Binary search trees

Comp Sci 1575 Data Structures

Missouri S&T | Computer Science

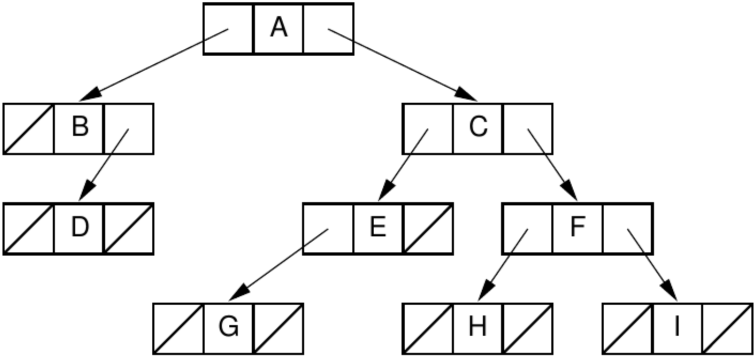
Trees

Q: What is the most common tree in silicon valley?
A: Binary search trees!

Outline

Outline

Pointer based binary trees



What does each node store?

Pointer based binary trees

Do all nodes need to store the same data?

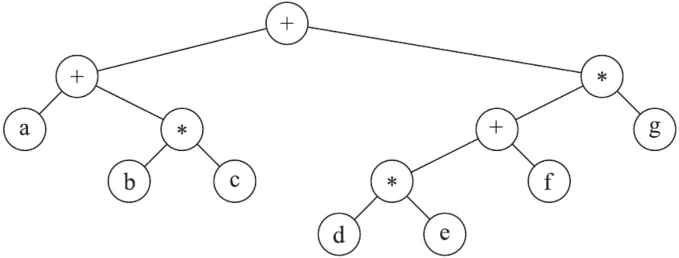


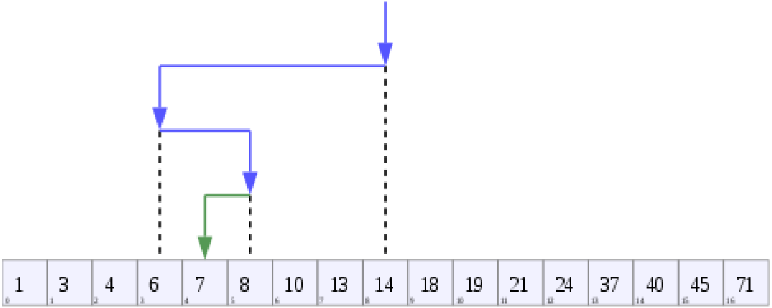
Figure 4.14 Expression tree for `(a + b * c) + ((d * e + f ) * g)`

Check out examples.

Outline

Review: binary search was efficient

Find 7:



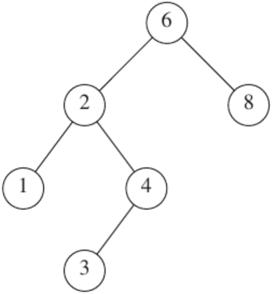
Reminder: cost of sort versus search

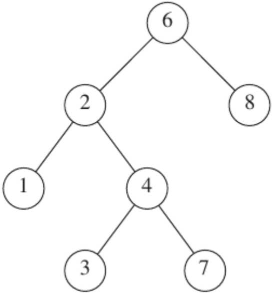
What is the complexity of the operation of inserting in sorted order, like our sorted array list (covered before)?

Binary search trees (BST)

- Binary search trees keep their keys in sorted order.
- BST is a tree whose internal nodes each store a comparable or key (and optionally an associated value), and each key must be greater than any key stored in its left sub-tree, and less than (or sometimes equal to) any key stored in its right sub-tree.
- For every node, X, in the tree, the values of all the keys in its left subtree are smaller than the key in X, and the values of all the keys in its right subtree are larger than the key in X.
- Fast lookup, addition and removal of keys ($\log_2 n$)
- On average, each lookup, insertion or deletion takes time proportional to the logarithm of the number of items stored in the tree. This is the key advantage over a continuously sorted list.

