CS 225

Lab_bst : Beautiful BSTs

Brad Solomon

Overview

In this week's lab you will work with binary search trees, review and implement the fundamentals of the ADT, and use your constructed trees to observe the distinction between 'worst case' and 'expected' performance on real-world datasets.

Pointers vs Reference vs Reference Pointers

Which of the following would change the value of input outside the function? Which functions *can* change the value if you change the contents (not the header) of the function?

pointers.cpp

```
void changeValue(int input){
    input = 10; // Changes value but is local only
    // No way to access original variable
}

void changeValue(int * input){
    input = 10; // Points to address 10 but is local only
    *input = 10; // Would change value
}

void changeValue(int & input){
    input = 10; // Changes value externally

}

void changeValue(int *& input){
    input = 10; // Points to address 10 now externally
    *input = 10; // Would change value
}
```

What are the values of x and y after this program runs?

pointers.cpp

```
void changeValue(int *& input){
    *input = 1;
    input = nullptr;
}
```

```
int main(int argc, char** argv)
{
    int x = 42;

    int* y = &x;

    changeValue(y);
}
```

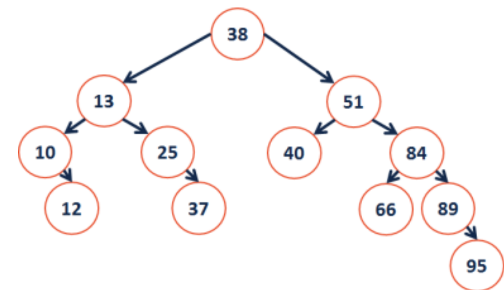
Finding an element in a BST:

What are the return types for find() and _find()?

BST.hpp

```
template <typename K, typename V>

________V___________________________ find(const K & key) {
...
}

template <typename K, typename V>

________Node * &_____________________________ _find
(Node *& node, const K & key) {

}
```

What returns when we call:



Find(25):

A reference to a pointer that points to Node 25. E.g. changing the value changes <Node13>->right

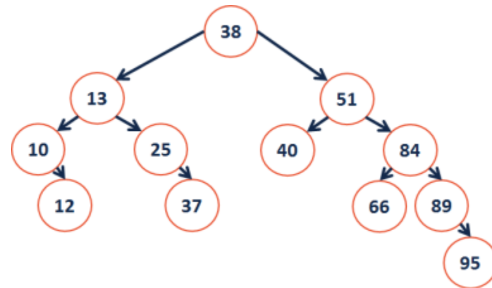
Find(9):

A reference to a pointer that points to nullptr. (<Node10>->left)

Running time? _____**O(n)**_______ **Bound by?** ____**height**____

Inserting an element in a BST:

Draw the changes to the tree when you insert:



Insert(9)

10->left = 9

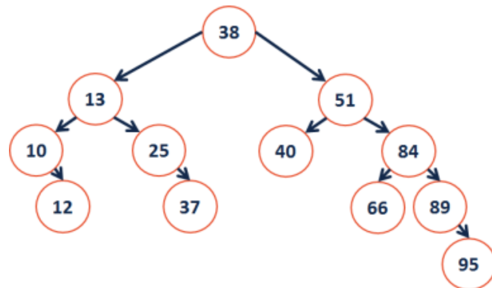
Insert(81)

66->right = 81

Running time? _____O(n)_______ **Bound by?** ____height____

Removing an element from a BST:

Redraw the tree after these changes.



`_remove(40)`

`_remove(25)`

`_remove(10)`

`_remove(13)`

Zero-child Remove	One-child remove
Find node **Delete node** **Set the parent node's pointer to null**	**Find node** **Make temporary pointer to node** **Set the parent node's pointer to node's child** **Delete node**

Two Child Remove
Find node **Find the IOP (or IOS) to node** **Swap IOP (or IOS) and node** **Recursively call remove to delete node in its new location**

The two child remove will always recurse to either a 0 or 1 child remove – why?

Because the in-order predecessor by definition is the right-most left child of the node and will not have a right child (or that node would be the IOP).

The same logic holds for IOS

Running time? _____O(n)_______ **Bound by?** ____height____