Trees

Bird's Eye View	A tree is a non-linear, acyclic, recursive data structure where data is arranged in a hierarchy with a set of connected nodes. There are many types of trees such as heaps, binary tree, or tries but the most important is the binary search tree which has O(log(N)) for access, inserting and searching. There are two main ways to traverse a tree and they are called BFS and DFS. DFS you can do in three different ways. You can always find the shortest distance with BFS but not DFS. DFS takes less memory however
Major Terms	binary tree, binary search tree, k-nary tree, BFS, DFS, inorder, preorder, post order
Type	Data Structure

Linear/Non-Linear

▼ What kind of data structures are arrays, linked lists, stacks and queues?

- linear data structures

▼ What does it mean for something to be a linear data structure again?

- data is arranged in a sequential manner and there is a logical start and end

▼ What is a non-linear data structure?

- where the data doesn‘t stay arranged linearly or sequentially

Trees - Introduction

▼ What is a tree?

- non-linear, recursive, acyclic data structure where data is arranged in a hierarchy with a set of connected nodes

▼ What are trees made of?

- nodes and links

▼ What makes trees different from other linear data structures?

- each node can have links to multiple other nodes

▼ Give a couple examples of data trees can be used to represent

1. company organization…CEO —> CTO, COO, CIO, etc.
2. geography…planet —> continents —> countries —> states/provinces —> etc.
3. file structure in a computer

▼ The structure of trees and the important terms

▼ Root

- the top most of the tree, which doesn't have any links connecting to it

▼ Link/edge

- the reference that a parent node contains that tells what its child node is

▼ Parent Node

- any node that has a reference or link to another node

▼ Does root have a parent?

- no root is the only node that doesn't have a relationship

▼ Child Node

- any node that has a parent node that links to it

▼ Sibling

- any group of nodes that are children of the same node
- effectively any group of nodes that have the same parent

