Unit - V

5 Graph and Symbol Table

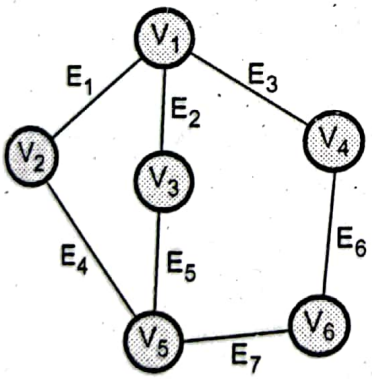
5.1 : Concept and Terminologies

Q.1 What is Graph ?

Ans. : A graph is a collection of two sets V and E where V is a finite non-empty set of vertices and E is a finite non-empty set of edges.

- **Vertices** are nothing but the nodes in the graph.
- Two adjacent vertices are joined by **edges.**
- Any graph is denoted as G = {V, E}.

For example :



$$G = \{\{V_1, V_2, V_3, V_4, V_5, V_6\}, \{E_1, E_2, E_3, E_4, E_5, E_6, E_7\}\}$$

Fig. Q.1.1 Graph G

Q.2 Define i) Complete Graph ii) Path ☞ **[SPPU : May-13, Marks 4]**

Ans. : i) Complete graph : If an undirected graph of n vertices consists of $\frac{n(n-1)}{2}$ number of edges then that graph is called a complete graph.

For example : The graph shown in Fig Q.2.1 is a complete graph.

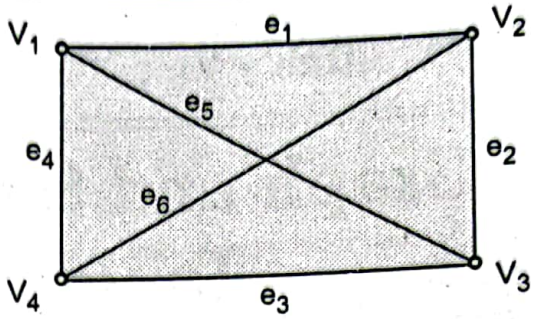


Fig. Q.2.1 Complete graph

ii) Path : A path is denoted using sequence of vertices and there exists an edge from one vertex to the next vertex.

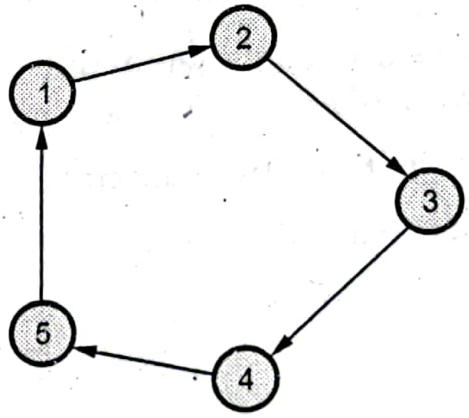


Fig. Q.2.2 Path of G is 1-2-3-4-5

Q.3 What is indegree and outdegree ?

Ans. : The degree of vertex is the number of edges associated with the vertex.

In-degree of a vertex is the number of edges that incident to that vertex. Out-degree of the vertex is total number of edges that are going away from the vertex.

Vertices	In-degree	Out-degree
V_1	1	1
V_2	2	1
V_3	1	1
V_4	1	2

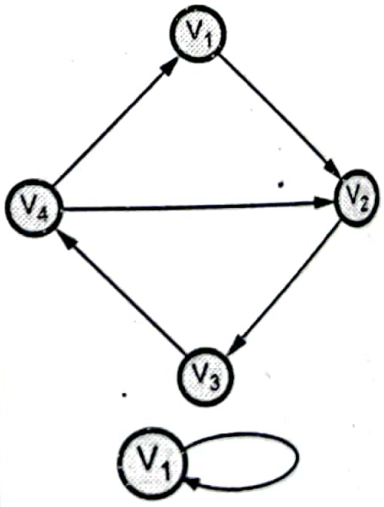


Fig. Q.3.1 Self loop

5.2 : Graph as ADT

Q.4 Give an ADT for graph.

Ans. : The abstract data type for graph is as given below.

AbstractDataType Graph

{

Instances

Graph is a collection of vertices and edges.

Operations

1) tot_vertices ( ) : This function returns total number of vertices.
2) tot_edges ( ) : This function returns total number of edges.
3) Is_edge (V_1, V_2) : This function returns true if there exists an edge between V_1 and V_2.
4) In_degree ( ) : This function returns total number of edges that are incident to the vertex.
5) Out_degree ( ) : This function returns out degree of the vertex.
6) Display ( ) : This function displays the graph in DFS or in BFS manner.

}

Q.5 What is the difference between trees and graphs ?

Ans. :

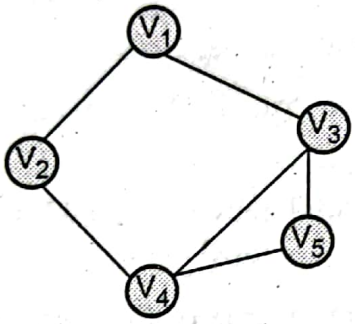
Sr. No.	Graph	Tree
1.	Graph is a non-linear data structure.	Tree is a non-linear data structure.
2.	It is a collection of vertices/nodes and edges.	It is a collection of nodes and edges.
3.	Each node can have any number of edges.	General trees consist of the nodes having any number of child nodes. But in case of binary trees every node can have at the most two child nodes.

4.	There is no unique node called root in graph.	There is a unique node called root in trees.
5.	A cycle can be formed.	There will not be any cycle.
6.	Applications : For finding shortest path in networking graph is used.	Applications : For game trees, decision trees, the tree is used.

5.3 : Representation of Graphs

Q.6 Describe various ways in which graph is represented.

☞ **[SPPU : May-10, Marks 6, May 13, Marks 4]**



	1	2	3	4	5
1	0	1	1	0	0
2	1	0	0	1	0
3	1	0	0	1	1
4	0	1	1	0	1
5	0	0	1	1	0

Fig. Q.6.1 An adjacency matrix representation

Ans. : 1) Adjacency matrix : In this representation, matrix or 2 dimensional array is used to represent the graph.

Consider a graph G of n vertices and the matrix M. If there is an edge present between vertices V_i and V_j then M[i][j] = 1 else M[i][j] = 0. Note that for an undirected graph if M[i][j] = 1 then for M[j][i] is also 1. Here are some graphs shown by adjacency matrix.

2) Adjacency list : In this representation, a **linked list** is used to represent a graph.

There are two methods of representing graph using adjacency list.

For example : Now in Fig. Q.6.2, graph has the nodes as a, b, c, d, e. So we will maintain the linked list of these head nodes as well as the adjacent nodes.

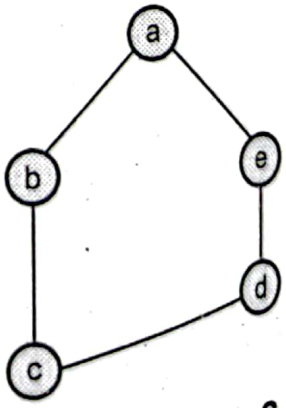


Fig. Q.6.2 Graph G

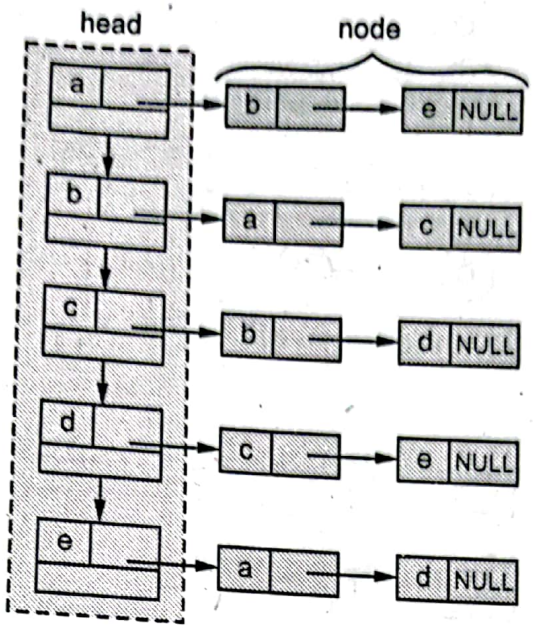


Fig. Q.6.3 Adjacency list

The down pointer helps us to go to each node in the graph where as the next node is for going to adjacent node of each of the head node.

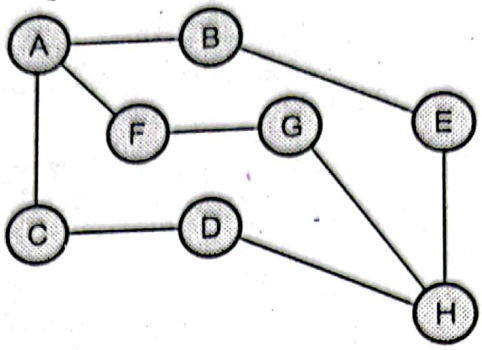
Q.7 Define a graph. For the given adjacency matrix draw the graph and its adjacency list :

	A	B	C	D	E	F	G	H
A	0	1	1	0	0	1	0	0
B	1	0	0	0	1	0	0	0
C	1	0	0	1	0	0	0	0
D	0	0	1	0	0	0	0	1
E	0	1	0	0	0	0	0	1
F	1	0	0	0	0	0	1	0
G	0	0	0	0	0	1	0	1
H	0	0	0	1	1	0	1	0

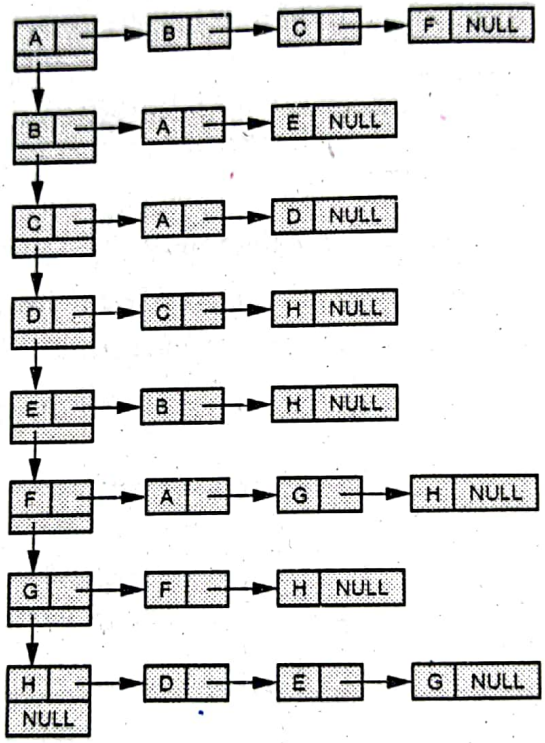
Find all the nodes adjacent to node A, node F and node G.

☞ **[SPPU : May-15, Marks 8]**

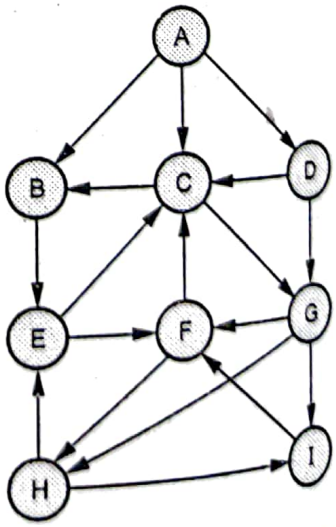
Ans. : Graph : Refer Q.1



Adjacency list :



Q.8 Consider the graph G given in Fig. below. Draw the adjacency list of G is also given. Assume that G represents the daily flights between different cities and we want to fly from city A to H with minimum stops. Find the minimum path P from A to H given that every edge has length of 1.



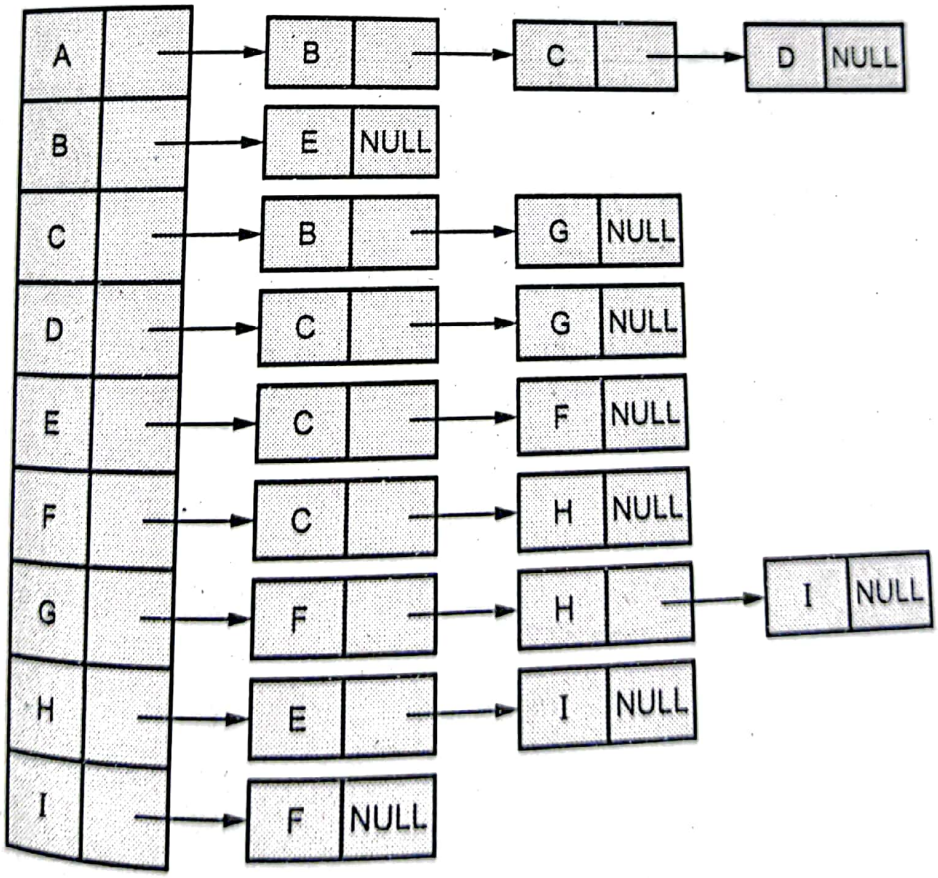
☞ **[SPPU : Dec.-15, Marks 6]**

Ans. : Adjacency List

The paths form A to H by considering every edge has lenght of 1 are listed as below :

Path	Total Length
A – B – E – F – H	4
A – B – E – F – C – G – H	6
A – C – G – H	3
A – C – G – F – H	4
A – D – C – G – H	4
A – D – G – H	3

The paths with minimum stops are (i) A– C – G – H (ii) A – D – G – H with path length 3.



Q.9 Draw the directed graph for the adjacency matrix representation given below :

	A	B	C	D	E	F
A	0	3	4	0	2	1
B	0	0	2	0	0	3
C	0	0	0	2	6	1
D	2	6	1	0	1	2
E	0	0	0	0	0	3
F	0	0	0	0	0	0

☞ **[SPPU : May-19, Marks 6]**

Ans. : Adjacency List

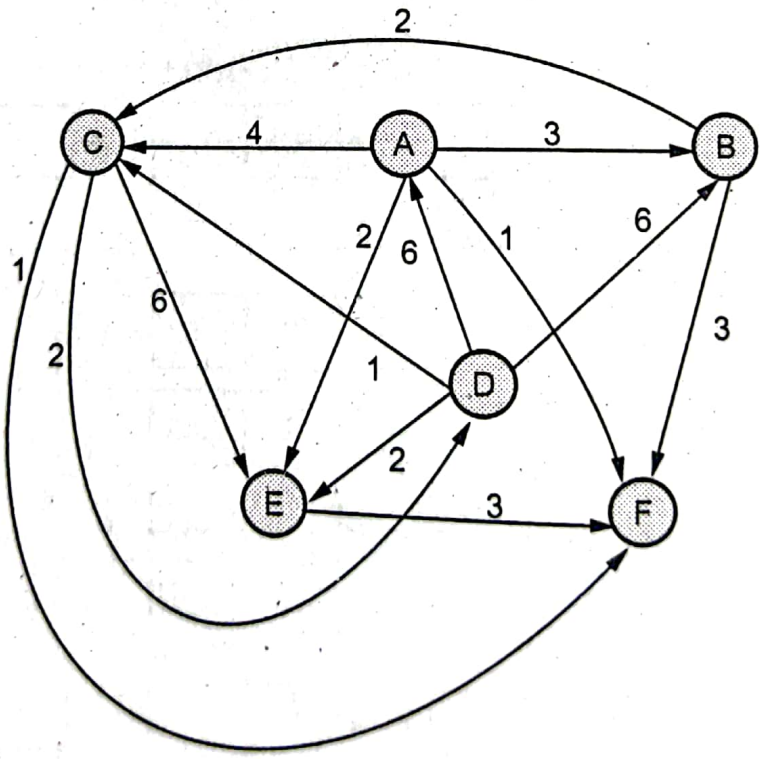


Fig. Q.9.1

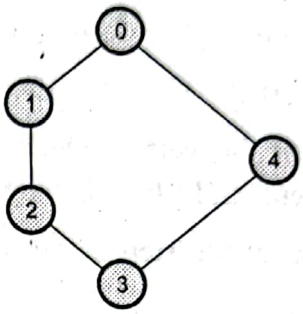
5.4 : Breadth First Search and Depth First Search Traversals

Q.10 What is Depth First Search Traversal ?

Ans. : • In depth first search traversal we start from one vertex and traverse the path as deeply as we can go. When there is no vertex further, we traverse back and search for unvisited vertex.

• An array is maintained for storing the visited vertex.

For example



- The DFS will be (if we start from vertex 0) 0 - 1 - 2 - 3 - 4
- The DFS will be (if we start from vertex 3) 3 - 4 - 0 - 1 - 2

Q.11 Write a recursive C/C++ code for depth first traversal.

Ans. :

```
void Gdfs::Dfs(int v1)
{
   int v2;

   cout<<endl<<v1;
   v[v1] = TRUE;
   for ( v2 = 0; v2 < n; v2++)
      if ( g[v1][v2] == TRUE && v[v2] == FALSE )
         Dfs(v2);
}
```

Q.12 What is breadth first search traversal ?

Ans. : In BFS we start from some vertex and find all the adjacent vertices of it. This process will be repeated for all the vertices so that the vertices lying on same breadth get printed.

• For avoiding repetition of vertices, we maintain array of visited nodes.

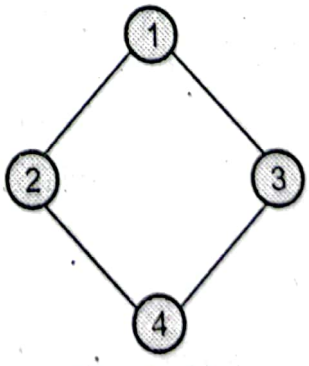


Fig. Q.12.1

- A queue data structure is used to store adjacent vertices.

The BFS traversal will be 1, 2 3, 4

Q.13 Write a C/C++ code for breadth first traversal.

Ans. :

1. Create a graph. Depending on the type of graph i.e. directed or undirected set the value of the flag as either 0 or 1 respectively.
2. Read the vertex from which you want to traverse the graph say V_i.
3. Initialize the visited array to 1 at the index of V_i.
4. Insert the visited vertex V_i in the queue.
5. Visit the vertex which is at the front of the queue. Delete it from the queue and place its adjacent nodes in the queue.
6. Repeat the step 5, till the queue is not empty.
7. Stop.

```
void Gbfs::bfs(int v1)
{
   int v2;

   visit[v1] = TRUE;
   front = rear = -1;
   Q[++rear] = v1;
   while ( front != rear )
   {
      v1 = Q[++front];
      cout<<"\n"<<v1;
      for ( v2 = 0; v2 < n; v2++)
      {
         if ( g[v1][v2] == TRUE && visit[v2] == FALSE )
         {
            Q[++rear] = v2;
            visit[v2] = TRUE;
         }
      }
   }
}
```

Q.14 Differentiate between BFS and DFS

Ans. :

Sr. No.	BFS	DFS
1.	BFS is simple to implement	DFS is complex to implement as it may suffer from infinite loop problem.
2.	BFS will perform poor is for large numbers of vertices in graph.	DFS will perform better in case of large complex graph.
3.	BFS requires more memory	DFS requires less memory
4.	BFS will find the shortest path if the weight on the links are uniform.	DFS can not obtain shSrtest path.
5.	BFS is not useful in sorting application	DFS is used in topological sorting.
6.	This algorithm works in single stage. The visited vertices are removed from the queue and then displayed at once.	This algorithm works in two stages - in the first stage the visited vertices are pushed onto the stack and later on when there is no vertex further to visit those are popped-off.
7.	If solution path is long, the whole tree must be searched up to that depth.	May settle for non-optimal solution.

Q.15 For given graph draw the adjacency list / matrix and perform BFS or DFS.

☞ **[SPPU : May-14, Marks 6]**

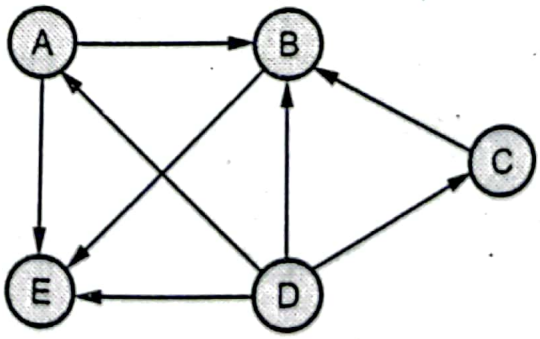
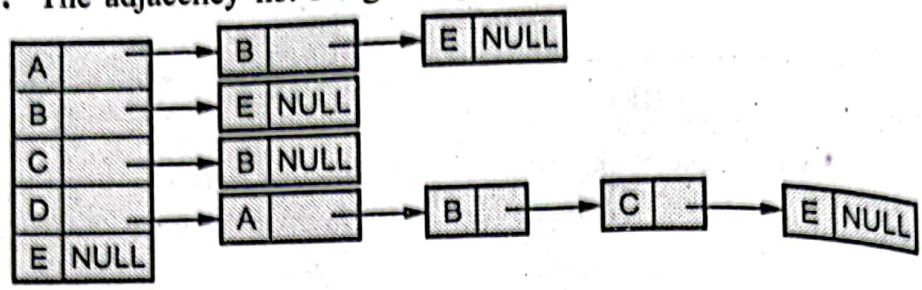


Fig. Q.15.1

Ans. : The adjacency list for given graph is

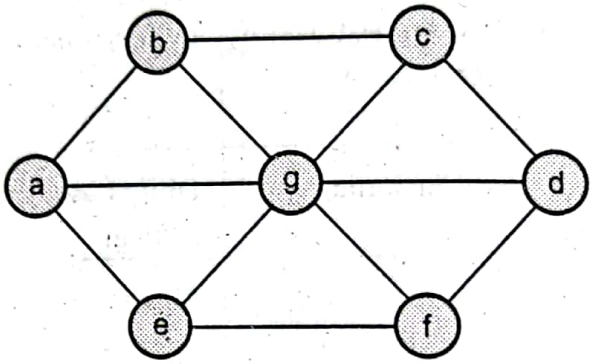


BFS sequence : D, A, B, C, E

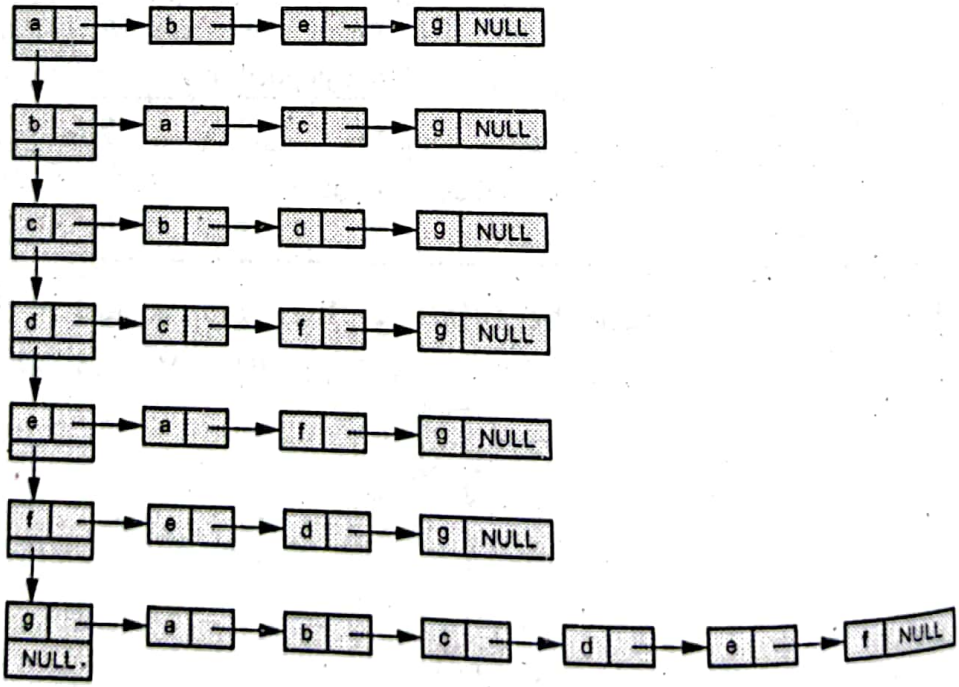
DFS sequence : D, A, B, E, C

Q.16 For the graph given below, find BFS and DFS stepwise

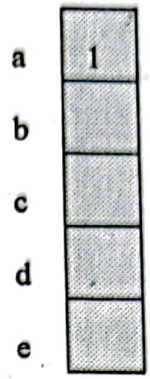
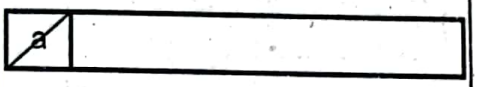
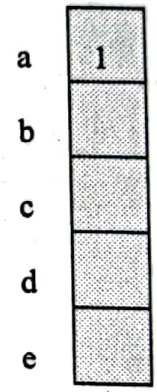
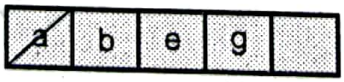
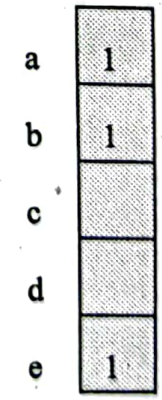
☞ **[SPPU : Dec.-14, Marks 6]**

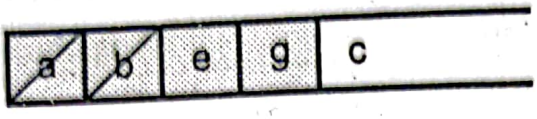
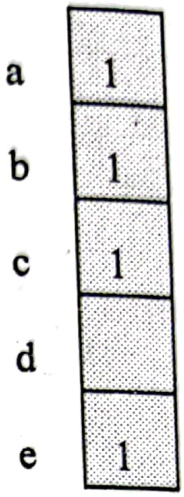
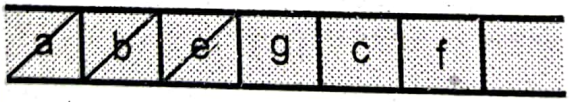
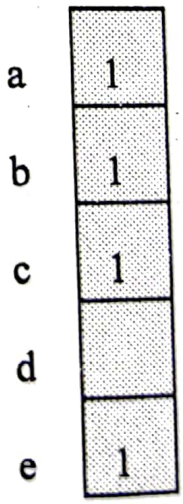


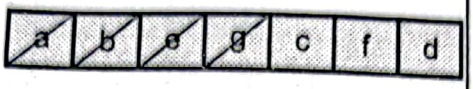
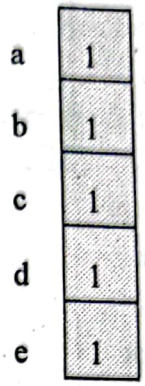
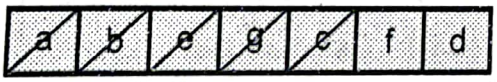
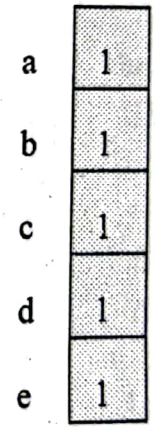
Ans. : We will create adjacency list for given graph.

