5 UNIT

Trees

CONTENTS

PART-1

Basic Terminology used with Tree, Binary Trees, Binary Tree Representation : Array Representation and Pointer (Linked List) Representation.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 5.1. Explain the following terms :

i. Tree
ii. Vertex of Tree
iii. Depth
iv. Degree of an element
v. Degree of Tree
vi. Leaf

Answer

i. **Tree :** A tree *T* is a finite non-empty set of elements. One of these elements is called the root, and the remaining elements, if any is partitioned into trees is called subtree of *T*. A tree is a non-linear data structure.

ii. **Vertex of tree :** Each node of a tree is known as vertex of tree.

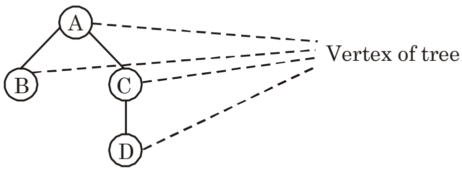


Fig. 5.1.1.

iii. **Depth :** The depth of binary tree is the maximum level of any leaf in the tree. This equals the length of the longest path from the root to any leaf.

Depth of Fig. 5.1.2 tree is 2.

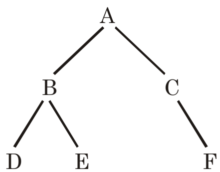


Fig. 5.1.2.

iv. **Degree of an element :** The number of children of node is known as degree of the element.

v. **Degree of tree :** In a tree, node having maximum number of degree is known as degree of tree.

vi. **Leaf :** A terminal node in tree is known as leaf node or a node which has no child is known as leaf node.

Que 5.2. **Show that the maximum number of nodes in a binarytree of height h is $2^{h+1} - 1$.**

Answer

If we consider the maximum nodes in a tree then all leaves will have the same depth and all internal nodes have left child and right child both.

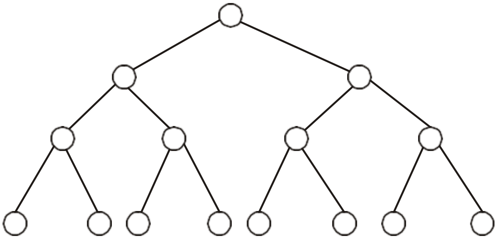


Fig. 5.2.1.

1. The root has 2 children at depth 1, each of which has 2 children at depth 2 *i.e.*, 4.
2. Thus, the number of leaves at depth h is 2^h, so we can calculate the maximum number of nodes in a binary tree as :

$$= 1 + 2 + 4 + 8 + 16 + \ldots 2^h$$
$$= 2^0 + 2^1 + 2^2 + 2^3 + \ldots 2^h$$
$$= \sum_{i=0}^{h} 2^i = \frac{2^{h+1} - 1}{2 - 1} = 2^{h+1} - 1$$

Thus, a binary tree having height h, has $2^{h+1} - 1$ maximum number of nodes.

Que 5.3. **Explain binary tree representation using array.**

Answer

1. In an array representation, nodes of the tree are stored level-by-level, starting from 0^{th} level.
2. Missing elements are represented by white boxes.
3. This representation scheme is wasteful of space when many elements are missing.

4. In fact, a binary tree that has n-elements may require an array of size up to 2^n (including position 0) for its representation.

5. This maximum size is needed when each element (except the root) of the *n*-element binary tree is the right child of its parent.
6. Fig. 5.3.1, shows such a binary tree with four elements. Binary trees of this type are called right-skewed binary trees.

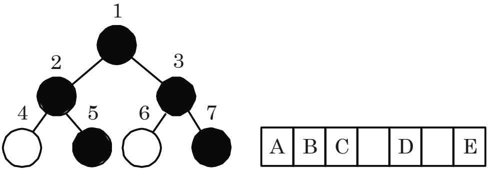


Fig. 5.3.1.

Que 5.4. **Explain binary tree representation using linked list.**

Answer

1. Consider a binary tree *T* which uses three parallel arrays, INFO, LEFT and RIGHT, and a pointer variable ROOT.
2. First of all, each node *N* of *T* will correspond to a location *K* such that :
   a. INFO[*K*] contains the data at the node *N*.
   b. LEFT[*K*] contains the location of the left child of node *N*.
   c. RIGHT[*K*] contains the location of the right child of node *N*.
3. ROOT will contain the location of the root *R* of *T*.
4. If any subtree is empty, then the corresponding pointer will contain the null value.
5. If the tree *T* itself is empty, then ROOT will contain the null value.
6. INFO may actually be a linear array of records or a collection of parallel arrays.

Que 5.5. **Write a C program to implement binary tree insertion, deletion with example.**

AKTU 2016-17, Marks 10

Answer

```
#include<stdlib.h>
#include<stdio.h>
struct bin_tree {
int data;
struct bin_tree *right, *left;
};
typedef struct bin_tree node;
void insert(node *tree, int val)
{
node *temp = NULL;
if(!(*tree))
{
```

```
temp = (node *)malloc(sizeof(node));
temp->left = temp->right = NULL;
temp->data = val;
*tree = temp;
return;
}
if(val < (*tree)->data)
{
insert(&(*tree)->left, val);
}
else if(val > (*tree)->data)
{
insert(&(*tree)->right, val);
}
}
void print_inorder(node *tree)
{
if (tree)
{
print_inorder(tree->left);
printf("%d\n",tree->data);
print_inorder(tree->right);
}
}
void deltree(node *tree)
{
if (tree)
{
deltree(tree->left);
deltree(tree->right);
free(tree);
}
}
void main()
{
node *root;
node *tmp;
//int i;
root = NULL;
/* Inserting nodes into tree */
insert(&root, 9);
insert(&root, 4);
insert(&root, 15);
insert(&root, 6);
insert(&root, 12);
insert(&root, 17);
insert(&root, 2);
```

```
/* Printing nodes of tree */
printf("After insertion inorder display\n");
print_inorder(root);
/* Deleting all nodes of tree */
deltree(root);
printf("Tree is empty");
}
```

Output of program :

```
After insertion inorder display
2
4
6
9
12
15
17
Tree is empty.
```

Que 5.6. Write the C program for various traversing techniquesof binary tree with neat example.

AKTU 2016-17, Marks 10

Answer

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int value;
node* left;
node* right;
};
struct node* root;
struct node* insert(struct node* r, int data);
void inorder(struct node* r);
void preorder(struct node* r);
void postorder(struct node* r);
int main()
{
root = NULL;
int n, v;
printf("How many data do you want to insert ?\n");
scanf("%d", &n);
for(int i=0; i<n; i++){
printf("Data %d: ", i+1);
scanf("%d", &v);
root = insert(root, v);
```

```
}
printf("Inorder Traversal :");
inorder(root);
printf("\n");
printf("Preorder Traversal :");
preorder(root);
printf("\n");
printf("Postorder Traversal :");
postorder(root);
printf("\n");
return 0;
}
struct node* insert(struct node* r, int data)
{
if(r==NULL)
{
r = (struct node*) malloc(sizeof(struct node));
r->value = data;
r->left = NULL;
r->right = NULL;
}
else if(data < r->value){
r->left = insert(r->left, data);
}
else {
r->right = insert(r->right, data);
}
return r;
}
void inorder(struct node* r)
{
if(r!=NULL){
inorder(r->left);
printf("%d ", r->value);
inorder(r->right);
}
}
void preorder(struct node* r)
{
if(r!=NULL){
printf("%d", r->value);
preorder(r->left);
preorder(r->right);
}
}
void postorder(struct node* r)
{
```

```
if(r!=NULL){
postorder(r->left);
postorder(r->right);
printf("%d", r->value);
}
}
```

Output :

How many data do you want to insert ?
5
Preorder Traversal :
3 2 1 4 5
Inorder Traversal :
1 2 3 4 5
Postorder Traversal :
1 2 5 4 3

PART-2

Binary Search Tree, Strictly Binary Tree, Complete Binary Tree,A Extended Binary Tree.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 5.7. **Explain binary search tree and its operations. Make a binary search tree for the following sequence of numbers, show all steps : 45, 32, 90, 34, 68, 72, 15, 24, 30, 66, 11, 50, 10.**

AKTU 2015-16, Marks 10

Answer

Binary search tree :

1. A binary search tree is a binary tree.
2. Binary search tree can be represented by a linked data structure in which each node is an object.
3. In addition to a key field, each node contains fields left, right and *P*, which point to the nodes corresponding to its left child, its right child and its parent respectively.
4. A non-empty binary search tree satisfies the following properties :
   a. Every element has a key (or value) and no two elements have the same value.
   b. The keys, if any, in the left subtree of root are smaller than the key in the node.

c. The keys, if any in the right subtree of the root are larger than the keys in the node.
d. The left and right subtrees of the root are also binary search tree.

Various operations of BST are :

a. Searching in a BST :

Searching for a data in a binary search tree is much faster than in arrays or linked lists. The TREE-SEARCH (x, k) algorithm searches the tree root at x for a node whose key value equals to k. It returns a pointer to the node if it exist otherwise NIL.

TREE-SEARCH (*x*, *k*)

```
If x = NIL or k = key [x]
then return x
If k < key [x]
then return TREE-SEARCH (left [x], k)
else return TREE-SEARCH (right [x], k)
```

b. Traversal operation on BST :

All the traversal operations are applicable in binary search trees. The inorder traversal on a binary search tree gives the sorted order of data in ascending (increasing) order.

c. Insertion of data into a binary search tree :

To insert a new value w into a binary search tree T, we use the procedure TREE-INSERT. The procedure passed a node z for which key[z] = w, left [z] = NIL and Right [z] = NIL.

