CSE 373
Data Structures & Algorithms

Lecture 05

Trees: BST

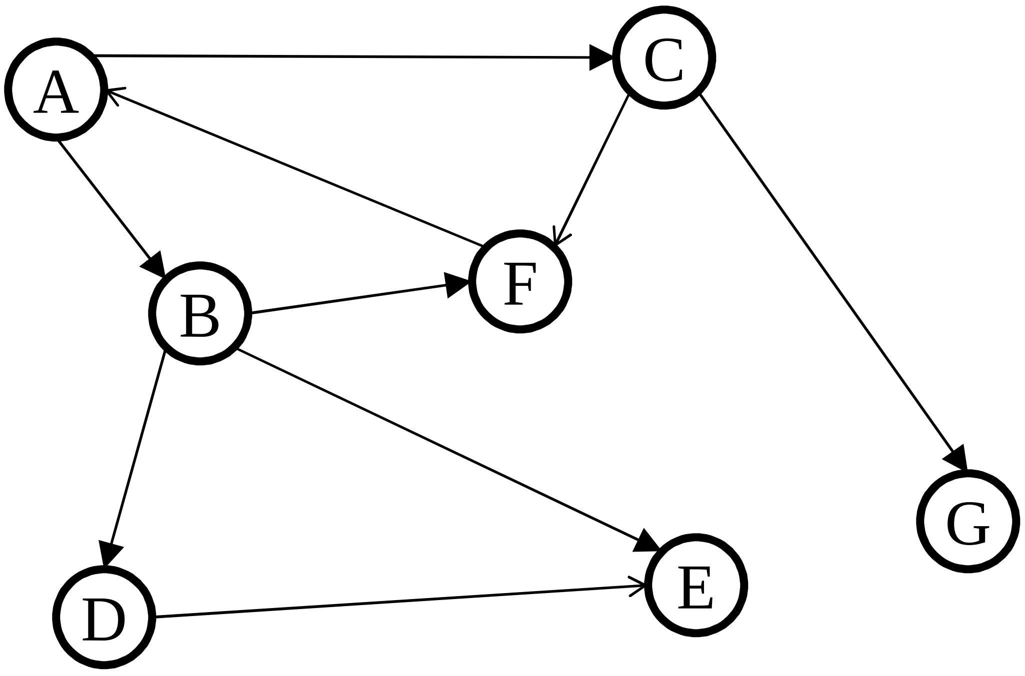
(Weiss 4.1, 4.2, 4.3)

Announcements

Homework 2

- Posted
- Due next Friday
- Turn-in in class OR drop box

Di-Graphs (Directed Graphs)



- Nodes: A,B,…
- Edges: A→B, …

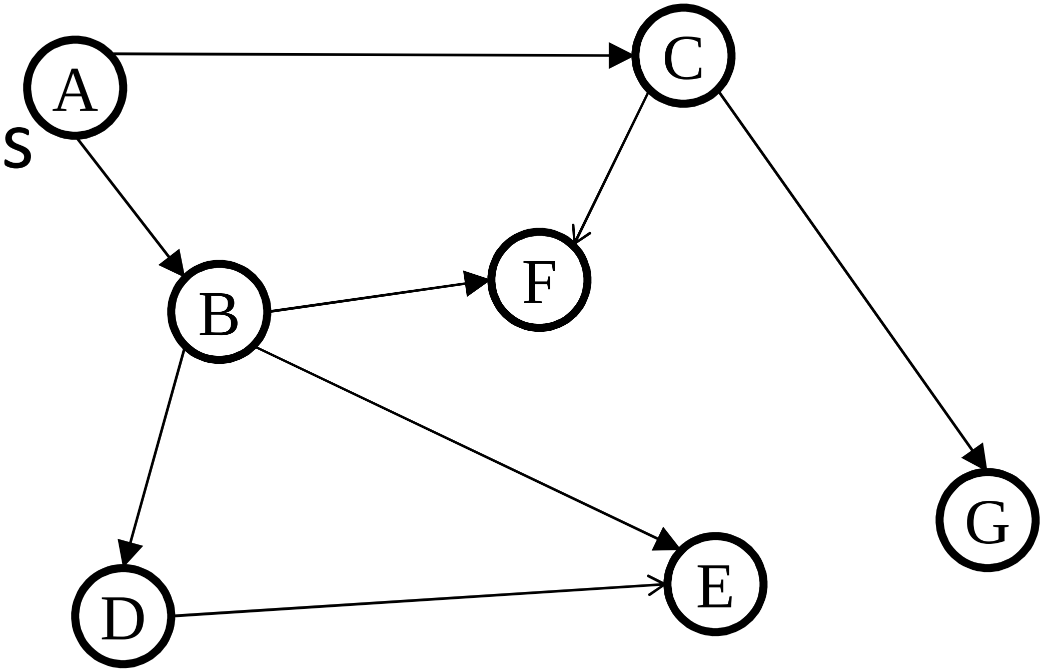
- **Paths** from A to E:
  A,B,E
  A,B,D,E
  A,B,F,A,B,F,A,B,E
- **Cycle**: A,B,F,A
- **Lengh** of a path = # of edges

What is a “tree” ?

- “A tree is a graph such that….”
  - How would you define a tree ?

Directed Acyclic Graph (DAG)

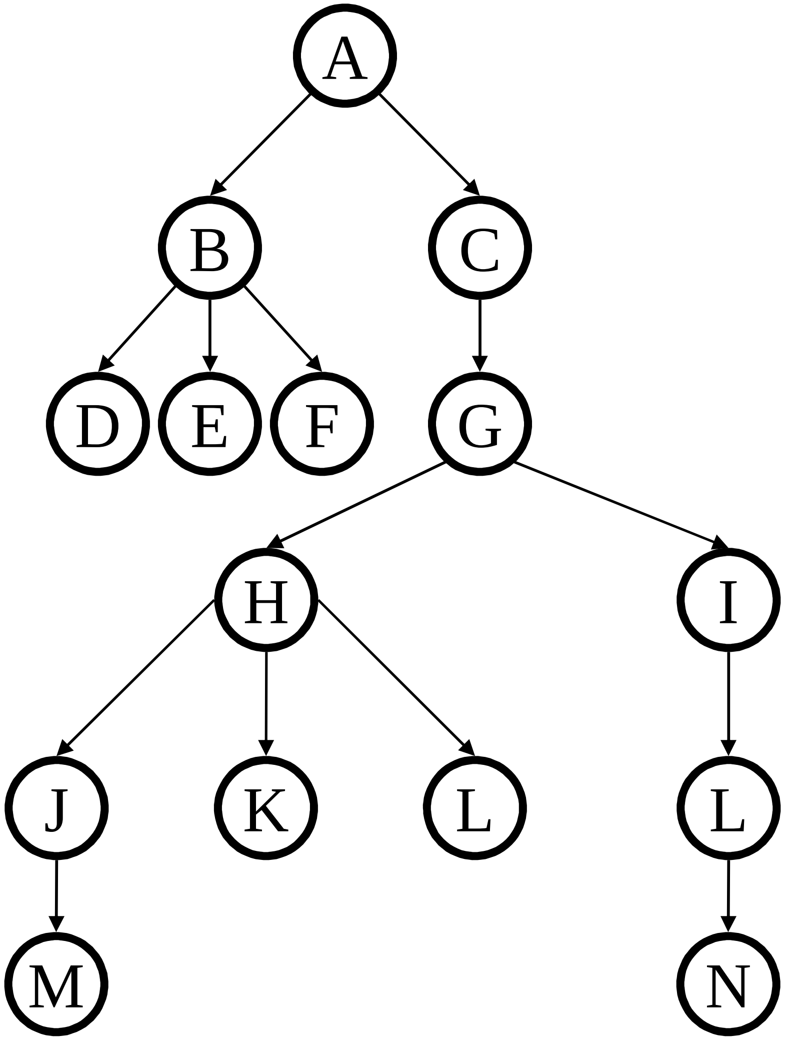
Definition: A DAG is a graph without cycles



