NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

CS 201 – DATA STRUCTURES LAB

Instructors: Faizan Yousuf & Abdul Aziz

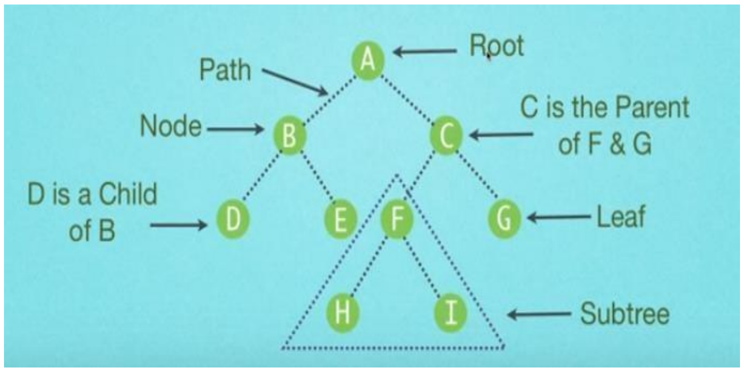
LAB SESSION # 05

Outline

- Tree
- Binary
- Binary Search Tree
- Depth First Traversing
- Breadth First Traversing
- Exercise

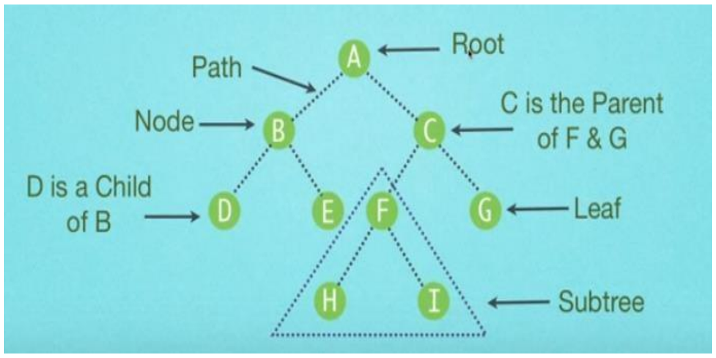
TREE

A tree is a structure in which each node can have multiple successors (unlike the linear structures that we have been studying so far, in which each node always had at most one successor). The graphical representation of a tree is given below.



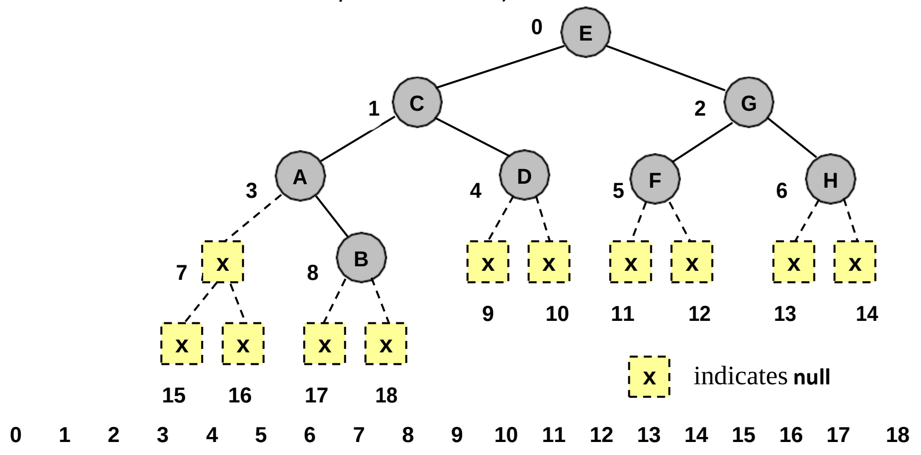
BINARY TREE

A tree is a structure in which each node can have multiple successors (unlike the linear structures that we have been studying so far, in which each node always had at most one successor). The graphical representation of a tree is given below.



The `buildTree()` method

This is a recursive method that builds a linked binary tree. It uses input node data which is represented as a linear array (**`null`** links are added to complete the tree).



