www.AllAbtEngg.com

PONJESLY COLLEGE OF ENGINEERING
NAGERCOIL.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SEMESTER III

CS8391- DATA STRUCTURES

UNIT III
NON LINEAR DATASTRUCTURES - TREES

Tree ADT – tree traversals - Binary Tree ADT – expression trees – applications of trees – binary search tree ADT –Threaded Binary Trees- AVL Trees – B-Tree - B+ Tree - Heap – Applications of heap.

AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com

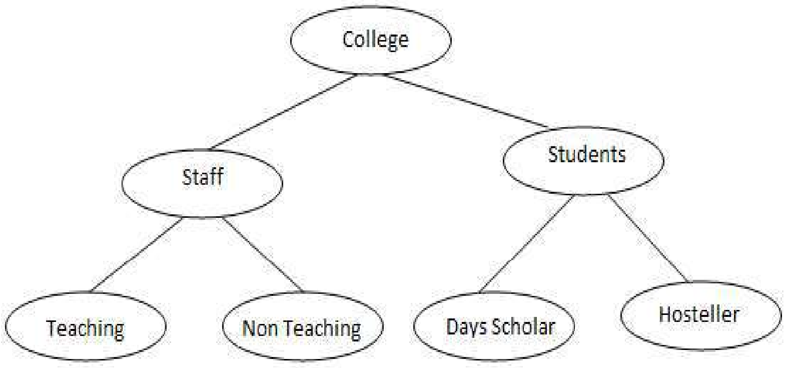
Q1) a)Define Tree.(2)

Or

Q1) b) What do you mean by Tree data Data Structure.(2)

ANSWER

A TREE is a finite set of one or more nodes such that there is a specially designated node called the ROOT and zero or more non empty subtrees T1,T2,T3,..............Tk each of whose roots are connected by a directed edge from ROOT R.



Q2) a) Write about the terms of Tree Data Structure.(8)

Or

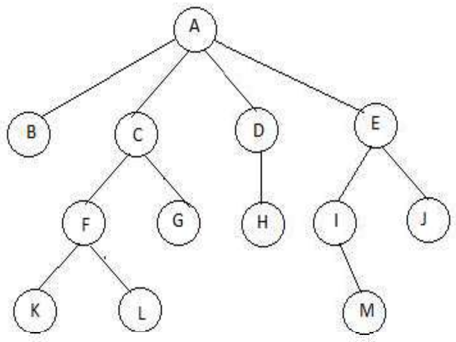
Q3) b) Define Basic Terminologies of Tree with examples.(13)

ANSWER

Basic Terminologies

<u>Node:</u> Item of information.

<u>Root or Parent:</u> A node which doesn't have a parent. A is a ROOT node.



<u>Leaf Or Child:</u>

A node which doesn't have child is called leaf or terminal node.b,K,L,G,H,M,J are leaf nodes.

<u>Siblings:</u>

Children of the same parents are said to be siblings.

AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com

B,C,D,E are siblings.
F,g are siblings
I,J, are siblings
K,L,are siblings.

Path:

- A path from node n1 to nk is defined as a sequence of nodes n1,n2,n3.............nk such that ni is the parent of ni+1 for 1<i<k.
- In a tree there is exactly only one path from root to each node.
- Path from A to L is A-C-F-L.where A is a parent for C,C is a parent for F and f is a parent for L.

Length: The length is defined as a number of edges on the path.The length for path a to l is 3.

Degree: The number of subtrees of anode is called its degree.

Degree of A is 4.
Degree of C is 2.
Degree of D is 1.
Degree of H is 0.

The degree of the tree is the maximum degree of any node in tree.
The degree of tree is 4.

Level:

The level of a node is defined by initially letting the root be at level one,if a node is at level L then its children are at level L+1.

Level of A is 1.
Level of B,C,D are 2.
Level of F,G,H,I,J is 3.
Level of K,L,M, is 4.

Depth: For any node n, the depthof n is the length of the unique path from root to n.

The depth of the root is 0.
Depth of node F is 2.
Depth of node L is 3.

Height: For any node n,the height of the node n is the length of the longest path from n to the leaf.

The height of the leaf is 0.
Height of node F is 1 and L is 0.

AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com

Ancestor And Descendant Nodes:

If there is a path from n1 to n2,then n1 is an ancestor of n2 and n2 is a descendant of n1. If n1=n2 then n1 is a proper ancestor of n2 and n2 is a proper descendane of n1.

A is a ancestor of B,C,D,E.

B,C,D,E is descendant of A.

Note:

- The height of the tree=height of the root
- Depth of the tree= Height of the tree.

Q3) a) Explain the tree traversals with algorithms and examples.(13)

Or

Q3) b) Explain the tree traversals. Give all the essential aspects.(15)

ANSWER:

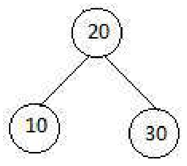
TREE TRAVERSAL

- Traversing means visiting each and every nodes only once.
- Tree Traversal is a method for visiting all the nodes in the tree exactly once.
- There are three types of tree traversal techniques namely,
  - ✓ Inorder traversal
  - ✓ Preorder traversal
  - ✓ Postorder traversal

INORDER TRAVERSAL:

The inorder Traversal of a binary tree is performed as,

- Traverse the leftsubtree in order.
- Visit the root or parent
- Traverse the right subtree in order.



- Inorder Traversal : 10 – 20 – 30
- The inorder traversal of the binary tree for an arithmetic expression is in infix form.

AllAbtEngg Android Application for Anna University, Polytechnic & School

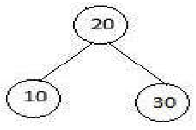
www.AllAbtEngg.com

Routine to perform in order traversal

```
void inorder (Tree T)
        {
                if(T!=NULL)
                {
                     inorder(T→left);
                     printf("%d",T→data);
                     inorder(T→right);
                }
}
```

PREORDER TRAVERSAL:

- The preorder Traversal of a binary tree is performed as,
  - Visit the root or parent.
  - Traverse the left subtree in pre order.
  - Traverse the right subtree in preorder.



- preorder Traversal 20 – 10- 30
- Pre order traversal of the binary tree for an arithmetic expression is in prefix form.

Routine to perform Pre order traversal

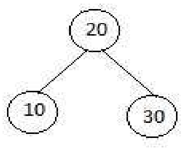
```
void preorder (Tree T)
        {
                if(T!=NULL)
                {
                     printf("%d",T→data);
                     preorder(T→left);
                     preorder(T→right);
                }
        }
```

POSTORDER TRAVERSAL:

- The postorder Traversal of a binary tree is performed as,
  - Traverse the leftsubtree in pre order.
  - Traverse the right subtree in preorder.
  - Visit the root or parent.

AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com

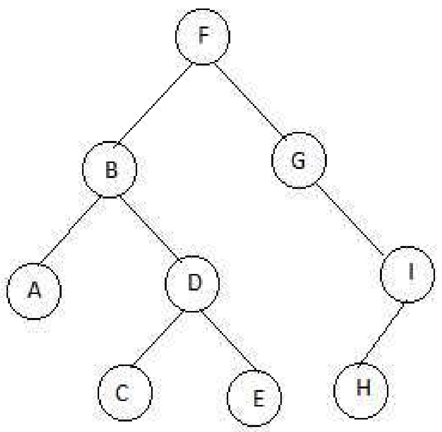


- Post order traversal of the binary tree for an arithmetic expression is in postfix form.

Routine to perform post order traversal

```
void postorder (Tree T)
        {
                if(T!=NULL)
                {
                     postorder(T→left);
                     postorder(T→right);
                     printf("%d",T→data);

                }
         }
```

Example: Binary Tree



- Inorder Traversal of the fig2 is A,B,C,D,E,F,G,H,I.
- preorder Traversal of the fig2 is F,B,A,D,C,E,G,I,H.
- Postorder Traversal of the fig2 is F,B,A,D,C,E,G,I,H

Q4) a) i) Define Binary tree.(2)

ii) What are the types of Binary Tree?(2)

iii) How do you represent Binary Tree?(2)

iv) What are the operations we can perform on Binary Tree?(2)

Or

Q4) b) Explain about Binary Tree in detail.(13)

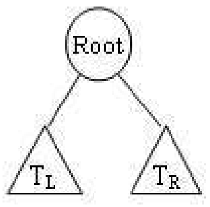
AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com

ANSWER:

BINARY TREE ADT

- Binary tree is a tree in which no node can have more than two children



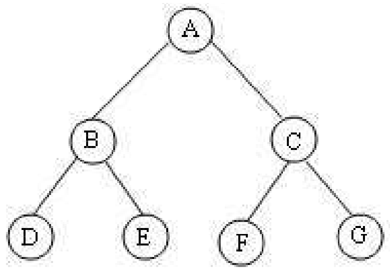
- A binary tree is a finite set of data items which is either empty or consists of a single item called the root and two disjoint binary trees called left subtree and right subtrees. The minimum degree of any node is atmost two and the depth of an average binary tree is smaller than N Average depth is O(n). Maximum number of nodes of level i of a binary tree is 2i+1

TYPES OF BINARY TREE:

1) Full Binary Tree
2) Complete Binary Tree

1) Full Binary Tree:

- A full binary tree of height h has $2^{h+1}-1$ nodes (ie) it contains maximum possible number of nodes in all level.



Here height is 2

No of nodes are $2^{h+1}-1 = 2^{2+1}-1$

$= 2^{3}-1 = 8-1 = 7$

OR

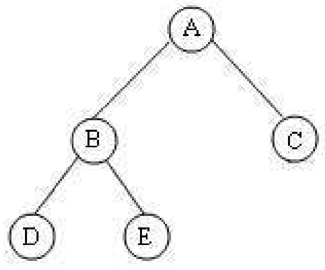
Level	Nodes
0	1
1	2
2	4

AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com

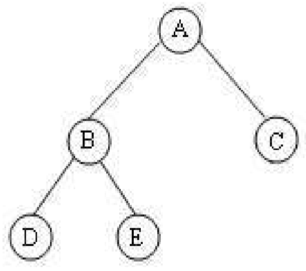
2) Complete binary tree:

- A complete binary tree of height h has nodes between 2^{h} and 2^{h+1}. In the bottom level the elements should be filled from left to right.



Here height is 2.
Nodes are between 2^{h} and $2^{h+1} = 2^{2}$ and 2^{3}
= 4 and 8

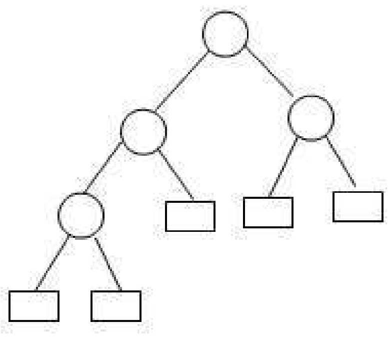
- Here there are nodes.
- ***Strict Binary Tree:***
    - Every non terminal node in a binary tree consists of a non empty left subtre and right subtree then such a tree is called strict binary tree.



- ***Extended Binary Tree***:
    - In each node of a tree has either o0 or 2 children.In that case the nodes with 2 children are called internal node and the nodes with 0 children are called external nodes.

AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com

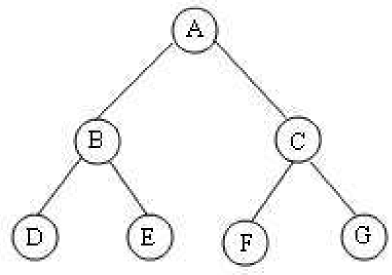


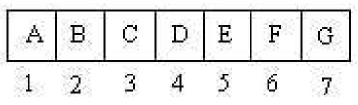
REPRESENTATION OF A BINARY TREE:

- There are two ways for representing binary tree .They are,
  1) Linear representation
  2) linked Representation

1) Linear Representation

- The elements are represented using arrays. For any element in position i, the left child is in position 2i,the right child is i position (2i+1), and the parent is in position (i/2)





A	B	C	D	E	F	G
1	2	3	4	5	6	7

For eg,i=1 for node a

Its left child at 2i=2*1=2==== B

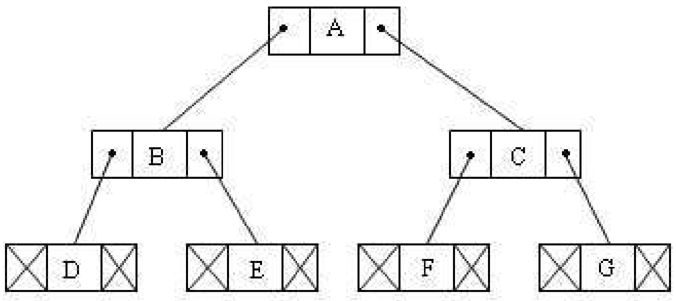
Its right child at 2i+1=2*1+1=2+1=3====C

2) Linked Representation:

- The elements are represented using pointers.Each node has 3 fields.
  1) Pointer to left subtree
  2) Data
  3) Pointer to right subtree.
- In leaf nodes,both pointers are NULL.

AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com



Declaration Of A Binary Tree

```
Struct TreeNode
{
int Element;
Struct TreeNode *Left;
Struct TreeNode * right;
};
```

<u>OPERATIONS OF BINARY TREE:</u>

- The basic operations that are performed onn binary tree are,

  1) CREATE

     It create an empty binary tree.

  2) MAKEBT

     It create a new binary tree having a single node with data field set to some value.

  3) EMPTYBT

     Return true,if the binary tree is empty else return false

  4) LCHILD

     Return a pointer to the left child of the node else return NULL pointer.

  5) RCHILD

     It rerurns a pointer to the right child of the node else return NULL pointer.

  6) FATHER

     It returns a pointer to the father of the node else it returns a NULL pointer.

  7) SIBLING

AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com

Reurns a pointer to the brother of the node else it return a NULL pointer.

8) DATA

Returns the content of the node.

Various operations are,

1) Tree Traversal
2) searching for a node
3) Insertion of a node
4) Deletion of a node
5) Copying the binary tree.

Q5) a) Define Expression Tree. How do you construct a expression tree.(13)

Or

Q5) b) Construct a Expression Tree for the expression ab+c*

ANSWER:

EXPRESSION TREE:

Expression tree is a binary tree in which the leaf node is operands and the interior nodes are operators. Like binary tree, expression tree can also be traversed by inorder, preorder and postorder traversal.

Constructing an Expression Tree:

- Let us consider postfix expression given as an input for constructing an expression tree by performing the following steps:
  1) Read one symbol at a time from the postfix expression
  2) Check whether the symbol is an operand or operator.
     a) If the symbol is an operand, create a one node tree and push a pointer on to the stack.
     b) If the symbol is an operator pop two pointers from the stack namely T1 and T2 and from a new tree with root as the operator and T2 as a left child and T1 as a right child a pointer to this new tree is then pushed onto the stack.

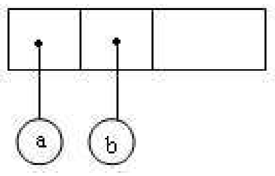
Eg:

ab+c*

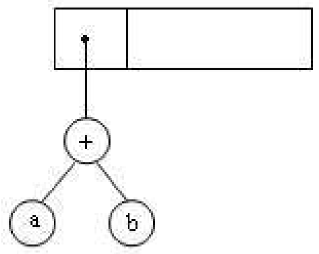
Step 1: The first two symbols are operand, so create a one node tree and push the pointer onto the stack.

AllAbtEngg Android Application for Anna University, Polytechnic & School

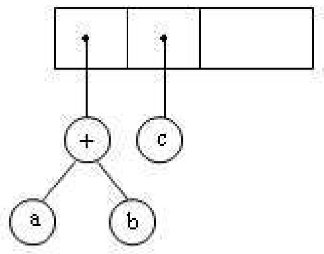
www.AllAbtEngg.com



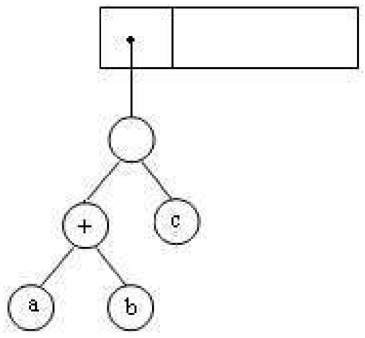
Step 2: Next + symbol is read,so two pointers are popped ,a new tree is formed and a pointer to this is pushed onto the stack.



Step 3: Next the operand C is read, so one node tree is created and the pointer is pushed onto the stack.

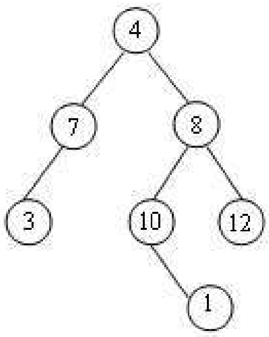
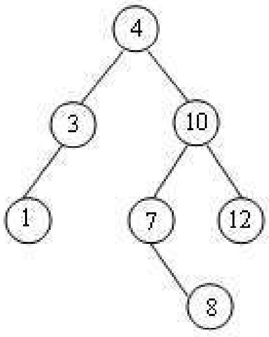


Step 4: Now * is read, so two trees are merged and the pointer to the final tree is pushed onto the stack.



AllAbtEngg Android Application for Anna University, Polytechnic & School

Q6) a) Differentiate Binary Tree and Binary Search Tree.
ANSWER:
Comparison Between Binary Tree & Binary Search Tree

BINARY TREE	**BINARY SEARCH TTREE**
1)It is a tree which has atmost 2 children	It is a binary tree in which the key values in the left node is less than the root and the key values in the right node is greater than the root.
2)It doesn't have any order 	

Q7) a) Explain Binary Search Tree ADT in detail. (15)

Or

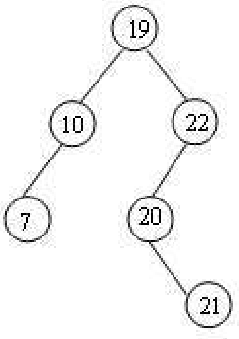
Q7) b) Define Binary Search Tree.Write an algorithm for binary search tree operations. Give example.(15)

ANSWER

BINARY SEARCH TREE ADT

Definition:

- Binary search tree is a binary tree in which for every node X in the tree,the values of all the keys in its left subtree are smaller than the key value in X and the values of all the keys in its right subtree and larger than the key value in X.



www.AllAbtEngg.com

Declaration Of A Binary Search Tree

```
Struct TreeNode
{
int Element;
Struct  SearchTreeNode *Left;
Struct  SearchTreeNode * right;
};
```

OPERATIONS ON BINARY SEARCH TREE

1) Make Empty:

- This operation is mainly for initialization when the programmer prefer to initialize the first element as a one node tree.

