

# Design and Analysis of Algorithms

## Lecture-26

Dharmendra Kumar (Associate Professor)

Department of Computer Science and Engineering

United College of Engineering and Research,

Prayagraj

# **Unit-3**

## **Greedy Algorithm**

# Greedy Algorithm

- A ***greedy algorithm*** always makes the choice that looks best at the moment. That is, it makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution.
- Greedy algorithms are used to solve optimization problems.
- Greedy algorithms do not always yield optimal solutions, but for many problems they do.
- We will solve the following problems using greedy algorithm:-
  - Activity selection problem
  - Knapsack problem
  - Minimum spanning tree problem
  - Shortest path problem

# Activity selection problem

- Suppose we have a set  $S = \{a_1, a_2, a_3, \dots, a_n\}$  of  $n$  proposed **activities** that wish to use a resource, such as a lecture hall, which can serve only one activity at a time.
- Each activity  $a_i$  has a **start time**  $s_i$  and a **finish time**  $f_i$ , where  $0 \leq s_i < f_i < \infty$ .
- Activities  $a_i$  and  $a_j$  are **compatible** if the intervals  $[s_i, f_i)$  and  $[s_j, f_j)$  do not overlap. That is,  $a_i$  and  $a_j$  are compatible if  $s_i \geq f_j$  or  $s_j \geq f_i$ .
- In the **activity-selection problem**, we wish to select a maximum-size subset of mutually compatible activities.

# Activity selection problem

**Example:** For example, consider the following set  $S$  of activities:

$i$	1	2	3	4	5	6	7	8	9	10	11
$s_i$	1	3	0	5	3	5	6	8	8	2	12
$f_i$	4	5	6	7	9	9	10	11	12	14	16

Solve this activity selection problem.'

**Solution:** The subset  $\{a_3, a_9, a_{11}\}$  consists of mutually compatible activities. It is not a maximum subset, however, since the subset  $\{a_1, a_4, a_8, a_{11}\}$  is larger. In fact,  $\{a_1, a_4, a_8, a_{11}\}$  is a largest subset of mutually compatible activities. Another largest subset is  $\{a_2, a_4, a_9, a_{11}\}$ .

# Activity selection problem

## A recursive greedy algorithm

We assume that the  $n$  input activities are already ordered by monotonically increasing finish time. If not, we can sort them into this order in  $O(n \lg n)$  time, breaking ties arbitrarily. Following algorithm solves the activity selection problem recursively.

RECURSIVE-ACTIVITY-SELECTOR( $s, f, k, n$ )

```
1   $m = k + 1$ 
2  while  $m \leq n$  and  $s[m] < f[k]$       // find the first activity in  $S_k$  to finish
3       $m = m + 1$ 
4  if  $m \leq n$ 
5      return  $\{a_m\} \cup \text{RECURSIVE-ACTIVITY-SELECTOR}(s, f, m, n)$ 
6  else return  $\emptyset$ 
```

The initial call, which solves the entire problem, is **RECURSIVE-ACTIVITY-SELECTOR**( $s, f, 0, n$ ).

**Time complexity:** the running time of this algorithm is  **$\theta(n)$** .

# Activity selection problem

## An iterative greedy algorithm

We assume that the  $n$  input activities are already ordered by monotonically increasing finish time. If not, we can sort them into this order in  $O(n \lg n)$  time, breaking ties arbitrarily. Following algorithm solves the activity selection problem iteratively

GREEDY-ACTIVITY-SELECTOR( $s, f$ )

```
1   $n = s.length$ 
2   $A = \{a_1\}$ 
3   $k = 1$ 
4  for  $m = 2$  to  $n$ 
5      if  $s[m] \geq f[k]$ 
6           $A = A \cup \{a_m\}$ 
7           $k = m$ 
8  return  $A$ 
```

**Time complexity:** The running time of this algorithm is  $\theta(n)$ .

# Greedy algorithm property

## Greedy choice property

we can assemble a globally optimal solution by making locally optimal (greedy) choices.

In other words, when we are considering which choice to make, we make the choice that looks best in the current problem, without considering results from subproblems.

## Optimal substructure property

A problem exhibits optimal substructure if an optimal solution to the problem contains within it optimal solutions to subproblems.



# Knapsack problem

- A thief is robbing a store and can carry a maximal weight of  $W$  into his knapsack. There are  $n$  items available in the store and weight of  $i^{th}$  item is  $w_i$  and its profit is  $p_i$ . What items should the thief take?
- Given weights and values of  $n$  items, put these items in a knapsack of capacity  $W$  to get the maximum total value in the knapsack. In other words, given two integer arrays  $val[0..n-1]$  and  $wt[0..n-1]$  which represent values and weights associated with  $n$  items respectively. Also given an integer  $W$  which represents knapsack capacity, find out the maximum value subset of  $val[]$  such that sum of the weights of this subset is smaller than or equal to  $W$ .
- Based on the nature of the items, Knapsack problems are categorized as
  1. Fractional Knapsack
  2. 0-1 Knapsack

# Knapsack problem

## Fractional Knapsack

In this version of Knapsack problem, items can be broken into smaller pieces. So, the thief may take only a fraction  $x_i$  of  $i^{\text{th}}$  item.

## 0-1 Knapsack

In this version of Knapsack problem, we cannot break an item, either pick the complete item or don't pick it (0-1 property).

# Knapsack problem

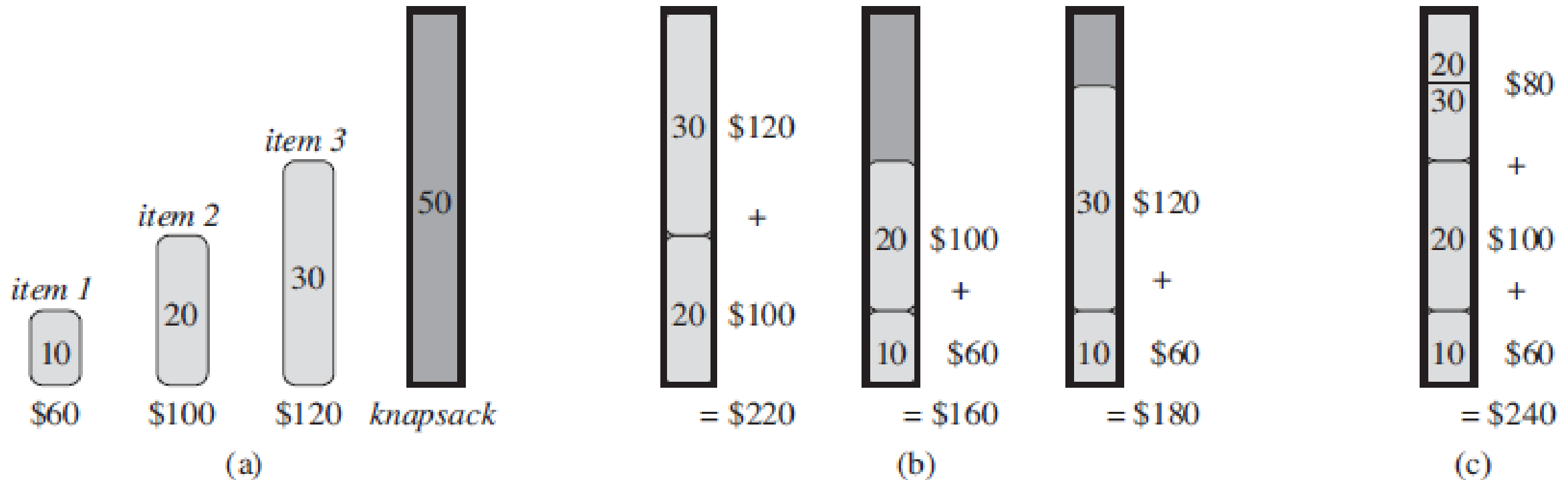
## Greedy approach to solve the knapsack problem

- First, we compute value  $v_i/w_i$  for each item  $i$ .
- Arrange all the items in descending order on the basis of  $v_i/w_i$ .
- Put first item in the knapsack fully if weight of first item is less than or equal to  $W$ . Put first item in the knapsack partially if weight of first item is greater than  $W$ .
- Similarly, put the second item in the knapsack fully if there is a sufficient space in the knapsack. Otherwise, we put second item partially.
- This process continue till Knapsack is filled.

# Knapsack problem

**Note:** Greedy algorithm solves the fractional knapsack but 0-1 Knapsack can not be solved by greedy algorithm.

To see that this greedy strategy does not work for the 0-1 knapsack problem, consider the problem instance illustrated in following figure:-



# Knapsack problem

**Example:** Consider the following instance for knapsack problem. Find the solution using Greedy method:

$N = 8, W = 130$

$W = \{21, 31, 43, 53, 41, 63, 65, 75\}$

$V = \{11, 21, 31, 33, 43, 53, 65, 65\}$

Solution: Compute

$$\frac{V}{W} = \left\{ \frac{11}{21}, \frac{21}{31}, \frac{31}{43}, \frac{33}{53}, \frac{43}{41}, \frac{53}{63}, \frac{65}{65}, \frac{65}{75} \right\}$$
$$= \{ 0.52, 0.68, 0.72, 0.62, 1.05, 0.84, 1, 0.87 \}$$

First arrange all items in descending order of their value per weight i.e.  $(v_i/w_i)$ .

$W' = \{41, 65, 75, 63, 43, 31, 53, 21\}$

$V' = \{43, 65, 65, 53, 31, 21, 33, 11\}$

Descending order of items = item5, item7, item8, item6, item3, item2, item4, item1

Therefore, optimal solution =  $(0, 0, 0, 0, 1, 0, 1, 24/75)$

Optimal value =  $43 + 65 + (24/75)*65 = 108 + 20.8 = 128.8$

# Knapsack problem

**Example:** Consider the following instance for knapsack problem. Find the solution using Greedy method:

$$w = (5, 10, 20, 30, 40), \quad v = (30, 20, 100, 90, 160)$$

The capacity of knapsack  $W = 60$

**Example:** Given the six items in the table below and a Knapsack with Weight 100, what is the solution to the Knapsack problem in all concepts. i.e. explain greedy all approaches and find the optimal solution.

ITEM ID	WEIGHT	VALUE	VALUE/WEIGHT
A	100	40	.4
B	50	35	.7
C	40	20	.5
D	20	4	.2
E	10	10	1
F	10	6	.6

# Knapsack problem

**Algorithm:** Assume all the items are arranged in descending order of their value per weight i.e.  $(v_i/w_i)$ .

Fractional\_Knapsack(w, v, W)

```
n = length[w]
for i=1 to n
  do      x[i] = 0
i=1
weight = 0
while( i <= n and weight < W)
  do      if(weight+w[i] <= W)
            then      x[i] = 1
                      weight = weight + w[i]
            else
                      x[i] = (W - weight)/wi
                      weight = W
return x
```

- Time complexity of this algorithm will be  $O(n)$  if all the items are arranged in descending order of their value per weight .
- Time complexity of this algorithm will be  $O(n \log n)$  if all the items are not arranged in descending order of their value per weight .

# Minimum Spanning Tree

## Spanning tree:

Spanning tree is a non-cyclic sub-graph of a connected and undirected graph  $G$  that connects all the vertices together.

