Trees 2 🌲

Binary Trees

- A binary tree is a rooted tree with no vertex
- has more than two children
  - left and right child nodes

```C++
// Syntax
struct BinaryNode{
        Object element;     // The Data in the node
        BinaryNode *left;   // & of Left Child
        BinaryNode *right;  // & of Right Child
}
```

- A binary tree is complete iff every layer except possibly the bottom is fully populated with vertices. All nodes at the bottom level must occupy the leftmost spots consecutively.
- A complete binary tree with n vertices and h height satisfies
  - $2^H \leq n < 2^{H+1}$
  - $2^2 \leq 7 < 2^{2+1}, 2^2 \leq 4 < 2^{2+1}$
  - $2^H \leq n < 2^{H+1}$
  - $H \leq log(n) < H + 1$
  - $H = floor(log(n))$
- **Proof:**
  - At level $k \leq H - 1$, there are 2^K vertices
  - At level $k = H$, there is at least 1 node, and at most 2^H vertices
  - Total number of vertices when all levels are fully populated (maximum level k)
    - $n = 2^0 + 2^1 + ... + 2^k$
    - $n = 1 + 2^1 + 2^2 + ... 2^k$ (Geometric Progression)
    - $n = \frac{1(2^{k+1} - 1)}{2 - 1}$
    - $n = 2^{k+1} - 1$

Case 1:

A tree has the maximum number of nodes when all levels are fully populated

- Let $k = H$
  - $n = 2^{H+1} - 1$
  - $n < 2^{H+1}$

Case 2:

The tree has a minimum number of nodes when there is only one node in the bottom level

- Let $k = H - 1$
  - $n' = 2^H - 1$
  - $n \geq n' + 1 = 2^H$

Combining two conditions we have

- $2^H \leq n \leq 2^{H+1}$

Representation of Complete Binary Tree

- All trees can be represented by the generic representation shown in the code above
- Due to the structure of a complete binary tree, it cannot be represented by a vector
  - As long as one can figure out the parent/child relationship
  - Parent/child relationship embedded in the index of parent and child
  - Vector elements carry data
- **Tree Structure : Vector**
  - Vector indices carry tree structure
  - Index order = levelorder
  - The tree structure is implicit
  - Uses integer arithmetic for tree navigation
  - No need to explicitly store the tree node pointers
- **Tree Navigation : Vector**
  - The root at `v[0]`
  - Parent of `v[k] = k[(k-1)/2]`
  - Left child of `v[k] = v[2*k + 1]`

- Right child of `v[k] = v[2*k]+1`

Binary Tree Traversal

Inorder traversal

- Definition
  - Left child
  - Vertex
  - Right Child (recursive)
- Algorithm
  - Depth-first search (visit between children)

Other Traversals that apply to binary case

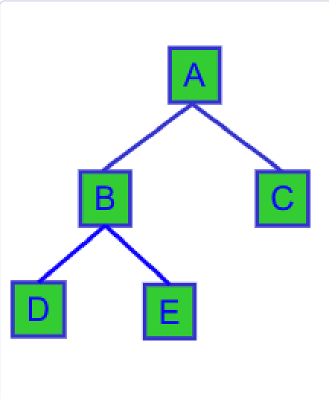
- Preorder
  - [Trees 🌲]()
- Postorder
  - [Trees 🌲]()
- Level order traversal

> A tree can be rebuilt from its inorder and preorder (or postorder) traversal results

---

Rebuild Tree from Traversal

- Let each node be associated with a letter, traversals print the letters when visiting a node. The results are:
  - Preorder: "ABDEC"
  - Postorder: "DEBCA"
  - Inorder: "DBEAC"



Rebuild tree from preorder + inorder traversal

- Find the root from the preorder result: A
- Decide left and right subtrees
    - Find the letter for the root in the inorder string and decide the inorder string for the two subtrees
- Decide the preorder string for left and right subtrees
    - Inorder for the traversal string length should be equal to another string length, extract preorder strings from the whole preorder string
- Recursively do this to the sub-trees

Build Expression Tree from Postfix Expression

```C++
stack<T> s;
while(s != /*end of postfix expression*/){
        // Get the next token
        if(token == /* operand */){
                // Create a new node with the operand
                s.push(/* New Node */);
        }
        if(token == /* operator */){
                s.pop(); // corresponding operands from S
                // Create a new node with the operator (and
corresponding operands as left/right children)
                s.push(/* New Node */);
        }
} // s.top is the final binary expression
```

Rebuilding with a tree

- Postorder string: FECAHJIGB
- Inorder string: CFEABHGJI

Root: B

- Last in the post-order string
  Left Subtree: CFEA
- Before root (B) in the inorder string
  Right Subtree: HGJI
- After root (B) in the inorder string