Chapter 5

Trees

LEARNING OBJECTIVES

- *Tree*
- *2-Tree*
- *Binary tree*
- *Properties of binary trees*
- *Complete binary tree*
- *Full binary tree*
- *Binary tree representation*
- *Linked representation*
- *Binary search tree*
- *Binary tree traversing methods*
- *AVL tree*
- *Binary heap*
- *Max-heap*
- *Min-heap*
- *Expression tree*

Tree

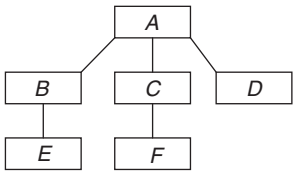
Tree is non-linear data structure designated at a special node called root and elements are arranged in levels without containing cycles.

(or)

The tree is

1. Rooted at one vertex
2. Contains no cycles
3. There is a sequence of edges from any vertex to any other
4. Any number of elements may connect to any node (including root)
5. A unique path traverses from root to any node of tree
6. Tree stores data in hierarchical manner
7. The elements are arranged in layers

Example:



- Root node is *A*.
- *A*'s children are *B*, *C* and *D*.
- *E*, *F* and *D* are leaves.
- Nodes *B*, *C* are called as intermediate nodes.
- *A* is parent of *B*, *C* and *D*.
- *B* is parent of *E* and *C* is parent of *F*.
- Number of children of a node is called degree of node.

2-Tree

A tree in which every node contains either 0 or 2 children.

Binary Tree

It is a special type of tree where each node of tree contains either 0 or 1 or 2 children.

(or)

Binary Tree is either empty, or it consists of a root with two binary trees called left-sub tree and right sub-tree of root (left or right or both the sub trees may be empty).

Properties of binary tree

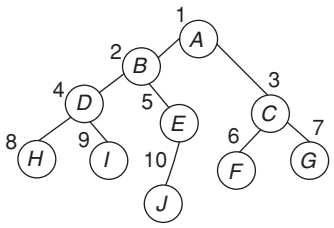
- Binary tree partitioned into three parts.
- First subset contains root of tree.
- Second subset is called left subtree.
- Another subset is called right subtree.
- Each subtree is a binary tree.
- Degree of any node is 0/1/2.
- The maximum number of nodes in a tree with height '*h*' is $2^{h+1}-1$.
- The maximum number of nodes at level '*i*' is 2^{i-1}.
- For any non-empty binary tree, the number of terminal nodes with n_2, nodes of degree 2 is $N_0 = n_2 + 1$
- The maximum number of nodes in a tree with depth *d* is $2^d - 1$.

Types of binary tree

Complete binary tree It is a binary tree, in which at every level, except possibly the last, is completely filled and all nodes at the last level are as left as possible.

Example:

Level	Height	Depth
1	3	1
2	2	2
3	1	3
4	0	4



For the given tree:

- Having 4 levels
- Height of the tree is 3
- Depth of the tree is 4
- The numbers at each node represents level order index.
- The level order Index, are assigned to nodes in the following manner
- Root of the tree is '1'
- For a node '*x*', the LOI is (2 * LOI (parent)), if '*x*' is left child of its parent.
- For a node '*y*', the LOI (2 * LOI (Parent) +1), if '*y*' is right child of its parent.

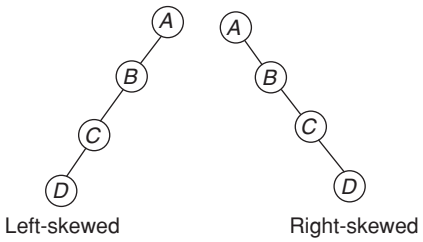
Now complete binary tree can be defined as a binary tree, which contains a sequence of numbers to its nodes as LOI's without any break in sequence.

Full binary tree It is a binary tree, for which all leaf nodes are at same level and all intermediate nodes contains exactly 2 children.
(or)
A tree with depth '*K*' contains exactly $2^K - 1$ nodes.

Strictly binary tree A binary tree in which every node contains exactly 0 or 2 children.

Skewed binary tree A binary tree in which elements are added only in one direction.

Example:



Application

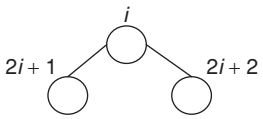
- A binary tree is useful data structure when two way decisions must be made at each point of process.

Binary tree representation

The binary trees can be represented in two ways.

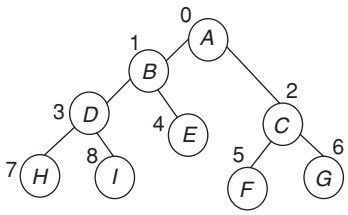
- Array
- Linked list

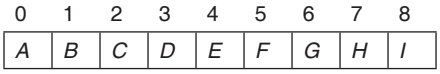
Array representation The elements of a binary tree are placed in an array using the level order index of each element.



