10 Greedy Technique

10.1 Introduction

In an algorithmic strategy like Greedy, the decision of solution is taken based on the information available. The Greedy method is a straight forward method. This method is popular for obtaining the optimized solutions. In Greedy technique, the solution is constructed through a sequence of steps, each expanding a partially constructed solution obtained so far, until a complete solution to the problem is reached. At each step the choice made should be,

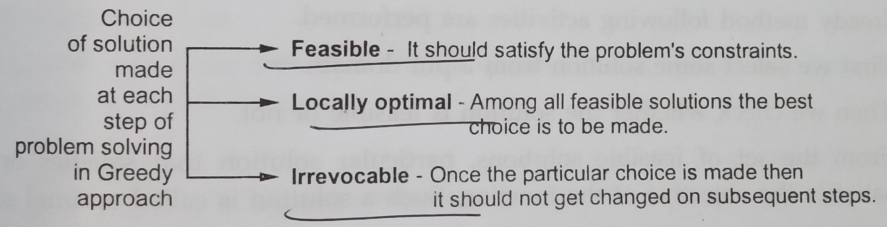


Fig. 10.1

In short, while making a choice there should be a Greed for the optimum solution.

In this chapter, we will first understand the general method of Greedy algorithm. Then we will discuss the minimum spanning tree algorithms : Prim's and Kruskal's algorithm. In the later part of this chapter we will discuss the shortest path algorithm and learn the method of obtaining Huffman's tree. All these problems always demand for the optimized solution and hence Greedy method is the specific approach which is to be used to solve these problems.

10.2 General Method

In this section we will understand *"What is Greedy method?"*.

Algorithm

```
Greedy (D, n)
//In Greedy approach D is a domain
//from which solution is to be obtained of size n
//Initially assume
    Solution ← 0
    for i ← 1 to n do
                    {
        s ← select (D) // section of solution from D
        if (Feasible (solution, s)) then
            solution ← Union (solution, s);
    }
    return solution
```

Check if the selected solution is feasible or not.

Make a feasible choices and select optimum solution.

Fig. 10.2

In Greedy method following activities are performed.

1. First we select some solution from input domain.
2. Then we check whether the solution is feasible or not.
3. From the set of feasible solutions, particular solution that satisfies or nearly satisfies the objective of the function. Such a solution is called optimal solution.
4. As Greedy method works in stages. At each stage only one input is considered at each time. Based on this input it is decided whether particular input gives the optimal solution or not.

10.2.1 Comparison between Greedy Algorithm and Dynamic Programming

In this section we will discuss *"What are the differences and similarities between Greedy algorithm and dynamic programming?"*

Sr. No.	Greedy Method	Dynamic Programming
1.	Greedy method is used for obtaining **optimum solution.**	Dynamic programming is also for obtaining **optimum solution.**
2.	In Greedy method a set of **feasible solutions** and the picks up the optimum solution.	There is **no** special set of **feasible solutions** in this method.

3.	In Greedy method the optimum selection is **without revising previously** generated solutions.	Dynamic programming considers **all possible sequences** in order to obtain the optimum solution.
4.	In Greedy method there is **no** as such **guarantee** of getting **optimum solution.**	It is guaranteed that the dynamic programming will generate optimal solution using **principle of optimality.**

With these fundamental issues of Greedy method, let us now discuss various applications of Greedy algorithm.

10.3 Prim's Algorithm

Prim's Algorithm works for obtaining minimum spanning tree. Let us first understand the concepts of spanning tree and minimum spanning tree.

Spanning Tree

A spanning tree of a graph G is a subgraph which is basically a tree and it contains **all the vertices** of G containing **no circuit.**

Minimum Spanning Tree

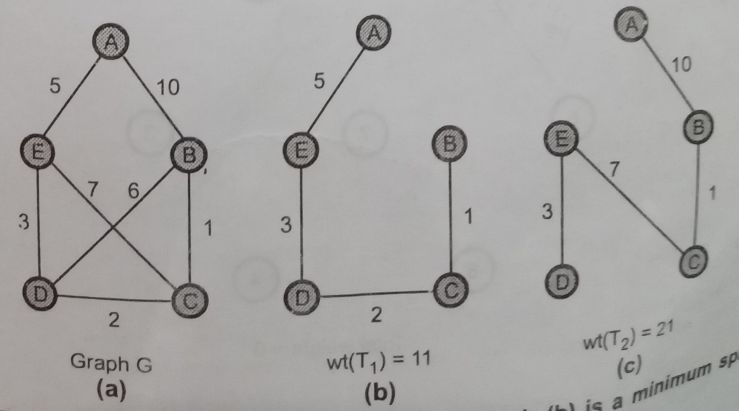
A minimum spanning tree of a weighted connected graph G is a spanning tree with minimum or smallest weight.

Weight of the Tree

A weight of the tree is defined as the sum of weights of all its edges.

For example :

Consider a graph G as given below. This graph is called weighted connected graph because some weights are given along every edge and the graph is a connected graph.



Graph G (a) (b) (c)

Fig. 10.3 Graph and two spanning trees out of which (b) is a minimum spanning tree

Applications of spanning trees :

1. Spanning trees are very important in designing efficient routing algorithms.
2. Spanning trees have wide applications in many areas such as network design.

Example of Prim's Algorithm

Let us understand the prim's algorithm with the help of example.

Example : Consider the graph given below :

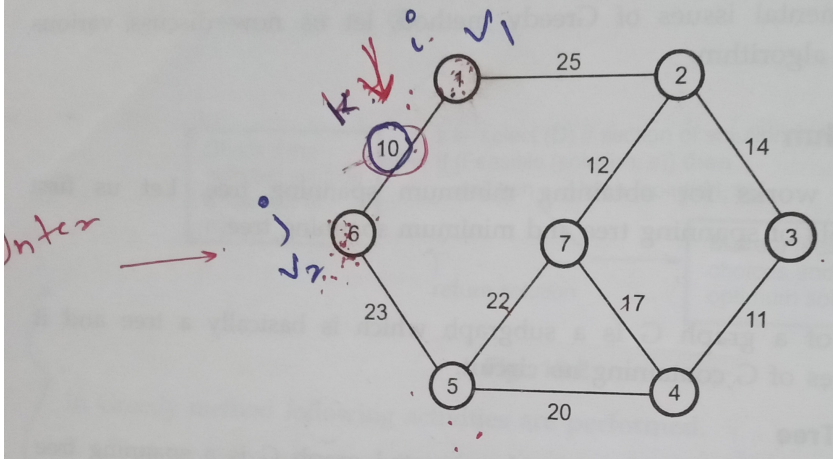
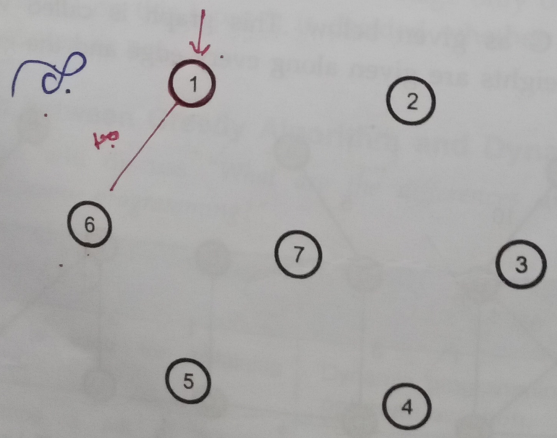


Fig. 10.4 Graph

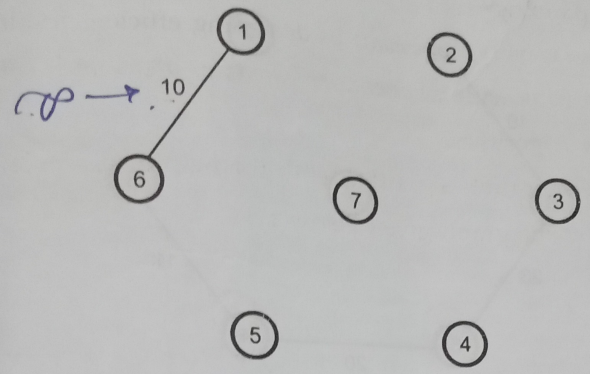
Now, we will consider all the vertices first. Then we will select an edge with minimum weight. The algorithm proceeds by selecting adjacent edges with minimum weight. Care should be taken for not forming circuit.

