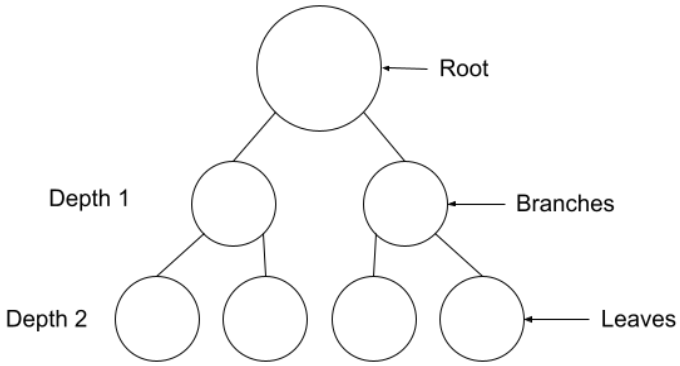
Topic 7 - Trees

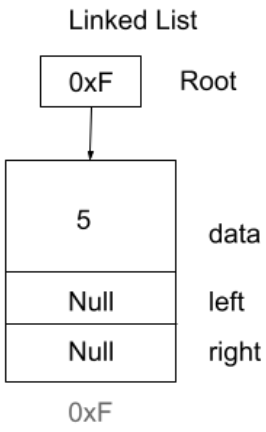
Trees: Introduction

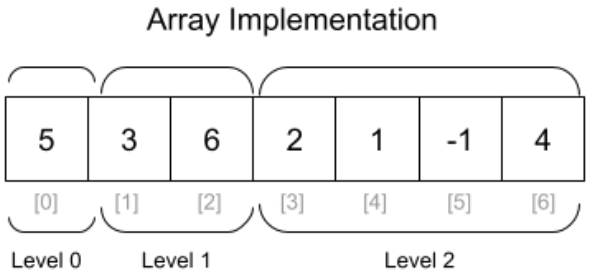


Keywords for trees:

- **Parent**
- **Child**
- **Ancestors**
- **Descendants**
- Every node in the tree has a **depth**, which is the number of branches between the root and the node itself.
- **Height** == longest dept
- When the number of children of every node in a tree is constrained to a maximum of two, that tree is known as a **binary tree**
- If every node has exactly two children except the ones in the last level, then we talk about a **full binary tree**
- Since every node has a maximum of two children, binary trees are drawn in such a way that one child is drawn to the left of its parent and the other to the right. Because of that, they are called the **left and the right child**. For the same reason, we can talk about **left and right sub-trees**
- Number of nodes of a full binary tree with **k levels:** $2^k - 1$ (levels starts at 0)

Binary trees: Implementation





Binary tree traversal: Introduction

- Traversal of the tree is the process of **visiting all the nodes of a tree**.
- There are two main tree traversal approaches, **breadth-first traversal: Horizontal approach** and d**epth-first traversal: Vertical approach.**
- Depth-first traversal types:
    - Pre-Order: Visit the root node **before** any other node
    - In-Order: The root node is visited in the **middle** of the traversal
    - Post-Order: Visit the root node **after** having visited all other nodes

Depth-first traversal

```
function pre-order(T)
  if !ISEMPTY(T) then
   visit Root(T)
   pre-order(left(T))

pre-order(right(T))
  end if
end function
```

```
function in-order(T)
  if !ISEMPTY(T) then
   pre-order(left(T))
   visit Root(T)

pre-order(right(T))
  end if
end function
```

```
function
post-order(T)
  if !ISEMPTY(T) then
   pre-order(left(T))

pre-order(right(T))
   visit Root(T)
  end if
end function
```

Breadth-first traversal

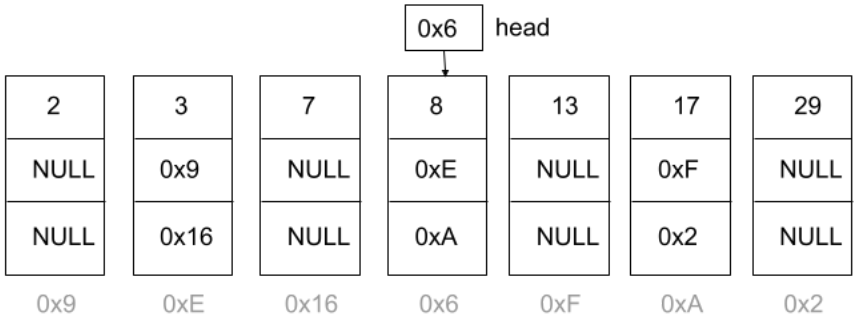
```
function breadth-first(root)
      Q ← new Queue()
      enqueue-if(Q,root)
      while !ISEMPTY(Q) do
            t ← PEEF(Q)
            visit(t)
            enqueue-if(Q,left(t))
            enqueue-if(Q,right(t))
            DEQUEUE(Q)
      end while
end function

function enqueue-if(Q,t)
      if !NULL(t) then
            ENQUEUE(Q,t)
end function
```

Binary search trees (BSTs)

Rules:

- Nodes in the left sub-tree must be less than the root
- Nodes in right sub-tree greater than root



BST: Insert

```
T(N)= Θ(log(N)) for a balanced tree and Θ(N) not balanced
function INSERT_BST(root,x)
      if (root == NULL)
            Node newNode = new Node(x)
            root = newNode
      else
            if (x < root->data)
                  INSERT_BST(root->left, x)
            else
                  INSERT_BST(root->right, x)
```

BST: Search

```
function  SEARCH_BST( root,x )
      if (root == NULL)
            return  False
      else
            if (x == root->data)
                  return True
            else
                  if ( x < root->data)
                        SEARCH_BST( root->left, x )
                  else
                        SEARCH_BST( root->right, x )
end function
```

BST: Delete

```
function DELETE_BST(root, x)
   if(root == NULL)
      return  NULL
   else
      if (x < root->data)
         root->left = DELETE_BST( root->right, x )
       else
         if (x > root->data)
            root->right = DELETE_BST( root->right , x )
         else // node found
            if (Case 1) Delete leaf
            else if (Case 2)
               if (Left Child) parent point to left child
               if (Right child) parent point to right child
            else if (Case 3)
               find minimum in right sub-tree
               copy it
               delete node with minimum
end function
```

Case 1: node with no children

```
function DELETE_BST (root, x)
[...]
     if ( root->left == NULL && root->right == NULL )
          root = NULL
          return root
[...]
```

Case 2: node with 1 child

```
function DELETE_BST (root, x)
[...]
     else if ( root->left == NULL )
          root = root->right
          return root
     else if ( root->right == NULL )
          root = root->left
          return root
[...]
```

Case 3: node with 2 children

```
function DELETE_BST (root, x)
[...]
      else
            Node tmp= getRMin(root->right)
            root->data = tmp->data
            root->right = DELETE_BST(root->right, root->data)
[...]
```

Get Right Minimum function

```
function getRMin (root)
      Node tmp = root->right
      if (tmp != NULL)
            while (tmp->left != NULL)
                  tmp=tmp->left
            return tmp
end function
```