```
y ← NIL
x ← root [T]
while x ≠ NIL
do y ← x
if key [z] < key [x]
then x ← left [x]
else x ← right [x]
P[z] ← y
if y = NIL
then root [T] ← z
else if key [z] < key [y]
then left [y] ← z
else right [y] ← z
```

d. Delete a node : Deletion of a node from a BST depends on the number of its children. Suppose to delete a node with key = z from BST T, there are 3 cases that can occur.

Case 1 : N has no children. Then N is deleted from T by simply replacingthe location of N in the parent node $P(N)$ by the null pointer.

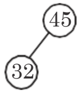
Case 2 : N has exactly one child. Then N is deleted from T by simply replacing the location of N in $P(N)$ by the location of the only child of N.

Case 3 : N has two children. Let $S(N)$ denote the inorder successor of N. (The reader can verify that $S(N)$ does not have a left child). Then N is deleted from T by first deleting $S(N)$ from T (by using Case 1 or Case 2) and then replacing node N in T by the node $S(N)$.

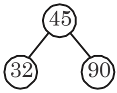
Numerical :

1. Insert 45 :

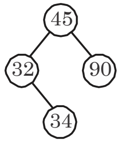
2. Insert 32 :



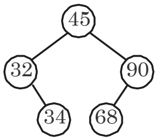
3. Insert 90 :



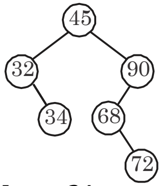
4. Insert 34 :



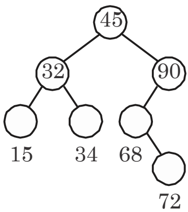
5. Insert 68 :



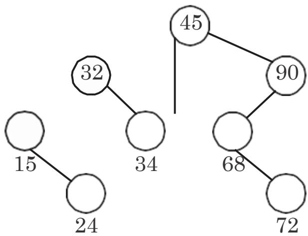
6. Insert 72 :



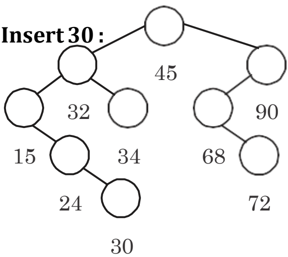
7. Insert 15 :



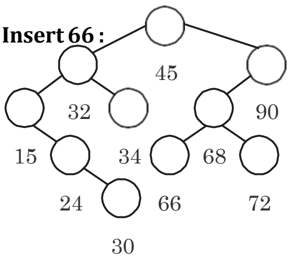
8. Insert 24 :



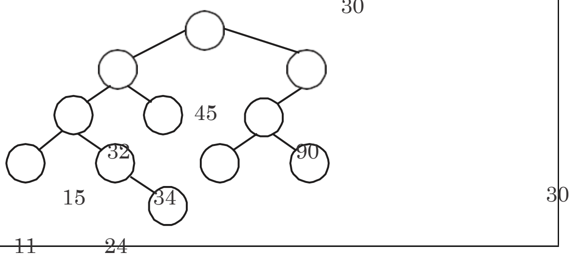
9. Insert 30 :



10. Insert 66 :



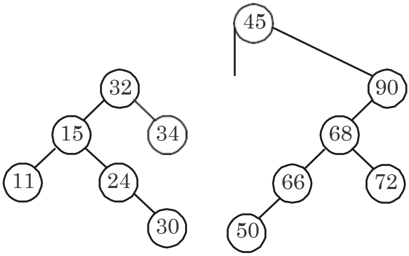
11. Insert 11 :



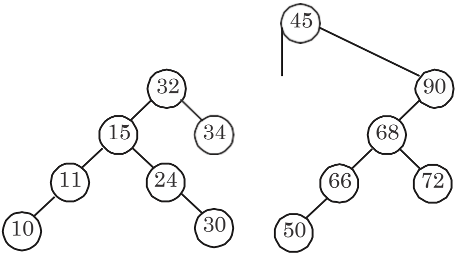
68

66 72

12. Insert 50 :



13. Insert 10 :



Que 5.8. **Define binary search tree. Create BST for the following data, show all steps :**

20, 10, 25, 5, 15, 22, 30, 3, 14, 13

AKTU 2014-15, Marks 10

Answer

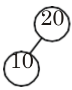
Binary search tree : Refer Q. 5.7, Page 5–8A, Unit-5.

Numerical :

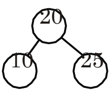
20, 10, 25, 5, 15, 22, 30, 3, 14, 13

1. Insert 20 :

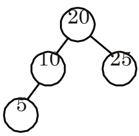
2. Insert 10 :



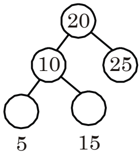
3. Insert 25 :



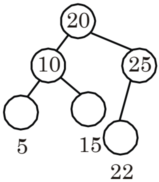
4. Insert 5 :



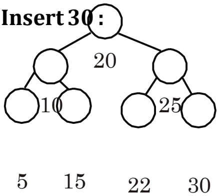
5. **Insert 15 :**



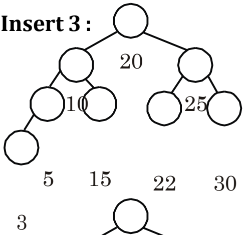
6. **Insert 22 :**



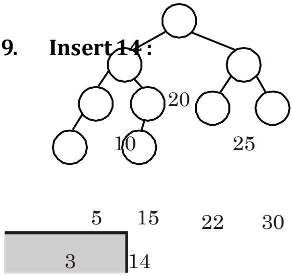
7. **Insert 30 :**



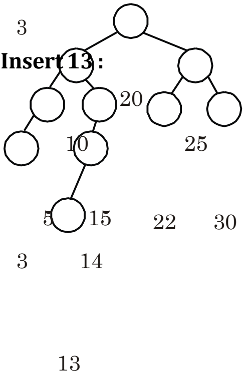
8. **Insert 3 :**



9. **Insert 14 :**



10. **Insert 13 :**



Que 5.9. Write a short note on strictly binary tree, completebinary tree and extended binary tree.

Answer

Strictly binary tree :

a. If every non-leaf node in a binary tree has non-empty left and right subtree, the tree is termed as strictly binary tree.
b. A strictly binary tree with n leaves always contains $2n - 1$ nodes.
c. If every non-leaf node in a binary tree has exactly two children, the tree is known as strictly binary tree.

Complete binary tree : A tree is called a complete binary tree if tree satisfies following conditions :

a. Each node has exactly two children except leaf node.
b. All leaf nodes are at same level.
c. If a binary tree contains m nodes at level l, it contains atmost $2m$ nodes at level $l + 1$.

Extended binary tree :

a. A binary tree T is said to be 2-tree or extended binary tree if each node has either 0 or 2 children.

b. Nodes with 2 children are called internal nodes and nodes with 0 children are called external nodes.

PART-3

Tree Traversal Algorithm : Inorder, Preorder and Postorder, Constructing Binary Tree From Given Tree Traversal.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 5.10. Define tree, binary tree, complete binary tree and full binary tree. Write algorithm or function to obtain traversals of abinary tree in preorder, postorder and inorder.

AKTU 2017-18, Marks 07

Answer

Tree : Refer Q. 5.1, Page 5–2A, Unit-5.

Binary tree :

1. A binary tree T is defined as a finite set of elements called nodes, such that :
   a. T is empty (called the null tree).
   b. T contains a distinguished node R, called the root of T, and the remaining nodes of T form an ordered pair of disjoint binary trees T_1 and T_2.
2. If T does contain a root R, then the two trees T_1 and T_2 are called, respectively, the left and right subtrees of R.
3. If T_1 is non-empty, then its root is called the left successor of R similarly, if T_2 is non-empty, then its root is called the right successor of R.

Complete binary tree : Refer Q. 5.10, Page 5–14A, Unit-5.

Full binary tree :

1. A full binary tree is formed when each missing child in the binary tree is replaced with a node having no children.
2. These leaf nodes are drawn as squares in the Fig. 5.10.1.

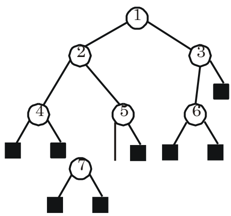


Fig. 5.10.1. Full binary tree.

3. Each node is either a leaf or has degree exactly 2.

Algorithm for preorder traversal :

Preorder (INFO, LEFT, RIGHT, ROOT)

1. [Initially push NULL onto STACK, and initialize PTR]
   Set TOP = 1, STACK [1] = NULL and PTR = ROOT
2. Repeat steps 3 to 5 while PTR ≠ NULL
3. Apply process to INFO [PTR]
4. [Right child?]
   If RIGHT [PTR] ≠ NULL
   Then
   [Push on STACK]
   Set TOP = TOP + 1 and
   STACK [TOP] = RIGHT [PTR]
   Endif
5. [Left child?]