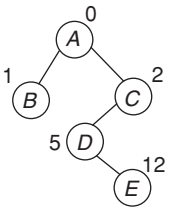
When LOI of Root is 0:

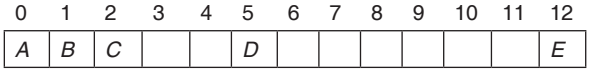
Example 1:





Example 2:

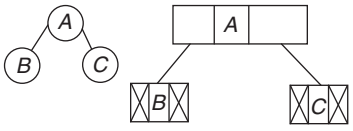




Linked representation Each node contains one data field and two link fields. Fist link point to the left child and another point to the right child.

In absence of any child, corresponding link field contains NULL.

Example:



Trade-off's between array and linked, representations

- Array representation is somewhat simpler. It must ensure elements are placed in array at proper position.
- Linked representation requires pointer to its left and right child.
- Array representation saves memory for almost complete binary trees.
- Linked representation allocates the number and nodes equal to the number of elements in tree.
- Array representation does not work efficiently for skewed binary trees.
- Array representation limits the size of binary tree to the array size.
- In linked representation, tree can be extended by adding an element dynamically and can be shrinked by deleting an element dynamically.

Binary search tree

It is a special type of binary tree that satisfies the following properties.

- All the elements of left sub tree of root are smaller than root.
- All the elements of right sub tree of root are greater than root.
- The above two properties satisfy for each subtree.

Example:

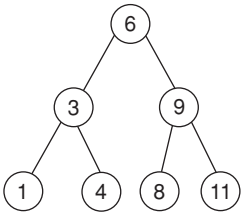


Figure 1 A data structure to encode binary search tree

The binary search tree node contains three fields, data field, left child, right child. Left child is a pointer which points to the predecessor of the node and right child is a pointer which points to the successor of the node.

A data structure to encode binary search tree is

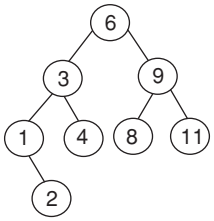
Left child	Data	Right child

The declaration is

```
Struct node
{
Struct node * left child;
Int data;
Struct node * Right child;
};
```

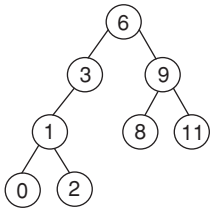
Insertion If a value to be inserted is smaller than the root, value, it must go in the left subtree, if larger it must go in the right subtree. This reasoning applies recursively until we reach a node where the required subtree does not exist and that is where we place the new value.

Example: It must go in 6's left subtree, 3's left subtree, 1's right subtree, 1 has no right subtree, so we make a singleton with 2 and it becomes 1's right subtree.

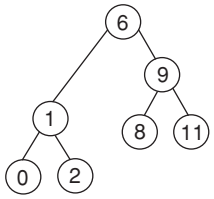


Deletion:

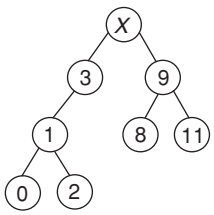
1. If a leaf node has to be deleted, just delete it and the rest of the tree is exactly as it was, so it is still a BST.
2. Suppose the node we are deleting has only one sub tree
   Example, In the following tree, '3' has only one sub-tree



To delete a node with 1 subtree, we just 'link past' the node, i.e., connect the parent of the node directly to the node's only subtree. This always works, whether the one subtree is on the left or on the right. Deleting 3 gives us.



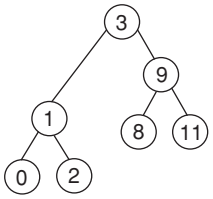
3. Deletion of node which has 2 subtrees
   Example: Delete 6.



Choose value '*X*'

1. Everything in the left subtree must be smaller than *X*.
2. Everything in the right subtree must be bigger than *X*.

We must choose *X* to be the largest value in the left subtree. In our example, 3 is the largest value in the left subtree. So we replace root node 6 with 3.



Note: We could do the same thing with the right subtree. Just use the smallest value in the right subtree.

Notes:

- The largest element in left subtree is the right most element.
- The smallest element in right subtree is the left most element.

Binary tree traversing methods

The binary tree contains 3 parts:

V – root

L – Left subtree

R – Right subtree

Pre-order: (*V*, *L*, *R*)

- Visit root of the tree first
- Traverse the left - subtree in pre-order
- Traverse the right - subtree in preorder

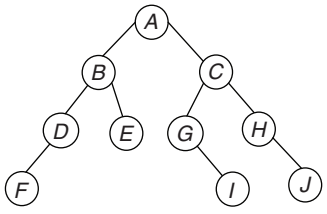
In-order: (*L*, *V*, *R*)

- Traverse the left – subtree in in-order
- Visit Root of the tree
- Traverse right - sub tree in in-order

Post-order: (*L*, *R*, *V*)

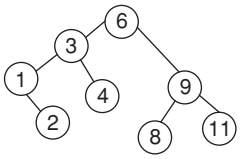
- Traverse the left subtree in post-order.
- Traverse the Right - subtree in post-order
- Visit root of the tree

Example 1:



Pre-order: *A B D F E C G I H J*
In-order: *F D B E A G I C H J*
Post-order: *F D E B I G J H C A*
Pre-order, In-order and post-order uniquely identify the tree.

