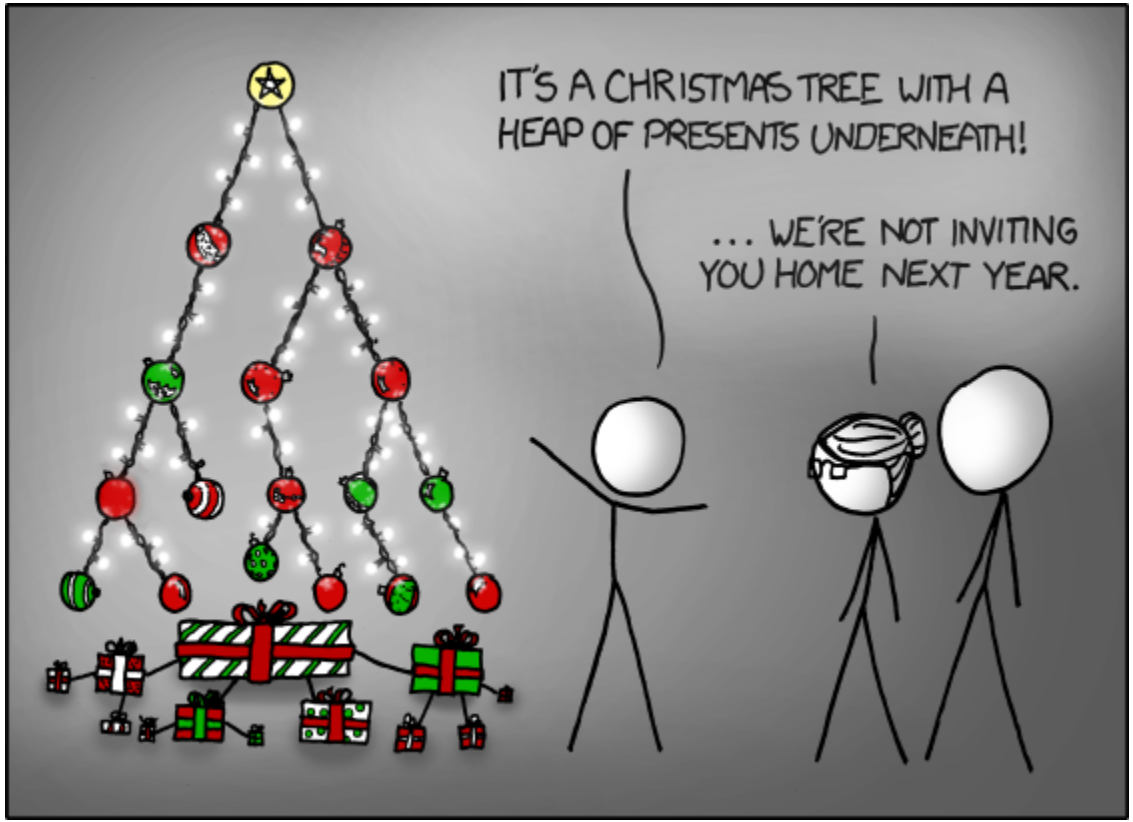
Binary Trees

Chapter 19

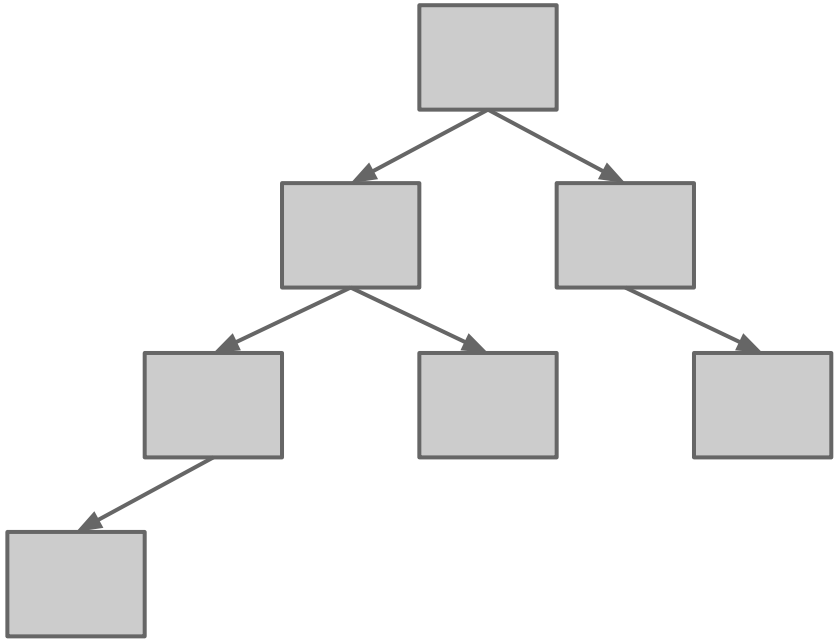


http://xkcd.com/835/

Binary Trees

- A binary tree is a non linear data structure where each node may have, or point at, 0, 1, or 2 other nodes

Binary Trees

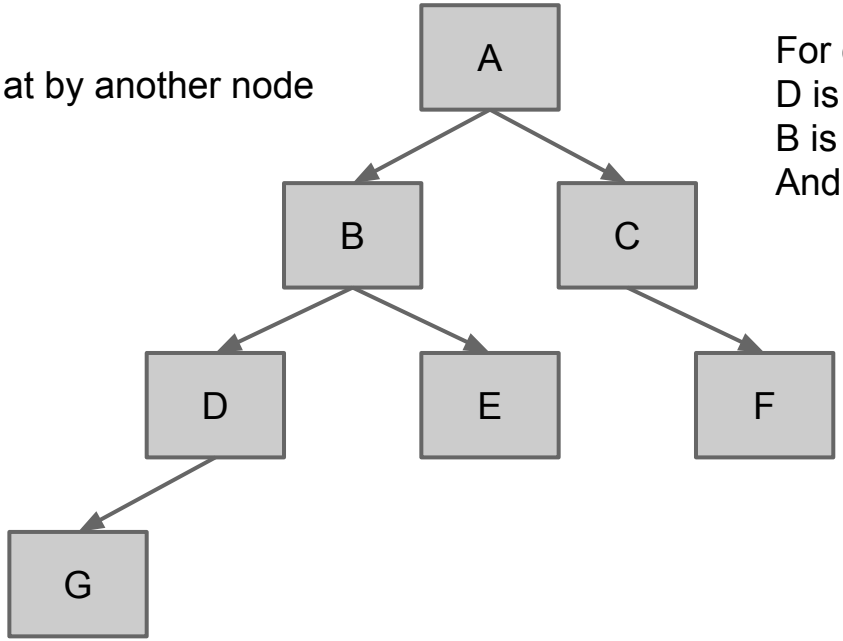


Binary Trees

- Root
  - A node with no parent node
- Parent
  - A node with 0 or more child nodes
- Child
  - A node with a parent
- leaves
  - A node with 0 children

Binary Trees

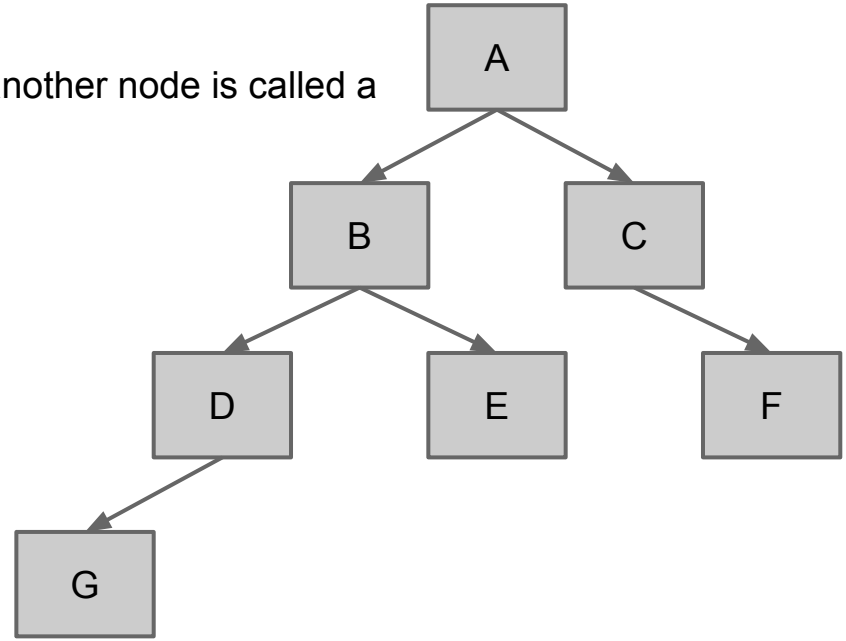
Nodes that are pointed at by another node are called children



For example:
D is a child of B
B is a child of A
And A is a child of no one

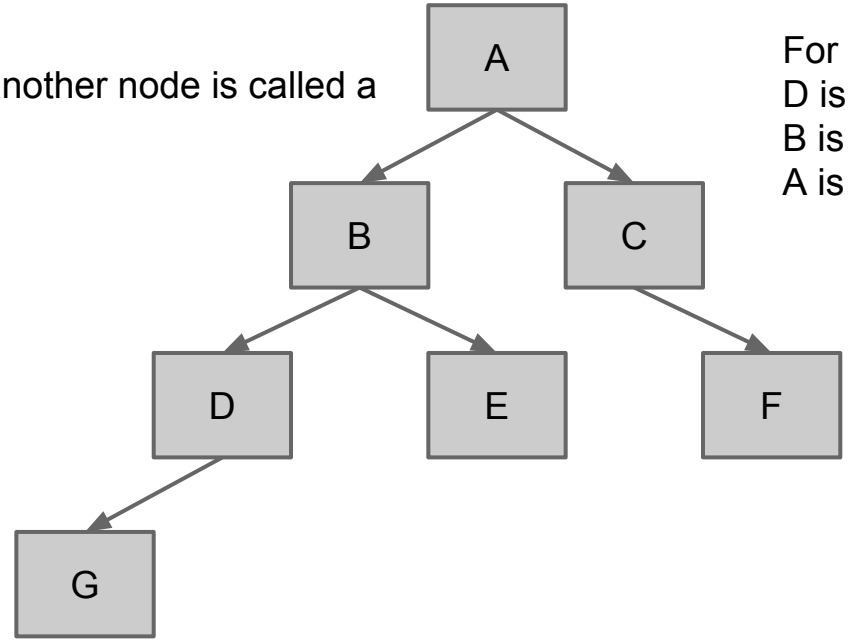
Binary Trees

A Node that points at another node is called a parent



Binary Trees

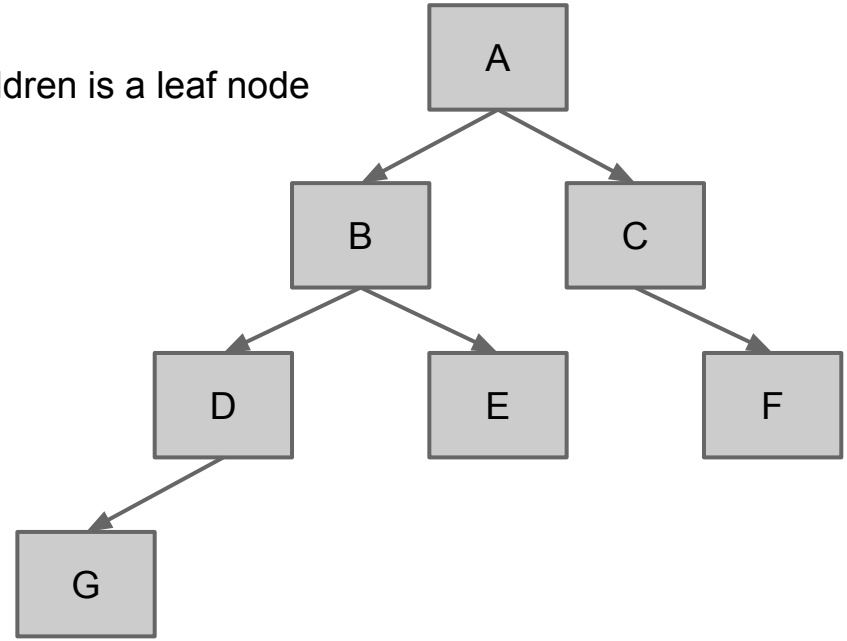
A Node that points at another node is called a parent



