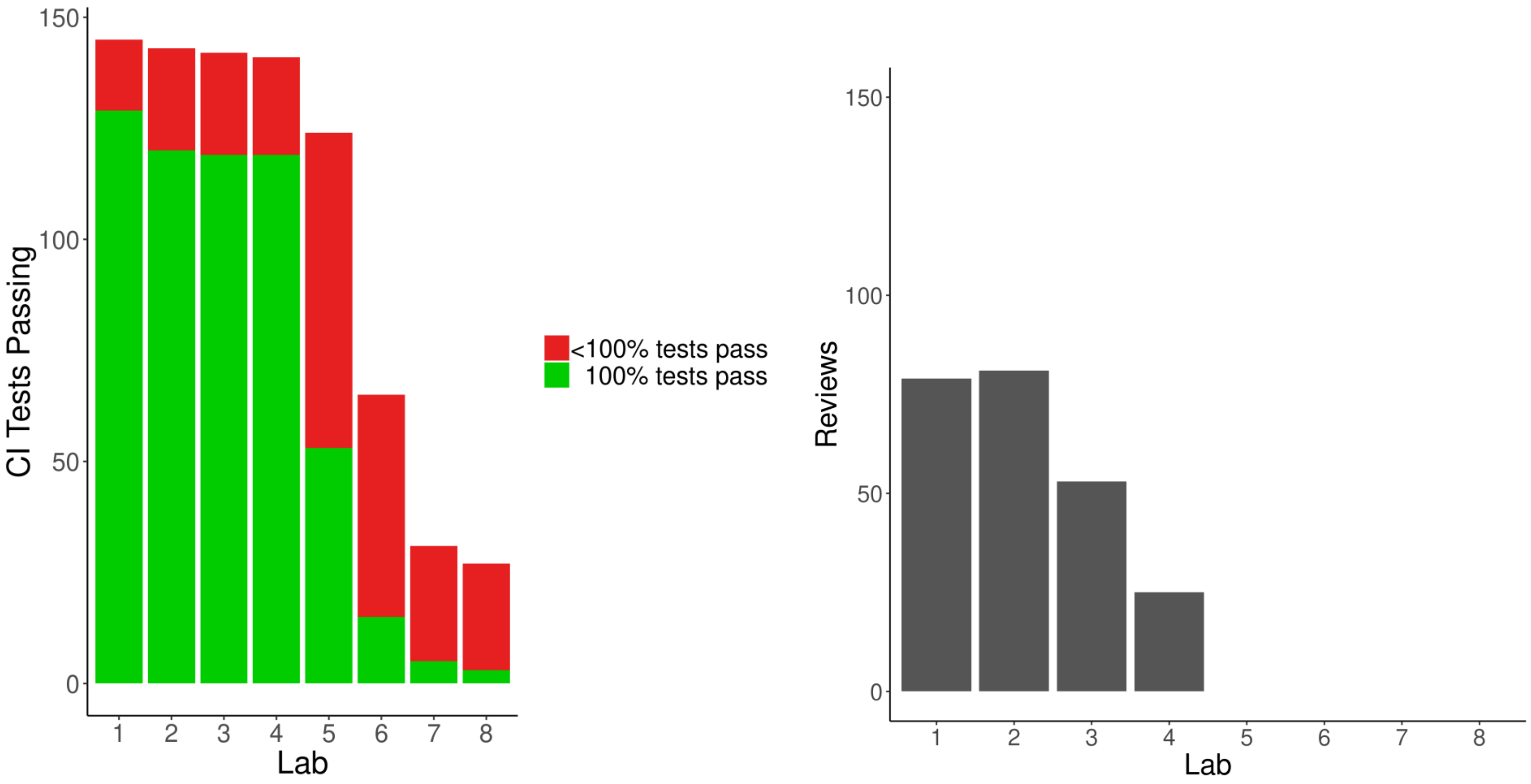


150
100
50
0
CI Tests Passing
1
2
3
4
5
6
7
8
Lab
<100% tests pass
100% tests pass
150
100
50
0
Reviews
1
2
3
4
5
6
7
8
Lab

Exercise

```java
public int binarySearch (int key, int first, int last)
  throws NotFoundException {
    if (first > last)
       throw new NotFoundException ("not found");
    else {
       int middle = (first + last) / 2;
       if (key == arr[middle])
          return arr[middle];
       else if (key < arr[middle])
          return binarySearch (key, first, middle-1);
        else
          return binarySearch (key, middle+1, last);
   }
}
```

Q1. Assume array **arr** has the following elements:

int[] arr = {2,4,6,8,10,12,14,16};

Draw the recursion trace of the algorithm with the following calls:

(a) **binarySearch(16,0,7);**

(b) **binarySearch(15,0,7);**

For each call describe

- the value of each parameter
- what the **middle**

Solution (a)

int[] arr = {2,4,6,8,10,12,14,16};

binarySearch(16,0,7);

Middle is 7 / 2 = 3.
Since arr[3] = 8 < 16, the right hand side is called

binarySearch(16,4,7);

Middle is 11 / 2 = 5.
Since arr[5] = 12 < 16, the right hand side is called:

binarySearch(16,6,7);

Middle is 13 / 2 = 6.
Since arr[6] = 14 < 16, the right hand side is called

binarySearch(16,7,7);

Middle is 14 / 2 = 7.
Since arr[7] = 16 we are done and **16 is returned**.

Solution (b)

int[] arr = {2,4,6,8,10,12,14,16};

binarySearch(15,0,7);

Here middle is 7 / 2 = 3.

Since arr[3] = 8 < 15, the right hand side is called

binarySearch(15,4,7);

Here middle is 11 / 2 = 5.

Since arr[5] = 12 < 15, the right hand side is called

binarySearch(15,6,7);

Here middle is 13 / 2 = 6.

Since arr[6] = 14 < 15, the right hand side is called

binarySearch(15,7,7);

Here middle is 14 / 2 = 7.

Since arr[7] = 16 > 15, the left hand side is called.

binarySearch(15,8,7);

Here middle first > last (8 >7) so **NotFoundException ("not found") is thrown**

Next few lectures

- Lecture 11 (yesterday): linear/binary search
  - Lots of live coding
  - You're now ready for **lab 6**
- Lecture 12 (today): trees
  - New concept: trees
- Lecture 13: implementing binary search trees
  - Lots of live coding
  - **Lab 7 Q1 and Q2**
- Lecture 14: priority queues
  - New concept: priority queues
  - **Lab 7 Q3 and Q4**

Software Development 3 (F27SG)

Lecture 12

(Binary Search) Trees

Rob Stewart

Motivation

- Last lecture we saw **Binary Search**
  - Linear search: O(N) growth
  - Binary searh: O(log N) growth
  - Data must be **sorted**
- On a **fixed, read only**, data collection
- What about **changing** data between searches?

Motivation

- We often need to **insert** and **remove** elements
- Key (**K**) - our lookup field
- Value (**V**) - what we want out from the data
- Consider some operations
  - **Get** number V from phone book with name K='Rob'
  - **Put** number V=0131 in phone book with name K='Rob'
  - **Remove** number for name K='Rob'
- will require re-ordering the array every time **O(N)**
- Could a **dynamic data structure** help improve efficiency of these operations?

Recap

- We've seen Dynamic Data Structures
  - Formed of Nodes
    - Value (s)
    - 1 or more object references to other nodes
  - Nodes are linked
- Variations in DDS are due to the **topology** of how nodes are connected to each other.
  - Singly Linked Lists
  - Doubly Linked Lists
  - Trees (today)

Overview

- Today: we will talk about Tree DDS in general
  - What are they?
  - What do they look like?
- From next week: we will start covering specific *types* of trees
  - There are LOTS
  - Binary Search Trees
- We will try to show the point of these

Trees and Terminology

This is a tree! ... or is it?