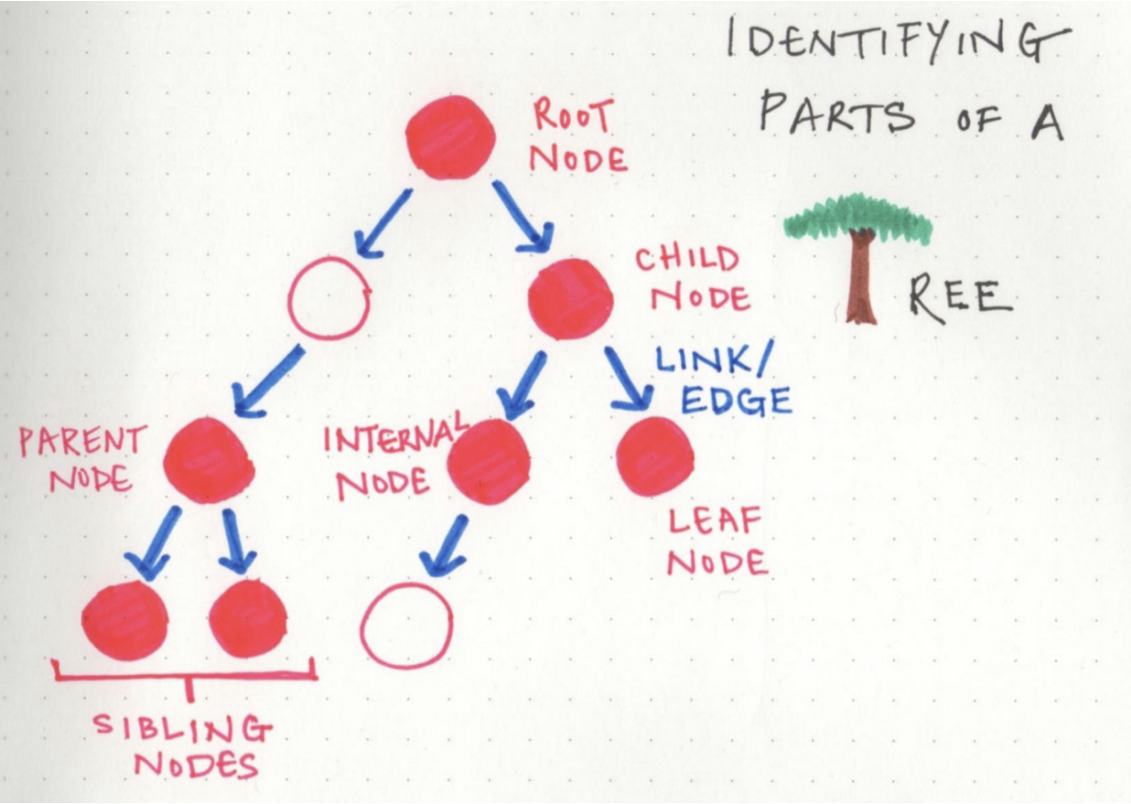
▼ Internal nodes

- any node that has at least one child node so basically all parent nodes

▼ Leaf

- any node that does not have a child node in the tree

▼ Picture



▼ Some universal tree truths

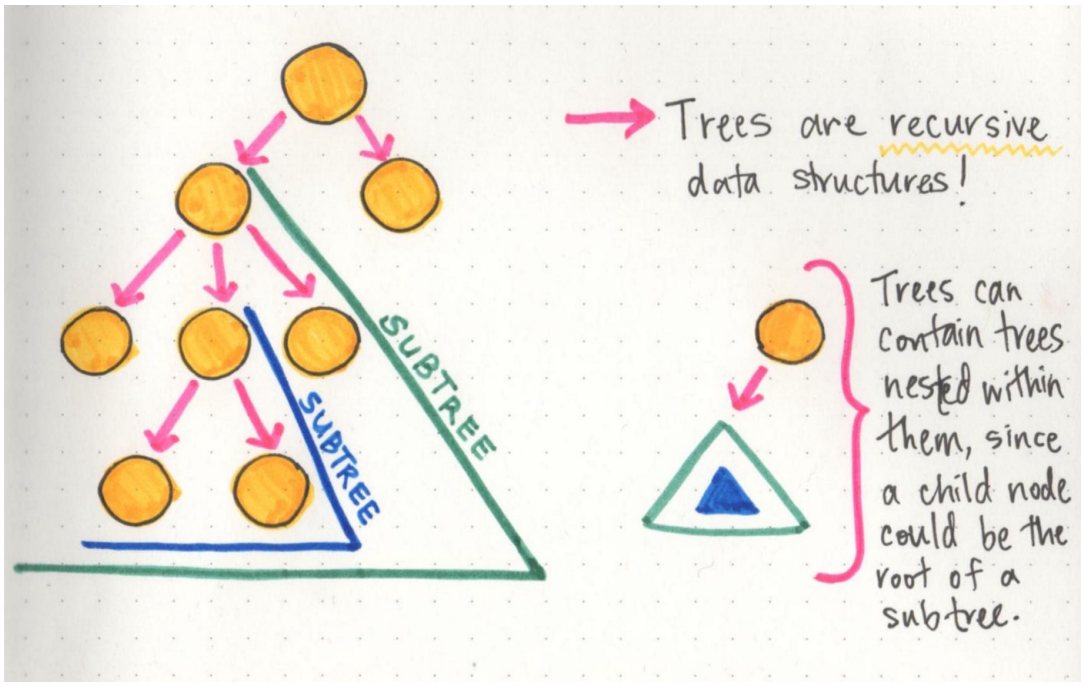
1. if a tree has n nodes, it will always have (n-1) edges

   ▼ Why is this the case?

   - root node has no node connecting to it so if every other node does, then the total number of edges should be n-1