For example:
D is the parent of G
B is the parent of D
A is the parent of B

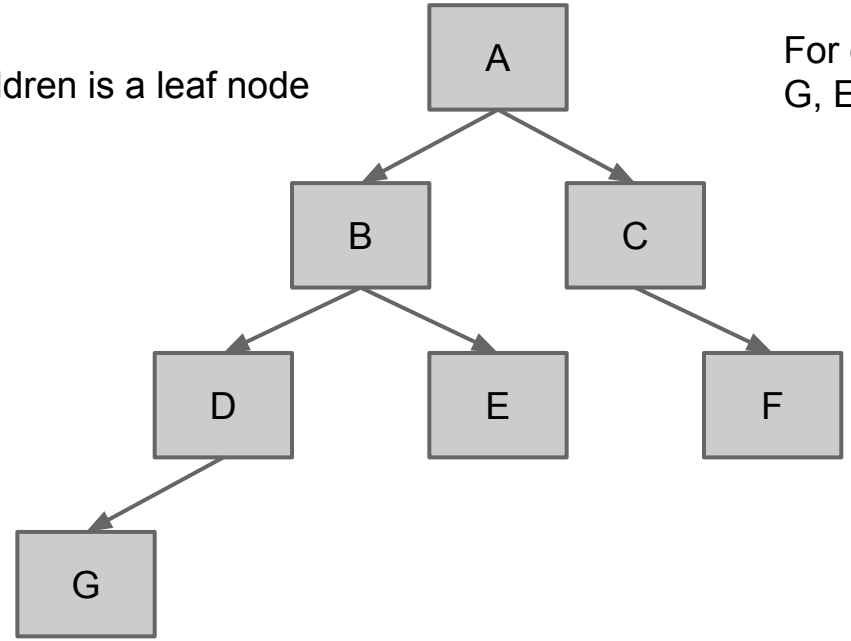
Binary Trees

A node that has no children is a leaf node



Binary Trees

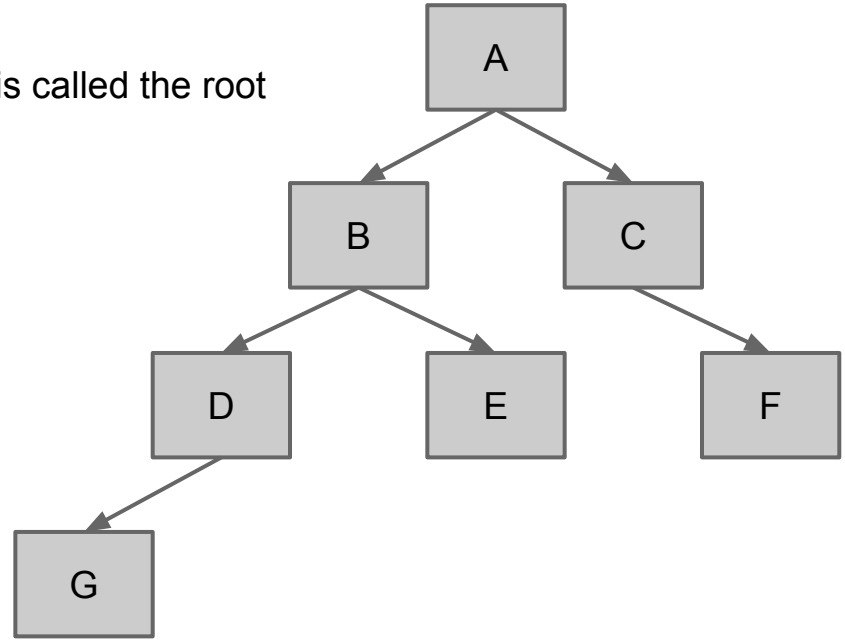
A node that has no children is a leaf node



For example:
G, E, and F are all leaf nodes

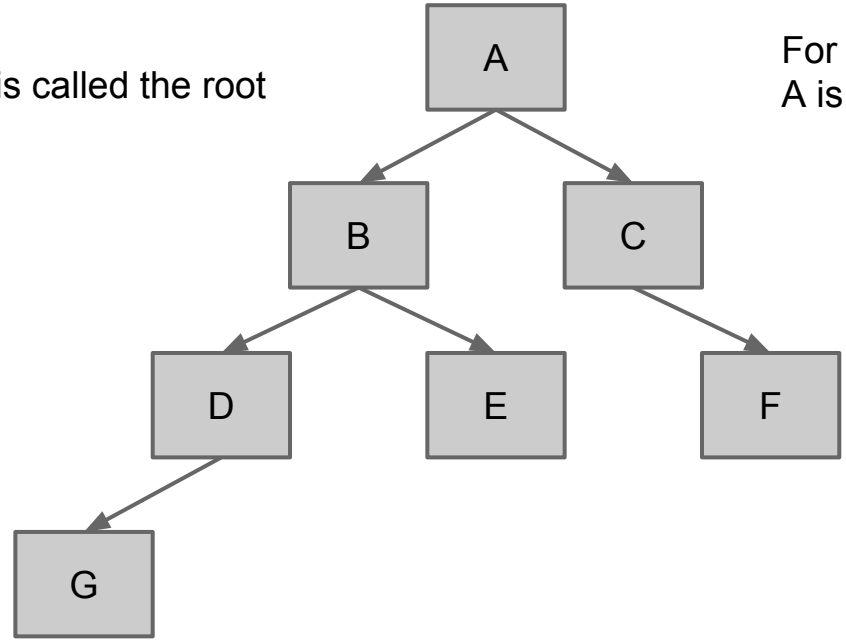
Binary Trees

A node with no parent is called the root



Binary Trees

A node with no parent is called the root



For example:
A is the only root

Binary Trees

- A binary tree is called a tree because it resembles an upside down tree

Binary Trees

- A binary tree is called a tree because it resembles an upside down tree

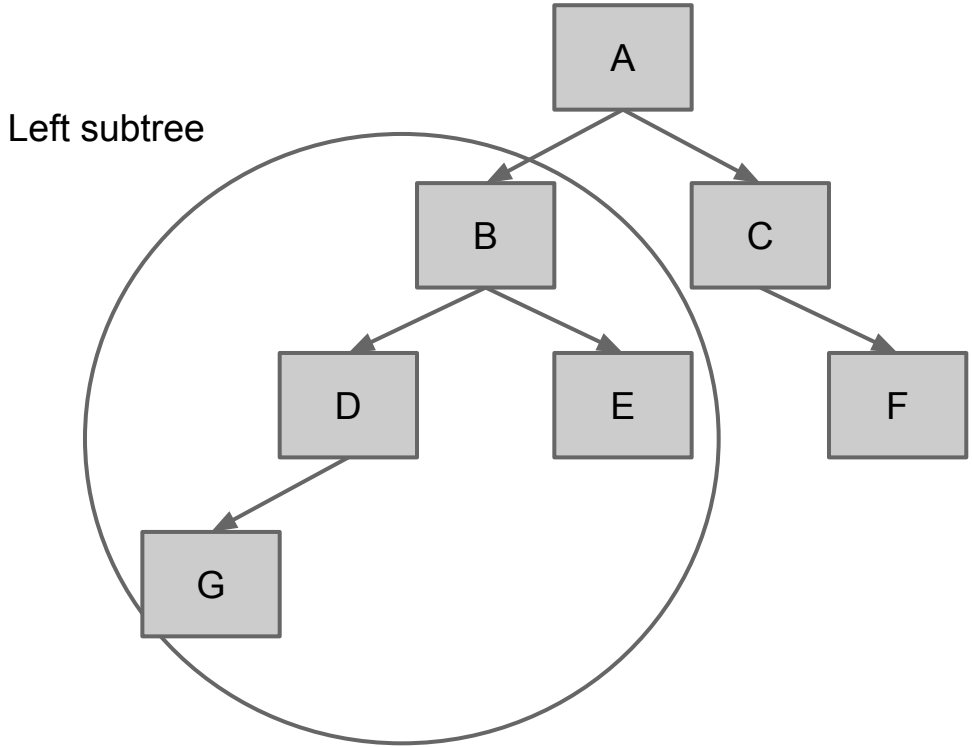
Binary Trees

- A tree can also be partitioned into a root node, a left subtree, and a right subtree

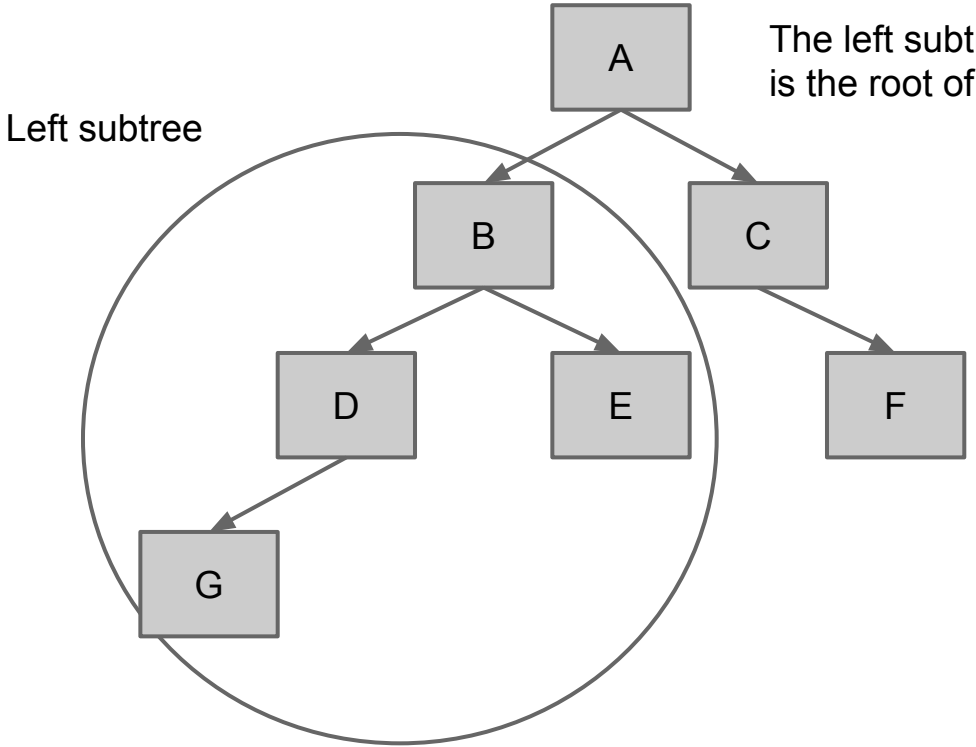
Binary Trees

- A tree can also be partitioned into a root node, a left subtree, and a right subtree
  - This can be used recursively

Binary Trees



Binary Trees

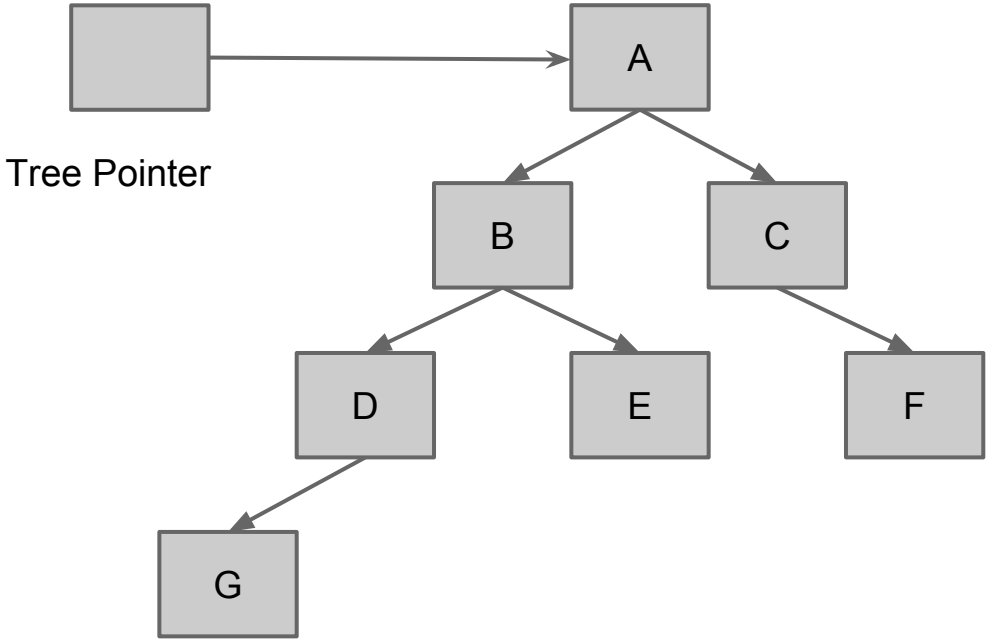


The left subtree is said to be rooted at B and B is the root of the subtree

Binary Trees

- A tree pointer is a pointer to the root node of a tree

Binary Trees



Binary Search Trees

- A Binary Search Tree is a Binary Tree that is setup to simplify searching

Binary Search Trees

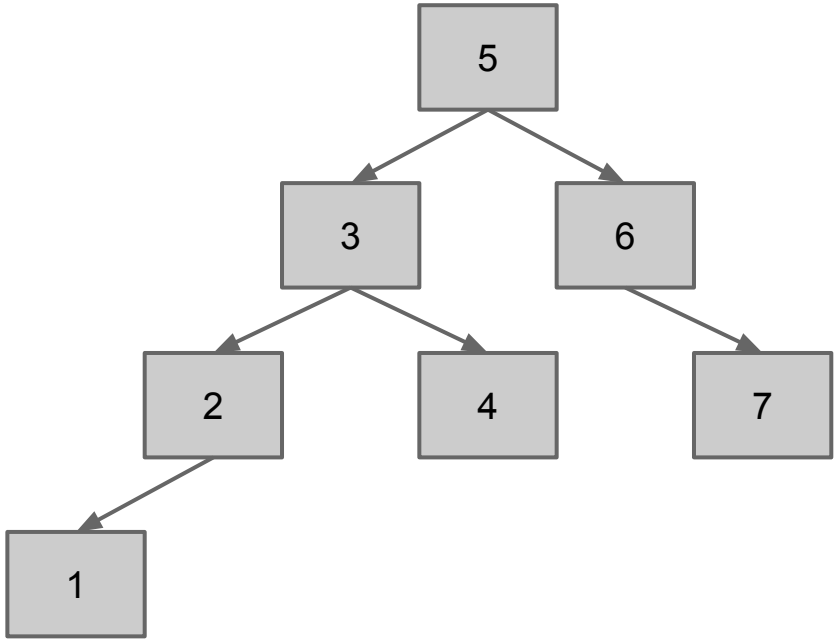
- A Binary Search Tree is a Binary Tree that is setup to simplify searching
  - Left subtree at each node contains data values less than the data in the node

