Lecture #1 Binary Tree Outline

Definition

How to insert a value into a binary search tree

How to search for a value

Traversals: Preorder, inorder, postorder (exercise)

Live Code: Kattis Problem – ceiling function

Typical tree functions: height, sum of nodes, max/min of a tree

Lecture #2 Binary Tree Outline

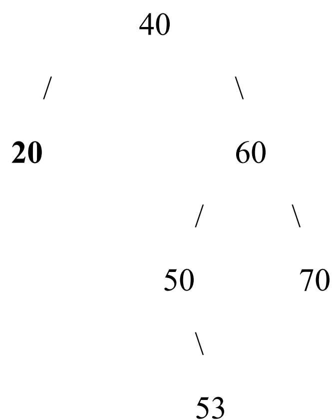
Deleting from a Binary Search Tree

One tracing question from a past foundation exam (May 2025)

Coding Questions – incorporate these and test them (old foundation exams)

Jan 25, Jan 24, Aug 23, Jan 23

Deleting items from a BST



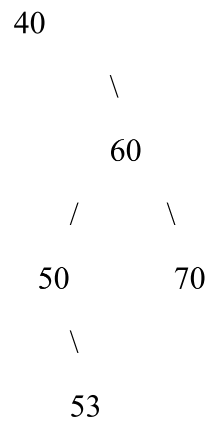
Case 1: Deleting a leaf node

1. Identify the parent node.

2. free the memory for that node

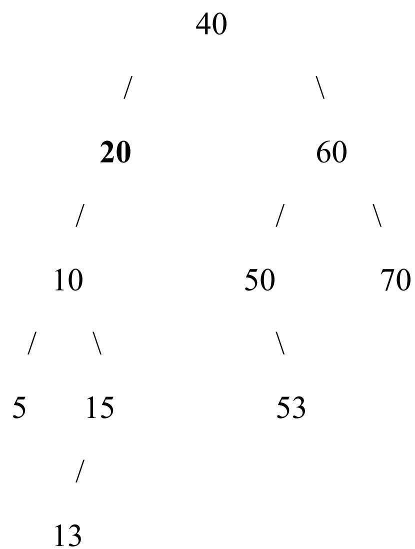
3. Set the appropriate pointer (left or right) of the parent node to NULL

Delete 20, get a pointer to that node and its parent. free the node. set 40's left to null



Note: we must return a pointer to the new root of the tree, because this could change.

Case 2: the node to delete has one child



20 has one child.

ISSUE: Can't set 40's left pointer to NULL, we'll lose a bunch of data!!!

SOLUTION: Make 40's left ptr point to 10 (left child of 20)

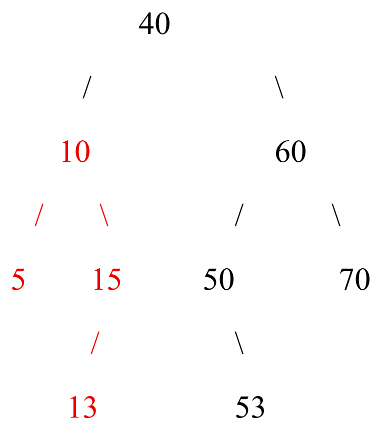
four cases for deleting a node with one child

1. Changing the parent's left child to point to the deleted node's left child. (example above)

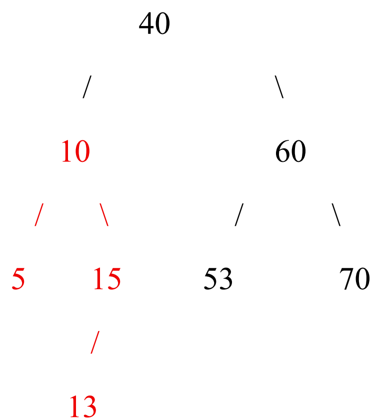
2. Changing the parent's left child to point to the deleted node's right child. (if there were a bunch of nodes in between 20 and 40 above, but nothing below 20)

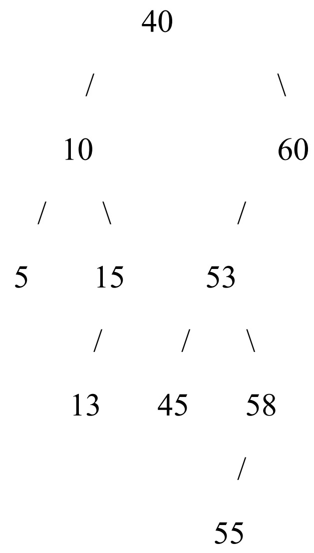
3. Changing the parent's right child to point to the deleted node's left child.

