جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

كلية العلـوم

قسـم علوم الذكاء الاصطناعي

المحاضرة الرابعة

Binary Tree

المادة: Searching and Sorting Algorithms
المرحلة: الثانية
اسم الاستاذ: م.م اية محمد حسين محمد علي

Binary Tree

In a normal tree, every node can have any number of children. Binary tree is a special type of tree data structure in which every node can have a maximum of 2 children. One is known as left child and the other is known as right child. Binary Tree: is a tree in which every node can have a maximum of two children. In a binary tree, every node can have either 0 children or 1 child or 2 children but not more than 2 children.

Example:

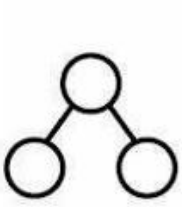


Figure 1:

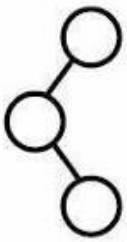


Figure 2:

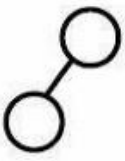


Figure 3:

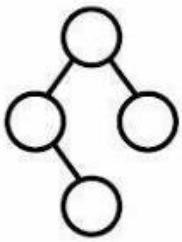
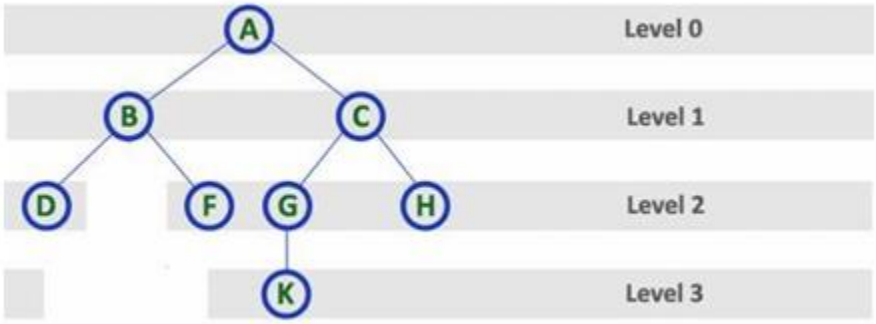


Figure 4:

Example:



1. The Maximum number of nodes in any level L

($Max\ Nodes\ in\ any\ level = 2L$)

2. The maximum number of the nodes in the binary tree ($2^{h+1} - 1$) where h is the height of the tree so in the example ($2^{3+1} - 1 = 15$) and the real number is 8.

3. The number of the leaves of the binary tree is equal to

No. of leaves= (no. of nodes which have degree 2)+1

In the above example 3+1=4.

2.2 Binary Tree Representations:

A binary tree data structure is represented using two methods. Those

methods are as follows...

1. Array Representation

2. Linked List Representation

We use double linked list to represent a binary tree. In a double linked list,

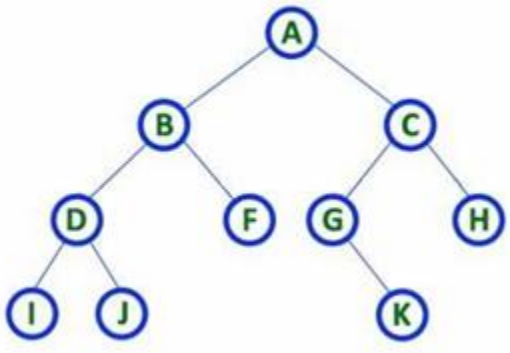
every node consists of three fields. First field for storing left child address,

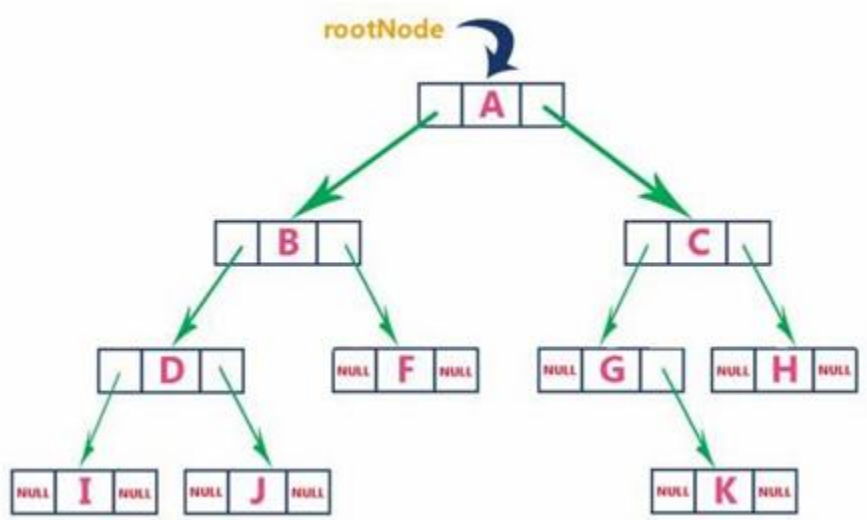
second for storing actual data and third for storing right child address.

In this linked list representation, a node has the following structure...

Left Child Address	Data	Right Child Address

The below example of binary tree represented using Linked list representation is shown as follows...




rootNode
A
B
C
D
NULL
F
NULL
NULL
G
NULL
NULL
H
NULL
NULL
I
NULL
NULL
J
NULL
NULL
K
NULL

s

2.3 Binary Tree Traversal

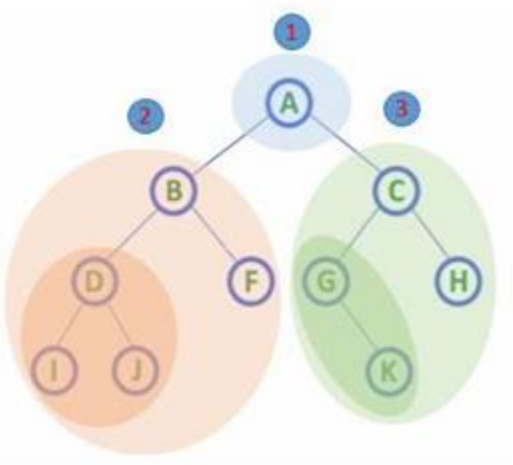
When we wanted to display a binary tree, we need to follow some order in which all the nodes of that binary tree must be displayed. In any binary tree displaying order of nodes depends on the traversal method. Displaying (or) visiting order of nodes in a binary tree is called as Binary Tree Traversal. There are three types of binary tree traversals.

