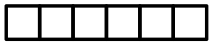
Introduction to Trees

Linear data structures : Arrays, Stacks, Linked List, Queue

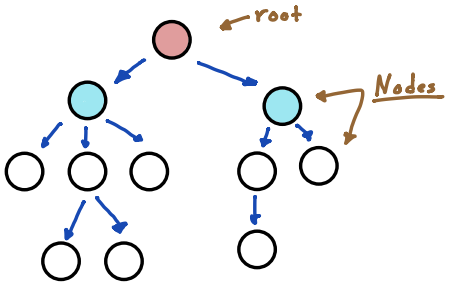


Arrays

How should I decide which data structure to use?

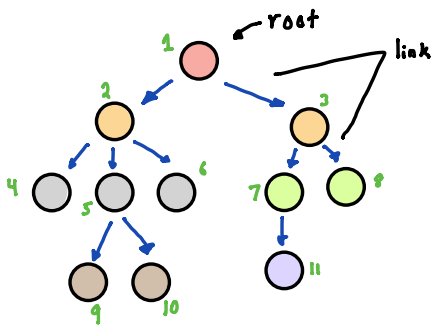
- What need to be stored?
- Cost of operation
- memory usage
- Ease of implementation

Tree used for Hierarchical data representation



Tree

- Non-linear
- Node contains data and possibly reference to another node



Common Terminology

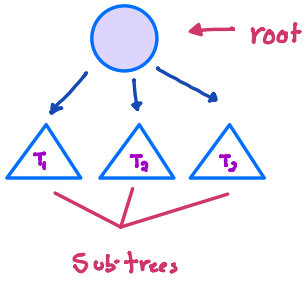
Note: #'s dont represent anything in our example

- 2 and 3 links to Node 1 and our "Children" of node 1
- Node 1 is called "Parent" of Nodes 2 and 3

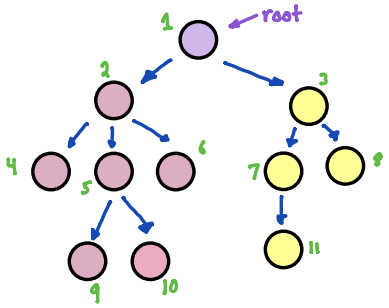
- Children of the same parent are called Siblings
- Any node in the tree w/o a child is called a leaf Node
- All nodes with 1 child are called internal nodes
- parent of a parent is called Grandparent
- When walking in the tree we can only go in one direction
- Ancestor and descendent
- Cousins, uncles

Properties of Trees

- **Recursive** data structure
- We can define a tree recursively as a structure that consists of a distinguished node called a root and some sub-trees and the arrangement is such that root of the tree contain link to roots of all the sub-trees (T_1, T_2, T_3) in our diagram.

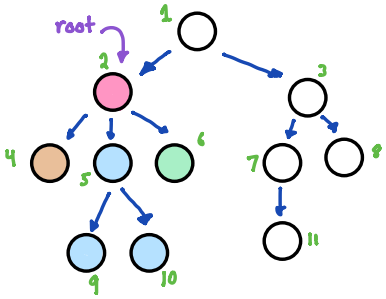


Example from our diagram above:



~ Root node in Purple, the left sub-tree in red and right subtree in yellow.

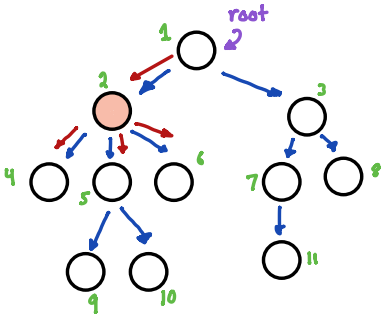
We can also look further at the sub-tree



- With number 2 as the root which has 3 sub-trees:

Properties of Trees Continued

- In a tree with n nodes, there will be <u>exactly</u> n-1 links or edges
- All nodes except the root node will have exactly <u>one</u> <u>edge</u>



