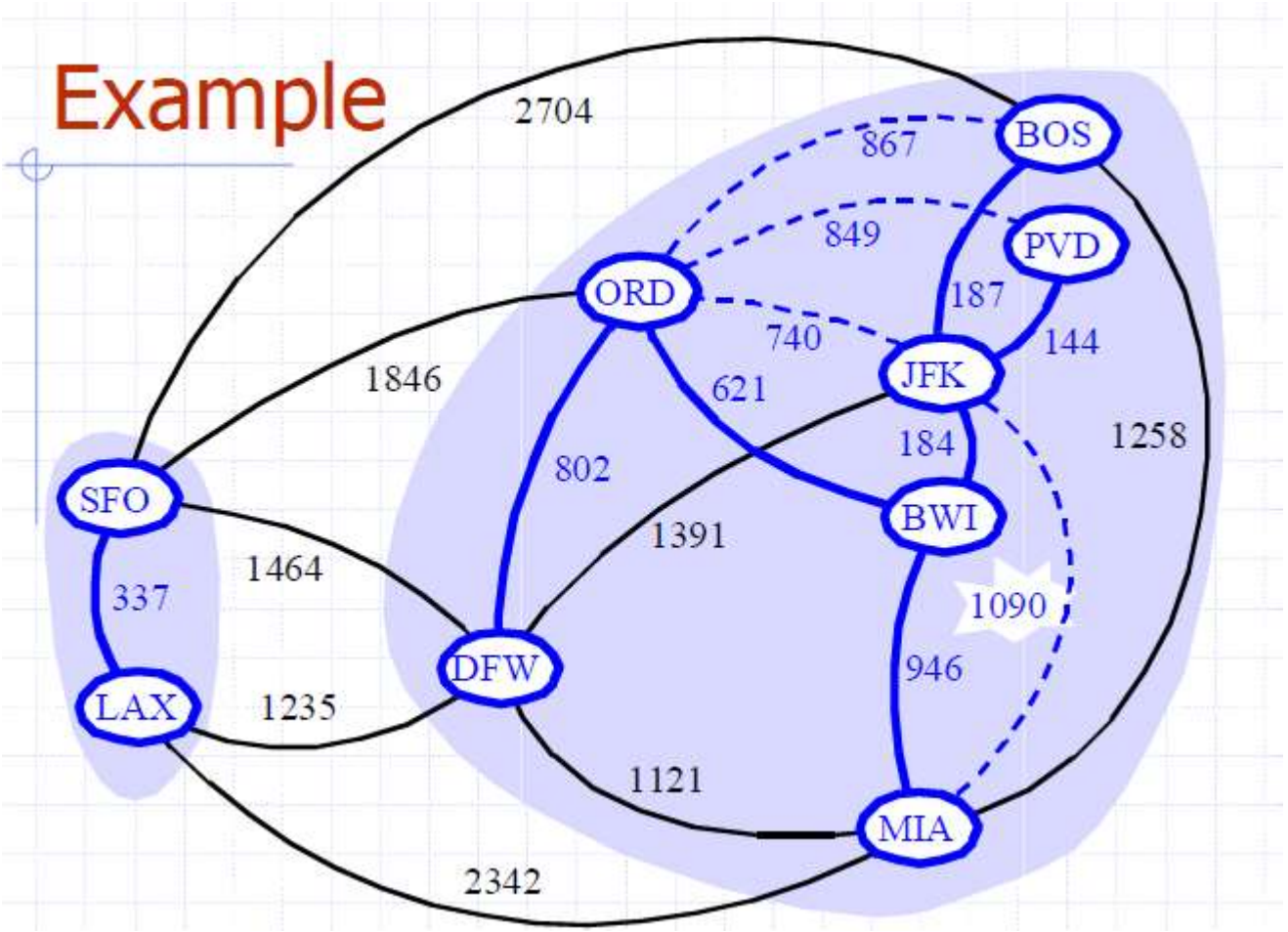
Minimum Spanning Tree



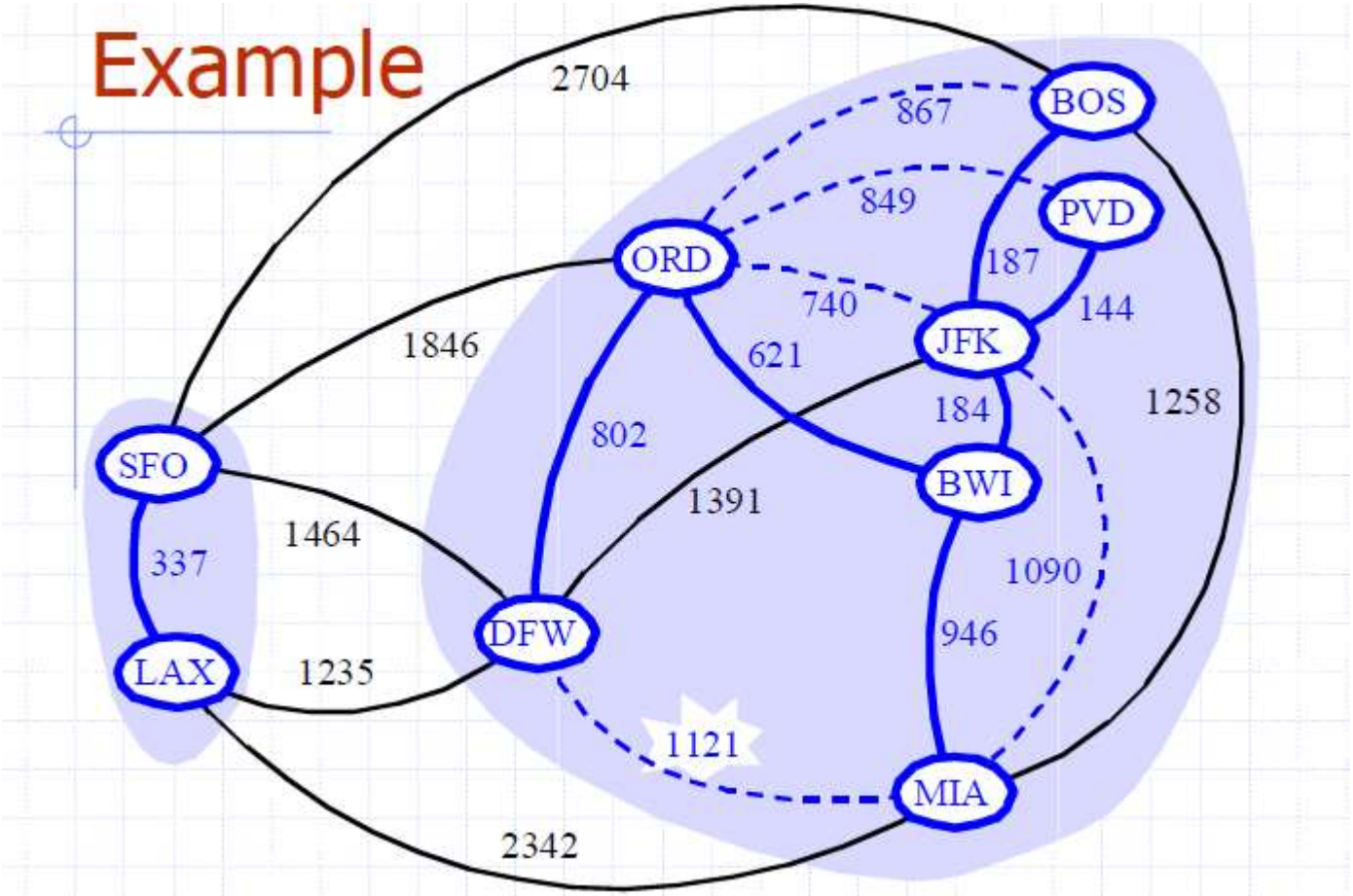
