LeetCamp

A project dedicated to DS&A

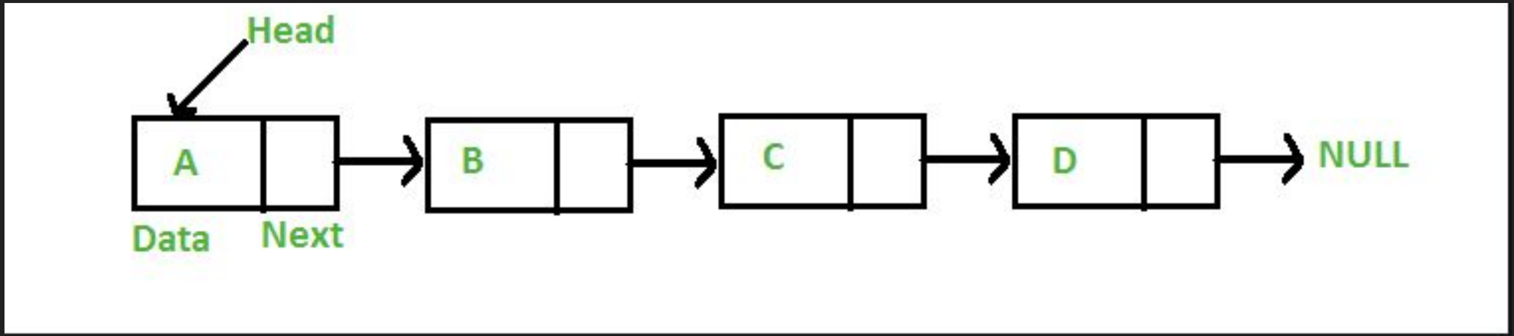
Agenda

1. Quick Review
2. Binary Trees
3. DFS Fundamentals
4. LeetCode Problems

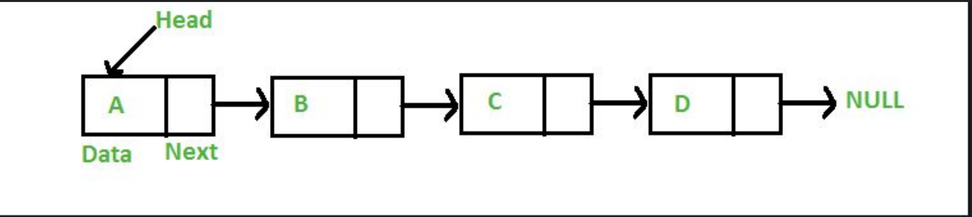
Quick Review

Linked Lists

- A linked list is a **linear data structure** that is not stored contiguously
    - i.e. the elements in the list are not stored next to each other in memory
- In a singly linked list, each node has data and a pointer to the next node

