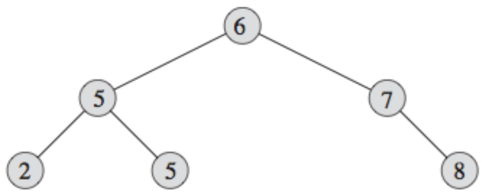
**Binary search tree**

**Operations:** search; min; max; predecessor; successor. Time O(h) with h height of the tree (more on later).

**Data strutcure fields usually include for a given node x, the following attributes:** x.left (left child); x.right (right child); x.p (parent); x.key. Root of entire tree pointed to by T.root

**<u>Example of storing keys in a binary search tree</u>:** (binary because branches either to left or to right)

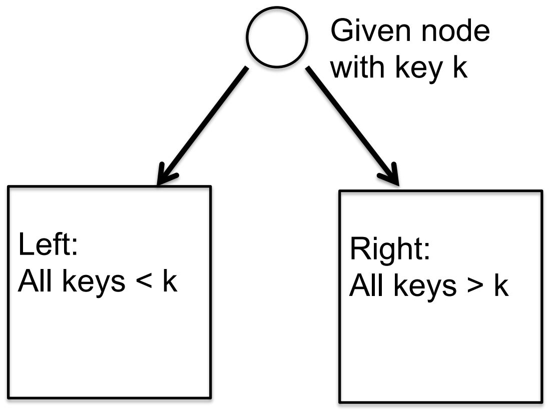


**Do you see a pattern?**

**Main property of binary search tree:**

**Left side:** key smaller equal to parent

**Right side:** key larger equal to parent

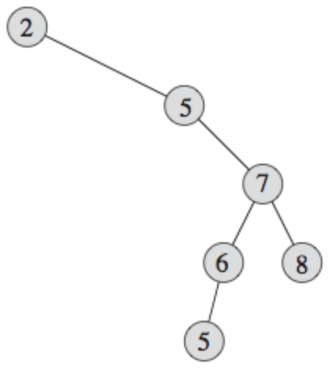


You can see this in tree example above node by node going from parent to child, and it is also true for the whole subtree to the left and to the right.

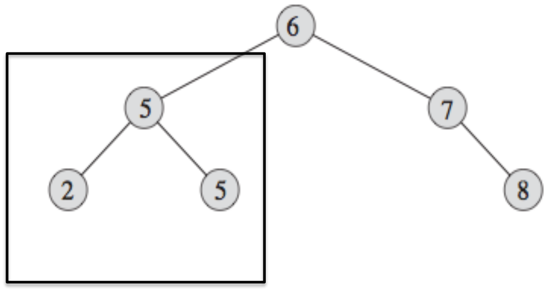
**<u>Is this tree unique for the given set of keys?</u>**

**<u>Answer: No; The same set of keys have many possible trees.</u>**

**Another example:**



**Question:** How can we traverse the tree and print out keys in ascending order:



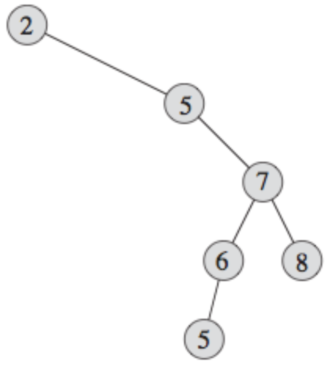
First left side:
2 5 5

Then root:
6

Then right:
7 8

```
INORDER-TREE-WALK(x)
1  if x ≠ NIL
2      INORDER-TREE-WALK(x.left)
3      print x.key
4      INORDER-TREE-WALK(x.right)
```

What about this one? (answer: same: 2 5 5 6 7 8)



**Runtime for inorder-tree-walk**? Phi(n) for a binary tree with n nodes

Why? Omega(n) because visits every node of the tree at least once and print the key; book shows also O(n) using subsitution method (it's intuitive in any case that you traverse all nodes no more than constant amount of times).

**Query operations on binary search tree**: search; max; min; successor; predecessor.

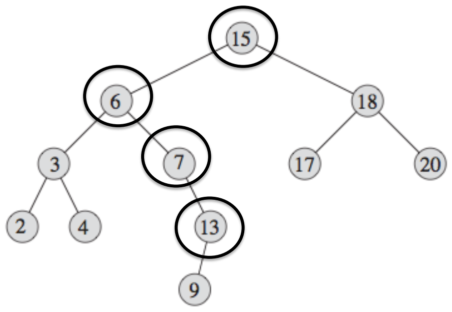
**Run time depends on? Height** of binary search tree.

**Question: Is the height always log n?**
**Answer:** Not necessarily, since tree could be made non efficiently and each time branch to the right only. Then its run time could be O(n) and as bad as a linked list.

Therefore height and run time for search queries could be O(log n) but could also be O(n)!

**Search:** Main idea of searching for key k: if looking for k smaller than node.key, search to the left, otherwise to the right.