2. Trees have trees within them…basically trees are made up of smaller and smaller **subtrees**

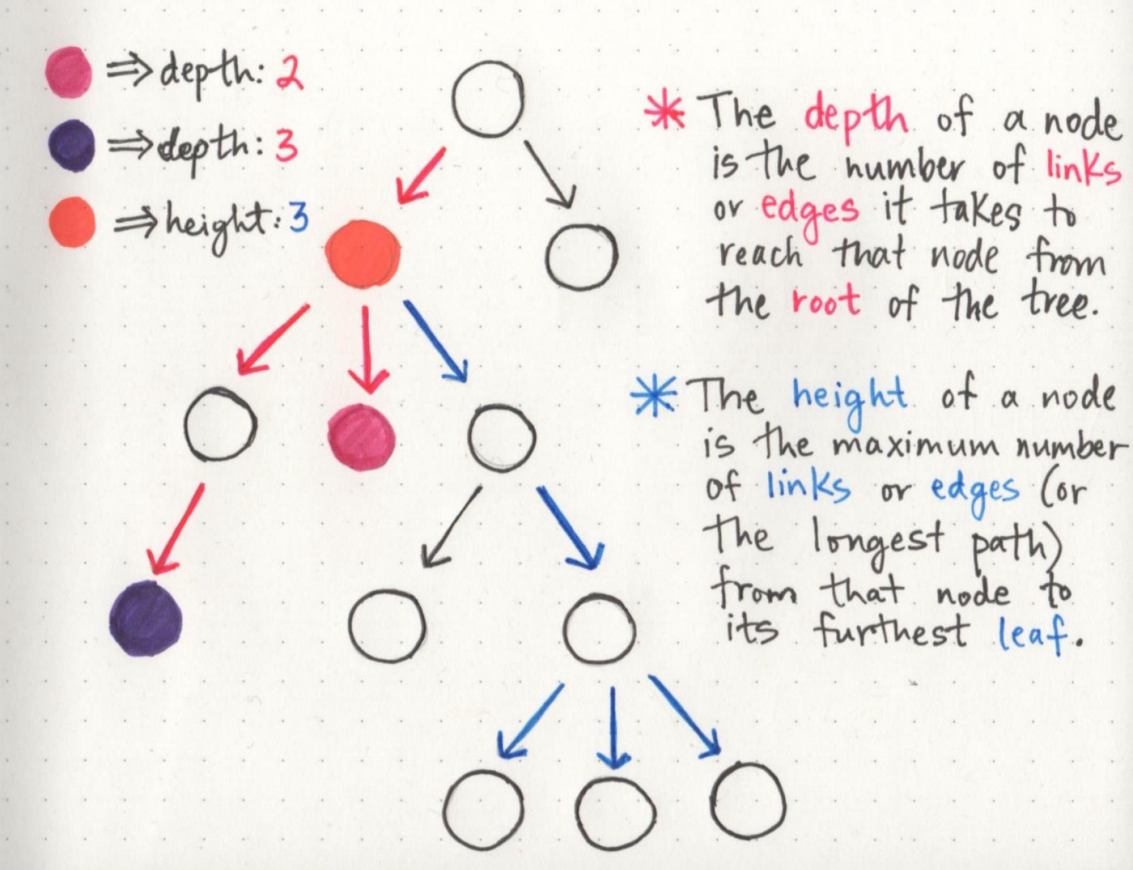
   ▼ Picture



- ▼ What is the implication of this fact?
  - trees are recursive data structures

▼ What's one important way to classify data in the tree?

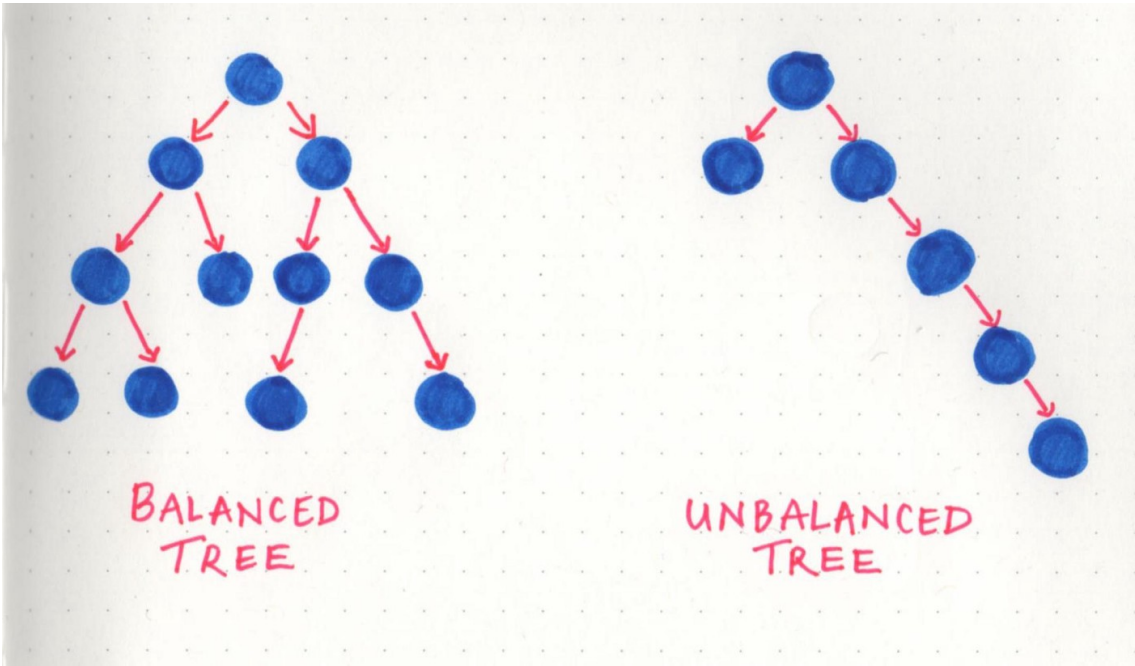
- where it lives in the tree —> the depth of the node or the height of the node
- ▼ What is depth of a node?
  - how far the node is from the root, basically count the number of links
- ▼ What is depth of root node?
  - 0
- ▼ What is the height of the node?
  - maximum number of links or edges from the node to its furthest leaf
  - basically longest path from node to a leaf
- ▼ Picture



▼ What is a balanced and unbalanced tree?

- a tree is balanced if any two sibling subtrees do not differ in height by more than one level
- a tree is unbalanced if any two sibling subtrees have more than one level of depth of difference

▼ Picture



- ▼ Why do we worry about this?
  - when traversing a tree we cut down on half the data each time, so having a well balanced tree means maximizing our efficiency
- ▼ When is a tree balanced?
  - if it maintains a log(n) time complexity if you traverse it by picking one subtree each time

▼ How can we classify tress?

1. by the number of children each node has

▼ What are all the kinds of trees with respect to interviews?

1. binary tree
2. heaps
3. binary search tree
4. tries (deal with strings)

▼ What is a binary tree?

- a tree where each node has no more than 2 child nodes

▼ What is a ternary tree?

