Trees

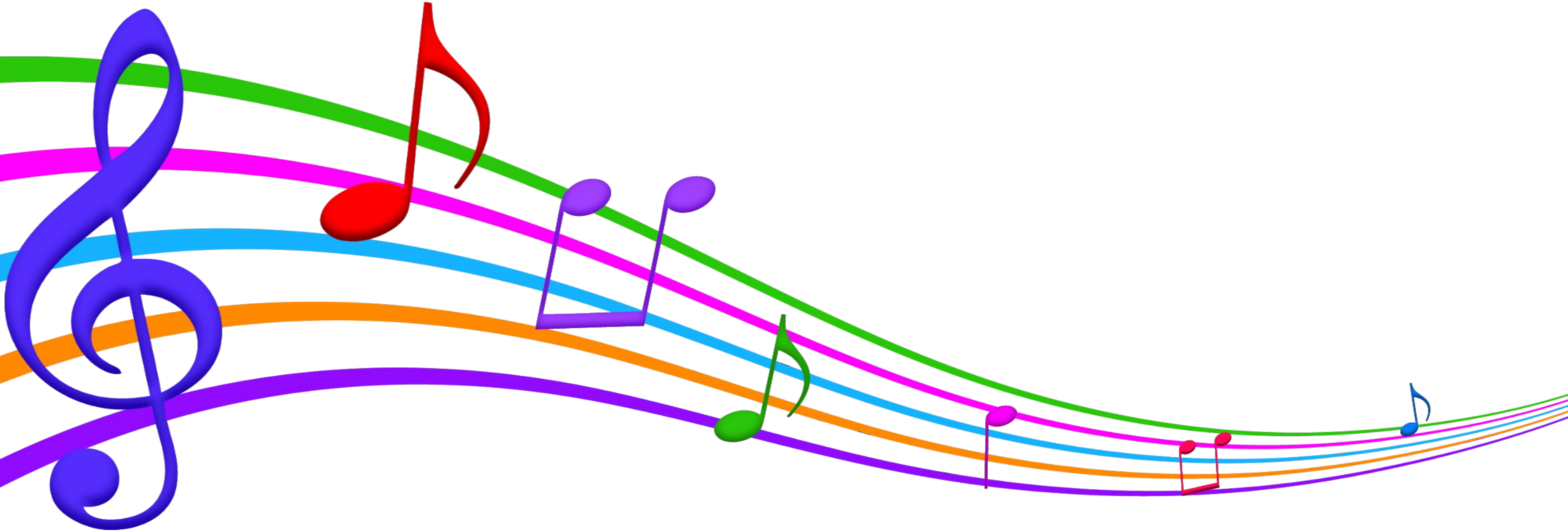
Learning Outcomes

- Be able to explain what it and isn’t a tree
- Become familiar with tree vocabulary
- Be able to explain what a Binary Search Tree is
- Be able to explain binary search tree methods

Check In

MAKE SCHOOL

What’s a song that you like to relax to?



Review

Draw a hash table with 4 buckets that uses chaining for collision resolution. The hash function will return the length of the key, for example for the key “Tara” the hash function will return 4.

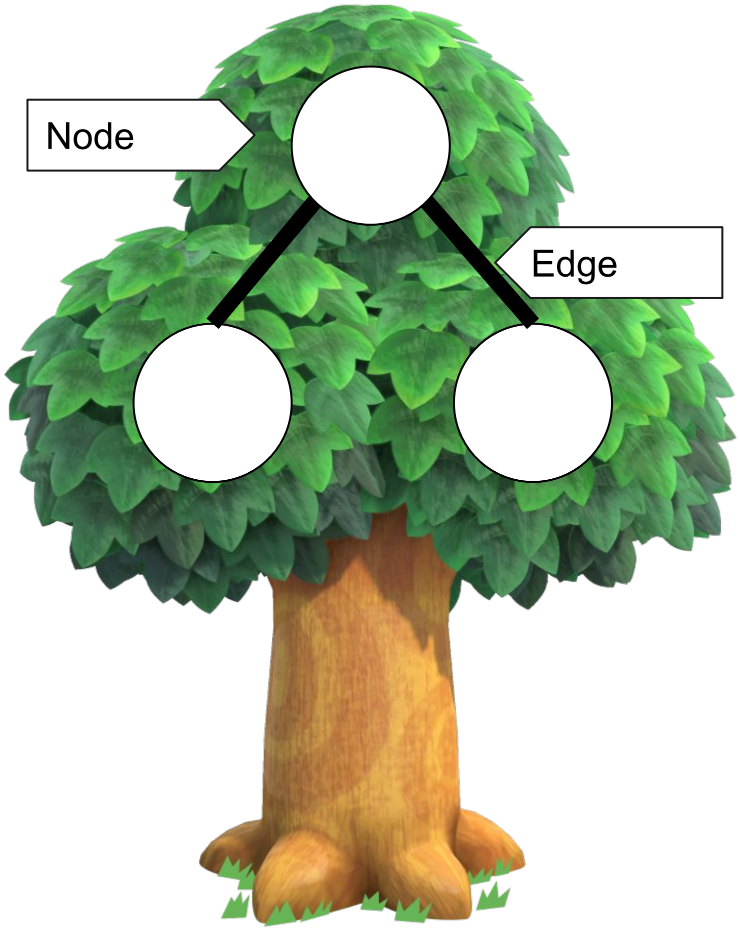
In your hash table insert these key value pairs: “Tara”:4, “Dagon”:2, “Nova”:4

