CSE 326: Data Structures
Topic #5
Branching Out

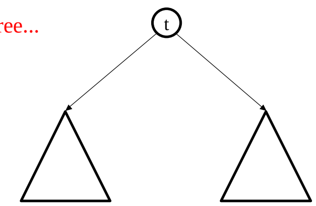
Luke McDowell
Summer Quarter 2003

Today's Outline

- Homework #3 Intro
- Some Tree Review
- Binary Trees
- Dictionary ADT / Search ADT
- Binary Search Trees

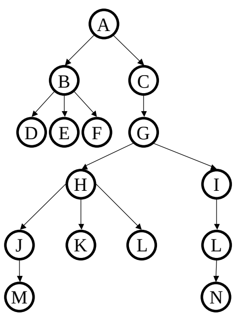
Tree Calculations

Find the height of the tree...



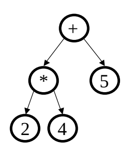
Runtime:

Tree Calculations Example



More Recursive Tree Calculations: Traversals

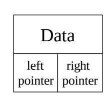
- A *traversal* is an order for visiting all the nodes of a tree
- Three types:
  - Pre-order
    - Root, left subtree, right subtree
  - In-order
    - Left subtree, root, right subtree
  - Post-order
    - Left subtree, right subtree, root

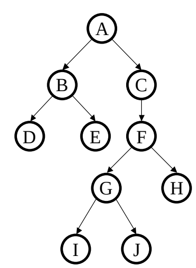


An expression tree

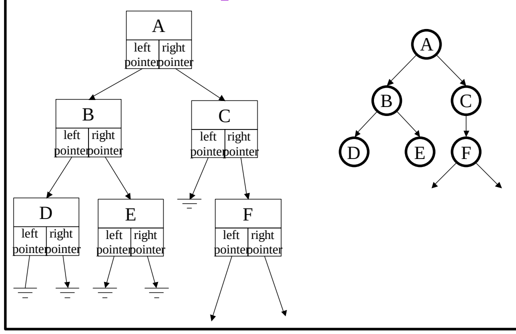
Binary Trees

- Binary tree is
  - a root
  - left subtree *(maybe empty)*
  - right subtree *(maybe empty)*
- For tree of depth d:
  - max # of leaves:
  - max # of nodes:
- Representation:

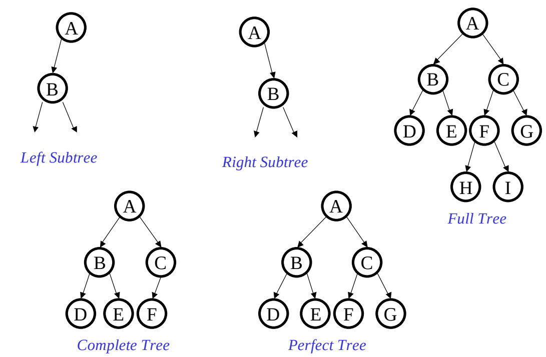




Representation



A Few More Trees



What We Can Do So Far

- Stack
  - Push
  - Pop
- Queue
  - Enqueue
  - Dequeue
- List
  - Insert
  - Remove
  - Find
- Priority Queue
  - Insert
  - DeleteMin

Remember decreaseKey?

New! The Dictionary ADT

- Dictionary ADT:
  - Maps *values* to user-specified *keys*
  - Or: a set of (key, value) pairs
    - Keys may be any (homogeneous) type
    - Values may be any (homogeneous) type
- Operations:
  - Insert (key, value)
  - Find (key)
  - Remove (key)

insert
- steve0@u
  - aka stevaroo…

find(erbad)
- erbad
  - Aiman Erbad, the …

- lucasm
  - Lucas Kelsey McDowell, your favorite 5th year grad student who needs help with Hollywood trivia
- stevaroo
  - Steven Loyd Martin, a recently graduated CSE ugrad who's destined fpr great things at Berkeley
- erbad
  - Aiman Mahmood Erbad, the man with the mysterious accent and real name that is even longer

The Dictionary ADT is sometimes called the "Map ADT"

Also New! The Search ADT

- Search ADT:
  - Contains unique user-specified *keys*
  - Or: a set of keys
    - Keys may be any (homogeneous) type
- Operations:
  - Insert (key)
  - Find (key)
    - Checks for membership
  - Remove (key)

insert
- Grill

find(Bake)
NOT FOUND

- Fry
- Simmer
- Puree
- Braise
- Poach
- Sear
- Stirfry
- Puree

The Search ADT is sometimes called the "Set ADT"

A Modest Few Uses

- Arrays
- Sets
- Dictionaries
- Router tables
- Page tables
- Symbol tables

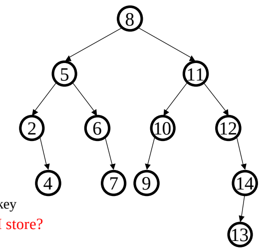
Naïve Implementations

	insert	find	delete
• Unsorted Linked list			
• Unsorted array			
• Sorted array			

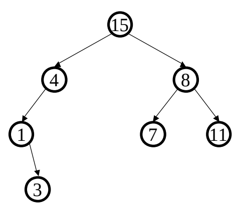
What limits the performance?

