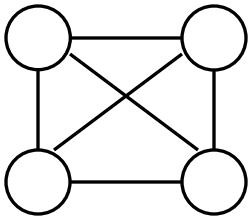
UNIT IV GREEDY TECHNIQUE

Greedy Technique – Minimum Spanning Tree – Prim‘s Algorithm – Kruskal’s Algorithm – Single-source-shortest-paths Problem – Dijkstra’s Algorithm – Huffman Coding – Fractional Knapsack Problem

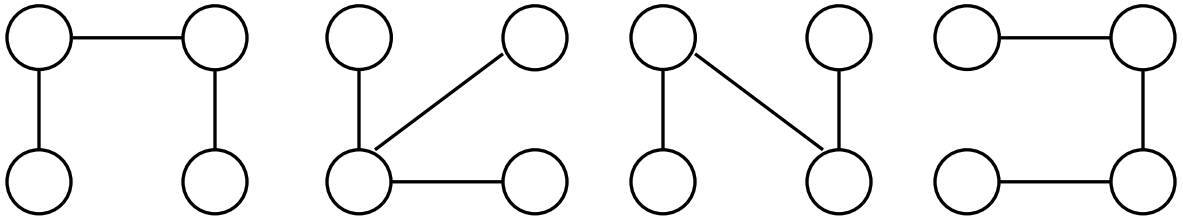
Spanning trees

- Suppose you have a connected undirected graph
  - Connected: every node is reachable from every other node
  - Undirected: edges do not have an associated direction

- ...then a spanning tree of the graph is a connected subgraph in which there are no cycles



A connected, undirected graph



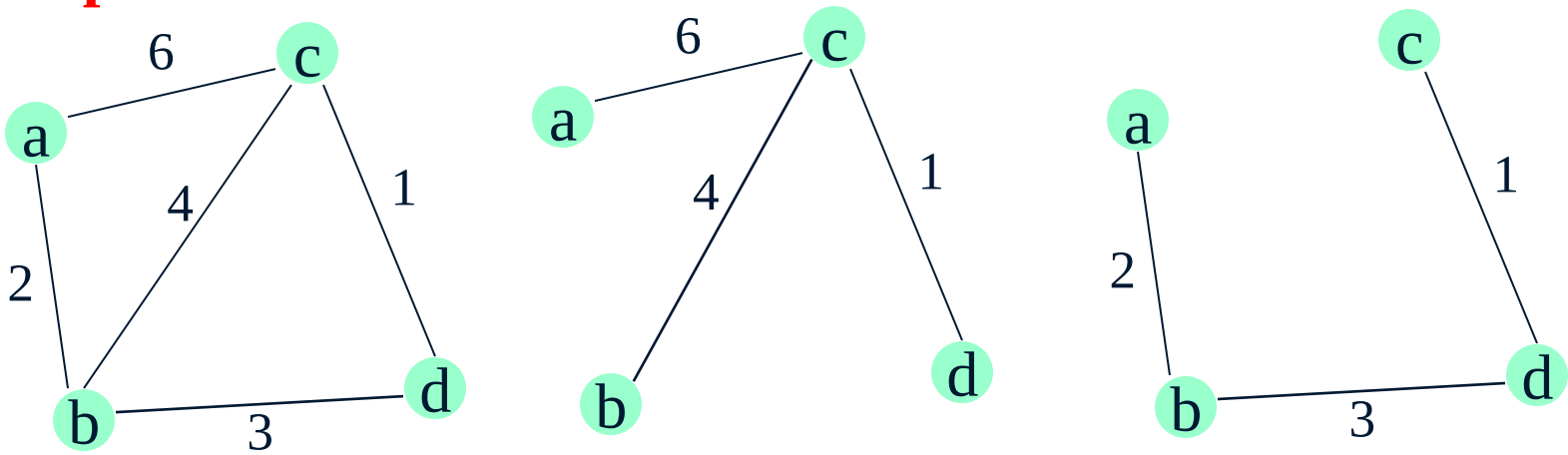
Four of the spanning trees of the graph

Minimum Spanning Tree (MST)

- ***Spanning tree*** **of a connected graph *G*: a connected acyclic subgraph of *G* that includes all of *G*'s vertices**

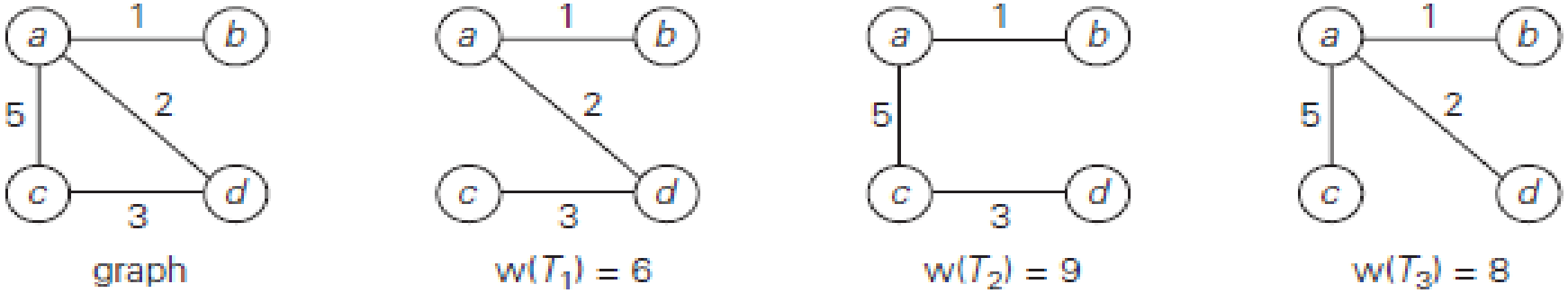
- ***Minimum spanning tree*** **of a weighted, connected graph *G*: a spanning tree of *G* of the minimum total weight**

Example:



Minimum Spanning Tree (MST)

A spanning tree of an undirected connected graph is its connected acyclic subgraph (i.e., a tree) that contains all the vertices of the graph. If such a graph has weights assigned to its edges, a minimum spanning tree is its spanning tree of the smallest weight, where the weight of a tree is defined as the sum of the weights on all its edges. The minimum spanning tree problem is the problem of finding a minimum spanning tree for a given weighted connected graph.