Not a tree yet…

Trees

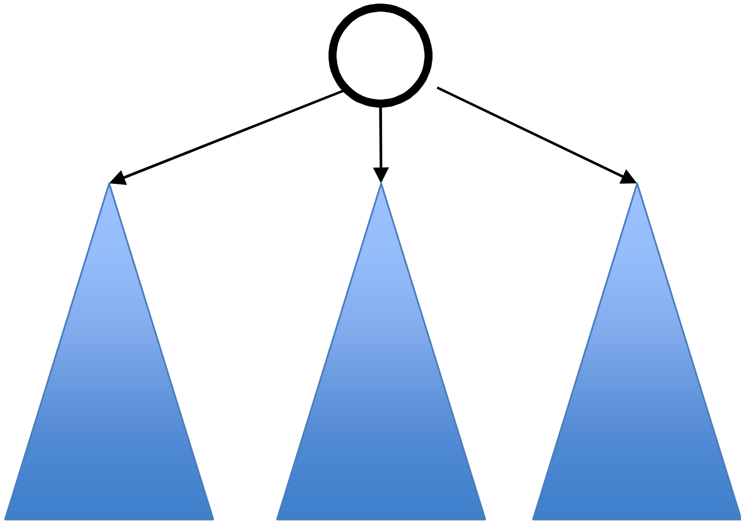
- A tree is a graph with a distinguished node A called *root* such that for any other node X, there exists a **unique path** from A to X
- See book for:
children, parent, sibling, leaf, depth, height



Trees

A recursive definition:

- A tree consists of a node (called the root) together with 0 or more (sub)trees $T_1, \ldots, T_k$



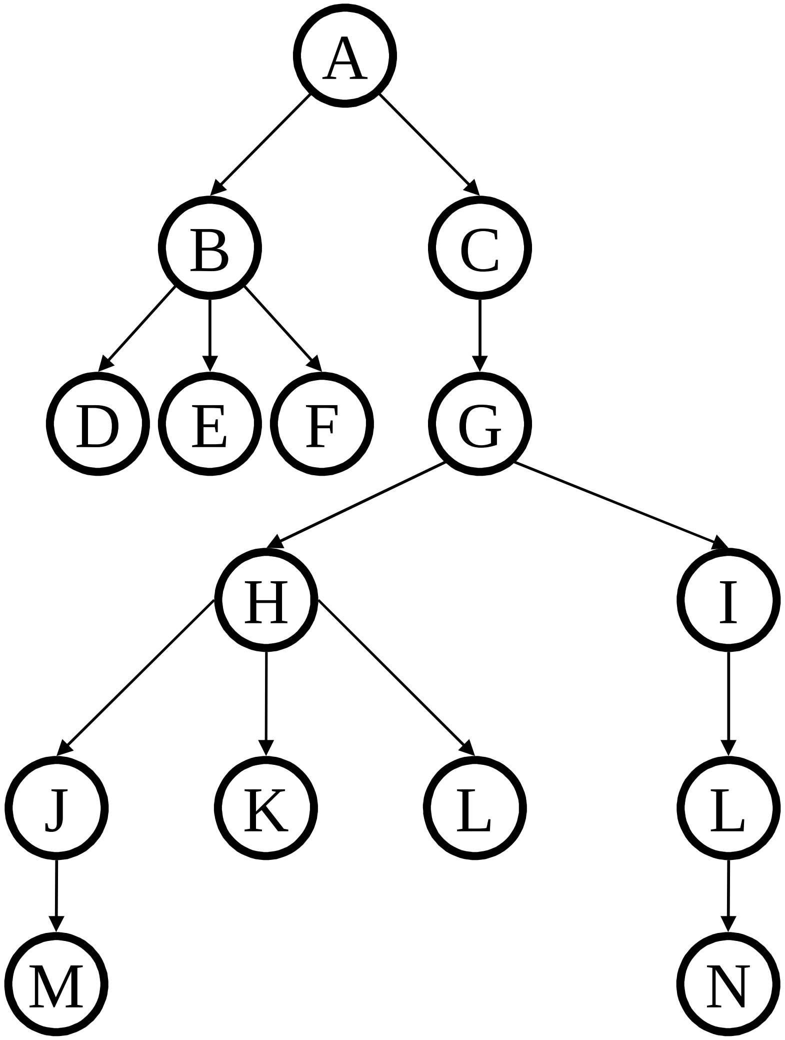
Trees

Please read these definitions in the book:

- Parent, children, leaves
- Path, length of a path (= # of edges)
- Depth of a node n (length of path root→n)
- Height of a node n (largest length n→leaf)
- Height of the tree

Tree Calculations Example

How high is this tree?



height(B) = 1
height(C) = 4

so height(A) = 5

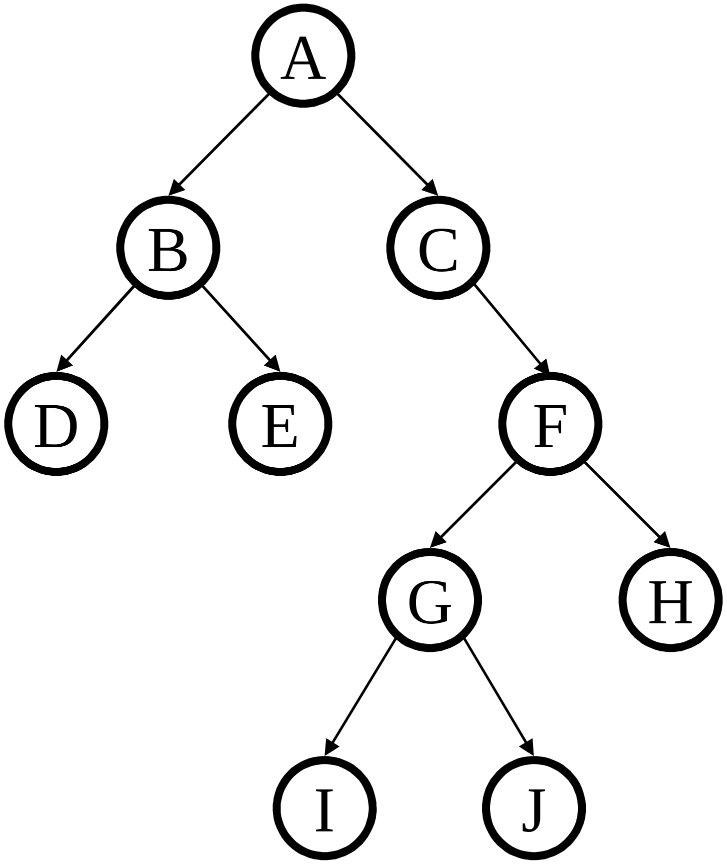
Quiz

- If a tree has n nodes, how many edges does it have ?
- If a tree has n nodes, how many leaves can it have ?