Minimum Spanning Tree



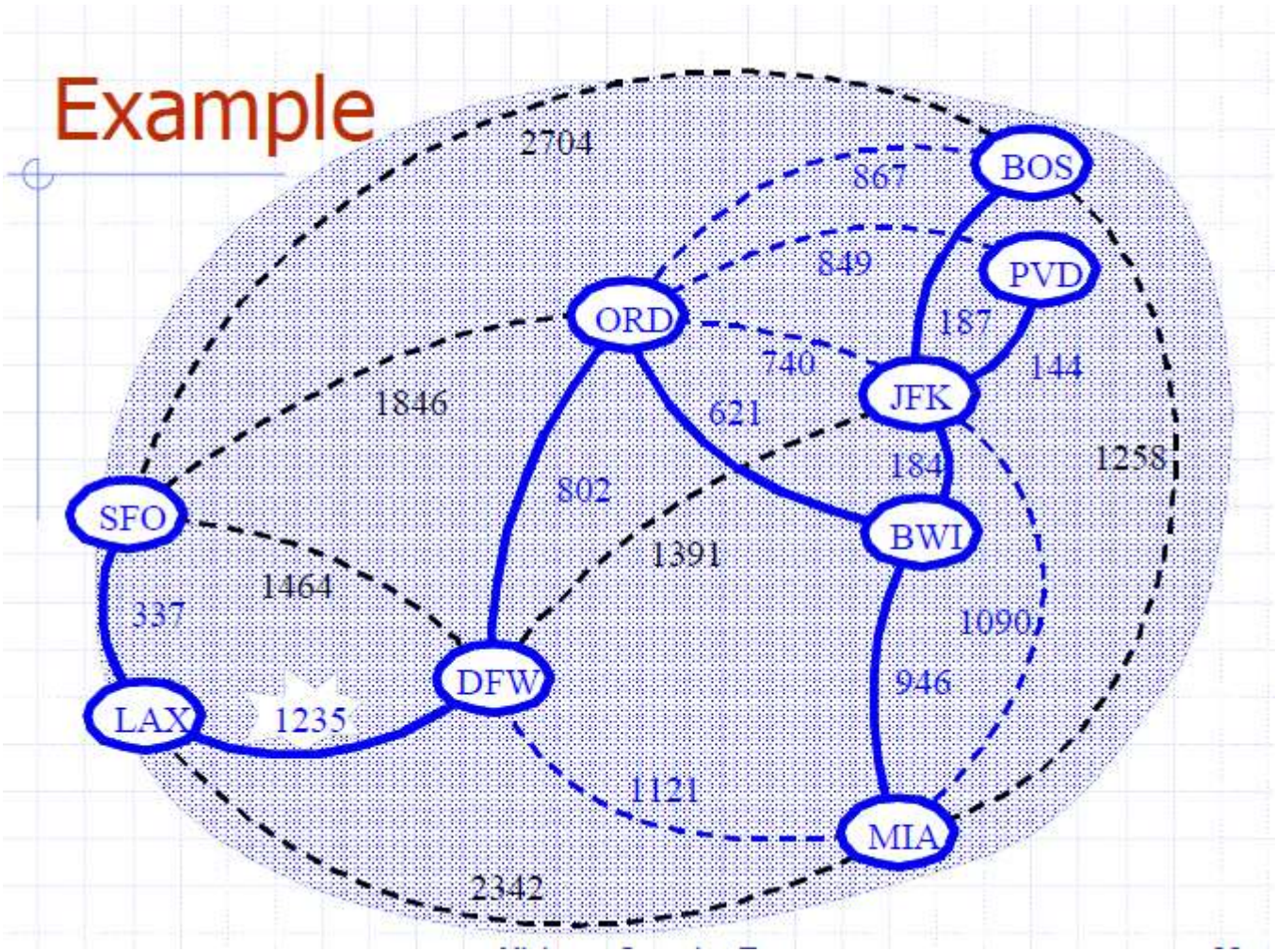
Minimum Spanning Tree



Minimum Spanning Tree



Minimum Spanning Tree



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.

MST-Problem -HW

MST QN