Example 2:



Pre-order: 6 3 1 2 4 9 8 11
In-order: 1 2 3 4 6 8 9 11
Post-order: 2 1 4 3 8 11 9 6

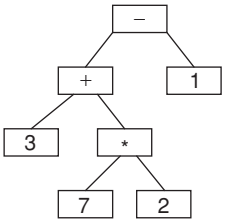
Points to remember

- Pre-order traversal contains root element as first element in traverse list.
- Post-order traversal contains root element as last in traversal list.
- For BST, in-order traversal is a sorted list.
- A unique binary tree can constructed if either pre-order or post-order traversal list provided with In order traversal list.
- If either pre-order or post-order only given then BST cannot be constructed.

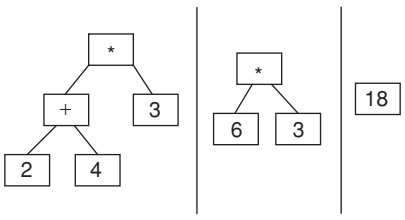
Applications

1. Binary trees can represent arithmetic expressions.
   - An infix expression will have a parent operator and two children operands.

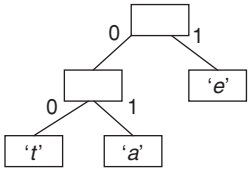
Consider the expression ((3 + (7 ∗ 2)) −1)
Each parenthesised expression becomes a tree.
Each operand is a leaf, each operator is an internal node.



2. To evaluate the expression tree:
   Take any two leaves
   Apply the parents operator to them
   Replace the operator with the value of the sub expression.



3. Binary trees in a famous file compression algorithm Huffman coding tree
   - Each character is stored in a leaf
   - The code is found by following the path 0 go left, 1 go right.
   - *a* is 01
   - *e* is 1



AVL Tree

An AVL tree is a self-balancing binary search tree, in which the heights of the two child subtrees of any node differ by atmost one.

Insertions and deletions may require the tree to be rebalanced by one or more tree rotations.

- The balance factor of a node is the height of its left subtree minus the height of its right subtree (sometimes opposite) and a node with balance factor—1, 0 or –1 is considered balanced. A node with any other balance factor is considered unbalanced and requires rebalancing the tree.
- The balance factor is either stored directly at each node or computed from the heights of the subtrees.

Insert operations

Step I: Insert a node into the AVL tree as it is inserted in a BST.

Step II: Examine the search path to see if there is a pivot node.

Three cases may arise

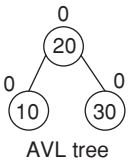
Case I: There is no pivot node. No adjustment required.

Case II: The pivot node exists and the subtree of the pivot node to which the new node is added has smaller height. No adjustment required.

Case III: The pivot node exists and the subtree to which the new node is added has the larger height, Adjustment required.

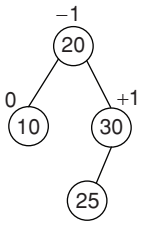
Example: The numbers at each node represents balance factor.

Example 1:



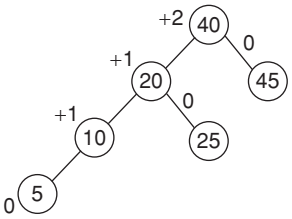
AVL tree

Example 2:



AVL tree

Example 3:



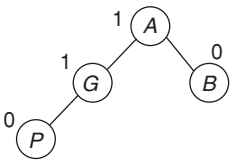
Not an AVL tree

Example 3 is not an AVL tree, because the balance factor of root node is +2.

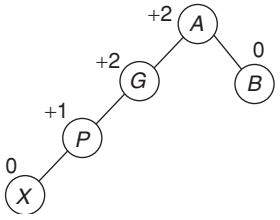
AVL tree becomes height in-balanced tree in following cases:

1. Left-Left case: An insertion in left subtree of left child of pivot node.

Example:

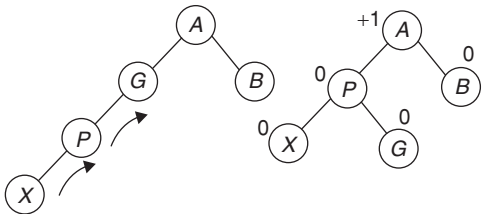


Insert '*X*' as left to node '*P*'. Here '*G*' is pivot node.



Solution:

To make the tree as balanced tree, perform **Left–Left Rotation** as follows:

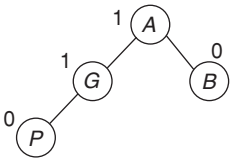


In left–left rotation

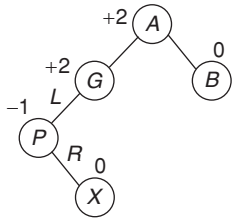
- Intermediate node '*P*' becomes root of subtree.
- Root of subtree '*G*' (pivot) becomes right subtree.
- New node '*X*' remains same as left child of '*P*'.

Left–Right Case

An insertion of left subtree of right child of pivot node.

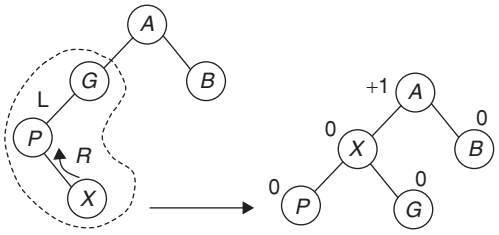


Example 1: Insert '*X*' as right child of '*P*'.