Trees and Terminology

This is how **Computer Scientists** Think of Trees!

Trees and Terminology

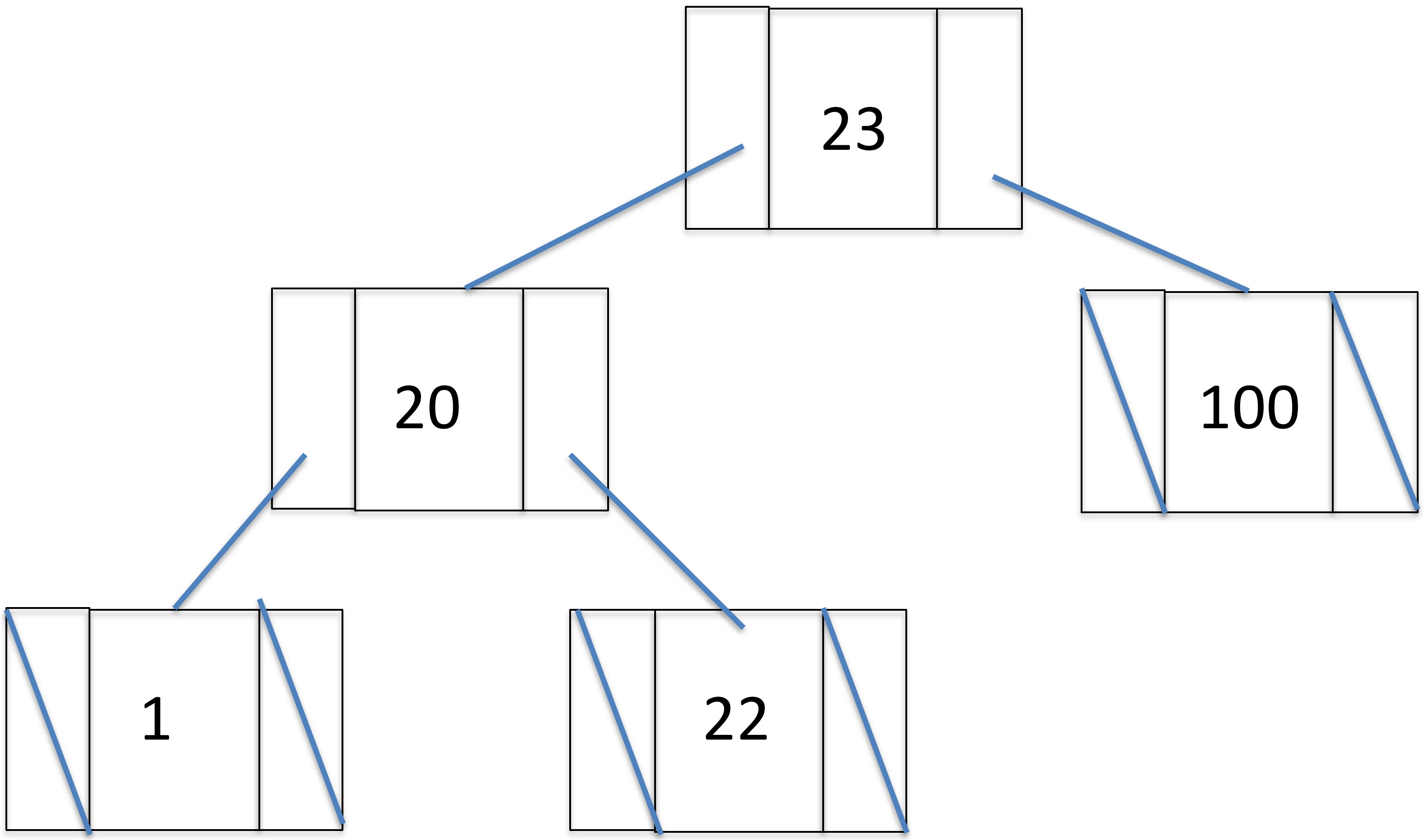
- All Trees are formed from Nodes
  - A value element
  - At least **two** references to the same type of object
- Trees have a **hierarchical** relationship

Trees and Terminology

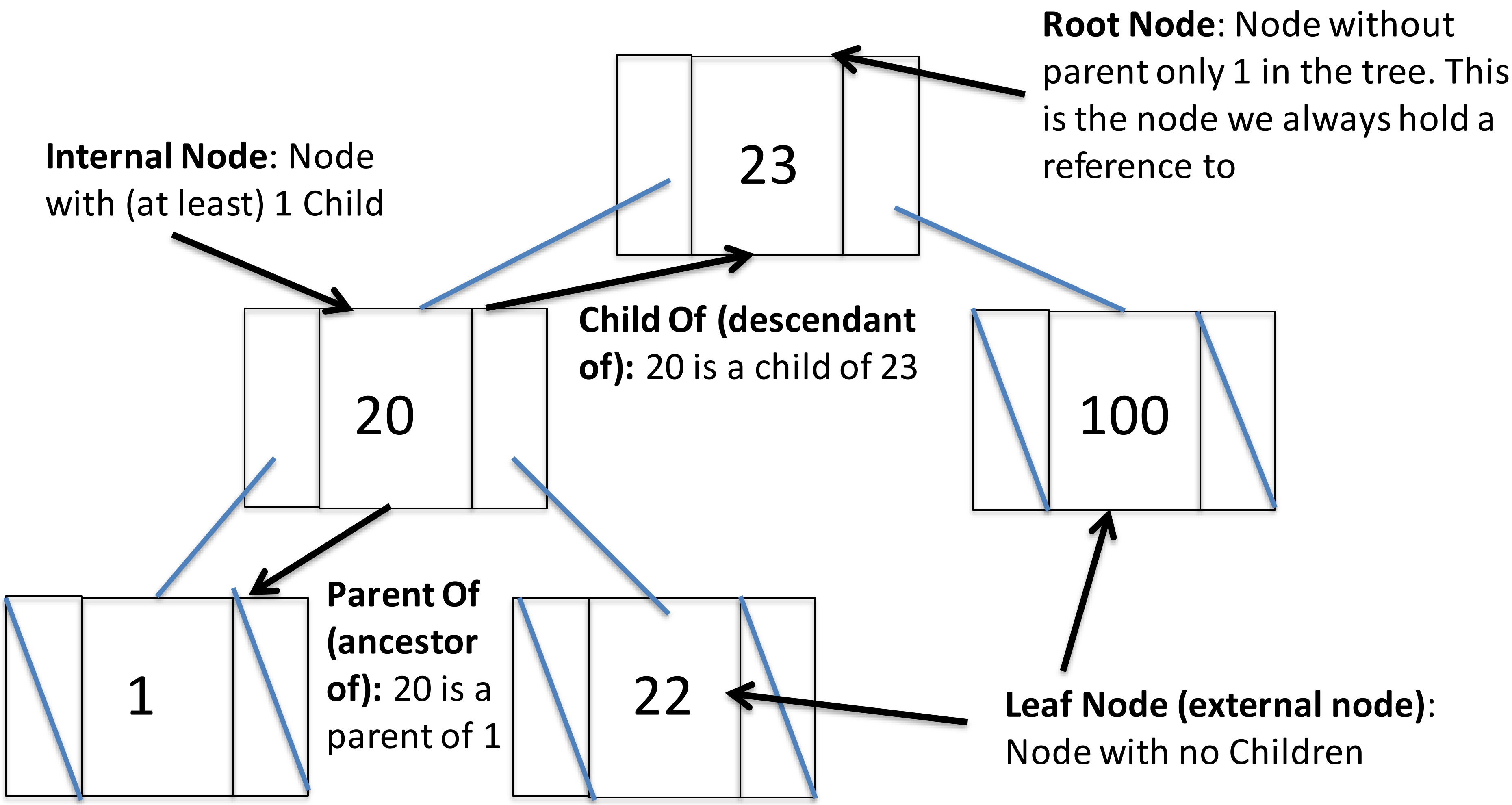
- Tree nodes similar to Doubly Linked List nodes
- As before
  - We can’t randomly access variables
  - Nodes have no idea where they are in the tree

Trees and Terminology

- This is how we draw trees!



