CSC148 – Introduction to Computer Science

Lecture 7: Binary Search Trees Part 1

Sean Henderson

Binary Trees

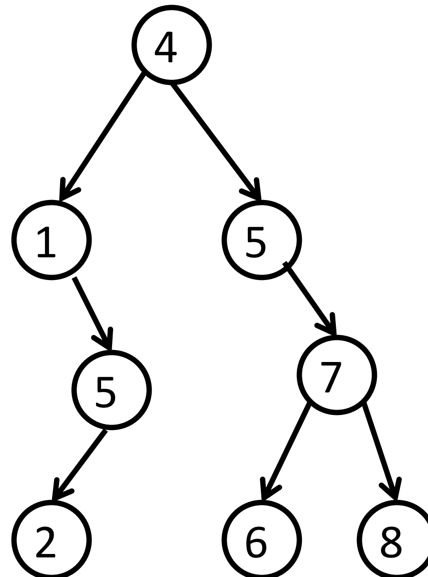
- Talked about these in prior lectures
- Usefulness not immediately apparent other than to directly model existing structures
- Let’s try and modify a binary tree so that it is more useful…

Binary Search Tree

- A binary tree with a special property
- For every node in the tree x:
  - For every node y in x’s left subtree, y.value < x.value
  - For every node z in x’s right subtree, x.value < z.right

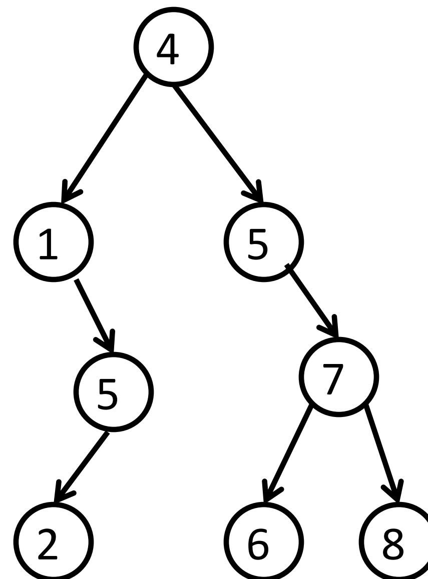
Example

- Is this tree a Binary Search Tree?



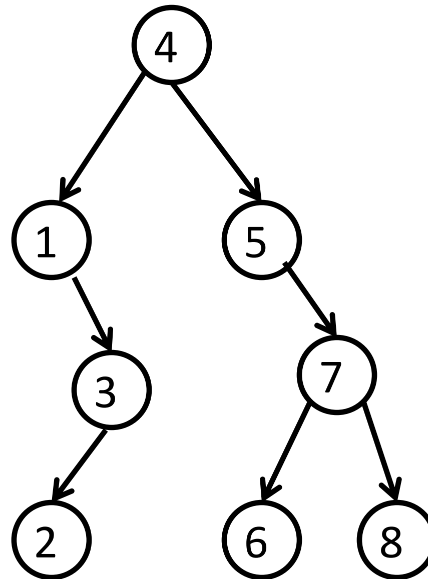
Example

- Is this tree a Binary Search Tree?
- No; 5 is in 4’s left subtree.



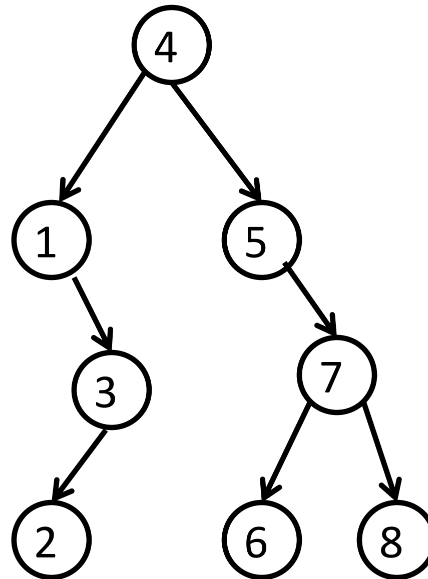
Example

- Is this tree a Binary Search Tree?