Binary Search Trees

- A Binary Search Tree is a Binary Tree that is setup to simplify searching
  - Left subtree at each node contains data values less than the data in the node
  - Right subtree at each node contains values greater than the data in the node

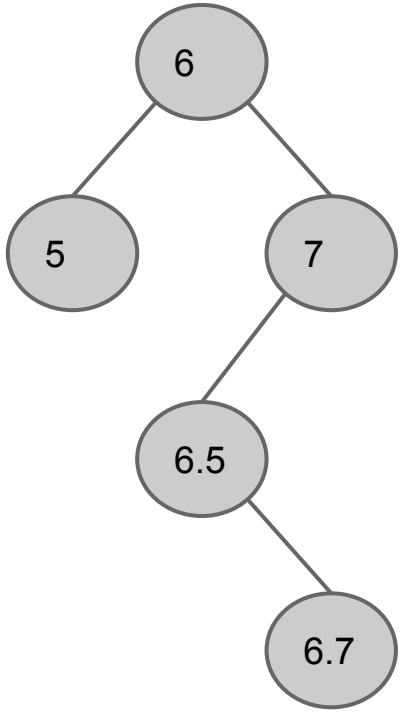
Binary Search Trees

Example



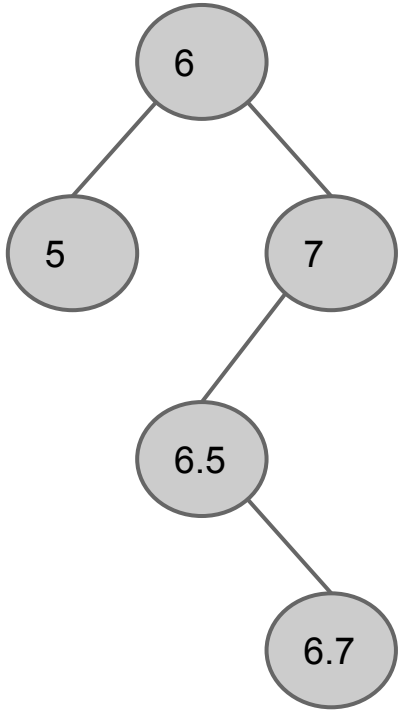
Binary Search Tree

- Is this a BST?



Binary Search Tree

- Is this a BST?



Yes.
What can we learn from looking at the structure?

Binary Search Trees

- The left side is always the minimum

- The right side is always the maximum

Binary Search Trees - Traversal

- There are three main ways of traversing a Tree
  - Inorder Traversal
    - Traverse left subtree of node
    - Process data in node
    - Traverse right subtree of node

Binary Search Trees - Traversal

- There are three main ways of traversing a Tree
  - Inorder Traversal
  - Preorder Traversal
    - Process data in node
    - Traverse left subtree of node
    - Traverse right subtree of node

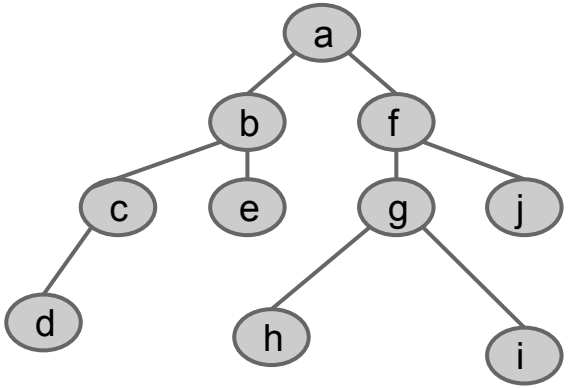
Binary Search Trees - Traversal

- There are three main ways of traversing a Tree
  - Inorder Traversal
  - Preorder Traversal
  - Postorder Traversal
    - Traverse left subtree of node
    - Traverse right subtree of node
    - Process data in node

Tree Traversal

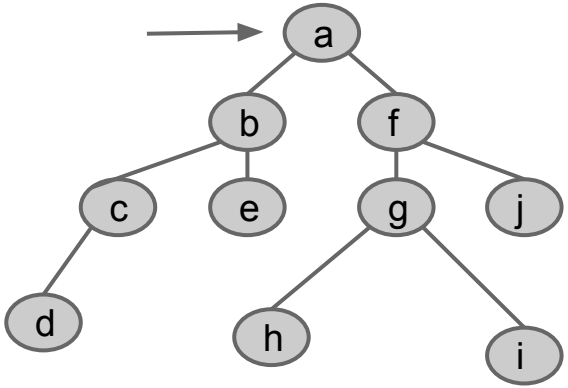
- Inorder Traversal
  - Traverse left subtree
  - Process data in node
  - Traverse right subtree