Is not an AVL tree. Height in-balance at node '*G*'.

Solution:
Perform Left–Right Rotation, to balance the height of tree.



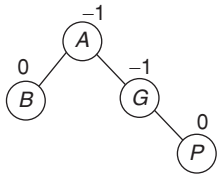
In Left–Right rotation:

- New node '*X*' becomes root of subtree.
- Root of subtree '*G*' (pivot) becomes right child of '*X*'.
- Intermediate node '*P*' becomes left child of new node.

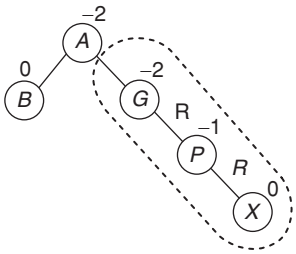
Right–Right case

An insertion of right subtree of right child of pivot node.

Example:

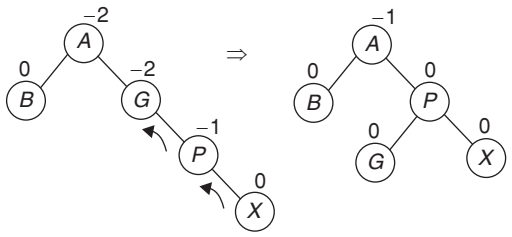


Insert '*X*' as right child of '*P*'



Is not an AVL tree, because of height in-balance at node '*G*'.

Solution:
To make the tree as balanced tree, perform the right–right rotation as follows:

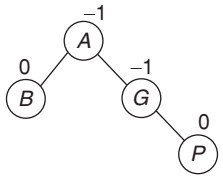


In Right–Right rotation:

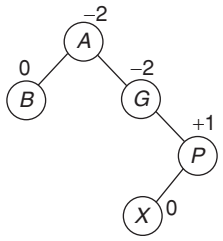
- Intermediate node '*P*' becomes root of subtree.
- Root of subtree '*G*' (pivot) becomes left child of '*P*'.
- New node '*X*' remains as right child to '*P*'.

Right–Left case

An insertion of right subtree of left child of pivot node.

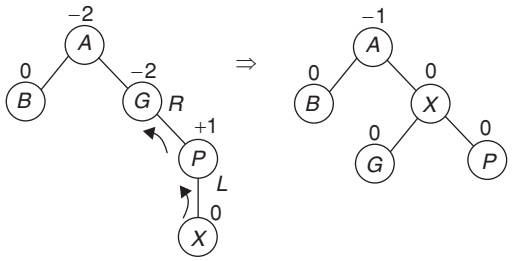


Insert '*X*' as left child of '*P*'



Is not AVL tree, because height in-balance at node '*G*'.

Solution:
To make the above tree as balanced, perform Right–Left rotation as follows:



In Right–Left Rotation:

- New node '*X*' becomes root of subtree.
- Root of subtree '*G*' (pivot) becomes left child of '*X*'.
- Intermediate node '*P*' becomes right of '*X*'.

Note: Left–Right and Right–Left rotation are also called as double rotations.

Binary Heap

A binary heap is a heap data structure created using a binary tree. It can be seen as a binary tree with two additional constraints.

The shape property: The tree is a complete binary tree; that is, all levels of the tree, except possibly the last one (deepest) level of the tree is not complete, the nodes of that level are filled, from left to right.

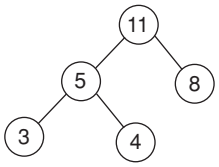
Max-Heap

A heap in which each node is greater than or equal to its children is called max-heap. Max-Heap generally used for heap sort.

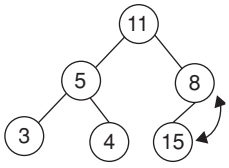
Min-Heap

A heap in which, each node is smaller than or equal to its children is called Min-Heap. Min-heap generally used to implement priority queue.

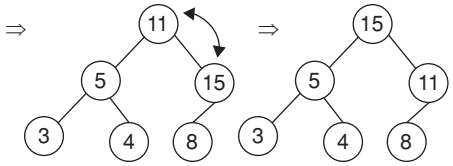
Note: By default heap represent Max-Heap:



Insert 15:

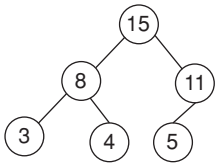


Is not satisfying heap property. So Heapify

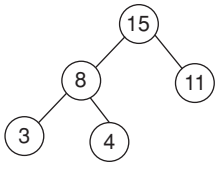


Delete 5: Deletion of a node from heap is always deletes a leaf node.

So interchange the value of last leaf node with node 5.



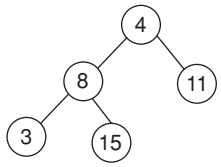
Now delete node '5'



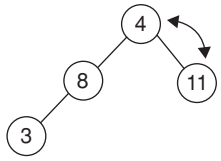
Is satisfying heap property.