1. Pre - Order Traversal

2. In - Order Traversal

3. Post - Order Traversal

Consider the following binary tree…

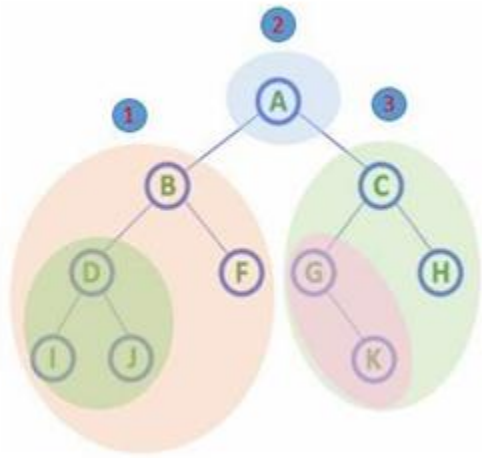


1. Pre - Order Traversal (root – left Child – right Child)

In Pre-Order traversal, the root node is visited before left child and right child nodes. In this traversal, the root node is visited first, then its left child and later its right child. This pre-order traversal is applicable for every root node of all subtrees in the tree. In the above example of binary tree, first we visit root node 'A' then visit its left child 'B' which is a root for D and F. So we visit B's left child 'D' and again D is a root for I and J. So we visit D's left child 'I' which is the left most child. So next we go for visiting D's right child 'J'. With this we have completed root, left and right parts of node D and root, left parts of node B. Next visit B's right child 'F'. With this we have completed root and left parts of node A. So we go for A's right child 'C' which is a root node for G and H. After visiting C, we go for its left child 'G' which is a root for node K. So next we visit left of G, but it does not have left child so we go for G's right child 'K'. With this

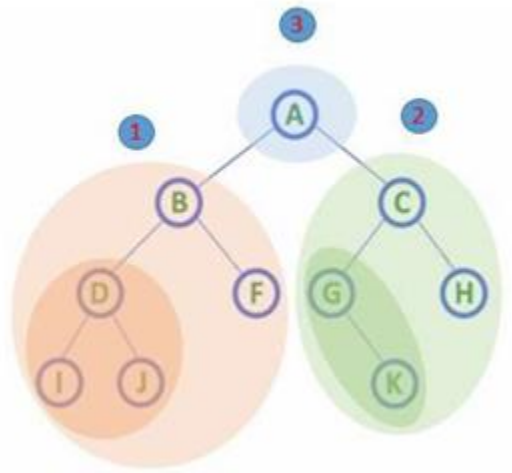
we have completed node C's root and left parts. Next visit C's right child 'H' which is the right most child in the tree. So we stop the process.

Pre-Order Traversal for above example binary tree is A - B - D - I - J - F - C - G - K – H



2. In - Order Traversal (left Child - root – right Child) In In-Order traversal, the root node is visited between left child and right child. In this traversal, the left child node is visited first, then the root node is visited and later we go for visiting right child node. This in-order traversal is applicable for every root node of all subtrees in the tree. This is performed recursively for all nodes in the tree.In the above example of binary tree, first we try to visit left child of root node 'A', but A's left child is a root node for left subtree. so we try to visit its (B's) left child 'D' and again D is a root for subtree with nodes I and J. So we try to visit its left child 'I' and it is the left most child. So first we visit 'I' then go for its root node 'D' and later we visit D's right child 'J'. With this we have completed the left part of node B. Then visit 'B' and next B's right child 'F' is visited. With this we have completed left part of node A. Then visit root node 'A'. With this we have completed left and root parts of node A. Then we go for right part of the node A. In right of A again there is a subtree with root C. So go for left child of C and again it is a subtree with root G. But G does not have left part so we visit 'G' and then visit G's right child K. With this we have completed the left part of node C. Then visit root node 'C' and next visit C's right child 'H' which is the right most child in the tree so we stop the process.

In-Order Traversal for above example of binary tree is I - D - J - B - F - A - G - K - C – H



3. Post - Order Traversal (left Child – right Child - root) In Post-Order traversal, the root node is visited after left child and right child. In this traversal, left child node is visited first, then its right child and then its root node. This is recursively performed until the right most node is visited. Post-Order Traversal for above example binary tree is I - J - D - F - B - K - G - H - C – A

2.4 The functions of the Binary tree

As we mentioned before that, the structure of the tree consists of subtrees so, that means the part looks like the all, so here we can make use of the recursion to represent the functions of the tree.

1. In - Order Traversal

```
void inorder(nodeptr t)
{     if(t!=0)
     {
         inorder(t->l);
         cout<<t->info<<' ';
         inorder(t->r);
     }
}
```

2. Pre - Order Traversal

```
void preorder (nodeptr& t)
{     if(t!=0)
     {
         cout<<t->info<<' ';
         preorder(t->l);
         preorder(t->r);
     }
}
```

3. Post - Order Traversal

```
void postorder(nodeptr t)
{    if(t!=0)
      {
            postorder(t->l);
            postorder(t->r);
            cout<<t->info<<' ';
      }
}
```