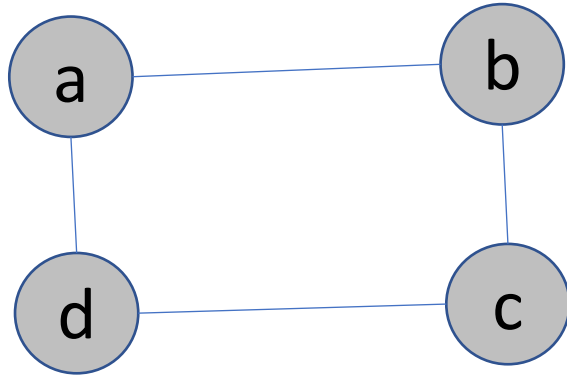
## General Properties of Spanning Tree

- A connected graph  $G$  can have more than one spanning tree.
- All possible spanning trees of graph  $G$ , have the same number of edges and vertices.
- The spanning tree does not have any cycle (loops).
- Removing one edge from the spanning tree will make the graph disconnected, i.e. the spanning tree is **minimally connected**.
- Adding one edge to the spanning tree will create a circuit or loop.
- Spanning tree has  **$n-1$**  edges, where  **$n$**  is the number of nodes (vertices).
- A complete graph can have maximum  **$n^{n-2}$**  number of spanning trees.

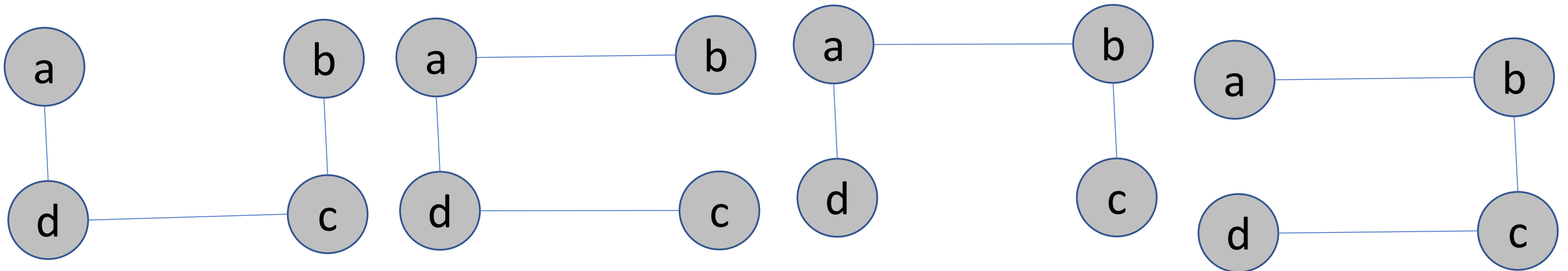


# Minimum Spanning Tree

**Example:** Consider the following graph:-



All the spanning tree of this graph are the following:-

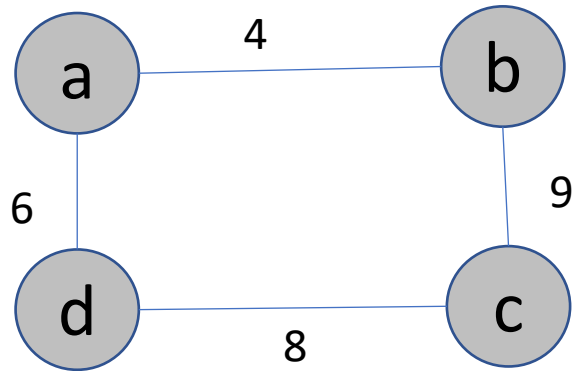


# Minimum Spanning Tree

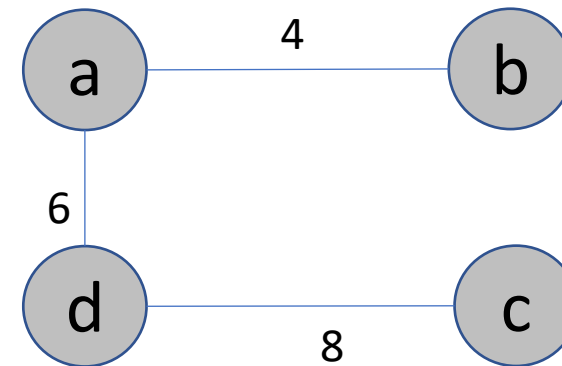
If the given graph is weighted graph, then we define the minimum spanning tree.

**Definition:** A spanning tree is said to be minimum spanning tree if sum of weights of all the edges in the tree is smallest.

**Example:** Consider the following graph:-



**Graph**



**Minimum spanning tree**

**Total cost = 18**

# Minimum Spanning Tree

- In this chapter, we will study two algorithms to find the minimum spanning tree.

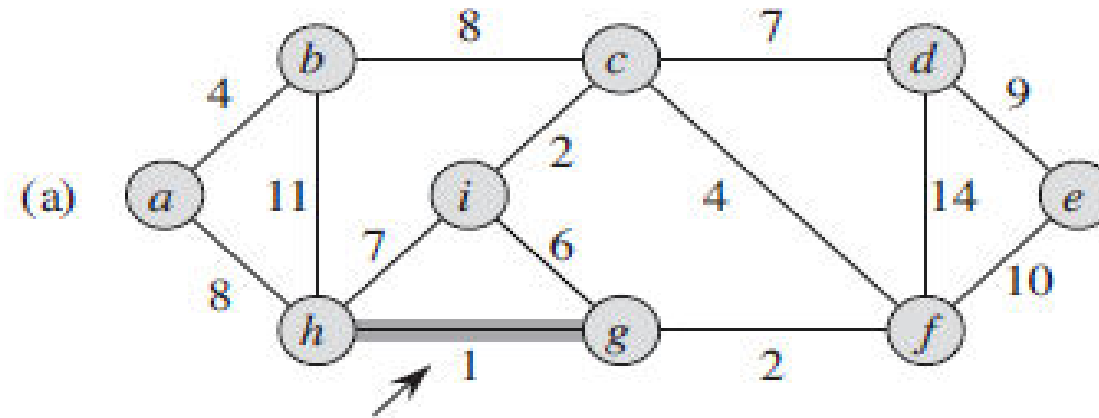
(1) Kruskal's Algorithm

(2) Prim's Algorithm

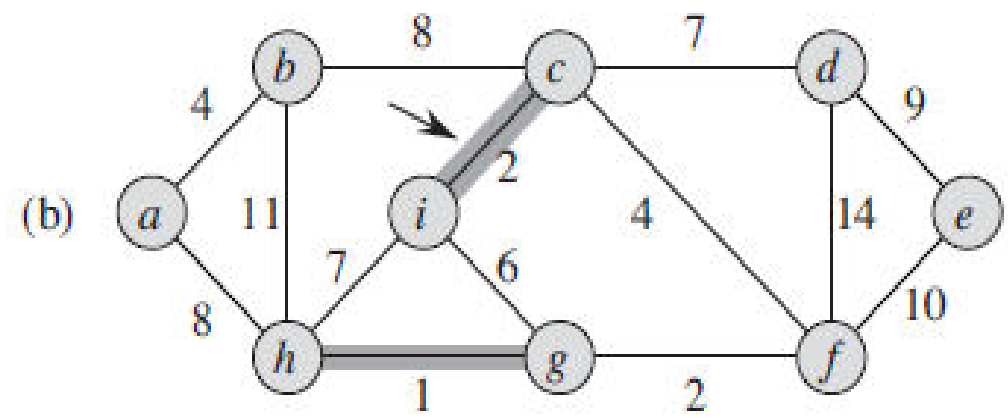
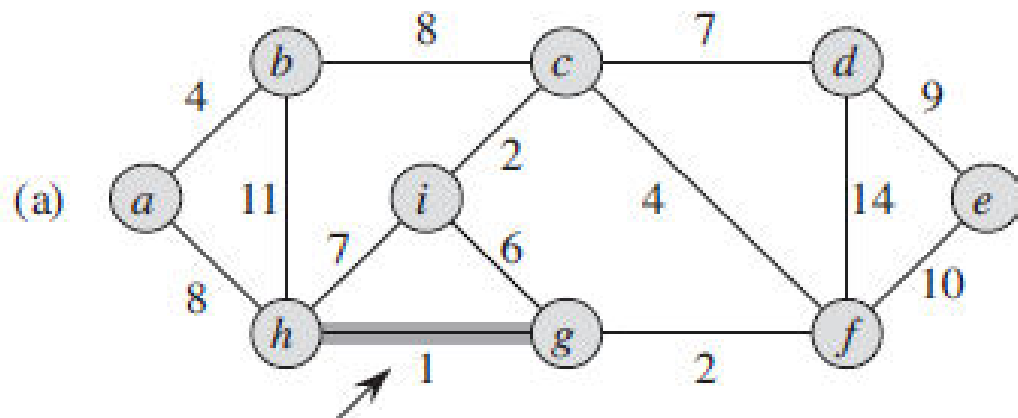
Both algorithms are based on Greedy approach.

