S. J. P. N. TRUST'S
HIRASUGAR INSTITUTE OF TECHNOLOGY, NIDASOSHI
Accredited at 'A' Grade by NAAC
Programmes Accredited by NBA: CSE, ECE, EEE & ME.

Department of Computer Science & Engineering

Course: Design And Analysis of Algorithms (18CS42)

Module 3: Greedy Method, Minimum Cost Spanning Tree, Single Source Shortest Path, Optimal Tree Problem, Transform And Conquer Approach

Prof. A. A. Daptardar
Asst. Prof. , Dept. of Computer Science & Engg., Hirasugar Institute of Technology, Nidasoshi

Module – 3
Greedy Method

1. **Introduction**
2. **Coin Change Problem**
3. **Knapsack problem**
4. **Job Sequencing with deadlines**
5. **Spanning Tree**
6. **Minimum Cost Spanning Tree**

Module – 3
Greedy Method

7. **Prim's Agorithm**
8. **Kruskal's Algorithm**
9. **Single Source Shortest Path**
10. **Dijkstra's Algorithm**
11. **Huffman Trees & Codes**
12. **Heaps and Heap Sort**

Introduction

- Greedy method is an optimization technique used to solve many real time examples
- Greedy method has a constraint that must be followed
- Greedy method has a objective to achieve
- Objective of greedy method is to find either minimum or maximum value by choosing feasible solution

- A greedy algorithm is an algorithm that always tries to find the best solution for each sub-problem with the hopes that this will yield a good solution for the problem as a whole.
- A greedy algorithm always makes the choice that looks best at that moment.
- While solving the problems using this technique at each step the choice made must be :
  - Feasible : Satisfying problem's constraints
  - Locally optimal : It has to be best local choice among all feasible choices available.
  - Irrevocable : Once the choice is made, it should not be changed in subsequent steps of the algorithm.

The greedy method suggests that one can devise an algorithm that works in stages, considering one input at a time. At each stage, a decision is made regarding whether a particular input is in an optimal solution. This is done by considering the inputs in an order determined by some selection procedure. If the inclusion of the next input into the partially constructed optimal solution will result in an infeasible solution, then this input is not added to the partial solution. Otherwise, it is added. The selection procedure itself is based on some optimization measure. This measure may be the objective function. In fact, several different optimization measures may be plausible for a given problem. Most of these, however, will result in algorithms that generate suboptimal solutions. This version of the greedy technique is called the *subset paradigm*.

Greedy Method Algorithm


```
Algorithm Greedy(a, n)
// a[1 : n] contains the n inputs.
{
    solution := ∅; // Initialize the solution.
    for i := 1 to n do
    {
        x := Select(a);
        if Feasible(solution, x) then
            solution := Union(solution, x);
    }
    return solution;
}
```

Coin change problem Statement

- A customer buys items valued less than 50 rupees and gives a 50 rupees note to the cashier (shopkeeper)
- Now, cashier wish to return remaining change to the customer with minimum number of coins available
- The cashier constructs the change in stages using greedy method
- In each stage increase the total amount of change constructed by as much as possible

Coin Change Problem Example

- Suppose customer buys items valued 39 rupees and gives 50 rupees note to cashier
- Then cashier needs to return 11 rupees change back to customer
- Also assume unlimited denominations of 1, 2, 5 and 10 rupees are available with cashier
- Then possible solutions to return remaining change i.e. 11 rupees back to customer will be-
- Solution1 - 10 + 1 = 11 and it takes 2 coins
- Solution2 - 5 + 5 + 1 = 11 and it takes 3 coins
- Solution3 - 5 + 2 + 2 + 1 + 1 = 11 and it takes 5 coins

Coin Change Problem Example

- Similarly N solutions are possible to solve above example
- Among N solutions, Solution1 will be optimal solution because it takes only 2 coins
- Also Solution1 achieved the objective of coin change problem i. e. to return 11 rupees change back to customer

Knapsack Problem

KNAPSACK PROBLEM STATEMENT

4.3 KNAPSACK PROBLEM

Let us try to apply the greedy method to solve the knapsack problem. We are given n objects and a knapsack or bag. Object i has a weight w_i and the knapsack has a capacity m. If a fraction x_i, $0 \leq x_i \leq 1$, of object i is placed into the knapsack, then a profit of $p_i x_i$ is earned. The objective is to obtain a filling of the knapsack that maximizes the total profit earned. Since the knapsack capacity is m, we require the total weight of all chosen objects to be at most m. Formally, the problem can be stated as

$$\text{maximize} \sum_{1 \leq i \leq n} p_i x_i \tag{4.1}$$

$$\text{subject to} \sum_{1 \leq i \leq n} w_i x_i \leq m \tag{4.2}$$

$$\text{and } 0 \leq x_i \leq 1, \quad 1 \leq i \leq n \tag{4.3}$$

The profits and weights are positive numbers.

A feasible solution (or filling) is any set $(x_1, \ldots, x_n)$ satisfying (4.2) and (4.3) above. An optimal solution is a feasible solution for which (4.1) is maximized.

Knapsack Problem

Objective:-

filling of knapsack(bag) that maximizes the total profit earned

Constraint:-

1. Total weight of all chosen object must be less than or equal to knapsack capacity m
2. Profits(Pi) and Weights(Wi) are positive integers

Example 4.1 Consider the following instance of the knapsack problem: $n = 3, m = 20, (p_1, p_2, p_3) = (25, 24, 15)$, and $(w_1, w_2, w_3) = (18, 15, 10)$. Four feasible solutions are:

	(x_1, x_2, x_3)	$\sum w_i x_i$	$\sum p_i x_i$
1.	(1/2, 1/3, 1/4)	16.5	24.25
2.	(1, 2/15, 0)	20	28.2
3.	(0, 2/3, 1)	20	31
4.	(0, 1, 1/2)	20	31.5

Of these four feasible solutions, solution 4 yields the maximum profit. As we shall soon see, this solution is optimal for the given problem instance. □

```
void GreedyKnapsack(float m, int n)
// p[1:n] and w[1:n] contain the profits and weights
// respectively of the n objects ordered such that
// p[i]/w[i] >= p[i+1]/w[i+1]. m is the knapsack
// size and x[1:n] is the solution vector.
{
    for (int i=1; i<=n; i++) x[i] = 0.0; // Initialize x.
    float U = m;
    for (i=1; i<=n; i++) {
        if (w[i] > U) break;
        x[i] = 1.0;
        U -= w[i];
    }
    if (i <= n) x[i] = U/w[i];
}
```

Knapsack Problem Example

Consider the following instance of the knapsack problem:

Number of Objects(n) = 7

Knapsack Capacity(m) = 15

Profits(p1,p2,p3,p4,p5,p6,p7) = (10,5,15,7,6,18,3)

Weights(w1,w2,w3,w4,w5,w6,w7) = (2,3,5,7,1,4,1)

Find the optimal solution. i.e. maximum profit

KNAPSACK PROBLEM EXAMPLE SOLUTION

Example-2: Find the optimal solution to the knapsack instant, $n=7$, $m=15$ using greedy method —

Object	1	2	3	4	5	6	7
Profit	10	05	15	07	06	18	03
Weight	02	03	05	07	01	04	01
P_i/W_i	5	1.6	3	1	6	4.5	3

Now, Reorder objects in descending order of P_i/W_i with their corresponding profit and weight to get maximum profit — & Given Maximum Knapsack Capacity = 15.

Objects	Profit (P_i)	Weight (W_i)	P_i/W_i in DSC order	Fraction of object choosen (x_i)	Remaining knapsack capacity (m)	Maximum Profit $\sum P_i x_i$
5	06	01	6	1	15-01 = 14	6×1 = 06
1	10	02	5	1	14-02 = 12	10×1 = 10
6	18	04	4.5	1	12-04 = 08	18×1 = 18
3	15	05	3	1	08-05 = 03	15×1 = 15
7	03	01	3	1	03-01 = 02	03×1 = 03
2	05	03	1.6	2/3	02-02 = 0	$5 \times \frac{2}{3}$ = 03.33
4	07	07	1	0	0	00

Maximum Profit = 06 + 10 + 18 + 15 + 03 + 3.33 = 55.33 //

JOB SEQUENCING WITH DEADLINES

Problem Statement

- We are given a set of n jobs.
- Each job is associated with an integer deadline $d_i \geq 0$ & a profit $p_i > 0$.
- For any job i the profit p_i is earned iff the job is completed by its deadline.
- To complete a job, one has to process the job on a machine for one unit of time.
- Only one machine is available for processing jobs.
- The objective is to find the subset J of jobs such that each job in this subset can be completed by its deadline & maximum profit will be earned.

What is the deadline of a Job?

Jobs	J1	J2	J3	J4
Profits	100	50	25	75
Deadlines	2	1	4	3

- Assume all jobs takes 1 hour to complete its processing
- Suppose machine starts processing a jobs at 8am then job(J1) needs to complete its processing within 10am because job(J1) has a deadline of 2 hour
- job(J2) needs to complete its processing within 9am because job(J2) has a deadline of 1 hour

JOB SEQUENCING WITH DEADLINES

OBJECTIVES:

To obtain feasible solution with maximum profit value

CONSTRAINTS:

1. Only one machine is available for processing all jobs
2. Only one unit of time is assigned to complete a job on a machine

High Level Description Algorithm

- Algorithm GreedyJob (d, J, n)

```
// J is a set of Jobs that can be completed by their deadlines
  J = {1}
  for i = 2 to n do
  {
      if (all Jobs in J U {i} can be completed by their deadlines)
      J = J U {i}
   }
}
```

```
Algorithm JS(d, J, n)
//d[i] ≥ 1,  1≤ i≤ n are the deadlines.
// The jobs are ordered such that p[1] ≥ p[2] ≥.........≥p[n].
//J[i] is the ith job in the optimal solution,   1≤ i≤ k.
// Also at termination d[J[i]] ≤ d[J[i+1]], 1≤ i≤ k.
begin
d[0] ← J[0] ← 0
J[1] ← 1
pf ← p[1]
k ← 1
for i ← 2 to n do
begin
    r ← k
    while ( ( d[J[r]] > d[i] ) and (d[J[r]] ≠ r)) do
    r ← r -1
    end while
    if ( ( d[J[r]]  ≤ d[i]) and (d[i] > r) ) then
    {
        for q ← k to (r+1) step -1 do
        J[q+1] ← J[q]
        end for
        J[r+1] ← i
        pf ← pf + p[i]
        k ← k +1
    }
    end for
    return k
end
```

The Method

Step 1 : Arrange the profit's of jobs & its concerned deadlines in non-increasing order.

Step 2 : Apply the algorithm steps one after the other.

JOB SEQUENCING WITH DEADLINES EXAMPLE-1

21.