Step 1 :



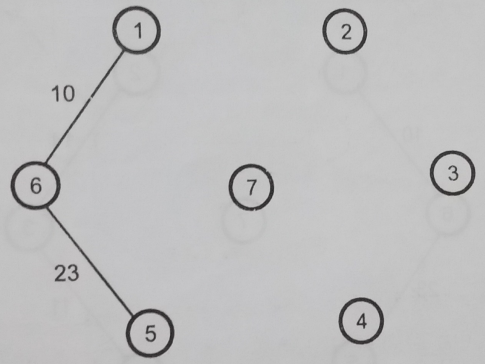
Total weight = 0

Step 2 :



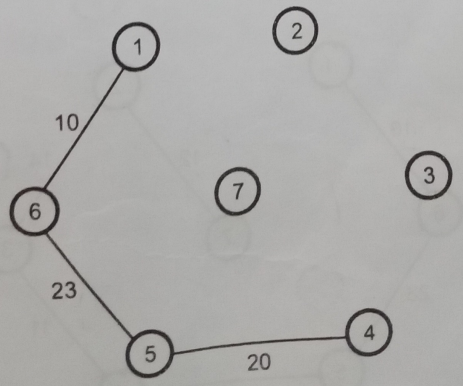
Total weight = 10

Step 3 :



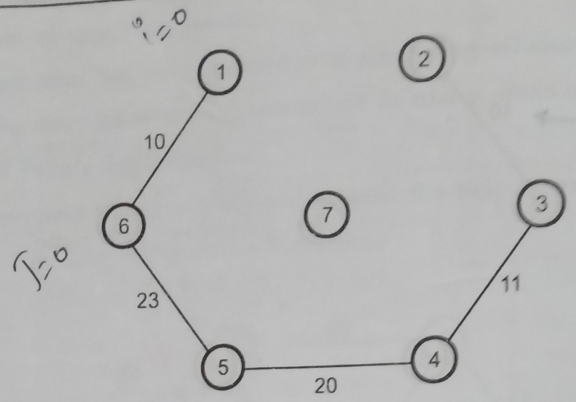
Total weight = 33

Step 4 :



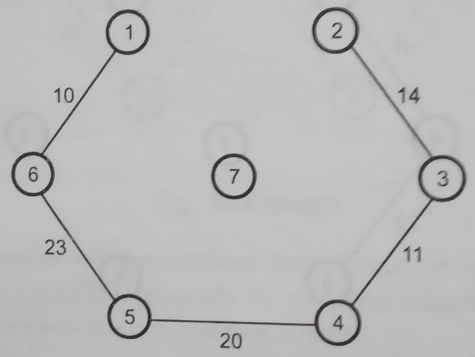
Total weight = 53

Step 5 :



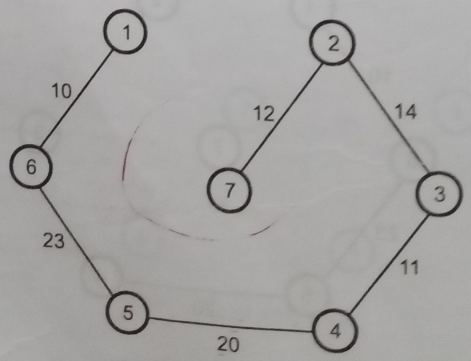
Total weight = 64

Step 6 :



Total weight = 78

Step 7 :



Total weight = 90

Prim's Algorithm

Algorithm

```
Prim(G[0…Size-1,0…Size-1],nodes)
//Problem Description :This algorithm is for implementing
//Prim's algorithm for finding spanning tree
//Input: Weighted Graph G and total number of nodes
//Output: Spanning tree gets printed with total path length
  total=0;
    // Initialize the selected vertices list
    for i←0 to nodes-1 do
      tree[i] ← 0
      tree[0] = 1;//take initial vertex
    for k←1 to nodes do
    {
      min_dist ← ∞
    //initially assign minimum dist as infinity
    for i←0 to nodes-1 do
     {
       for j←0 to nodes-1 do
        {
          if (G[i,j]AND((tree[i]AND tree[j])OR
              (!tree[i] AND tree[j])))then
            if(G[i,j] < min_dist)then
              {
                min_dist←G[i,j]
                v1←i
                v2←j
              }
        }
     }
```

Select an edge such that one vertex is selected and other is not and the edge has the least weight.

Obtained edge with minimum wt.

Picking up those vertices yielding minimum edge.

```
Write(v1,v2,min_dist);
   tree[v1] ← tree[v2] ← 1
  total ← total+min_dist
  }
  Write("Total Path Length Is",total)
```

Mark the corresponding edge of selected vertices as an edge in spanning tree.

C function

```
void Prim(int G[][SIZE], int nodes)
{
    int tree[SIZE], i, j, k;
    int min_dist, v1, v2,total=0;
    // Initialize the selected vertices list
    for (i=0 ; i<nodes ; i++)
          tree[i] = 0;
    printf("\n\n The Minimal Spanning Tree Is :\n");
    tree[0] = 1;
    for (k=1 ; k<=nodes-1 ; k++)
    {
      min_dist = INFINITY;
     //initially assign minimum dist as infinity
       for (i=0 ; i<=nodes-1 ; i++)
         {
            for (j=0 ; j<=nodes-1 ; j++)
        {
          if (G[i][j] && ((tree[i] && !tree[j]) ||
              (!tree[i] && tree[j])))
          {
           if (G[i][j] < min_dist)
            {
              min_dist = G[i][j];
              v1 = i;
              v2 = j;
            }
```

```
            }
        }
    }

    printf("\n Edge (%d %d )and weight= %d",v1,v2,min_dist);
     tree[v1] = tree[v2] = 1;
    total =total+min_dist;
    }
    printf("\n\n\t Total Path Length Is = %d",total);
}
```

Analysis

The algorithm spends most of the time in selecting the edge with minimum length. Hence the **basic operation** of this algorithm is to find the edge with **minimum path length**. This can be given by following formula.

$$T(n) = \sum_{k=1}^{nodes-1} \left(\sum_{i=0}^{nodes-1} 1 + \sum_{j=0}^{nodes-1} 1 \right)$$

Time taken by for k=1 to nodes–1 loop

Time taken by for i=0 to nodes–1 loop

Time taken by for j=0 to nodes–1 loop

We take variable n for 'nodes' for the sake of simplicity of solving the equation then

$$T(n) = \sum_{k=1}^{n-1} \left(\sum_{i=0}^{n-1} 1 + \sum_{j=0}^{n-1} 1 \right)$$

$$= \sum_{k=1}^{n-1} [((n-1) - 0 + 1) + ((n-1) - 0 + 1)]$$

$$= \sum_{k=1}^{n-1} (2n)$$

$$= 2n \sum_{k=1}^{n-1} 1$$

$$= 2n[(n-1) - 1 + 1] = 2n(n-1)$$

$$= 2n^2 - 2n$$

$\therefore \quad T(n) = n^2$

$\therefore \quad T(n) = \Theta(n^2)$

But n stands for total number of nodes or vertices in the tree. Hence we state

> Time complexity of Prim's Algorithm is $\Theta(|V^2|)$.

But if the Prim's algorithm is implemented using binary heap with the crea graph using adjacency list then its time complexity becomes $\Theta(E \log_2 V)$ wh stands for total number of edges and V stands for total number of vertices.