Trees and Terminology

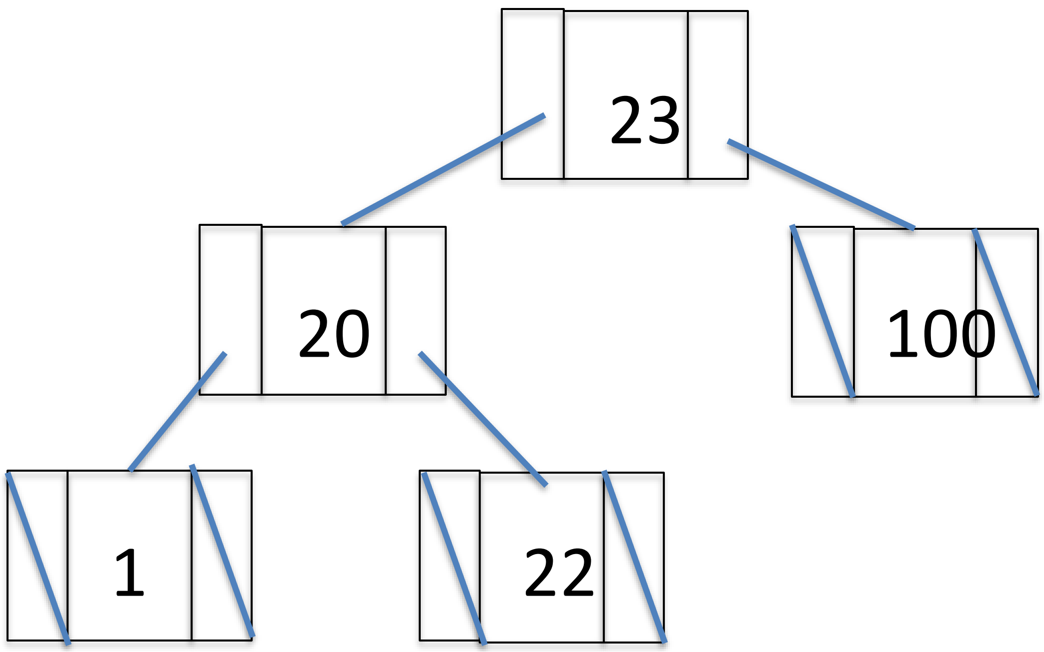


Trees and Terminology

- links between nodes are **Edges**
- We also talk about **Paths** through the tree.
  - The sequence of nodes we go through to get to any node (usually starting from the root)
    - E.g. 23->20->1 or 23->100

Trees height

- What is the height of this tree?

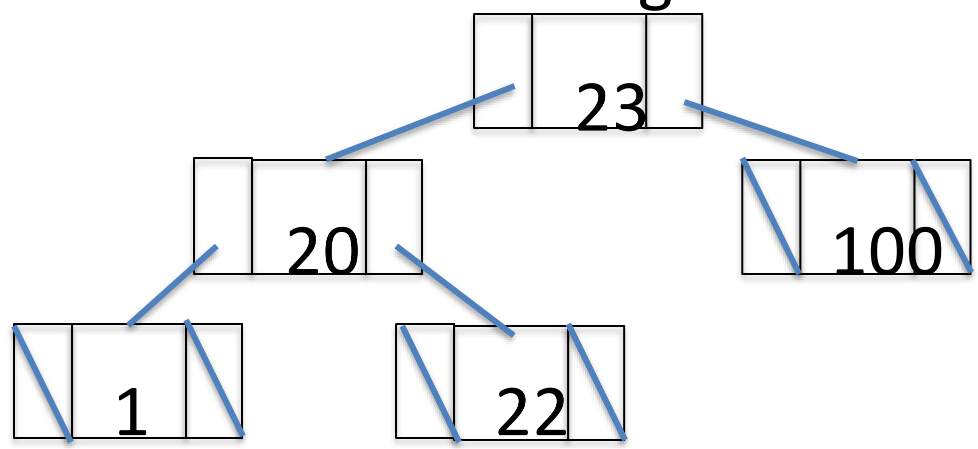


Trees and Terminology

- Must only be **1 path** from the root to any Node
  - If not we have a **Graph** not a Tree
  - Graphs covered in ***Data Structures & Algorithms***
    - Year 2, semester 1
- An important number is **Tree Height** or **Depth** (h)
  - h = the length of the longest path in the tree
    - i.e. the number of levels of nodes in the hierarchy
  - this is important for Time Complexity calculations

Trees and Terminology

- As with Linked-Lists Trees are **recursive**
- The Child of any node (and its children)
  - is a valid tree
  - We call this **a sub tree**
- They lend themselves to recursive algorithms
  - actually **really hard** to do non-recursive algorithms



Tree Variants

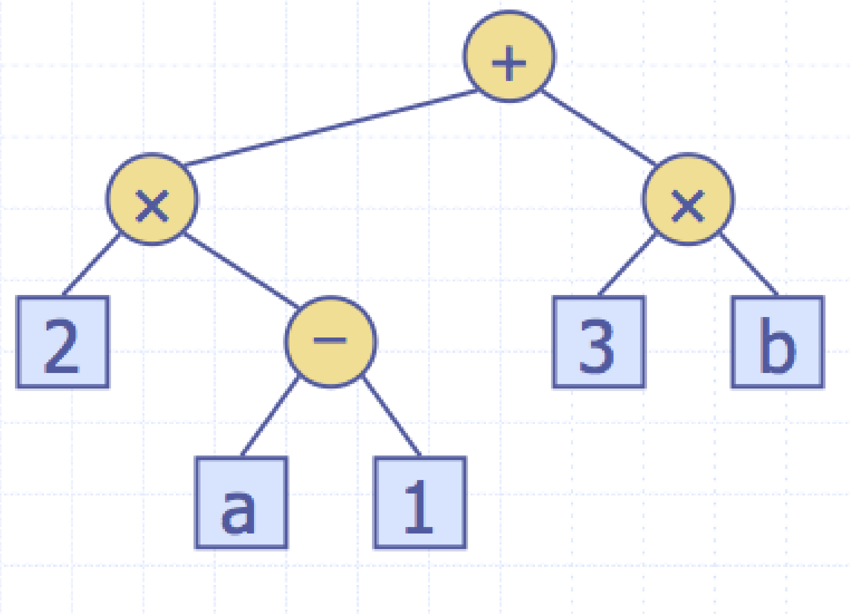
- Multiple Types of Tree
  - **Binary Trees**
    - Red/Black Trees, AVL-Tree
      - Self balancing
  - **K-ary Trees**
  - **B-Trees**
  - **R-Tree**
    - for geospatial searching
      - e.g. Find nearby venues in Foursquare

Tree Variants

- Multiple uses of trees
  - Artificial Intelligence
    - Decision trees
  - Operating Systems
    - Computer File Storage
  - Data Storage and Process
    - e.g. fast dictionaries
  - Indexing Databases
- All have the same basic properties
  - (with minor variations)
- We aren’t going to cover them all