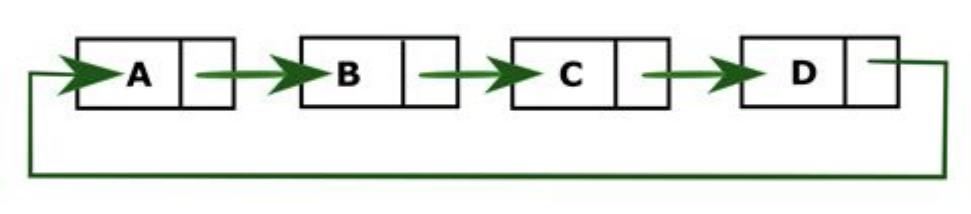


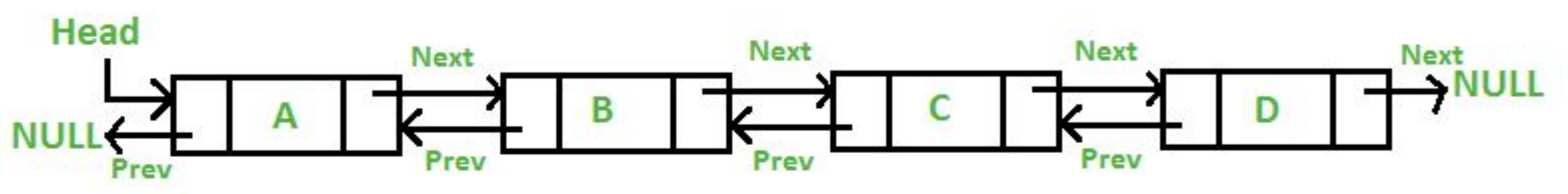
Types of Linked Lists



Single Linked Lists

Circular Linked Lists



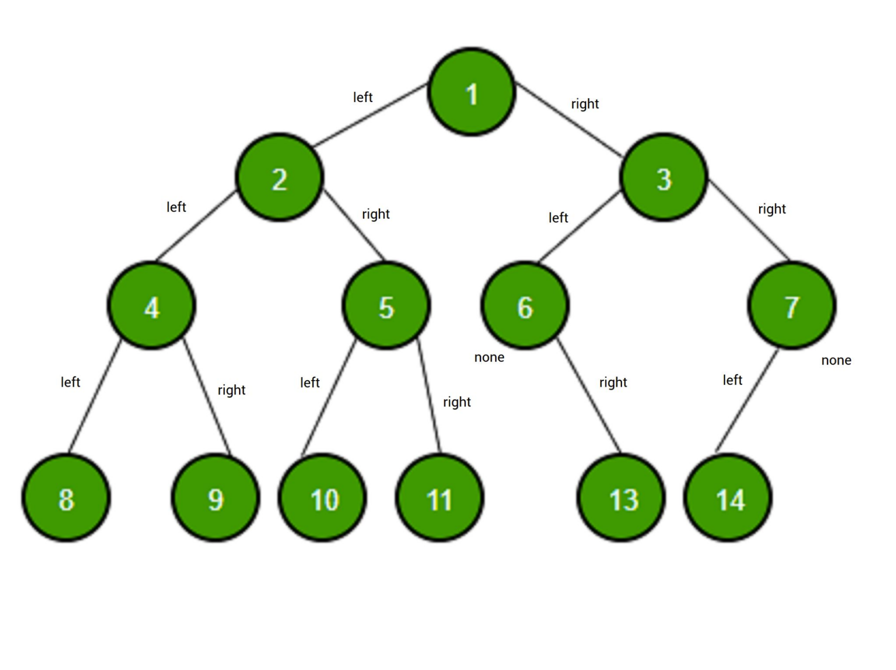


Doubly Linked Lists

Binary Trees

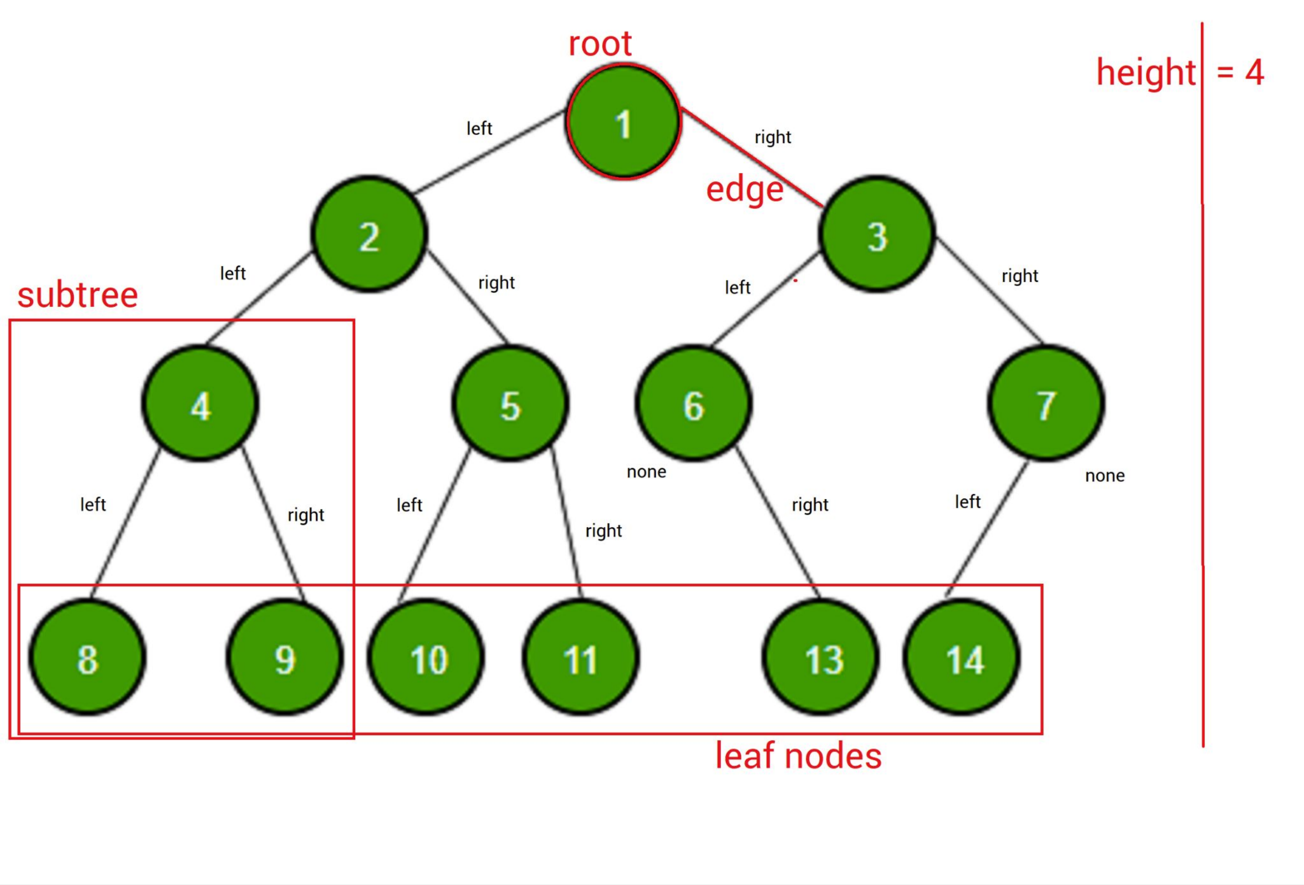
Binary Trees

- A binary tree is represented by a pointer to the topmost node of the tree
- If the tree is empty, then the value of the root is None



Tree Terminology