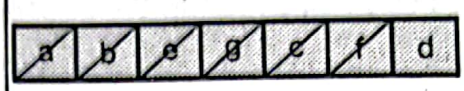
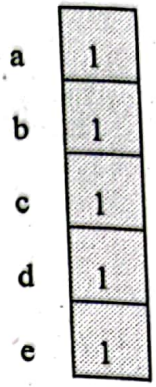
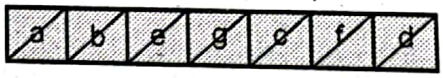
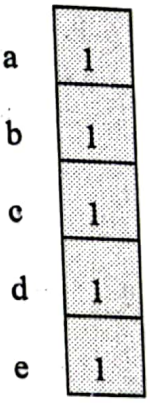


Bredth First Search (BFS)

Action	Queue	Visited array	Output
Step 1 : Insert a in queue. Mark visited (a) = 1			
Step 2 : Delete a and print			a
Step 3 : Find adjacent of a and insert them in queue. Mark corresponding adjacent nodes as visited.			a

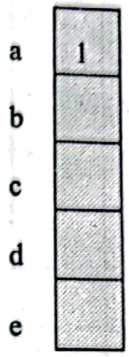
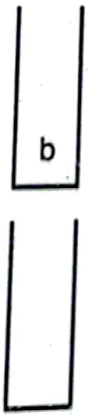
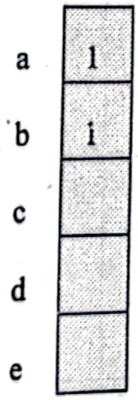
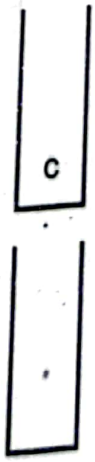
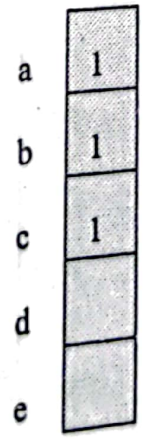
Step 4 : Delete b and print. Find adjacent of b and if those nodes are not visited then insert them in the queue. Mark corresponding nodes as visited.			a, b
Step 5 : Delete e and print. Find adjacent of e and if those nodes are not visited then insert them in queue. Mark corresponding nodes as visited.			a, b, e

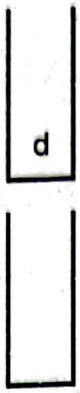
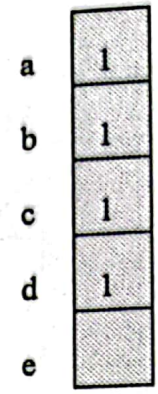
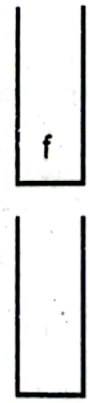
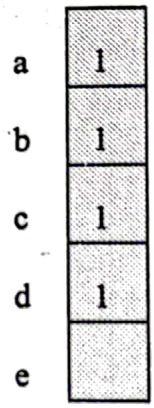
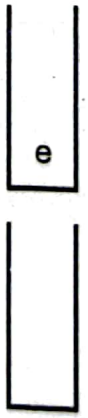
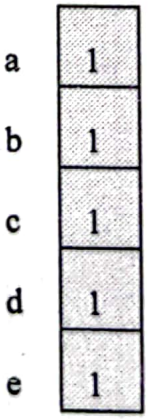
Step 6 : Delete g and print. Find adjacent of g and if those nodes are not visited then insert them in queue. Mark corresponding nodes as visited.			a, b, e, g
Step 7 : Delete c and print. Find adjacent of c and if those nodes are not visited then insert them in queue. Mark corresponding nodes as visited.			a, b, e, g, c

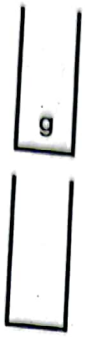
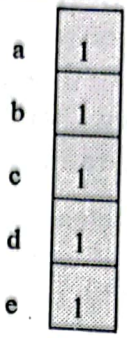
Step 8 : Delete f and print. Find adjacent of f and if those nodes are not visited then insert them in queue. Mark corresponding nodes as visited.			a, b, e, g, c, f
Step 9 : Delete d and print. Find adjacent of d and if those nodes are not visited then insert them in queue. Mark corresponding nodes as visited.			a, b, e, g, c, f, d
Step 10 : As queue is empty and all entries in the visited array are visited them stop. At display we will get BFS sequence.			

Hence BFS sequence is **a, b, e, g, c, f, d.**

Depth first search

Action	Stack	Visited array	Output
Step 1 : Start with vertex a, mark it visited and print it. Then			a
Step 2 : Find adjacent of a i.e. b, and mark it visited. Push it onto the stack. Later b will be popped and printed.			a, b
Step 3 : Find adjacent b. The a is already visited, hence ignore. Next node will be c. Insert c onto the stack mark it as visited. Push it. Later c will be popped and printed.			a, b, c

Step 4 : Find adjacent of c. Push d, mark it visited. Later d will be popped and printed.			a, b, c, d
Step 5 : Find adjacent of d. Push f, mark it as visited. Later f will be popped and printed.			a, b, c, d, f
Step 6 : Find adjacent of f. Push e, mark it as visited. Later e will be popped and printed.			a, b, c, d, f, e

Step 7 : Find adjacent of e. Push g, mark it as visited. Later g will be popped and printed.			a, b, c, d, f, e, g
As all nodes are visited and stack is empty we will stop traversing.			

The DFS sequence is **a, b, c, d, f, e, g.**

Q.17 What are various applications of graph ?

Ans. :

1. In computer networking such as Local Area Network (LAN), Wide Area Networking (WAN), internetworking.
2. In telephone cabling graph theory is effectively used.
3. In job scheduling algorithms

5.5 : Minimum Spanning Tree

Q.18 What is spanning Tree ?

Ans. : A Spanning tree S is a subset of a tree T in which all the vertices of tree T are present but it may not contain all the edges.

Q.19 Explain the term minimum spanning tree.

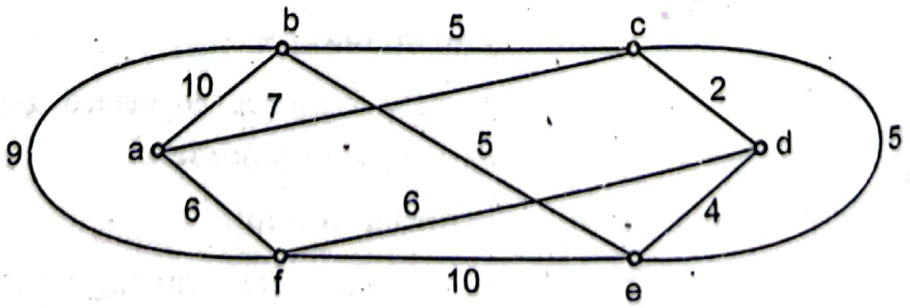
Ans. : The minimum spanning tree of a weighted connected graph G is called **minimum** spanning tree if its **weight is minimum.**

Q.20 Explain Prim's algorithm with an example.

Ans. : Prim's algorithm is to obtain the minimum spanning tree. In this algorithm the minimum edge is selected each time. The edge selection is done for adjacent vertices only.

```
Prim(G,Cost, n, tree)
{
 /* G is a graph of n nodes. cost is an array of cost of edges. tree is
an array of minimum spanning tree */
min=cost[p,q] //represents minimum cost edge
tree[1][1]=p;
tree[1][2]=q;
for(i=1; i<n; i++)
{
   if(cost[i][q] < cost[i][p])
      optimum[i]=p;
   else
   optimum[i]=q // storing minimum cost
   optimum[p]=optimum[q]=0;
   for(i=2; i<n-1; i++)
{//store vertex j at optimum distance and neighbouring of j in 'tree'
      tree[i][1] = j;
      tree[i][2] = optimum[j];
      min = min+cost[j][optimum[j]];
   optimum[j]=0;
   for(k=1 to n)
   if((optimum[k]!=0)AND (cost[k,optimum[k]]>cost[k,j]))then
      optimum[k]=j;
   }
      return min;
   }
```

For example : Consider the graph as given below -



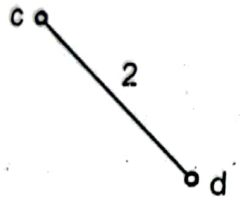
We will find the minimum spanning tree using Prim's algorithm using following rules -

1. Select the edge with minimum edge. The corresponding vertices will form the set of vertices.
2. Each time select an edge with minimum weight and the edge to be selected from the graph should be such that at least one of the vertex of the selected edge is adjacent to already selected vertices.
3. In selection of edges the circuit should not be formed.

From above graph,

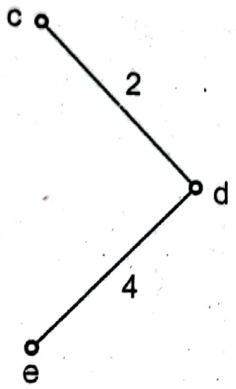
First select edge c-d with minimum weight 2

Step 1 :



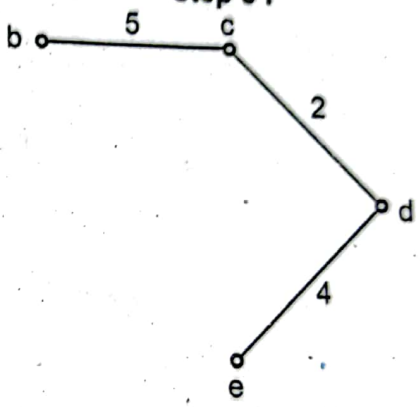
weight = 2

Step 2 :



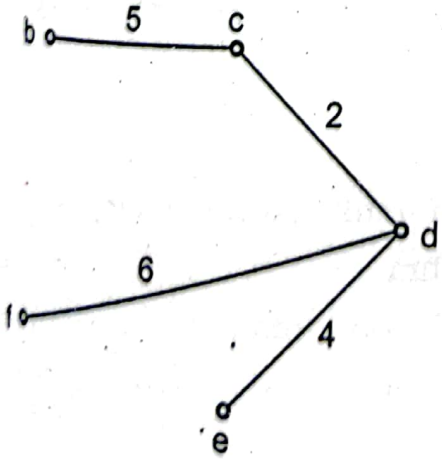
weight = 6

Step 3 :



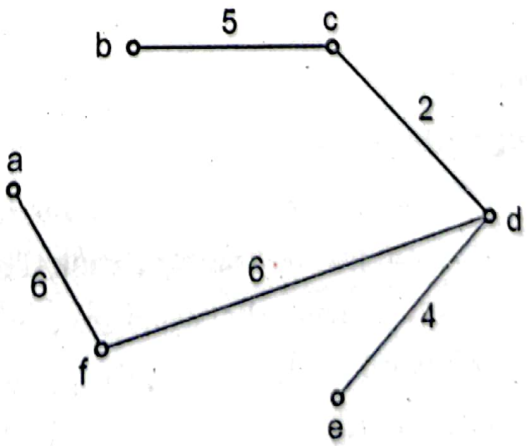
weight = 11

Step 4



weight = 16.

Step 5

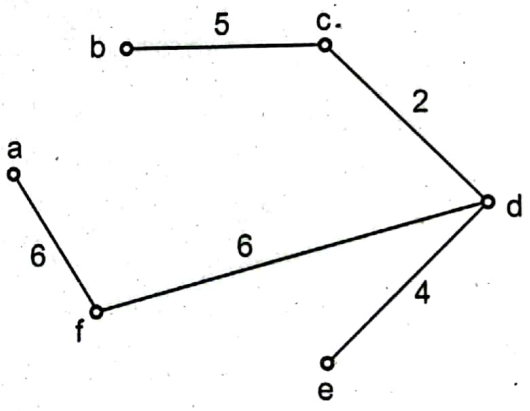


weight = 23 - is minimum spanning tree

The edge selection can be summarised as below -

Edge	Set
<c-d>	{c, d}
<d-e>	{c, d, e}
<b-c>	{b, c, d, e}
<d-f>	{b, c, d, e, f}
<a-f>	{a, b, c, d, e, f}

Thus the minimum spanning tree will be -



Minimum spanning tree with weight 23

Q.21 Write Pseudo code for Kruskal's algorithm.

☞ **[SPPU : May-17, Marks 6]**

Ans. :

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 of
//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.

Q.22 Find the minimum spanning tree using Prim's and Kruskal's method for the following graph : ☞ **[SPPU : May-15, Marks 4]**

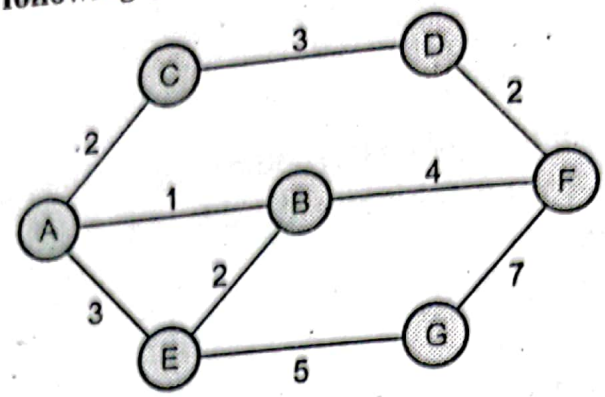
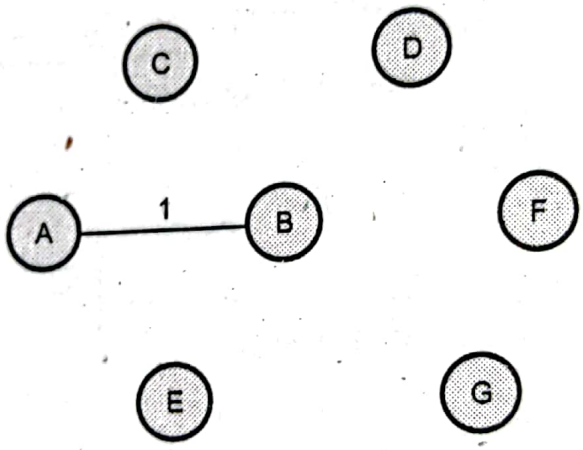


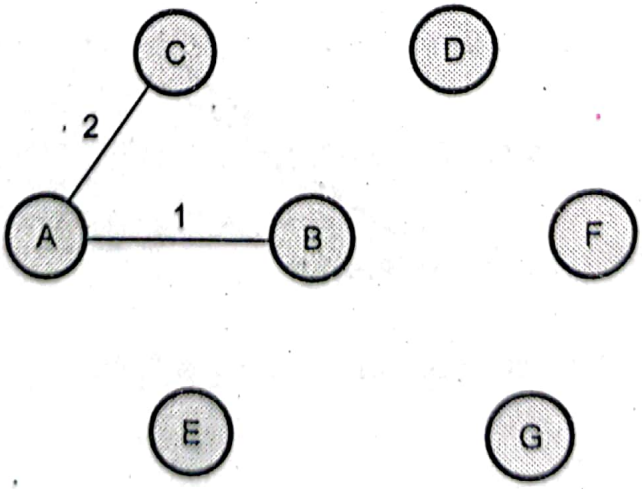
Fig. Q.22.1

Ans. :

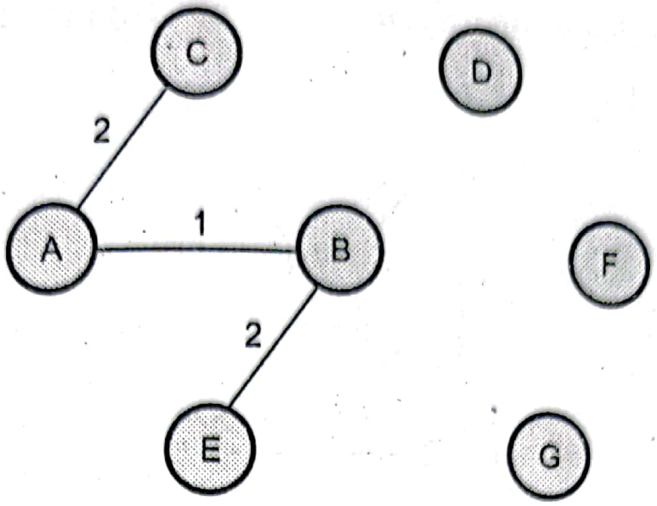
Step 1 :



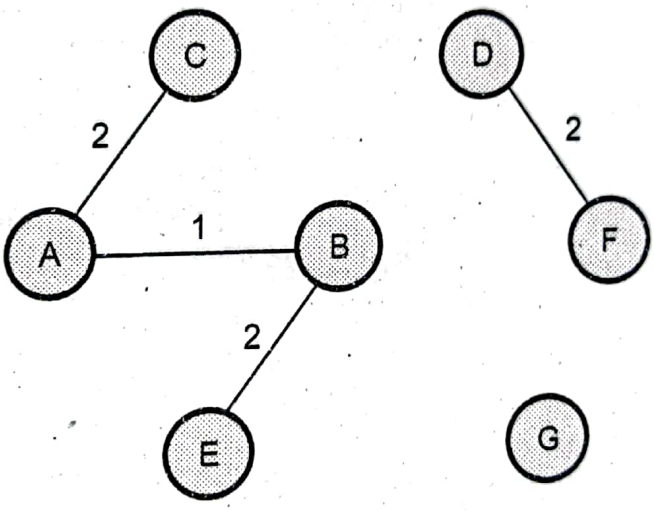
Step 2 :



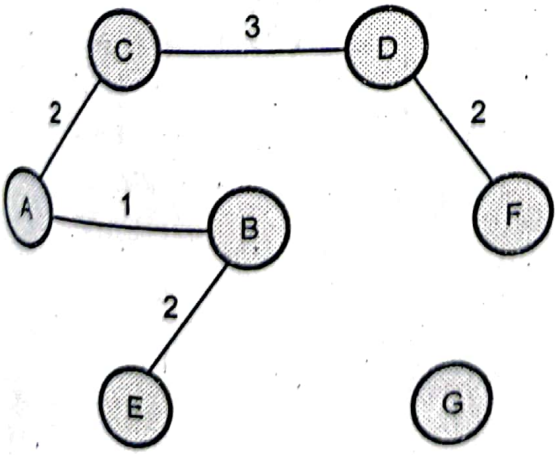
Step 3 :



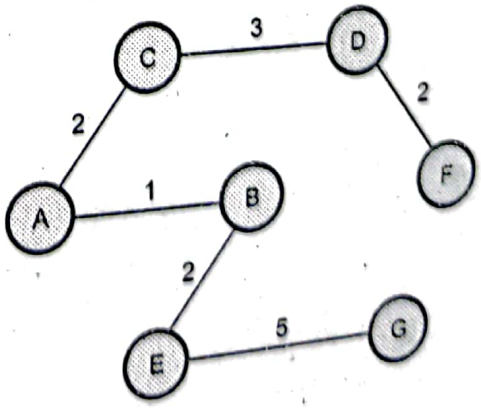
Step 4 :



Step 5 :



Step 6 :



Total path length = 15

Q.23 For the graph given below, show stepwise representation of MST using Kruskal's algorithm. ☞ **[SPPU : May-16, Marks 4]**

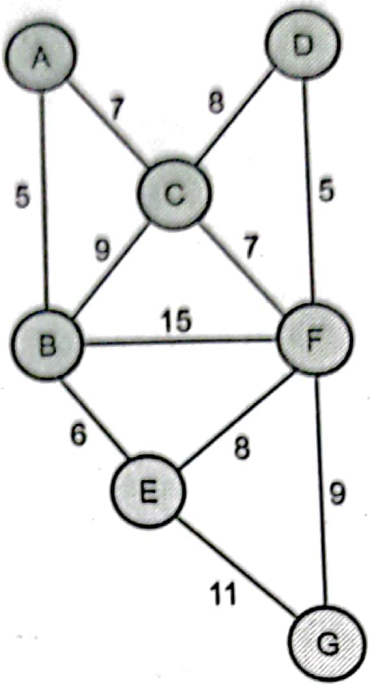
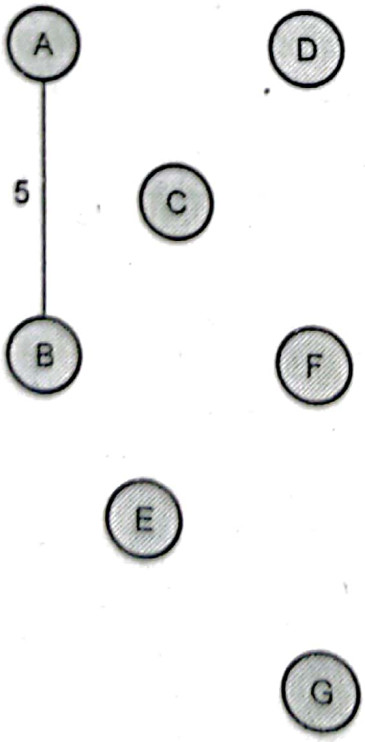


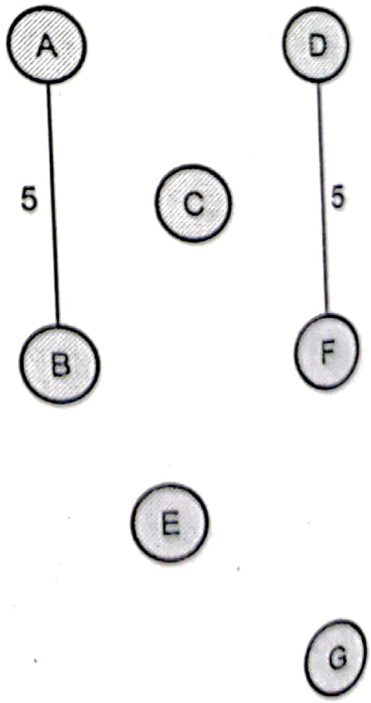
Fig. Q.23.1

Ans.

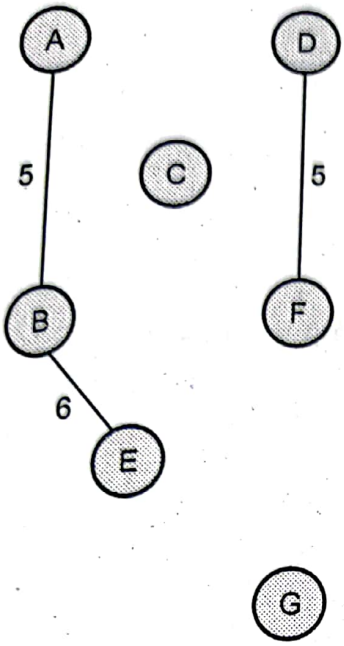
Step 1 :



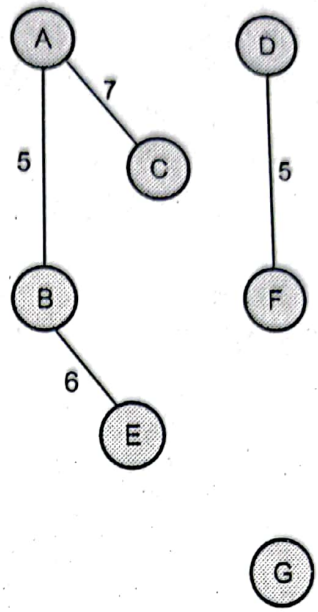
Step 2 :



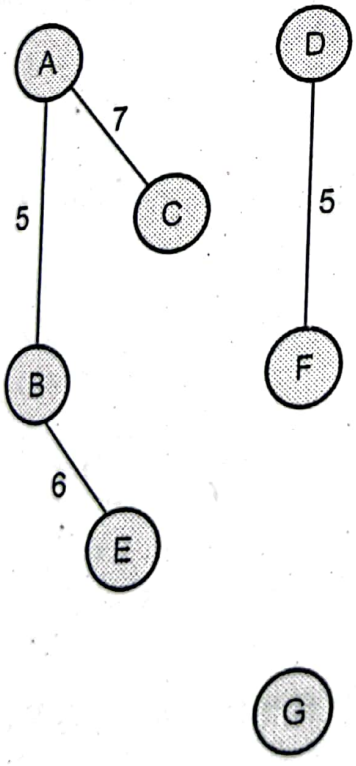
Step 3 :



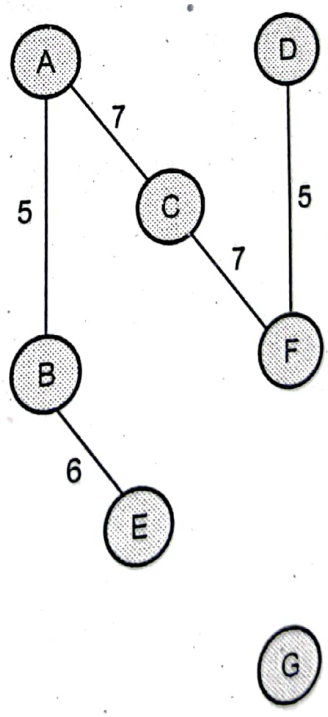
Step 4 :



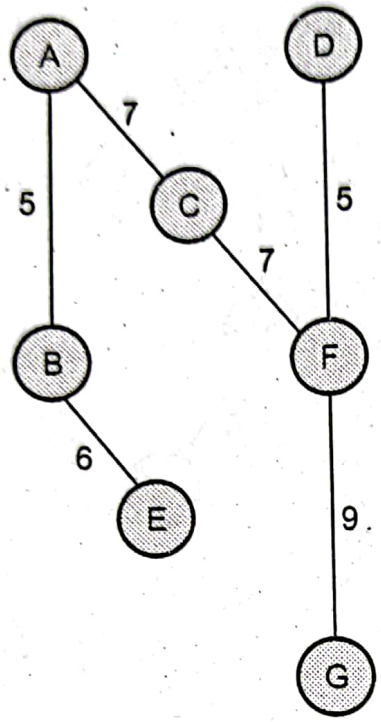
Step 5 :



Step 6 :



Step 7 :



Total path length = 39

Q.24 Write the pseudo code for Kruskal's algorithm and find Minimum spaming tree for the following graph :

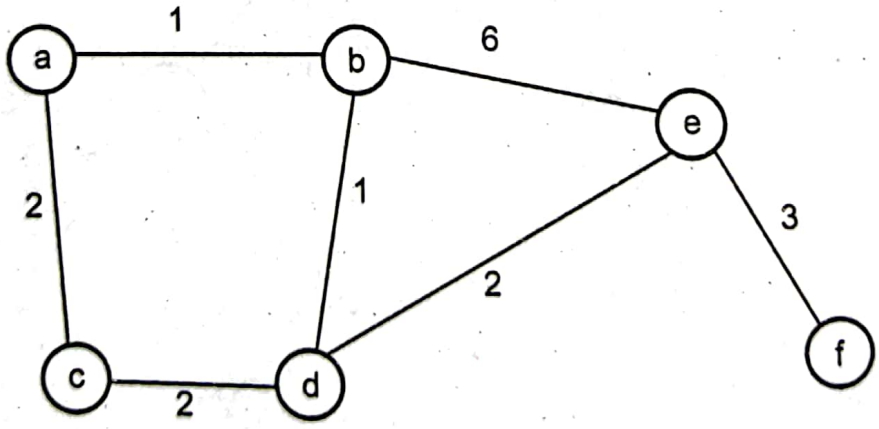
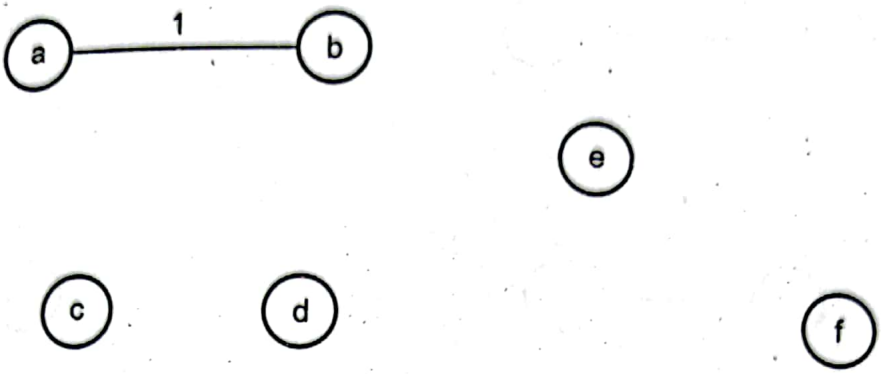


Fig. Q.24.1

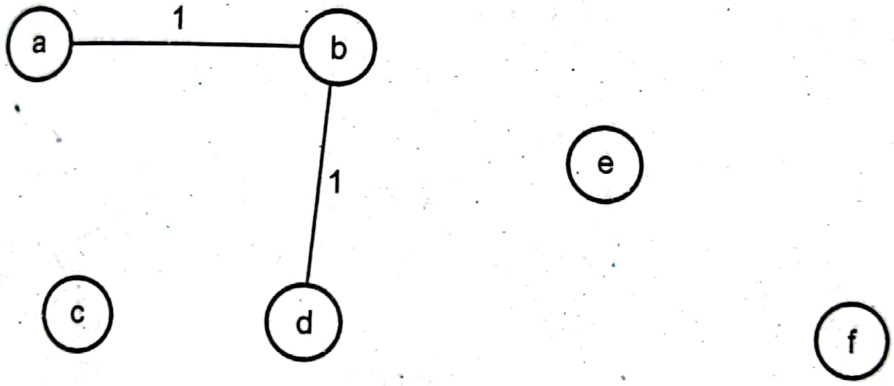
☞ **[SPPU : Dec.-17, Marks 6]**

Ans. : Kruskal's Algorithm - Refer Q.21.

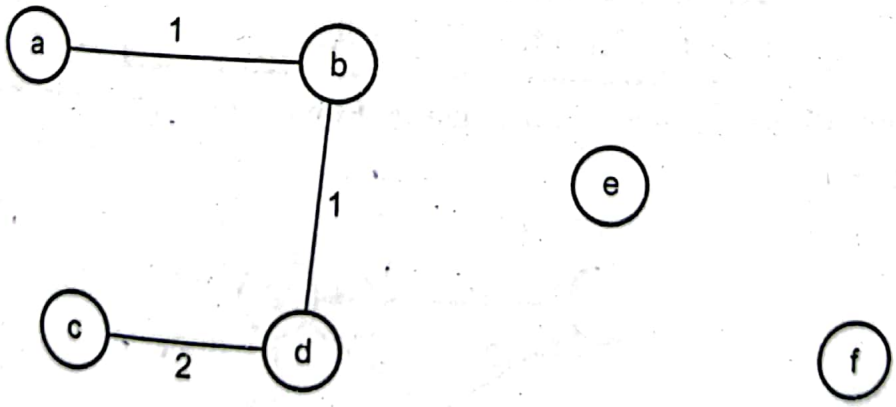
Step 1 : Select a - b.