Routine To Make A Empty Tree

```
SearchTree MakeEmpty (SearchTree T)
{
If(T!= NULL)
{
MakeEmpty ( T→ Left);
MakeEmpty (T→Right);
Free(T);
}
Return NULL;
}
```

2) Insert:

- To insert a element X into the tree,

  1) Check with the root node T
  2) If it is less than the root,
     Traverse the left subtree recursively until; it reaches the T→left equals to NULL.Then X is placed in T→Left.
  3) If X is greater than the root,

AllAbtEngg Android Application for Anna University, Polytechnic & School

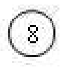
www.AllAbtEngg.com

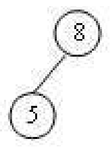
Traverse the right subtree recursively until it reaches the T →right equals to NULL .then X is placed in T→Right.

Routine To Insert Into A Binary Search Tree

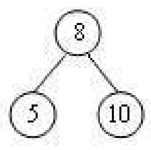
```
SearchTree Insert(int X, SearchTree T)
{
If(T == NULL)
{
T= malloc ( size of (Struct TreeNode));
If(T!= NULL) // First element is placed in the root.
{
T→ Element = X;
T→ Left = NULL:
T→ Right = NULL;
}
}
Else if(X< T → Element)
T → Left = Insert ( X, T→ Left);
Else if(X> T → Element)
T → Right = Insert ( X, T→ Right);
// Else X is in the tree already
Return T;
}
```

Example:





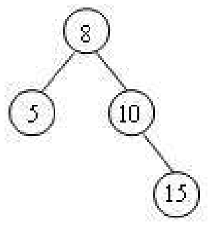
As 5 < 8, Traverse towards left

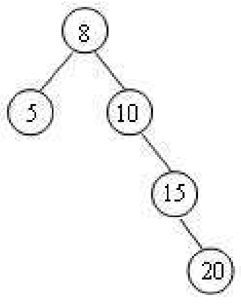


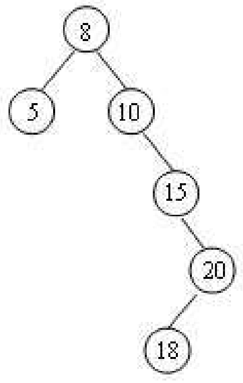
10 > 8, Traverse towards right

AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com







3)Find

1) Check whether the root is NULL if so then return NULL.
2) Otherwise,check the value X with root node value(ie)T→data
   * If X is equal to T→datya,return T
   * If x is less than T→data,
     Traverse the left of T recursively.
   * If x is greater than T→data,traverse the right of T recursively.

<u>Routine For Find Operation</u>

```
Int find(int X, SerachTree T)
{
If (T == NULL)
Return NULL:
If( X < T → Element)
```

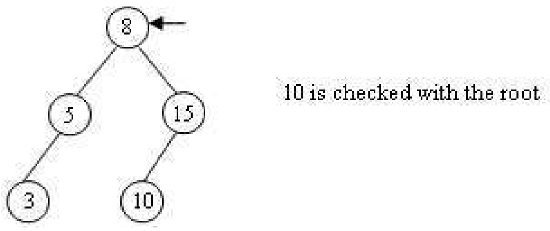
AllAbtEngg Android Application for Anna University, Polytechnic & School

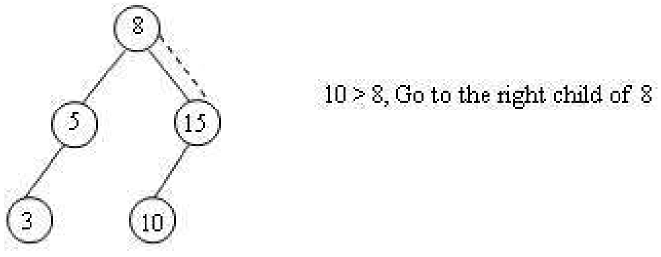
www.AllAbtEngg.com

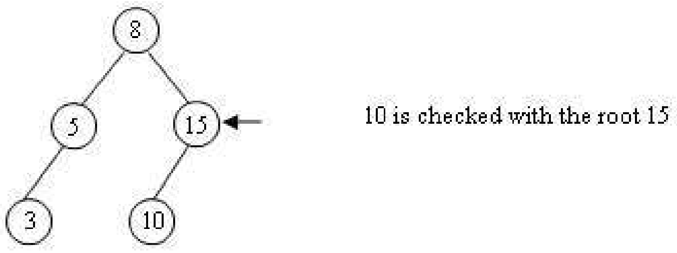
```
Return Find (X , T → Left);
Else if ( X >  T→ Element)
Return  Find ( X, T→ Right);
Else
Return T;   // returns the position of the search element
}
```

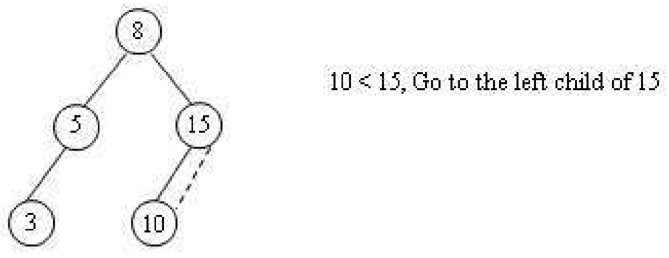
Example:

- To find an element 10 (consider, X= 10)



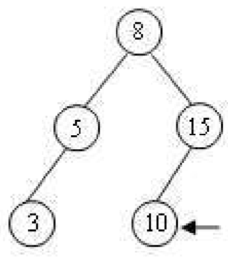






AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com



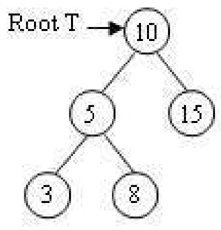
10 is checked with the root 10 (Found)

4) FindMin

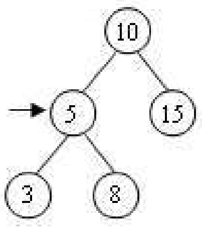
➤ This operation returns the position of the smallest element in the tree. To perform FINDMIN,start at the root and go left as long as there is a left child.The stopping point is the smallest element.

<u>Recursive Routine For Findmin Operation</u>

```
Int FindMin(int X, SerachTree T)
{
If (T == NULL)
Return NULL;
Else If( T → Left = = NULL)
Return T;
Else
Return  FindMin ( T→ Left);
}I
```



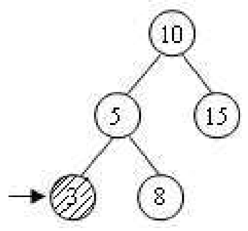
T!= NULL and T--> left != NULL, Traverse left



T != NULL and T --> left != NULL, Traverse left

AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com



Since T --> left is NULL, Return T as a minimum element

Non Recursive Routine For Findmin Operation

```
Int FindMin(int X, SerachTree T)
{
If (T != NULL)
while( T → Left1! = NULL)
T = T → Left;
Return   T;
}l
```

5) Find Max:

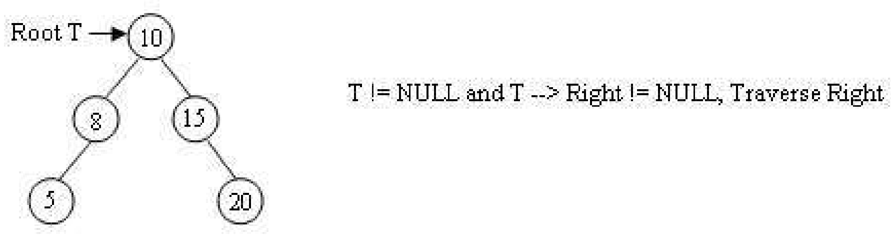
- Find max routine return the position of largest element in the tree. To perform a FINDMAX start at the root and go right as long s there is a right child.The stopping point is the largest element.

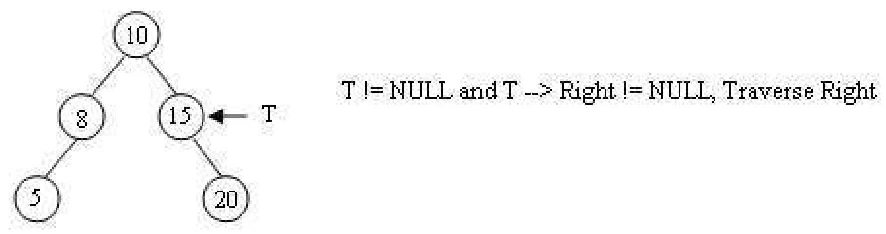
Recursive Routine For Findmax Operation

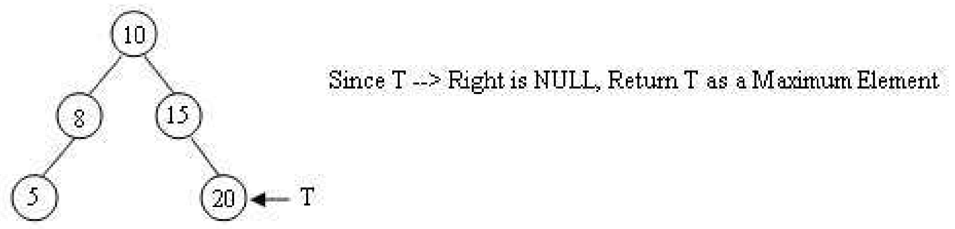
```
Int FindMax(int X, SerachTree T)
{
If (T == NULL)
Return NULL;
Else If( T → Right = = NULL)
Return T;

Else
Return  FindMax ( T→ Right);
}l
```

AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com







<u>**Non Recursive Routine For Findmin Operation**</u>

```
Int FindMax(int X, SerachTree T)
{
If (T != NULL)
while( T → Right1! = NULL)
T = T → Right;
Return   T;
}l
```

6) DELETE

- Deletion peartion is a complex operation in the binary search tree. To delete an element consider the following three possibilities

  Case 1: Node to be deleted is a leaf noe(ie)No children

  Case 2 : Node with one child

  Case 3 : Node with two children

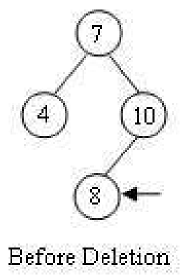
<u>***Case 1 : Node with no children***</u>

- If the node is a leaf node,it can be deleted immediately.

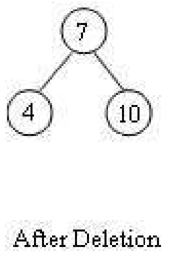
  Delete 8:

AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com



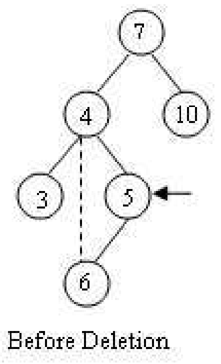
Before Deletion



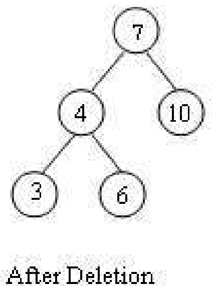
After Deletion

Case 2: Node with one child

- If the node has one child ,it can be deleted by adjusting its parant pointer that points to its child
  node.
  Delete 5



Before Deletion



After Deletion

Case 3: Node with two children

- It is difficult to delete a node which has two children. The general strategy is to replace the data of node to be deleted with its smallest data of the right subtree and recursively delete that node

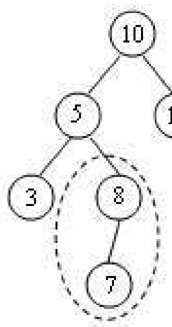
Deletion Routine for Binary Search Trees:

```
SerachTree Delete (int X, SearchTree T)
{
Int Tmpcell;
If( t == NULL)
Error ( “ Element not found’);
Else
If(X < T → Element) // Traverse towards left
T → Left = Delete ( X, T → Left);
Else if( X > T → Element) // Traverse towards right
T --.> Right= Delete ( X , T→ Right);
};
```

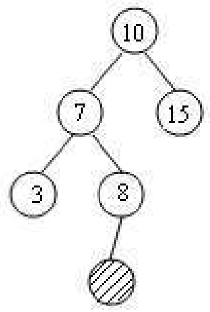
AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com

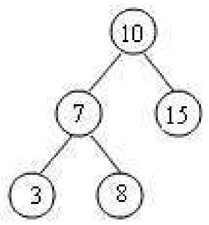
To delete 5



The maximum element at the right subtree is 7.



- Now the value 7 is replaced in the position of 5.
- Since the position of 7 is the leaf delete immediately



After deleting the node 5.

Q8) a) Write down the applications of Trees.(2)
ANSWER:

1) A binary tree is used for manipulation of arithmetic expression.
2) A binary search tree is an appropriate data structure for searching a value and sorting the records.
3) It is also used for construction and maintenance of symbol tables.
4) It plays important role in the area of syntax analysis.
5) It is also used for playing games such as tic-tac-toe, chess, etc .
6) Trees are used in data compression techniques for encoding the compressed data.

Q9) a) Explain clearly the logic behind using the THREADED BINARY TREES in Dta Structure.Draw a labeled diagram to show the working of the Threaded Binary Tree.(15)
Or
Q9) b) What is the need for Threaded Binary Tree.Explain it witn an examples(15)

AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com

ANSWER:

Need For Threaded Binary Tree:

In a Binary Tree or binary search tree, there are many nodes that have an empty left child or empty right child or both.• You can utilize these fields in such a way so that the empty left child of a node points to its inorder predecessor and empty right child of the node points to its inorder successor.

Threaded Binary Tree definition

The structure of a node in a threaded binary tree is a bit different from that of a normal binary tree.

Unlike a normal binary tree, each node of a threaded binary tree contains two extra pieces of information, namely left thread and right thread.

We have the pointers reference the next node in an inorder traversal; called threads The left and right thread fields of a node can have two values:

1: Indicates a normal link to the child node

0: Indicates a thread pointing to the inorder predecessor or inorder successor

Two types of Threading

- ✓ ***One way Threading***
- ✓ ***Two way Threading***

1) One Way Threading:

Thread appears only on the RLINK of a node, pointing to the inorder successor of the node.

2) Two Way Threading

Threads are appear in both the links LLINK and RLINK and points to the inorder predecessor and inorder successor respectively.

Logic Explanation:

- In threaded binary tree,the NULL pointer are avoided.Instead of left NULL pointer the link points to inorder predecessor of that node. Similarly, instead of right NULL pointer the link points to the inorder successor of that node.
- There are two additional fields in each node named as LThread or RThread these fields indicates whether left or right thread is present. Thread present means the NULL link is replaced by a link to inorder predecessor or inorder successor. To represent that there exists thread the Lth or Rth fields are set to 0.

AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com

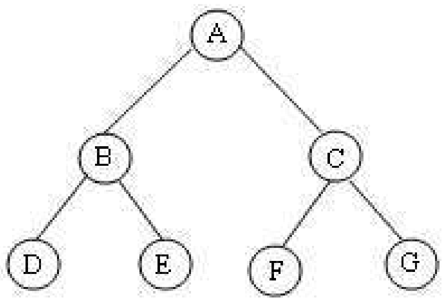
- The basic idea in inorder threading is that the left thread should point to the inorder predecessor and the right thread points to the inorder successor. Here we assuming the head(dummy) node as the starting node and the root node of the tree is attached to left of head(dummy) node.

Node representation:

TLPOINT	LLINK	DATA	RLINK	TRPOINT

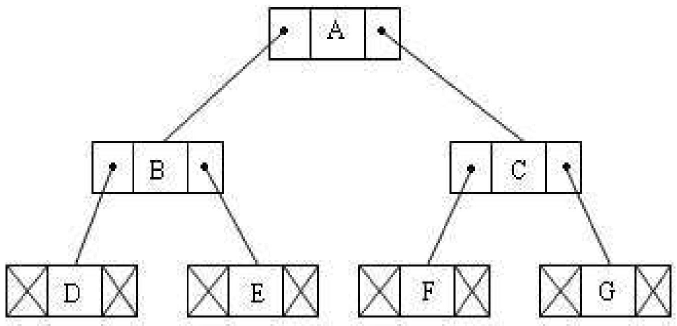
Example :

Consider the following Binary Tree and create Threaded Binary Tree



Solution:

Step1: Represent the binary tree by using linked list representation.



Step 2:Inorder Traversal of Binary Tree

D B E A C F G

Step 3: Nodes D, E, F and G contain left and right null poniters. In threaded binary tree,the NULL pointer are avoided.So find inorder predecessor and inorder successor

1) Inorder predecessor of D is dummy node.

AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com

Inorder successor of D is B

2) Inorder predecessor of E is B.

   Inorder successor of E is A

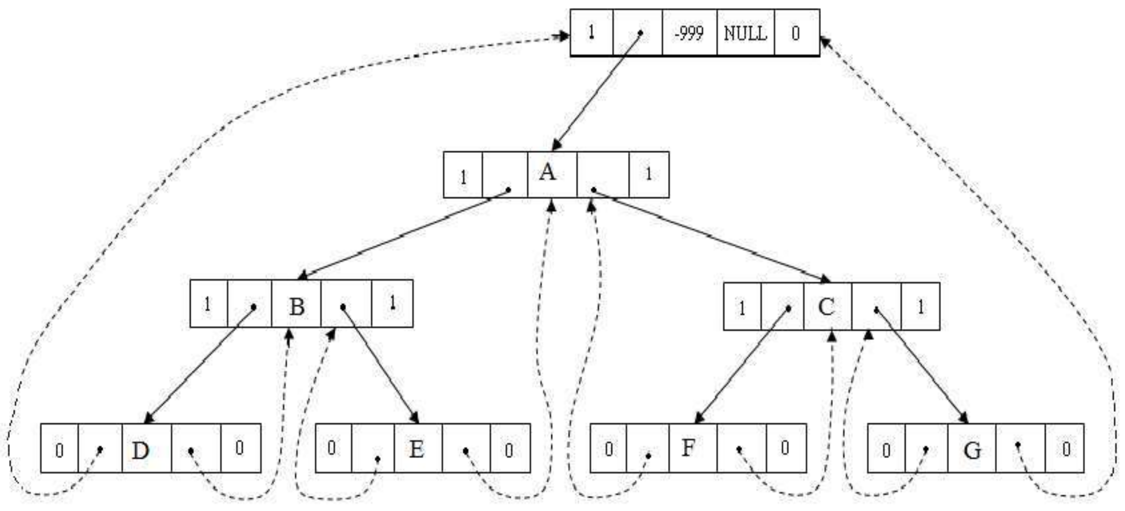
3) Inorder predecessor of F is C.