Example-3 – Solve below example of Job Sequencing With Deadlines – Using Greedy Method –

Jobs	1	2	3	4	5
Profits	20	15	10	5	1
Deadlines	2	2	1	3	3

Solution: Maximum deadline given is 3, Hence, maximum 3 jobs will be completed out of 5 jobs.
Now, Reorder All the jobs in descending order of profits in a two table shown below –

Jobs	Profits	Deadlines	JOB SCHEDULING			Profit Earned
J1	20	2		J_1		20
J2	15	2	J_2	J_1		15
J3	10	1	J_2	J_1		J_3 Rejected
J4	05	3	J_2	J_1	J_4	05
J5	01	3	J_2	J_1	J_4	J_5 Rejected

Job Completed – J_1, J_2, J_4

Job Rejected – J_3, J_5

Maximum Profit = 20+15+05

= 40 //.

JOB SEQUENCING WITH DEADLINES EXAMPLE-2

profit value.

Example 1: Given $n = 4$, jobs with their corresponding profits & deadlines as $(P_1, P_2, P_3, P_4) = (100, 10, 15, 27)$ & $(d_1, d_2, d_3, d_4) = (2, 1, 2, 1)$. find the maximum profit value.

Solution:

Jobs	1	2	3	4
Profits	100	10	15	27
Deadlines	2	1	2	1

Reorder Above Jobs Based on descending order of profit earned - Maximum deadline given in above example is 2. Hence, Maximum two jobs are completed -

Jobs	Profits	Deadlines	JOB Scheduling	Profit Earned
1	100	2	[ ] [J1]	100
4	27	1	[J4] [J1]	27
3	15	2	[J4] [J1]	JOB-3 Rejected
2	10	1	[J4] [J1]	JOB-2 Rejected-

Jobs Completed - Job1, Job4 Jobs Rejected - J_2, J_3

Total Profit = 100 + 27 = 127 //

Steps

- Step – 01 : Sort all the given jobs in decreasing order of their profit.
- Step-02:
  - Check the value of maximum deadline.
  - Draw a Gantt Chart where the maximum time on the Gantt chart is the value of maximum deadline.
- Step-03:
  - Pick up the jobs one by one.
  - Put the job on Gantt chart as far as possible from 0 ensuring that the job gets completed before the deadline.

Job Sequencing with Deadlines Problem

Let n =6,

(p1,p2,p3,p4,p5,p6)=(200,180,190,300,120,100) and (d1,d2,d3,d4,d5,d6)=(5,3,3,2,4,2)

- Answer the following questions:
  - Write the optimal schedule that gives maximum profit.
  - Are all the jobs completed in the optimal schedule?
  - What is the maximum earned profit?

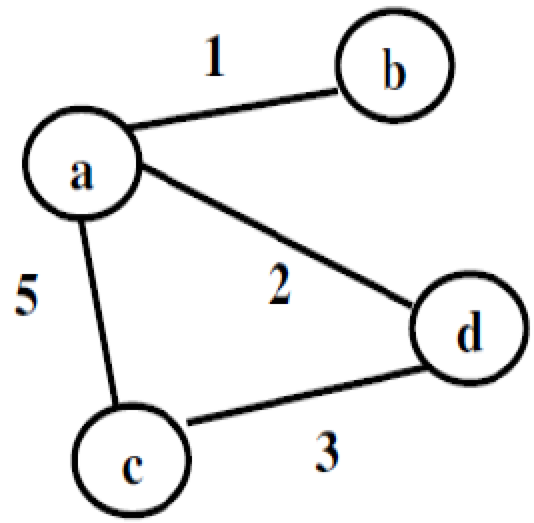
Spanning Tree

Minimum Cost Spanning Tree

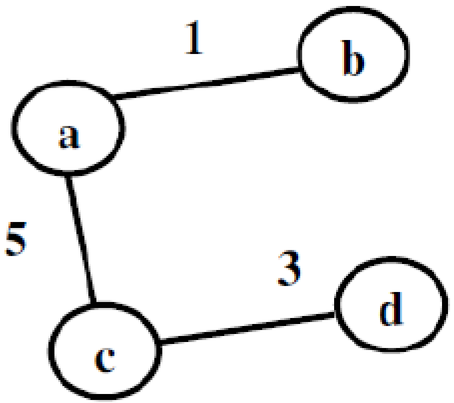
Spanning Tree

- **Definition:** Spanning tree is a connected acyclic sub-graph (tree) of the given graph (G) that includes all of G's vertices.

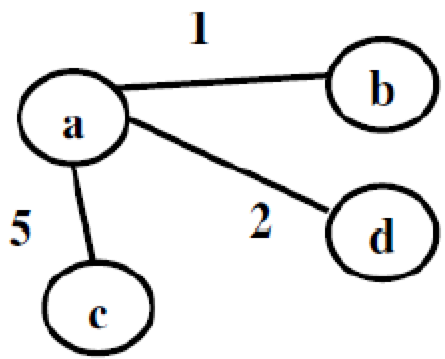
Example : Consider the following graph



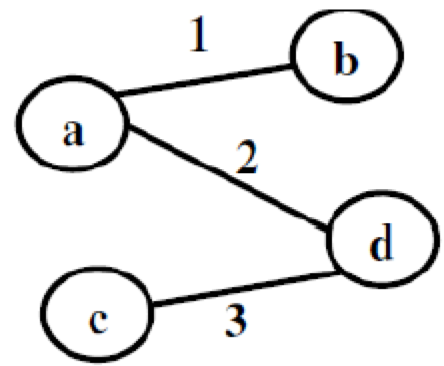
The spanning trees for the above graph are :



Weight (T_1) = 9



Weight (T_2) = 8

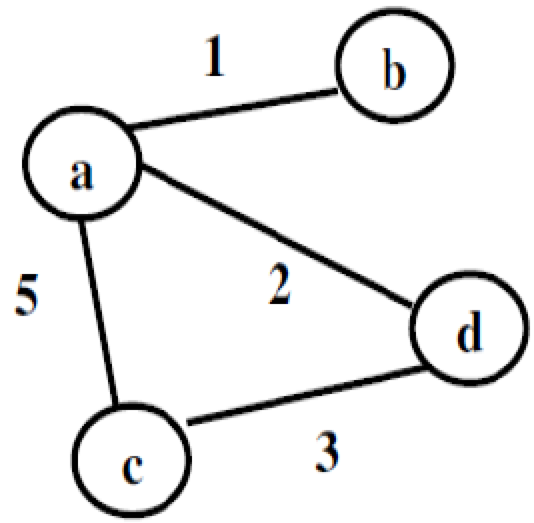


Weight (T_3) = 6

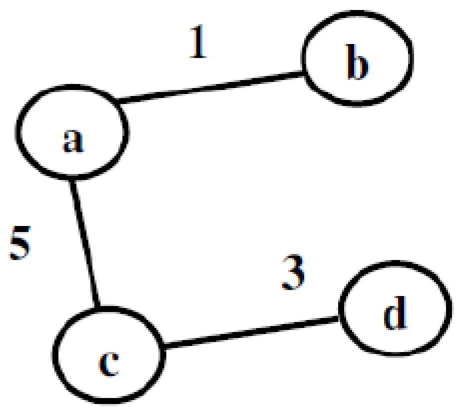
Minimum Spanning Tree (MST)

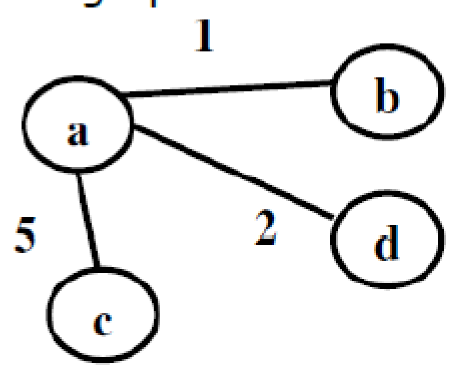
- **Definition:** MST of a weighted, connected graph G is defined as: A spanning tree of G with minimum total weight.

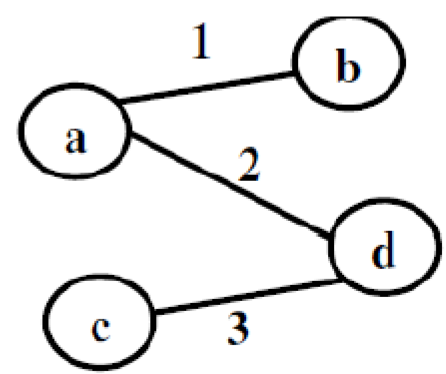
Example : Consider the following graph



The spanning trees for the above graph are :







Weight (T_1) = 9

Weight (T_2) = 8

Weight (T_3) = 6

- Two algorithms are used to generate minimum Cost Spanning Tree :
  - Prim’s Algorithm
  - Kruskal’s Algorithm

Prim’s Algorithm

- **Fringe edge:** An edge which has one vertex is in partially constructed tree Ti and the other is not.
- **Unseen edge:** An edge with both vertices not in Ti.

Algorithm

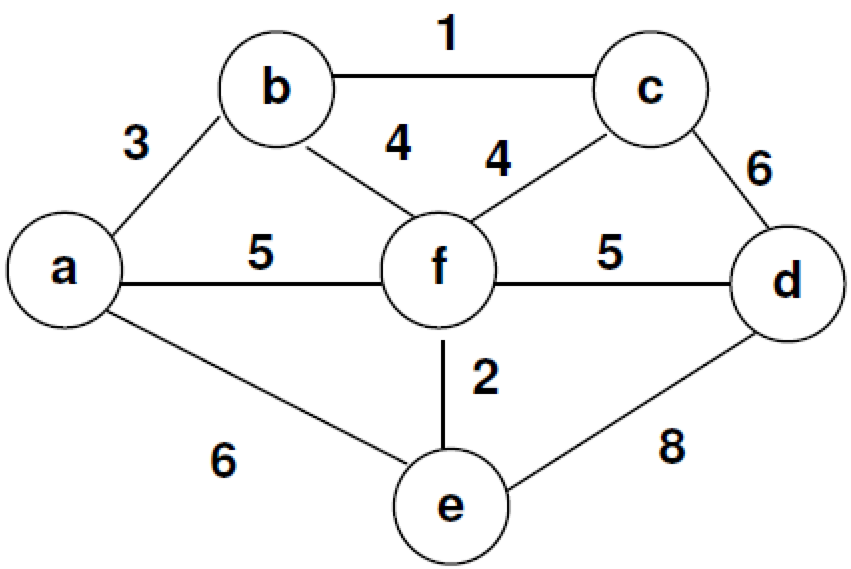
Algorithm Prim (G)
//Prim's algorithm for constructing a MST
//Input: A weighted connected graph G = { V, E }
//Output: ET the set of edges composing a MST of G
// the set of tree vertices can be initialized with any vertex
$V_T \leftarrow \{ v_0 \}$
$E_T \leftarrow$ O
for i $\leftarrow$ 1 to |V| - 1 do
Find a minimum-weight edge e* = (v*, u*) among all the edges (v, u) such that v is in V_T and u is in $V - V_T$
$V_T \leftarrow V_T$ U { u*}
$E_T \leftarrow E_T$ U { e*}
return E_T

