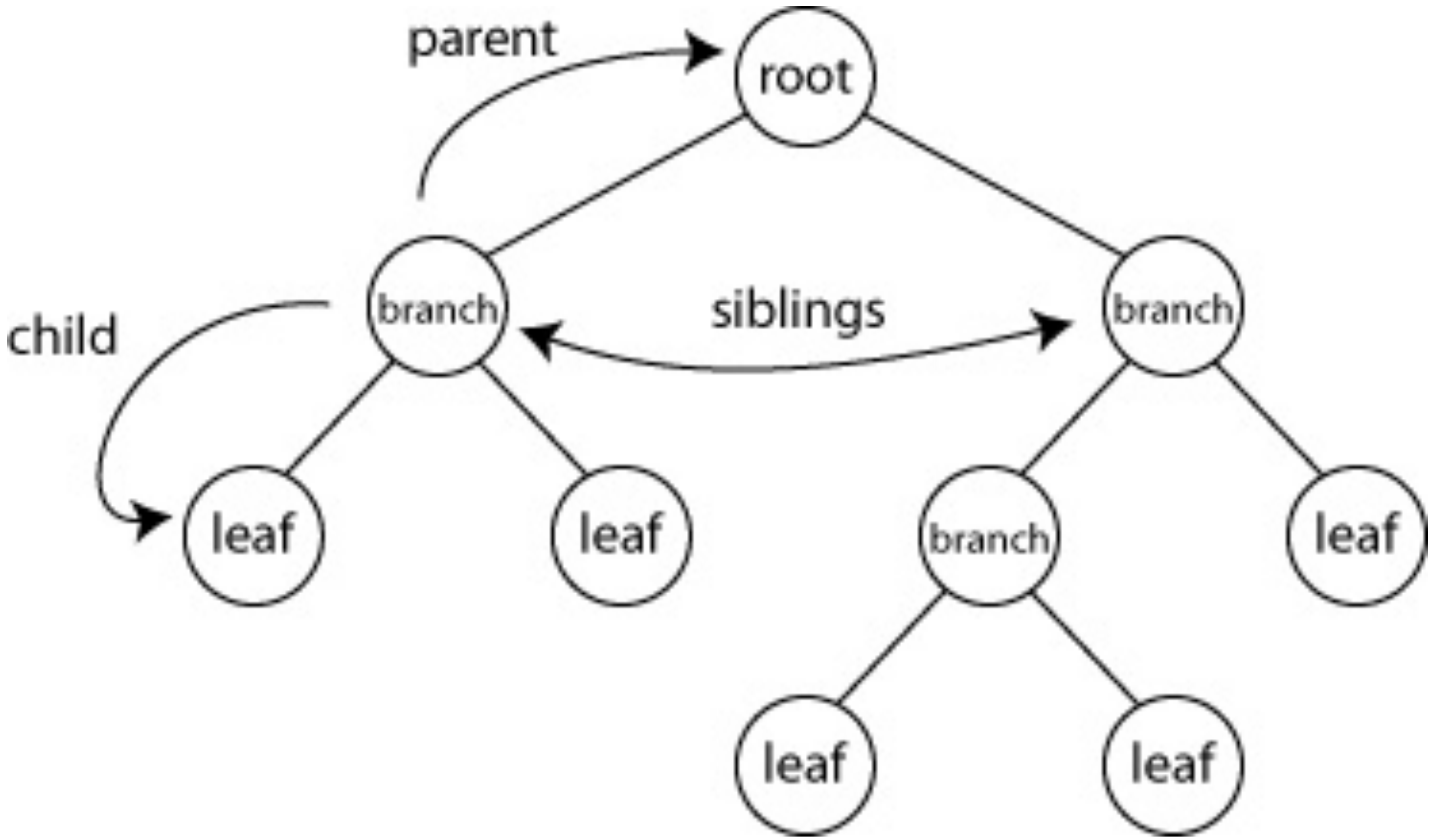
Tree

data structure composed of nodes

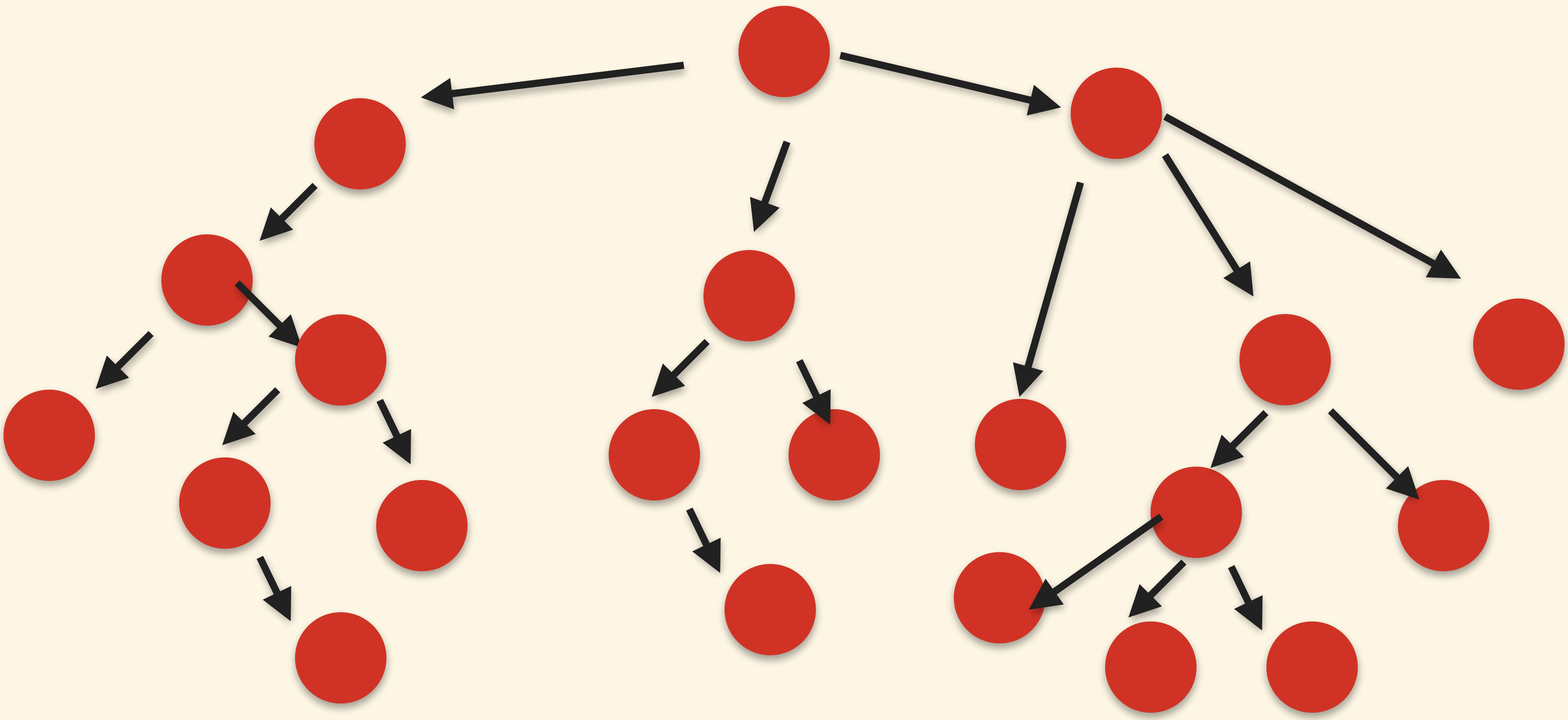
each node has a value, and a set of (zero or more) nodes that it references