4. Changing the parent's right child to point to the deleted node's right child.

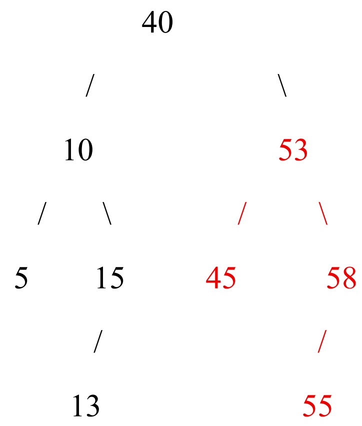


From tree above, delete 50: (case 2)

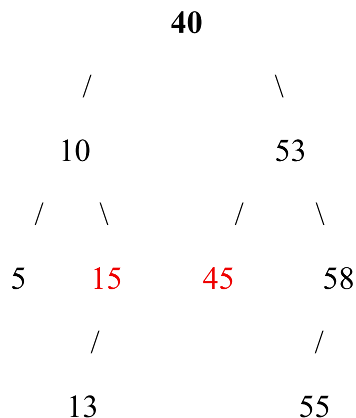




Delete 60 (case 3) Make 40's right pointer point to 60's left pointer:



Deletion in the 2 child case



Consider deleting 40...

Deleting root node is a bit special since it has no parent and our previous code assumed a parent existed...

Consider the fact that we have to maintain the binary search tree property.

What node naturally "fits into the root node?"

Observation #1: The greatest value on the left or the least value on the right, are the only two values that naturally fit into the node's natural ordering if that node value were to be deleted.

Observation #2: The greatest value on the left and the least value on the right, can have AT MOST 1 child

SOLUTION

Do NOT delete the physical node.

Copy the value of the greatest element in the left subtree or smallest element in the right subtree into the deleted nodes value.

Then delete the node where the original value is stored.

The way I code it is

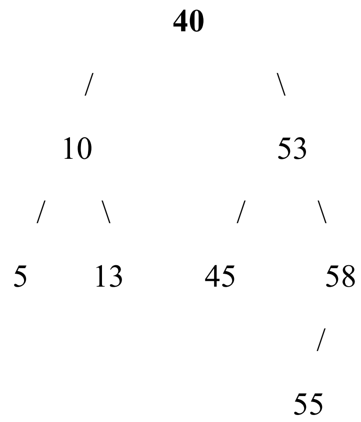
1) Store the greatest value on the left or least value on the right in a temporary variable.

2) Recursively call delete on this value.

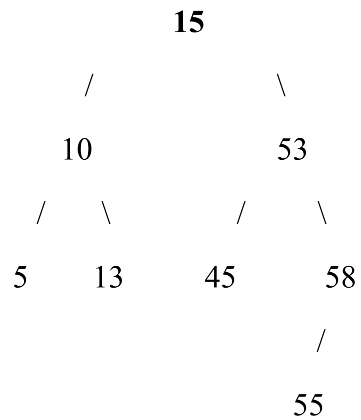
3) Copy my saved value into the original node that was going to be delete.

step 1: Store 15 in a tempvar

step 2: delete 15 (case 3 for one child delete)



step 3: copy tempvar into the original deleted node:



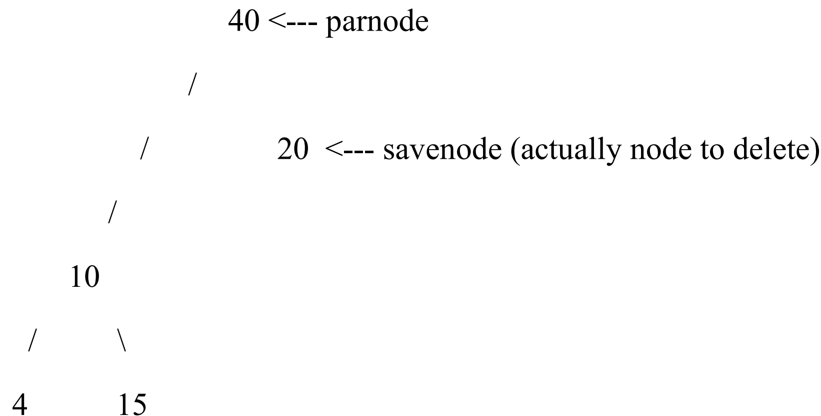
<u>Line 400 trace of bintree.c</u>

```
                20  <--- delnode
           /
        10  <---- savenode
      /     \
    4        15
```

free(delnode)

return savenode; // new root

Line 408 trace of bintree.c



parnode->left = parnode->left->left;

free(savenode)