Binary search trees (BST)





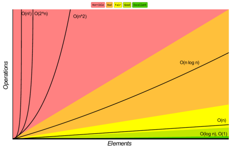
Left is a binary search tree, right is just a binary tree

Outline

Data structures

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

Color key:



Outline

Binary node abstract class: BinNode.h

// Return the node's value
// Set the node's value
// Return the node's left child
// Set the node's left child
// Return the node's right child
// Set the node's right child
// Return true if the node is a leaf, false otherwise

Outline

BST node class: BSTNode.h

key,
value,
pointer to left child,
pointer to right child

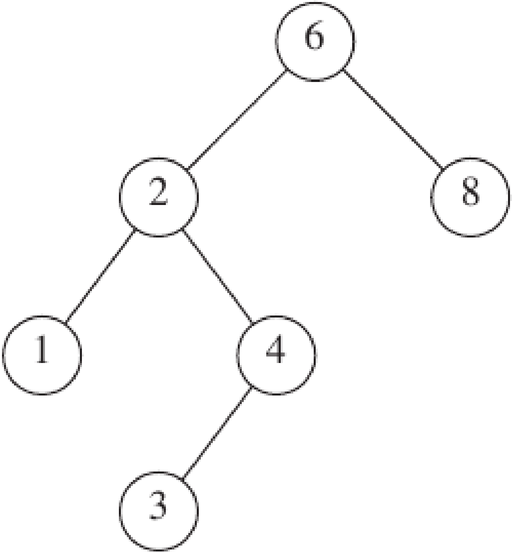
Outline

BST tree class: BST.h

We will walk through each function

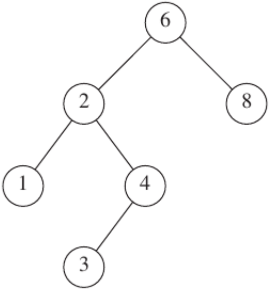
Outline

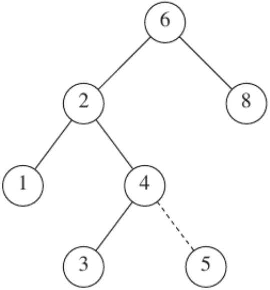
Search



What are steps for search?

Insert

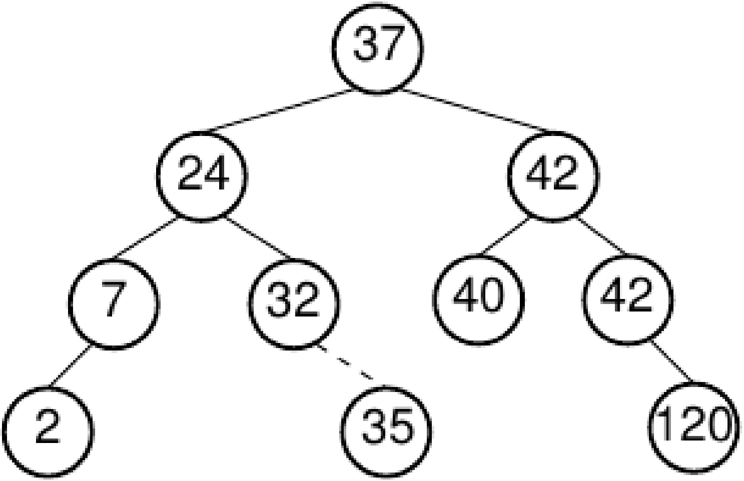




What are steps?

Delete with 0 children?

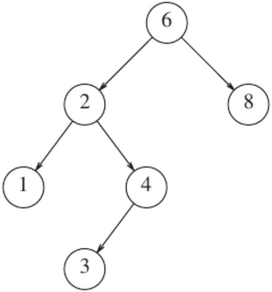
Delete 35

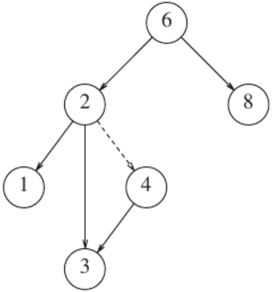


What are steps?

Delete with one child

Delete 4

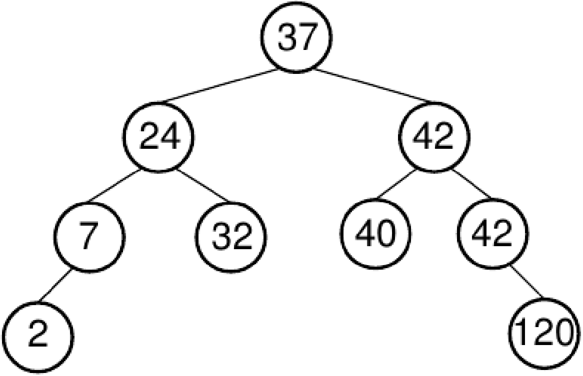




What are steps?

