Unit III

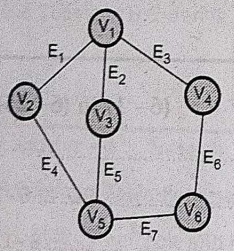
3 Graphs

3.1 : Basic Concepts

Q.1 Define the term - 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 What is the difference between graph and tree?

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.

Q.3 Explain the concept of directed and undirected graph.

Ans. : 1. Directed graph

2. Undirected graph

- In the directed graph the **directions** are shown on the edges. As shown in the Fig. Q.3.1, the edges between the vertices are ordered. In this type of graph, the edge E_1 is in between the vertices V_1 and V_2. The V_1 is called head and the V_2 is called the tail. Similarly for V_1 head the tail is V_3 and so on.
- We can say E_1 is the set of (V_1, V_2) and not of (V_2, V_1).
- Similarly, in an undirected graph, the **edges are not ordered**. Refer Fig. Q.3.2. In this type of graph the edge E_1 is set of (V_1, V_2) or (V_2, V_1).

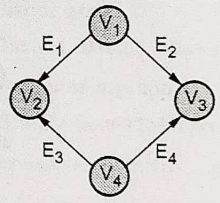


Fig. Q.3.1 Directed graph

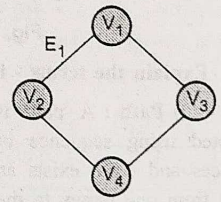


Fig. Q.3.2 Undirected graph

Q.4 Define the following :
i) A complete graph ii) A weighted graph.

[SPPU : May-10, Marks 4]

Ans. : i) A 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.4.1 is a complete graph.

Here $n = 4$

$\therefore e = \frac{n(n-1)}{2} = \frac{4(3)}{2} = 6$

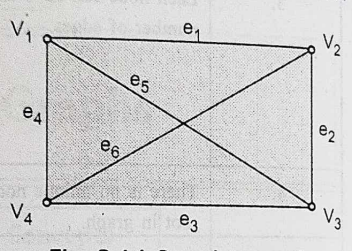


Fig. Q.4.1 Complete graph

ii) Weighted Graph : A weighted graph is a graph which consists of weights along its edges.

For example :

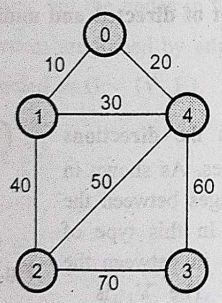


Fig. Q.4.2 Weighted graph

Q.5 Explain the terms - i) Path and ii) Cycle.

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

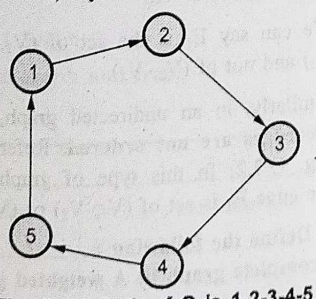


Fig. Q.5.1 Path of G is 1-2-3-4-5

Cycle : A closed walk through the graph with repeated vertices, mostly having the same starting and ending vertex is called a cycle.

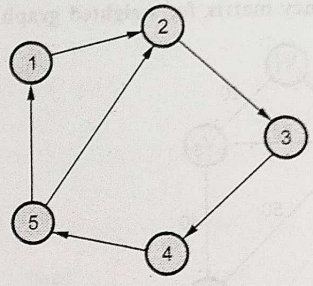


Fig. Q.5.2 Cycle 2-3-4-5-2 or 1-2-3-4-5-1

3.2 : Storage Representation

Q.6 Explain the following terms :

i. Adjacency matrix of graph ii. Adjacency list of graph.

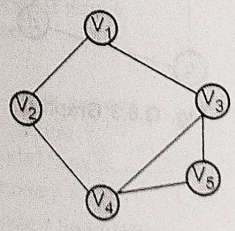
☞[SPPU : May-05, Dec.-05, Dec.07, Marks 4]

OR Write a short note on different representation of a graph.

☞ [SPPU : May-11, Marks 6]

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

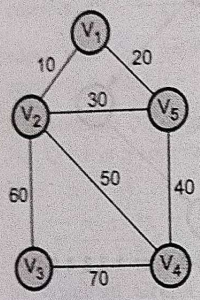
- Consider a graph G of n vertices and the matrix M. If there is an edge present between vertices V_i and V_j then M[i][j] = 1 else M[i][j] = 0. Note that for an undirected graph if M[i][j] = 1 then for M[j][i] is also 1. Here are some graphs shown by adjacency matrix.



	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

Adjacency matrix for weighted graph



Weighted graph

	1	2	3	4	5
1	0	10	0	0	20
2	10	0	60	50	30
3	0	60	0	70	0
4	0	50	70	0	40
5	20	30	0	40	0

Adjacency matrix

Fig. Q.6.2 Adjacency matrix representation for weighted graph

- In the weighted graph, weights or distances are given along every edge. Hence in an adjacency matrix representation any edge which is present between vertices V_i and V_j is denoted by its weight. Hence M[i][j] = Weight of edge.
- If there is no edge between V_i and V_j then, M[i][j] = 0.

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

There are two methods of representing graph using adjacency list.

Method 1 : As we know the graph is a set of vertices and edges, we will maintain the two structures, for vertices and edges respectively.

For example : Now in Fig. Q.6.3, 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.

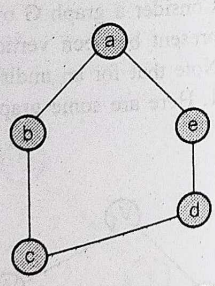


Fig. Q.6.3 Graph G

```
typedef struct head
{
    char data;
    struct head *down;
```



```
    struct head *next;
};
typedef struct node
{
 int ver;
struct node *link;
};
```

Explanation : This is purely the adjacency list graph. 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.

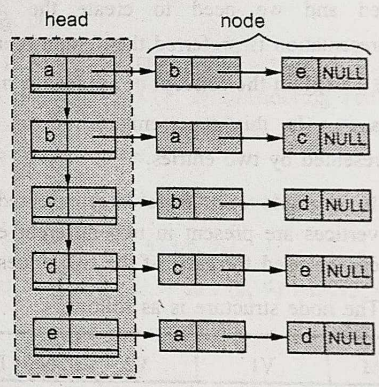


Fig. Q.6.4 Adjacency list (Method 1)

Method 2 : In this method of representing the adjacency list, we take mixed data structure. That means instead of taking the head list as a linked list we will take an array of head nodes. So only one 'C' structure will be there representing the adjacent nodes. See the Fig. Q.6.5 representing the adjacency list for the above graph.

First let us see the 'C' structure.

```
typedef struct
node1
{
    char vertex;
    struct
node1*nex ;
}node ;
node * head [10] ;
```

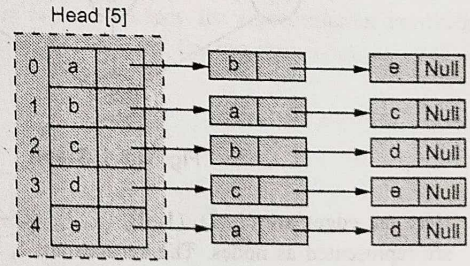


Fig. Q.6.5 Adjacency list (Method 2)

Explanation : This is the graph which can be represented with the array and linked list data structures. Array is used to store the head nodes. The node structure will be the same through out.

Q.7 Explain the situation in which linked representation of a graph is more beneficial than array representation.

[SPPU : Dec.-13, Marks 2]

Ans. : • The situation in which the number of nodes in the graph is not fixed and we need to create the graph dynamically, then linked representation is preferred than graph representation.

Q.8 Explain the concept of adjacency multilist in graph with example.

Ans. : • In this representation of an undirected graph each edge is represented by two entries.

- In a graph, each edge is represented by two vertices but these two vertices are present in two different edges. Hence adjacency multilists can be used for each of the two nodes incident to.
- The node structure is as follows -

M	V1	V2	Next Link for V1	Next Link for V2

- Here M is a one bit field that is used to represent whether the edge is visited or not.

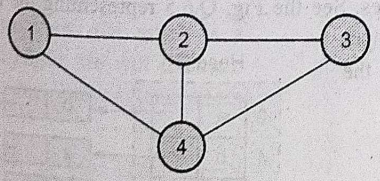


Fig. Q.8.1 Graph

- Here the edges are (1, 2), (1, 4), (2, 3), (2, 4) and (3, 4). These edges are represented as nodes. The vertices are 1, 2, 3, 4.

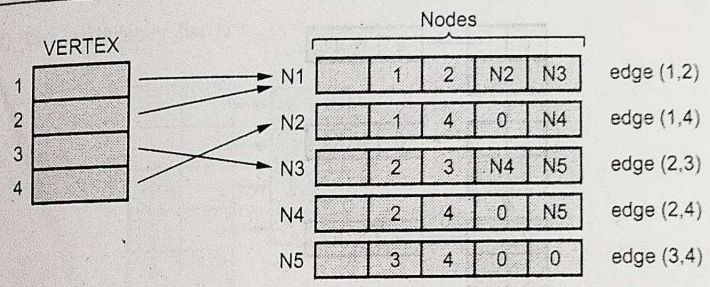


Fig. Q.8.2

- **In node N1** the edge (1, 2) is specified. Third field N2 is for edge (1, 4), this link is the adjacent edge of edge (1, 2) considering vertex V1 i.e., 1 Forth field-N3 is for edge (2, 3), this link is the adjacent edge of edge (1, 2) considering vertex V2 i.e. 2.
- In this way remaining nodes N2, N3, N4 and N5 are created.
- Now Vertex 1 is connected to (1, 2) and (1, 4) i.e. N1, N2.
- Similarly Vertex 2 is connected to (1, 2), (2, 3) and (2, 4) i.e. N2, N3 and N4

$$\left.\begin{array}{l} \text{Vertex 1 : N1} \rightarrow \text{N2} \\ \text{Vertex 2 : N2} \rightarrow \text{N3} \rightarrow \text{N4} \\ \text{Vertex 3 : N3} \rightarrow \text{N5} \\ \text{Vertex 4 : N2} \rightarrow \text{N4} \rightarrow \text{N5} \end{array}\right\} \text{Adjacency Multilists}$$

Q.9 Write short note on - Inverse adjacency list.

Ans. : Inverse adjacency list is an adjacency list representation in which only incoming edge to a vertex is represent. For example -

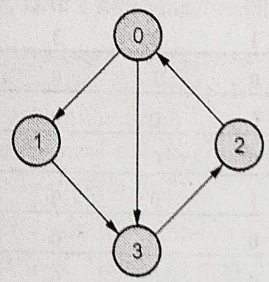


Fig. Q.9.1 Graph G

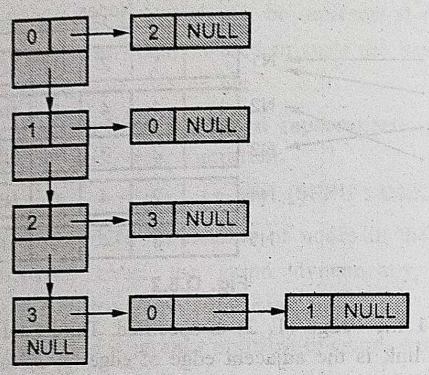


Fig. Q.9.2 Inverse adjacency list of G

Q.10 Draw any directed graph with minimum 6 it nodes and represent graph using adjacency matrix, adjacency list, adjacency multilist and inverse adjacency list.

☞ [SPPU : May-18, June-22, Marks 6]

Ans. : The required directed graph is as shown below.

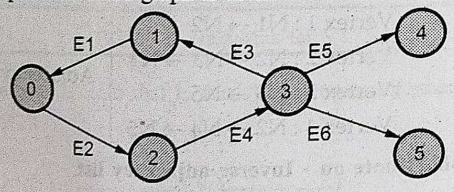
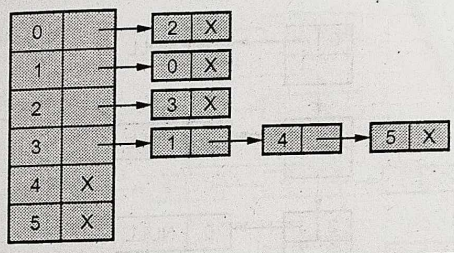


Fig. Q.10.1

i) The Adjacency matrix is

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

ii) The Adjacency list is



Edge 1		1	0	Edge 3	NULL	edge (0, 1)
Edge 2		0	2	NULL	NULL	edge (0, 2)
Edge 3		3	1	Edge 4	NULL	edge (3, 1)
Edge 4		2	3	NULL	NULL	edge (2, 3)
Edge 5		3	4	NULL	NULL	edge (3, 4)
Edge 6		3	5	NULL	NULL	edge (3, 5)

Fig. Q.10.2 Adjacency multilist

- In above multilist various edges are represented. In the node of edge 1 the edge (1,0) is represented. For vertex 1, the incident edge is E3. Hence, we enter 'Edge 3' in the fourth field. For vertex 0, the incident edge is E1 itself. So we need not have to mention this self defining edge. So we enter NULL in fifth field.
- Again for Edge 2, for vertex 0 the incident edge is 1 - 0 (Edge 1) but this edge is already defined. Hence, we mention NULL in 4th field of Edge 2.

- The inverse adjacency list is a list in which only incoming edge to a vertex is represented.

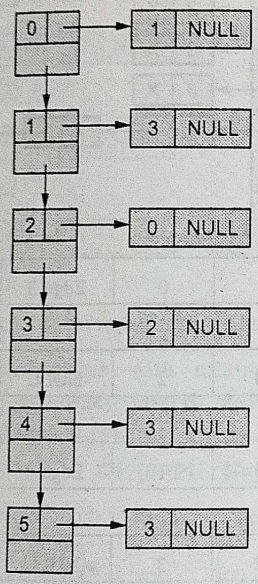


Fig. Q.10.3

Q.11 What is graph ? Draw how the following graph can be represented using linked organization : ☞ [SPPU : Dec.-10, Marks 8]

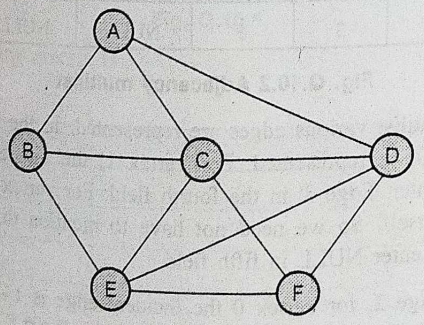


Fig. Q.11.1

Ans. : Graph : The graph is a collection of nodes and vertices. The linked representation of given graph is -

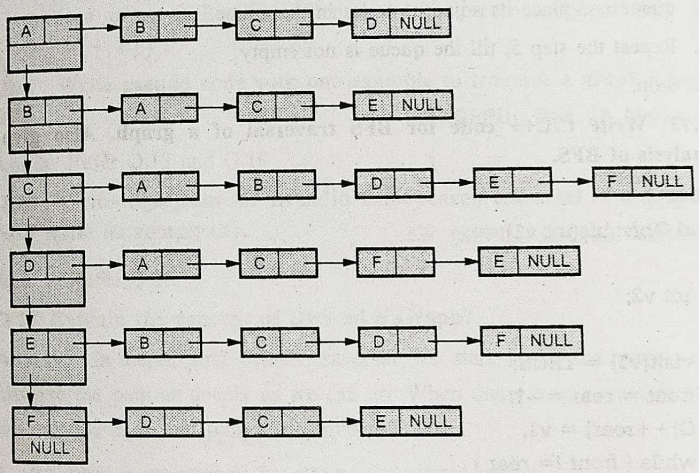


Fig. Q.11.2

3.3 : Traversals

Q.12 Write an algorithm for Breadth First Search traversal of a graph.

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

Algorithm :

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.

Q.13 Write C/C++ code for BFS traversal of a graph. Also give analysis of BFS.

Ans. :

```
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;
         }
      }
   }
}
```

Analysis of BFS

- If the graph is created using **adjacency matrix** in BFS routine then while statement executes for n times. Inside this while statement there is a for loop which executes for all the vertices. Hence time complexity of BFS is $O(V^2)$.

- If the graph is created using **adjacency list** then each vertex V is enqueued and dequeued at most once. Scanning for all the adjacent vertices takes O(|E|) time. Hence the time complexity of BFS is $O(|V|+|E|)$.

Q.14 Write pseudo code with one example to traverse a graph using BFS. ☞ **[SPPU : Dec.-22, Marks 6]**

Ans. : Refer Q.13 and Q.19

Q.15 Write algorithm for Breadth First search traversal of a graph. Also write its complexity. ☞ **[SPPU : Dec.-22, Marks 6]**

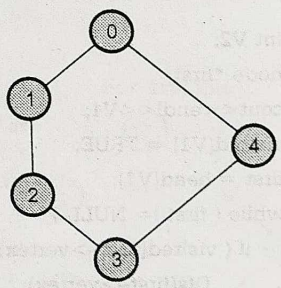
Ans. : Refer Q.13

Q.16 Explain the concept of DFS od a Graph?

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.17 Write an algorithm for Depth First Search traversal of a graph.

Ans. : 1. Define stack with size as total number of vertices.

2. Select any vertex as a starting point for traversal. Visit it, print it and push it onto the stack.

3. Visit any one of the adjacent node which is on the top of the stack and that adjacent node must be unvisited, print it and push it onto the stack, marks it as visited.
4. Repeat step 3, until there is no new vertex to be visited for the vertex which is on the top of the stack.
5. When there is no new vertex to be visited, backtrack and pop one vertex from the stack.
6. Repeat step 3,4,5 until stack becomes empty.

Q.18 Write a recursive C/C++ function for DFS traversal of a graph.

Ans. :

```
/*
-----------------------------------------------------------------
The Dfs function
-----------------------------------------------------------------
*/
void Lgraph::Dfs(int V1)
{
    int V2;
    node *first;
    cout<<endl<<V1;
    visited[V1] = TRUE;
    first = head[V1];
    while ( first != NULL )
        if ( visited[first->vertex] == FALSE )
            Dfs(first->vertex);
        else
            first = first -> next;
}
```

Q.19 Define DFS and BFS for a graph. Show BFS and DFS for the following graph with starting vertex as 1.

☞ **[SPPU : Dec.-17, June-22, Marks 6]**

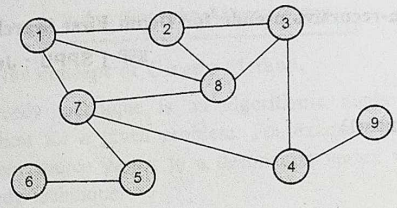
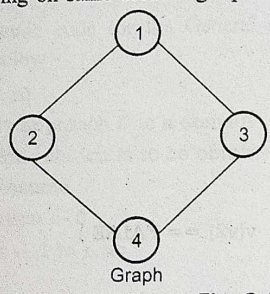


Fig. Q.19.1

Ans. : BFS : 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.



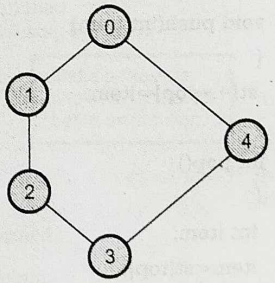
BFS order : 1, 2, 3, 4

Fig. Q.19.2

DFS : • 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.

- 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

For example



For example -

BFS sequence for given example : 1, 2, 7, 8, 3, 4, 5, 9, 6

DFS sequence for given example : 1, 2, 3, 4, 7, 5, 6, 9, 8

Q.20 Write non-recursive pseudo for Depth First Search (DFS).

☞ [SPPU : June-22, Marks]

Ans. :

```
void Gdfs::Dfs(int v1)
{
    int v2;
    push(v1);
    while(top!=-1)
    {
    v1=pop();
    if(v[v1]==FALSE)
    {
    cout<<endl<<v1;
    v[v1] = TRUE;
    }
    for ( v2 = 0; v2 < n; v2++)
        if ( g[v1][v2] == TRUE && v[v2] == FALSE )
            push(v2);
    }
}
void push(int item)
{
st[++top]=item;
}
int pop()
{
int item;
item=st[top];
top--;
return item;
}
```


3.4 : Greedy Algorithms

Q.21 Explain the concept of Greedy Method.

Ans. : The Greedy technique is an algorithmic method for obtaining **optimized solution** for a given problem. For example- for obtaining the shortest path from source vertex to a destination vertex within a graph - we can use greedy technique.

- In Greedy technique, the solution is constructed through a sequence of steps, each expanding a partially constructed solution obtained so far, until a complete solution to the problem is reached. At each step the choice made should be optimal.
- The pseudo code for the General method of Greedy technique is as given below -

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

Check if the selected solution is feasible or not.

Make a feasible choices and select optimum solution.

Explanation of Algorithm

In Greedy method following activities are performed.

1. First we select some solution from input domain.
2. Then we check whether the solution is feasible or not. Feasible soltion is a solution that satisfies problem's constraint.

3. From the set of feasible solutions, particular solution that satisfies or nearly satisfies the objective of the function. Such a solution is called **optimal solution.**

4. As Greedy method works in stages. At each stage only one input is considered at each time. Based on this input it is decided whether particular input gives the optimal solution or not.

Q.22 Explain the terms, feasible solution, optimal solution and objective function.

Ans. : • **Feasible solution** - For solving a particular problem there exists n inputs and we need to obtain a subset that satisfies some constraints. Then any subset that satisfies these constraints is called **feasible solution.**

- **Optimal solution** - From a set of feasible solutions, particular solution that satisfies or nearly satisfies the objectives of the function such a solution is called optimal solution.
- **Objective function** - A feasible solution that either minimizes or maximizes a given objective function is called objective function.

Q.23 What are the applications of Greedy Method ?

Ans. : 1) Knapsack problem 2) Minimum spanning trees

3) Single source shortest path algorithm

4) Huffman's Tree construction.

3.5 : Minimum Spanning Tree

Q.24 Explain Prim's algorithm with an example.

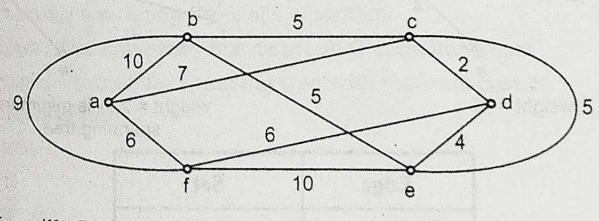
Ans. : Prim's algorithm is to obtain the minimum spanning tree. In this algorithm the minimum edge is selected each time. The edge selection is done for adjacent vertices only.

```
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 -

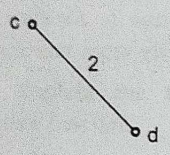
1. Select the edge with minimum edge. The corresponding vertices will form the set of vertices.
2. Each time select an edge with minimum weight and the edge to be selected from the graph should be such that at least one of the vertex of the selected edge is adjacent to already selected vertices.

3. In selection of edges the circuit should not be formed.

From above graph,

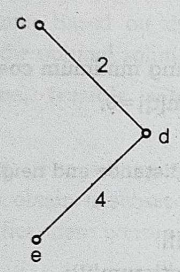
First select edge c-d with minimum weight 2

Step 1 :



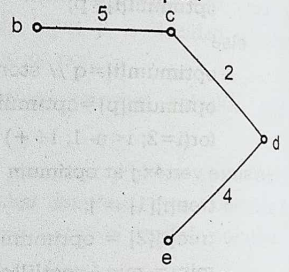
weight = 2

Step 2 :



weight = 6

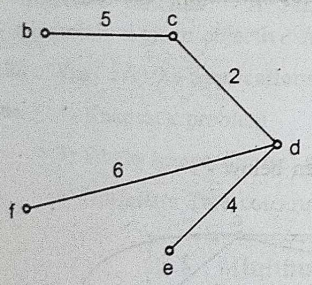
Step 3 :



weight = 11

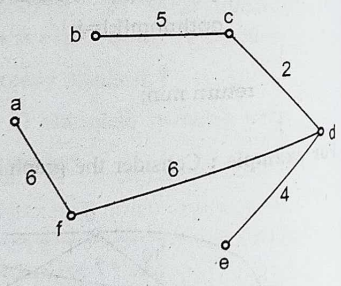
The edge selection can be summarized as below -

Step 4



weight = 16

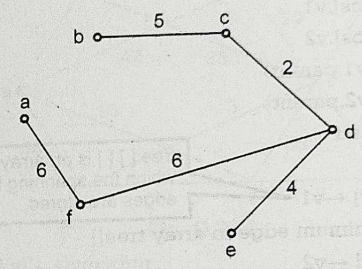
Step 5



weight = 23 - is minimum spanning tree

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.25 What is Kruskal's algorithm ? Explain it with suitable example. **☞[SPPU : Dec.-22, Marks 6]**

Ans. : • In Kruskal's algorithm the **minimum** weight is obtained. In this algorithm also the **circuit should not be formed**. Each time the edge of minimum weight has to be selected, from the graph.It is **not necessary** in this algorithm to have edges of minimum weights to be **adjacent**.

Algorithm

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

xample - Refer Q.**

.26 Consider the following graph (Fig. Q.26.1)
Obtain minimum spanning tree by Kruskal's algorithm.

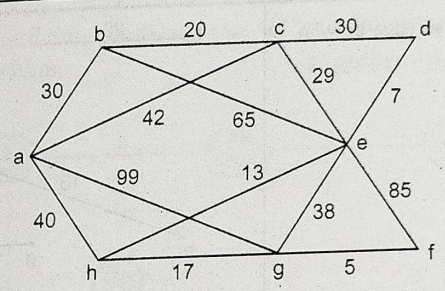
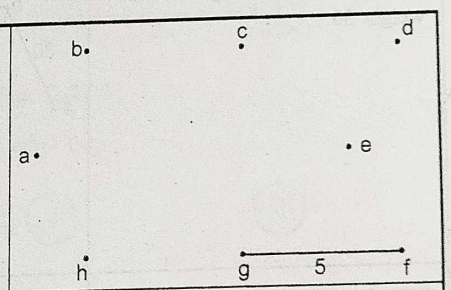
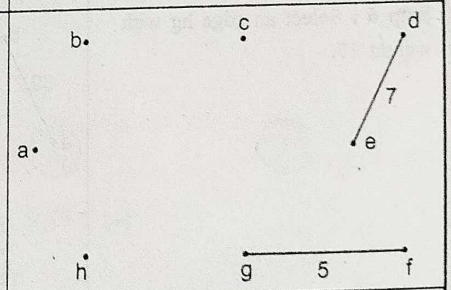
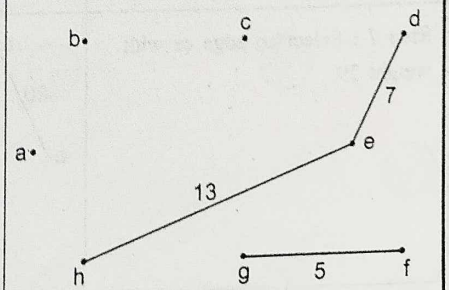
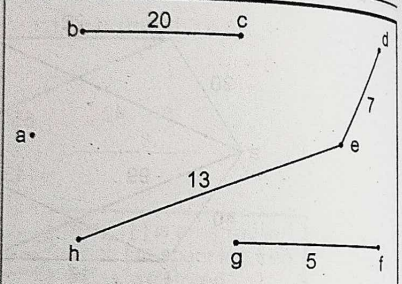
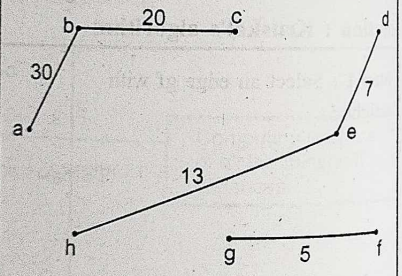
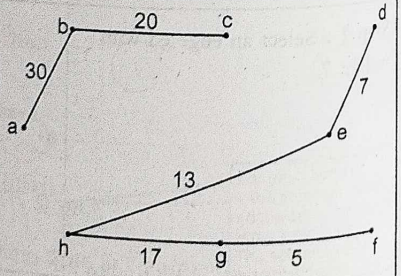
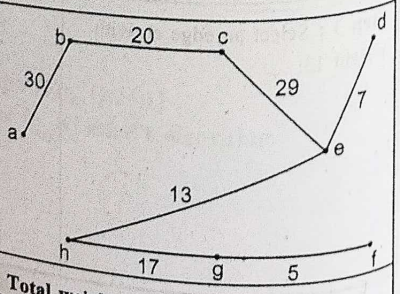


Fig. Q.26.1

Solution : Kruskal's algorithm

Step 1 : Select an edge gf with weight 5.	
Step 2 : Select an edge ed with weight 7.	
Step 3 : Select an edge eh with weight 13.	

Step 4 : Select an edge bc with weight 20.	
Step 5 : Select an edge ab with weight 30.	
Step 6 : Select an edge hg with weight 17.	
Step 7 : Select an edge ce with weight 29.	
	Total weight = 134

Q.27 Find minimum spanning tree of the following graph using Kruskals algorithm. ☞ **[SPPU : Dec.-22, Marks 6]**

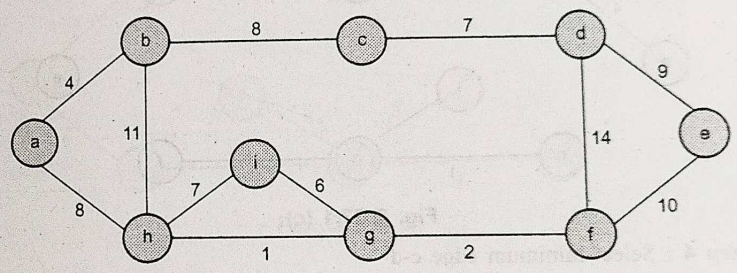


Fig. Q.27.1

Ans. : Step 1 : Select minimum edge h-g

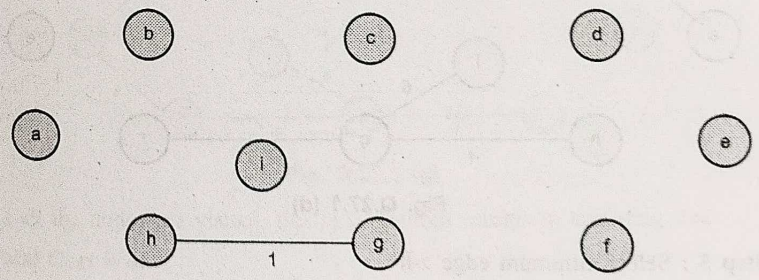


Fig. Q.27.1 (a)

Step 2 : Select minimum edge g-f

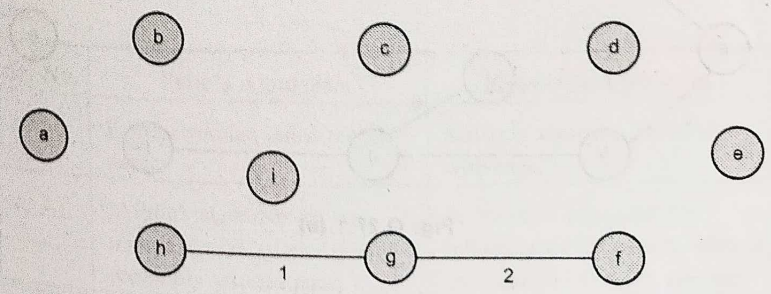


Fig. Q.27.1 (b)

POCO M3 | ❤️❤️ DIRECT ❤️❤️

Step 3 : Select minimum edge i-g

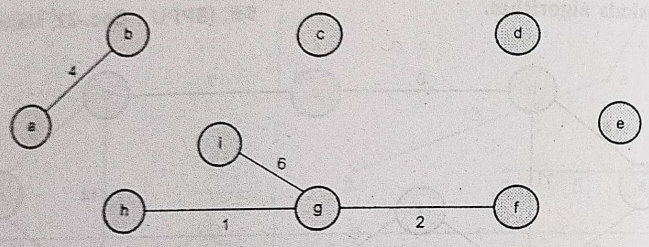


Fig. Q.27.1 (c)

Step 4 : Select minimum edge c-d

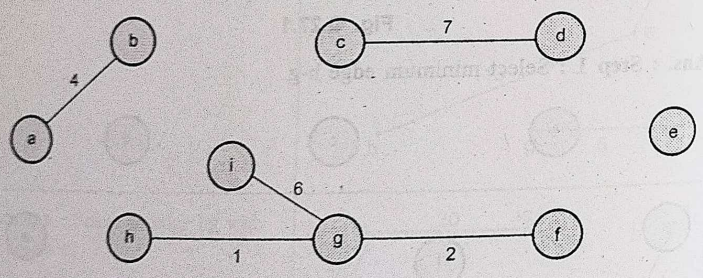


Fig. Q.27.1 (d)

Step 5 : Select minimum edge a-h

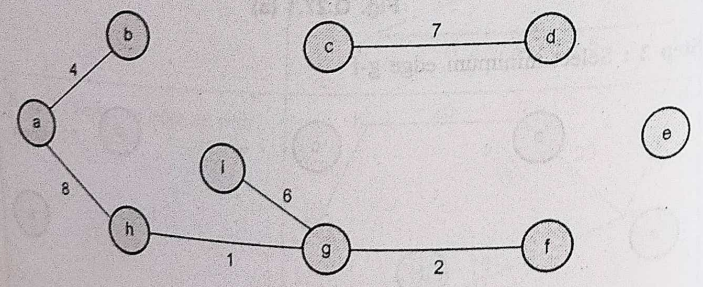


Fig. Q.27.1 (e)

Step 6 : Select minimum edge b-c

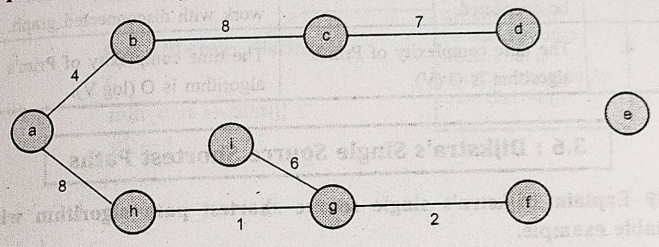


Fig. Q.27.1 (f)

Step 7 : Select minimum edge d-e

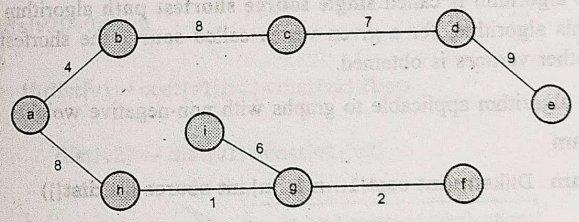


Fig. Q.27.1 (g)

As all the nodes are visited, this is a required minimum spanning tree.

Total Cost = 45

Q.28 What is the difference between Prim's and Kruskal's algorithm for minimum spanning trees ?

☞ **[SPPU : May-13, Marks 4, Dec.-13, Marks 3]**

Ans. :

Sr. No.	Prim's Algorithm	Kruskal's Algorithm
1.	Prim's algorithm initializes with node.	Kruskal's algorithm initializes with edge.
2.	In Prim's algorithm the next node is always selected from previously selected node.	In Kruskal's algorithm the minimum weight edge is selected independent of earlier selection.

3.	In Prim's algorithm graph must be connected.	The Kruskal's algorithm can work with disconnected graph.
4.	The time complexity of Prim's algorithm is O (V).	The time complexity of Prim's algorithm is O (log V).

3.6 : Dijkstra's Single Source Shortest Paths

Q.29 Explain Dijkstra's single source shortest path algorithm with suitable example.

Ans. : • Dijkstra's Algorithm is a popular algorithm for finding shortest path.

- This algorithm is called **single source shortest path** algorithm because in this algorithm, for a given vertex called **source** the shortest path to all other vertices is obtained.
- This algorithm applicable to graphs with non-negative weights only.

Algorithm

```
Algorithm Dijkstra(int cost[1...n,1...n],int source,int dist[])

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

Start from each source node


```
        if(s[j]=0)then
        {
            if(dist[j]<min_dist)then
            {
                min_dist ← dist[j]
                v1 ← j
            }
        }
    }
    s[v1] ← 1
    for(v2 ← 0 to tot_nodes-1)
    {
     if(s[v2]=0)then
     {
        if(dist[v1]+cost[v1][v2]<dist[v2])then
        {
            dist[v2] ← dist[v1]+cost[v1][v2]
            path[v2] ← v1
        }
     }
    }
  }
}
```

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

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

Example - Refer Q.30.