   If LEFT [PTR] ≠ NULL then
   set PTR = LEFT[PTR]
   Else
   [Pop from STACK]
   set PTR = STACK[TOP] and TOP = TOP – 1
   Endif
   End of step 2
6. Exit

Algorithm for inorder traversal :

Inorder (INFO, LEFT, RIGHT, ROOT)

1. [Push NULL onto STACK and initialize PTR]
   Set TOP = 1, STACK[1] = NULL and PTR = ROOT
2. Repeat while PTR ≠ NULL
   [Push leftmost path onto STACK]
   a. Set TOP = TOP + 1 and
   STACK [TOP] = PTR
   b. Set PTR = LEFT [PTR]
   End loop
3. Set PTR = STACK[TOP] and TOP = TOP – 1
4. Repeat steps 5 to 7 while PTR ≠ NULL
5. Apply process to INFO[PTR]
6. [Right Child?] If RIGHT [PTR] ≠ NULL
   Then
   a. Set PTR = RIGHT [PTR]
   b. goto step 2
   Endif
7. Set PTR = STACK[TOP] and TOP = TOP – 1
   End of Step 4 Loop
8. Exit

Algorithm for postorder traversal :

Postorder (INFO, LEFT, RIGHT, ROOT)

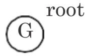
1. [Push NULL onto STACK and initialize PTR]
   Set TOP = 1, STACK[1] = NULL and PTR = ROOT
2. [Push leftmost path onto STACK]
   Repeat steps 3 to 5 while PTR ≠ NULL
3. Set TOP = TOP + 1 and STACK [TOP] = PTR
   [Pushes PTR on STACK]
4. If RIGHT [PTR] ≠ NULL
   Then
   Set TOP = TOP + 1 and STACK [TOP] = RIGHT [PTR]
   Endif
5. Set PTR = LEFT [PTR]
   End of step 2 loop
6. Set PTR = STACK [TOP] and TOP = TOP – 1
   [Pops node from STACK]
7. Repeat while PTR > 0
   a. Apply process to INFO [PTR]
   b. Set PTR = STACK [TOP] and TOP = TOP – 1
   End loop
8. If PTR < 0 Then
   a. Set PTR = – PTR
   b. goto step 2
   Endif
9. Exit

Que 5.11. Construct a binary tree for the following :Inorder : ***Q, B, K, C, F, A, G, P, E, D, H, R***
Preorder : ***G, B, Q, A, C, K, F, P, D, E, R, H***

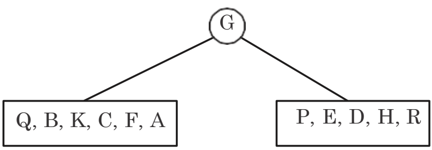
Find the postorder of the tree. **AKTU 2018-19, Marks 07**

Answer

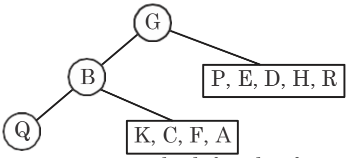
Step 1 : In preorder traversal root is the first node. So, *G* is the root node of the binary tree. So,



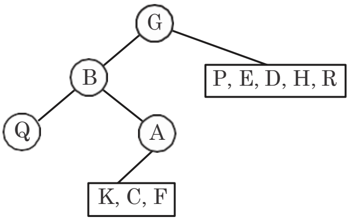
Step 2 : We can find the node of left sub-tree and right sub-tree with inorder sequence. So,



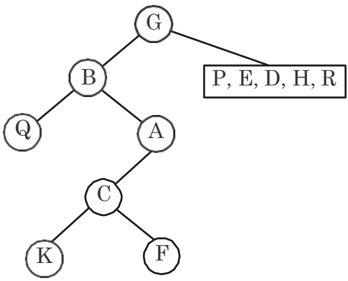
Step 3 : Now, the left child of the root node will be the first node in the preorder sequence after root node G. So,



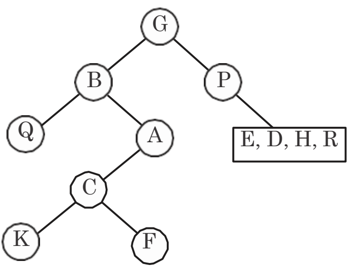
Step 4 : In inorder sequence, Q is on the left side of B and A is on the rightside B. So,



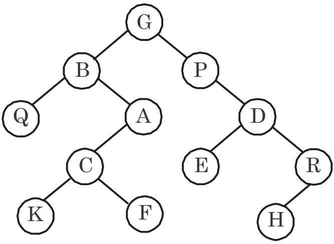
Step 5 : In inorder sequence, C is on the left side of A . Now according to inorder sequence, K is on the left side of C and F is on the right side of C.



Step 6 : Similarly, we can go further for right side of G.



So, the final tree is



Postorder of tree : *Q, K, F, A, B, E, H, R, D, P, G*

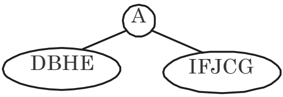
Que 5.12. Draw a binary tree with following traversal :Inorder : *D B H E A I F J C G*

Preorder : *A B D E H C F I J G*

AKTU 2015-16, Marks 10

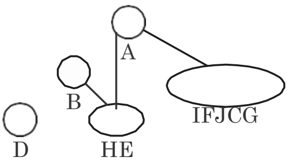
Answer

From preorder traversal, we get root node to be *A*.

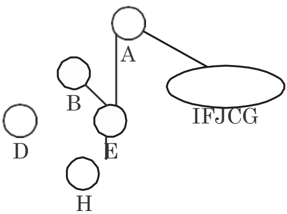


Now considering left subtree.

Observing both the traversal we can get *B* as root node and *D* as left child and *HE* as a right subtree.



Now observing the preorder traversal we get *E* as a root node and *H* as a left child.



Repeating the above process with the right subtree of root node *A*, we finally obtain the required tree in given Fig. 5.12.1.

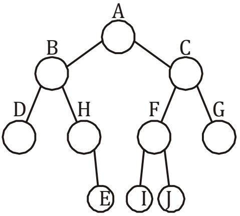


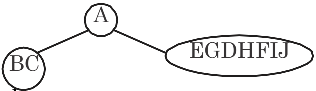
Fig. 5.12.1.

Que 5.13. Draw a binary tree with following traversals :Inorder : *B C A E G D H F I J*

Preorder : *A B C D E F G H I J* **AKTU 2017-18, Marks 07**

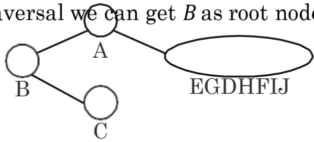
Answer

From preorder traversal, we get root node to be *A*.

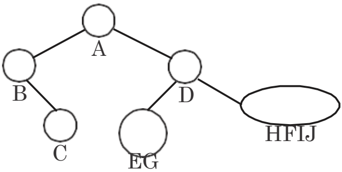


Now considering left subtree.

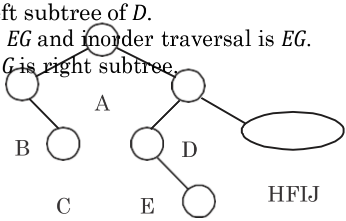
Observing both the traversal we can get *B* as root node and *C* as right child.



Now, consider the right subtree.
Preorder traversal is *DEGFHIJ*, which shows *D* is root node.
Inorder traversal is *EGDHFIJ*, which shows *EG* is left subtree and *HFIJ* is right subtree.

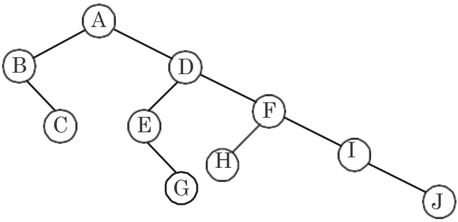


Now, consider the left subtree of *D*.
Preorder traversal is *EG* and inorder traversal is *EG*.
∴ *E* is root node and *G* is right subtree.



G

Similarly, following the same procedure, we finally get



PART-4

Operation of Insertion, Deletion, Searching and Modification ofData in Binary Search.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 5.14. **Write a procedure to insert a new element in a binary search tree.**

Answer

INSBST(INFO, LEFT, RIGHT, ROOT, AVAIL, ITEM, LOC)
A binary search tree *T* is in memory and an ITEM of information is given. This algorithm finds the location LOC of ITEM in *T* or adds ITEM as a new node in *T* at location LOC.

1. Call FIND(INFO, LEFT, RIGHT, ROOT, ITEM, LOC, PAR).
2. If LOC ≠ NULL, then Exit.
3. [Copy ITEM into new node in AVAIL list.]
   a. If AVAIL = NULL, then write OVERFLOW, and Exit.
   b. Set NEW := AVAIL, AVAIL := LEFT[AVAIL] and INFO[NEW] := ITEM.
   c. Set LOC := NEW, LEFT[NEW] := NULL and RIGHT[NEW] := NULL.
4. [Add ITEM to tree.]
   If PAR = NULL, then :
      Set ROOT := NEW.
   Else if ITEM < INFO[PAR], then:
      Set LEFT[PAR] := NEW.