- a tree where each node has no more than 3 child nodes

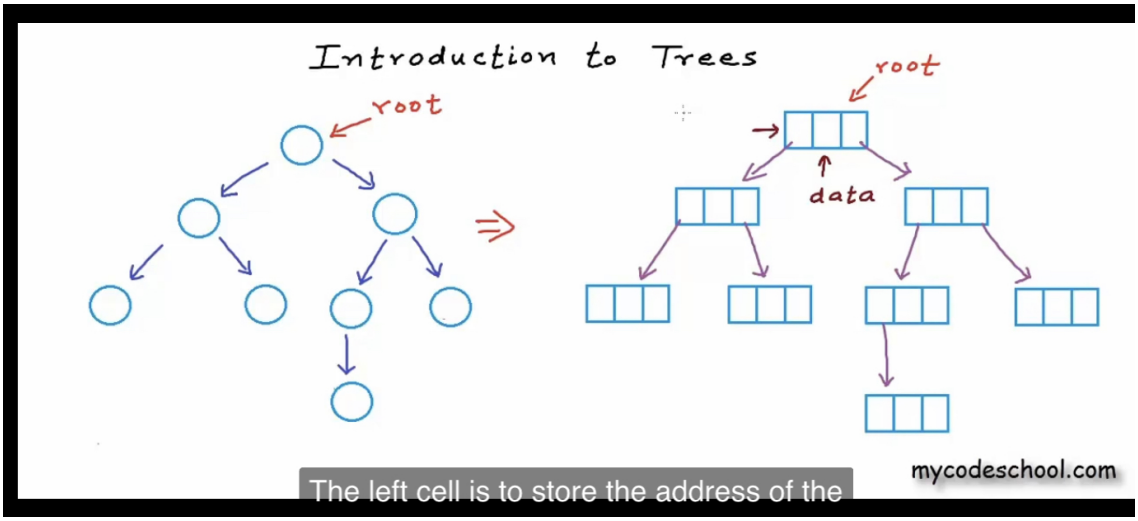
▼ What is a k-ray trees?

- a tree where each node has no more than k child nodes

▼ What is the simplest way to implement trees in code?

- via linked list with 3 fields one for data and two for pointers to next node

▼ Picture



▼ You can use an array, but why is this a dumb idea?

- trees are dynamic data structures where the size can change during runtime. Arrays, on the other hand, are static data structures where the size is fixed at the time of creation.
- To create an array of a fixed size that can accommodate any possible tree structure, we would have to allocate a large amount of memory, most of which would go unused. This can be very inefficient in terms of both memory usage and processing time.
- Furthermore, inserting or deleting nodes in an array-based representation of a tree can be very expensive in terms of time complexity. This is because

the array needs to be resized and all elements need to be shifted to maintain the proper order of the tree.

▼ Applications

1. storing naturally hierarchical data
2. organizing data for quick search, insertion, deletion
3. Trie —> used to search dictionary and spell checking
4. Network routing algorithm

Tree Traversal

▼ What is tree traversal?

- process of visiting each node in a tree exactly once in some order

▼ What the 2 main ways to traverse a tree?

- Depth-First
- Breadth-First

▼ Describe Breadth-First Search

▼ What does breadth mean?

- broad/wide

▼ How does it work?

- level order traversal —> traverse all nodes in one level before going to the next
- visit all node’s children before going to a node’s grandchildren

▼ How do we keep track of the nodes we visited?

- using a queue, first in first out principle

▼ How does it work with the queue?

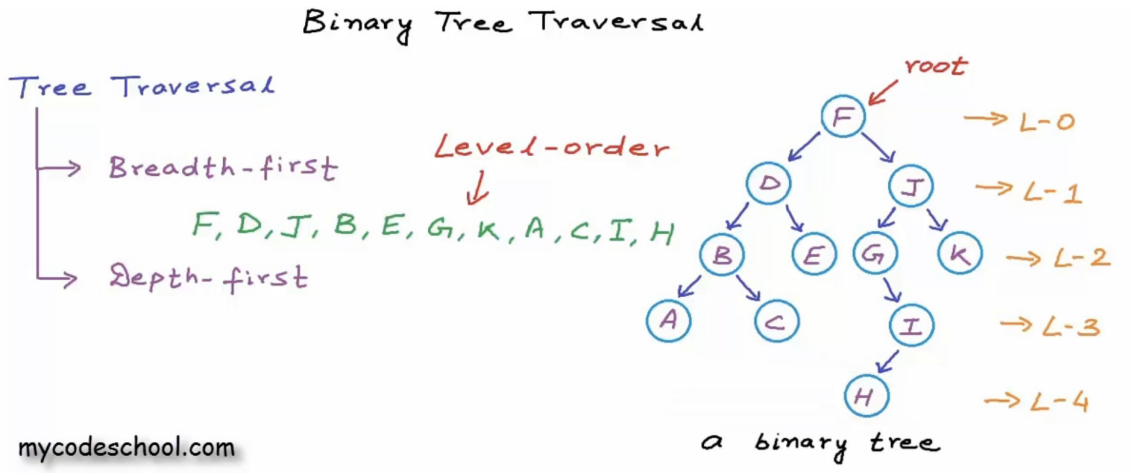
- visit the root and add it into the queue, pop it from the queue and you’ve visited this node do with it what you want. then add its children to the

queue, and then go to the first child and its children to the queue. and pop that first child off and go to the next one.

▼ What is the time and space complexity

- Time: O(N)
- Space: O(1) best, O(N) worst

▼ Picture



▼ Video

https://youtu.be/HZ5YTanv5QE

▼ Describe Depth-First Search

▼ How does it work?