Tree Traversal: Inorder



List

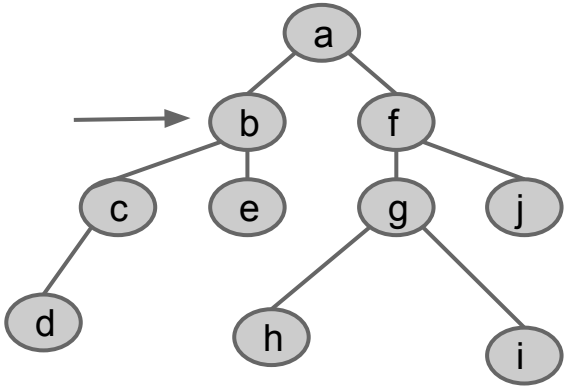
Tree Traversal: Inorder



List

_____

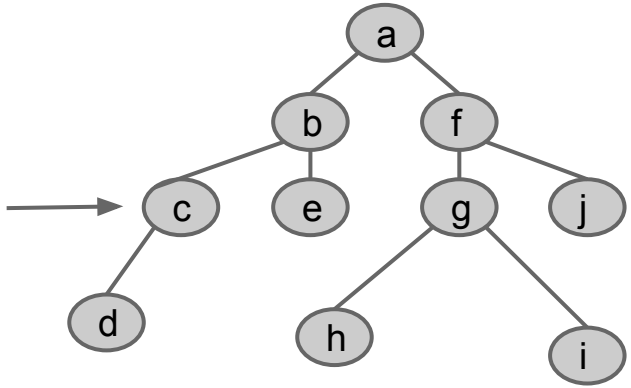
Tree Traversal: Inorder



List

_____

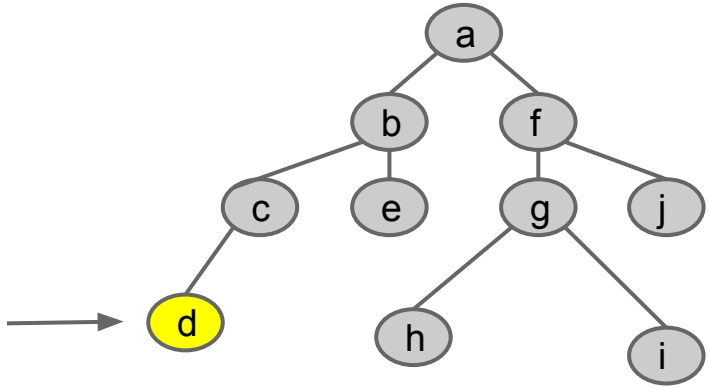
Tree Traversal: Inorder



List

_____

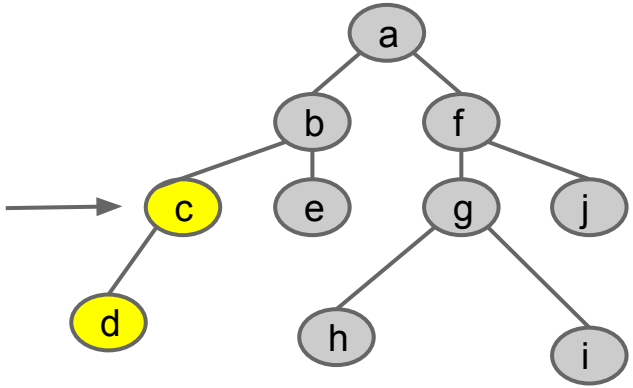
Tree Traversal: Inorder



List

d

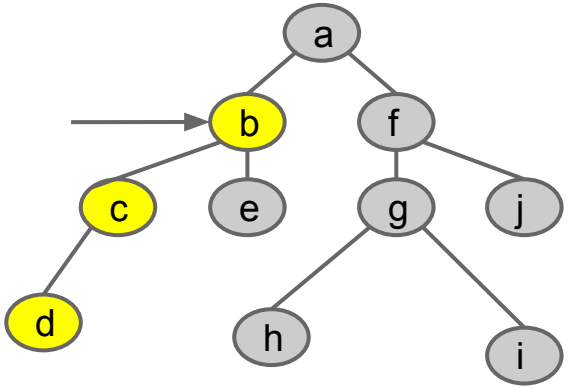
Tree Traversal: Inorder



List

d
c

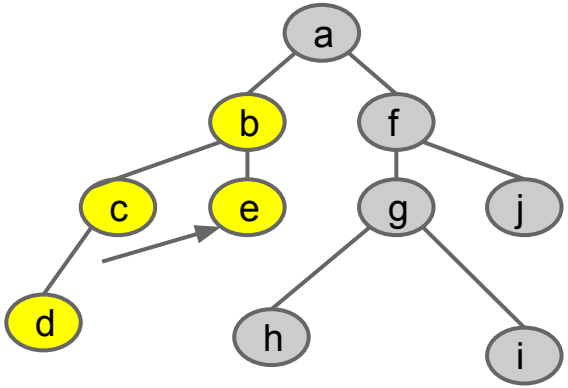
Tree Traversal: Inorder



List

d
c
b

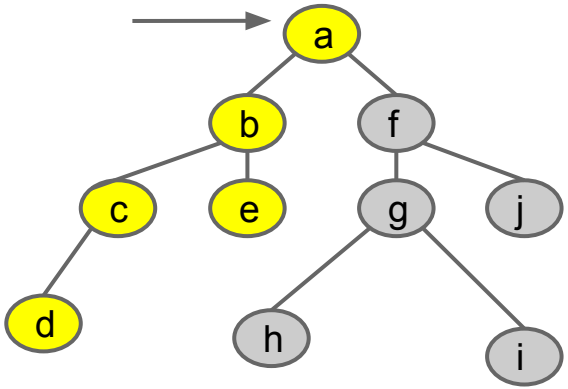
Tree Traversal: Inorder



List

d
c
b
e

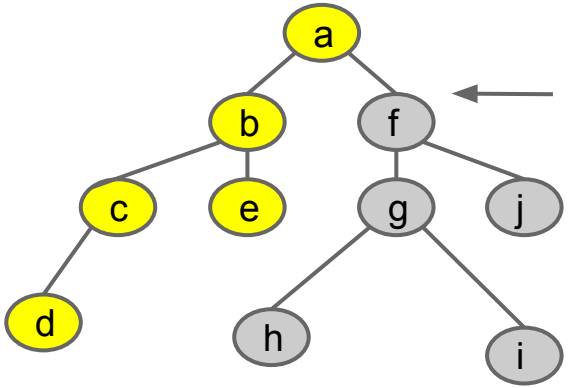
Tree Traversal: Inorder



List

d
c
b
e
a

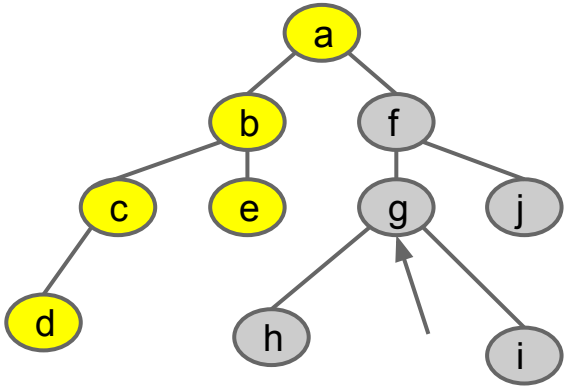
Tree Traversal: Inorder



List

d
c
b
e
a

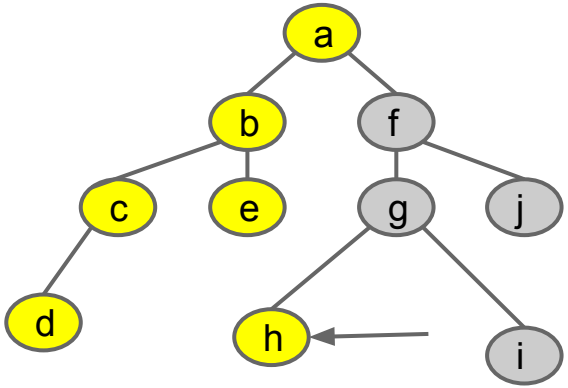
Tree Traversal: Inorder



List

d
c
b
e
a

Tree Traversal: Inorder

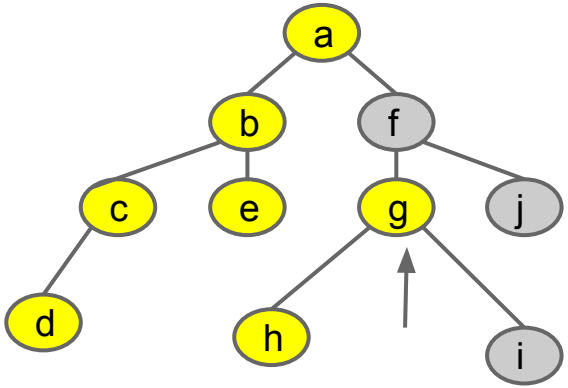


List

_____

d
c
b
e
a
h

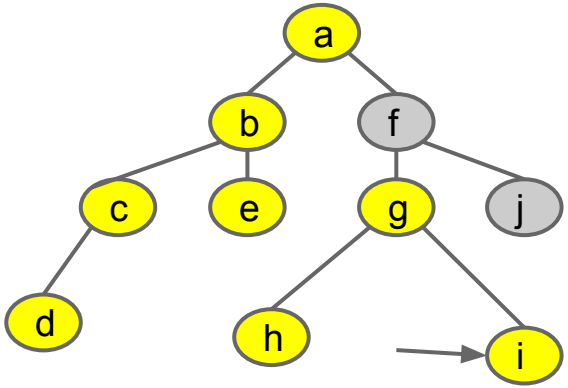
Tree Traversal: Inorder



List

d
c
b
e
a
h
g

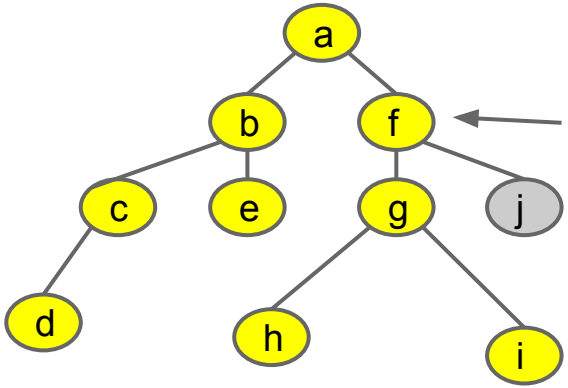
Tree Traversal: Inorder



List

d
c
b
e
a
h
g
i

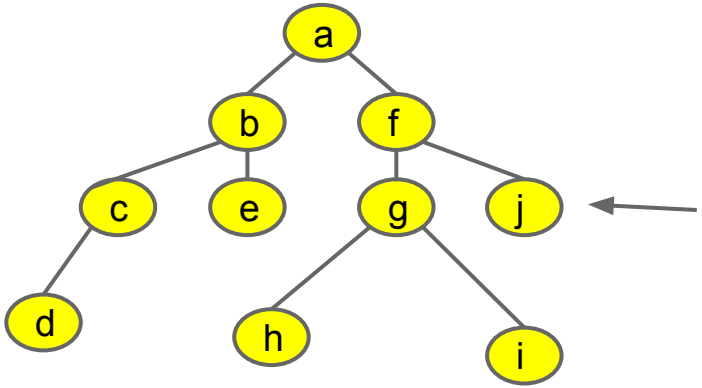
Tree Traversal: Inorder



List

d
c
b
e
a
h
g
i
f

Tree Traversal: Inorder



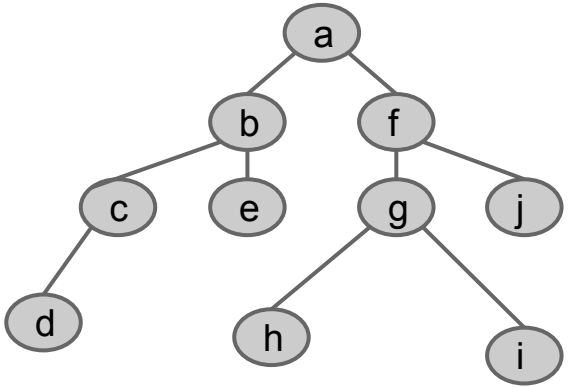
List

d
c
b
e
a
h
g
i
f
j

Tree Traversal

- Preorder Traversal
  - Process data in node
  - Traverse left subtree
  - Traverse right subtree

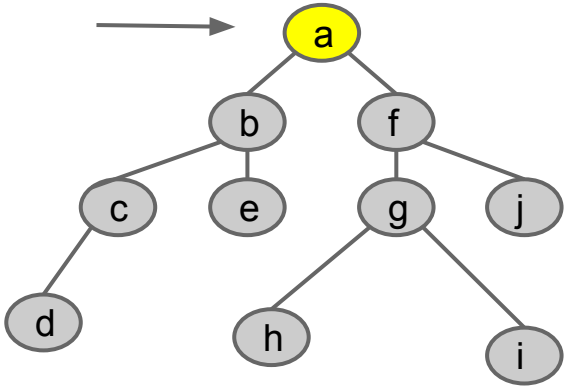
Tree Traversal: Preorder



List

_____

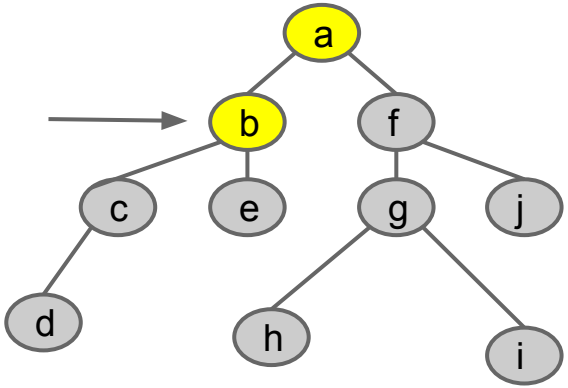
Tree Traversal



List

a

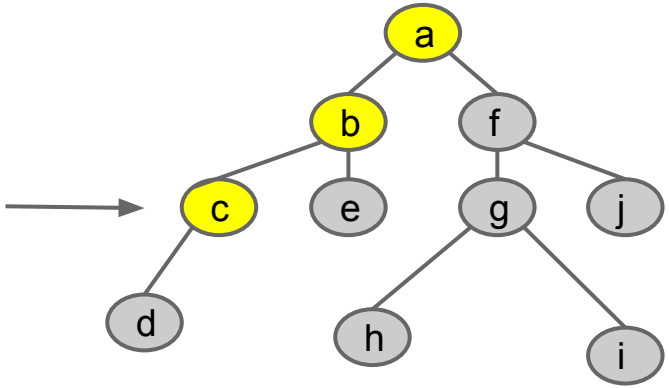
Tree Traversal



List

a
b

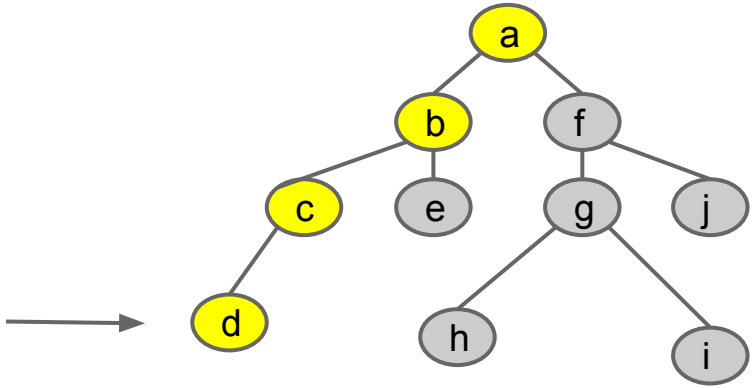
Tree Traversal



List

a
b
c

Tree Traversal

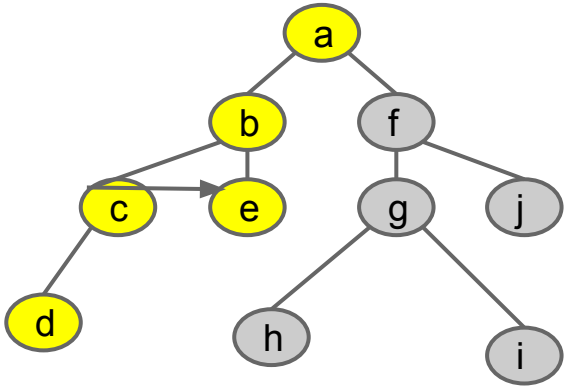


List

_____

a
b
c
d

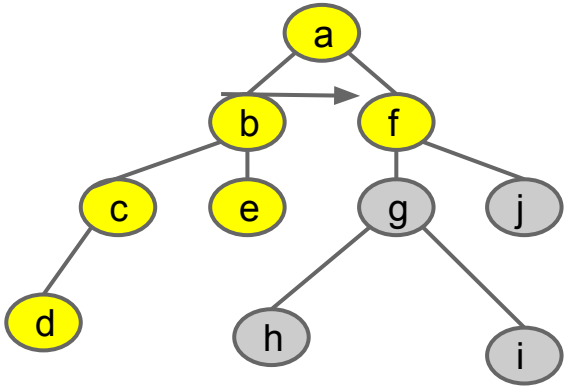
Tree Traversal



List

a
b
c
d
e

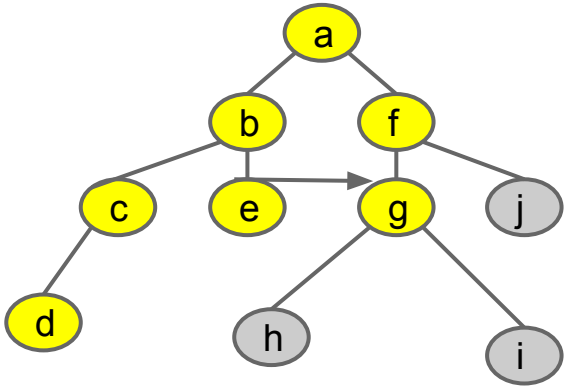
Tree Traversal



List

a
b
c
d
e
f

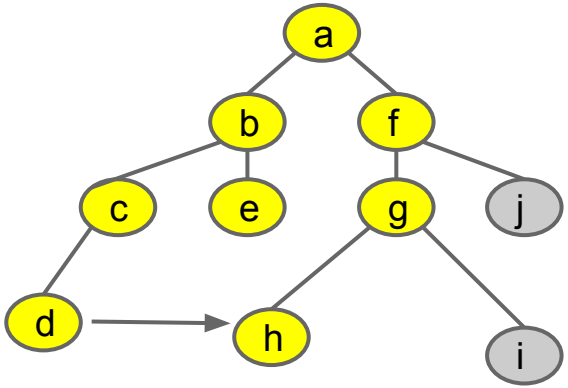
Tree Traversal



List

a
b
c
d
e
f
g

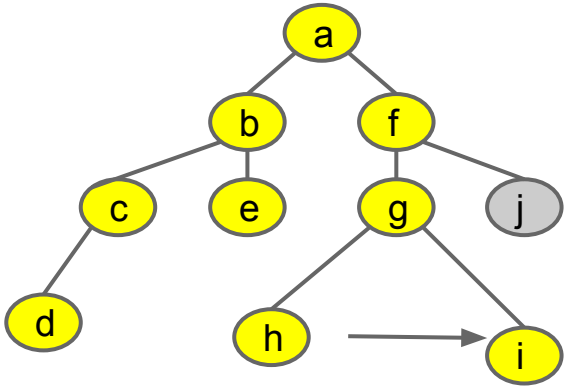
Tree Traversal



List

a
b
c
d
e
f
g
h

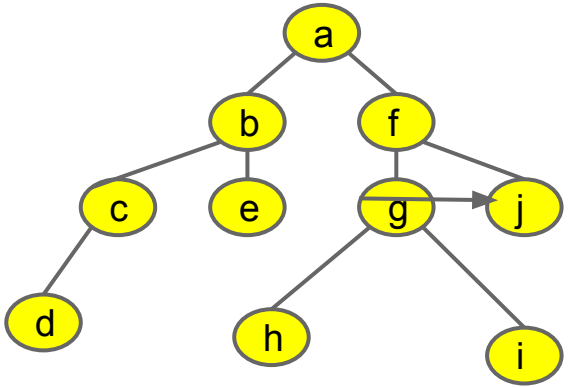
Tree Traversal



List

a
b
c
d
e
f
g
h
i

Tree Traversal



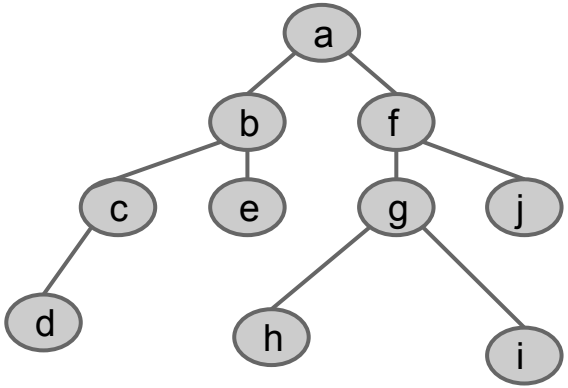
List

a
b
c
d
e
f
g
h
i
j

Tree Traversal

- Postorder traversal
  - Traverse left subtree
  - Traverse right subtree
  - Process data

