CS 400

Graphs: Minimum Spanning Trees
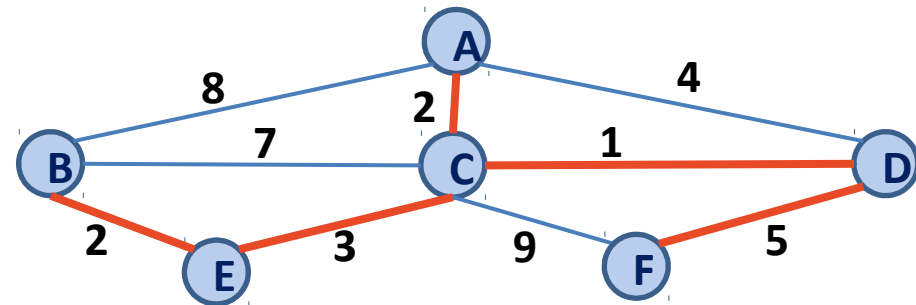
ID: 14-01

Minimum Spanning Tree Algorithms

Input: Connected, undirected graph **G** with edge weights (unconstrained, but must be additive)

Output: A graph G' with the following properties:

- G' is a spanning graph of G
- G' is a tree (connected, acyclic)
- G' has a minimal total weight among all spanning trees

CS 400

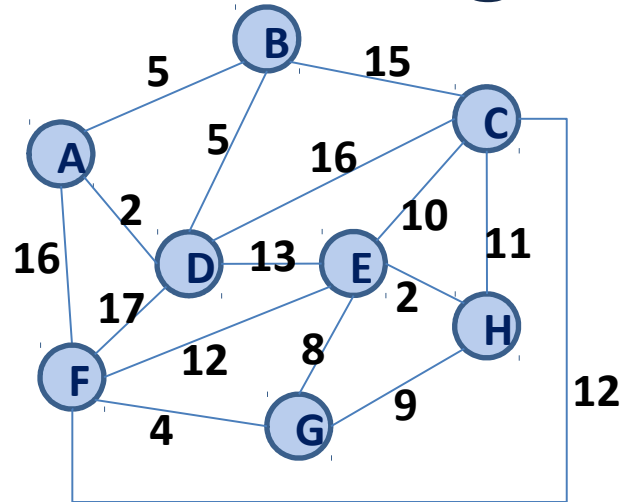**Graphs: MST – Kruskal's Algorithm**

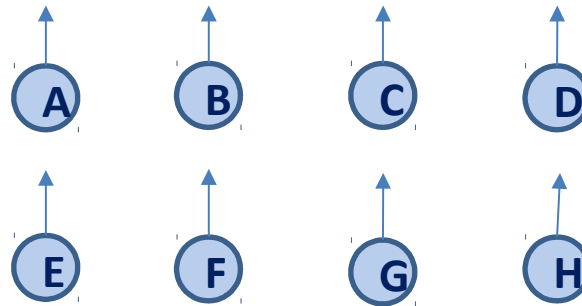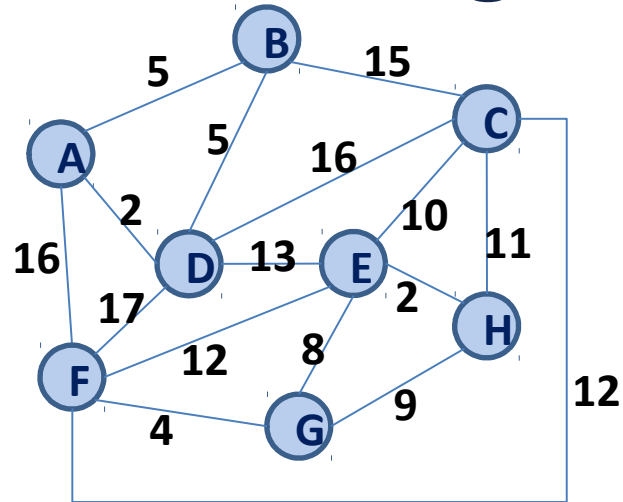**ID: 14-02**

Kruskal's Algorithm

(A, D)
(E, H)
(F, G)
(A, B)
(B, D)
(G, E)
(G, H)
(E, C)
(C, H)
(E, F)
(F, C)
(D, E)
(B, C)
(C, D)
(A, F)
(D, F)