- we traverse each subtree completely before going into the next

▼ What do we use to keep track of the nodes we visited?

- stacks

▼ What is the time and space complexity?

- Time: O(N)
- Space: O(log(n)) average, O(N) worst

▼ How many ways can we achieve this and what are their names?

PreOrder	root, left, right
InOrder	left, root, right
PostOrder	left, right, root

▼ Inorder

- An inorder traversal does an inorder traversal on the left subtree, followed by a visit to the root node, followed by an inorder traversal of the right subtree.

▼ Preorder

A preorder traversal visits the root node first, followed by a preorder traversal of the left subtree, followed by a preorder traversal of the right subtree.

▼ Postorder

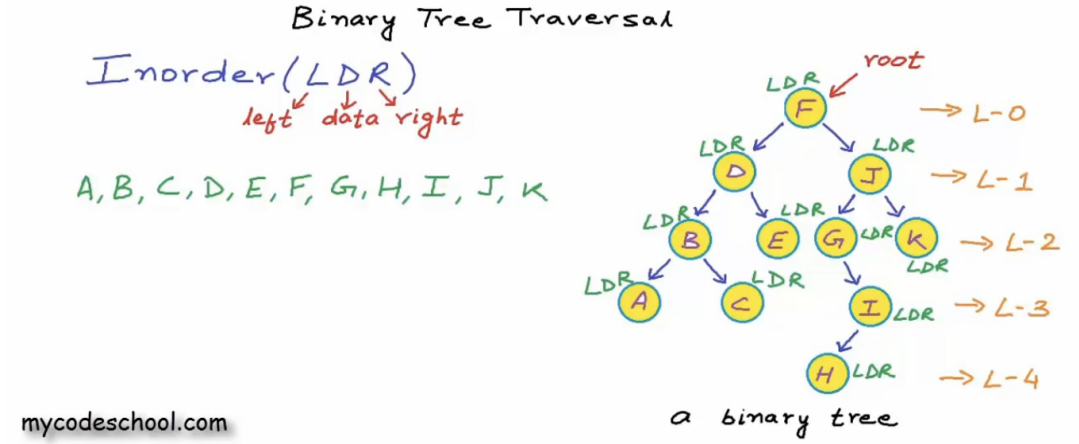
- A postorder traversal does a postorder traversal of the left subtree, followed by a postorder traversal of the right subtree, followed by a visit to the root node.

▼ Where do you use each one?

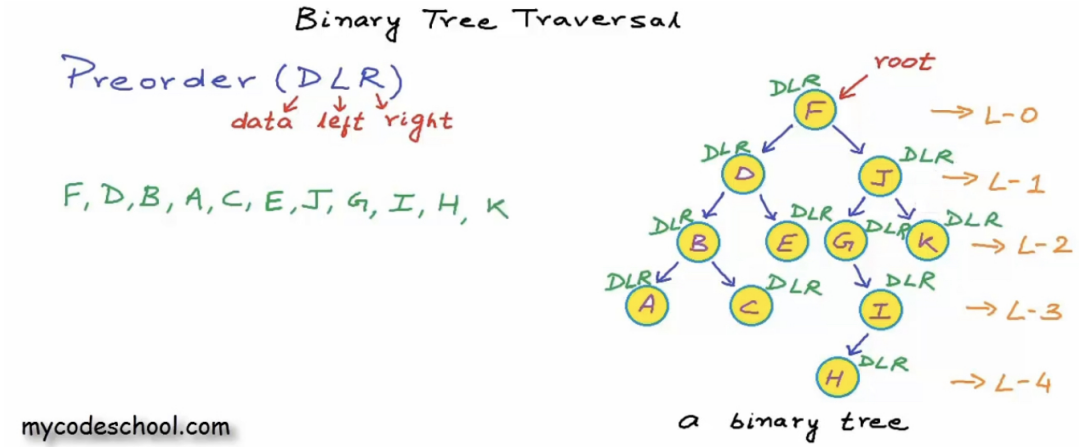
- Inorder: If you know that the tree has an inherent sequence in the nodes, and you want to flatten the tree back into its original sequence, than an **in-order** traversal should be used. The tree would be flattened in the same way it was created. A pre-order or post-order traversal might not unwind the tree back into the sequence which was used to create it.
- Preorder: If you know you need to explore the roots before inspecting any leaves, you pick **pre-order** because you will encounter all the roots before all of the leaves.
- Postorder: If you know you need to explore all the leaves before any nodes, you select **post-order** because you don't waste any time inspecting roots in search for leaves.