each node is itself a tree

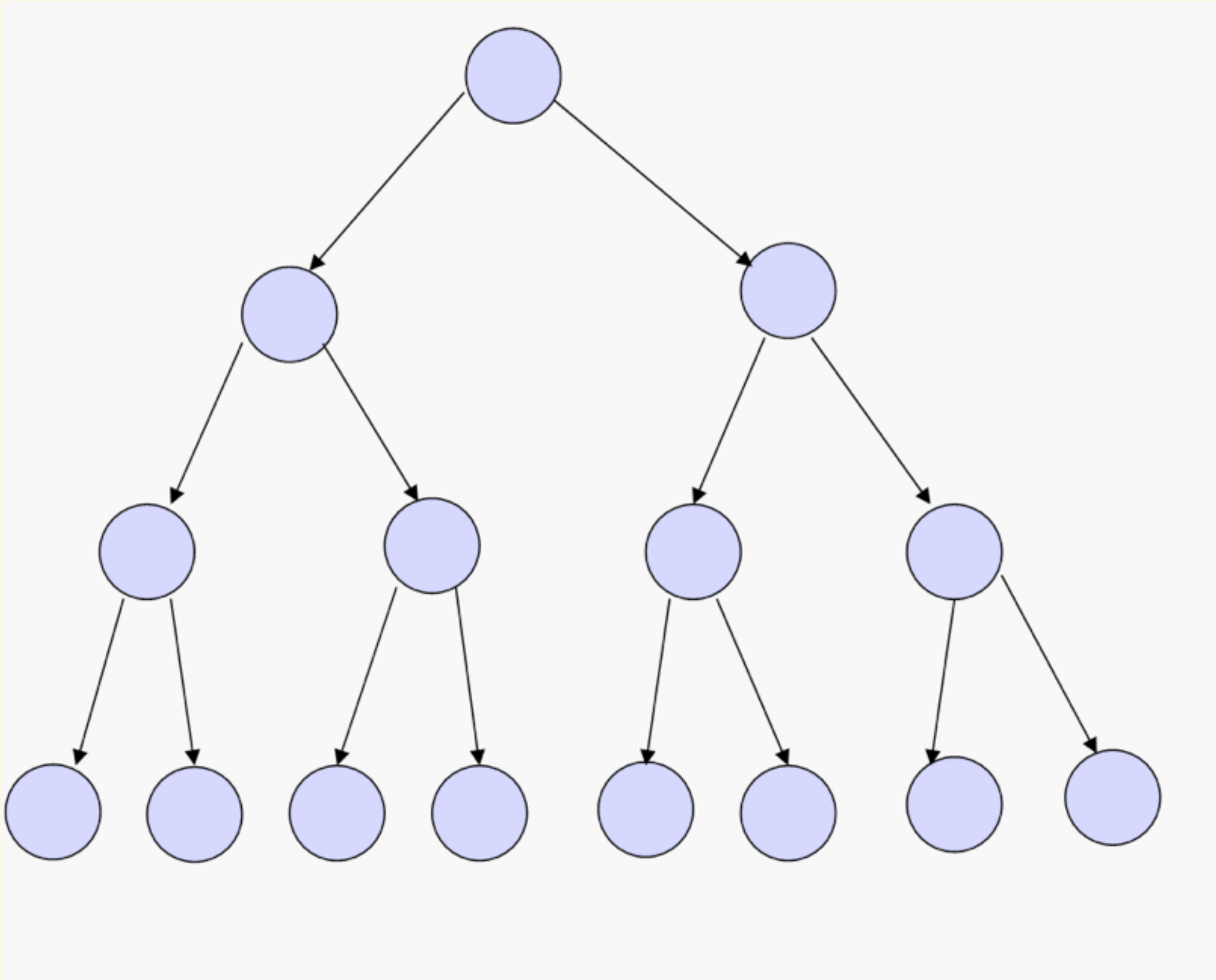


http://www.jslab.dk/gfx/trees1.jpg

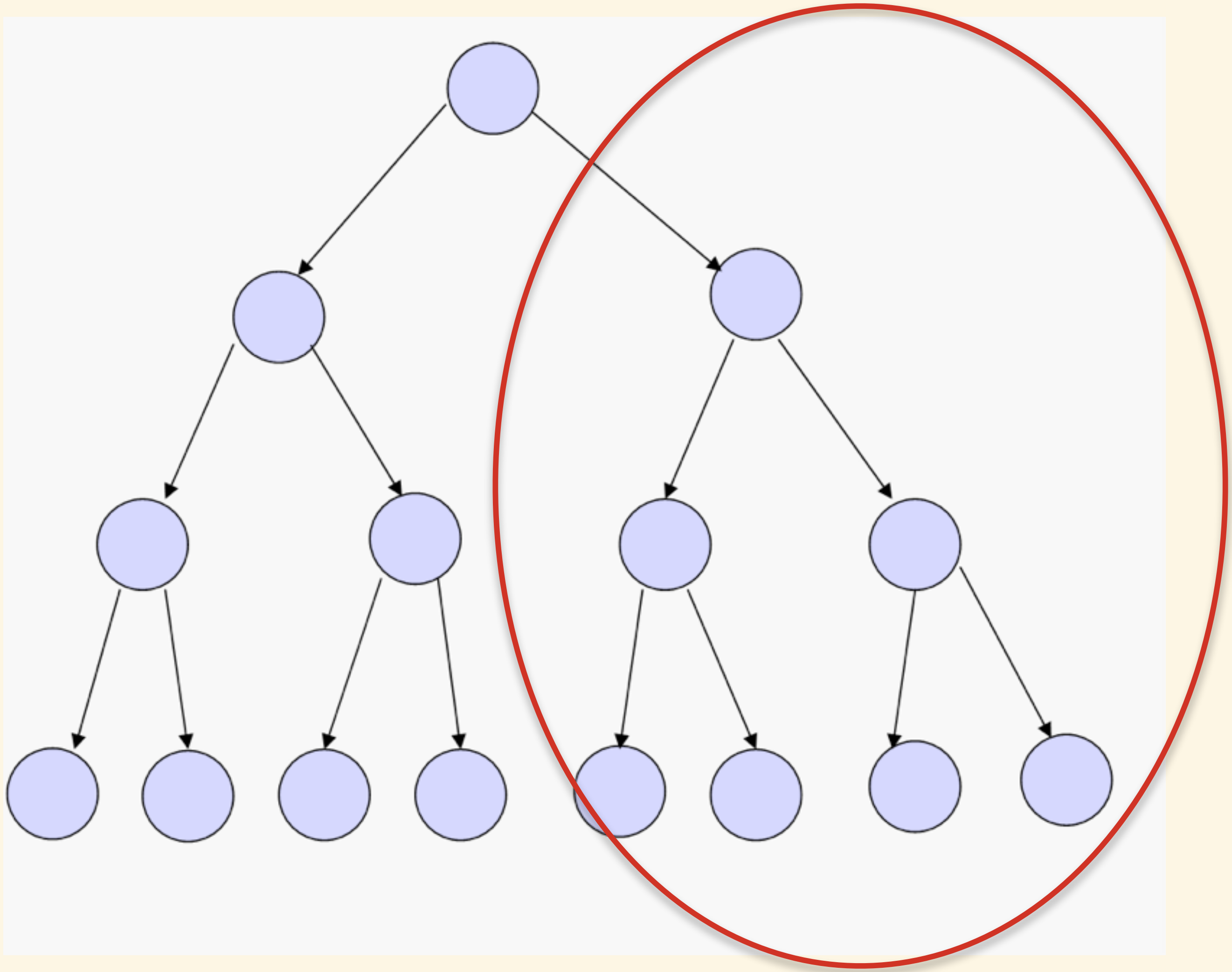
Tell your neighbor: what is this?



