

# CSE 332: Data Structures and Parallelism

Fall 2022

Richard Anderson

Lecture 6: Binary Search Trees

# Announcements

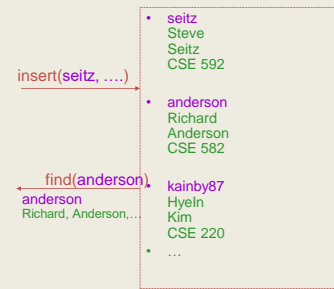
- Reading: Weiss
  - Today: Binary Search Trees, 4.1-4.3, 4.6
  - Wednesday: AVL Trees, 4.4
  - Friday: AVL Trees and B-Trees, 4.7
- Project 1, Due Thursday
- Exercises 3 and 4, Due next week
- Minor change in lecture schedule

# Abstract Data Types seen so far

- Stack
  - Push
  - Pop
- Queue
  - Enqueue
  - Dequeue
- Priority Queue
  - Insert
  - DeleteMin
- None of these support Find(x)
  - Test if x is in the data structure
  - Return data associated with x

# The Dictionary ADT

- Data:
  - a set of (key, value) pairs
- Operations:
  - Insert (key, value)
  - Find (key)
  - Remove (key)



The Dictionary ADT is also called the "Map ADT".

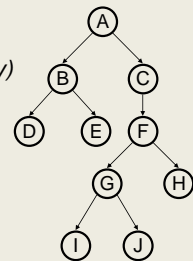
# Dictionary Implementations

- Unsorted Linked List insert find delete
- Unsorted Array
- Sorted Array

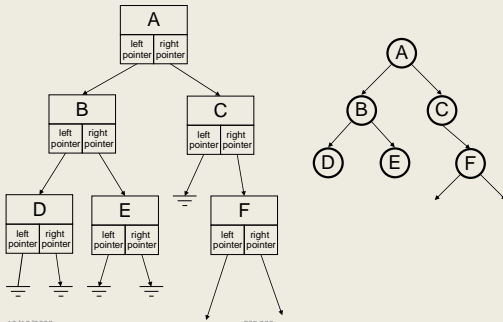
# Binary Trees

- Binary tree is
  - a root
  - left subtree (maybe empty)
  - right subtree (maybe empty)
- Representation:

Data	
left pointer	right pointer



## Binary Tree: Representation



10/10/2022

CSE 332

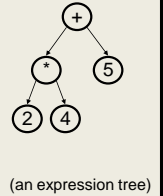
7

## Tree 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



10/10/2022

CSE 332

8

## Inorder Traversal

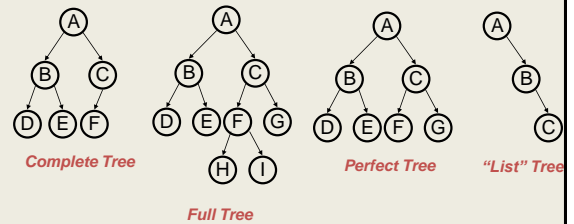
```
void traverse(BNode t) {
    if (t != NULL)
        traverse (t.left);
        process t.element;
        traverse (t.right);
    }
}
```

10/10/2022

CSE 332

9

## Binary Trees: Special Cases



10/10/2022

CSE 332

10

## Binary Tree of height h

Height of a tree: longest path from root to leaf

- max # of leaves:
- max # of nodes:
- min # of leaves:
- min # of nodes:

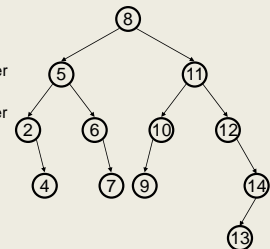
10/10/2022

CSE 332

11

## Binary Search Tree Data Structure

- Structural property
  - each node has  $\leq 2$  children
- Order property
  - all keys in left subtree smaller than root's key
  - all keys in right subtree larger than root's key