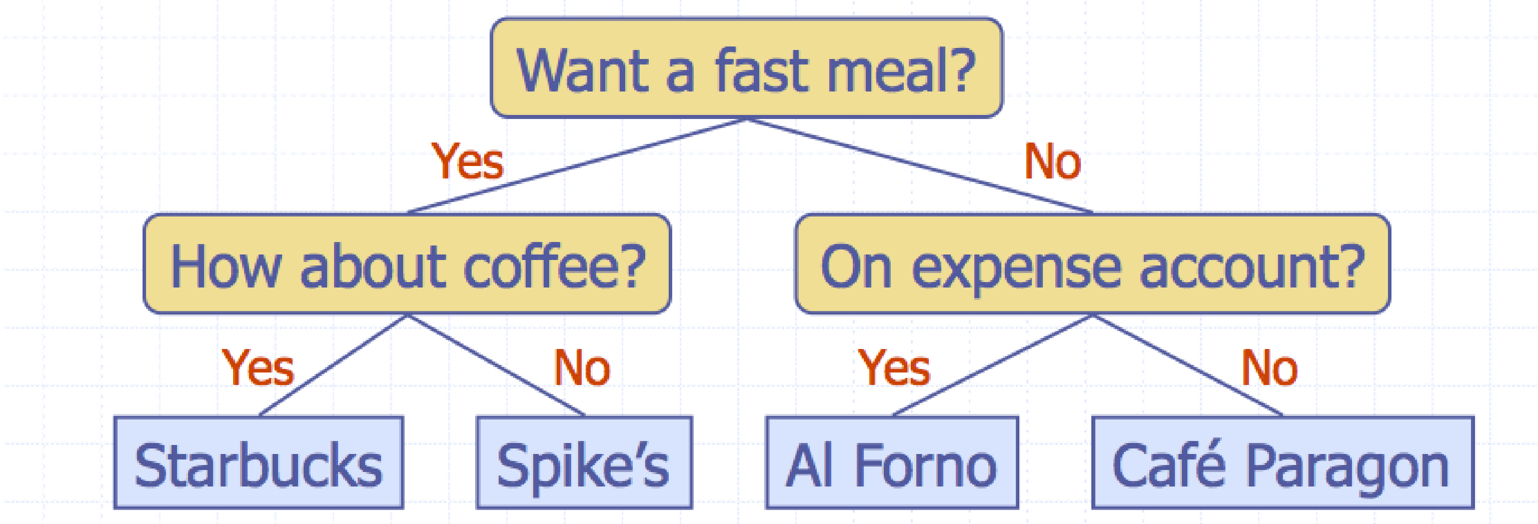
Arithmetic Expression Tree

- A way to represent an arithmetic expression in tree form
  - Internal Nodes: operators
  - External Nodes: operands
- E.g. (2x(a-1) + (3 x b))
- Allows us to logically traverse the expression



Decision Trees

- Represents a decision process to achieve a goal
  - Basic form of Artificial Intelligence
    - (e.g. Robot moving towards a goal)
- Internal Nodes = yes/no answer question
- External Nodes = decisions



Binary Trees

- As the name suggests Each Node has (at most) 2 Children
  - Left Child
  - Right Child
- The Children are typically ordered
  - i.e. there is some relationship between the left child and the node value.
  - E.g.
    - **Left Child < node Value < right child** (numerically)
    - Left Child (comes before) node Value (comes before) right child
    - Node is question, left child is yes to that question, right child is no.

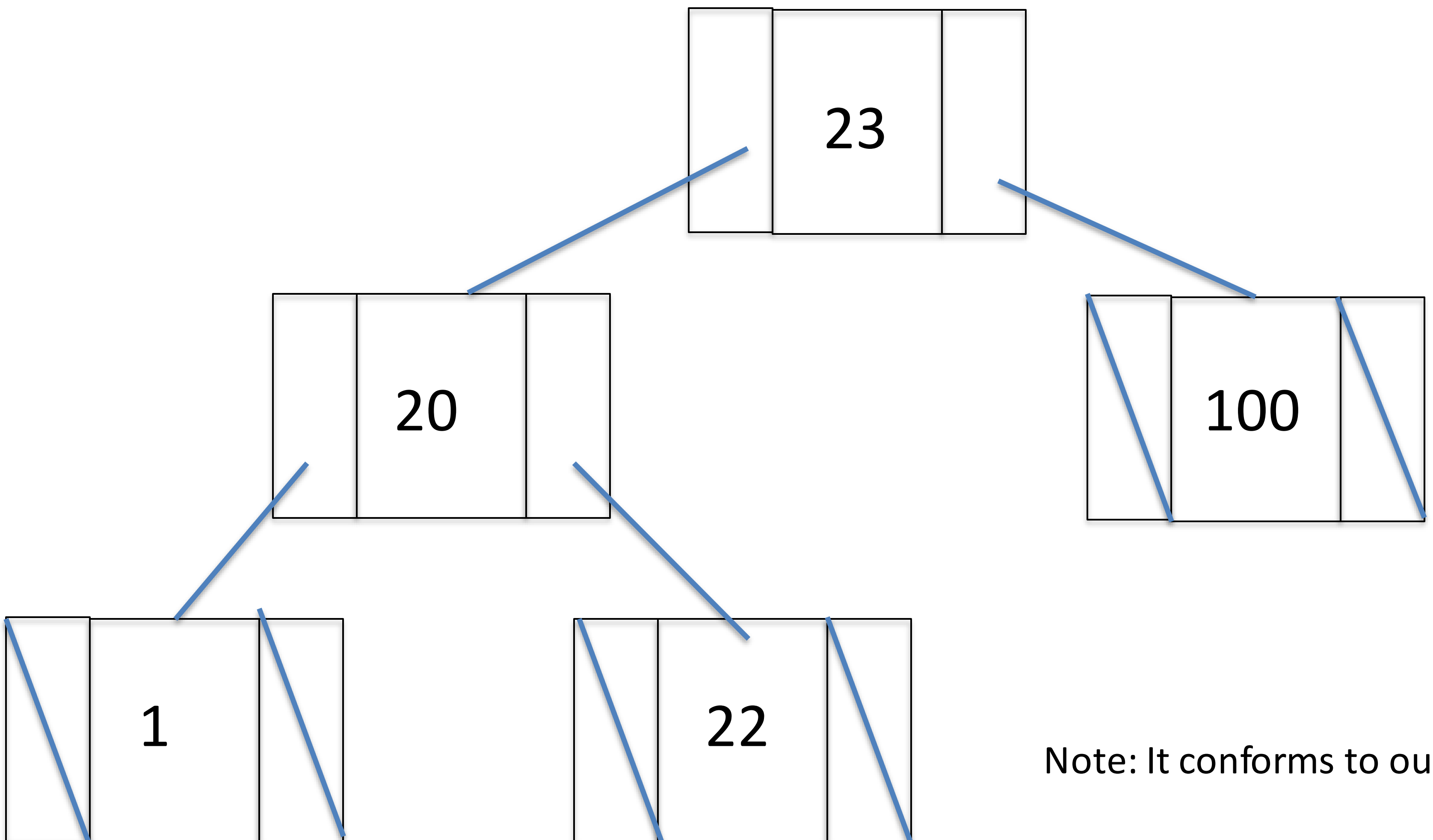
The *Map* ADT

- **Binary Search Trees** efficient way of storing a map
- A **map** allows us to store elements so that they can be quickly located with a key
  - E.g. a **dictionary**
- They key operations are
  - v = `get`(k)
    - Get element associated with key k
  - `put`(k,v)
    - Add value v associated with key k
  - `remove`(k)
    - Remove element identified by key k