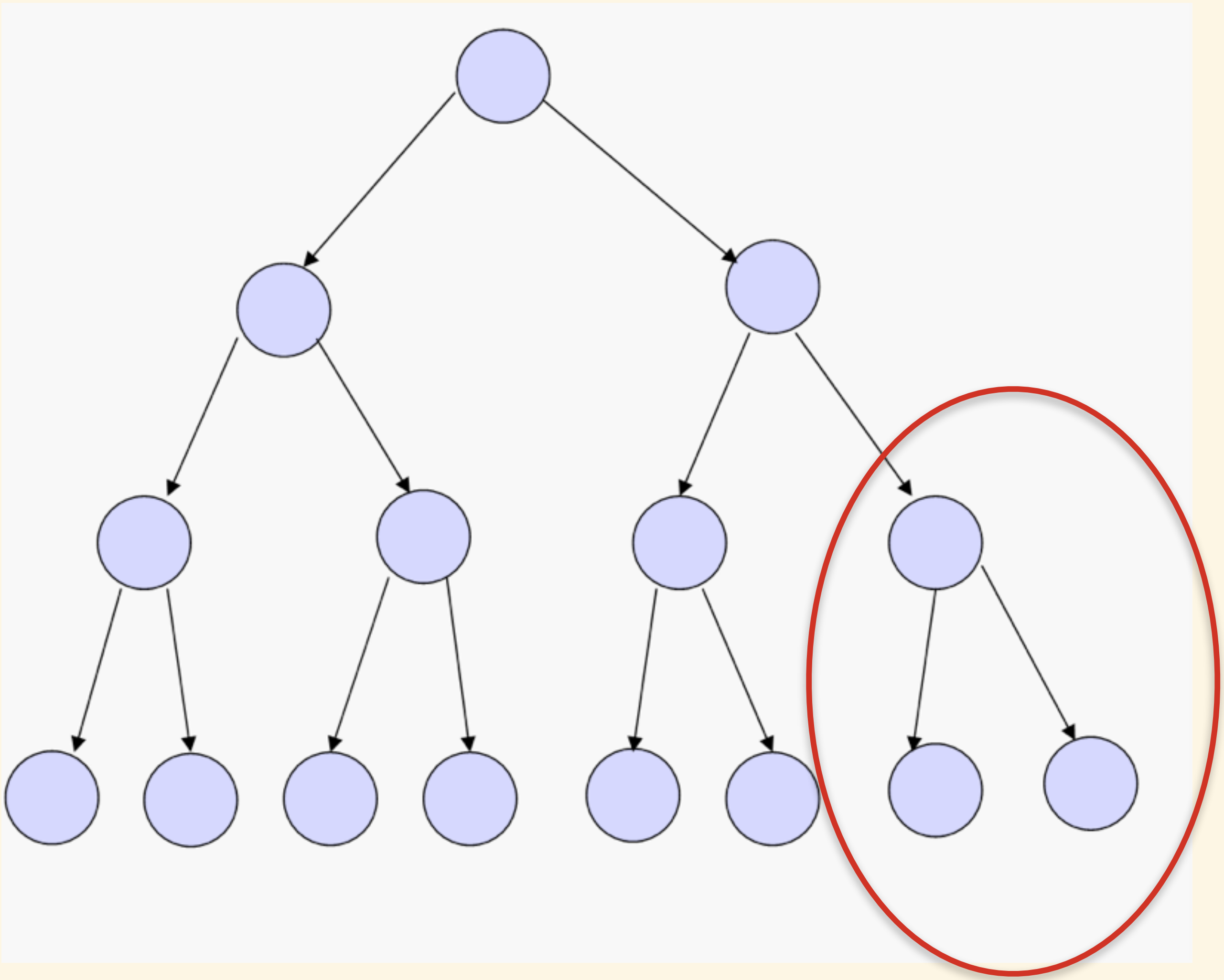
What is this?