- At node 2, we have one incoming link and 3 outgoing links

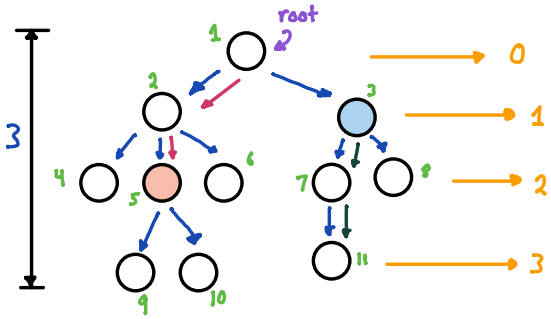
Depth and Height Properties

Depth of x: length of path from root to x
or
Number of edges in path from root to x

Height of x: Number of edges in longest path from x to a leaf.

Height of tree: Height of root node

Example of Depth and Height:



Depths of nodes in Orange

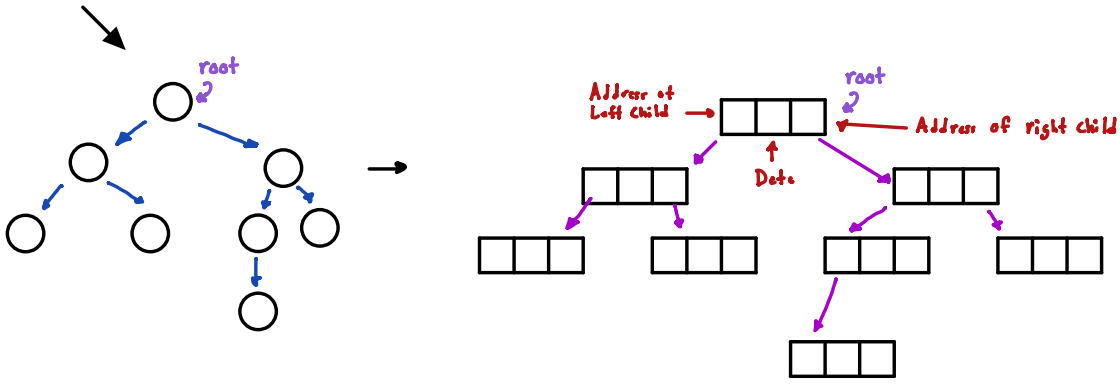
- In node 5 we have 2 edges in the path from root. So the depth is 2
- In node 3, the longest path from this node to any leaf is 2.

Different types of Trees

Binary Tree:

- a tree in which each node can have at most 2 children

- most famously



Code in Python:

Node Class:

```
Class Node:

    def __init__(self, data):

        self.left = None
        self.right = None
        self.data = data
```

Application of trees

1) Storing naturally hierarchical data
   → e.g. file system

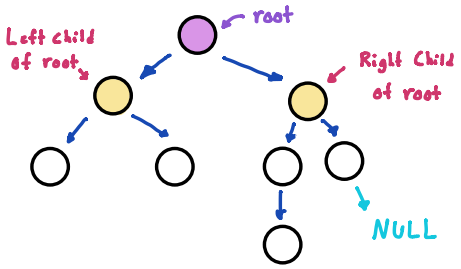
2) Organizing data for quick search, insertion, deletion
   → e.g. Binary Search trees

3) Trie
   → Dictionary

4) Network Routing Algorithm

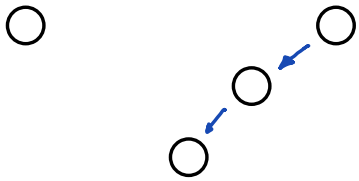
Binary Tree

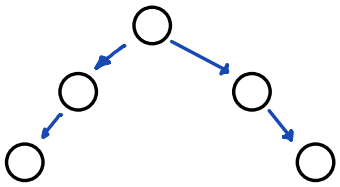
Properties:



- A node may have both right and left child or a node can have either of right or Left child
- In a program we'd set reference of empty child to node
- For Leaf, we can set both Left and right to NULL

Different types of Binary trees

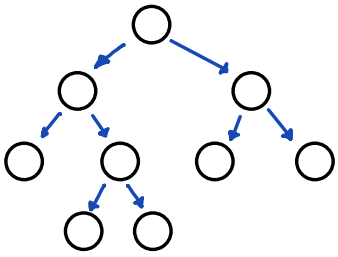




Strict/Proper binary Tree

↳ each node can have either 2 or 0 childern.