Review

True or False: to retrieve a value from the hash table we need to search through all the buckets

Basic Trees

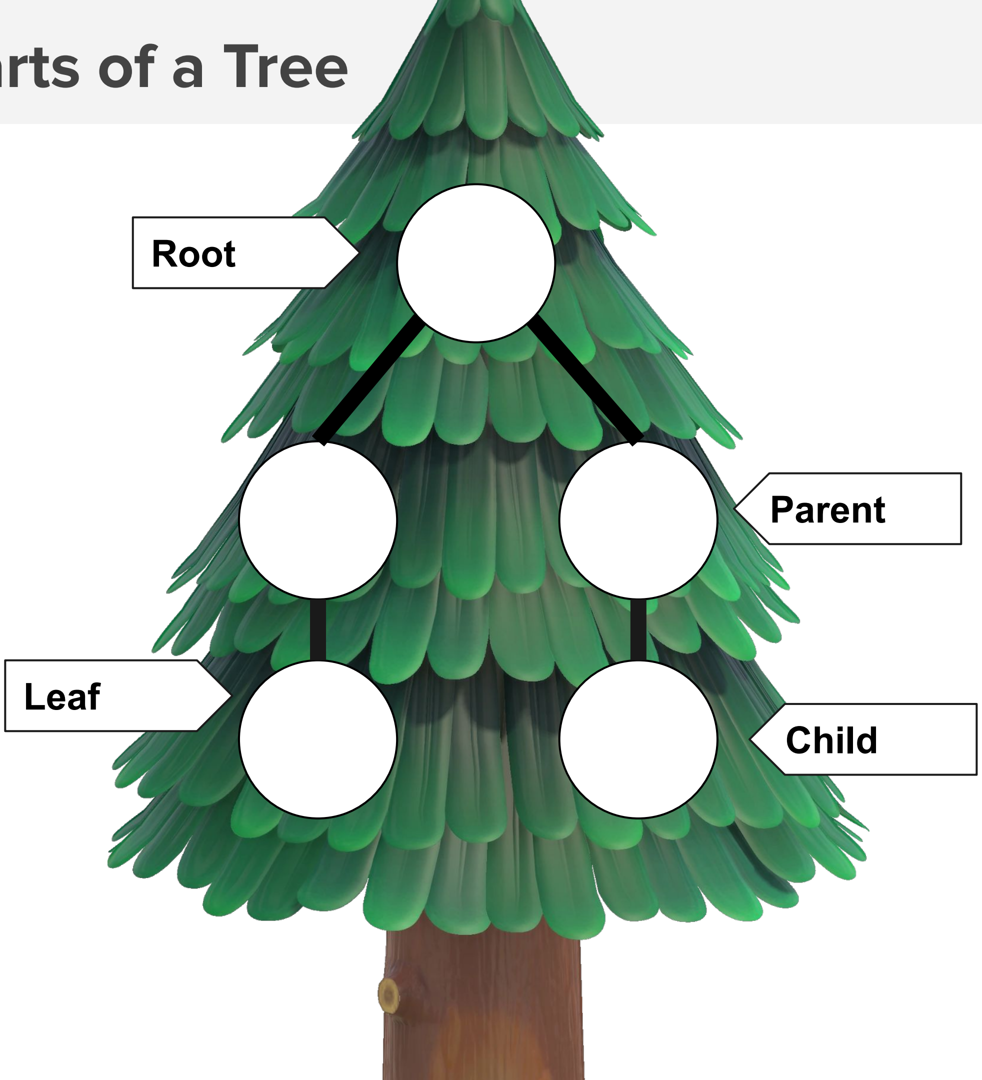
Parts of a Tree



These trees are Valid

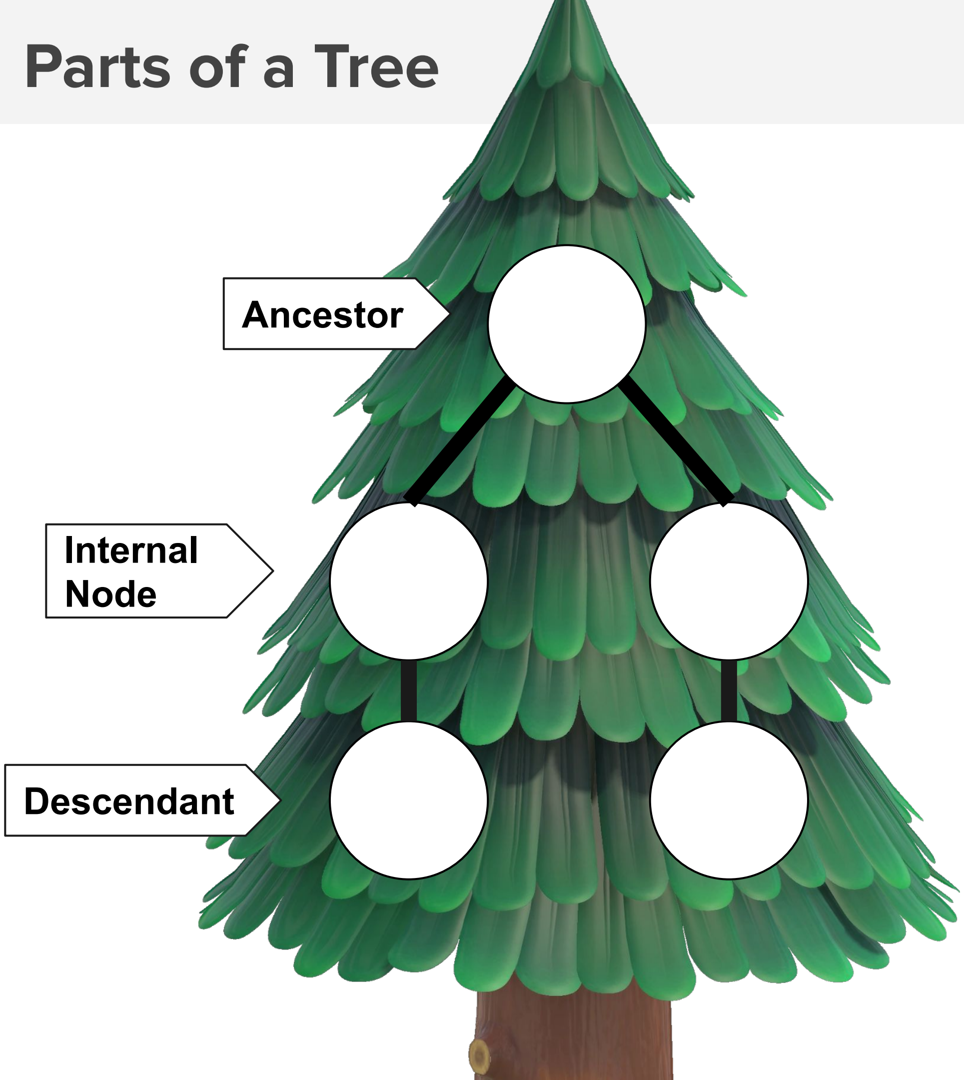
Trees don’t have cycles

Parts of a Tree



Root	**Topmost node**