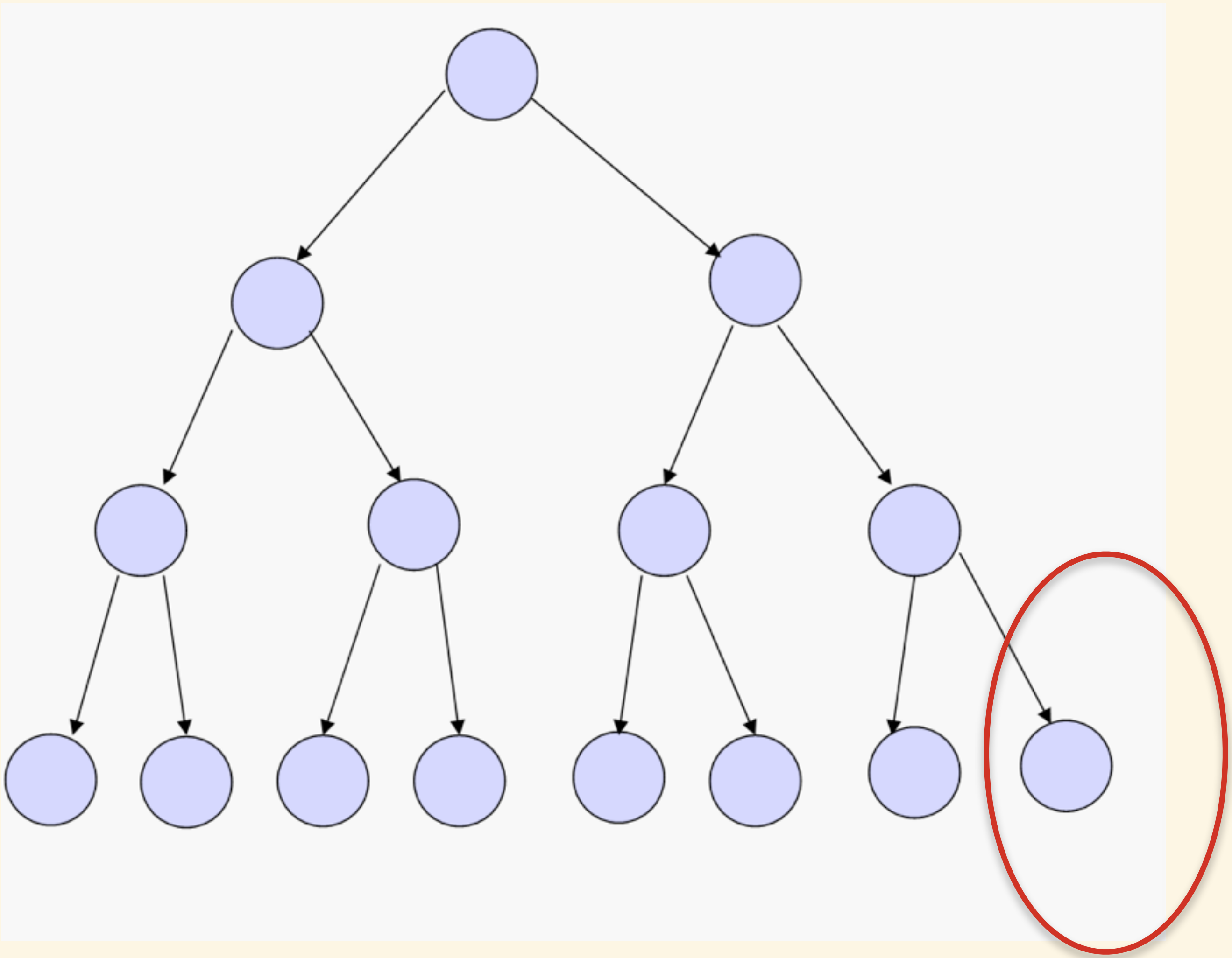
and then… ?



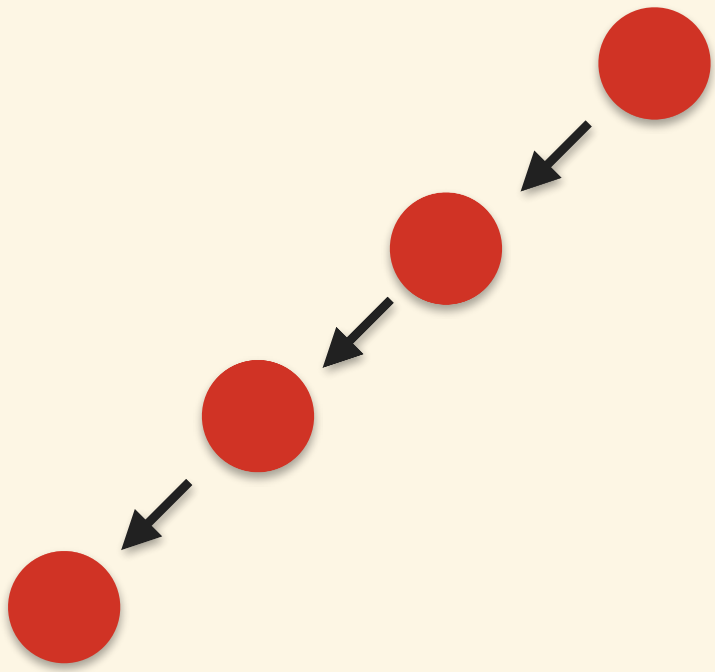
and then… ?



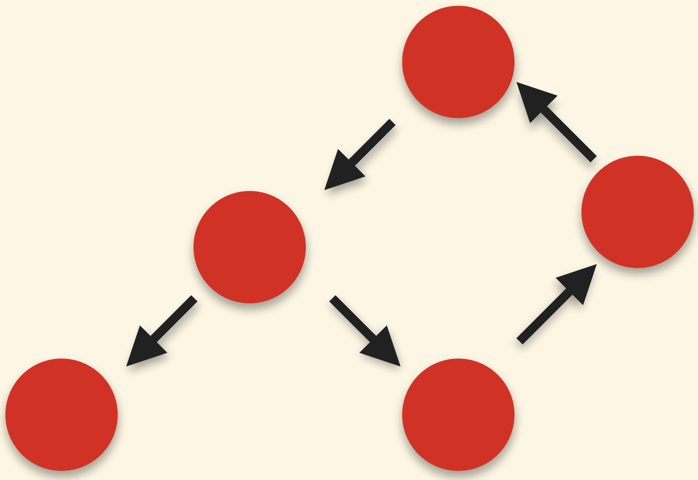
and then… ?



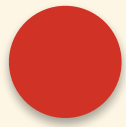
What about this?



This?



How about this?