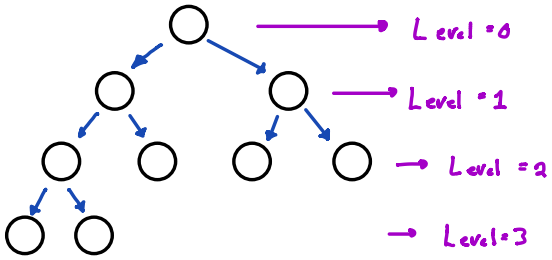
Example



Complete Binary Tree

all levels except possibly the last are completely filled and all nodes are as left as possible

- max number of nodes at level $i = 2^i$



Perfect Binary Tree

Maximum number of nodes in a binary tree with height h

$= 2^0 + 2^1 + \dots + 2^h$
$= 2^{h+1} - 1$
$= 2^{(\text{no of levels})} - 1$

$n = 2^{h+1} - 1$ ← n = # of nodes

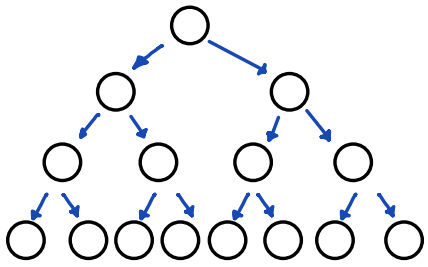
$\rightarrow 2^{h+1} = (n+1)$

$h = \log_2(n+1) - 1$
↓ is

$h = \log_2(15+1) - 1$

$h = \log_2 16 - 1$

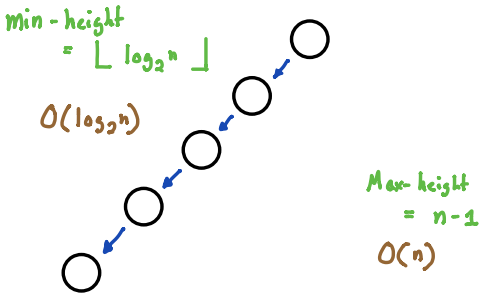
$h = 4 - 1$ $\quad 2^x = 16$
$h = 3$ $\quad$ - Solve for x



Height of Complete Binary tree

$= \lfloor \log_2 n \rfloor$ ← floor function

Time Complexity: $O(h)$ → h = height of tree



n	$\log_2 n$
2^{100}	100

Balanced Binary Tree

- difference b/w height of Left and right subtree for every node is not more than K (mostly 1)

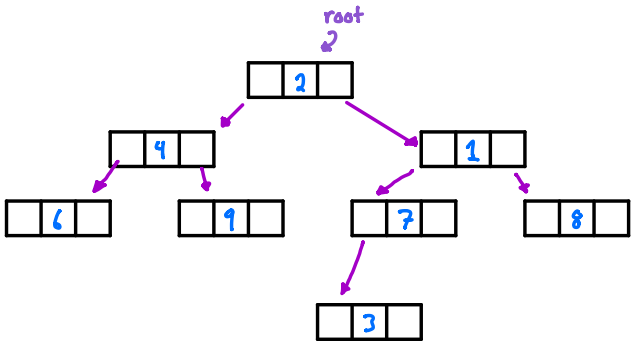
> Height → number of edges in longest path from root to a leaf
>
> Height of an Empty tree = -1
> Height of tree with 1 node = 0

$diff = | h_{left} - h_{right} |$

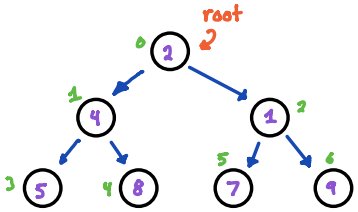
We can implement Binary tree using:

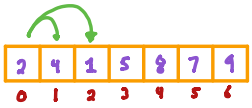
a) Dynamically created nodes

b) arrays
↓
Used for heaps



a) Dynamically created nodes





for node at index i,
left-child-index = 2i + 1
right-child-index = 2i + 2
→ in a Complete Binary tree