Parent	**Node that comes before**
Child	**Node that comes after**
Leaf	**Node with no children**

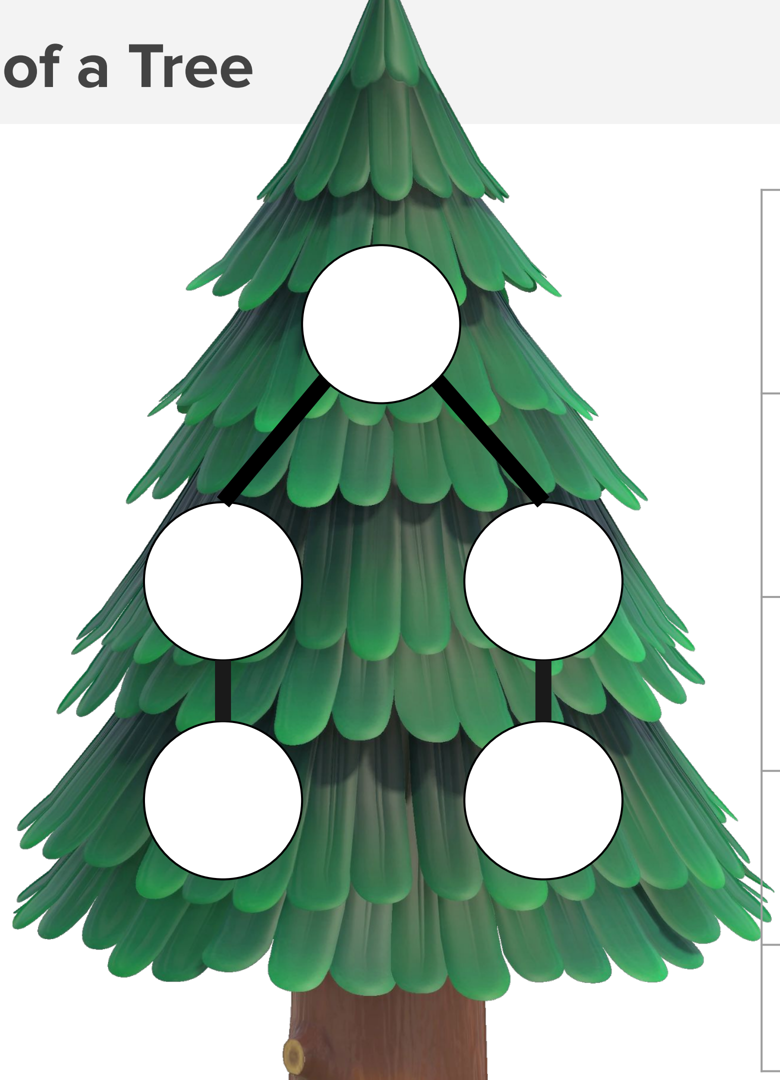
Parts of a Tree



Ancestor	Node reachable from child to parent
Descendant	Node reachable from parent to child
Internal Node	Node with at least one child

