Lec 23: Trees

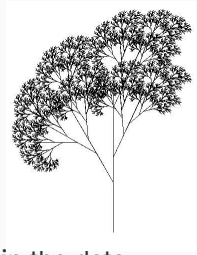
April 20, 2026

Admin

- Midterm back as soon as I can (likely end of the week)
- Apply to be a TA!!
    - Deadline April 24
- Any questions?

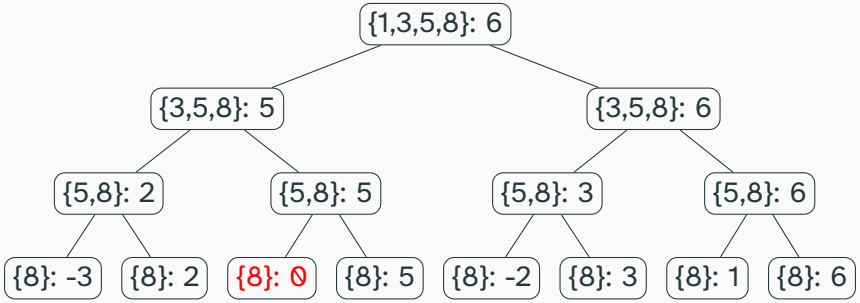
Trees

Trees



- All the ways we've had to store data have been one-dimensional.
  - At the end of the day: every item in our data structure is the *i*th item in the data structure for some *i*
  - All of our access has (indirectly) been through such a one-dimensional mapping
- With trees, we add a second dimension to how we store data
- *Drastic* improvements in what we can store and the performance we can achieve

Trees We've Seen



We can draw the method calls made by a recursive algorithm using a tree! (The above is `canMakeSum()` from lab 4.)

Here: each of the rectangles above (called a *node*) represents a recursive call. We connect each node to the nodes it calls.

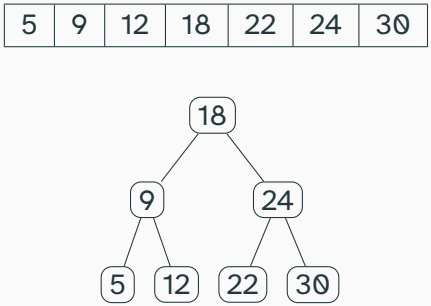
Trees We've Seen

5	9	12	18	22	24	30

How do we do binary search on this array? What do we compare to?

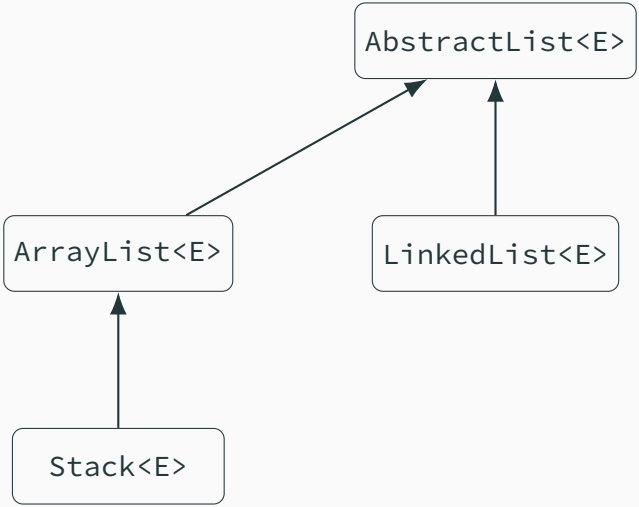
Something like: first, we compare our query element to 18. Based on the result, we then compare it to either 9 or 24.