E	C	G	A	D	F	H	x	B	x	x	x	x	x	x	x	x	x	x

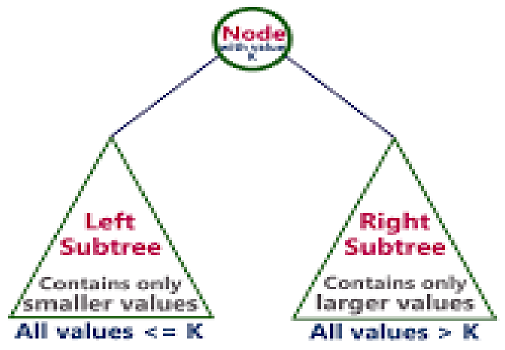
The buildTree() method takes the index of the array as an input parameter. Initially, the indexis

0 (root node). The method calls recursively with updated index – for left child, the index is (2*index+1) and for right child, it is (2*index+2). The routine in Java is as follows:

```
public Node buildTree( int index )
{     Node p = null;                          // ‘p’  refers to current node
      if( tree[index] != null )
      {     p = new Node(tree[index]);        // create a node from array data
            // call buildTree() to create a left node
            p.left = buildTree(2*index+1);
            // call buildTree() to create a right node
            p.right = buildTree(2*index+2);
      }
      return p;
}
```

BINARY SEARCH TREE (BST)

A binary search tree is a binary tree with additional properties, in binary search tree each node has a comparable key (and an associated value) and satisfies the restriction that the key in any node is larger than the keys in all nodes in that node’s left sub tree and smaller than the keys in all nodes in that node’s right sub tree. A graphical representation of binary search tree is given below.



Basic implementation of BST

Each node of binary tree contains the following information:

- A value (user’s data)
- A link to the left child
- A link to the right child

In some implementations, node may store a link to the parent, but it depends on algorithm, programmer want to apply to BST. For basic operations, like addition, removal and search a link to the parent is not necessary. It is needed in order to implement iterators.

Operations on a BST

Following are the basic operations which can be applied to a binary search tree

- ✓ Add/ Insert a new value
- ✓ Search for a value
- ✓ Remove/Delete a value

Search Operation in a BST

Searching in a BST always starts at the root. We compare a data store at the root with the key we are searching for. If the node does not contain the key we proceed either to the left or right child depending upon comparison. If the result of comparison is negative we go to the left child, otherwise to the right child. Here are two implementations of the dynamic set operation search

```
TREE-SEARCH(x, k)
if x == NIL or k == x.key
    return x
if k < x.key
    return TREE-SEARCH(x.left, k)
else return TREE-SEARCH(x.right, k)
```

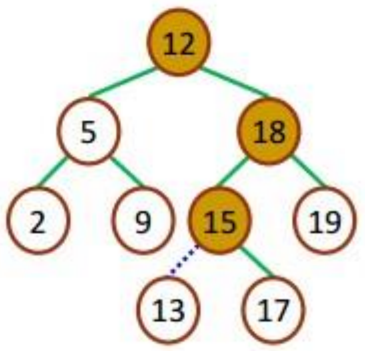
```
ITERATIVE-TREE-SEARCH(x, k)
while x ≠ NIL and k ≠ x.key
    if k < x.key
        x = x.left
    else x = x.right
return x
```

Insert/Add operation in a BST

To insert any element in a BST requires the concept of search operation. In a search operation when the element is not present then its output is nil. Insert operation will replace nil with a new node. Like search operation, insertion can also be recursive and iterative.

Algorithm for Insertion:

```
TREE-INSERT(T, z)
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      /* Tree T was empty */
    else if key[z] < key[y]
        then left[y] ← z
        else right[y] ← z
```



Inserting an item with key 13

Minimum and Maximum operations in a BST

The binary search tree property guarantees that

- ➢ The minimum key of a BST is located at the leftmost node, and
- ➢ The maximum key of a BST is located at the rightmost node Traverse the

appropriate pointers (left or right) until NIL is reached.

Algorithm for Minimum and Maximum

```
TREE-MINIMUM(x)
1.    while left[x] ≠ NIL
2.        do x ← left[x]
3.    return x
```

```
TREE-MAXIMUM(x)
1.    while right[x] ≠ NIL
2.        do x ← right[x]
3.    return x
```

Height of a BST

For a tree with just one node, the root node, the height is defined to be 0, if there are 2 levels of nodes the height is 1 and so on. A null tree (no nodes except the null node) is defined to have a height of -1.