Parts of a Tree

MAKE SCHOOL

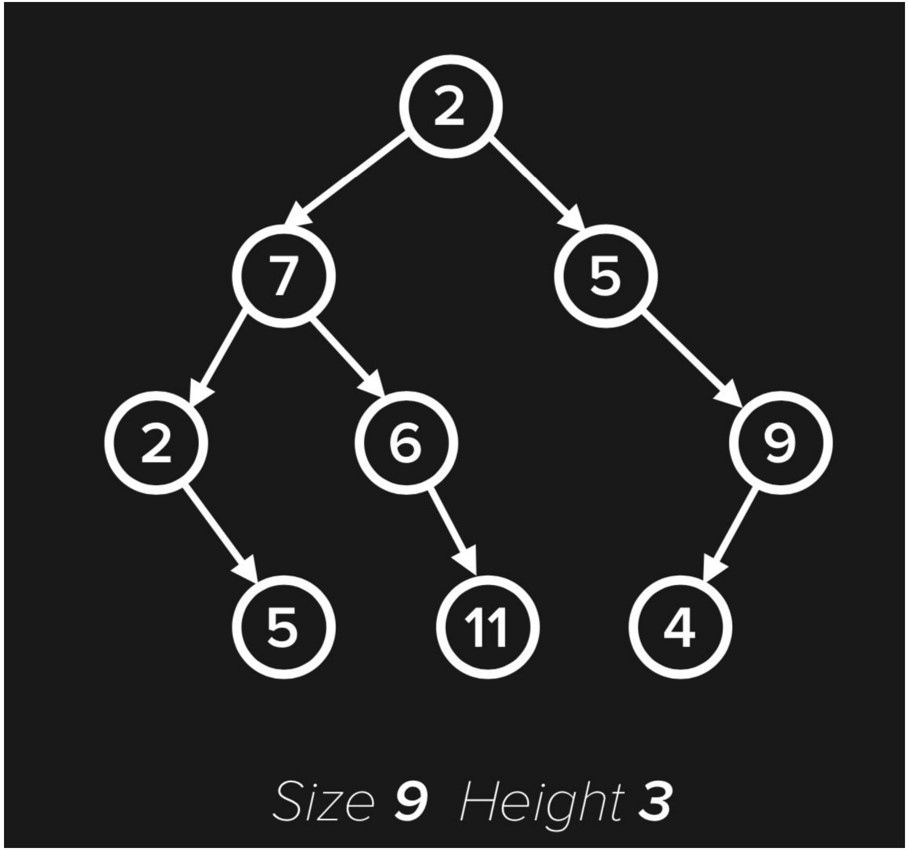


Height (tree)	number of edges on longest downward path from root to leaf
Height (node)	number of edges on longest downward path from node to leaf
level	1 + number of edges between the node and the root
Depth	number of edges between the node and the root
Size	number of nodes in the tree

Binary Trees

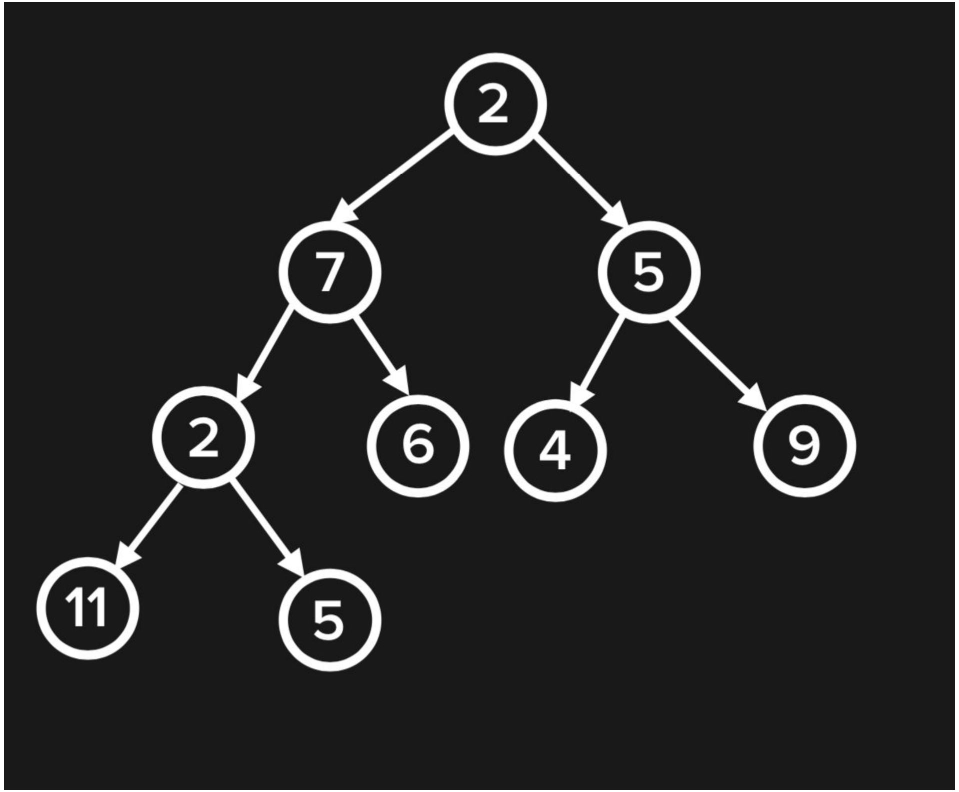
Binary Tree

A tree in which each node has at most two children



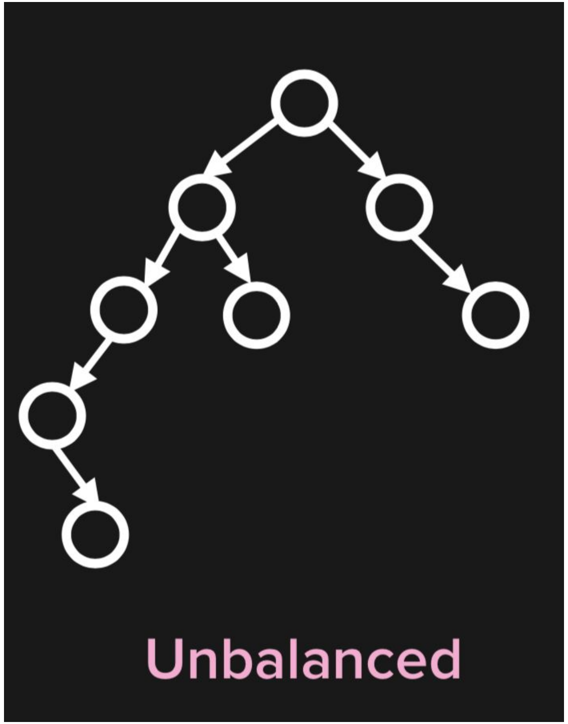
Complete Tree

Every level except possibly last is completely filled and nodes are as far left as possible