Trees We've Seen

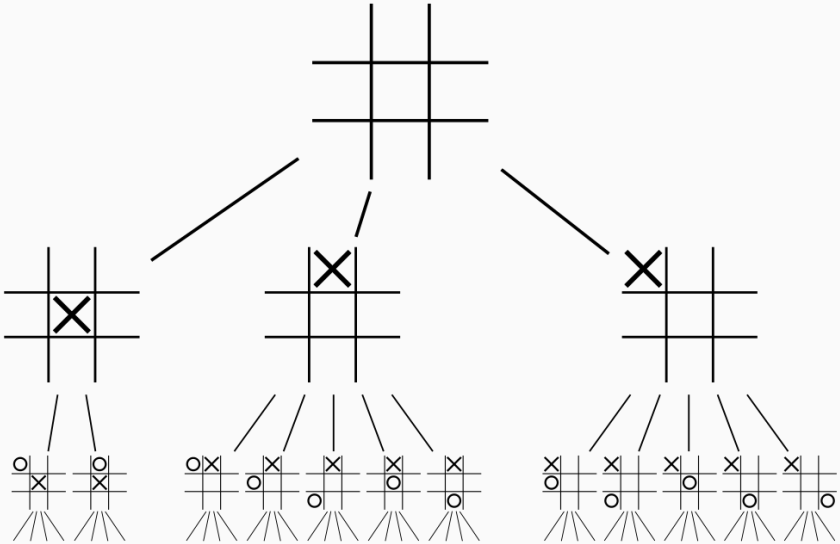


Binary search seems to also have a tree-like structure. We'll see how to store data in a very similar tree very soon.

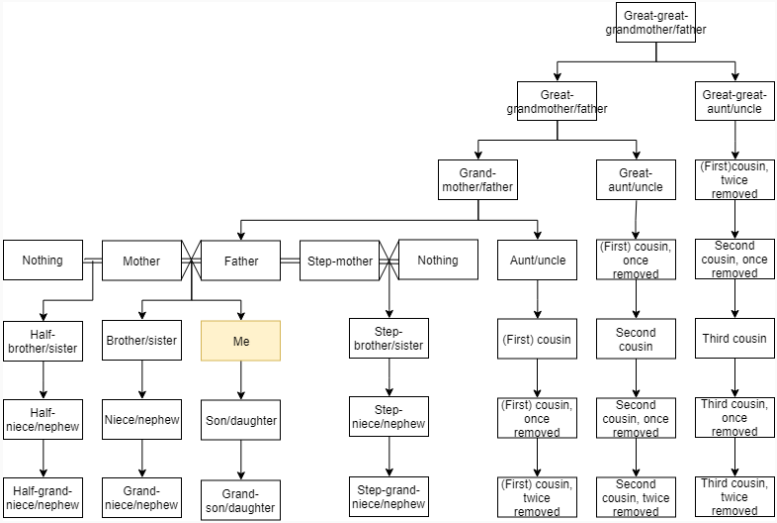
Inheritance



Game Tree



Family “Tree”

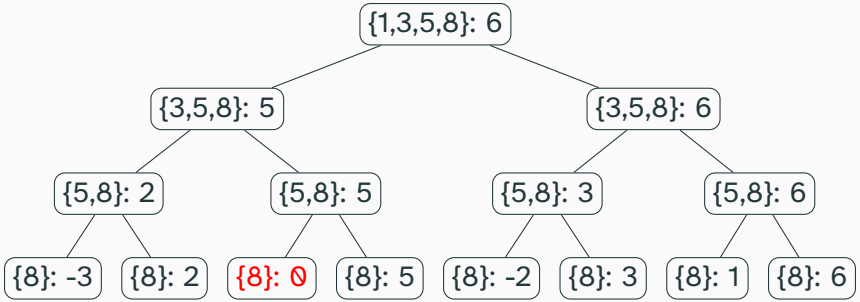


Same basic idea. Though note: not quite a tree by our definition.

Definition of a Tree

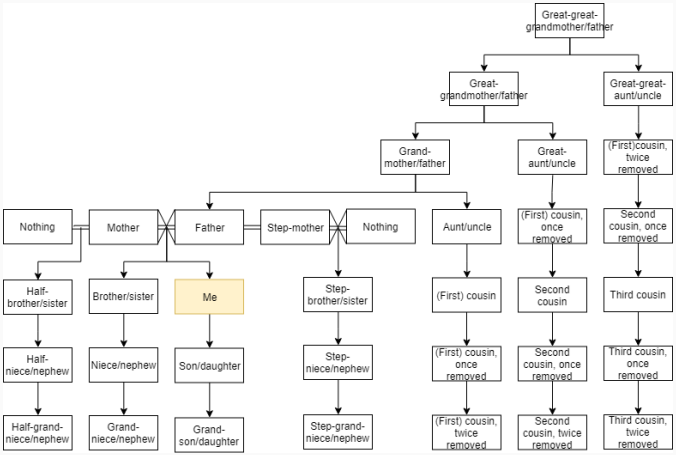
- Tree consists of nodes (the boxes in the images we saw above)
- Nodes are connected by edges (lines in the images we saw above)
- There is one root node that does not have a *parent node*
- Every other node has exactly one *parent node*
- Nodes may have some children.
- A node without a child is called a *leaf*

Labeling nodes



What is the root node in this tree? What are the leaves?