- Root
- Edge
- Leaves
- Subtree
- Height

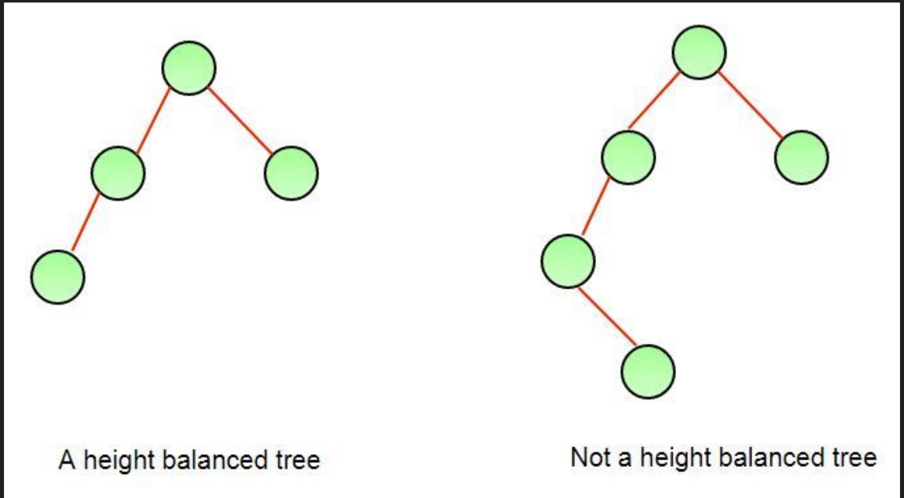


Full, Complete, and Perfect Binary Trees

- Full Binary Tree
  - Each node has 0 or 2 children
- Complete Binary Tree
  - All levels are completely filled except for the last row
  - The last row’s nodes are all as left as possible
- Perfect Binary Tree
  - All levels are completely filled
  - Total Number of Nodes = 2 * Number of Leaf Nodes - 1