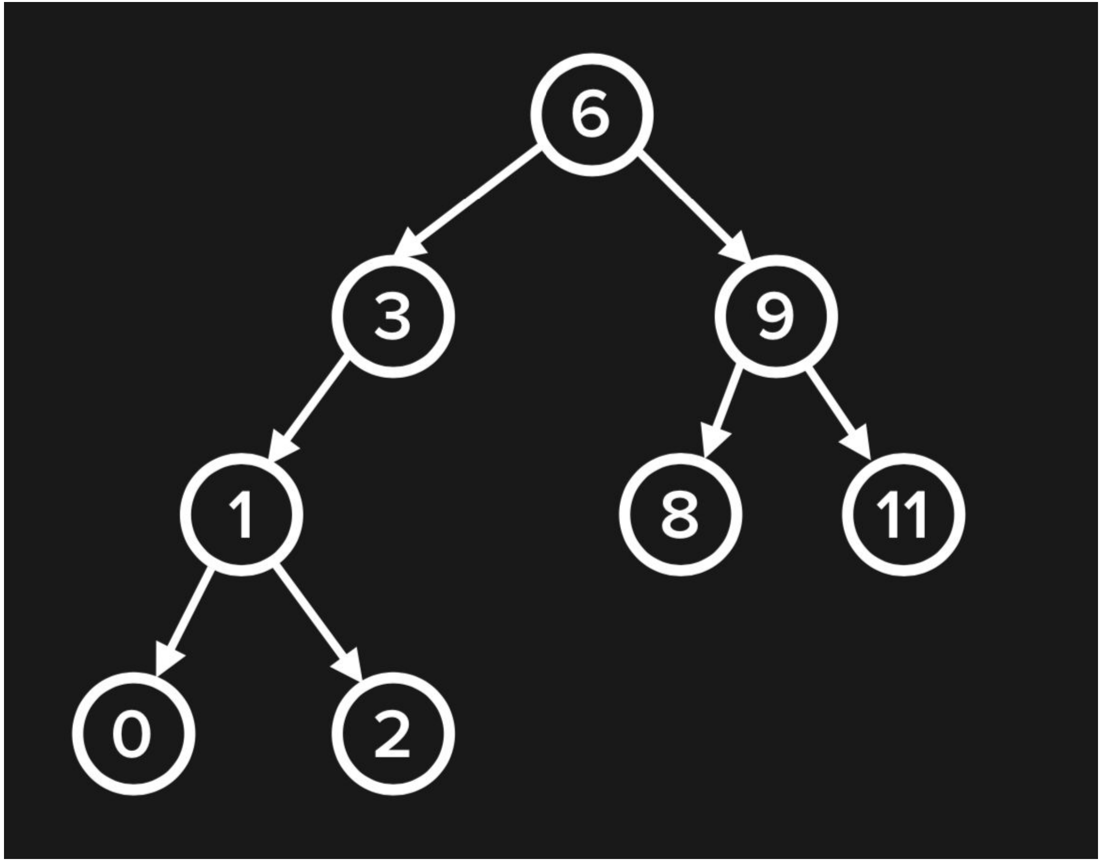
Balanced Tree

All leaves are at minimum possible depth



Binary SEARCH Tree

- Always sorted
- For each node:
  - Left children are smaller
  - Right children are larger
- No duplicate keys



Why Use a BST? What do you think it's good at?

Students, write your response!

Pear Deck Interactive Slide
Do not remove this bar

Why Use a BST?

MAKE SCHOOL

- Fast search, insertion, deletion - especially when balanced
- Sort as you go instead of all at once
- Fairly simple implementation for good performance
- Only allocates memory as it's needed
- Doesn't have to reallocate memory to grow (like a hash table)