Kruskal's Algorithm
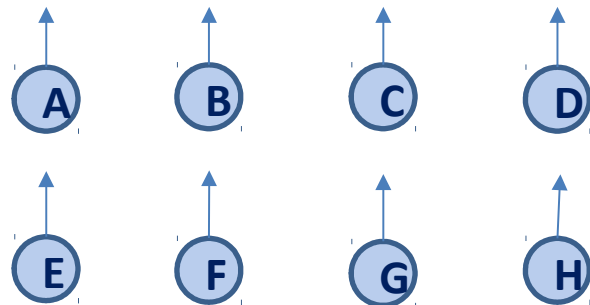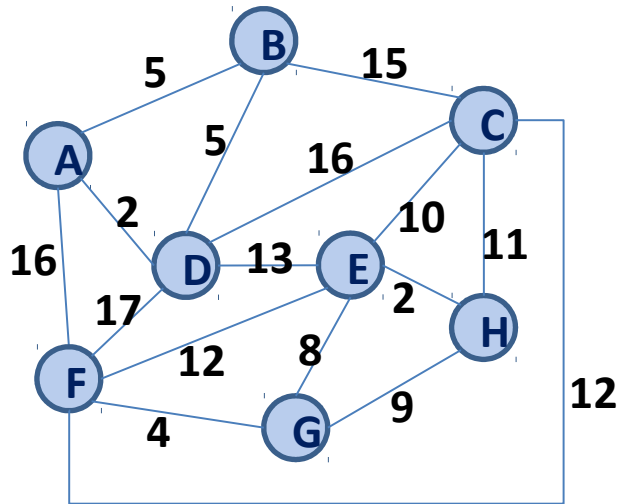
Kruskal's Algorithm

(A, D)
(E, H)
(F, G)
(A, B)
(B, D)
(G, E)
(G, H)
(E, C)
(C, H)
(E, F)
(F, C)
(D, E)
(B, C)
(C, D)
(A, F)
(D, F)


```
1   KruskalMST(G):
2     DisjointSets forest
3     foreach (Vertex v : G):
4       forest.makeSet(v)
5
6     PriorityQueue Q     // min edge weight
7     foreach (Edge e : G):
8       Q.insert(e)
9
10    Graph T = (V, {})
11
12    while |T.edges()| < n-1:
13      Edge (u, v) = Q.removeMin()
14      if forest.find(u) != forest.find(v):
15        T.addEdge(u, v)
16        forest.union( forest.find(u),
17                      forest.find(v) )
18
19    return T
```

Kruskal's Algorithm

Priority Queue:	Heap	Sorted Array
Building :6-8		
Each removeMin :13		

```
 1   KruskalMST(G):
 2     DisjointSets forest
 3     foreach (Vertex v : G):
 4       forest.makeSet(v)
 5
 6     PriorityQueue Q     // min edge weight
 7     foreach (Edge e : G):
 8       Q.insert(e)
 9
10     Graph T = (V, {})
11
12     while |T.edges()| < n-1:
13       Edge (u, v) = Q.removeMin()
14       if forest.find(u) != forest.find(v):
15           T.addEdge(u, v)
16           forest.union( forest.find(u),
17                         forest.find(v) )
18
19     return T
```

Kruskal's Algorithm

Priority Queue:	Total Running Time
Heap	
Sorted Array	

```
1   KruskalMST(G):
2     DisjointSets forest
3     foreach (Vertex v : G):
4       forest.makeSet(v)
5
6     PriorityQueue Q     // min edge weight
7     foreach (Edge e : G):
8       Q.insert(e)
9
10    Graph T = (V, {})
11
12    while |T.edges()| < n-1:
13      Edge (u, v) = Q.removeMin()
14      if forest.find(u) != forest.find(v):
15        T.addEdge(u, v)
16        forest.union( forest.find(u),
17                      forest.find(v) )
18
19    return T
```
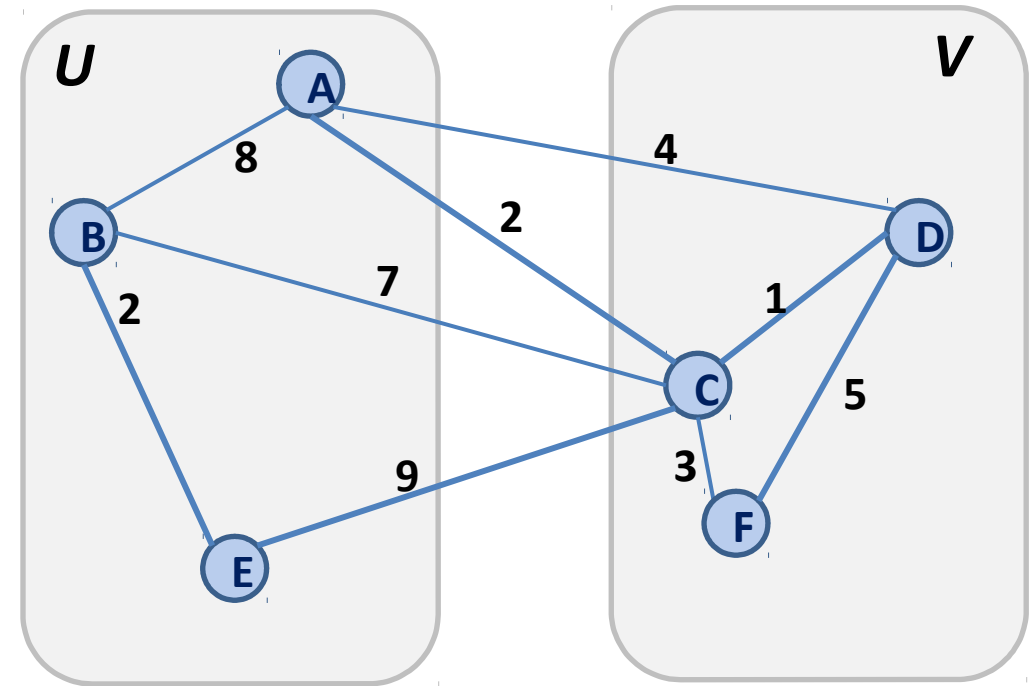
CS 400