   Inorder successor of F is G

4) Inorder predecessor of G is F

   Inorder successor of G is dummy node

Step4: .Instead of left NULL pointer the link points to inorder predecessor of that node. Similarly, instead of right NULL pointer the link points to the inorder successor of that node.Hence we will get the theaded binary tree as follows



Advantages:

1) IN THREADED BINARY TREE is avoid NULL pointer. Hence memory wastage in occupying predecessor and successor nodes.

Q10) a) Create a threaded binary tree for the nodes having values. 10,8,6,12,9,11,14(13)

ANSWER

1) initially we will create a dummy node which will act as a header of the tree.

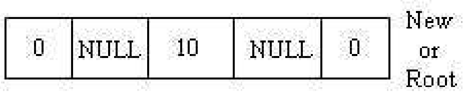
lth	Left	Data	Right	rth
0	NULL	-999	NULL	0

Dummy node

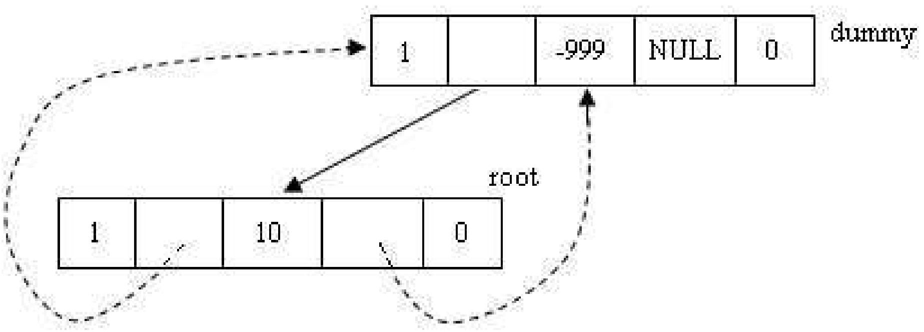
AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com

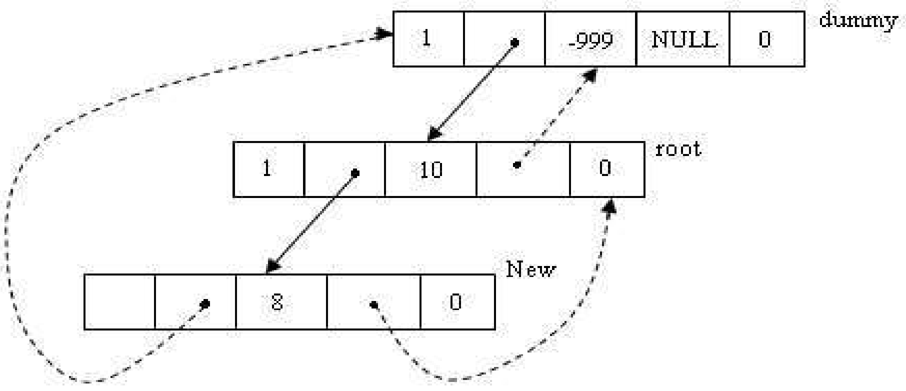
2) Now let us take first value(ie)10.The node with value 10 will be the root node.we will attach this root node as left child of dummy node.



3) The NULL links of root's left and right child will be pointed to dummy.



4) Now next comes 8.The 8 will be compared with 10,as the value is less than 10,we will attach 8 as left child of 10 and we will set root's Lth field to 1,indicating that the node 10 is having the child. Then links are arranged are,



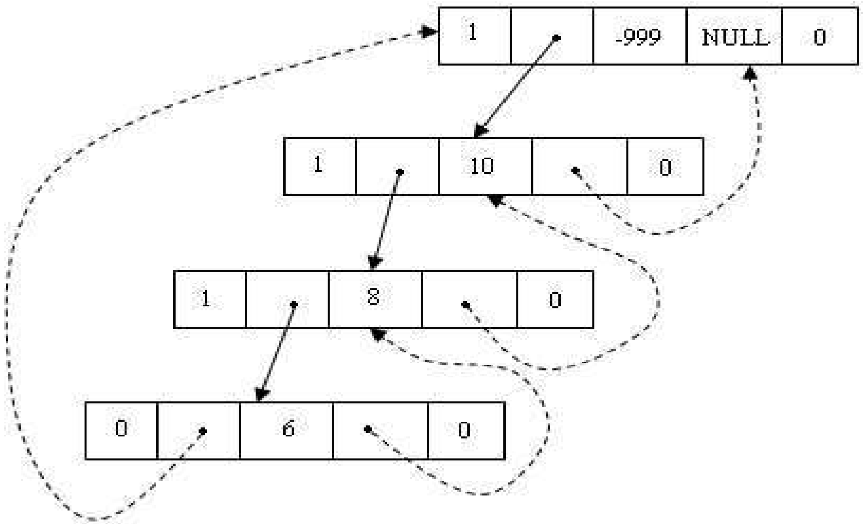
Note:

The left link of node 8 points to its inorder predecessor and right link of node 8 points to its inorder successor.

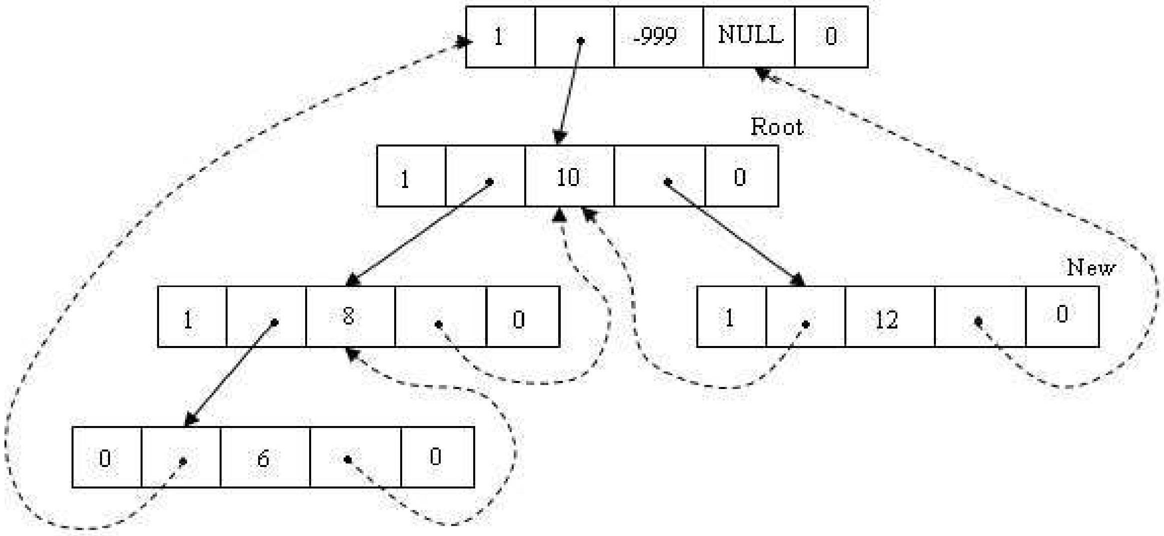
5) Next comes 6.The value 6 will be first compared with root(ie) with 10.As 6 is less than 10,we will move on left subbranch. The 6 will then be compared with 8,so we have to move on left subbranched of 8 but lth of node 8is 0,indicating that there is no left child of 8,so we will attach 6 as left child to 8.

AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com



6) Then comes 12.We will compare 12 with10.As 12 is greater than 10,we will attach the node 12 as right child of 10.

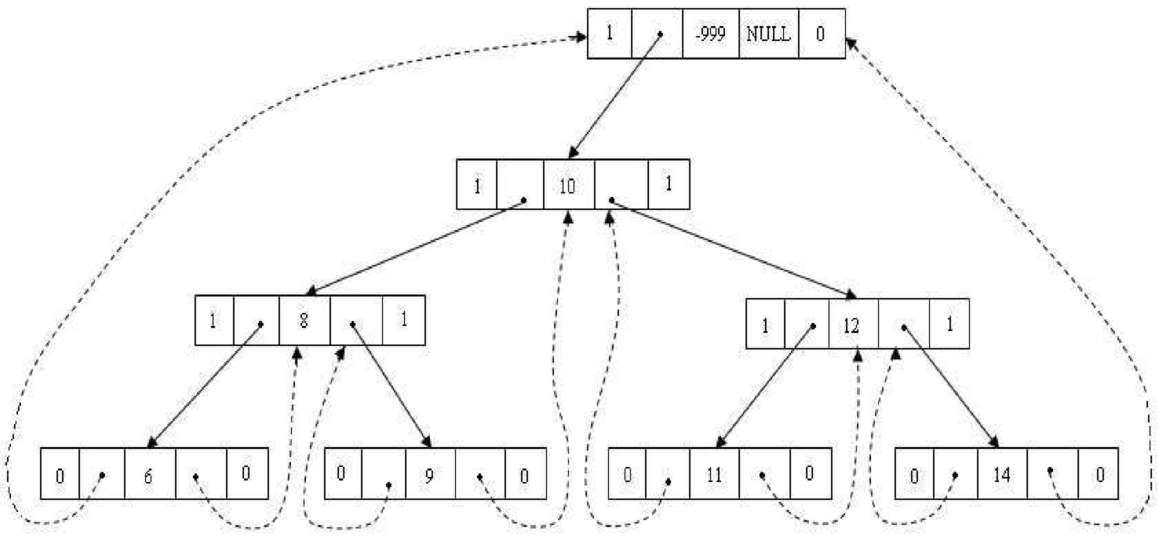


Note that the left field of node 12 points to its inorder predecessor and right field of node 12 points to its inorder successor.

7) Thus by attaching 9,11,14 as appropriate children, the threaded binary tree will look like this

AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com



Q11) a) Define AVL Tree. Explain its rotation operations with example.(15)

Or

Q11) b) Show the results of inserting43, 11, 69, 72 and 30 into an initially empty AVL tree.

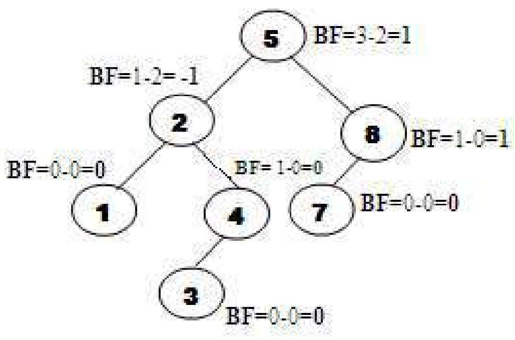
ANSWER:

AVL TREE:

- An AVL (Adelson –Velskii and Landis) tree is a binary search tree with a ***balance condition***. The ***balance information*** is termed as **Balance Factor (BF)**, it is the height of the left sub tree minus height of the right sub tree.

$$\boxed{\textbf{BF=Height of left sub tree-Height of right sub tree}}$$

- For an AVL tree all ***balance factor*** value will be ***+1,0,-1.***
- For example, consider the AVL tree. The balance factor of each node will be calculated as follows,



AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com

Balanced AVL tree with Balance Factor Value

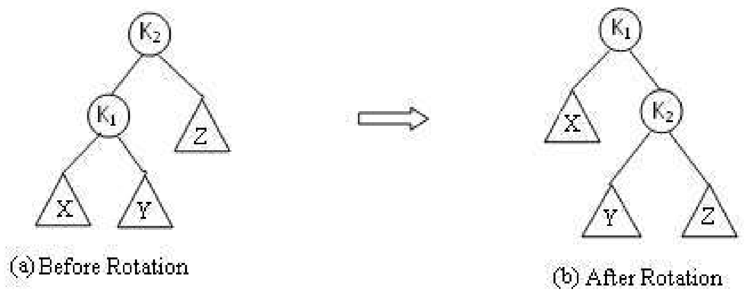
- (ie) An AVL Tree is identical to a binary search tree, except that for every node in the tree, must have balanced factor of 1 or 0 or -1
- Insertion in AVL tree is performed in the same way as it is performed in a binary search tree. However, it may lead to violation in the AVL tree property and therefore the tree may need balancing. The tree can be balanced by applying rotations.
- Then any one of the following conditions would occur due to an insertion in node ***a***.
    - ***CASE 1:*** *An insertion into the left subtree of the left child of node a.*
    - ***CASE 2:*** *An insertion into the right subtree of the left child of node a.*
    - ***CASE 3:*** *An insertion into the left subtree of the right child of node a.*
    - ***CASE 4:*** *An insertion into the right subtree of right child of node.*
- To rebalance the AVL tree we need to perform rotation on the imbalanced subtree. There are two types of rotation are available
    - ***Single Rotation***
    - ***Double Rotation***
- For case 1 and case 4 we need to perform single rotation. And for case 2 and case 3 we need to perform double rotation

SINGLE ROTATION:

- There are two types of single rotations
    1. ***Single rotation with Left (Case 1)***
    2. ***Single rotation with Right (Case 4)***

SINGLE ROTATION WITH LEFT:

General Representation of* Single rotation with Left(*case 1):



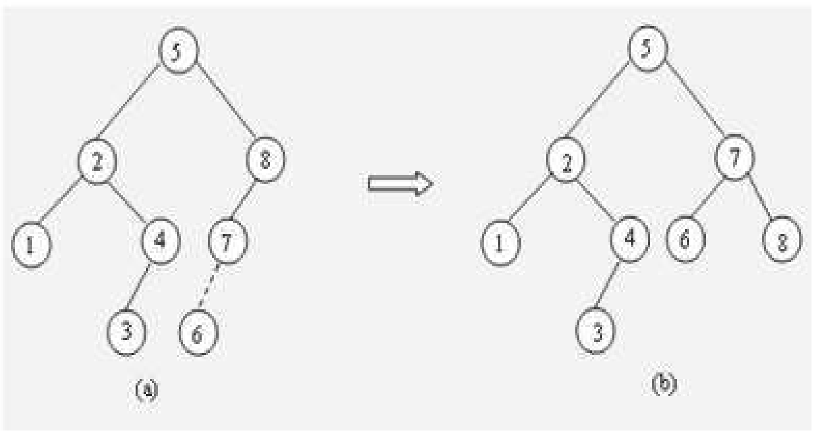
AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com

Routine to perform single rotation with left

```
Static Position  SingleRotateWithLeft (Position K2)
{
        Position K1;
        K1 = K2 → Left;
        K2 → Left = K1 → Right;
        K1 → Right = K2;
        K2 → Height = Max (Height (K2 → Left), Height(K2 → Right))+1;
        K1 → Height = Max (Height (K1 → Left), K2 → Height) +1;
        Return K1; /* New Root */
}
```

Example:-



Before Rotation **After Rotation**

- In the above example, when we try to insert a new node **6** in an AVL tree, imbalance will occur at node **8** (BF=2-0=2). So rotation should be performed.
- Here imbalance is occurred due to the insertion at the left subtree of the left child of node **8.** So ***single rotation with left should be performed***.
- Now we want to rotate the subtree a, based on this the node **7** should be placed as root and the node **8** should be placed as right child of 7.