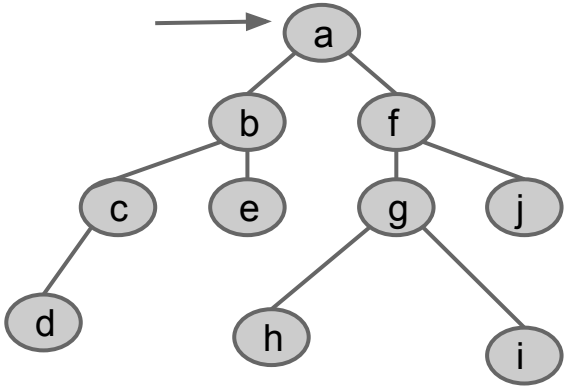
Tree Traversal: Postorder



List

_____

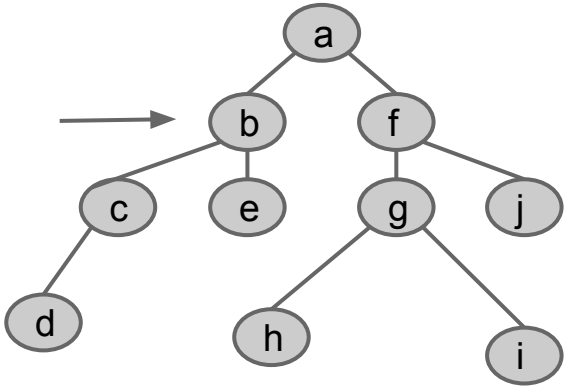
Tree Traversal



List

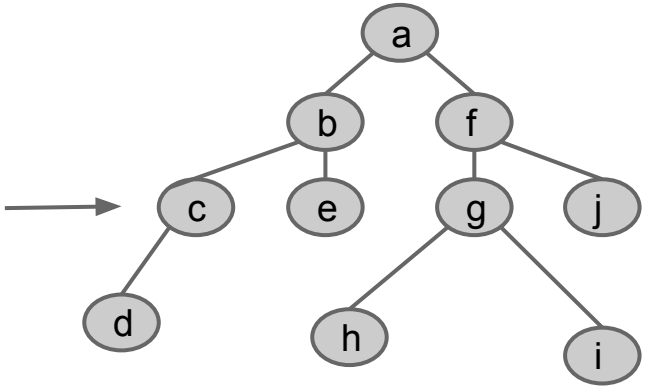
_____

Tree Traversal



List

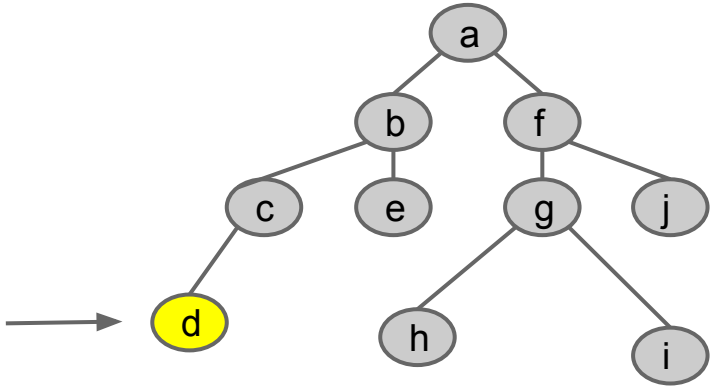
Tree Traversal



List

_____

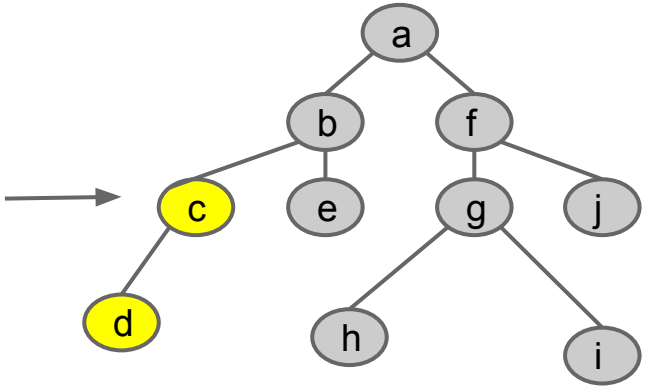
Tree Traversal



List

d

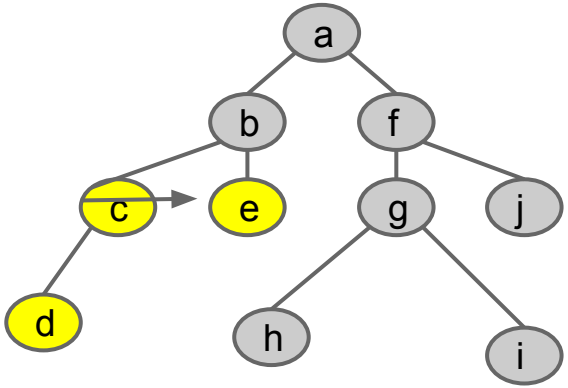
Tree Traversal



List

d
c

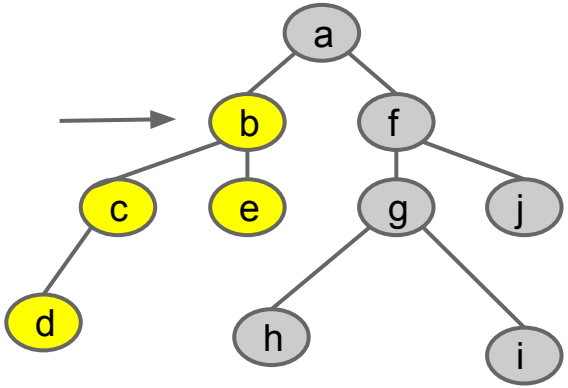
Tree Traversal



List

d
c
e

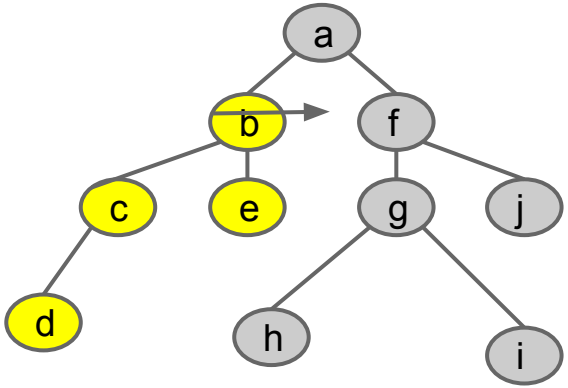
Tree Traversal



List

d
c
e
b

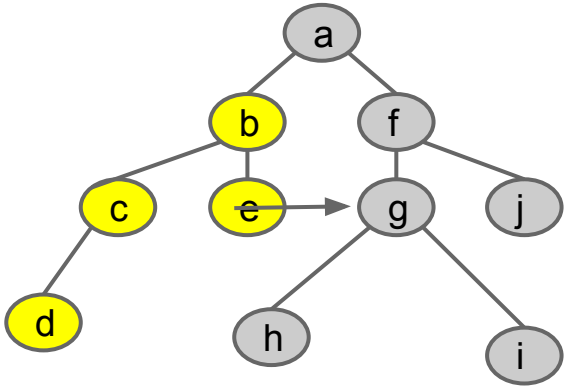
Tree Traversal



List

d
c
e
b

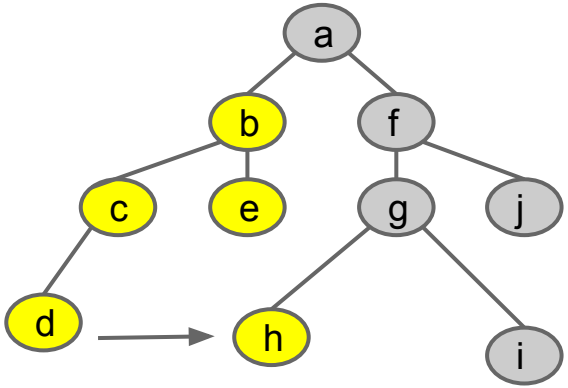
Tree Traversal



List

d
c
e
b

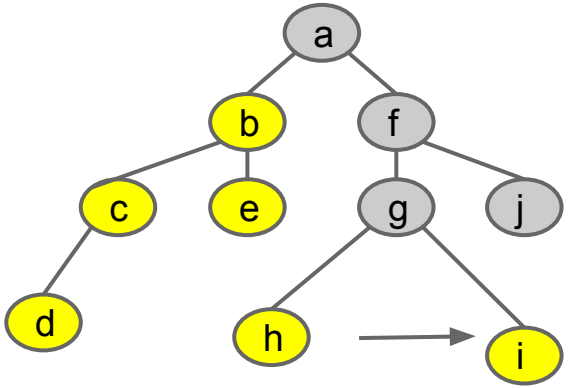
Tree Traversal



List

d
c
e
b
h

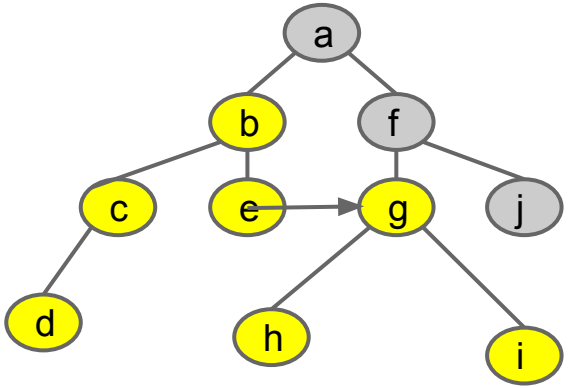
Tree Traversal



List

d
c
e
b
h
i

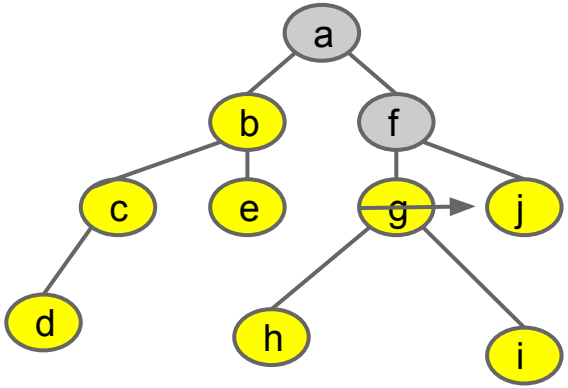
Tree Traversal



List

d
c
e
b
h
i
g

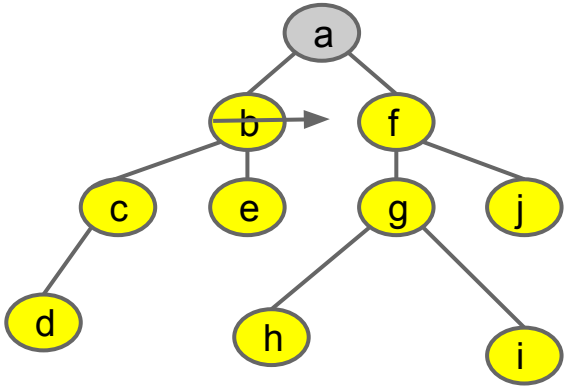
Tree Traversal



List

d
c
e
b
h
i
g
j

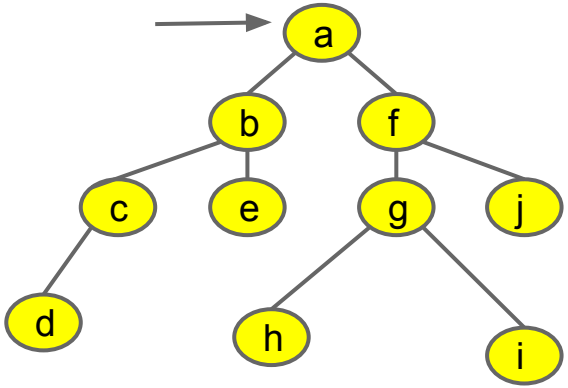
Tree Traversal



List

d
c
e
b
h
i
g
j
f

Tree Traversal



List

d
c
e
b
h
i
g
j
f
a

Binary Search Trees

- Binary Search Tree Operations
  - Create
    - Create the list

Binary Search Trees

- Binary Search Tree Operations
  - Create
    - Create the list
  - Insert
    - insert a node into the correct position and maintain order in the list