Example

- Is this tree a Binary Search Tree?
- Yes



Basic Operations

- Search for a key in the tree
  - Test for membership in a set, or modify our try like the question from the midterm to store (key, value) pairs
- Insert a key into the tree
- Remove a key from the tree

Searching

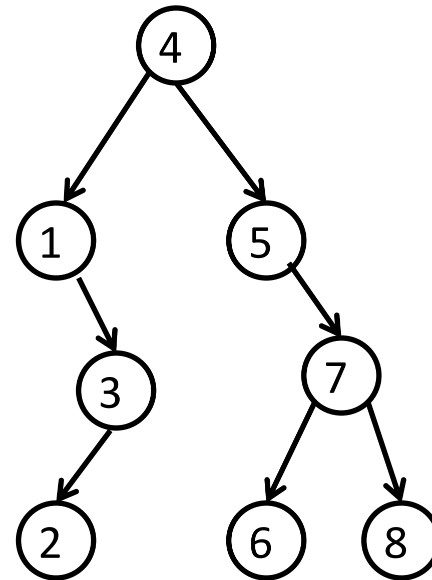
- Very similar to binary search
- Can be defined recursively
- Base case: empty tree (return false)
- Otherwise: check if the value stored at our current node is the same as they key; if so, return true; otherwise recursively search left and right subtrees
- Pre-order traversal

Searching

- The runtime of that algorithm would be O(n)
- Did not make use of this new, shiny binary search tree property
- Consider searching at a node x
- If x.value == key, we're done
- What if x.value < key? x.value > key?

Searching

- What if we're searching for 2?
- 7?
- 5.5?



Searching

- Only need to search the right subtree if key > x.value; similar argument for left subtree
- Let’s implement this…

Searching

- What is the runtime of search now?
- In the worst case we start at the root, and never find what we are looking for
- Continue until we hit a leaf, and then stop
- So the runtime is O(height of the tree)
- It would be nice if we could show some bound on the height of the tree…

Insertion

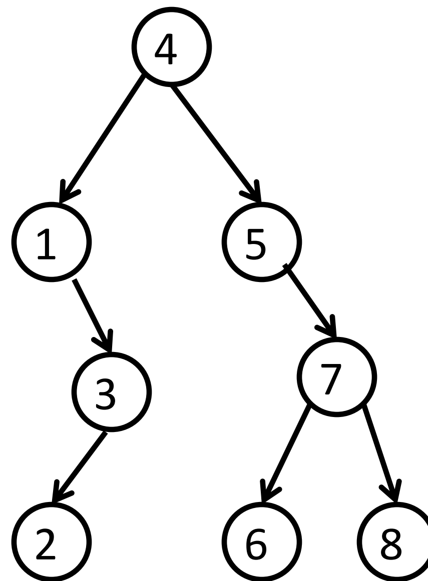
- How can we insert a value into the tree, while maintaining the binary search tree property?
- Could scrap the entire tree and rebuild it from scratch… or…

Insertion

- Using similar reasoning from search…
- If x is the root of some subtree into which we want to insert key
  - If key == x.value then it is already in the tree
  - If key < x.value then it **must** be inserted into the left subtree (or this would violate our BST property)
  - If key > x.value, must insert it to the right…