SINGLE ROTATION WITH RIGHT :

- An insertion into the right subtree of the right child of K1

General Representation of Single rotation with Right (Case 4):

AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com

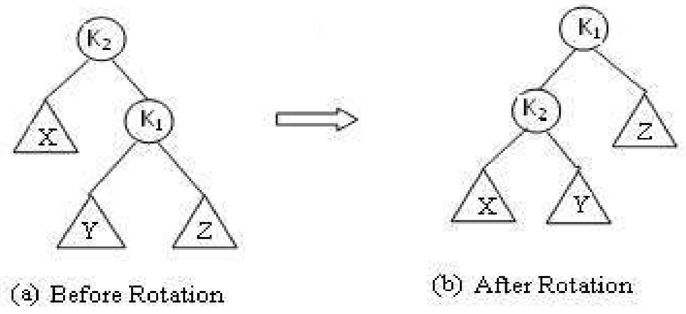


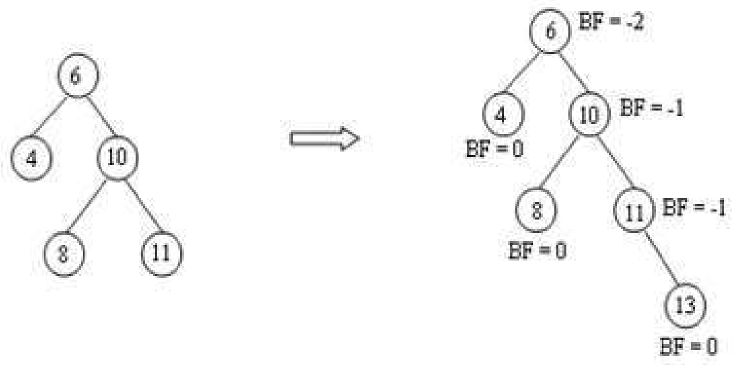
Figure 3.7: Single rotation with Right

<u>Routine to perform single rotation with right</u>

```
Static Position SingleRotateWithRight(Position K2)
{
        Position K1;
        K1 = K2 → Right;
        K2 → Right = K1 → Left;
        K1 → Left = K2;
        K2 → Height = Max(Height(K2 → Left), Height(K2 → Right))+1;
        K1 → Height = Max(Height(K1 → Right), K2 → Height)+1;
        return K1; /* New Root */
}
```

Example:-

- Inserting node **13** in the following AVL tree will leads to imbalance (violate AVL tree property) at the node **6 (BF= -2**). So we want to rebalance it by performing rotation.
- As the imbalance happen due to the insertion at the ***right subtree of the right child of 6,*** we want to perform ***single rotation with right***.
- Now if we rotate the tree in single rotation with right, we get the balanced AVL tree as follows

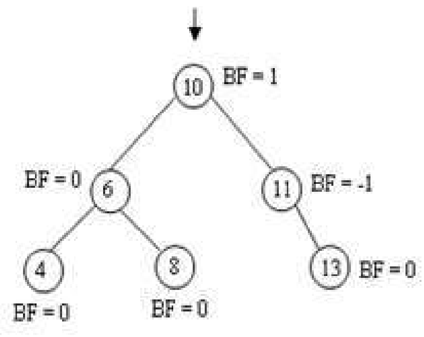


(a): AVL tree with imbalance

AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com

Rotation with Right



(b) Balanced AVL Tree

DOUBLE ROTATION:

- Double rotation is performed for case 2 and case 3.ie) imbalance due to insertion at the left subtree of the right child of node **a** and vice-versa**.**
- There are two types of double rotation. They are,
  - ***Left-Right Double Rotation (Case 2)***
  - ***Right-Left Double Rotation (Case 3)***

LEFT – RIGHT DOUBLE ROTATION:

- The Left-Right double rotation is performed, when an imbalance occur due to the insertion at the ***right subtree of the left child of an node.***

General Representation of Left-Right Double Rotation:

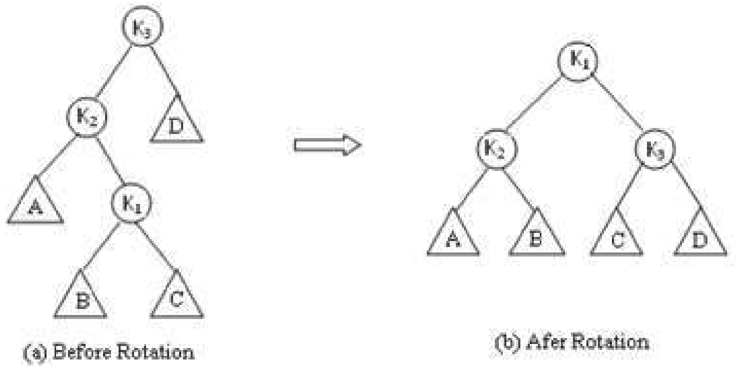


Figure 3.9: Left-Right Double Rotation

- The above double rotation can be done by using the following two single rotation,
  1. Single rotation with right, then
  2. Single rotation with left.

AllAbtEngg Android Application for Anna University, Polytechnic & School

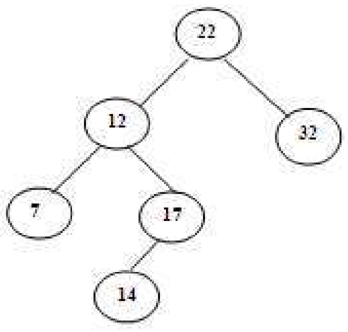
www.AllAbtEngg.com

Routine to perform Left-Right Double Rotation:

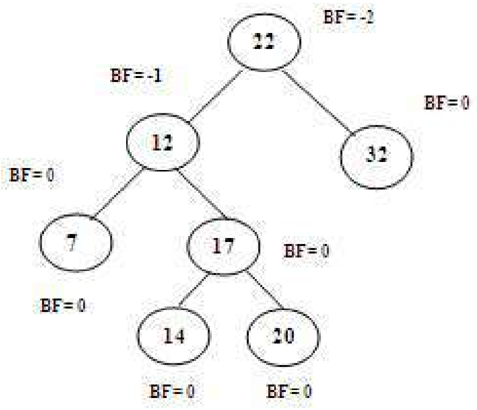
```
Static  Position  DoubleRotateWithLeft (Position K3)
{
        /* Rotate between K1 and K2 */
        K3 → Left = SingleRotateWithRight(K3→Left);
        /* Rotate between K3 and K1 */
        return SingleRotateWithLeft(K3);
}
```

Example:-

- Consider the following AVL tree in the figure 4.10.(a). If we insert **20** in the tree, then imbalance will occur at the node **22** as follows



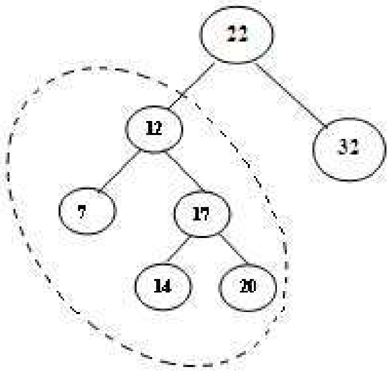
AVL Tree



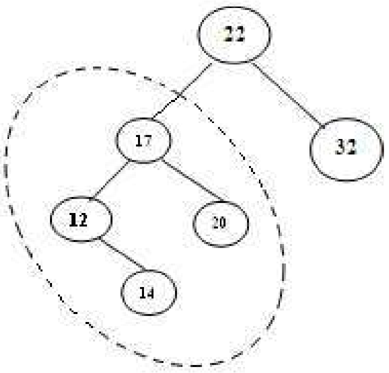
Imbalanced AVL Tree after inserting 20.

AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com

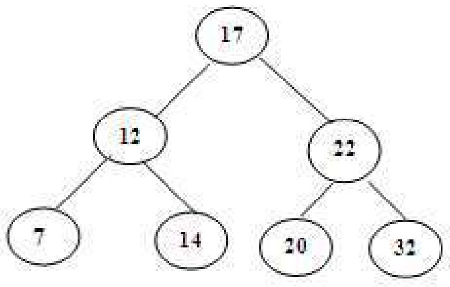


- After performing single rotation with right on the subtree we get the intermediate result as shown below,



After single rotation with Right

- Now perform single rotation with left on the above tree. The values are, **K1= 17, K3= 22, D= 32.** We get the balanced AVL Tree as in figure 4.10 (d)



After Left-Right Double Rotation

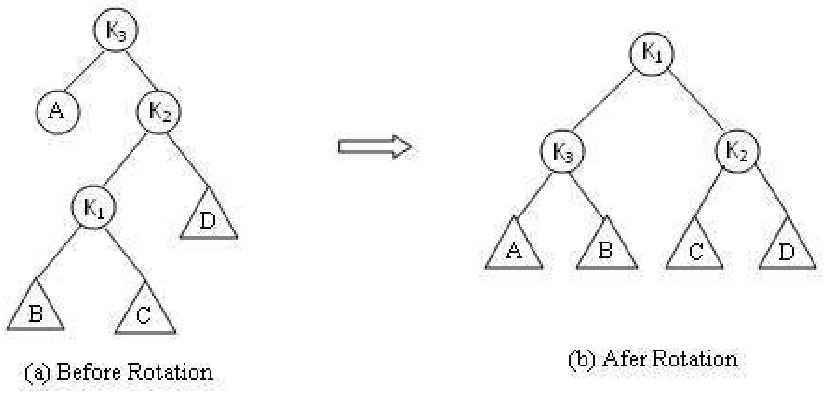
<u>RIGHT – LEFT ROTATION:</u>

- The Right-Left double rotation is performed, when an imbalance occur due to the insertion at the ***left subtree of the right child of an node.***

AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com

General Representation of Right- Left Double Rotation:



General Representation of Right- Left Double Rotation

- The above double rotation can be done by using the following two single rotation,

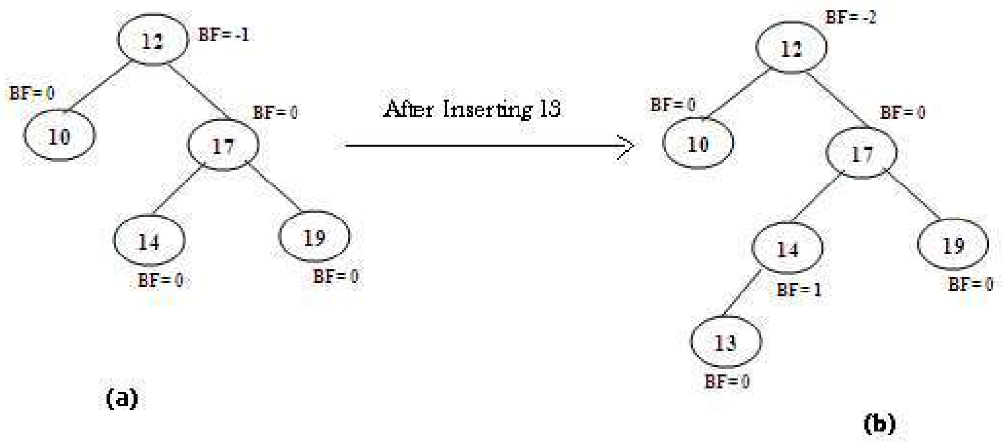
 1. Single rotation with left, then

 2. Single rotation with right.

Routine to perform Right-Left Double Rotation:

```
Static Position DoubleRotateWithRight (Position K3)
{
        /* Rotate between K1 and K2 */
K3 → Right = SingleRotateWithLeft(K3 →Right);
/* Rotate between K3 and K1 */
return SingleRotateWithRight(K3);
}
```

- For example, consider the following AVL tree in figure 4.12 (a), if we insert a node **13** in that AVL tree it will become imbalanced it node **12** as shown in figure 4.12 (b)

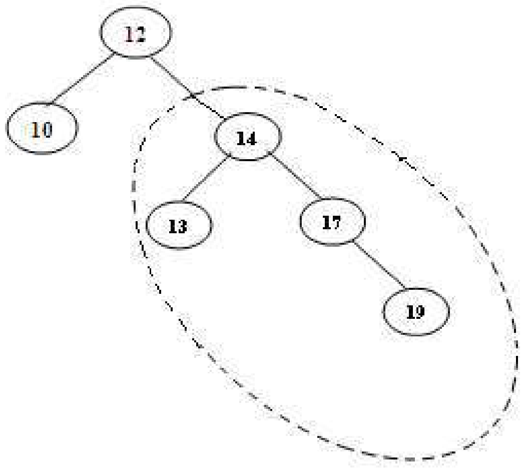


AVL tree **Imbalanced AVL Tree.**

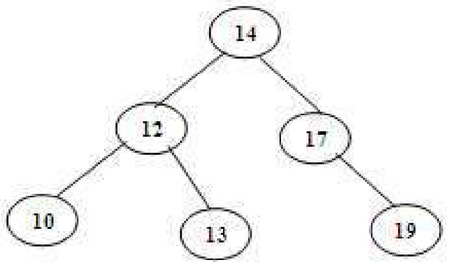
AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com

- To make it as a balanced tree, perform Right-Left double rotation.
- For that, first perform single rotation with left on the subtree (17, 14, 19, 13), in which the vaules of keys are, **K2= 17, K1= 14, B= 13, D=19,** and no value for **C.**
- After the single rotation with left, we obtain the tree as,



- The balanced AVL tree will be as follows



AVL tree after Right-Left double rotation.

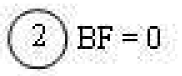
Q12) Show the results of inserting 2, 1, 4, 5, 9, 3, 6, 7into an initially empty AVL tree.

ANSWER:

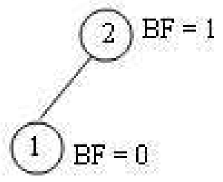
Example

Let us consider how to balance a tree while inserting the numbers 2, 1, 4, 5, 9, 3, 6, 7

1. Insert the value 2



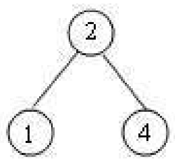
2. Insert 1



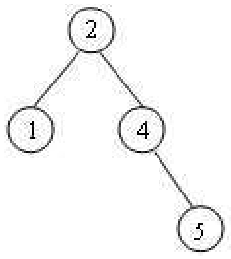
AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com

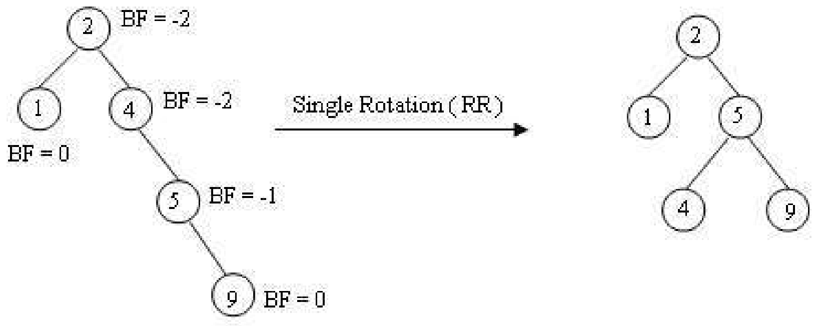
3. Insert 4



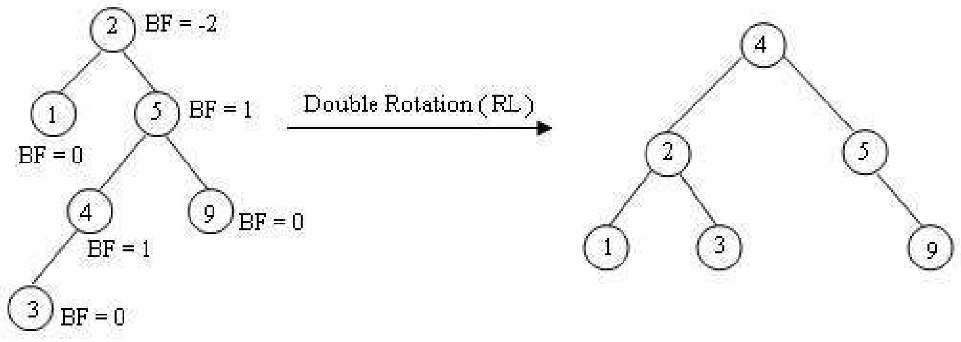
4. Insert 5



5. Insert 9



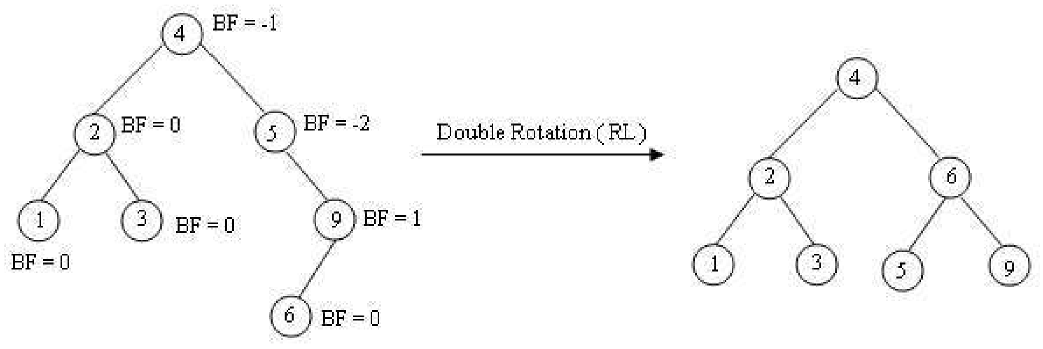
6. Insert 3



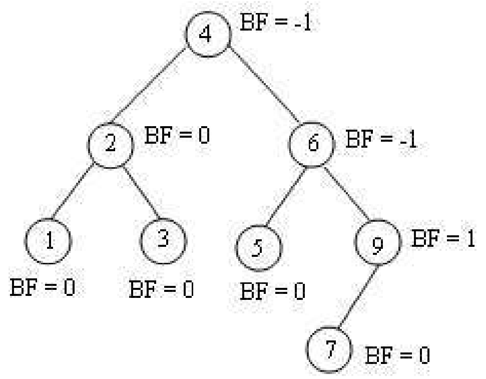
7. Insert 6

AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com



8. Insert 7



Q13)a) Discuss in detail the B +-tree. What are its advantages? (15)

Or

Q13) b) Explain the oprations which are done in B Tree with examples.(15)

ANSWER

B Tree

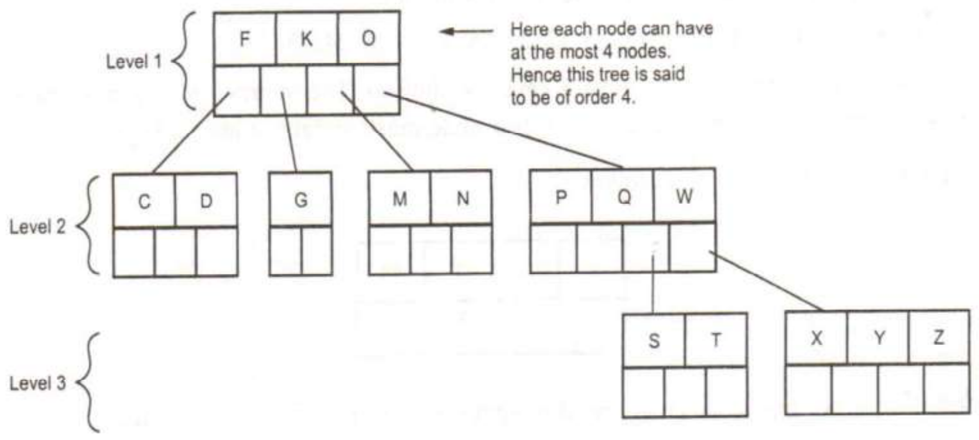
- B-tree is a specialized multiway tree used to store the records in a disk. There are number of subtrees to each node. So that the height of the tree is relatively small. So that only small number of nodes must be read from disk to retrieve an item. The goal of B-trees is to get fast access of the data.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

- Following is a tree of order 4.

AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com


Level 1
Level 2
Level 3
F K O
Here each node can have at the most 4 nodes. Hence this tree is said to be of order 4.
C D
G
M N
P Q W
S T
X Y Z

- From above tree following observations can be made -

1. The node which has n children posses (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 then 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 0 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, 0 are called keys and branches are called children.

Properties of B+ Tree

- All the leaf nodes are on the botom level.
- The root node should have at least two children.
- All the internal nodes except root node have at least ceil (m/2) nonempty children. The ceil is a function such that ceil (3.4) = 4, ceil (9.3) = 10, ceil (2.98) = 3 ceil (7) = 7.
- Each leaf node must contain at least ceil (m/2) — 1 keys.