   Else :
      Set RIGHT[PAR] := NEW
   [End of If structure]

5. Exit.

Que 5.15. Write the algorithm for deletion of an element in binarysearch tree.

AKTU 2018-19, Marks 07

Answer

DEL(INFO, LEFT, RIGHT, ROOT, AVAIL, ITEM)

A binary search tree *T* is in memory, and an ITEM of information is given. This algorithm deletes ITEM from the tree.

1. [Find the locations of ITEM and its parent]
   Call FIND(INFO, LEFT, RIGHT, ROOT, ITEM, LOC, PAR)
2. [ITEM in tree ?]
   If LOC = NULL, then write ITEM not in tree, and Exit.
3. [Delete node containing ITEM]
   If RIGHT[LOC] ≠ NULL and LEFT[LOC] ≠ NULL, then :
   Call CASEB(INFO, LEFT, RIGHT, ROOT, LOC, PAR)
   Else :
   Call CASEA(INFO, LEFT, RIGHT, ROOT, LOC, PAR)
   [End of If structure]
4. [Return deleted node to the AVAIL list]
   Set LEFT[LOC] := AVAIL and AVAIL := LOC
5. Exit.

Que 5.16. Write a procedure to delete an element from binary search tree where node does not have two children.

Answer

CASEA(INFO, LEFT, RIGHT, ROOT, LOC, PAR)

This procedure deletes the node N at location LOC, where N does not have two children. The pointer PAR gives the location of the parent of N, or else PAR = NULL indicates that N is the root node. The pointer CHILD gives thelocation of the only child of N, or else CHILD = NULL indicates N has nochildren.

1. [Initializes CHILD]
   If LEFT[LOC] = NULL and RIGHT[LOC] = NULL, then:
   Set CHILD := NULL.
   Else if LEFT(LOC) ≠ NULL, then :
   Set CHILD := LEFT[LOC].
   Else
   Set CHILD := RIGHT[LOC].
   [End of If structure.]
2. If PAR ≠ NULL, then :
   If LOC = LEFT[PAR], then :
      Set LEFT[PAR] := CHILD.
   Else :
      Set RIGHT[PAR] := CHILD.

[End of If structure.]
Else :
Set ROOT := CHILD.
[End of If structure.]
3. Return.

Que 5.17. Write procedure to delete an element from binary search tree where node has two children.

Answer

CASEB(INFO, LEFT, RIGHT, ROOT, LOC, PAR)
This procedure will delete the node N at location LOC, where N has two children. The pointer PAR gives the location of the parent of N, or else PAR = NULL indicates that N is the root node. The pointer SUC gives the location of the inorder successor of N, and PARSUC gives the location of the parent of the inorder successor.

1. [Find SUC and PARSUC]
   a. Set PTR := RIGHT[LOC] and SAVE := LOC.
   b. Repeat while LEFT[PTR] ≠ NULL:
      Set SAVE := PTR and PTR := LEFT[PTR].
      [End of loop.]
   c. Set SUC := PTR and PARSUC := SAVE.
2. [Delete inorder successor]
   Call CASEA(INFO, LEFT, RIGHT, ROOT, SUC, PARSUC).
3. [Replace node N by its inorder successor.]
a. If PAR ≠ NULL, then:
   If LOC = LEFT[PAR], then:
   Set LEFT[PAR] := SUC.
   Else :
   Set RIGHT[PAR] := SUC
   [End of If structure.]
   Else :
   Set ROOT := SUC.
   [End of If structure.]
b. Set LEFT[SUC] := LEFT[LOC] and
   RIGHT[SUC] := RIGHT[LOC].
4. Return.

Que 5.18. Write a procedure to search an element in the binarysearch tree.

Answer

FIND(INFO, LEFT, RIGHT, ROOT, ITEM, LOC, PAR)
A binary search tree T is the memory and an ITEM of information is given. This procedure finds the location LOC of ITEM in T and also the location PAR of the parent of ITEM. There are three special cases :

i. LOC = NULL and PAR = NULL will indicate that the tree is empty.

ii. LOC ≠ NULL and PAR = NULL will indicate that ITEM is the root of T.

iii. LOC = NULL and PAR ≠ NULL will indicate that ITEM is not in T and can be added to T as a child of the node N with location PAR.

1. [Tree empty ?]
   If ROOT = NULL, then: Set LOC := NULL and PAR := NULL, and Return.
2. [ITEM at root ?]
   If ITEM = INFO[ROOT], then: Set LOC := ROOT and PAR := NULL, and Return.
3. [Initialize pointers PTR and SAVE.]
   If ITEM < INFO[ROOT], then:
      Set PTR := LEFT[ROOT] and SAVE := ROOT.
   Else :
      Set PTR := RIGHT[ROOT] and SAVE := ROOT.
   [End of If structure.]
4. Repeart steps 5 and 6 while PTR ≠ NULL:
5. [ITEM found ?]
   If ITEM = INFO[PTR], then: Set LOC := PTR and PAR := SAVE, and Return.
6. If ITEM < INFO[PTR], then:
      Set SAVE := PTR and PTR := LEFT[PTR].
   Else :
      Set SAVE := PTR and PTR := RIGHT[PTR].
   [End of If structure]
   [End of step 4 loop.]
7. [Search unsuccessful.] Set LOC := NULL and PAR := SAVE.
8. Exit.

PART-5

Threaded Binary Trees, Traversing Threaded Binary Trees,Huffman Coding using Binary Tree.

Questions-Answers

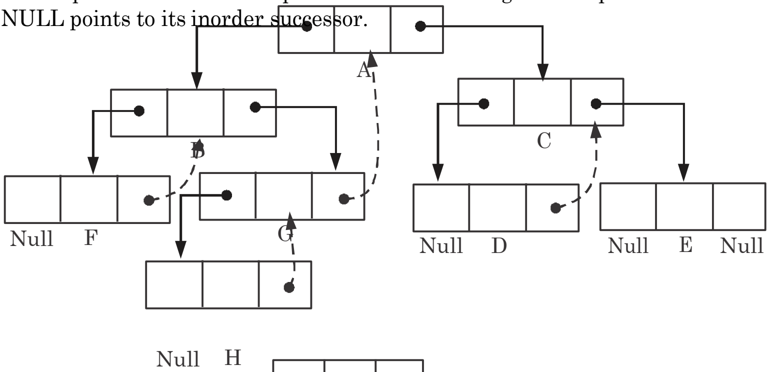
Long Answer Type and Medium Answer Type Questions

Que 5.19. What is a threaded binary tree ? Explain the advantages of using a threaded binary tree.

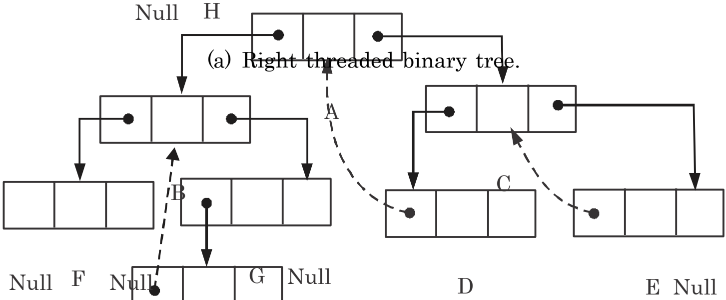
AKTU 2017-18, Marks 07

Answer

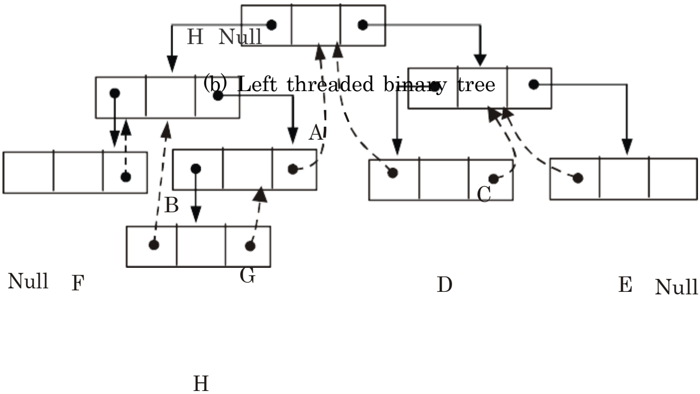
Threaded binary tree is a binary tree in which all left child pointers that are NULL points to its inorder predecessor and all right child pointers that are NULL points to its inorder successor.



(a) Right threaded binary tree.



(b) Left threaded binary tree



(c) Fully threaded binary tree

Fig. 5.19.1.

Advantages of using threaded binary tree :

1. In threaded binary tree the traversal operations are very fast.

2. In threaded binary tree, we do not require stack to determine the predecessor and successor node.
3. In a threaded binary tree, one can move in any direction *i.e.*, upward or downward because nodes are circularly linked.