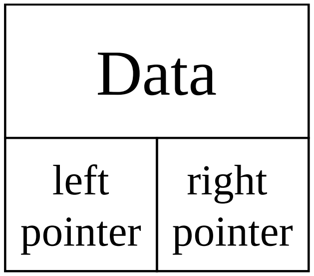
Binary Trees

Recursive definition

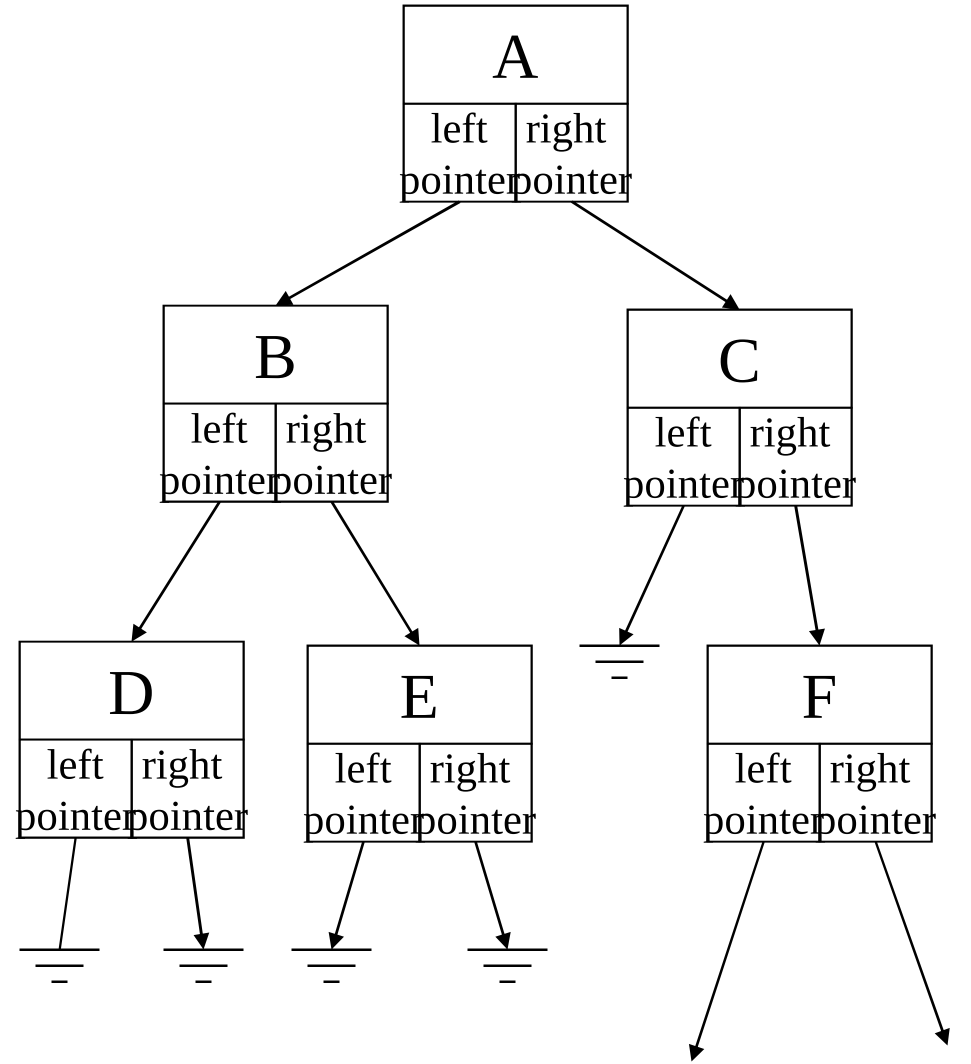
- A binary tree is
  - Either an empty tree
  - Or a node plus a left (sub)tree and a right (sub)tree

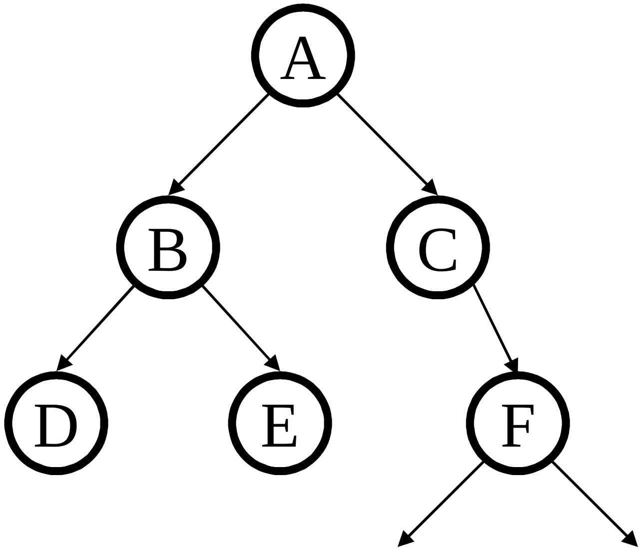


- Representation:



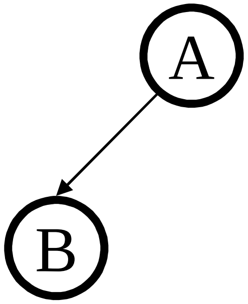
Binary Tree: Representation



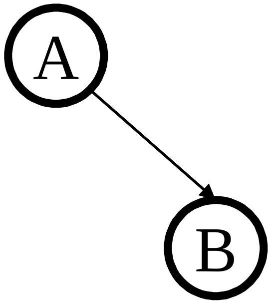


Subtle Distinction

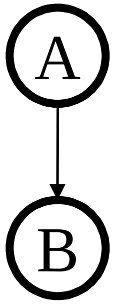
If a node has a single child we distinguish between
the case when it is a left child and when it is a right child



Left child only

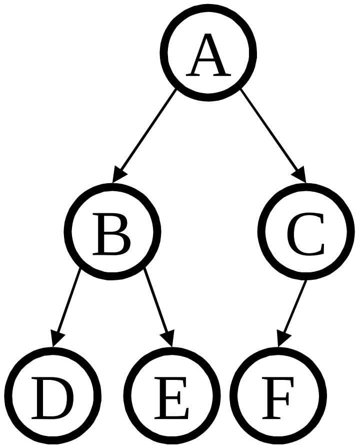


Right child only



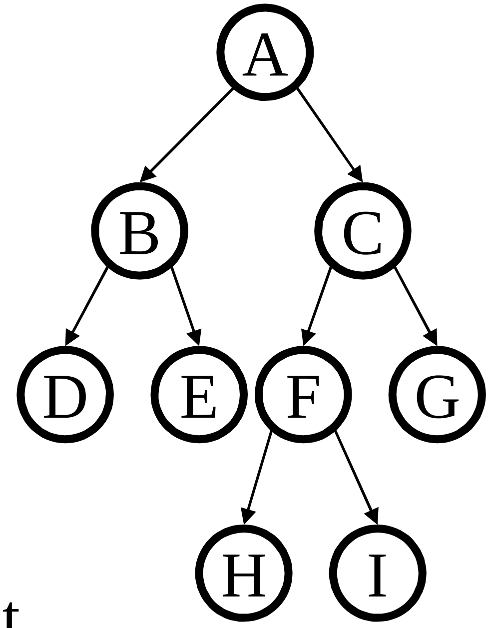
Not a “binary” tree

Binary Tree: Special Cases



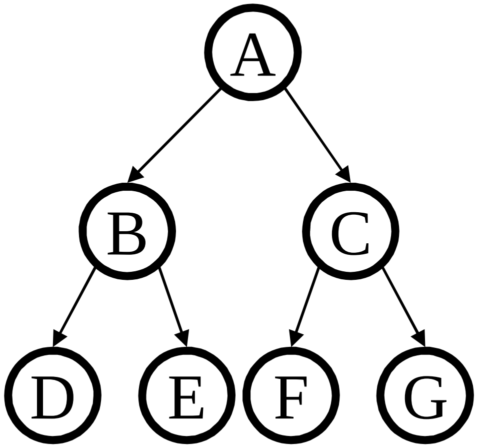
Complete Tree

Every level,except
possibly the last,
is completely filled,
and all nodes ar
as far left as possible.