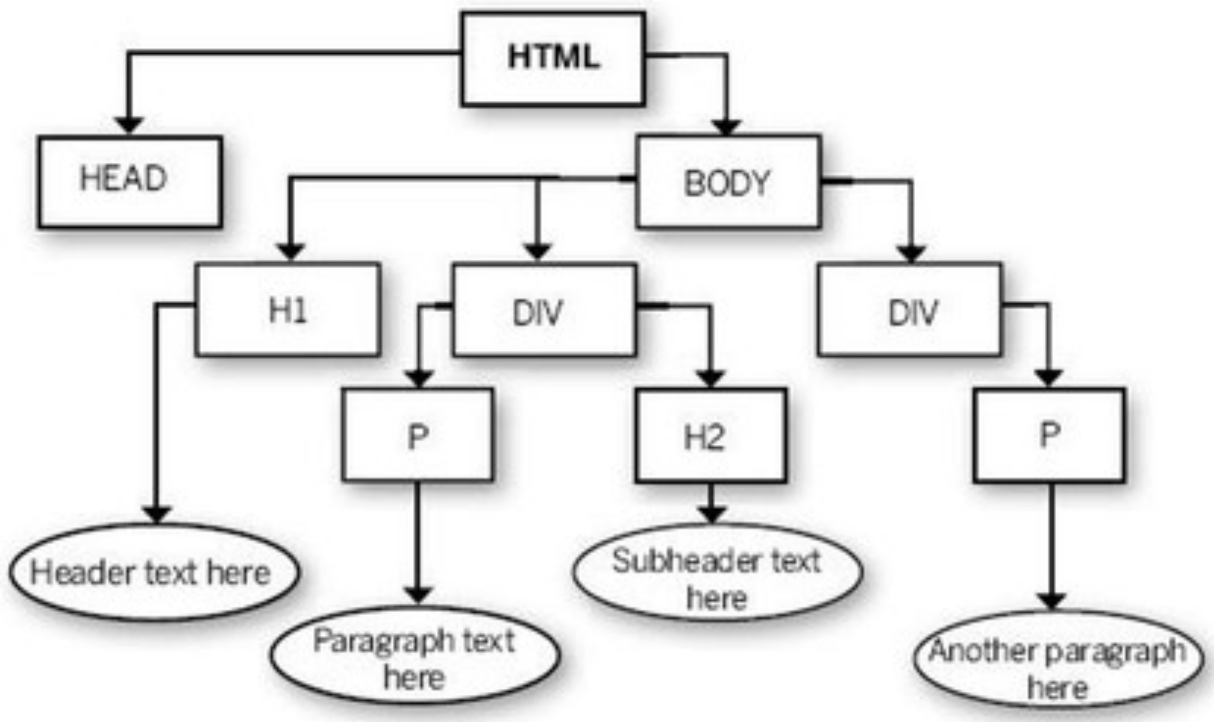
Recursive Data Structure

one that's composed of smaller versions of the same type of data structure

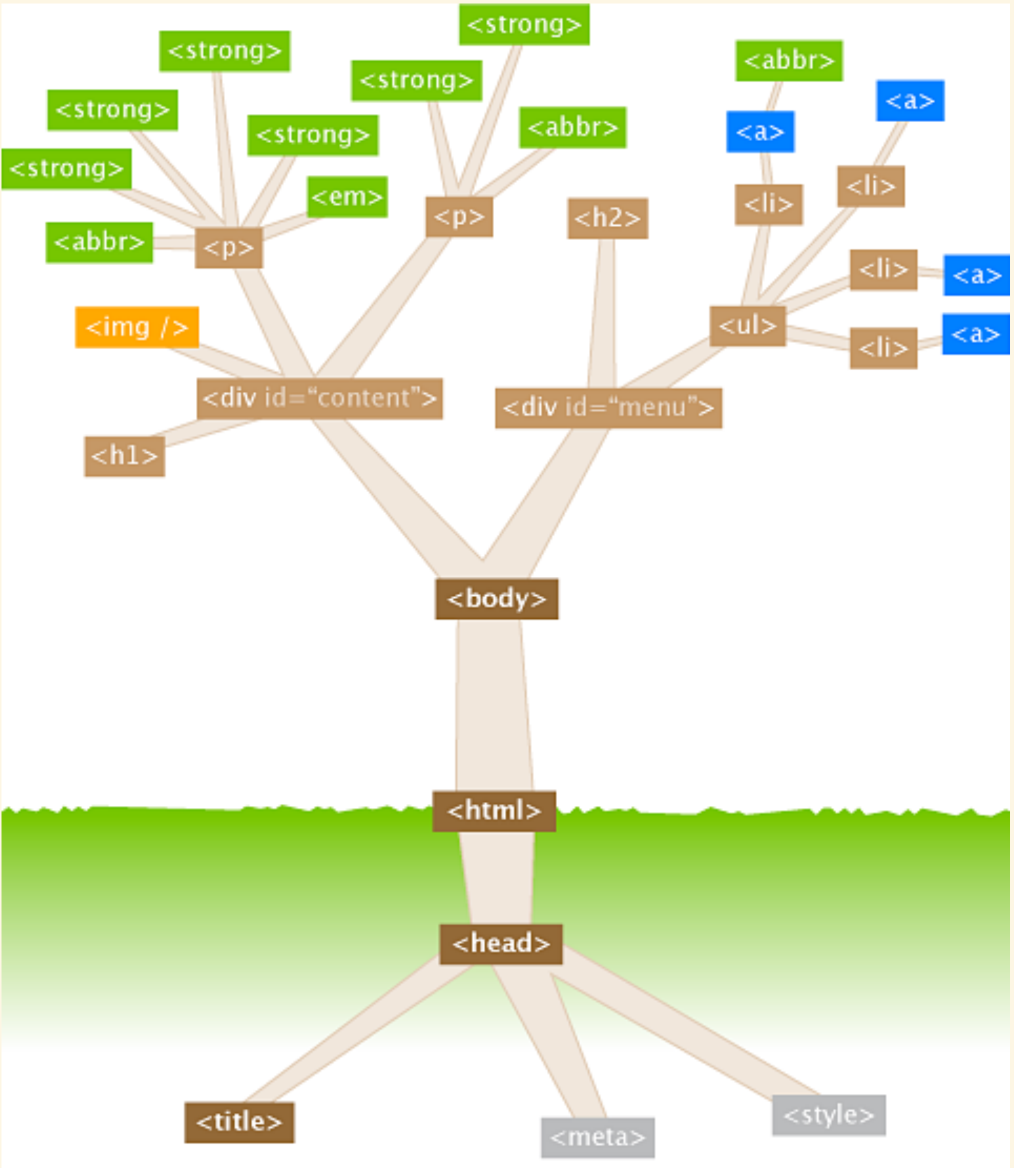
Uses for trees

- DOM
- file system (today)
- searching (2nd half of the course)