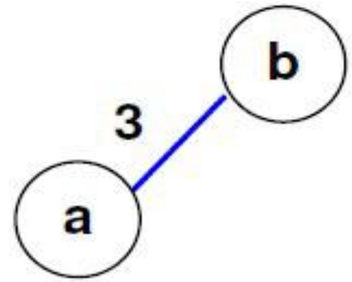
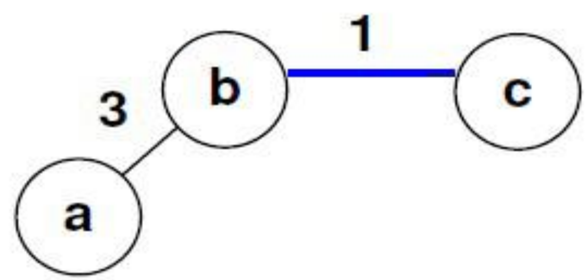
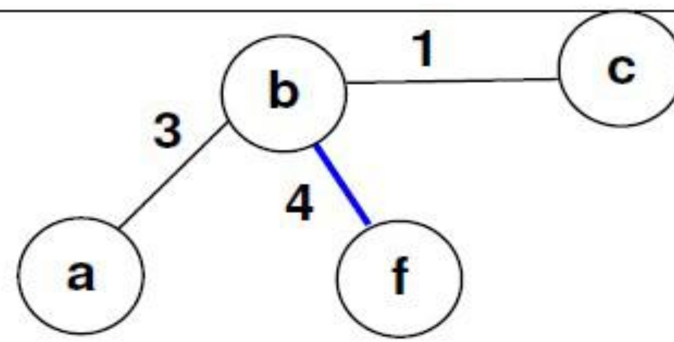
The Method:

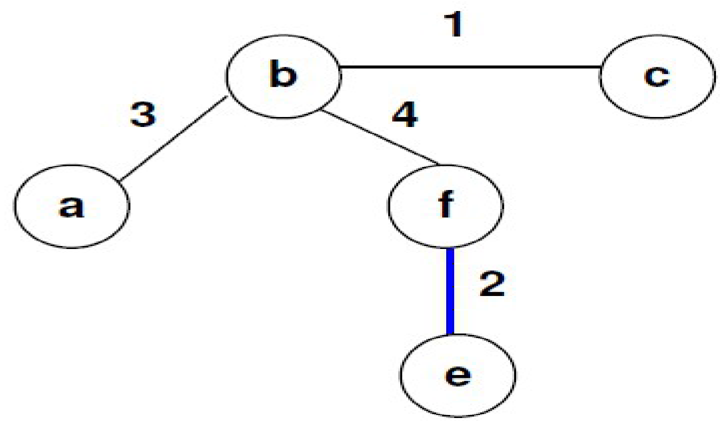
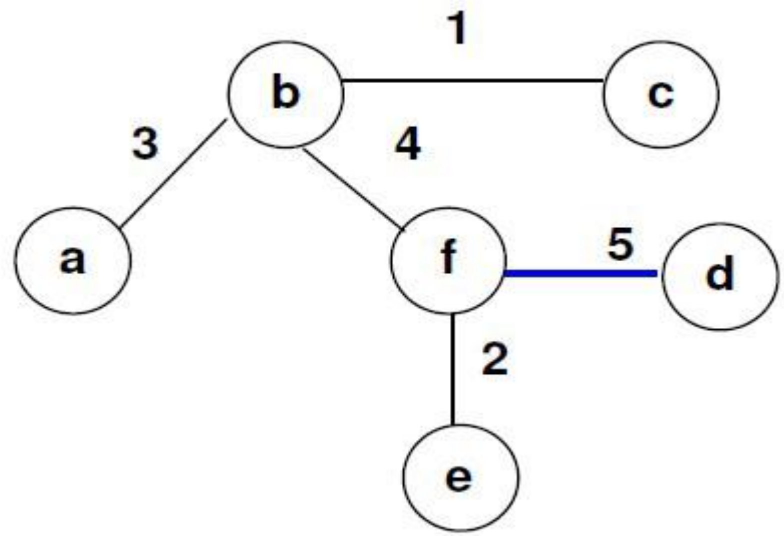
- Step 1 : Start with a tree, T_0 , consisting of one vertex
- Step 2 : “Grow” tree one vertex/edge at a time
  - Construct a series of expanding sub-trees T1, T2, ... Tn-1
  - At each stage construct Ti + 1 from Ti by adding the minimum weight edge connecting a vertex in tree (Ti) to one vertex not yet in tree, choose from “fringe” edges (this is the “greedy” step!)
  - Algorithm stops when all vertices are included

Example:

- Apply Prim's algorithm for the following graph to find MST.



Tree vertices	Remaining vertices	Graph
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)	

f (b, 4)	d (f , 5) e (f , 2)	
e (f, 2)	d (f , 5)	
d(f, 5)		**Algorithm stops since all vertices are included.** The weight of the minimum spanning tree is **15**

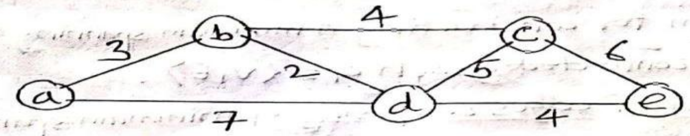
Example:

- Apply Prim's algorithm for the following graph to find MST.

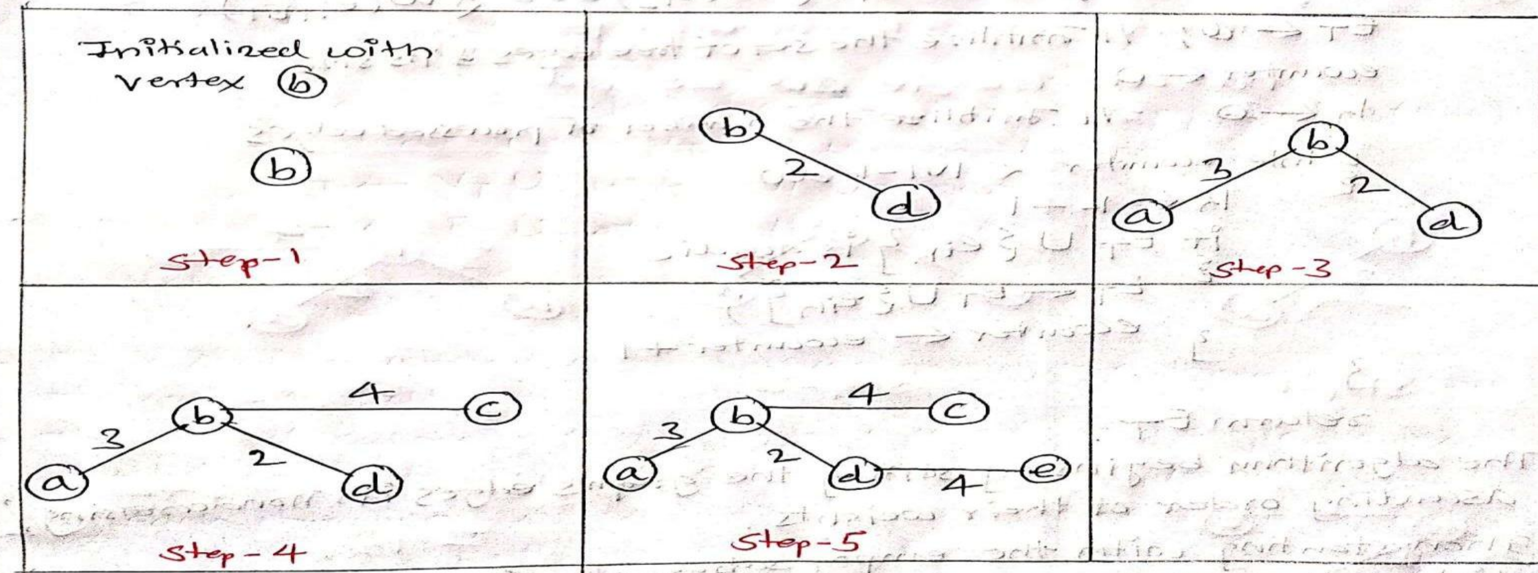
PRIM'S ALGORITHM EXAMPLE-2

Example 2: Apply Prims Algorithm for below given graph & find cost of minimum spanning tree –

25



Solution:



Minimum cost = W(T) = 3+4+2+4

= 13.

Kruskal’s Algorithm

Algorithm

Algorithm Kruskal (G)
//Kruskal's algorithm for constructing a MST
//Input: A weighted connected graph G = { V, E }
//Output: E_T the set of edges composing a MST of G
Sort E in ascending order of the edge weights
$E_T \leftarrow \emptyset$
ecounter $\leftarrow$ 0 // initialize the set of tree edges and its size
k $\leftarrow$ 0 //initialize the number of processed edges
while ecounter < |V| - 1
k $\leftarrow$ k + 1
if E_T U $\{e_{ik}\}$ is acyclic
$E_T \leftarrow E_T$ U $\{ e_{ik}\}$
ecounter $\leftarrow$ ecounter + 1
return E_T

The Method

Step 1 : Sort the edges by increasing weight

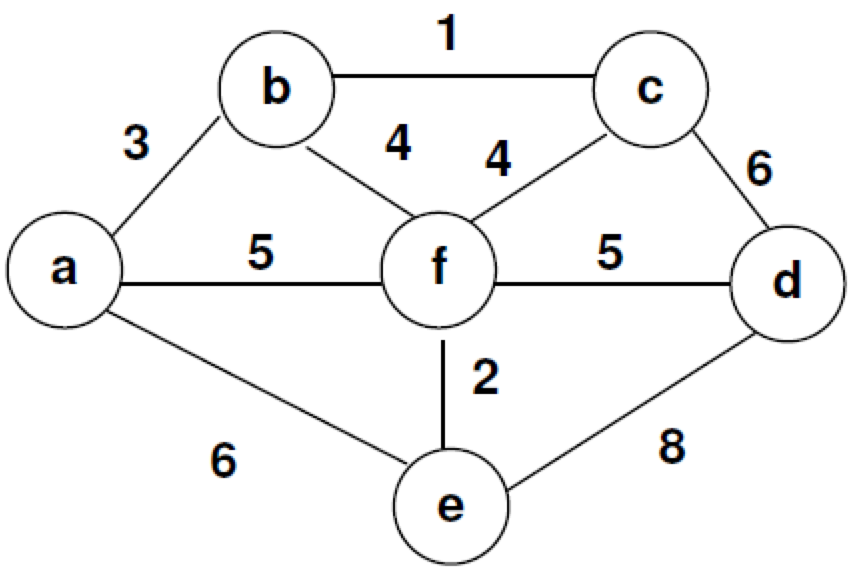
Step 2 : Start with a forest having |V| number of trees.

Step 3 : Number of trees are reduced by ONE at every inclusion of an edge

- At each stage:
  - Among the edges which are not yet included, select the one with minimum weight AND which does not form a cycle.
  - The edge will reduce the number of trees by one by combining two trees of the forest.
  - Algorithm stops when |V| -1 edges are included in the MST i.e : when the number of trees in the forest is reduced to ONE.