4. Insertion into and deletions from a threaded tree are all although time consuming operations but these are very easy to implement.

Que 5.20. **Write algorithm/function for inorder traversal of threaded binary tree.**

Answer

Algorithm for inorder traversal in threaded binary tree :

1. Initialize current as root
2. While current is not NULL
   If current does not have left child
   a. Print current's data
   b. Go to the right, *i.e.*, current = current->right
   Else
      a. Make current as right child of the rightmost node in current's left subtree
      b. Go to this left child, *i.e.*, current = current->left

Que 5.21. **What is Huffman tree ? Create a Huffman tree withfollowing numbers :**

24, 55, 13, 67, 88, 36, 17, 61, 24, 76

AKTU 2014-15, Marks 10

Answer

Huffman tree is a binary tree in which each node in the tree represents a symbol and each leaf represent a symbol of original alphabet.

Huffman algorithm :

1. Suppose, there are n weights W_1, W_2,......., W_n.
2. Take two minimum weights among the n given weights. Suppose W_1 and W_2 are first two minimum weights then subtree will be :

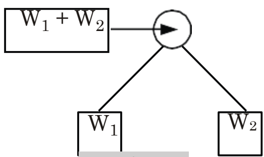


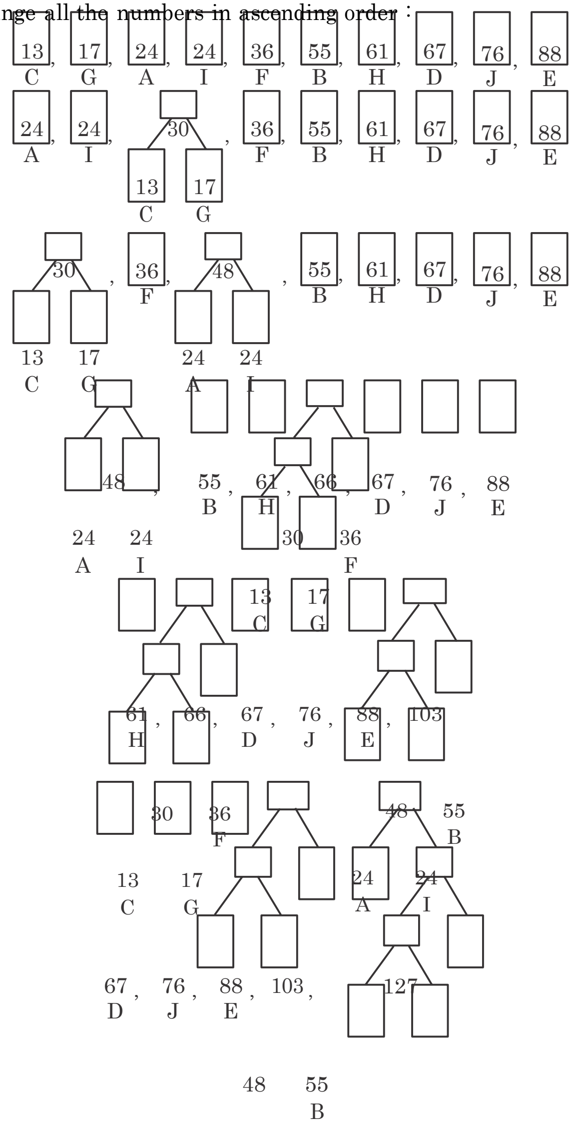
Fig. 5.21.1.

3. Now the remaining weights will be $W_1 + W_2$, W_3, W_4, ... , W_n.
4. Create all subtree at the last weight.

Numerical :

24	55	13	67	88	36	17	61	24	76
A	B	C	D	E	F	G	H	I	J

Arrange all the numbers in ascending order :



24 61 66

A I H

30 36

F

13 17

C G

88 , 103 , 127 , 143
E

48 55 61 66 67 76
B H D J

24 24 30 36
A I F

13 17
C G

191

127 , 143 , 88 103
E

61 66 67 76 48 55

H D J B

30 36 24 24
F A I

13 17
C G

191 , 270

88 103 127 143
E

48 55 61 66 67 76
B H D J

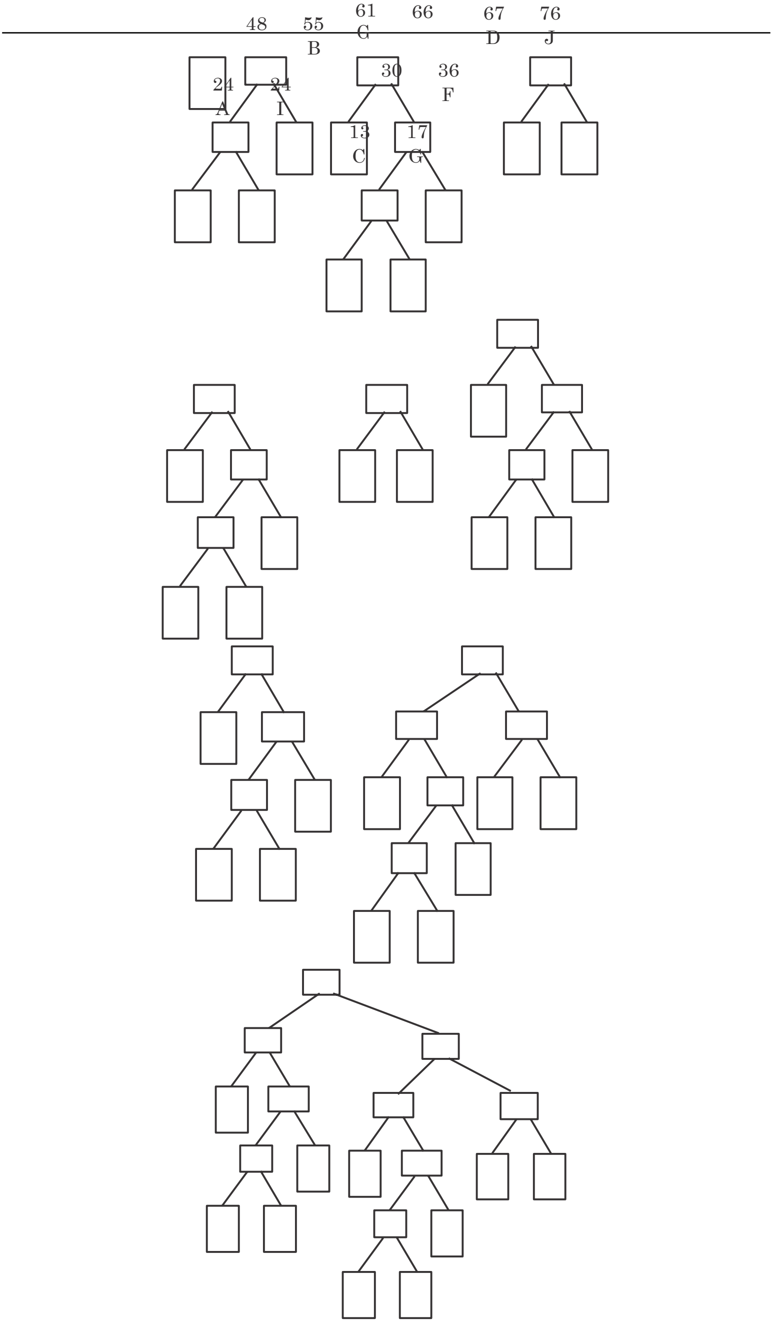
24 24 30 36
A I F

13 17
C G

461

191 270

88 103 127 143
E


48
55
B
61
G
66
67
D
76
J
24
A
24
I
30
36
F
13
C
17
G

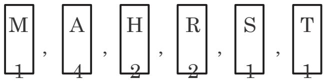
Que 5.22. **Explain Huffman algorithm. Construct Huffman tree for *MAHARASHTRA* with its optimal code.**

AKTU 2018-19, Marks 07

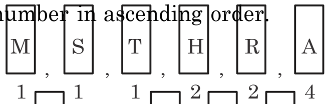
Answer

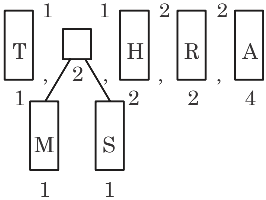
Huffman algorithm : Refer Q. 5.21, Page 5–24A, Unit-5.

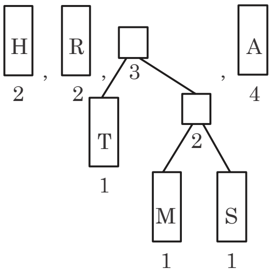
Numerical :

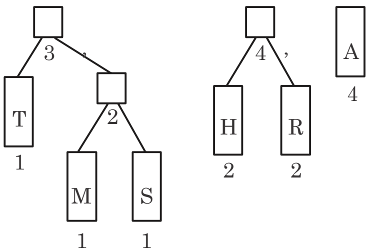


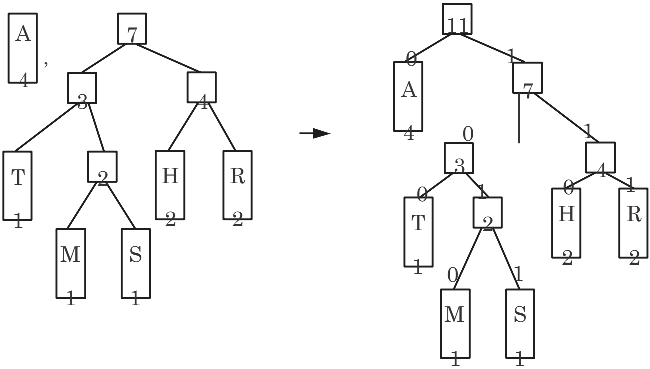
Arrange all the number in ascending order.











Character	Code
M	1010
A	0
H	110
R	111
S	1011
T	100

Optimal code for *MAHARASHTRA* is :

1010011001110101111010011110

PART-6

Concept and Basic Operation for AVL Tree, B tree and BinaryHeaps.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 5.23. Define AVL trees. Explain its rotation operations with example. Construct an AVL tree with the values 10 to 1 numbersinto an

initially empty tree. **AKTU 2016-17, Marks 15**

Answer

i. An AVL (or height balanced) tree is a balanced binary search tree.

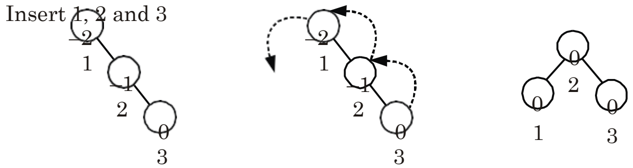
ii. In an AVL tree, balance factor of every node is either –1, 0 or +1.

iii. Balance factor of a node is the difference between the heights of left and right subtrees of that node.

Balance factor = height of left subtree – height of right subtree

iv. In order to balance a tree, there are four cases of rotations :

1. **Left Left rotation (LL rotation) :** In LL rotation every node movesone position to left from the current position.



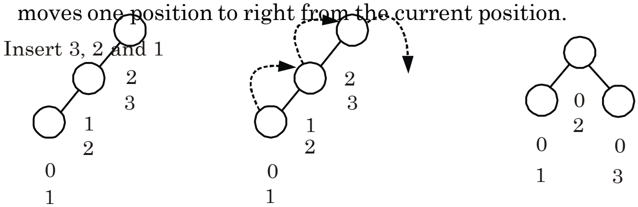
Tree is unbalanced

To make tree balance we use LL rotation which moves nodes one position to left

After LL rotation tree is balanced

Fig. 5.23.1.

2. **Right Right rotation (RR rotation) :** In RR rotation every node moves one position to right from the current position.



Tree is unbalanced because node 3 has balance factor 2

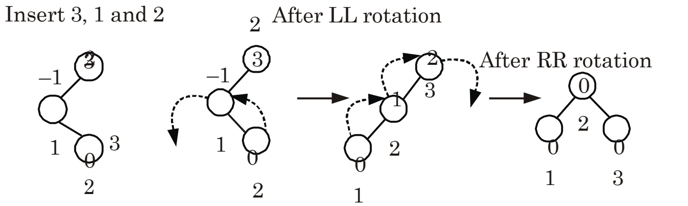
To make tree balance we use RR rotation which moves nodes one position to right

After RR Rotation tree is balanced

Fig. 5.23.2.

3. **Left Right rotation (LR rotation) :** The LR Rotation is combination of

single left rotation followed by single right rotation. In LR rotation, first every node moves one position to left then one position to right from the current position.



Tree is unbalanced because node 3 has balanced factor 2

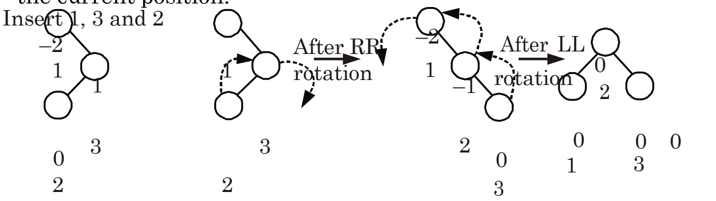
LL rotation

RR rotation

After LR rotation tree is balanced

Fig. 5.23.3.

4. **Right Left rotation (RL rotation) :** The RL rotation is the combination of single right rotation followed by single left rotation. In RL rotation, first every node moves one position to right then one position to left from the current position.



Tree is unbalanced because node 1 has balance factor –2

RR rotation

LL rotation

After RL rotation tree is balanced

Fig. 5.23.4.

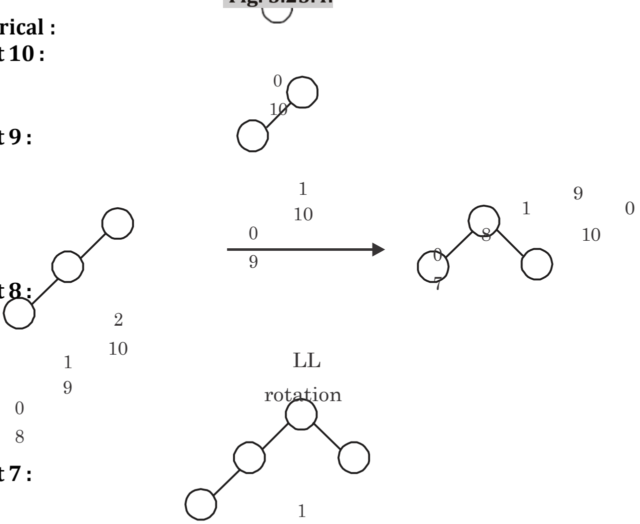
Numerical :

Insert 10 :

Insert 9 :

Insert 8 :

Insert 7 :



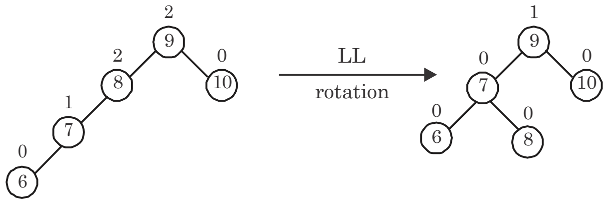
1

9

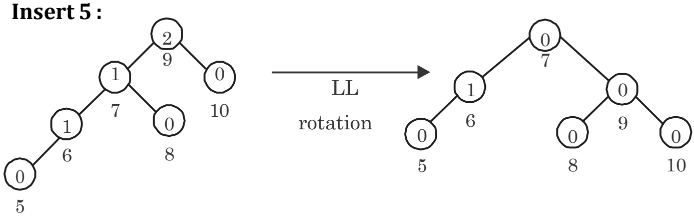
0 0

8 10

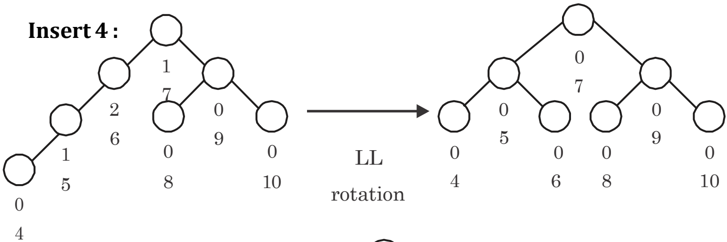
Insert 6 :



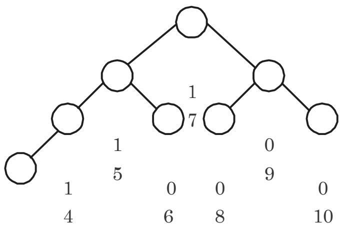
Insert 5 :



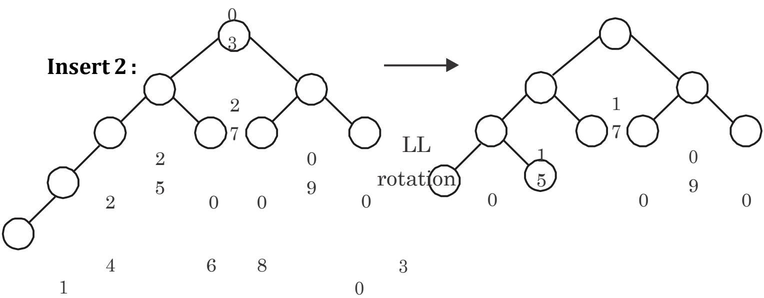
Insert 4 :



Insert 3 :



Insert 2 :



2

10

3

6

8

10

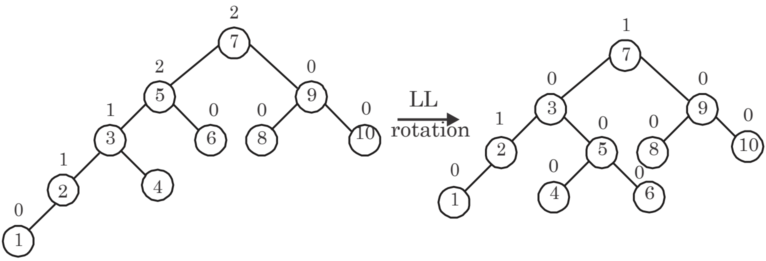
0

0

2

4

Insert 1 :



Que 5.24. **Consider the following AVL tree and insert 2, 12, 7 and 10 as new node. Show proper rotation to maintain the tree as AVL.**