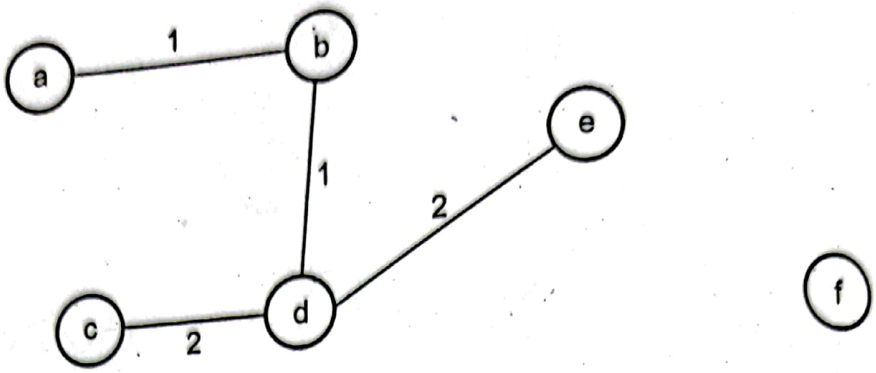
Step 2 : Select b - d



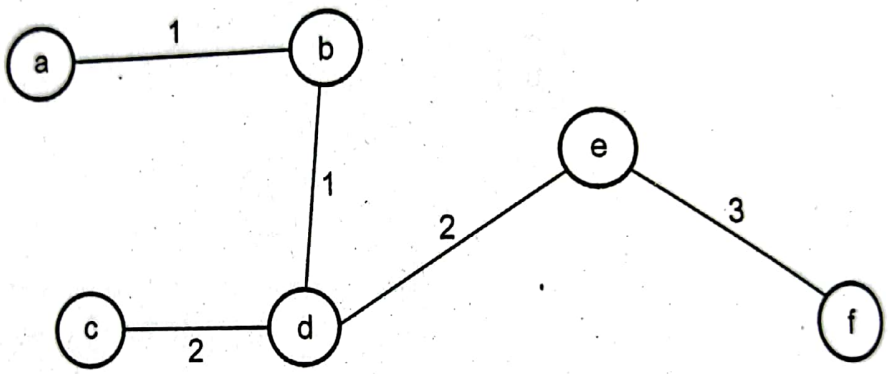
Step 3 : Select c - d



Step 4 : Select d - e



Step 5 : Select e - f



Minimum spanning tree

Total path length = 9

5.6 : Shortest Path Algorithm

Q.25 Find all pair shortest path using Floyd Warshall algorithm for given graph.

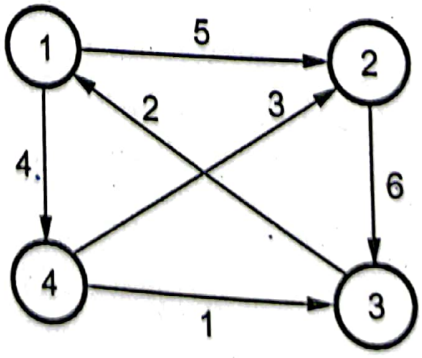


Fig. Q.25.1

Ans. :

We will obtain $D^{(0)}$

$D^{(0)}$ =

	1	2	3	4
1	0	5	∞	4
2	∞	0	6	∞
3	2	∞	0	∞
4	∞	3	1	0

For $D^{(1)}$, k = 1 Hence

$$D^{(1)}[1, 2] = \min \{D^0[i, j], (D^0[i, k] + D^0[k, j])\}$$
$$= \min \{5, (0 + 5)\}$$
$$D^{(1)}[1, 2] = 5$$
$$D^{(1)}[1, 3] = \min \{D^0[i, j], (D^0[i, k] + D^0[k, j])\}$$
$$= \min \{D^0[1, 3], (D^0[1, 1] + D^0[1, 3])\}$$
$$D^{(1)}[1, 3] = \min \{\infty, (0 + \infty)\}$$
$$D^{(1)}[1, 3] = \infty$$
$$D^{(1)}[1, 4] = \min \{D^0[i, j], (D^0[i, k] + D^0[k, j])\}$$
$$= \min \{D^0[1, 4], (D^0[1, 1] + D^0[1, 4])\}$$
$$= \min \{4, (0 + 4)\}$$
$$D^{(1)}[1, 4] = 4$$

Continuing in this fashion we will get changed values as -

$$D^{(1)}[3, 2] = \min \{D^0[i, j], (D^0[i, k] + D^0[k, j])\}$$
$$= \min \{\infty, (D^0[3, 1], D^0[1, 2])$$
$$= \min \{\infty, (2 + 5)\}$$
$$D^{(1)}[3, 2] = 7$$
$$D^{(1)}[3, 4] = \min \{D^0[3, 4], (D^0[3, 1] + D^0[1, 4])$$
$$= \min \{\infty, (2 + 4)\}$$
$$D^{(1)}[3, 4] = 6$$

Thus we get

$D^{(1)}$ =

	1	2	3	4
1	0	5	∞	4
2	∞	0	6	∞
3	2	7	0	6
4	∞	3	1	0

Now consider k = 2

$D^2 [1, 2] = \min \{D^1 [i, j], (D^1 [i, k] + D^1 [k, j])\}$

$= \min \{D^1 [1, 2], (D^2 [1, 2] + D^1 [2, 2])\}$

$= \min \{5, (5 + 0)\}$

$D^2 [1, 2] = 5$

$D^2 [1, 3] = \min \{D^1 [i, j], (D^1 [i, k] + D^1 [k, j])\}$

$= \min \{D^1 [1, 3], (D^1 [1, 2] + D^1 [2, 3])\}$

$= \min \{\infty, (5 + 6)\}$

$D^2 [1, 3] = 11$

The other values remain unchanged. Hence the matrix will be

$D^{(2)}$ =

	1	2	3	4
1	0	5	11	4
2	∞	0	6	∞
3	2	7	0	6
4	∞	3	1	0

Let k = 3

$D^3 [1, 2] = \min \{D^2 [i, j], (D^2 [i, k] + D^{(2)} [k, j])\}$

$= \min \{D^2 [1, 2], (D^2 [1, 3] + D^2 [3, 2])\}$

$= \min \{5, (11 + 7)\}$

$D^3 [1, 2] = 5$

The change in value occurs at location D[2, 1]. This computation is as given below.

$D^3 [2, 1] = \min \{D^3 [2, 1], (D^2 [2, 3] + D^2 [3, 1])\}$

$= \min \{\infty, (6 + 2)\}$

$D^3 [2, 1] = 8$

$D^3[2, 4] = \min \{D^3[2, 4], (D^3[2, 3] + D^3[3, 4])\}$
$= \min \{\infty, (6 + 6)\}$

$D^3[2, 4] = 12$

$D^3[4, 1] = \min \{D^3[4, 1], (D^2[4, 3] + D^3[3, 1])\}$
$= \min \{\infty, (1 + 2)\}$

$D^3[4, 1] = 3$

Thus we get following matrix -

	1	2	3	4
1	0	5	11	4
2	8	0	6	12
3	2	7	0	6
4	3	3	1	0

Now consider k = 4

$D^4[1, 3] = \min \{D^4[1, 3], (D^4[1, 4] + D^4[4, 3])\}$
$= \min \{11, (4 + 1)\}$

$D^4[1, 3] = 5$

All other values remain unchanged. Hence the matrix will be -

$D^{(4)} =$

	1	2	3	4
1	0	5	5	4
2	8	0	6	12
3	2	7	0	6
4	3	3	1	0

Q.26 Obtain the shortest path for the given graph.

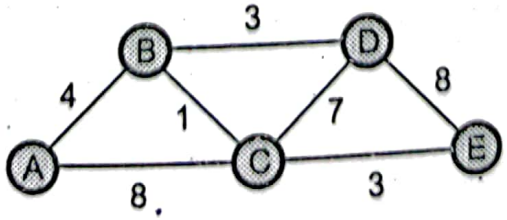
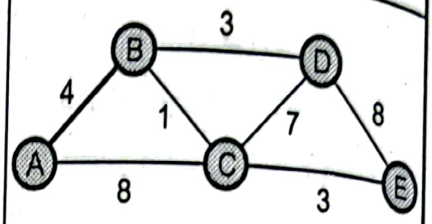
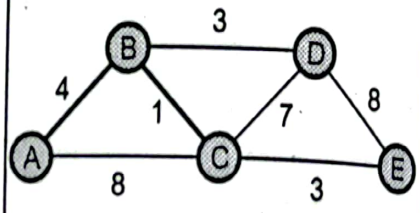
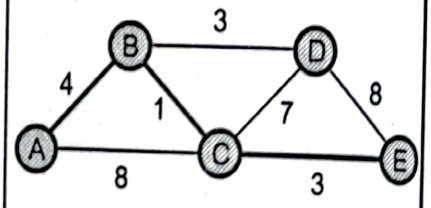


Fig. Q.26.1

Ans. : 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 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	

Thus the shortest distance from A to E is obtained.

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

Q.27 Find the shortest path using Dijkstra algorithm between node A and node F.

☞ **[SPPU : May-19, Marks 6]**

Ans. : Consider source node A.

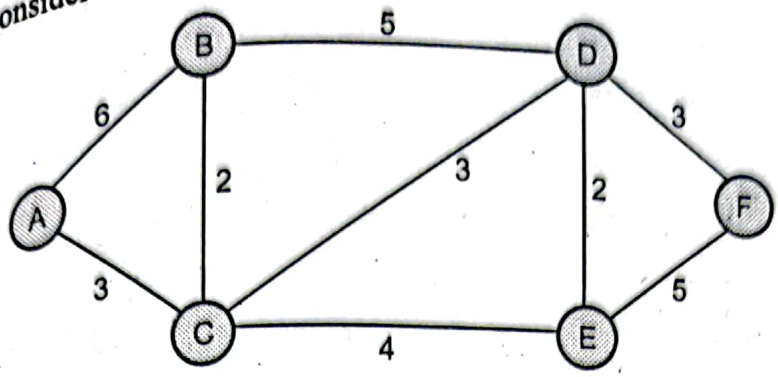


Fig. Q.27.1

Step 1 : S = {A} T = {B, C, D, E, F}

d (A, B) = 6 d (A, D) = ∞

d (A, C) = 3 d (A, E) = ∞

d (A, F) = ∞

The d(A, C) = 3 is minimum distance. Hence we will select vertex C.

Step 2 : S = {A, C} T = {B, D, E, F}

d (A, B) = 6

d (A, D) = d (A, C) + d (C, D) = 6

d (A, E) = d (A, C) + d (C, E) = 7

d (A, F) = ∞

We select d (A, B) as minimum distance. Hence select vertex B.

Step 3 : S = {A,C, B}, T = {D, E, F}

d (A, D) = min {[d (A, B) + d (B, D)], [d (A, C) + d (C, D)]}

= min {[6+5], [3+3]}

d (A, D) = 6

d (A, E) = 7

d (A, F) = ∞.

We will select vertex D.

Step 4 : S = {A, C, B, D}, T = {E, F}

d (A, E) = 7

d (A, F) = d (A, D) + d (D, F) = 6 + 3

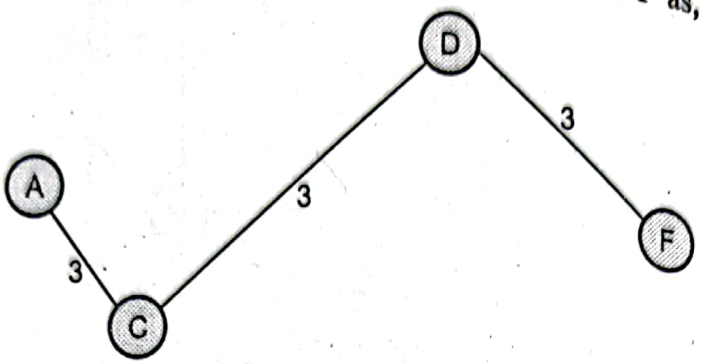
d (A, F) = 9

We will select vertex E.

Step 5 : S = {A, C, B, D, E} T = {F}

d (A, F) = 9

Step 6 : Thus we obtain the shortest path from A and F as,



5.7 : Topological Sorting

Q.28 Write an algorithm for topological sort.

Ans. : Following are the steps to be followed in this algorithm -

1. From a given graph find a vertex with no incoming edges. Delete it along with all the edges outgoing from it. If there are more than one such vertices then break the tie randomly.
2. Note the vertices that are deleted.
3. All these recorded vertices give topologically sorted list.

Q.29 Sort the digraph for topological sort.

Ans. : We will follow following steps to obtain topologically sorted list.

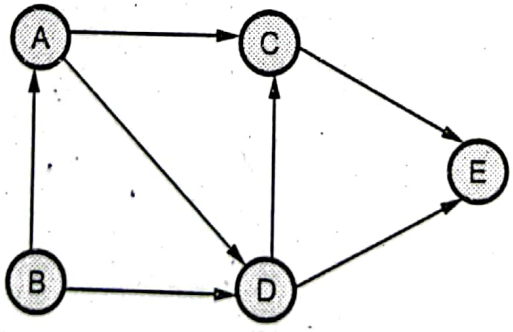


Fig. Q.29.1

Choose vertex B, because it has no incoming edge, delete it along with its adjacent edges.

Hence the list after topological sorting will be B, A, D, C, E.

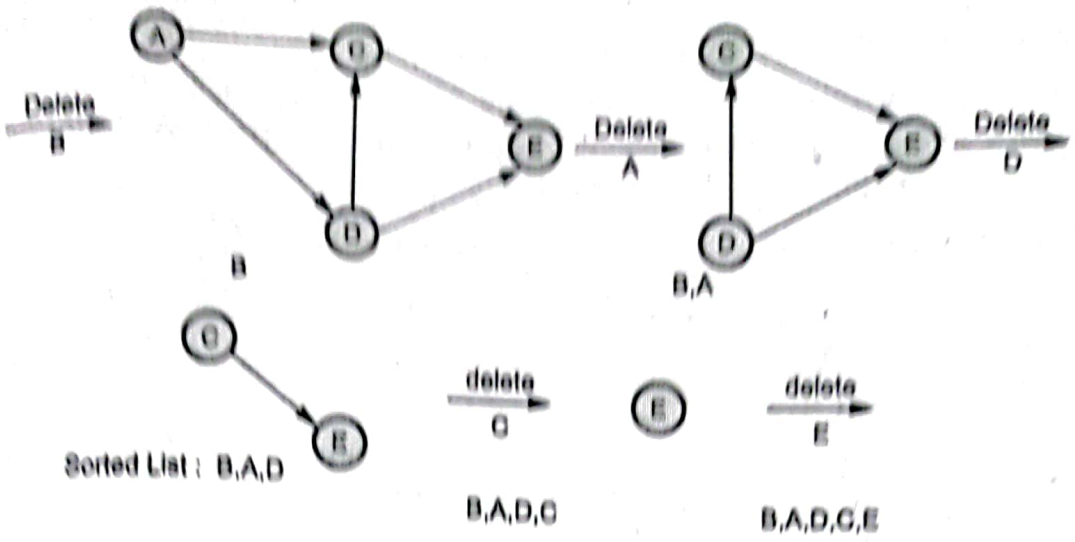


Fig. Q.20.2

Q.30 Consider the following graph, each node contains subject course number :

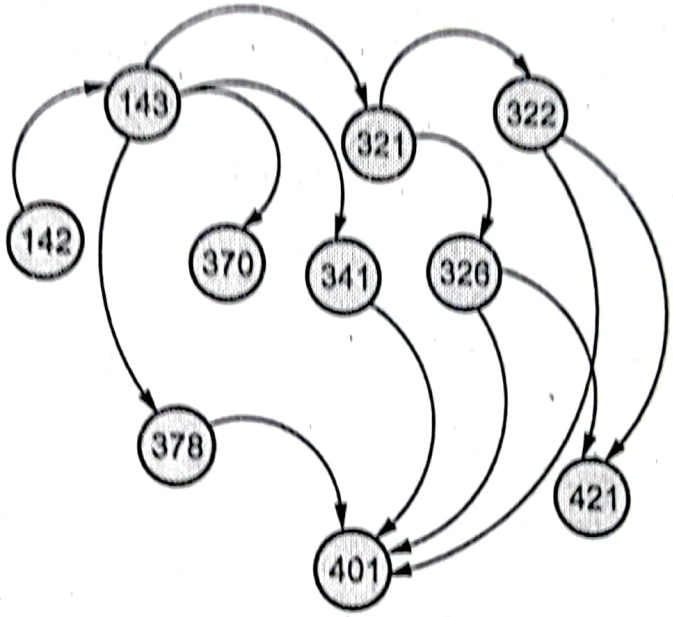


Fig. Q.30.1

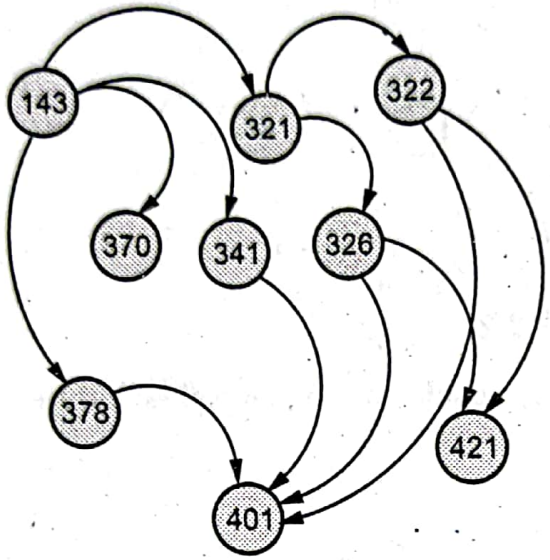
To complete a course, students need to take all the course subjects. Students need to take into consideration which courses are prerequisites for other courses when making a schedule for the upcoming semester so that 370 can not taken before 143, but the former can be taken along with 341 and 370. Help the students by giving the order of subjects they need to take to complete the course. [Hint : Topological sort]

☞ **[SPPU : May-16, Marks 4]**

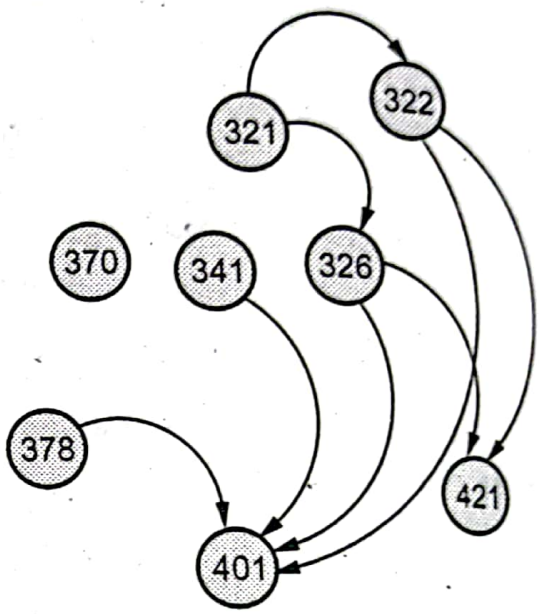
Ans. : We will apply topological sorting technique.

Each time we will find a vertex with no incoming edges. Delete that vertex along with all the outgoing edges from it. If there are more than such vertices then break the tie randomly.

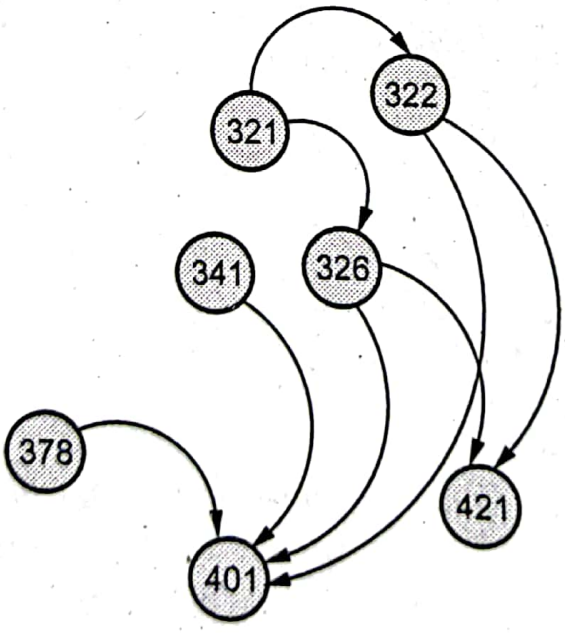
Step 1 : Delete 142



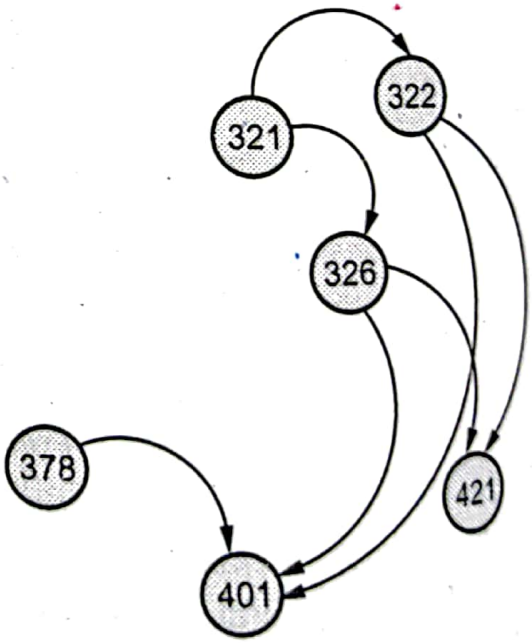
Step 2 : Delete 143



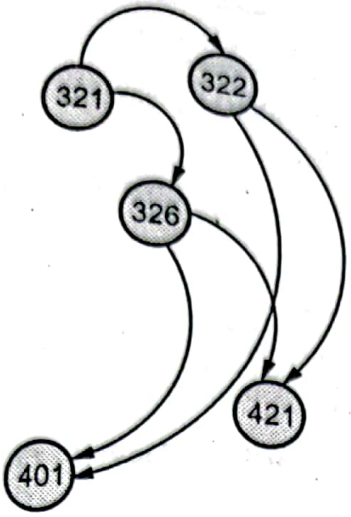
Step 3 : Delete 370



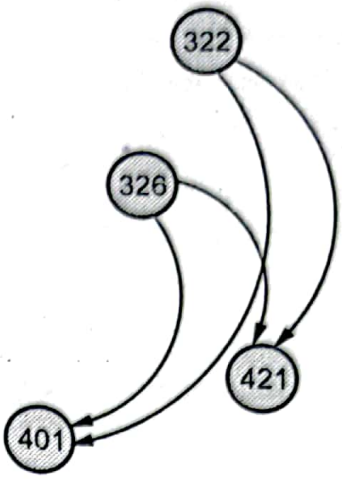
Step 4 : Delete 341



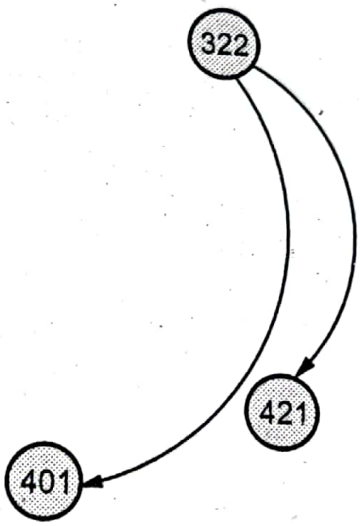
Step 5 : Delete 378



Step 6 : Delete 321

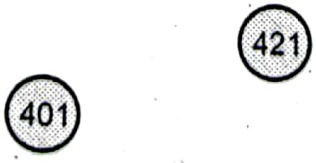


Step 7 : Delete 326



Step 8 : Delete 322

Step 9 : Delete 401



Step 10 : Delete 421

Thus the course schedule can be in following order.

142	143	370, 341	378	321	326, 322	401, 421

Q.31 What is topological sorting ? Where topological sorting can be applicable ? Explain it with suitable example.

☞ **[SPPU : Dec.-19, Marks 6, June-22, Marks 9]**

Ans. : Refer Q.28 and Q.29.

5.8 : Notion of Symbol Table

Q.32 Explain symbol table with its applications.

☞ **[SPPU : May-12, Marks 6]**

Ans. : • Definition

The symbol table is defined as the set of **Name** and **Value** pairs.

For example

	Name	Value
0	a	10
1	b	0
2	c	0
3		
4		
5		
6		

Fig. Q.32.1 Symbol table storing 3 identifiers

Use of symbol table

- The symbol tables are typically used in **compilers**. Basically compiler is a program which scans the application program (for instance : your C program) and produces machine code. During this scan compiler stores the identifiers of that application program in the symbol table. These identifiers are stored in the form of name, value address, type. Here the name represents the name of identifier, value represents the

value stored in an identifier, the address represents memory location of that identifier and type represents the data type of identifier. Thus compiler can keep track of all the identifiers with all the necessary information.

Q.33 What are the advantages of using symbol table ?

Ans. : Following are some advantages of using symbol tables. As symbol table is normally used in compiler, these advantages are relevant to compilers -

1. During compilation of source program, **fast look up** for the required identifiers is possible due to use of symbol table.
2. The runtime allocation for the identifiers is managed using symbol tables.
3. Use of symbol table allows to handle certain issues like scope of identifiers, and implicit declarations.

Q.34 Explain static and dynamic tree tables.

☞[SPPU : Dec.-10, Marks 4]

Ans. : The static symbol table stores the fixed amount of information whereas dynamic symbol table stores the dynamic information.

These symbol tables are normally used to implement static and dynamic data structures. Hence there can be static tree tables or dynamic tree tables, which are implemented using static and dynamic symbol tables respectively.

Examples of static symbol table : Optimal Binary Search Tree (OBST), Huffman's coding can be implemented using static symbol table.

Example of dynamic symbol table : An AVL tree is implemented using dynamic symbol table.

5.9 : OBST

Q.35 What is Optimal Binary Search Tree (OBST) ? What is it's use ?

☞ [SPPU : May-10, Marks 4]

Ans. : Let $\{a_1, a_2, \ldots a_n\}$ be a set of identifiers such that $a_1 < a_2 < a_3$. Let P(i) be the probability with which we can search for a_i and $q_{(i)}$ be the probability of searching element x such that $a_i < x < a_{i+1}$ and $0 \leq i \leq n$.

In other words $p_{(i)}$ is the probability of successful search and q(i) be the probability of unsuccessful search. Also $\sum_{1 \le i \le n} p_{(i)} + \sum_{1 \le i \le n} q_{(i)}$ then obtain a tree with minimum cost. Such a tree with optimum cost is called optimal **binary search tree.** .

- The element having more probability of frequent access is placed near the root, and the elements having less probability of access is placed far from the root. Thus the optimal binary search tree is used to access the frequently required elements more efficiently.

Q.36 Consider n = 4 and (q_1, q_2, q_3, q_4) = (do, if, read, while). The values for p's and q's are given as p(1 : 4) = (3, 3, 1, 1) and q(0 : 4) = (2, 3, 1, 1, 1). Construct the optimal binary search tree.

☞**[SPPU : May-16, Marks 8]**

Ans.: We will apply following formulae for the computation of W, C and r.

$W_{i,i} = q_i$, $r_{i,i} = 0$, $C_{i,i} = 0$

$$W_{i,i+1} = q_i + q_{(i+1)} + p_{(i+1)}$$

$$r_{i,i+1} = i + 1$$

$$C_{i,i+1} = q_i + q_{(i+1)} + p_{(i+1)}$$

$$W_{i,j} = W_{i,j-1} + p_j + q_j$$

$$r_{i,j} = k$$

$$C_{i,j} = \min_{i < k \le j} \{C_{(i,k-1)} + C_{k,j}\} + W_{i,j}$$

- We will construct tables for values of W, C and r.

Let $i = 0$

$$W_{00} = q_0 = 2$$

When i = 1

$$W_{11} = 3$$

When i = 2

$$W_{22} = 1$$

When i = 3

$$W_{33} = 1$$

When i = 4

$$W_{44} = q_4 = 1$$

When i = 0 and j – i = 1 then

$$W_{01} = q_0 + q_1 + p_1 = 2 + 3 + 3$$

$$W_{01} = 8$$

When i =1 and j – i = 1

$$W_{12} = q_1 + q_2 + p_2 = 3 + 1 + 3$$

$$W_{12} = 7$$

When i = 2 and j – i = 1

$$W_{23} = q_2 + q_3 + p_3 = 1 + 1 + 1$$

$$W_{23} = 3$$

When i = 3 and j – i = 1

$$W_{34} = q_3 + q_4 + p_4 = 1 + 1 + 1$$

$$W_{34} = 3$$

Now, when i = 0 and j – i = 2

$$W_{ij} = W_{i, j-1} + p_j + q_j$$

$$W_{02} = W_{01} + p_2 + q_2 = 8 + 3 + 1$$

$$W_{02} = 12$$

When i =1 and j – i = 2

$$W_{13} = W_{12} + p_3 + q_3 = 7 + 1 + 1$$

$$W_{13} = 9$$

When i = 2 and j – i = 2 then

$$W_{24} = W_{23} + p_4 + q_4 = 3 + 1 + 1$$

$$W_{24} = 5$$

Now, when i = 0 and j – i = 3 then

$$W_{03} = W_{02} + p_3 + q_3 = 12 + 1 + 1$$

$$W_{03} = 14$$

When i = 1 and j – i = 3 then

$$W_{14} = W_{13} + q_4 + p_4 = 9 + 1 + 1$$

$$W_{14} = 11$$

When i = 0 and j – i = 4 then

$$W_{04} = W_{03} + q_4 + p_4 = 14 + 1 + 1$$

$$W_{04} = 16$$

- The table for W can be represented as

j – i \ i	0	1	2	3	4
0	$W_{00} = 2$	$W_{11} = 3$	$W_{22} = 1$	$W_{33} = 1$	$W_{44} = 1$
1	$W_{01} = 8$	$W_{12} = 7$	$W_{12} = 3$	$W_{34} = 3$	
2	$W_{02} = 12$	$W_{13} = 9$	$W_{24} = 5$		
3	$W_{03} = 14$	$W_{14} = 11$			
4	$W_{04} = 16$				

- We will now compute for C and r.