Tree Applications

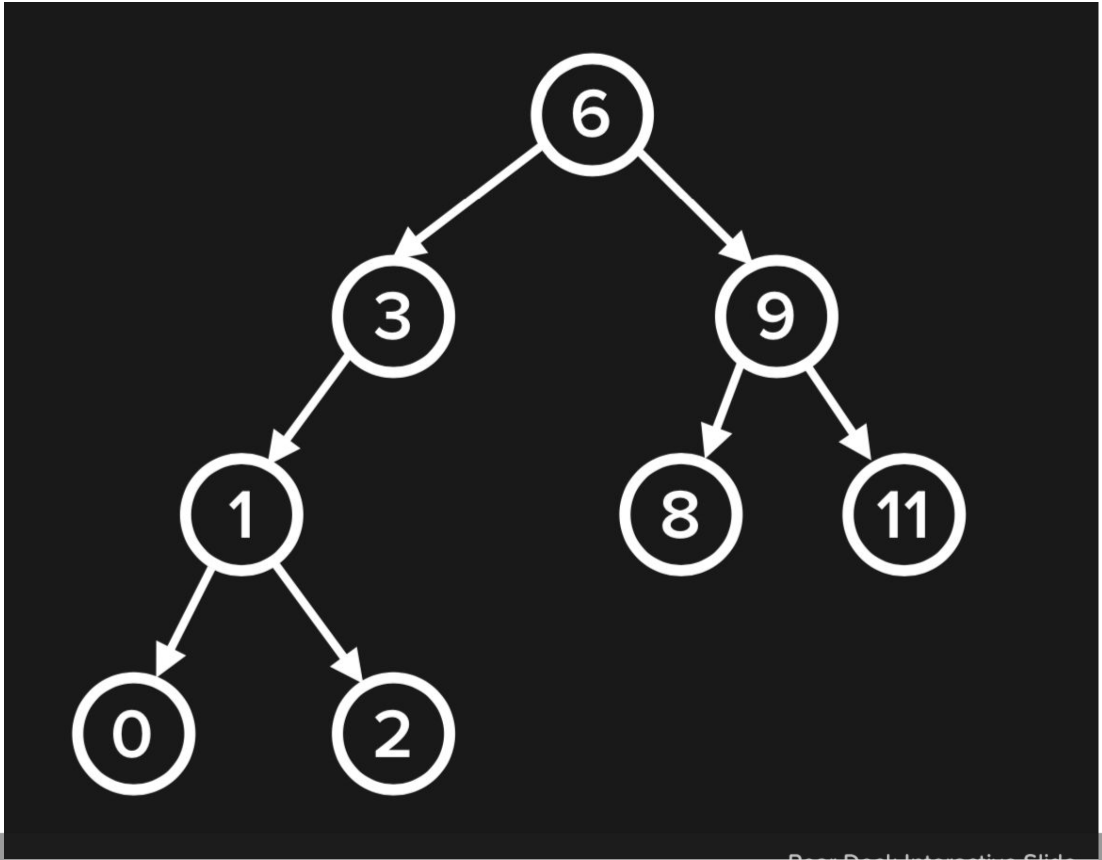
MAKE SCHOOL

- Binary Space Partitioning in Doom
- Search Applications
- Routing
- Syntax parsing
- Databases

How to SEARCH?

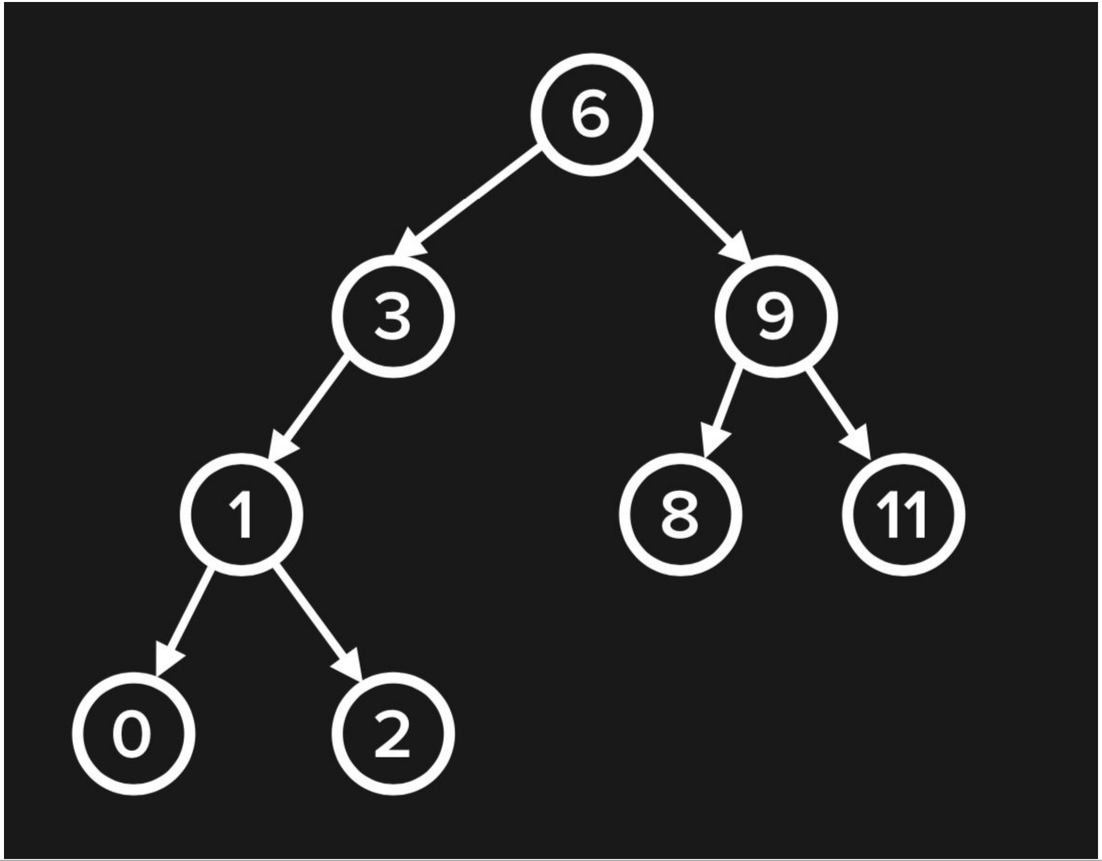
MAKE SCHOOL

- Always sorted
- For each node:
  - Left children are smaller
  - Right children are larger
- No duplicate keys
- How might you approach search?