Binary Search Trees

- Binary Search Tree Operations
  - Create
    - Create the tree
  - Insert
    - insert a node into the correct position and maintain order in the tree
  - Delete
    - Delete a node from the tree and maintain order in the tree

Binary Search Trees

- Binary Search Tree Operations
  - Insert
    - insert a node into the correct position and maintain order in the tree
  - Delete
    - Delete a node from the tree and maintain order in the tree
  - Find
    - Find a node in the tree

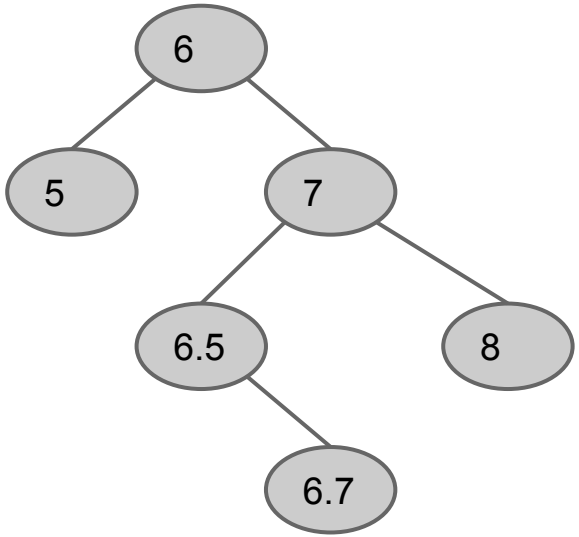
Binary Search Trees - Find

- If the current node is the node you want
  - return
- else if the node you want is greater than the current node
  - recursively find on the right subtree
- else (the node you want is less than the current node)
  - recursively find on the left subtree

Binary Search Trees - Insert

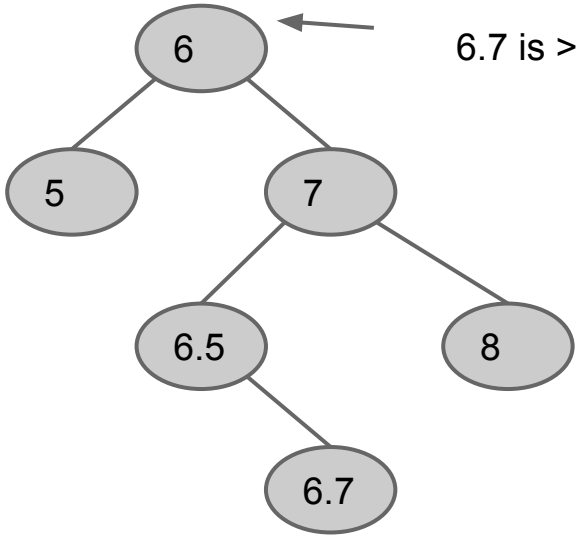
- If the tree is empty
  - place the node as the root with NULL left and right pointers

Binary Search Trees - Find



Find node 6.7

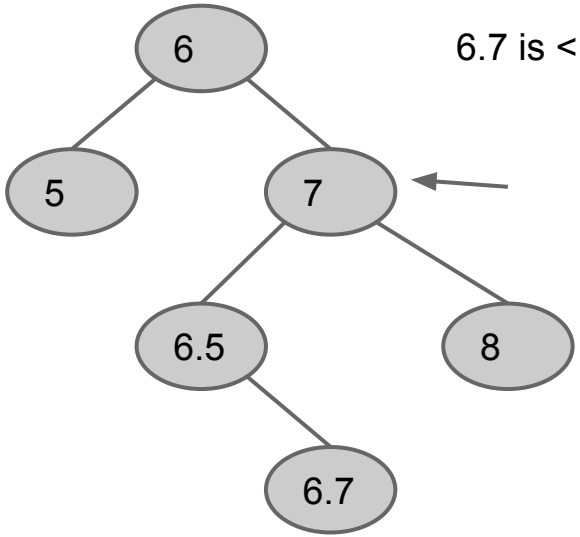
Binary Search Trees - Find



6.7 is > 6, find in right subtree of 6

Find node 6.7

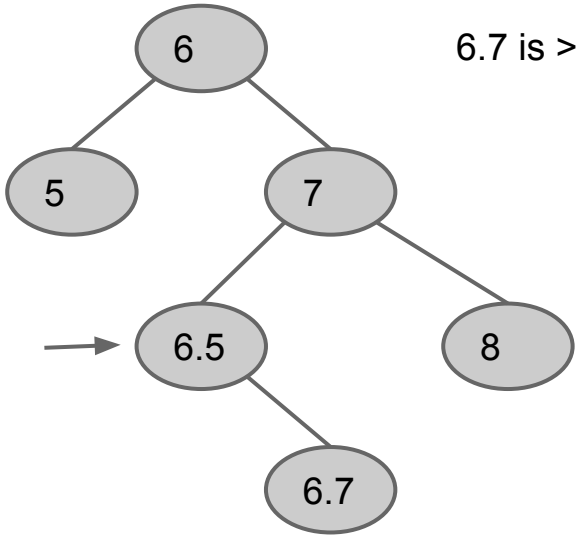
Binary Search Trees - Find



6.7 is < 7, find in left subtree of 7

Find node 6.7

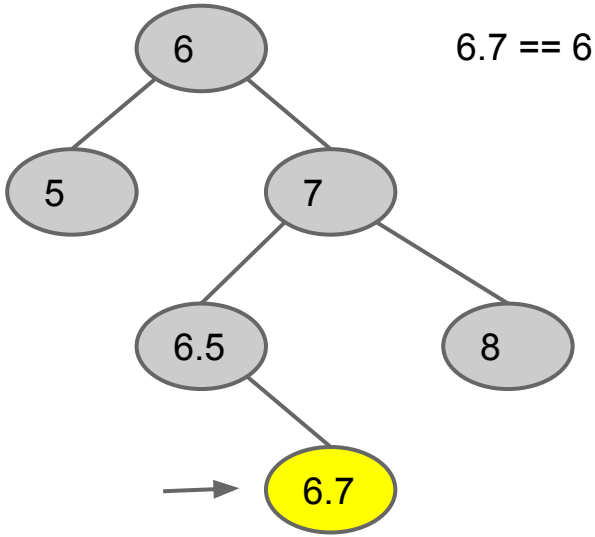
Binary Search Trees - Find



6.7 is > 6.5, find in right subtree of 6.5

Find node 6.7

Binary Search Trees - Find



6.7 == 6.7, return

Find node 6.7

Binary Search Trees - Insert

- How do we find the minimum node in our Binary Search Tree?

Binary Search Trees - Insert

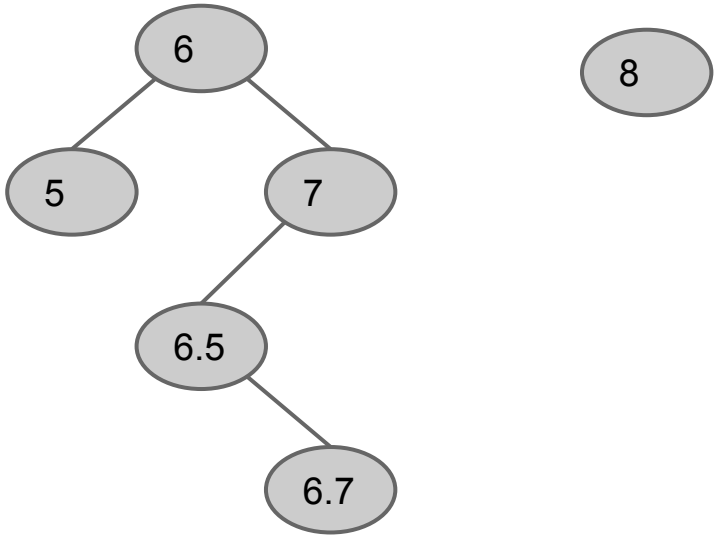
- How do we find the minimum node in our Binary Search Tree?

- How do we find the maximum node in our Binary Search Tree?

Binary Search Trees - Insert

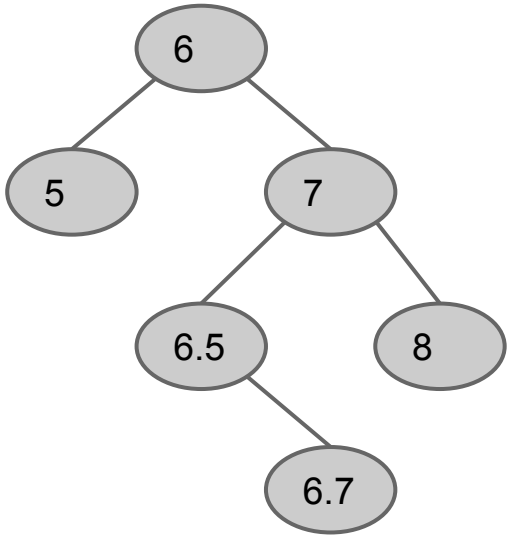
- If the tree is empty
  - place the node as the root with NULL left and right pointers
- else
  - if the item is less than the root, recursively insert left
  - if the item is greater than the root, recursively insert right

Binary Search Trees - Insert

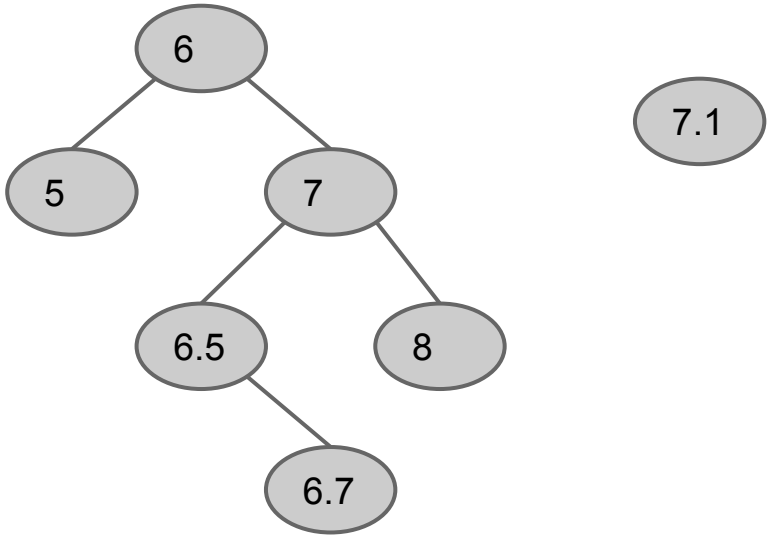


Insert the new node

Binary Search Tree: Insertion

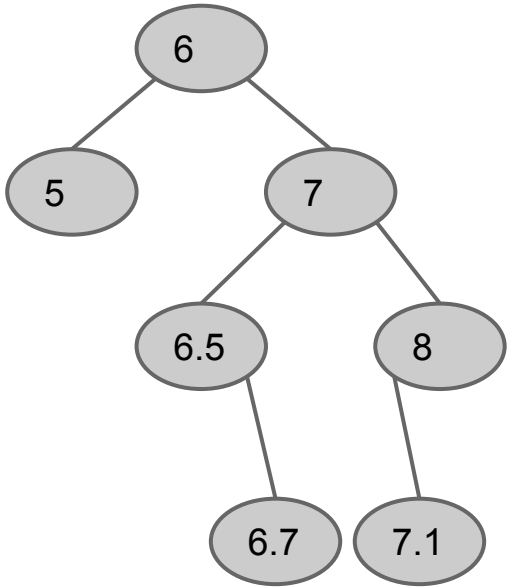


Binary Search Tree: Insertion



Insert the new node

Binary Search Tree: Insertion



Binary Search Trees - Delete

- Cases
  - Case 1: node is a leaf node
  - Case 2: node has only 1 child
  - Case 3: node has 2 children

Binary Search Trees - Delete

- If the node has a left and right child
  - Promote one child to take the place of the deleted one
  - Locate correct position for the other child in subtree of promoted child