DOM TREE

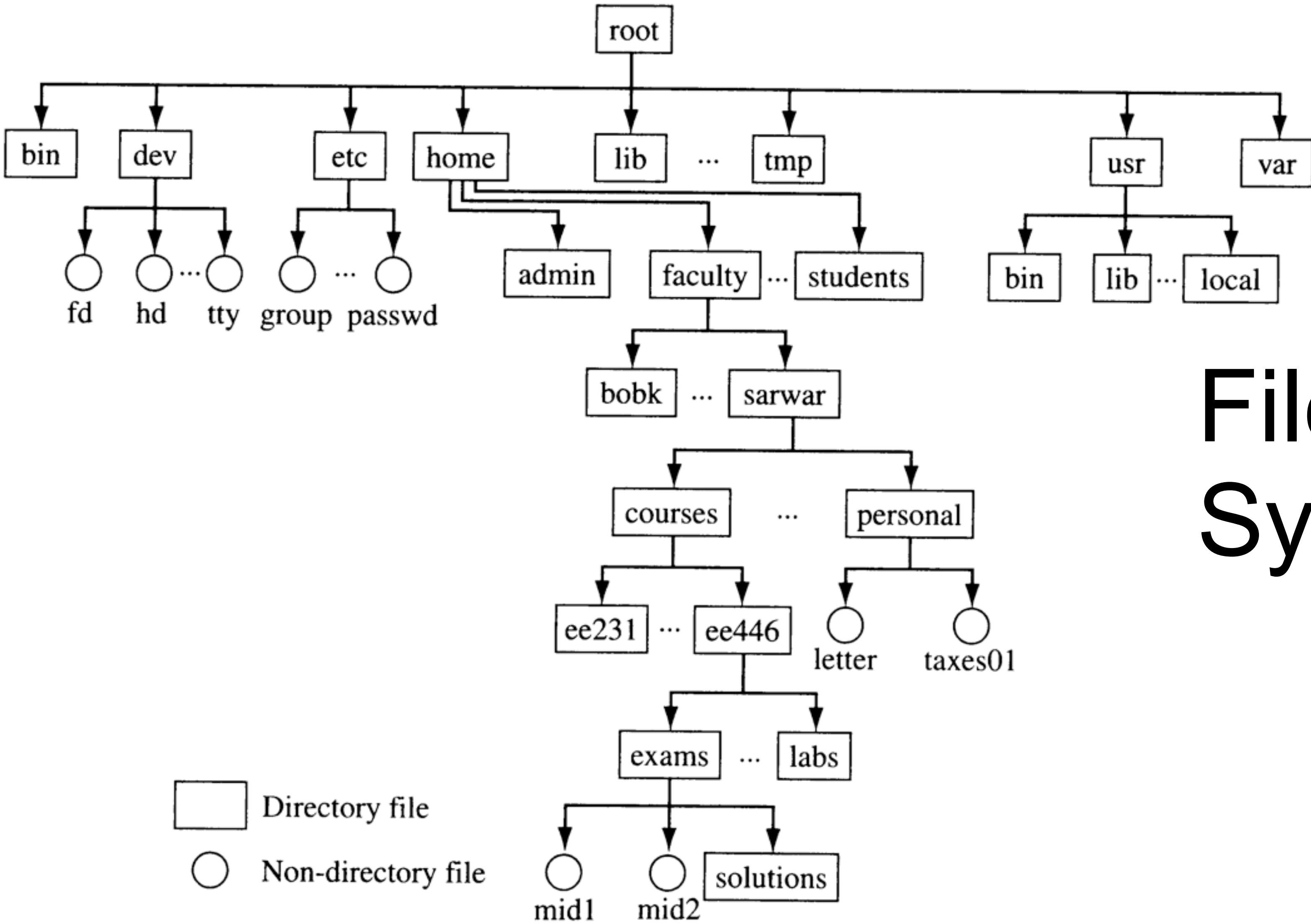


http://cdn0.mos.techradar.futurecdn.net/Review%20images/Linux%20Format/Issue%20118/DOM%20tree%20inline2-420-90.jpg



What is the root element of this tree?

File System


root
bin
dev
etc
home
lib
...
tmp
usr
var
fd
hd
tty
group
passwd
admin
faculty
students
bin
lib
local
bobk
sarwar
courses
personal
ee231
ee446
letter
taxes01
exams
labs
mid1
mid2
solutions
Directory file
Non-directory file

Tree Node

object that holds a value and references to child nodes

```
const node = {
  data: someValue,
  children: [ ]
}
```

```
class TreeNode {
  constructor(data) {
    this.data = data;
    this.children = [];
  }
}
```

Tree Traversal

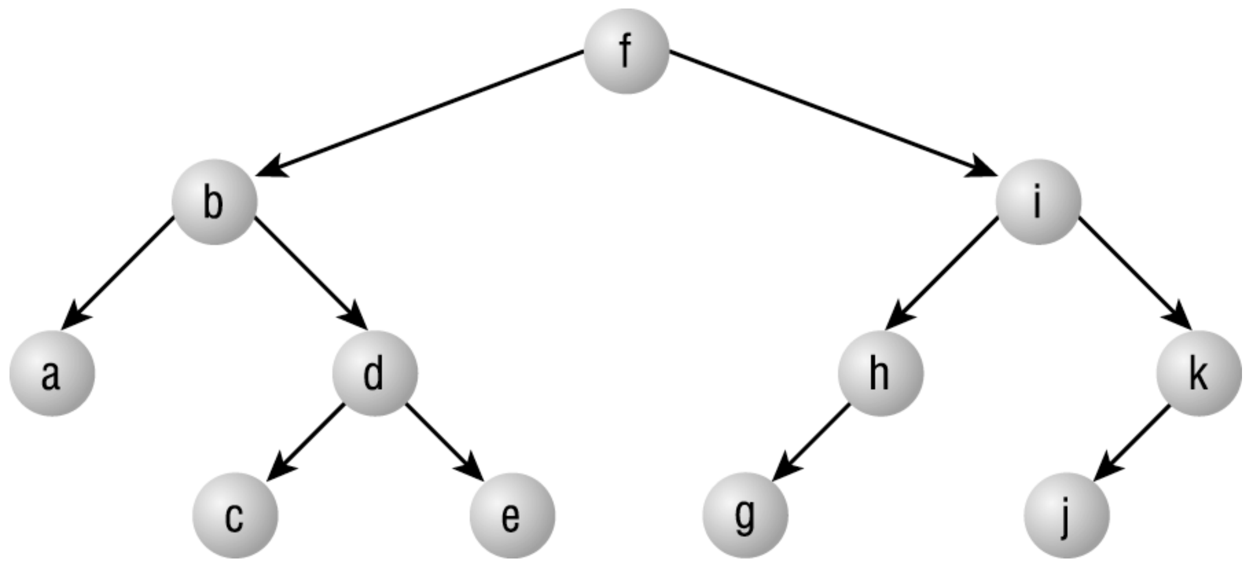
iterating through every element in a tree

```
const D = {
    data: 'D',
    children: []
};

const B = {
    data: 'B',
    children: [D]
};

const C = {
    data: 'C',
    children: []
};

const A = {
    data: 'A',
    children: [B, C]
};
```

Consider this set of nodes

1. Which of the nodes is the root? How do you know?

2. Draw a tree diagram for these nodes similar to the tree below:



How to whiteboard a recursive function

```
function countTo(input, max) {

    console.log(`before ${input}`);

    if(input < max) {
        const message = countTo(input + 1, max);
        console.log(message);
    }

    return `after ${input}`;
}
```

Depth First Traversal

```
const D = {
    data: 'D',
    children: []
};

const B = {
    data: 'B',
    children: [D]
};

const C = {
    data: 'C',
    children: []
};

const A = {
    data: 'A',
    children: [B, C]
};
```