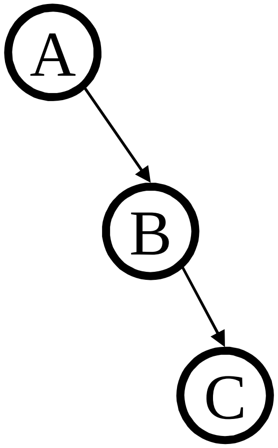
Full Tree

Every non-leaf
has two children



Perfect Tree

Full+complete

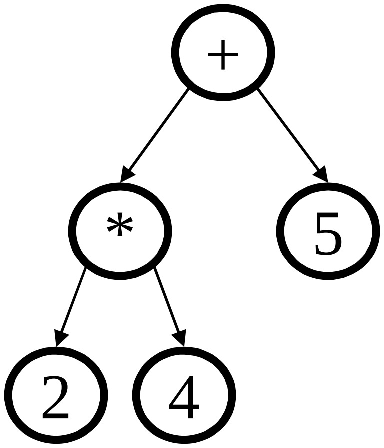


“List” Tree

Tree Traversals

A *traversal* is an order for visiting all the nodes of a tree

An expression tree:



Four types:

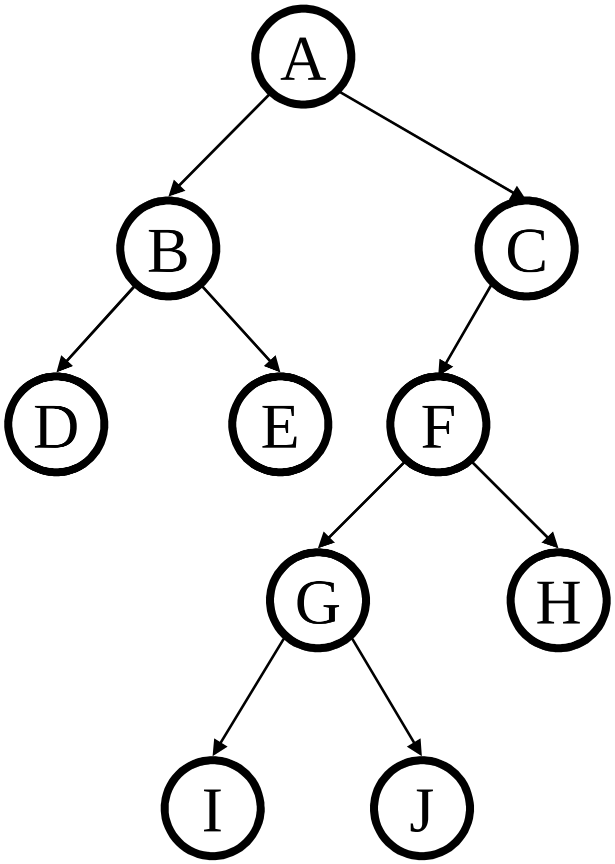
- Pre-order Root, left-subtree, right-subtree
- In-order: Left-subtree, root, right-subtree
- Post-order: Lef- subtree, right-subtree, root
- Breadth-first: left-right, top-down

Inorder Traversal

```
void traverse(BNode t){
   if (t != NULL)
     traverse (t.left);
  process t.element;
     traverse (t.right);
}
```

Tree Traversals

- Preorder: ABDECFGIJH
- Inorder: DBEAIGJFHC
- Postorder: DEBIJGHFCA
- Breadth-first: ABCDEFGHIJ



A binary tree is *complete* if and only if all nodes in breadth-first order are present

Quiz

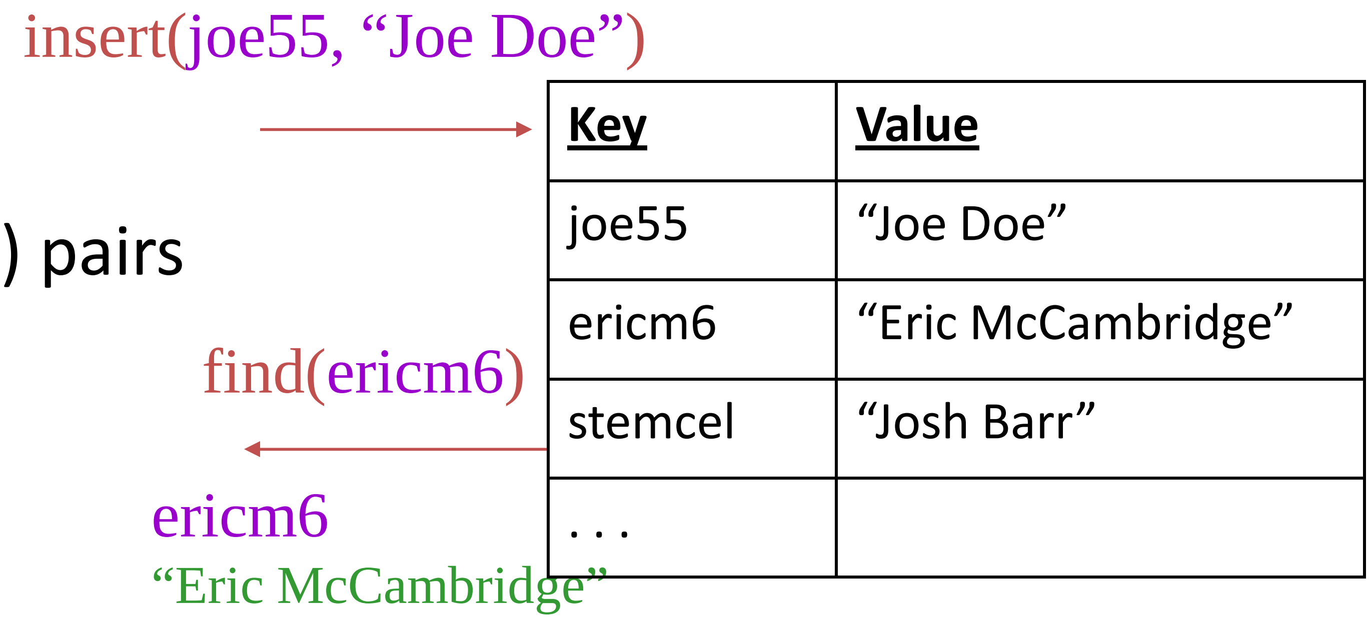
- If a binary tree has n nodes, what can its height be ?
- If a binary tree has n nodes, how many leaves can it have ?
- If the binary tree is full and has n nodes, how many leaves does it have ?

ADTs Seen So Far

- Stack
  - push, pop, top
- Queue
  - enqueue, dequeue, front

The Dictionary ADT (aka Map ADT)

- Data:
  - a set of (key, value) pairs



insert(joe55, "Joe Doe")

Key	**Value**
joe55	"Joe Doe"
ericm6	"Eric McCambridge"
stemcel	"Josh Barr"
. . .	

find(ericm6)

ericm6
"Eric McCambridge"

- Operations:
  - Insert (key, value)
  - Find (key)
  - Remove (key)

We will tend to emphasize the keys, don't forget about the stored values

A Modest Few Uses

- Sets
- Dictionaries
- Networks : Router tables
- Operating systems : Page tables
- Compilers : Symbol tables

- Anytime you want to store information according to some key and be able to efficiently retrieve it

Probably the most widely used ADT!