Delete 15:

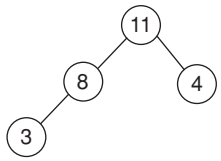
Interchange 4 and 15



Now delete Node '15'



Is not satisfying heap property. So heapify



Note: Insertion or deletion operation on a heap may require heapify process.

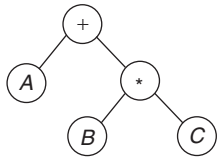
Expression Tree

The expressions can also represented by using a binary tree called expression tree.

Expression tree contains:

- Operators as intermediate nodes.
- Operands as leaf nodes (or) childs to operator nodes.
- The operator at lowest level will be having highest priority.

Example: $A + B * C$



Traversing:

Pre-order: $+ A * B C$

In-order: $A + B * C$

Post-order: $A B C * +$

Note: In-order traversal of expression tree generates In-fix expression. Similarly pre-order and post-order generates prefix and postfix, respectively.

EXERCISES

Practice Problems I

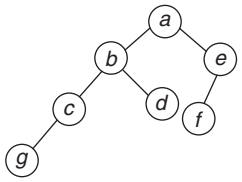
Directions for questions 1 to 15: Select the correct alternative from the given choices.

1. A binary tree T has n leaf nodes. The number of nodes of degree two in T is ____.
(A) n (B) $n - 1$
(C) $\log n$ (D) $n + 1$

2. How many numbers of binary tree can be created with 3 nodes which when traversed in post-order gives the sequence C, B, A?
(A) 3 (B) 5
(C) 8 (D) 15

3. A binary search tree contains the values 3, 6, 10, 22, 25, 30, 60, 75. The tree is traversed in pre-order and the values are printed out. Which of the following sequence is a valid output?
(A) 25 6 3 10 22 60 30 75
(B) 25 6 10 3 22 75 30 60
(C) 25 6 75 60 30 3 10 22
(D) 75 30 60 22 10 3 6 25

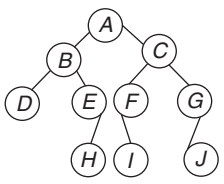
4. Figure shows a balanced tree. How many nodes will become unbalanced when a node is inserted as a child of the node 'g'?



(A) 7 (B) 2
(C) 3 (D) 8

5. A full binary tree with n non-leaf nodes contains
(A) $2n$ nodes (B) $\log_2 n$ node
(C) $n + 1$ nodes (D) $2n + 1$ nodes

6. Which of the following list of nodes corresponds to a post order traversal of the binary tree shown below?



(A) *A B C D E F G H I J* (B) *J I H G F E D C B A*
(C) *D H E B I F J G C A* (D) *D E H F I G J B C A*

7. Which of the following sequence of array elements forms as heap?
(A) {23, 17, 14, 6, 13, 10, 1, 12, 7, 5}
(B) {23, 17, 14, 6, 13, 10, 1, 5, 7, 12}
(C) {23, 17, 14, 7, 13, 10, 1, 5, 6, 12}
(D) {23, 17, 14, 7, 13, 10, 1, 12, 5, 6}

8. What is the maximum height of any AVL tree with 7 nodes? Assume that the height of a tree with a single node is 0.
(A) 2 (B) 3
(C) 4 (D) 5

9. A binary search tree is generated by inserting in order the following integers:

55, 15, 65, 5, 25, 59, 90, 2, 7, 35, 60, 23.

The number of nodes in the left subtree and right subtree of the root respectively are
(A) 8, 3 (B) 7, 4
(C) 3, 8 (D) 4, 7

10. In a complete binary tree of n nodes, how far are the most distant two nodes? Assume each in the path counts as 1.
(A) about $\log_2 n$ (B) about $2\log_2 n$
(C) about $3\log_2 n$ (D) about $4\log_2 (n)$

11. A complete binary tree of level 5 has how many nodes?
(A) 20 (B) 63
(C) 30 (D) 73

Common data for questions 12 and 13: A 3-ary max-heap is like a binary max-heap, but instead of 2 children, nodes have 3 children. A 3-ary heap can be represented by an array as follows:

The root is stored in the first location, $a[0]$, nodes in the next level from left to right is stored from $a[1]$ to $a[3]$ and so on. An item x can be inserted into a 3-ary heap containing n items by placing x in the location $a[n]$ and pushing it up the tree to satisfy the heap property.

12. Which one of the following is a valid sequence of elements in an array representing 3-ary max-heap?
(A) 1, 3, 5, 6, 8, 9 (B) 9, 6, 3, 1, 8 , 5
(C) 9, 3, 6, 8, 5, 1 (D) 9, 5, 6, 8, 3, 1

13. Suppose the elements 7, 2, 10 and 4 are inserted, in that order, into the valid 3-ary max-heap found in the above question. Which one of the following is the sequence of items in the array representing the resultant heap?
(A) 10, 7, 9, 8, 3, 1, 5, 2, 6, 4
(B) 10, 9, 8, 7, 6, 5, 4, 3, 2, 1
(C) 10, 9, 4, 5, 7, 6, 8, 2, 1, 3
(D) 10, 8, 6, 9, 7, 2, 3, 4, 1, 5

14. Consider the nested representation of binary trees : (X Y Z) indicated Y and Z are the left and right subtrees respectively, of node X(Y and Z may be null (or) further nested) which of the following represents a valid binary tree?