C program

```
/*************************************************************
This program is to implement Prim's Algorithm using Greedy Method
*************************************************************/
# include<stdio.h>
# include<conio.h>
# define SIZE 20
# define INFINITY 32767

/*This function finds the minimal spanning tree by Prim's
Algorithm */

void Prim(int G[][SIZE], int nodes)
{
    int tree[SIZE], i, j, k;
    int min_dist, v1, v2,total=0;
// Initialize the selected vertices list
for (i=0 ; i<nodes ; i++)
    tree[i] = 0;
printf("\n\n The Minimal Spanning Tree Is :\n");
tree[0] = 1;
for (k=1 ; k<=nod
```

```
Edge  (0  4 )and weight = 5
Edge  (3  4 )and weight = 3
Edge  (2  3 )and weight = 2
Edge  (1  2 )and weight = 1
```

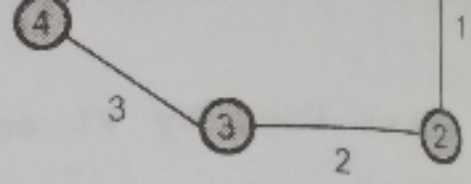


Fig. 10.6

```
Total Path Length Is = 11
```

10.4 Kruskal's Algorithm

Kruskal's algorithm is another algorithm of obtaining minimum spanning tree. This algorithm is discovered by a second year graduate student Joseph Kruskal. In this algorithm always the minimum cost edge has to be selected. But it is not necessary that selected optimum edge is adjacent.

Difference between Prim's and Kruskal's Algorithm

Prim's Algorithm	Kruskal's Algorithm
This algorithm is for obtaining minimum spanning tree by selecting the adjacent vertices of already selected vertices.	This algorithm is for obtaining minimum spanning tree but it is not necessary to choose adjacent vertices of already selected vertices.

Let us understand this algorithm with the help of some example.

Step 1

Step 2

Example : Consider the graph given below :

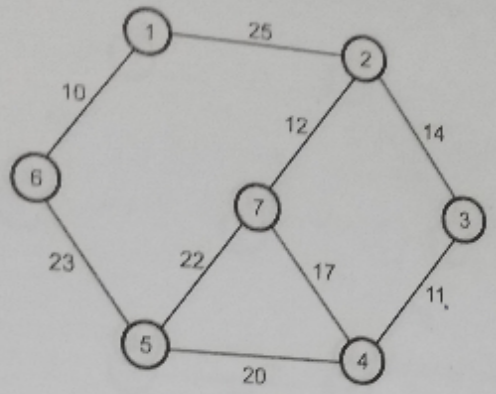
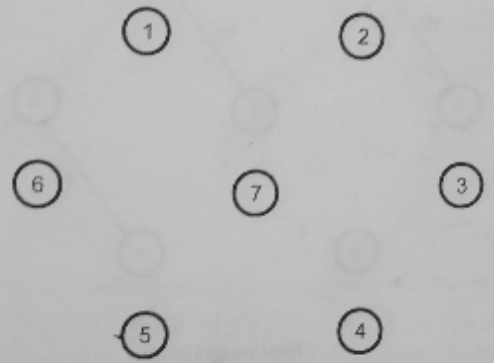


Fig. 10.7 Graph

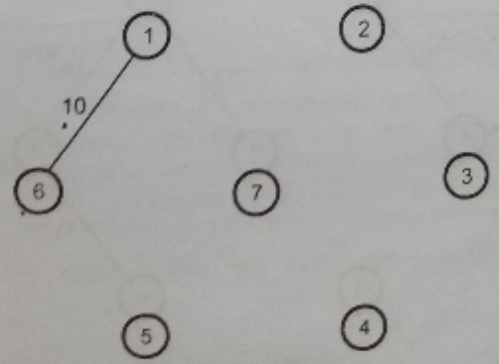
First we will select all the vertices. Then an edge with optimum weight is selected from heap, even though it is not adjacent to previously selected edge. Care should be taken for not forming circuit.

Step 1 :



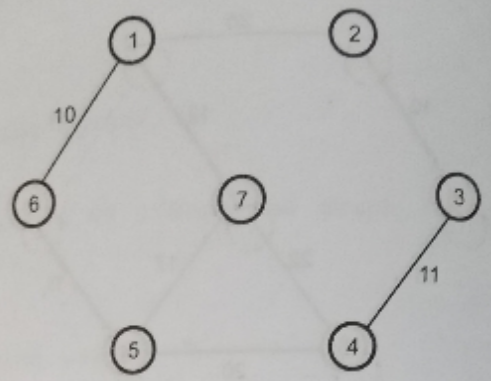
Total weight = 0

Step 2 :



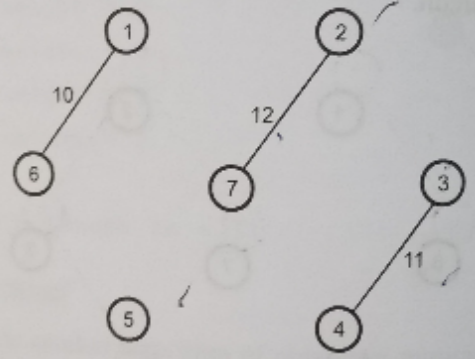
Total weight = 10

Step 3 :



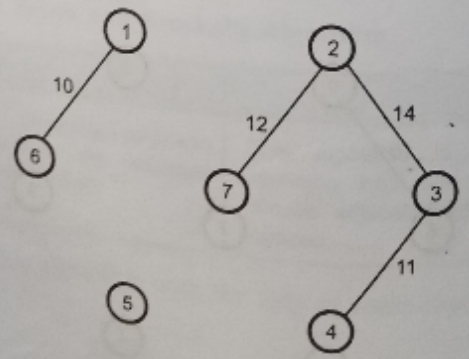
Total weight = 21

Step 4 :



Total weight = 33

Step 5 :



Total weight = 47

Analysis and D

Step 6 :

Step 7 :

10.4.1 Algo

Algorith

//Proble

//spanni

//Input:

//Output

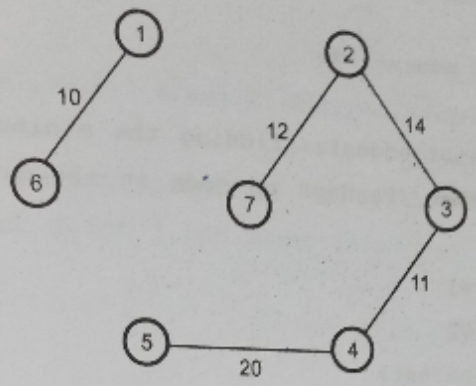
//spanni

count←

k←0

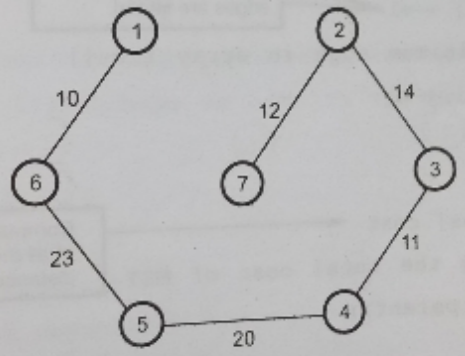
sum←0

Step 6 :



Total weight = 67

Step 7 :



Total weight = 90

10.4.1 Algorithm

```
Algorithm  spanning_tree()
//Problem Description: This algorithm finds the minimum
//spanning tree using Kruskal's algorithm
//Input:The adjacency matrix graph G containing cost
//Output: prints the spanning tree with the total cost
//spanning tree
 count←0
 k←0
 sum←0
```

```
for i←0 to tot_nodes do
   parent[i]←i
while(count!=tot_nodes-1)do
 {
   pos←Minimum(tot_edges);//finding the minimum cost edge
    if(pos=-1)then//Perhaps no node in the graph
      break
    v1←G[pos].v1
    v2←G[pos].v2
    i←Find(v1,parent)
    j←Find(v2,parent)
    if(i!=j)then
    {
     tree[k][0] ←v1
//storing the minimum edge in array tree[]
     tree[k][1] ←v2
     k++
     count++;
     sum+←G[pos].cost
  //accumulating the total cost of MST
     Union(i,j,parent);
    }
    G[pos].cost INFINITY
   }
   if(count=tot_nodes-1)then
   {
    for i←0 to tot_nodes-1
    {
       write(tree[i][0],tree[i][1])
    }
    write("Cost of Spanning Tree is ",sum)
   }
```

Tree [ ] [ ] is an array in which the spanning tree edges are stored.

