Binary Search Tree

Class 21

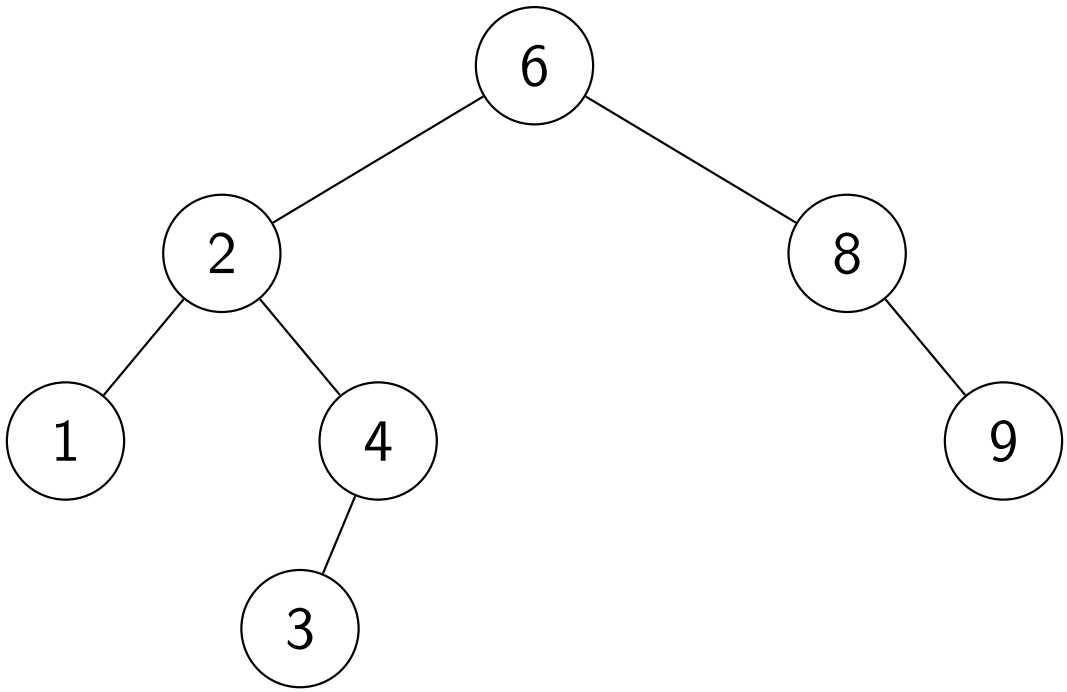
BST

- binary search tree is the name for a container with the following characteristics
  - a key which identifies an entry (like a hash)
  - a corresponding value consisting of data (like a hash) (like a hash, we'll mostly focus on key and ignore data)
- and with the following behaviors
  - find a specific element in the collection by key
  - insert a new element into the collection
  - remove an element from the collection by key
  - findmin find the minimum value in the collection
  - findmax find the maximum value in the collection
  - miscellaneous: is_empty, size, etc.

BST Data Structure

BST

A binary tree in which each node contains data which includes a key field. The root node is either empty or contains a key and two children. A non-empty root's left child is either empty or has a key of lower value; a non-empty root's right child is either empty or has a key of higher value; both children are recursively BSTs.



find(key)

- start at the root
- if the root is empty (nullptr), return false
- else if the root's key is the searched-for key, find returns true
- else if the root's key is larger than the searched-for key, recurse on the left child
- else recurse on the right child

findmin()