(A) (1 2 (4 5 6 7)) (B) (1(2 3 4)5 6)7
(C) (1(2 3 4) (5 6 7)) (D) (1(2 3 NULL)(4 5))

15. A scheme for storing binary trees in an array X is as follows:

Indexing of X starts at 1 instead of 0. The root is stored at $X[\ 1\ ]$. For a node stored at $X[i]$, the left child, if any, is stored in $X[\ 2i\ ]$ and the right child, if any, in $X[2i + 1]$. To store any binary tree on 'n' vertices the minimum size of X should be

(A) $2n$ (B) n
(C) $3n$ (D) n^2

Practice Problems 2

Directions for questions 1 to 15: Select the correct alternative from the given choices.

1. A binary search tree contains the values 1, 2, 3, 4, 5, 6, 7, 8. The tree is traversed in pre-order and the values are printed out. Which of the following is a valid output?

(A) 53124786 (B) 53126487
(C) 53241678 (D) 53124768

2. A binary search tree is generated by inserting in order the following integers : 50, 15, 62, 5, 20, 58, 91, 3, 8, 37, 60, 24. The number of nodes in the left subtree and right subtree of the root respectively are:

(A) (4, 7) (B) (7, 4)
(C) (8, 3) (D) (3, 8)

3. A full binary tree (with root at level 0) of height h has a total number of nodes equal to:

(A) 2^h (B) $2^{h+1} - 1$
(C) $2^h - 1$ (D) 2^{h-1}

4. The number of null pointers of a binary tree of n nodes is :

(A) $n + 1$ (B) $n(n + 1)$
(C) n^2 (D) $2n$

5. Which of the following is false?

(A) A tree with n nodes has $(n - 1)$ edges.
(B) A labeled rooted binary tree can be uniquely constructed, given its post-order, in-order traversal results.
(C) The complete binary tree with n internal nodes has $(n + 1)$ leaves.
(D) The maximum number of nodes in a binary tree of height h is $(2^{h+1} - 1)$.

6. The maximum number of nodes in a binary tree at level i is

(A) 2^i (B) $2^i - 1$
(C) $2^i + 1$ (D) $\log_2 i + 1$

7. The number of leaf nodes in a rooted tree of n nodes, with each node having 0 or 3 children is

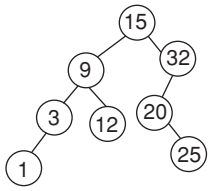
(A) $\frac{n}{3}$ (B) $\frac{(n-1)}{3}$
(C) $\frac{(n-1)}{2}$ (D) $\frac{(2n+1)}{3}$

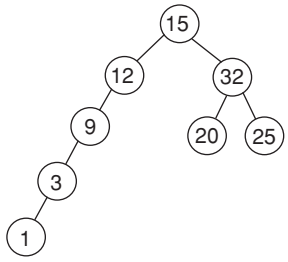
8. A complete n-ary tree is one in which every node has 0 or n children. If x is the number of internal nodes of a complete n-ary tree, the number of leaves in it is given by

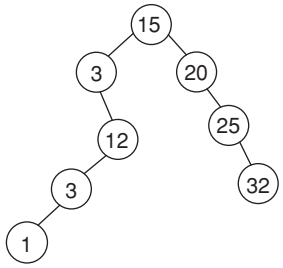
(A) $x(n - 1) + 1$
(B) $xn + 1$
(C) $xn - 1$
(D) $x(n + 1) - 1$

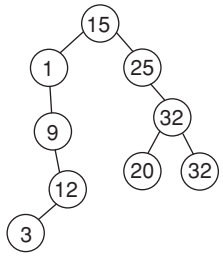
Common data for questions 9 and 10:

9. Insert the keys into a binary search tree in the order specified 15, 32, 20, 9, 3, 25, 12, 1. Which one of the following is the binary search tree after insertion of all elements?

(A)


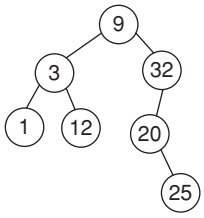
(B)


(C)


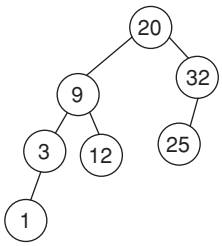
(D)


10. Which of the following is the binary tree after deleting 15?

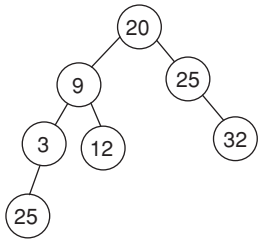
(A)



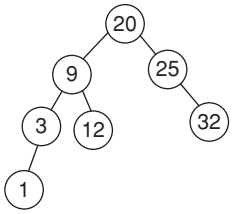
(B)



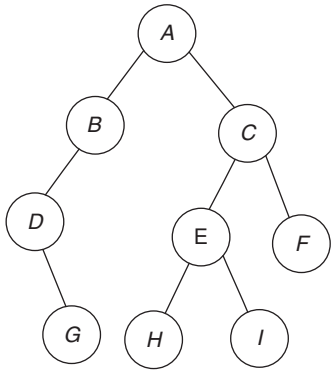
(C)