As $C_{i,i} = 0$ and $r_{i,i} = 0$

$C_{00} = 0$ $C_{11} = 0$ $C_{22} = 0$ $C_{33} = 0$ $C_{44} = 0$

$r_{00} = 0$ $r_{11} = 0$ $r_{22} = 0$ $r_{33} = 0$ $r_{44} = 0$

Similarly, $C_{i,i+1} = q_i + q_{(i+1)} + p_{(i+1)}$ and $r_{i,i+1} = i+1$

∴ When i = 0

$$C_{01} = q_0 + q_1 + p_1 = 2 + 3 + 3$$

$$C_{01} = 8$$

$$r_{01} = 1$$

When i = 1

$$C_{12} = q_1 + q_2 + p_2 = 3 + 1 + 3$$

$$C_{12} = 7 \quad \text{and} \quad r_{12} = 2$$

When i = 2

$$C_{23} = q_2 + q_3 + p_3 = 1 + 1 + 1$$

$C_{23} = 3$ and $r_{23} = 3$

When $i = 3$

$C_{34} = q_3 + q_4 + p_4 = 1 + 1 + 1$

$C_{34} = 3$ and $r_{34} = 4$

Now we will compute C_{ij} and r_{ij} for $j - 1 \geq 2$.

$$C_{i,j} = \min_{i<k\leq j} \{C_{(i,k-1)} + C_{k,j}\} + W_{i,j}$$

As

Hence we will find k.

- For C_{02} we have $i = 0$ and $j = 2$. Value of K will be between i and j i.e. between 0 and 2. For $r_{i,j-1}$ to $r_{i+1,j}$ i.e. for r_{01} to $r_{1,2}$. We will compute minimum value of C_{ij}.
- Let $r_{01} = 1$ and $r_{12} = 2$. Then we will assume value of $k = 1$ and will compute C_{ij}. Similarly with $k = 2$ we will compute C_{ij} and will pick up minimum value of C_{ij} only.
- Let us compute C_{ij} with following formula,

$$C_{ij} = C_{i,k-1} + C_{kj}$$

For k = 1, i = 0, j = 2.

$$C_{02} = C_{00} + C_{12} = 0 + 7$$

$$C_{02} = 7 \quad \text{...(Q.36.1)}$$

For k = 2, i = 0, j = 2

$$C_{02} = C_{01} + C_{22} = 8 + 0$$

$$C_{02} = 8 \quad \text{...(Q.36.2)}$$

- From equations (Q.36.1) and (Q.36.2) we can select minimum value of C_{02} is 7. That means $\mathbf{k = 1}$ gives us minimum value of $C_{i,j}$.

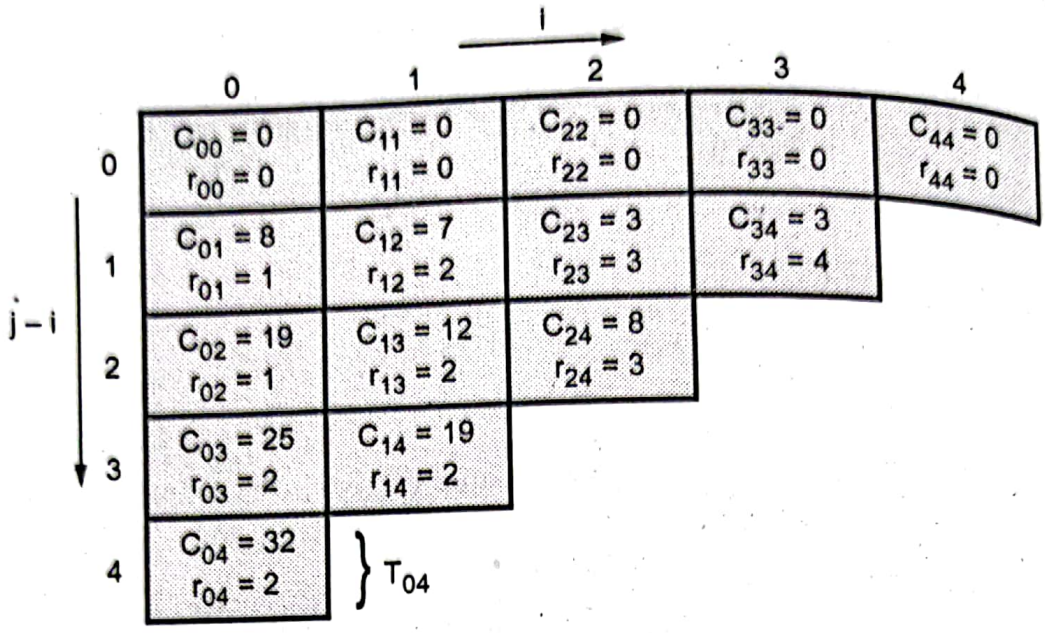
Hence Now $\mathbf{r_{ij} = r_{02} = k = 1}$

$$C_{02} = \min\{C_{(i,k-1)} + C_{kj}\} + W_{ij}$$

$$= 7 + W_{02} = 7 + 12$$

$$C_{02} = 19$$

- Continuing in this fashion we can compute C_{ij} and r_{ij}. It is given as follows.



- Therefore T_{04} has a root r_{04}. The value of r_{04} is 2. From (a_1, a_2, a_3, a_4) = (do, if, read, while) a_2 becomes the root node. Hence root is if.

$$r_{ij} = k$$

$$r_{04} = 2$$

- Then r_{ik-1} becomes left child and r_{kj} becomes the right child. In other words r_{01} becomes the left child and r_{24} becomes right child of r_{ij}. Here $r_{01} = 1$ so a_1 becomes left child of a_2 and $r_{24} = 3$ so a_3 becomes right child of a_2.

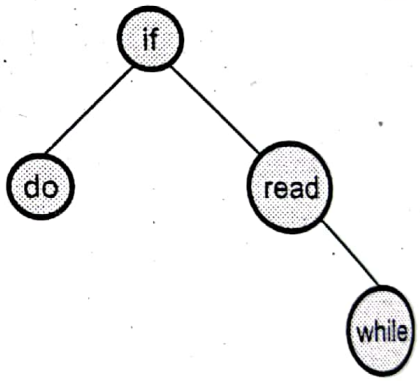


Fig. Q.36.1 OBST

- For the node r_{24} i = 2, j = 4 and k =3. Hence left child of it is $r_{ik-1} = r_{22} = 0$. That means left child of $r_{24} = a_3$ is empty. The right child of r_{24} is $r_{34} = 4$. Hence a_4 becomes right child of a_3. Thus all n = 4 nodes are used to build a tree. The optimal binary search tree is show in Fig. Q.36.1

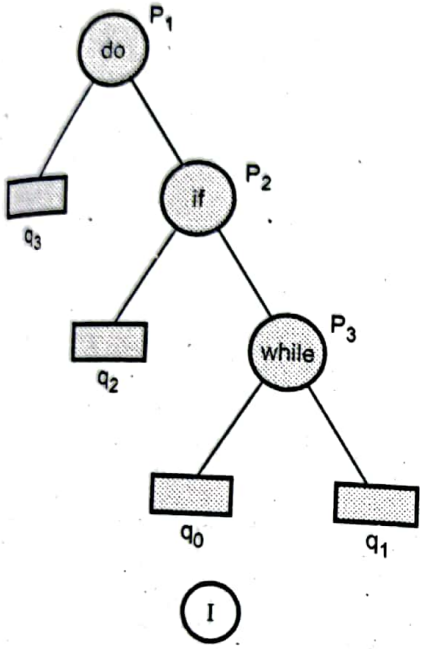
Q.37 How many binary search trees (BSTs) can be constructed for given 'n' identifiers ? i) Construct all possible Binary Search Trees for the identifier set (q_1, q_2, q_3) = (do, if, while)
ii) Compute the total cost of each BST constructed by you assuming equal probabilities for successful and unsuccessful search i.e.

$(p_1, p_2, p_3) = (1/7, 1/7, 1/7)$ for successful search and $(q_0, q_1, q_2, q_3) = (1/7, 1/7, 1/7, 1/7)$ for unsuccessful search. Which BST is an optimal binary search tree ?

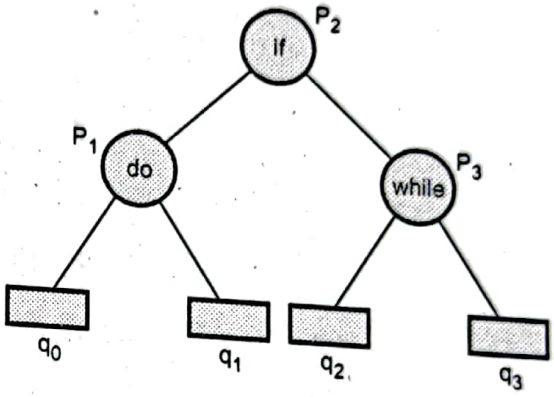
iii) Compute the total cost of each BST constructed by you assuming probabilities for successful and unsuccessful search as : $(p_1, p_2, p_3) = (0.5, 0.1, 0.05)$ for successful search and $(q_0, q_1, q_2, q_3) = (0.15, 0.1, 0.05, 0.05)$ for unsuccessful search. Which BST is an optimal binary search tree ?

☞[SPPU : Dec.-09, 17, Marks 10]

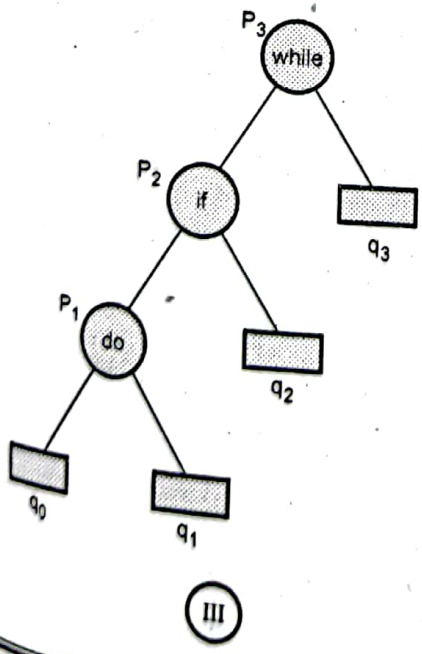
Ans.: i) All possible binary search tree for (do, if, while) are -



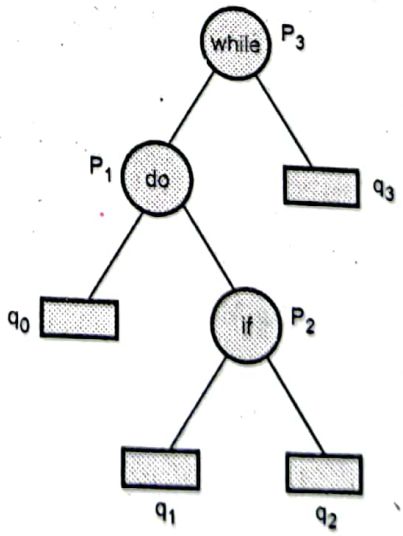
I



II



III



IV

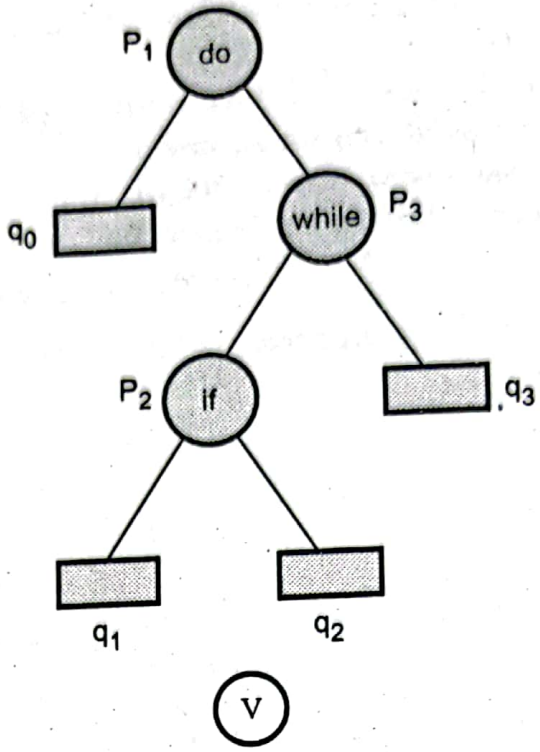


Fig. Q.37.1 All possible BSTs for 3 nodes

- Thus there are 5 possible BSTs for 3 identifiers. Let a_i be a node then the cost can be computed by following formula -

ii) Cost (tree) = $\sum_{1 \le i \le n} P(i) * \text{level}(a_i) + \sum_{0 \le i \le n} q(i) * (\text{level}(E_i) - 1)$

where E_i represents the equivalence class for all the identifiers a_i.

Cost (tree I) =

$$\left[\frac{1}{7}*3+\frac{1}{7}*2+\frac{1}{7}*1\right]+\left[\frac{1}{7}*(4-1)+\frac{1}{7}*(4-1)+\frac{1}{7}*(3-1)+\frac{1}{7}*(2-1)\right]$$

successful nodes (such as do, if, while)

(unsuccessful nodes denoted by square)

$$=\frac{6}{7}+\frac{9}{7}$$

Cost (tree I) = $\frac{15}{7}$

Cost (treeII) =

$$\left[\left(\frac{1}{7}*1\right)+\left(2*\frac{1}{7}+2*\frac{1}{7}\right)\right]+\left[\frac{1}{7}*(3-1)+\frac{1}{7}*(3-1)+\frac{1}{7}*(3-1)+\frac{1}{7}*(3-1)\right]$$

$$=\frac{5}{7}+\frac{8}{7}$$

Cost(tree II) = $\frac{13}{7}$

$$\text{Cost(tree III)} = \left[\frac{1}{7}*1+\frac{1}{7}*2+\frac{1}{7}*3\right]+\left[\frac{1}{7}*(2-1)+\frac{1}{7}*(3-1)+\frac{1}{7}*2*(4-1)\right]$$

$$=\frac{6}{7}+\frac{9}{7}$$

Cost (tree III) = $\frac{15}{7}$

Cost (tree IV) =

$$\left[\frac{1}{7}*1+\frac{1}{7}*2+\frac{1}{7}*3\right]+\left[\frac{1}{7}*(2-1)+\frac{1}{7}*(3-1)+\frac{1}{7}*(4-1)+\frac{1}{7}*(4-1)\right]=\frac{6}{7}+\frac{9}{7}$$

Cost (tree IV) = $\frac{15}{7}$

Cost (tree V) =

$$\left[\frac{1}{7}*1+\frac{1}{7}*2+\frac{1}{7}*3\right]+\left[\frac{1}{7}*(2-1)+\frac{1}{7}*(3-1)+\frac{1}{7}*(4-1)+\frac{1}{7}*(4-1)\right]$$

Cost (tree V) = $\frac{15}{7}$

- From all these computations tree II is an optimal binary search tree because it has least cost.

iii) Using the same formula as mentioned in (ii) we will compute the cost of all the trees shown in Fig. Q.37.1.

Cost (tree I)

$$=[(0.5*1+0.1*2+0.05*3)]+[0.05*1+0.05*2+0.15*3+0.1*3]$$

$$=[0.5+0.2+0.15]+[0.90]=0.80+0.90$$

Cost (tree I) = 1.70

Cost (tree II) = $[P_2 * 1 + P_1 * 2 + P_3 * 2] + [q_0 * 2 + q_1 * 2 + q_2 * 2 + q_3 * 2]$
$= [0.1 * 1 + 0.5 * 2 + 0.05 * 2] + [0.15 * 2 + 0.1 * 2 + 0.05 * 2 + 0.05 * 2]$

Cost (tree II) = 1.90

Cost (tree III) = $[P_1 * 3 + P_2 * 2 + P_3 * 1] + [3 * (q_0 + q_1) + 2 * (q_2) + 1 * q_3]$
$= [0.5 * 3 + 0.1 * 2 + 0.05 * 1] + [3 * (0.15 + 0.1) + 2 * 0.05 + 0.05]$

Cost (tree III) = 1.30

Cost (tree IV) = $[P_1 * 2 + P_2 * 3 + P_3 * 1] + [2 * q_0 + 3 * (q_1 + q_2) + 1 * q_3]$
$= [0.5 * 2 + 0.1 * 3 + 0.05 * 1] + [2 * 0.15 + 3 * (0.1 + 0.05) + 0.05]$

Cost (tree IV) = 1.25

Cost (tree V) = $[P_1 * 1 + P_2 * 3 + P_3 * 2] + [1 * q_0 + 3 * q_1 + 3 * q_2 + 2 * q_3]$
$= [0.5 * 1 + 0.1 * 3 + 0.05 * 2] + [1 * 0.15 + 3 * 0.1 * 3 * 0.05 + 2 * 0.05]$

Cost (tree V) = 1.60

All these computations show that tree IV is optimal binary search tree.

5.10 AVL Trees

Q.38 What is height balance tree ? Explain with one example.

Ans. : An empty tree is height balanced if T is a non empty binary tree with T_L and T_R as its left and right subtrees. The T is height balanced if and only if

i) T_L and T_R are height balanced.

ii) $h_L - h_R <= 1$ where h_L and h_R are heights of T_L and T_R.

For any node in AVL tree the balance factor i.e. BF(T) is **–1, 0 or +1.**

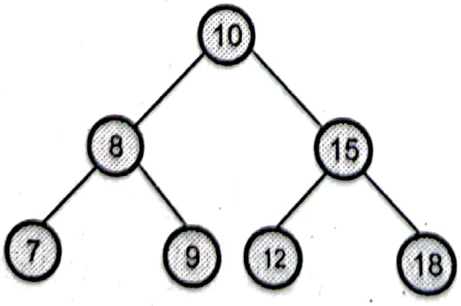


Fig. Q.38.1 AVL tree

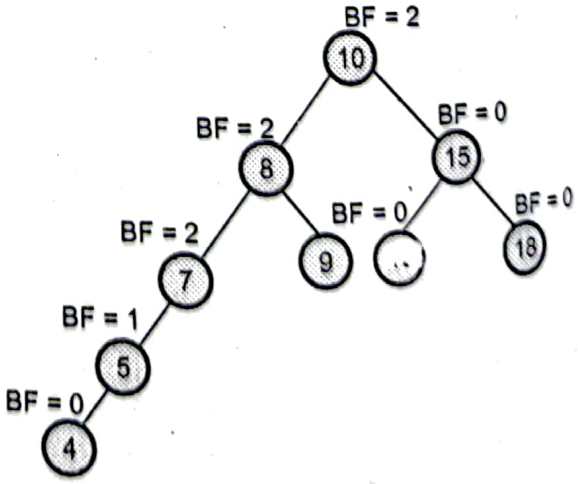


Fig. Q.38.2 Not an AVL tree

Q.39 Explain the following terms with respect to height balanced tree : i) LL ii) RR iii) LR iv) RL

Ans. : There are four different cases when rebalancing is required after insertion of new node.

1. An insertion of new node into left subtree of left child (LL).
2. An insertion of new node into right subtree of left child (LR).
3. An insertion of new node into left subtree of right child (RL).
4. An insertion of new node into right subtree of right child (RR).

Q.40 Explain with suitable examples LL, LR, RR and RL rotations for AVL tree.

Ans. : 1. LL rotation

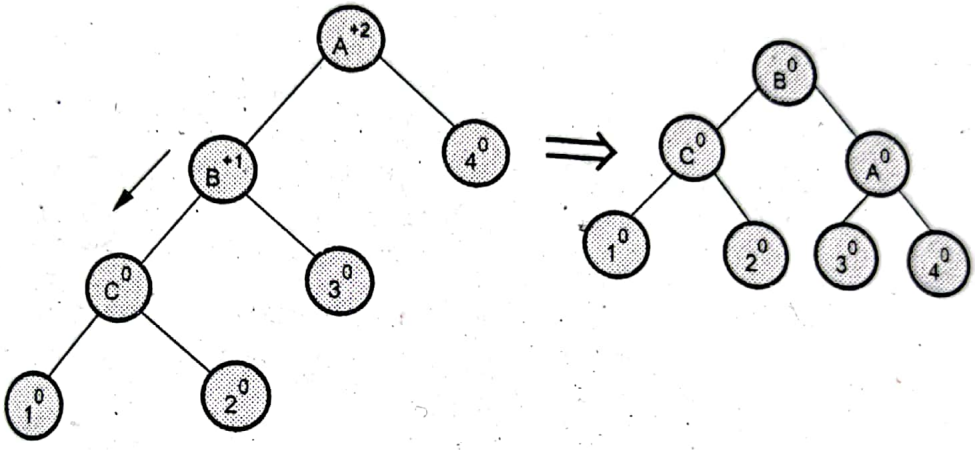


Fig. Q.40.1 LL rotation

2. RR rotation

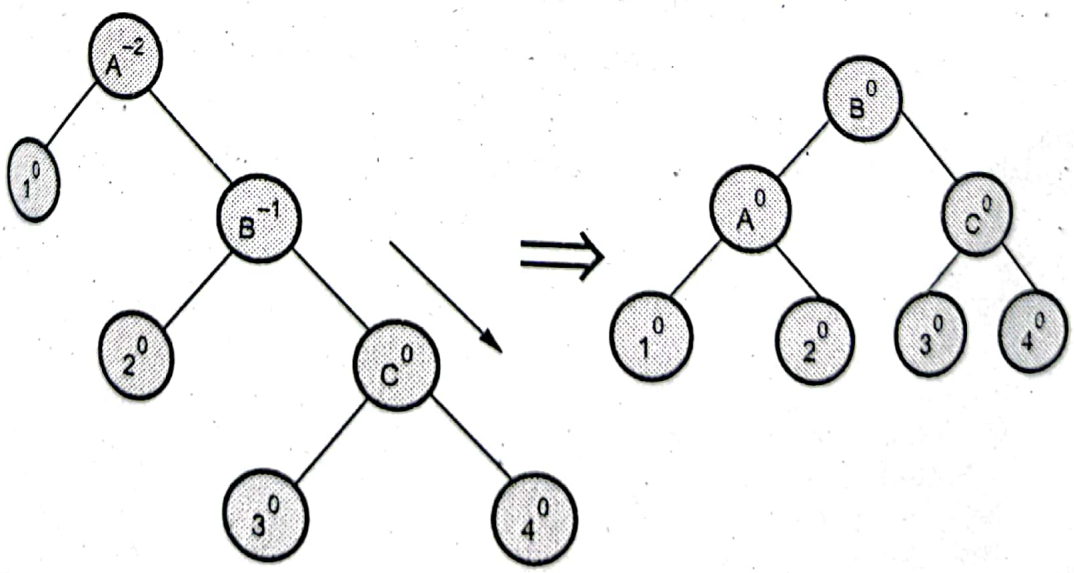


Fig. Q.40.2 RR rotation

3. LR rotation

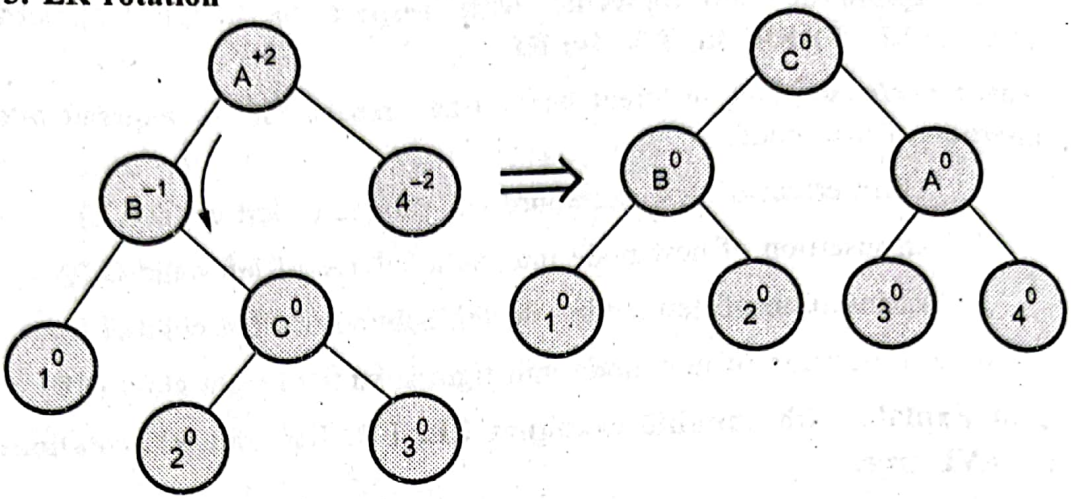


Fig. Q.40.3 LR rotation

4. RL rotation

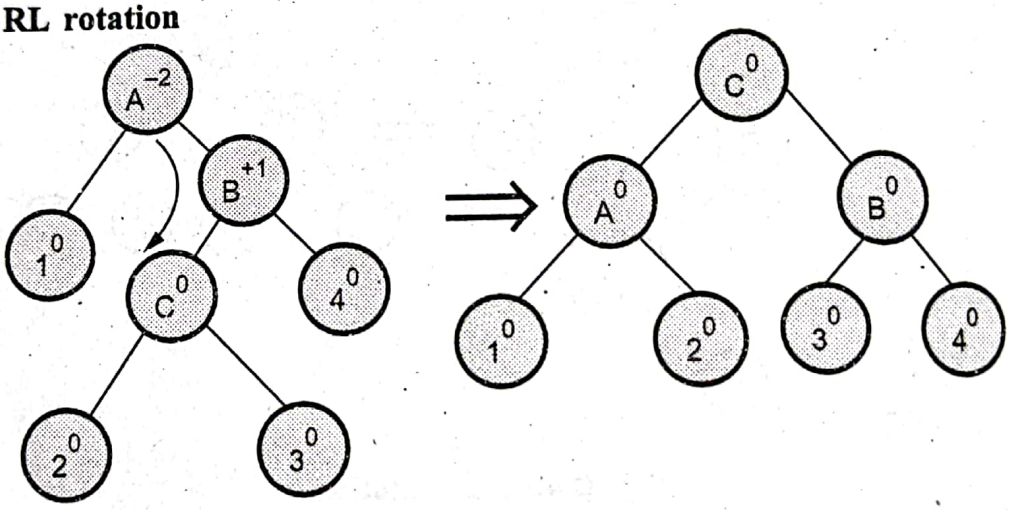


Fig. Q.40.4 RL rotation

Q.41 Write pseudo C/C++ code for LL, LR, RR and RL.

Ans. :

```
node *AVL::create(struct Node *root,int data,int *current)
{
 node *temp1,*temp2;
 if(root==NULL)//initial node
 {
    root= new node;
    root->data=data;
    root->left=NULL;
    root->right=NULL;
    root->BF=0;
```

```
    *current=TRUE;
    return(root);
}
if(data<root->data)
{
  root->left=create(root->left,data,current);
  // adjusting left subtree
  if(*current)
  {
  switch(root->BF)
  {
    case 1:temp1=root->left;
                if(temp1->BF==1)
                {
                   cout<<"\n single rotation: LL rotation";
                   root->left=temp1->right;
                   temp1->right=root;
                   root->BF=0;
                   root=temp1;
                }
                else
                {
                   cout<<"\n Double roation:LR rotation";
                   temp2=temp1->right;
                   temp1->right=temp2->left;
                   temp2->left=temp1;
                   root->left=temp2->right;
                   temp2->right=root;
                   if(temp2->BF==1)
                    root->BF=-1;
                   else
                    root->BF=0;
                   if(temp2->BF==-1)
                    temp1->BF=1;
                   else
                    temp1->BF=0;
                   root=temp2;
                }
                root->BF=0;
                *current=FALSE;
                break;
case 0:

                root->BF=1;
                break;
```

```
        case –1:
                    root–>BF=0;
                    *current=FALSE;
    }
 }
}
if(data> root–>data)
{
root–>right=create(root–>right,data,current);
//adjusting the right subtree
if(*current!=NULL)
{
   switch(root–>BF)
   {
    case 1:
                    root–>BF=0;
                    *current=FALSE;
                    break;
    case 0:
                    root–>BF=–1;
                    break;
    case –1:
                    temp1=root–>right;
                    if(temp1–>BF==–1)
                    {
                       cout<<"\n single rotation:RR rotation";
                       root–>right=temp1–>left;
                       temp1–>left=root;
                       root–>BF=0;
                       root=temp1;
                    }
                    else
                    {
                       cout<<"\n Double rotation:RL rotation";
                       temp2=temp1–>left;
                       temp1–>left=temp2–>right;
                       temp2–>right=temp1;
                       root–>right=temp2–>left;
                       temp2–>left=root;
                       if(temp2–>BF==–1)
                        root–>BF=1;
                       else
                        root–>BF=0;
                       if(temp2–>BF==1)
```

```
                    temp1->BF=-1;
                  else
                    temp1->BF=0;
                  root=temp2;
                }
                root->BF=0;
                *current=FALSE;
            }
        }
    }
    return(root);
}
```

Q.42 Write and explain algorithm to insert a node into AVL tree.

Ans. : 1. Insert a new node as new leaf just as in ordinary binary search tree.

2. Now trace the path from insertion point (new node inserted as leaf) towards root. For each node 'n' encountered, check if heights of left (n) and right (n) differ by at most 1.
   a) If yes, move towards parent (n).
   b) Otherwise restructure by doing suitable rotations such as LL, LR, RR, RL.

Thus once we perform a rotation at node 'n' we do not require to perform any rotation at any ancestor on 'n'.

For example - Refer Q.41.

Q.43 Write and explain algorithm to delete node from AVL tree.

Ans. : The deletion algorithm is more complex than insertion algorithm.

1. Search the node which is to be deleted.
2. a) If the node to be deleted is a leaf node then simply make it NULL to remove.
   b) If the node to be deleted is not a leaf node i.e. node may have one or two children, then the node must be swapped with its inorder successor. Once the node is swapped, we can remove this node.
3. Now we have to traverse back up the path towards root, checking the balance factor of every node along the path. If we encounter unbalancing in some subtree then balance that subtree using appropriate single or double rotations.