Visit a tree's children before its siblings

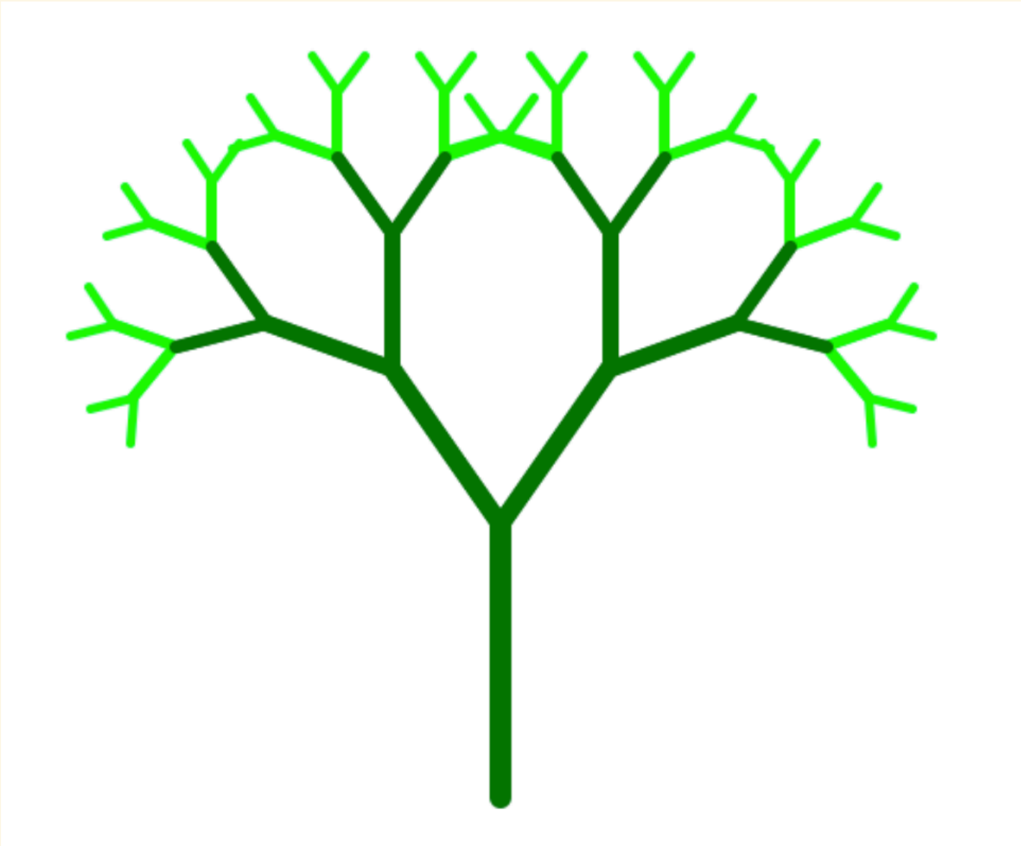
1. <u>whiteboard</u> a function that takes a node and console logs the data for that node and then the data for each of its children but indented. Passing A, it would look like this:

two spaces indent
per level down

```
A
  B
    D
  C
```

2. step through your code, keeping a call stack and showing variables change

What is special about THIS tree?



Binary Tree Node

each node has **at most** two children

they are referred to as “**left**” and “**right**”

```javascript
const D = {
    data: 'D',
    children: []
};

const B = {
    data: 'B',
    children: [D]
};

const C = {
    data: 'C',
    children: []
};

const A = {
    data: 'A',
    children: [B, C]
};
```

```javascript
const D = {
    data: 'D',
    left: null,
    right: null
};

const B = {
    data: 'B',
    left: null,
    right: D
};

const C = {
    data: 'C',
    left: null,
    right: null
};

const A = {
    data: 'A',
    left: B,
    right: C
};
```

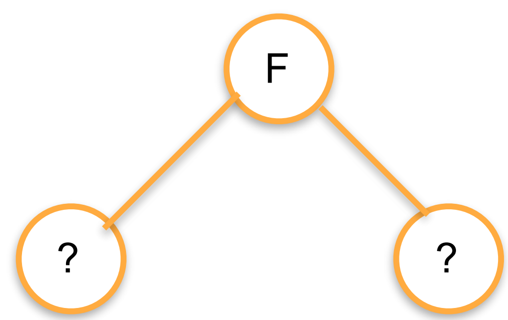
Try Me! 1. Draw the Binary Tree (**rooted at index 6**) for these data:

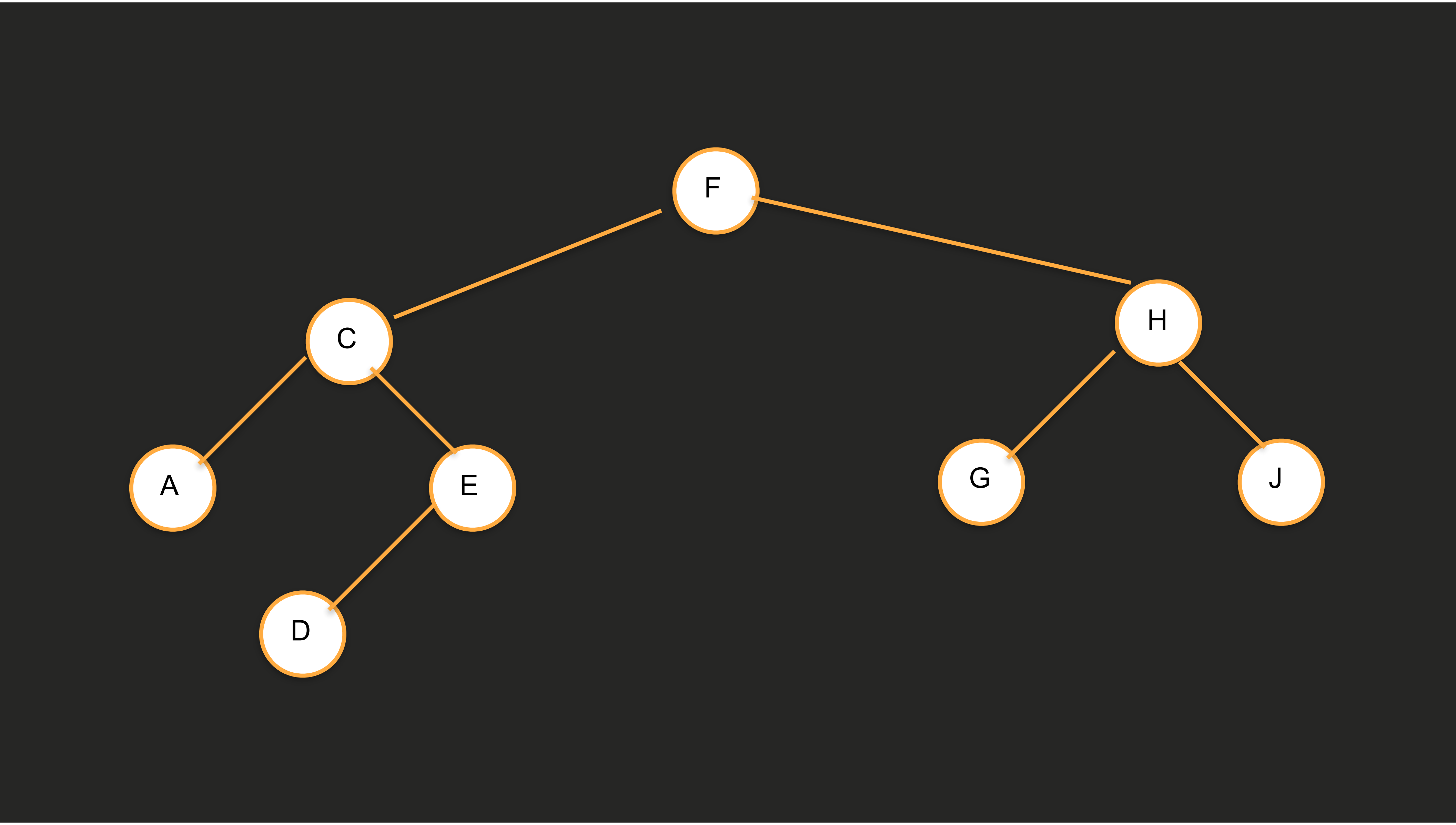
index	value	left (index #)	right (index #)
1	A	null	null
2	B	8	null
3	C	1	5
4	D	null	null
5	E	4	null
6 (root)	F	3	8
7	G	null	null
8	H	7	10
9	I	null	null
10	J	null	null