Computing total cost of all the minimum distances.

For each node of i, the minimum distance edges are collected in array **tree [ ] [ ]**. The spanning tree is printed here.

```
Enter Edge in (V1 V2) form 4 3

Enter Corresponding Cost 5

Spanning tree is...
-------------------------
[1 - 4][1 - 2][1 - 3]
-------------------------
Cost of Spanning Tree is = 6
```

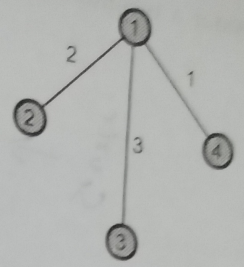


Fig. 10.9 Spanning tree

10.5 Dijkstra's Algorithm

Dijkstra's Algorithm is a popular algorithm for finding shortest path. This algorithm is called **single source shortest path** algorithm. In this algorithm, for a given vertex called source the shortest path to all other vertices is obtained. In this algorithm the main focus is not to find only one single path but to find the shortest paths from any vertex to all other remaining vertices.

This algorithm applicable to graphs with non-negative weights only.

Dijkstra's algorithm finds shortest paths to graph's vertices in order of their distance from a given source. In this process of finding shortest path, first it finds the shortest path from the source to a vertex nearest to it, then second nearest and so on.

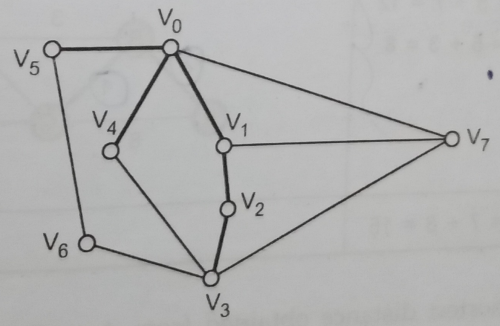


Fig. 10.10

The shortest path from V_0 is obtained. First we find shortest path from $V_0 - V_1$ then $V_1 - V_2$ then from $V_2 - V_3$ the shortest distance is obtained. Let us understand this algorithm with some example.

Consider a weighted connected graph as given below.

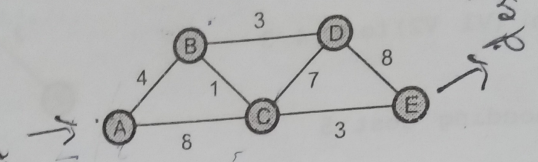
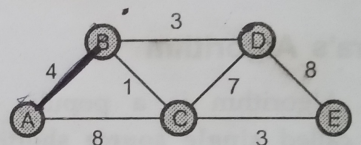
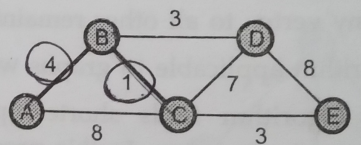
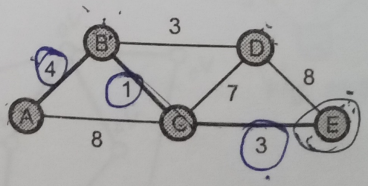


Fig. 10.11

Now we will consider each vertex as a source and will find the shortest distance from this vertex to every other remaining vertex. Let us start with vertex A.

Source vertex	Distance with other vertices	Path shown in graph
A	A-B, path = 4 A-C, path = 8 A-D, path = ∞ A-E path = ∞	
B	B-C, path = 4 + 1 = 5 B-D, path = 4 + 3 B-E path = ∞	
C	C-D, path = 5 + 7 = 12 C-E, path = 5 + 3 = 8	
D	D-E, path = 7 + 8 = 15	

But we have one shortest distance obtained from A to E and that is A - B - C - E with path length = 4 + 1 + 3 = 8. Similarly other shortest paths can be obtained by choosing appropriate source and destination.

```
Algorithm Dijkstra(int cost[1...n,1...n],int source,int dist[])
for i ← 0 to tot_nodes
{
   dist[i] ← cost[source,i]//initially put the
   s[i] ← 0 //distance from source vertex to i
   //i is varied for each vertex
   path[i] ← source//all the sources are put in path
}
s[source] ← 1
for(i ← 1 to tot_nodes)
{
   min_dist ← infinity;
   v1 ← -1//reset previous value of v1
   for(j ← 0 to tot_nodes-1)
   {
      if(s[j]=0)then
      {
         if(dist[j]<min_dist)then
         {
            min_dist ← dist[j]
            v1 ← j
         }
      }
   }
   s[v1] ← 1
   for(v2 ← 0 to tot_nodes-1)
   {
     if(s[v2]=0)then
     {
        if(dist[v1]+cost[v1][v2]<dist[v2])then
        {
           dist[v2] ← dist[v1]+cost[v1][v2]
```

Start from each source node

Finding minimum distance from selected source node.That is : source-j represents min. dist. edge

```
                path[v2] ← v1
            }
        }
    }
}
```

v_1 is next selected destination vertex with shortest distance. All such vertices are accumulated in array **path[ ]**

C function

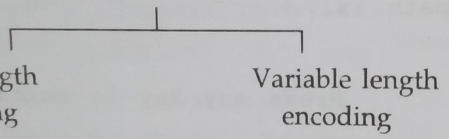
```
void Dijkstra(int tot_nodes,int cost[10][10],int
dist[])
{
  int i,j,v1,v2,min_dist;
  int s[10];
  for(i=0;i<tot_nodes;i++)
  {
     dist[i]=cost[source][i];//initially put the
     s[i]=0; //distance from source vertex to i
     //i is varied for each vertex
     path[i]=source;//all the sources are put in pa
  }
  s[source]=1;
  for(i=1;i<tot_nodes;i++)
  {
     min_dist=infinity;
     v1=-1;//reset previous value of v1
     for(j=0;j<tot_nodes
```

```
Step by Step shortest path is...
    The length=3
Step by Step shortest path is...
    The length=7
Step by Step shortest path is...
    The length=0
```

10.6 Huffman Tree

The Huffman trees are constructed for encoding a given text of n characters. While encoding a given text, each character is associated with some bit sequence. Such a bit sequence is called **code word**.

The encoding can be of two types.



Fixed length encoding is a kind of encoding in which each character is associated with a bit string of some fixed length.

Variable length encoding is a kind of encoding in which each character is associated with a codeword of different length.

The Huffman tree can be well understood with the help of some example.

Consider five character alphabets {A, B, C, D, E} with following occurrence probabilities.

Character	A	B	C	D	E
Probability	0.40	0.1	0.25	0.2	0.15

Now first arrange the characters in ascending order of their probabilities.