Binary Tree Traversals:

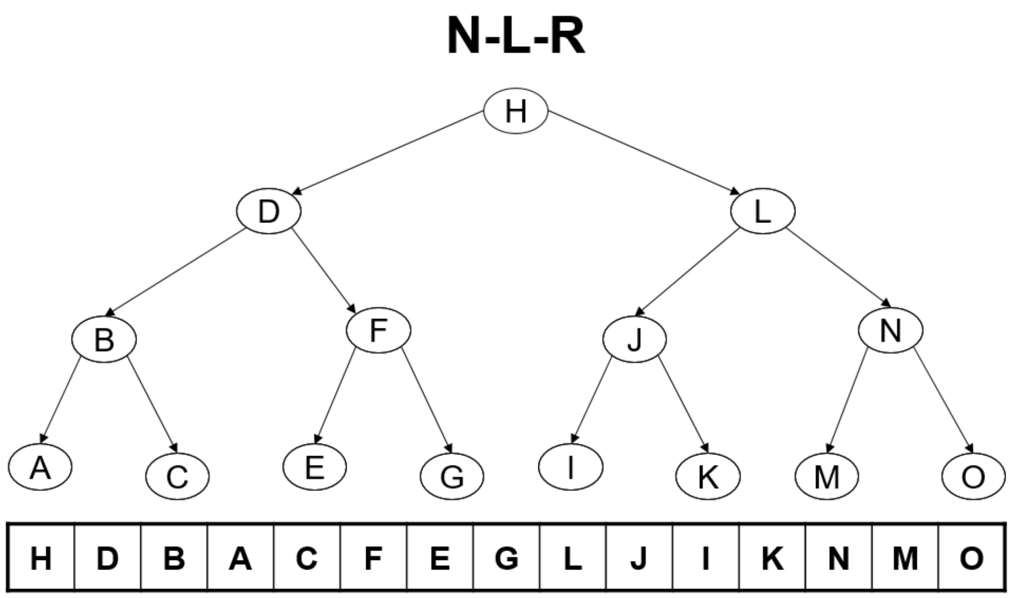
A traversal is a process that visits all the nodes in the tree. Since a tree is a nonlinear data structure, there is no unique traversal. We will consider several traversal algorithms with we group in the following two kinds.

- Depth First Traversal
- Breadth First Traversal

Depth First Search:

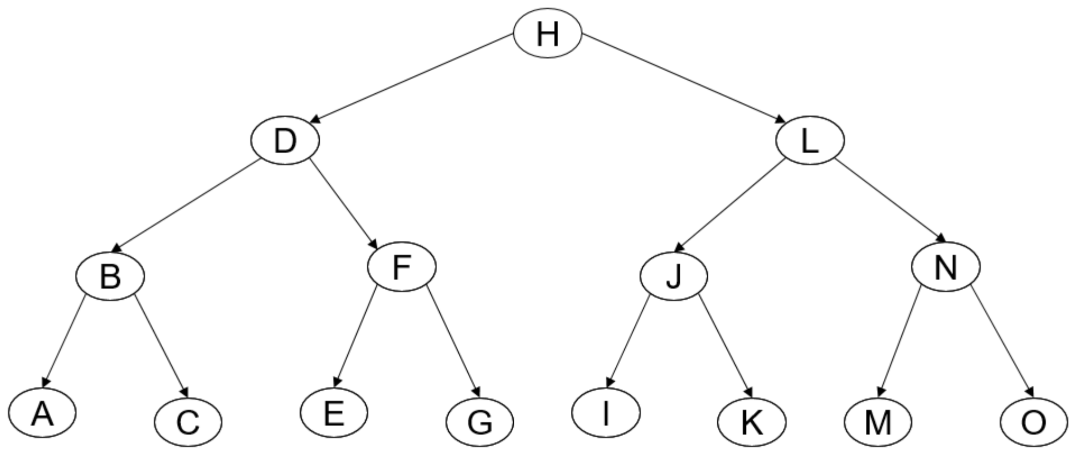
There are three different types of depth first traversal