Q.30 Obtain the shortest path for the given graph.

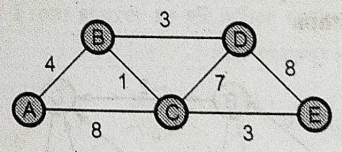
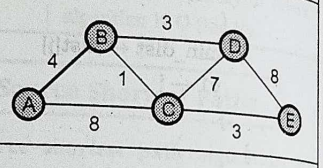
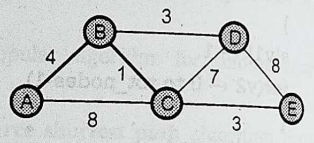
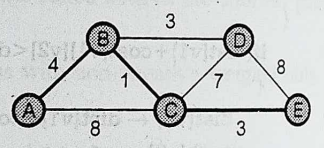


Fig. Q.30.1

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.31 Find the shortest path in the following graph from mode A, using Dijkstra algorithm.

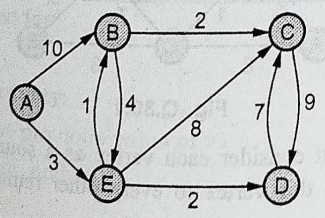


Fig. Q.31.1

Ans. : Consider source vertex A.

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

d(A, B) = 10 d(A, D) = ∞

d(A, C) = ∞ d(A, E) = 3 ← **Min. distance**

∴ Select vertex E.

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

d(A, B) = d(A, E) + d(E, B) = 4 ← **Min. distance**

d(A, C) = d(A, E) + d(E, C) = 11

d(A, D) = d(A, E) + d(E, D) = 5

Hence select vertex B.

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

d(A, C) = d(A, B) + d(B, C) = 4 + 2 = 6

d(A, D) = d(A, E) + d(E, D) = **5** ← **Min. distance**

∴ Select vertex D.

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

Now the only remaining vertex is C.

d(A, C) = 6

Thus we obtain shortest paths from node A as follows.

d(A, B) = 4 A - E - B

d(A, C) = 6 A - E - B - C

d(A, D) = 5 A - E - D

d(A, E) = 3 A - E

Q.32 Consider the given graph and find the shortest path by using Dijkstra's algorithm. From source to all other nodes.

[SPPU : June-22, Marks 6]

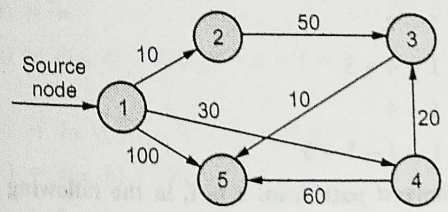


Fig. Q.32.1

Ans. : Consider source vertex 1

Step 1 : S {1} T = {2,3,4,5}

d(1,2) = 10 d(1,3) = ∞

d(1,5) = 100 d(1,4) = 30

The minimum distance is **10**

Hence, select vertex 2.

Step 2 : S{1, 2} T = {3, 4, 5}

d(1,3) = {d(1,2) + d(2,3)} = 10 + 50

d(1,3) = 60

d(1,4) = 30

d(1,5) = 100

The minimum distance is **30**

Hence, select vertex **4**

Step 3 : S{1,2,4} T = {3, 5}

d(1,3) = {d(1,4) + d(4,3)} = 30 + 20

d(1,3) = 50

d(1,5) = {d(1,4) + d(4,5)} = 30 + 60

d(1,5) = 90

The minimum distance is **50**

Hence, select vertex **3**

Step 4 : S = {1,2,4,3} T = {5}

d (1, 5) = {d(1,4)+ d(4,3) + d(3,5)} = 30 + 20 + 10

d(1,5) = 60

Thus we obtain shortest paths from vertex 1 as follows :

d(1,2) = 10 **1 – 2**

d(1,3) = 50 **1 – 4 – 3**

d(1,4) = 30 **1 – 4**

d(1,5) = 60 **1 – 4 – 3 – 5**

Q.33 Find the shortest path from a to f, in the following graph using Dijkstra's Algorithm. ☞ **[SPPU : Dec.-22, Marks 6]**

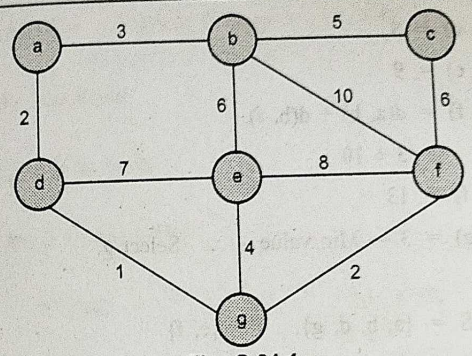


Fig. Q.31.1

Ans. : Consider source node 'a' and remaining nodes are target node.

Step 1 :

S = {a}, T = {b, c, d, e, f, g}

d(a, b) = 3 d(a, e) = ∞ d(a, g) = ∞

d(a, c) = ∞ d(a, f) = ∞

d(a, d) = 2 ← Min. value

∴ Select vertex d.

Step 2 :

S = {a, d}, T = {b, c, e, f, g}

d(a, b) = 3 ← **Min value,** select vertex b.

d(a, c) = ∞

d(a, e) = d(a, d) + d(d, e)

= 2 + 7

d(a, e) = 9

d(a, f) = ∞

d(a, g) = d(a, d) + d(d, g) = 2 + 1 = 3

Step 3 :

S = {a, d, b}, T = {c, e, f, g}

d(a, c) = d(a, b) + d(b, c)

= 3 + 5

d(a, c) = 8

d(a, e) = 9

d(a, f) = d(a, b) + d(b, f)

= 3 + 10

d(a, f) = 13

d(a, g) = 3 ← Min value ∴ Select g.

Step 4 :

S = {a, b, d, g}, T = {e, f}

d(a, e) = d(a, g) + d(g, e)

= 3 + 4

d(a, e) = 7

d(a, f) = d(a, g) + d(g, f)

= 3 + 2

d(a, f) = 5 ← Min value

∴ Select vertex f.

Step 5 :

S = {a, b, d, g, f}, T = {e}

d(a, e) = 7

Step 6 : The shortest distance from a to f is **5.**

The shortest path is **a- d - g - f.**

3.7 : All Pairs Shortest Paths

Q.34 Write an algorithm for Warshall's all pair shortest path. Also analyse the algorithm.

Ans. : Algorithm

```
Algorithm Warshall (Matrix [1... n, 1... n]
//Problem Description : This algorithm is for computing
//transitive closure using Warshall's algorithm
//Input : The adjacency matrix given by Matrix[1...n, 1...n]
//Output : The transitive closure of digraph
```



```
R(0) ← Matrix    //Initially adjacency matrix of
                 //digraph becomes R(0)
for (K ← 1 to n) do
{
    for (i ← 1 to n) do
    {
        for (j ← 1 to n) do
        {
            R(k)[i,j] ← R(k-1)[i,j] OR R(k-1)[i,k] AND
                                       R(k-1)[k,j]
        }
    }
}
return R(n)
```

i) We are computing the Transitive closure R for n vertices. The matrix is accessed for n^2 elements. Hence total number of elements computed is n^3. Since computing each element takes constant time, the time efficiency of the algorithm is $\Theta(n^3)$

ii) The traversal based algorithm can be using Depth First Traversal (DFS) or Breadth First Traversal (BFS) technique. We will first compute the time complexity of computing transitive closure using DFS or BFS.

- Suppose the graph is represented using adjacency list with n vertices and m edges, then the graph is accessed using $\Theta(n + m)$ time. For computing transitive closure of n vertices the time taken by these algorithms is $n\Theta(n+m) = \Theta(n^2 + nm)$.
- For the sparse graph, edges $m \in O(n)$. Hence we get the time complexity using the traversal based algorithm as $\Theta(n^2+nm) = \Theta(n^2+nO(n)). = \Theta(n^2)$.
- The warshall algorithm requires $\Theta(n^3)$ time. This shows that the traversal based algorithm for sparse graph is superior to Warshall's algorithm.

Q.35 Apply Warshall's algorithm to find the transitive closure of the digraph defined by the following adjacency matrix.

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

i) Prove that the time efficiency of Warshall's algorithm is cubic.

ii) Explain why the time efficiency of Warshall's algorithm is inferior to that of the traversal based algorithm for sparse graphs represented by their adjacency lists.

Ans. :

$$R(0) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Step 1 :

i = 1, j = 1, k = 1	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{11}^{1} = r_{11}^{0}$ or r_{11}^{0} and r_{11}^{0} = 0 or 0 and 0 = 0
i = 1, j = 2, k = 1	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{12}^{1} = r_{12}^{0}$ or r_{11}^{0} and r_{12}^{0} = 1 or 0 and 1 = 1 or 0 = 1

i = 1, j = 3, k = 1	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{13}^{1} = r_{13}^{0}$ or r_{11}^{0} and r_{13}^{0} = 0 or 0 and 0 = 0
i = 1, j = 4, k = 1	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{14}^{1} = r_{14}^{0}$ or r_{11}^{0} and r_{14}^{0} = 0 or 0 and 0 = 0
i = 2, j = 1, k = 1	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{21}^{1} = r_{21}^{0}$ or r_{21}^{0} and r_{11}^{0} = 0 or 0 and 0 = 0
i = 2, j = 2, k = 1	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{22}^{1} = r_{22}^{0}$ or r_{21}^{0} and r_{12}^{0} = 0 or 0 and 1 = 0 or 0 = 0

i = 2, j = 3, k = 1	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{23}^{1} = r_{23}^{0}$ or r_{21}^{0} and r_{13}^{0} = 1 or 0 and 0 = 1 or 0 = 1
i = 2, j = 4, k = 1	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{24}^{1} = r_{24}^{0}$ or r_{21}^{0} and r_{14}^{0} = 0 or 0 and 0 = 0
i = 3, j = 1, k = 1	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{31}^{1} = r_{31}^{0}$ or r_{31}^{0} and r_{11}^{0} = 0 or 0 and 0 = 0
i = 3, j = 2, k = 1	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{32}^{1} = r_{32}^{0}$ or r_{31}^{0} and r_{12}^{0} = 0 or 0 and 1 = 0 or 0 = 0

i = 3, j = 3, k = 1	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{33}^{1} = r_{33}^{0}$ or r_{31}^{0} and r_{13}^{0} = 0 or 0 and 0 = 0
i = 3, j = 4, k = 1	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{34}^{1} = r_{34}^{0}$ or r_{31}^{0} and r_{14}^{0} = 1 or 0 and 0 = 1 or 0 = 1
i = 4, j = 1, k = 1	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{41}^{1} = r_{41}^{0}$ or r_{41}^{0} and r_{11}^{0} = 0 or 0 and 0 = 0
i = 4, j = 2, k = 1	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{42}^{1} = r_{42}^{0}$ or r_{41}^{0} and r_{12}^{0} = 0 or 0 and 1 = 0 or 0 = 0

i = 4, j = 3, k = 1	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{43}^{1} = r_{43}^{0}$ or r_{41}^{0} and r_{13}^{0} = 0 or 0 and 0 = 0
i = 4, j = 4, k = 1	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{44}^{1} = r_{44}^{0}$ or r_{41}^{0} and r_{14}^{0} = 0 or 0 and 0 = 0

The adjacency matrix table for representing the graph is as shown below.

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

Step 2 :

i = 1, j = 1, k = 2	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{11}^{2} = r_{11}^{1}$ or r_{12}^{1} and r_{21}^{1} = 0 or 1 and 0 = 0 or 0 = 0

i = 1, j = 2, k = 2	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{12}^{2} = r_{12}^{1}$ or r_{12}^{1} and r_{22}^{1} = 1 or 1 and 0 = 1 or 0 = 1
i = 1, j = 3, k = 2	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{13}^{2} = r_{13}^{1}$ or r_{12}^{1} and r_{23}^{1} = 0 or 1 and 1 = 0 or 1 = 1
i = 1, j = 4, k = 2	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{14}^{2} = r_{14}^{1}$ or r_{12}^{1} and r_{24}^{1} = 0 or 1 and 0 = 0 or 0 = 0
i = 2, j = 1, k = 2	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{21}^{2} = r_{21}^{1}$ or r_{22}^{1} and r_{21}^{1} = 0 or 0 and 0 = 0

i = 2, j = 2, k = 2	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{22}^{2} = r_{22}^{1}$ or r_{22}^{1} and r_{22}^{1} = 0 or 0 and 0 = 0 or 0 = 0
i = 2, j = 3, k = 2	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{23}^{2} = r_{23}^{1}$ or r_{22}^{1} and r_{23}^{1} = 1 or 0 and 1 = 1 or 0 = 1
i = 2, j = 4, k = 2	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{24}^{2} = r_{24}^{1}$ or r_{22}^{1} and r_{24}^{1} = 0 or 0 and 0 = 0
i = 3, j = 1, k = 2	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{31}^{2} = r_{31}^{1}$ or r_{32}^{1} and r_{21}^{1} = 0 or 0 and 0 = 0

i = 3, j = 2, k = 2	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{32}^{2} = r_{32}^{1}$ or r_{32}^{1} and r_{22}^{1} = 0 or 0 and 0 = 0 or 0 = 0
i = 3, j = 3, k = 2	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{33}^{2} = r_{33}^{1}$ or r_{32}^{1} and r_{23}^{1} = 0 or 0 and 1 = 0 or 0 = 0
i = 3, j = 4, k = 2	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{34}^{2} = r_{34}^{1}$ or r_{32}^{1} and r_{24}^{1} = 1 or 0 and 0 = 1 or 0 = 1
i = 4, j = 1, k = 2	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{41}^{2} = r_{41}^{1}$ or r_{42}^{1} and r_{21}^{1} = 0 or 0 and 0 = 0 or 0 = 0

i = 4, j = 2, k = 2	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{42}^{2} = r_{42}^{1}$ or r_{42}^{1} and r_{22}^{1} = 0 or 0 and 0 = 0 or 0 = 0
i = 4, j = 3, k = 2	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{43}^{2} = r_{43}^{1}$ or r_{42}^{1} and r_{23}^{1} = 0 or 0 and 1 = 0 or 0 = 0
i = 4, j = 4, k = 2	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{44}^{2} = r_{44}^{1}$ or r_{42}^{1} and r_{24}^{1} = 0 or 0 and 0 = 0

The adjacency matrix table for representing the graph is an shown below -

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

Step 3 :

i = 1, j = 1, k = 3	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{11}^{3} = r_{11}^{2}$ or r_{13}^{2} and r_{31}^{2} = 0 or 1 and 0 = 0 or 0 = 0
i = 1, j = 2, k = 3	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{12}^{3} = r_{12}^{2}$ or r_{13}^{2} and r_{32}^{2} = 1 or 1 and 0 = 1 or 0 = 1
i = 1, j = 3, k = 3	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{13}^{3} = r_{13}^{2}$ or r_{13}^{2} and r_{33}^{2} = 1 or 1 and 1 = 1 or 1 = 1
i = 1, j = 4, k = 3	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{14}^{3} = r_{14}^{2}$ or r_{13}^{2} and r_{34}^{2} = 0 or 1 and 1 = 0 or 1 = 1

i = 2, j = 1, k = 3	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{21}^{3} = r_{21}^{2}$ or r_{23}^{2} and r_{31}^{2} = 0 or 1 and 0 = 0
i = 2, j = 2, k = 3	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{22}^{3} = r_{22}^{2}$ or r_{23}^{2} and r_{32}^{2} = 0 or 1 and 0 = 0 or 0 = 0
i = 2, j = 3, k = 3	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{23}^{3} = r_{23}^{2}$ or r_{23}^{2} and r_{33}^{2} = 1 or 1 and 0 = 1 or 0 = 1
i = 2, j = 4, k = 3	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{24}^{3} = r_{24}^{2}$ or r_{23}^{2} and r_{34}^{2} = 0 or 1 and 1 = 1

i = 3, j = 1, k = 3	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{31}^{2} = r_{31}^{2}$ or r_{33}^{2} and r_{31}^{2} = 0 or 0 and 0 = 0
i = 3, j = 2, k = 3	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{32}^{3} = r_{32}^{2}$ or r_{33}^{2} and r_{32}^{2} = 0 or 0 and 0 = 0 or 0 = 0
i = 3, j = 3, k = 3	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{33}^{3} = r_{33}^{2}$ or r_{33}^{2} and r_{33}^{2} = 0 or 0 and 0 = 0 or 0 = 0
i = 3, j = 4, k = 3	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{34}^{3} = r_{34}^{2}$ or r_{33}^{2} and r_{34}^{2} = 1 or 0 and 1 = 1 or 0 = 1

i = 4, j = 1, k = 3	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{41}^{3} = r_{41}^{2}$ or r_{43}^{2} and r_{31}^{2} = 0 or 0 and 0 = 0 or 0 = 0
i = 4, j = 2, k = 3	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{42}^{3} = r_{42}^{2}$ or r_{43}^{2} and r_{32}^{2} = 0 or 0 and 0 = 0 or 0 = 0
i = 4, j = 3, k = 3	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{43}^{3} = r_{43}^{2}$ or r_{43}^{2} and r_{33}^{2} = 0 or 0 and 0 = 0 or 0 = 0
i = 4, j = 4, k = 3	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{44}^{3} = r_{44}^{2}$ or r_{43}^{2} and r_{34}^{2} = 0 or 0 and 1 = 0

The adjacency matrix table for representing the graph is as shown below -

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

Step 4 :

i = 1, j = 1, k = 4	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{11}^{4} = r_{11}^{3}$ or r_{14}^{3} and r_{41}^{3} = 0 or 1 and 0 = 0 or 0 = 0
i = 1, j = 2, k = 4	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{12}^{4} = r_{12}^{3}$ or r_{14}^{3} and r_{42}^{3} = 1 or 1 and 0 = 1 or 0 = 1
i = 1, j = 3, k = 4	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{13}^{4} = r_{13}^{3}$ or r_{14}^{3} and r_{43}^{3} = 1 or 1 and 0 = 1 or 0 = 1

POCO M3 | ❤️❤️ DIPS ❤️❤️

i = 1, j = 4, k = 4	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{14}^{4} = r_{14}^{3}$ or r_{14}^{3} and r_{44}^{3} = 1 or 1 and 0 = 1 or 0 = 1
i = 2, j = 1, k = 4	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{21}^{4} = r_{21}^{3}$ or r_{24}^{3} and r_{41}^{3} = 0 or 0 and 0 = 0
i = 2, j = 2, k = 4	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{22}^{4} = r_{22}^{3}$ or r_{24}^{3} and r_{42}^{3} = 0 or 0 and 0 = 0 or 0 = 0
i = 2, j = 3, k = 4	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{23}^{4} = r_{23}^{3}$ or r_{24}^{3} and r_{43}^{3} = 1 or 0 and 0 = 1 or 0 = 1

i = 2, j = 4, k = 4	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{24}^{4} = r_{24}^{3}$ or r_{24}^{3} and r_{44}^{3} = 1 or 1 and 0 = 1
i = 3, j = 1, k = 4	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{31}^{4} = r_{31}^{3}$ or r_{34}^{3} and r_{41}^{3} = 0 or 1 and 0 = 0
i = 3, j = 2, k = 4	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{32}^{4} = r_{32}^{3}$ or r_{34}^{3} and r_{42}^{3} = 0 or 1 and 0 = 0 or 0 = 0
i = 3, j = 3, k = 4	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{33}^{4} = r_{33}^{3}$ or r_{34}^{3} and r_{43}^{3} = 0 or 1 and 0 = 0 or 0 = 0

i = 3, j = 4, k = 4	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{34}^{4} = r_{34}^{3}$ or r_{34}^{3} and r_{44}^{3} = 1 or 1 and 0 = 1 or 0 = 1
i = 4, j = 1, k = 4	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{41}^{4} = r_{41}^{3}$ or r_{44}^{3} and r_{41}^{3} = 0 or 0 and 0 = 0 or 0 = 0
i = 4, j = 2, k = 4	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{42}^{4} = r_{42}^{3}$ or r_{44}^{3} and r_{42}^{3} = 0 or 0 and 0 = 0 or 0 = 0
i = 4, j = 3, k = 4	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{43}^{4} = r_{43}^{3}$ or r_{44}^{3} and r_{43}^{3} = 0 or 0 and 0 = 0 or 0 = 0

i = 4, j = 4, k = 4	**Formula Used** $r_{ij}^{k} = r_{ij}^{(k-1)}$ or $r_{ik}^{(k-1)}$ and $r_{kj}^{(k-1)}$ $r_{44}^{4} = r_{44}^{3}$ or r_{44}^{3} and r_{44}^{3} = 0 or 0 and 0 = 0

The adjacency matrix table for representing the graph is as shown below -

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

Q.36 Write an algorithm for Floyd's all pair shortest path. Also analyse the algorithm.

Ans. : Algorithm

```
Algorithm Floyd_shortest_path (wt[1...n, 1...n)
//Problem Description : This algorithm is for computing
//shortest path between all pairs of vertices.
//Input : The weighted matrix. wt[1...n, 1...n] for
//given graph.
//Output : The distance matrix D containing shortest
//paths.
    D ← wt        //Copy weighted matrix to matrix D
    //This initialization gives D(0)
    for k ← 1 to n do
    {
        for i ← 1 to n do
```

```
    {
        for j ← 1 to n do
        {
            D[i,j] ← min{D[i,j], (D[i,k] + D[k,j])}
        }
    }
}
return D  //computed D^(n) is returned
```

Computing $D^{(k)}$ using two cases ① with k ② without k as intermediate vertex

Note that this is the basic operation in this algorithm

Analysis

In the above given algorithm the basic operation is -

$$D[i, j] \leftarrow \min\{D[i,j], D[i,k] + D[k, j]\}$$

This operation is with in three nested for loops, we can write

$$C(n) = \sum_{k=1}^{n}\sum_{i=1}^{n}\sum_{j=1}^{n} 1$$

$$C(n) = \sum_{k=1}^{n}\sum_{i=1}^{n} (n-1+1) \qquad \because \sum_{i=l}^{u} 1 = u - l + 1$$

$$= \sum_{k=1}^{n} \sum_{i=1}^{n} n$$

$\sum_{i=1}^{n} i \approx \frac{1}{2} n^2$, we can neglect constants and write it as n^2

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

$$C(n) = n^3 \qquad \because \sum_{i=1}^{n} n^2 \approx \frac{1}{3} n^3$$, we neglect constants.

The time complexity of finding all pair shortest path is $\Theta(n^3)$.

Q.37 Solve the all-pairs shortest path problem for the diagraph with the following weight matrix.

$$\begin{bmatrix} 0 & 2 & \infty & 1 & 8 \\ 6 & 0 & 3 & 2 & \infty \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & \infty & \infty & \infty & 0 \end{bmatrix}$$

Ans. : The shortest path can be obtained using formula

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

We will compute $D^{(1)}, D^{(2)}, D^{(3)}, D^{(4)}, D^{(5)}$

$D^{(0)} =$

	1	2	3	4	5
1	0	2	∞	1	8
2	6	0	3	2	∞
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	∞	∞	∞	0

Step 1 :

Computing $D^{(1)}$

(1) k = 1, i = 1, j = 1

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

$$= \min\left\{D^0[1,1], D^0[1,1] + D^0[1,1]\right\}$$

$$= \min\{0, 0 + 0\}$$

$$D^1[1, 1] = 0$$

(2) k = 1, i = 1, j = 2

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

$$= \min\left\{D^0[1,2], D^0[1,1] + D^0[1,2]\right\}$$

$= \min\{2, 0 + 2\}$

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

(3) $k = 1, i = 1, j = 3$

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

$= \min\left\{D^0[1,3], D^0[1,1] + D^0[1,3]\right\}$

$= \min\{\infty, 0 + \infty\}$

$D^1[1,3] = \infty$

(4) $k = 1, i = 1, j = 4$

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

$= \min\left\{D^0[1,4], D^0[1,1] + D^0[1,4]\right\}$

$= \min\{1, 0 + 1\}$

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

(5) $k = 1, i = 1, j = 5$

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

$= \min\left\{D^0[1,5], D^0[1,1] + D^0[1,5]\right\}$

$= \min\{8, 0 + 8\}$

$D^1[1,5] = 8$

Thus we have computed 1st row of D^1 matrix. Now

(1) $k = 1, i = 2, j = 1$

$= \min\left\{D^0[i,j], D^0[i,k] + D^0[k,j]\right\}$

$= \min\left\{D^0[2,1], D^0[2,1] + D^0[1,1]\right\}$

$= \min\{6, 6 + 0\}$

$D^1[2,1] = 6$

(2) $k = 1, i = 2, j = 2$

$= \min\left\{D^0[i,j], D^0[i,k] + D^0[k,j]\right\}$

$= \min\left\{D^0[2,2], D^0[2,1] + D^0[1,2]\right\}$

$= \min\{0, 6 + 2\}$

$D^1[2,2] = 0$

(3) $k = 1, i = 2, j = 3$

$= \min\left\{D^0[i,j], D^0[i,k] + D^0[k,j]\right\}$

$= \min\left\{D^0[2,3], D^0[2,1] + D^0[1,3]\right\}$

$= \min\{3, 6 + \infty\}$

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

(4) $k = 1, i = 2, j = 4$

$= \min\left\{D^0[i,j], D^0[i,k] + D^0[k,j]\right\}$

$= \min\left\{D^0[2,4], D^0[2,1] + D^0[1,4]\right\}$

$= \min\{2, 6 + 1\}$

$D^1[2,4] = 2$

(5) $k = 1, i = 2, j = 5$

$= \min\left\{D^0[i,j], D^0[i,k] + D^0[k,j]\right\}$

$= \min\left\{D^0[2,5], D^0[2,1] + D^0[1,5]\right\}$

$= \min\{\infty, 6 + 8\}$

$D^1[2,5] = 14$

Now we will compute third row of $D^{(1)}$

(1) $k = 1, i = 3, j = 1$

$= \min\left\{D^0[i,j], D^0[i,k] + D^0[k,j]\right\}$

$= \min\left\{D^0[3,1], D^0[3,1] + D^0[1,1]\right\}$

$= \min\{\infty, \infty + 0\}$

$D^1[3,1] = \infty$

(2) $k = 1, i = 3, j = 2$

$= \min\{D^0[i,j], D^0[i,k] + D^0[k,j]\}$

$= \min\{D^0[3,2], D^0[3,1] + D^0[1,2]\}$

$= \min\{\infty, \infty + 2\}$

$D^1[3,2] = \infty$

(3) $k = 1, i = 3, j = 3$

$= \min\{D^0[i,j] + D^0[i,k] + D^0[k,j]\}$

$= \min\{D^0[3,3], D^0[3,1] + D^0[1,3]\}$

$= \min\{0, \infty + \infty\}$

$D^1[3,3] = 0$

(4) $k = 1, i = 3, j = 4$

$= \min\{D^0[i,j], D^0[i,k] + D^0[k,j]\}$

$= \min\{D^0[3,4], D^0[3,1] + D^0[1,4]\}$

$= \min\{4, \infty + 1\}$

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

(5) $k = 1, i = 3, j = 5$

$= \min\{D^0[i,j], D^0[i,k] + D^0[k,j]\}$

$= \min\{D^0[3,5], D^0[3,1] + D^0[1,5]\}$

$= \min\{\infty, \infty + 8\}$

$D^1[3,5] = \infty$

(1) $k = 1, i = 4, j = 1$

$= \min\{D^0[i,j], D^0[i,k] + D^0[k,j]\}$

$= \min\{D^0[4,1], D^0[4,1] + D^0[1,1]\}$

$= \min\{\infty, \infty + 0\}$

$D^1[4,1] = \infty$

(2) $k = 1, i = 4, j = 2$

$= \min\{D^0[i,j], D^0[i,k] + D^0[k,j]\}$

$= \min\{D^0[4,2], D^0[4,1] + D^0[1,2]\}$

$= \min\{\infty, \infty + 2\}$

$D^1[4,2] = \infty$

(3) $k = 1, i = 4, j = 3$

$= \min\{D^0[i,j], D^0[i,k] + D^0[k,j]\}$

$= \min\{D^0[4,3], D^0[4,1] + D^0[1,3]\}$

$= \min\{2, \infty + \infty\}$

$D^1[4,3] = 2$

(4) $k = 1, i = 4, j = 4$

$= \min\{D^0[i,j], D^0[i,k] + D^0[k,j]\}$

$= \min\{D^0[4,4], D^0[4,1] + D^0[1,4]\}$

$= \min\{0, \infty + 1\}$

$D^1[4,4] = 0$

(5) $k = 1, i = 4, j = 5$

$= \min\{D^0[i,j], D^0[i,k] + D^0[k,j]\}$

$= \min\{D^0[4,5], D^0[4,1] + D^0[1,5]\}$

$= \min\{3, \infty + 8\}$

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

(1) $k = 1, i = 5, j = 1$

$= \min\{D^0[i,j], D^0[i,k] + D^0[k,j]\}$

$= \min\{D^0[5,1], D^0[5,1] + D^0[1,1]\}$

$= \min\{3, 3 + 0\}$

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

(2) $k = 1, i = 5, j = 2$

$= \min\{D^0[i,j], D^0[i,k] + D^0[k,j]\}$

$= \min\{D^0[5,2], D^0[5,1] + D^0[1,2]\}$

$= \min \{ \infty, 3 + 2 \}$

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

(3) $k = 1, i = 5, j = 3$

$= \min\{D^0[i, j], D^0[i, k] + D^0[k, j]\}$

$= \min\{D^0[5,3], D^0[5,1] + D^0[1,3]\}$

$= \min \{ \infty, 3 + \infty\}$

$D^1[5, 3] = \infty$

(4) $k = 1, i = 5, j = 4$

$= \min\{D^0[i, j], D^0[i, k] + D^0[k, j]\}$

$= \min\{D^0[5,4], D^0[5,1] + D^0[1,4]\}$

$= \min \{ \infty, 3 + 1 \}$

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

(5) $k = 1, i = 5, j = 5$

$= \min\{D^0[i, j], D^0[i, k] + D^0[k, j]\}$

$= \min\{D^0[5,5], D^0[5,1] + D^0[1,5]\}$

$= \min \{ 0, 3 + 8 \}$

$D^1[5, 5] = 0$

The $D^{(1)}$ matrix will be

$D^{(1)} =$

	1	2	3	4	5
1	0	2	∞	1	8
2	6	0	3	2	14
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	5	∞	4	0

Continueing in this fashion we get the matrices as follows -

$D^{(2)} =$

	1	2	3	4	5
1	0	2	5	1	8
2	6	0	3	2	14
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	5	8	4	0

$D^{(3)} =$

	1	2	3	4	5
1	0	2	5	1	8
2	6	0	3	2	14
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	5	8	4	0

$D^{(4)} =$

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

$D^{(5)} =$

0	2	3	1	4
6	0	3	2	5
10	12	0	4	7
6	8	2	0	3
3	5	6	4	0

POCO M3 | DIPS

3.8 : Topological Ordering

Q.38 Write short note on Topological Ordering.