Binary Search Trees

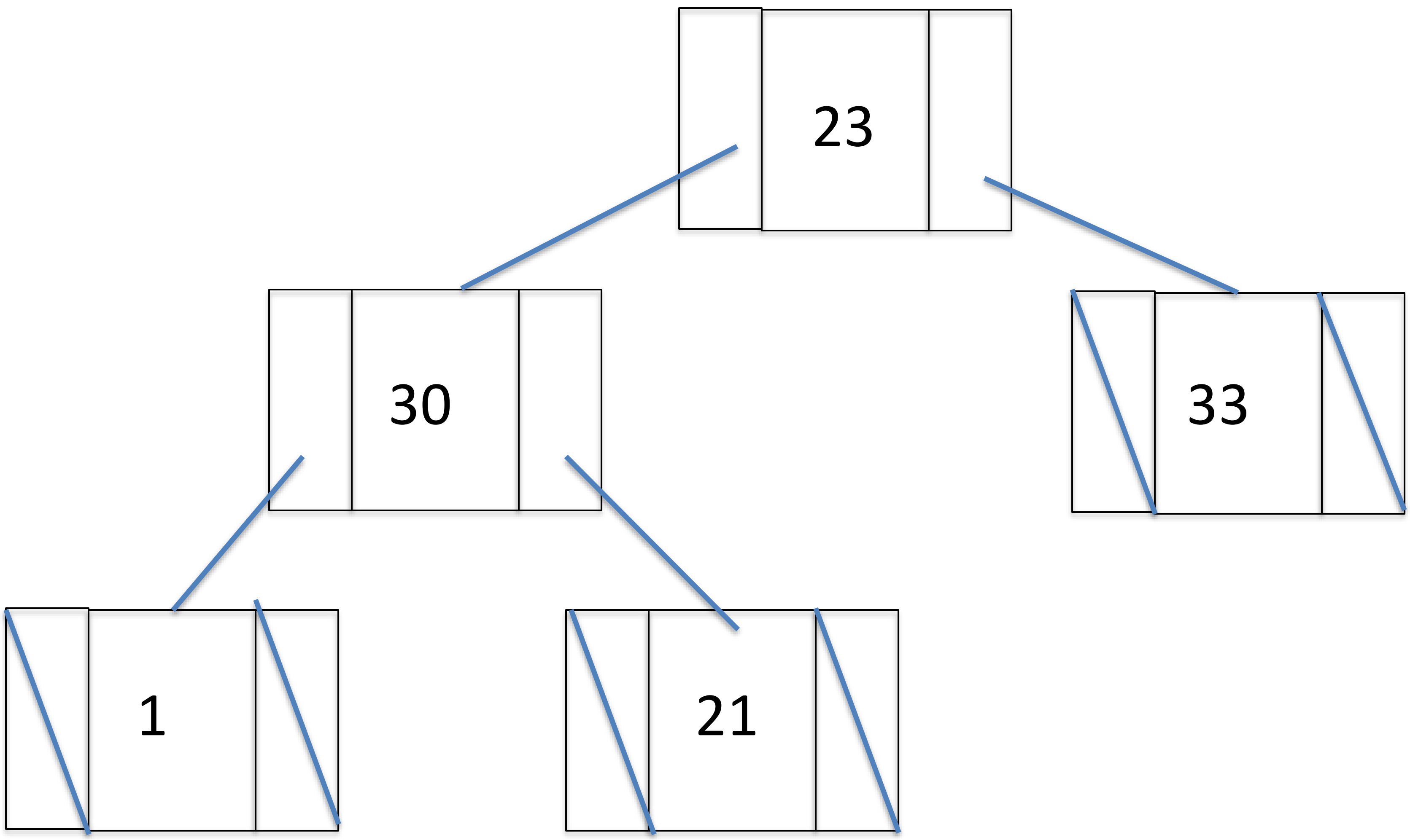
- An Efficient way to store map or ordered data
- Some additional rules
  - For the **value** (index) in the **current** node
    - All **values** (keys) in its **left** subtree are **smaller**
    - All **values** (keys) in its **right** subtree are **greater**
  - There is no = (i.e. no duplicate values allowed)

Binary Search Tree Example

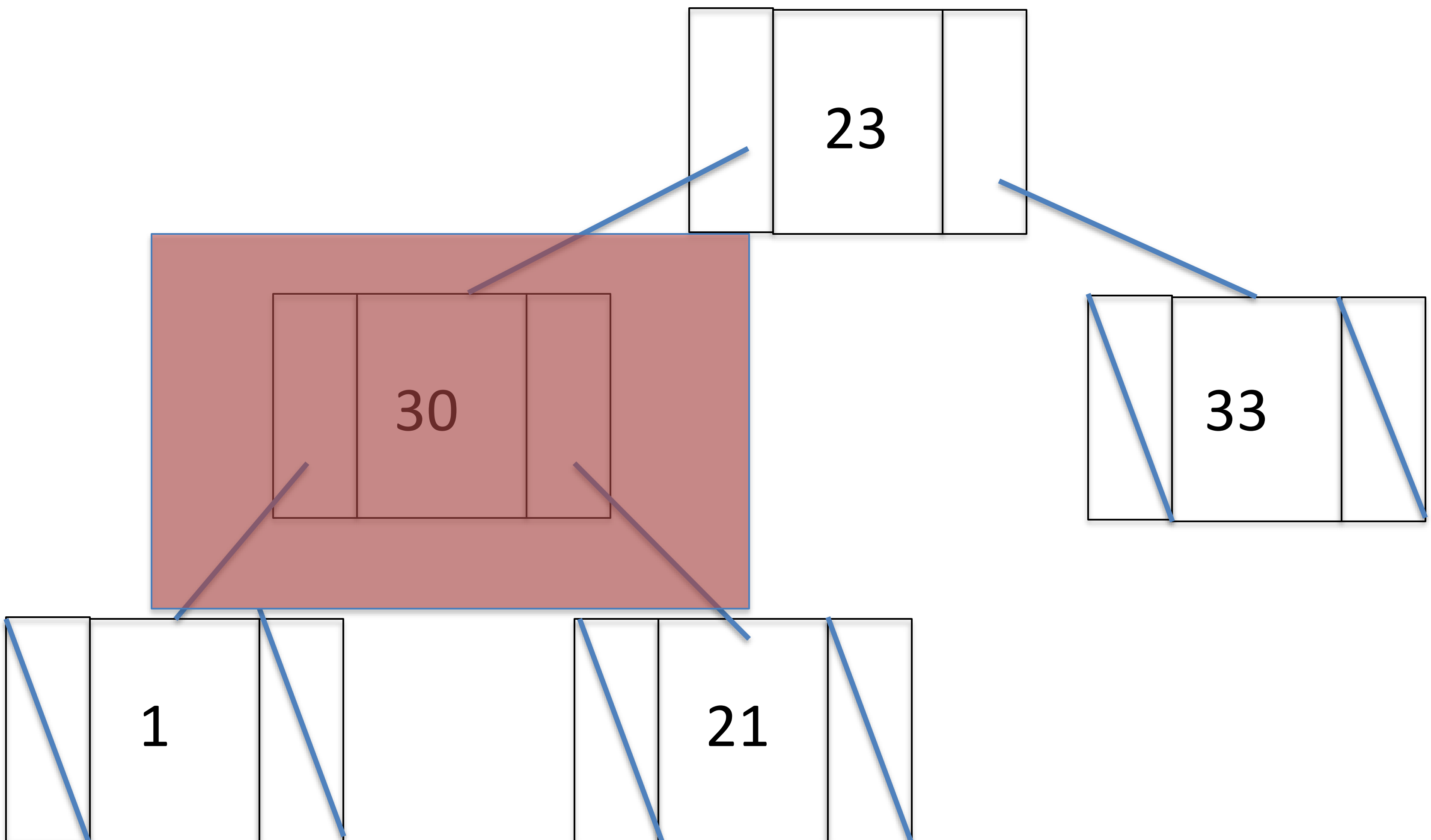


Note: It conforms to our rules

Not a Binary Search Tree – WHY?