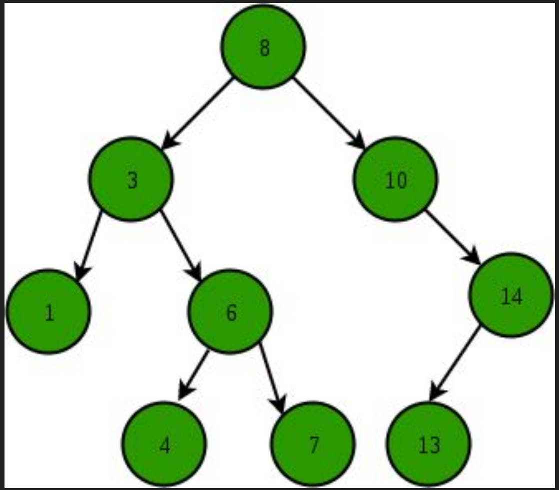
Balanced Binary Tree

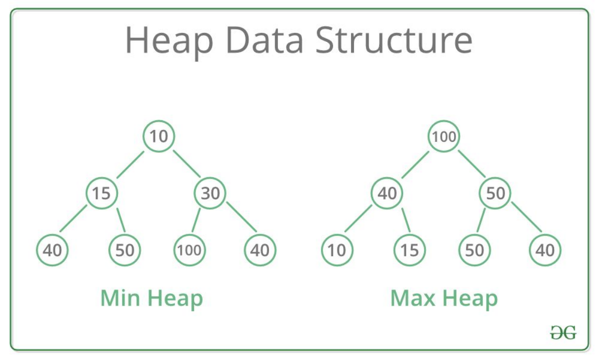
- The height difference between the left and right subtree of the node is not more than 1.
- This property is important for ensuring the effectiveness of our algorithms...

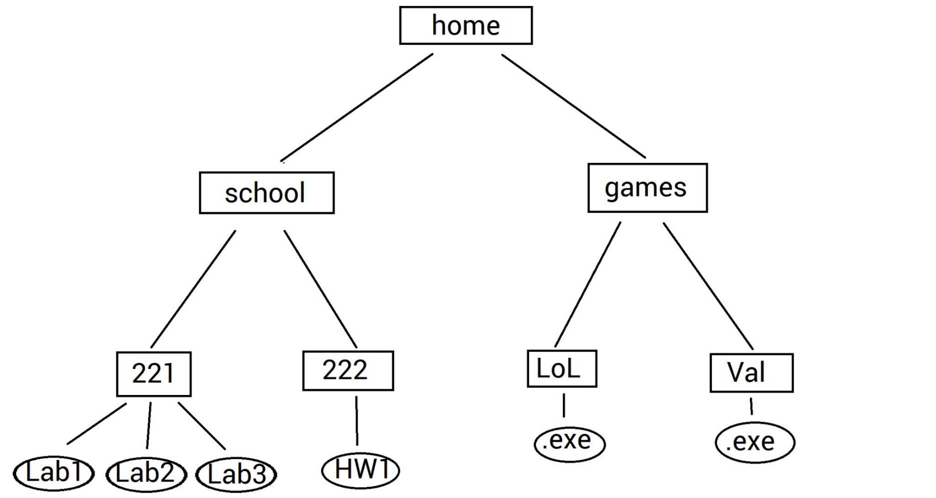


When should I use a tree?

- When making “decisions” in an algorithm
  - I.e. less than or greater than
- Storing hierarchical data
  - I.e. a file system
- Inside of other data structures
  - Heaps
  - Priority Queues







Time Complexity

- **Searching**: Since we haven't introduced special types of trees, searching is worst case O(n)
- **Insertion**: Insertion also has a worst case of O(n), though it is O(h) if the tree is balanced
- **Deletion**: Since we have to search before deleting, it is also O(n)