▼ Pictures

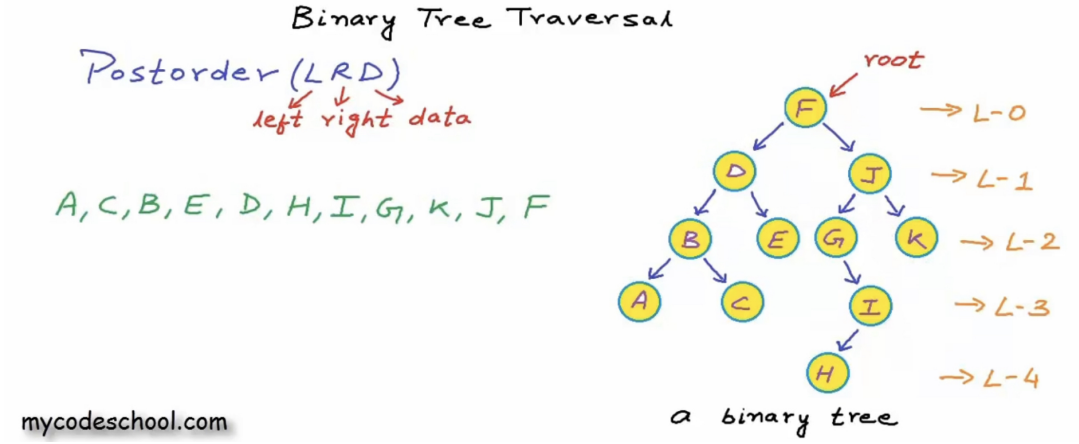
▼ Inorder



▼ Preorder



▼ Postorder

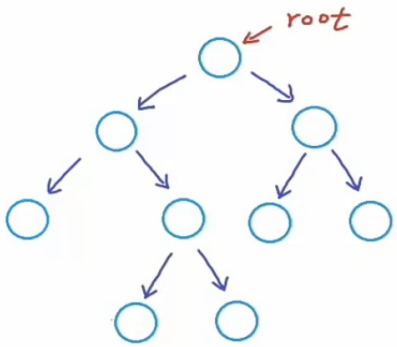


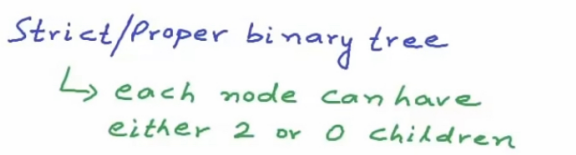
▼ Video

https://youtu.be/Urx87-NMm6c

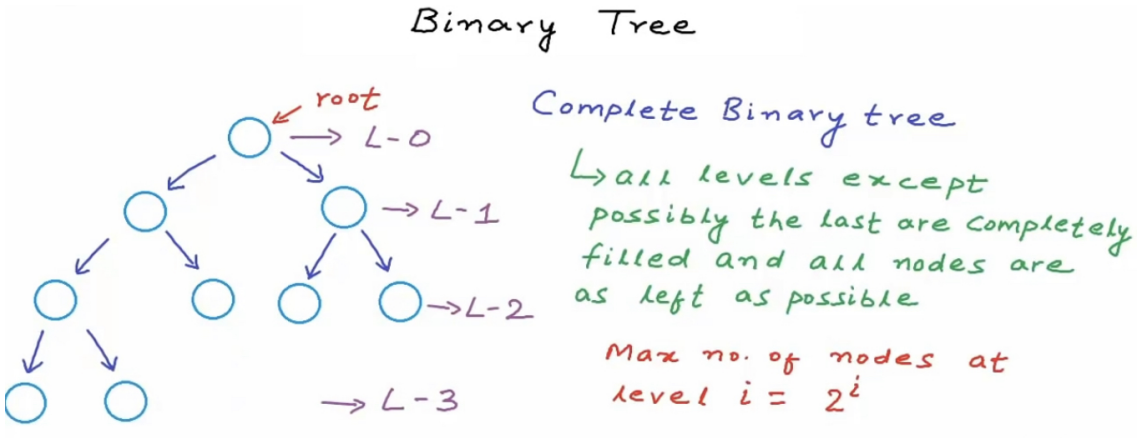
Binary Tree

- For binary tree, what do we call the children nodes?
  - left child
  - right child
- What is a strict/proper binary tree?
  - nodes either have 2 children or 0 children but nothing in between
  - Picture

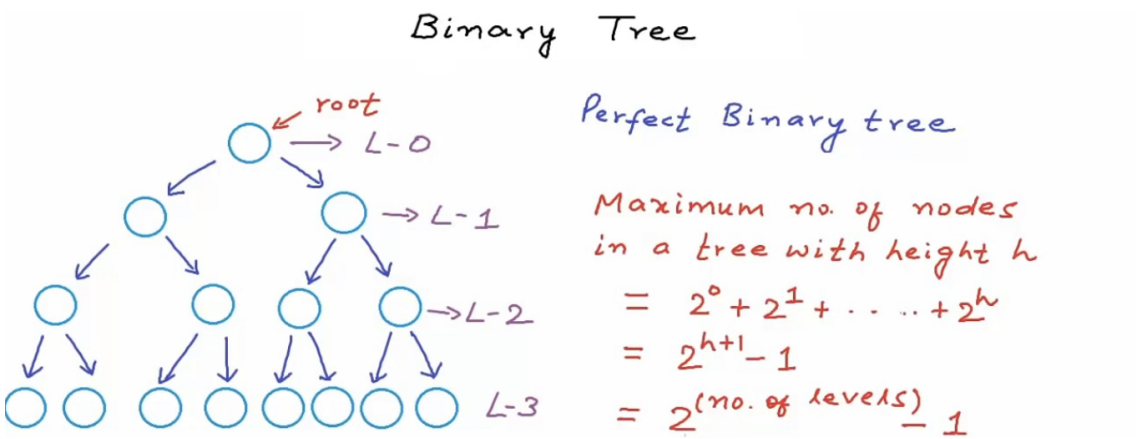




- What is a complete binary tree?
  - all levels except possibly the last are completely filled and if the last level isn't filled it will be as left as possible
  - Maximum number of nodes at level i
    - 2^i nodes
  - Picture