Graphs: MST – Prim's Algorithm

ID: 14-03

Partition Property

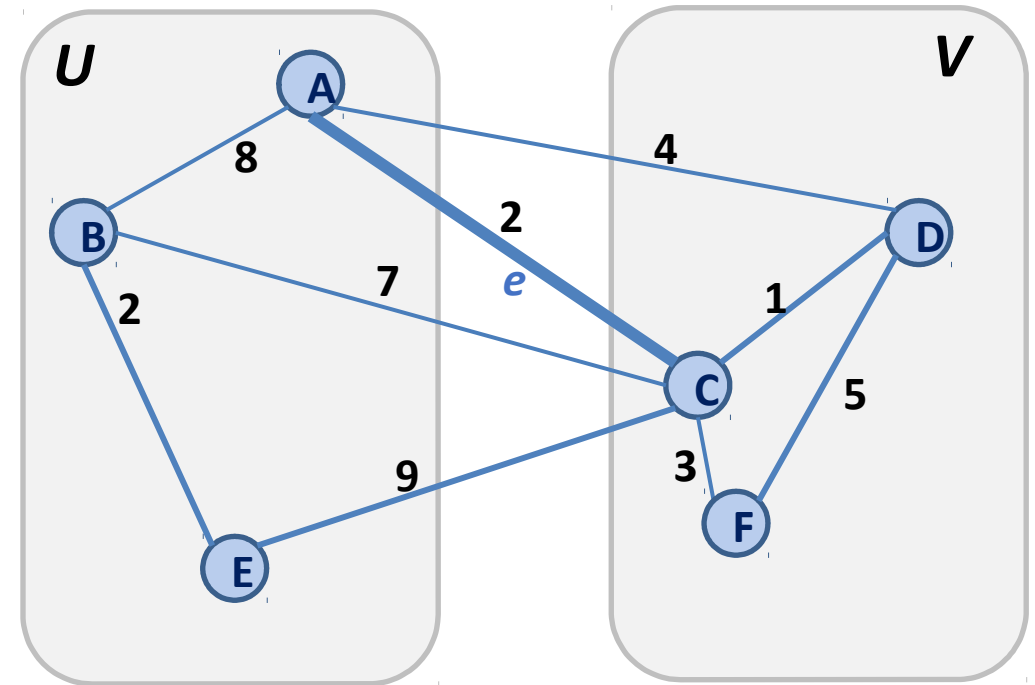Consider an arbitrary partition of the vertices on **G** into two subsets **U** and **V**.