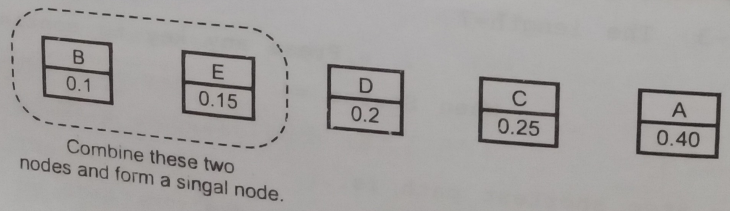


Fig. 10.13

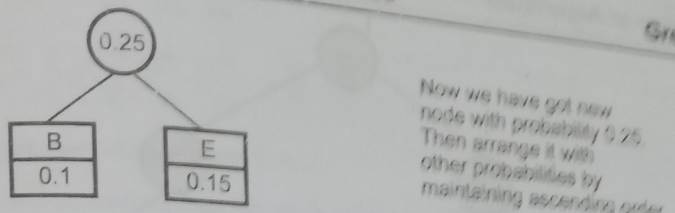


Fig. 10.14

Again we arrange probabilities with ascending order.

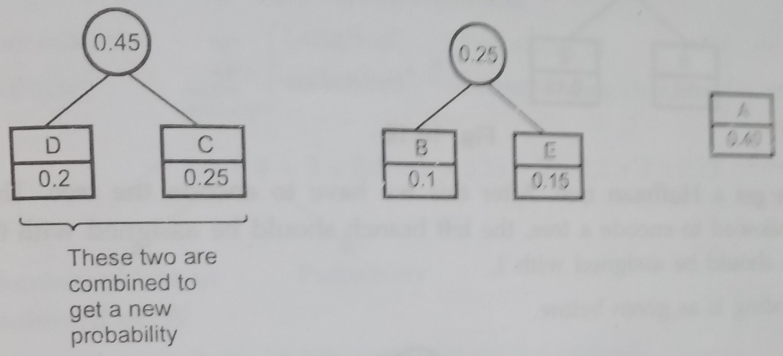


Fig. 10.15

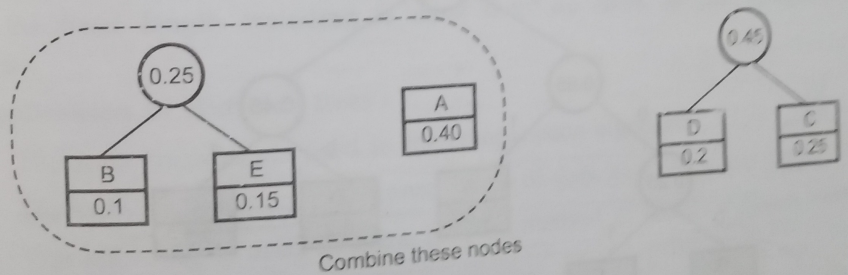


Fig. 10.16

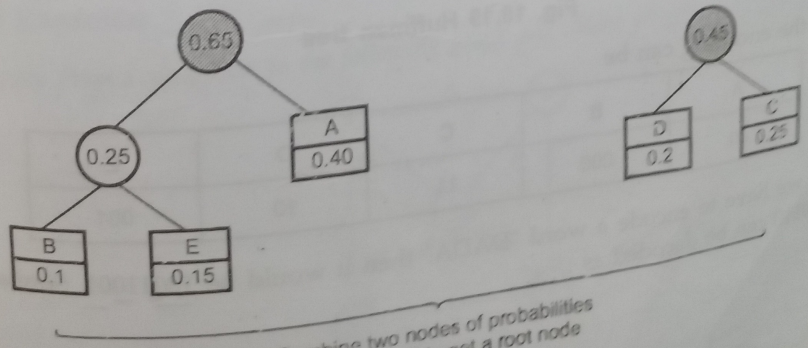


Fig. 10.17

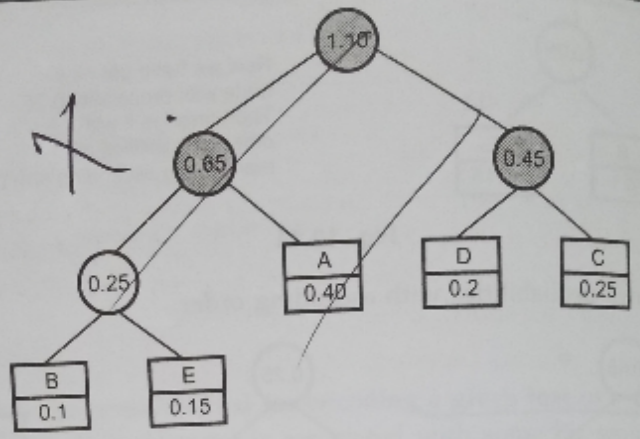


Fig. 10.18

Thus we get a Huffman tree. After this we have to encode the tree. The rule to be followed to encode a tree, the left branch should be assigned with 0 and right branch should be assigned with 1.

The encoding is as given below.

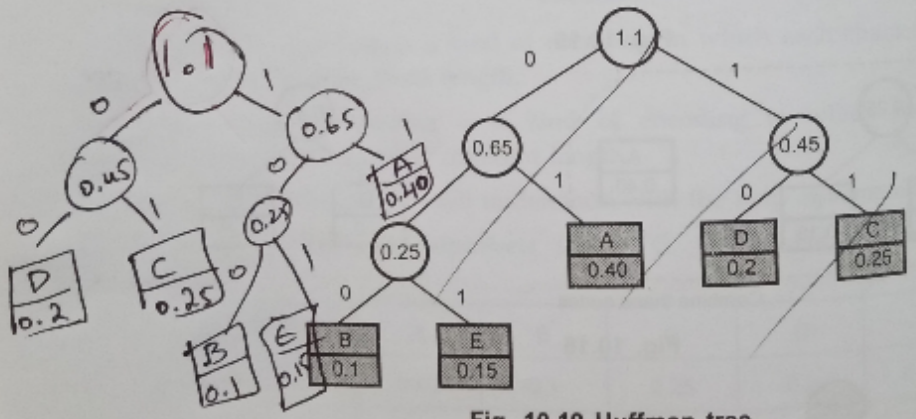


Fig. 10.19 Huffman tree

Thus the encoding can be

A	B	C	D	E
1	00	01	00	01

Now we have to encode a word "DADA" then it would be ~~10011001~~. Similarly word 110110 can be decoded as :

0 0 11 0 0 11

11 01 10
↓ ↓ ↓
C A D

Hence we get decoded string CAD.

The codeword associated with each character is called prefix code. And the tree in Fig. 10.19 is called Huffman code. Now the basic question that arises here is "What should be the number of bits per character in a given code ?" We can obtain the answer for this question by applying some formula.

$$\text{Number of bits per character} = \sum_{\text{All the characters}} \left(\text{Length of codeword} \times \text{Probability of corresponding character} \right)$$

$$= 2 \times 0.4 + 3 \times 0.1 + 2 \times 0.25 + 2 \times 0.2 + 3 \times 0.15$$

Number of bits in codeward of 'A' — Probability

$$= 2.45 \approx 3 \text{ bits per character are allowed.}$$

If the fixed length encoding is used then we have to use atleast 3 bits per character.