Binary Search Trees - Delete

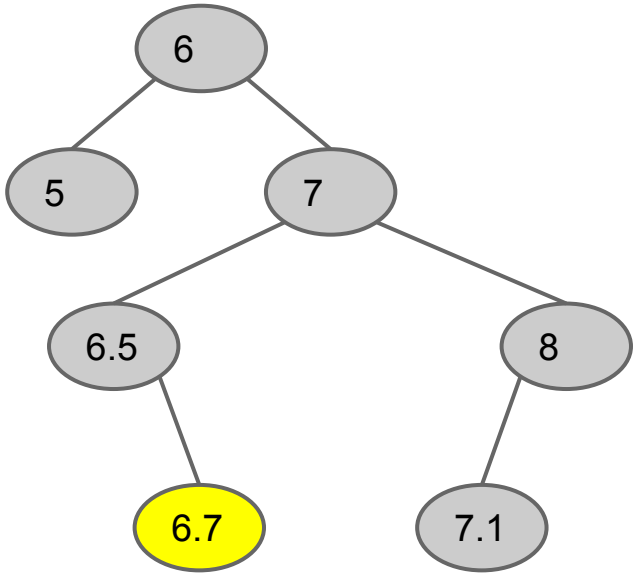
- It is the books convention to promote the immediate right child
  - position left subtree underneath

Binary Search Trees - Delete

- It is the books convention to promote the immediate right child
  - position left subtree underneath

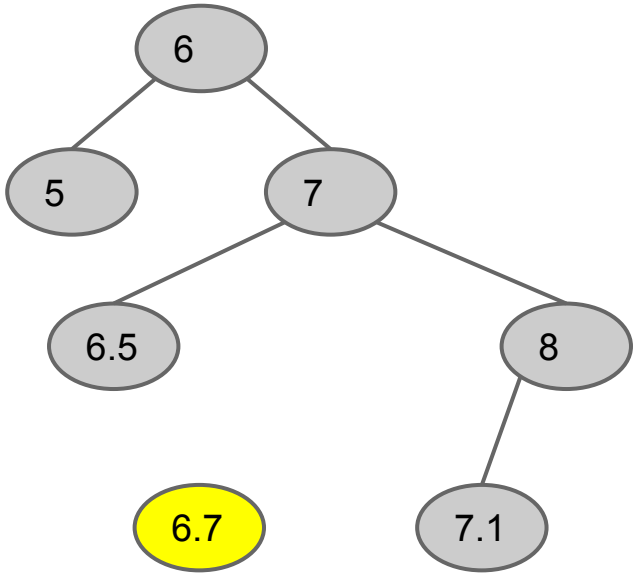
- It is better to promote the smallest leaf node in the left subtree

Binary Search Tree: Delete case 1



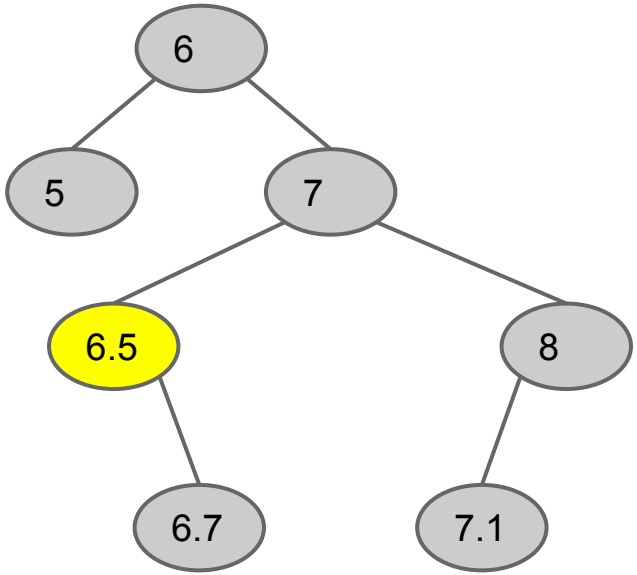
Delete node 6.7

Binary Search Tree: Delete case 1



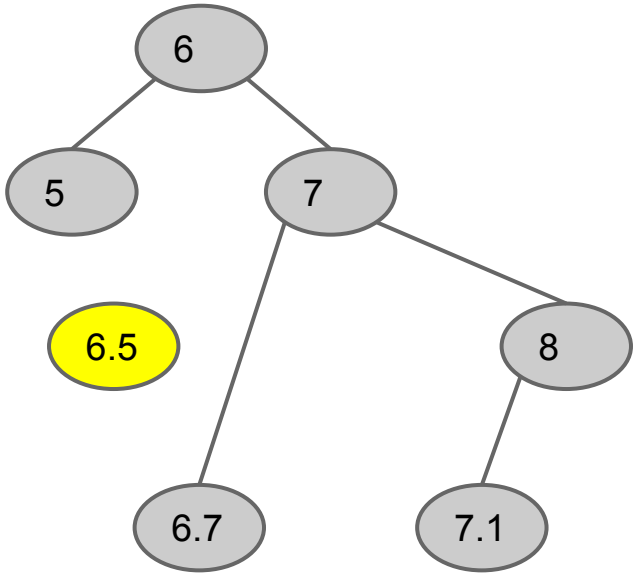
Delete node 6.7: remove the link from 6.5 to 6.7

Binary Search Tree: Delete case 2



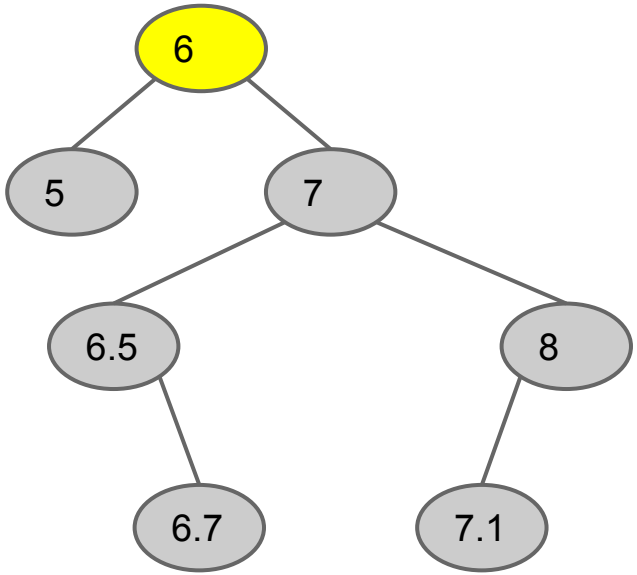
Delete node 6.5

Binary Search Tree: Delete case 2



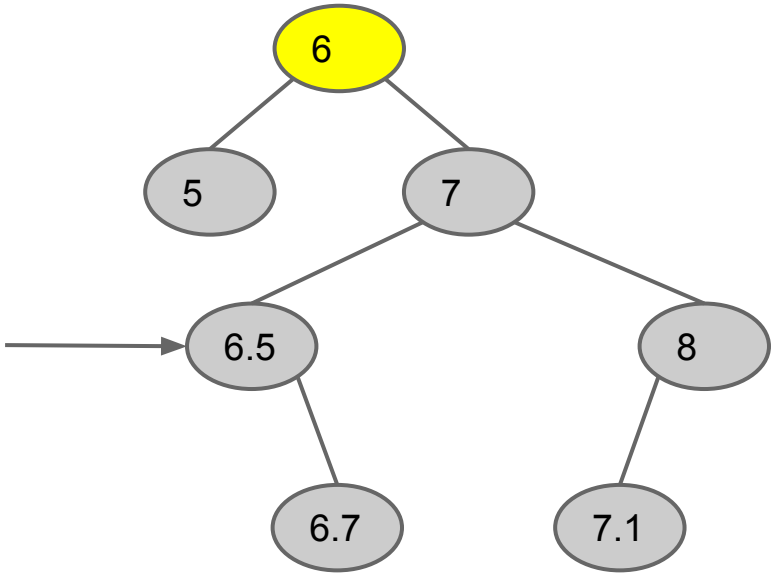
Delete node 6.5: update 7 to point to 6.7

Binary Search Tree: Delete case 2



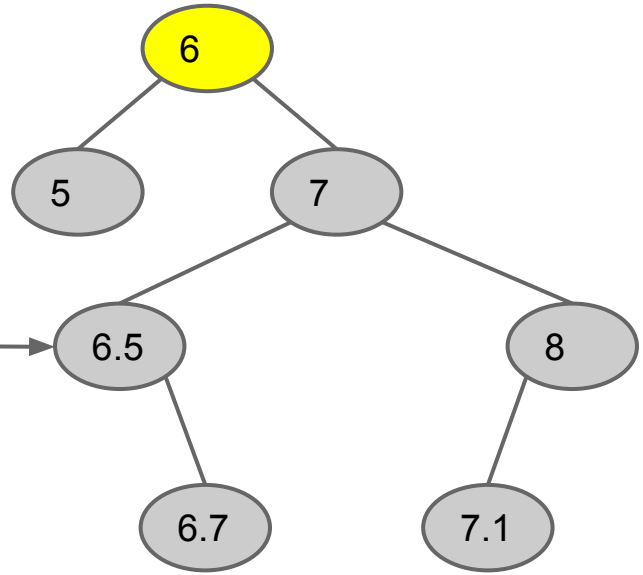
Delete node 6

Binary Search Tree: Delete case 2



Delete node 6: find smallest value to right of 6

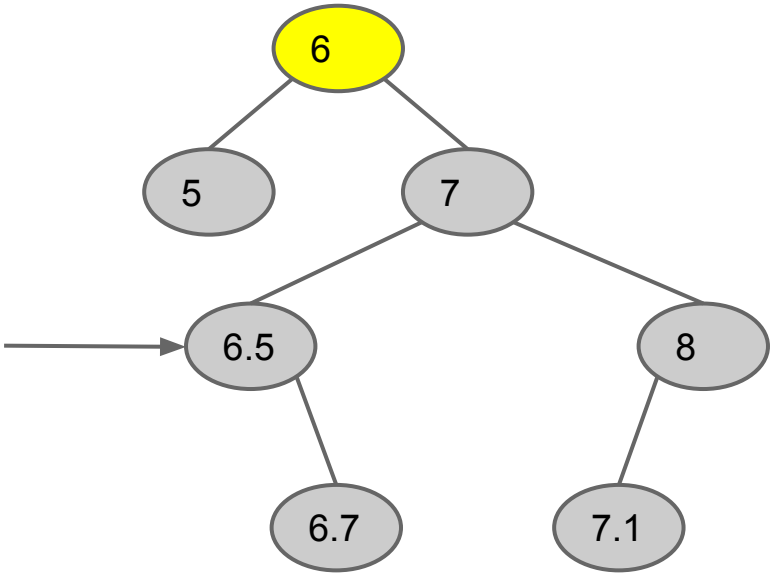
Binary Search Tree: Delete case 2



Since it is the smallest value, it cannot have anything to its left

Delete node 6: find smallest value to right of 6

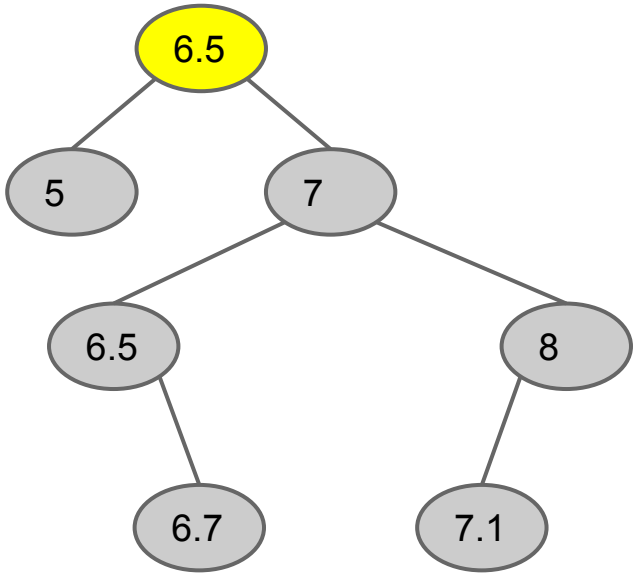
Binary Search Tree: Delete case 2



Since it is the smallest value, it cannot have anything to its left

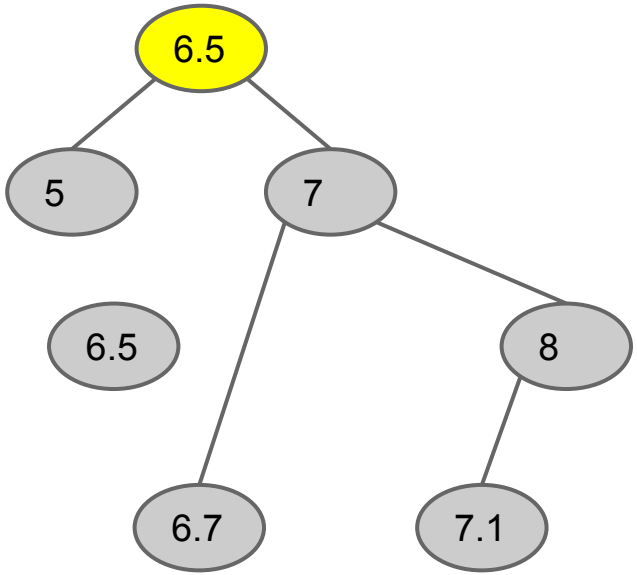
Delete node 6: find smallest value to right of 6; replace 6 with 6.5 and recursively delete