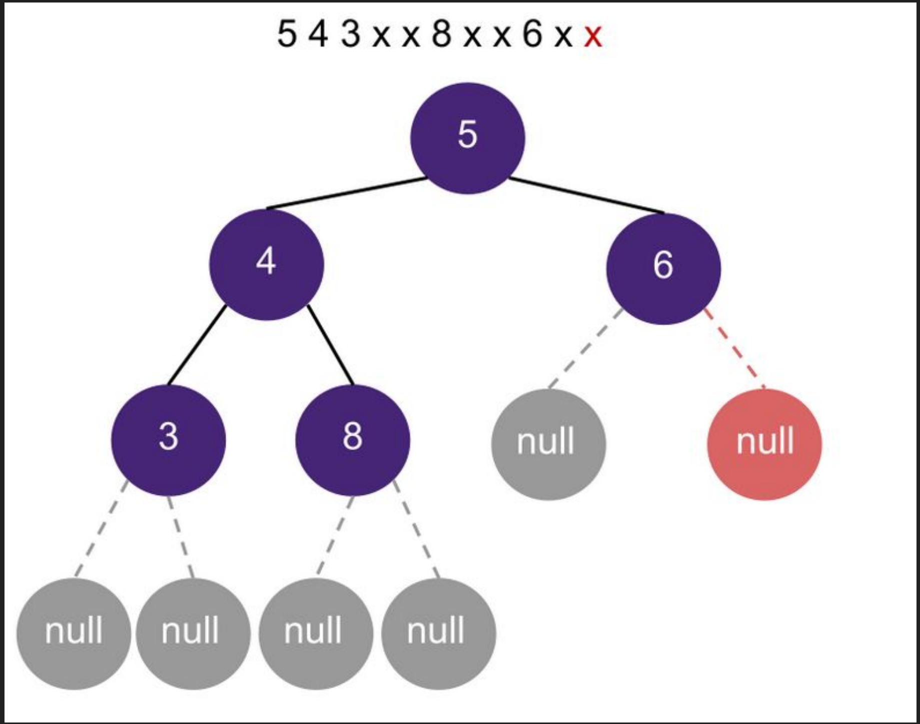
- These time complexities can be improved significantly when we introduce special tree types, such as Binary Search Trees.

Local Setup

- Get code from our new GitHub: https://tx.ag/LCBinaryTree
- Replace "yourFunction" with your function name
- Replace "yourFunction" with your function name in the last line

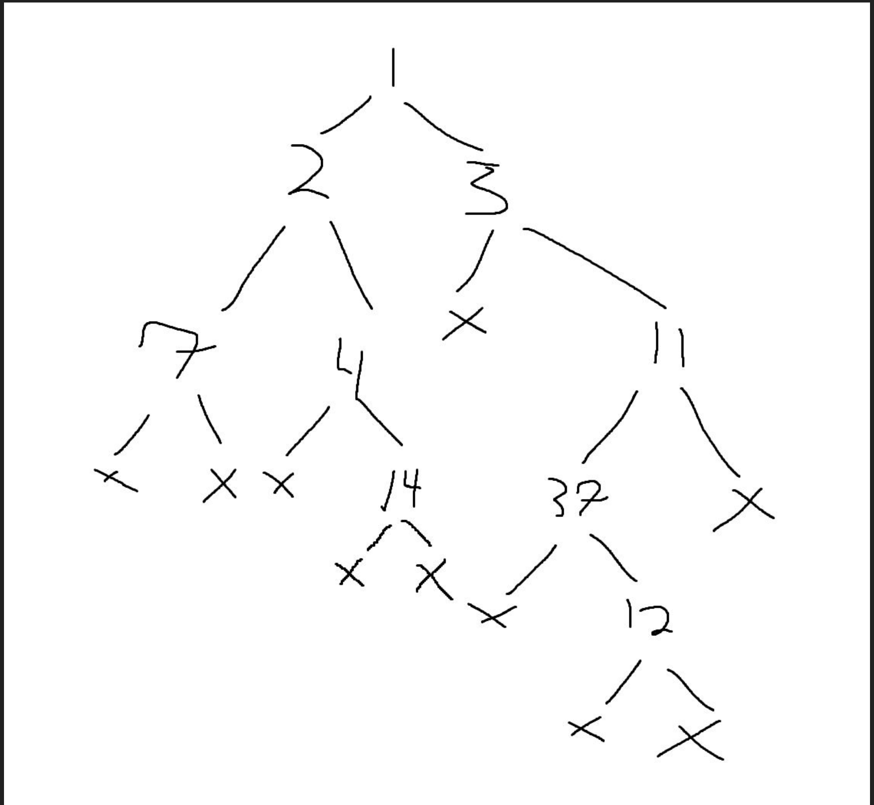
- Input will be given in the form shown on the right —>

Example Input



Another Example of Input

- This is just the way that I have encoded input in the GitHub...
    - You can do this differently if you write your own!