[SPPU : June-22, Marks 6]

Ans. : This is a direct implementation of decrease and conquer method. 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.

Example - Refer Q.39.

Q.39 Sort the digraph for topological sort using source removal algorithm.

[SPPU : SPPU : May-14, Marks 3]

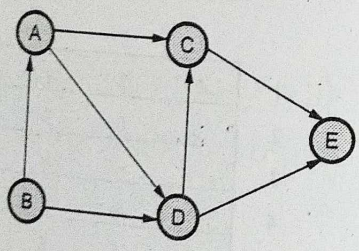


Fig. Q.39.1

Ans. :

- We will follow following steps to obtain topologically sorted list.
- Choose vertex B, because it has no incoming edge, delete it along with its adjacent edges.

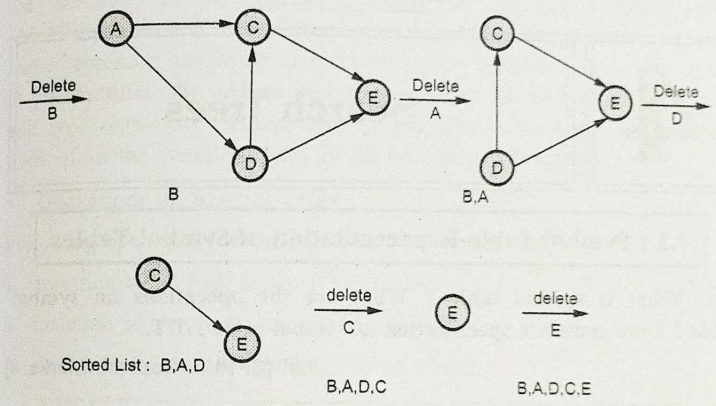


Fig. Q.39.2

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

END...

Unit IV

4 Search Trees

4.1 : Symbol Table-Representation of Symbol Tables

Q.1 What is symbol table ? What are the operations on symbol table ? Give complete specification of symbol table ADT.

☞ **[SPPU : Dec.-11, Marks 8]**

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.1.1 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.

- **Operations on Symbol Table**

Following operations can be performed on symbol table -

1. Insertion of an item in the symbol table.
2. Deletion of any item from the symbol table.
3. Searching of desired item from symbol table.

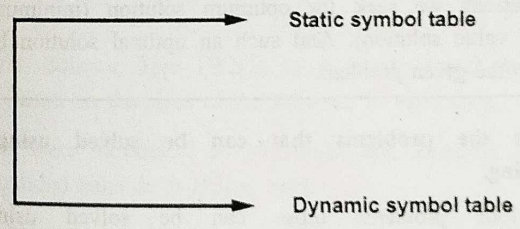
4.2 : Static Tree Table and Dynamic Tree Table

Q.2 Explain static and dynamic tree table.

☞ **[SPPU : June-22, Marks 3]**

Ans. : Types of Symbol Tables

- There are two types of symbol tables.



- 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.

4.3 : Introduction to Dynamic Programming

Q.3 Explain in brief the concept of Dynamic Programming.

Ans. :

- **Dynamic programming** is a method in which the solution to a problem is obtained by making sequence of decisions. The optimal solution to the given problem is obtained from the sequence of all possible solutions being generated.
- In the word dynamic **programming** the word programming stands for **planning** and it does not mean by computer programming.
- The dynamic programming works on **principle of optimality** to solve the problems.

General method

Dynamic programming is typically applied to optimization problems.

For a given problem we may get any number of solutions. From all those solutions we seek for optimum solution (minimum value or maximum value solution). And such an optimal solution becomes the solution to the given problem.

Q.4 Enlist the problems that can be solved using Dynamic Programming.

Ans. : Various problems those can be solved using dynamic programming are,

i) For computing n^{th} Fibonacci number.

ii) Computing binomial coefficient.

iii) Warshall's algorithm.

iv) Floyd's algorithm.

v) Optimal binary search trees.

Q.5 What is the principle of optimality of dynamic programming ?

☞ [SPPU : June-22, Marks 3]

Ans. :

- The dynamic programming makes use of principle of optimality when finding solution to given problem. The principle of optimality states that "in an optimal sequence of choices or decisions, each subsequence must also be optimal."
- When it is not possible to apply principle of optimality, it is almost impossible to obtain the solution using dynamic programming approach.
- **For example :** While constructing optimal binary search tree we always select the value of k which is obtained from minimum cost. Thus it follows principle of optimality.

4.4 : Weight Balanced Tree

Q.6 Explain the concept of Weight Balance Tree with suitable example.

Ans. :

- The Weight Balance Tree (WBT) is a binary search tree, whose balance is based on the sizes of the subtrees, in each node.
- These trees were introduced by Nievergelt and Reingold in 1970 as trees of bounded balance or BB[α] trees.
- A weight balance tree is a binary tree in which the number of nodes in the left subtree is at least half and at most twice the number of nodes in right subtree.
- The balance factor at each node =

$$\frac{\text{Number of external nodes of left subtree}}{\text{Total number of external nodes of tree}}$$

- **Example**

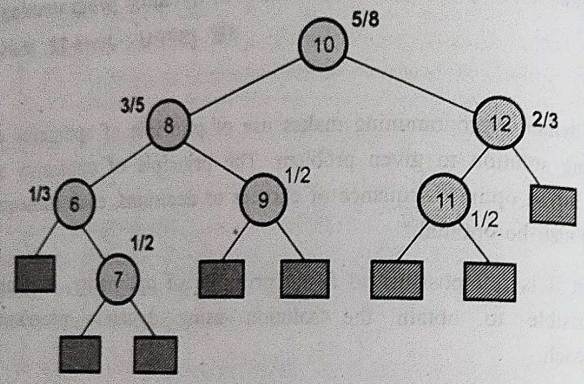


Fig. Q.6.1

- The depth of weight balance tree is O(logn).

4.5 : Optimal Binary Search Tree (OBST)

Q.7 What is Optimal Binary Search Tree (OBST) in data structure? And what are the advantages of OBST ?

☞[SPPU : June-22, Marks 5]

OR What is optimal binary search tree ? What is its use ?

☞ [SPPU : May-10, Marks 4]

Ans. : • **Concept of OBST :** 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 $0 \le i \le n$. 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.**

Advantages of OBST :

1) The cost of searching a node from OBST is reduced.
2) The optimal binary search is not modified once it is constructed.
3) The input sequence is already known for building the search tree and hence this data structure is suitable for offline algorithms.
4) The most frequently accessed node is nearer to root node. Hence most frequently accessed nodes can be accessed efficiently.

Q.8 Write an algorithm for optimal binary search tree.

Ans. : In order to obtain the optimal binary search tree we will follow :

1. Compute the weight using Wt function.
2. Compute C_{ij} and r_{ij} using OBST.
3. Build the tree using build_tree function.

Algorithm

```
Algorithm Wt(p,q,W,n)
//p is an input array [1..n]
//q is an input array [0..n]
// W[i,j] will be output for this function such that W[0..n,0..n]
{
  for i←1 to n do
    W[i,i] ← q[i]; // initialize
  for m←1 to n do
  {
   for i←0 to n–m do
    {
      k ← i+m;
      W[i,k]←W[i,k–1] + p[k] + q[k];
    }
  }
write(W[0:n]);//print the values of W
}
```

- For computing C_{ij}, r_{ij} following algorithm is used.

```
Algorithm OBST(p,q,W,C,r)
{
  //computation of first two rows
for i=0 to n do
{
      C[i,i] ← 0;
          r[i,i] ← 0;
          C[i,i+1] ← q[i]+q[i+1]+p[i+1];
          r[i,i+1]←i+1;
      }
      for m ← 2 to n do
      {
         for ← 0 to n–m do
         {
            j←i+m;
            k←Min_Value(C,r,i,j); //call for algorithm Min_Value
            // minimum value of Cij is obtained for deciding value of k
            C[i,j]←W[i,j] + C[i,k–1] + C[k,j];
            r[i,j]←k;
         }
      }
write(C[0:n],r[0:n]);//print values of Cij and rij
}

Algorithm Min_Value(C,r,i,j)
{
   minimum← ∞;
// finding the range of k
   for (m←r[i,j–1] to r[i+1,j]) do
   {
      if(C[i,m–1]+C[m,j])< minimum then
      {
         minimum ← C[i,m–1]+C[m,j];
```



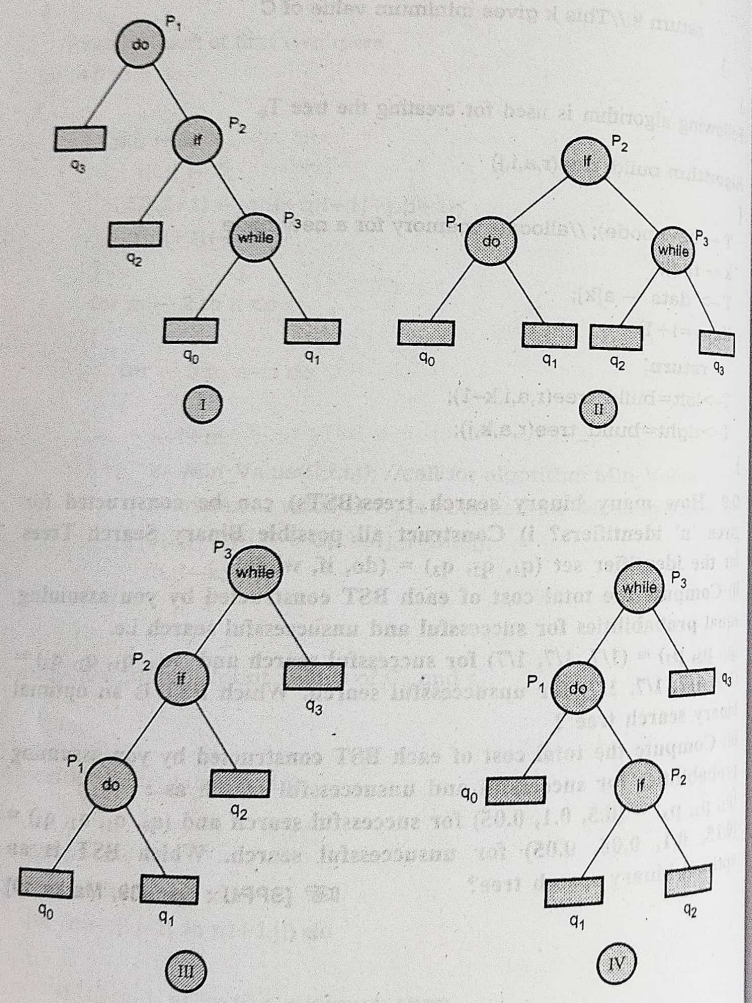
```
         k←m;
      }
      return k;//This k gives minimum value of C
   }
}
```

Following algorithm is used for creating the tree T_{ij}.

```
Algorithm build_tree(r,a,i,j)
{
  T ← new(node); //allocate memory for a new node
  k ← r[i,j];
  T -> data ← a[k];
  if (j==i+1)
     return;
  T->left=build_tree(r,a,i,k–1);
  T->right=build_tree(r,a,k,j);
}
```

Q.9 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, Marks 10]**

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



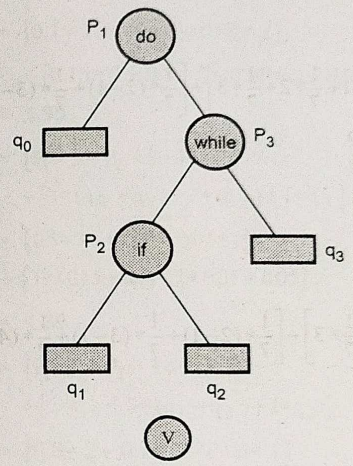


Fig. Q.9.1 All possible BSTs for 3 nodes

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) =

$$\underbrace{\left[\frac{1}{7}*3+\frac{1}{7}*2+\frac{1}{7}*1\right]}_{\text{successful nodes (such as do, if,}}+\underbrace{\left[\frac{1}{7}*(4-1)+\frac{1}{7}*(4-1)+\frac{1}{7}*(3-1)+\frac{1}{7}*(2-1)\right]}_{\text{(unsuccessful nodes denoted by square)}}=\frac{6}{7}+\frac{9}{7}$$

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) = $\frac{13}{7}$

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.9.1.

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.

Q.10 Find the optimal binary search tree for the given data using dynamic programming approach. Explain the solution stepwise.

[SPPU : May-19, Dec.-22, Marks 6]

Index	0	1	2	3
Data	10	20	30	40
Frequency	4	2	6	3

Ans. :

Step 1 : We will build initial tables cost table and root table as follows.

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

Fill up diagonally

Cost table

$$\left.\begin{array}{l} C[1,0]=0 \\ C[2,1]=0 \\ C[3,2]=0 \\ C[4,3]=0 \\ C[5,4]=0 \end{array}\right\} \begin{array}{l}\text{Using formulae} \\ C[i, i-1] = 0 \text{ and} \\ C[n+1, n]=0\end{array}$$

$$\left.\begin{array}{l} C[1,1] = 1 \\ C[2,2] = 2 \\ C[3,3] = 3 \\ C[4,4] = 4 \end{array}\right\} \begin{array}{l}\text{Using Formula} \\ C[i,i] = \text{Frequency } [i]\end{array}$$

The root table

$$\left.\begin{array}{l} R[1,1] = 1 \\ R[2,2] = 2 \\ R[3,3] = 3 \\ R[4,4] = 4 \end{array}\right\} \begin{array}{l}\text{Using Formula} \\ R[i,i] = i\end{array}$$

Step 2 : Now we will compute C[i, j] diagonally as

$$C[i, j] = \min_{i \le K \le j} \{C[i, K-1] + C[K+1, j]\} + \sum_{s=i}^{j} P_s$$

Compute -

C[1, 2], i = 1, j = 2

C [1, 2] =
- When k = 1 → C [1, 0] + C [2, 2] + F [1] + F [2] = 0 + 2 + 4 + 2 = 8
- When k = 2 → C [1, 1] + C [3, 2] + P [1] + P [2] = 4 + 0 + 4 + 2 = 10

As C[1, 2] = 8, gives minimum value consider value of **k = 1**.

∴ Cost [1, 2] = 8, R[1, 2] = 1

Now compute

C[2, 3] = i = 2, j = 3

C [2, 3] =
- k = 2 → C [2, 1] + C [3, 3] + F [2] + F [3] = 0 + 6 + 2 + 6 = 14
- k = 3 → C [2, 2] + C [4, 3] + F [2] + F [3] = 2 + 0 + 2 + 6 = 10

C[2, 3] = 10 with k = 3

R[2, 3] = 10, k = 3

Compute C[3, 4] with i = 3, j = 4

C [3, 4] =
- When k = 3 → C [3, 2] + C [4, 4] + F [3] + F [4] = 0 + 3 + 6 + 3 = 12
- When k = 4 → C [3, 3] + C [5, 4] + F [3] + F [4] = 6 + 0 + 6 + 3 = 15

C[3, 4] = 12 with k = 3

∴ R[3, 4] = 3

The partially field tables are

	0	1	2	3	4
1	0	4	8		
2		0	2	10	
3			0	6	12
4				0	3
5					0

	1	2	3	4
1	1	1		
2		2	3	
3			3	3
4				4

Cost table

Step 3 :

C [1, 3] =
- When k = 1 → C [1, 0] + C [2, 3] + F [1] + F [2] + F [3] = 0 + 10 + 4 + 2 + 6 = 22
- When k = 2 → C [1, 1] + C [3, 3] + F [1] + F [2] + F [3] = 4 + 6 + 4 + 2 + 6 = 22
- When k = 3 → C [1, 2] + C [4, 3] + F [1] + F [2] + F [3] = 8 + 0 + 4 + 2 + 6 = 20

C[1, 3] = 20, with k = 3

Compute C[2, 4]

C [2, 4] =
- When k = 2 → C [2, 1] + C [3, 4] + F [2] + F [3] + F [4] = 0 + 12 + 2 + 6 + 3 = 23
- When k = 3 → C [2, 2] + C [4, 4] + F [2] + F [3] + F [4] = 2 + 3 + 2 + 6 + 3 = 16
- When k = 4 → C [2, 3] + C [5, 4] + F [2] + F [3] + F [4] = 10 + 0 + 2 + 6 + 3 = 21

C[2, 4] = 16 with K = 3

R[2, 4] = 3

	0	1	2	3	4
1	0	4	8	20	
2		0	2	10	16
3			0	6	12
4				0	3
5					0

	1	2	3	4
1	1	1	3	
2		2	3	3
3			3	3
4				4

C[1, 4] can be computed with k = 1, 2, 3 or 4

Consider, i = 1, j = 4

C [1, 4] =
- When k = 1 → C [1, 0] + C [2, 4] + F [1] + F [2] + F [3] + F [4] = 0 + 16 + 4 + 2 + 6 + 3 = 31
- When k = 2 → C [1, 1] + C [3, 4] + F [1] + F [2] + F [3] + F [4] = 4 + 12 + 4 + 2 + 6 + 3 = 31
- When k = 3 → C [1, 2] + C [4, 4] + F [1] + F [2] + F [3] + F [4] = 8 + 3 + 4 + 2 + 6 + 3 = 26
- When k = 4 → C [1, 3] + C [5, 4] + F [1] + F [2] + F [3] + F [4] = 20 + 0 + 4 + 2 + 6 + 3 = 35

C[1, 4] = 26 with k = 3

	0	1	2	3	4
	0	4	8	20	26
		0	2	10	16
			0	6	12
				0	3
					0

	1	2	3	4	
1	1	1	3	3	← Root
2		2	3	3	
3			3	3	
4				4	

To build a tree R[1] [n] = R[1] [4] = 3 is root data

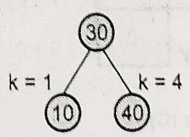
1	2	3	4
10	20	30	40

↑ Root

Here i = 1, j = 4, k = 3

Left child = R[i, k − 1] → T[1, 2] i.e. 1

Right child = R[k + 1, j] → T[4, 4] i.e. 4



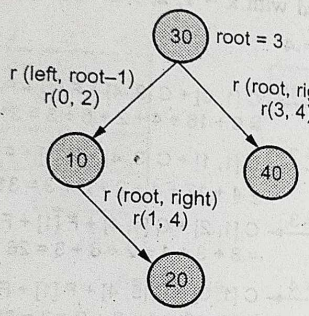


Fig. Q.10.1 Required OBST

Q.11 What is OBST? List binary search tree with 3 words (w1, w2, w3) = (do, if, stop) words occurs with probabilities (P1, P2, p3) = (0.4, 0.5, 0.1) find expected access time in each case.

[SPPU : Dec.-22, Marks 6]

Ans. : We will first number out the nodes (do, if, stop) as 1, 2, 3 respectively. There are 3 nodes. Hence n = 3.

	1	2	3
P[i]	0.4	0.5	0.1
a[i]	do	if	stop

The two tables are created as **cost table** and **root table** are as given below -

	0	1	2	3
1	0	0.4		
2		0	0.5	
3			0	0.1
4				0

Fil-up diagonally

Cost table (C)

	1	2	3
1	1		
2		2	
3			3

Fill-up diagonally

Root table (R)

The following formula is used to fill up the above tables -

$$\left.\begin{aligned} C[1,0] &= 0 \\ C[2,1] &= 0 \\ C[3,2] &= 0 \\ C[4,3] &= 0 \end{aligned}\right\} \text{Using formula } C[i,i-1]=0 \text{ and } C[n+1,n]=0$$

$$\left.\begin{aligned} C[1,1] &= 0.4 \\ C[2,2] &= 0.5 \\ C[3,3] &= 0.1 \\ C[4,3] &= 0 \end{aligned}\right\} \text{Using formula, } C[i,i]=P[i]$$

The root table

$$\left.\begin{aligned} R[1,1] &= 1 \\ R[2,2] &= 2 \\ R[3,3] &= 3 \end{aligned}\right\} \text{Using formula, } R[i,i]=i$$

Now let us compute C[i, j] diagonally using formula,

$$C[i, j] = \min_{i \le k \le j} \{C[i,k-1]+C[k+1,j]\} + \sum_{S=i}^{j} P_S \qquad \ldots(Q.1.1)$$

To compute C[1, 2] the k can be either 1 or 2.

Let i = 1, j = 2, we use formula in equation (Q.1.1),

C [1,2] =

k = 1 → $C[i, k-1] + C[k+1, j] + \sum_{S=i}^{j} P_S$

$C[1, 0] + C[2, 2] + P[1] + P[2]$

$0 + 0.5 + 0.4 + 0.5 = 1.4$

k = 2 → $C[i, k-1] + C[k+1, j] + \sum_{S=i}^{j} P_S$

$C[1, 1] + C[3, 2] + P[1] + P[2]$

$0 + 0 + 0.4 + 0.5 = 0.9$ ← **Minimum value**

∴ Consider **k = 2**

POCO M3 | ❤️❤️ DIPS ❤️❤️

C[1, 2] = 0.9 and R[1, 2] = 2

Now consider C[2,3]

∴ i = 2, j = 3 with k = 2 and k = 3

C [2, 3] =

- k = 2 → $C[i, k-1] + C[k+1, j] + \sum_{S=i}^{j} P_S$

  C[2, 1] + C[3, 3] + P[2] + P[3]

  0 + 0.1 + 0.5 + 0.1 = 0.7 ← Minimum value

- k = 3 → $C[i, k-1] + C[k+1, j] + \sum_{S=i}^{j} P_S$

  C[2, 2] + C[4, 3] + P[2] + P[3]

  0.5 + 0 + 0.5 + 0.1 = 1.1

C[2, 3] = 0.7 and R[2, 3] = k = 2

	0	1	2	3
1	0	0.4	0.9	
2		0	0.5	0.7
3			0	0.1
4				0

Cost table (C)

	1	2	3
1	1	2	
2		2	2
3			3

Root table (R)

Now consider C[1, 3]

i = 1, j = 3 with k = 1, k = 2 or k = 3

C [1, 3] =

- k = 1 → C[1, 0] + C[2, 3] + P[1] + P[2] + P[3]

  = 0 + 0.7 + 0.4 + 0.5 + 0.1

  = 1.7

- k = 2 → C[1, 1] + C[3, 3] + P[1] + P[2] + P[3]

  = 0.4 + 0.1 + 0.4 + 0.5 + 0.1

  = 1.5 ← Minimum value with **k = 2**

- k = 3 → C[1, 2] + C[4, 3] + P[1] + P[2] + P[3]

  = 0.9 + 0 + 0.4 + 0.5 + 0.1

  = 1.9

Hence C[1, 3] = 1.5 and R[1, 3] = 2

	0.	1	2	3
1	0	0.4	0.9	1.5
2		0	0.5	0.7
3			0	0.1
4				0

Cost table (C)

	1	2	3
1	1	2	2
2		2	2
3			3

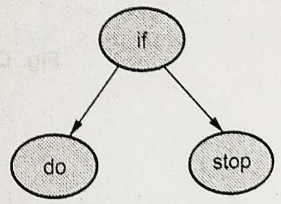
Root table (R)

As **R[1, 3] = 2, Key = 2**

	1	2	3
a [i]	do	if	stop

→ Key = 2. Then that means "if" is a root.

The optimal binary search is -



The tree has cost = **C[1, 3] = 1.5,** this is optimum cost.

The expected access time for this OBST is $O(n^3)$

Q.12 Enlist the names of static tree tables with suitable example.

[SPPU : Dec.-13, Marks 2]

Ans. : 1) A reserved word table used in a compiler is a static tree table.

2) Optimal binary search tree is an example of static tree table. For example - Refer final tree from Q.11.

4.6 : Height Balanced Tree- AVL Tree

Q.13 What is height balance tree? Explain with one example.

[SPPU : Dec.-10, Marks 4]

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.

The idea of balancing a tree is obtained by calculating the balance factor of a tree.

Definition of Balance Factor

The balance factor BF(T) of a node in binary tree is defined to be $h_L - h_R$ where h_L and h_R are heights of left and right subtrees of T.

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

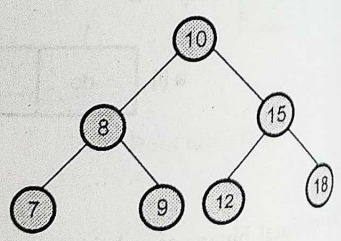


Fig. Q.13.1 AVL tree

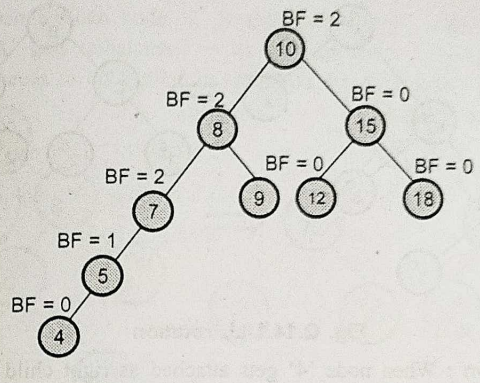


Fig. Q.13.2 Not an AVL tree

Q.14 Write an algorithm to insert node into AVL tree.

[SPPU : May-10, 12, Marks 8, June-22, Dec.-22, Marks 5]

Ans. : Insertion algorithm

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 either a single rotation or a double rotation.

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

Different rotations in AVL tree

1. LL rotation : When node '1' gets inserted as a left child of node 'C' then AVL property gets destroyed i.e. node A has balance factor + 2. The LL rotation has to be applied to rebalance the nodes.

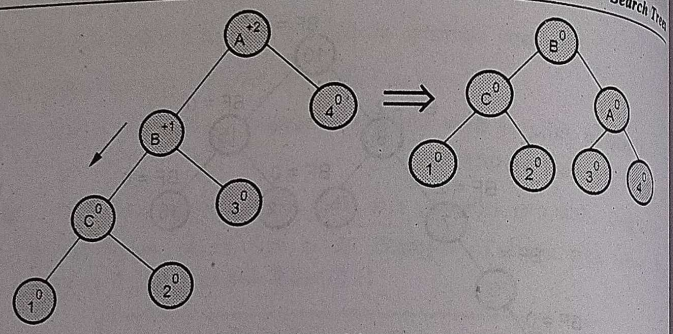


Fig. Q.14.1 LL rotation

2. RR rotation : When node '4' gets attached as right child of node 'C' then node 'A' gets unbalanced. The rotation which needs to be applied is RR rotation as shown in Fig. Q.14.2.

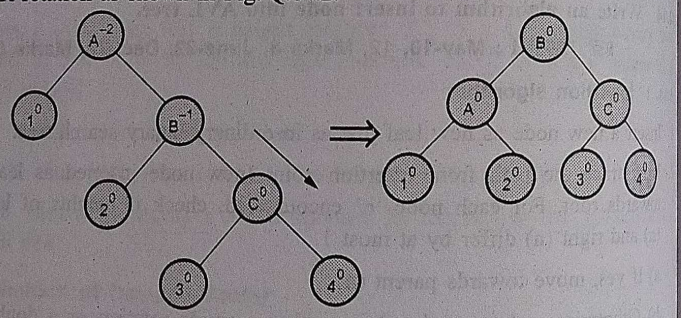


Fig. Q.14.2 RR rotation

3. LR rotation : When node '3' is attached as a right child of node 'C' then unbalancing occurs because of LR. Hence LR rotation needs to be applied.

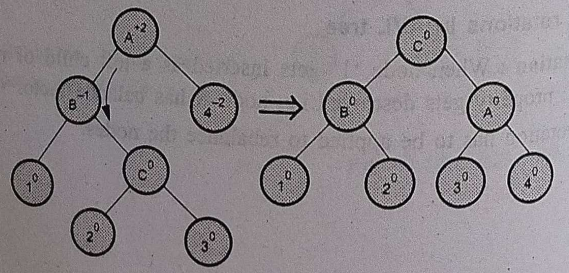


Fig. Q.14.3 LR rotation

4. RL rotation : When node '2' is attached as a left child of node 'C' then node 'A' gets unbalanced as its balance factor becomes – 2. Then RL rotation needs to be applied to rebalance the AVL tree.

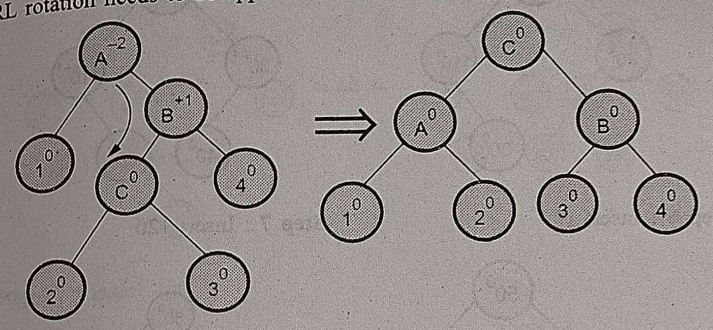


Fig. Q.14.4 RL rotation

Q.15 Explain with examples LL, LR, RR and RL rotations for AVL tree ☞ **[SPPU : May-11,June-22, Marks 6, May-12, Marks 8]**

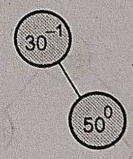
Ans. : Refer Q.14.