# Kruskal's Algorithm

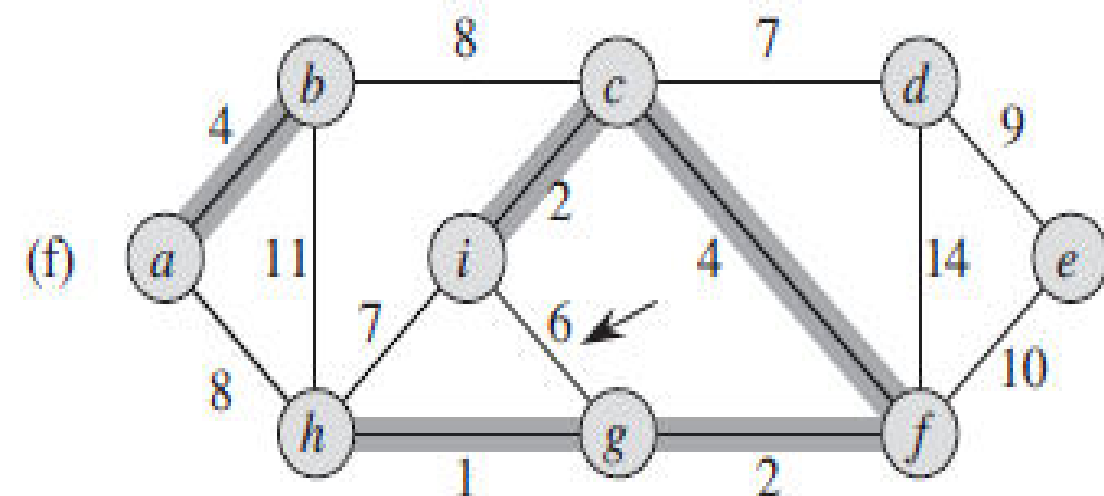
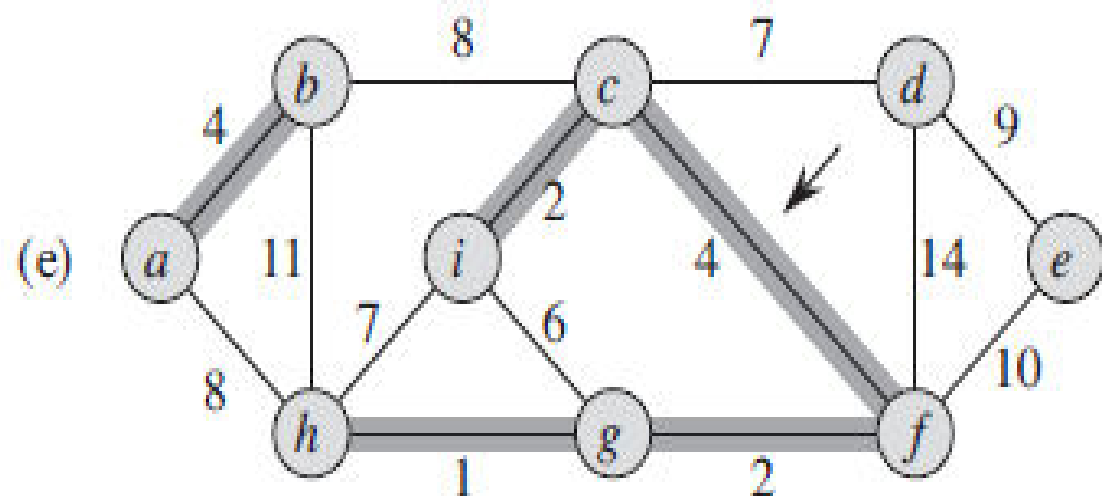
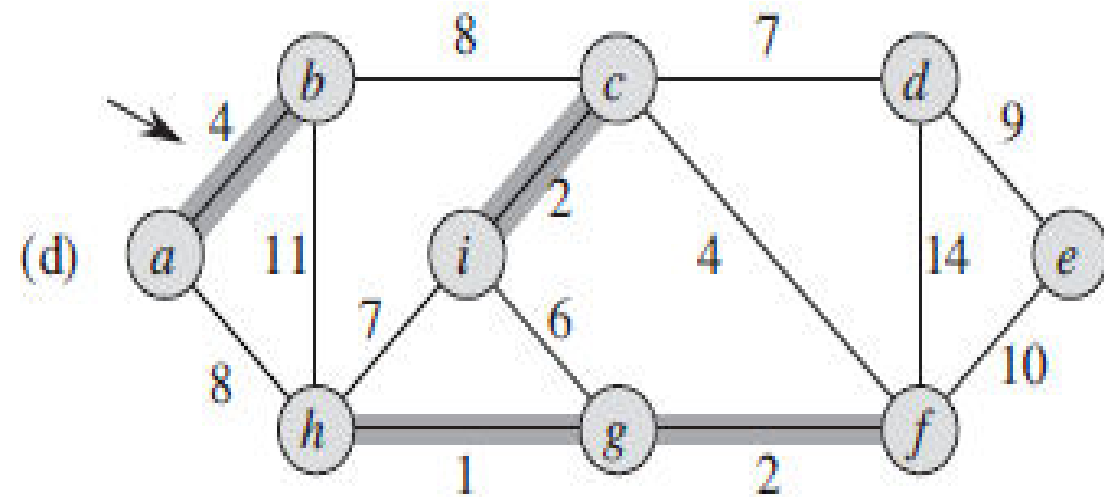
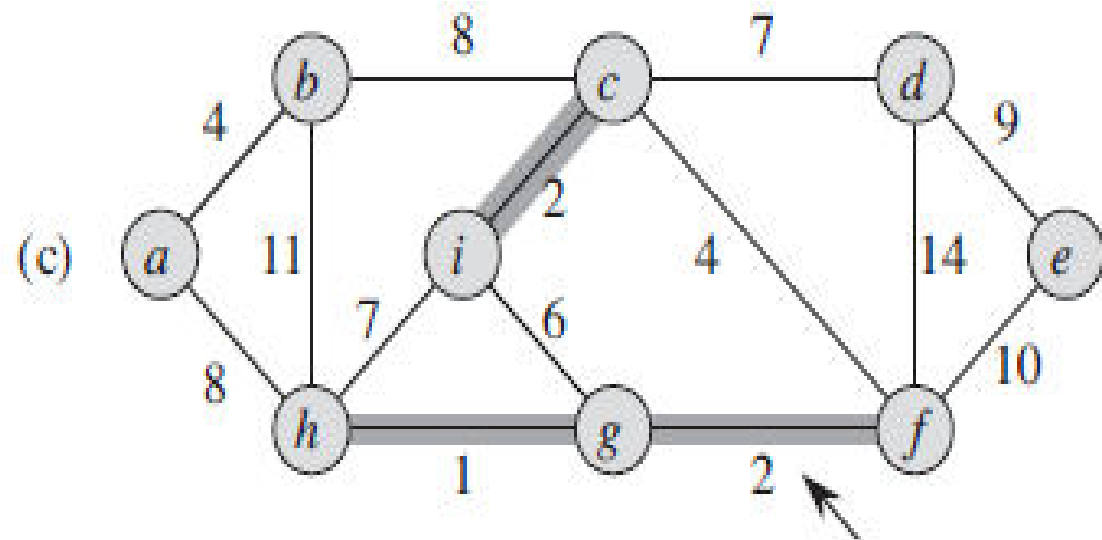
Example: Find the minimum spanning of the following graph using Kruskal algorithm.



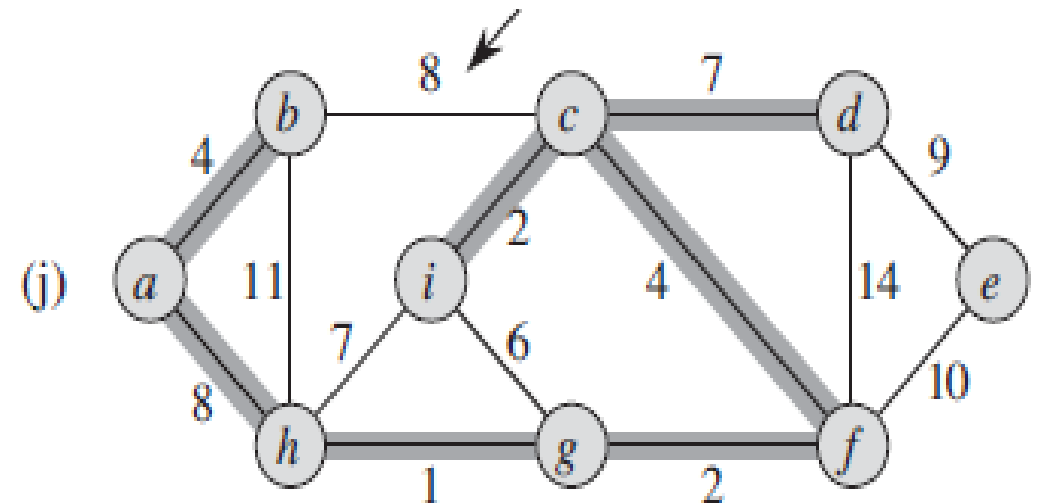
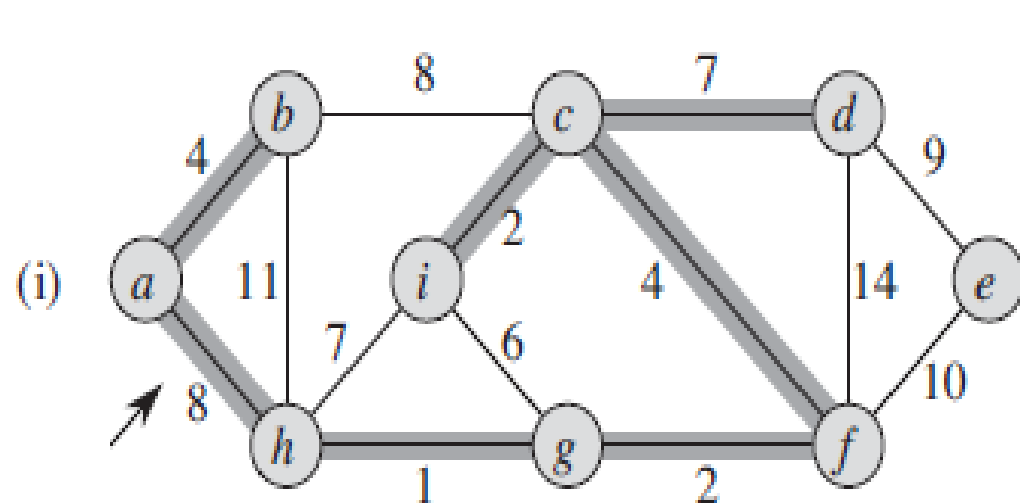
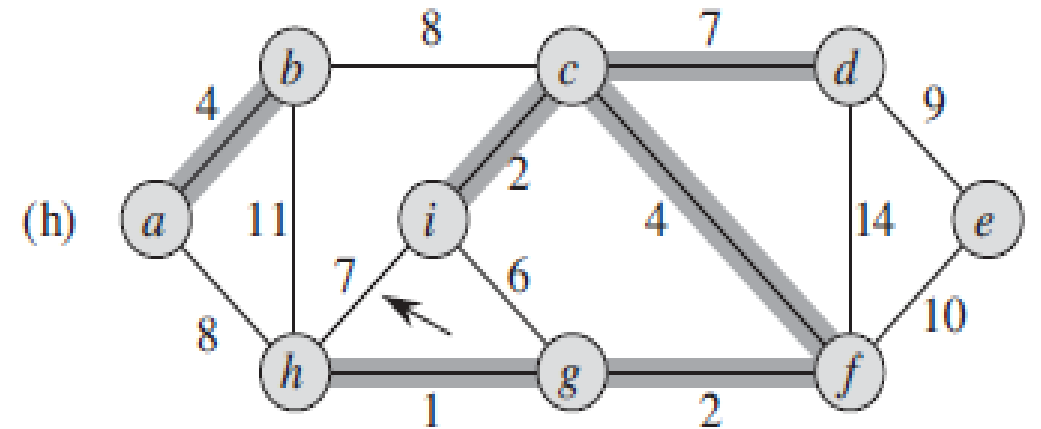
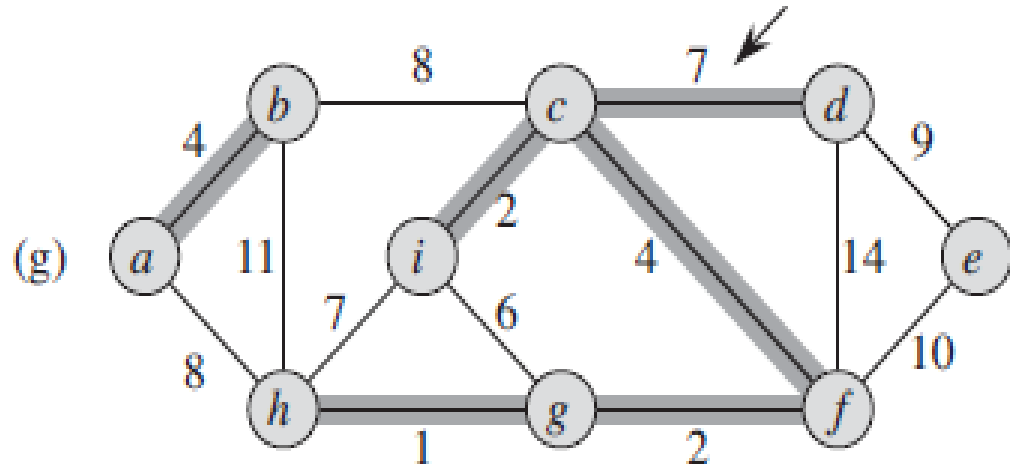
Solution:



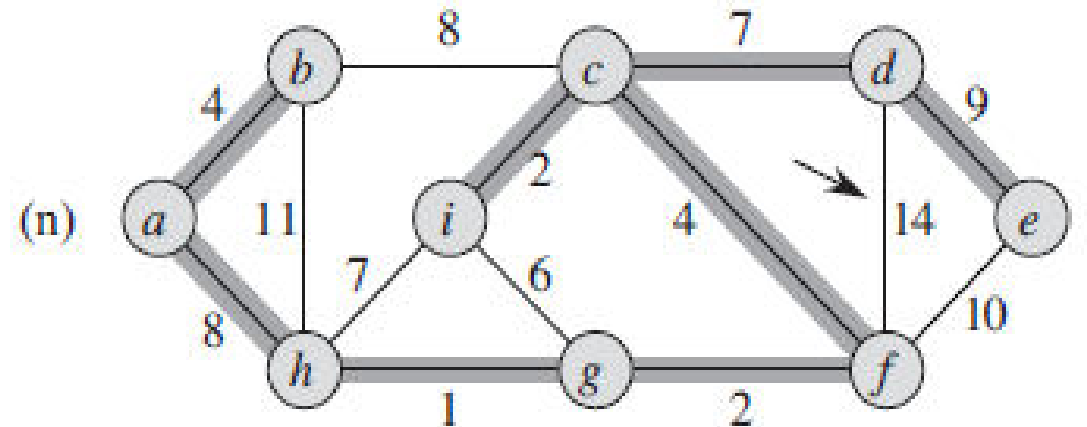
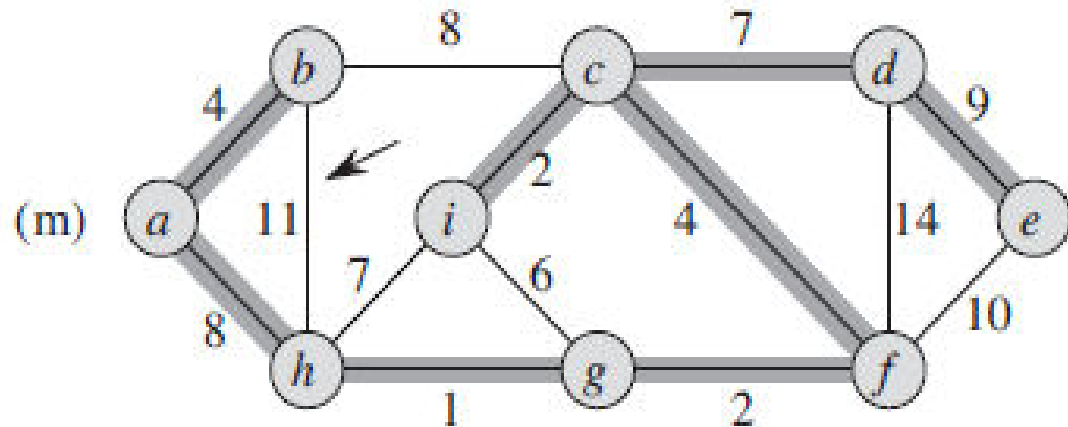
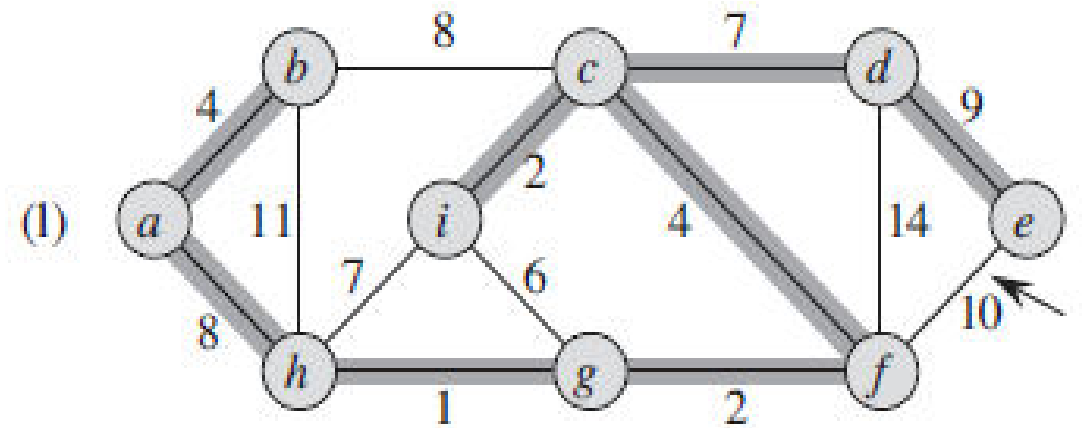
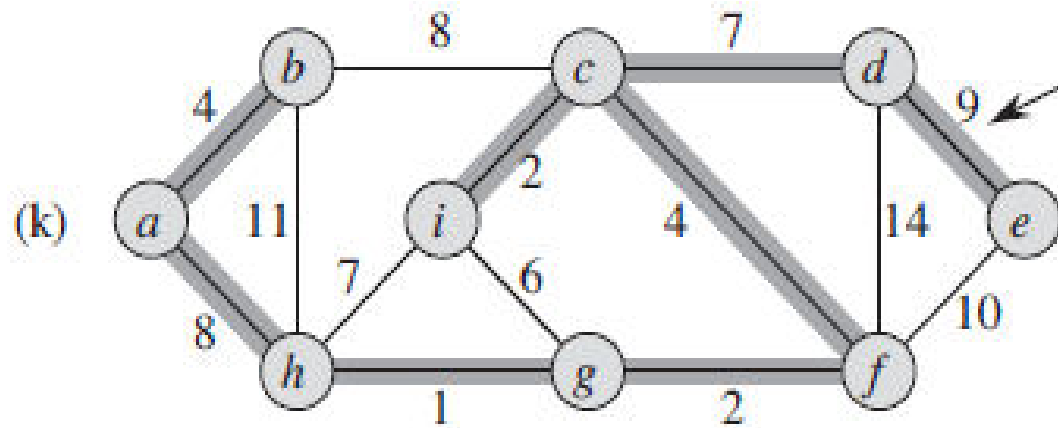
# Kruskal's Algorithm



# Kruskal's Algorithm



# Kruskal's Algorithm



Total cost = 37

Final minimum spanning tree

# Kruskal's Algorithm

MST-KRUSKAL( $G, w$ )

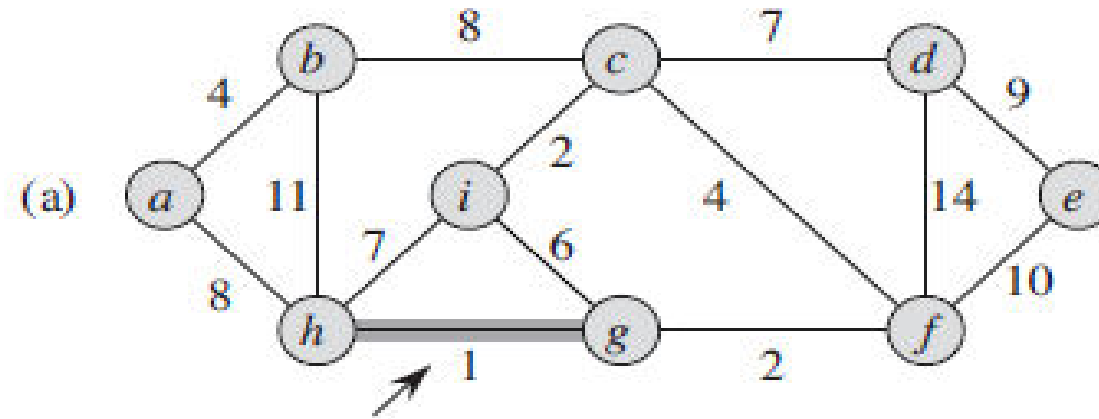
```
1   $A = \emptyset$ 
2  for each vertex  $v \in G.V$ 
3      MAKE-SET( $v$ )
4  sort the edges of  $G.E$  into nondecreasing order by weight  $w$ 
5  for each edge  $(u, v) \in G.E$ , taken in nondecreasing order by weight
6      if FIND-SET( $u$ )  $\neq$  FIND-SET( $v$ )
7           $A = A \cup \{(u, v)\}$ 
8          UNION( $u, v$ )
9  return  $A$ 
```

Time complexity of this algorithm is  $O(E \lg E)$  or  $O(E \lg V)$ .

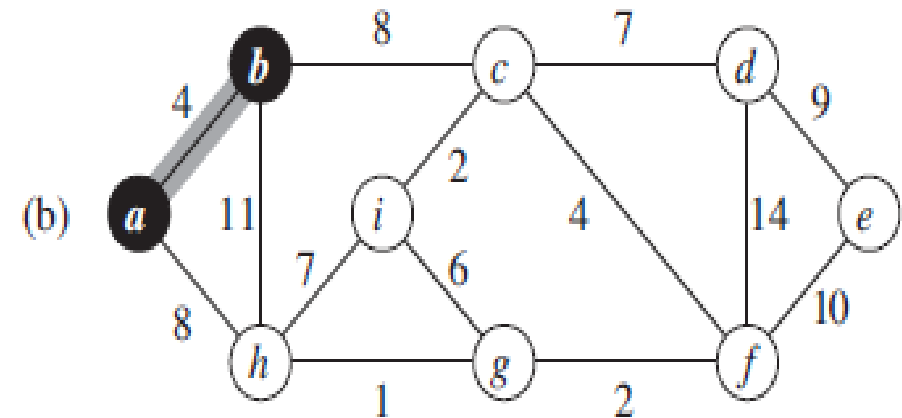
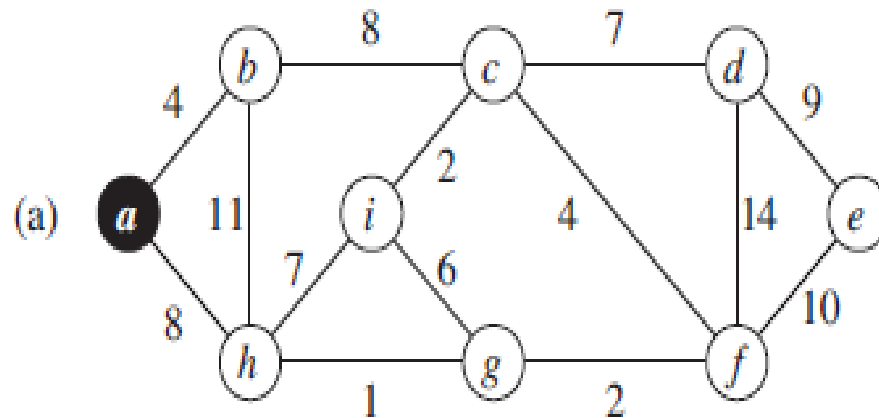


