7.1 binary trees and searching

Tuesday, October 18, 2022

Today:
- 20 questions game
- introduction of trees, binary trees, and binary search trees
- (if time) dictionary ADT, keys, values

Reminder: lab 6 is due Oct 26th (next week) and has three parts:
1. implement stacks and queues
2. solve mazes using BFS and DFS
3. induction proofs

You must come to lab this week unless your team has completed and pushed *all parts*.

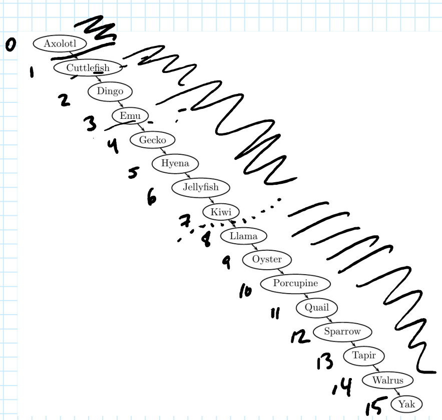
Think of your favorite animal.

I have a list of animals I can think of:

- axolotl
- yak
- cuttlefish
- walrus
- dingo
- tapir
- emu
- sparrow
- gecko
- quail
- hyena
- porcupine
- jellyfish
- oyster
- kiwi
- llama

(I'm storing this in an arrayList or LinkedList, probably.)

How can I get better (faster) at finding if your favorite animal is on my list?
(by changing my list or my strategy)



Binary search for X

① If X is in the list, it must be somewhere 0…15

$\left\lfloor \frac{0+15}{2} \right\rfloor = 7$ array[7] = kiwi
X is alphabetically < kiwi

② If X is in the list, it's in 0…6

$\frac{0+6}{2} = 3$ array[3] = Emu
X is < Emu

③ If X is in the list, it's in 0…2

$\frac{0+2}{2} = 1$ array[1] = Cuttlefish *
X is < Cuttlefish

④ If X is in the list, it's in 0…1

$\left\lfloor \frac{0+1}{2} \right\rfloor = 0$ array[0] = axolotl
X > axolotl *

⑤ If X is in the list, it's after index 0 before index 1
return "X not found!"

Once the data is in sorted order, I can use binary search:

idea: repeatedly reduce the size of search space by half (until you find element or determine it's not there)

Worst case takes $O(\log_2(n))$ where n = size of list

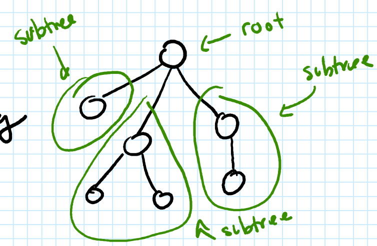
$\log_2(1 \text{ million}) \approx 20$

$\log_2(1 \text{ billion}) \approx 30$

So far all our data structures have been LINEAR.
We'll use the idea of binary search to build
our first HIERARCHICAL data structure: a TREE.

TREE:

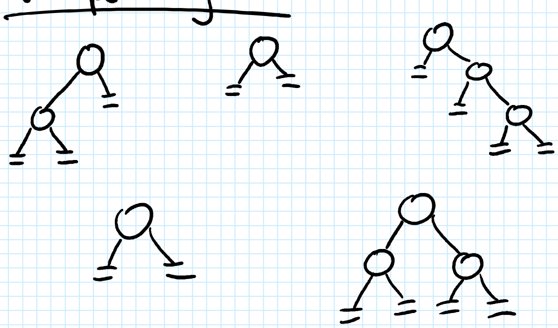
- collection of nodes
- in general, each node may have any number of branches
- recursive structure



We will focus on BINARY TREES:

either: empty
or: consists of a node that contains links to 2 trees below it, called left and right

example binary trees:



(Usually we won't bother to draw all the null pointers.)

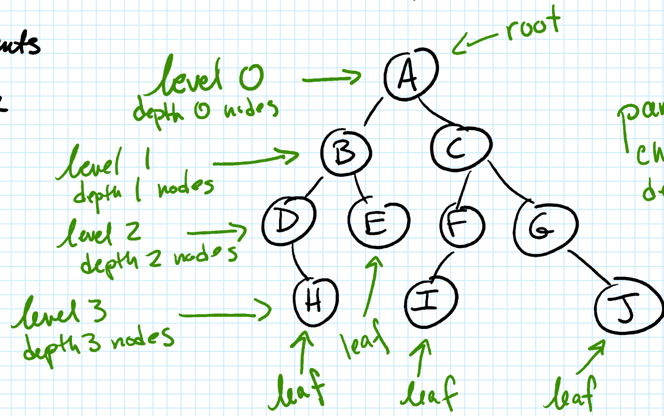
Terminology

node — contains { data ← called "key"; pointers to left & right subtrees }

children (of a node) — the nodes directly linked below (pointers to left & right subtrees

children (of a node)	the nodes directly linked below
parent (of a node)	the unique node directly above
root (of a tree)	the unique node with no parents
size (of a tree)	number of nodes in the tree
leaf	a node with no children
depth (of a node)	how far node is from root (number of links you have to traverse) (root is depth zero)
height (of a tree)	the max depth of any node in the tree (tree with only root → height zero) (empty tree → height −1)

example — This one is a binary tree.



parent of F is C
child of F is I
depth of F is 2

height = 3
size = 10

If we just put all animals in the tree structure arbitrarily, that won't help make the search more efficient.