Not a Binary Search Tree



Getting Around A Tree

- We term a structured order of (recursively) visiting each Node as a **Traversal**
- There are three types
  - **InOrder** Traversal
  - **PreOrder** Traversal
  - **PostOrder** Traversal

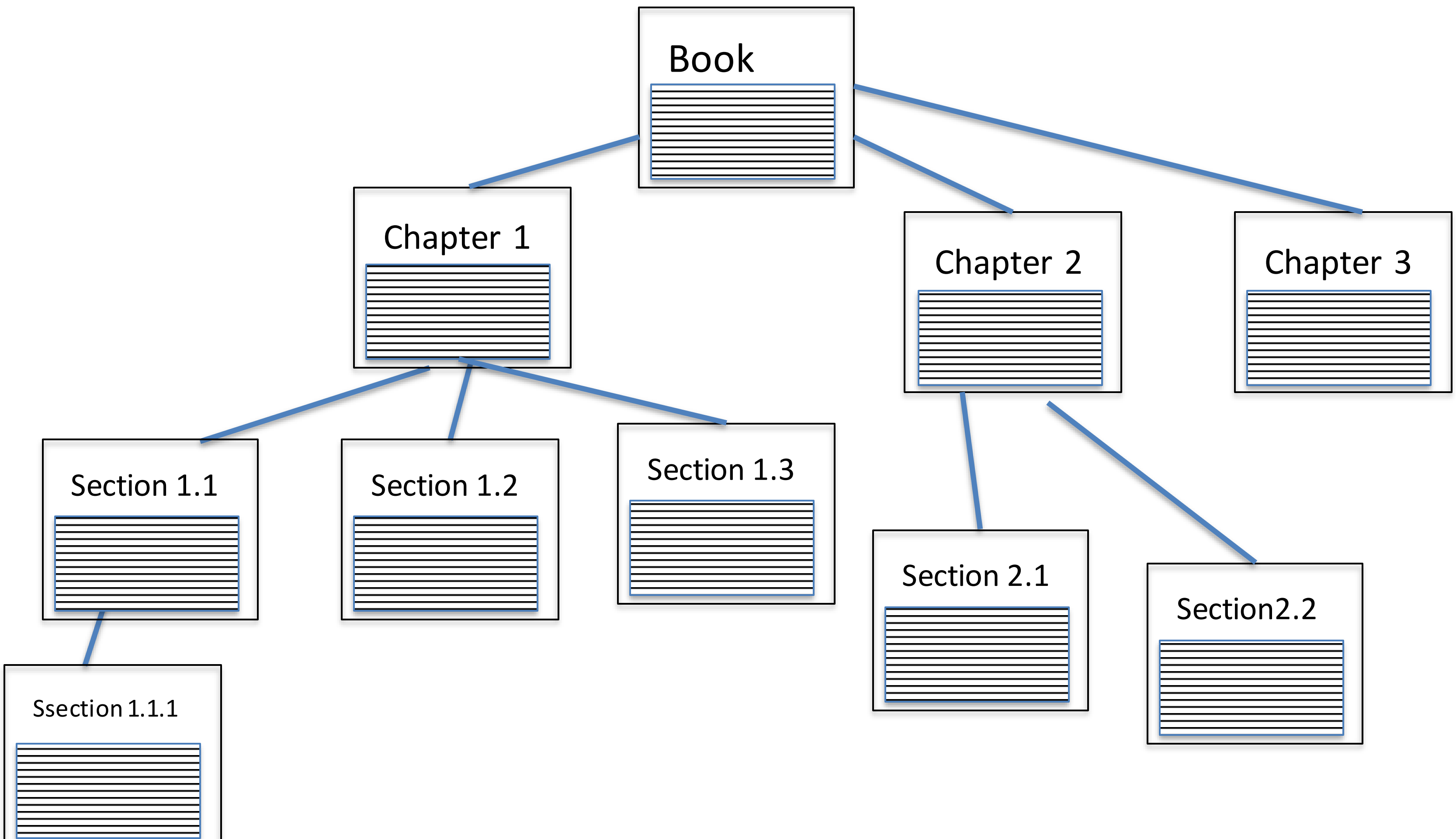
Getting Around A Tree

- InOrder, PreOrder and PostOrder Traversal
  – differ in terms of what we deal with first:
    - Current node value or children?
    - Which child?

- This applies to all forms of Trees
  – (not just binary trees)

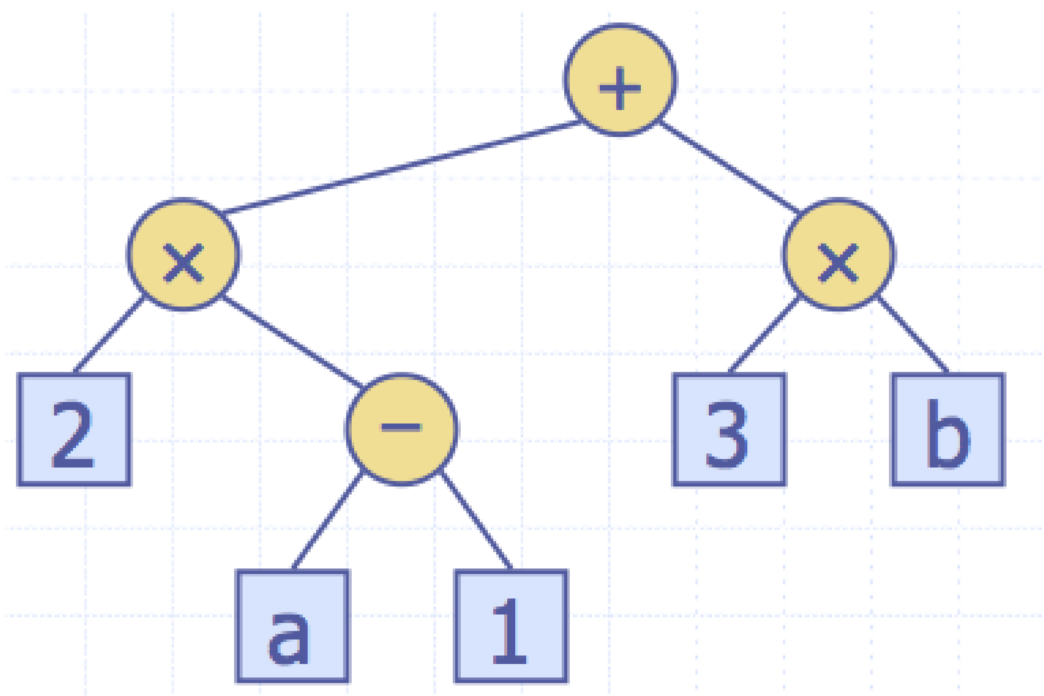
PreOrder Traversal

- Deal with the Node value, then the Children
  - E.g. **Printing** a Structured Document