Delete with two children

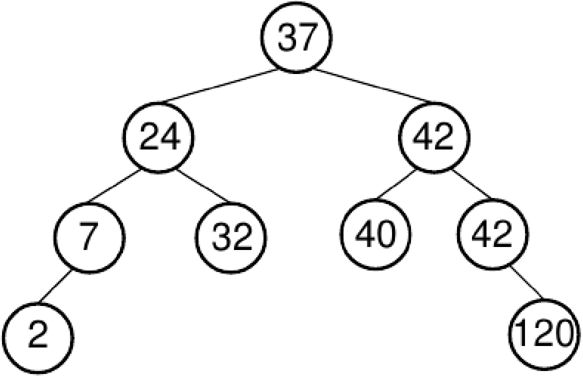
Delete 42, or 37, or 24?



What to do with the two remaining subtrees?

Delete with two children

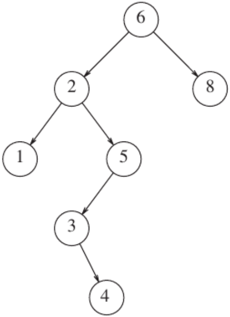
Delete 42, or 37, or 24?

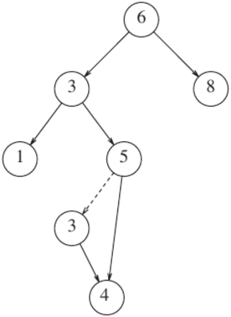


Inefficient option: Set R's parent to point to one of R's subtrees, and then reinsert the remaining subtree's nodes one at a time.

Delete with two children

Delete the node labeled 2

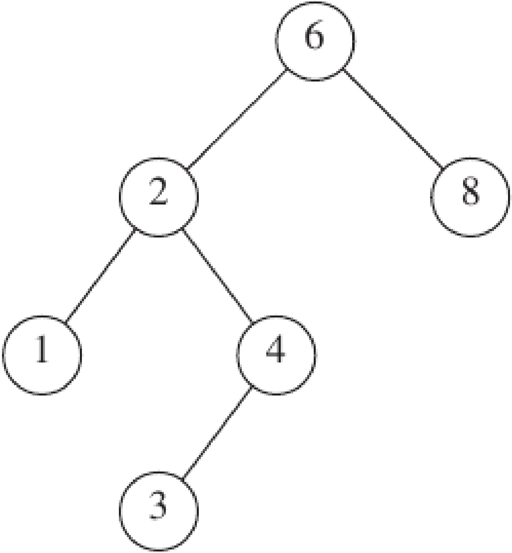




- Find value in one of the subtrees that can replace the value in R.
- Least key value greater than (or equal to) the one being removed, or else the greatest key value less than the one being removed.
- Min of Max and Max of Min can be promoted to root

Clear or delete all?

For a recursive traversal that deletes all nodes, what order should the traversal be?



Outline

Insert order

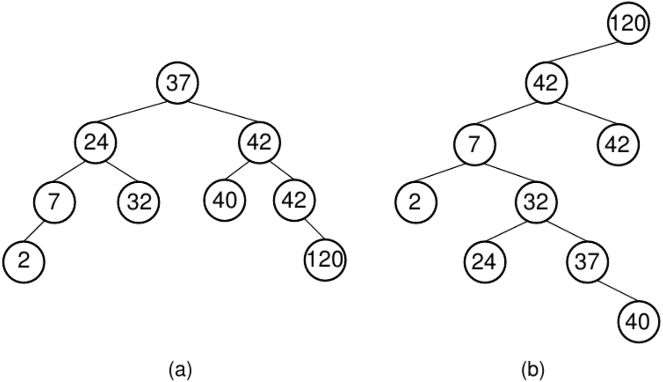
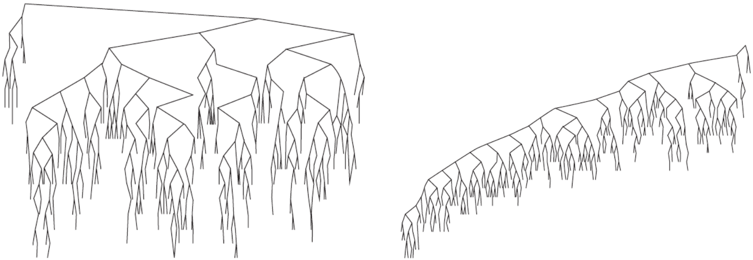


Figure 5.13 Two Binary Search Trees for a collection of values. Tree (a) results if values are inserted in the order 37, 24, 42, 7, 2, 40, 42, 32, 120. Tree (b) results if the same values are inserted in the order 120, 42, 42, 7, 2, 32, 37, 24, 40.

Insert order

Random tree before (left) and after (right) 250,000 insert and remove operations



Advanced variations maintain balance during insert and remove (like std::map and std::set)

Drawing