Applications of Huffman trees :

1. Huffman encoding is used in file compression algorithm.
2. Huffman's code is used in transmission of data in an encoded form.
3. This encoding is used in game playing method in which decision trees need to be formed

Solved Exercise

Q.1 : *Apply Prim's algorithm to the following graph and obtain minimum spanning tree.*

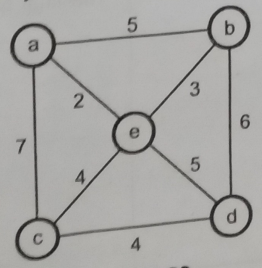
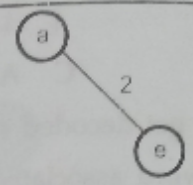
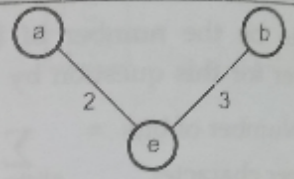
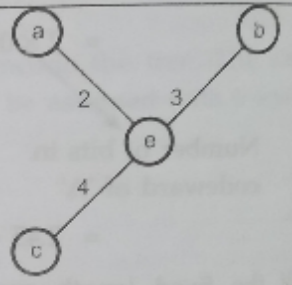
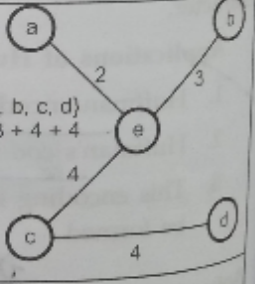
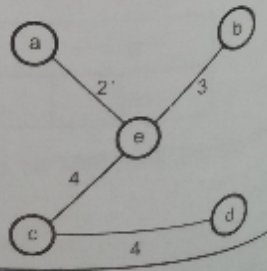


Fig. 10.20

Ans. :

We will first select a minimum distance edge from given graph.	**Step 1 :**  V = {a,e} Cost = 2
The next minimum distance edge is b-e. This edge is adjacent to previously selected vertex e.	**Step 2 :**  V = {a,e,b} Cost = 2 + 3 = 5
The next minimum distance edge is c-e which is adjacent to already selected edge b-e.	**Step 3 :**  V = {a,e,b,c} Cost = 2 + 3 + 4 = 9
Then next select an edge c-d which is with minimum distance and it is adjacent to already selected edge c-e.	**Step 4 :**  V = {a, e. b, c, d} Cost = 2 + 3 + 4 + 4 = 13
Since all the vertices are visited and we get a connected tree as minimum spanning tree.	**Step 5 :** V{a, e ,b, c, d} and Cost = 2 + 3 + 4 + 4 = 13. The final minimum spanning tree will be 

Q.2 : *Apply Prim's algorithm to the following graph.*

Obtain MST.

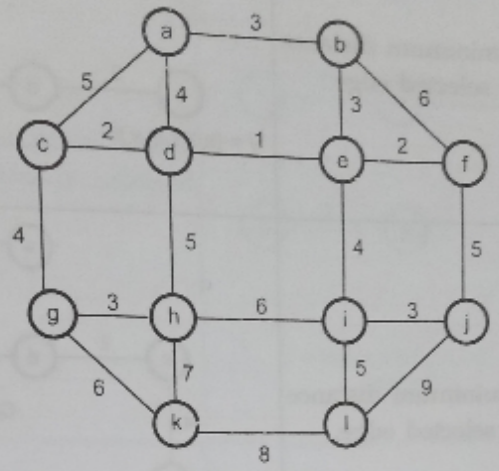
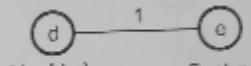
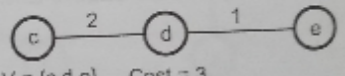
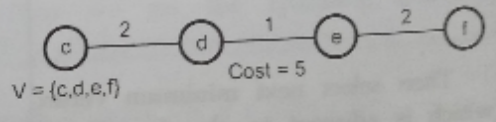
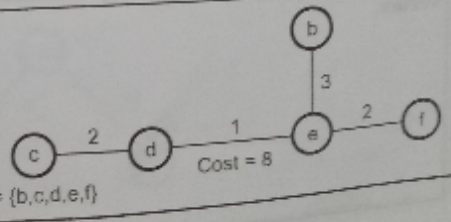
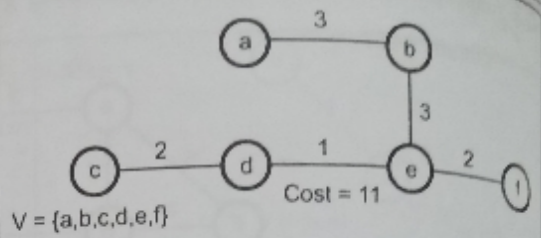


Fig. 10.21

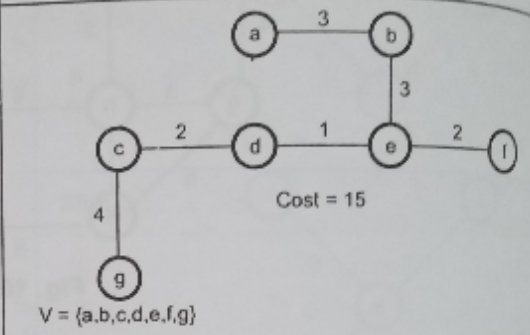
Ans. :

We first select a minimum distance edge from given graph.	
Then select next minimum distance edge with previously selected edge.	
Then select next minimum distance edge with previously selected edge.	
Then select next minimum distance edge with previously selected edge.	

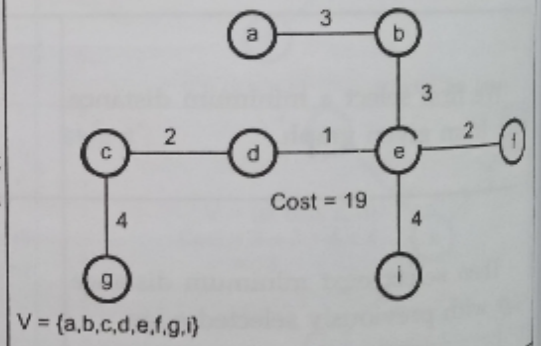
Then select next minimum distance edge with previously selected edge.



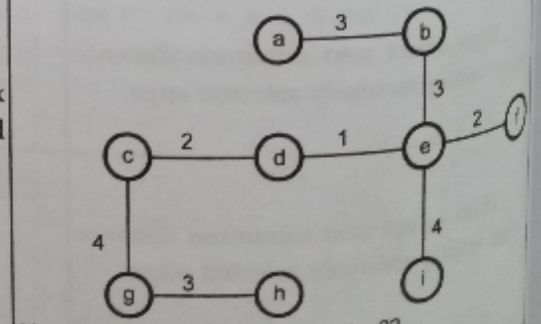
Then select next minimum distance edge with previously selected edge.



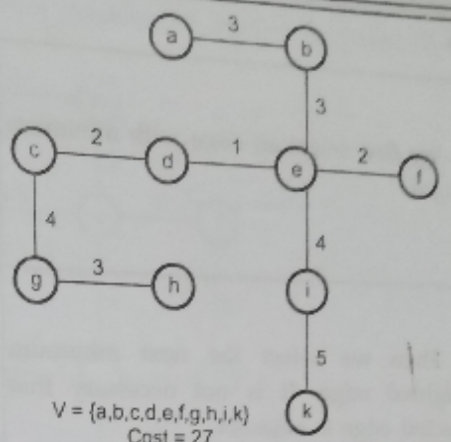
Then select next minimum vertex which is adjacent to already selected vertex.



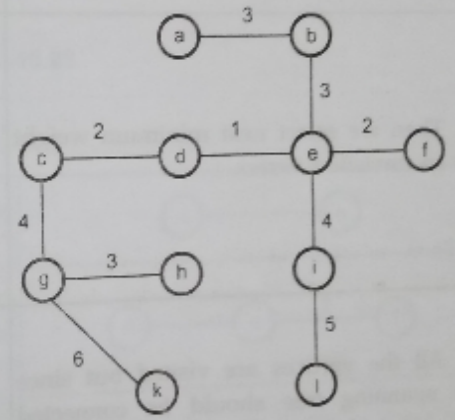
Then select next minimum vertex which is adjacent to already selected vertex.



Then select next minimum vertex which is adjacent to already selected vertex.



Then select next minimum vertex which is adjacent to already selected vertex.



Thus we get the spanning tree by visiting all the vertices and cost = 33.