Example:

- Apply Kruskal's algorithm for the following graph to find MST.

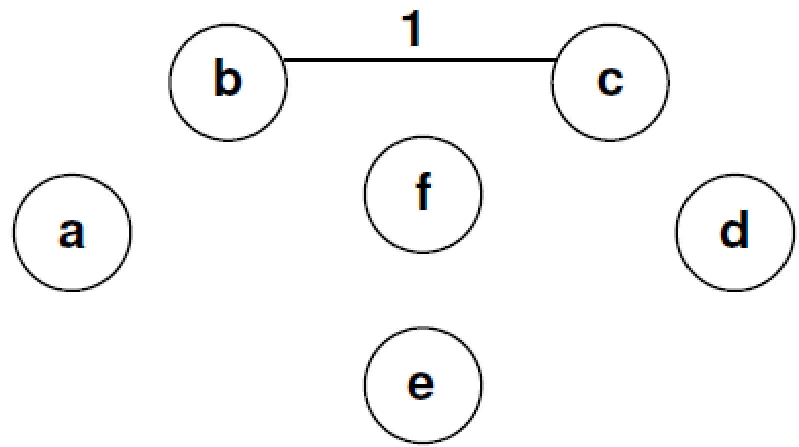
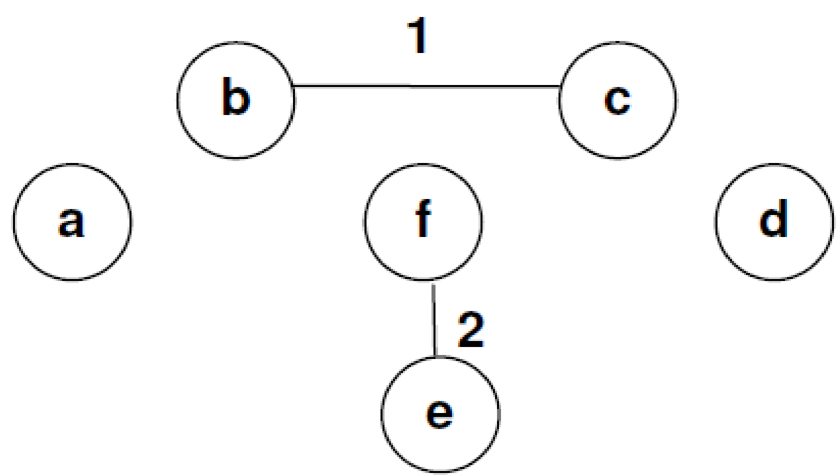


- The list of edges :

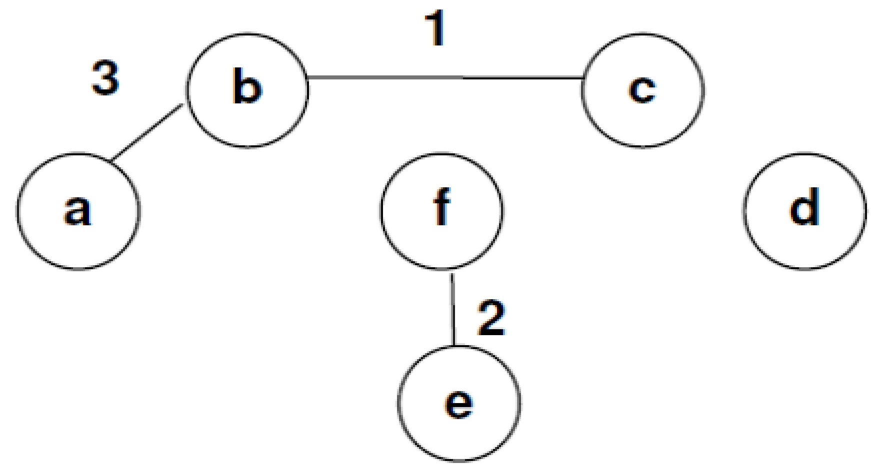
Edge	ab	af	ae	bc	bf	cf	cd	df	de	ef
Weight	3	5	6	1	4	4	6	5	8	2

- Sort the edges in ascending order :

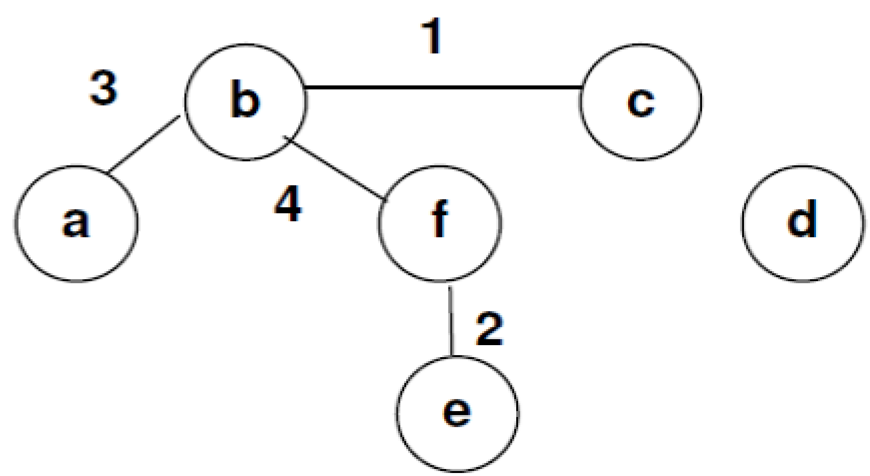
Edge	bc	ef	ab	bf	cf	af	df	ae	cd	de
Weight	1	2	3	4	4	5	5	6	6	8

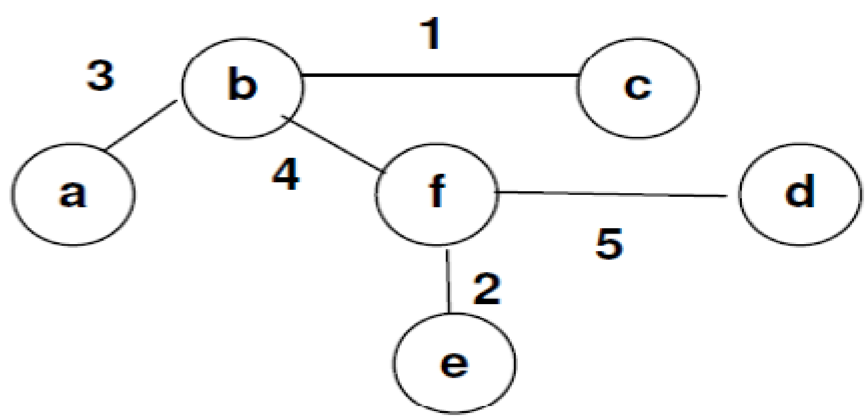
Edge	bc	
Weight	1	
Insertion status	YES	
Insertion order	1	
Edge	ef	
Weight	2	
Insertion status	YES	
Insertion order	2	

Edge	ab
Weight	3
Insertion status	YES
Insertion order	3

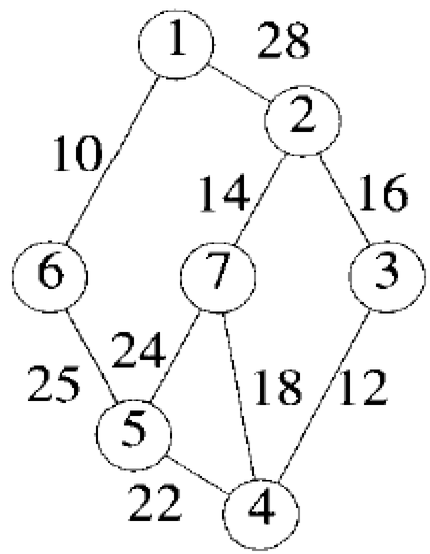


Edge	bf
Weight	4
Insertion status	YES
Insertion order	4



Edge	cf	
Weight	4	
Insertion status	**NO**	
Insertion order	-	
Edge	af	
Weight	5	
Insertion status	**NO**	
Insertion order	-	
Edge	df	
Weight	5	
Insertion status	YES	
Insertion order	5	
Algorithm stops as |V| -1 edges are included in the MST		

- Apply Kruskal’s algorithm for the following graph to find MST.



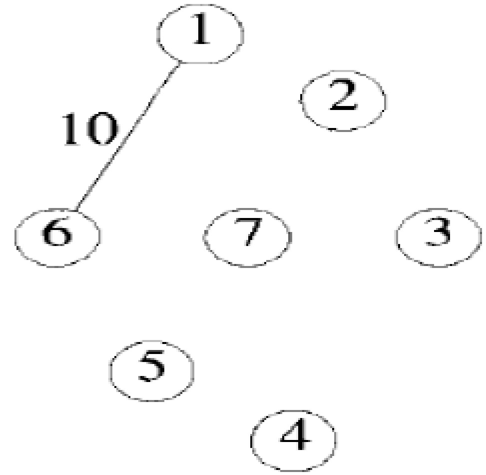
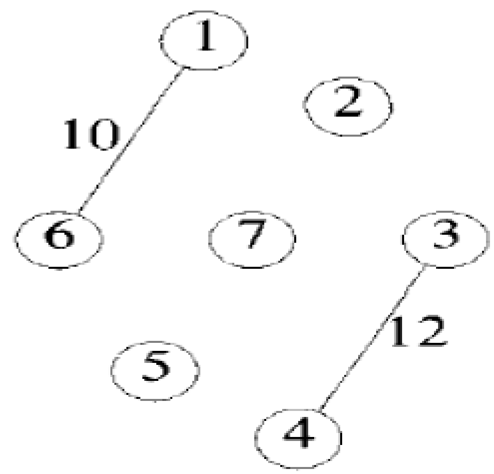
(a)

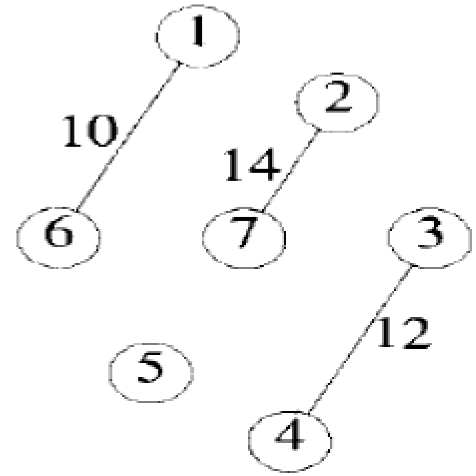
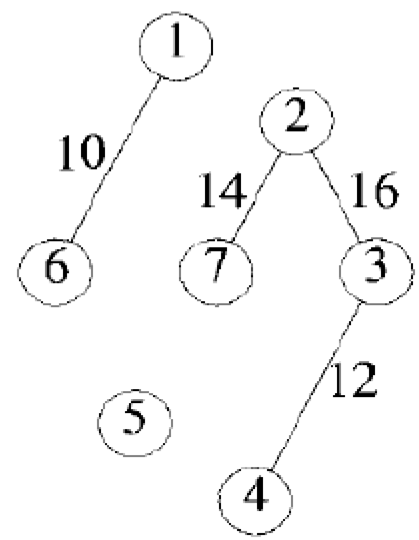
- The list of edges :