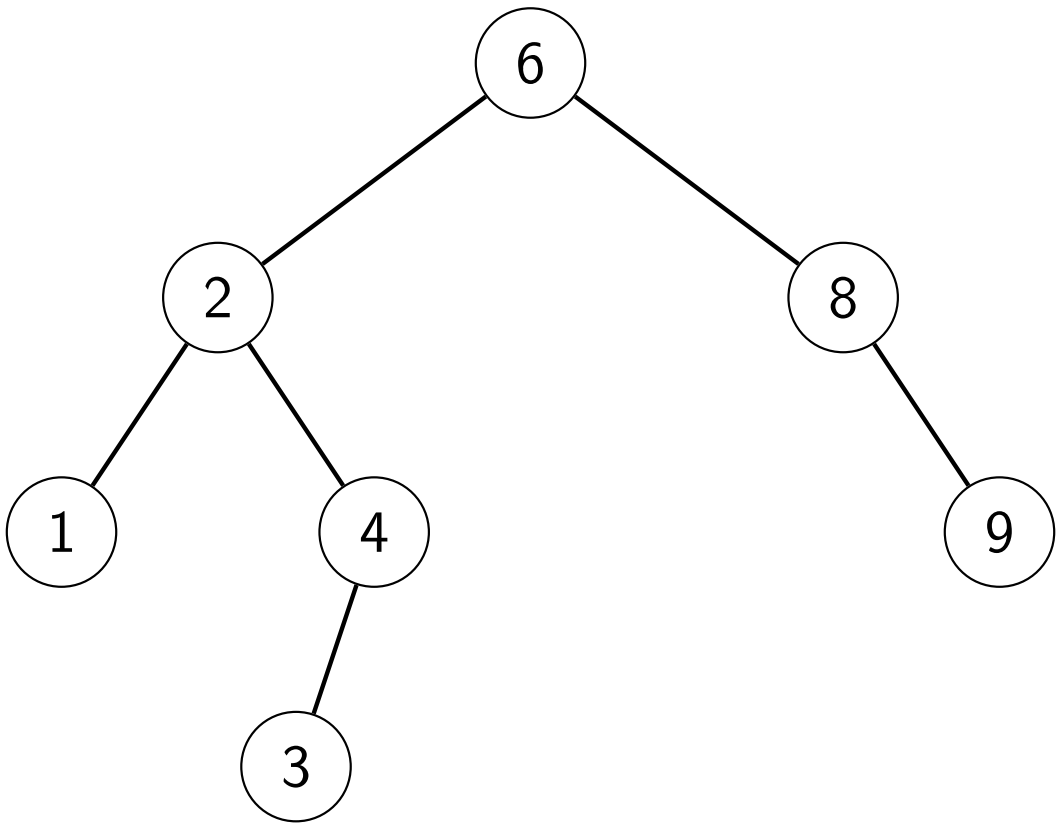
- traverse the tree following strictly left children
- continue as long as left children exist
- the last node, that has a null left child, is the minimum node of the collection

findmax()

- the mirror image of findmin()
- traverse all right child links until the last node

insert(key, data)

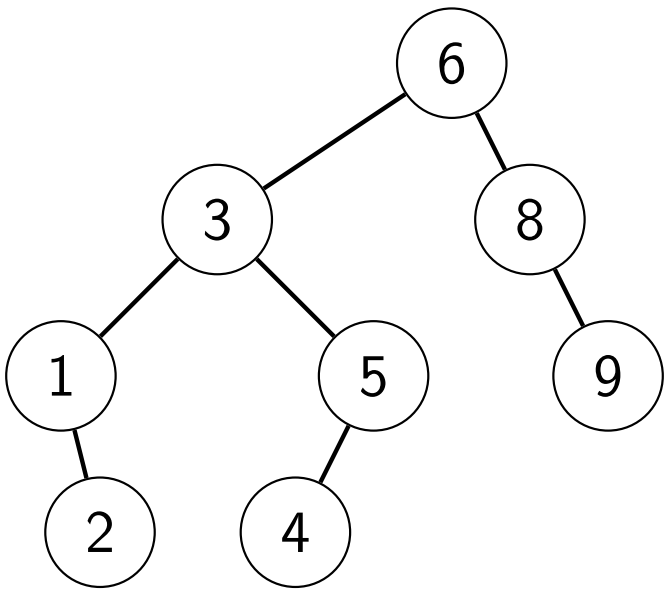
- almost as simple as find
- start out as in find, until an empty child is found
- create a new node at that spot, and insert the data



remove(key)

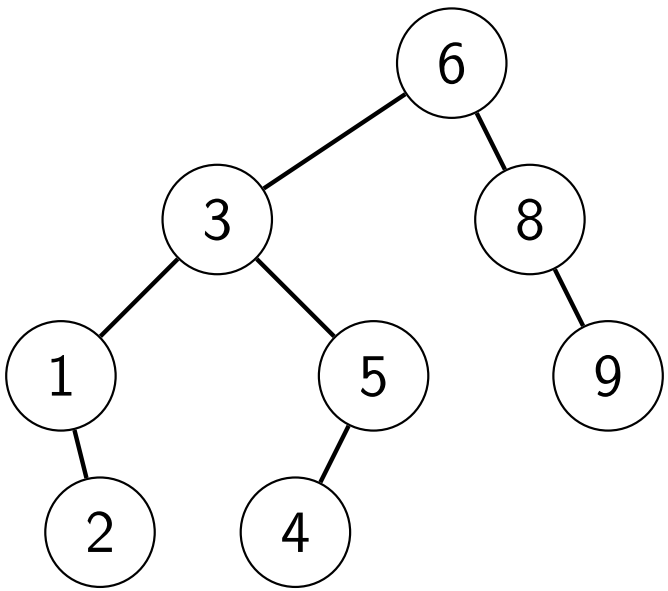
- the hardest operation
- proceed with find until the node to be deleted is found
- now there are three cases

Case 1: if the node to be deleted is a leaf (both children nullptrs), just delete the node. Example: remove(9) or remove(4)



Remove With One Child

Case 2: if the node to be deleted has a single child, delete the node and promote its child to the position the deleted node occupied.
Examples: remove(1), remove(5), remove(8)



