COP 3502 3/21/2022

(1) Quiz - Returned tomorrow/Fri in Recitation

(2) Binary Trees

Storage/Retrieval Data

Arrays
Bin Search Sorted Array] → finding something O(log n) time, but data can't change.

Linked Lists] dynamic inserts/deletes, O(n) operations not restricted in access

Stacks
Queues] dynamic adding/delete data
fast O(1)
restrictions on what we can delete/look at.

What we want but don't have

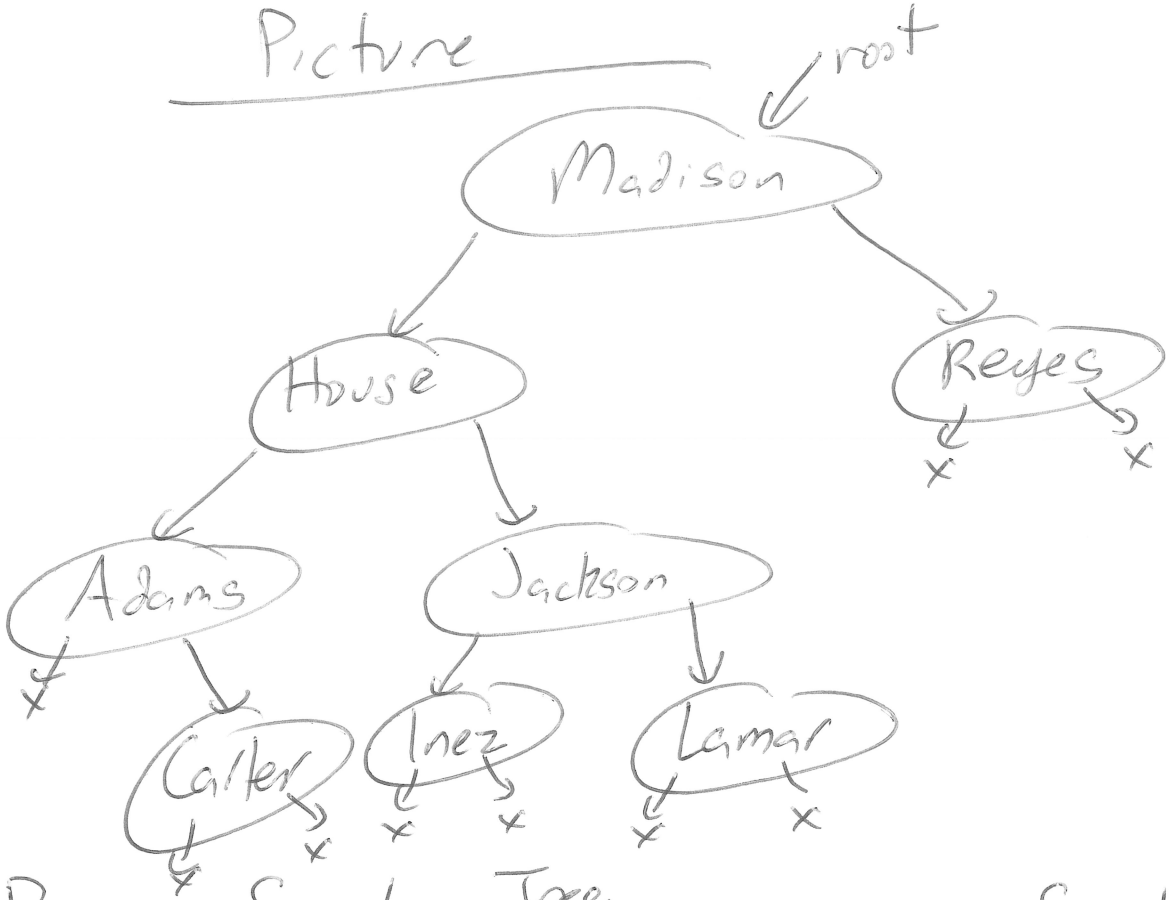
ALL TOGETHER
- dynamically insert/delete data
- fast operations
- no big restrictions on what add or remove.

How to combine ideas:

(1) less → left, greater → right

(2) LL → easy way to add/remove items

Picture



Binary Search Tree

Binary tree

Struct components

1) Data
2) Pointer to left subtree
3) Pointer to right subtree

Search Property

all nodes ≤ root
↳ left side
all nodes > root
↳ right side
TRUE for <u>all</u> nodes

How to <u>traverse</u> each node in a binary tree

3 methods

1. Pre-order Traversal
2. In-order Traversal
3. Post-order Traversal

```
typedef struct bintreenode {
    int data;
    struct bintreenode* left;
    struct bintreenode* right;
} bintreenode;
```