Q.16 Obtain AVL trees for the following data :
30, 50, 110, 80, 40, 10, 120, 60, 20, 70, 100, 90

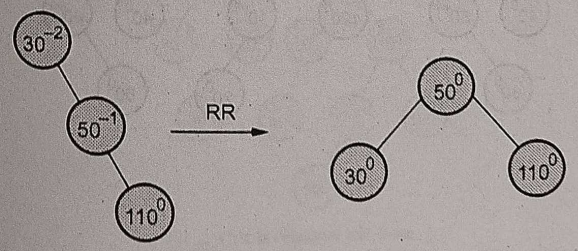
☞ **[SPPU : Dec.-17, Marks 6]**

Ans. : Step 1 :

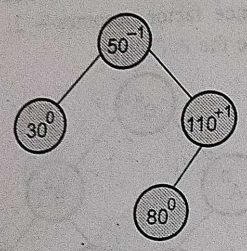
Step 2 :



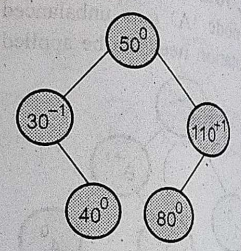
Step 3 : Insert 110



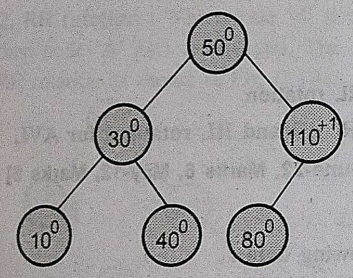
Step 4 : Insert 80



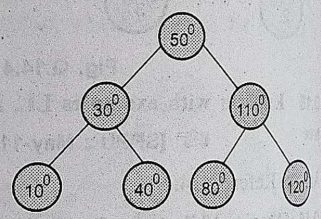
Step 5 : Insert 40



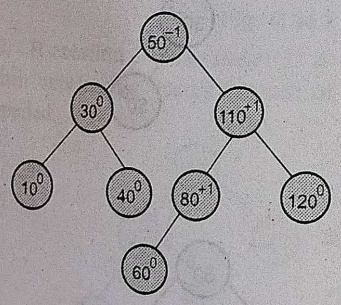
Step 6 : Insert 10



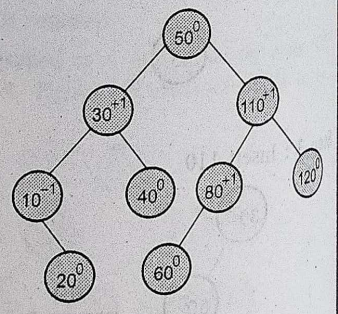
Step 7 : Insert 120



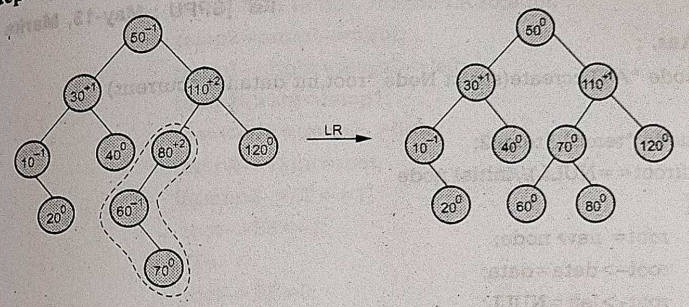
Step 8 : Insert 60



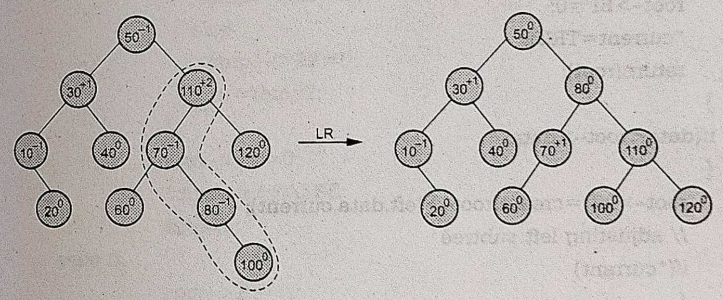
Step 9 : Insert 20



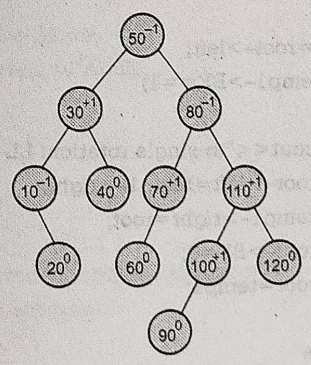
Step 10 : Insert 70



Step 11 : Insert 100



Step 12 : Insert 90



This is required AVL Tree

Q.17 Write pseudo C/C++ code for LL, LR, RR and RL.

[SPPU : May-13, Marks 8]

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.18 Write and explain algorithm to delete node from AVL tree.

[SPPU : Dec.-12, Marks 8, Dec.-22, Marks 6]

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.

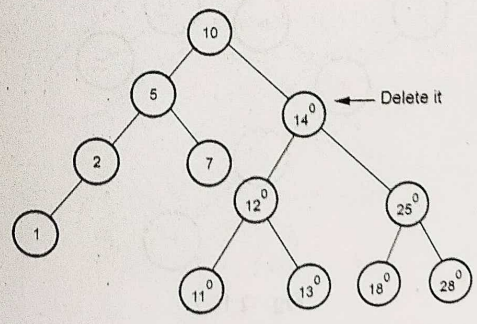


Fig. Q.18.1

The tree becomes

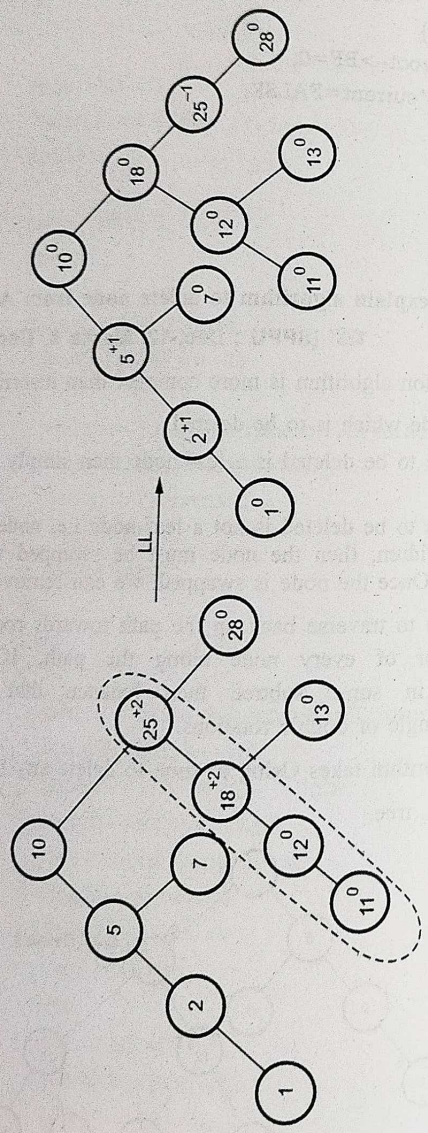


Fig. Q.18.2

Thus the node 14 gets deleted from AVL tree.

Q.19 Define AVL tree. Obtain an AVL tree by accepting the following numbers (keys) one at a time. Show balance factor of each node, the type of rotation and the transformation required for each insertion in AVL tree.
30, 31, 32, 23, 22, 28, 24, 29, 26, 27, 34, 36.

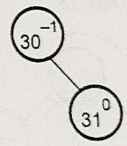
☞ **[SPPU : Dec.-08, May-10, Marks 10]**

Ans. : An AVL tree is a height balanced tree in which every node has a balance factor which is 0, –1 or +1.

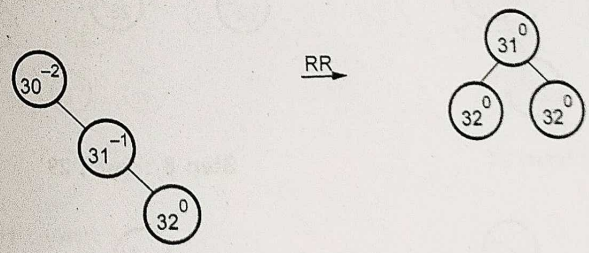
Example

Step 1 : Insert 30

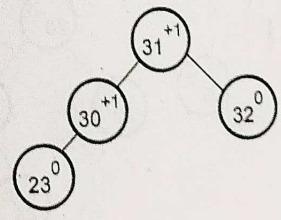
Step 2 : Insert 31



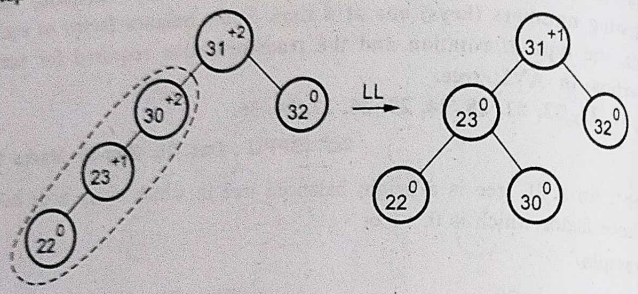
Step 3 : Insert 32



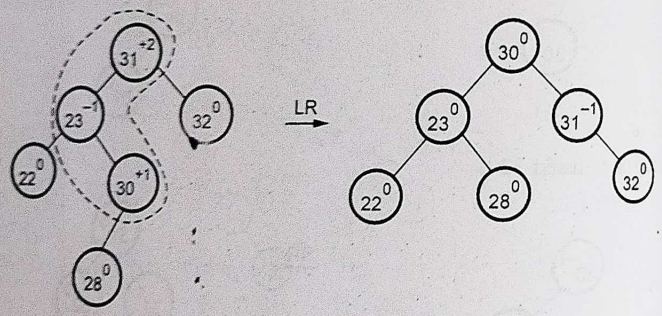
Step 4 : Insert 23



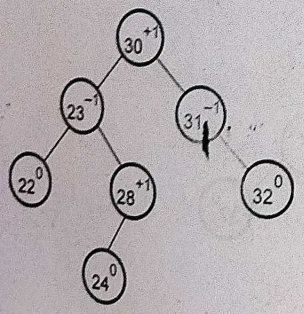
Step 5 : Insert 22



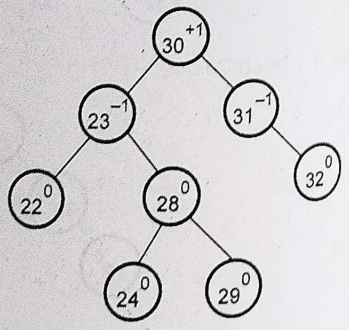
Step 6 : Insert 28



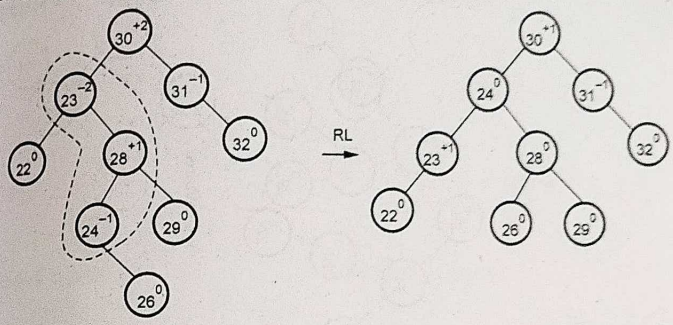
Step 7 : Insert 24



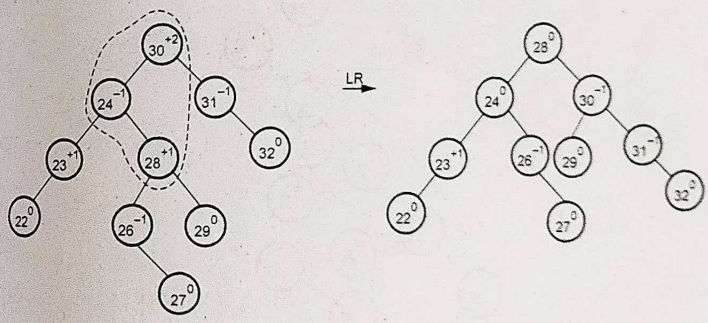
Step 8 : Insert 29



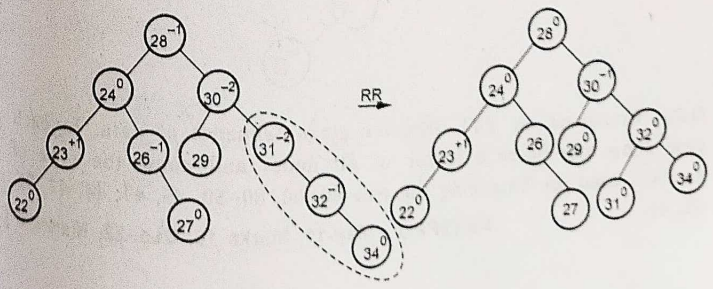
Step 9 : Insert 26



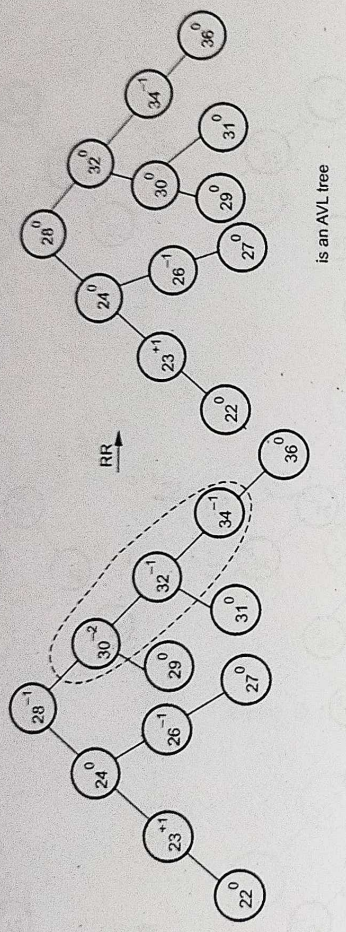
Step 10 : Insert 27



Step 11 : Insert 34



Step 12 : Insert 36



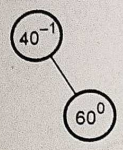
Q.20 Construct an AVL tree for given sequence of data. In each case show the balance factor of all nodes and name the type of rotation used for balancing the tree 40, 60, 80, 50, 45, 47, 44, 42, 75, 46, 41.

☞ **[SPPU : May-11, Marks 10, Dec.-22, Marks 6]**

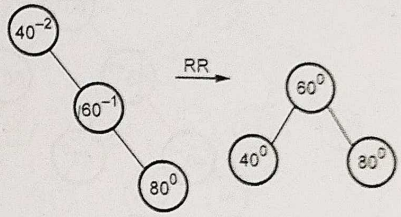
Ans. :

Step 1 :

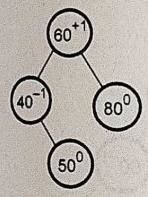
Step 2 : Insert 60



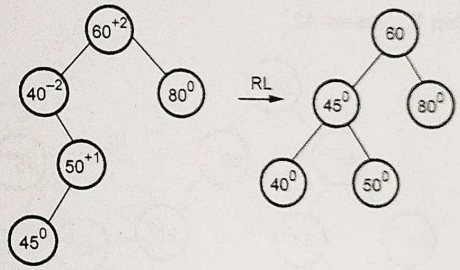
Step 3 : Insert 80



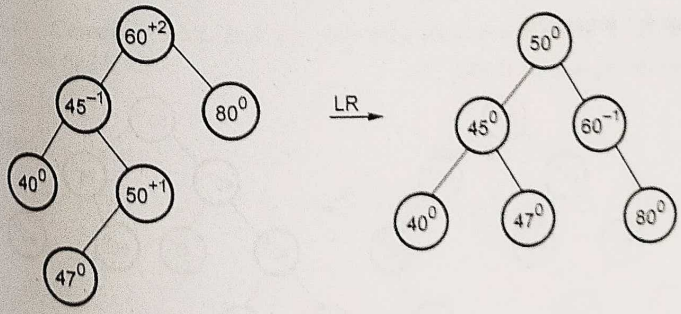
Step 4 : Insert 50



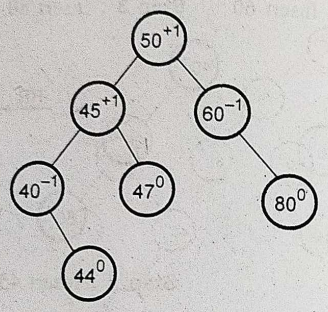
Step 5 : Insert 45



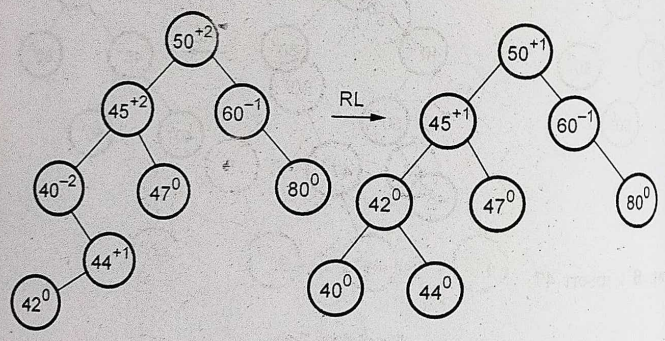
Step 6 : insert 47



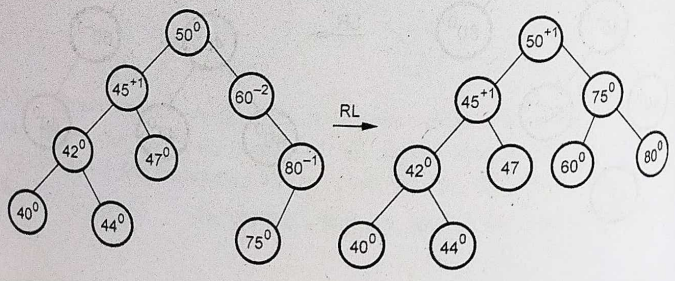
Step 7 : Insert 44



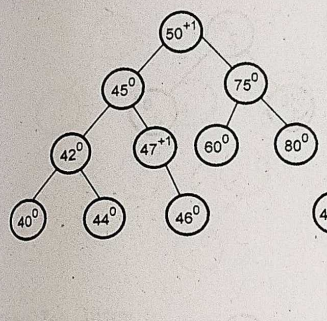
Step 8 : Insert 42



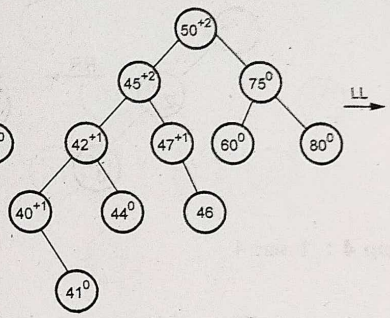
Step 9 : Insert 75

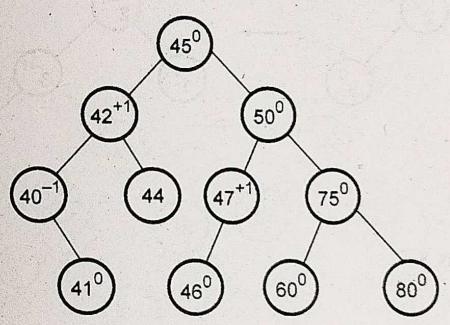


Step 10 : Insert 48



Step 11 : Insert 41





Final AVL Tree

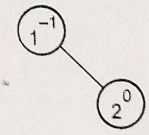
Q.21 Construct AVL tree for following sequence of keys. 1, 2, 3, 4, 8, 7, 6, 5, 11, 10

☞ **[SPPU : June-22, Marks 6]**

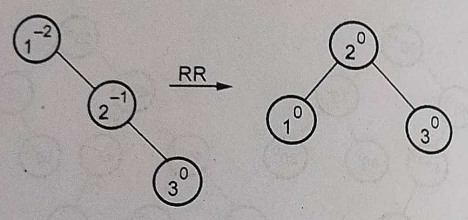
Ans. :

Step 1 : Insert 1

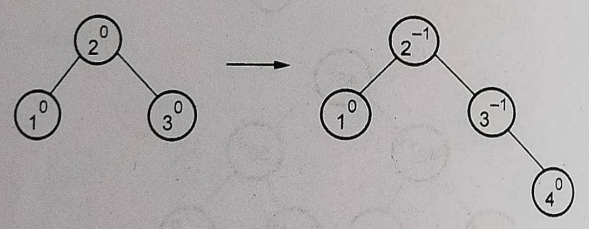
Step 2 : Insert 2



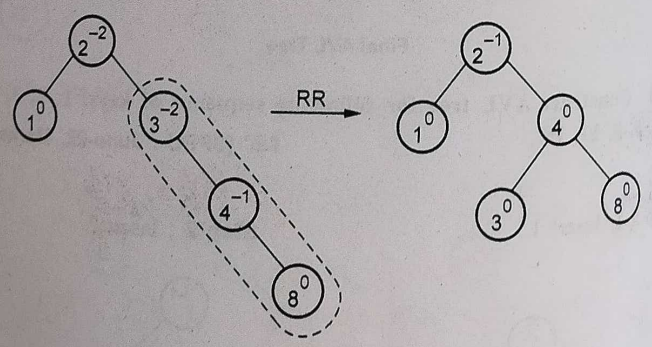
Step 3 : Insert 3



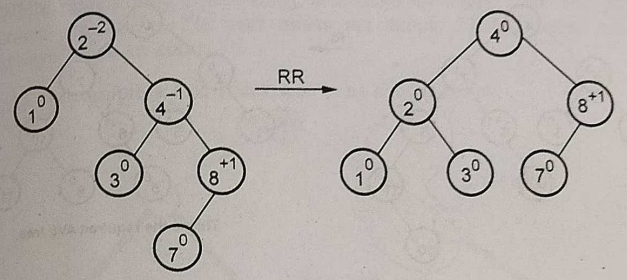
Step 4 : Insert 4



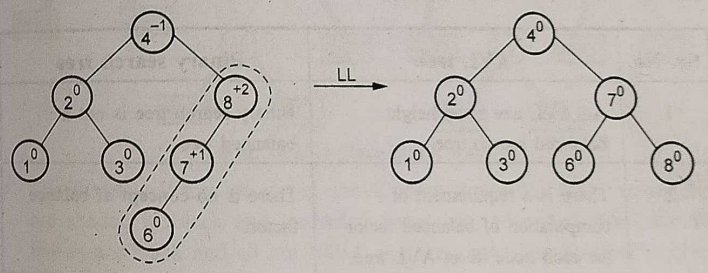
Step 5 : Insert 8



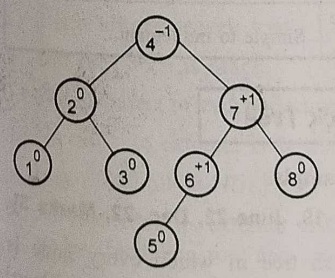
Step 6 : Insert 7



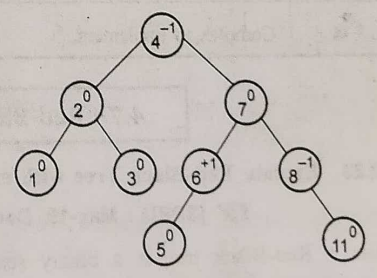
Step 7 : Insert 6



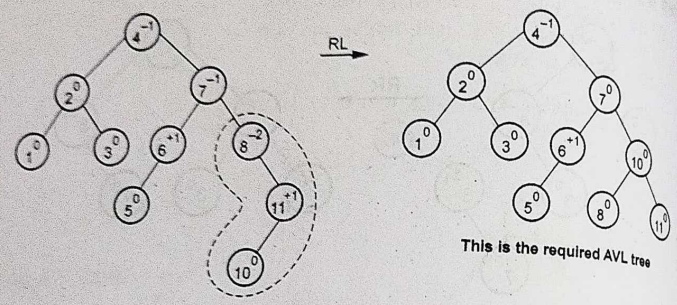
Step 8 : Insert 5



Step 9 : Insert 11



Step 10 : Insert 10



Q.22 Compare AVL tree with Binary Search Tree.

Ans. :

Sr. No.	AVL tree	Binary search tree
1.	An AVL tree is an height balanced search tree.	Binary search tree is not a balanced tree.
2.	There is a requirement of computation of balanced factor for each node in an AVL tree.	There is no concept of balance factor.
3.	Searching of any desired node is faster due to balancing of height.	Searching is not efficient when there are large number of nodes in the tree.
4.	Complex to implement.	Simple to implement.

4.7 : Red-Black Tree

Q.23 Explain Red-Black Tree with example.

[SPPU : May-19, Dec.-19, June-22, Dec.-22, Marks 3]

Ans. : Red-Black tree is a binary search tree in which every node is colored with either **red** or **black**. It is a type of self balancing binary search tree. It has a good efficient worst case running time complexity.

Representation :

- While representing a Red-Black tree color of every node and pointer colors are shown. The **leaf nodes** are simply NULL nodes and are always **black in color.**
- As an example a Red-Black tree is as shown below.

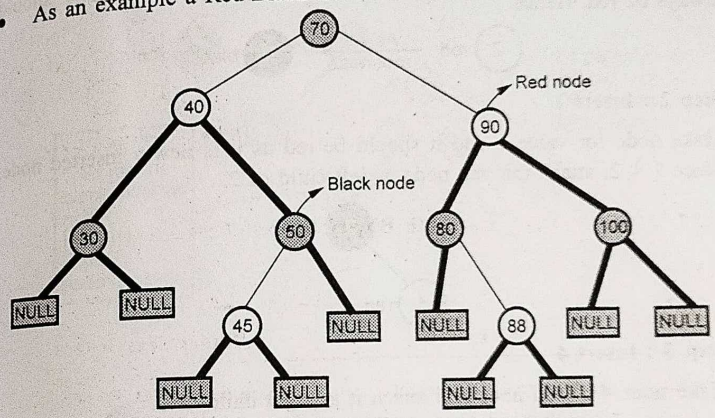


Fig. Q.23.1 Red-Black tree

- The Red-Black tree shown in above given figure has black nodes that are shaded in black and unshaded nodes are red nodes. Similarly the leaves are black and all are NULL pointers only. The pointers to black node are black pointers which are shown by thick lines remaining are red pointers. But in practice, we explicitly mention the color of nodes. The above given Red-Black tree follows all the **properties of Red-Black tree -**

1. It is a **binary search tree**.
2. The **root** node is **black**.
3. The **children of red node are black**.
4. No root - to external node path has two consecutive red nodes (e.g. 70-90-80-88-NULL).
5. All the root to external node paths contain same number of black nodes (including root and external node).

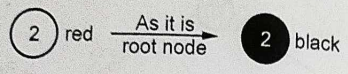
- For e.g. : Consider path 70-40-30-NULL and 70-90-80-88-NULL in both these paths 3 black nodes are there.

Q.24 Insert 2, 1, 4, 5, 9, 3, 6 and 7 for Red black tree.

Ans. :

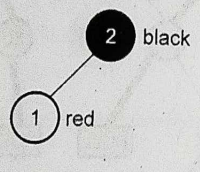
Step 1 : Insert 2

Initially the red black tree is empty. The newly inserted node should always be red. Hence.



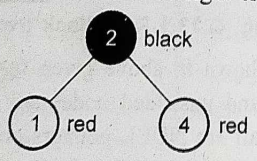
Step 2 : Insert 1

Make node for value 1 and it should be red as it is newly inserted node. Since 1 < 2, attach this red node as left child of 2.



Step 3 : Insert 4

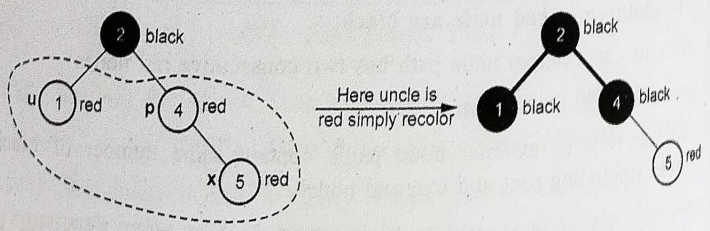
Make node 4 as red node and attach it as right child of 2.



Step 4 : Insert 5

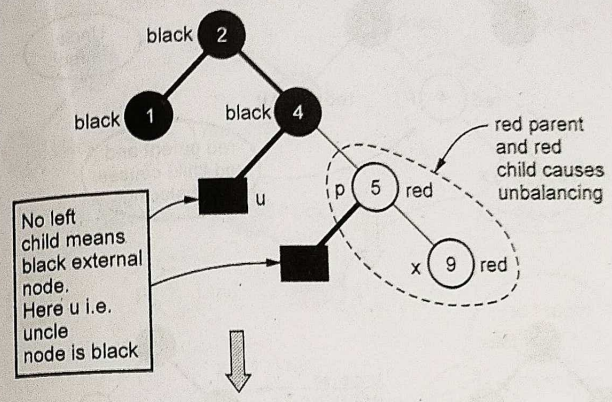
Here node 5 is a red node attached as right child of red node 4.

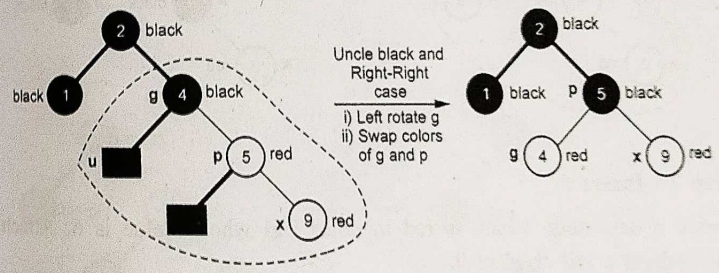
But there should not be red child of red node. Hence we need to make adjustments.



Step 5 : Insert node 9

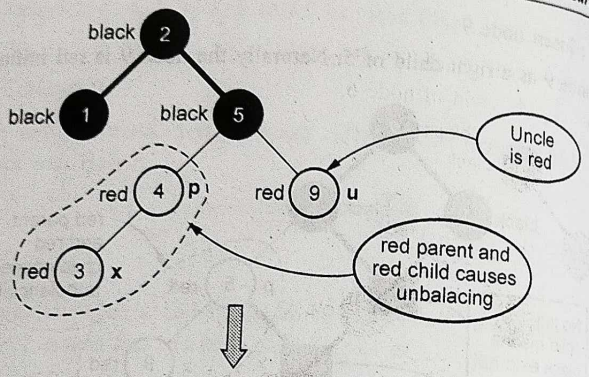
Insert node 9 as a right child of 5. Naturally the node 9 is red initially.

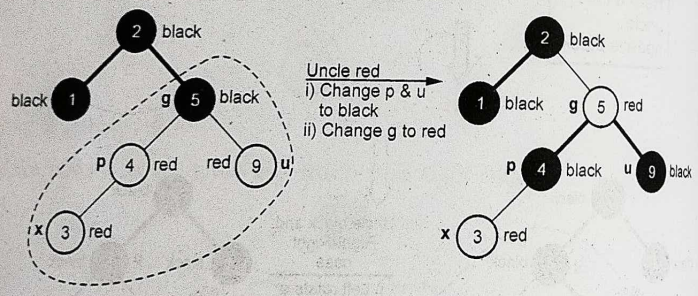




Step 6 : Insert 3

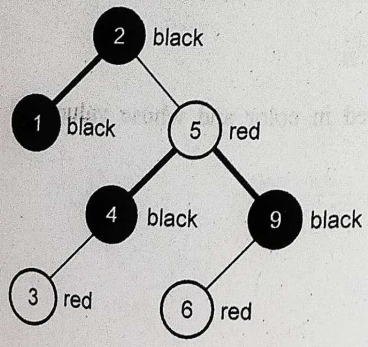
Create a new node which is red in color and whose value is 3. Attach it as left child of 4.





Step 7 : Insert 6

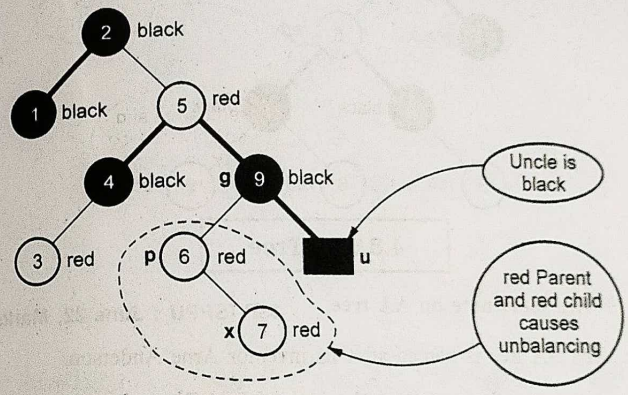
Create a new node which is red in color and whose value is 6. Attach this node as a left child of 9.

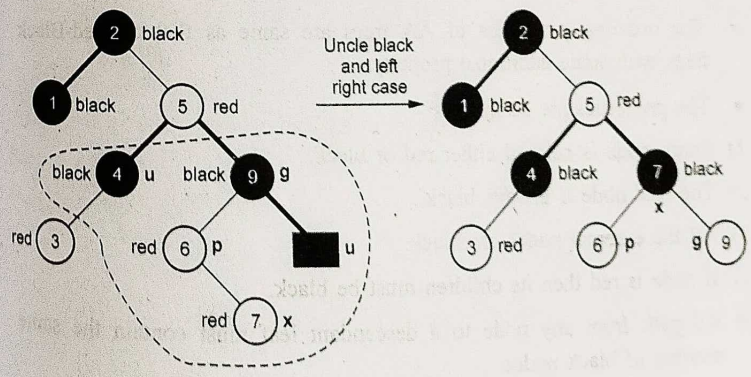


As here parent node i.e. 9 is black and child node i.e. 6 is red, there is no need for adjustment.

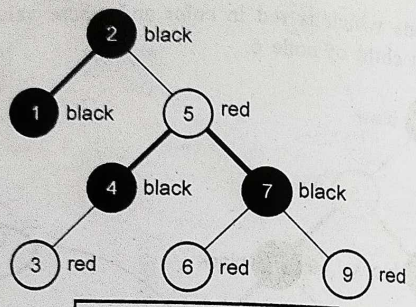
Step 8 : Insert 7

Create a new node which is red in color and whose value is red. Attach this node as right child of node 6.