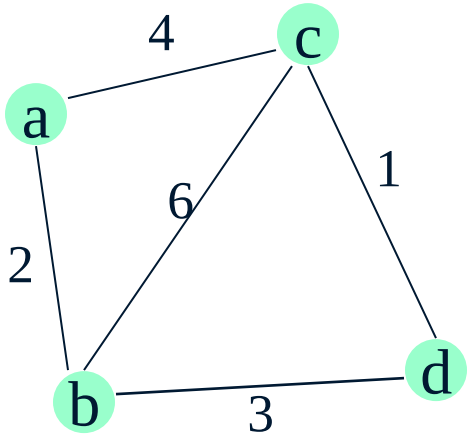


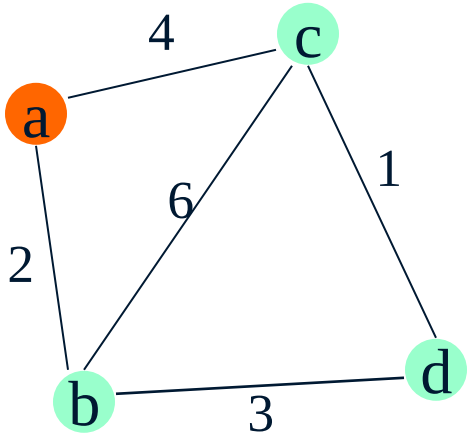
Graph and its spanning trees, with T_1 being the minimum spanning tree.

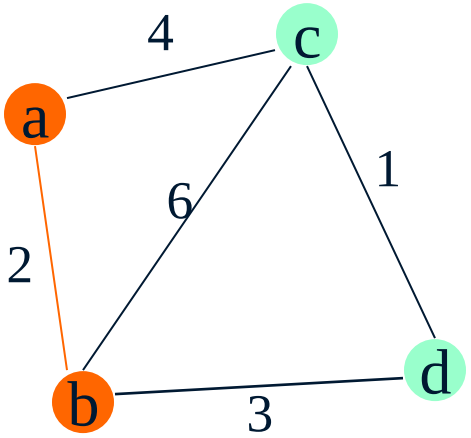
Prim's MST algorithm

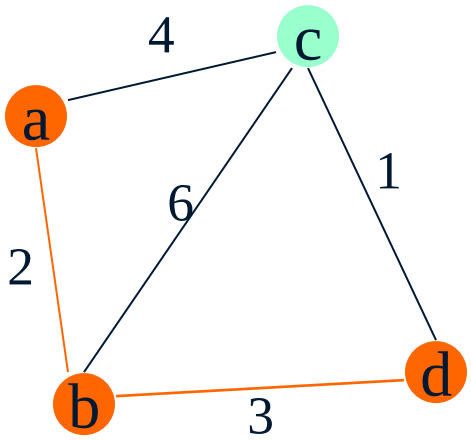
- Start with tree T_1 consisting of one (any) vertex and "grow" tree one vertex at a time to produce MST through a series of expanding subtrees $T_1, T_2, \ldots, T_n$
- On each iteration, construct T_{i+1} from T_i by adding vertex not in T_i that is closest to those already in T_i (this is a "greedy" step!)
- Stop when all vertices are included

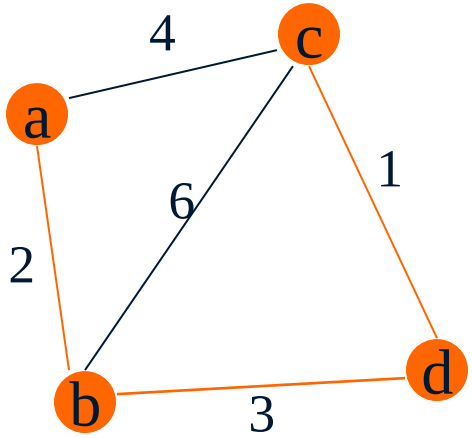
Example











Notes about Prim's algorithm

- Proof by induction that this construction actually yields an MST (CLRS, Ch. 23.1). Main property is given in the next page.