Q.3 : *Apply Kruskal's algorithm to find a minimum spanning tree of a given graph.*

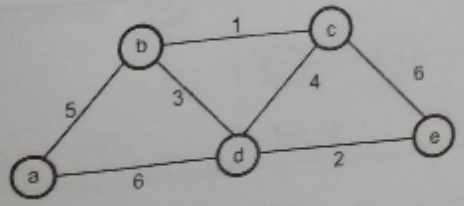
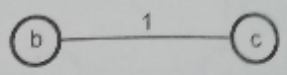
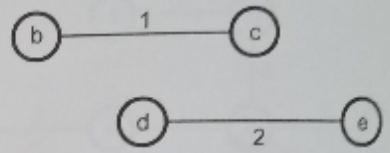
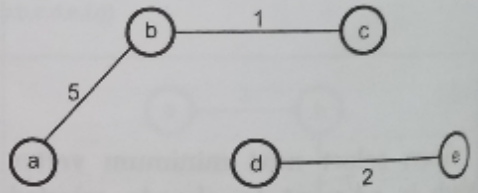
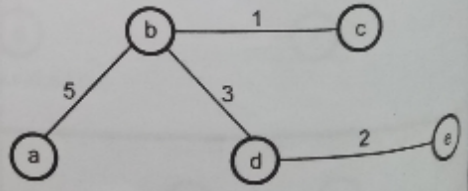


Fig. 10.22

Ans. :

We first select an edge with minimum weight.	 Total cost = 1
Then we select the next minimum weighted edge. It is not necessary that selected edge is adjacent.	 Total cost = 3
Then we select next minimum weight for an unvisited vertex.	 Total cost = 8
All the vertices are visited but since the spanning tree should be connected one. Hence we select an edge with minimum weight. Thus we get a minimum spanning tree with Kruskal's algorithm.	 total cost = 11

Q.4 : *Apply Kruskal's algorithm to find minimum spanning tree of following graph.*

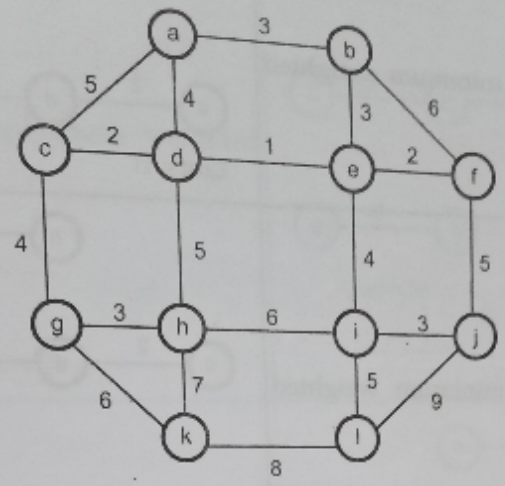
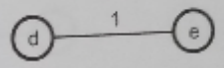
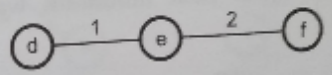
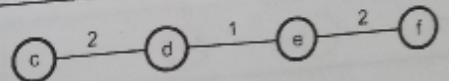
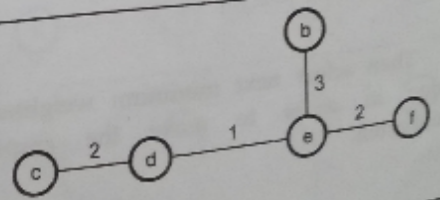
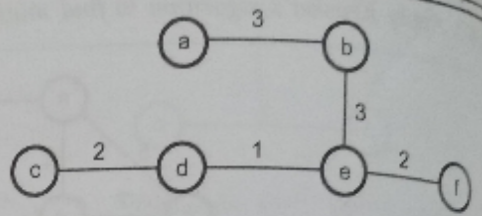


Fig. 10.23

Ans. :

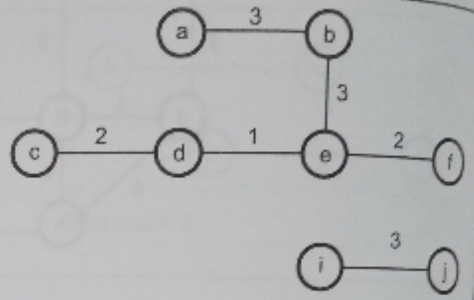
We first select an edge with minimum weight.	
Then we select the next minimum weighted edge. It is not necessary that selected edge is adjacent.	
Then select next minimum weighted edge.	 Cost = 5
Then select next minimum weighted edge.	 Cost = 8

Then select next minimum weighted edge.



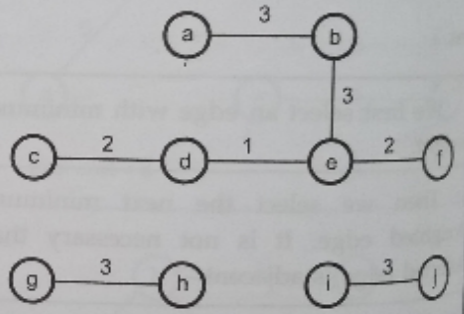
Cost = 11

Then select next minimum weighted edge.



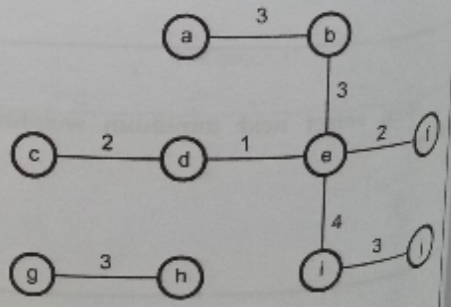
Cost = 14

Then select next minimum weighted edge.



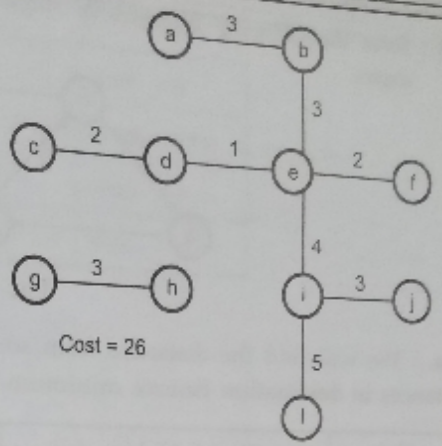
Cost = 17

Then select next minimum weighted edge in order to make the graph connected.

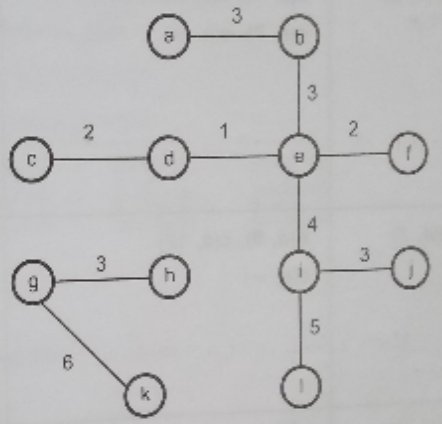


Cost = 21

Then select next minimum weighted edge.



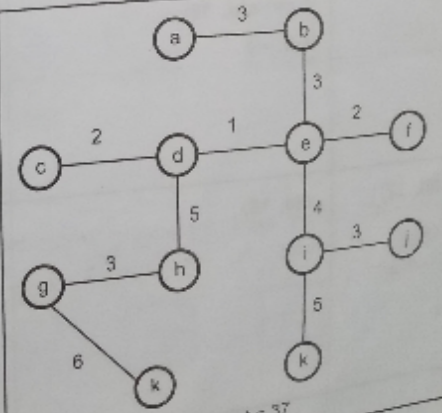
Then select next minimum weighted edge.



In order to make graph connected, select the minimum weighted edge.

Thus we get a spanning tree.