The deletion algorithm takes **O(log n)** time to delete any node.

Consider an AVL tree.

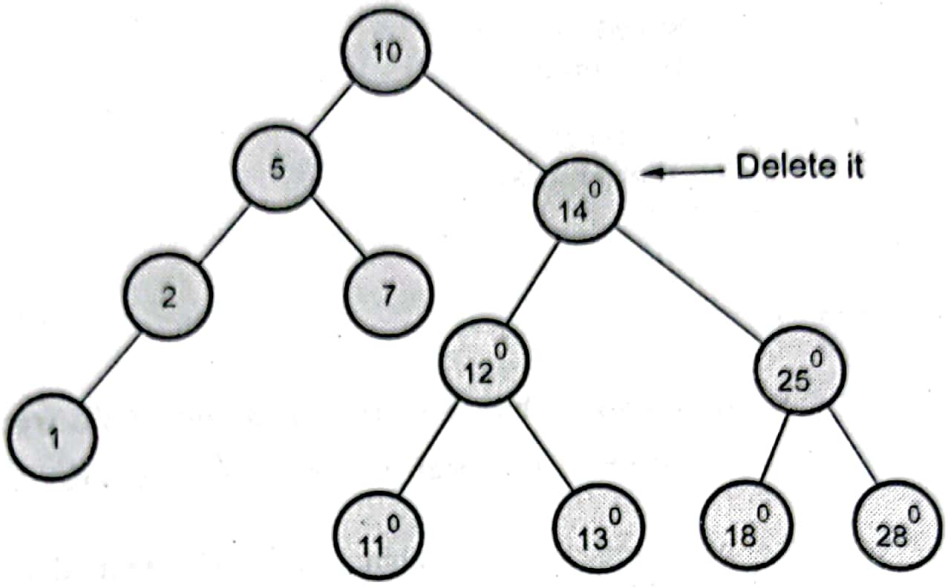


Fig. Q.43.1

The tree becomes

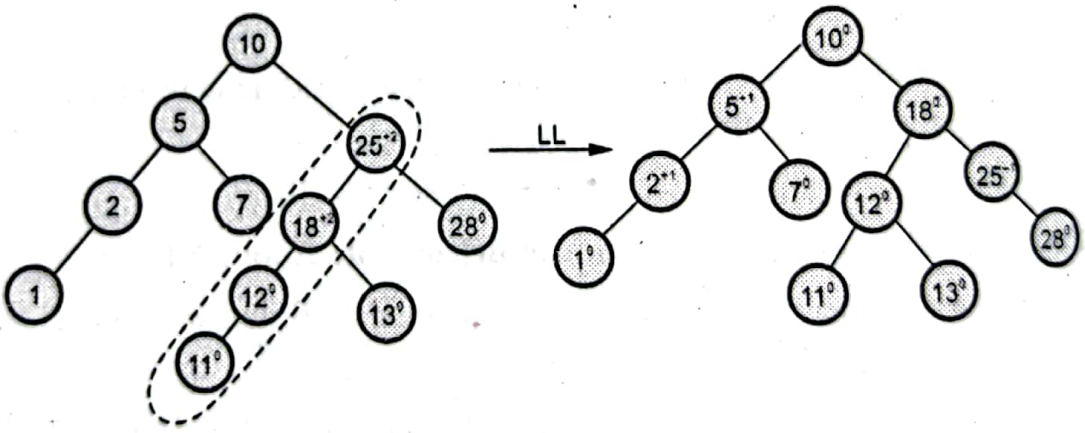


Fig. Q.43.2

Thus the node 14 gets deleted from AVL tree.

Q.44 What is the size of the smallest AVL tree with height 8 ?

☞ **[SPPU : Dec.-05, Marks 4]**

Ans. : If h is the height of an AVL tree then minimum number of nodes can be obtained for such AVL tree using the following formula.

$$N(h) = 1 + N(h - 1) + N(h - 2) \quad \ldots(Q.44.1)$$

If there is only one node then h = 0, for 2 nodes the h = 1. We can formulate and say

$$N(0) = 1, \quad N(1) = 2 \qquad ...(Q.44.2)$$

Hence using equation (1) we can obtain minimum number of nodes for height 8. .

$$N(8) = 1 + N(7) + N(6)$$

$$N(7) = 1 + N(6) + N(5) \qquad ...(Q.44.3)$$

$$N(6) = 1 + N(5) + N(4) \qquad ...(Q.44.4)$$

$$N(5) = 1 + N(4) + N(3) \qquad ...(Q.44.5)$$

$$N(4) = 1 + N(3) + N(2) \qquad ...(Q.44.6)$$

$$N(3) = 1 + N(2) + N(1) \qquad ...(Q.44.7)$$

$$N(2) = 1 + N(1) + N(0) \qquad ...(Q.44.8)$$

$\therefore$ $N(2) = 1 + 2 + 1$ ... $\because$ From equation (Q.44.2)

$$\mathbf{N(2) = 4}$$

Put N(2) in equation (Q.44.8)

$\therefore$ $N(3) = 1 + 4 + 2$

$$\mathbf{N(3) = 7}$$

Put N(3) in equation (Q.44.7)

$\therefore$ $N(4) = 1 + 7 + 4$

$$\mathbf{N(4) = 12}$$

Put N(4) in equation (Q.44.6)

$\therefore$ $N(5) = 1 + 12 + 7$

$$\mathbf{N(5) = 20}$$

Put N(5) in equation (Q.44.5)

$\therefore$ $N(6) = 1 + 20 + 12$

$$\mathbf{N(6) = 33}$$

Put N(6) in equation (Q.44.4)

$\therefore$ $N(7) = 1 + 33 + 20$

$$\mathbf{N(7) = 54}$$

Put N(7) in equation (Q.44.3)

$\therefore$ $N(8) = 1 + 54 + 33$

$N(8) = 88$

That means minimum number of nodes are 88 for an AVL tree of height 8.

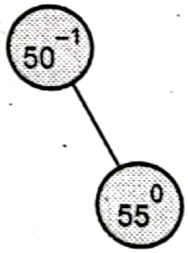
Q.45 Obtain an AVL tree by inserting one data element at a time in the following sequence :
50, 55, 60, 15, 10, 40, 20, 45, 30, 70, 80
Label the rotations appropriately at each stage.

☞ **[SPPU : Dec.-14, Marks 8]**

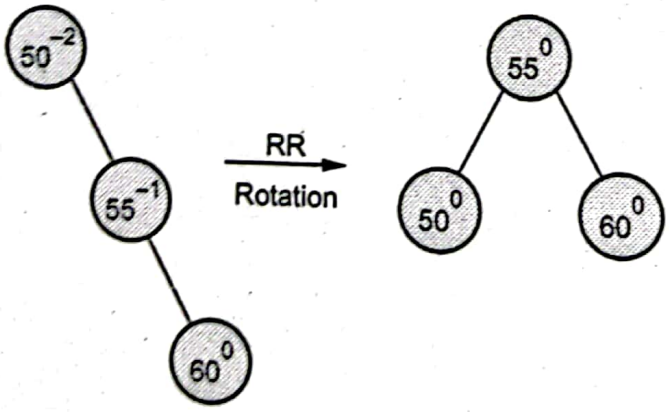
Ans. :

Step 1 : Insert 50

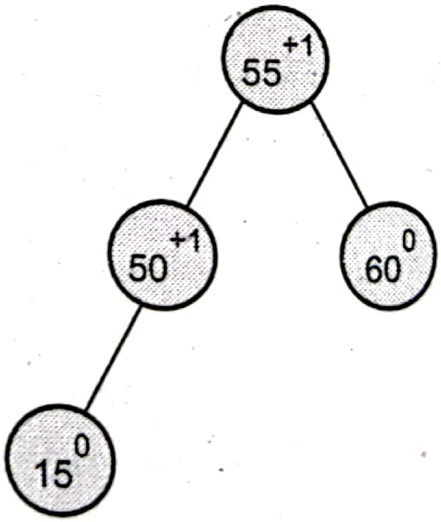
Step 2 : Insert 55



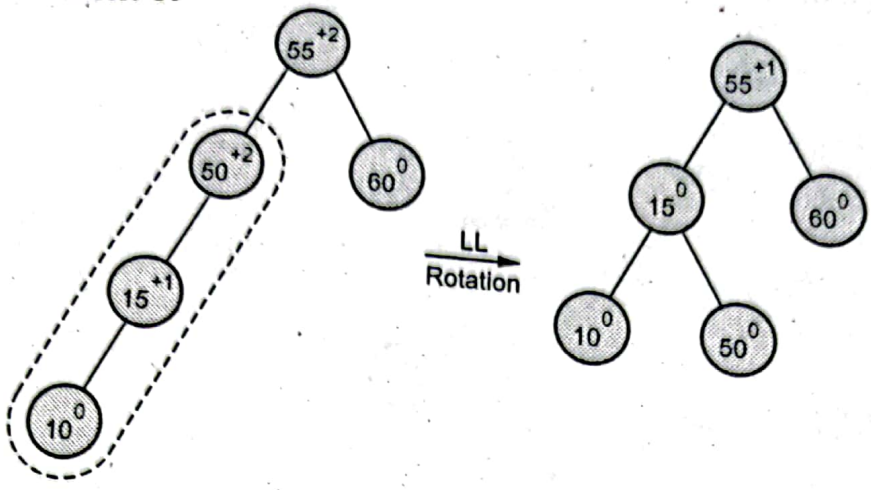
Step 3 : Insert 60



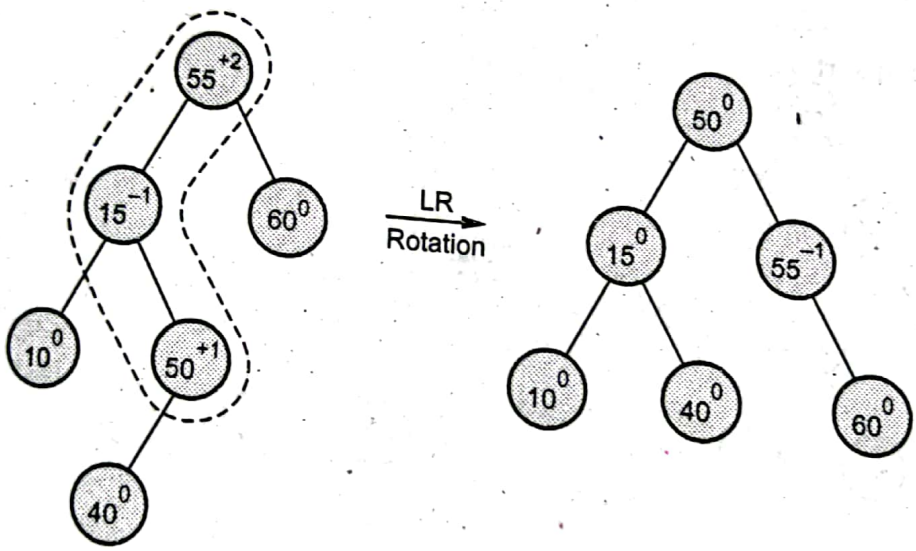
Step 4 : Insert 15



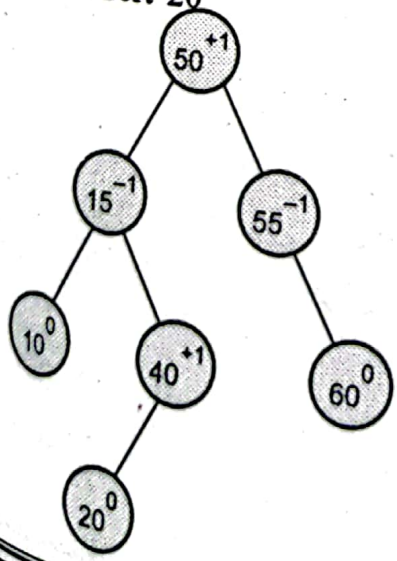
Step 5 : Insert 10



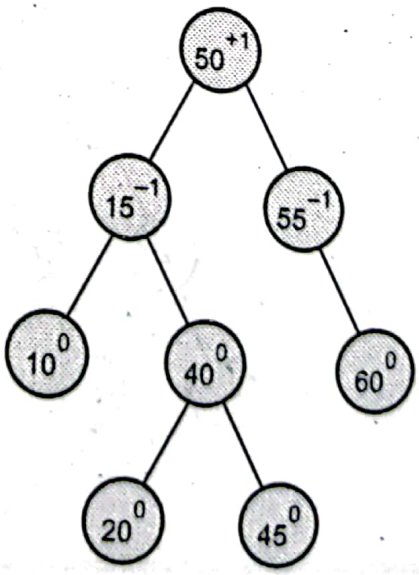
Step 6 : Insert 40



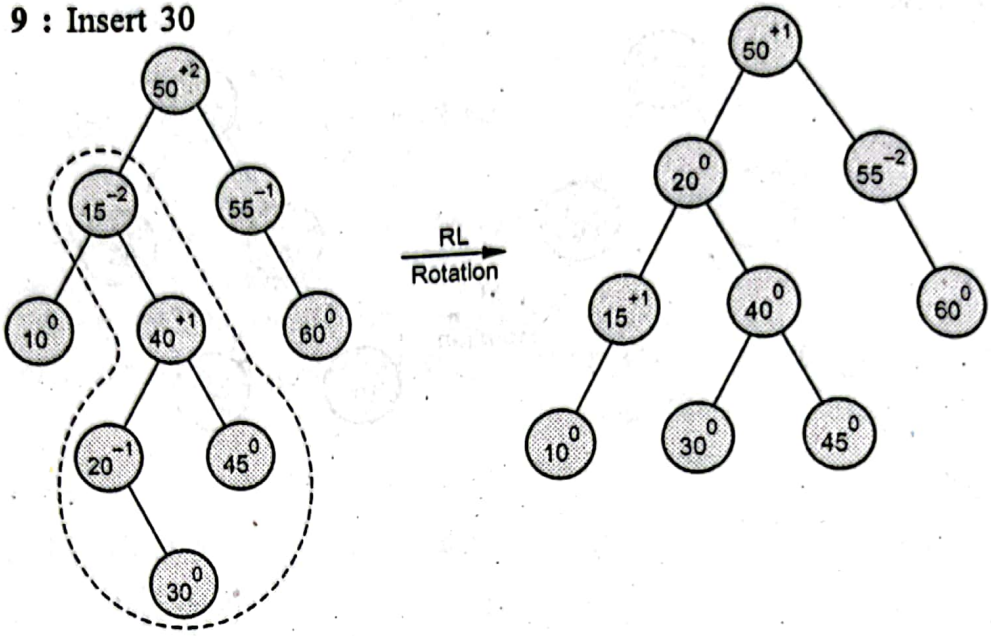
Step 7 : Insert 20



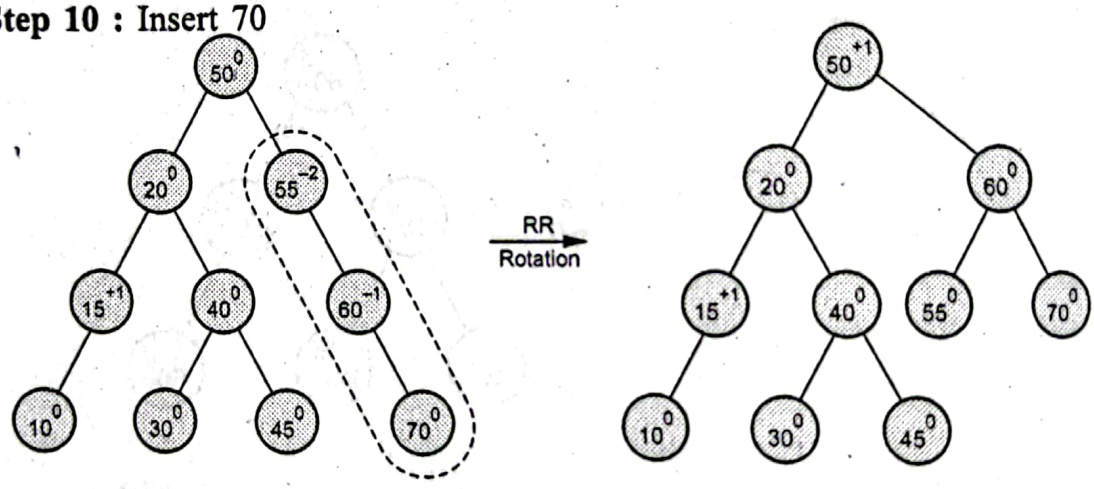
Step 8 : Insert 45



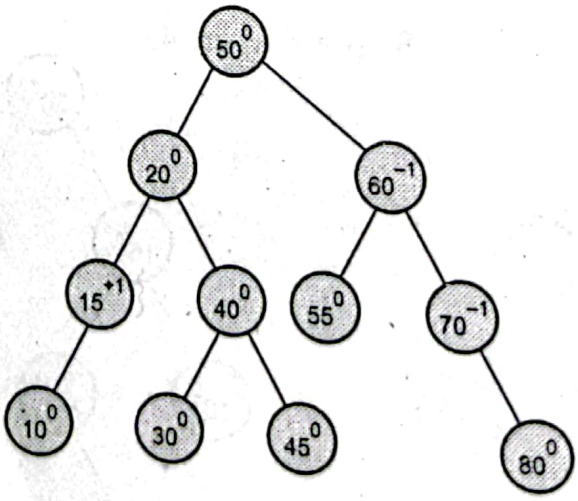
Step 9 : Insert 30



Step 10 : Insert 70



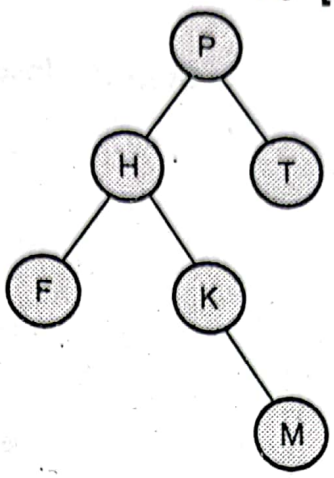
Step 11 : Insert 80



AVL tree

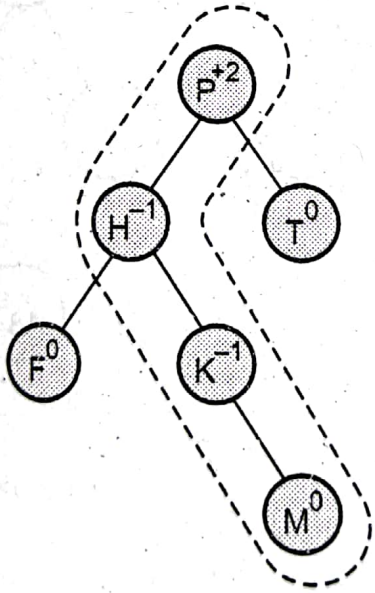
Q.46 Convert the following binary tree into AVL tree.

☞ **[SPPU : May-16, Marks 4]**



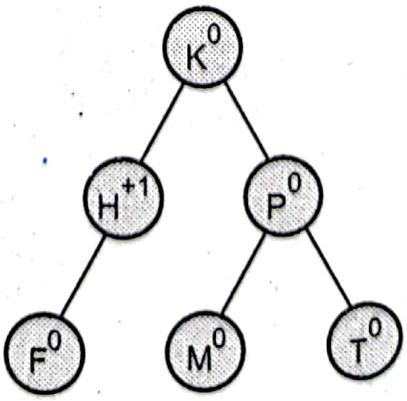
Ans. :

Step 1 : We will add balance factor to each node.



Due to balance factor = + 2 at node P, the rotation is required. The LR rotation is applicable here.

Step 2 :



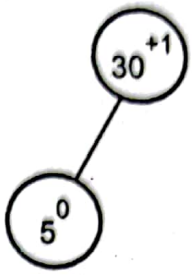
Balanced AVL tree

Q.47 Construct an AVL for the following data set 30, 5, 3, 18, 19, 4, 6, 35, 33, 15. ☞ [SPPU : Dec.-17, Marks 10]

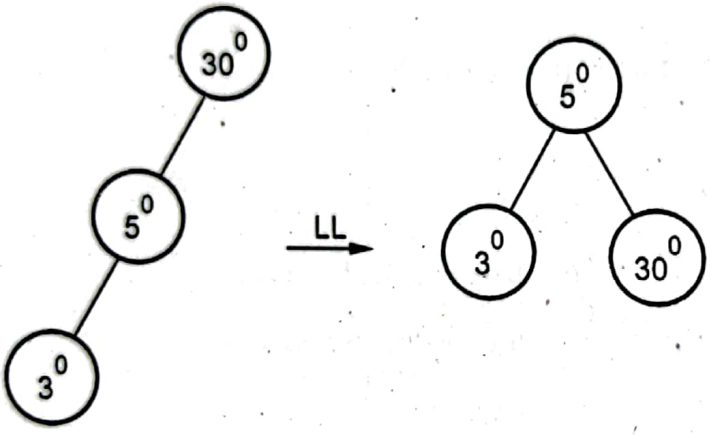
Ans. :

Step 1 : Insert 30

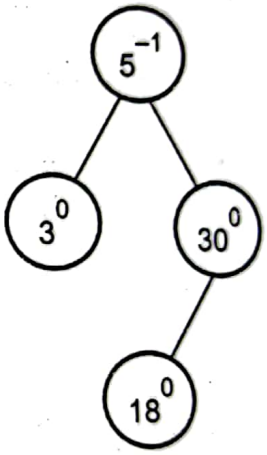
Step 2 : Insert 5



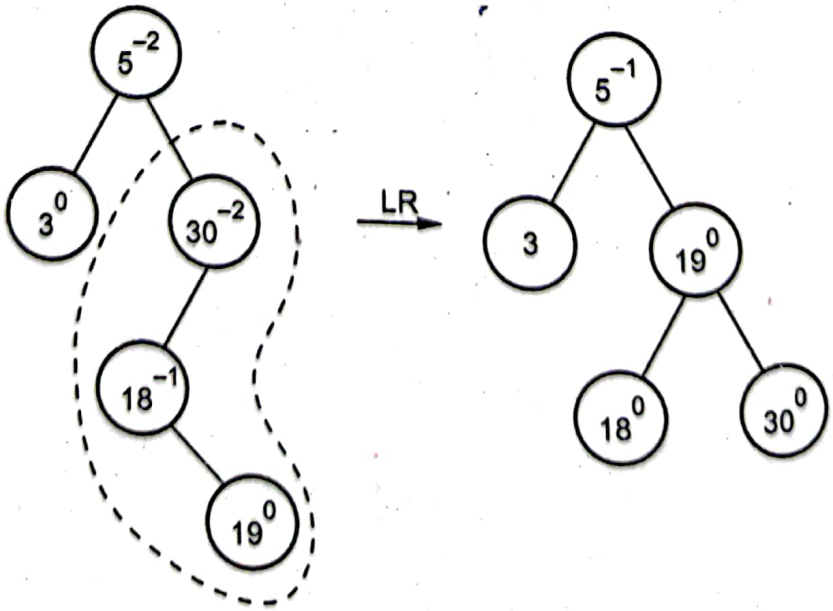
Step 3 : Insert 3



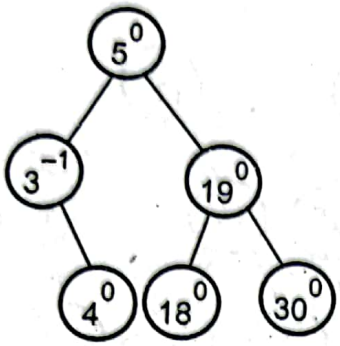
Step 4 : Insert 18



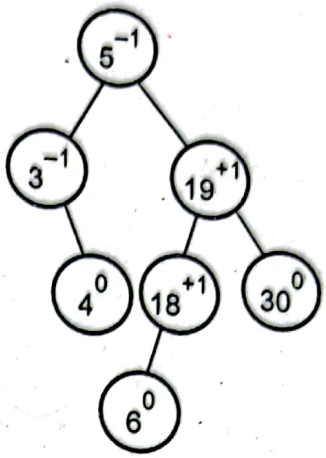
Step 5 : Insert 19



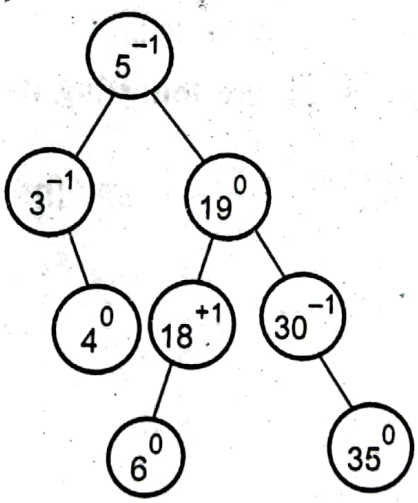
Step 6 : Insert 4



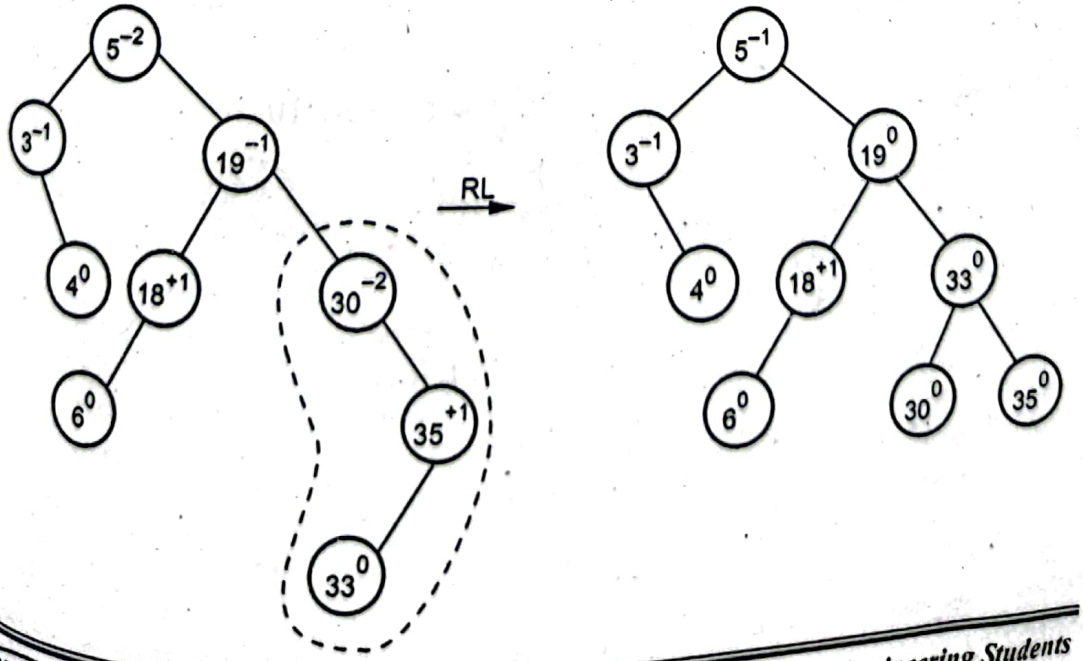
Step 7 : Insert 6



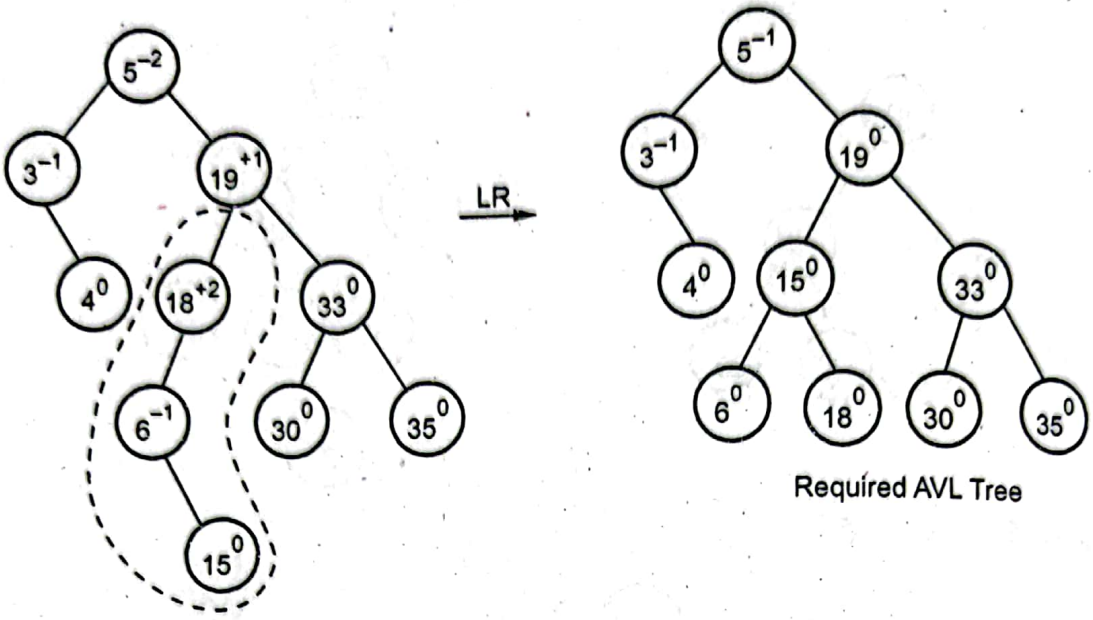
Step 8 : Insert 35



Step 9 : Insert 33



Step 10 : Insert 15



Required AVL Tree

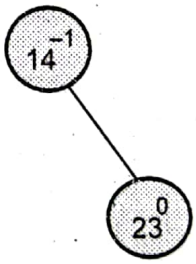
Q.48 Create an AVL tree using the following data, show the balance factor :