(D)



For questions 11, 12 and 13 below, use this figure



11. What is the post-order expression?
(A) ABDGCEJHIF (B) GDBHIEFCA
(C) DGBAHEICF (D) ABHIEFCDG

12. What is the pre-order expression?
(A) ABDGCEHIF (B) ABHIEFCDG
(C) DGBAHEIFCF (D) GDBHIEFCA

13. What is the in-order expression?
(A) ABDGCEHIF (B) GDBHIEFCA
(C) DGBAHEICF (D) ABHIEFCDG

14. In a 3-ary tree every internal node has exactly 3 children. The number of leaf nodes in such a tree with 6 internal nodes will be
(A) 13 (B) 12 (C) 11 (D) 10

15. Minimum number of swaps needed to convert the array 89, 19, 14, 40, 17, 12, 10, 2, 5, 7, 11, 6, 9, 70 into a max heap
(A) 2 (B) 3 (C) 1 (D) 0

Previous Years' Questions

1. In a binary tree with n nodes, every node has an odd number of descendants. Every node is considered to be its own descendant. What is the number of nodes in the tree that have exactly one child? **[2010]**
(A) 0 (B) 1
(C) $(n-1)/2$ (D) $n-1$

2. The following C function takes a singly-linked list as input argument. It modifies the list by moving the last element to the front of the list and returns the modified list. Some part of the code is left blank.

```
typedef struct node {
int value;
struct node *next;
}     Node;
Node *move_to_front(Node *head) {
Node *p, *q;
if ((head = = NULL || (head->next = = NULL)) return head;
q = NULL; p = head;
while (p-> next !=NULL) {
                q = p;
                p = p->next;
}

return head;
}
```

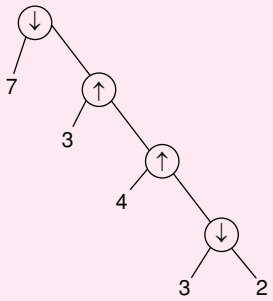
Choose the correct alternative to replace the blank line. **[2010]**
(A) `q = NULL; p->next = head; head = p;`
(B) `q->next = NULL; head = p; p->next = head;`
(C) `head = p; p->next = q; q->next = NULL;`
(D) `q->next = NULL; p->next = head; head = p;`

3. Consider two binary operators '↑' and '↓' with the

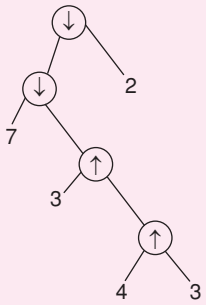
precedence of operator ↓ being lower than that of the operator ↑. Operator ↑ is right associative while operator ↓ is left associative. Which one of the following represents the parse tree for expression (7 ↓ 3 ↑ 4 ↑ 3 ↓ 2)?

[2011]

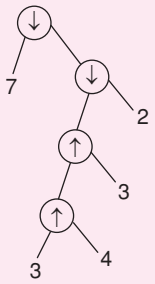
(A)



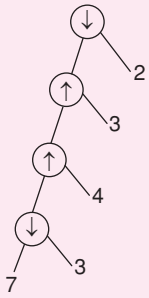
(B)



(C)



(D)



4. The height of a tree is defined as the number of edges on the longest path in the tree. The function shown in the pseudocode below is invoked as "height(root)" to compute the height of a binary tree rooted at the tree pointer "root".

```
int height(treeptr n)
{if (n == NULL) return -1;
if (n → left ==NULL)
if (n → right == NULL) return 0;
else return [B1] ;        //Box 1
else {h1 = height (n → left);
if (n → right== NULL) return (1 + h1);
else {h2 = height (n → right);
return [B2] ;      //Box 2
          }
    }
}
```

The appropriate expressions for the two boxes B_1 and B_2 are **[2012]**

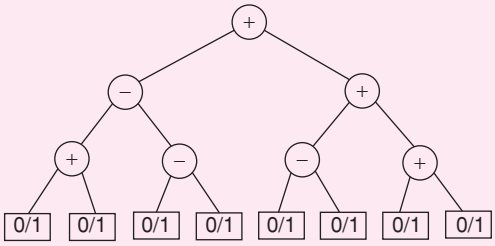
(A) B_1: (1 + height ($n \rightarrow$ right))
B_2: (1 + max (h_1, h_2))

(B) B_1: (height ($n \rightarrow$ right))
B_2: (1+ max(h_1, h_2))

(C) B_1: height ($n \rightarrow$ right)
B_2: max(h_1, h_2)

(D) B_1: (1 + height ($n \rightarrow$ right))
B_2: max(h_1, h_2)

5. Consider the expression tree shown. Each leaf represents a numerical value, which can either be 0 or 1. Over all possible choices of the values at the leaves, the maximum possible value of expression represented by the tree is ————. **[2014]**