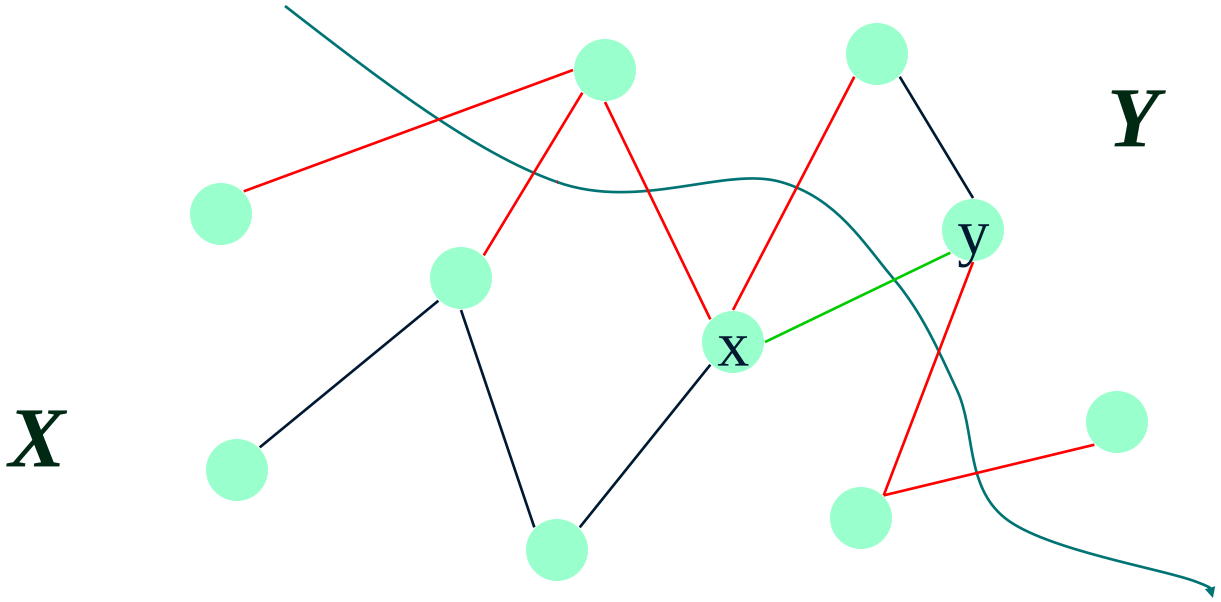
- Needs priority queue for locating closest fringe vertex. The Detailed algorithm can be found in Levitin, P. 310.

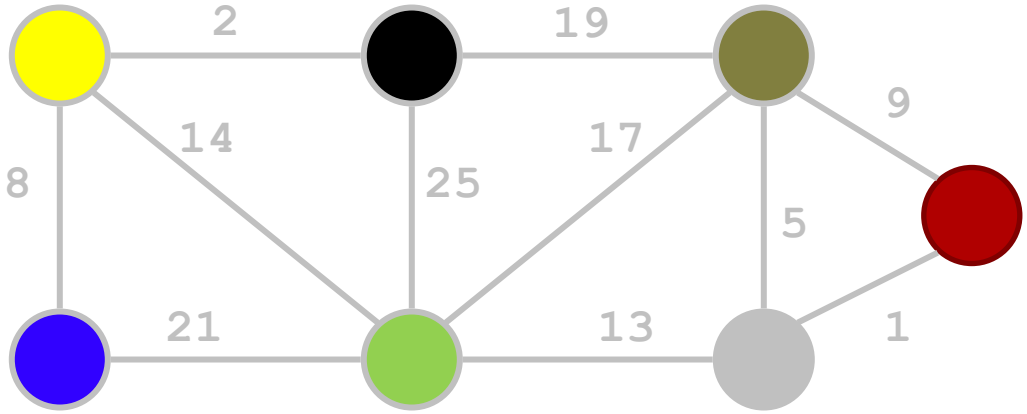
- Efficiency
  - $O(n^2)$ for weight matrix representation of graph and array implementation of priority queue
  - $O(m \log n)$ for adjacency list representation of graph with n vertices and m edges and min-heap implementation of the priority queue

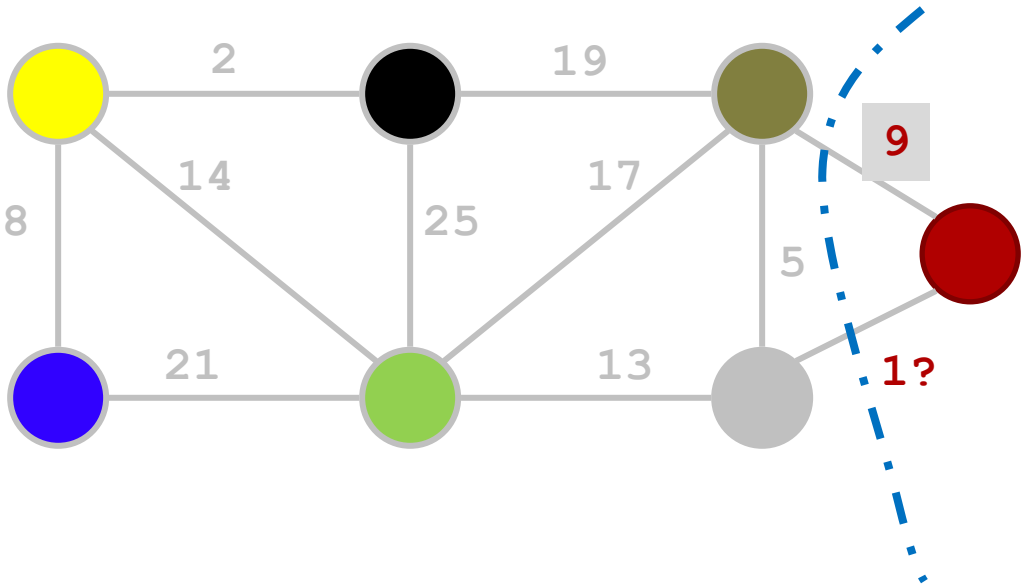
The Crucial Property behind Prim's Algorithm