14, 23, 7, 10, 33, 56, 80, 66, 70

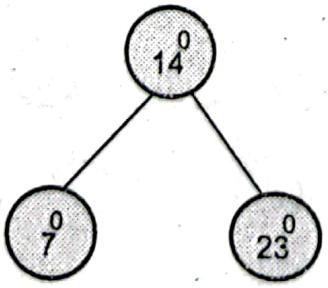
☞ **[SPPU : May-19, Marks 8]**

Ans. : Step 1 : Insert 14

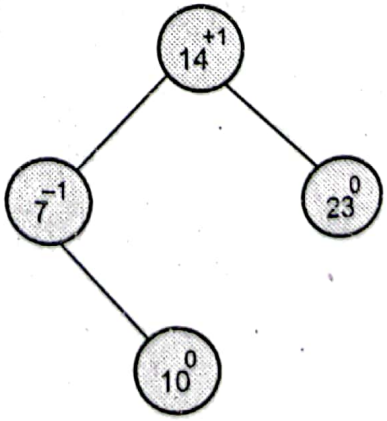
Step 2 : Insert 23



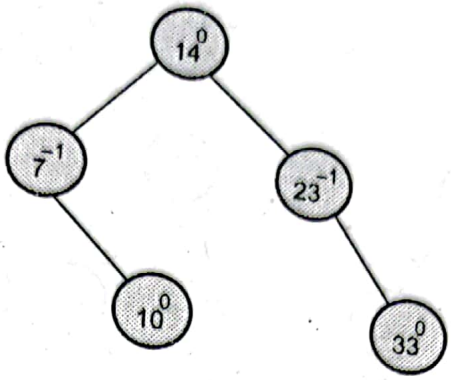
Step 3 : Insert 7



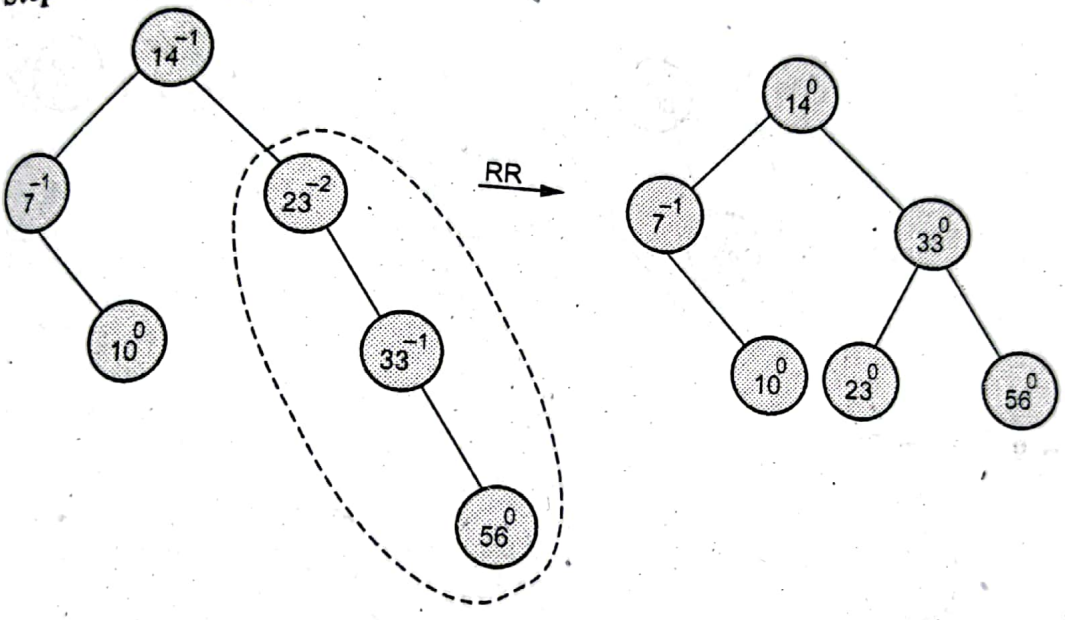
Step 4 : Insert 10



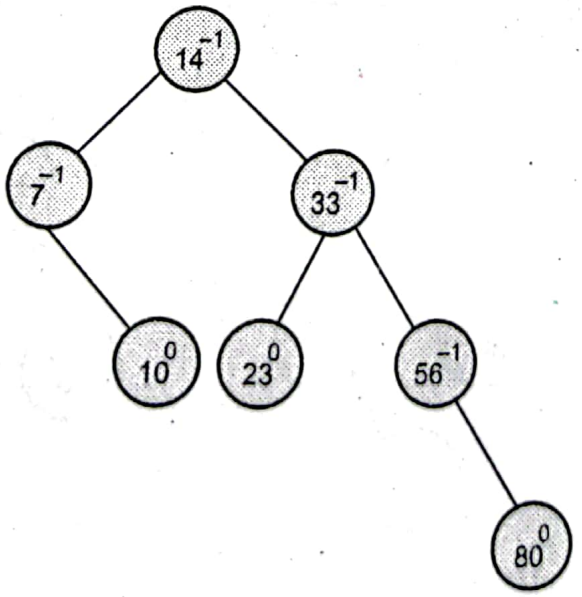
Step 5 : Insert 33



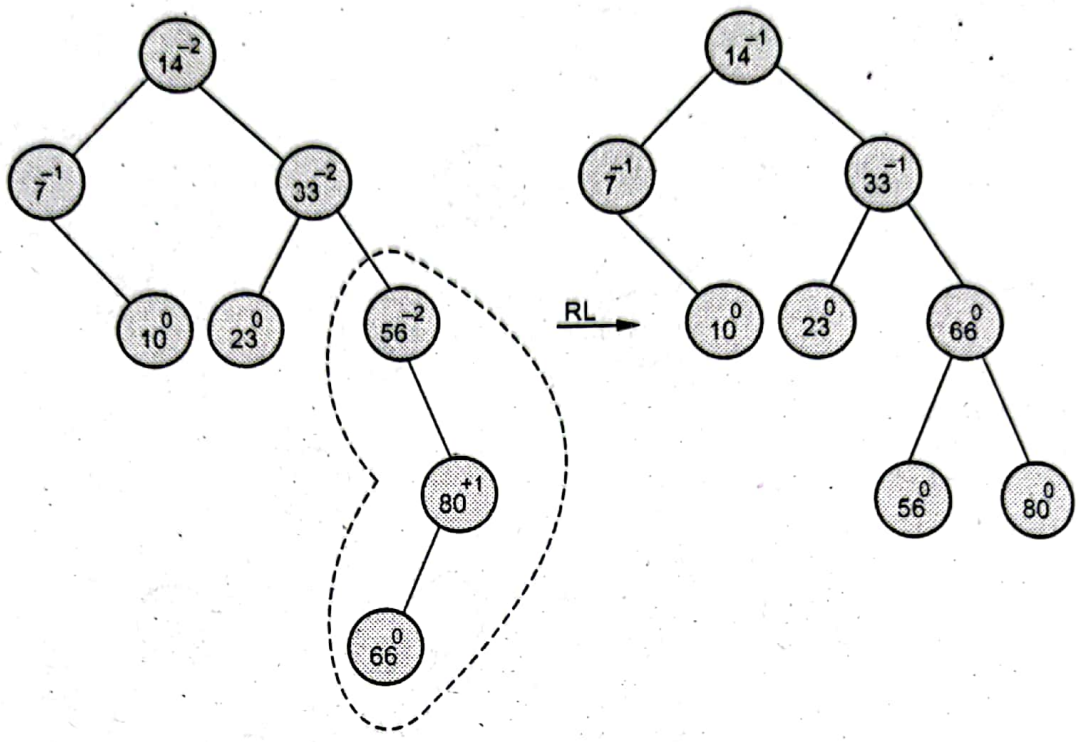
Step 6 : Insert 56



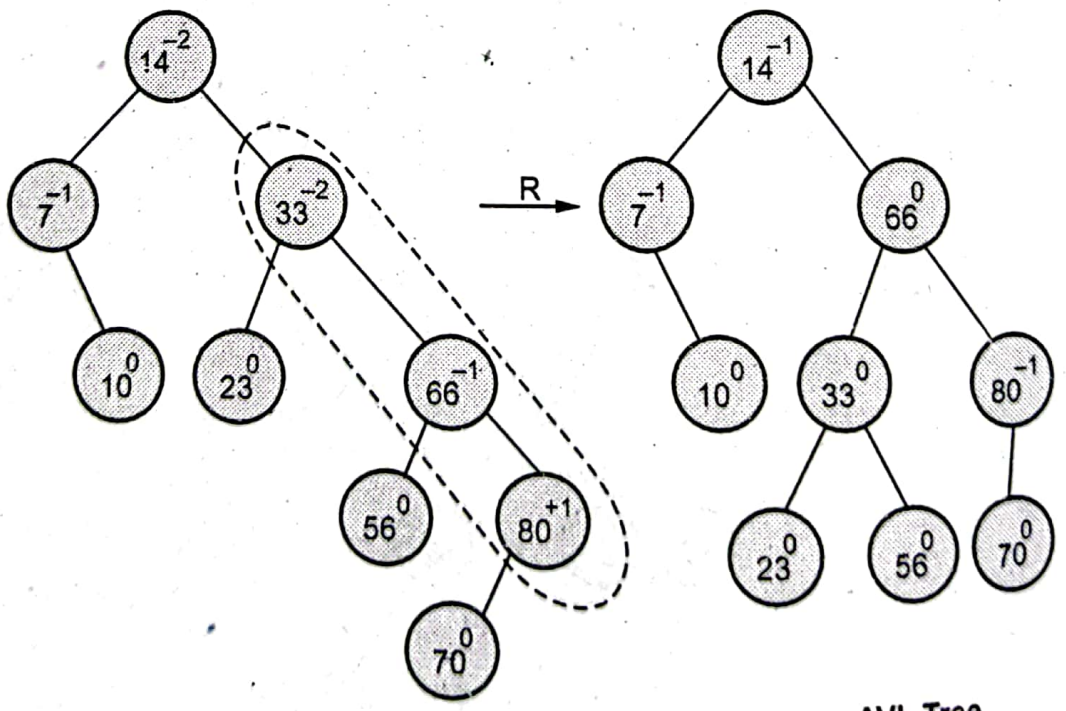
Step 7 : Insert 80



Step 8 : Insert 66



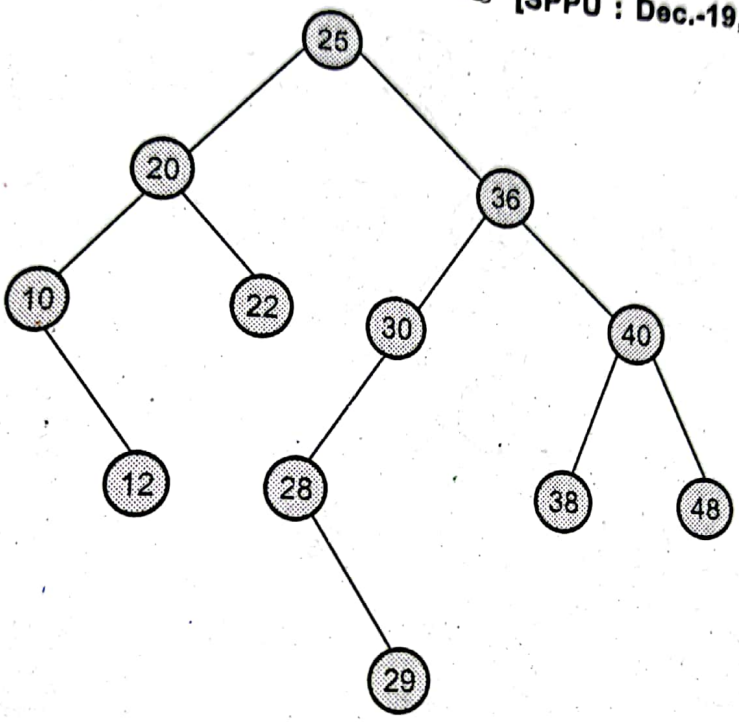
Step 9 : Insert 70



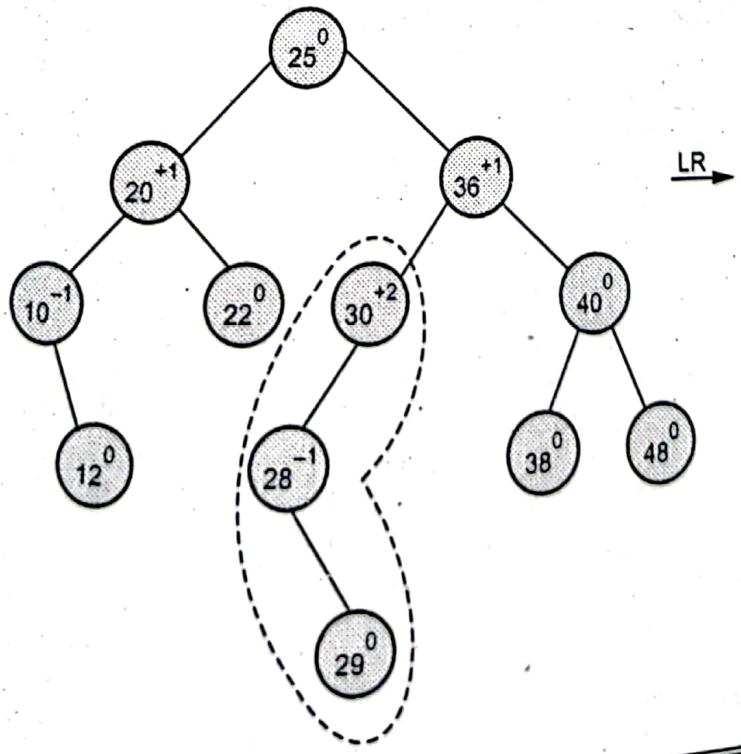
AVL Tree

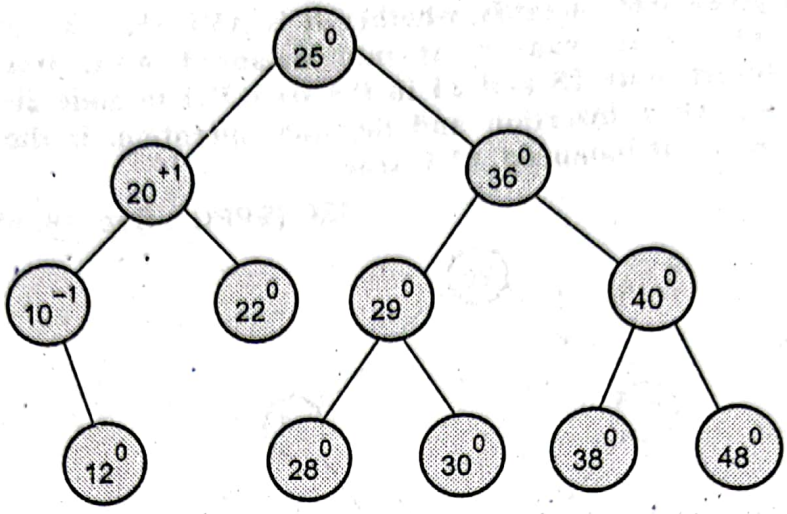
Q.49 For a given tree, identify whether it is AVL tree or not ? If it is not an AVL tree, convert it into balanced AVL tree. After conversion, insert node 15 and 24 in the tree. Delete node 20 and 22 from the tree. After insertion and deletion operation, if the tree is imbalanced, make it balanced AVL tree.

☞ **[SPPU : Dec.-19, Marks 10]**

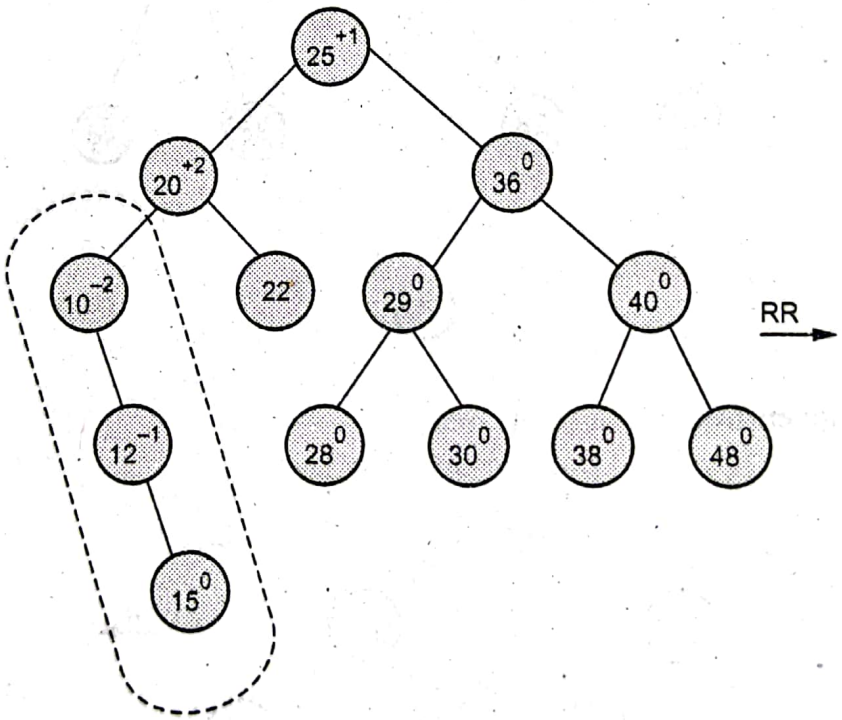


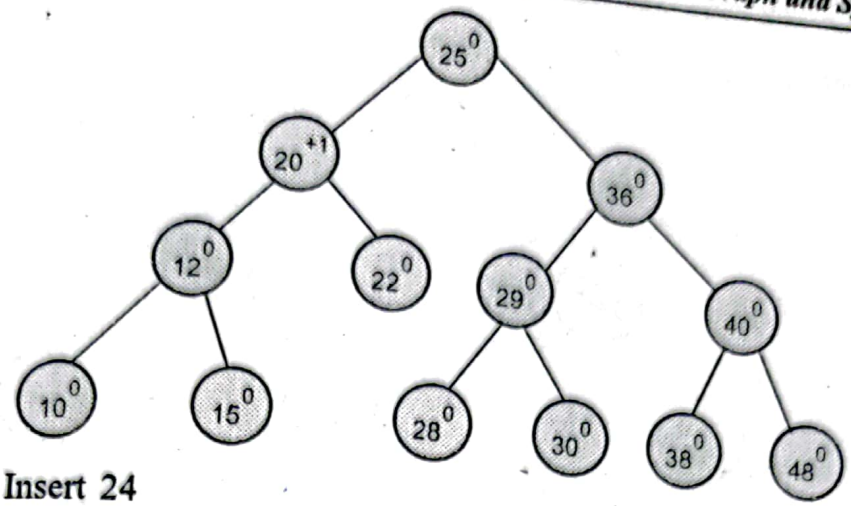
Ans. : The given tree is not AVL tree,



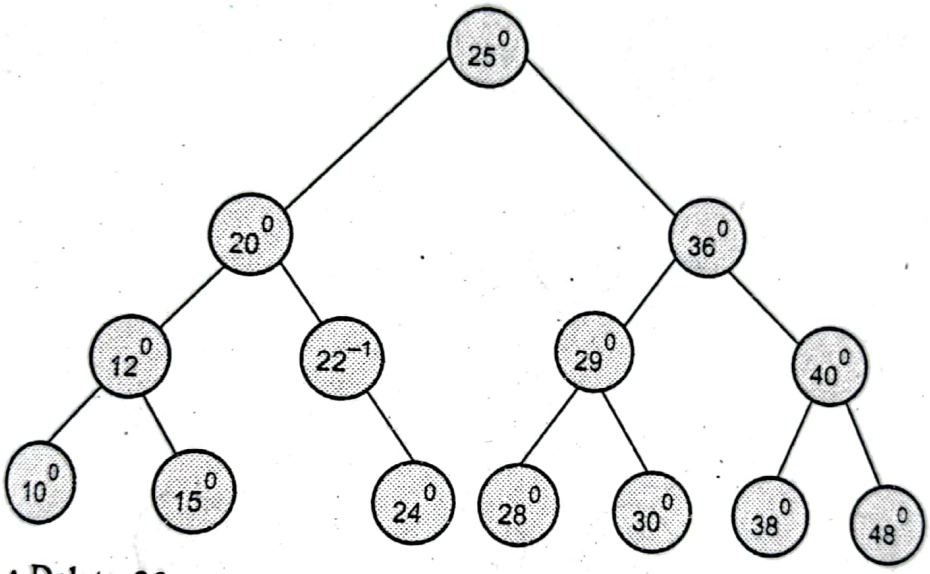


Step 1 : Insert 15

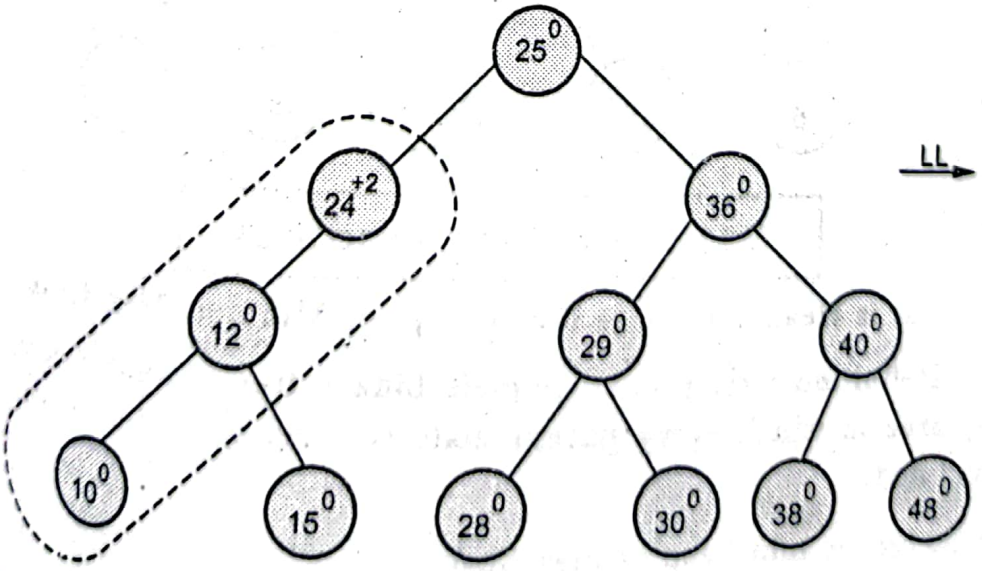




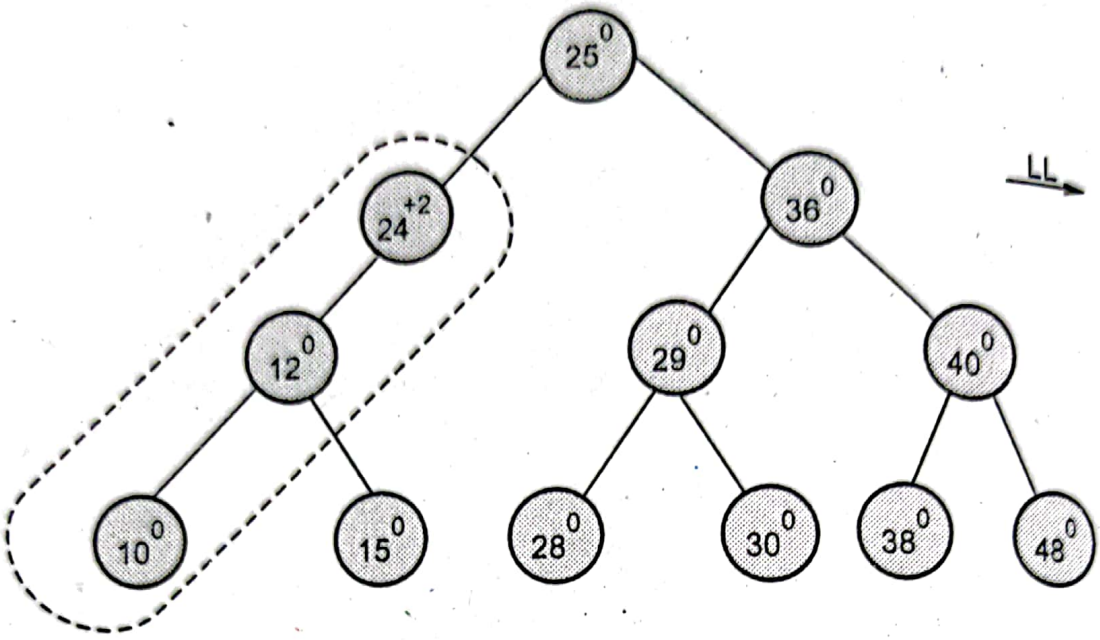
Step 2 : Insert 24

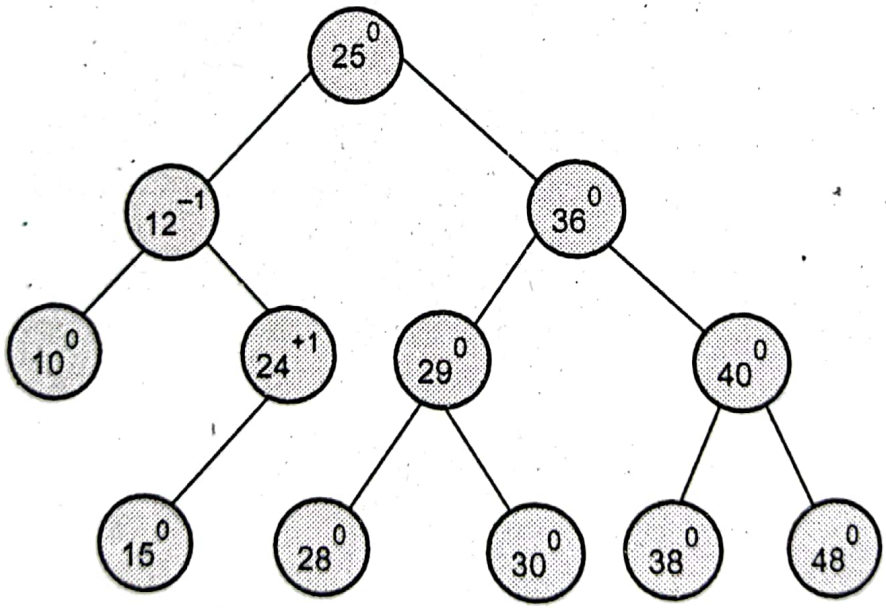


Step 3 : Delete 20



Step 4 : Delete 22





5.11 : Heap Data Structures

Q.50 What is Heap ? Explain the concept of Min and Max Heap.

Ans. : Definition : Heap is a complete binary tree or almost complete binary tree in which every parent node is either greater or lesser than its child nodes.

- Heap can be **min heap** or **max heap.**

- A **Max heap** is a tree in which value of each node is greater than or equal to the value of its children nodes.

For example :

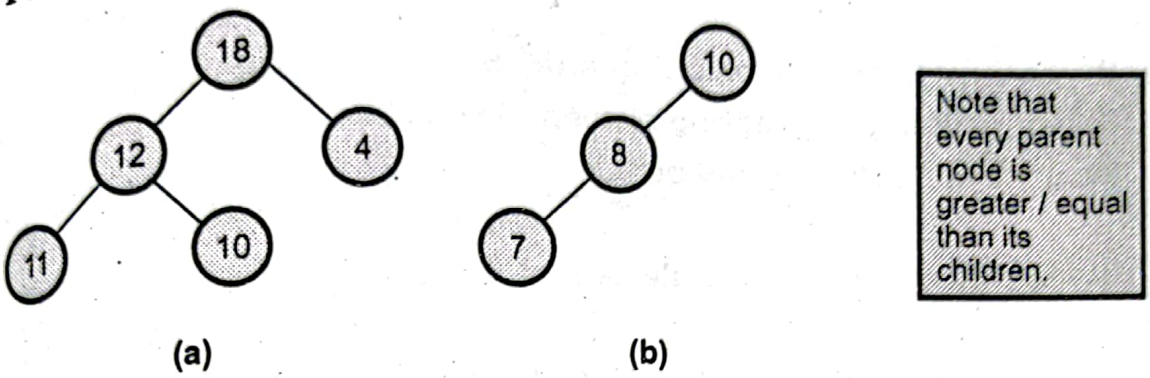


Fig. Q.50.1 Max heap

- A **Min heap** is a tree in which value of each node is less than or equal to value of its children nodes.

For example :

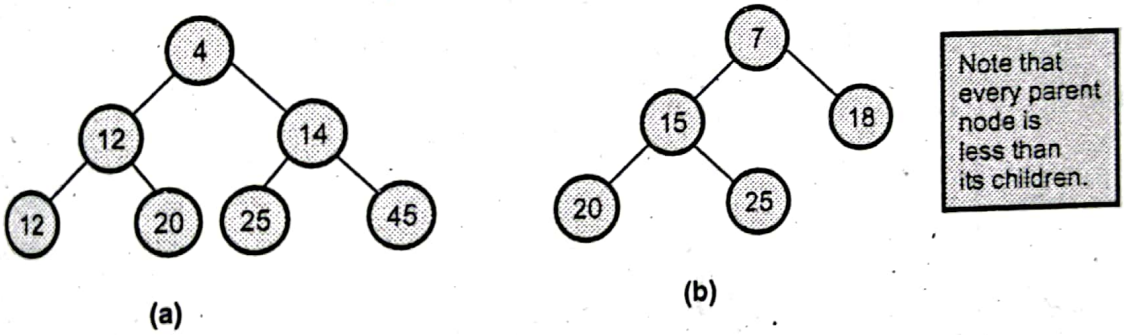


Fig. Q.50.2 Min heap

Q.51 Write an algorithm for insertion of element in heap.

Ans.: 1. Insert element at the last position in heap.