Partition Property

Consider an arbitrary partition of the vertices on **G** into two subsets **U** and **V**.
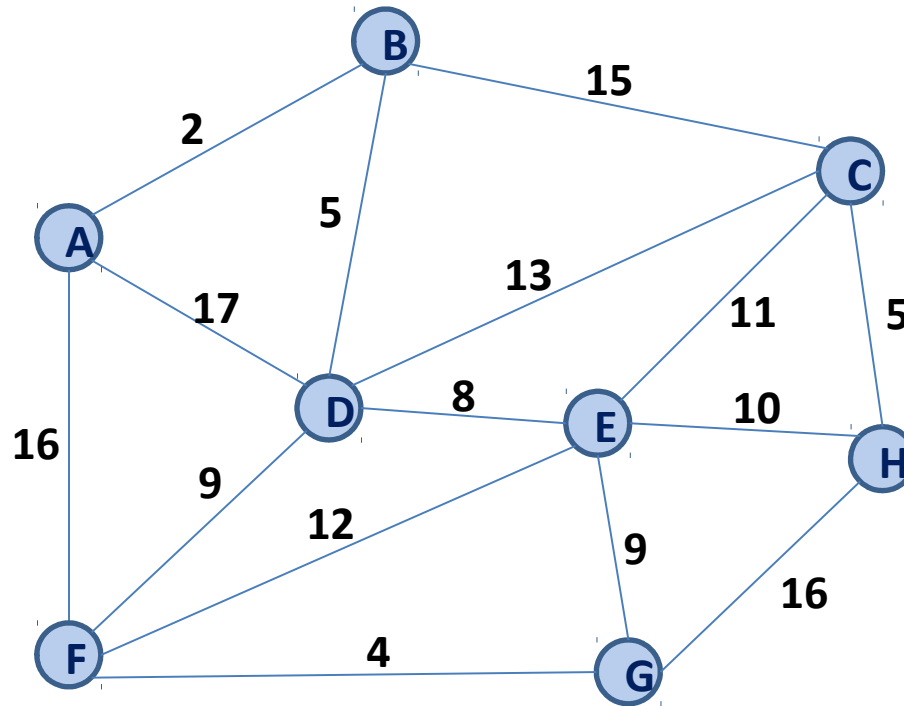
Let **e** be an edge of minimum weight across the partition.

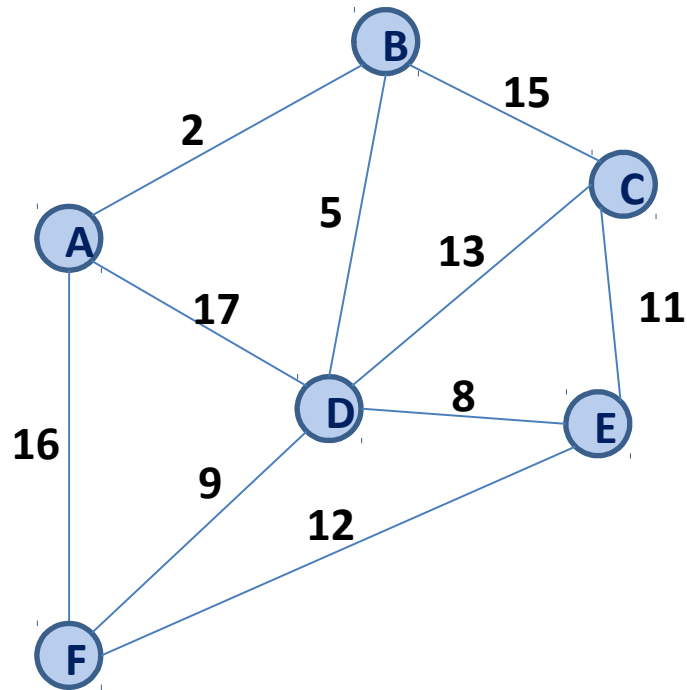Then **e** is part of some minimum spanning tree.

Partition Property

The partition property suggests an algorithm:

Prim's Algorithm


```
1  PrimMST(G, s):
2    Input: G, Graph;
3           s, vertex in G, starting vertex
4    Output: T, a minimum spanning tree (MST) of G
5
6    foreach (Vertex v : G):
7      d[v] = +inf
8      p[v] = NULL
9    d[s] = 0
10
11   PriorityQueue Q    // min distance, defined by d[v]
12   Q.buildHeap(G.vertices())
13   Graph T            // "labeled set"
14
15   repeat n times:
16     Vertex m = Q.removeMin()
17     T.add(m)
18     foreach (Vertex v : neighbors of m not in T):
19       if cost(v, m) < d[v]:
20         d[v] = cost(v, m)
21         p[v] = m
22
23   return T
```

Prim's Algorithm

Sparse Graph:

Dense Graph:

```
6   PrimMST(G, s):
7     foreach (Vertex v : G):
8       d[v] = +inf
9       p[v] = NULL
10    d[s] = 0
11
12    PriorityQueue Q // min distance, defined by d[v]
13    Q.buildHeap(G.vertices())
14    Graph T          // "labeled set"
15
16    repeat n times:
17      Vertex m = Q.removeMin()
18      T.add(m)
19      foreach (Vertex v : neighbors of m not in T):
20        if cost(v, m) < d[v]:
21          d[v] = cost(v, m)
22          p[v] = m
```

	Adj. Matrix	Adj. List
Heap	O(n² + m lg(n))	O(n lg(n) + m lg(n))
Unsorted Array	O(n²)	O(n²)

CS 400

Graphs: MST – Runtime Analysis

ID: 14-04

MST Algorithm Runtime:

- Kruskal's Algorithm:
  O(n + m lg(n))

- Prim's Algorithm:
  O(n lg(n) + m lg(n))

MST Algorithm Runtime:

- Upper bound on MST Algorithm Runtime:
O(m lg(n))