Thus the B-tree is fairly balanced tree

<u>Insertion Example of B-tree</u>

Construct a B-tree of order 5 by following numbers.

- 3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20, 26, 4, 16, 18, 24, 25, 19.

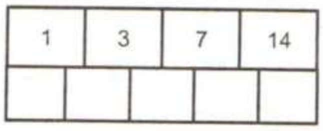
Solution:

AllAbtEngg Android Application for Anna University, Polytechnic & School

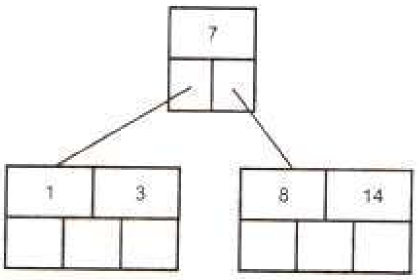
www.AllAbtEngg.com

- The order 5 means at the most 4 keys are allowed.
- The internal node should have at least 3 non empty children and each leaf node must contain at least 2 keys.

Step 1 : Almost 4 keys are allowed.So Insert 3, 14, 7, 1 as follows

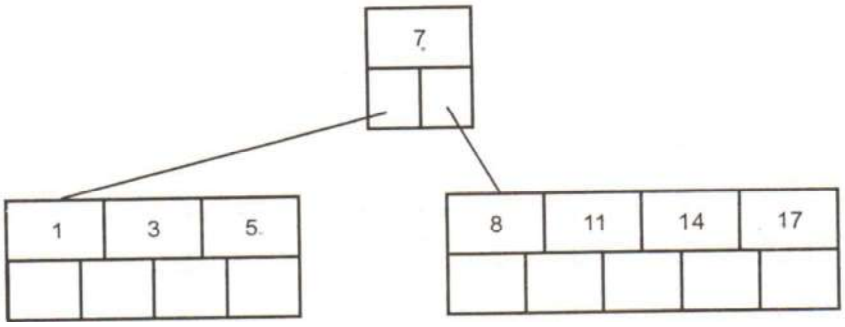


Step 2 : If we insert 8 then we need to split the node 1, 3, 7, 8, 4 at medium. Hence,



Here, 1 and 3 are smaller than 7.So they are in left and then 8 and 14 are greater than 7.So they are in right.

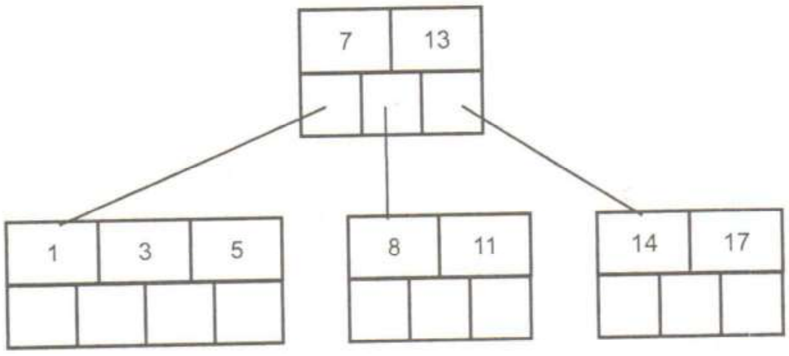
Step 3 : Insert 5, 11, 17 which can be easily inserted in a B-tree.Hence,



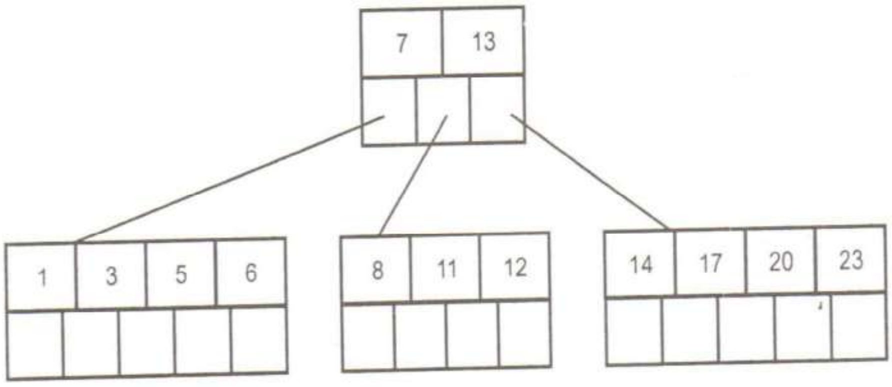
Step 4 : Now insert 13. But if we insert 13 then the leaf node will have 5 keys which is not allowed. Hence 8, 11, 13 , 14, 17 is split and medium node 13 is moved up.Hence,

AllAbtEngg Android Application for Anna University, Polytechnic & School

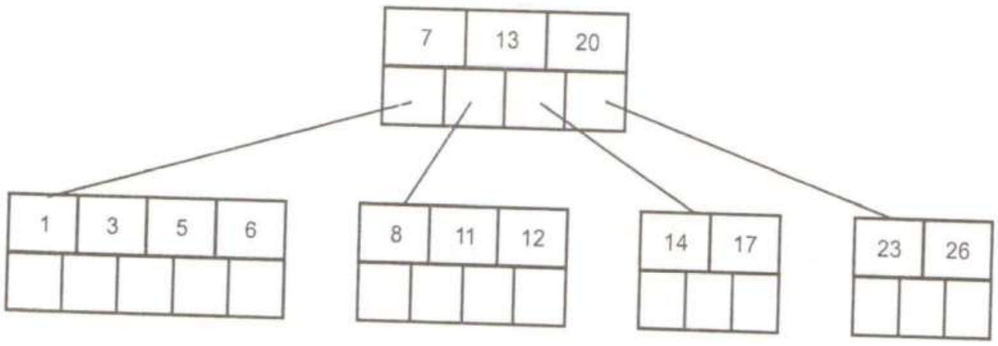
www.AllAbtEngg.com



Step 5 : Now insert 6, 23, 12, 20 without any split as folows



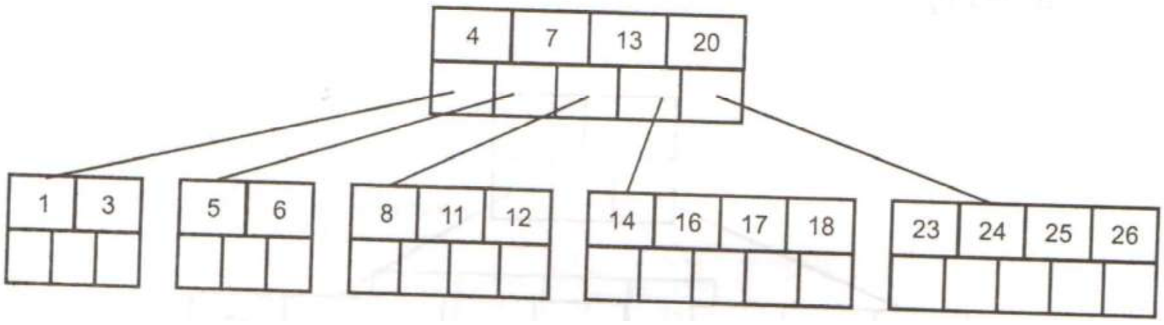
Step 6: Now insert 26.If we insert 26 in rightmost leaf node then it will have 5 keys which is not allowed.Hence 14, 17, 20, 23, 26 the node is split and 20 will be moved up.



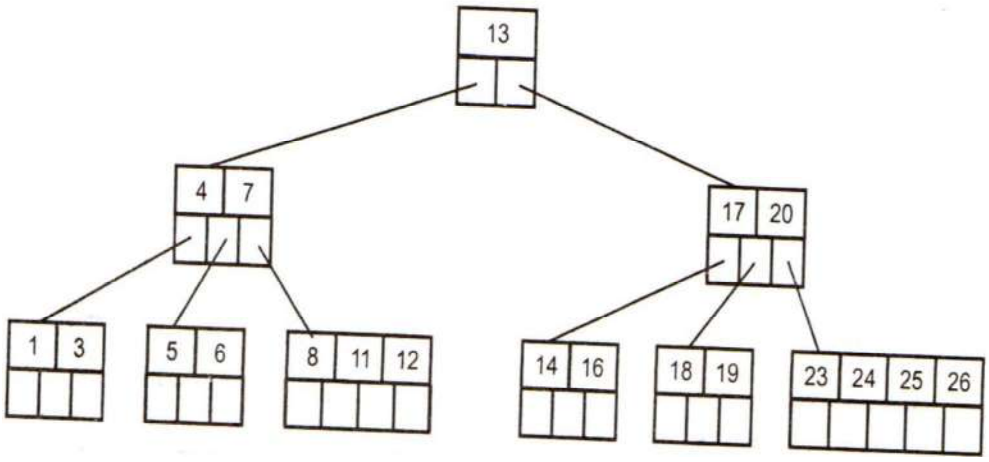
Step 7: Now insert 4.If we insert 4 in leftmost leaf node ,then it will have five keys which is not allowed.Hence, 1, 3, 4 , 5, 6 is split and medium node 4 is moved up and then insert 16, 18, 24, 25 without any split.

AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com



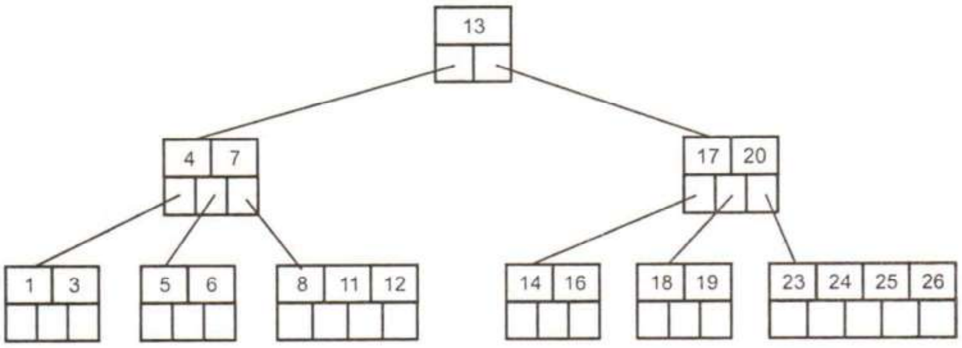
Step 8: Finally insert 19. If we insert 19 after 18 means that node contains 5 keyswhich is not allowed.Hence 14,16,17,18,19 is split and medium node 17 is moved up and the root node contains 5 keys which is not allowed.So split the root node 4,7,13,17,20 at medium and 13 is moved up as follows.



- Thus the B-tree of order 5 is constructed.

<u>**DELETION**</u>

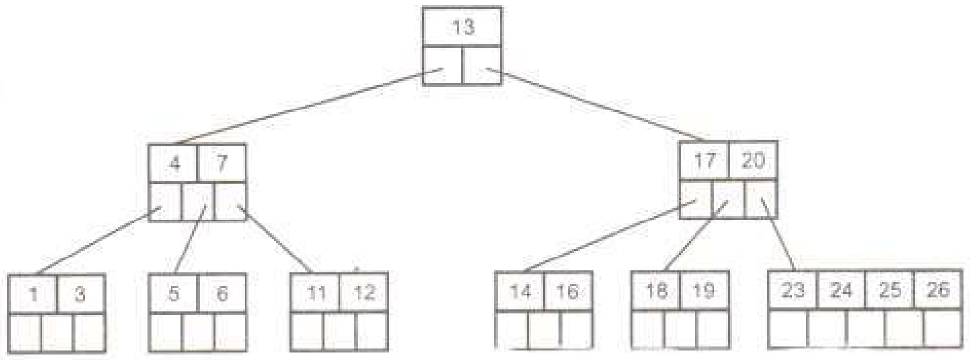
- Consider a below B-tree and delete 8,20,18,5



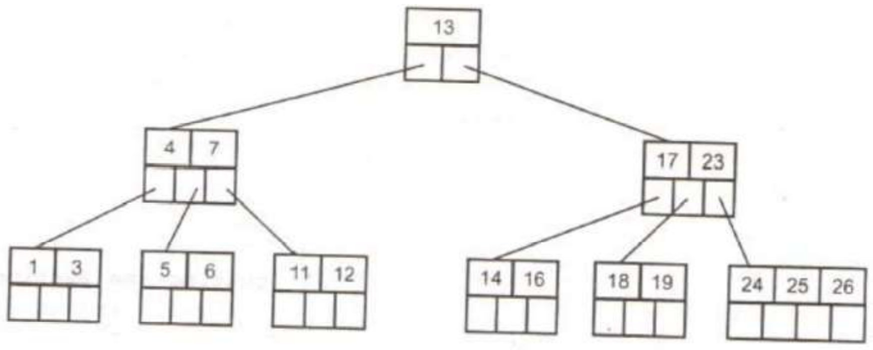
AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com

Step 1:To delete 8, simply delete it from leaf node.Hence,



Step2 : Now we will delete 20, the 20 is not in a leaf node so we will find successor of 20 which is 23. Hence 23 will be moved up to replace 20.Otherwise that node contain only one key 17.



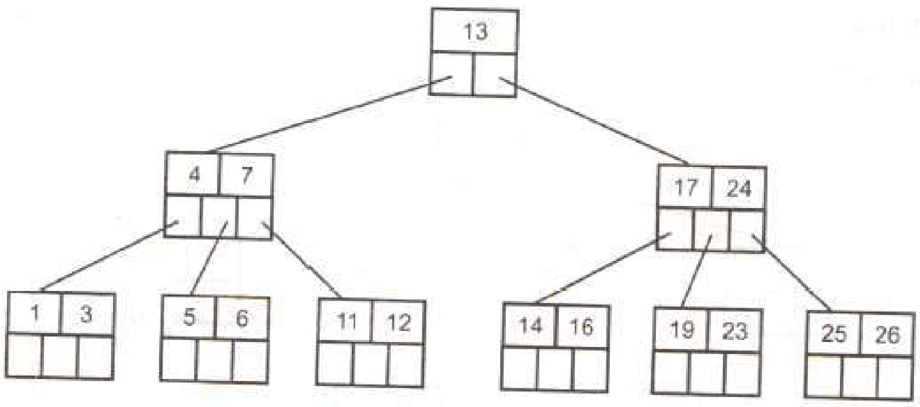
Step 3:Next we will delete 18. Deletion of 18 from the corresponding node causes the node with only one key, which is not allowed (as per property 4) in B-tree of order 5.

So, The sibling node to immediate right has an extra key(it contain 3 keys). In such a case we can borrow a key from parent and move spare key of sibling node to moved up (ie) move 24 up. See the following figure.

.

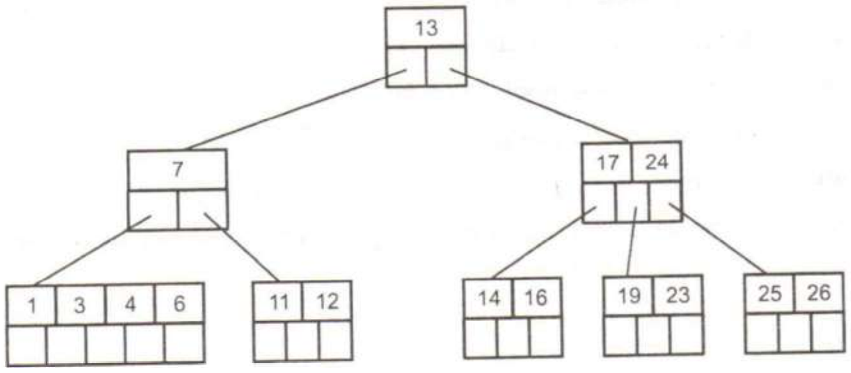
AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com

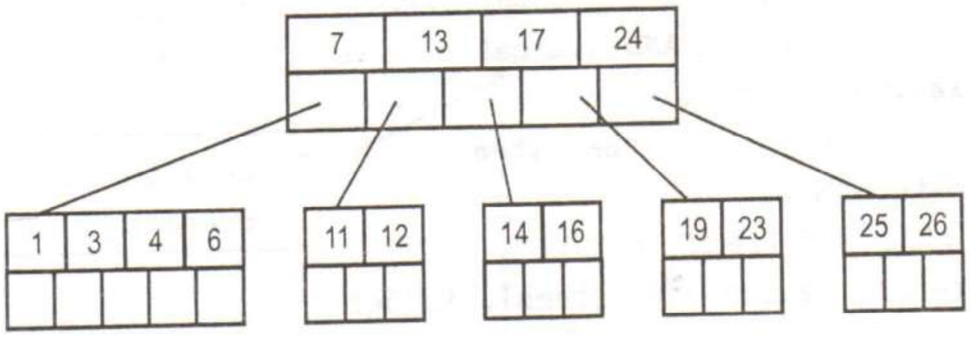


Step 4: Now delete 5. But deletion of 5 is not easy. The first thing is 5 is from leaf node. Secondly this leaf node has no extra keys nor siblings to immediate left or right. In such a situation we can combine this node with one of the siblings

- That means remove 5 and combine 6 with the node 1, 3. To make the tree balanced we have to move parent's key down. Hence we will move 4 down as 4 is between 1, 3 and 6. Then the tree will be



- But again internal node of 7 contains only one key which not allowed in B-tree (as per property 3). We then will try to borrow a key from sibling. But sibling 17, 24 has no spare key. Hence what we can do is that, combine 7 with 13 and 17, 24. Hence the B-tree will be



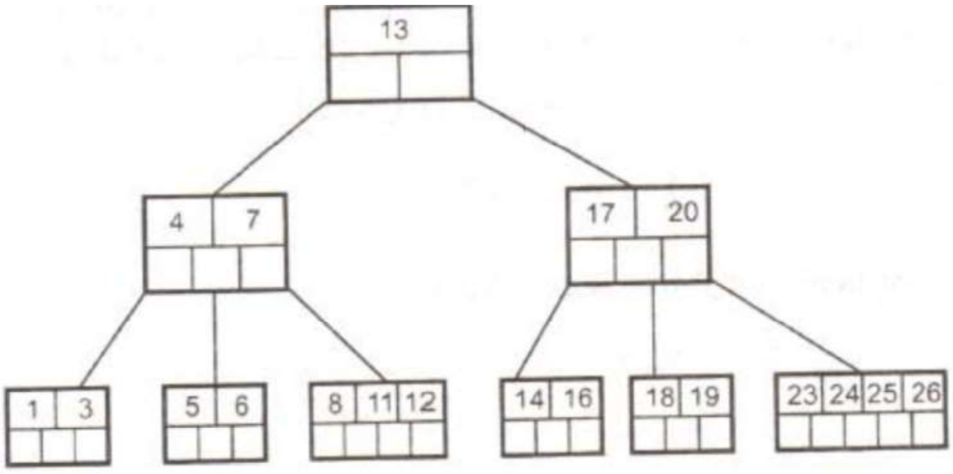
AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com

SEARCHING

➢ 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.