Thus finally we get, following Red black tree.



4.8 : AA Tree

Q.25 Write short note on AA tree. ☞ **[SPPU : June-22, Marks 3]**

Ans. : The AA tree is named after its inventor Arne Anderson.

- The AA tree is created from the idea of Red Black tree, but contains fewer rotations and less complex.
- The time complexity for operations of AA tree is **O(log n).**

Properties of AA trees

- The ordering properties of AA trees are same as that of Red-Black trees with some additional property.
- The properties are as follows -

1) Every node is colored either red or black.
2) The root node is always black.
3) All the external nodes are black.
4) If node is red then its children must be black.
5) All path from any node to a descendant leaf must contain the same number of black nodes.
6) The left child may not be red.

For example

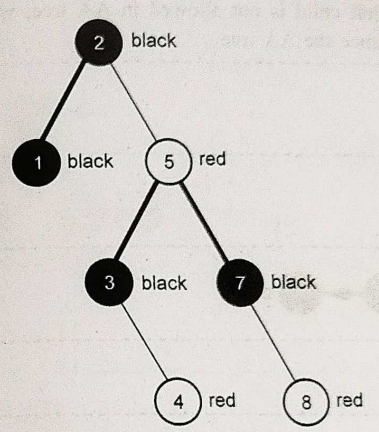
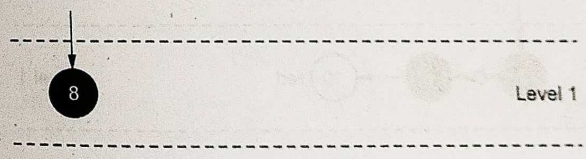


Fig. Q.25.1 AA tree

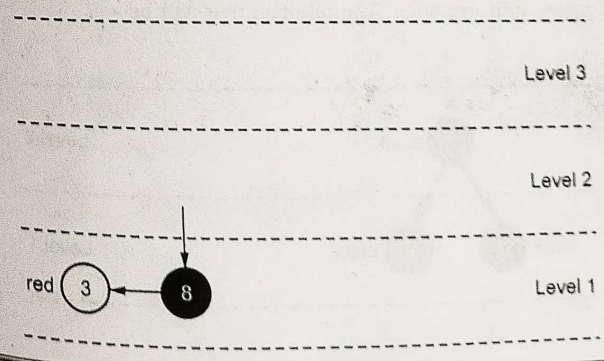
Q.26 Insert the elements 8, 3, 10, 18, 12, 2 in AA tree.

Ans. : We will insert the elements one by one. We will apply skew and split operations when horizontal links cause unbalancing in the tree.

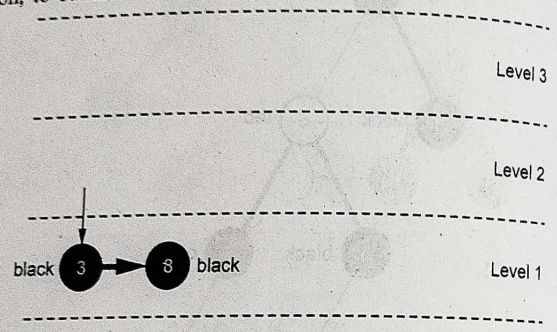
Step 1 : Insert 8



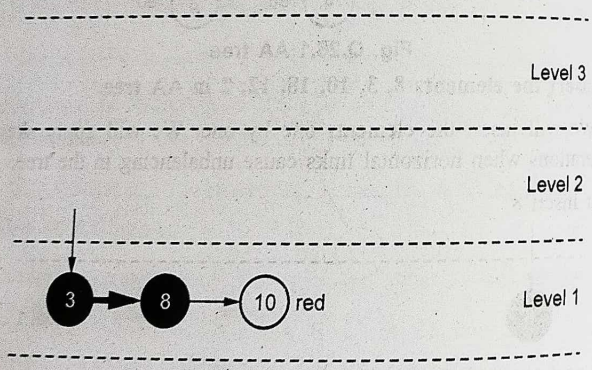
Step 2 : Insert 3



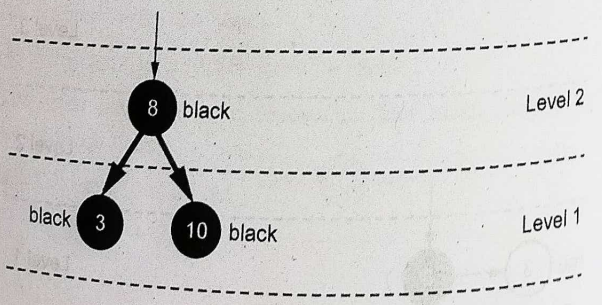
But since right left child is not allowed in AA tree, we will apply skew operation, to balance the AA tree



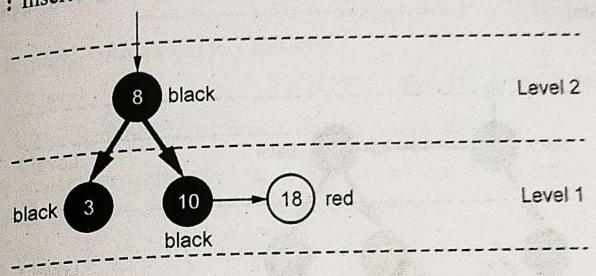
Step 3 : Insert 10



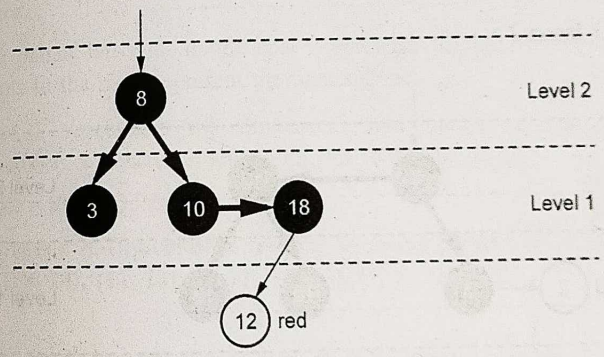
Here two consecutive horizontal links occur. This is not allowed. Hence we will apply **split** operation. The resultant tree will be -



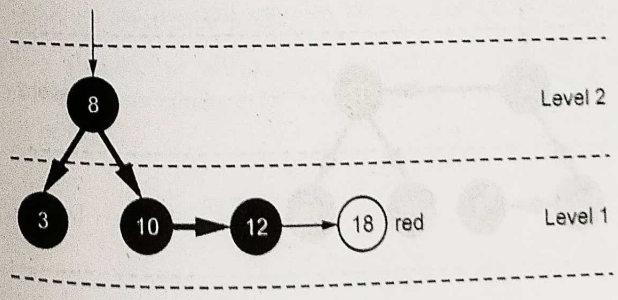
Step 4 : Insert 18



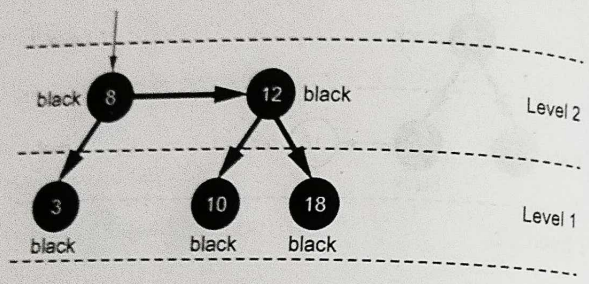
Step 5 : Insert 12



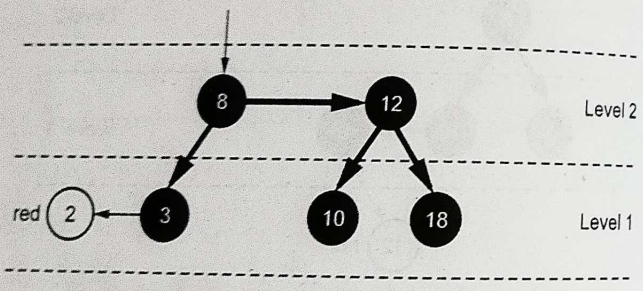
Again left link from 18 to 12 is not allowed. Hence we apply skew operation. The balanced AA tree will then be,



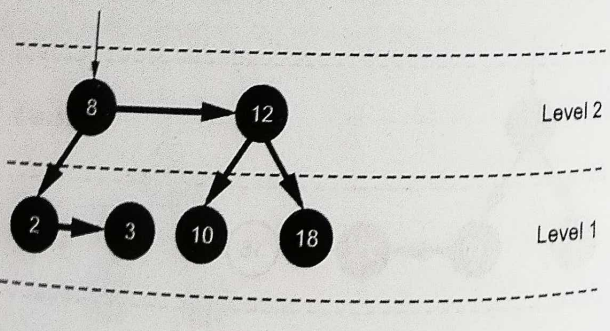
But still two horizontal links are not allowed. Hence we will apply split operation.



Step 6 : Insert 2



The left horizontal link is not allowed. Hence we apply skew operation.



hus, the AA tree is created.

4.9 : K-dimensional Tree

Q.27 Explain with example - KD tree.

☞ **[SPPU : May-19, June-22, Marks 3]**

Ans. : • The K-dimensional tree is also known as **KD tree.**

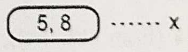
- It is a data structure used in Computer science for organizing some number of points in a space with K dimensions.
- It is basically a **binary search tree** with some constraints.
- The points are arranged in a space which is called as **K-Dimensional space.**
- The points left side in the space represents the left subtree and the points in the right represent the right subtree.
- The K represents the dimensions. That means a tree with 2 dimension, 3 dimension and so on can be drawn.

Example - Refer Q.28.

Q.28 Draw a K-Dimensional tree for following points.
(5, 8), (18, 14), (12, 14), (8, 11), (10, 2), (3, 6), (11, 20).

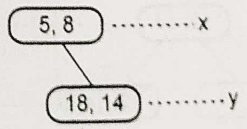
Ans. :

Step 1 : We will read the point **(5, 8)** make it as root node. The root is said to be x-aligned.



Step 2 : Now the next point is **(18, 14).**

As root is x aligned, we will compare x value of (5, 8) with x value of (18, 14). That means compare 18 with 5. As 18 > 5, make (18, 14) as right child of (5, 8).

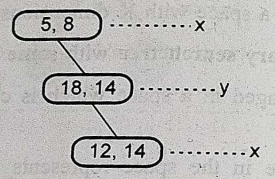


This node is y-aligned.

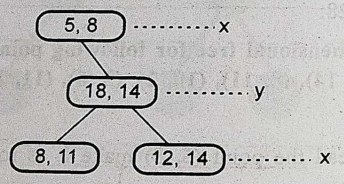
Step 3 : Now read **(12, 14).** We will start from root node. The x-value of (5, 8) is compared with x-value of (12, 14).

As 12 > 5, move on to right branch.

Here (18, 14) is y-aligned. Hence the y-value of (18, 14) is compared with y-value of (12-14). As 14 = 14 move on to right branch and attach (12, 14) as right child of (18,14).



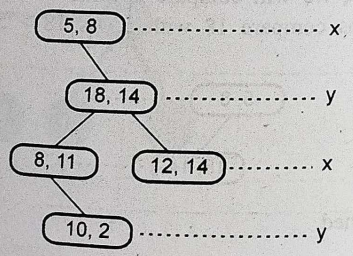
Step 4 : The next point is **(8,11).** Starting from root we will compare x and y values of (8, 11) alternatively and attach (8, 11) as left child of (18, 14). The partial construction is as follows -



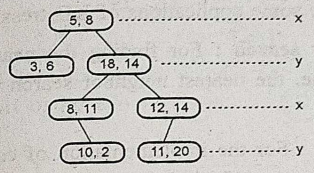
Step 5 : Next point is **(10, 2).**

10 > 5 (From 5, 8)), 10 < 14 (from (18, 14)),

10 > 8 (from 8, 11)). Hence attach (10, 2) as a right child of (8, 11).



Continuing in this fashion, we get following tree -



Step 6 : These points can be plotted on X-Y plane.

i) For point (5, 8), X = 5. Hence draw a vertical line at x = 5.

ii) For point (3, 6), Y = 6. Hence draw a horizontal line at Y = 6. But as (3, 6) is a left child of (5, 8), draw this line at the left of x = 5 line.

In this way, the lines X and Y can be drawn to divide the space. The K-D space spotting is as shown below -

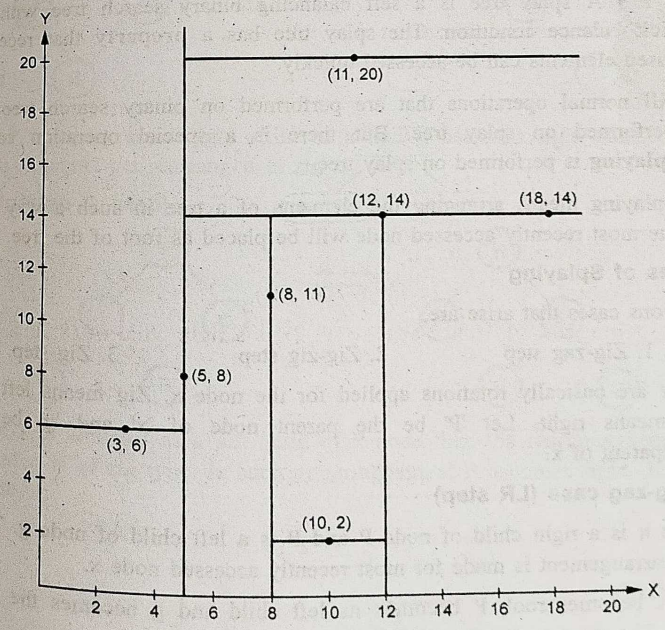


Fig. Q.28.1 K-D space

Q.29 Enlist the applications of KD trees.

Ans. : Following are some applications of KD tress -

1. **Nearest neighbor search :** For finding the nearest police station for reporting the crime, the nearest neighbor search is conducted with the help of K-D tree.
2. **Database queries :** For the implementation of complex queries, which involves multidimensional search key, the query problem is converted geometrical problem in order to get the answer.

4.10 : Splay Tree

Q.30 Explain with example - Splay tree.

☞[SPPU : May-19, Dec.-19, June-22, Dec.-22, Marks 3]

Ans. : • A splay tree is a self balancing binary search tree with no explicit balance condition. The splay tree has a **property** that recently accessed elements can be accessed quickly.

- All normal operations that are performed on binary search tree are performed on splay tree. But there is a special operation called **splaying** is performed on splay tree.
- Splaying means arranging the elements of a tree in such a way that the most recently accessed node will be placed as root of the tree.

Cases of Splaying

Various cases that arise are,

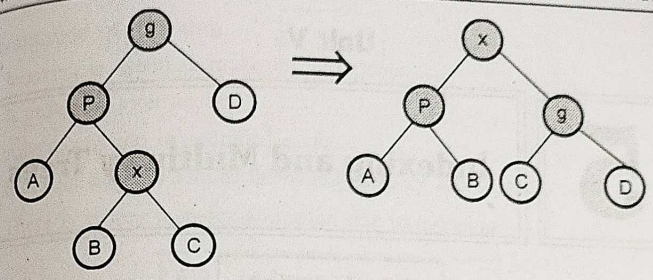
1. Zig-zag step 2. Zig-zig step 3. Zig step.

These are basically rotations applied for the node x. Zig means left and zag means right. Let 'P' be the parent node of 'x' and 'g' be the grandparent of x.

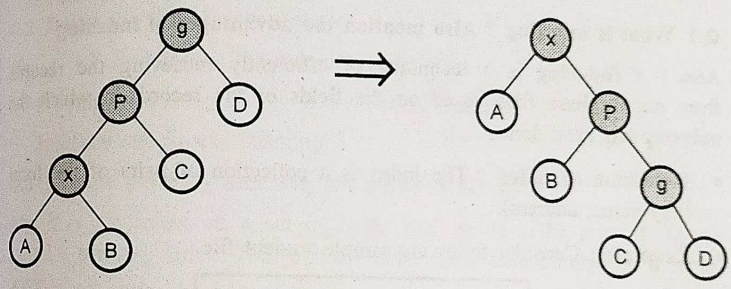
1. Zig-zag case (LR step)

When x is a right child of node P and P is a left child of node g. Then the rearrangement is made for most recently accessed node x.

The x becomes root, P becomes its left child and g becomes the right child. The zig-zag step is equivalent to rotation about x.

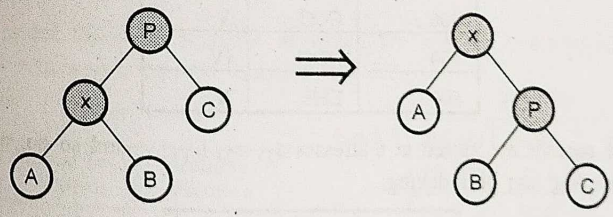


2. Zig-zig step (LL step)



The P node becomes right child of x and g becomes right child of P, when elements get rearranged in zig-zig step.

3. Zig step (L step)



When x is attached as a left child of P node then the rearrangement that is needed is of zig type. In such a rearrangement P becomes right child of x node.

END...

Unit V

5 Indexing and Multiway Trees

5.1 : Indexing

Q.1 What is indexing ? Also mention the advantages of indexing

Ans. : • **Indexing** is a technique of efficiently retrieving the records from the database files based on the fields of the record on which the indexing has been done.

- **Definition of index :** The index is a collection of pairs of the form (key value, address).
- **Example :** Consider following sample student file

Roll No.	Name	Std.
100	AAA	IX
200	BBB	VI
300	CCC	V
400	DDD	IV
500	EEE	X

If these records are stored at addresses $a_1, a_2, a_3, \ldots$ and so on. on disk then indexing can be indexing.

Name	disk Address
AAA	a_1
BBB	a_2
CCC	a_3
DDD	a_4
EEE	a_5

POCO M3 | ❤️❤️DIPS ❤️❤️

Advantages of Indexing

1) Indexing is useful in managing large data.
2) It provides faster access.
3) Any record can be searched easily.
4) Indexing helps to reduce unwanted memory access.
5) There is proper memory allocation due to indexing.

5.2 : Indexing Techniques

Q.2 Enlist the indexing techniques

Ans. : • Following are the indexing techniques -

1. Primary indexing 2. Secondary indexing
3. Dense and Sparse indexing

Q.3 Explain primary indexing technique in detail

Ans. : • An index on a set of fields that includes the primary key is called a primary index. The primary index file should be always in sorted order.

- The primary indexing is always done when the data file is arranged in sorted order and primary indexing contains the primary key as its search key.
- Consider following scenario in which the primary index consists of few entries as compared to actual data file.

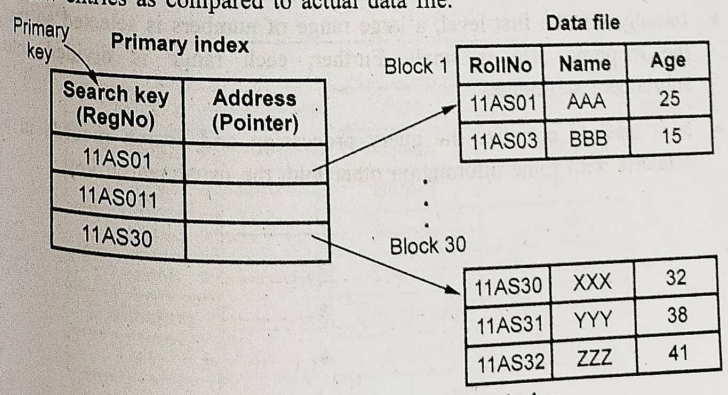


Fig. Q.3.1 Example of primary index

- Once if you are able to locate the first entry of the record containing block, other entries are stored continuously. For example if we want to search a record for RegNo 11AS32 we need not have to search for the entire data file. With the help of primary index structure we come to know the location of the record containing the RegNo 11AS30, now when the first entry of block 30 is located, then we can easily locate the entry for 11AS32.
- We can apply binary search technique. Suppose there are n = 300 blocks in a main data file then the number of accesses required to search the data file will be $\log_2 n + 1 = (\log_2 300) + 1 \approx 9$.
- If we use primary index file which contains at the most n = 3 blocks then using binary search technique, the number of accesses required to search using the primary index file will be $\log_2 n + 1 = (\log_2 3) + 1 \approx 3$.
- This shows that using primary index the access time can be deduced to great extent.

Q.4 Explain secondary indexing technique with suitable example

Ans. :

- In this technique two levels of indexing are used in order to reduce the mapping size of the first level.
- Initially, for the first level, a large range of numbers is selected so that the mapping size is small. Further, each range is divided into subsequent sub ranges.
- It is used to optimize the query processing and access records in a database with some information other than the usual search key.

For example -

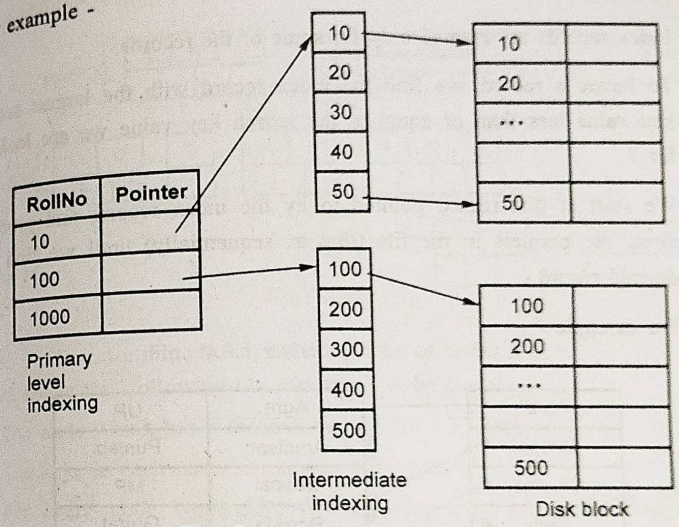


Fig. Q.4.1 Secondary indexing

Q.5 Explain dense and sparse indexing techniques with illustrative examples

Ans. : 1) Dense index :

- An index record appears for every search key value in file.
- This record contains search key value and a pointer to the actual record.
- For example :

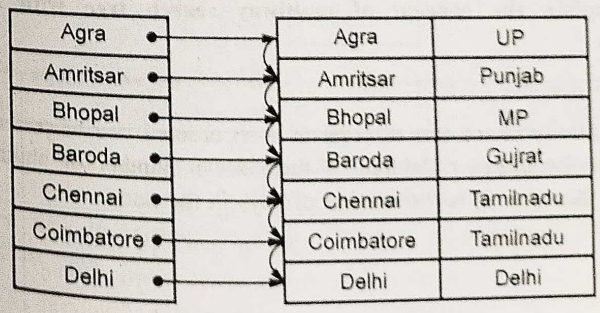


Fig. Q.5.1 : Dense index

2) Sparse Index :

- Index records are created only for some of the records.
- To locate a record, we find the index record with the largest search key value less than or equal to the search key value we are looking for.
- We start at that record pointed to by the index record, and proceed along the pointers in the file (that is, sequentially) until we find the desired record.
- For example -

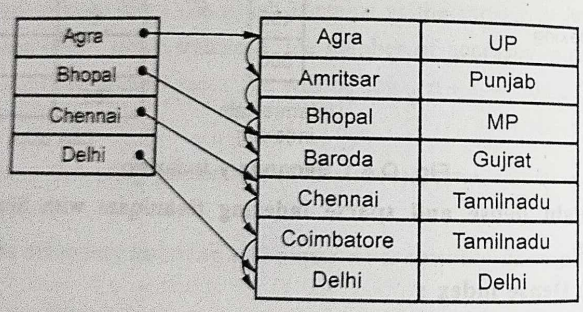


Fig. Q.5.2 : Sparse index

5.3 : Multiway Search Trees

Q.6 Explain the concept of multiway search tree with suitable example

Ans. :

- A multiway search tree of order m is an ordered tree where each node has at the most m children. If there are n number of children in a node then (n – 1) is the number of keys in the node.

For example :

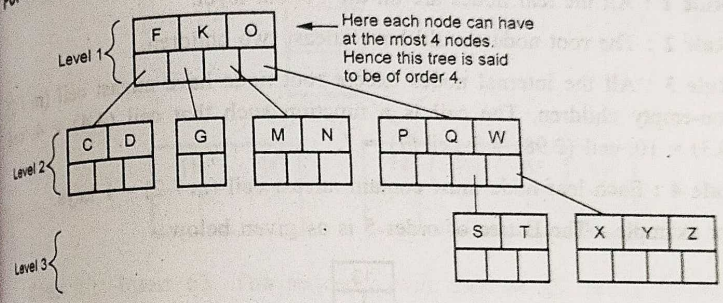


Fig. Q.6.1 Multiway tree of order 4

From above tree following observations can be made -

1. The node which has n children possess (n – 1) keys.
2. The keys in each node are in ascending order.
3. For every node Node.child [0] has only keys which are less than Node.key [0] similarly, Node.child [1] has only keys which are greater than Node.key [0].

- In other words, the node at level 1 has F, K and O as keys. At level 2 the node containing keys C and D are arranged as child of key F (the cell before F). Similarly the node containing key G will be child of F but should be attached after F and before K, as G is between F and K. Here the alphabets F, K, O are called keys and branches are called children.

5.4 : B-Tree

Q.7 Write short note on B-tree. ☞ **[SPPU : Dec.-09, Marks 4]**

Ans. : • B-tree is a specialized multiway tree used to store the records in a disk. There are number of subtrees to each node. So that the height of the tree is relatively small. So that only small number of nodes must be read from disk to retrieve an item. The goal of B-trees is to get fast access of the data.

The B-tree of order m should be constructed such that -

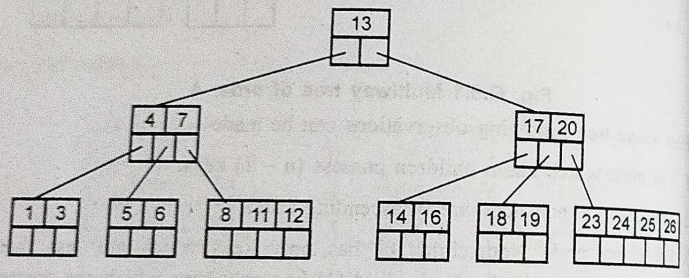
Rule 1 : All the leaf nodes are on the bottom level.

Rule 2 : The root node should have atleast two children.

Rule 3 : All the internal nodes except root node have atleast ceil (m / 2) non-empty children. The ceil is a function such that ceil (3.4) = 4, ceil (9.3) = 10, ceil (2.98) = 3 ceil (7) = 7.

Rule 4 : Each leaf node must contain atleast ceil (m / 2) – 1 keys.

For example - The B tree of order 5 is as given below -

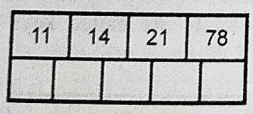


Q.8 Explain the steps to built a B-tree of order 5 for following data : 78, 21, 14, 11, 97, 85, 74, 63, 45, 42, 57, 20, 16, 19, 52, 30, 21.

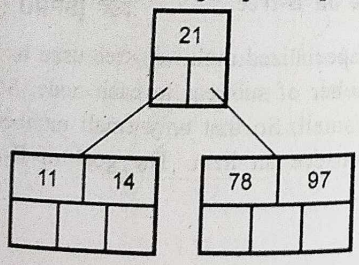
☞ **[SPPU : Dec.-05, May-07, June-22, Marks 6]**

Ans. : The order 5 means at the most 4 keys are allowed.

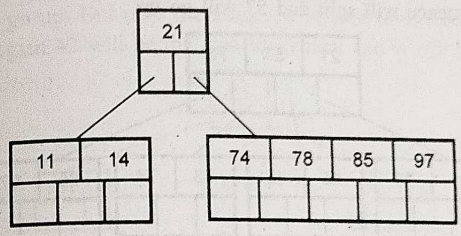
Step 1 : Insert 78, 21, 14, 11 as follows -



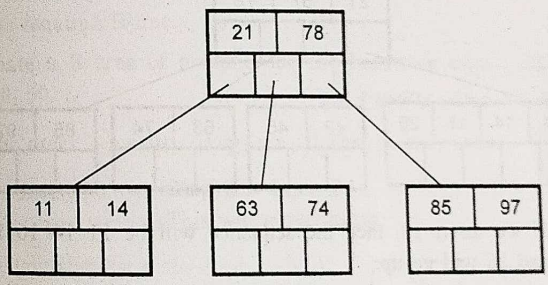
Step 2 : If we insert 97 then the sequence becomes 11 14 21 78 97. Then, the sequence will split, 21 will go up. The partial tree will be -



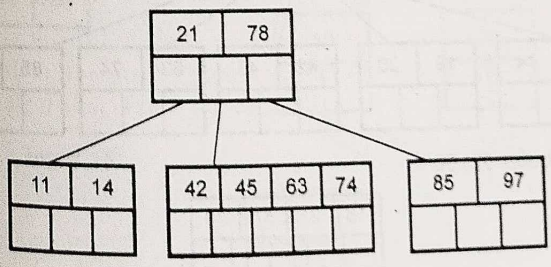
Step 3 : Now insert 85, 74. Then the tree will be -



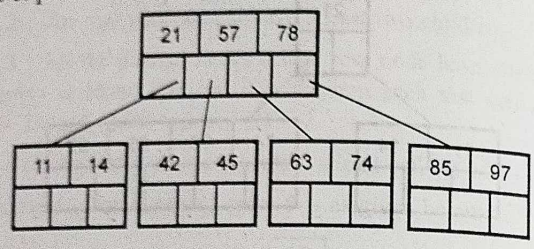
Step 4 : Insert 63. The sequence will then be 63 74 78 85 97. This sequence will then split, 78 will go up. The partial tree will be -



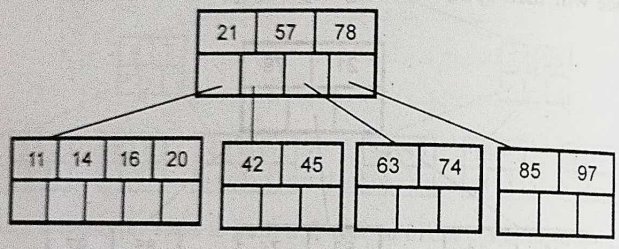
Step 5 : Insert 42, 45.



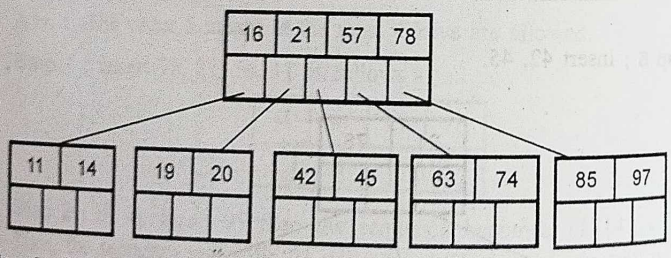
Step 6 : If we insert 57, then the sequence will be 42 45 57 63 74. Then this sequence will split and 57 will go up.



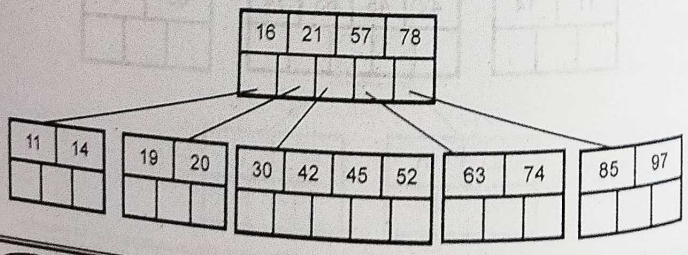
Step 7 : Insert 20, 16. The tree will be -



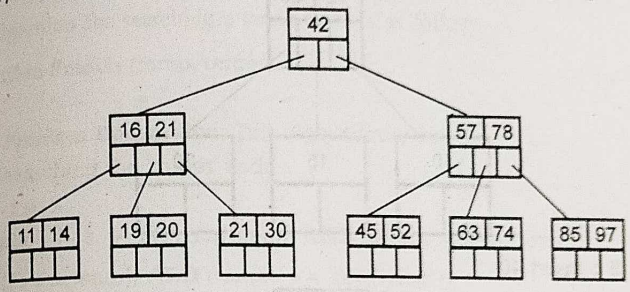
Step 8 : If we insert 19, then the sequence will be 11 14 16 19 20. It will split and 16 will go up.



Step 9 : Insert 52, 30.



Step 10 : Insert 21. The sequence will becomes 21 30 42 45 52. Then 42 will go up. But now the sequence at root level becomes 16 21 42 57 78. Again 42 will be the middle element and it will go up.

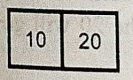


This is the required B-tree.