Questions to answer:

2. Which nodes are terminal (leafs)? How an you tell from just the data table?

3. Which data are NOT part of this tree? (hint: you must start by drawing the root:



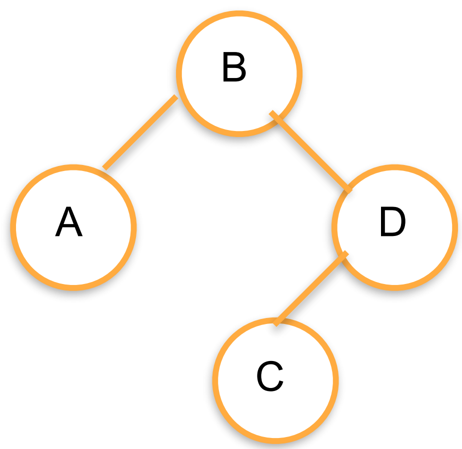

F
C
H
A
E
G
J
D

What do you notice about this tree?

Node Insertion

Implement the add method for the BinaryNode class

```
const A = new BinaryNode('A');
const B = new BinaryNode('B');
const C = new BinaryNode('C');
const D = new BinaryNode('D');
B.add(A);
B.add(D);
B.add(C);
```




```
class BinaryNode {
    constructor(data) {
        this.data = data ;
        this.left = null;
        this.right = null;
    }

    add(node) {
        // Implement Me!
    }
}
```

Height (this will be important next time)

distance (number of edges) from root to furthest-away leaf

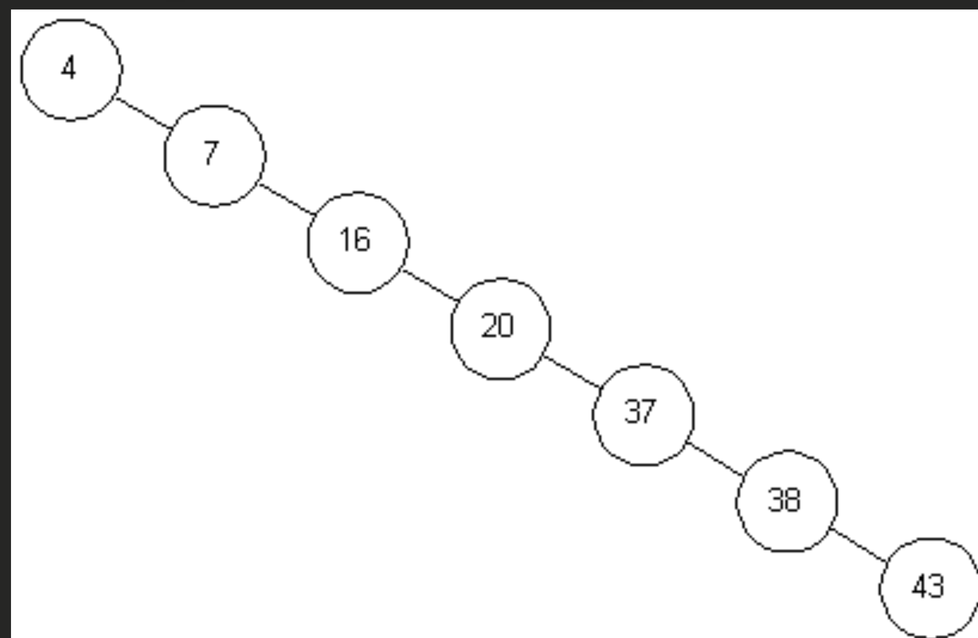
Try Me!

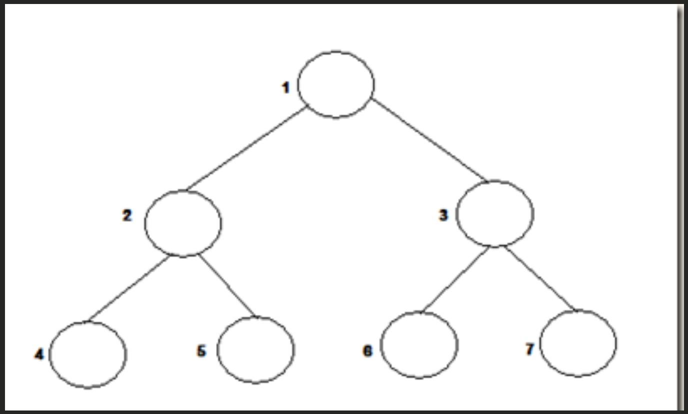
what are the largest and smallest possible heights for a binary tree with n nodes?

largest: n - 1

so approximately n for large values of n

ex: n = 7





smallest: $\log_2(n+1) - 1$

so **approximately log(n)**

("log" is generally base 2 in computer science)

n = 7 has a height of 2

$\log_2(7+1) - 1 \;\; = = = 2$

($\log_2(n+1) - 1$ is approximately 2 for non-perfect trees)