- What is a perfect binary tree?
  - interior nodes all have two child-nodes and whose leaf nodes all have the same depth
  - it legit looks perfect
  - What is the maximum number of nodes in a tree with height h?
    - 2^(h+1)-1
    - taking into account that the root doesn't have 2 nodes
  - What is the height given n number of nodes?
    - log2(n+1) - 1
  - Picture



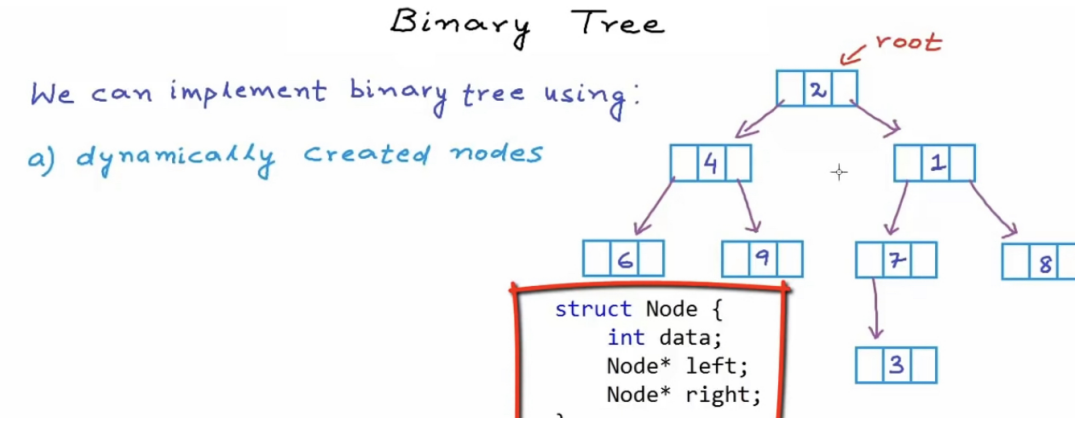
- What is a balanced binary tree?

- binary tree whose nodes all have left and right subtrees whose heights differ by no more than 1

▼ How can we implement binary trees?

1. dynamically created nodes

    ▼ Picture

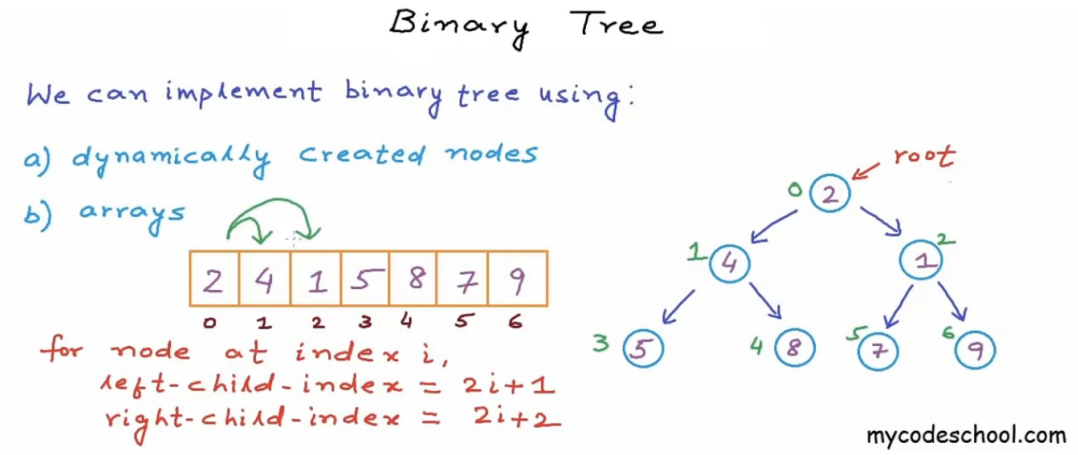


2. arrays

    ▼ How does that work?