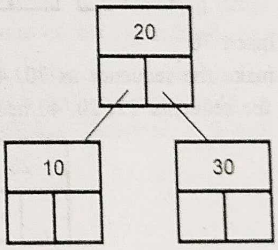
Q.9 Create a B tree of order 3 for the following data : 20, 10, 30, 15, 12, 40, 50. ☞ **[SPPU : Dec.-13, Marks 4]**

Ans. : The order 3 means, 2 keys are allowed in each node.

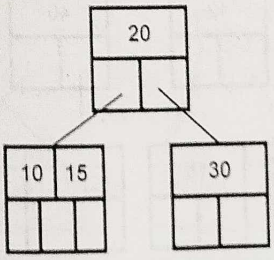
Step 1 : Insert 20, 10



Step 2 : Insert 30
This will move 20 up.

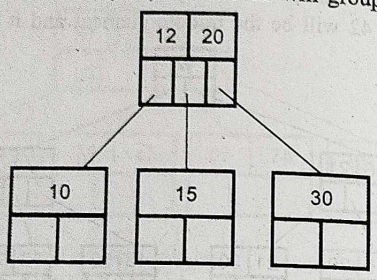


Step 3 : Insert 15

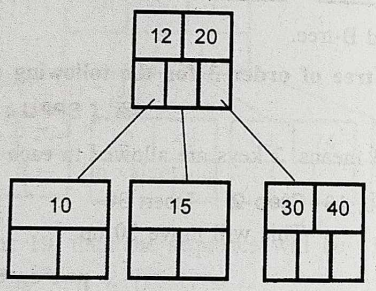


Step 4 : Insert 12.

This will make the list as 10, 12, 15. The 12 will group and join the 20.

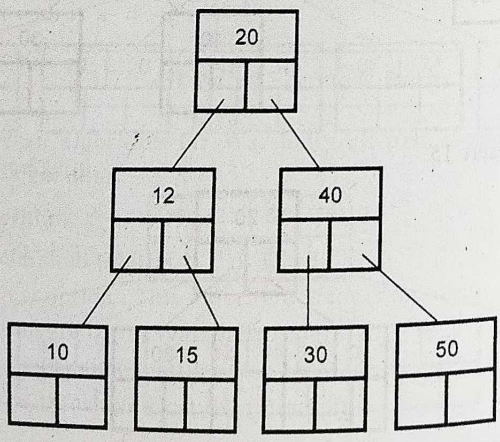


Step 5 : Insert 40



Step 6 : Insert 70

This will make the sequence as 30, 40, 70. Due to which 40 will go up. But again the sequence 12, 20, 40 need to get split. The 20 will go up.



Q.10 Write a pseudo C/C++ code to search the data stored in a B tree. ☞ **[SPPU : Dec.-13, Marks 6 , May-14, Marks 4]**

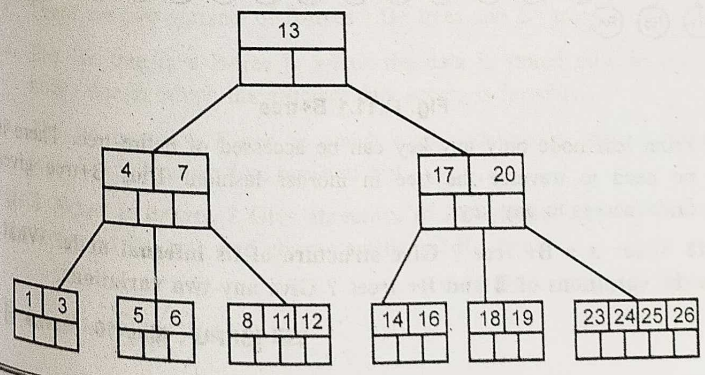
Ans. : Algorithm

The algorithm for searching a target node is as follows.

```
Algorithm Search (temp, target)
{
  // Problem Description: This algorithm is for
  // searching the 'target' node
     i ← 1
  while i ← i + 1 (( i ≤ n[temp]) AND (target > key_i [temp]))
  if (( i ≤ n[temp]) AND (target == key_i[temp]))
     return (temp, i)
  if (leaf [temp] == TRUE) then
     return NULL
  else READ ( C_i[temp] )
     return search ((C_i [temp], target)
}
```

Desired node is present.

- The **search** operation on B-tree is similar to a search operation on a binary tree. The desired child is chosen by performing linear search of values in the node. The running time of this algorithm is O(log n).
- The search operation on B-tree is similar to a search on binary search tree. Instead of choosing between a left and right child as in binary tree, B-tree makes an m-way choice. Consider a B-tree as given below -



If we want to search node 11 then

i) 11 < 13 : Hence search left node.

ii) 11 > 7 : Hence rightmost node.

iii) 11 > 8 , move in second block.

iv) Node 11 is found.

5.5 : B+Tree

Q.11 What is B+ tree ?

Ans. : • In B+ trees from leaf nodes reference to any other node can be possible. The leaves in B+tree form a linked list which is useful in scanning the nodes sequentially. The insertion and deletion operations are similar to B-trees.

• Consider following B+tree.

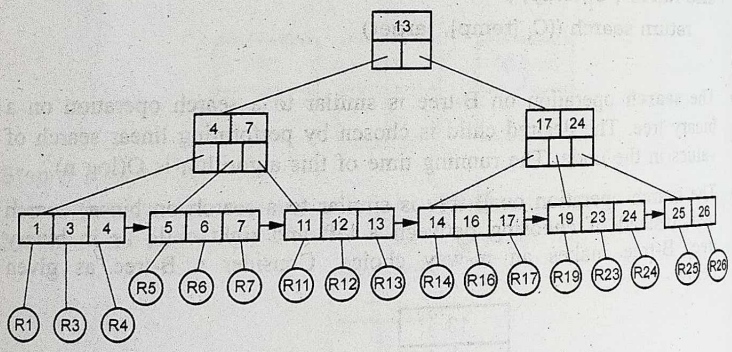


Fig. Q.11.1 B+tree

• From leaf node only any key can be accessed of entire tree. There is no need to traverse the tree in inorder fashion. Thus B+tree gives faster access to any key.

Q.12 What is a B+ tree ? Give structure of its internal node. What are the variations of B and B+ trees ? Give any two variations.

[SPPU : May-10, Marks 6]

Ans. : B+ Tree : Refer Q.11

Structure of internal node :

Let, **n** is the order of B+ tree,

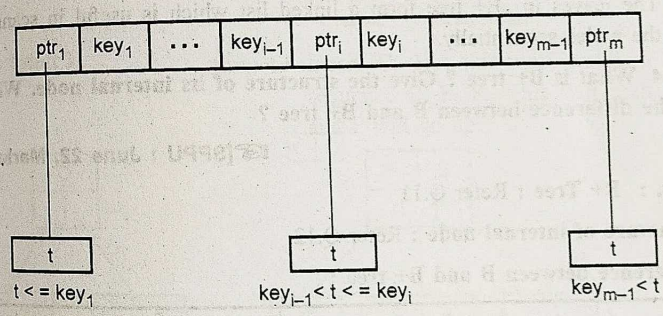
1) Each internal node is of the form **<ptr$_1$, key$_1$, ptr$_2$,key$_2$,..., ptr$_{m-1}$, key$_{m-1}$,ptr$_m$ >** where **m<=n**, where **ptr$_i$** is a tree pointer and **key$_i$** is the key value. The pointer **ptr** points to another node of the tree.

2) Every internal node has $key_1 < key_2 < \ldots key_{m-1}$.

3) Each search field value 't' in the sub-tree pointed by ptr_i, the following condition holds

a) $key_{i-1} < t <= key_i$ for $1 < i < m$ and

b) $key_{i-1} < t$ for i=m



Variations of B-trees :

• There are two variants of B-trees - B+ trees and B* trees.

• The B+ tree is a B-tree in which the data is stored only in the leaf nodes due to which the efficient data access is possible.

• The B* tree is a B-tree in which each node except root node is at least 2/3 full rather than just half full.

Q.13 What is B+tree ? Give structure of its internal node. What are the order of B+ tree and characteristics of B+ tree.

[SPPU : May-14, Marks 7]

Ans. : B+ tree : Refer Q.11

Structure of internal node : Refer Q.12

Order of B+ tree : The order of a B+ tree is the maximum number of children that an internal node may have.

Characteristics of B+ tree :

- If m is the order of B+ tree then the properties of B+ tree are as follows -

1. Every node in a B+ Tree contains at most "m" children.
2. Every node in a B+ Tree, except the root node and the leaf node contain at least "m/2" children.
3. Nodes can have a maximum of (m – 1) keys.
4. Nodes can have a minimum of [m/2] – 1 keys.
5. The leaves in B+ tree form a linked list which is useful in scanning the nodes sequentially.

Q.14 What is B+ tree ? Give the structure of its internal node. What is the difference between B and B+ tree ?

☞[SPPU : June 22, Marks 6]

Ans. : B+ Tree : Refer Q.11

Structure of internal node : Refer Q.12

Difference between B and B+ tree :

Sr.No.	B Tree	B+ Tree
1.	In the B tree, all the keys and records are stored in both internal as well as leaf nodes.	In the B+ tree, keys are stored in internal node whereas records are stored in leaf nodes.
2.	No duplication is allowed.	Duplication may occur.
3.	Linking of leaf nodes is not done with each other.	Linking of leaf nodes is done with each other.
4.	Sequential access is not possible.	Sequential access is possible.
5.	Searching is less efficient.	Searching is very efficient.

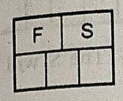
Q.15 Construct a B+tree for F, S, Q, K, C, L, H, T, V, W, M, R.

☞ [SPPU : Dec.-10, Marks 8, June-22, Marks 6]

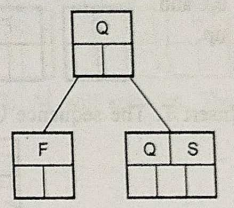
Ans. : • The method for constructing B+tree is similar to the building of B tree but the only difference here is that, the parent nodes also appear in the leaf nodes. We will build B+tree for order 3.

- The order 3 means at the most 2 keys are allowed.

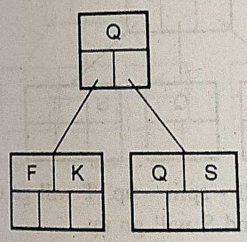
Step 1 : Insert F and S



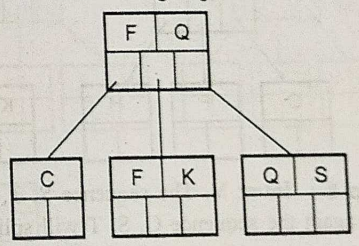
Step 2 : If we insert Q then the sequence will be F, Q, S. The Q will go up.



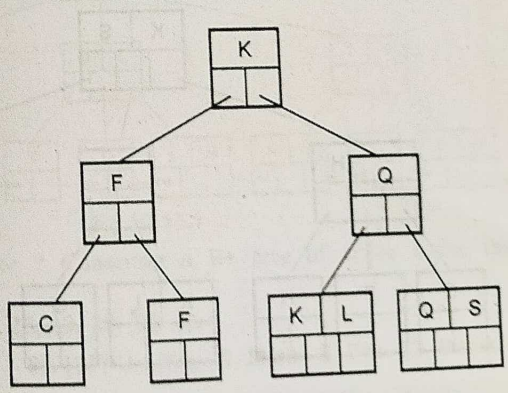
Step 3 : Insert K.



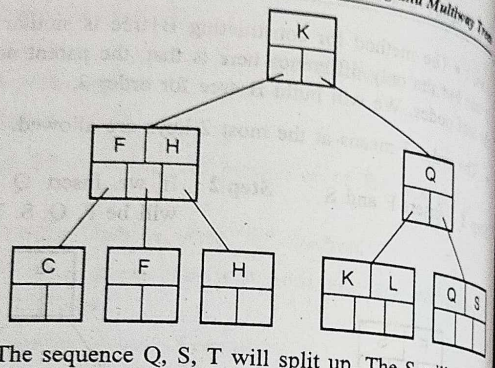
Step 4 : Insert C. But this will create a sequence C, F, K. This will split and F will go up.



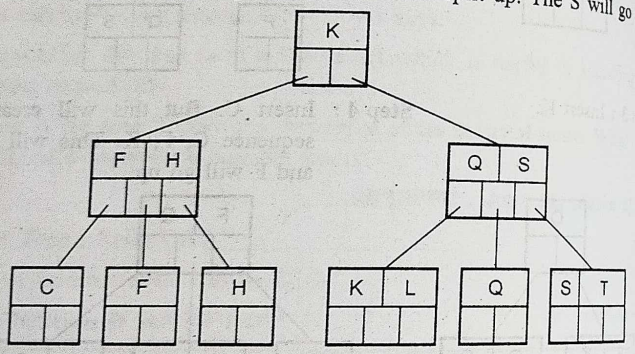
Step 5 : Insert L. This will make the sequence F, K, L. Again this sequence will split up and K will go up. Then the sequence F, K, Q will split up and K will go up.



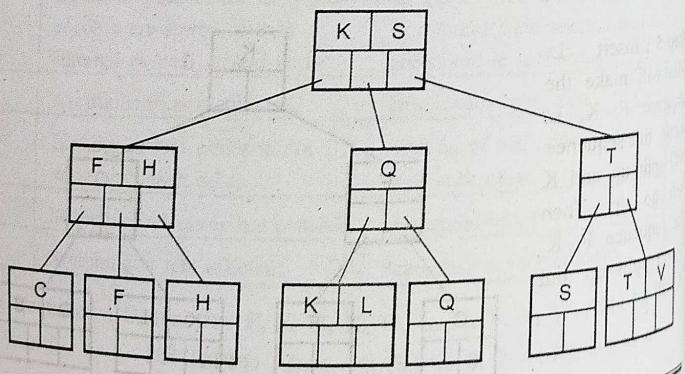
Step 6 : Insert H. This will make the sequence F, H, K. The sequence will split up and H will go up.



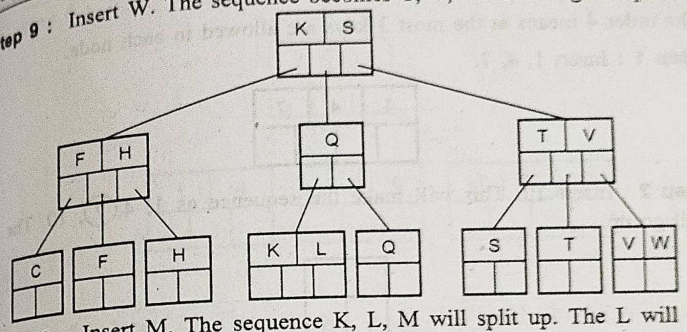
Step 7 : Insert T. The sequence Q, S, T will split up. The S will go up.



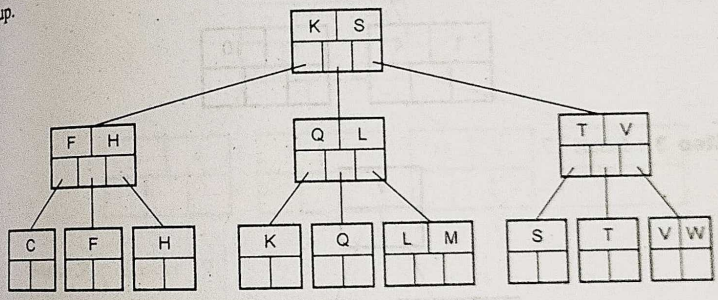
Step 8 : Insert V. The sequence S, T, V will - split - up. T will go up. But again the sequence Q, S, T will split up and S will go up.



Step 9 : Insert W. The sequence becomes T, V, W. The V goes up.



Step 10 : Insert M. The sequence K, L, M will split up. The L will go up.



Step 11 : Insert R.

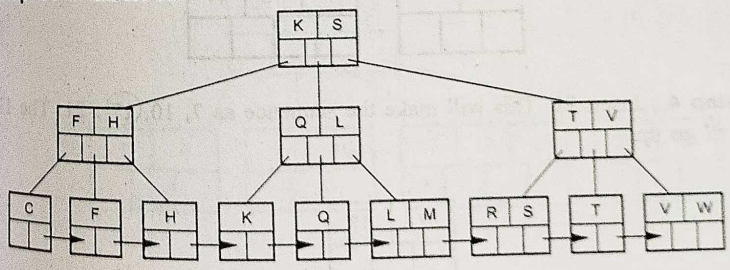


Fig. Q.15.1

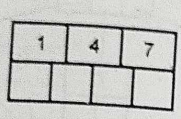
Q.16 What is B+ tree ? Construct a B+ tree of order 4 for the following data.
1, 4, 7, 10, 17, 21, 31, 25, 19, 20, 28, 42.

☞[SPPU : May-19, Marks 8, Dec.-22, Marks 6]

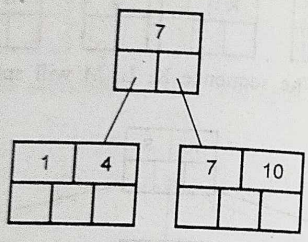
Ans. : B + Tree - Refer Q.11.

The order 4 means at the most 3 keys are allowed in each node.

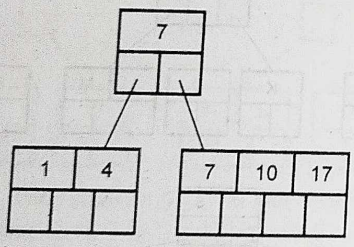
Step 1 : Insert 1, 4, 7.



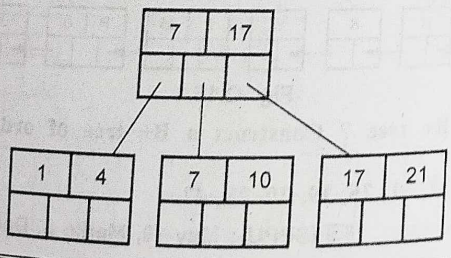
Step 2 : Insert 10. This will make the sequence as 1, 4, 7, 10. The 7 will go up.



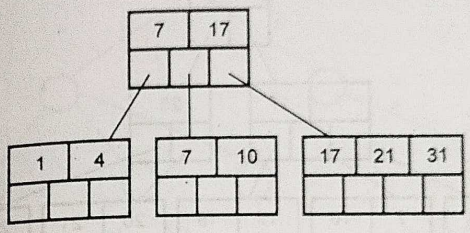
Step 3 : Insert 17.



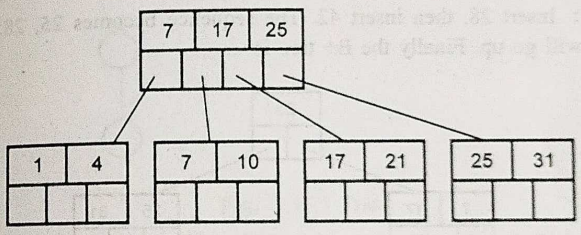
Step 4 : Insert 21. This will make the sequence as 7, 10, 17, 21. The 17 will go up.



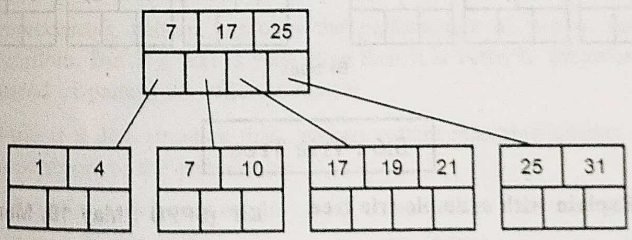
Step 5 : Insert 31.



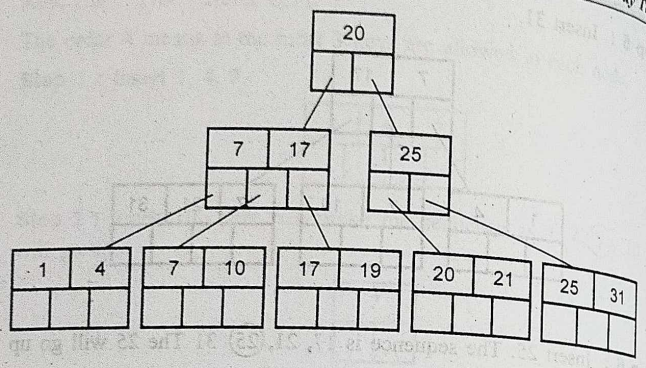
Step 6 : Insert 25. The sequence is 17, 21, 25, 31 The 25 will go up



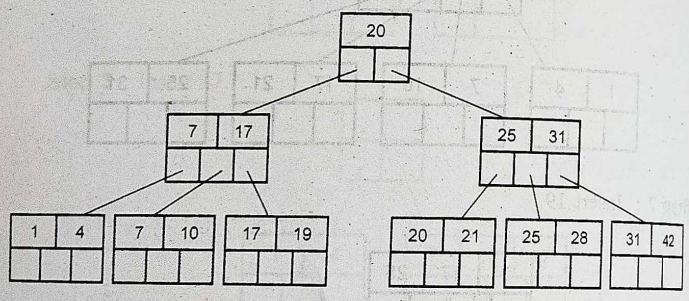
Step 7 : Insert 19.



Step 8 : Insert 20. The sequence becomes 17, 19, 20, 21. The 20 will go up. The sequence becomes 7, 17, 20, 25. Again 20 will go up.



Step 9 : Insert 28, then insert 42. The sequence becomes 25, 28, 31 42. The 31 will go up. Finally the B+ tree is



B+ tree

5.6 : Trie Tree

Q.17 Explain with example trie tree. ☞ **[SPPU : May 19, Marks 3]**

Ans. : • **Definition :** A trie (derived from retrieval) is a multiway tree data structure used for storing strings over an alphabet. It is used to store a large amount of strings. The pattern matching can be done efficiently using tries. Following Fig. Q.17.1 represents the trie.

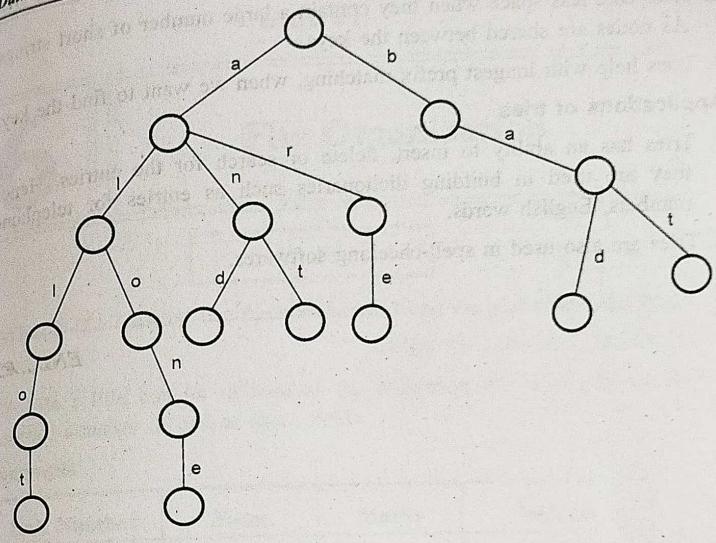


Fig. Q.17.1 Trie

- The trie shows words like allot, alone, ant, and, are, bat, bad. The idea is that all strings sharing common prefix should come from a common node. The tries are used in spell checking programs.
- Preprocessing pattern improves the performance of pattern matching algorithm. But if a text is very large then it is better to preprocess text instead of pattern for efficient search.
- A trie is a data structure that supports pattern matching queries in time proportional to the pattern size.

Q.18 Explain with example trie tree. Give advantage and applications of trie tree ☞ **[SPPU : June 22, Marks 6]**

Ans. : Trie Tree : Refer Q.17

Advantages of tries

1. In tries the keys are searched using common prefixes. Hence it is faster. The lookup of keys depends upon the height in case of binary search tree.

2. Tries take less space when they contain a large number of short strings. As nodes are shared between the keys.
3. Tries help with longest prefix matching, when we want to find the key.

Applications of tries

1. Tries has an ability to insert, delete or search for the entries. Hence they are used in building dictionaries such as entries for telephone numbers, English words.
2. Tries are also used in spell-checking softwares.

END...

DIPS

Unit VI

6 File Organization

6.1 : Files

Q.1 What is file ? Enlist the operations that can be performed on file.

☞[SPPU : June-22, Marks 3]

Ans. : File : File can be defined as the collection of records in which each record consists of one or more fields.

For example

Roll Number	Name	Marks	Address
10	AAA	90	MG Road
20	BBB	88	Shivaji Nagar
30	CCC	76	S.K.Marg
.	.	.	.
.	.	.	.
.	.	.	.

Fig. Q.1.1 Sample File containing student record

- **Various operations that can be performed on the file are -**
  1. Insertion of record into the file
  2. Deletion of record
  3. Updation of record
  4. Retrieval of record

Q.2 Explain the stream classes used in file handling in C++.

☞[SPPU : Dec.-22, Marks 2]

Ans. : • C++ provides following classes to perform input and output of characters to and from the files.

DIPS

ofstream	This stream class is used to write on files.
ifstream	This stream class is used to read from files.
fstream	This stream class is used for both read and write from/to files.

To perform file I/O, we need to include **<fstream.h>** in the program. It defines several classes, including **ifstream, ofstream** and **fstream**. These classes are derived from **ios**, so **ifstream, ofstream** and **fstream** have access to all operations defined by **ios**. While using file I/O we need to do following tasks -

1. To create an input stream, declare an object of type **ifstream**.
2. To create an output stream, declare an object of type **ofstream**.
3. To create an input/output stream, declare an object of type **fstream**.

Example code

```
#include <iostream.h>
#include <fstream.h>
void main()
{
 ofstream fobj; //creating object
 fobj.open ("output.txt"); //opening the file using object
 fobj << "Writing...Tested OK!!!\n";//writing to the file
 fobj.close();//closing the file
}
```

- The above program basically creates a file called **output.txt** and writes the message "Writing...Tested OK!!!" in that file. At the end of the program the file is closed.

Q.3 What is the difference between opening a file using constructor function and open() function ? ☞[SPPU : Dec.-22, Marks 2]

Ans. : 1. Opening a file using constructor function

- Using the constructor the object is created. In the same way using the file stream constructor the **file stream object** is created. Suppose we want to open the file for reading then we make use of **ifstream** class and create an object. Using this object the desired file is opened and is available for reading data. For example

```
ifstream myfile("input.txt",ios::in);
```

POCO M3 | ❤️❤️ DIPS ❤️❤️

- The above statement creates **myfile** object of input stream class **ifstream**.
- Opening a file using constructor function is better to be used when a **single file** is to be opened using single stream.

2. Opening a file using open function

- The open function of **single stream** is used to open the desired file. For example

```
ifstream myfile;  //This line creates an object of ifstream
myfile.open("input1.txt",ios::in);
myfile.close();
myfile.open("input2.txt,ios::in);
myfile.close();
myfile.open("input3.txt,ios::in);
myfile.close();
```

When we want to open **multiple files** one after the other then use **open** function.

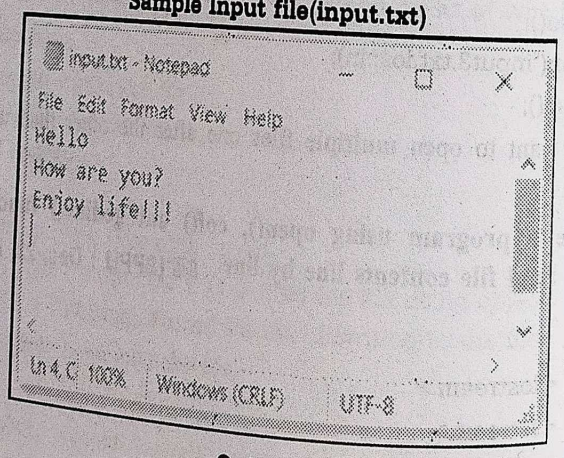
Q.4 Write a program using open(), eof() anf getline() function to open and read file contents line by line. ☞[SPPU : Dec.-22, Marks 2]

Ans. :

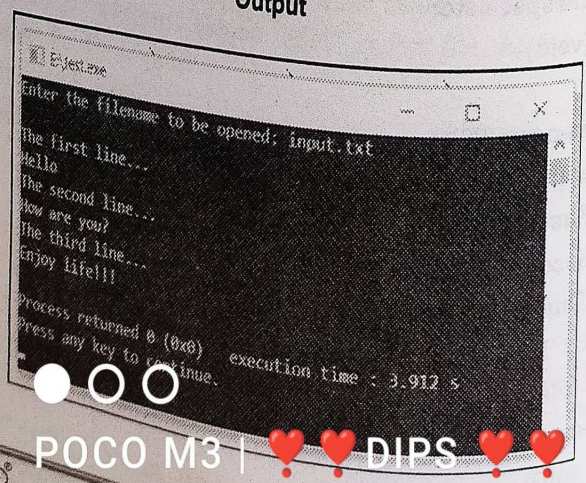
```
#include <iostream>
#include <fstream>
using namespace std;
int main(void)
{
  char filename[70];
  ifstream ifs;
  char FirstLine[70];
  char SecondLine[70];
  char ThirdLine[70];
  cout<<"Enter the filename to be opened: ";
  cin>>filename;
  ifs.open(filename);
  if(!ifs.eof())
```

```
{
    cout<<"\nThe first line... \n";
    ifs.getline(FirstLine, 70);
    cout<<FirstLine<<'\n';
    cout<<"The second line... \n";
    ifs.getline(SecondLine, 70);
    cout<<SecondLine<<endl;
    cout<<"The third line... \n";
    ifs.getline(ThirdLine, 70);
    cout<<ThirdLine<<endl;
}
```

Sample Input file(input.txt)



Output



POCO M3 | ❤️❤️ DIPS ❤️❤️

Q.5 Explain the various modes of opening the file in C or C++.

☞ [SPPU : Dec.-13, 17, 19, Marks 7, June-22, Marks 3, Dec.-22, Marks 5]

Ans. : Modes in which file can be opened

ios::in	Open for input operation.
ios::out	Open for output operation.
ios::binary	Open for binary operations.
ios::ate	If this flag is set then initial position is set at the end of the file otherwise initial position is at the beginning of the file.
ios::app	The output operations are appended to the file. This is an appending mode. That means contents are inserted at the end of the file.
ios::trunc	The contents of pre existing file get destroyed and it is replaced by new one.

We can use open() function using the above given syntax as -

```
ofstream obj;
obj.open("sample.bin",ios::out|ios::binary)
```

That means the file sample.bin is opened for output operation in binary mode. Thus we can combine the flags using OR operator (|).

Q.6 Explain seek and tell operations in file handling in C++ with example.

Ans. : • While performing file operations, we must be able to reach at any desired position inside the file. For this purpose there are two commonly used functions -

1) seek

The seek operation is using two functions seekg and seekp.

- **seekg** means get pointer of specific location for reading of record.
- **seekp** means get pointer of specific location for writing of record.

The syntax of **seek** is

seekg (offset, reference - position);

seekp (offset, reference - position);

where, **offset** is any constant specifying the location.

reference - position is for specifying beginning, end or current position. It can be specified as,

ios : : beg	for beginning location
ios : : end	for end of file
ios : : cur	for current location

2) tell

This function tells us the current position.

For example

seqfile. tellg ()	-	gives current position of get pointer (for reading the record).
seqfile. tellp ()	-	gives current position of put pointer (for writing the record).

Q.7 What is file organization ? What are its types ?

Ans. :

Definition : File organization deals with representation or arrangement of data on external storage.

- The method by which records can be retrieved or updated from the file is called **access method.**
- The records can be organized in a way file in various ways. These ways are represented by following figure.

POCO M3 | ❤️❤️ DIPS ❤️❤️

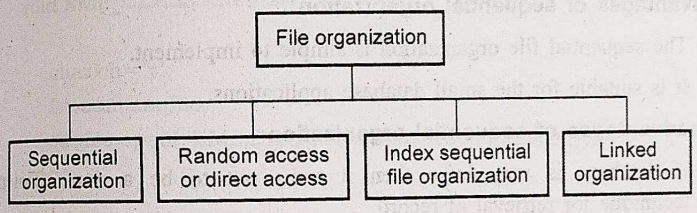


Fig. Q.7.1 Types of file organization

6.2 : Sequential File Organization

Q.8 Describe sequential access file organization method in detail. Also state the advantages and disadvantages.

[May 19, Dec 19, June 22, Marks 6]

Ans. : • This type of file organization makes use of ISMA scheme.

- The ISMA scheme i.e. Index Sequential Memory Access scheme is a kind of file organization in which record can be arranged using cylinder and surfaces.

For example

Consider the arrangement of students' record in which cylinders and surfaces are used.