Implementation

	Insert	Find	Delete
Unsorted linked lists			
Unsorted array			
Sorted array			

What are the asymptotic running times ?

Implementation

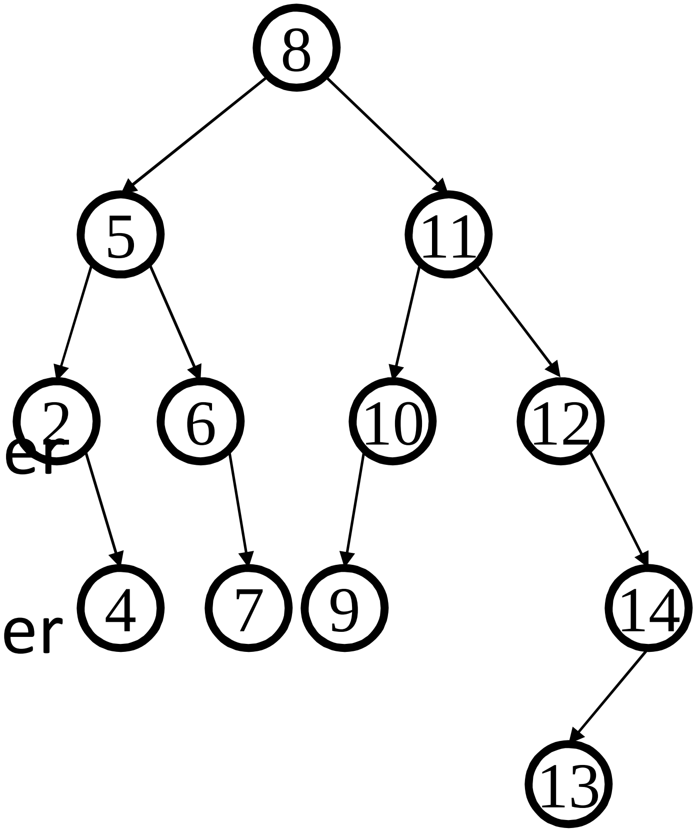
	Insert	Find	Delete
Unsorted linked lists	O(1)	O(n)	O(n)
Unsorted array	O(1)	O(n)	O(n)
Sorted array	O(log(n)+n)	O(log n)	O(log(n)+n)

What limits the performance ?

Binary Search Tree Data Structure

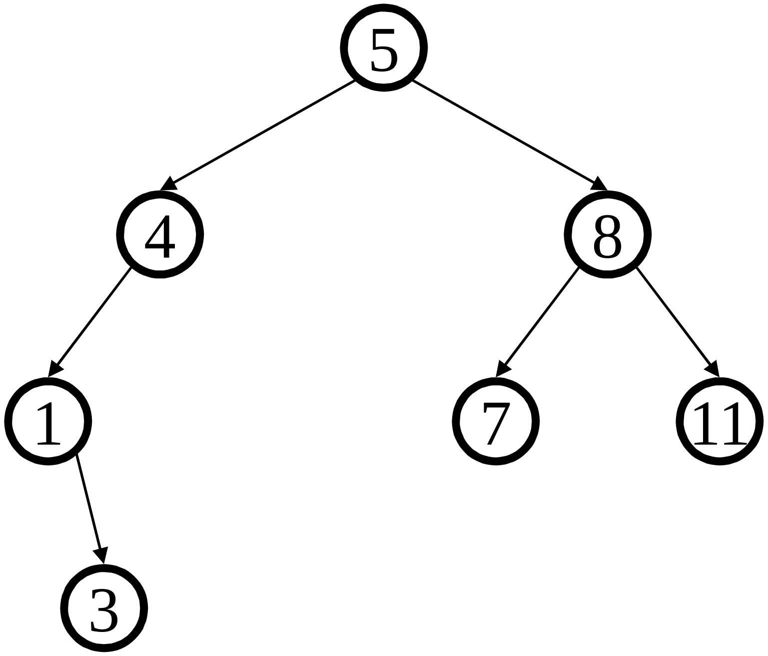
A Binary Search Tree (BST) is a binary tree with the following *ordering property*:

- For every node n with key k:
  - all keys in left subtree are smaller than k
  - all keys in the right subtree larger than k