2. Compare with its parent, swap the two nodes if parental dominance condition gets violeted.

   [Parental dominance = Parent is greater than its children]

3. Continue comparing the new element with nodes up, each time by satisfying parental dominance condition.

```
void Heap::insert(int item)
{
int temp;
```

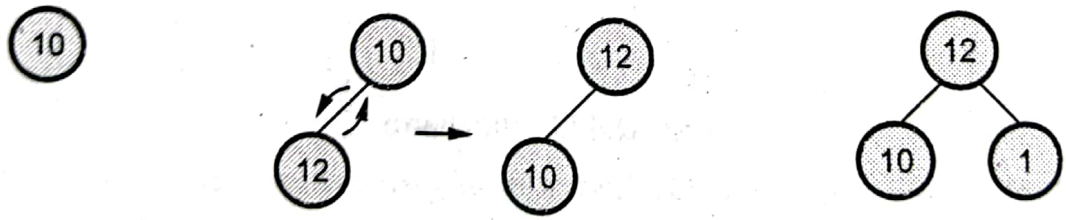
```
//temp node starts at leaf and moves up
 temp=++size;
 while(temp!=1 && heap[temp/2]<item)
 {
// the element cannot be placed in array H
  H[temp]=H[temp/2];//moving element down
   temp=temp/2;//finding the parent
  }
  H[temp]=item;//placing the element at its position
}
```

Q.52 Show the result of inserting 10, 12, 1, 14, 6, 5, 8, 15, 3, 9, 7, 4, 11, 13 and 2 at a time, into initially empty binary heap. After creating such heap delete the element 8 from heap, how do you repair the heap? Then insert the element in the heap and show the final result (insertion should be at other than lead node).

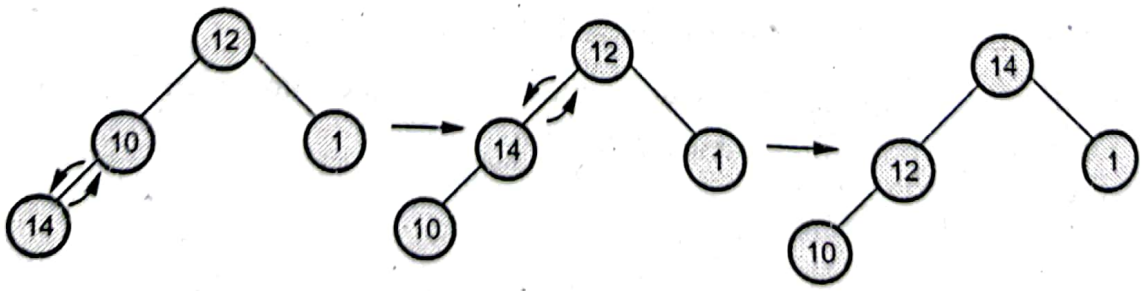
Ans. : We will create max heap for the set

10 12 1 14 6 5 8 15 3 9 7 4 11 13 2

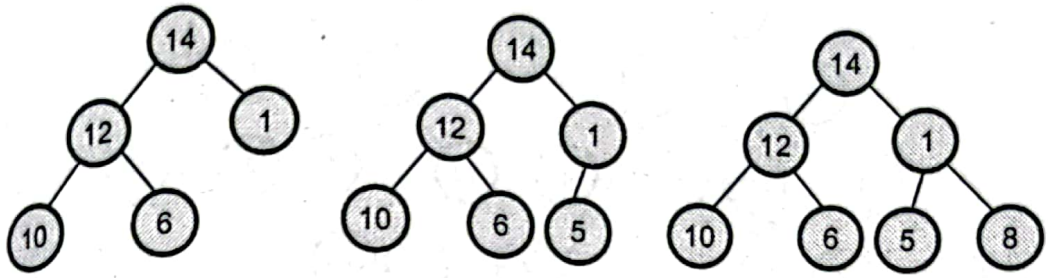
Step 1 : Insert 10. **Step 2 :** Insert 12. **Step 3 :** Insert 1.



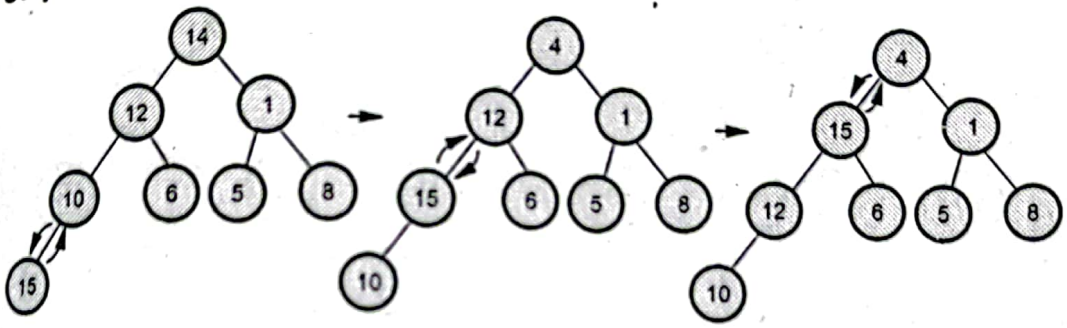
Step 4 : Insert 14.

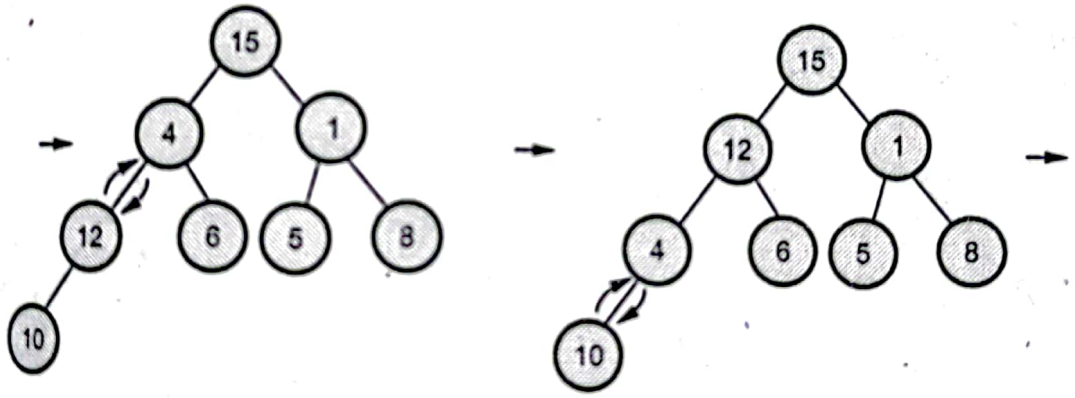


Step 5 : Insert 6. **Step 6 :** Insert 5. **Step 7 :** Insert 8.

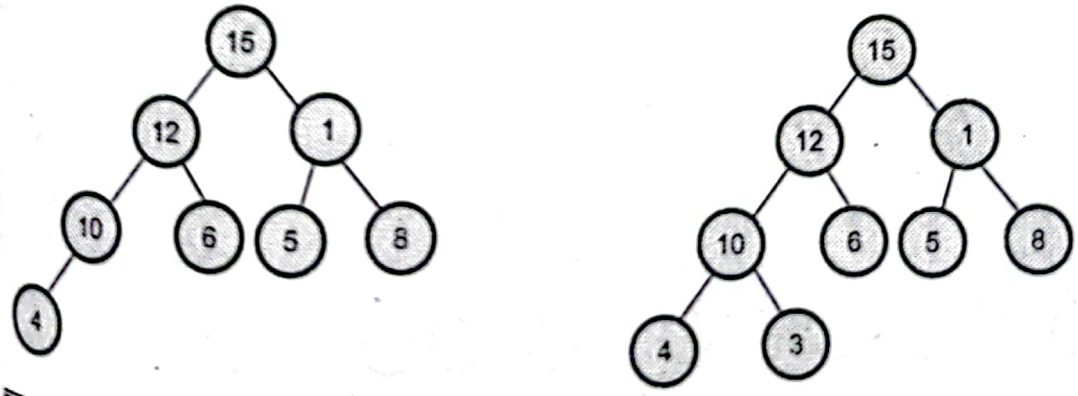


Step 8 : Insert 15

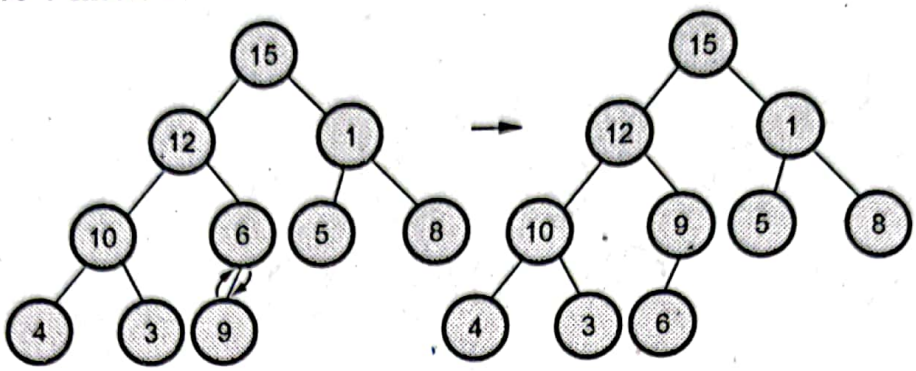




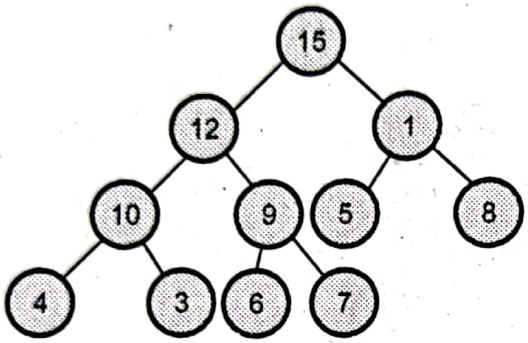
Step 9 : Insert 3.



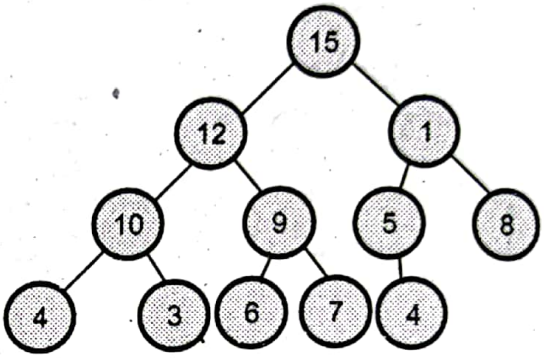
Step 10 : Insert 9.



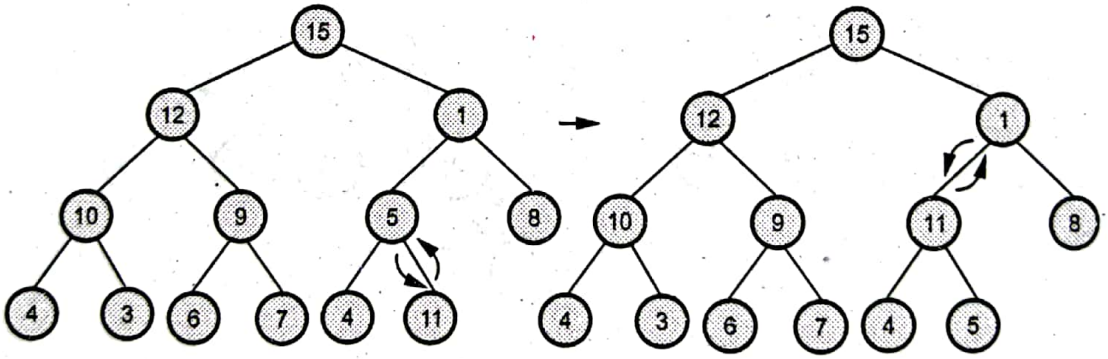
Step 11 : Insert 7.

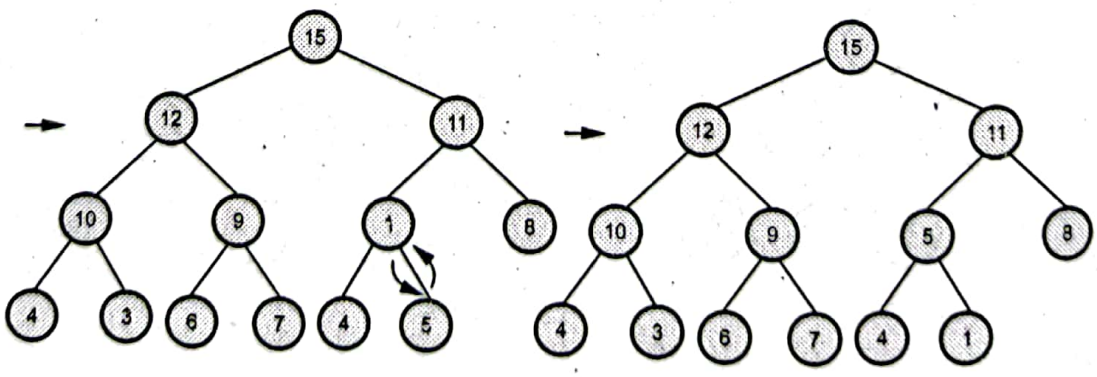


Step 12 : Insert 4.

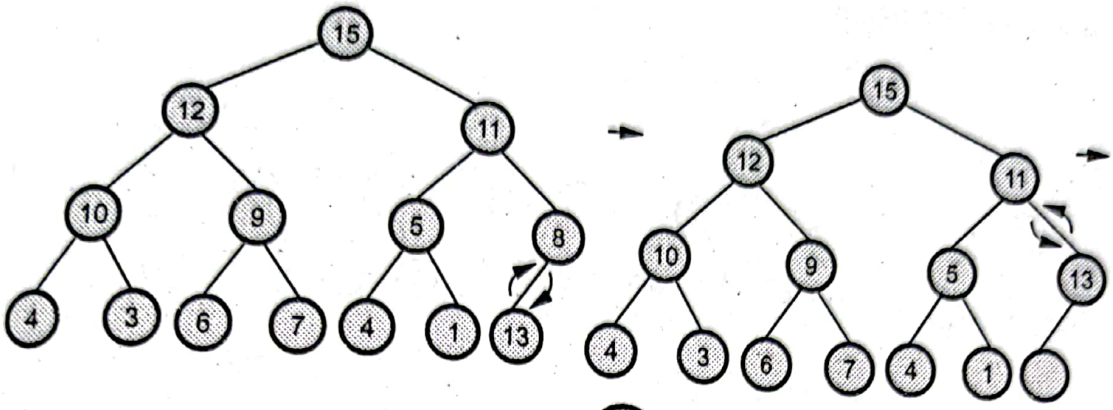


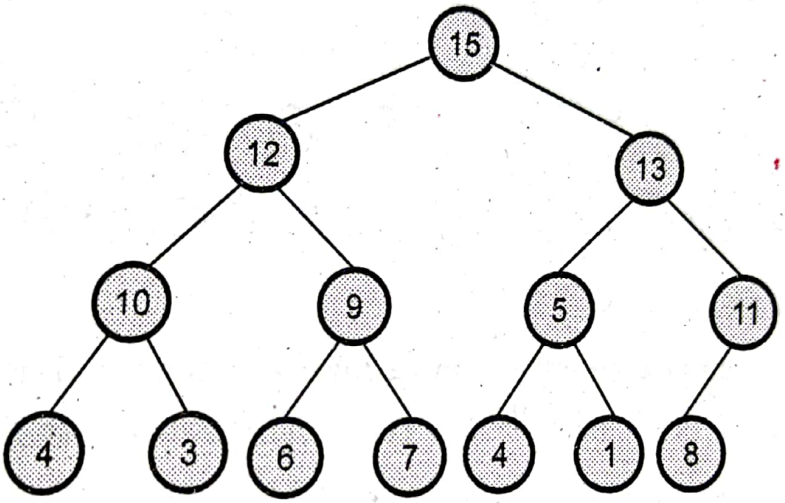
Step 13 : Insert 11.





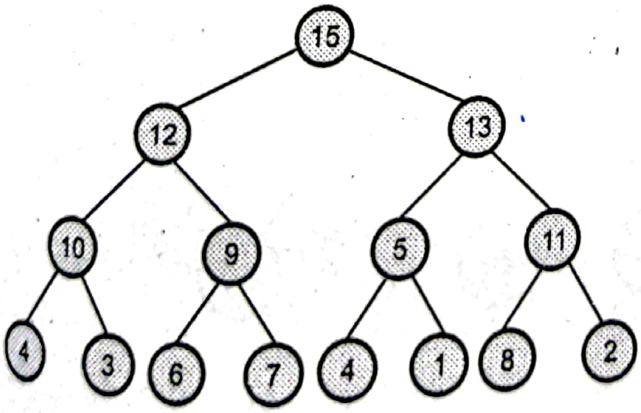
Step 14 : Insert 13.

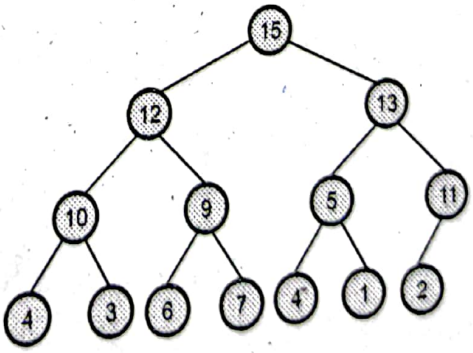




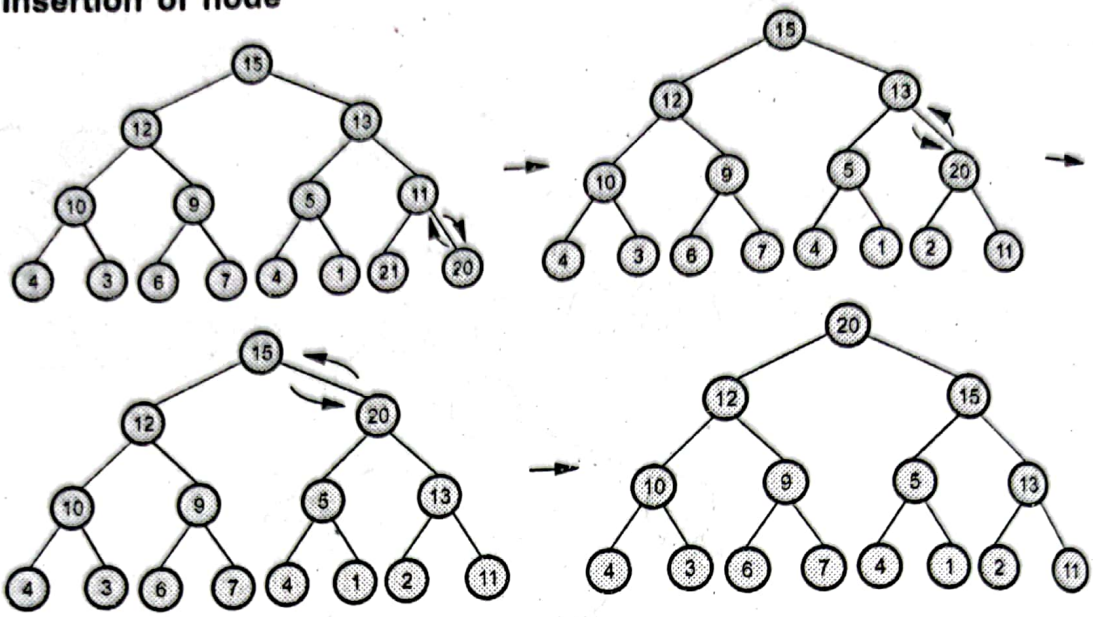
Step 15 : Insert 2

Deletion of 8





Insertion of node

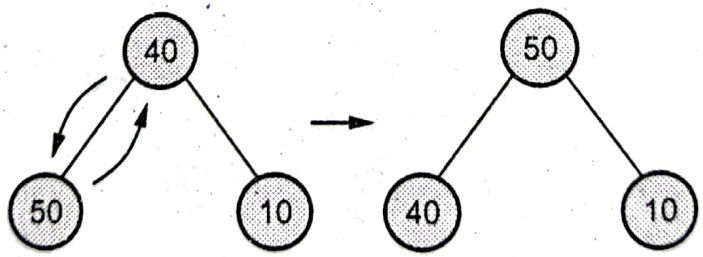


If we insert node 20 then,

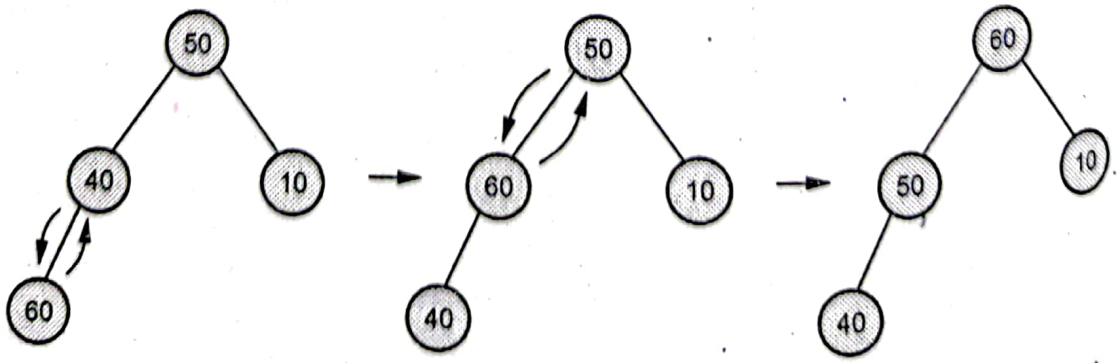
Q.53 Show stepwise construction of maxheap for the data :
40, 50, 10, 60, 20, 30, 70 **☞[SPPU : May-16, Marks 5]**

Ans.: Max heap is a complete or almost complete binary tree having parent node greater than its child nodes.

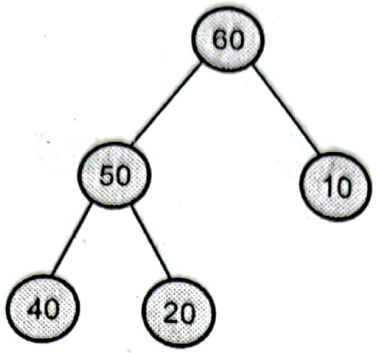
Step 1 :



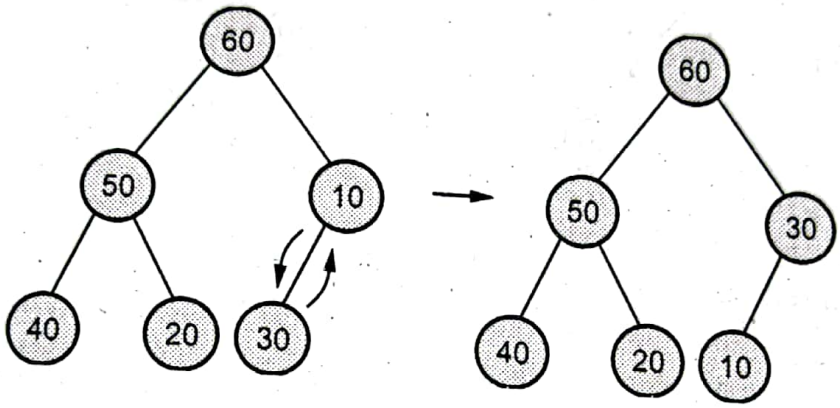
Step 2 :



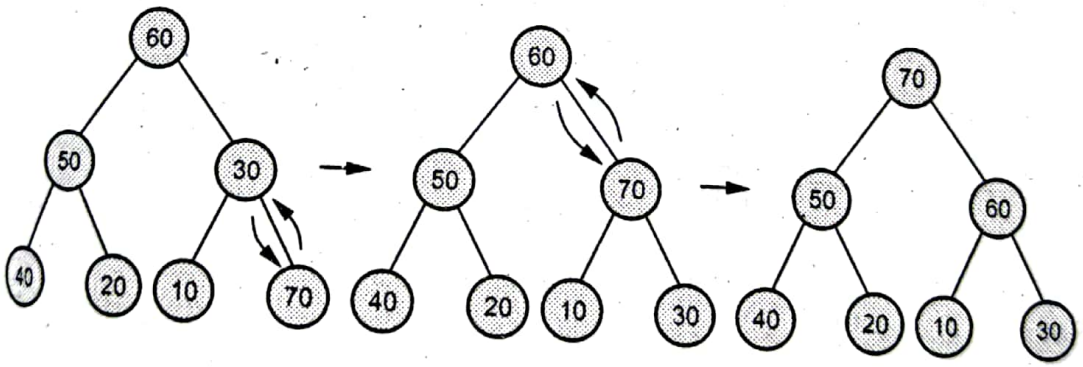
Step 3 :



Step 4 :



Step 5 :



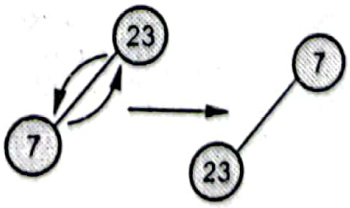
Max heap

Q.54 Construct heap out of following data read from the keyboard : 23, 7, 92, 6, 12, 14, 40, 44, 20, 21 ☞ **[SPPU : May-19, Marks 8]**

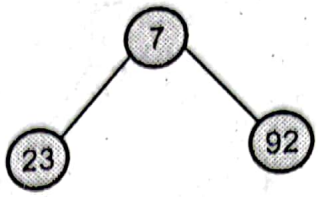
Ans. : We will build a min-heap structure.

Step 1 : Insert 23

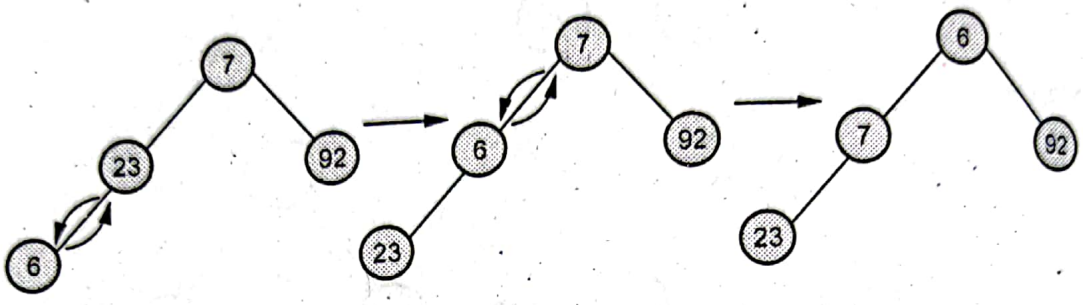
Step 2 : Insert 7



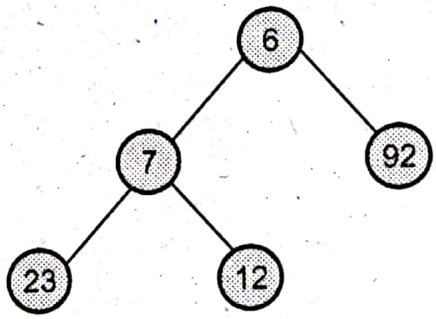
Step 3 : Insert 92



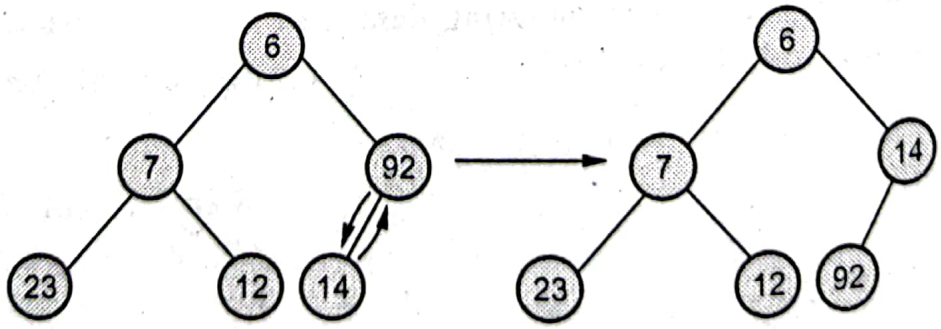
Step 4 : Insert 6



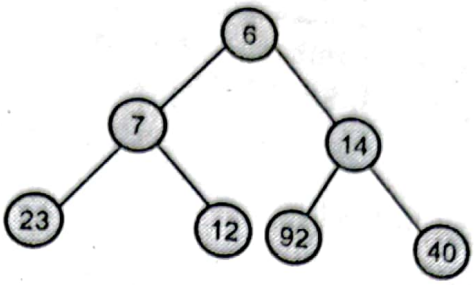
Step 5 : Insert 12



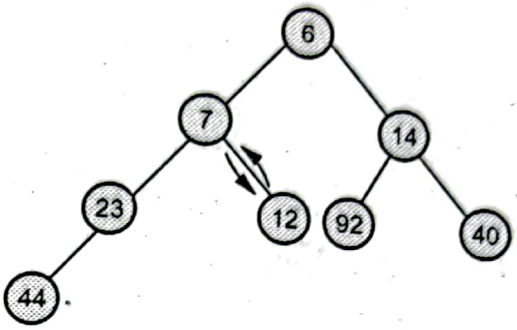
Step 6 : Insert 14



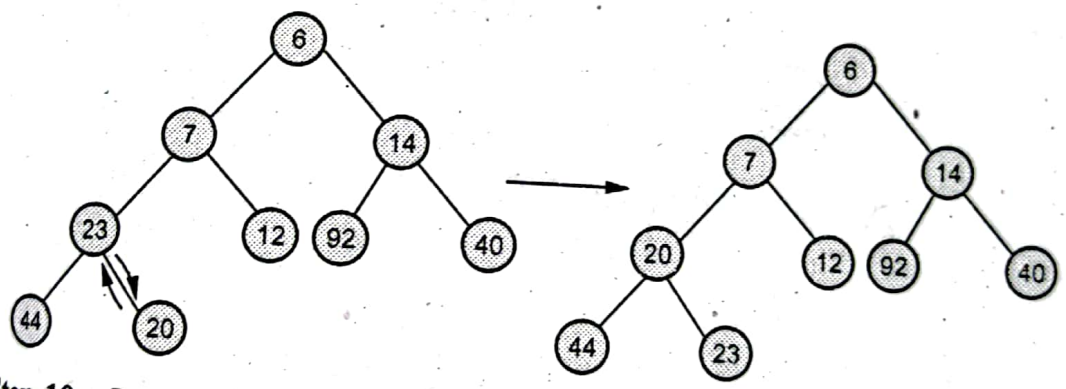
Step 7 : Insert 40



Step 8 : Insert 44

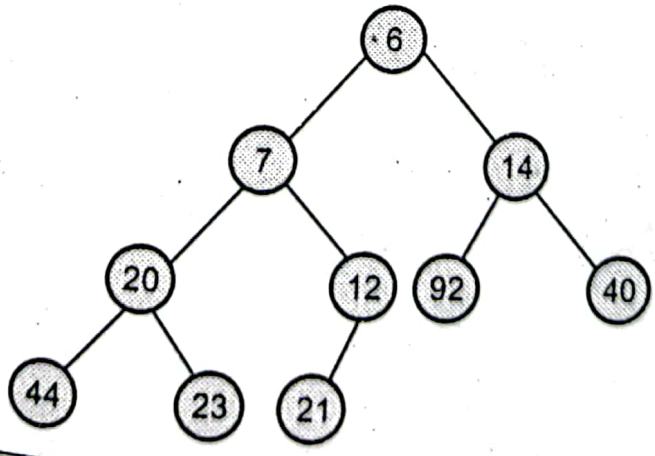


Step 9 : Insert 20



Step 10 : Insert 21

This is required heap structure.