Edge	1,2	1,6	2,3	2,7	3,4	6,5	7,4	7,5	4,5
Weight	28	10	16	14	12	25	18	24	22

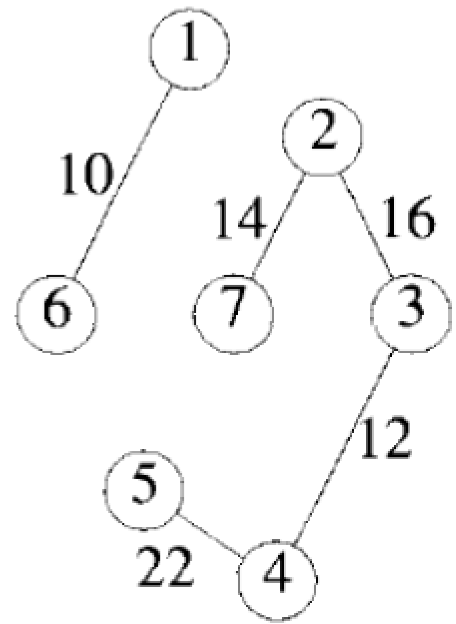
- Sort the edges in ascending order :

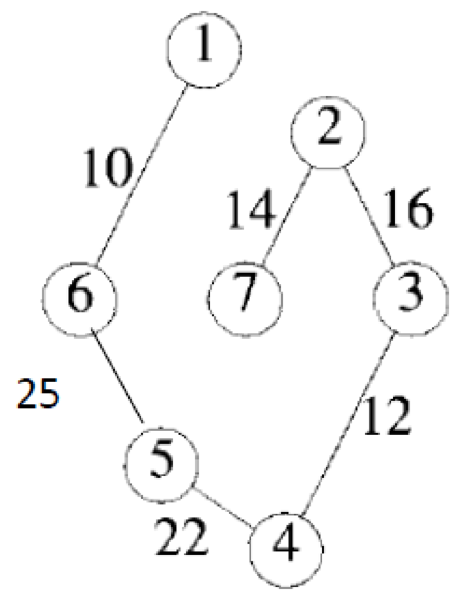
Edge	1,6	3,4	2,7	2,3	7,4	4,5	7,5	6,5	1,2
Weight	10	12	14	16	18	22	24	25	28

Edge	1,6	
Weight	10	
Insertion Status	Yes	
Insertion Order	1	
Edge	3,4	
Weight	12	
Insertion Status	Yes	
Insertion Order	2	

<table>
<tr><td>Edge</td><td>2,7</td><td rowspan="4">
</td></tr>
<tr><td>Weight</td><td>14</td></tr>
<tr><td>Insertion Status</td><td>Yes</td></tr>
<tr><td>Insertion Order</td><td>3</td></tr>
<tr><td>Edge</td><td>2,3</td><td rowspan="4">
</td></tr>
<tr><td>Weight</td><td>16</td></tr>
<tr><td>Insertion Status</td><td>Yes</td></tr>
<tr><td>Insertion Order</td><td>4</td></tr>
</table>

Edge	7,4
Weight	18
Insertion Status	No
Insertion Order	--
Edge	4,5
Weight	22
Insertion Status	Yes
Insertion Order	5



Edge	7,5	
Weight	24	
Insertion Status	No	
Insertion Order	--	
Edge	5,6	
Weight	25	
Insertion Status	Yes	
Insertion Order	6	

Algorithm Stops as |V| - 1 edges are included in the MST

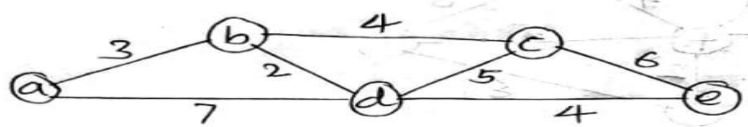
Minimum Cost (MST) = 10+12+14+16+22+25

= 99

KRUSKAL'S ALGORITHM EXAMPLE-2

28

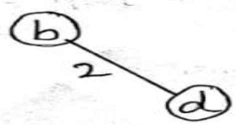
Example 2: Apply kruskal's Algorithm for given graph & find the minimum cost.



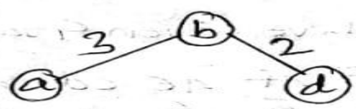
Solution:

Generate sorted list of edges for given Graph.

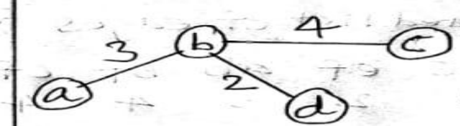
bd	ab	bc	de	cd	ce	ad
2	3	4	4	5	6	7



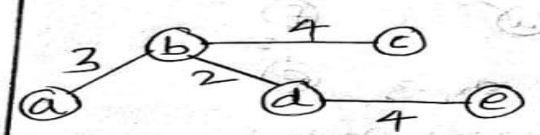
Step-1



Step-2



step-3



Step-4

Minimum cost = $W(T) = 3+4+2+4$

$= 13.$

Single Source Shortest Path

Dijkstra’s Algorithm

SINGLE SOURCE SHORTEST PATH PROBLEM

- It is a problem in which, consider one source vertex in a given weighted connected graph and find shortest paths to all its other vertices from source vertex
- That is to generate separate paths from source vertex to remaining vertex of shortest distance.
- Dijkstra’s algorithm is the best-known algorithm used to solve single source shortest-paths problem.

Algorithm

Algorithm Dijkstra(G, s)
//Input: Weighted connected graph G and source vertex s
//Output: The length Dv of a shortest path from s to v and its penultimate vertex Pv for every vertex v in V
for every vertex v in V do
$D_v \leftarrow \infty$
$P_v \leftarrow$ null // Pv , the parent of v
$d_s \leftarrow 0$
$V_T \leftarrow \emptyset$
for i $\leftarrow$ 0 to |V| - 1 do
$V_T \leftarrow V_T \cup \{u^*\}$
for every vertex u in V – V_T that is adjacent to u* do
if Du* + w (u*, u) < Du
Du $\leftarrow$ Du + w (u*, u)
Pu $\leftarrow$ u*

The Method :

Dijkstra's algorithm solves the single source shortest path problem in 2 stages.

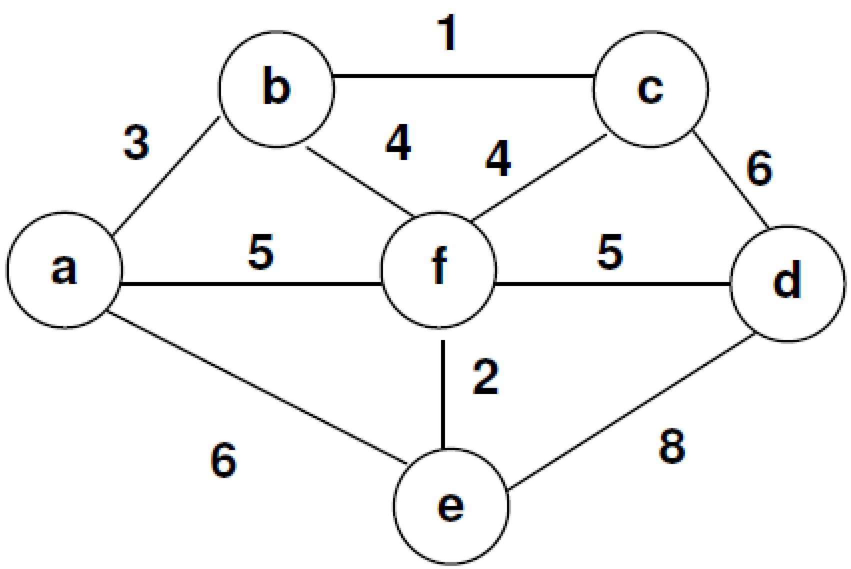
- Stage 1 : A greedy algorithm computes the shortest distance from source to all other nodes in the graph and saves in a data structure.
- Stage 2 : Uses the data structure for finding a shortest path from source to any vertex v.
  1. At each step, and for each vertex x, keep track of a "distance" D(x) and a directed path P(x) from root to vertex x of length D(x).
  2. Scan first from the root and take initial paths P(r, x) = (r, x) with

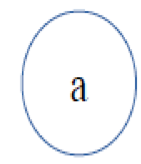
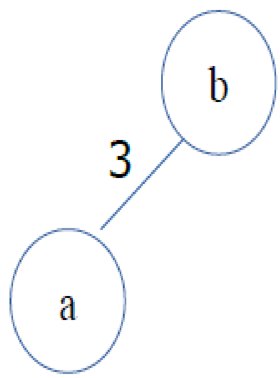
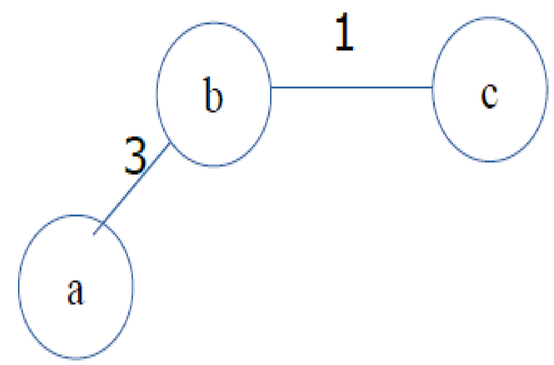
     D(x) = w(rx)   when rx is an edge,

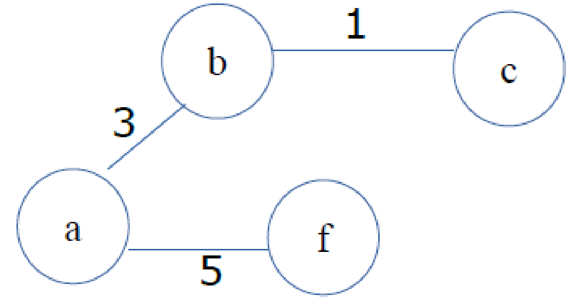
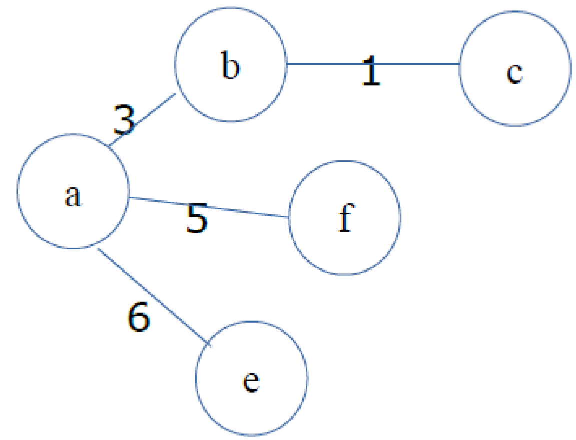
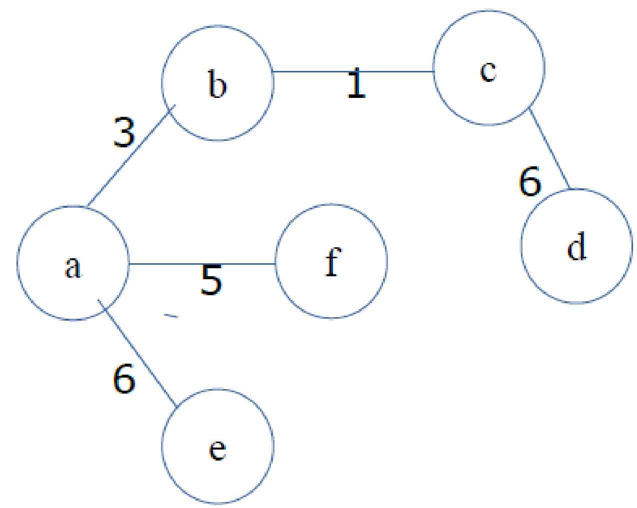
     D(x) = ∞ when rx is not an edge.
  3. For each temporary vertex y distinct from x, set D(y) = min{ D(y), D(x) + w(xy) }

Example:

- Apply Dijkstra's algorithm to find Single source shortest paths with vertex a as the source.