# Prim's Algorithm

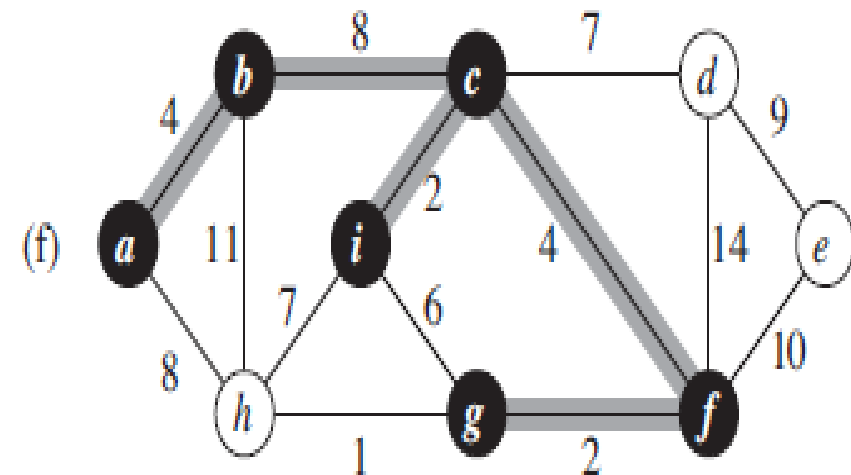
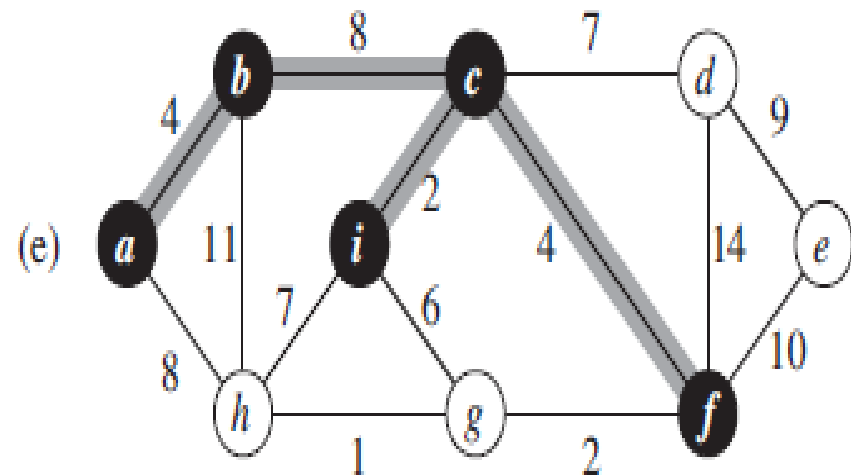
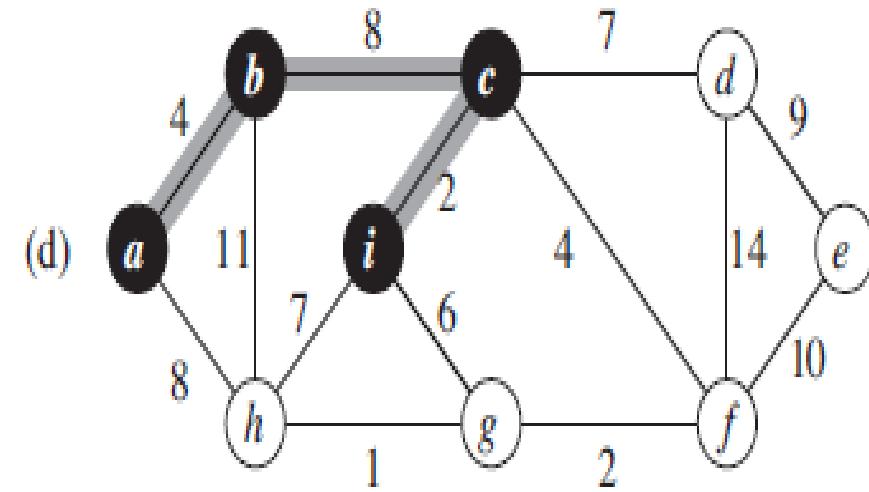
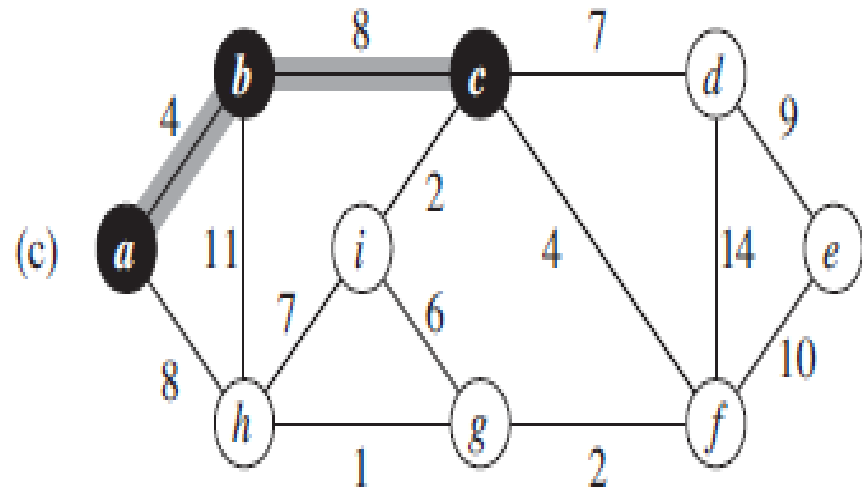
Example: Find the minimum spanning of the following graph using Prim's algorithm.



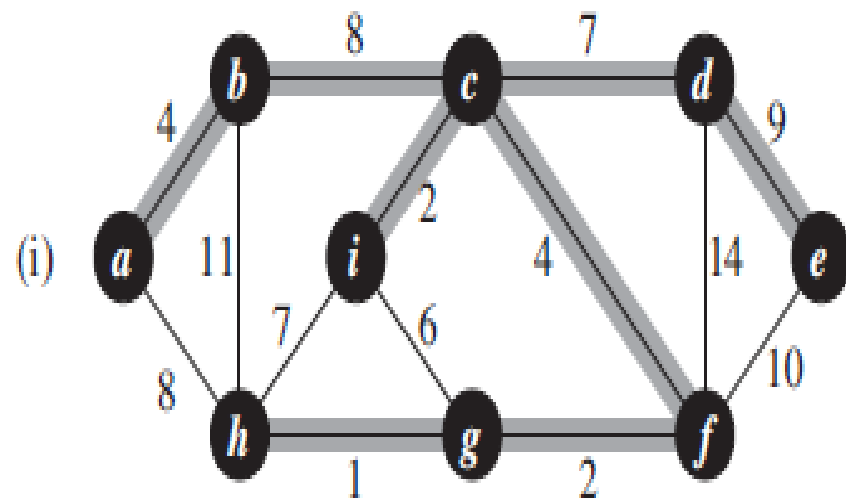
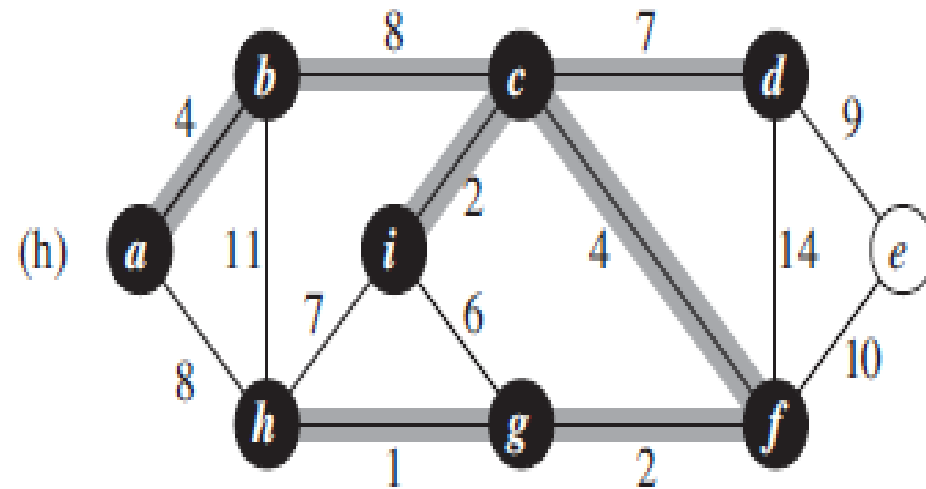
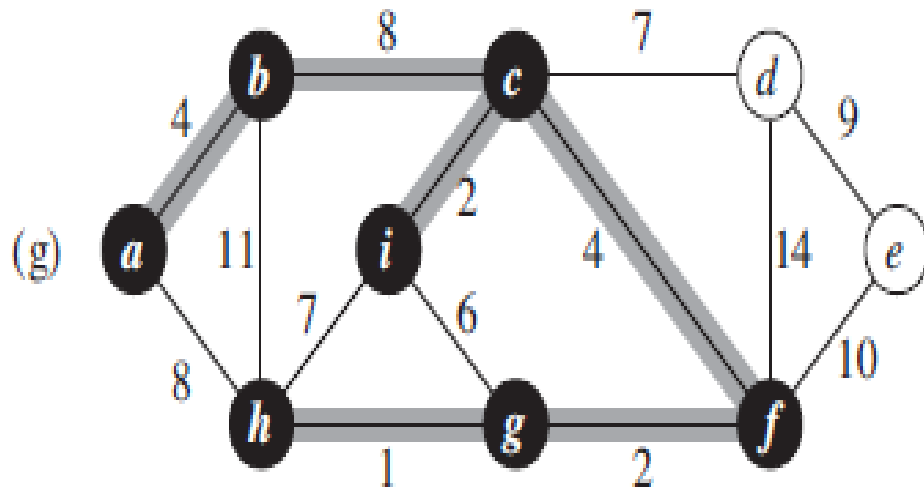
Solution:



# Prim's Algorithm



# Prim's Algorithm



Total cost = 37

Final minimum spanning tree

# Prim's Algorithm

MST-PRIM( $G, w, r$ )

```
1  for each  $u \in G.V$ 
2       $u.key = \infty$ 
3       $u.\pi = \text{NIL}$ 
4   $r.key = 0$ 
5   $Q = G.V$ 
6  while  $Q \neq \emptyset$ 
7       $u = \text{EXTRACT-MIN}(Q)$ 
8      for each  $v \in G.Adj[u]$ 
9          if  $v \in Q$  and  $w(u, v) < v.key$ 
10              $v.\pi = u$ 
11              $v.key = w(u, v)$ 
```

Time complexity of this algorithm is  $O(E \lg V + V \lg V) = O(E \lg V)$ .

# Shortest path algorithm

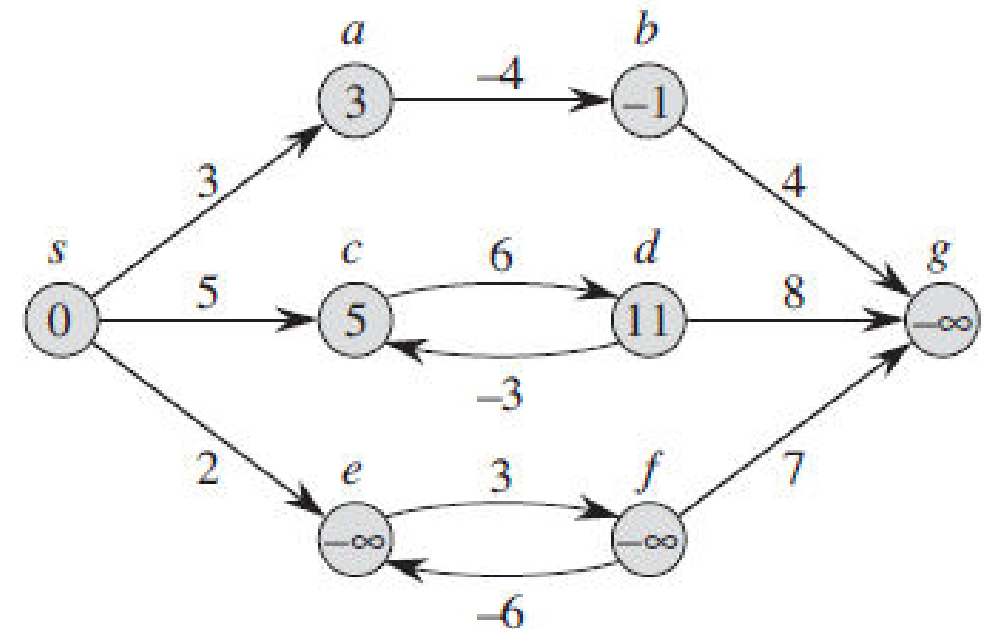
## Single-source shortest-paths problem

Given a graph  $G = (V, E)$ , we want to find a shortest path from a given source vertex  $s \in V$  to each vertex  $v \in V$ .