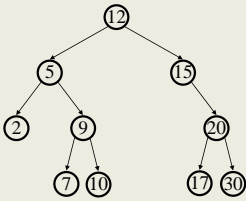


10/10/2022

CSE 332

12

## Find in BST, Recursive



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;
}
```

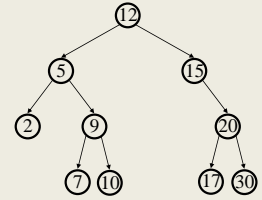
10/10/2022

CSE 332

13

## 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:

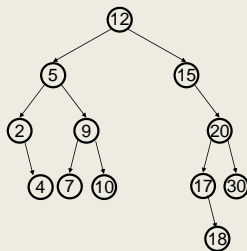
CSE 332

14

## Bonus: FindMin/FindMax

- Find minimum

- Find maximum

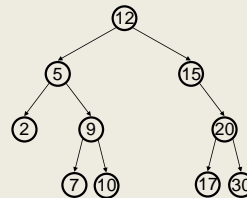


10/10/2022

CSE 332

15

## Insert in BST



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

Insertions happen only  
at the leaves – easy!

Runtime:

10/10/2022

CSE 332

16

## BuildTree for BST

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

If inserted in given order, what is the tree? What big-O runtime for this kind of sorted input?

If inserted in reverse order, what is the tree? What big-O runtime for this kind of sorted input?

10/10/2022

CSE 332

17

## BuildTree for BST

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

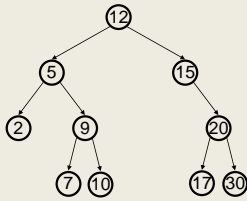
– If inserted median first, then left median, right median, etc., what is the tree? What is the big-O runtime for this kind of sorted input?

10/10/2022

CSE 332

18

## Deletion in BST



Why might deletion be harder than insertion?

10/10/2022

CSE 332

19

## 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

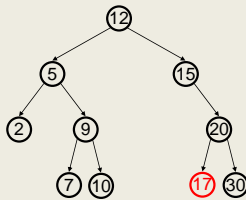
10/10/2022

CSE 332

20

## Deletion – The Leaf Case

Delete(17)



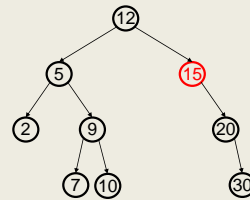
10/10/2022

CSE 332

21

## Deletion – The One Child Case

Delete(15)



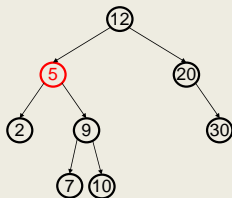
10/10/2022

CSE 332

22

## Deletion: The Two Child Case

Delete(5)



What can we replace 5 with?

10/10/2022

CSE 332

23

## Deletion – The Two Child Case

Idea: Replace the deleted node with a value *between* the two child subtrees

Options:

- *succ* from right subtree:  $\text{findMin}(t.\text{right})$
- *pred* from left subtree:  $\text{findMax}(t.\text{left})$

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

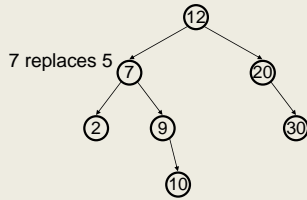
- Leaf or one child case – easy!

10/10/2022

CSE 332

24

## Finally...



Original node containing  
7 gets deleted

10/10/2022

CSE 332

25

## Balanced BST

### Observations

- BST: the shallower the better!
- For a BST with  $n$  nodes
  - Average depth (averaged over all possible insertion orderings) is  $O(\log n)$
  - Worst case maximum depth is  $O(n)$
- Simple cases such as insert(1, 2, 3, ..., n) lead to the worst case scenario

### Solution: Require a **Balance Condition** that

1. ensures depth is  $O(\log n)$  – strong enough!
2. is easy to maintain – not too strong!

10/10/2022

CSE 332

26