Binary Search Tree Dictionary Data Structure

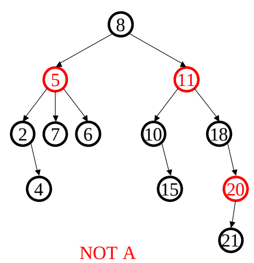
- Binary tree property
  - each node has ? 2 children
  - result:
    - storage is small
    - operations are simple
    - average depth is small
- Search tree property
  - all keys in left subtree smaller than root's key
  - all keys in right subtree larger than root's key
  - result: easy to find any given key
- What must I know about what I store?



Example and Counter-Example

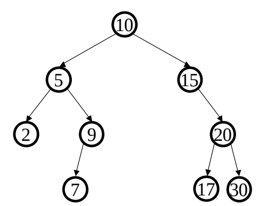


BINARY SEARCH TREE



NOT A
BINARY SEARCH TREE

Finding a Node

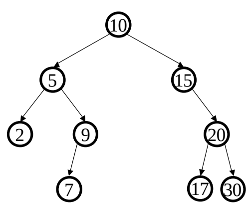


Runtime:

```
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;
}
```

Iterative Find

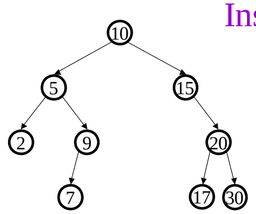
```
Node Find(Object key,
            Node root) {

  while (root != NULL &&
         root.key != key) {
    if (key < root.key)
      root = root.left;
    else
      root = root.right;
  }

  return root;
}
```



Why It's Called a "*Binary Search* Tree"

2	5	7	9	10	15	17	20	30

Insert



Runtime:

BuildTree for BSTs

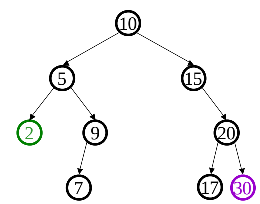
- Suppose the data 1, 2, 3, 4, 5, 6, 7, 8, 9 is inserted into an initially empty BST:
  - in order
  - in reverse order
  - median first, then left median, right median, etc.

Analysis of BuildTree

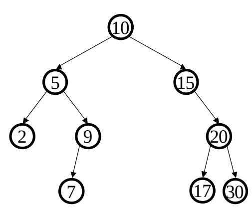
- Worst case: $O(n^2)$ as we've seen
- Average case assuming all orderings equally likely:
  - Sum of all depths:
    - $D(N) = D(I) + D(N - I - 1) + (N - 1)$
      $=$
  - Average depth of a node:
  - Total runtime:

Bonus: FindMin/FindMax

- Find minimum
- Find maximum



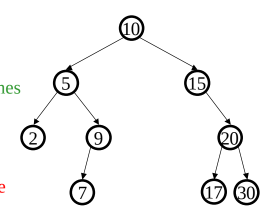
Deletion



Why might deletion be harder than insertion?

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)



Lazy Deletion

Delete(17)

Delete(15)

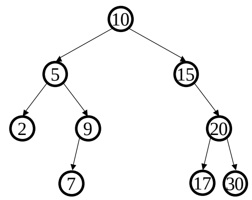
Delete(5)

Find(9)

Find(16)

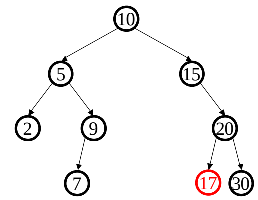
Insert(5)

Find(17)



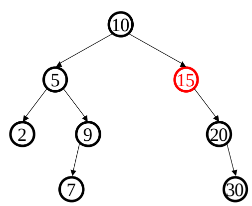
Deletion - Leaf Case

Delete(17)



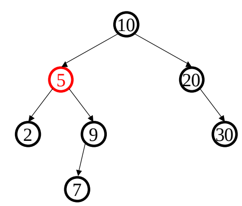
Deletion - One Child Case

Delete(15)

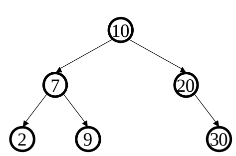


Deletion - Two Child Case

Delete(5)



Finally…



Thinking about Binary Search Trees

- Observations
  - Each operation views two new elements at a time
  - Elements (even siblings) may be scattered in memory
  - Binary search trees are fast *if they're shallow*
- Realities
  - For large data sets, disk accesses dominate runtime
  - Some deep and some shallow BSTs exist for any data

Solutions?

- Keep BSTs shallow?
- Reduce disk accesses even for shallow tree?

To Do

- Start Homework 3
  - Find a partner
- Read chapter 4 in the book

Coming Up

- A bit more Binary Search Trees
- Self-balancing Binary Search Trees
- **Huge** Search Tree Data Structure