In real binary search tree code, it's useful to store extra data in the struct

Our struct

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

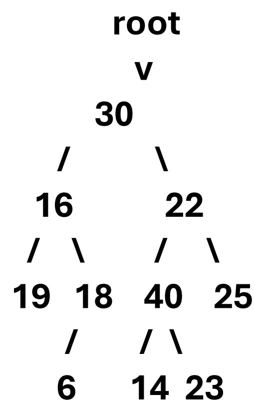
An example what we could do:

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

Note: if we store this extra stuff, we have to maintain it, but if we do, then we have fast access to some other information.

May 25 tracing question

```
#include <stdio.h>
#include <stdlib.h>
typedef struct bintreenode {
    int data;
    struct bintreenode* left;
    struct bintreenode* right;
} bintreenode;

int whatDoesItDo(bintreenode* root) {

    if (root == NULL) return 0;
    if (root->left == NULL && root->right == NULL) return root->data;

    if (root->left == NULL) return root->data + whatDoesItDo(root->right);
    if (root->right == NULL) return root->data + whatDoesItDo(root->left);

    if (root->left->data > root->right->data)
        return root->data + whatDoesItDo(root->left) - whatDoesItDo(root-
>right);

    return root->data + whatDoesItDo(root->right) - whatDoesItDo(root->left);
}
```

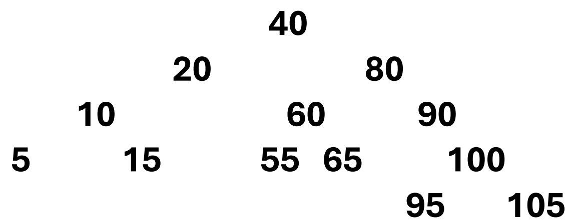


30 + (reccall(22)) – (reccall(16)
30 + (22 + reccall(40) – reccall(25)) - reccall(16)
30 + 22 +(40+23-14) -25 – (16 + 19 – reccall(18))
30 + 22 +(40+23-14) -25 – (16 + 19 – (18+6))
30 + 22 +40+23-14 -25 – 16 - 19 + 18+6

Added: 30, 22, 40, 23,18,6
Subtracted: 14, 25,16,19

Foundaton Exam january 25

New example




```
int sumAtDepth(struct tree_node *root, int depth) {

    // Two base cases.
    if (root == NULL) return 0;
    if (depth == 0) return root->data;

    return sumAtDepth(root->left, depth-1) +
sumAtDepth(root->right, depth-1);
}
```

Foundation Exam January 2024

```
int sumSingleParents(struct tree_node *root) {

    // Get base cases out of the way.
    if (root == NULL) return 0;
    if (root->left == NULL && root->right == NULL)
return 0;

    // See if root needs to be added (ie is a single
parent).
    int res = 0;
    if (root->left == NULL || root->right == NULL) res
= root->data;

    // Just add other nodes below me on left and right
that are single parents!
    return res + sumSingleParents(root->left) +
sumSingleParents(root->right);
}
```

```c
typedef struct node {
    int data;
    int height;
    struct node* left;
    struct node* right;
} node;

void assignHeights(node* root) {

    if (root==NULL) return;

    assignHeights(root->left);
    assignHeights(root->right);

    int max = 0;
    if (root->left != NULL) max = root->left->height;

    if (root->right != NULL &&
        root->right->height > max)
        max = root->right->height;

    root->height = max + 1;
}
```

Foundation Exam January 2023

Prune case we tried:

```
                  40
             20            80
       10             60        90
  5        15       55  65        100
        13            57        95
```

should be answer

```
                  40
             20            80
       10              60        90
         15          55            100
```

code for it:

```
struct tree_node* prune(struct tree_node *root) {

    // Just to be safe.
    if (root == NULL) return NULL;

    // Base case.
    if (root->left == NULL && root->right == NULL) {
        free(root);
        return NULL;
    }

    // Recursively prune nodes on both the left and right,
reassign my left
    // and right accordingly and return me!
    root->left = prune(root->left);
    root->right = prune(root->right);
    return root;
};
```