Family “Tree”



Why isn’t this a tree?

- Answer: nodes have multiple parents! (Plus there are a bunch of extra edges in this image.)

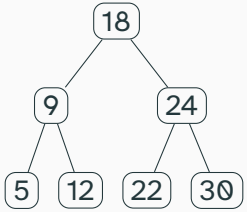
Binary Tree

Binary Tree

- Binary Tree: A tree where each node has at most 2 children
- The degree of a node is the number of children it has. So a binary tree is a tree where all nodes have degree at most 2.
    - We'll be using "degree" quite a bit in this part of the course
- Let's see an example of a binary tree. Then, we'll discuss how to implement a binary tree to hold data in Java

Tree of Binary Search Calls

5	9	12	18	22	24	30



How to Store a Binary Tree?

- Nodes should probably be objects of some class type; we'll call it `BinaryTreeNode<E>`
- Each node keeps track of its children, its parent, and some data (of type E)
    - Similar to a `LinkedList<E>`
- The `BinaryTree<E>` just keeps track of the root of the tree
- Let's look at the code

What's Missing?

- We are storing a bunch of data in a tree
- But so far, we can't *do* anything with it
- With `LinkedList<E>` we could call `set()` and `add()` to add elements, or `get()` and `indexOf()` and `contains()` to query the elements
- How do we do that here?

Tree Methods

- How we interact with a tree depends on the type of data being stored in the tree
- For example: a game tree and a family tree aren't going to be used in the same way
- Plan: let's look at one of the most common kinds of tree, the Binary Search Tree
- Then we'll see some examples of other kinds of trees

Binary Search Trees

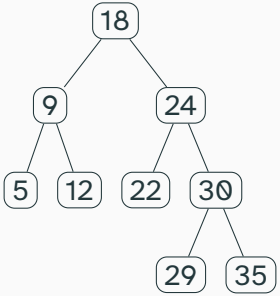
Finding Items Using Trees

- Goal: store items in a tree such that we can implement methods like `add()` and `contains()` efficiently
- Don’t want to traverse the entire tree
- We will assume our data is “ordered” (is `Comparable` or has a `Comparator`): we want to get binary-search-like performance

Binary Search Tree Invariant

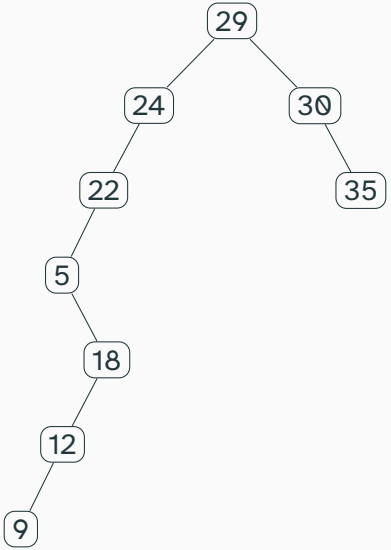
- The following always holds for a binary search tree.
- For *every* node n in a binary search tree with value v:
  - All values v_ℓ of nodes that are descendants of the left child of n have values $v_\ell \leq v$
  - All values v_r of nodes that are descendants of the right child of n have values $v_r > v$

Binary Search Tree Examples



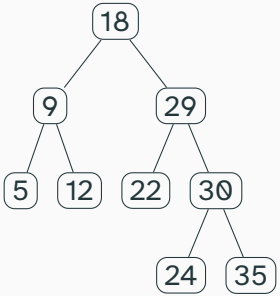
Is this a binary search tree?

Binary Search Tree Examples



Is this a binary search tree? (It has the same elements!)

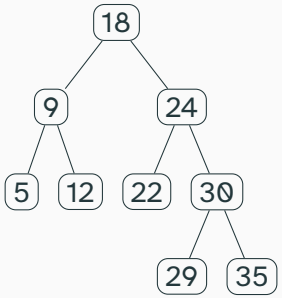
Binary Search Tree Examples



Is this a binary search tree?

