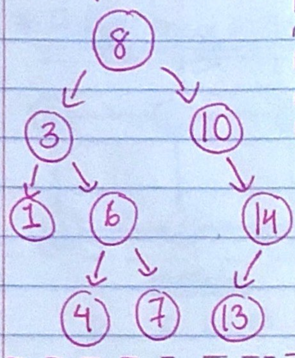
Binary Trees!

Example Tree:



Tree Tips:

* trees directed acyclic graph (non-circle, never twice)
* a binary tree each node has 2 children (might be null)
* Binary Search Tree: all elements to the left, < n value
  " " right, > n value
* In the examples we have, there will be be duplicate values.

Traversals:

Pre: 8,3,1,6,4,7,10,14,13
In: 1,3,4,6,7,8,10,13,14
Post: 1,4,7,6,3,13,14,10,8

Patterns:

- Preorder: root left right
- Inorder: left root right
- Postorder: left right root

runtime: (All) $\Theta(n)$

More Definitions regarding height, trees, root, etc.

Height of a Tree: number of edges from root → deepest leaf (3)
- height root w/ no children = 0
- height w/ nullptr = -1

Internal Nodes: nodes w/ at least one non-null child

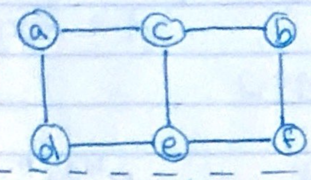
Leaves: nodes that have 2 null children

Max width: number of nodes that contain most nodes

Diameter: number of nodes on longest path between 2 nodes

Test 2 Topics

Graphs: * Adjacency lists vs. Matrices: Adjacency lists is made up a collection of linked lists
Adjacency Matrix → undirected always symmetric. nxn boolean one row/one col for each val



undirected Graph!

matrix

$$
\begin{array}{c|cccccc}
 & a & b & c & d & e & f \\
\hline
a & 0 & 0 & 1 & 1 & 0 & 0 \\
b & 0 & 0 & 1 & 0 & 0 & 1 \\
c & 1 & 1 & 0 & 0 & 1 & 0 \\
d & 1 & 0 & 0 & 0 & 1 & 0 \\
e & 0 & 0 & 1 & 1 & 0 & 1 \\
f & 0 & 1 & 0 & 0 & 1 & 0
\end{array}
$$

Notice how the list attachs all the paths of "1"

list

a	→ c → d
b	→ c → f
c	→ b → f → *a
d	→ a → e
e	→ c → d → f
f	→ b → e

Running Times: Breadth-first search
Adj List: ~~$\Theta(v^2)$~~ $\Theta(v+E)$
Adj Matrix: $\Theta(v^2)$

Depth-first search
Adj List: $\Theta(v+E)$
Adj Matrix: $\Theta(v^2)$

Searches: * Breadth-first search, depth-first, topological: exhaustive searchs

stack { Depth-First Search: starts at vertex → visit adj vertex (dead-end = vertex w/ no vert) at a dead-end → backs up one edge

queue { Breadth-First Search: starting vertex → visits ALL adj vertex (marks visited, removes) identify unvisited → adjacent to front

DFS & BFS check connectivity & acyclicity of a graph
BFS → find path w/ fewest number of edges between 2

Topological Sorting: list vertices for every edges in the graph

Selection/Sorting: Bubble Sort, selection Sort, Insertion Sort, Counting sort
Partitioning, Quickselect, MergeSort, Quicksort, Radix sort

Sparse → vertices > edges → Adj List
dense → few possible edges missing → Adj Matrix

height - edges
diameter - nodes

DFS → good for cyclic

Best case → $O(n)$
Worst → $O(n^2)$

* Bubble
* Selection
* Insertion

Red Black Trees

Potential Violations during insertion:

* the root is black
* the node is red -> both children black

Case 1: Z's uncle y is red

* p[z].color = black
* y.color = black
* p[p[z]].color = red
* z = p[p[z]]

Z's parent is a left child and

Case 2a: z's uncle y is black & z is right child

* z = p[z]
* left-rotate(z)

Case 3a: z's uncle y is black & z is left child

* p[z].color = black
* p[p[z]].color = red
* right-rotate(p[p[z]])

Z's parent is a right child and

Case 2b: z's uncle y is black & z is left child

* z = p[z]
* right-rotate[z]

Case 3b: z's uncle y is black & z is right child

* p[z].color = black
* p[p[z]].color = red
* left-rotate(p[p[z]])

* ROOT MUST BE BLACK!! *

Test 3 Prep

Horner's Method: Horner's Rule is for evaluating a polynomial

$p(x) = 2x^4 - x^3 + 3x^2 + x - 5 \Rightarrow x(x(x(2x-1)+3)+1)-5$

Program → evaluates a polynomial @ a given input P[0…n], x

Pseudocode → p ← P(n), for i ← n-1 downto 0 do, p ← x * p + P[i]

Example: P = [-5, 1, 3, -1, 2]

x = 3, n = 4, P = P[4]
* p = 2

x	p	n	i
3	2	4	
	(6-1)		3
	18		2
	55		1
	$\boxed{160}$		0

3	2	-1	3	1	-5
	↓	6	15	54	165
	2	5	18	55	$\boxed{160}$

Correct answer!

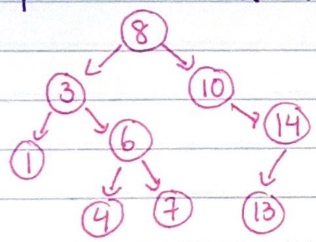
Binary Exponentiation: Let $n = b_I \ldots b_i \ldots b_0$, $p(x) = b_I x^I + \ldots b_i x^i + \ldots$

Computing a^n let n = … where $x = 2 \Rightarrow 13 = 1\cdot2^3 + 1\cdot2^2 + 0\cdot2^1 + 1\cdot2^0$

Program → p ← 1 for i ← I-1 downto 0 do p ← 2p + b

LR Binary → product ← a for i ← I-1 downto 0 do prod = prod * prod
if $b_i = 1$: prod ← prod * a, return prod

Example: 2^{13} → a = 2, b(n) = 1101



8192 {

product	a	i
2	2	2
4		1
8		
64		0
4096		
8192		

Binary Search Trees: Preorder root left right ⇒ 8, 3, 1, 6, 4, 7, 10, 14, 13

Runtime: Θ(n) Inorder left root right ⇒ 1, 3, 4, 6, 7, 8, 10, 13, 14

Height: # edges root→deep Postorder left right root ⇒ 1, 4, 7, 6, 3, 13, 14, 10, 8

↳ no children = 0, null = -1, Max Width = # nodes on lvl w/ most, Dia = longest path

Red Black Trees: Red Black Trees are the search tree scheme that is balanced to guarantee that ops take O(lg n) W.C.

* Each Node contains the attributes ⇒ color, key, left, right & p.

1) Every Node is Red/Black
2) Root is Black
3) Every leaf (NIL) is Black
4) If Red → both children Black
5) Simple paths # black nodes same

{ Key Movements:
* Left-Right Rotation
* Right-Left Rotation
* Insertion ↑ }

* After insert, check 5 requirements *