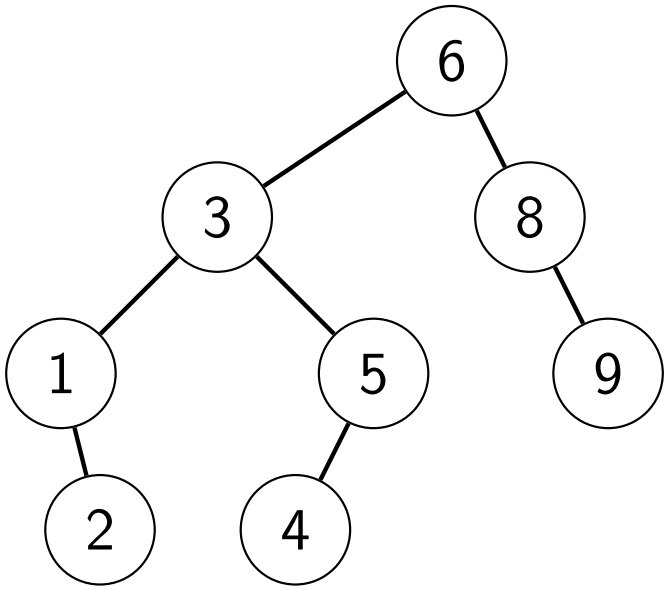
Remove With Two Children

Case 3: if the node to be deleted has two children, we have a most-complicated situation

- two equivalent solutions
  1. replace this node with the smallest child of the right subtree
  2. replace this node with the largest child of the left subtree
- then recursively remove that smallest or largest child
- that child by definition will have zero or one child

Remove With Two Children

Example: remove(3)



- for a valid BST, the spot now occupied by 3 could be replaced with either
  - 2
  - 4
- 2 is the largest leaf of 3's left subtree
- 4 is the smallest leaf of 3's right subtree

BST Analysis

- the find algorithm is integral to all the other operations
- once find completes, the final actions are mostly a constant number of operation
- find's best case is always $\Omega(1)$
- the analysis of find's worst case depends on the structure of the tree
- the best worst case is when the tree is complete
  - in that case, find takes $O(\lg n)$ operations, just like binary search
- the worst worst case is when the tree is all left (or right) children
  - in that case, find takes $O(n)$ operations; linear search

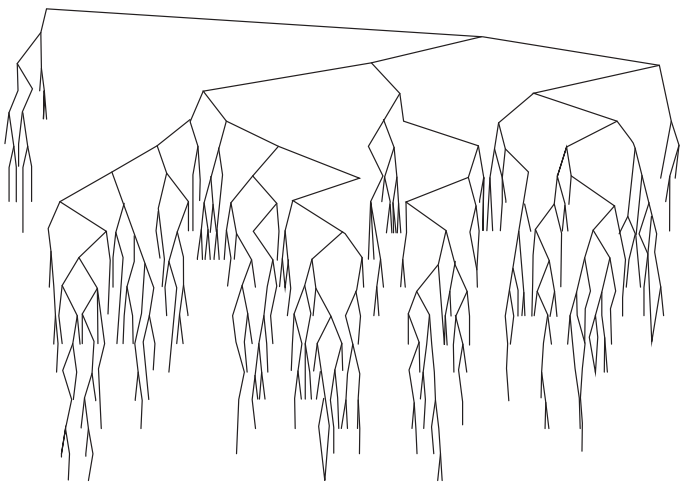
Balancing

- a balanced binary tree is one whose height $h \approx \lg n$
- more precisely, a balanced BST is one each of whose nodes has approximately equal numbers of children
- how likely is a balanced binary tree?

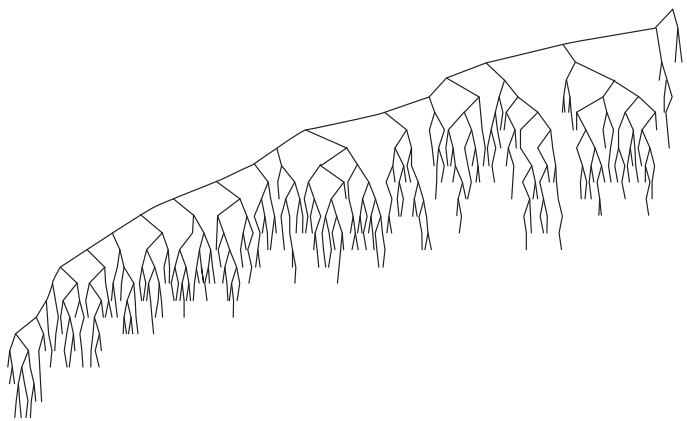
- repeated inserts in ascending order
- repeated inserts in descending order
- repeated inserts in random order; from random.org:
  11 13 2 6 5 20 17 9 14 12 7 19 8 10 16 3

Delete Strategy

- imagine starting with a balanced BST, i.e, a complete BST
- now perform n random inserts and n random removes
- the removes always use the case-3 strategy of replacing the deleted node with the smallest leaf of the right subtree



500 Random Inserts



After 250K Random Insert & Remove