Example: Search 11 from a below B-tree



Solution: To search 11,
Step 1: Compare 11 and 13, 11 < 13 So go to left side nodes.
Step 2: Compare 11 and 7, 11 > 7 So go to right side nodes.
Step 3: Compare 11 nad 8, 11 > 8 So go to the second key.
Step 4: Second key 11.(ie) Found search element.

Q14)a) Discuss in detail the B +-tree. What are its advantages? (15)

Or

Q14) b) Explain the oprations which are done in B+ Tree with examples.(15)

ANSWER

B-Trees

B tree is a popular search tree that is not binary.

A B-tree of order m is a tree with the following structural properties:

- The root is either a leaf or has between 2 and m children.
- All or internal nodes or non leaf nodes (except the root) have between m/2 and m children.
- The number of keys(elements) in each internal node is one less than its number of child nodes.
- All leaves are at the same depth.

All data is stored at the leaves. Contained in each interior node are pointers

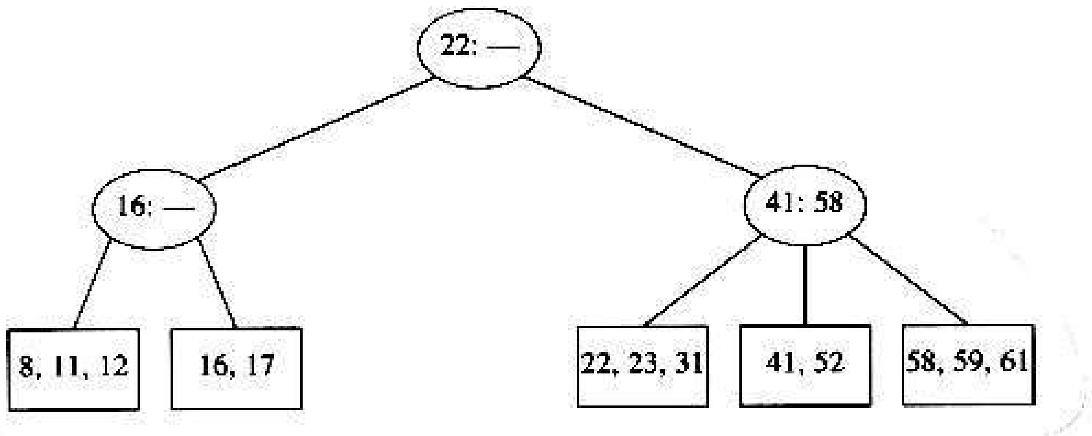
AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com

p1, p2, . . . , pm to the children, and values k1, k2, . . . , km - 1,representing the smallest key found in the subtrees p2, p3, . . . , pm respectively.

For every node, all the keys in subtree p1 are smaller than the keys in subtree p2, and so on. The leaves contain all the actual data, which is either the keys themselves or pointers to records containing the keys.

B Tree representation



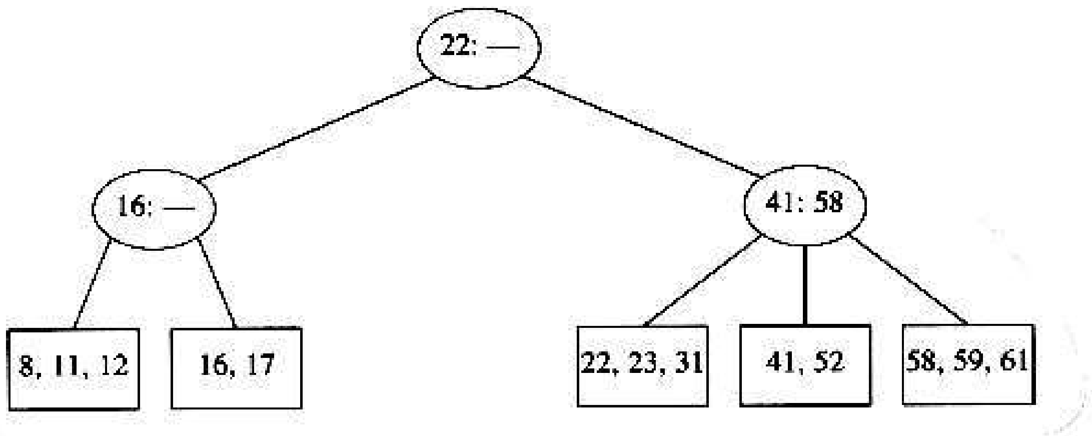
The root node and non leaf node is represented as ellipses which contains two pieces of data for each node.First part represents the key value, which can be the largest element of the left sibling or the smallest element of the right sibling and dash line indicates that the node has only two children.

Leaves are drawn in boxes, which contain the keys.The keys in the leaves are ordered.

A B-tree of order 4 is more popularly known as a 2-3-4 tree, and a B-tree of order 3 is known as a 2-3 tree.

INSERTION

In the following B Tree of order 3, insert values 18,1,19,28 and 28,19,1,18.

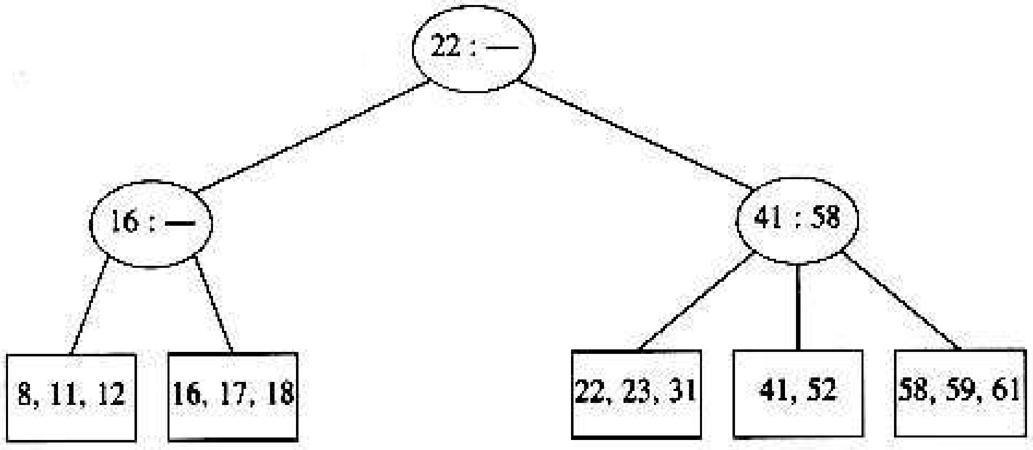


Step 1: Insert 18

AllAbtEngg Android Application for Anna University, Polytechnic & School

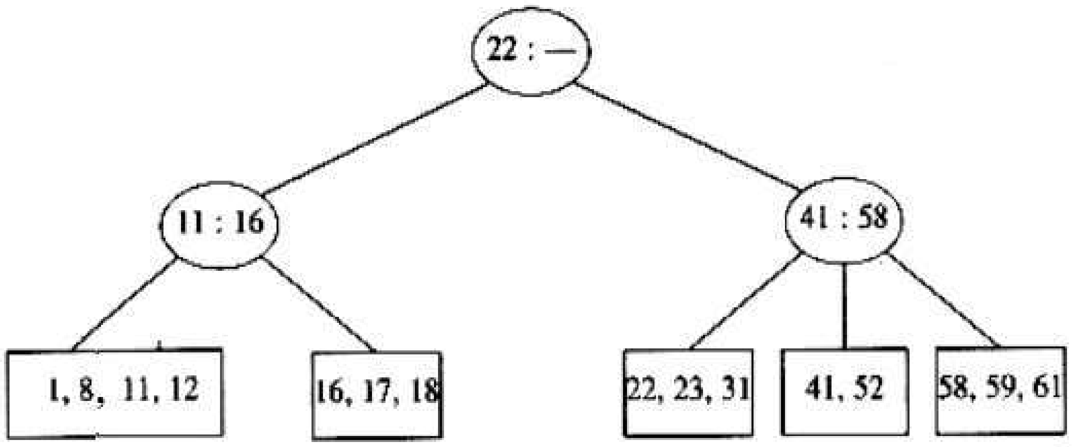
www.AllAbtEngg.com

Compare 18 with 22.18< 22 ao move to left then 18> 16. So move to right and insert 18 without causing any violations of the 2-3 tree.



Step 2: Insert 1

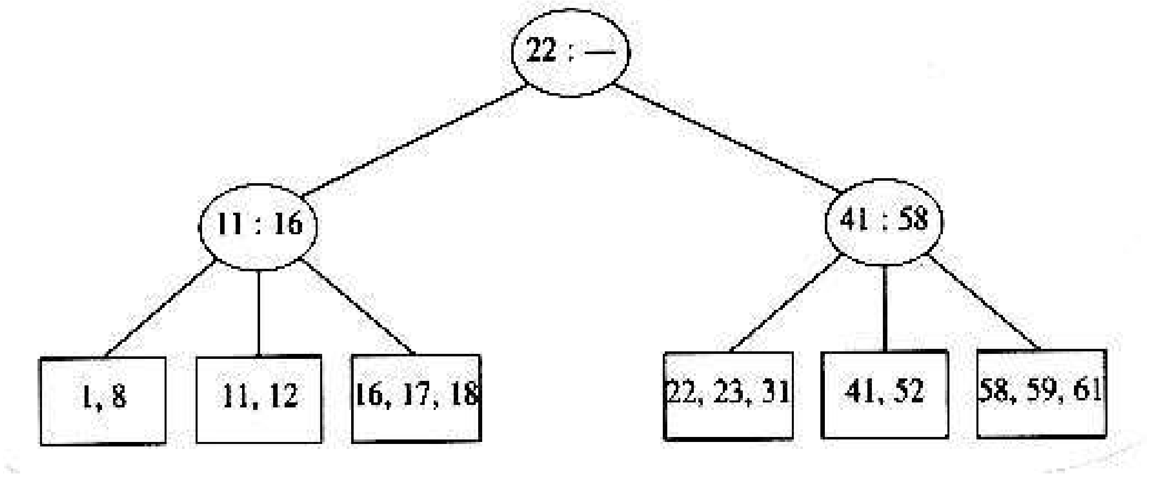
Compare 1 with 22. 1 < 22 So move 1 to left .Then 1 < 16. So move 1 to right.



But leaf node can contain only 3 keys .So insertion of 1 violates the property. So insert 1 by splitting the node as follows

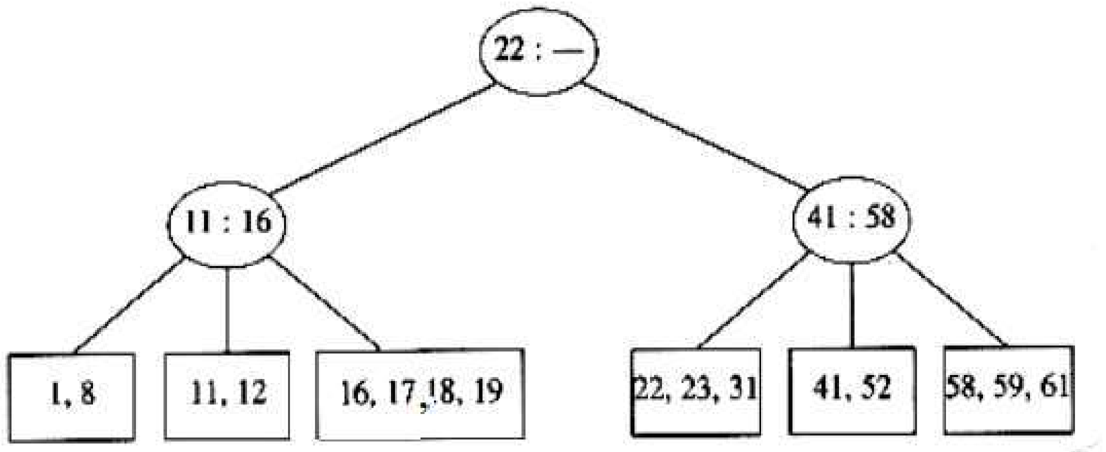
AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com



Step 3: Insert 19

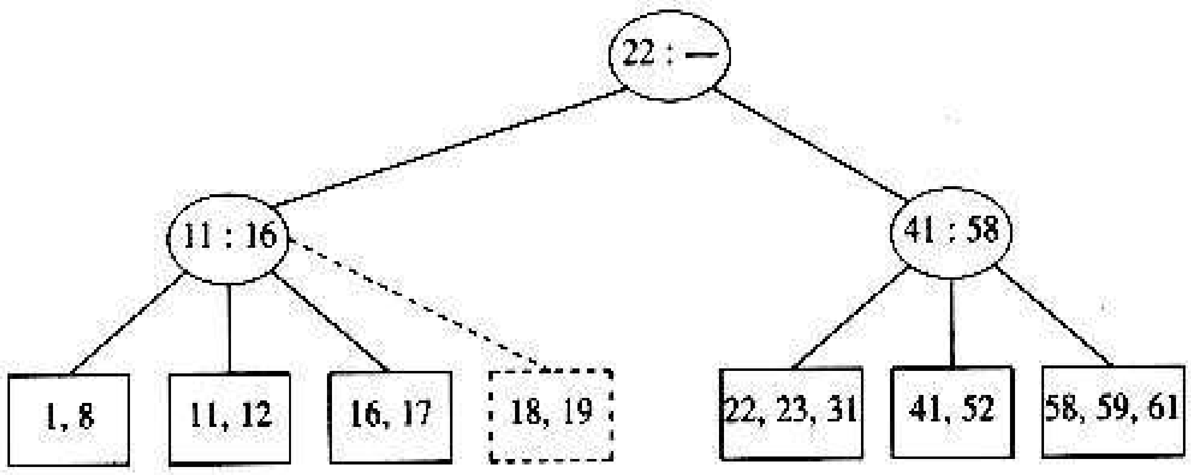
Compare 19 and 22. 19 < 22 So move to left .Then 19 > 16. So move to right and insert 19.



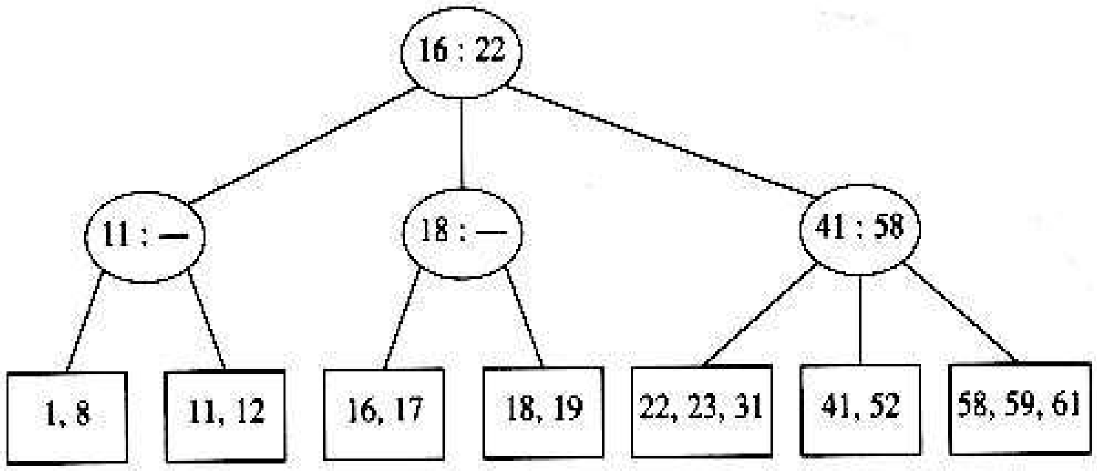
But a B Tree of order 3 can contain only 3 keys in leaf.So property violated.So insert 19 by splitting a node as follows

AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com



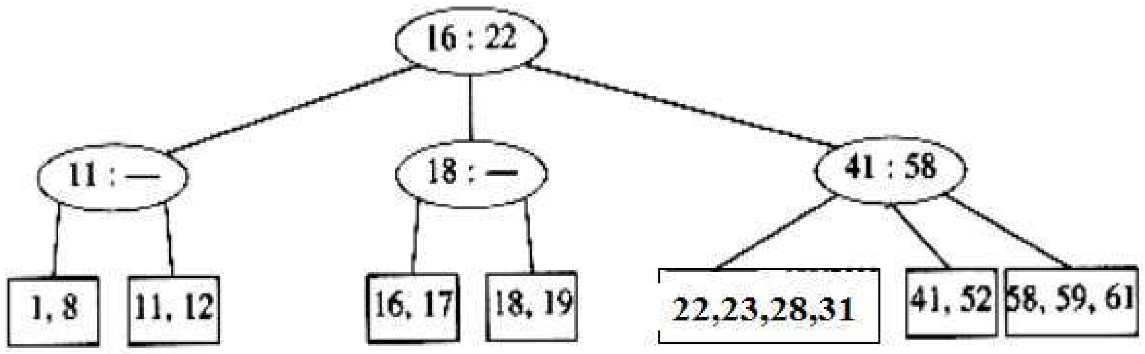
But it also violates properties of 2-3 tree because internal node can have only 3 children but here internal node contain 4 children which is not allowed.
So we can split this node into two node with two children as follows,



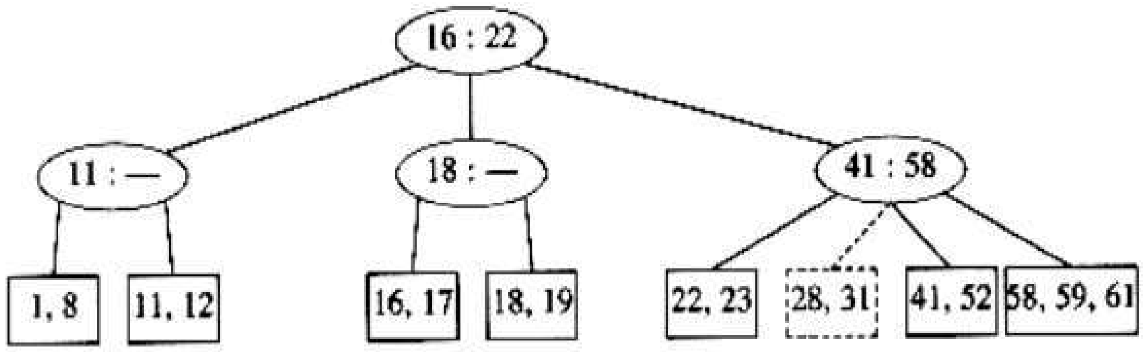
Step 4: Insert 28
Compare 22 & 28.28 > 22So move to right then 28 < 41. So move to left and insert 28.