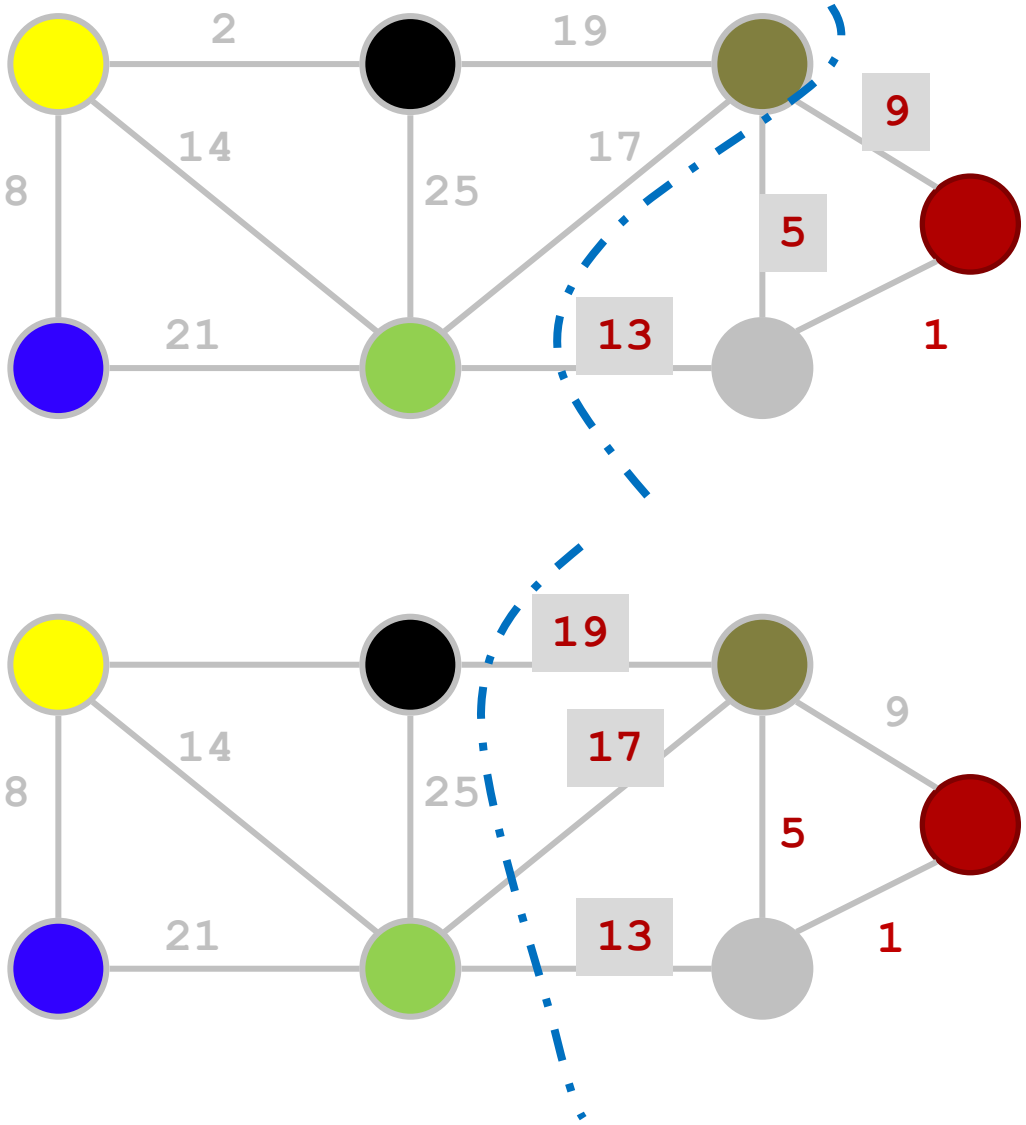
Claim:

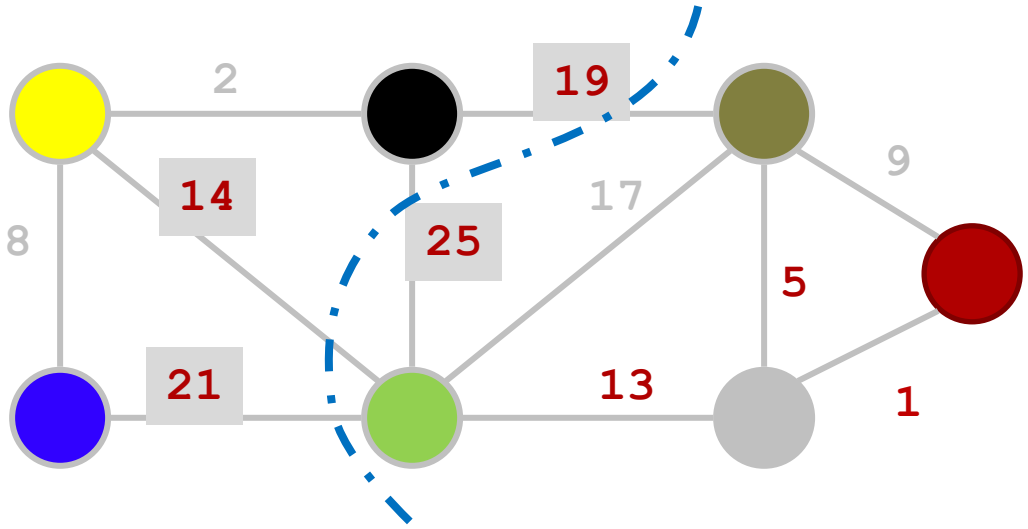
Let *G = (V,E)* be a weighted graph and *(X,Y)* be a partition of V (called a *cut*). Suppose *e = (x,y)* is an edge of *E* across the cut, where *x* is in *X* and *y* is in *Y*, and *e* has the minimum weight among all such crossing edges (called a light edge). Then there is an MST containing *e*.