Record number	Name	Roll_number	Marks	Cylinder	Surface
1	AAA	10	70	1	1
2	BBB	20	90	1	2
3	CCC	30	80	2	1
4	DDD	40	55	2	2
5	EEE	50	78	3	1

- Now if we want to find record of Roll_number 40 then we can search it based on the cylinder. That is we will first search for the cylinder number 2, then we will make the search based on the surface. When we reach at the surface 2 of cylinder 2 then the desired record can be obtained within the track.

Advantages of sequential organization

1. The sequential file organization is simple to implement.
2. It is suitable for the small database applications.

Disadvantages of sequential organization

1. If database is very huge then it turns out to be a non efficient technique for retrieval of record.
2. If the number of tracks are more on the surfaces then multiple disk access may be required to obtain the desired record.

Q.9 Explain any three operations on sequential file organization with example.

☞ **[SPPU : May 17, June 22, Dec. 22, Marks 6]**

Ans. : C++ class

```
class EMP_CLASS
{
typedef struct EMPLOYEE
{
   char name[10];
   int emp_id;
   int salary;
}Rec;
Rec Records;
      public:
      void Create();
      void Display();
      void Update();
      void Delete();
      void Append();
      int Search();
};
```

This record structure is for the sequential file For example

name	emp_id	salary
AAA	10	1000
BBB	20	2000
CCC	30	3000

Various operations on File -

1) Create file operation
2) Display file operation
3) Update file operation
4) Delete operation

POCO M3 | ❤️❤️ DIPS ❤️❤️


```
void EMP_CLASS::Create()
{
   char ch='y';
   fstream seqfile;
   seqfile.open("EMP.DAT",ios::in|ios::out|ios::binary);
   do
   {
      cout<<"\n Enter Name: ";
      cin>>Records.name;
      cout<<"\n Enter Emp_ID: ";
      cin>>Records.emp_id;
      cout<<"\n Enter Salary: ";
      cin>>Records.salary;
      //then write the record containing this data in the file
      seqfile.write((char*)&Records,sizeof(Records));
      cout<<"\nDo you want to add more records?";
      cin>>ch;
   }while(ch=='y');
   seqfile.close();
}
void EMP_CLASS::Display()
{
   fstream seqfile;
   int n,m,i;
   seqfile.open("EMP.DAT",ios::in|ios::out|ios::binary);
   //positioning the pointer in the file at the beginning
   seqfile.seekg(0,ios::beg);
   cout<<"\n The Contents of file are ..."<<endl;
   //read the records sequentially
   while(seqfile.read((char *)&Records,sizeof(Records)))
   {
      if(Records.emp_id!=-1)
      {
         cout<<"\nName: "<<Records.name;
         cout<<"\nEmp_ID: "<<Records.emp_id;
```

User must enter the desired data in the members of the structure **Records**

```
        cout<<"\nSalary: "<<Records.salary;
        cout<<"\n";
    }
}
int last_rec=seqfile.tellg();//last record position
//formula for computing total number of objects in the file
n=last_rec/(sizeof(Rec));
cout<<"\n\n Total number of objects are "<<n<<"(considering
logical deletion)";
seqfile.close();
}
void EMP_CLASS::Update()
{
  int pos;
  cout<<"\n For updation,";
  fstream seqfile;
  seqfile.open("EMP.DAT",ios::in|ios::out|ios::binary);
  seqfile.seekg(0,ios::beg);
//obtaining the position of desired record in the file
  pos=Search();
  if(pos==-1)
  {
   cout<<"\n The record is not present in the file";
   return;
  }
//calculate the actual offset of the desired record in the file
  int offset=pos*sizeof(Rec);
  seqfile.seekp(offset);//seeking the desired record for modification
  cout<<"\n Enter the values for updation...";
  cout<<"\n Name: ";cin>>Records.name;
  cout<<"\n Emp_Id: ";cin>>Records.emp_id;
  cout<<"\n Salary: ";cin>>Records.salary;
  seqfile.write((char*)&Records,sizeof(Records))<<flush;
  seqfile.seekg(0);
  seqfile.close();
```

New values for the updation of record


```
  cout<<"\n The record is updated!!!";
}
void EMP_CLASS::Delete()
{
  int id,pos;
  cout<<"\n For deletion,";
  fstream seqfile;
  seqfile.open("EMP.DAT",ios::in|ios::out|ios::binary);
  seqfile.seekg(0,ios::beg);//seeking for reading purpose
  pos=Search();//finding pos. for the record to be deleted
  if(pos==-1)
  {
   cout<<"\n The record is not present in the file";
   return;
  }
//calculate offset to locate the desired record in the file
  int offset=pos*sizeof(Rec);
  seqfile.seekp(offset);//seeking the desired record for deletion
  strcpy(Records.name,"");
  Records.emp_id=-1;
  Records.salary=-1;
  seqfile.write((char*)&Records,sizeof(Records))<<flush;
  seqfile.seekg(0);
  seqfile.close();
  cout<<"\n The record is Deleted!!!";
}
```

It's a logical deletion

6.3 : Direct Access File

Q.10 Explain random access file organization.

Ans. : • Random organization is a kind of file organization in which records are stored at random locations on the disks.

- There are three techniques used in random organization and those are given in following Fig. Q.10.1.

POCO M3 | ❤️❤️ DIPS ❤️❤️

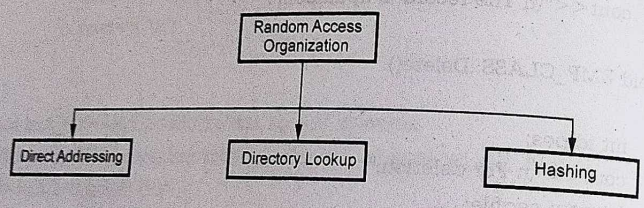


Fig. Q.10.1 Random access organization

1. Direct addressing

- In direct addressing two types of records are handled: fixed length record and variable length record.
- For storing the fixed length records the disk space is divided into the nodes. These nodes are large enough to hold individual record.
- Every fixed length record is stored in node number # which is equal to the primary key value. For example: If a primary key value is 185 then the record must be present in the node number 185.

For example

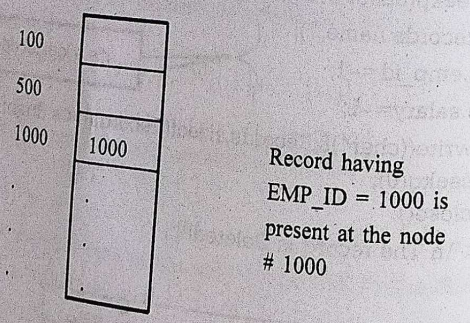


Fig. Q.10.2 Storing of fixed length record

- If we consider that the records are stored on the external storage devices then **deletion** and **searching** of the record requires one disk access. If we want to **update** a record then it requires two disk access, one for reading the record and one for writing the updated data back to the disk.

POCO M3 | ❤❤ DIPS ❤❤

- For storing the **variable length records** on the disk, the address (pointer) of each individual record is stored in the file at specific index. We can locate the variable length record using the index of the pointer. This pointer will point to desired record which is present on the disk.

  Variable length records make the storage management more **complex**.

2. Directory lookup

- In this scheme the index for the pointers to the records is maintained.
- For retrieving the desired record first of all the index for the record address is searched and then using this record address the actual record is accessed.
- The **drawback** of this method is that it requires more disk access than direct address method.
- **Advantage** of this method is that effective disk space utilization in it as compared to direct addressing method.

3. Hashing

- Hashing is a technique in which hash key is obtained using some suitable **hash function** and record is placed in the hash table with the help of this **hash key**.
- Thus in this random organization, the record can be quickly searched with the help of hash function being used.
- For creation of hash table the available file space is divided into **buckets** and **slots**.
- Some file space is left aside for handling the overflow situation.
- The total number of slots per bucket is equal to the total number of records each bucket can hold.

6.4 : Indexed Sequential File Organization

Q.11 Describe indexed sequential access file organization method in detail. Also state the advantages and disadvantages.

☞[SPPU : May 19, Marks 6, June 22, Dec. 22, Marks 5]

Ans. : • The main drawback of sequential file is that searching operation is not efficient. Because in sequential organization primary key of every record is compared with searching key. To optimize this operation concept of index sequential file is introduced.

- In index sequential file organization, a separate file for storing indexes of every record is maintained along with the master file.

For example

- There are two files "IND.DAT" and "EMP.DAT".
- Using the positions field of "IND.DAT" file we can quickly retrieved the desired record.

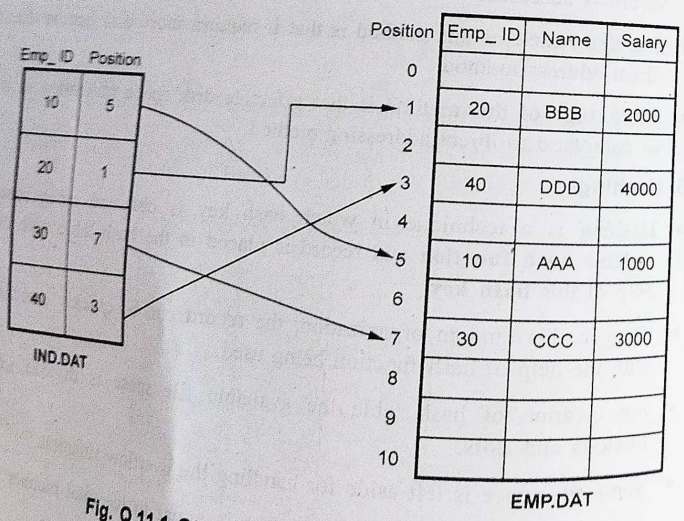


Fig. Q.11.1 Structure of index sequential file

- The index sequential organization, accelerates the retrieval of any desired record. In this case, we need not have to scan the entire memory block of record. Instead of that using primary key (such as EMP_ID) and position we can access the record from master file.

Advantages

1. Desired record can be accessed efficiently by using index which is maintained in separate file.
2. Variable length records can also be handled using index sequential file.

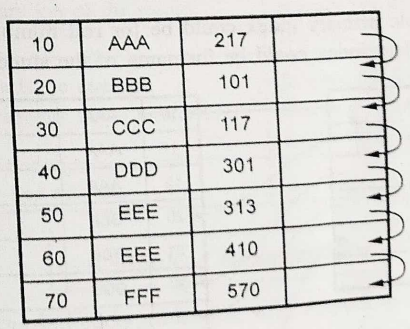
Disadvantages

1. At least two files need to be maintained : One master file and another index file. Hence extra amount of memory is required in order to maintain index file.
2. While performing insertion and deletion index manipulation is required.

Q.12 Explain advantages of indexing over sequential file. Enlist types of indices. Explain any two. ☞ **[SPPU : Dec 19, Marks 7]**

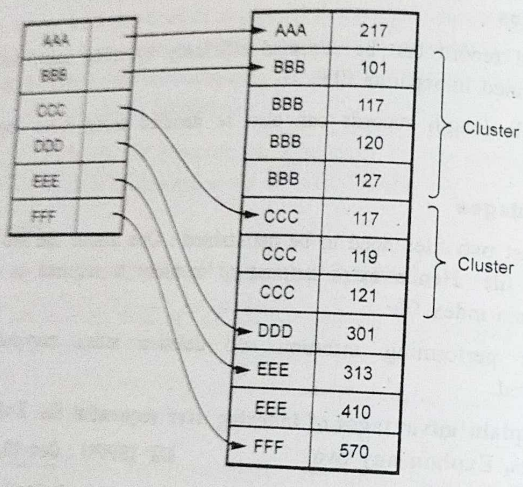
Ans. : Advantage of indexing over sequential File : Refer Q.11.

Types of Indices : i) Primary indexes : A file can be made ordered using a key. This key index is called primary key index. For example -



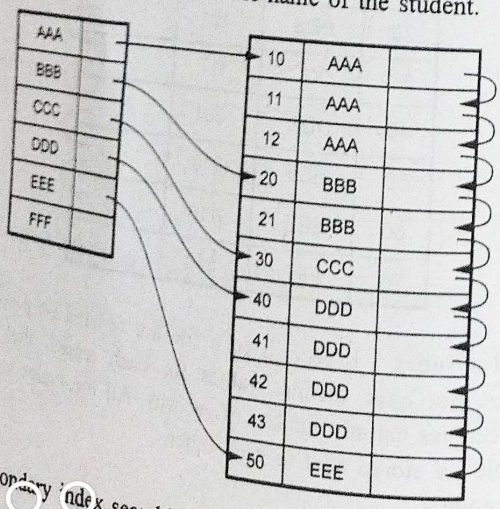
ii) Clustering index : If records of a file are ordered on a non-key field that does not have distinct values for each record, then same disk block occupies certain number of records. All the tuples with identical key value are stored next to each other.

For example -



iii) **Secondary indexes :** Secondary index is a key which is other than a primary key is used for indexing. When certain fields other than the defined key fields are used frequently then in order to improve the performance secondary index is used.

For example primary index could be for roll number of the student and secondary index could be for name of the student.



- Using secondary index searching data becomes efficient.

POCO M3 | ❤️❤️ DIPS ❤️❤️

Q.13 Compare index sequential and direct access files.

☞ **[SPPU : Dec.-17, Marks 6, June-22, Marks 3]**

Ans. :

Sr. No.	Index sequential file	Direct access file
1.	Desired record can be obtained using the index which is maintained in a separate file.	Desired record can be obtained using the hash key. This hash key is returned by some hash function.
2.	Insertion and deletion operations can be performed by simple index manipulation.	On insertion or deletion of records, collision may occur. Hence collision handling techniques are required.
3.	Variable length records are allowed.	Record length must be fixed.
4.	Index key is obtained using primary key of the record.	Hash key is obtained by passing primary key of record to hash function.
5.	Records are arranged sequentially in the master file.	Records can be arranged randomly. This arrangement is influenced by hash key.
6.	It is less efficient.	It is more efficient

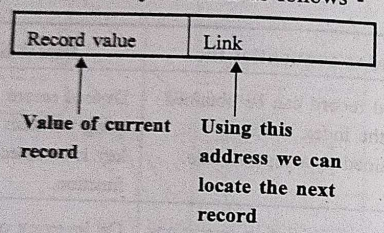
6.5 : Linked Organization

Q.14 Explain the linked organization with respect to file handling.

☞ **[SPPU : May 17, Marks 6]**

Ans. : • In linked organization the logical sequence of the records is different than the physical sequence. In any sequential organization if we are accessing n^{th} node at Loc_i then $(n+1)^{th}$ record may be located at $(Loc_i + c)$ where c is the constant which represents the length of the record or it may be some inter-record spacing.

- In linked organization we can access next logical record by following the **link-value** pair. The link-value pair denotes each individual record.
- The typical structure of every record is as follows -



Thus records in the linked organization can be stored as follows -

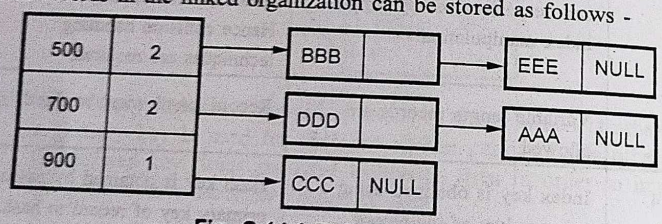
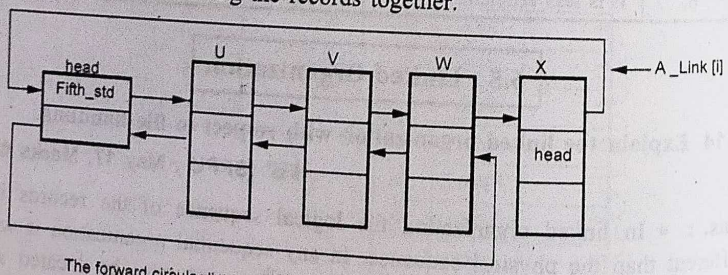


Fig. Q.14.1 Roll_No index

Q.15 Write a short note on - Coral Ring.

Ans. : • Coral ring is a file organization in which doubly linked multi-list structure is maintained. Each list is a circular list. Thus coral ring structure is a kind of structure in which **circular doubly** linked list is maintained for connecting the records together.



The forward circular list contains nodes head U, V, W, X.

The reverse circular list contains nodes head W, V, U.

Fig. Q.15.1 Coral ring structure

- Associated with each record there are two link fields i.e. forward link and back link. Thus records get associated with each other by circular linked list.

Q.16 Explain the linked organization with respect to inverted files.

[SPPU : Dec 19, Marks 7]

Ans. : • Inverted files are similar to multi-lists.

- The difference between multi-list and inverted files is that in multi-lists records with the same key value are linked together along with the **link information** being kept in **individual record**. But in case of inverted files this **link information** is kept in the **index itself**.

For example

437	BBB
488	EEE
689	DDD
695	AAA
778	CCC

Fig. Q.16.1 (a) Roll_No index

Fifth	BBB, CCC
Tenth	EEE, DDD, AAA

Fig. Q.16.1 (b) Class index

Female	BBB, DDD, CCC
Male	EEE, AAA

Fig. Q.16.1 (c) Sex index

First class	EEE
Second class	AAA, DDD, CCC
Pass class	BBB

Fig. Q.16.1 (d) Marks index

- Consider Fig. Q.16.1 (a) of Roll-no index which shows records BBB, EEE, DDD, AAA and CCC. In Fig. Q.16.1 (b) of class index two class are there fifth and tenth and we can observe that in the link information is stored in the index itself. Hence for fifth class records are BBB and CCC. And for tenth class records are EEE, DDD and AAA.

POCO M3 | DIPS

- Similarly from sex index Fig. Q.16.1 (c), it is clear that BBB, DDD and CCC are females and EEE and AAA are males.
- The above index structure is a dense index structure **Dense Indexing**. The dense index is a kind of indexing in which record appers for every search key value in the file.
- Thus in inverted files the index entries is of variable length. Hence inverted files structure is **more complex** than multi-list file structure.
- Following are the two steps that are adopted while searching a record from inverted files -

  i) Index of required record is searched first of all.

  ii) Then actual record is retrieved.
- In inverted files the index structure is important. The records can be arranged sequentially, randomly or linked depending on primary key.
- The number of disk accesses required = Number of records being retrieved + Processing for indexes.

Advantage

1) Inverted files are space saving as compared to other file structures when record retrieval does not require retrieval does not require retrieval of key fields.

Disadvantages

1) Insertion and deletion of records is complex because it requires the ability to insert and delete within indexes.
2) Index maintainance is complicated as compared to multi-list.

Q.17 What is linked organization ? Explain inverted files and cellular partitions with respect to linked organization.

☞ [SPPU : May 19, Marks 6]

Ans. : Linked organization : Refer Q.14.

Inverted files : Refer Q.16.

Cellular partition : • For reducing the searching time during file operations, the storage media
(e.g. secondary memory, magnetic disk, magnetic tape etc.) may be divide d into cells.

- The cells can be of two types -

  i) Entire disk pack can be a cell

  ii) A cylinder can be a cell.
- A list of records can occupy either entire disk pack or it may lie on particular cylinder.
- If all the records lie on the same cylinder then without moving the read/write head the records can be accessed.
- If the cell is nothing but entire disk pack then the disk is partitioned into different partitions. Such partitions are called **cellular partitions**. Then these different cells can be searched in parallel.

Advantages of cellular partitions

1) Various read operations can be performed parallely in order to reduce the search time.
2) Faster execution of any query.

Disadvantage

1) If multiple records lie in the same cell then reading a single cell becomes a time consuming process.

6.6 : External Sort

Q.18 Write short note on external sort.

☞ [SPPU : May 17, June 22, Marks 6]

Ans. : • In external sorting, the **data stored on secondary memory** is part by part loaded into main memory, sorting can be done over there.

- The sorted data can be then stored in the intermediate files. Finally these intermediate files can be merged repeatedly to get sorted data.
- Thus **huge amount** of data can be sorted using this technique.

For example :

- Consider that there are 10,000 records that has to be sorted. Clearly we need to apply external sorting method. Suppose main memory has a capacity to store 500 records in blocks, with each block size of 100 records.

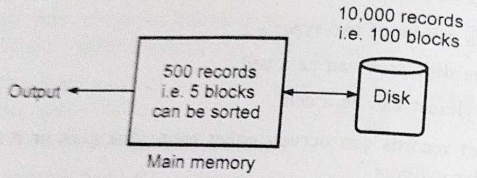


Fig. Q.18.1

- The sorted 5 blocks (i.e. 500 records) are stored in intermediate file. This process will be repeated 20 times to get all the records sorted in chunks.
- In the second step, we start merging a pair of intermediate files in the main memory to get output file.

For example -

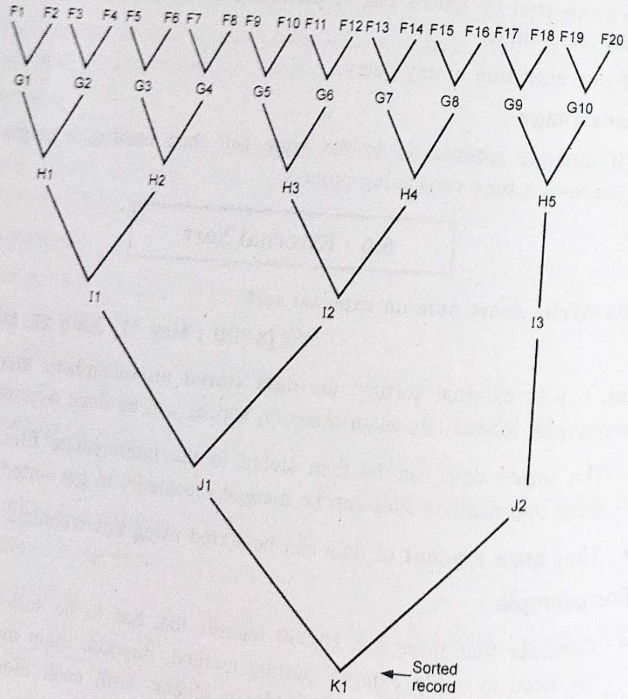


Fig. Q.18.2

Q.19 Sort the following list of elements using two way merge sort with M = 3.
20, 47, 15, 8, 9, 4, 40, 30, 12, 17, 11, 56, 28, 35.

☞ **[SPPU : June-22, Marks 6]**

Ans. : • As M = 3, we will break the records in the group of 3 and sort them. Then we will store them on tape. We will store data on alternate tapes.

Stage I : Sorting Phase

1) 20, 47, 15. We arrange in sorted order → 15, 20, 47

Tb1 :

15	20	47

2) Read next three records sort them and store them on Tape Tb2

8, 9, 4 → 4, 8, 9

Tb2 :

4	8	9

3) Read next three records, sort them and store on tape Tb1.

40, 30, 12 → 12, 30, 40

Tb1 :

15	20	47	12	30	40

4) Read next three records, sort them and store on tape Tb2.

17, 11, 56 → 11, 17, 56

Tb2 :

4	8	9	11	17	56

5) Read next two remaining records, sort them and store on Tape Tb1

28, 35, → 28, 35

Tb1 :

15	20	47	12	30	40	28	35

POCO M3 | DIPS

- At the end of this process we get

Tb1 :	15	20	47	12	30	40	28	35
Tb2 :	4	8	9	11	17	56		

Stage II : Merging of runs

- The input tapes Tb1 and Tb2 will use two more output tapes Ta1 and Ta2, for sorting. Finally the sorted data will be on tape Ta1.

Tb1 :	15	20	47	12	30	40	28	35
Tb2 :	4	8	9	11	17	56		

- We will read the elements from both the tapes Tb1 and Tb2, compare them, and store on Ta1 in sorted order.

Ta1 :	4	8	9	15	20	47

- Now we will read second blocks from Tb1 and Tb2. Sort the elements and store on Ta2.

Ta2 :	11	12	17	30	40	56

- Finally read the third block from Tb1 and store in sorted manner on Ta1. We will not compare this block with Ta2 as there is no third block. Hence we will get

Ta1 :	4	8	9	15	20	47	28	35
Ta2 :	11	12	17	30	40	56		

POCO M3 | DIPS

- Now compare first blocks of Ta1 and Ta2 and store sorted elements on Tb1.

Tb1 :	4	8	9	11	12	15	17	20	30	40	47	56
Tb2 :	28	35										

- Now both Tb1 and Tb2 contains only single block each. Sort the elements from both the blocks and store the result on Ta1.

Ta1 :	4	8	9	11	12	15	17	20	28	30	35	40	47	56

Thus we get **sorted list.**

END...

JUNE - 2022 [5869]-277 | Solved Paper

Course 2019

Time : $2\frac{1}{2}$ Hours] **[Max. Marks : 70**

Instructions to the candidates :

1) ***Answer Q.1 or Q.2, Q.3 or Q.4, Q.5 or Q.6, Q.7 or Q.8.***
2) ***Neat diagrams must be drawn wherever necessary.***
3) ***Figures to the right indicate full marks.***
4) ***Assume suitable data, if necessary.***

Q.1 a) *Draw any directed graph with minimum 6 nodes and represent graph using adjacency matrix, adjacency list and adjacency multi list.* **(Refer Q.10 of Chapter - 3)** **[6]**

b) *Consider the graph represented by the following adjacency matrix :* **[6]**

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

Find minimum spanning tree of this graph using Prim's algorithm. **(Refer similar Q.24 of Chapter - 3)**

c) *Write a short note on topological sorting.* **(Refer Q.38 of Chapter - 3)** **[6]**

OR

Q.2 a) *Write non-recursive pseudo for Depth First Search (DFS).* **(Refer Q.20 of Chapter - 3)** **[6]**

DIPS

b) *Consider the given graph and find the shortest path by using Dijkstra's algorithm. From source to all other nodes.* **(Refer Q.32 of Chapter - 3)** **[6]**

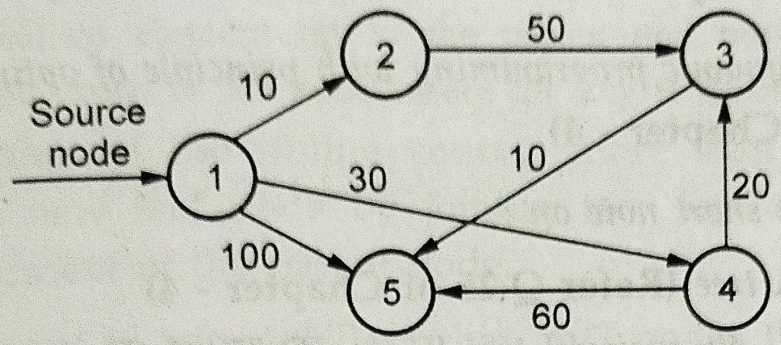


Fig. 1

c) *Show BFS and DFS for the following graph with starting vertex as 1. Explain with proper steps.* **(Refer Q.19 of Chapter - 3)** **[6]**

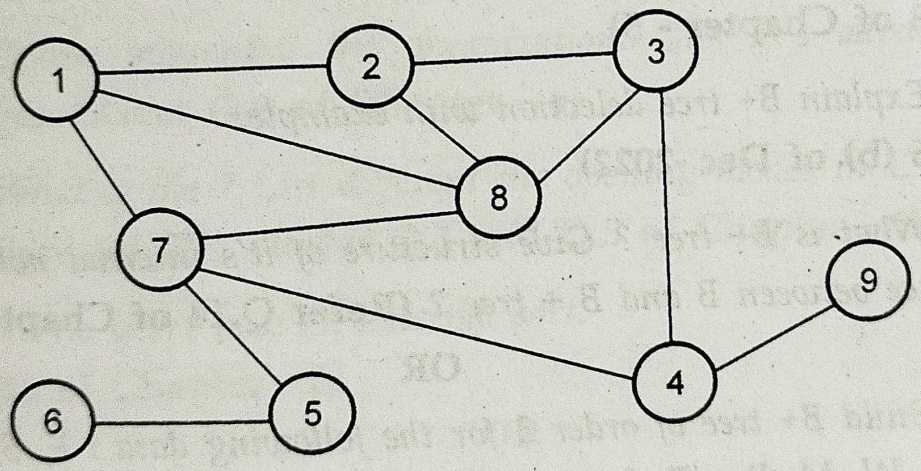


Fig. 2

Q.3 a) *Explain with example :*

i) Red - Black Tree **(Refer Q.23 of Chapter - 4)**

ii) Splay Tree **(Refer Q.30 of Chapter - 4)** **[6]**

b) *Construct AVL tree for following sequence of keys. 1, 2, 3, 4, 8, 7, 6, 5, 11, 10* **(Refer Q.21 of Chapter - 4)** **[6]**

c) *What is OBST in data structure ? and what are advantages of OBST ?* **(Refer Q.7 of Chapter - 4)** **[5]**

OR

Q.4 a) *Explain the following :*

i) Static and dynamic tree tables with suitable example.
(Refer Q.2 of Chapter - 4) [3]

ii) Dynamic programming with principle of optimality.
(Refer Q.5 of Chapter - 4) [3]

b) *Write short note on :* [6]

i) AA tree **(Refer Q.25 of Chapter - 4)**

ii) K - dimensional tree **(Refer Q.27 of Chapter - 4)**

c) *Explain AVL tree rotations with example.*
(Refer Q.14 of Chapter - 4) [5]

Q.5 a) *Construct B tree of order 5 for the following data :* [6]

78, 21, 14, 11, 97, 85, 74, 63, 45, 42, 57
(Refer Q.8 of Chapter - 5)

b) *Explain B+ tree delection with example.*
(Refer Q.6 (b) of Dec.-2022) [5]

c) *What is B+ tree ? Give structure of it's internal note. What is the difference between B and B + tree ?* **(Refer Q.14 of Chapter - 5)** [6]

OR

Q.6 a) *Build B+ tree of order 3 for the following data : F, S, Q, K, C, L, H, T, V, W, M, R.* **(Refer Q.15 of Chapter - 5)** [6]

b) *Write an algorithm of B tree deletion.* [6]

Ans. :

Step 1 : Locate the leaf node.

Step 2 : If there are more than m/2 keys in the leaf node then delete the desired key from the node.

Step 3 : If the leaf node doesn't contain m/2 keys then complete the keys by taking the element from eight or left sibling.

- If the left sibling contains more than m/2 elements then push its largest element up to its parent and move the intervening element down to the node where the key is deleted.
- If the right sibling contains more than m/2 elements then push its smallest element up to the parent and move intervening element down to the node where the key is deleted.

Step 4 : If neither of the sibling contain more than m/2 elements then create a new leaf node by joining two leaf nodes and the intervening element of the parent node.

Step 5 : If parent is left with less than m/2 nodes then, apply the above process on the parent too.