AllAbtEngg Android Application for Anna University, Polytechnic & School

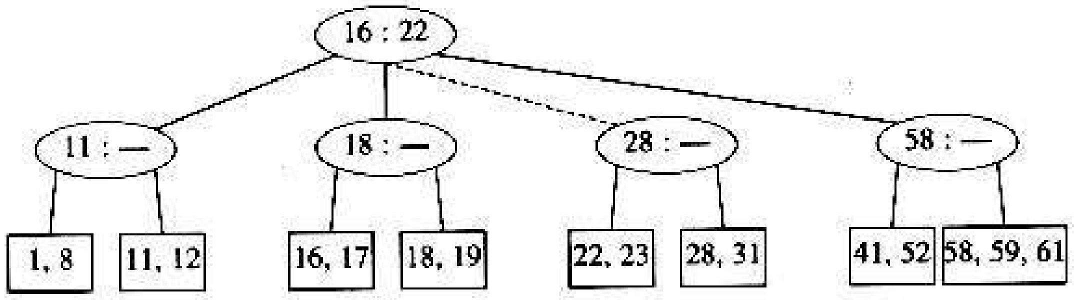
www.AllAbtEngg.com



But it also violates properties of 2-3 tree because leaf node contain 4 key which is not allowed.So we can split that node into two nodes as follows



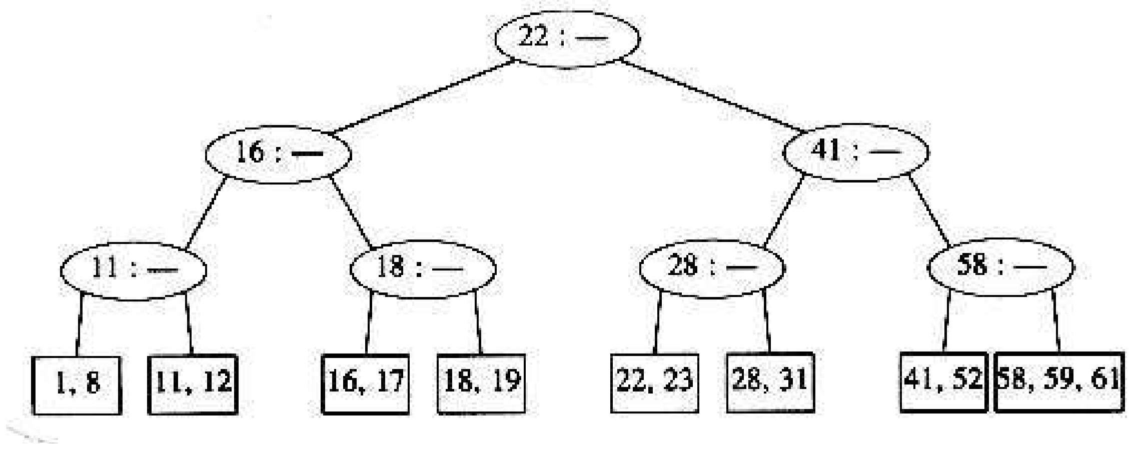
But it violates the properties of 2-3 tree because internal node contain 4 children which is not allowed.So split internal root node into two as follows.



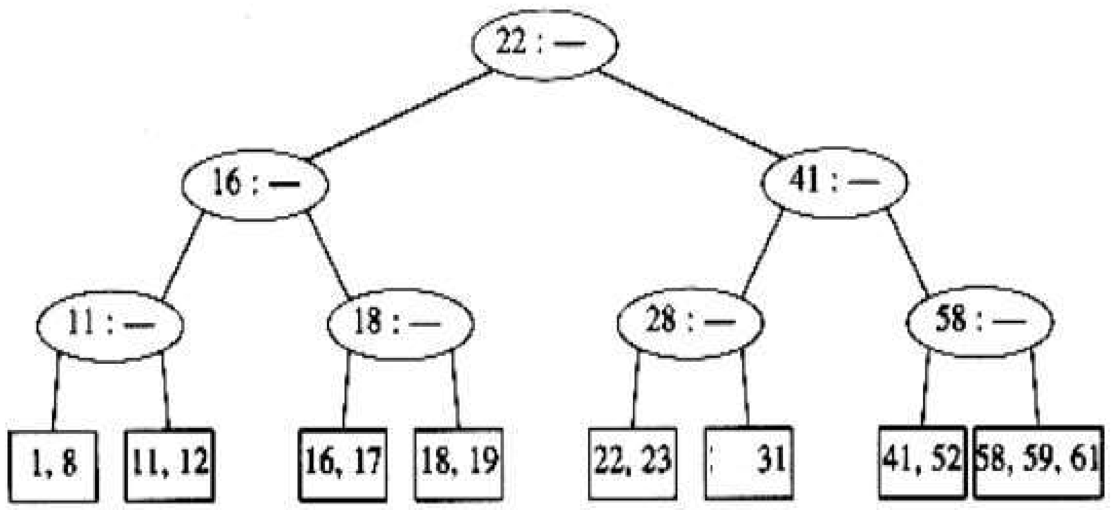
But it also violates the property of 2-3 tree because root node have 4 children which is not allowed. So split this root node into two and create new root node.

AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com



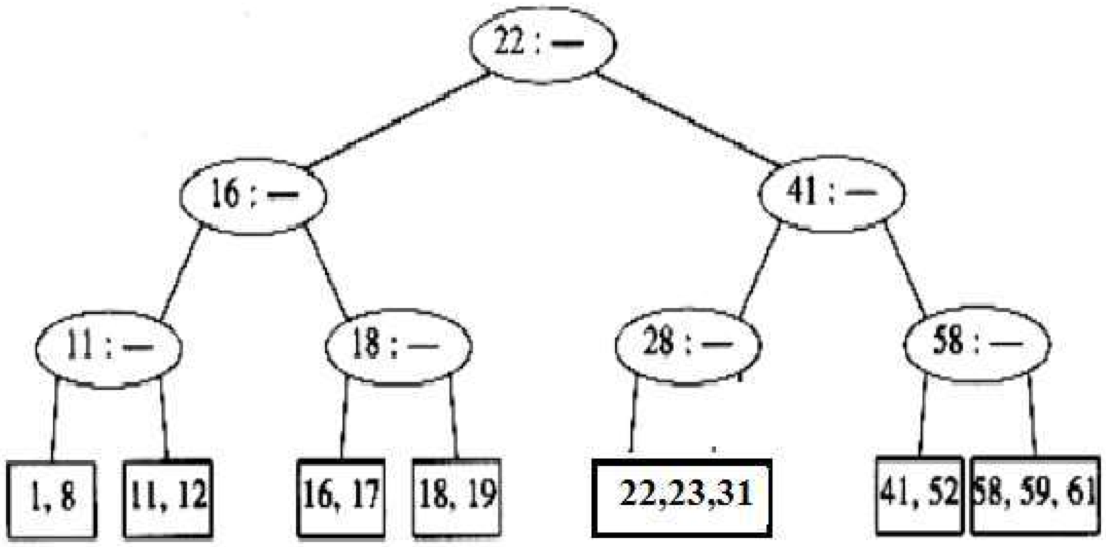
Deletion: Delete 28,19,1,18
Step 1: Delete 28.
We just delete 28 as follows,



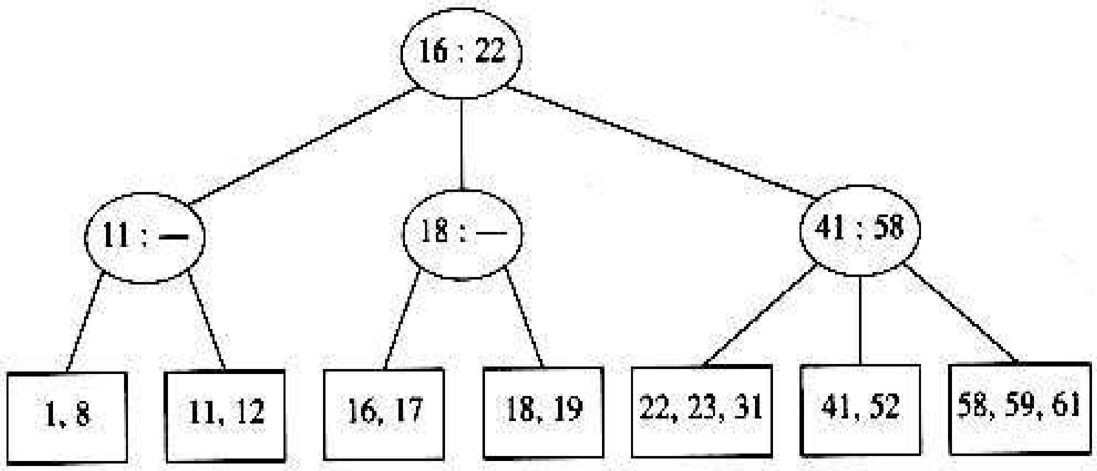
Deletion of 28 creates wastage of memory ,so we can merge two nodes as follows,

AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com



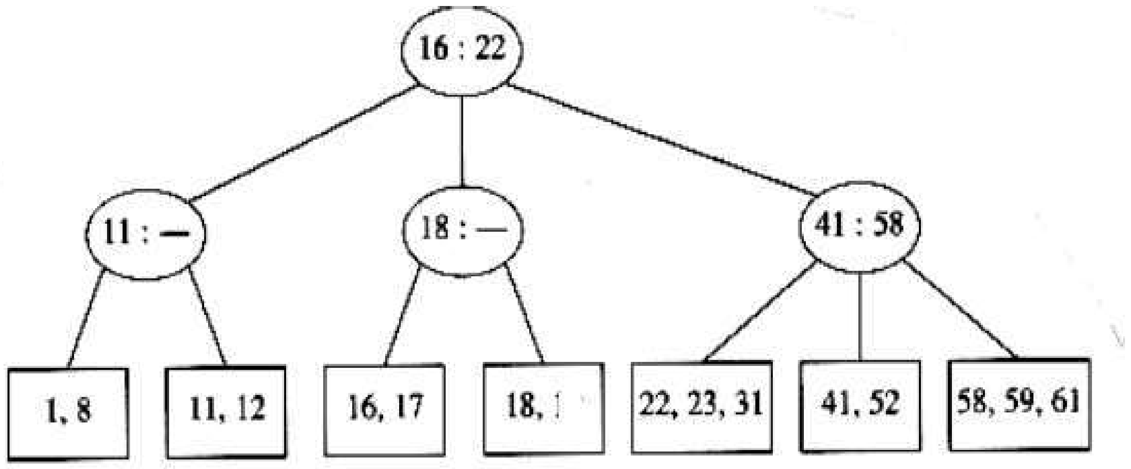
Deletion of 28 creates wastage of memory ,so we can avoid it by merging internal nodes,



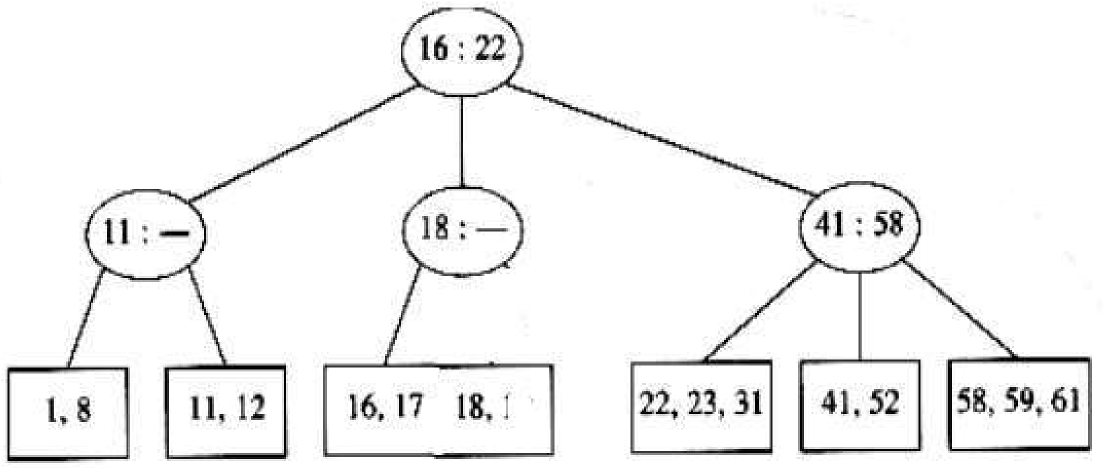
Step 2: Delete 19
We can delete it as follows

AllAbtEngg Android Application for Anna University, Polytechnic & School

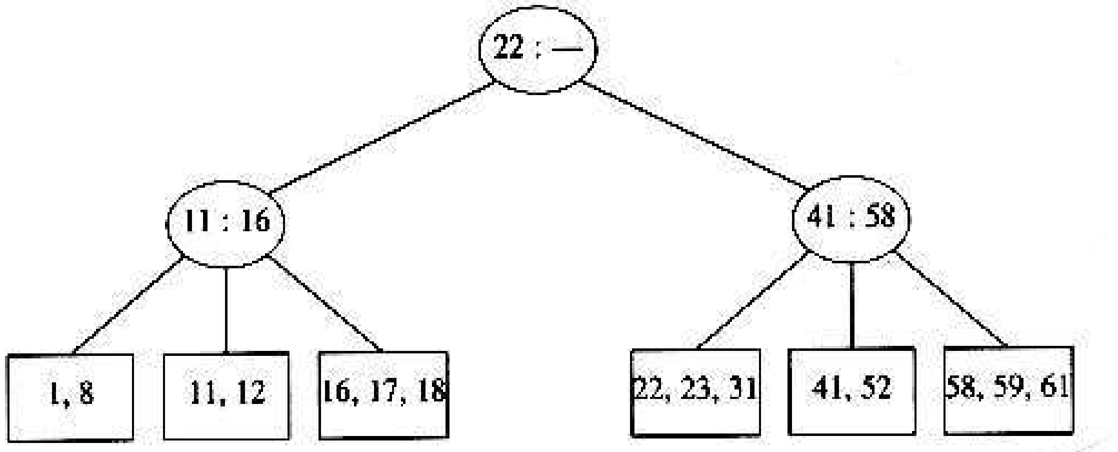
www.AllAbtEngg.com



It creates wastage of memory , so we can merge two nodes as follows



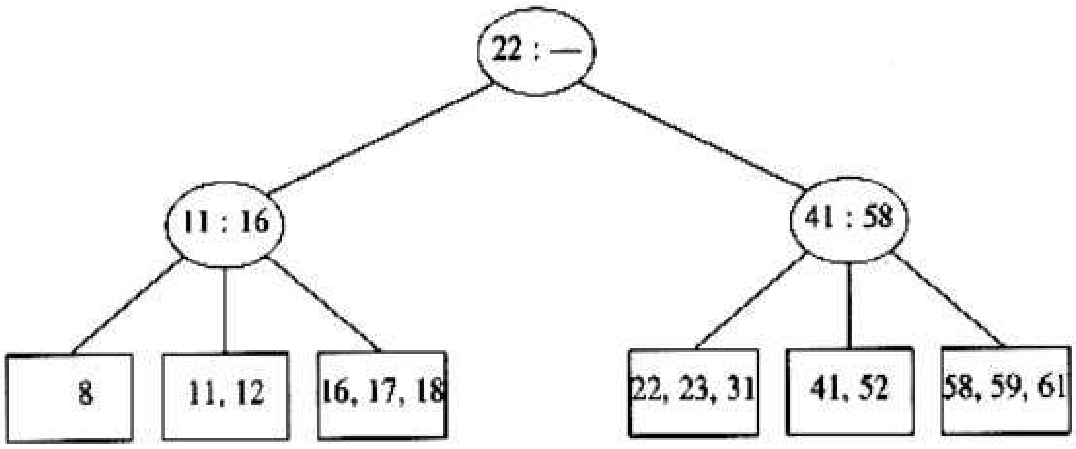
It creates wastage of memory , so we can avoid it by merging internal node as follows,



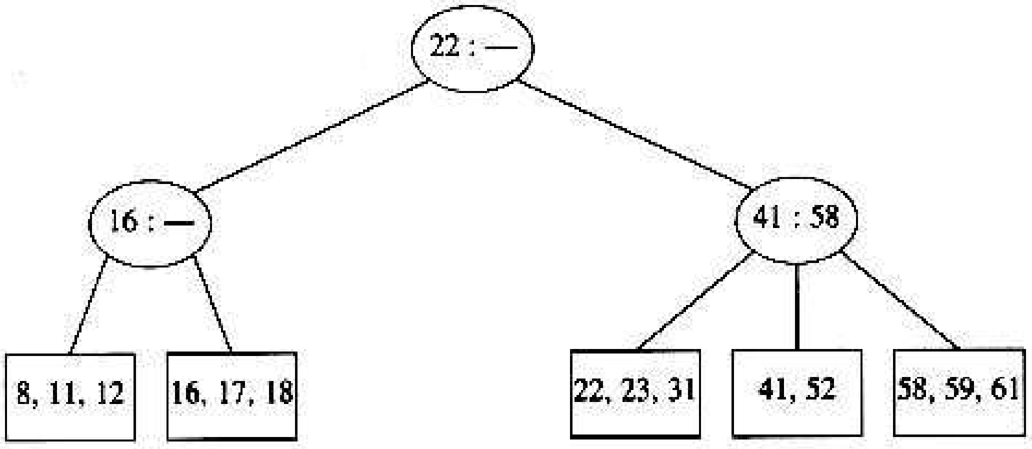
Step 3:Delete 1, we can delete it as follows

AllAbtEngg Android Application for Anna University, Polytechnic & School

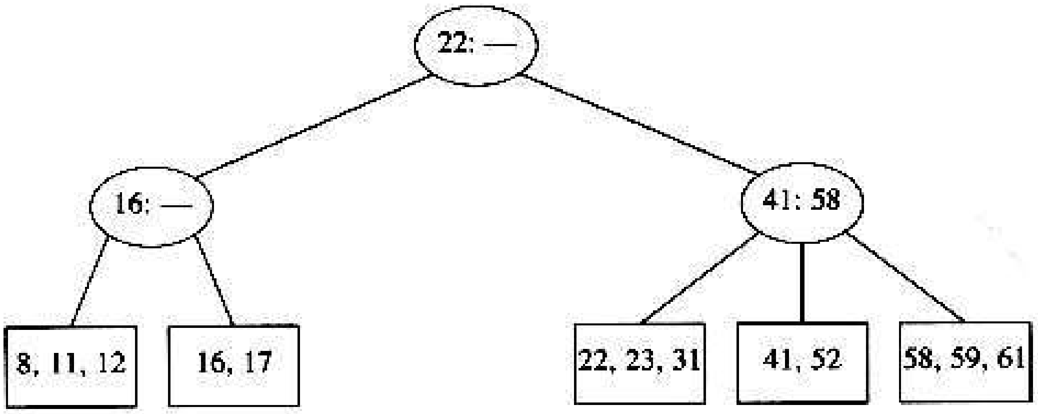
www.AllAbtEngg.com



But it creates wastage of memory,so we can merge two nodes as follows,



Step 4: Delete 18
We can delete it as follows



AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com

Q15) a) Explain binary heaps in detail. Give its merits.(13)

Or

Q15) b) What is structure property and heap order property?Explain the operations which can be done in heap with examples.