5.12 : Heap Sort

Q.55 Sort the following number using heap sort and show the sorting stepwise : 44, 66, 33, 88, 77, 55, 22.

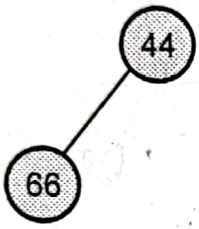
☞[SPPU : Dec.-14, Marks 6]

Ans.: Stage I : Heap construction

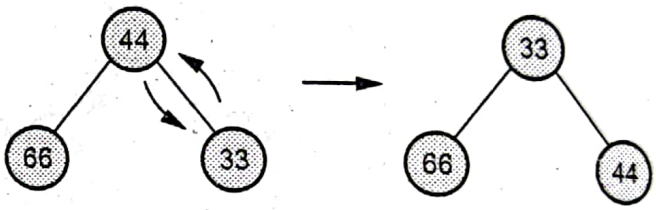
We will construct min-heap.

Step 1 :

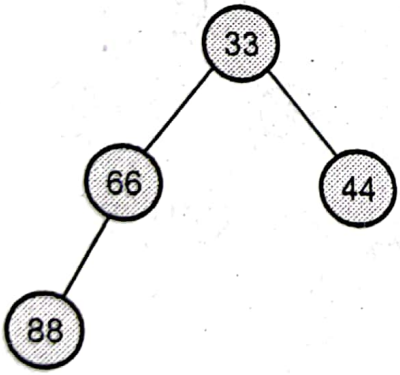
Step 2 :



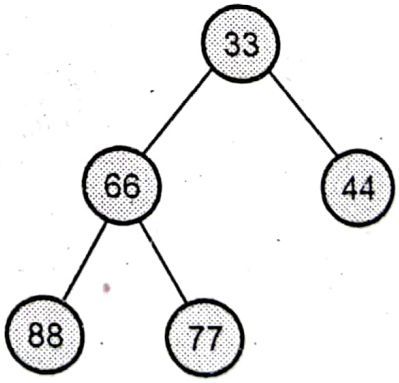
Step 3 :



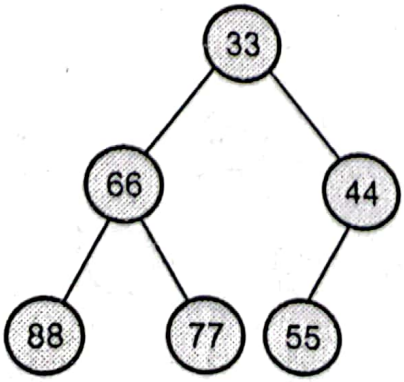
Step 4 :



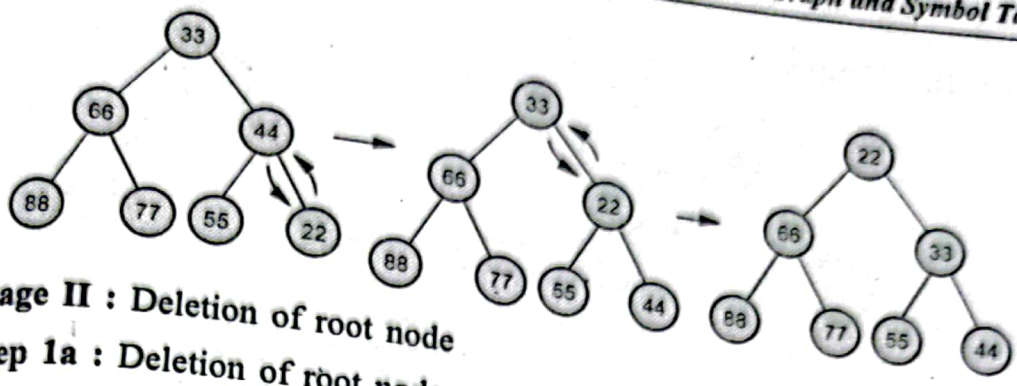
Step 5 :



Step 6 :

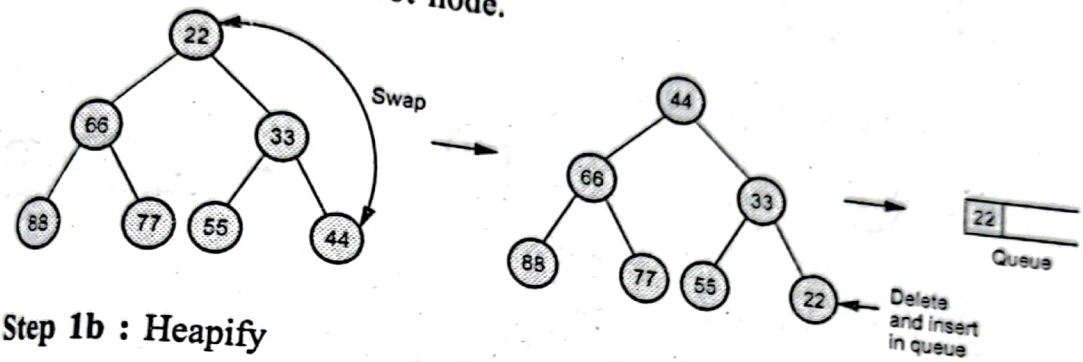


Step 7 :

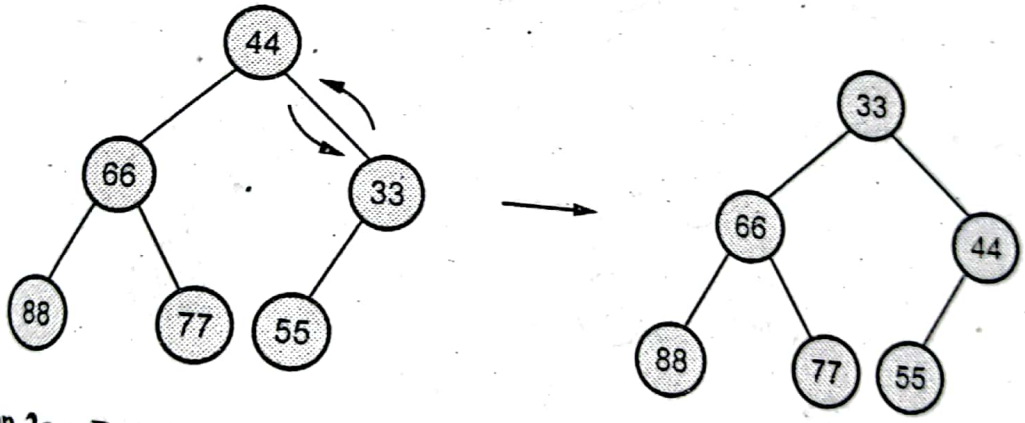


Stage II : Deletion of root node

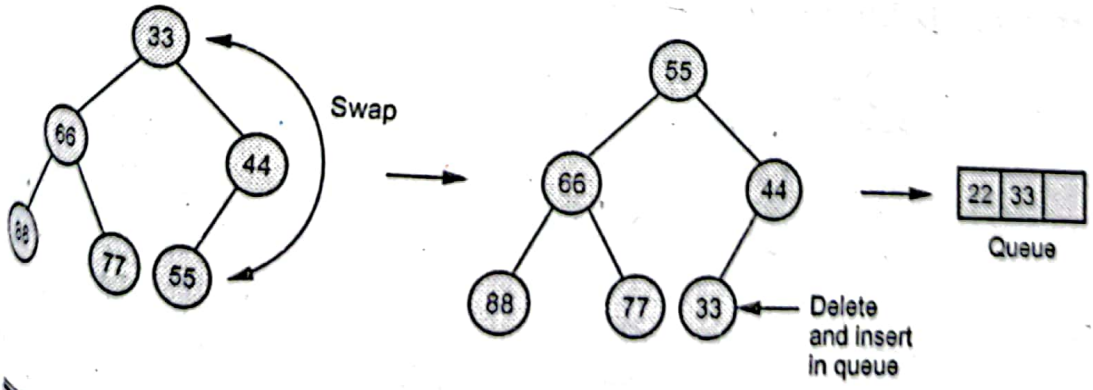
Step 1a : Deletion of root node.



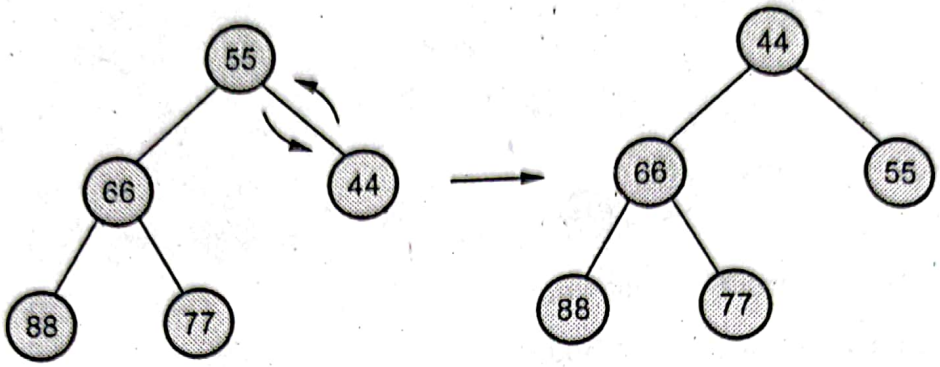
Step 1b : Heapify



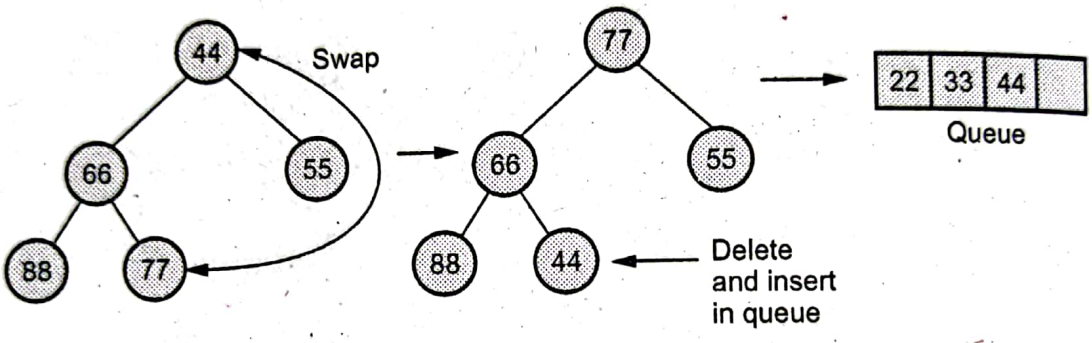
Step 2a : Deletion of root node



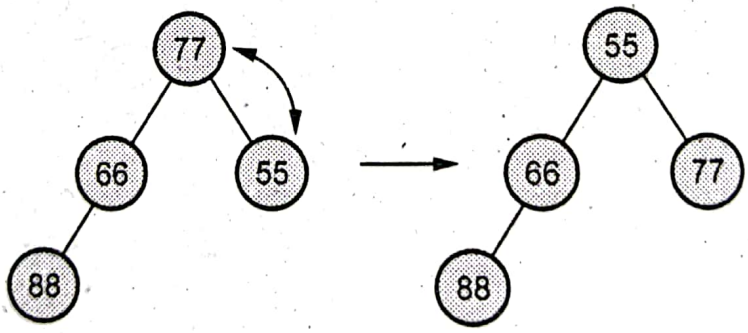
Step 2b : Heapify



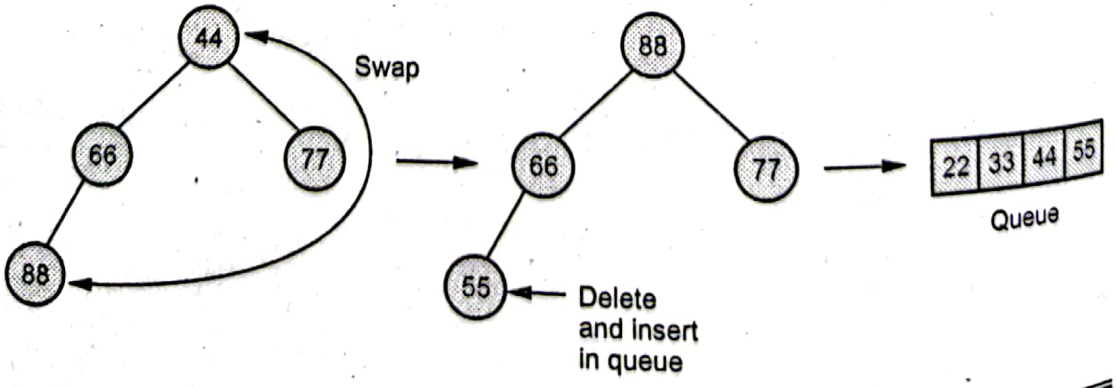
Step 3a : Deletion of root node



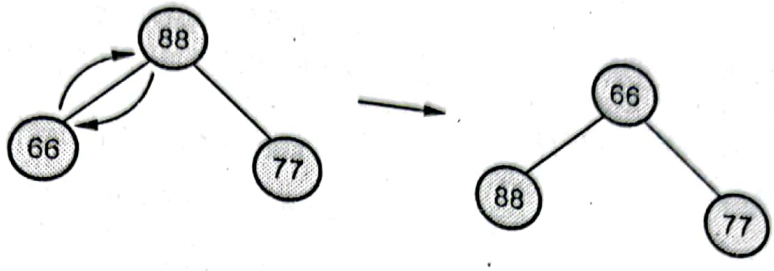
Step 3b : Heapify



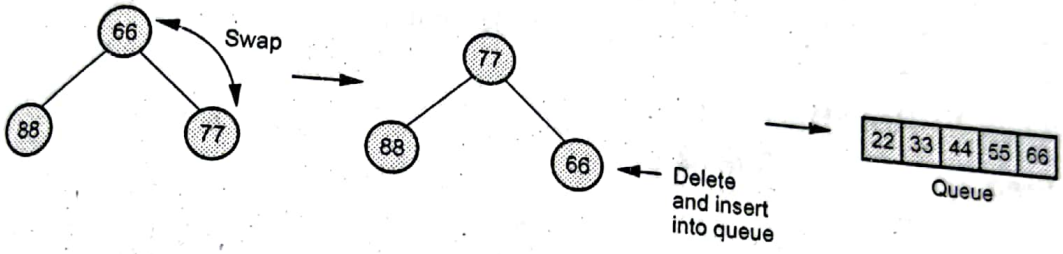
Step 4a : Deletion of root node



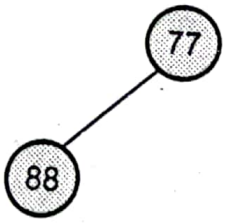
Step 4b : Heapify



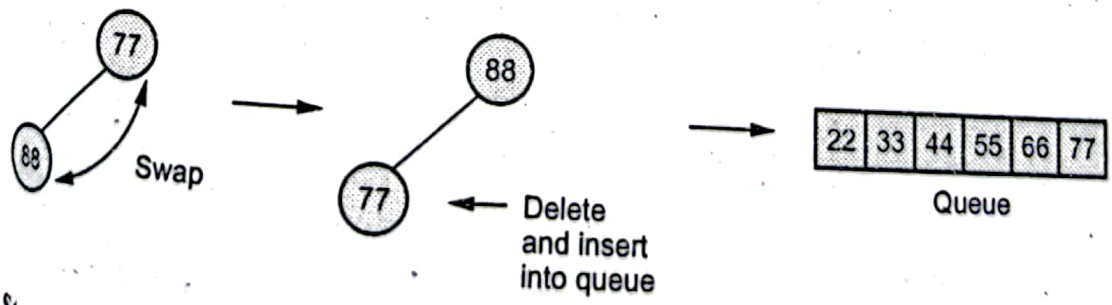
Step 5a : Deletion of root node



Step 5b : Heapify



Step 6a : Deletion of root node



Step 6b : Heapify

- As there is only one node remaining, delete it and insert in queue. Finally, we will get sorted elements in queue as

22	33	44	55	66	77	88

Q.56 Which data structures supports to perform sorting using heap data structure ? Explain their use in detail for the following list to sort it in ascending order :
1, 12, 9, 5, 6, 10 ☞ **[SPPU : Dec.-19, Marks 8]**

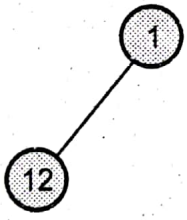
Ans. : The priority queue is used to perform sorting using heap data structure.

In the following example we will create a min heap structure. We will each time delete a root node from this heap structure and place it in priority queue to form a sorted list of elements.

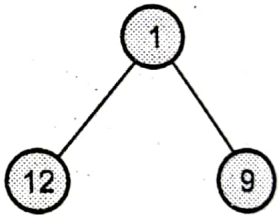
Stage I : Heapify

Step 1 :

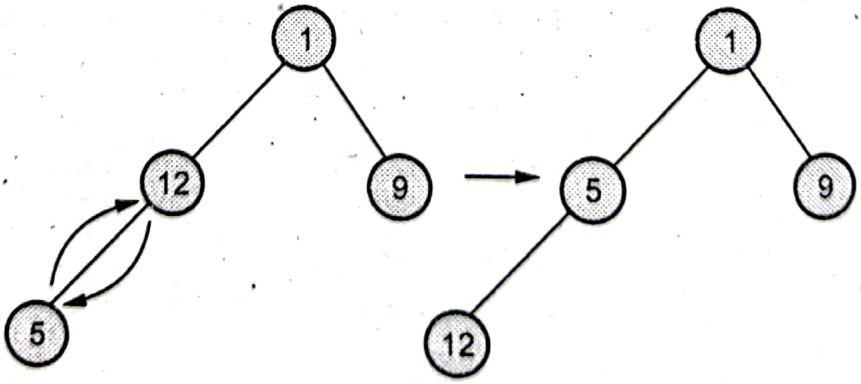
Step 2 :



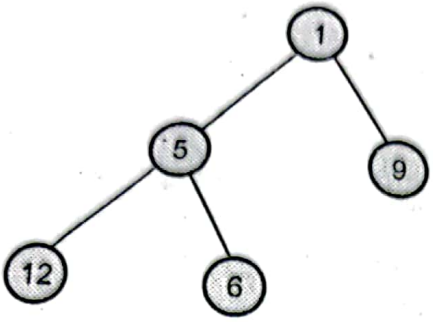
Step 3 :



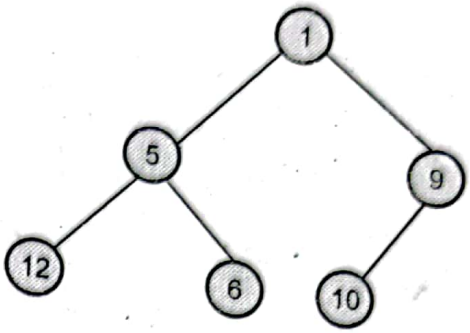
Step 4 :



Step 5 :

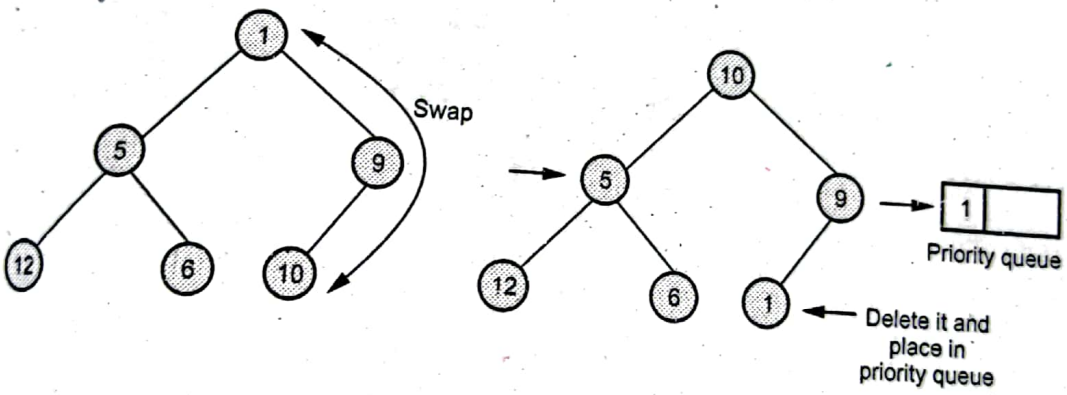


Step 6 :

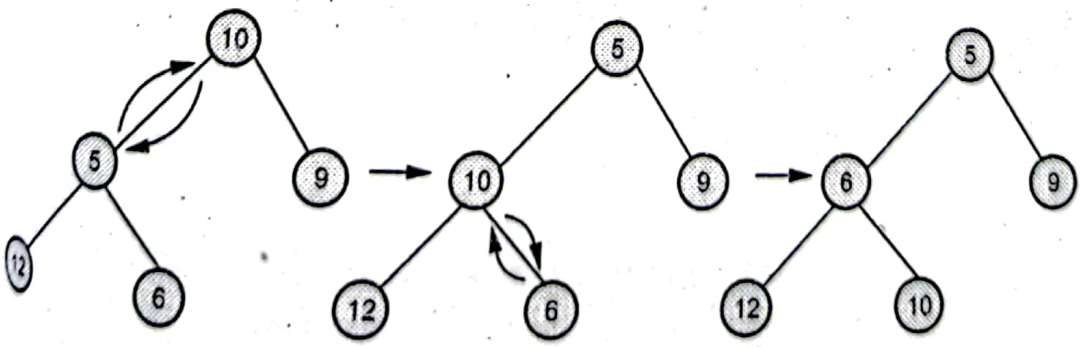


Stage II : Deletion of root

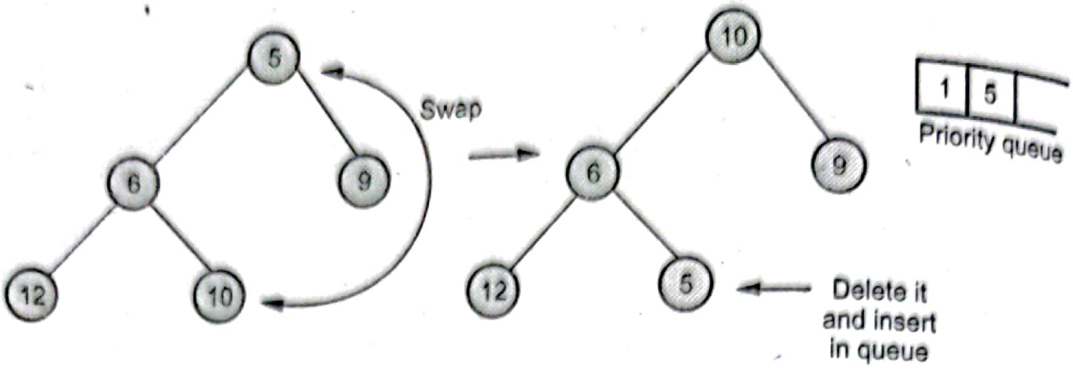
Step 1 :



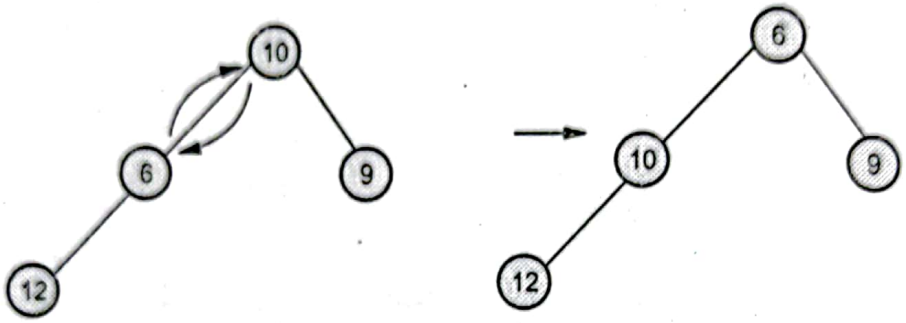
Heapify for creating min heap.



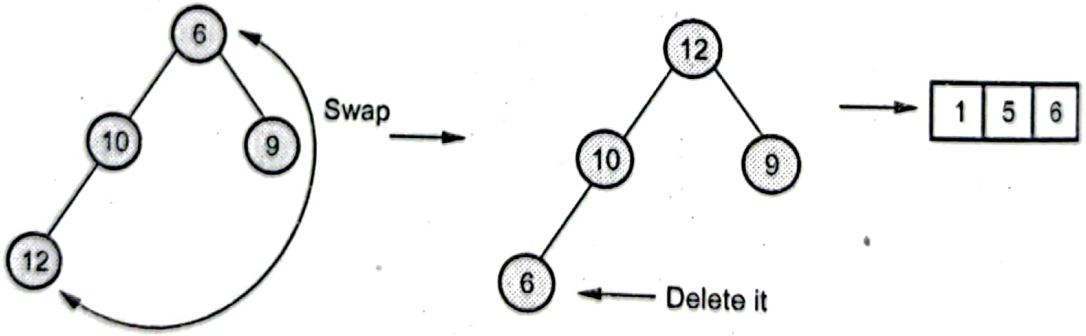
Step 2 :



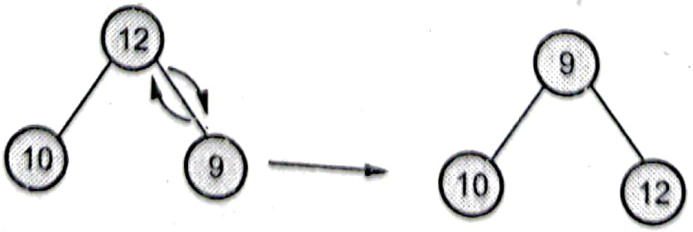
Heapify



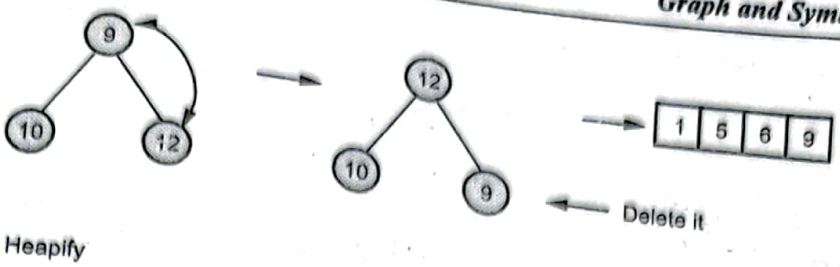
Step 3 :

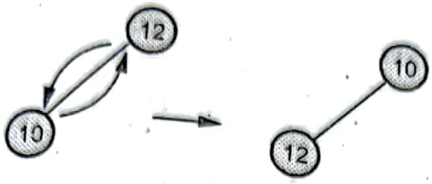


Heapify

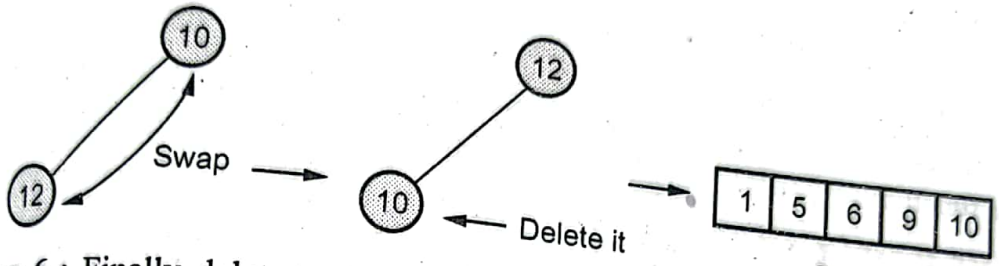


Step 4 :





Step 5 :



Step 6 : Finally delete node 12 and insert it into priority queue. Thus we get a list in ascending order in a priority queues as follows,

1	5	6	9	10	12

5.13 : Applications of Heap

2.57 What are different applications of heap data structure ?

Ans. : Various applications of heap are

Heap sort method is based on heap data structure. This is one of the best sorting methods.

Heap is used in graph algorithms such as finding minimal spanning tree using Prim's algorithm and Dijkstras shortest path algorithm.

Heap allows access to min or max element in constant time. Thus heap is used in selection algorithms.

END...