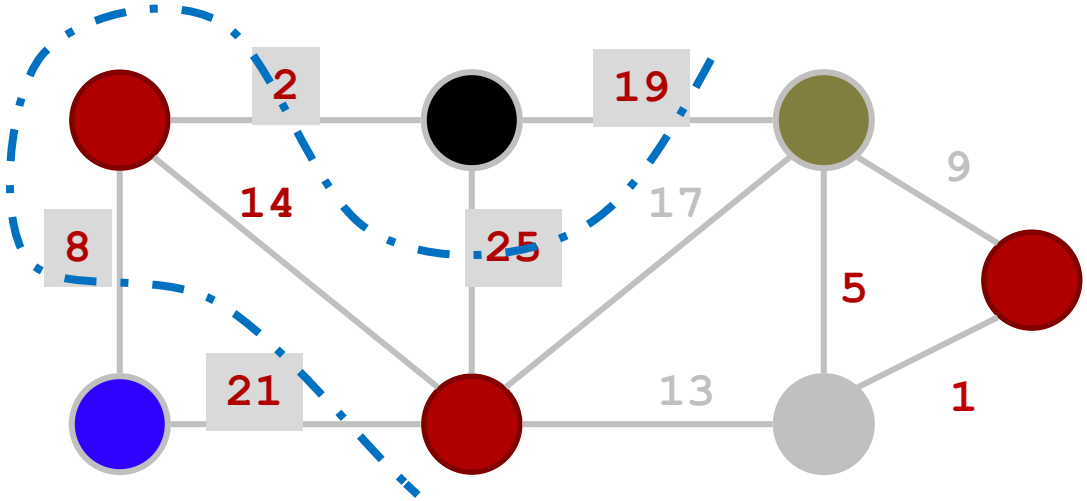


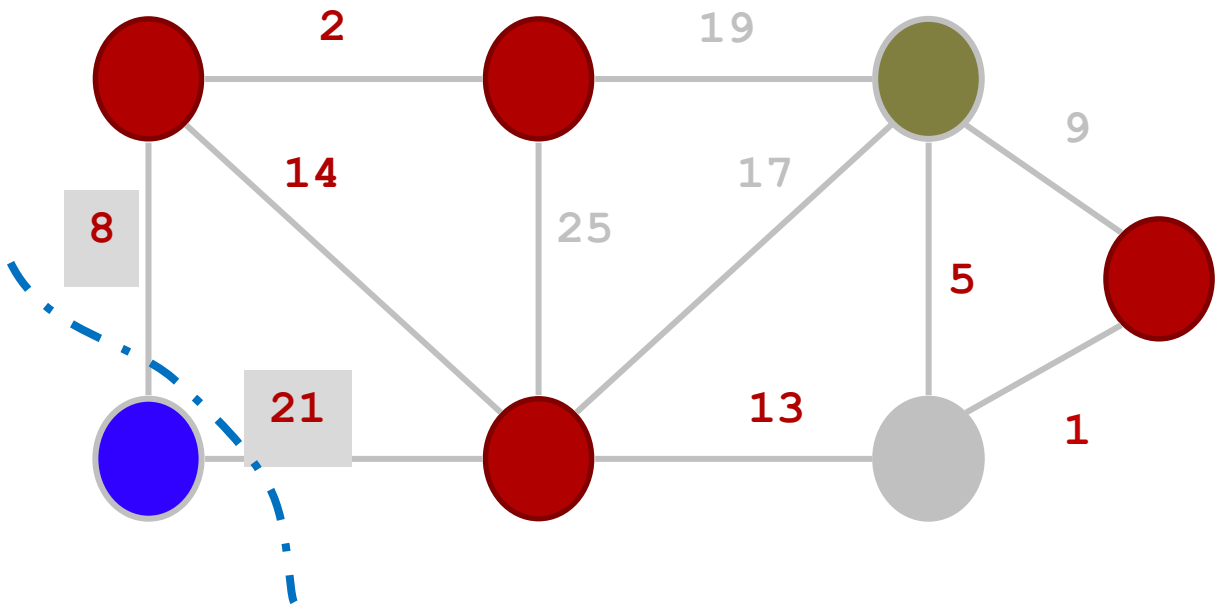

2
19
9
14
17
8
25
5
21
13
1

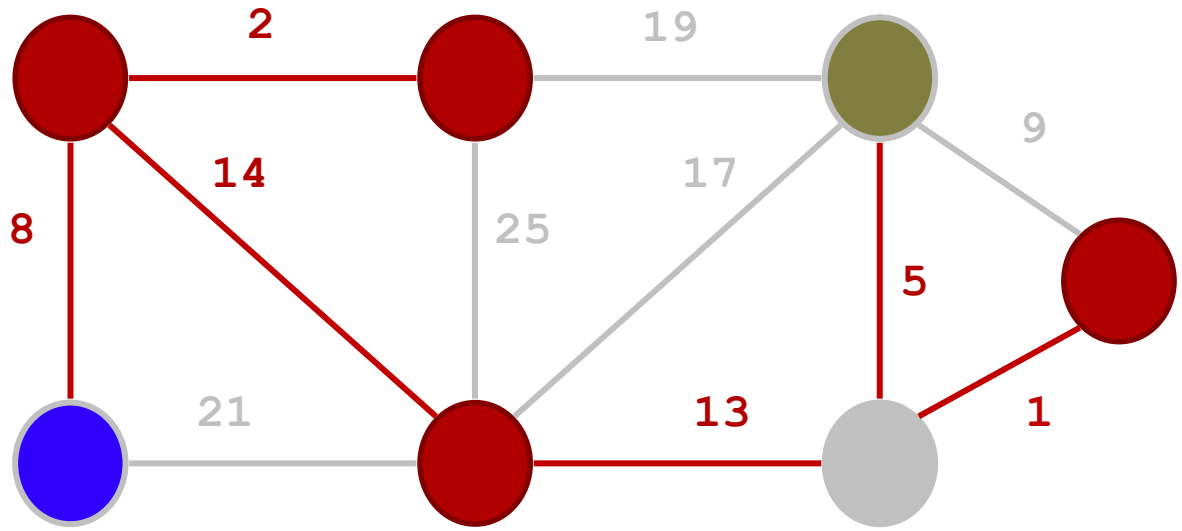

2
19
9
14
17
8
25
5
21
13
1?


2
19
9
14
17
8
25
5
21
13
1
19
9
14
17
8
25
5
21
13
1

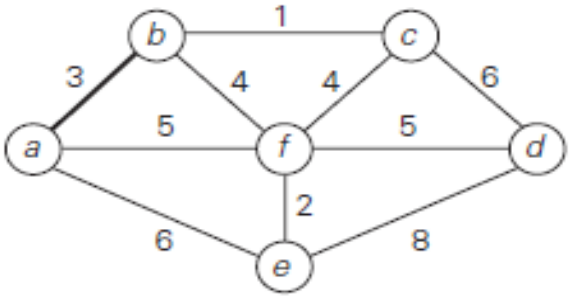
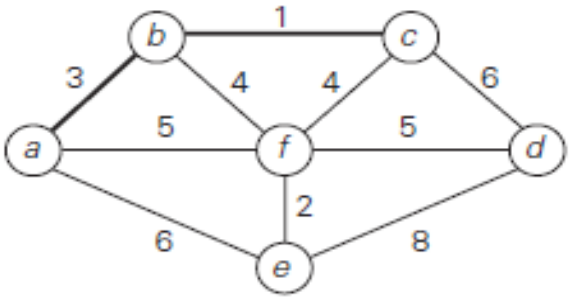
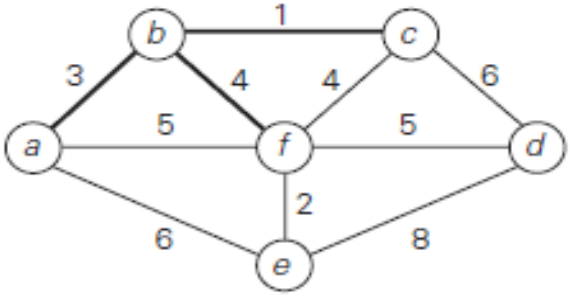

2
19
14
17
9
8
25
5
21
13
1

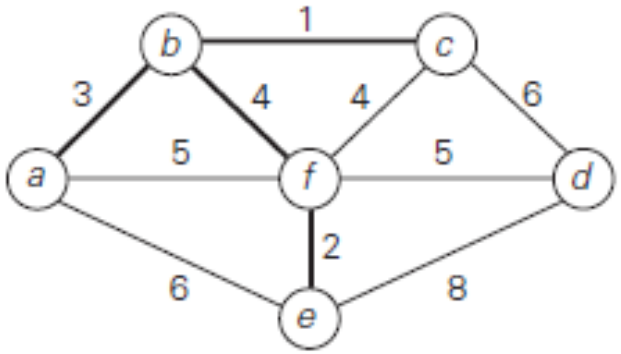
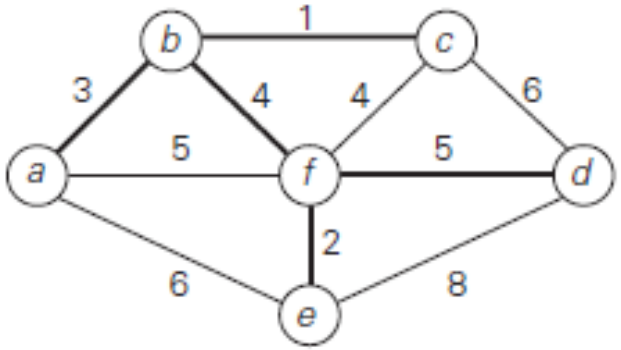

2
19
14
17
9
8
25
5
21
13
1





Cost
1+5+13+14+2+8
= 43

Tree vertices	Remaining vertices	Illustration
a(−, −)	**b(a, 3)** c(−, ∞) d(−, ∞) e(a, 6) f(a, 5)	
b(a, 3)	**c(b, 1)** d(−, ∞) e(a, 6) f(b, 4)	
c(b, 1)	d(c, 6) e(a, 6) **f(b, 4)**	

Tree vertices	Remaining vertices	Illustration
f(b, 4)	d(f, 5) **e(f, 2)**	
e(f, 2)	**d(f, 5)**	
d(f, 5)		

Cost: 3+1+4+2+5 = 15

Kruskal's Algorithm

- Kruskal's algorithm after Joseph Kruskal, who discovered this algorithm.