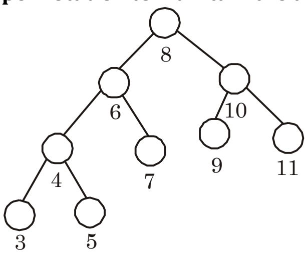
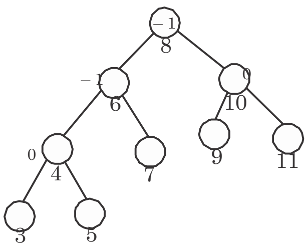


Fig. 5.24.1.

AKTU 2017-18, Marks 07

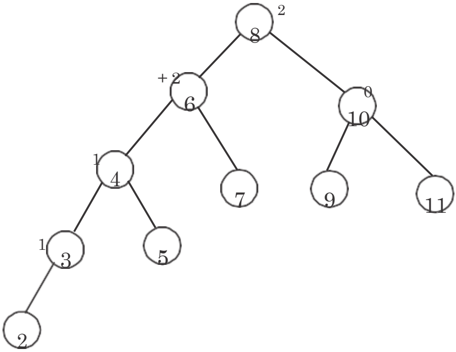
Answer

Given tree :

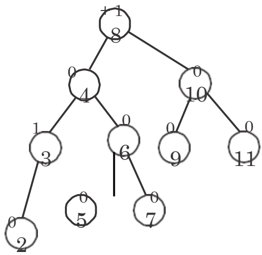


Balanced tree

Insert 2 :

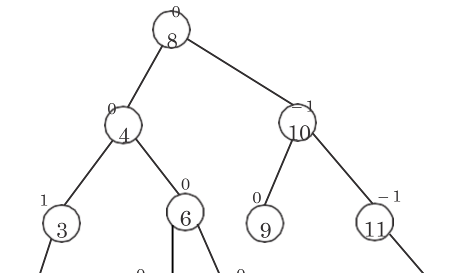


Tree is unbalanced, now LL rotation is required to balance it.



Now the tree is balanced.

Insert 12 :



Tree is balanced, so there is no need to balance the tree.

Insert 7 : 7 is already in the tree hence it cannot be inserted in the AVL tree.

Insert 10 : 10 is also in the tree hence it cannot be inserted in the AVL tree.

Que 5.25. What is height balanced tree ? Why height balancing of tree is required ? Create an AVL tree for the following elements : *a, z, b, y, c, x, d, w, e, v, f*.

AKTU 2018-19, Marks 07

Answer

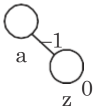
Height balanced tree : Refer Q. 5.23, Page 5–28A, Unit-5.

Height balancing of tree is required : Height balancing of tree is required to implement an AVL tree. Each node must contain a balance factor, which indicates its states of balance relative to its sub-tree.

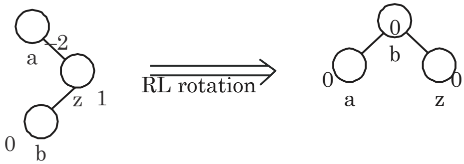
Numerical :

Insert *a* :

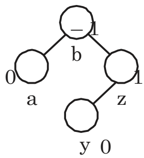
Insert *z* :



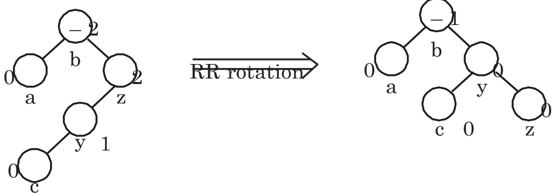
Insert *b* :



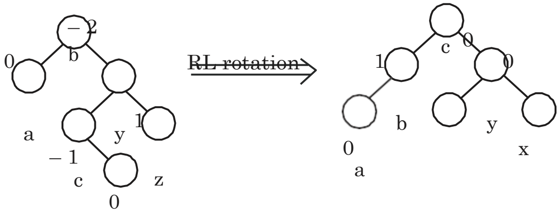
Insert *y* :



Insert *c* :

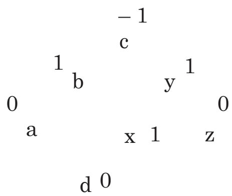


Insert *x* :

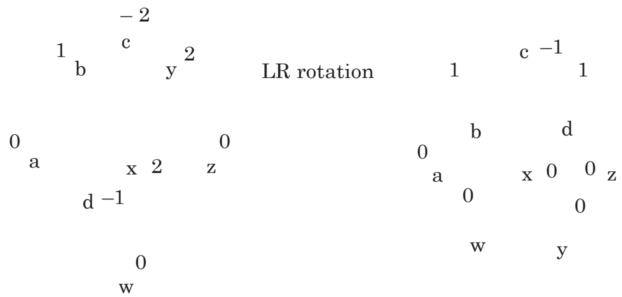


$x \quad 0 \qquad z^{0}$

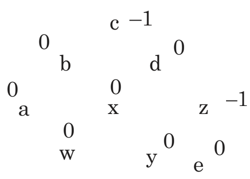
Insert *d* :



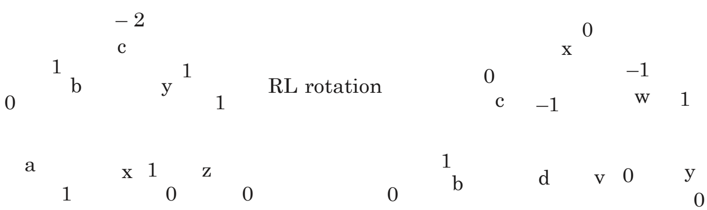
Insert *w* :



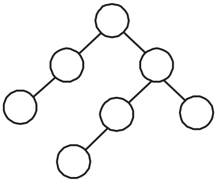
Insert *e* :

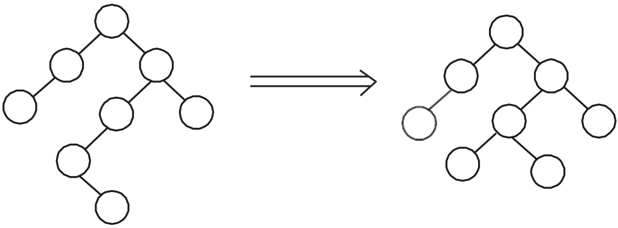


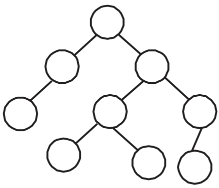
Insert *v* :

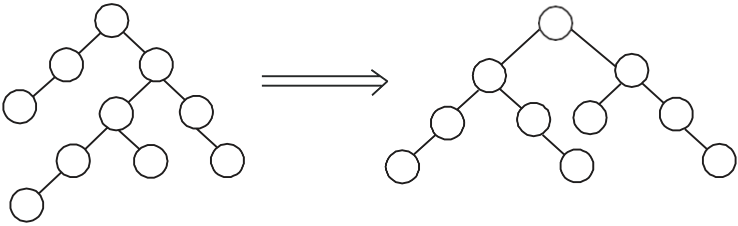


w y c a 0 e z
0
v

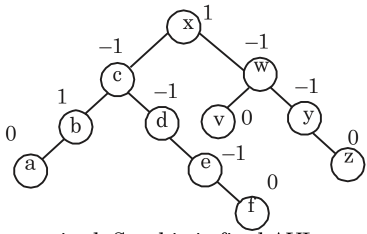








Insert *f* :



No, rebalancing required. So, this is final AVL search tree.

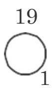
Que 5.26. **Construct a height balanced binary search tree by performing following operations :**
Step 1 : Insert
19, 16, 21, 11, 17, 25, 6, 13
Step 2 : Insert3
Step 3 : Delete
16

AKTU 2014-15, Marks 10

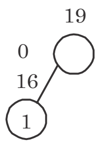
Answer

Step 1 : Insert 19, 16, 21, 11, 17, 25, 6, 13 :

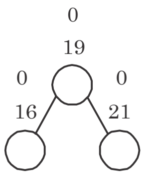
Insert 19 :



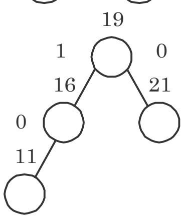
Insert 16 :



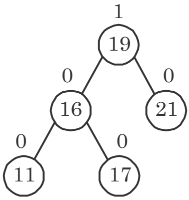
Insert 21 :



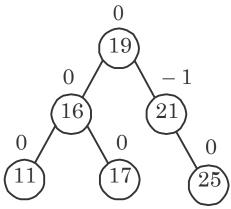
Insert 11 :



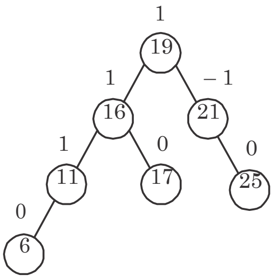
Insert 17 :



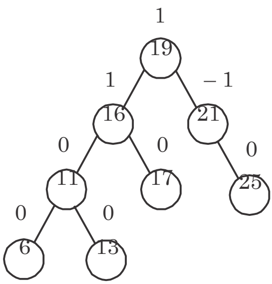
Insert 25 :



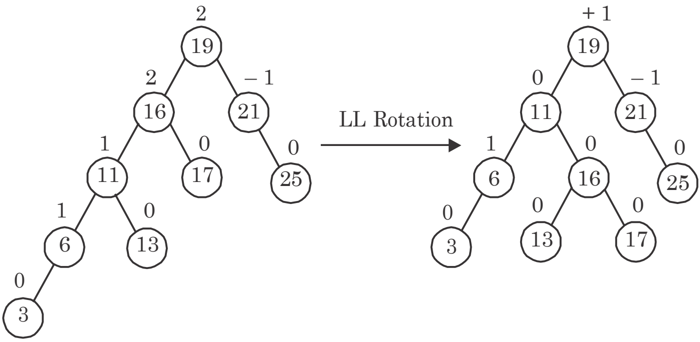
Insert 6 :



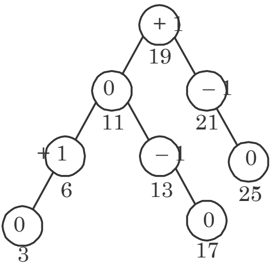
Insert 13 :



Step 2 : Insert 3 :



Step 3 : Delete 16 :



Que 5.27. **Describe all rotations in AVL tree. Construct AVL tree from the following nodes : *B, C, G, E, F, D, A*.**

AKTU 2015-16, Marks 10

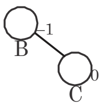
Answer

AVL rotations : Refer Q. 5.23, Page 5–28A, Unit-5.

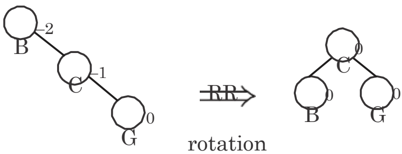
Construction of AVL tree : B, C, G, E, F, D, A

Insert B :

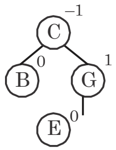
Insert C :



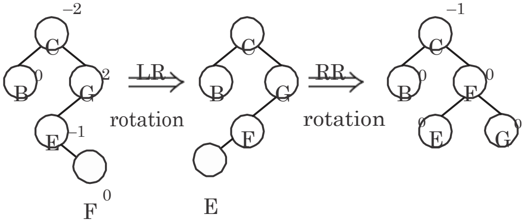
Insert G :



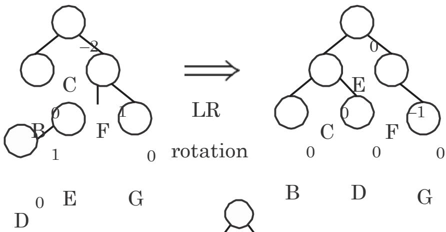
Insert E :



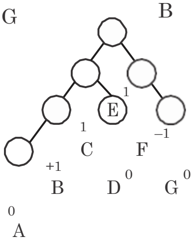
Insert F :



Insert D :



Insert A :



Que 5.28. Define a B-tree. What are the applications of B-tree ? Draw a B-tree of order 4 by insertion of the following keys in order : *Z, U, A, I, W, L, P, X, C, J, D, M, T, B, Q, E, H, S, K, N, R, G, Y, F, O, V.*

AKTU 2015-16, Marks 15

OR

Write a short notes on B-tree. **AKTU 2014-15, Marks 05**

Answer

B-tree :

1. A B-tree is a self-balancing tree data structure that keeps data sorted and allows searches, sequential access, insertions, and deletions in logarithmic time.
2. A B-tree of order *m* is a tree which satisfies the following properties :

a. Every node has at most m children.
b. Every non-leaf node (except root) has at least $\lceil m/2 \rceil$ children.
c. The root has at least two children if it is not a leaf node.

d. A non-leaf node with k children contains $k-1$ keys.
e. All leaves appear in the same level.

Application of B-tree : The main application of a B-tree is the organizationof a huge collection of records into a file structure. The organization should be in such a way that any record in it can be searched very efficiently *i.e.*, insertion, deletion and modification operations can be carried out perfectly and efficiently.

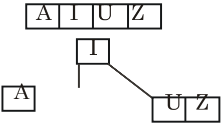
Construction of B-tree :

Insert Z :

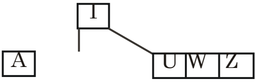
Insert U :

Insert A :

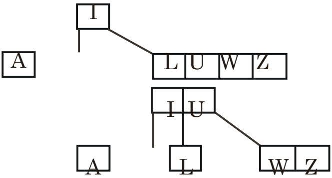
Insert I :



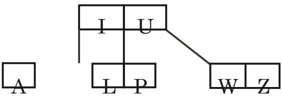
Insert W :



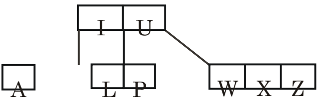
Insert L :



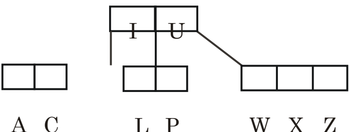
Insert P :



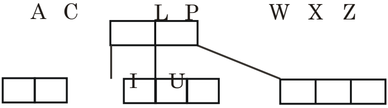
Insert X :



Insert C :



Insert J :



A C J L P W X Z

Insert D :

I U

A C D J L P W X Z

Insert M :

I U

A C D J L M P W X Z

I L U

A C D J M P W X Z

Insert T :

I L U

A C D J M P T W X Z

Insert B :

I L U

A B C D J M P T W X Z

B I L U