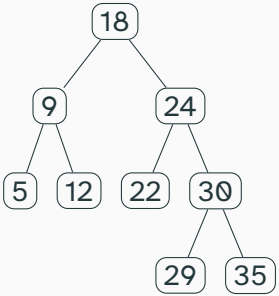
No: note that *all* right descendants must be greater than the node

Finding an element in a binary search tree



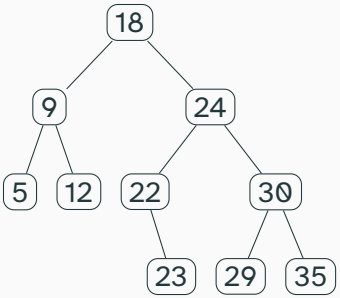
- In pairs: How can I search for an element (say 14)?
- Idea: we can look at a node and know immediately if the element we're searching for is a descendant of the left child, or of the right child
- Go to the appropriate node, and continue
- If we find the element, or if we hit an empty node, we're done
- This is essentially a *recursive* algorithm. (We will implement it with a loop however.)

Adding an element to a binary search tree



- In pairs: How can I add an element (say 23)?
- Idea: we can look at a node and know immediately if the element we're adding should be a descendant of the left child, or of the right child
- Recurse on the appropriate node
- If we hit an empty node, replace it with the element we want to add

Adding an element to a binary search tree



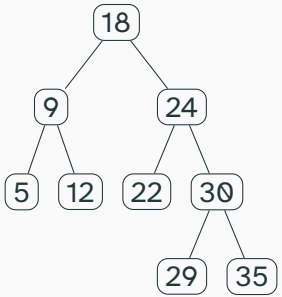
- In pairs: How can I add an element (say 23)?
- Recursively!
- Idea: we can look at a node and know immediately if the element we're adding should be a descendant of the left child, or of the right child
- Recurse on the appropriate node
- If we hit an empty node, replace it with the element we want to add

Implementing a Binary Search Tree

Comparing Elements

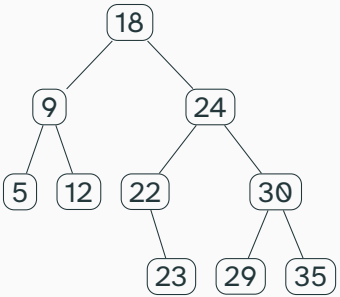
- Need some kind of way to compare elements
- What are our options?
    - Store `Comparable` items, or use a `Comparator`
    - We'll use a `Comparator` so that our tree works with any kind of item

Finding an element in a binary search tree



- Idea: we can look at a node and know immediately if the element we're searching for is a descendant of the left child, or of the right child
- Recurse on the appropriate node
- If we find the element, or if we hit an empty node, we're done
- Let's write the code for this

Adding an element to a binary search tree



- Idea: we can look at a node and know immediately if the element we're adding should be a descendant of the left child, or of the right child
- Recurse on the appropriate node
- If we reach `null`, instead add a new node with the element we want as data

Natural Comparator

- Let's say we have an item of type `E` that implements `Comparable<E>`
- That means we can already compare items of type `E`
- But, we want the flexibility to compare them other ways using a `Comparator<E>`
- The `NaturalComparator<E>` implements `Comparator<E>`, and compares items using their `compareTo()` method
- That way, we can write code assuming we always have a comparator; if we want we can replace it with a different comparator
- Let's look at the code

Tree Vocabulary

- *Descendant*: A node n' is a descendant of node n if there exists a sequence of nodes $n = n_1, n_2, \ldots, n_i = n'$ such that for all $1 \leq j < i$, n_{j+1} is a child of n_j. (*Ancestor* is the opposite)

- *Siblings:* Two nodes are siblings if they share the same parent

- *Path*: the unique shortest sequence of edges between two nodes n_1 and n_2. Each successive edge in the path must share one of its nodes with the previous edge

Tree Vocabulary

- The *degree* of the tree is the maximum number of children of any node
- The *depth* of a node is the number of edges on the path to the root
- The *height* of a tree is the maximum depth of any node

Binary Search Tree Analysis

- How much time does a call to `contains()` take?
  - Worst case
  - Definitely not worse than $O(n)$ (we never look at a node multiple times)
  - Is there a tree where it's actually $O(n)$? Yes; let's try to create an example on the board
- Let's say we have a tree of height h. How long does a call to `contains()` take in terms of h?
  - Each time we call the method the depth of the node increases by one, so $O(h)$
  - If we have time: how can we prove this by induction?

Binary Search Tree Analysis

- How much time does a call to `add()` take?
  - $O(n)$ in a tree of size n
  - $O(h)$ in a tree of height h

Tree Discussion

- How many nodes are in a full tree of depth *h*?
- How can we sort using a Binary Search Tree?
- How much time does this take?