Pre-order

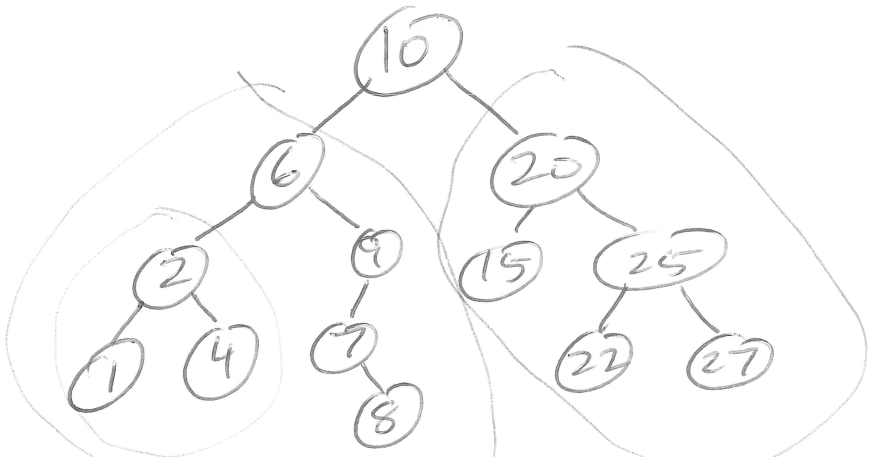
1) Root
2) Left
3) Right

In-order

1) Left
2) Root
3) Right

Post-order

1) Left
2) Right
3) Root



Pre-order: 10, 6, 2, 1, 4, 9, 7, 8, 20, 15, 25, 22, 27

left 6 | right 6

In-order: 1, 2, 4, 6, 7, 8, 9, 10, 15, 20, 22, 25, 27

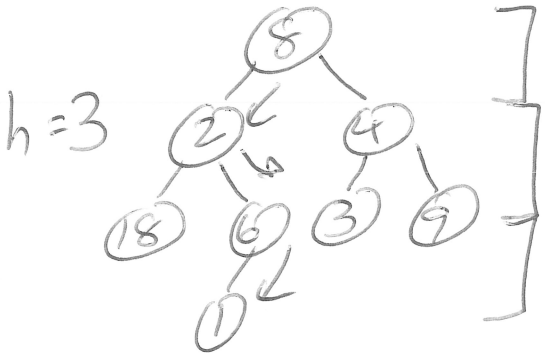
Post-order: 1, 4, 2, 8, 7, 9, ~~15~~ 6, 15, 22, 27, 25, 20, 10

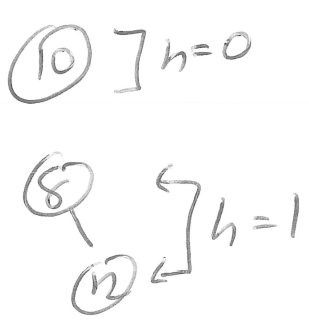
```
Void preorder (bintreenode* root) {
    if (root == NULL) return;
    printf("%d ", root->data);
    preorder (root->left);
    preorder (root->right);
}
```

Given the pre-order and in-order traversal of a binary tree, determine its post-order traversal.

Code w/o changes to a binary tree

Define height: length of the longest path from the root to any node in the tree.






```
int height(bintreenode* root) {
    if (root == NULL) return -1;
    int leftH = height(root->left);
    int rightH = height(root->right);
    if (leftH > rightH)
        return leftH+1;
    return rightH+1;
}
```

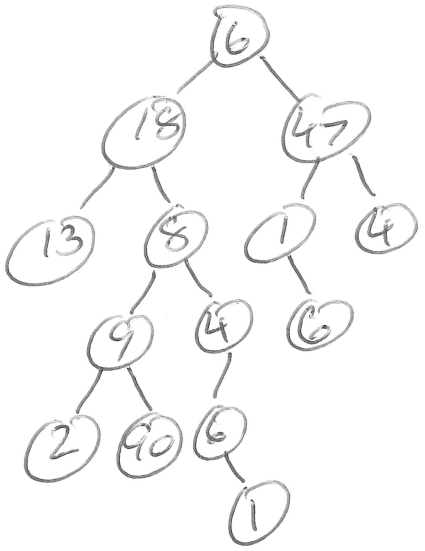
// Take larger of the 2 and add 1.

```
int sum(bintreenode* root) {
    if (root == NULL) return 0;
    return root->data + sum(root->left) + sum(root->right);
}
```

left child -> node to left, right child -> node to right

leaf node is a node w/no children

```
int numLeafNodes(bintreenode * root) {
    if (root == NULL) return 0;
    if(root->left == NULL && root->right == NULL)
        return 1;
    return numLeafNodes(root->left) + numLeafNodes(root->right);
        // root isn't a leaf node!
}
```




```
int maxGold(bintreenode * root) {
    if (root == NULL) return 0;
    int leftG = maxGold(root->left);
    int rightG = maxGold(root->right);
    if (leftG > rightG)
        return leftG + root->data;
    return rightG + root->data;
}
```