Announcements

Assignment#6 ⇒ Due tomorrow

- Should support mazes of low thousands^2
- Looking for completly separate DisjointSet and Maze classes
- Only do num_sets -- when the sets are actually united

Assignments

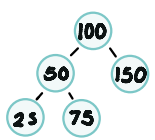
- Only have assignments #8 and #9 left
- Assignment #9 is extra credit

Binary Search Trees (continued)

Traversals ⇒ Touches every node in O(n) time complexity

- We will traverse recursively
- In all three traversals go left then right. Only difference is where x is
- Inorder traversal ⇒ x.left x x.right

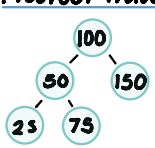
```
//public wrapper, invoke on root
private inorder(Node x) {
    //bounds?
    if(x==null) return;
    inorder(x.left);
    //x - eg print(x)
    inorder(x.right);
}
```



→ 25 50 75 100 150

All values in asending order

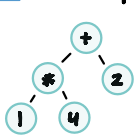
- Preorder traversal ⇒ x x.left x.right



→ 100 50 25 75 150

Not a super useful order given BST

- Postorder traversal ⇒ x.left x.right x
- Traversals on other binary trees ⇒ Example with expression 1*4+2
  - Inorder ⇒ 1*4+2
  - Preorder ⇒ + * 1 4 2
  - Postorder ⇒ 1 4 * 2 +



Insert ⇒ Add a new value to the tree. O(h)

- Assume values inserted ave unique
- New node will be inserted as a child of a leaf node. The position is determined by the values

of the leaf nodes

- Similar to the search() function. Starts at root and works its way down to the leaf

```
//public wrapper (maybe?) may be okay to be public with no wrapper
insert(int value){
    Node node = new Node(value);     //more * if cpp, this is Java style
    Node*prev=null,temp=root;     //node references
    //traversal (no look-ahead)
    while(temp!=null){
        prev=temp;
        if(node.key<temp.key)temp=temp.left;
        else temp=temp.right;
    }
    //temp is null here, because tree is empty or because at end
    node.parent=prev;
    if(prev==null){ //If tree was empty
        root=node;
        return;
    }
    //Attach to left or right
    if(node.key<prev.key)prev.left=node;
    else  prev.right=node;
}
```

Remove ⇒ Delete an element from the tree

- There are several cases, some are easier than others
  1. Node with no children (Easy) ⇒ Delete root
  2. Node with one child (Mid) ⇒ Move child
  3. Node with multiple children (Hard) ⇒ Overwrite other node (successor)

- Code either one function or helper functions (helper function version is in the textbook)

```
private remove(Node node){
    if(node.left==null||node.right==null)target=node;
    else target=successor(node);
    if(target.left!=null)temp=target.left;
    else temp=target.right;
    if(temp!=null)temp.parent=target.parent;
```

* Not on final

```
        if(target.parent==null){
            root=temp;
        }else{
            if(target==target.parent.left)target.parent.left=temp;
            else target.parent.right=temp
        }
        if(target!=node)  //Copy node target into node node
        //delete target
    }
```

Destructor ⇒ For C++ only

- How do you delete nodes safely? Use one of the traversal methods