Total cost = **37**



Q.5 : *Solve the following instances of single-source shortest path problem with vertex a as source.*

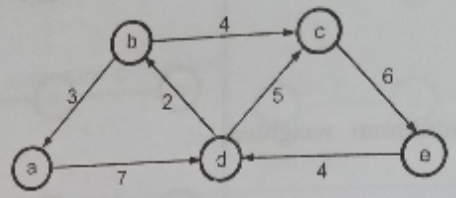
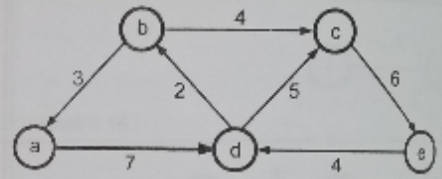
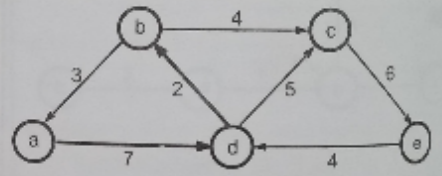
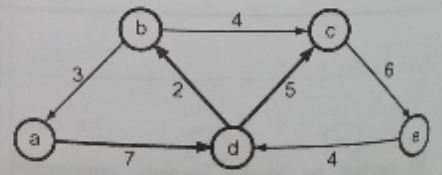
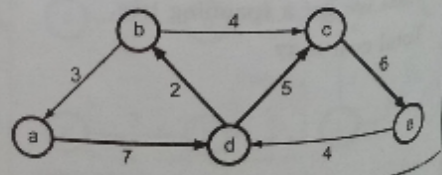


Fig. 10.24

Ans. : We will find the distances from source a to every other vertices. We write the distances in **destination (source minimum distance)** form.

a (–, 0)	b(a, ∞), c(a, ∞) **d(a, 7)**, e(a, ∞)	
d(a, 7)	**b(d, 9)**, c(d, 12) e(d, ∞)	
b(d, 9)	min[c(b, 13), c(d, 12)] = **c(d, 12)** e(b, ∞)	
c(d, 12)	e(d, 18)	

Thus from source a to all other remaining vertices the shortest paths are-

a to b	(a-d-b)	Path = 9
a to c	(a-d-c)	Path = 12
a to d	(a-d)	Path = 7
a to e	(a-d-c-e)	Path = 18

Q.6 : *Construct a Huffman code for the following data.*

Character	A	B	C	D	–
Probability	0.4	0.1	0.2	0.15	0.15

a) Encode the text ABACABAD using generated code.

b) Decode the text whose encoding is 100010111001010

Ans. : Let us arrange the probabilities in ascending order.

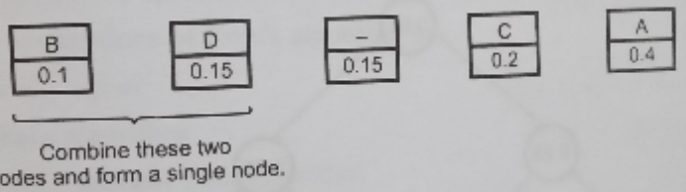


Fig. 10.25

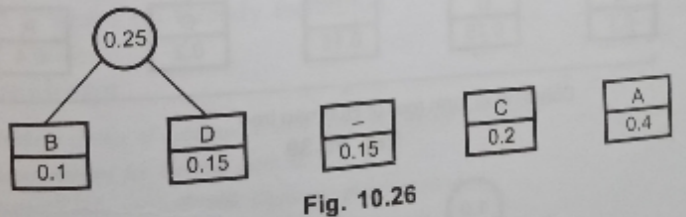


Fig. 10.26

Rearrange the nodes in ascending order.

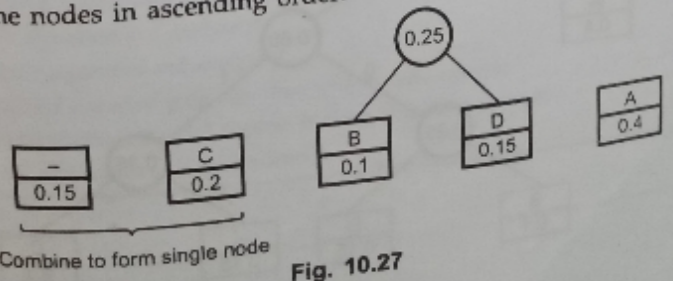
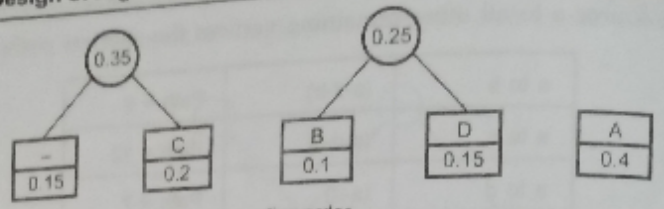
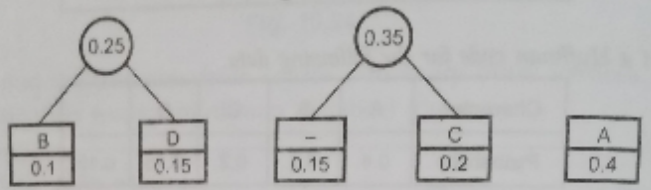


Fig. 10.27



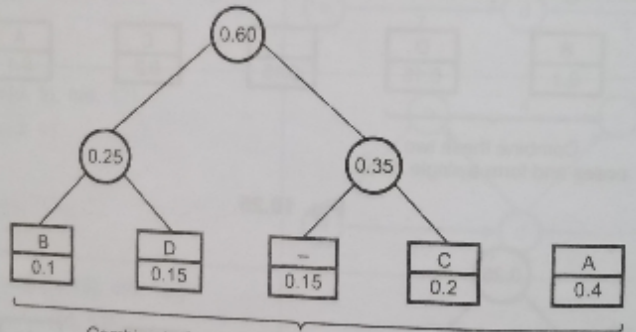
Rearrange the nodes in ascending order

Fig. 10.28



Combine two subtrees

Fig. 10.29



Combine to form root of Huffman tree

Fig. 10.30

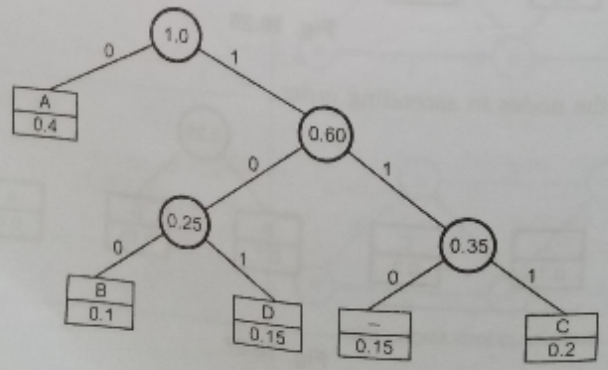


Fig. 10.31

The encoding for each character will be

A = 0

B = 100

C = 111

D = 101

– = 110

∴ ABACABAD = 0100011101000101

The decoding of : $\underbrace{100}_{B}\ \underbrace{0}_{A}\ \underbrace{101}_{D}\ \underbrace{110}_{-}\ \underbrace{0}_{A}\ \underbrace{101}_{D}\ \underbrace{0}_{A}$

Hence text will be BAD-ADA.

Summary

- In Greedy method all the feasible solutions are obtained and from these feasible solutions, optimal solution is selected as a solution to the problem.
- Typical applications of Prim's algorithm are -

  i) Prim's algorithm

  ii) Kruskal's algorithm

  iii) Dijkstra's shortest path algorithm
- Huffman's tree is constructed by which Huffman's code can be obtained. This method is based on Greedy technique.

Review Questions

1. *How to make a choice of solution at each step in Greedy approach?*
2. *Give general method for Greedy method.*
3. *Differentiate Greedy method with Dynamic programming.*
4. *Explain Prim's algorithm with some suitable example.*
5. *Explain Kruskal's algorithm with some suitable example.*
6. *Give Prim's algorithm and analyse it.*
7. *Give Kruskal's algorithm and analyse it.*
8. *Explain Dijkstra's shortest path algorithm with suitable example.*
9. *How to obtain Huffman's code? Explain it along with some example.*
10. *What are the application of Huffman trees?*

❑❑❑