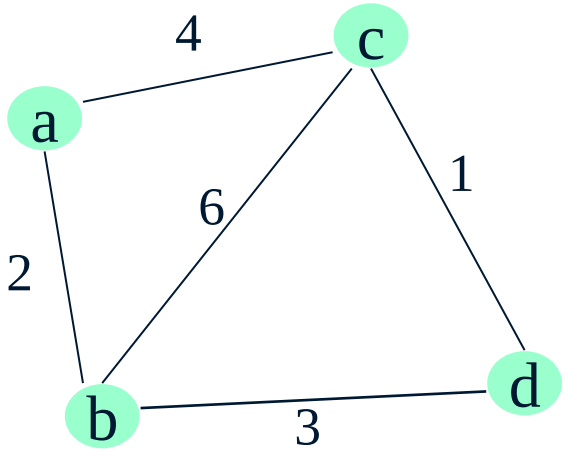
- Kruskal's algorithm looks at a minimum spanning tree of a weighted connected graph G = V, E as an acyclic subgraph with $|V| - 1$ edges for which the sum of the edge weights is the smallest.

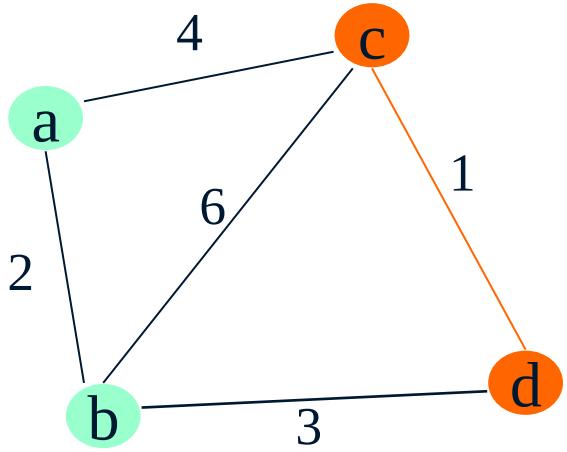
- The algorithm begins by sorting the graph's edges in nondecreasing order of their weights. Then, starting with the empty subgraph, it scans this sorted list, adding the next edge on the list to the current subgraph if such an inclusion does not create a cycle and simply skipping the edge otherwise.

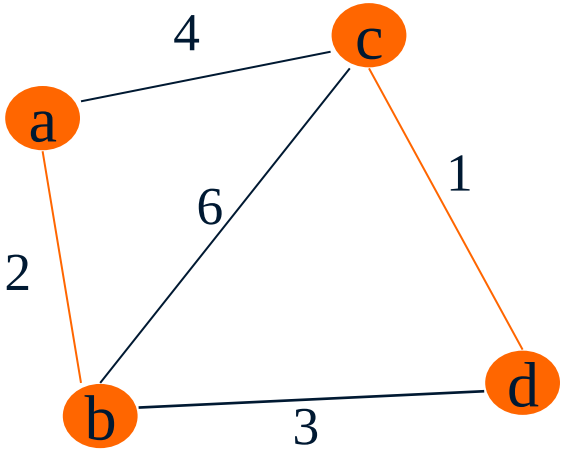
greedy algorithm for MST: Kruskal's

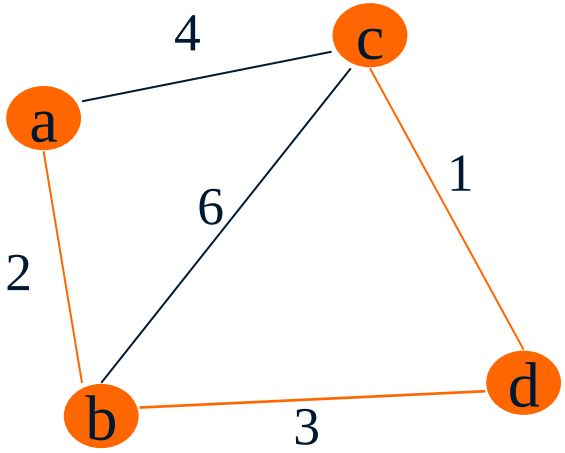
- Sort the edges in nondecreasing order of lengths
- "Grow" tree one edge at a time to produce MST through a series of expanding forests $F_1, F_2, \ldots, F_{n-1}$
- On each iteration, add the next edge on the sorted list unless this would create a cycle. (If it would, skip the edge.)