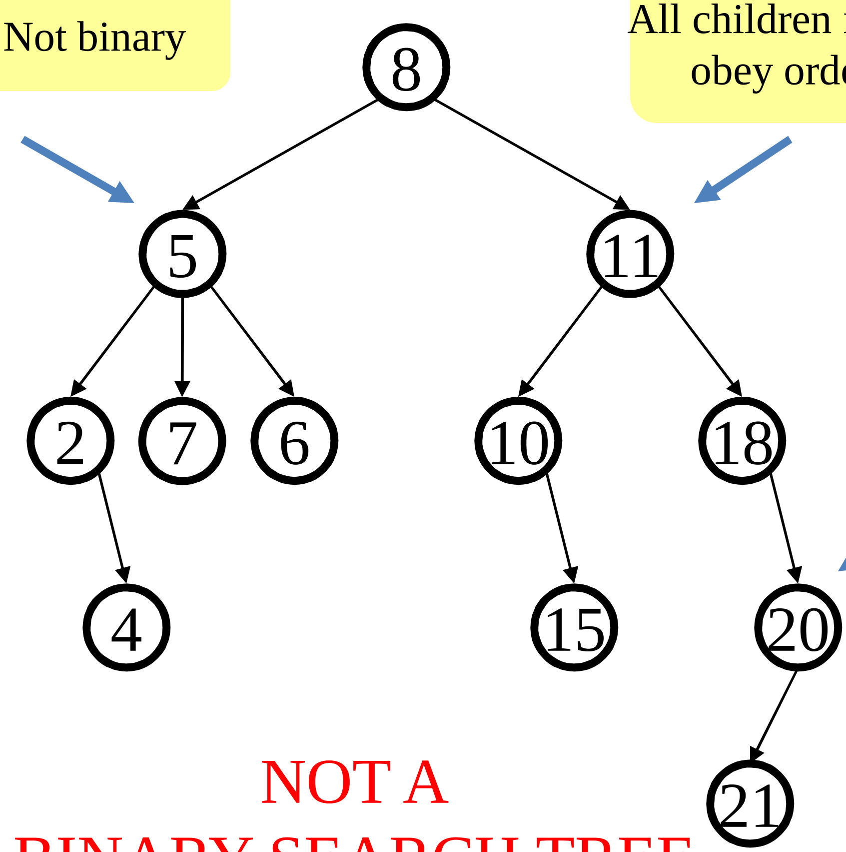


Comparison, equality testing

Example and Counter-Example



BINARY SEARCH TREE

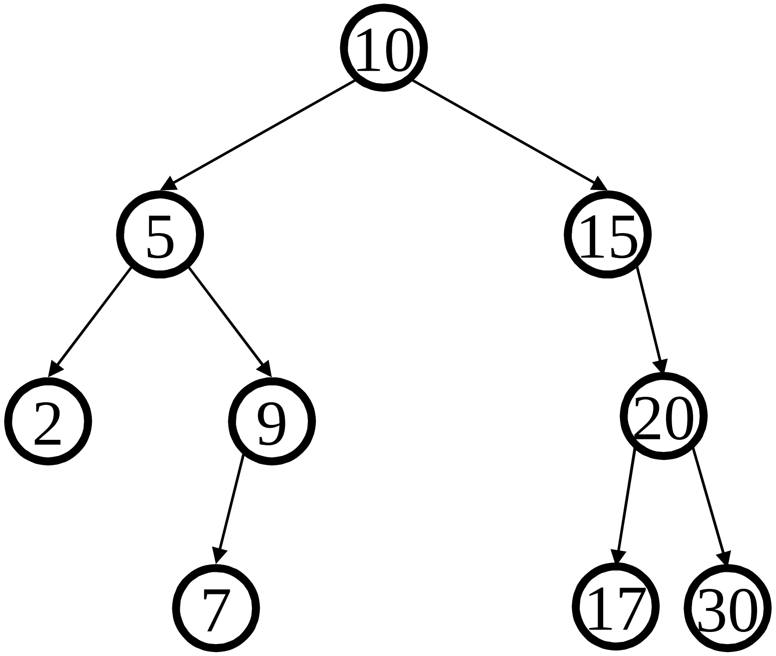


Not binary

All children must obey order

NOT A
BINARY SEARCH TREE

Find in BST, Recursive



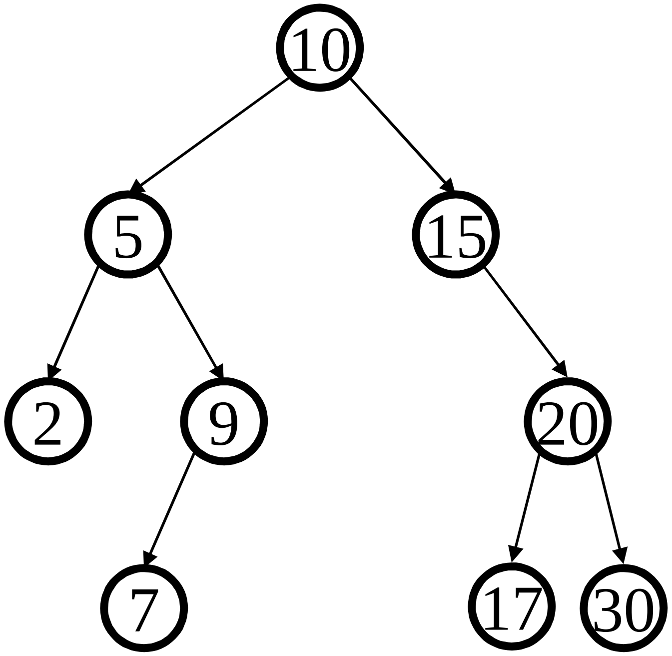
```
Node Find(Object key,
            Node root) {
  if (root == NULL)
    return NULL;

  if (key < root.key)
   return Find(key,root.left);
  else if (key > root.key)
   return Find(key,root.right);
  else
    return root;
}
```

Runtime:

Θ(depth) = $\Theta(n)$ worst, $\Theta(\log n)$ avg

Find in BST, Iterative

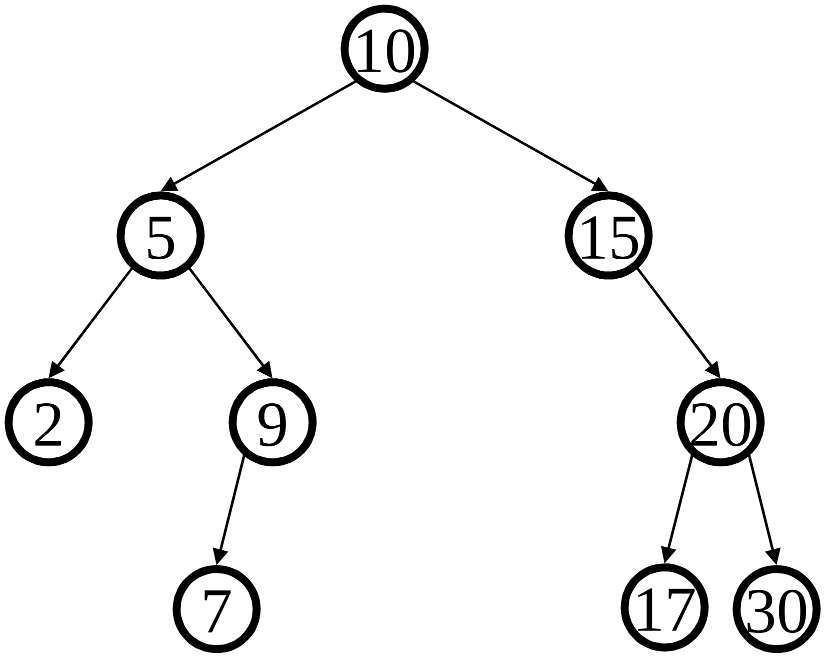
```
Node Find(Object key,Node root)
  {
  while (root != NULL &&
         root.key != key) { if
  (key < root.key)
      root = root.left;
    else
      root = root.right;
  }

  return root;
}
```



Runtime:

same as before!

Insert in BST



Insert(13)
Insert(8)
Insert(31)

Insertions happen only at the leaves – easy!

Runtime:

O(depth) = O(n) worst, O(log n) avg

The Height of a BST

- Important question: if a BST has n nodes, what is its height ?
  - Best case: O(log n)
  - Worst case: O(n)

- Simpler question: if we insert n keys into an empty BST, what is its height ?

Insertions Only

- Suppose keys 1, 2, 3, 4, 5, 6, 7, 8, 9 are inserted into an initially empty BST.

  Runtime depends on the order!

  - in given order

$\Theta(n^2)$

  - in reverse order

$\Theta(n^2)$

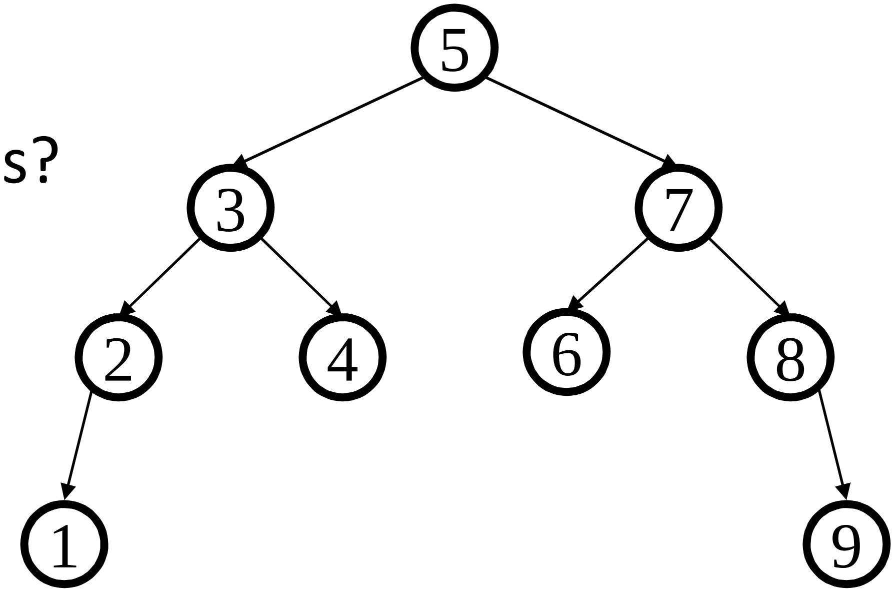
  - median first, then left median, right median, etc.