Binary Search Tree: Delete case 2



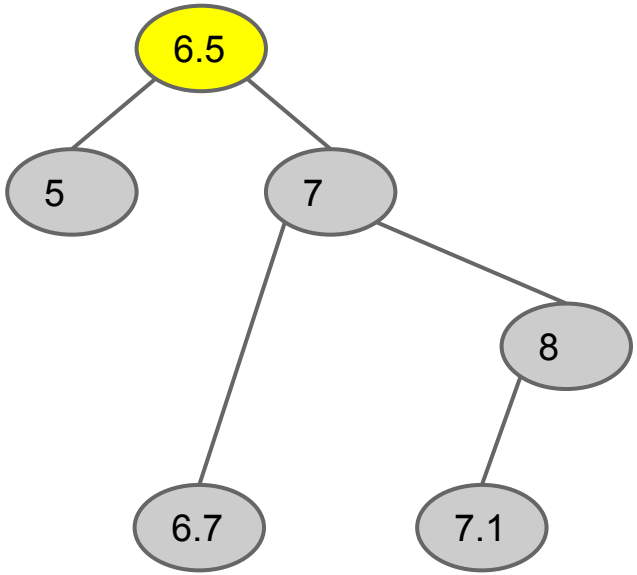
Delete node 6: find smallest value to right of 6;
replace 6 with 6.5 and recursively delete

Binary Search Tree: Delete case 2



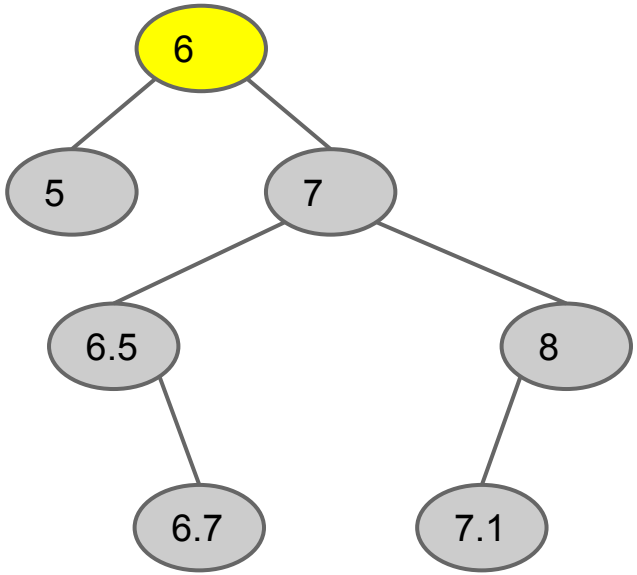
Delete node 6: find smallest value to right of 6;
replace 6 with 6.5 and recursively delete

Binary Search Tree: Delete case 2



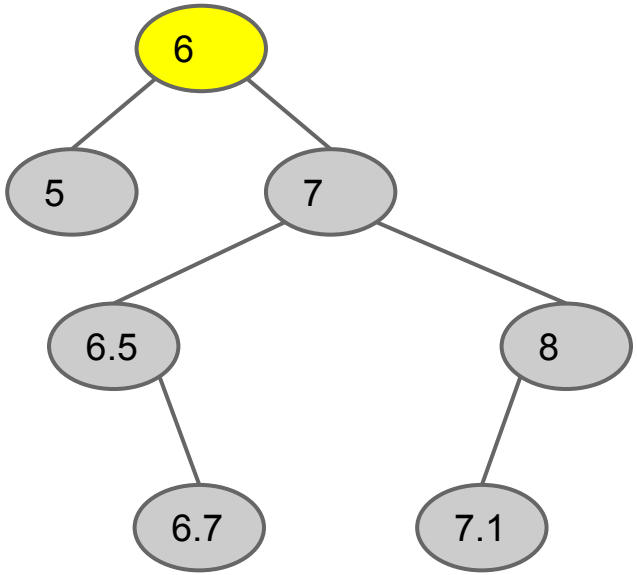
Delete node 6: find smallest value to right of 6;
replace 6 with 6.5 and recursively delete

Binary Search Tree: Delete case 2



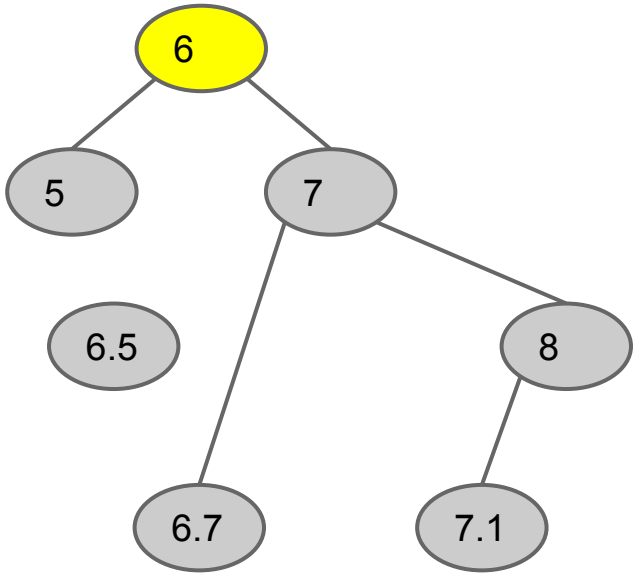
Delete node 6 with pointers?

Binary Search Tree: Delete case 2



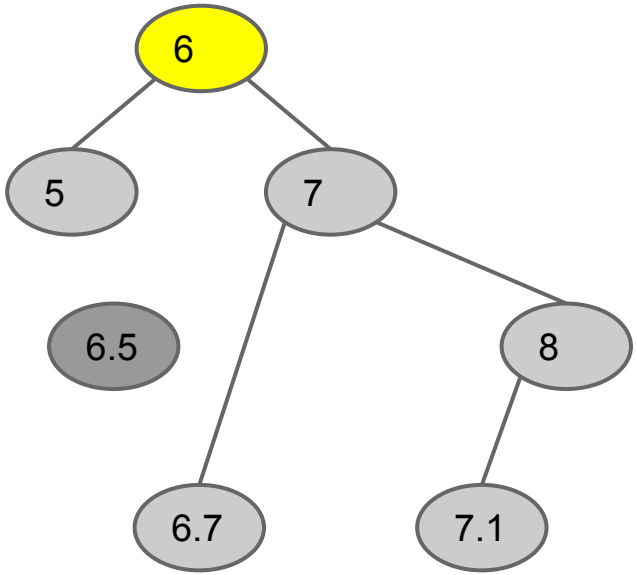
Delete node 6 with pointers: remove min of 6->right

Binary Search Tree: Delete case 2



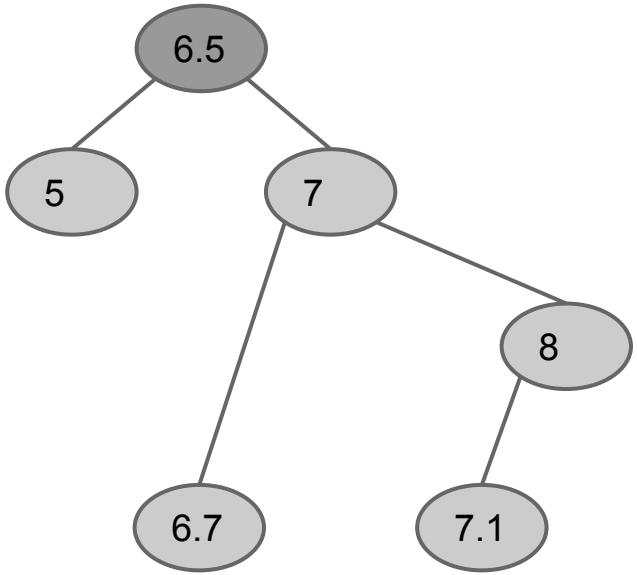
Delete node 6 with pointers: remove min of 6->right

Binary Search Tree: Delete case 2



Delete node 6 with pointers: remove min of 6->right; replace 6 with 6.5;

Binary Search Tree: Delete case 2



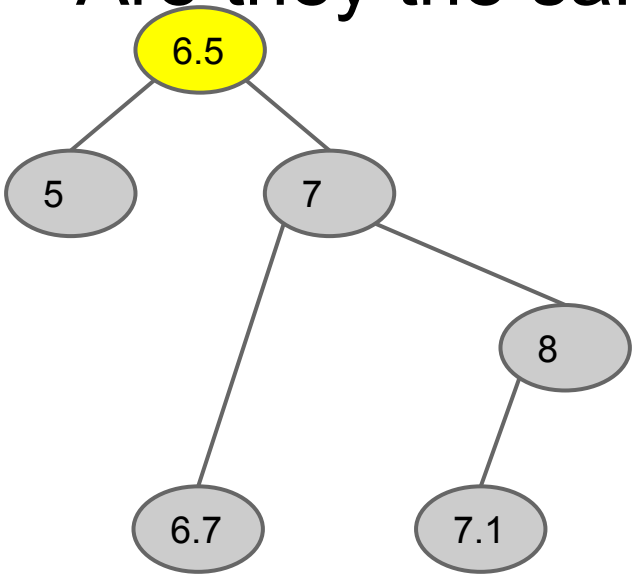
Delete node 6 with pointers: remove min of 6->right; replace 6 with 6.5;

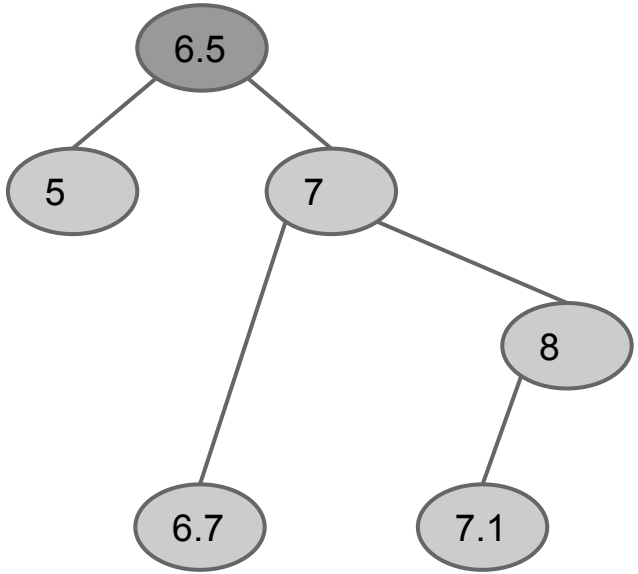
Binary Search Tree: Delete case 2

- Are they the same?

Binary Search Tree: Delete case 2

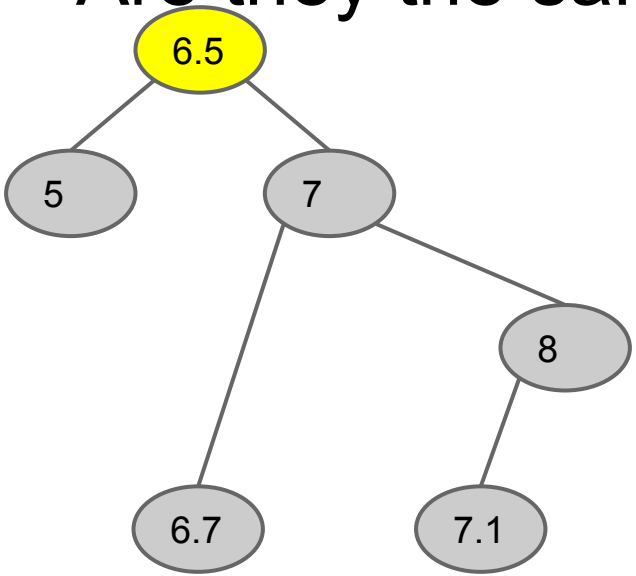
- Are they the same?

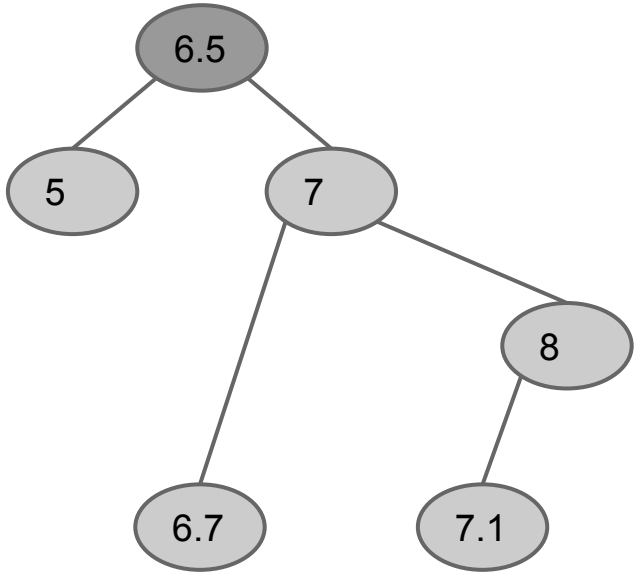




Binary Search Tree: Delete case 2

- Are they the same? Yes





- What's the difference?