6. Consider the pseudocode given below. The function **Dosomething()** takes as argument a pointer to the root of an arbitrary tree represented by the *leftMostChild-rightSibling* representation. Each node of the tree is of type **treenode.** **[2014]**

```
type def struct treeNode* treeptr;
struct treeNode
{
treeptr leftMostChild, rightSibling;
};
int Dosomething (treeptr tree)
```

```
{
int value = 0;
if (tree ! = NULL){
if (tree - > leftMostChild = = NULL)
value = 1;
else
value = Dosomething (tree - > leftMostChild);
value = value +Dosomething (tree - > right
Sibling);
}
return (value);
}
```

When the pointer to the root of a tree is passed as the argument to **DoSomething**, the value returned by the function corresponds to the

(A) Number of internal nodes in the tree
(B) Height of the tree
(C) Number of nodes without a right sibling in the tree
(D) Number of leaf nodes in the tree

7. The height of a tree is the length of the longest root-to-leaf path in it. The maximum and minimum number of nodes in a binary tree of height 5 are **[2015]**

(A) 63 and 6, respectively
(B) 64 and 5, respectively
(C) 32 and 6, respectively
(D) 31 and 5, respectively

8. Which of the following is/are correct inorder traversal sequence(s) of binary search tree(s)? **[2015]**

I. 3, 5, 7, 8, 15, 19, 25
II. 5, 8, 9, 12, 10, 15, 25
III. 2, 7, 10, 8, 14, 16, 20
IV. 4, 6, 7, 9, 18, 20, 25

(A) I and IV only (B) II and III only
(C) II and IV only (D) II only

9. Consider a max heap, represented by the array: 40, 30, 20, 10, 15, 16, 17, 8, 4 **[2015]**

Array Index	1	2	3	4	5	6	7	8	9
Value	40	30	20	10	15	16	17	8	4

Now consider that a value 35 is inserted into this heap. After insertion, the new heap is

(A) 40, 30, 20, 10, 15, 16, 17, 8, 4, 35
(B) 40, 35, 20, 10, 30, 16, 17, 8, 4, 15
(C) 40, 30, 20, 10, 35, 16, 17, 8, 4, 15
(D) 40, 35, 20, 10, 15, 16, 17, 8, 4, 30

10. A binary tree T has 20 leaves. The number of nodes in T having two children is _____ **[2015]**

11. Consider a binary tree T that has 200 leaf nodes. Then, the number of nodes in T that have exactly two children are _____. **[2015]**

12. While inserting the elements 71, 65, 84, 69, 67, 83 in an empty binary search tree (BST) in the sequence shown, the element in the lowest level is **[2015]**

(A) 65 (B) 67
(C) 69 (D) 83

13. Consider the following New–order strategy for traversing a binary tree: **[2016]**

- Visit the root;
- Visit the right subtree using New – order;
- Visit the left subtree using New – order;

The New – order traversal of the expression tree corresponding to the reverse polish expression

3 4 * 5 – 2 ^ 6 7 * 1 + – is given by:

(A) + – 1 6 7 * 2 ^ 5 – 3 4 *
(B) – + 1 * 6 7 ^ 2 – 5 * 3 4
(C) – + 1 * 7 6 ^ 2 – 5 * 4 3
(D) 1 7 6 * + 2 5 4 3 * – ∧ –

14. Let T be a binary search tree with 15 nodes. The minimum and maximum possible heights of T are: **[2017]**

Note: The height of a tree with a single node is 0.

(A) 4 and 15 respectively
(B) 3 and 14 respectively
(C) 4 and 14 respectively
(D) 3 and 15 respectively

15. The pre-order traversal of a binary search tree is given by 12,8,6,2,7,9,10,16,15,19,17,20. Then the post-order traversal of this tree is: **[2017]**

(A) 2, 6, 7, 8, 9, 10, 12, 15, 16, 17, 19, 20
(B) 2, 7, 6, 10, 9, 8, 15, 17, 20, 19, 16, 12
(C) 7, 2, 6, 8, 9,10, 20, 17, 19, 15, 16, 12
(D) 7, 6, 2, 10, 9, 8, 15, 16, 17, 20, 19, 12

16. The postorder traversal of a binary tree is 8, 9, 6, 7, 4, 5, 2, 3, 1. The inorder traversal of the same tree is 8, 6, 9, 4, 7, 2, 5, 1, 3. The height of a tree is the length of the longest path from the root to any leaf. The height of the binary tree above is _____. **[2018]**

17. The number of possible min-heaps containing each value from {1, 2, 3, 4, 5, 6, 7} exactly once is _____. **[2018]**

Answer Keys

Exercises

Practice Problems 1

1. B	**2.** B	**3.** A	**4.** C	**5.** D	**6.** C	**7.** C	**8.** B	**9.** B	**10.** B
11. B	**12.** D	**13.** A	**14.** C	**15.** A					

Practice Problems 2

1. D	**2.** B	**3.** B	**4.** A	**5.** C	**6.** B	**7.** D	**8.** A	**9.** A	**10.** B
11. B	**12.** A	**13.** C	**14.** A	**15.** B					

Previous Years' Questions

1. A	**2.** D	**3.** B	**4.** A	**5.** 6	**6.** D	**7.** A	**8.** A	**9.** B	**10.** 19
11. 199	**12.** B	**13.** C	**14.** B	**15.** B	**16.** 4	**17.** 80			