A C D J M P T W X Z

I

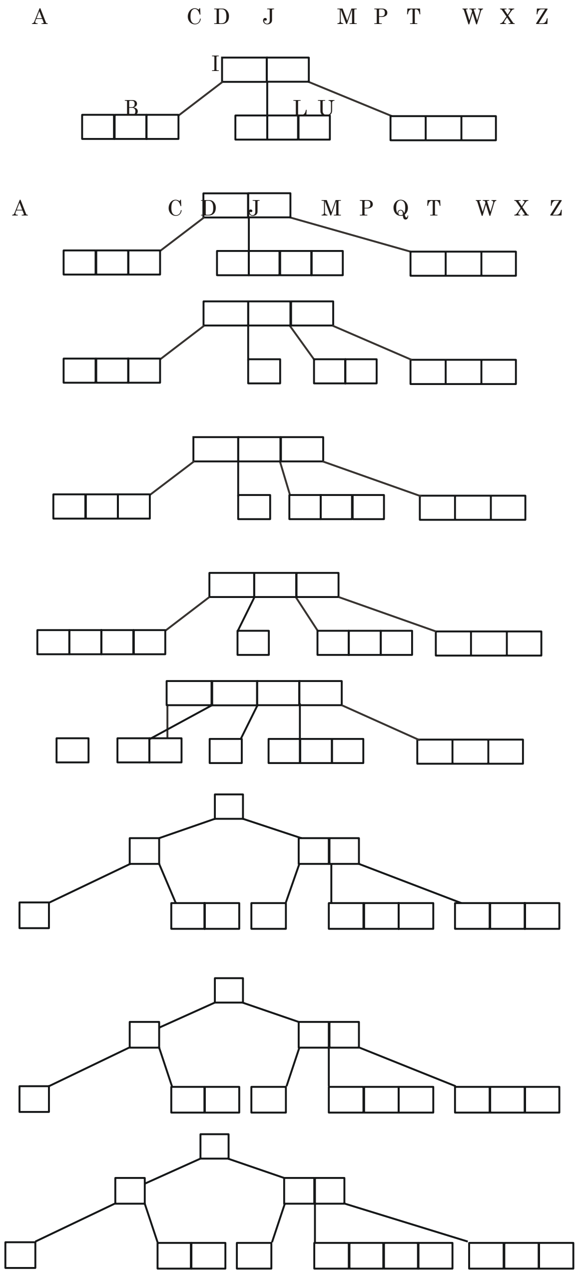
B L U

A C D J M P T W X Z

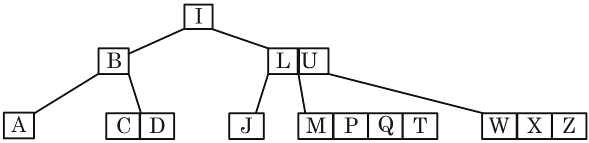
Insert Q :

I

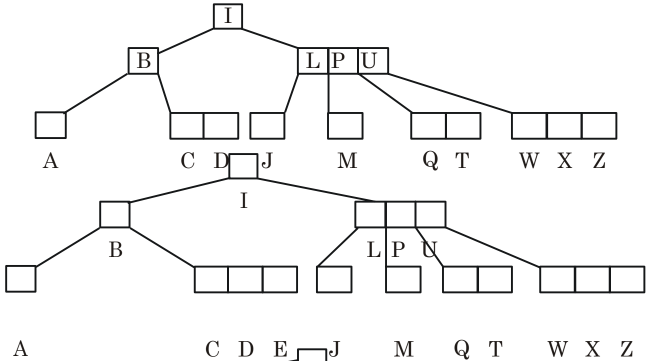
B L U


A C D J M P T W X Z
I
B
L U
A C D J M P Q T W X Z

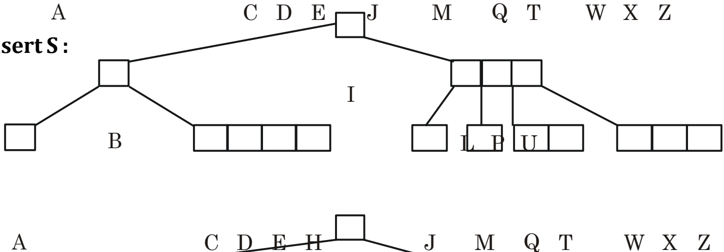
Insert E :



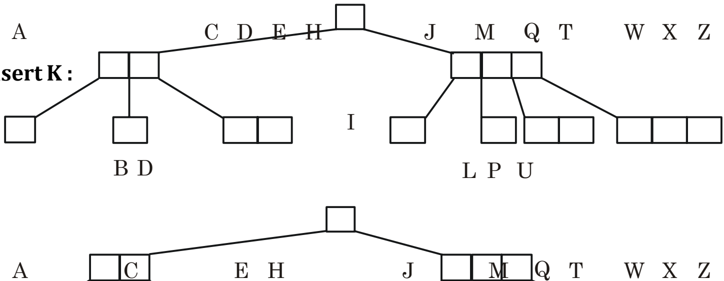
Insert H :



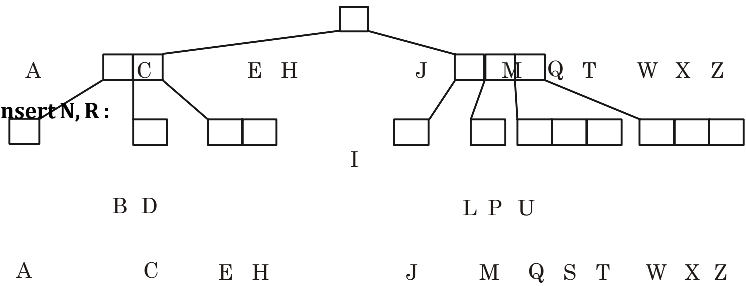
Insert S :



Insert K :



Insert N, R :



I

B D L P U

A C E H J M Q S T W X Z

I P

B D L R U

A C E H J K M N Q S T W X Z

Insert G, Y :

I P

B D L R U

A C E G H J K M N Q S T W X Y Z

I P

B D L R U X

A C E G H J K M N Q S T W Y Z

Insert F, O & V :

I P

B D L R U X

A C E F G H J K M N O Q S T V W Y Z

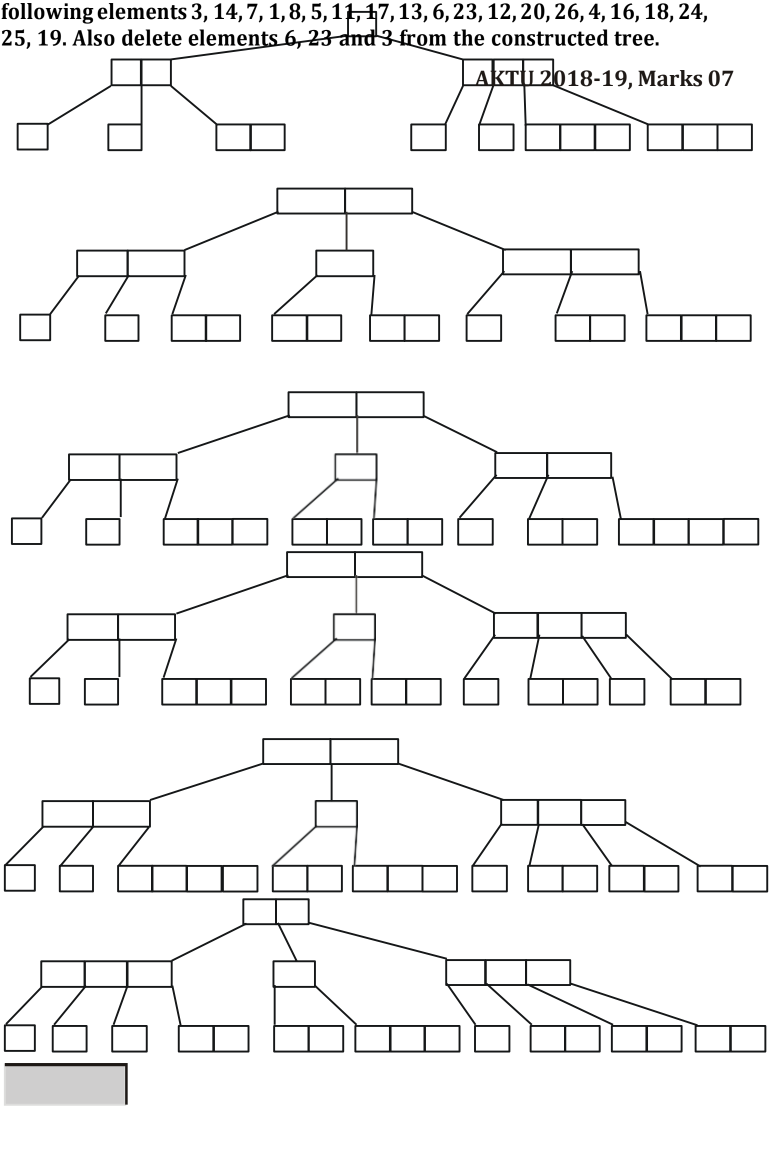
I P

B D F L R U X

A C E G H J K M N O Q S T V W Y Z

Que 5.29. **Construct a B-tree of order 5 created by inserting the following elements 3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20, 26, 4, 16, 18, 24, 25, 19. Also delete elements 6, 23 and 3 from the constructed tree.**

AKTU 2018-19, Marks 07



Answer

Insert 3 : 3

Insert 14 : 3 14

Insert 7 : 3 7 14

Insert 1 : 1 3 7 14

Insert 8 :

8

1 3 7 14

Insert 5 :

8

1 3 5 7 14

Insert 11 :

8

1 3 5 7 11 14

Insert 17 :

8

1 3 5 7 11 14 17

Insert 13 :

8

1 3 5 7 11 13 14 17

Insert 6 :

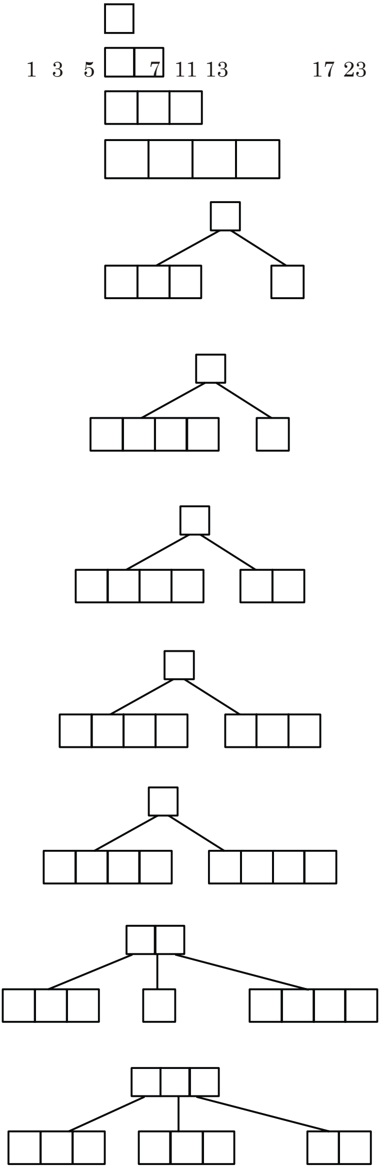
6 8

1 3 5 7 11 13 14 17

Insert 23 :

6 8 14

1 3 5 7 11 13 17 23



Insert 12 :

6 8 14

1 3 5 7 11 12 13 17 23

Insert 20 :

6 8 14

1 3 5 7 11 12 13 17 20 23

Insert 26 :

6 8 14

1 3 5 7 11 12 13 17 20 23 26

Insert 4 :

6 8 14

1 3 4 5 7 11 12 13 17 20 23 26

Insert 16 :

6 8 14 20

1 3 4 5 7 11 12 13 16 17 23 26

Insert 18, 24, 25 :

6 8 14 20

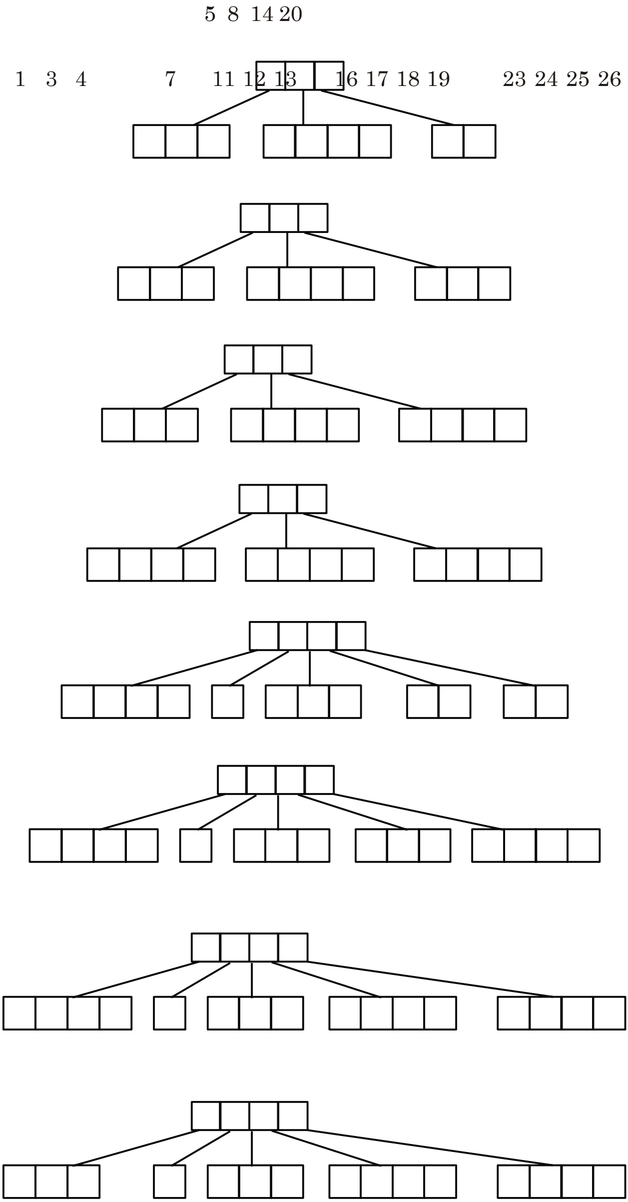
1 3 4 5 7 11 12 13 16 17 18 23 24 25 26

Insert 19 :

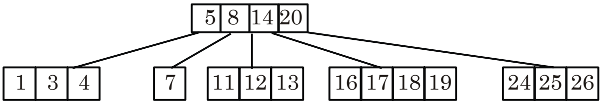
6 8 14 20

1 3 4 5 7 11 12 13 16 17 18 19 23 24 25 26

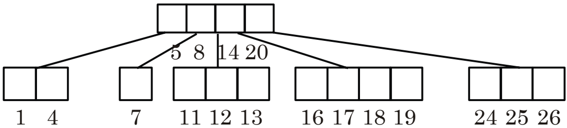
Delete 6 :


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

Delete 23 :



Delete 3 :



Que 5.30. **Construct a B-tree on following sequence of inputs.**

10, 20, 30, 40, 50, 60, 70, 80, 90

Assume that the order of the B-tree is 3.

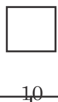
AKTU 2017-18, Marks 07

Answer

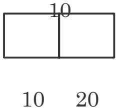
10, 20, 30, 40, 50, 60, 70, 80, 90

Order of the *B*-tree is 3.

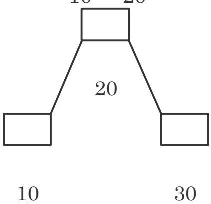
1. Insert 10 :



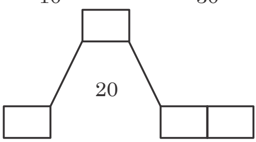
2. Insert 20 :



3. Insert 30 :

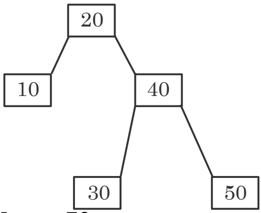


4. Insert 40 :

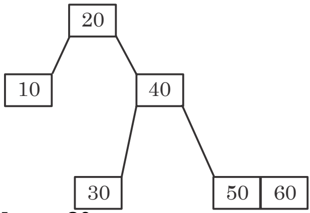


10 30 40

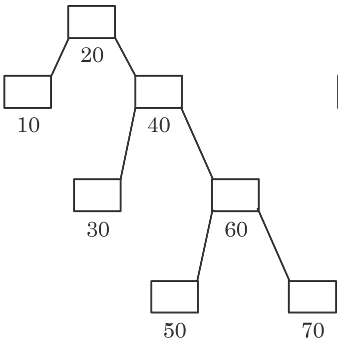
5. Insert 50 :



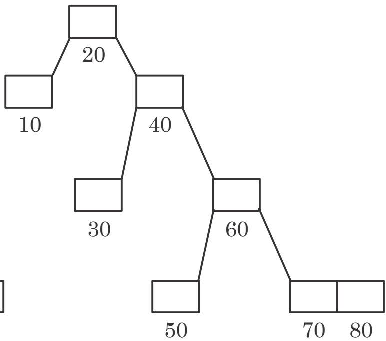
6. Insert 60 :



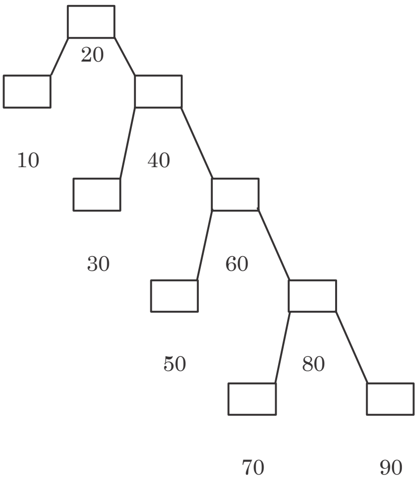
7. Insert 70 :



8. Insert 80 :



9. Insert 90 :



This is final B-tree of order 3.

Que 5.31. Compare and contrast the difference between B+ tree index files and B-tree index files with an example.

AKTU 2016-17, Marks 10

Answer

S. No.	Basis	B^+ tree	B-tree
1.	Definition	B^+ tree is an n-array tree with a variable but often large number of children per node. A B^+ tree consists of a root, internal nodes and leaves. The root may be either a leaf or a node with two or more children.	A B-tree is an organizational structure for information storage and retrieval in the form of a tree in which all terminal nodes are at the same distance from the base, and all non-terminal nodes have between n and $2n$ sub-trees or pointers (where n is an integer).
2.	Space complexity	$O(n)$	$O(n)$
3.	Storage	In a B^+ tree, data is stored only in leaf nodes.	In a B-tree, search keys and data are stored in internal or leaf nodes.
4.	Data	The leaf nodes of the tree store the actual record rather than pointers to records.	The leaf nodes of the tree store pointers to records rather than actual records.
5.	Space	These trees do not waste space.	These trees waste space.
6.	Function of leaf nodes	In B^+ tree, leaf node data are ordered in a sequential linked list.	In B-tree, the leaf node cannot store using linked list.
7.	Searching	In B^+ tree, searching of any data is very easy because all data is found in leaf nodes.	In B-tree, searching becomes difficult as data cannot be found in the leaf node.
8.	Search accessibility	In B^+ tree, the searching becomes easy.	In B-tree, the search is not that easy as compared to a B^+ tree.
9.	Redundant key	They store redundant search key.	They do not store redundant search key.

Example :

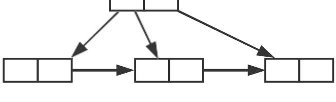
B^+ tree :

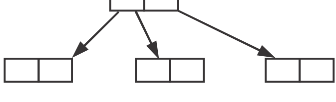
3 5

B-tree :

3 5

1 2
6 4
8 9
1 2
6 4
8 9





Que 5.32. **Write a short note on binary heaps.**

Answer

1. The binary heap data structure is an array that can be viewed as a complete binary tree.
2. Each node of the binary tree corresponds to an element of the array.
3. The array is completely filled on all levels except possibly lowest.
4. We represent heaps in level order, going from left to right.
5. If an array A contains key values of nodes in a heap, length [*A*] is the total number of elements.

   Heap-size [*A*] = Length [*A*] = Number of elements.
6. The root of the tree A[1] and given index i of a node the indices of its parent, left child and right child can be computed :