## Negative weight cycles

A cycle in the graph is said to be negative weight cycle if the sum of weights of all the edges in the cycle is negative value.

**Example:** Consider the following graph.



# Shortest path algorithm

## Initialize the single source function

INITIALIZE-SINGLE-SOURCE( $G, s$ )

```
1  for each vertex  $v \in G.V$ 
2       $v.d = \infty$ 
3       $v.\pi = \text{NIL}$ 
4   $s.d = 0$ 
```

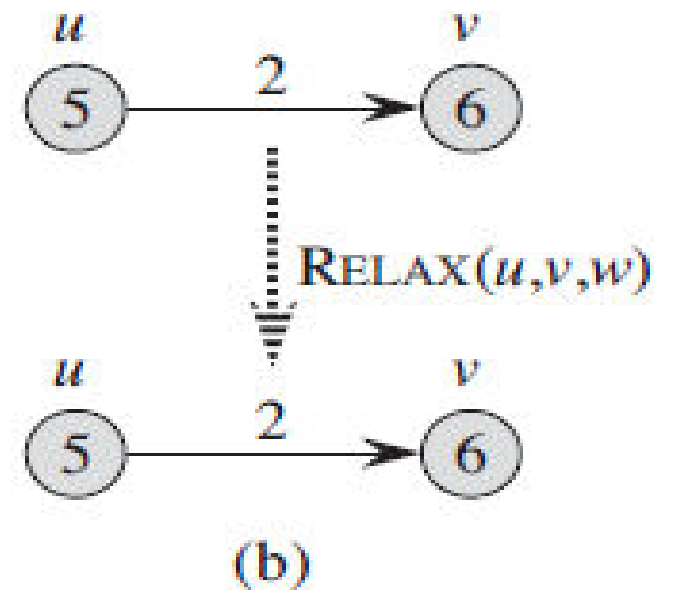
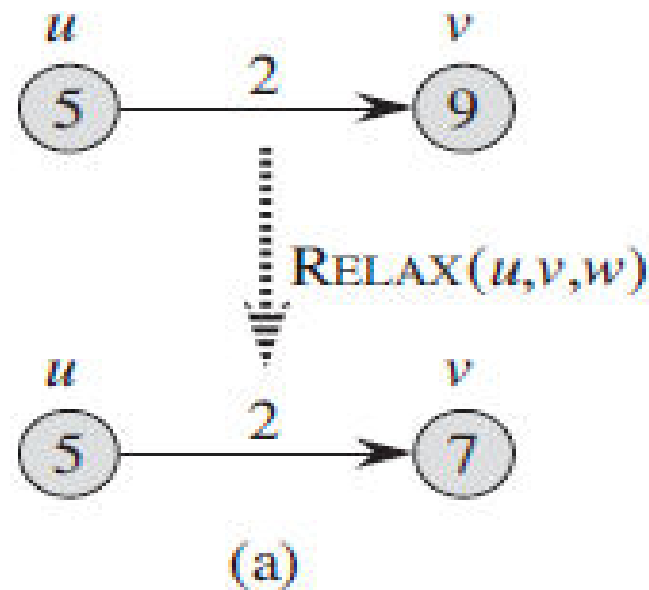
$s$  is the source vertex.

$v.d$  is the distance of vertex  $v$  from  $s$ .

$v.\pi$  is the predecessor vertex of  $v$ .

# Shortest path algorithm

## Relaxation function



$\text{RELAX}(u, v, w)$

- 1 if  $v.d > u.d + w(u, v)$
- 2      $v.d = u.d + w(u, v)$
- 3      $v.\pi = u$

# Shortest path algorithm

Here, we shall study two single source shortest path algorithms. These algorithms are the following:-

1. Bellman-Ford algorithm
2. Dijkstra algorithm

Both algorithms are based on **greedy approach**.



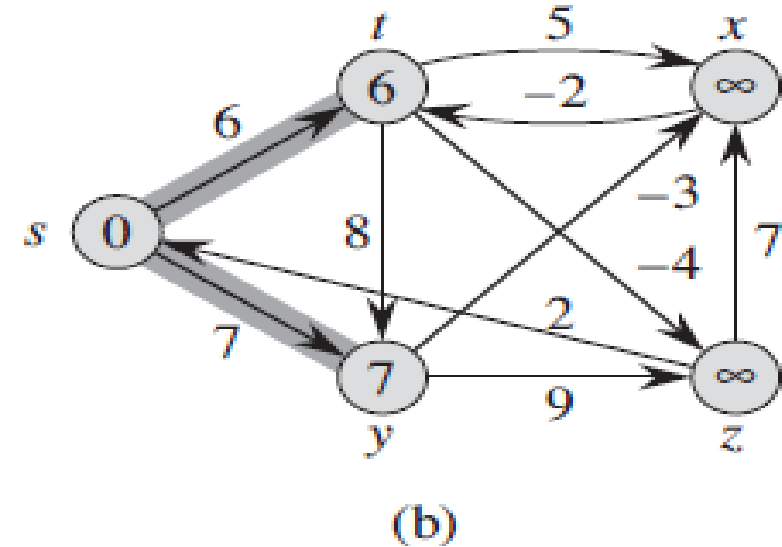
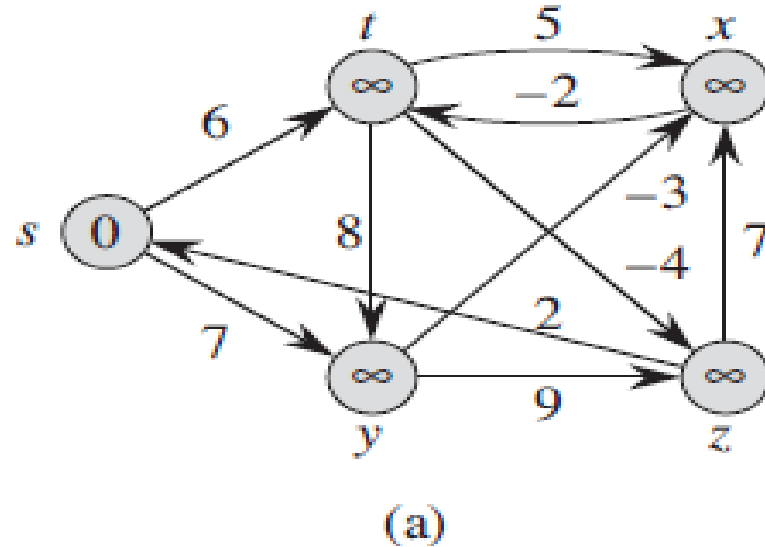
# Bellman-Ford algorithm

The **Bellman-Ford algorithm** solves the single-source shortest-paths problem in the general case in which edge weights may be negative. Given a weighted, directed graph  $G = (V, E)$  with source  $s$  and weight function  $w : E \rightarrow \mathbb{R}$ , the Bellman-Ford algorithm returns a boolean value indicating whether or not there is a negative-weight cycle that is reachable from the source. If there is such a cycle, the algorithm indicates that no solution exists. If there is no such cycle, the algorithm produces the shortest paths and their weights.

The algorithm returns TRUE if and only if the graph contains no negative-weight cycles that are reachable from the source.

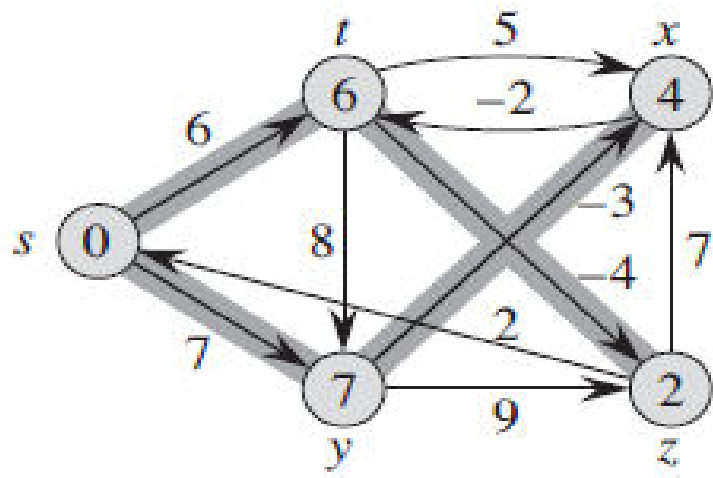
# Bellman-Ford algorithm

## Example:

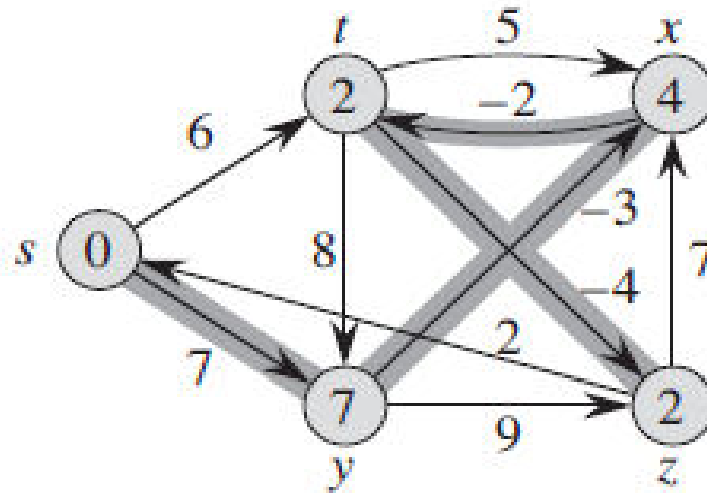


The execution of the Bellman-Ford algorithm. The source is vertex  $s$ . The  $d$  values appear within the vertices, and shaded edges indicate predecessor values: if edge  $(u, v)$  is shaded, then:  $v.\pi = u$ . In this particular example, each pass relaxes the edges in the order  $(t, x)$ ,  $(t, y)$ ,  $(t, z)$ ,  $(x, t)$ ,  $(y, x)$ ,  $(y, z)$ ,  $(z, x)$ ,  $(z, s)$ ,  $(s, t)$ ,  $(s, y)$ .

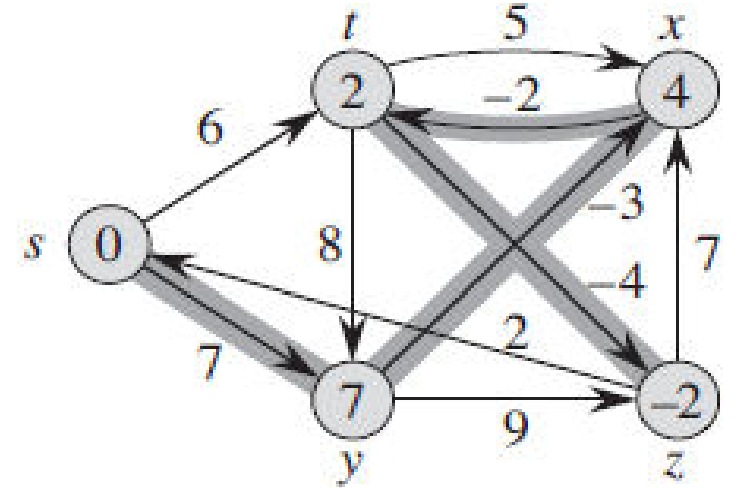
# Bellman-Ford algorithm



(c)



(d)



(e)

Order  $(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)$ .