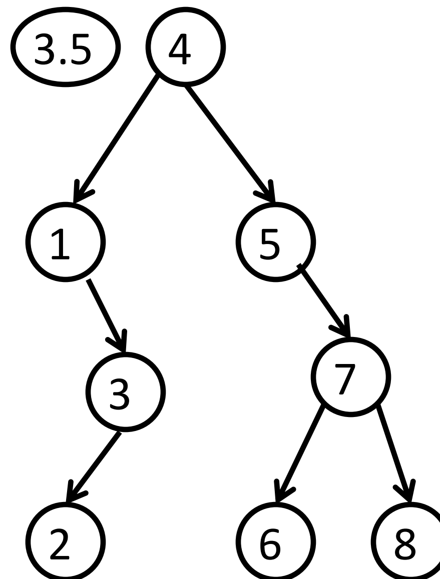
Insertion

- Say we want to add 3.5 to the tree



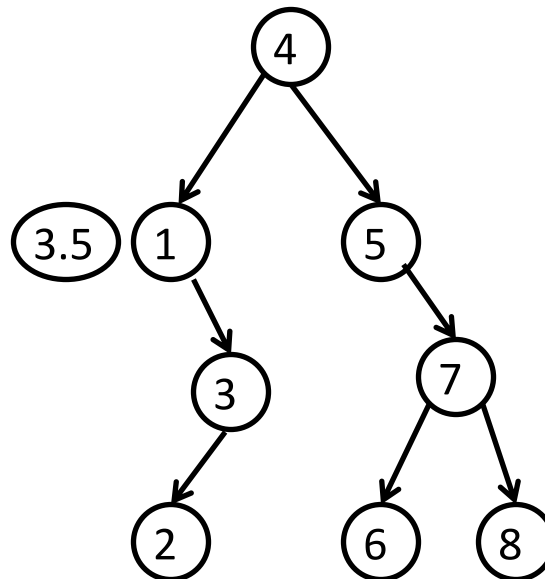
Insertion

- Say we want to add 3.5 to the tree
- Compare 3.5 to 4



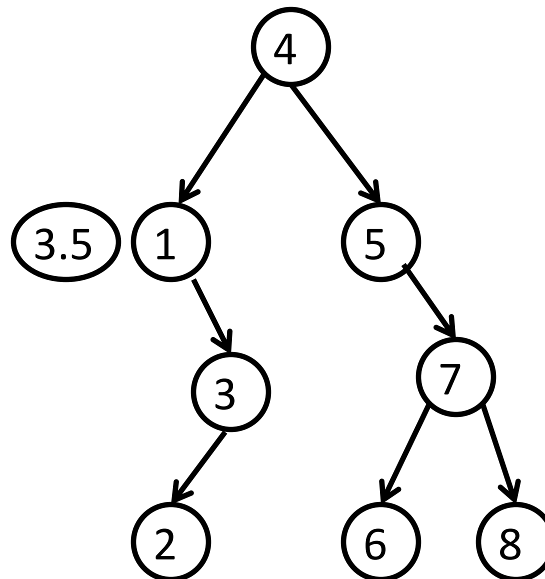
Insertion

- Say we want to add 3.5 to the tree
- Compare 3.5 to 4
- Insert left



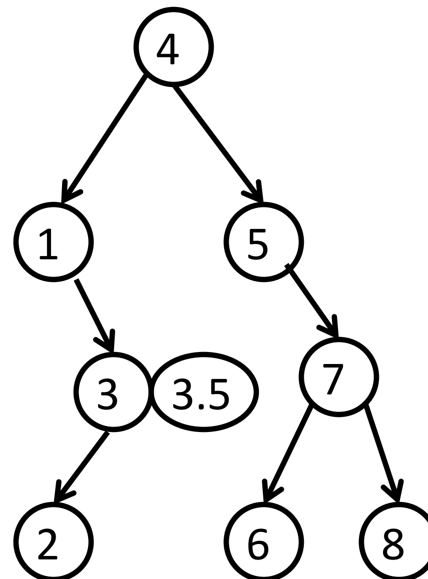
Insertion

- Say we want to add 3.5 to the tree
- Compare 3.5 to 4
- Insert left
- Compare 3.5 to 3



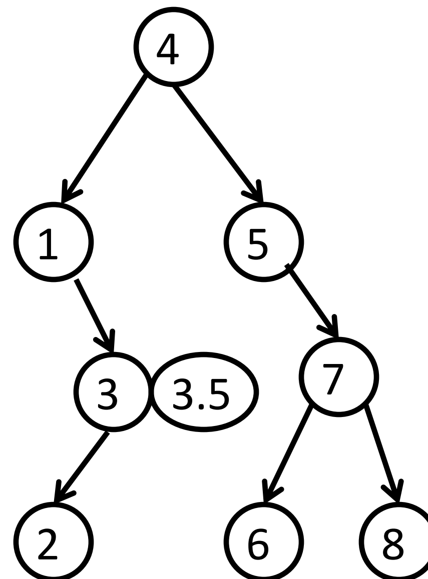
Insertion

- Say we want to add 3.5 to the tree
- Compare 3.5 to 4
- Insert left
- Compare 3.5 to 3
- Insert right



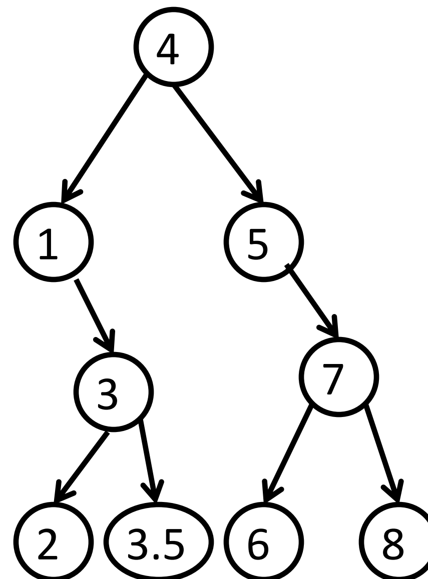
Insertion

- Say we want to add 3.5 to the tree
- Compare 3.5 to 4
- Insert left
- Compare 3.5 to 3
- Insert right
- Compare 3.5 to 3



Insertion

- Say we want to add 3.5 to the tree
- Compare 3.5 to 4
- Insert left
- Compare 3.5 to 3
- Insert right
- Compare 3.5 to 3
- Insert right (done!)



Insertion

- Let’s implement this…

Insertion

- Theorem: a randomly constructed binary search tree has (expected) height O(log n)
- That means we can insert and search in time O(log n)
- How does this compare to a linked list?
- Bonus: inserting things into a BST then outputting an in-order traversal sorts the list
- O(n log n) sorting algorithm (with high probability)