Nodes	Tree Vertices	Remaining vertices	Graph
{a}	a (-, 0)	b (a , 3) c (- , ∞) d (- , ∞) e (a , 6) f (a , 5)	
{a,b}	b (a, 3)	c (b , 3+1) d (- , ∞) e (a , 6) f (a , 5)	
{a,b,c}	c (b, 4)	d (c , 4+6) e (a , 6) f (a , 5)	

{a,b,c,f}	f (a, 5)	d (c , 10) e (a , 6)	
{a,b,c,f,e}	e (a, 6)	d (c, 10)	
{a,b,c,f,e,d}	d(c, 10)	----	

Example:

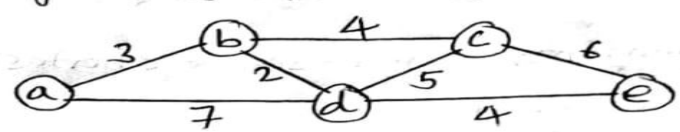
- Apply Dijkstra’s algorithm to find Single source shortest paths with vertex a as the source.

Dijkstra’s Algorithm – Example1

30.

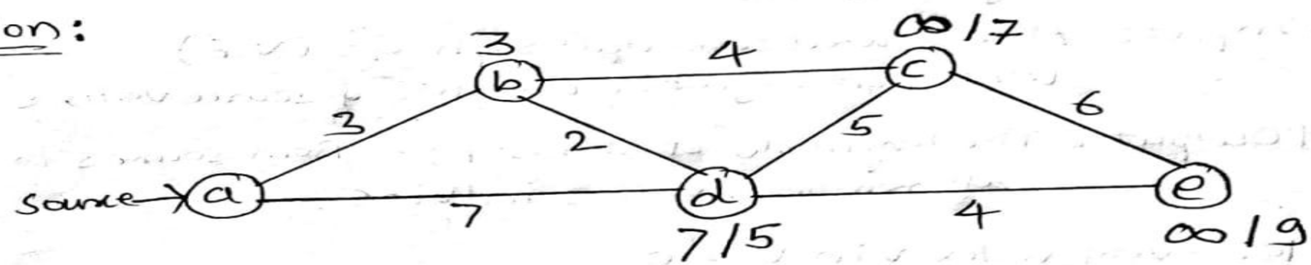
DIJKSTRA'S ALGORITHM EXAMPLE

Exaple- Solve single source shortest path problem for below given graph using Dijkstra's Algorithm -



Consider vertex (a) as the source vertex -

Solution:



Shortest Path from Source to All vertices are -

From a to b : $a \xrightarrow{3} b$ of length 3

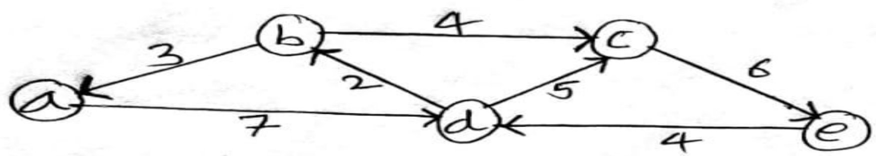
From a to c : $a \xrightarrow{3} b \xrightarrow{4} c$ of length 7

From a to d : $a \xrightarrow{3} b \xrightarrow{2} d$ of length 5

From a to e : $a \xrightarrow{3} b \xrightarrow{2} d \xrightarrow{4} e$ of length 9

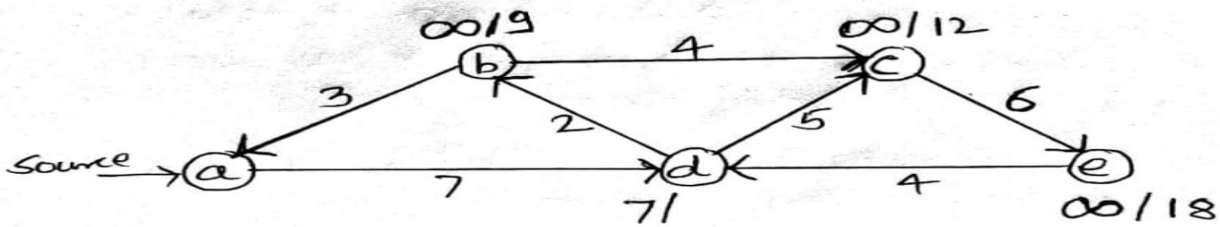
Dijkstra’s Algorithm – Example2

DIJKSTRAM'S ALGORITHM EXAMPLE 31



Find single source shortest path for above given graph from source vertex ⓐ. –

Solution:



Shortest Path from source ⓐ to all vertices –

From a to b – $a \xrightarrow{7} d \xrightarrow{2} b$ of length 9

From a to c – $a \xrightarrow{7} d \xrightarrow{5} c$ of length 12

From a to d – $a \xrightarrow{7} d$ of length 7

From a to e – $a \xrightarrow{7} d \xrightarrow{5} c \xrightarrow{6} e$ of length 18

Huffman Trees & Codes

Huffman Coding

- Huffman coding is lossless data compression technique widely used while transmitting data over network or storing data on the disk
- Huffman coding assigns codeword's of different lengths to different characters. Hence, it is a variable-length encoding technique
- Fixed-length encoding assigns codeword’s of same length to different characters

Fixed Length Vs. Variable Length

Fixed Length Encoding Example:

- Assume 4 characters A, B, C, D are given
- Then minimum 3-bits are required to encode all 4-characters
- Each character are represented by using 3-bits as shown below-A=000, B=001, C=010, D=011
- Hence final codeword results as- 000 001 010 011
- Total 12 bits are used to encode 4-characters A, B, C, D
- It takes more time to transfer data over network
- Also takes more storage space to save data on disk

Variable Length Encoding Example:

- Assume 4 characters A, B, C, D are given
- Then different length bits are used to encode all 4-characters
- Each character are represented by using variable length bits as shown below- A=00, B=001, C=10, D=111
- Hence final codeword results as-

  00 001 10 111
- Only 10 bits are used to encode 4-charactes A, B, C, D
- It takes less time to transfer data over network
- Also takes less storage space to save data on disk

Huffman algorithm to Construct Huffman Tree

- **Step 1:-** Initialize n one-node trees and label them with the characters of the alphabet. Record the frequency of each character in its tree's root to indicate the tree's weight
- **Step 2:-** Repeat the following operation until a single tree is obtained. Find two trees with the smallest weight and make them the left and right subtree of new tree and record the sum of their weights in the root of new tree as its weight

A tree constructed by the above algorithm is called a ***Huffman tree***. It defines—in the manner described—a ***Huffman code***.

EXAMPLE Consider the five-character alphabet {A, B, C, D, _} with the following occurrence probabilities:

character	A	B	C	D	_
probability	0.35	0.1	0.2	0.2	0.15

The Huffman tree construction for this input is shown in Figure 9.11. The resulting codewords are as follows:

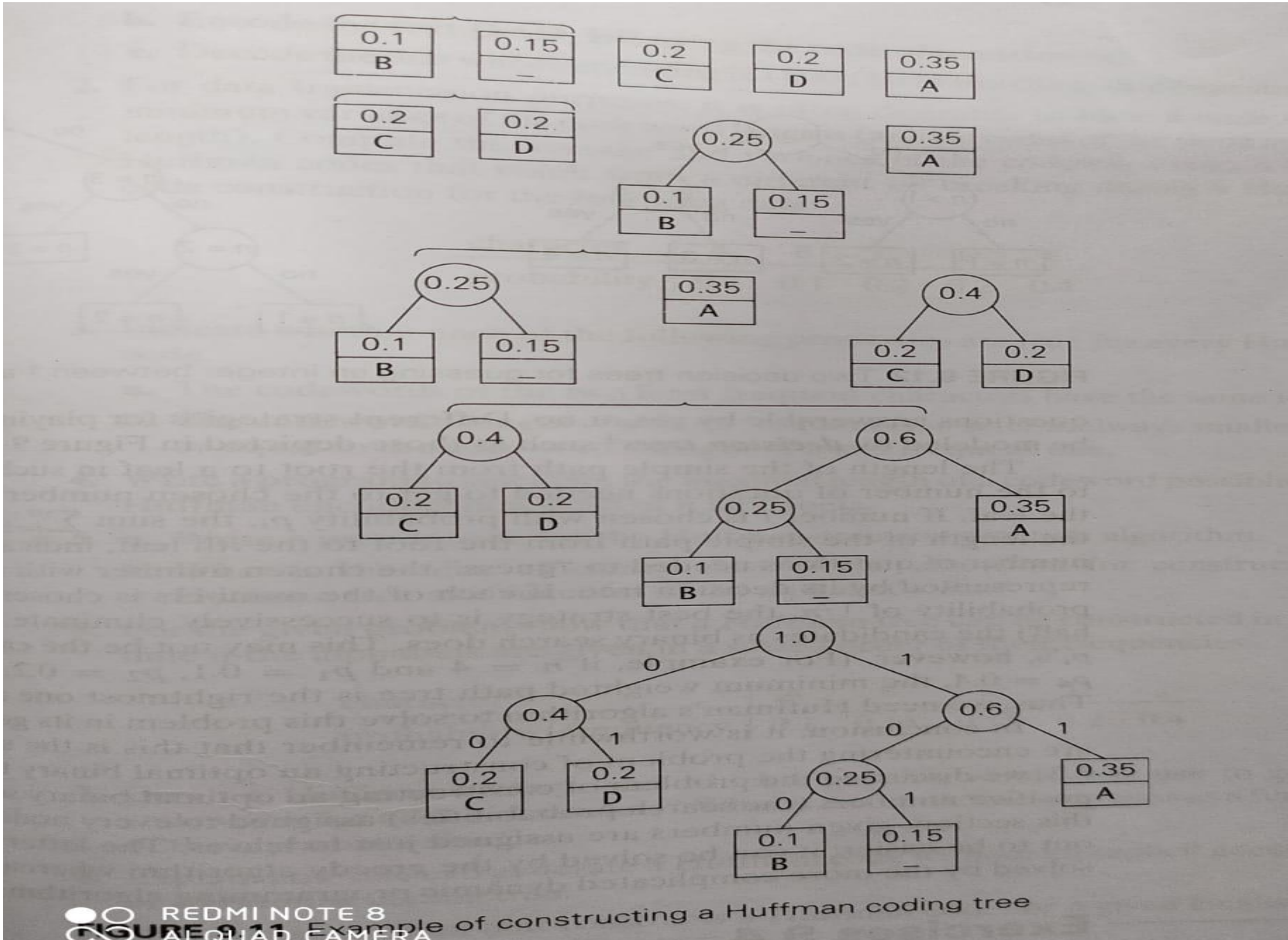


FIGURE 9.11 Example of constructing a Huffman coding tree

Encoding & Decoding

character	A	B	C	D	_
probability	0.35	0.1	0.2	0.2	0.15
codeword	11	100	00	01	101

- From above table, now characters DAD is encoded as 01 11 01
- BAD is encoded as 100 11 01
- Similarly codeword 100 11 01 101 11 01 is decoded as BAD_AD

Transform and Conquer

The General Method

- It deals with a group of design methods that are based on the idea of transformation.
- These methods work as two-stage procedures.
  - First, in the transformation stage, the problem's instance is modified to be, for one reason or another, more amenable to solution.
  - Then, in the second or conquering stage, it is solved.