5, 3, 7, 2, 1, 6, 8, 9 better: $n \log n$

BuildTree for BST

Insert n keys into an empty BST = "bulk insertion"

- Example: 1, 2, 3, 4, 5, 6, 7, 8
- What we if could somehow re-arrange them
  - median first, then left median, right median, etc.
  - 5, 3, 7, 2, 1, 4, 8, 6, 9
  - What tree does that give us?
  - What big-O runtime?



O(N log N)

The Height of a BST after Insertions Only

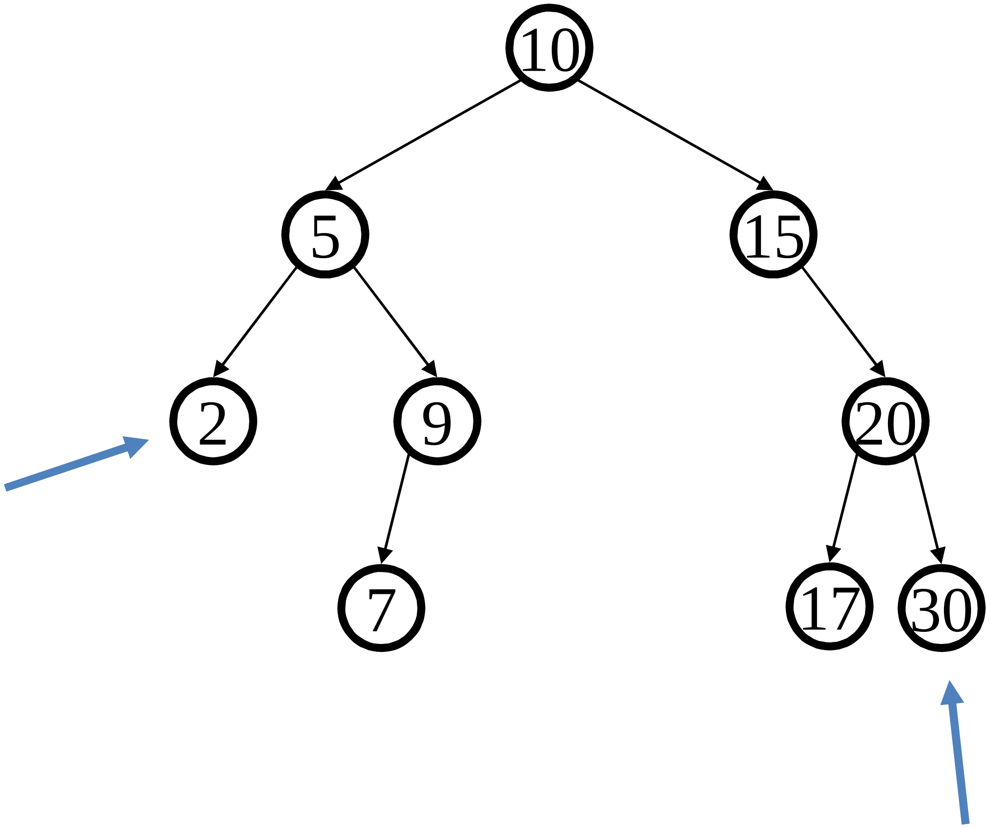
- Bulk insertion of n keys → height = O(log n)

- Regular insertion of n keys:
  - Worst case O(n)
  - Best case O(log n)
  - Average case O(log n) READ THE BOOK

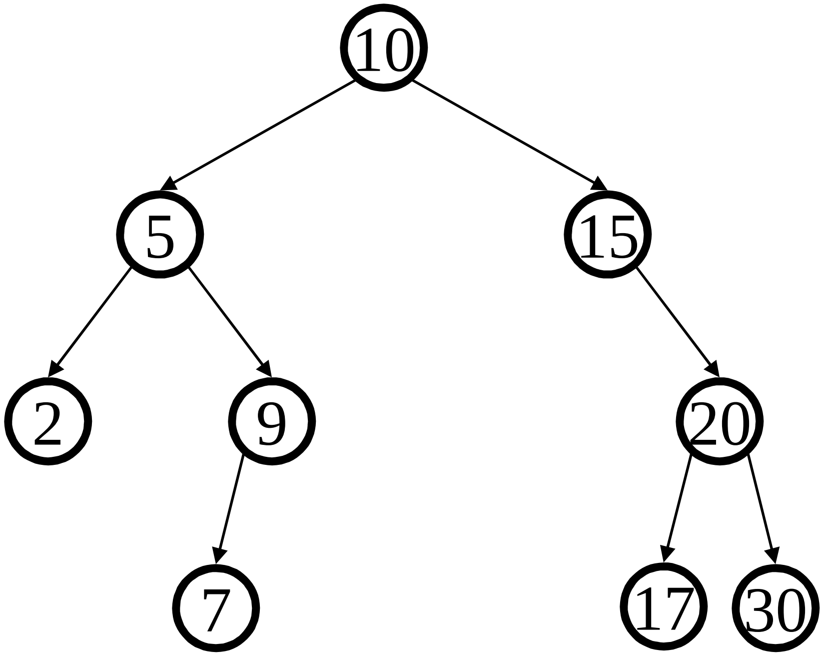
FindMin/FindMax

- Find minimum

- Find maximum



Deletion in BST



Why might deletion be harder than insertion?

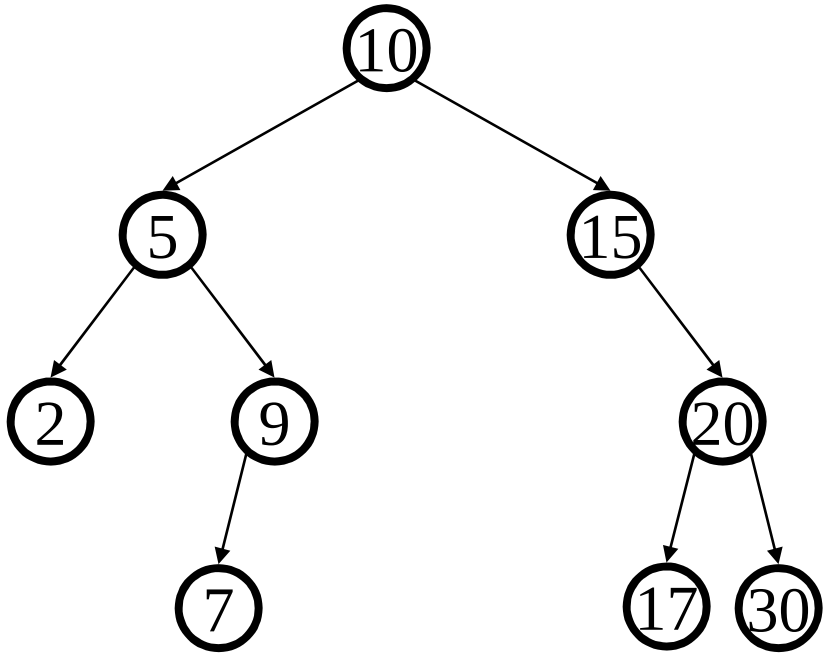
May be in middle, instead of at leaf

Lazy Deletion

Instead of physically deleting nodes,
just mark them as deleted

- \+ simpler
- \+ physical deletions done in batches
- \+ some adds just flip deleted flag

- – extra memory for deleted flag
- – many lazy deletions slow finds
- – some operations may have to be modified (e.g., min and max)