Searching for k = 13:



Search trajectory for k=13: 15 -> 6 -> 7 -> 13

Tree-Search(x,k)

```
1. if k=x.key or x==nil return x
2. if k < x.key return Tree-Search(x.left, k)
3. else if k > x.key return Tree-Search(x.right, k)
```

(more formal in book and can also be written iterative in book)

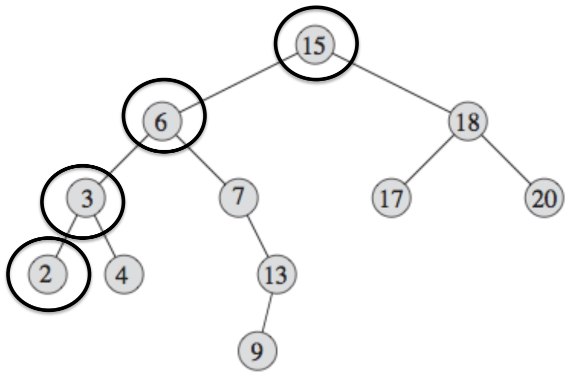
**Run time: height of tree. Again, could be O(log n) or O(n) depending on tree and its balance.**

**<u>Min/max:</u>**

**Question: finding the min?**

**Answer:** keep traversing to the left subtree until we encounter a nil

Finding the min:

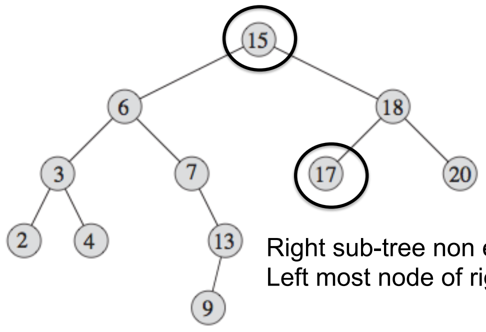


**Finding the max:** keep traversing to the right subtree…

**Successor: (similar for predecessor)**

Case 1: Right sub-tree non empty: left-most node of right subtree (why? It's the smallest value that is larger than the given node)

Example case 1: successor of 15



Right sub-tree non empty:
Left most node of right sub-tree

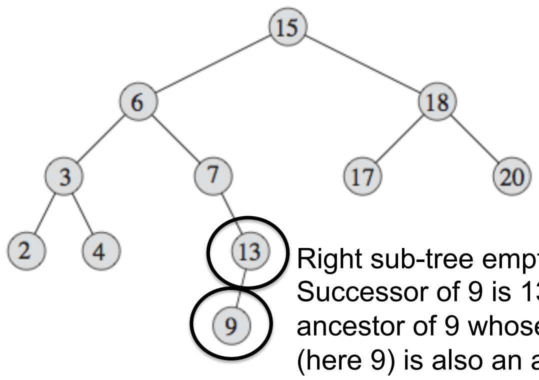
---

Case 2: Right subtree is empty, then successor is an **ancestor** (we'll first define ancestor and then more specifically what we mean here)

**Ancestor definition in Cormen book:** an ancestor of node x is any node from the root to node x. This is either a "parent" of x or the node x itself.

**More specifically: successor of node x is** an ancestor that is the lowest ancestor of node x, whose left child is also an ancestor of x. Put differently:

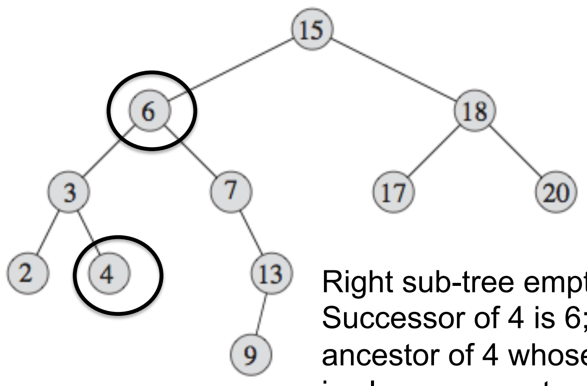
Example case 2: successor of 9



Right sub-tree empty:
Successor of 9 is 13; lowest ancestor of 9 whose left child (here 9) is also an ancestor of 9

---

Another example case 2: successor of node 4



Right sub-tree empty:
Successor of 4 is 6; lowest ancestor of 4 whose left child is also an ancestor of 4

---

Why does this work? Consider node x (here node 4). Intuitively, if left child of ancestor y is also an ancestor of node x (here node 6), then node x is smaller than the ancestor y. All ancestor nodes before y (here before node 6 = node 3) do not have a left child that is an ancestor of x, which means that x is a right child of that ancestor and so that ancestor is smaller than x.

Another example case 2: successor of node 20

Answer: none. Why?

Another example case 2: successor of node 13 is 15. Why?

Run time of successor: O(height), since we either follow a path up the tree or a path down the tress