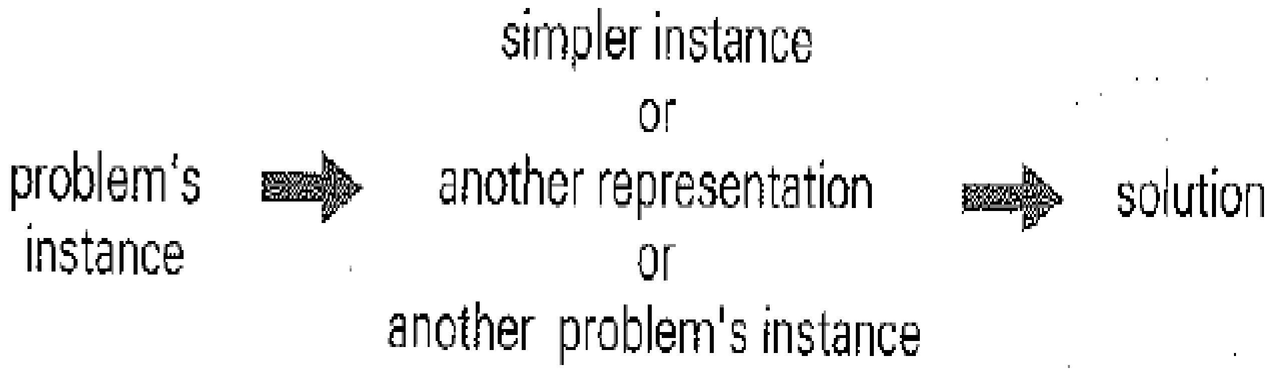


FIGURE 6.1 Transform-and-conquer strategy

Major Variations of Transform and Conquer

- There are three major variations of this idea that differ by what we transform a given instance to
  - *transformation to a simpler or more convenient instance of the same problem-we call it instance simplification*
  - *transformation to a different representation of the same instance-we call it representation change*
  - *transformation to an instance of a different problem for which an algorithm is already available-we call it problem reduction*

Heap Definition

- A heap can be defined as a BINARY TREE with keys assigned to its each nodes that satisfies following two conditions-
- **Condition 1: The Binary Tree shape Requirement**
  - BT is complete BT, that is, all its levels are full expect last level , where only some rightmost leaves may be missing
- **Condition 2: The Parental dominance Requirement**
  - The key at each parent node is greater than or equal to the keys at its children. It is called as Max heap.
  - The key at each parent node is less than or equal to the keys at its children. It is called as Min heap.

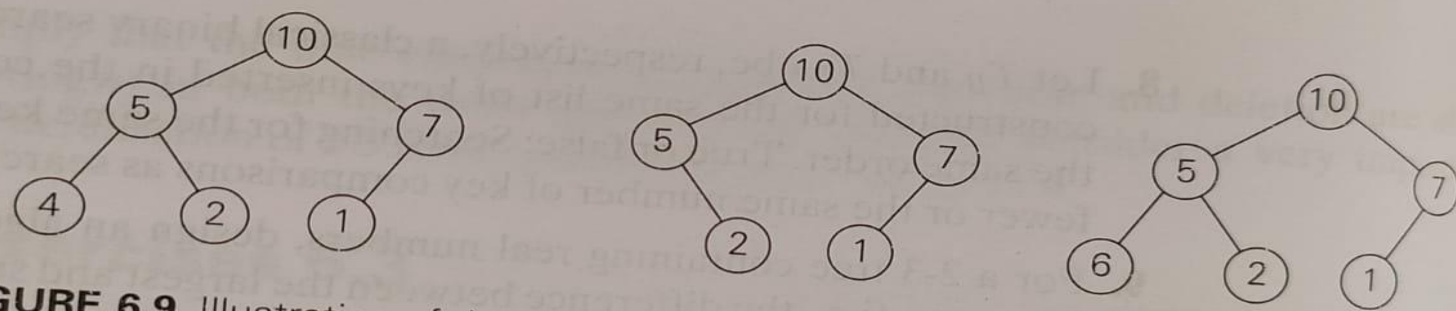


FIGURE 6.9 Illustration of the definition of "heap": only the leftmost tree is a heap.

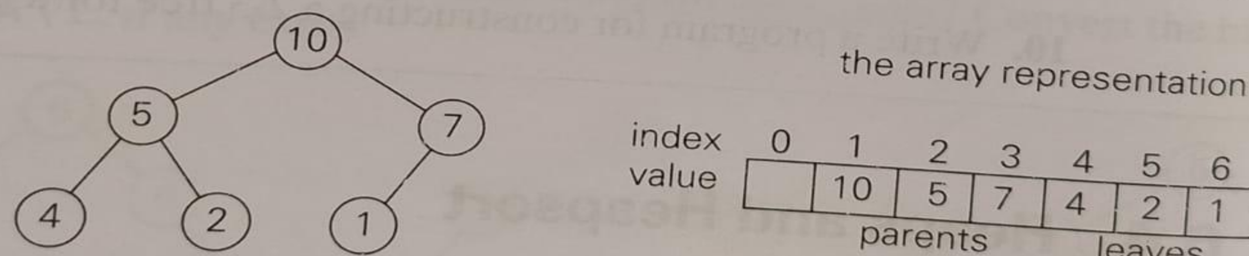
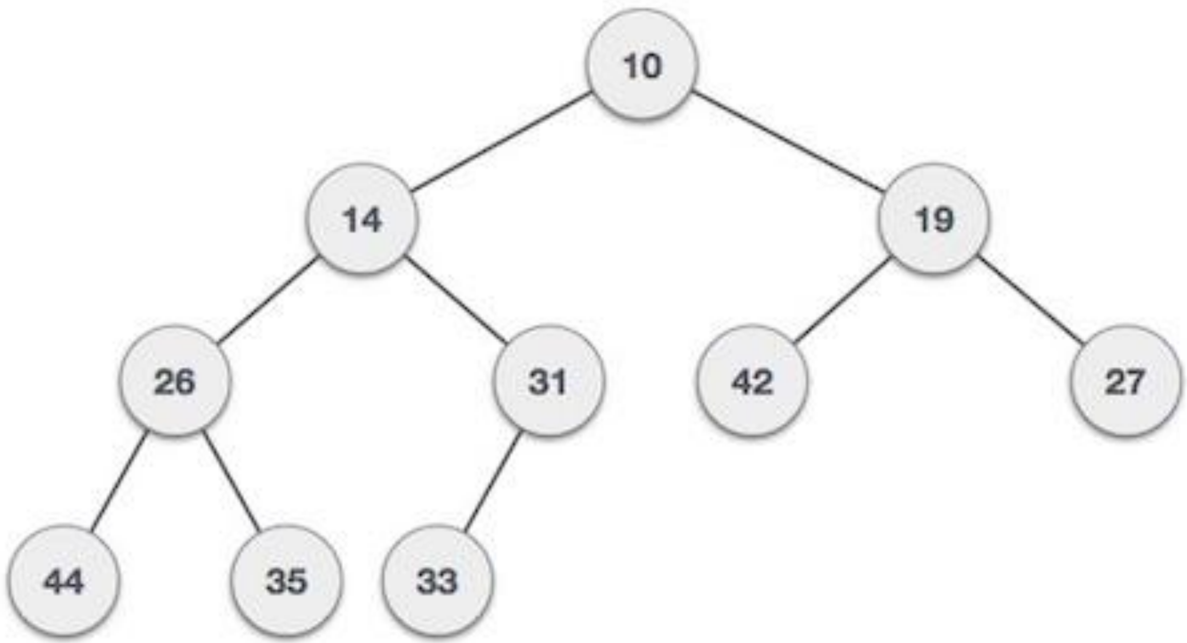