We need a special kind of tree:

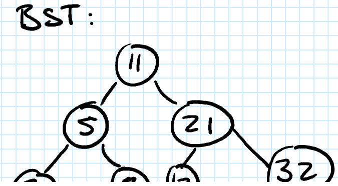
BINARY SEARCH TREES!

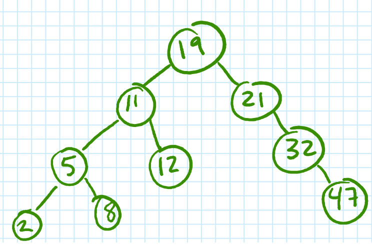
— a binary tree
has a special property governing all keys stored within nodes,
property is true at all nodes & helps with searching

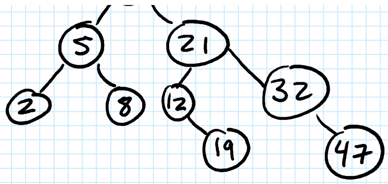
BINARY SEARCH TREE PROPERTY

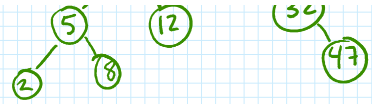
All keys in the left subtree of a node must be less than the key at that node.
All keys in the right subtree of a node must be greater than the key at that node.

example BST:



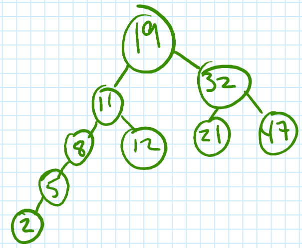


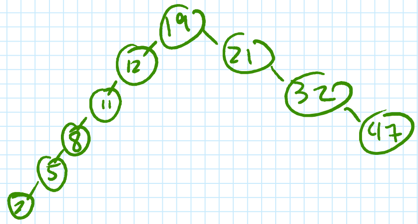




Make another BST with 19 at the root:

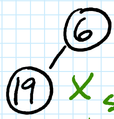
2, 5, 8, 11, 12, 19, 21, 32, 47





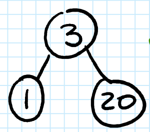
Which of these are valid BSTs?

A



X 19 > 6 so it needs to be in the right subtree

B

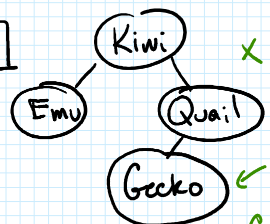


✓

C

✓

D

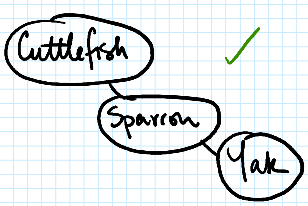


X

Gecko is in the wrong place

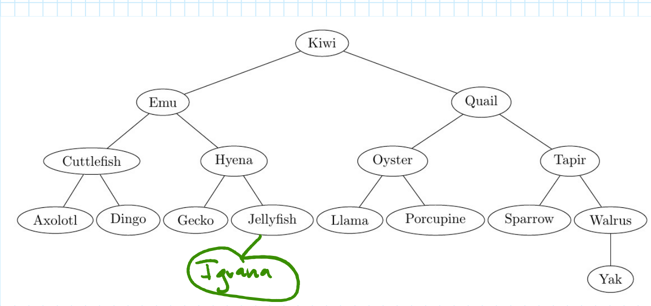
Gecko < Kiwi

E



✓

Let's do our binary search again, but this time with a BST:



Kiwi? no, after

Quail? no, before

Oyster? no, after

Porcupine? yes!

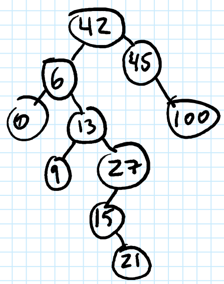
Note: search takes O(height of tree) many questions.

How to add a new key to the tree?

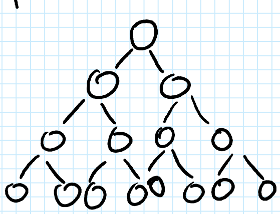
How to add a new Key to the tree?

ex: Iguana : do the same steps as search, add a new node once we reach an empty tree.

Create a BST from these keys by inserting them in order: 42, 6, 13, 45, 27, 9, 15, 0, 21, 100



Best possible structure is a fully-packed tree:



level 0

level 1

level 2

level 3

\# nodes = 15

$2^{(\text{height}+1)} - 1$ = # nodes in a full binary tree of that height

If the tree has n nodes:

best case height: $O(\log_2(n))$ fully packed

worst case height: $O(n)$ is basically a linked list

Questions to think about:

- If we don't know all the keys beforehand, how do we keep the BST balanced as we insert new keys?
- What should we do if we want to remove a key from the BST?

implementation details:

Linked BST Node

K key	V value
left	right

contains 4 data members

```
private: // data
    K key
    V value
    LinkedBSTNode<K,V> *left
    LinkedBSTNode<K,V> *right
public: // methods
    getKey, setKey
    getValue, setValue
    getLeft, setLeft
    getRight, setRight
```

Linked BST

private: //data

Linked BSTNode<k,v>* root

int size

Note: you can reach the entire tree by starting at the root.