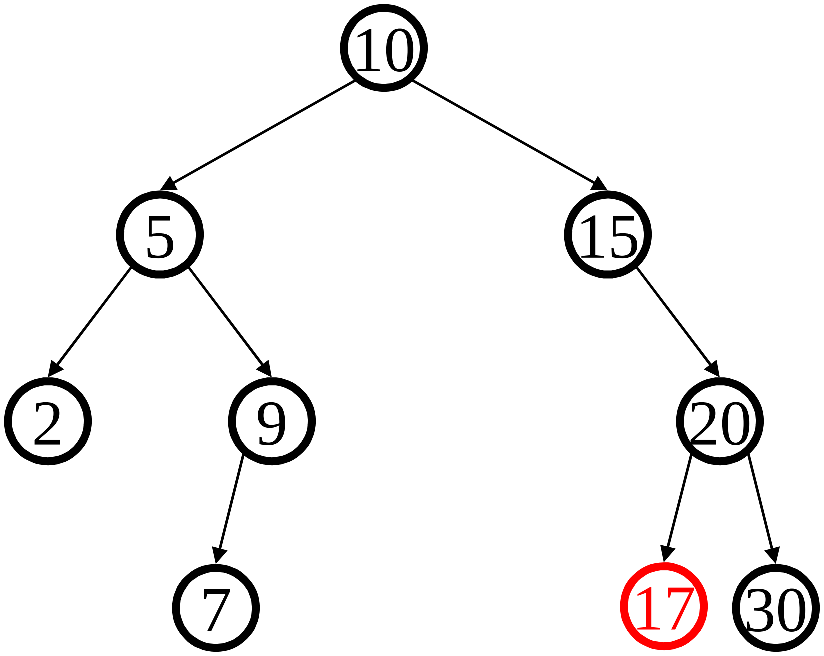


Non-lazy Deletion

- Removing an item disrupts the tree structure.
- Basic idea: find the node that is to be removed. Then “fix” the tree so that it is still a binary search tree.
- Three cases:
  - node has no children (leaf node)
  - node has one child
  - node has two children

Non-lazy Deletion – The Leaf Case

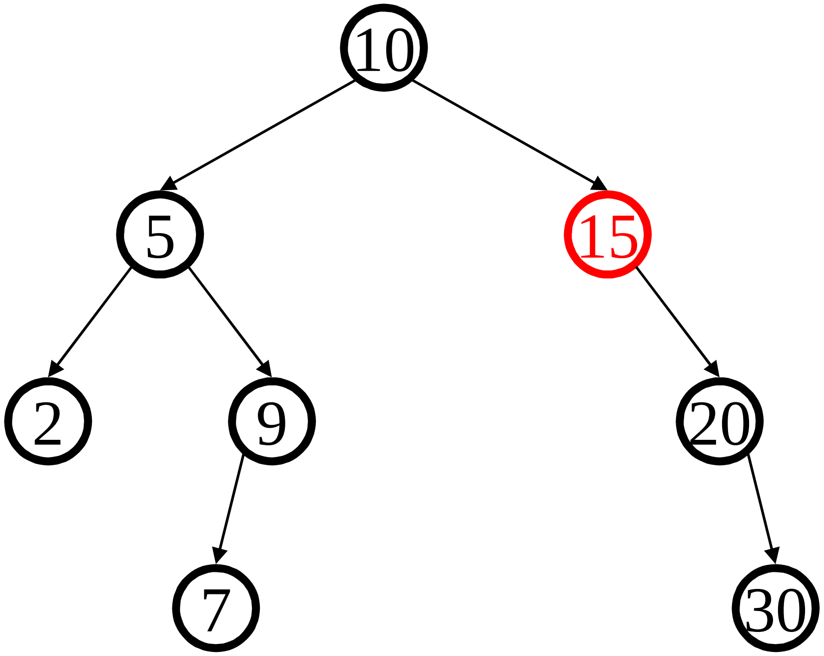
Delete(17)



Easy – prune

Deletion – The One Child Case

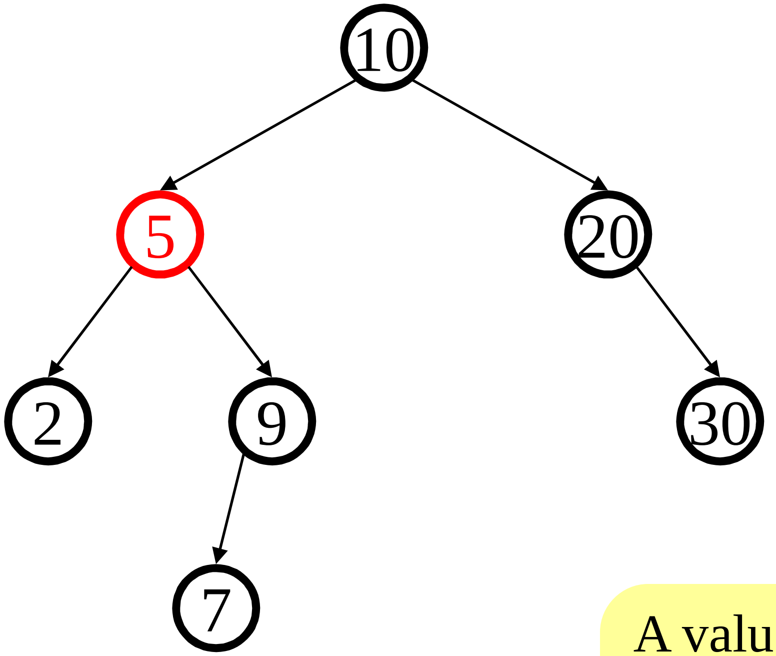
Delete(15)



Pull up child – will this always work?

Deletion – The Two Child Case

Delete(5)



A value guaranteed to be between the two subtrees!
- *succ* from right subtree
- *pred* from left subtree

What can we replace 5 with?

How long do these operations take? (find, insert, delete)

Deletion – The Two Child Case

Idea: Replace the deleted node with a value guaranteed to be between the two child subtrees!

Options:

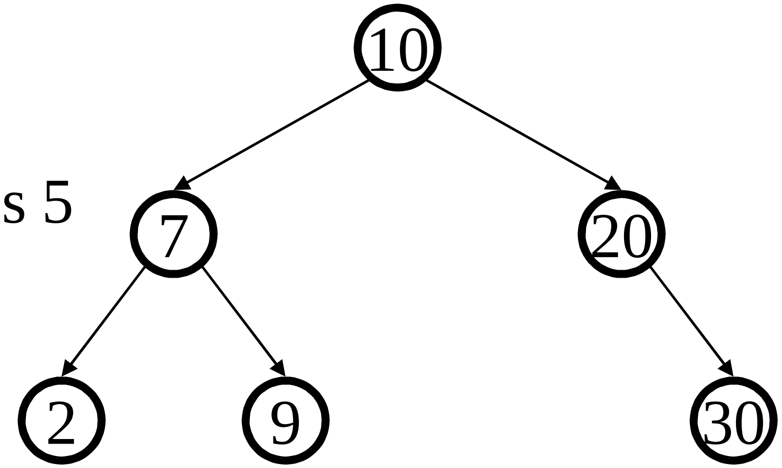
- *succ* from right subtree: findMin(t.right)
- *pred* from left subtree : findMax(t.left)

Now delete the original node containing *succ* or *pred*

- Leaf or one child case – easy!

Why leaf or one child case?

Finally…



7 replaces 5

Original node containing
7 gets deleted

Binary Trees: Some Numbers

Recall: height of a tree = longest path from root to leaf.

For binary tree of height *h*:

- max # of leaves: 2^h
- max # of nodes: $2^{(h+1)} - 1$
- min # of leaves: 1
- min # of nodes: $h + 1$

We're not going to do better than log(n) height, and we need something to keep us away from n