Students, write your response!

How to SEARCH?



BST Methods: Search

```python
#  call initially with node == root node
def find_recursive(key, node):
    if node is None or node.key == key:
        return node
    elif key < node.key:
        return find_recursive(key, node.left)
    else:
        return find_recursive(key, node.right)
```

BST Methods: Insert

MAKE SCHOOL

It’s the same as search except once you find a node without a child on the next side you’re traversing, add it there

BST Methods: Delete

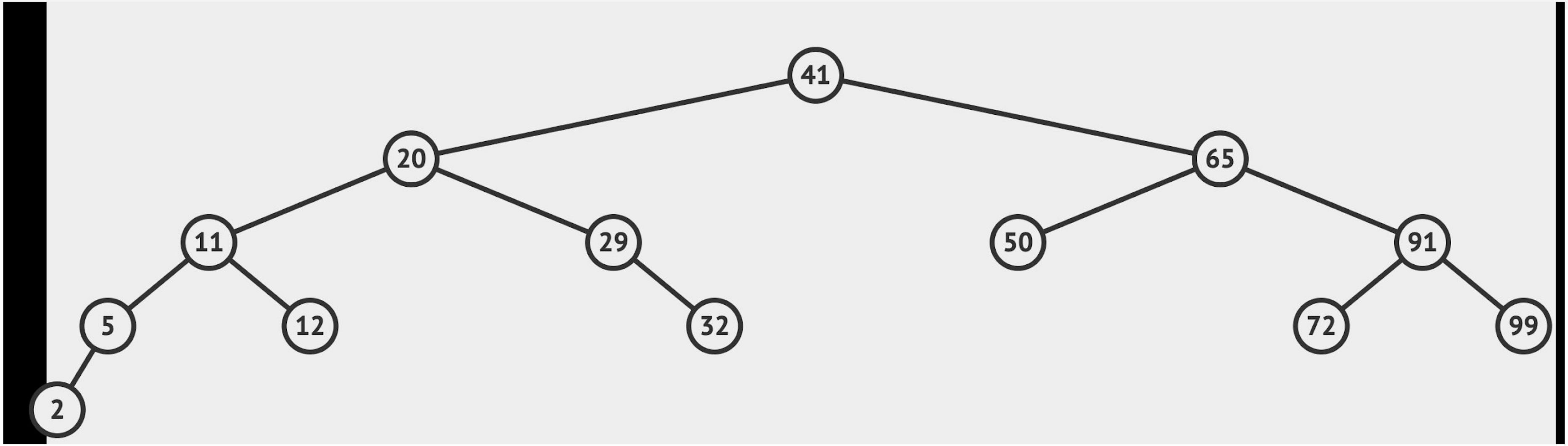
How do you think delete might work?

Check Your Understanding: Draw a valid BST with 5 nodes

Students, draw anywhere on this slide!

Check Your Understanding

MAKE SCHOOL



Answer these questions about the properties of the given tree:

1. What kind of node is 2?
2. What kind of node is 11?
3. What is the height of the tree?
4. What is the height of 50?

5. What is the depth of 91?
6. Which nodes are 32’s ancestors?
7. What is the size of the tree?
8. What level is 29 on?

Let’s build a basic BST

Students browse: repl.it/@MakeSchool/basicbst?lite=true

Binary Search Tree Methods

Search

Check if the current node is None or if it contains the key

If not, if key is less than current node's key:

    Search the left subtree

If key greater than current node's key:

    Search the right right subtree

https://visualgo.net/bn/bst

Search

```
#  call initially with node == root node
def find_recursive(key, node):
    if node is None or node.key == key:
        return node
    elif key < node.key:
        return find_recursive(key, node.left)
    else:
        return find_recursive(key, node.right)
```

Insert

Check if tree is empty

If not, find parent will return parent of node we want to insert

Insert as left child if smaller than parent

Insert as right child if greater than parent

https://visualgo.net/bn/bst

Delete

MAKE SCHOOL

3 cases

1. Delete a leaf
2. Delete with one child
    a. “Skip over”
3. Delete with two children
    a. Find the in-order successor and replace
    b. (smallest in the right subtree)

https://visualgo.net/bn/bst

Tree Traversals

MAKE SCHOOL

What do you think the goal of a traversal is?

Students, write your response!

Traversal

MAKE SCHOOL

Goal - visit each node once and only once

Three operations:

visit current node

traverse to **left** node

traverse to **right** node

Two Types

- Depth First Search (DFS)
  - Down first - visit child, then next descendent
- Breadth First Search (BFS)
  - Across first - visit all siblings before going deeper

Depth First Search

Always go left before going right

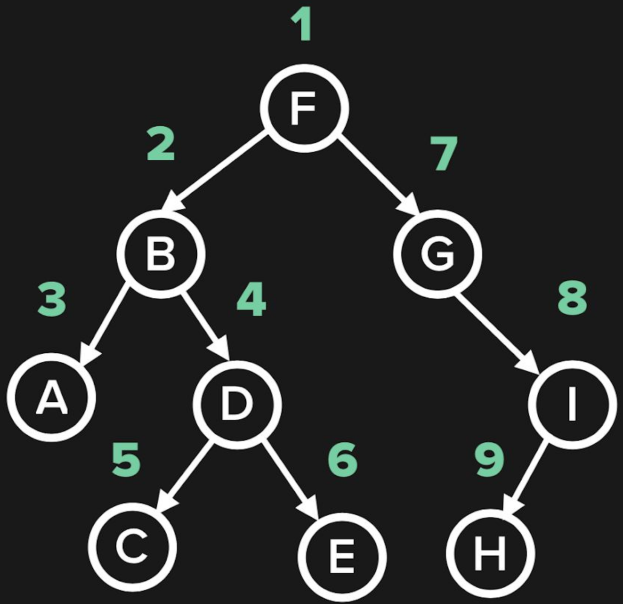
Three types of visitation:

1. Pre-order: Copy the tree
2. In-order: get underlying values in order
3. Post-order: delete tree from leaves to roots

Pre-Order DFS


```python
def pre_order_dfs(node):
    if node is not None:
        visit(node)
        pre_order_dfs(node.left)
        pre_order_dfs(node.right)
```

F B A D C E G I H



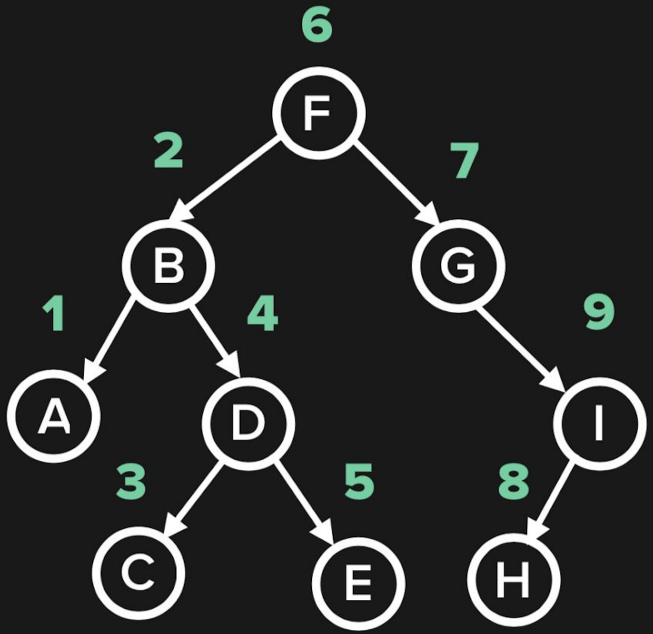
Let’s look at a different way to visualize!

In-Order DFS

MAKE SCHOOL

```
def in_order_dfs(node):
    if node is not None:
        in_order_dfs(node.left)
        visit(node)
        in_order_dfs(node.right)
```

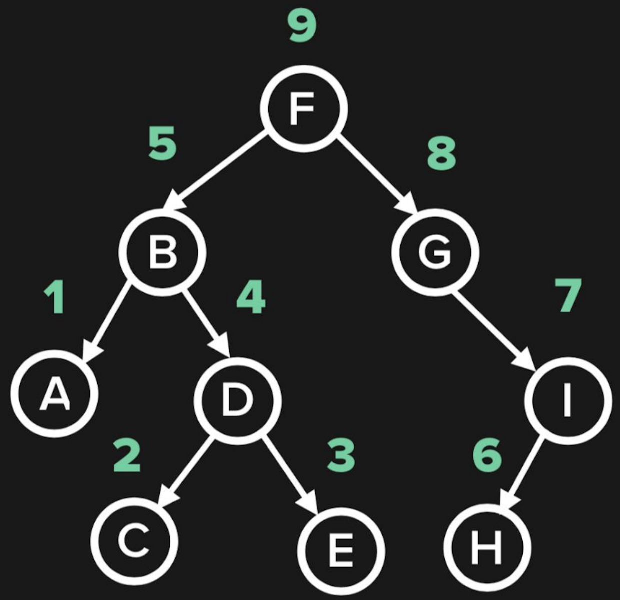
A B C D E F G H I



Post-Order DFS

POST-ORDER DFS

```
def post_order_dfs(node):
    if node is not None:
        post_order_dfs(node.left)
        post_order_dfs(node.right)
        visit(node)
```



A C E D B H I G F

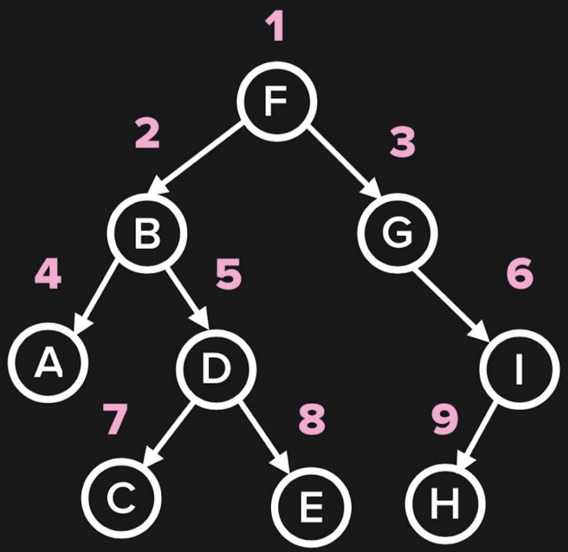
Breadth First Search

MAKE SCHOOL

BREADTH-FIRST SEARCH

```
from collections import deque

def bfs(root_node):
    queue = deque()
    queue.append(root_node)
    while len(queue) > 0:
        node = queue.popleft()
        visit(node)
        if node.left is not None:
            queue.append(node.left)
        if node.right is not None:
            queue.append(node.right)
```



F B G A D I C E H

Shout Outs