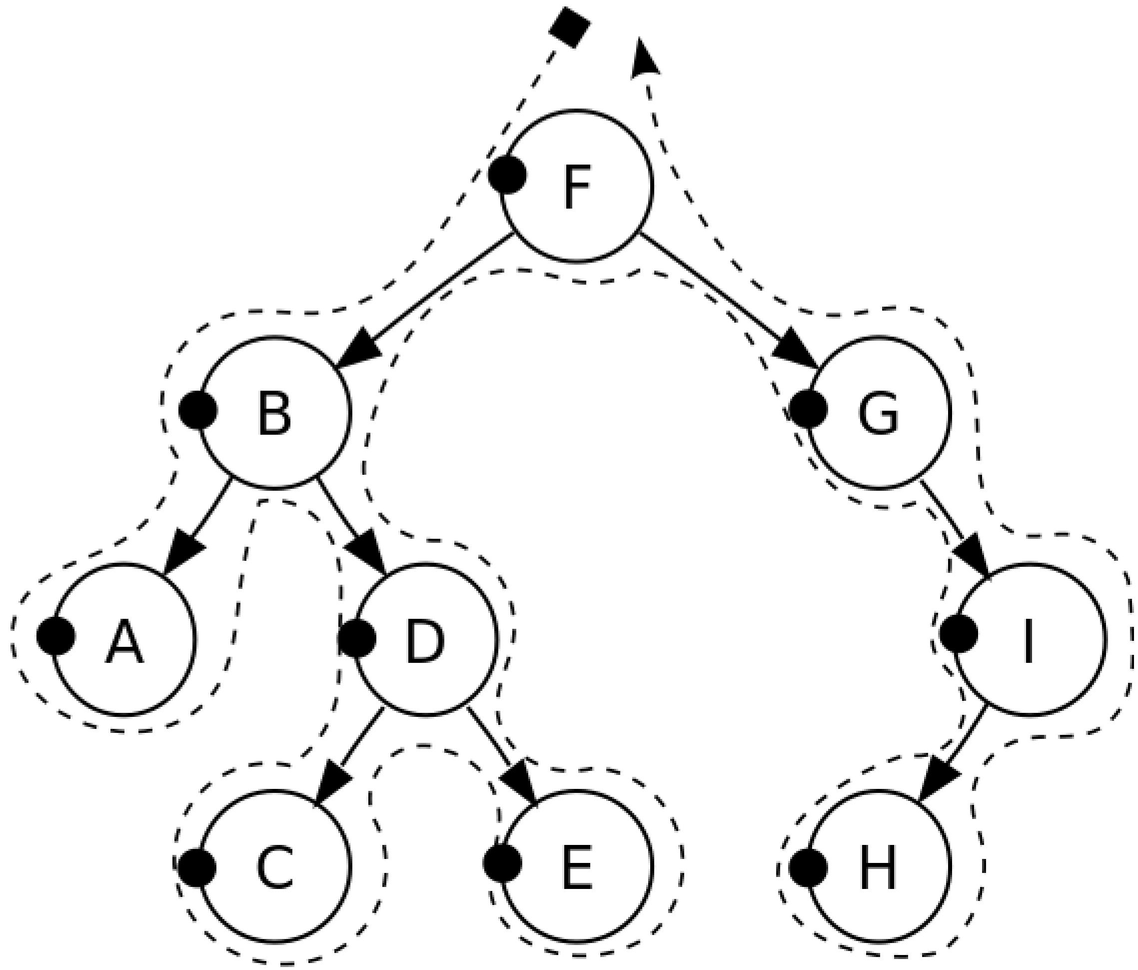


Post Order Traversal

- Deal with the children, then with node
  - E.g. Calculating an arithmetic expression



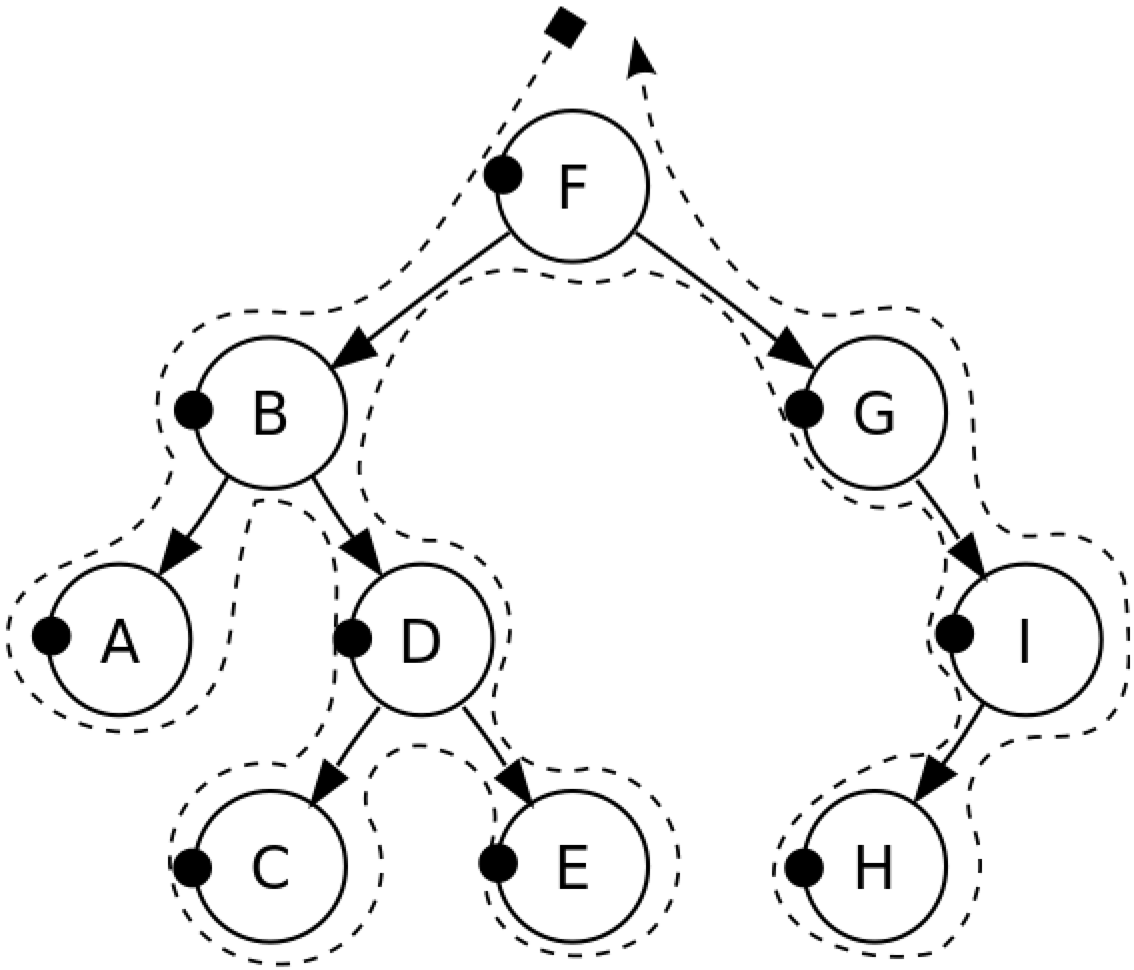
Pre-Order Traversal



Pre-order: F, B, A, D, C, E, G, I, H.