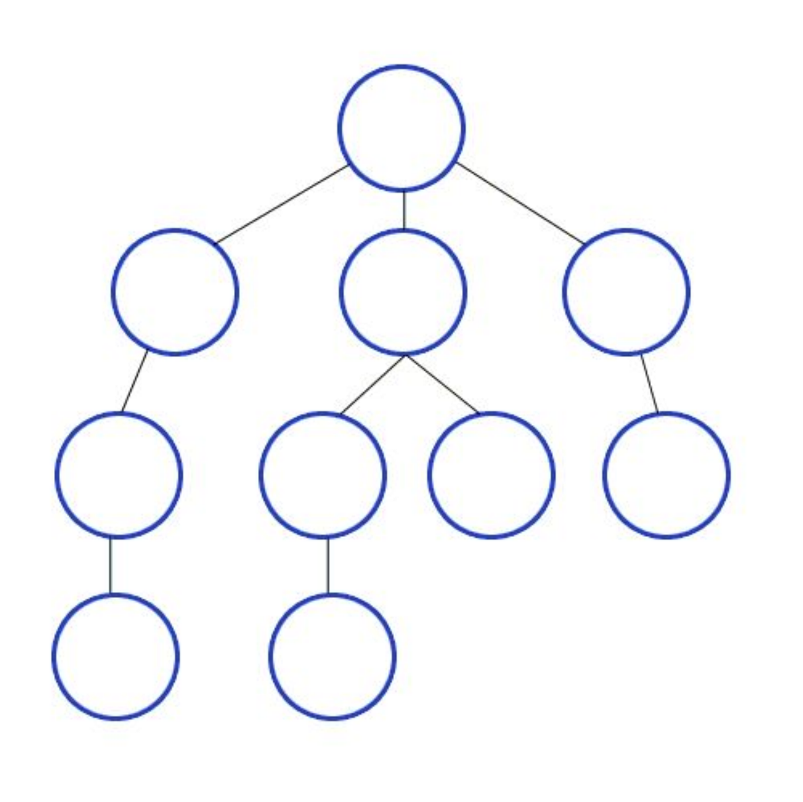


1 2 7 x x 4 x 14 x x 3 x 11 37 x 12 x x x

DFS Fundamentals

Depth First Search (DFS)

- Depth First means we go as deep as we can to look for a value
  - I.e. we go down the rabbit hole as far as possible, then come back up and try again with another rabbit hole
- Two different options
  - Iterative
  - Recursive



Recursive DFS Boilerplate Code

```python
def dfs(root, target):
    if root is None:
        return None
    if root.val == target:
        return root
    # return non-null return value from the recursive calls
    left = dfs(root.left, target)
    if left is not None:
        return left

    # at this point, we know left is null, and right could be null or non-null
    # we return right child's recursive call result directly because
    # - if it's non-null we should return it
    # - if it's null, then both left and right are null, we want to return null
    return dfs(root.right, target)
    # the code can be shortened to: return dfs(root.left, target) or dfs(root.right, target)
```

Iterative DFS Code

```python
def dfsIterative(root: TreeNode, target):
    """Iterative Depth First Search (DFS)"""
    if root is None:
        return
    stack = []
    curr = root
    prev = None
    while stack or curr is not None:  # While stack isn't empty OR curr isn't None
        if curr is not None:
            if curr.val == target:  # if curr node is target, return it
                return curr
            stack.append(curr)  # append curr node to stack
            curr = curr.left  # update curr node to curr.left
        else:
            prev = stack.pop()  # pop node off of stack
            curr = prev.right  # update curr to prev.right
    return # target is not in the tree
```

LeetCode Problems

Example Problem #1 - LeetCode 104.

- https://leetcode.com/problems/maximum-depth-of-binary-tree/

- Can we implement what we have learned to solve this problem?
  - DFS will work
    - You can use an iterative or recursive solution
  - Other algorithms will also work, but we haven't talked about them yet.
- Time Complexity?
  - O(n)
- Space Complexity?
  - O(1)

Group Work (if there is time)

Until next time...
Keep practicing

Practice Problems

Easy (supposedly):

- https://leetcode.com/problems/invert-binary-tree/
- https://leetcode.com/problems/subtree-of-another-tree/

A bit more difficult:

- https://leetcode.com/problems/binary-tree-maximum-path-sum/

Quite difficult (but also cool):

- https://leetcode.com/problems/serialize-and-deserialize-binary-tree/

Questions?