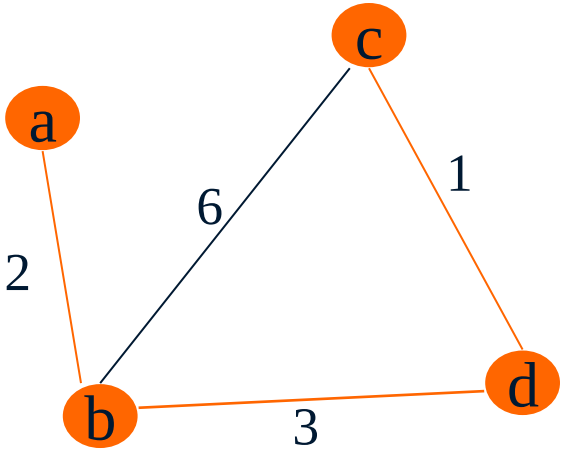
Example

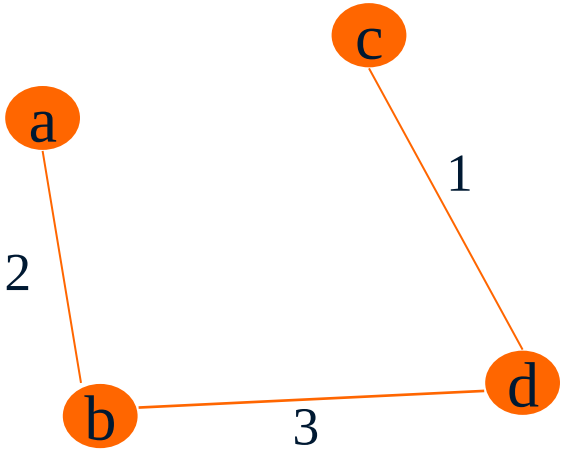

4
c
a
6
1
2
b
3
d


4
c
a
6
1
2
b
3
d


4
c
a
6
1
2
b
3
d

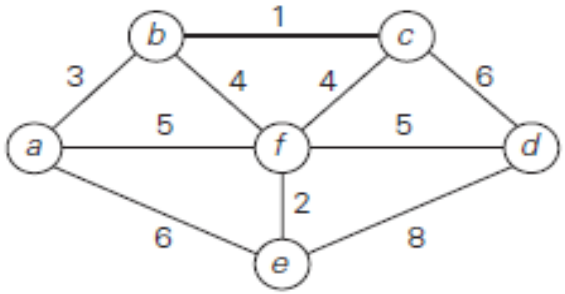
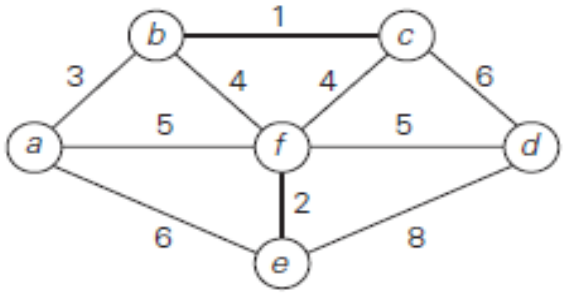
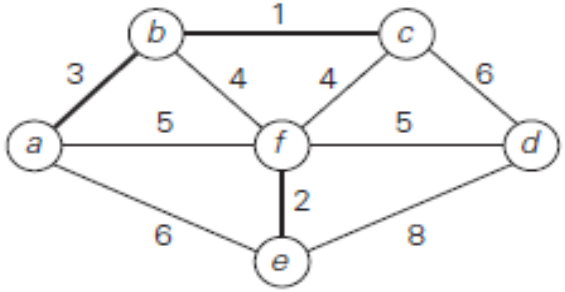

4
c
a
6
1
2
b
3
d

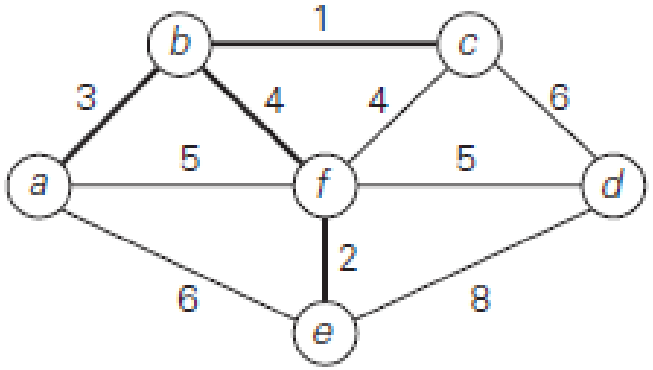
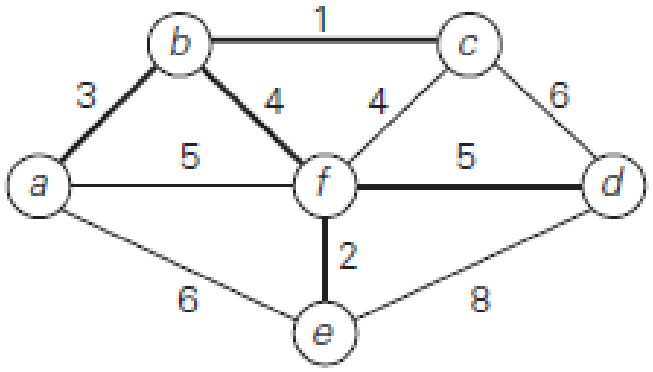

c
a
6
1
2
b
3
d


c
a
1
2
b
3
d

Notes about Kruskal's algorithm

- **Algorithm looks easier than Prim's but is harder to implement (checking for cycles!)**
- **Cycle checking: a cycle is created iff added edge connects vertices in the same connected component**
- ***Union-find* algorithms – see section 9.2**
- **Runs in $O(m \log m)$ time, with $m = |E|$. The time is mostly spent on sorting.**

Tree edges	Sorted list of edges	Illustration
	bc ef ab bf cf af df ae cd de 1 2 3 4 4 5 5 6 6 8	
bc 1	bc **ef** ab bf cf af df ae cd de 1 2 3 4 4 5 5 6 6 8	
ef 2	bc ef **ab** bf cf af df ae cd de 1 2 3 4 4 5 5 6 6 8	

Tree edges	Sorted list of edges	Illustration
ab 3	bc ef ab **bf** cf af df ae cd de 1 2 3 4 4 5 5 6 6 8	
bf 4	bc ef ab bf cf af **df** ae cd de 1 2 3 4 4 5 5 6 6 8	
df 5		


Thank You