If the node which is to be deleted is an internal node, then replace the node with its in-order successor or predecessor.

c) *Explain with example trie tree. Give advantage and applications of trie tree.* **(Refer Q.18 of Chapter - 5)** [5]

Q.7 a) *Define sequential file organization. Give it's advantages and disadvantages.* **(Refer Q.8 of Chapter - 6)** [6]

b) *What is file ? List different file opening modes in C++. Explain concept of inverted files.* **(Refer Q.1 and Q.5 of Chapter - 6)** [6]

c) *Write short note on external sort.*
(Refer Q.18 of Chapter - 6) [5]

OR

Q.8 a) *A write a C++ program to create a file. Insert records into the file by opening file in append mode. Search for a specific record into file.*
(Refer Q.9 of Chapter - 6) [6]

b) *Sort the following elements using two way merge sort with m = 3.*

20, 47, 15, 8, 9, 4, 40, 30, 12, 17, 11, 56, 28, 35

(Refer Q.19 of Chapter - 6) [6]

c) *Explain indexed sequential file organization. Compare it with direct access file.* **(Refer Q.11 and Q.13 of Chapter - 6)** [5]

M3 | DIPS

DECEMBER - 2022 [5925]-261 | Solved Paper

Course 2019

Time : 2 $\frac{1}{2}$ Hours] **[Max. Marks : 70**

Q.1 a) *Find minimum spanning tree of the following graph using kruskals algorithm.* **(Refer Q.27 of Chapter - 3)** **[6]**

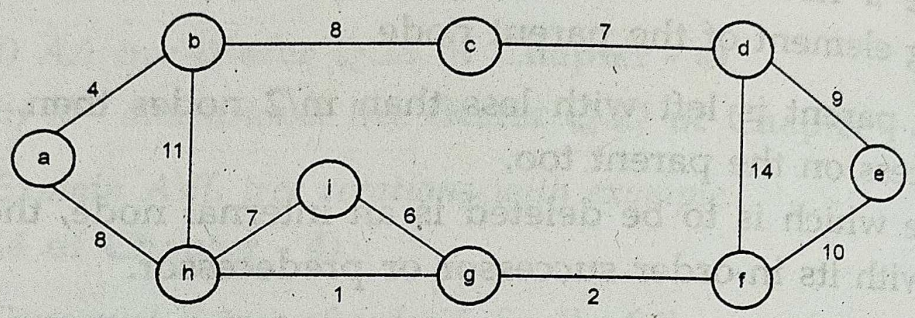


Fig. 1

b) *Write algorithm for Breadth First Traversal of the graph. Also write its complexity.* **(Refer Q.15 of Chapter - 3)** **[6]**

c) *Write Kruskal's algorithm for minimum spanning trees and explain with example.* **(Refer Q.25 of Chapter - 3)** **[6]**

OR

Q.2 a) *Apply Prim's Algorithm to construct minimum spanning tree, for below drawn graph : Starting vertex is 'a'.* **(Refer similar Q.24 of Chapter - 3)** **[6]**

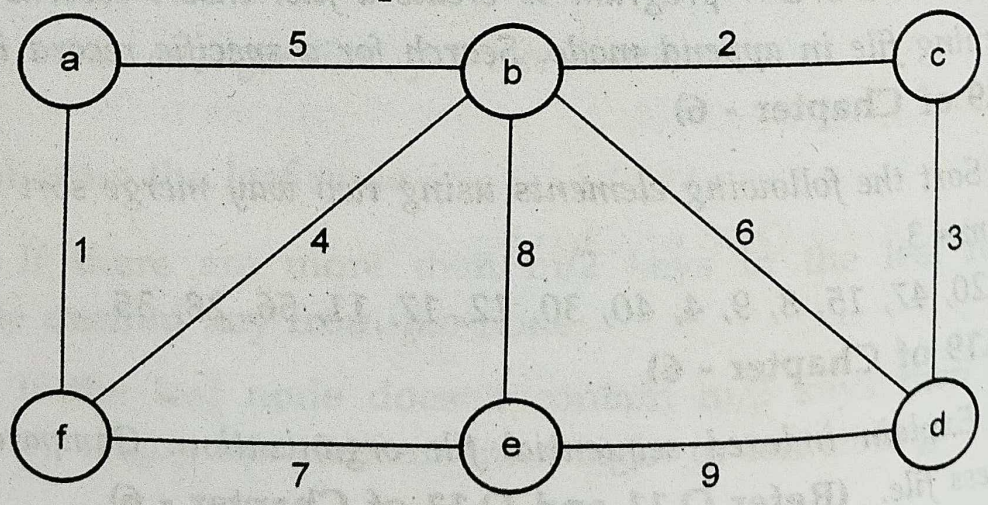


Fig. 2

O M3 | ❤️❤️DIPS ❤️❤️

b) *Develop pseudo code with one example to traverse a graph using BFS.* **(Refer Q.14 of Chapter - 3)** **[6]**

c) *Find the shortest path from a to f, in the following graph using Dijkstra's Algorithm.* **(Refer Q.33 of Chapter - 3)** **[6]**

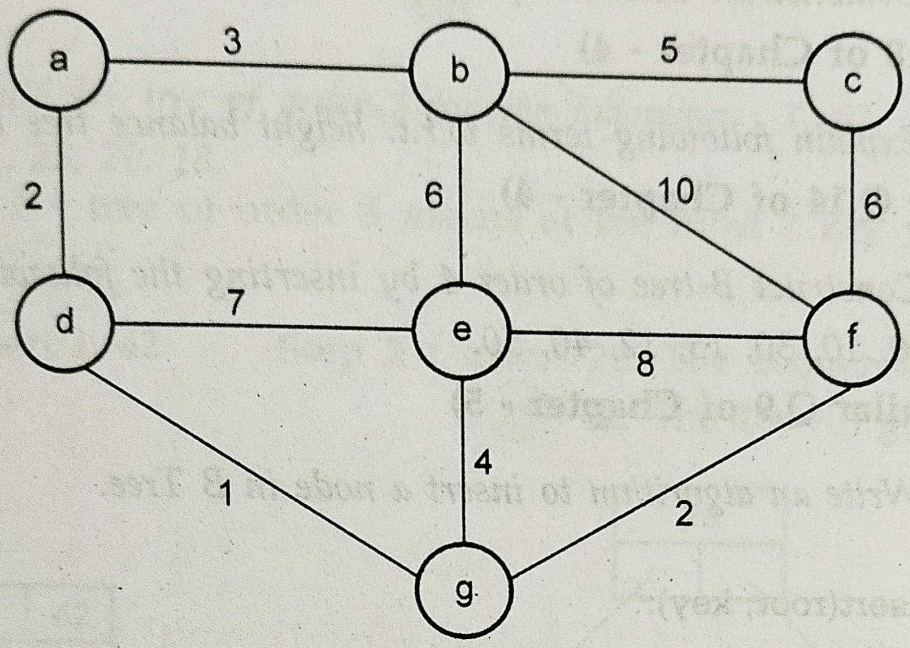


Fig. 3

Q.3 a) *What is OBST ? List binary search tree with 3 words* (w_1, w_2, w_3) = *(do, if, stop) words occurs with probabilities (P1, P2, p3) = (0.4, 0.5, 0.1) find expected access time in each case.*
(Refer Q.11 of Chapter - 4) **[6]**

b) *Build AVL tree for given sequence of data. Show balance factor of all nodes and name the rotation used for balancing the tree 40, 60, 80, 50, 45, 47, 44, 42, 75, 46, 41.* **(Refer Q.20 of Chapter - 4)** **[6]**

c) *Write short notes on :*

i) Red Black tree **(Refer Q.23 of Chapter - 4)**

ii) Splay tree **(Refer Q.30 of Chapter - 4)**

OR

Q.4 a) *Construct OBST for given data using dynamic programming approach. Explain stepwise.* **(Refer Q.10 of Chapter - 4)** **[6]**

POCO M3 | ❤❤ DIPS ❤❤

Index	0	1	2	3
Data	10	20	30	40
Frequency	4	2	6	3

b) *Demonstrate deletion operation in AVL with example.*
(Refer Q.18 of Chapter - 4) [6]

c) *Explain following terms w.r.t. height balance tree LL, RR, LR, RL.* **(Refer Q.14 of Chapter - 4)** [5]

Q.5 a) *Construct B-tree of order 4 by inserting the following data one at a time 20, 10, 30, 15, 12, 40, 50.*
(Refer similar Q.9 of Chapter - 5) [6]

b) *Write an algorithm to insert a node in B Tree.* [6]

Ans. :

```
function insert(root, key):
  if root is empty:
    create a new leaf node with key as its only element
  else :
    find the appropriate leaf node to insert the key
    if the key is already present in the tree:
      return an error message indicating that the value already
exists
    else:
      if the leaf node has space for the new key:
        insert the key in the leaf node in sorted order
      else :
        split the leaf node into two nodes
        promote the middle key to the parent node
        repeat the split operation until there is a parent node
        with space for the new key
  if the root is full :
    split the root into two nodes
```

create a new root node with the middle key as its only element

set the two split nodes as children of the new root node

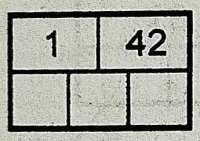
c) *Construct the B+ tree of order 4 for the following data : 1, 4, 7, 10, 17, 21, 31, 25, 19, 20, 28, 42.* **(Refer Q.16 of Chapter - 5)** **[6]**

OR

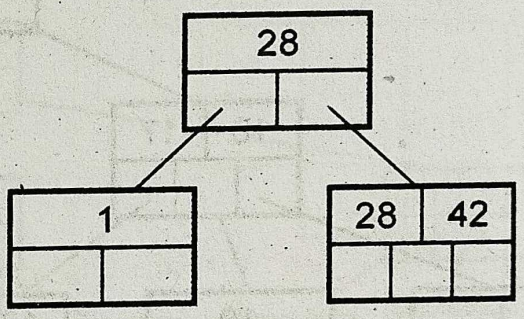
Q.6 a) *Build B+ tree of order 3 for the following : 1, 42, 28, 21, 31, 10, 17, 7, 31, 25, 20, 18.* **[6]**

Ans. : The B + tree of order 3 means at the most 2 key values are allowed in a tree.

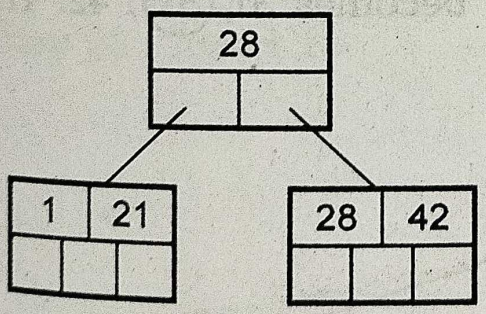
Step 1 : Insert 1, 42



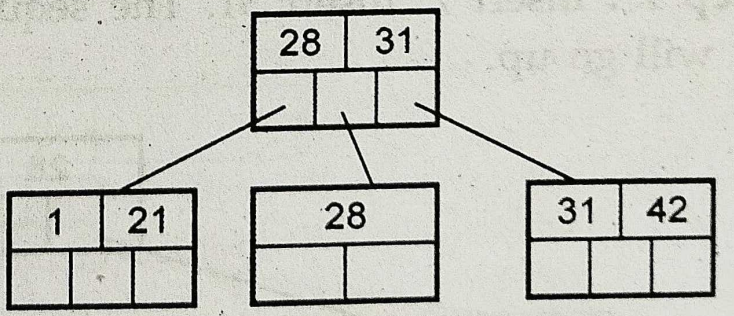
Step 2 : Insert 28. The sequence becomes 1, (28), 42. The 28 will go up.



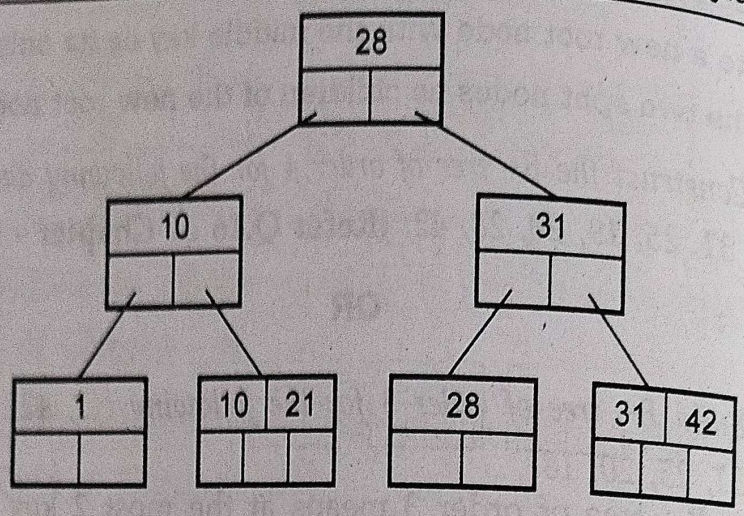
Step 3 : Insert 21.



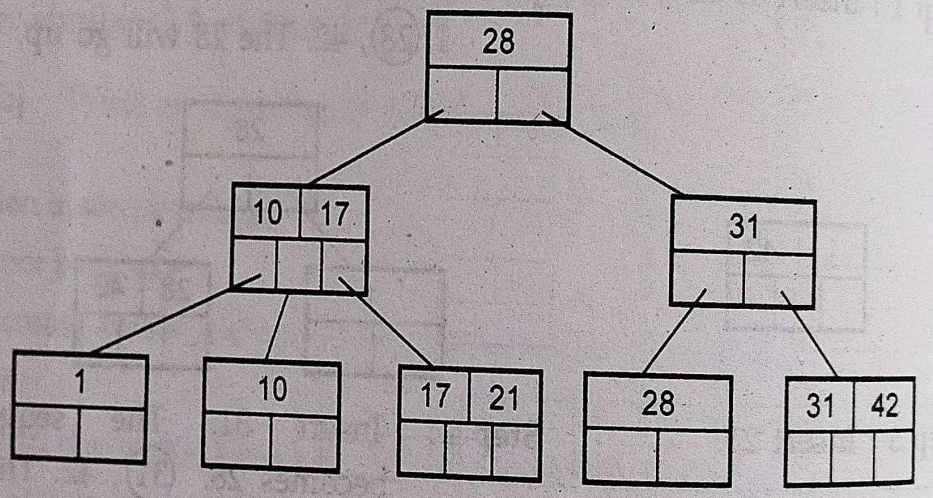
Step 4 : Insert 31. The sequence becomes 28, (31), 42. The 31



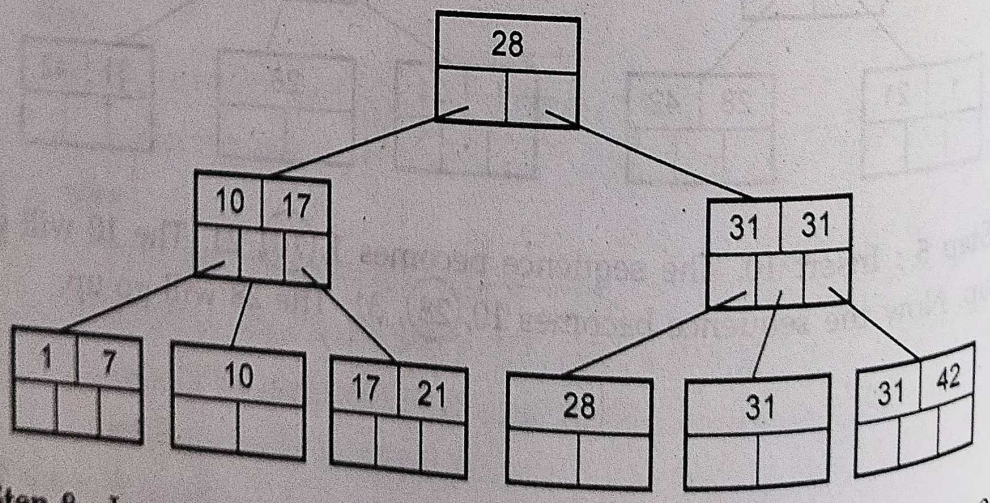
Step 5 : Insert 10. The sequence becomes 1, (10), 21. The 10 will go up. Now the sequence becomes 10, (28), 31. The 28 will go up.



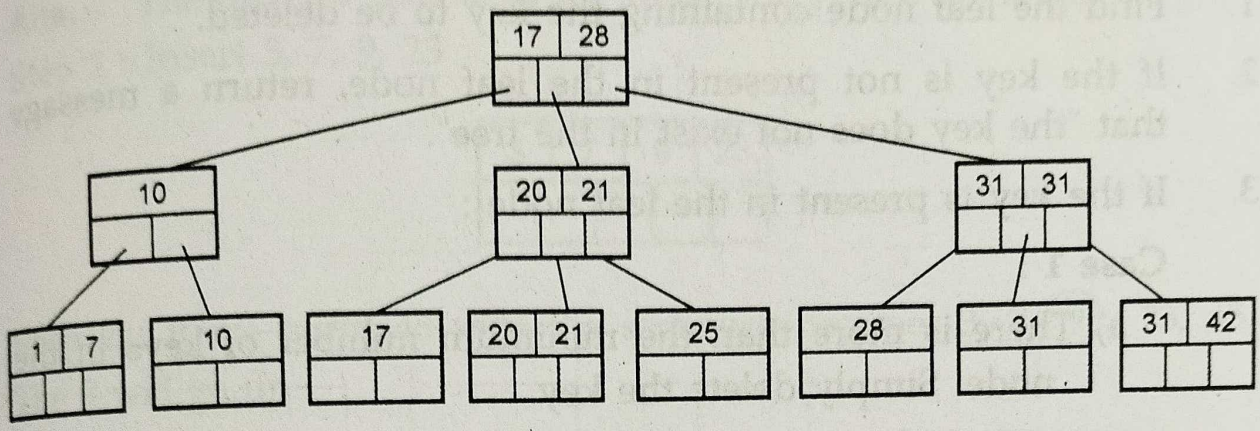
Step 6 : Insert 17. The sequence becomes 10, (17), 21. The 17 will go up.



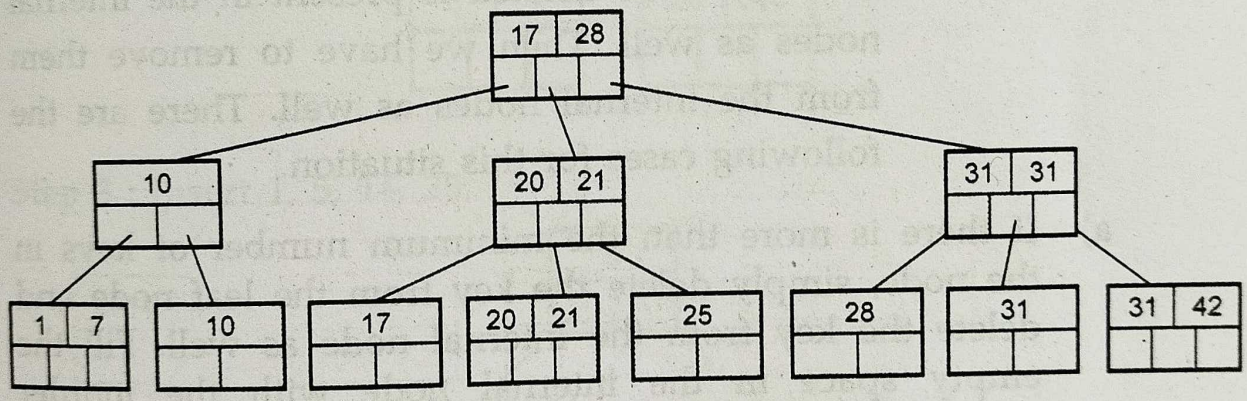
Step 7 : Insert 7. Insert 31. The sequence becomes 31, (31), 42. The 31 will go up.



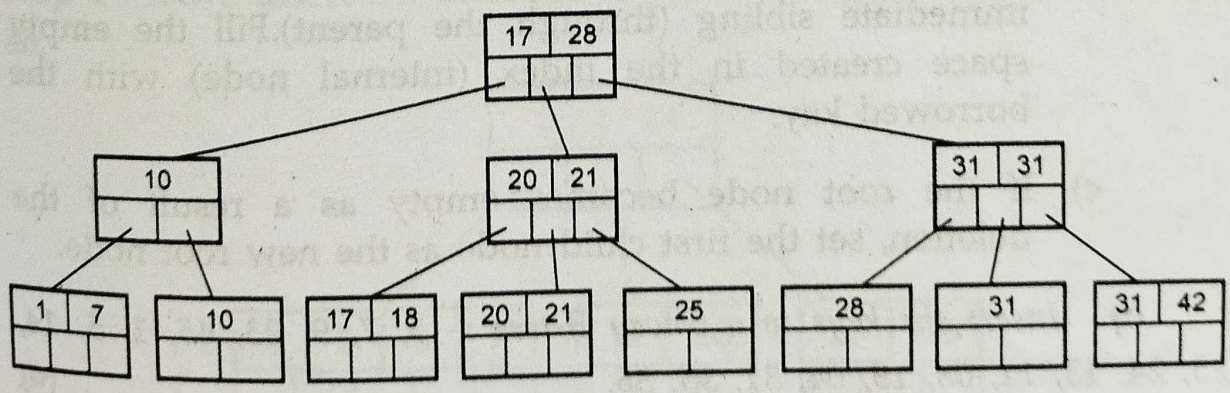
Step 8 : Insert 25. The sequence becomes 17, (21), 25. The 21 will go up. The sequence becomes 10, 17, 21. Now 17 will go up.



Step 9 : Insert 20. The sequence becomes 17, 20, 21. The 20 will go up.



Step 10 : Insert 18



b) *Write an algorithm to delete a node from B+ tree.* **[6]**

Ans. :

1. Find the leaf node containing the key to be deleted.
2. If the key is not present in the leaf node, return a message that "the key does not exist in the tree".
3. If the key is present in the leaf node :

Case 1 :

a) There is more than the minimum number of keys in the node. Simply delete the key.

b) There is an exact minimum number of keys in the node. Delete the key and borrow a key from the immediate sibling. Add the median key of the sibling node to the parent.

Case 2 : The key to be deleted is present in the internal nodes as well. Then we have to remove them from the internal nodes as well. There are the following cases for this situation.

a) If there is more than the minimum number of keys in the node, simply delete the key from the leaf node and delete the key from the internal node as well. Fill the empty space in the internal node with the inorder successor.

b) If there is an exact minimum number of keys in the node, then delete the key and borrow a key from its immediate sibling (through the parent).Fill the empty space created in the index (internal node) with the borrowed key.

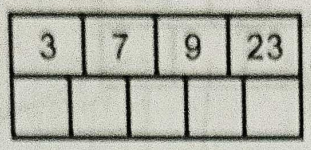
c) If the root node becomes empty as a result of the deletion, set the first child node as the new root node.

c) *Insert the keys to a 5-way B-tree :* 3, 7, 9, 23, 45, 1, 5, 14, 25, 24, 13, 11, 08, 19, 04, 31, 35, 56.

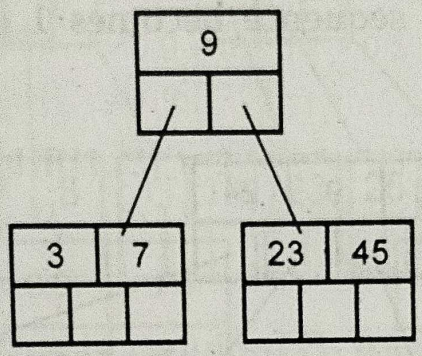
[6]

Ans. : The order 5 means at the most 4 keys are allowed.

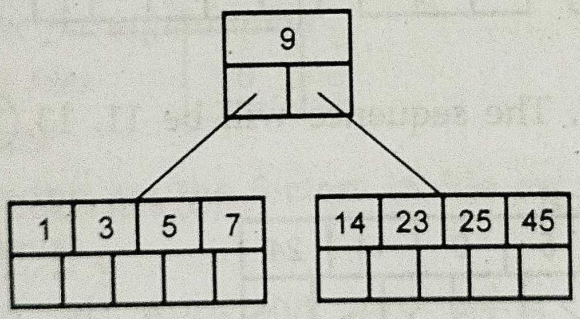
Step 1 : Insert 3, 7, 9, 23



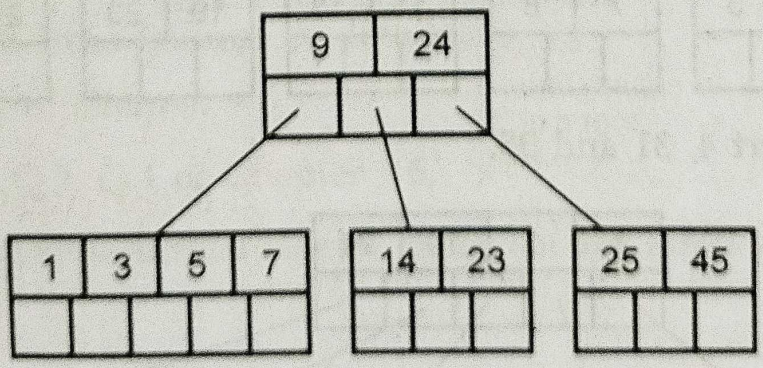
Step 2 : If we insert 45 then the sequence becomes 3, 7, 9, 23, 45. The 9 will go up



Step 3 : Insert 1, 5, 14, 25.

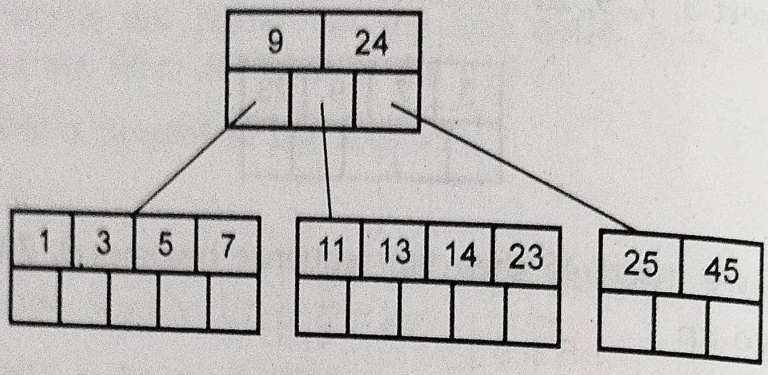


Step 4 : If we insert 24, then the sequence becomes 14, 23, 24, 25, 45.

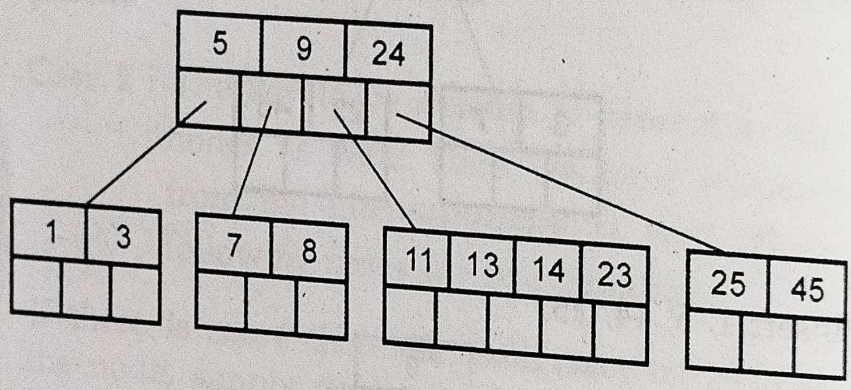


CO M3 | ❤❤ DIPS ❤❤

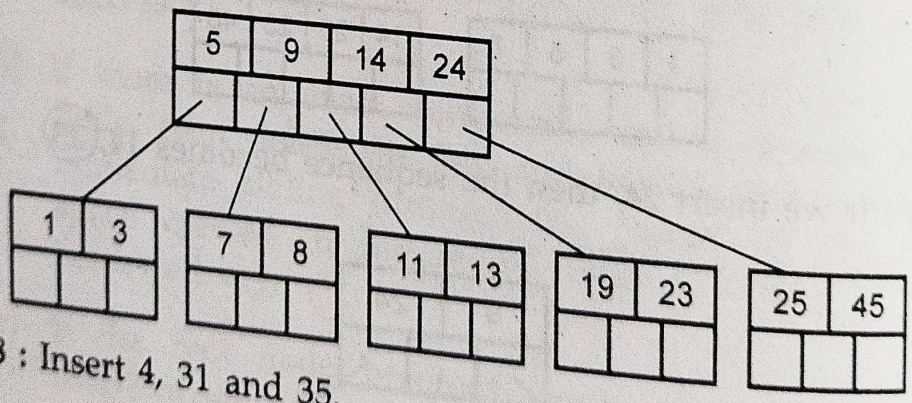
Step 5 : Insert 13, 11.



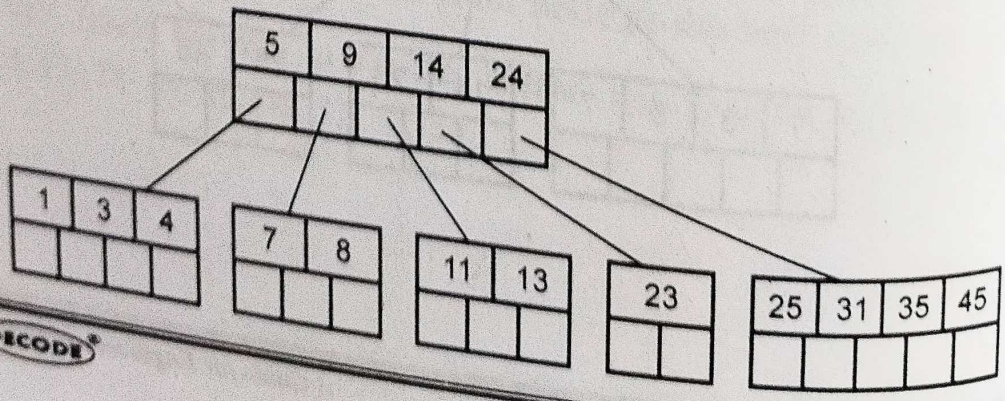
Step 6 : Insert 8. The sequence becomes 1, 3, (5), 7, 8. The 5 will go up.



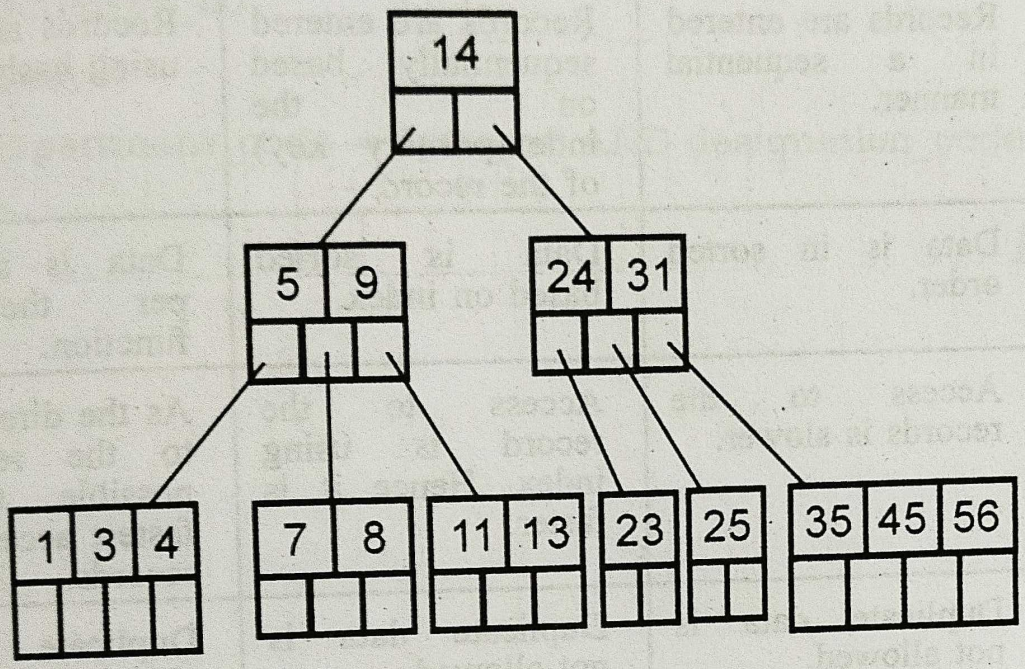
Step 7 : Insert 19. The sequence will be 11, 13, (14), 19, 23. The 14 will go up.



Step 8 : Insert 4, 31 and 35.



Step 9 : Insert 56. The sequence becomes 25, 45, 31, 35, 56. The 31 will go up. Then the sequence **becomes** 5, 9, 14 , 24, 31. The 14 will go up. Finally B-tree will be



This is required B- tree

Q.7 a) *Write short notes on :*

i) Factors affecting the file organization

ii) Indexed sequential files **[6]**

iii) Indexing techniques

Ans. : i) Following are the factors in file organization -

1. Fast data retrieval.
2. High amount of work for processing data input and maintenance transaction.
3. Efficient use of storage space.
4. Protection from failures or data loss.

ii) Refer Q.11 of Chapter - 6.

iii) Refer Q.2, Q.3, Q.4 of Chapter - 6.

b) *Compare sequential indexed sequential and direct access files.* **[6]**

POCO M3 | ❤️❤️DIPS ❤️❤️

Ans. :

Sr. No	Sequential file	Indexed sequential file	Direct access file
1.	Records are entered in a sequential manner.	Records are entered sequentially based on the index(primary key) of the record.	Records are entered using hash function.
2.	Data is in sorted order.	Data is sorted based on index.	Data is sorted as per the hash function.
3.	Access to the records is slower.	Access to the record is using index. Hence it is faster.	As the direct access to the record is possible, there is fastest access to the records.
4.	Duplicate data is not allowed.	Duplicate data is not allowed.	Duplicate data is not allowed.
5.	There is no key	Index key is used.	Hashed key or direct access key is used.
6.	Data is stored on the tape.	Data is stored on disk.	Data is stored on the disk.

c) *Explain any 4 modes of opening the file in C or C++.* **(Refer Q.5 of Chapter - 6)** [5]

OR

Q.8 a) *Explain following operations carried out on sequential files. i) Add ii) Delete iii) Search* **(Refer Q.9 of Chapter - 6)** [6]

b) *Explain any 3 operations carried out on sequentail file and its pseudo code.* **(Refer Q.9 of Chapter - 6)** [6]

c) *A file of employees records, has 'employee no' as primary key and the 'department code' and the 'designation code' as the secondary keys. Write a procedure to answer the following query - 'Which employees from systems department are above designation level 4 ?* [5]

Ans. : Retrieve all the records from employee file where department_code='system' and designation level > 4. The Structured query language(SQL) is as follows -

```
SELECT employee_no
FROM employee
WHERE department_code ='Systems' AND designation_code>4
```

END...

OCO M3 | ❣❣DIPS ❣❣