ANSWER

BINARY HEAPS

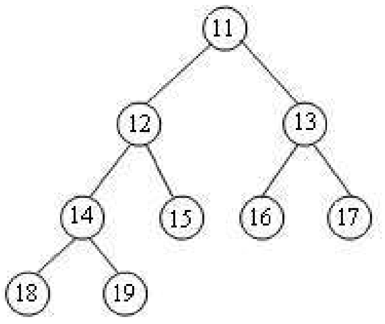
- The efficient way of implenting priority queue is Binary heap. Binary heap is merely referred as Heaps. Heap have two propertie namely,
  1) ***Structure Property***
  2) ***Heap Order Property***
- Like AVL Trees,an operation on a heap like AVL trees,an operation on a heap can destory one of the properties,so a heap operation must not terminate until alll heap properties are in order.Both the operations require the average running time as O(log N)

1) Structure Property

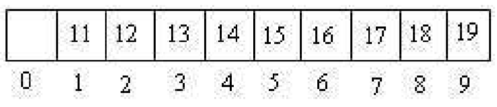
- A heap should be complete binary tree ,which is a completely filled binary tree with the possible exception of the bottom level,which is filled from left to right.
- A complete binary tree of height H has between 2H and 2H+1 -1 nodes.This implies that the height of a complete binary tree is [log n] (ie) O(log n)

Eg:

If the height is #.Then the no of nodes will be between 8 and 15.(ie 23 and 23 – 1)



- We can represent a tree in an array because of complete binary tree and no pointers are necessary.



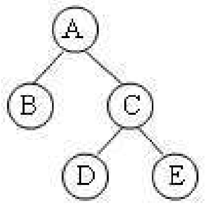
	11	12	13	14	15	16	17	18	19
0	1	2	3	4	5	6	7	8	9

- For any elemnt in array position i, the left child is in position 2i,the right child is in position 2i+i, and the parent is in i/2.

AllAbtEngg Android Application for Anna University, Polytechnic & School

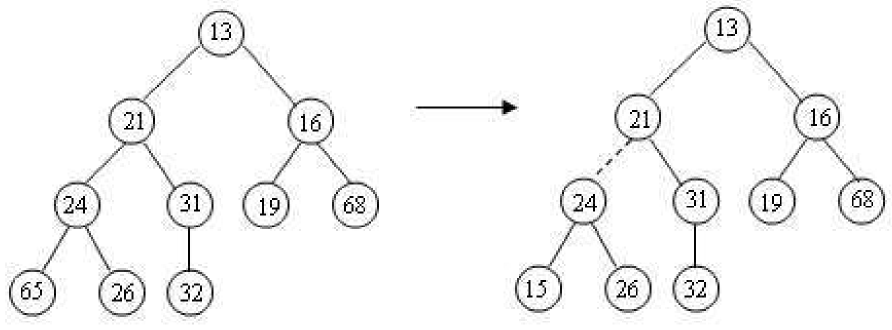
www.AllAbtEngg.com

- As it is represented as array it doesn't require pointers and also the opreation s required to traverse the tree are extremely simple and fast.But the only disadvantagge is to specify the maximum heap size in advance.



2) Heap Order Property

- The property that allows the operation to be performrd quickly is the heap order property. To find the minimum element quickly it makes sense that the smallest element should be at the root.
- In a heap,for every node X,the key in the parent of X is smallest than or equal to the key in X with the exception of the root.(Which has no parent). Thus,we get the findmin opeartion in constant time.
- In below figure,the tree on thhe left is a heap but the tree on the right is not.



Declaration

```
Struct Heapstruct
{
        int capacity;
        int size;
        int * elements;
};
```

Initialization

```
Priority Queue Initialize( int MaxElements)
{
        Priority Queue H;
        H = malloc ( size of ( Struct Heapstruct))
        H → Capacity = MaxElements;
        H → size =0;
        H → elements[0] = MinData;
```

AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com

```
        Return H;
}
```

3) Basic Heap Operations

To perform the insert and Deletemin opeartions ensure, that the heap order property is maintained.

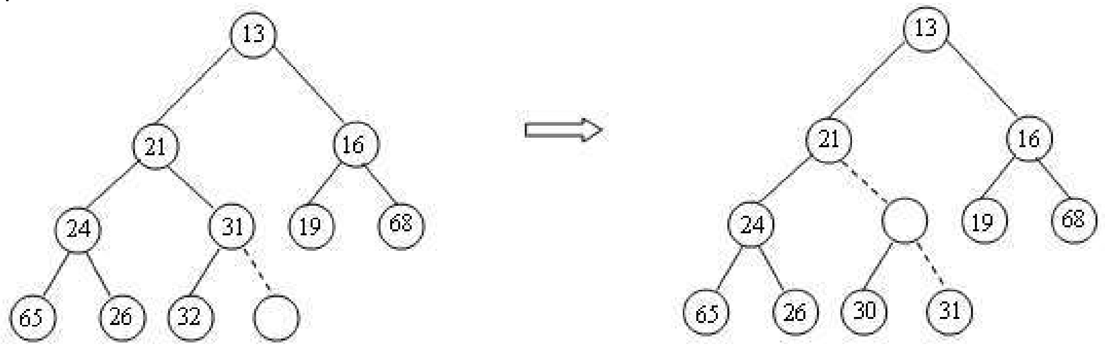
(i) Insert

To insert an element X into the heap. Create the hole in the next available location.If X can be placed in the hole without violating heap order, then place the element X there itself. Otherwise, we slide the element that is in the hole's parent node into the hole, thus bubbling the hole up toward the root. This process continues until X can be placed in the hole. This general strategy is known as percolate up. In which the new element is percolated up the heap until the correct location is found.

Routine to perform INSERT operation

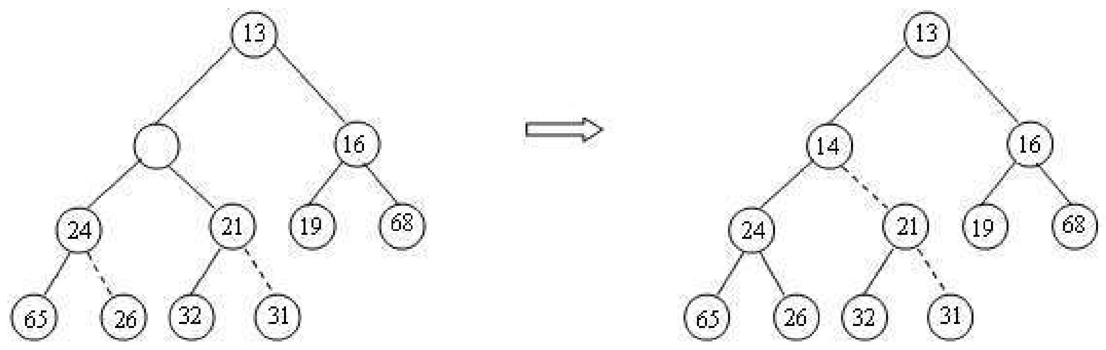
```
Void insert ( int X, Priority Queue H)
        {
        int I;
        if( IsFull(H))
        {
           Error ( "Priority Queue is Full");
           Return;
        }
        for(i=++ H→ size; H→Elements[i/2]>X;i/=2)
            H → Elements [i] = H → Elements [i/2]
            H → Elements [i] = X;
        }
```

Example:



AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com

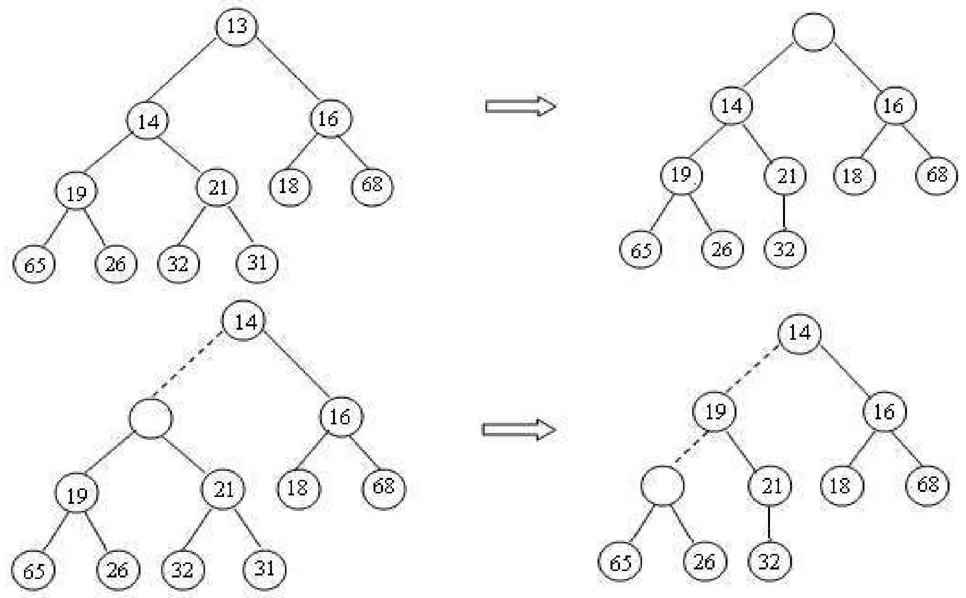


ii) DeleteMin

DeleteMin operation is deleting the minimum element from the heap. In binary heap the minimum element is found in the root. When this minimum is removed a hole is created at the root. Since the heap becomes one smaller makes the last element X in the heap is to move somewhere in the heap.

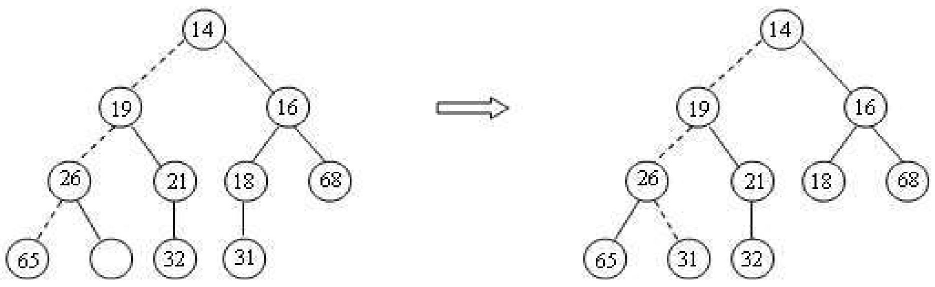
If X can be placed in hole without violating heap order property place it. Otherwise we slide the smaller of the hole's children into the hole, thus pushing the hole down one level.

We repeat until X can be placed in the hole. This general strategy is known as percolate down.



AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com



Routine to perform DELETEMIN

```
    int Deletemin ( Priority Queue H)
    {
    int I, child;
    int MinElement, LastElement;
    if(IsEmpty(H))
    {
    Error("Priority Queue is Empty ");
    Return H → Elements [0];
    }
    MinElement = H → Elements[i];
    LastElement = H → Elements [ H→ Size --];
    for(i=1; i*2<=H→size;i= child)
    {
    child=i*2;
    if(child!= H→size && H→Elements[child +1]< H→Elements [child])
    child++;
    if(LastElement > H→ Elements [child])
    H→ Elements [i] = H→ Elements [child];
    else
    break;
    }
    H→. Elements [i]= LastElement;
    Return MinElement;
}
```

OTHER HEAP OPERATIONS

The other heap operations are

1) Decrease key
2) Increase Key
3) Delete
4) Bulid Heap

Decrease Key

AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com

The decrease key(p, ,H). Operation decreases the value of the key at position p by a positive amount .This may violates the heap order property which can be fixed by percolate up.

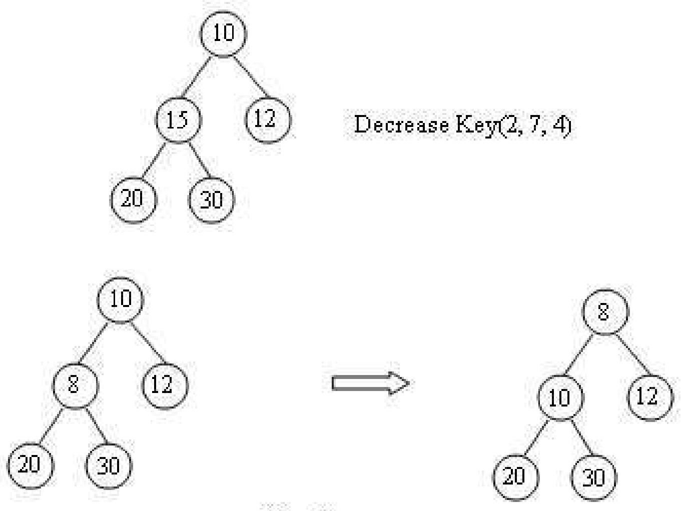


Fig. A

Element at position 2 is 15.decrease that element by 7.Now the position 2 has the value 8.which violates the heap order property .

This can be fixed by percolating up strategy.(Fig A)

Increase key

The increase key (p, , H), operation increases the value of the key at position p by a positive amount. This may violate order property.

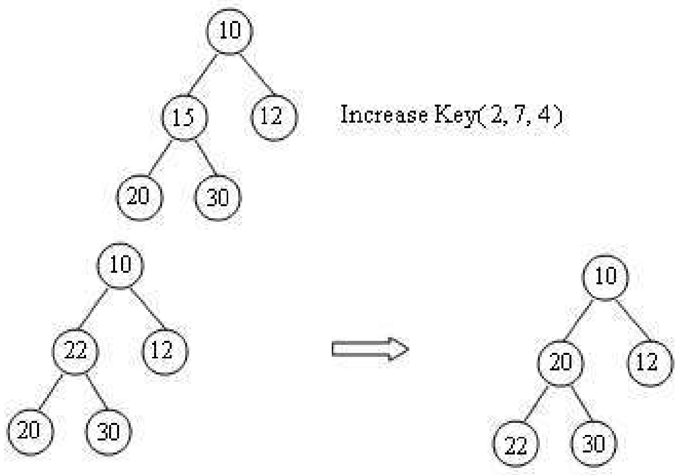


Fig. B

Here the element at position 2 is 15.Increase that value by 7.Now the position 2 has the value 22,which violates the heap order property.This can be fixed by percolate down.

AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com

Delete:

The delete(p,H) operation removes the node at the position p from the heap. This can be done by,

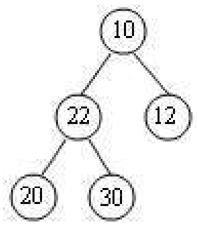
(i) Perform the decrease key operation

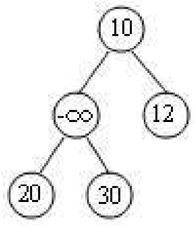
Decrease key(p, ,H)

(ii) Perform DeleteMin operation

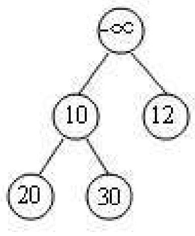
DeleteMin(H)

(i) Decreasing by Infinity





After decreasing the value at position .The value changes to which is the least element in heap.



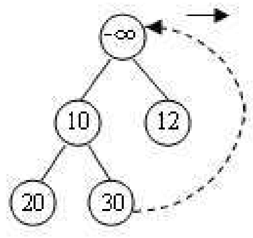
Since occupies the root position,apply DeleteMin opration.

(ii) DeleteMin

After deleting the minimumelement ,the last element will occupy the hole.Then rearrange the heap till it satisfies heap order property.

AllAbtEngg Android Application for Anna University, Polytechnic & School

www.AllAbtEngg.com



Bulid Heap

The BuildHeap(H) operations takes an input N keys and places them into an empty heap by maintaining structure property and heap irder property. This can be done with N successive insertion, since each insert will take O(1) average and o(log N) worst case time.

The total running time of the algorithm would be O(N) average but O(N log N) worst case. The general algorithm is to place the N keys into the tree in any order, maintaining structure property.

APPLICATIONS OF BINARY HEAPS

1) Bandwidth management
2) Discrete event simulation
3) A* and SMA * search algorithm
4) ROAM triangulation algorithm
5) Relationship to sorting algorithm

AllAbtEngg Android Application for Anna University, Polytechnic & School