# Bellman-Ford algorithm

BELLMAN-FORD( $G, w, s$ )

```
1  INITIALIZE-SINGLE-SOURCE( $G, s$ )
2  for  $i = 1$  to  $|G.V| - 1$ 
3      for each edge  $(u, v) \in G.E$ 
4          RELAX( $u, v, w$ )
5  for each edge  $(u, v) \in G.E$ 
6      if  $v.d > u.d + w(u, v)$ 
7          return FALSE
8  return TRUE
```

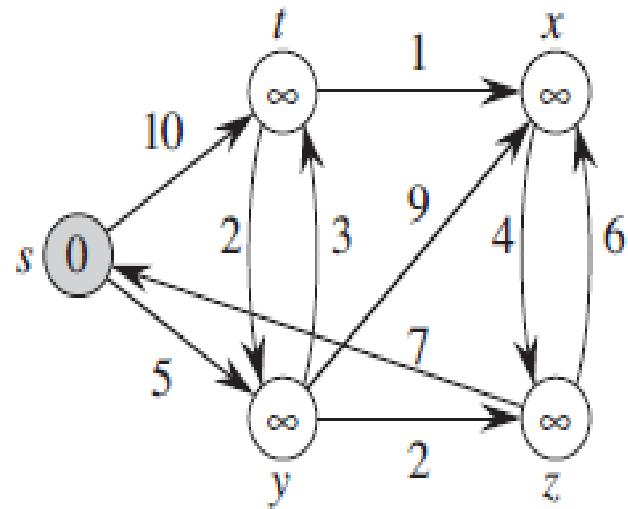
Time complexity of this algorithm is  **$O(VE)$** .

# Dijkstra's algorithm

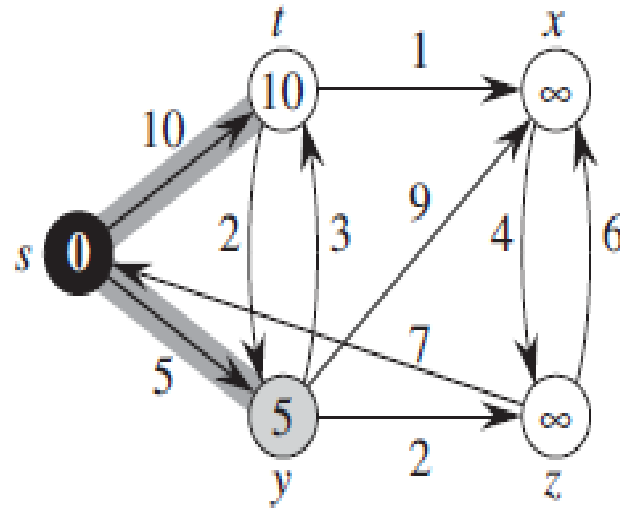
- Dijkstra's algorithm solves the single-source shortest-paths problem on a weighted, directed graph  $G = (V, E)$  for the case in which all edge weights are nonnegative.
- Dijkstra's algorithm maintains a set  $S$  of vertices whose final shortest-path weights from the source  $s$  have already been determined. The algorithm repeatedly selects the vertex  $u \in V-S$  with the minimum shortest-path estimate, adds  $u$  to  $S$ , and relaxes all edges leaving  $u$ .
- In the following implementation, we use a min-priority queue  $Q$  of vertices, keyed by their  $d$  values.

# Dijkstra's algorithm

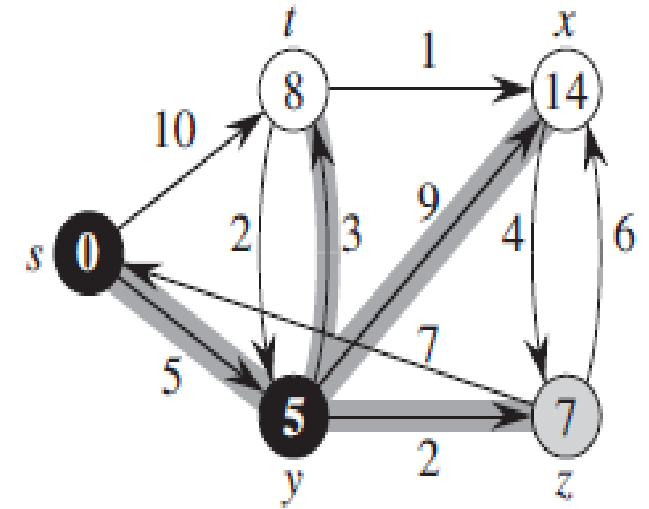
## Example:



(a)



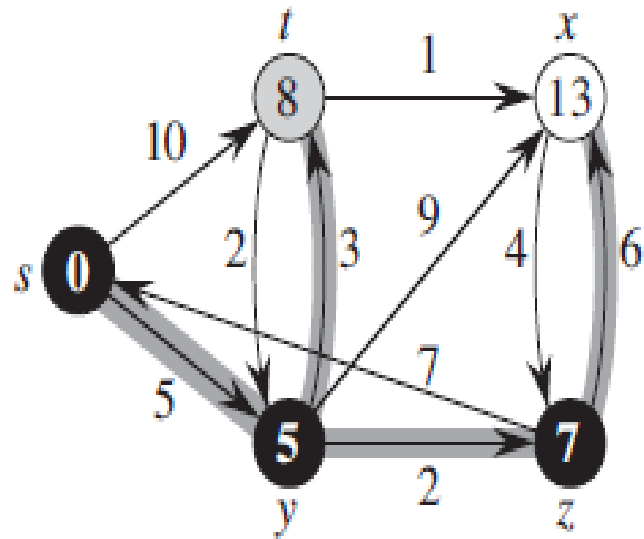
(b)



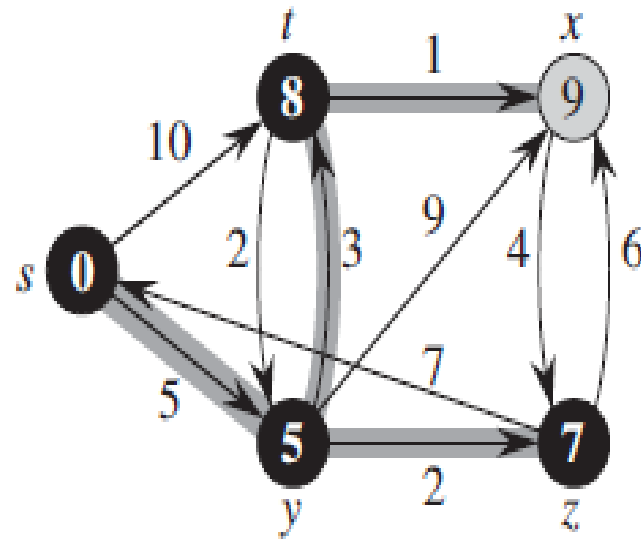
(c)

The execution of Dijkstra's algorithm.  $s$  is the source vertex. The shortest-path estimates appear within the vertices, and shaded edges indicate predecessor values. Black vertices are in the set  $S$ , and white vertices are in the min-priority queue  $Q = V - S$ .

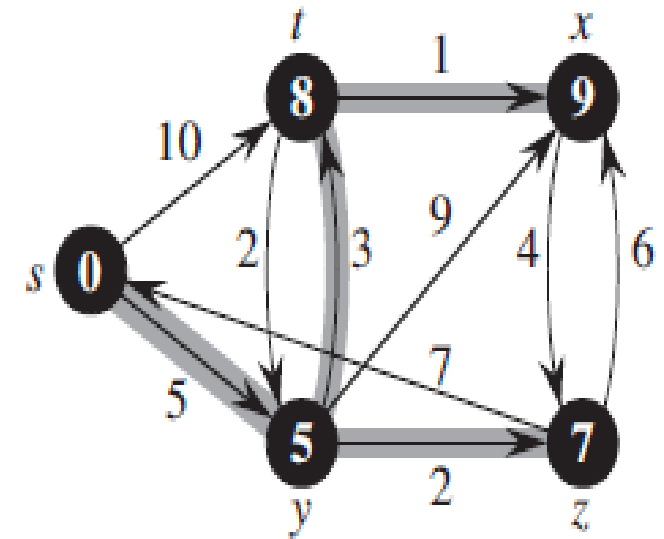
# Dijkstra's algorithm



(d)



(e)



(f)

# Dijkstra's algorithm

DIJKSTRA( $G, w, s$ )

```
1  INITIALIZE-SINGLE-SOURCE( $G, s$ )
2   $S = \emptyset$ 
3   $Q = G.V$ 
4  while  $Q \neq \emptyset$ 
5       $u = \text{EXTRACT-MIN}(Q)$ 
6       $S = S \cup \{u\}$ 
7      for each vertex  $v \in G.Adj[u]$ 
8          RELAX( $u, v, w$ )
```

Time complexity of this algorithm is  **$O(E \log V)$** .



# AKTU Examination Questions

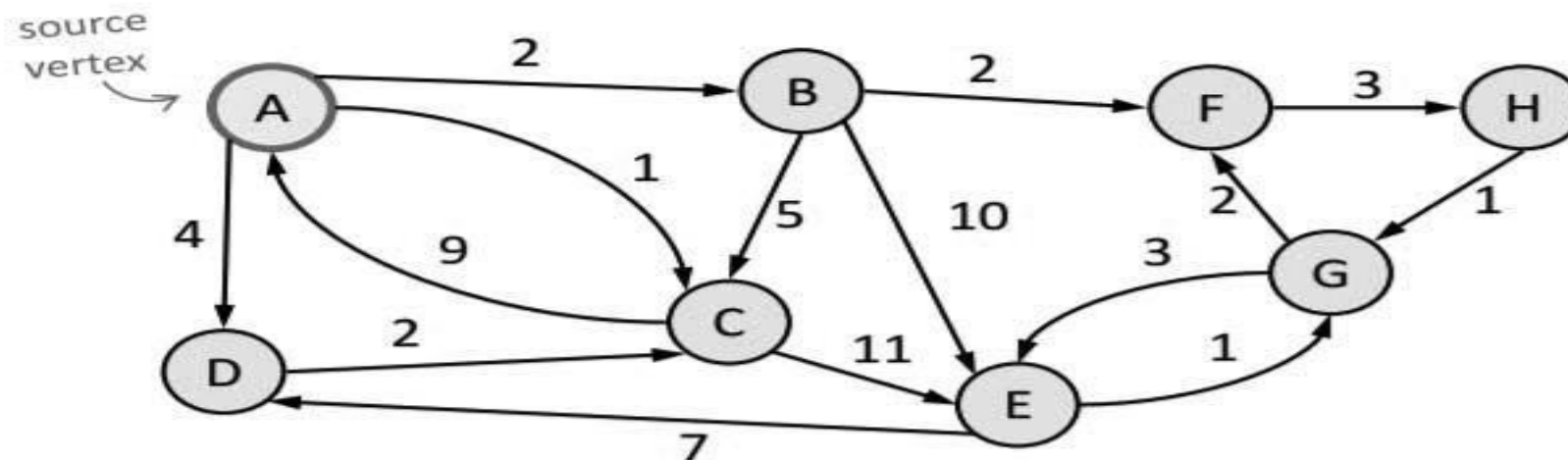
1. Consider the following instance for knapsack problem. Find the solution using Greedy method:

$N = 10, W = 130$

$P [] = \{21, 31, 43, 53, 41, 63, 65, 75\}$

$V [] = \{11, 21, 31, 33, 43, 53, 65, 65\}$

2. Apply the greedy single source shortest path algorithm on the following graph:

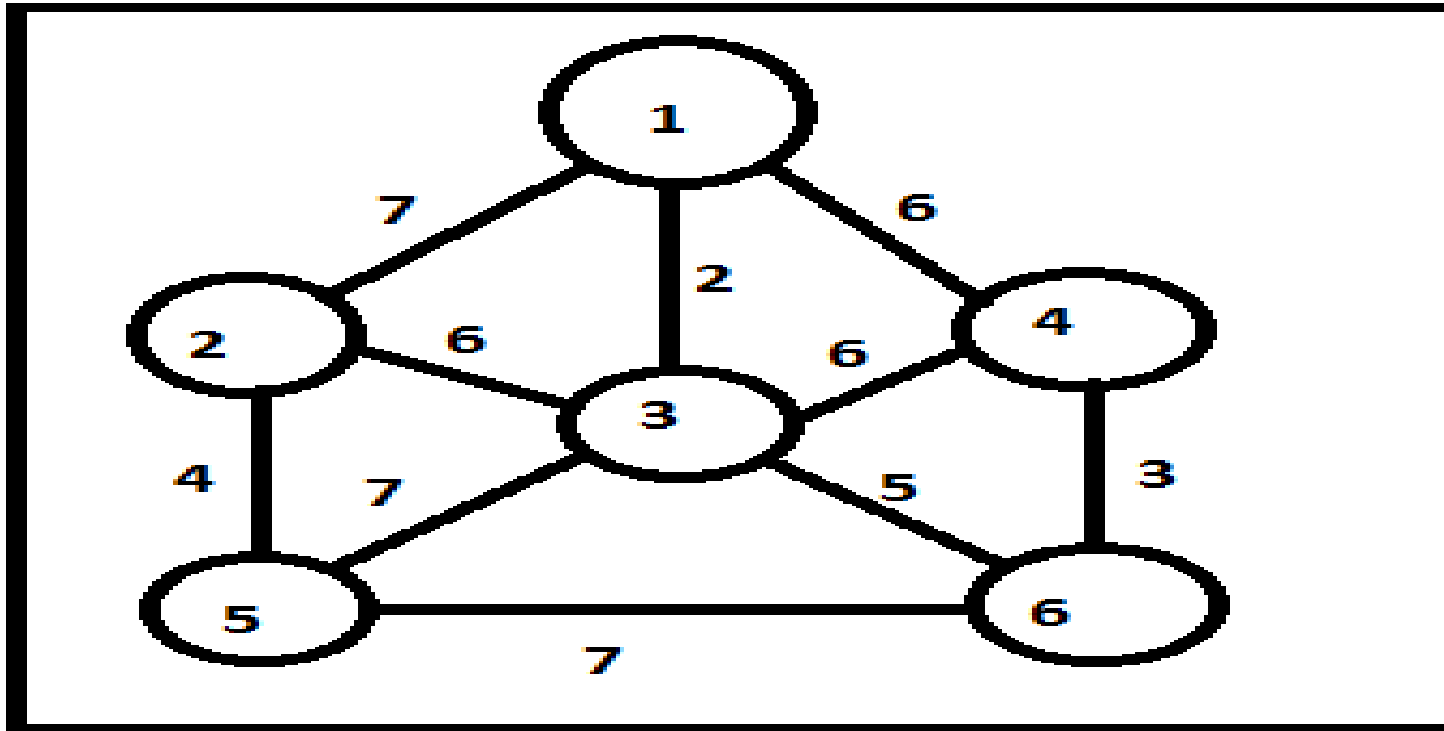


# AKTU Examination Questions

3. Describe Activity selection problem.
4. Write an algorithm for minimum spanning tree with example.
5. What are greedy algorithms? Explain their characteristics?
6. Define feasible and optimal solution.
7. Define spanning tree. Write Kruskal's algorithm for finding minimum cost spanning tree. Describe how Kruskal's algorithm is different from Prim's algorithm for finding minimum cost spanning tree.
8. Explain Single source shortest path.

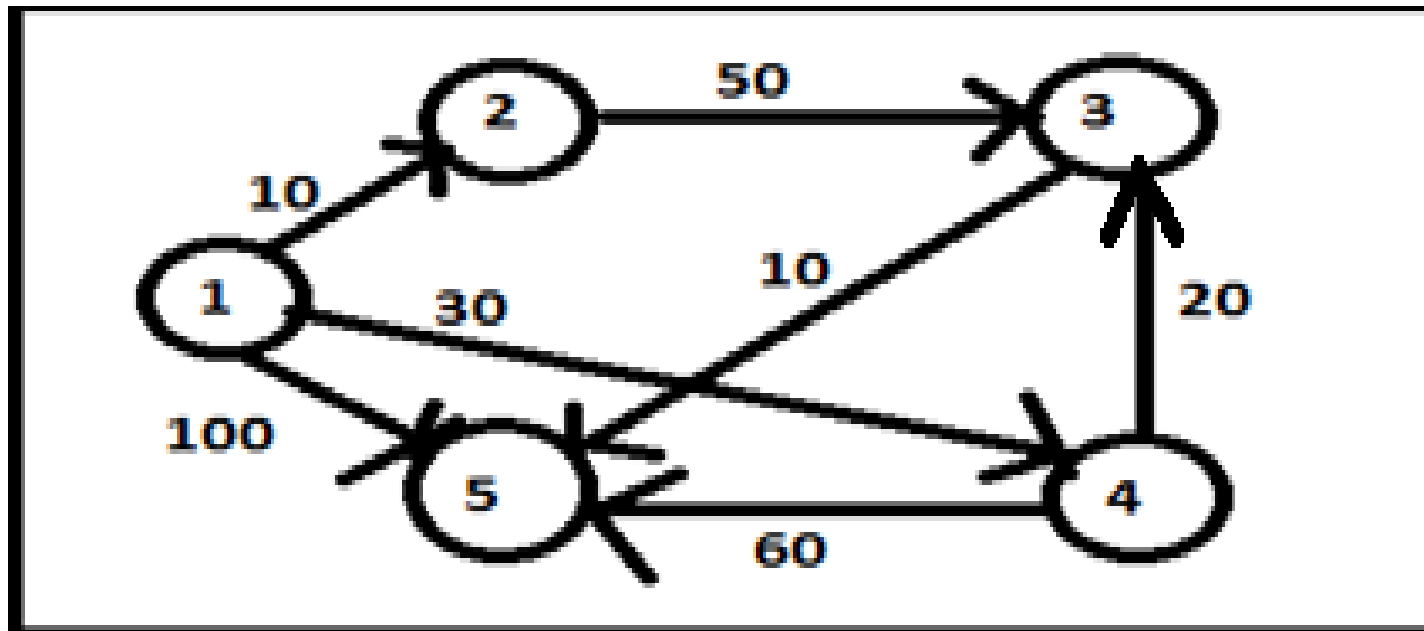
# AKTU Examination Questions

9. What is Minimum Cost Spanning Tree? Explain Kruskal's Algorithm and Find MST of the Graph. Also write its Time-Complexity.



# AKTU Examination Questions

10. Find the shortest path in the below graph from the source vertex 1 to all other vertices by using Dijkstra's algorithm.



# AKTU Examination Questions

11. Given the six items in the table below and a Knapsack with Weight 100, what is the solution to the Knapsack problem in all concepts. I.e. explain greedy all approaches and find the optimal solution.

ITEM ID	WEIGHT	VALUE	VALUE/WEIGHT
A	100	40	.4
B	50	35	.7
C	40	20	.5
D	20	4	.2
E	10	10	1
F	10	6	.6

12. Prove that if the weights on the edge of the connected undirected graph are distinct then there is a unique Minimum Spanning Tree. Give an example in this regard. Also discuss Prim's Minimum Spanning Tree Algorithm in detail.

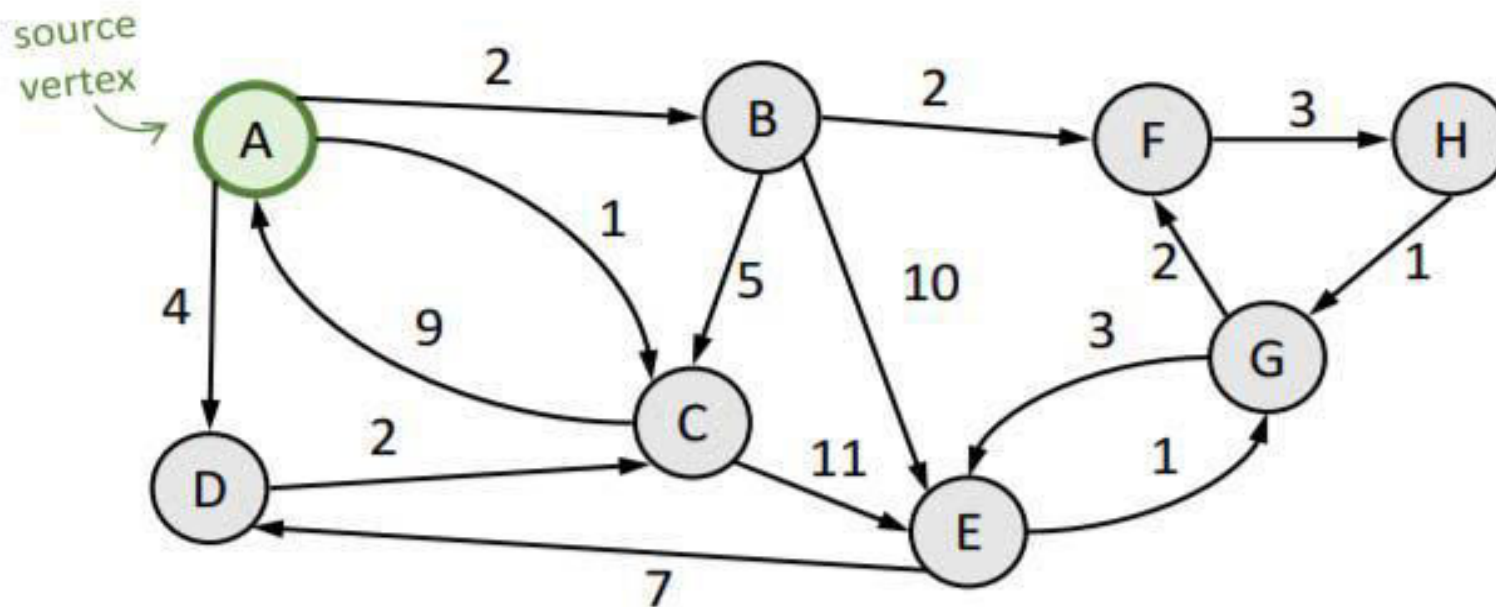
# AKTU Examination Questions

13. Consider the weights and values of items listed below. Note that there is only one unit of each item. The task is to pick a subset of these items such that their total weight is no more than 11 Kgs and their total value is maximized. Moreover, no item may be split. The total value of items picked by an optimal algorithm is denoted by  $V_{\text{opt}}$ . A greedy algorithm sorts the items by their value-to-weight ratios in descending order and packs them greedily, starting from the first item in the ordered list. The total value of items picked by the greedy algorithm is denoted by  $V_{\text{greedy}}$ . Find the value of  $V_{\text{opt}} - V_{\text{greedy}}$ .

Item	$I_1$	$I_2$	$I_3$	$I_4$
W	10	7	4	2
V	60	28	20	24

# AKTU Examination Questions

14. When do Dijkstra and the Bellman-Ford algorithm both fail to find a shortest path? Can Bellman ford detect all negative weight cycles in a graph? Apply Bellman Ford Algorithm on the following graph:



**Thank You.**