- Pre Order traversal – visit the parent first and then left and right children



- In Order traversal – visit the left child, then the parent and the right child

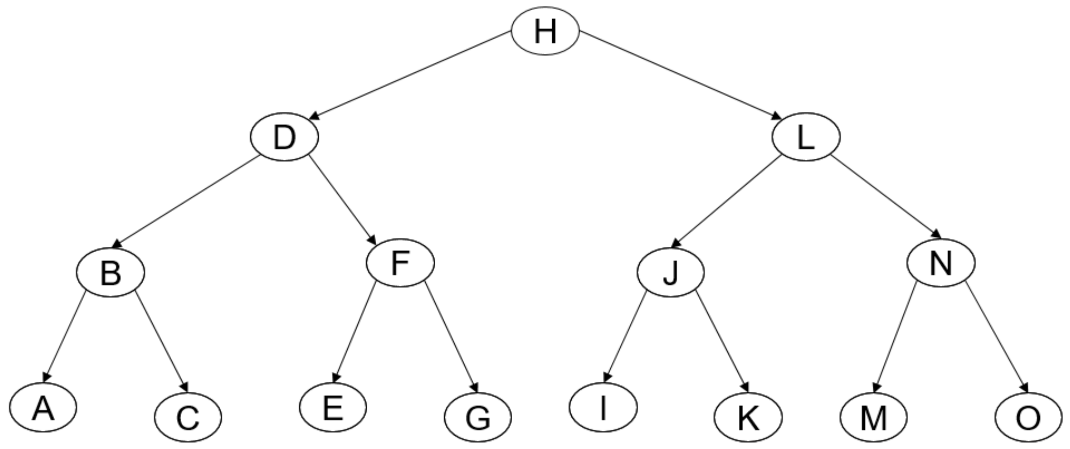
L-N-R



A	B	C	D	E	F	G	H	I	J	K	L	M	N	O

- Post Order traversal – visit left child, then the right child and then the parent

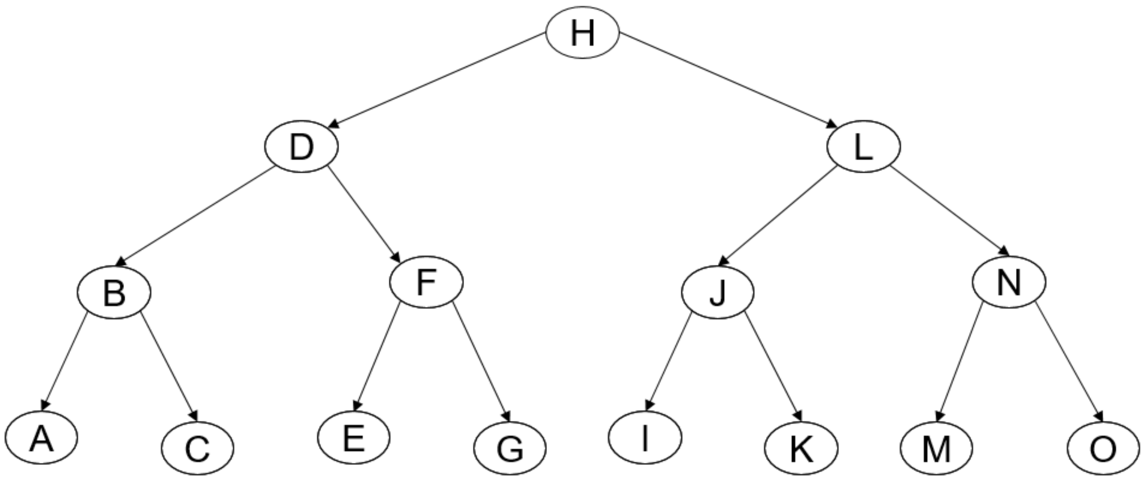
L-R-N



A	C	B	E	G	F	D	I	K	J	M	O	N	L	H

Breadth First Traversal:

There in only one kind of breadth first traversal – the level order traversal. This traversal visits nodes by levels from top to bottom and from left to right.



H	D	L	B	F	J	N	A	C	E	G	I	K	M	O

Exercise:

Question No. 1:

Write Java programs that use recursive and non-recursive functions to traverse the given binary tree in

(a) Preorder
(b) Inorder
(c) Postorder.

Question No. 2:

Write a java program that reads an infix expression, converts the expression to postfix form and then evaluates the postfix expression (Using Tree and Depth First Traversing Techniques)

Question No. 3:

Write a Java program that displays node values in a level order traversal (Traverse the tree one level at a time, starting at the root node) for a binary tree.

Question No. 4:

Write a Java program that uses recursive functions.

(a) To create a binary search tree.
(b) To count the number of leaf nodes.
(c) To copy the above binary search tree.

Question No. 5:

Write a Java program to perform the following operations:

(a) Insert an element into a binary search tree.
(b) Delete an element from a binary search tree.
(c) Search for a key element in a binary search tree.