https://en.wikipedia.org/wiki/Tree_traversal

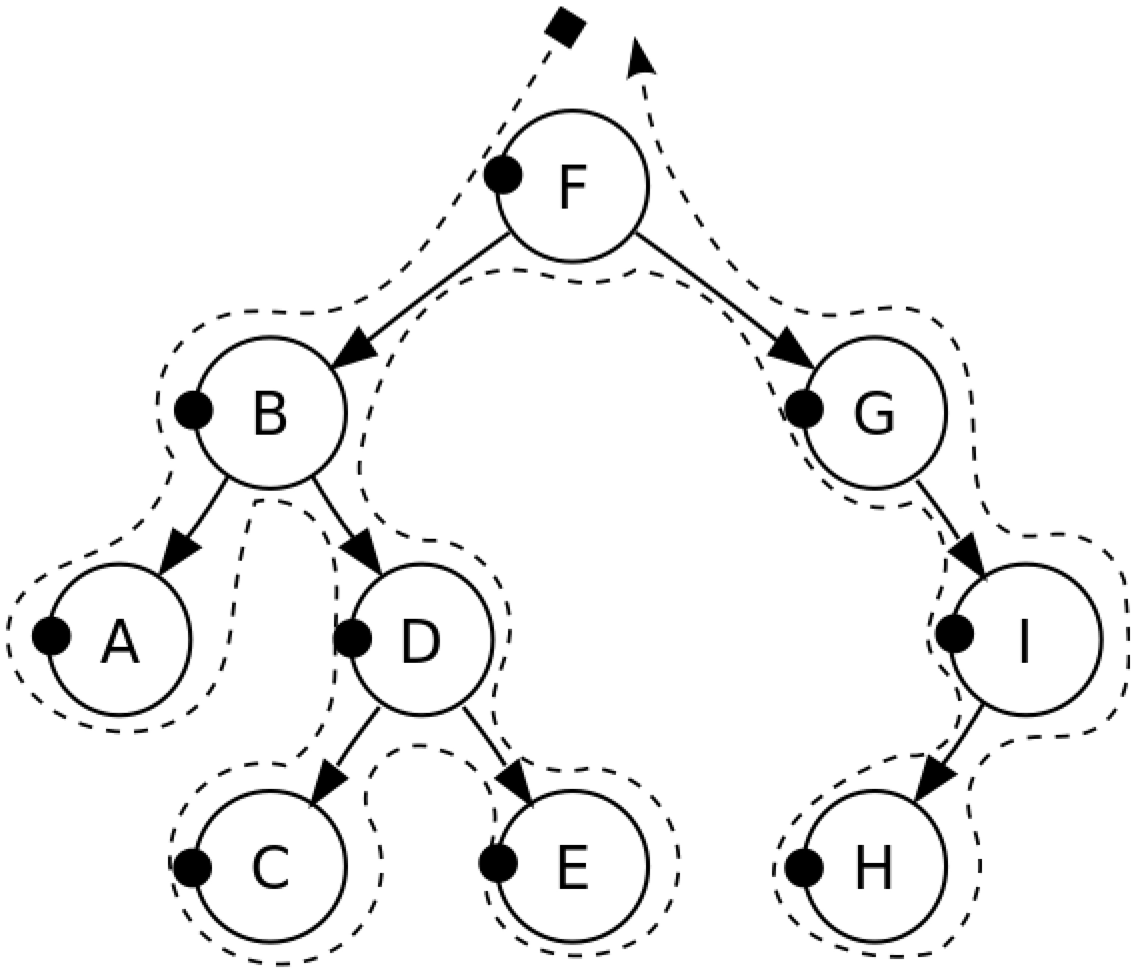
Post-Order Traversal



Post-order: A, C, E, D, B, H, I, G, F.

https://en.wikipedia.org/wiki/Tree_traversal

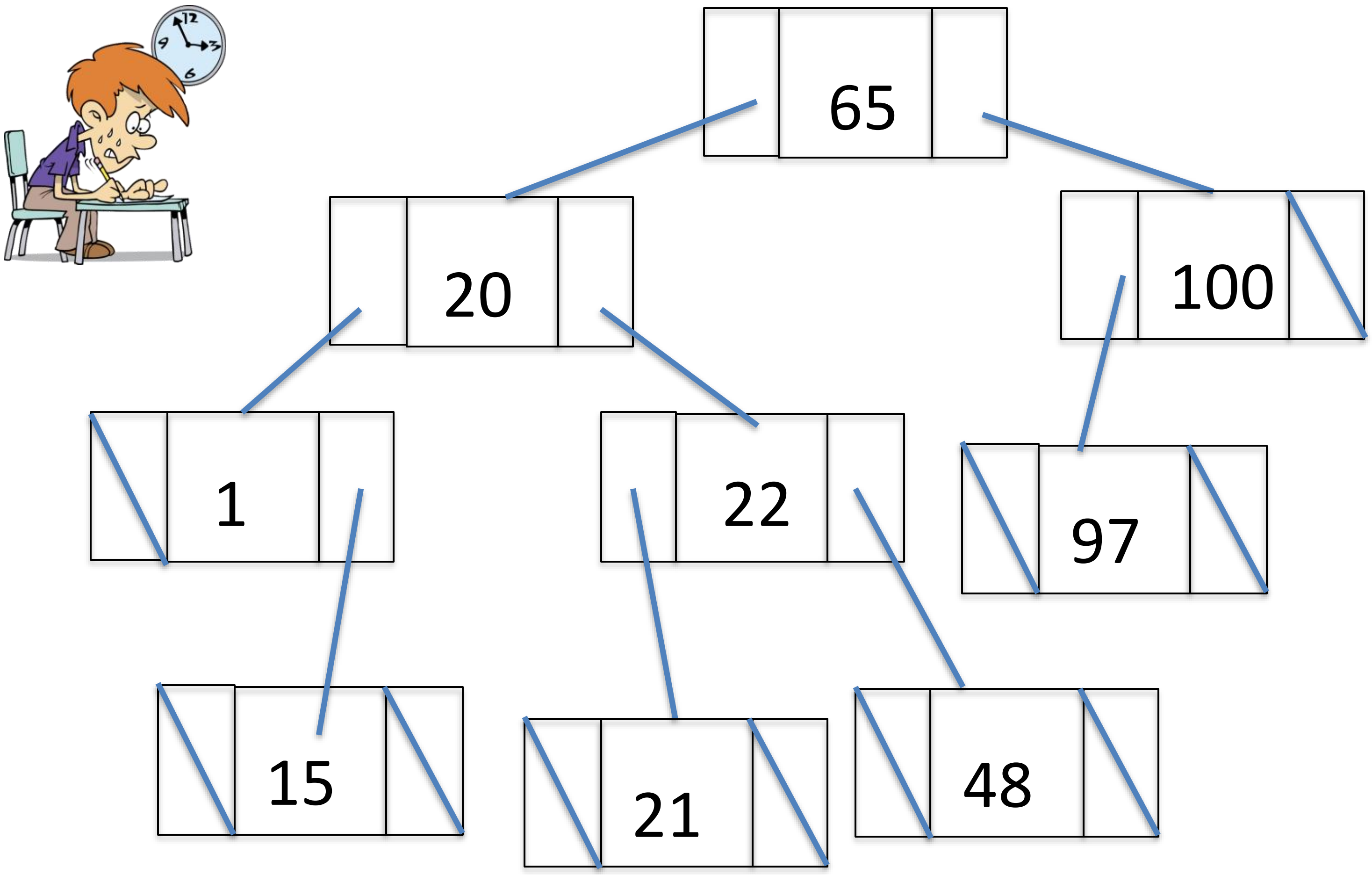
In-Order Traversal



In-order: A, B, C, D, E, F, G, H, I.

https://en.wikipedia.org/wiki/Tree_traversal

Traversal Exercise (homework)



Print numbers using **PreOrder**, **PostOrder** and **InOrder** traversal

Trees Space Race

Summary

- We have introduced the ***Map*** ADT
- We have introduced Trees
  - **Hierarchical** Data Structures
  - Support multiple types of data
  - Very useful in Computer Science
- **Efficient** ways to **store**, **search** and **reason** over data
- Various types of Tree
  - We've focused on **Binary Search Trees**
- **Next** lecture:
  - operations on trees, and how to implement them
  - and some of the pitfalls…
- Attendance sheet