FIGURE 6.10 Heap and its array representation

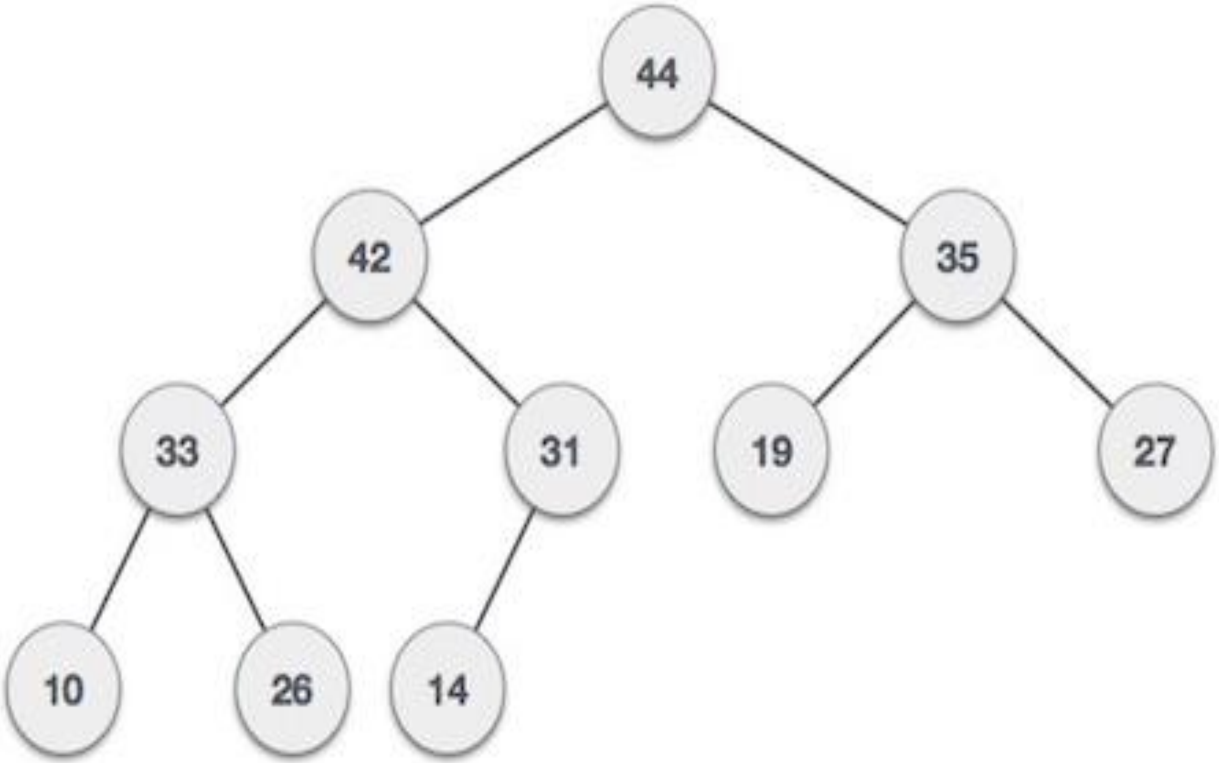
Min Heap Example

For Input → 35, 33, 42, 10, 14, 19, 27, 44, 26, 31



Max Heap Example

For Input → 35, 33, 42, 10, 14, 19, 27, 44, 26, 31



Max Heap Construction Algorithm

- **Step 1** – Create a new node at the end of heap.
- **Step 2** – Assign new value to the node.
- **Step 3** – Compare the value of this child node with its parent.
- **Step 4** – If value of parent is less than child, then swap them.
- **Step 5** – Repeat step 3 & 4 until Heap property holds.

Bottom-Up Heap Construction

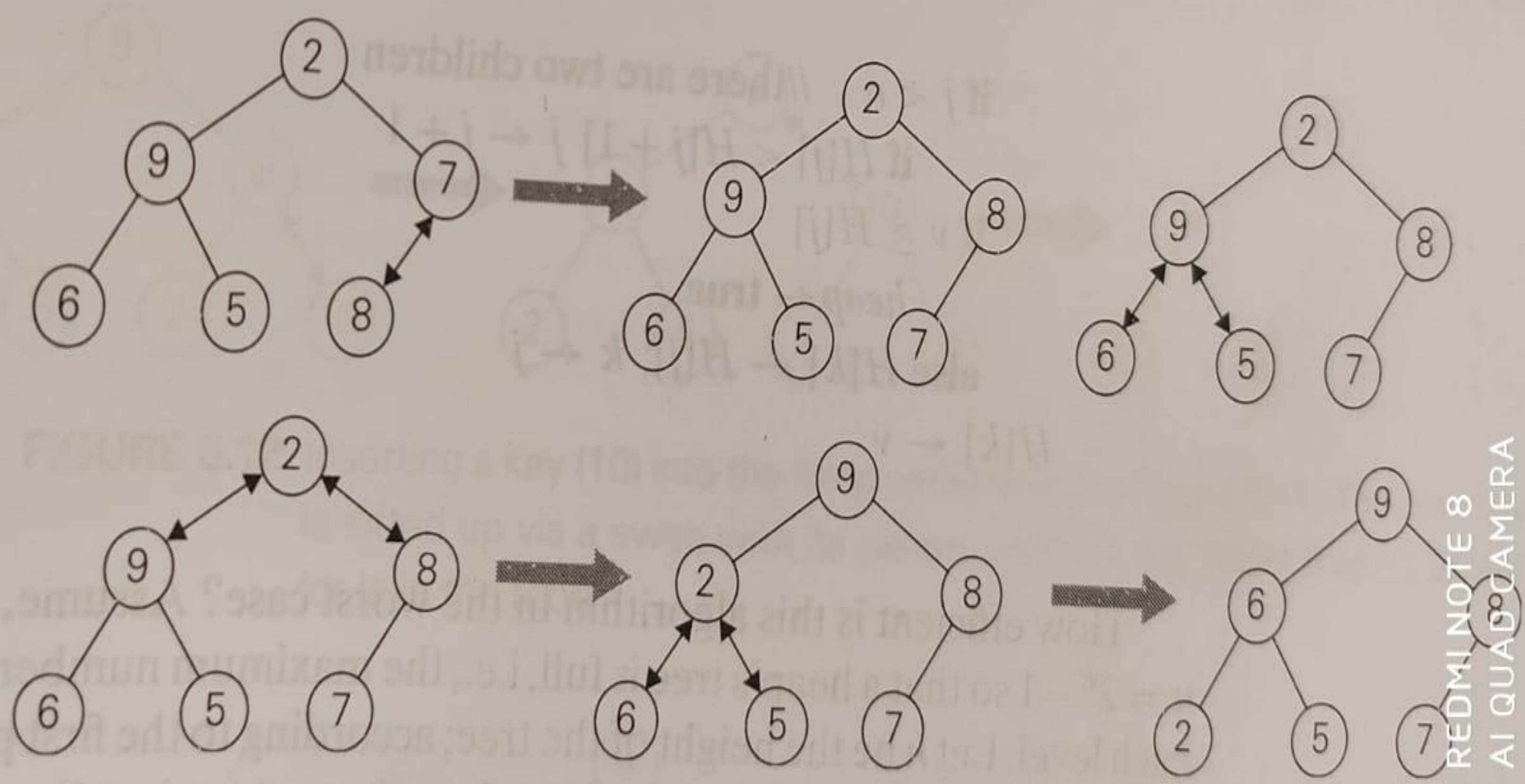


FIGURE 6.11 Bottom-up construction of a heap for the list 2, 9, 7, 6, 5, 8

Max Heap Deletion Algorithm

- **Step 1** – Remove root node.
- **Step 2** – Move the last element of last level to root.
- **Step 3** – Compare the value of this child node with its parent.
- **Step 4** – If value of parent is less than child, then swap them.
- **Step 5** – Repeat step 3 & 4 until Heap property holds.

Maximum Key Deletion from a Heap

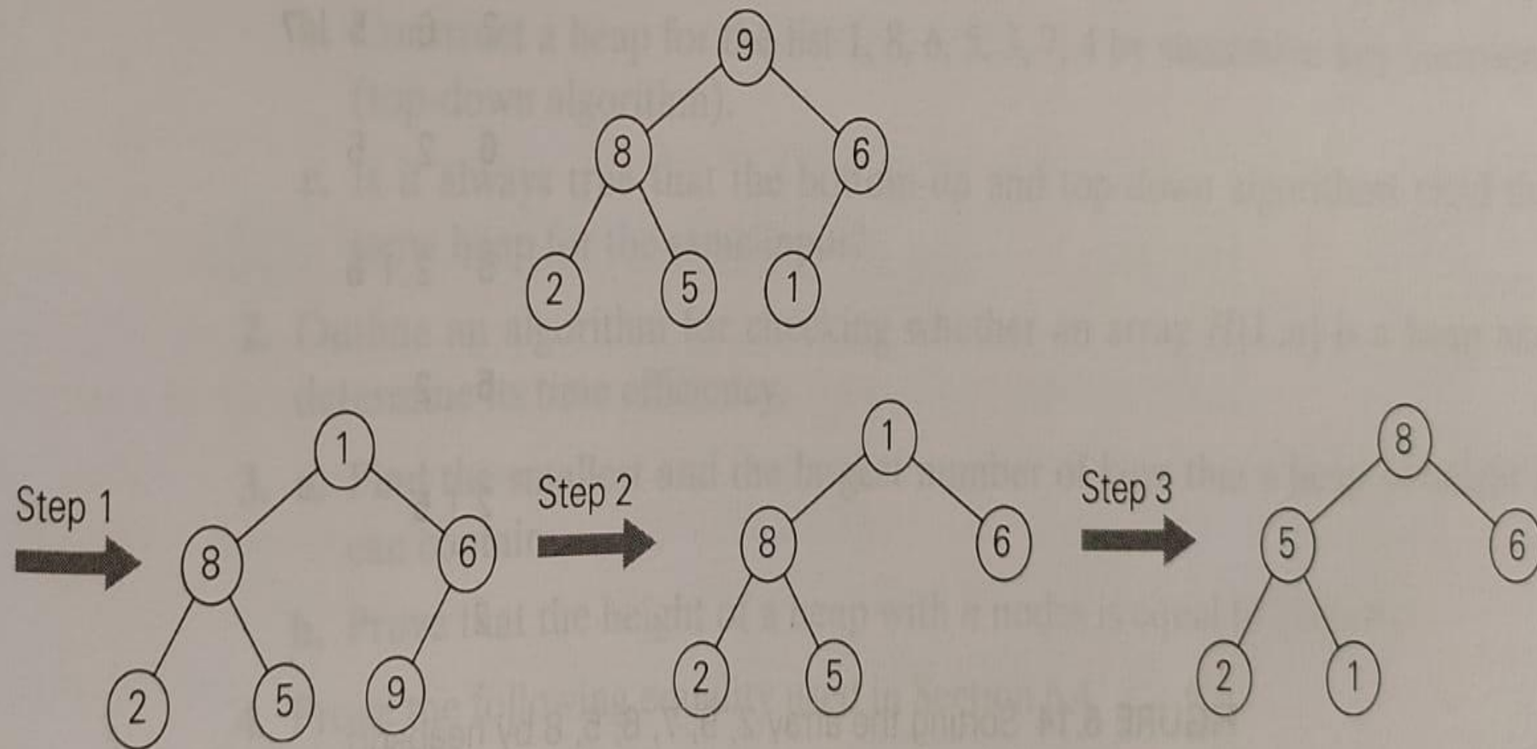


FIGURE 6.13 Deleting the root's key from a heap. The key to be deleted is swapped with the last key, after which the smaller tree is "heapified" by exchanging the new key at its root with the larger key at its children until the parental dominance requirement is satisfied.

Heapsort Algorithm

Heapsort algorithm has two stages as shown below-

stage 1: Heap Construction

Construct a heap for a given array

stage 2: Maximum Deletion

Apply the root deletion operation n-1 times for remaining heap

HeapSort Example

Stage 1 (heap construction)

2	9	**7**	6	5	8
2	**9**	8	6	5	7
2	9	8	6	5	7
9	**2**	8	6	5	7
9	6	8	2	5	7

Stage 2 (maximum deletions)

9	6	8	2	5	7
7	6	8	2	5	I **9**
8	6	7	2	5	
5	6	7	2	I **8**	
7	6	5	2		
2	6	5	I **7**		
6	2	5			
5	2	I **6**			
5	2				
2	I **5**				
2					

FIGURE 6.14 Sorting the array 2, 9, 7, 6, 5, 8 by heapsort

- Construct a heap for the list 1, 8, 6, 5, 3, 7, 4 by the bottom-up algorithm.