    - each index is a node
    - more on how we know left and right in heaps

    ▼ Picture



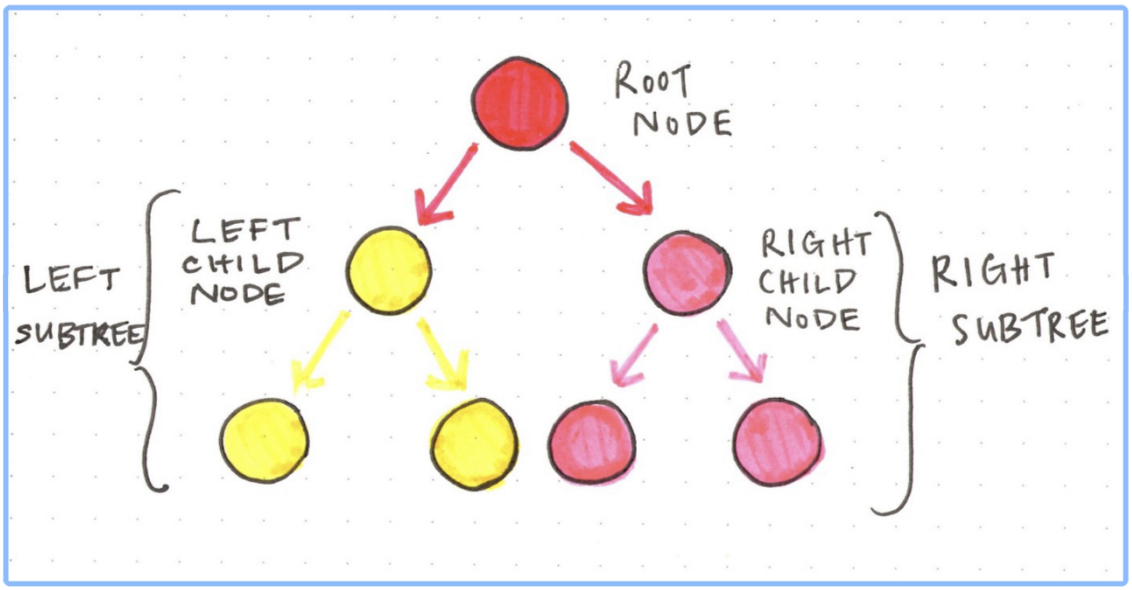
▼ How many subtrees do binary trees have?

- 2

▼ What search algorithm is this important?

- binary search

▼ Picture

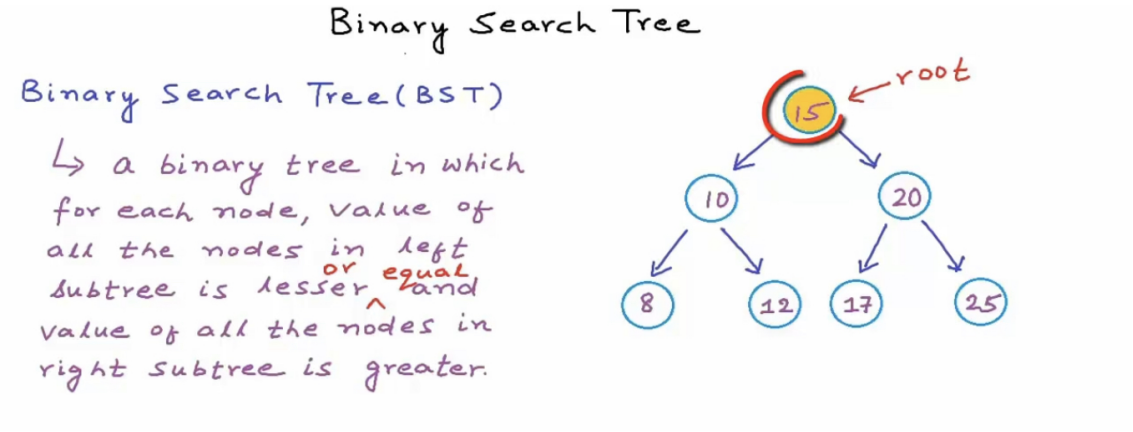


Binary Search Tree

▼ What is a binary search tree?

- a modified version of a binary tree where the value of the nodes in the left subtree is lesser or equal and the value of all the nodes in the right subtree is greater
- each subtree are they themselves their own binary search tree effectively

▼ Picture



▼ BST operations

Search	O(log(n))
Insert	O(log(n))
Remove	O(log(n))
Access	O(log(n))

- average case and only if its balanced for unbalanced search trees you can treat it as basically a linked list
- space complexity is O(H) where h is the height of the tree while traversing very skewed trees

▼ How to find min/max in a binary search tree?

▼ What the ways to do it?

- can use a loop
- can use recursion

▼ What is the logic to find the minimum?

- keep moving pointer left until you hit the null and that is the smallest number

▼ What is the logic to find the maximum?

- keep moving pointer right until you hit the null and that is the largest number

▼ How to do this with recursion?

- keep going down the subtree until its empty and if it is then that node value is the smallest or largest

▼ How do we find height or maximum depth of a binary tree?

▼ What is height?

- number of edges in the longest path from root to a leaf node

▼ How does this work?

- you can use recursion and keep finding a subtree until you hit the null and then return -1. with each subtree take the maximum of the number of edges to the leaf and add +1

▼ How do we check if a binary tree is a binary search tree?

- recursively check if the left node is less than the root and the right node is greater than the root

▼ Delete a node from a binary search tree.

▼ What are the three cases you have to keep in mind when deleting a node?

▼ leaf node

- just delete the node and set the value to Null or None

▼ node with one child

- delete the node and make its child the new parent

▼ node with two children

- take the minimum value in right subtree and assign it to the node you want to delete
- take maximum value in the left subtree and assign it to the node you want to delete