Salazar

Assignment 6

Assignment 6 (Binary search tree)(two week assignment)

Completion requirements

Create a design, before you start coding, that shows how your binary tree functions and what attributes it keeps track of to function (yes, you can add to this design once you start coding, but please get some design down to start with and make note of when you add new design features based on your implementation work 🙄),

Create some tests (at least one per function), before you start coding, that you want your Binary Search Tree (BST) to pass as evidence that it would be working correctly if it passed the tests,

Implement a binary search tree that includes:

nodes to store values,

an add function that adds a new value in the appropriate location based on our ordering rules,

(I likely used less than or equal to going to the left and greater than values going to the right)

a remove function that finds and removes a value and then picks an appropriate replacement node,

(successor is a term often used for this)

we have at least one tree traversal function (I recommend starting with an in-order traversal!)

Bonus if you implement the three common traversals (pre-order, post-order, in-order)

More Bonus if you also include a breadth-first traversal (sometimes called a level-order search)

Analyze and compare the complexity of insert and search as compared to a binary tree without any order in its nodes (what is the run-time of an unordered tree...?).

Once you have implemented and tested your code, add to the README file what line(s) of code or inputs and outputs show your work meeting each of the above requirements (or better, include a small screen snip of where it meets the requirement!).

//Desigin

// node struture

// sauce: a string for the hot sauce

// hotness: a value for the hot sauce hotness , changed to shu

// node pointers for back and front. // front = left and back = right,

// start with my base from assigment 3 then made lots of changes from there

// add function to place sauces into tree

// remove function delete elements

// a search function

// a travel function that start from the left to the right.

//requirements

//an add function that adds a new value in the appropriate location based on our ordering rules,

```
// places information in node
void insert(Node*& node, const string& sauce, int SHU) {
    // creates new node when null
if (node == nullptr) {
    node = new Node(sauce, SHU);
    return;
}


if (SHU <= node->SHU) { // checks and compares shu number, if its small it will place in front
    insert(node->front, sauce, SHU);
} else { // back if not
    insert(node->back, sauce, SHU);
}

}
```

//traverse function

```
void inOrderTraversal(Node* node) const {
    //checks if null
    if (node == nullptr){
        return;
    }

    inOrderTraversal(node->front); // loops to it reach the end
    // prints the sauce and shu value
    cout << node->sauce << " (" << node->SHU << " SHU), ";

    inOrderTraversal(node->back); // loops on the back till it reachs the end.
}
```

// remove function

```
class BST {
    Node* remove(Node* node, int SHU) {
        // checks for empty node
        if (node == nullptr) {
        return node;
        }

        // tree search
        if (SHU < node->SHU) {
            //searchs the tree by going towards the front
            node->front = remove(node->front, SHU);

        } else if (SHU > node->SHU) {
            //search toward the back
            node->back = remove(node->back, SHU);

        } else {
            //  nodes with one only front side
            if (node->front == nullptr) {
                Node* temp = node->back;
                delete node;  // deletes
                return temp;

            // same as font but for back
            } else if (node->back == nullptr) {
                Node* temp = node->front;
                delete node; // deletes
                return temp;
            }

            // Node with  both front and back
            Node* temp = findSmallest(node->back); //finds smallest
            node->sauce = temp->sauce; //saves to temp
            node->SHU = temp->SHU; // saves to temp
            node->back = remove(node->back, temp->SHU);
        }
        return node;
```

// search function

```
// search function
bool search(Node* node, const string& sauce, int& SHU) const {
    //checks if null
    if (node == nullptr){
        return false;

    }

    // when it matches it assignes the shu value and returns
    if (node->sauce == sauce) {
        SHU = node->SHU; // assigns value
        return true;
    }
    // the front of the tree
    if (sauce < node->sauce){

        return search(node->front, sauce, SHU);
    }
    // the back of the tree
    return search(node->back, sauce, SHU);
}
```

// test functions

```
// All Test functions
void testBSTTraversal() {
    // startes by puting 10 hot sauces into tree
    BST bst;
    bst.insert(bst.root,"Sriracha", 2200); // meh
    bst.insert(bst.root,"Tabasco", 3500); //okay
    bst.insert(bst.root,"Cholula", 3600); // great
    bst.insert(bst.root,"Frank's RedHot", 450); // sucks
    bst.insert(bst.root,"Crystal", 800);  // n/a
    bst.insert(bst.root,"Valentina", 900); // okay
    bst.insert(bst.root,"Tapatío", 3000); // number one hot sauce
    bst.insert(bst.root,"El Yucateco", 11600); //n/a
    bst.insert(bst.root,"Louisiana", 450); //n/a
    bst.insert(bst.root,"Dave's Insanity Sauce", 180000); // no thanks

    // start by listing hot sauce in how they are traveld
    std::cout << "In-order traversal of hot sauces (sorted by SHU):\n";
    std::cout << std::endl;
    bst.inOrderTraversal(bst.root);
    std::cout << std::endl;
    std::cout << std::endl;

}
```

//search test

```
void testBSTSearch() {
    // startes by puting 10 hot sauces into tree
    BST bst;
    bst.insert(bst.root,"Sriracha", 2200); // meh
    bst.insert(bst.root,"Tabasco", 3500); //okay
    bst.insert(bst.root,"Cholula", 3600); // great
    bst.insert(bst.root,"Frank's RedHot", 450); // sucks
    bst.insert(bst.root,"Crystal", 800);  // n/a
    bst.insert(bst.root,"Valentina", 900); // okay
    bst.insert(bst.root,"Tapatío", 3000); // number one hot sauce
    bst.insert(bst.root,"El Yucateco", 11600); //n/a
    bst.insert(bst.root,"Louisiana", 450); //n/a
    bst.insert(bst.root,"Dave's Insanity Sauce", 180000); // no thanks


    // testing search function and the expected values returned
    int SHU;
    std::cout << "Search for 'Tabasco': " << (bst.search(bst.root,"Tabasco", SHU) ? "Found, SHU = " + to_string(SHU) : "Not Found"
    std::cout << std::endl;
    std::cout << "Search for 'Mega Sauce': " << (bst.search(bst.root,"Mega Sauce", SHU) ? "Found, SHU = " + to_string(SHU) : "Not
    std::cout << std::endl;


}
```

//removal test

```
void testBSTRemoval() {
    // startes by puting 10 hot sauces into tree
    BST bst;
    bst.insert(bst.root,"Sriracha", 2200); // meh
    bst.insert(bst.root,"Tabasco", 3500); //okay
    bst.insert(bst.root,"Cholula", 3600); // great
    bst.insert(bst.root,"Frank's RedHot", 450); // sucks
    bst.insert(bst.root,"Crystal", 800);  // n/a
    bst.insert(bst.root,"Valentina", 900); // okay
    bst.insert(bst.root,"Tapatío", 3000); // number one hot sauce
    bst.insert(bst.root,"El Yucateco", 11600); //n/a
    bst.insert(bst.root,"Louisiana", 450); //n/a
    bst.insert(bst.root,"Dave's Insanity Sauce", 180000); // no thanks


    // test removing hot sauce and checking if it in fact gone on high sauce so back
    std::cout << "Removing 'Tabasco'\n";
    std::cout << std::endl;
    bst.remove(bst.root,3500);
    std::cout << " after removal:\n";
    bst.inOrderTraversal(bst.root);
    std::cout << std::endl;
    std::cout << std::endl;

}
```

// test results