```
PARENT (i)
return floor (i/2)
LEFT(i)
return 2i
RIGHT (i)
return 2i + 1
```

VERY IMPORTANT QUESTIONS

Following questions are very important. These questionsmay be asked in your SESSIONALS as well as UNIVERSITY EXAMINATION.

Q. 1. Write a C program to implement binary tree insertion, deletion with example.

Ans. Refer Q. 5.5.

Q. 2. Write the C program for various traversing techniques of binary tree with neat example.

Ans. Refer Q. 5.6.

Q. 3. Explain binary search tree and its operations. Make a binarysearch tree for the following sequence of numbers, show all steps : 45, 32, 90, 34, 68, 72, 15, 24, 30, 66, 11, 50, 10.

Ans. Refer Q. 5.7.

Q. 4. Define binary search tree. Create BST for the following data, show all steps :
20, 10, 25, 5, 15, 22, 30, 3, 14, 13

Ans. Refer Q. 5.8.

Q. 5. Define tree, binary tree, complete binary tree and full binarytree. Write algorithm or function to obtain traversals of a binary tree in preorder, postorder and inorder.

Ans. Refer Q. 5.10.

Q. 6. Construct a binary tree for the following :Inorder : *Q, B, K, C, F, A, G, P, E, D, H, R* Preorder : *G, B, Q, A, C, K, F, P, D, E, R, H* Find the postorder of the tree.

Ans. Refer Q. 5.11.

Q. 7. Draw a binary tree with following traversal :Inorder : *D B H E A I F J C G*
Preorder : *A B D E H C F I J G*

Ans. Refer Q. 5.12.

Q. 8. Draw a binary tree with following traversals :Inorder : *B C A E G D H F I J*
Preorder : *A B C D E F G H I J*

Ans. Refer Q. 5.13.

Q. 9. What is a threaded binary tree ? Explain the advantages of using a threaded binary tree.

Ans. Refer Q. 5.19.

Q. 10. What is Huffman tree ? Create a Huffman tree with following numbers :
24, 55, 13, 67, 88, 36, 17, 61, 24, 76

Ans. Refer Q. 5.21.

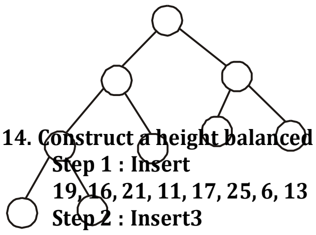
Q. 11. Explain Huffman algorithm. Construct Huffman tree for *MAHARASHTRA* with its optimal code.

Ans. Refer Q. 5.22.

Q. 12. Define AVL trees. Explain its rotation operations with example. Construct an AVL tree with the values 10 to 1 numbers into an initially empty tree.

Ans. Refer Q. 5.23.

Q. 13. Consider the following AVL tree and insert 2, 12, 7 and 10 asnew node. Show proper rotation to maintain the tree as AVL.



Q. 14. Construct a height balanced binary search tree by performing following operations :

Step 1 : Insert

19, 16, 21, 11, 17, 25, 6, 13

Step 2 : Insert3

Step 3 : Delete16

Ans. Refer Q. 5.26.

Q. 15. What is height balanced tree ? Why height balancing of treeis required ? Create an AVL tree for the following elements : ***a, z, b, y, c, x, d, w, e, v, f.***

Ans. Refer Q. 5.25.

Q. 16. Describe all rotations in AVL tree. Construct AVL tree fromthe following nodes : ***B, C, G, E, F, D, A.***

Ans. Refer Q. 5.27.

Q. 17. Define a B-tree. What are the applications of B-tree ? Drawa B-tree of order 4 by insertion of the following keys in order : ***Z, U, A, I, W, L, P, X, C, J, D, M, T, B, Q, E, H, S, K, N, R,G, Y, F, O, V.***

Ans. Refer Q. 5.28.

Q. 18. Construct a B-tree of order 5 created by inserting the following elements 3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20, 26, 4, 16, 18, 24, 25, 19. Also delete elements 6, 23 and 3 from theconstructed tree.

Ans. Refer Q. 5.29.

Q. 19. Construct a B-tree on following sequence of inputs.10, 20, 30, 40, 50, 60, 70, 80, 90

Assume that the order of the B-tree is 3.

Ans. Refer Q. 5.30.

☺☺☺