Algorithms Assignment 2

Binary Search Trees

Due Date: Oct. 30, 2024

Max Thursby | 010967047 | mathursb@uark.edu

0 Late Days

Implementation:

The first function i implemented was the Binary Search Tree (BST) deconstructor helper, this was made to recursively search through the BST and set every node to NULL, it starts by taking in a reference to a BSTNode and calls itself on node -> left and node -> right, after which it sets node equal to NULL. Next was the find helper, this first checked if the node was NULL and proceeds to check if the node's key and value match the key and value passed into the function. If that doesn't match it checks if either the key is less than the nodes key, or the key is equal to the node's key, but the value is less than the node's value, then it return calls itself on the initial key value pair and the node -> left. Otherwise, it return calls itself on the initial key value pair again but with the node -> right. Following the find I did the sum help function which just consisted of checking if (node) and returning itself with node -> value + sum help(node->left) + sum help(node ->right). The next were popMaximum and popMinimum which were very similar aside from which way they went but both started with checking the root if it was NULL or not and then create a reference to the root, p, and a reference to NULL, q. This is followed by running a while loop on a bool variable set to false which checks if p -> right != NULL, for popMax and p -> left != NULL, for popMin, inside this it sets q = p, and p = p -> right for popMax and left for popMin. When p -> right or left == NULL, the bool variable is set to true and it moves on to check if p is still equal to the root in which case for popMax the root becomes set to p -> left and right for min, otherwise q -> right = p -> left, q left and p right for min, then p -> left = p -> right = NULL and returns p for both. The penultimate function was the insert help which checks if node is equal to NULL if so if creates a new BSTNode with key and value and 0, NULL, NULL for everything else then returns node. If not, it checks the same as the find except it sets node -> left equal to itself with the initial key value and node -> left for less than and the same with right for greater than and returns node at the end. Finally, I took on the remove helper, same as some others it starts by checking to see if node is equal to NULL and then checks if the key and value are equal to the node's key and value and enters the deletion portion of the helper. It starts by checking if the node has neither a left nor a right subtree, if not it deletes the node and returns NULL. Next if checks if the node has a left subtree and doesn't have a right subtree, it creates a holder reference equal to the left subtree and deletes node and returns the holder reference. Then it does the same thing for if the node does have a right subtree

but not a left, except it points right. The last case of the deletion portion is if the node has both left and right subtrees, for this it creates a reference to the maxValue of the left subtree by calling a reference node to BSTNode = itself -> left until the pointer equaled NULL. Using this point it sets node -> equal to the maxLeft -> key and same for the value. It then calls itself using the maxLeft -> key and value but called on node -> left and the return value is set equal to node -> left. After the deletion portion of the remove helper are the less than greater than checks for the helper to go down the left and right subtrees in respect to the key and value give.

Outputs:

```
max@max-envy-linux:~/Desktop/Homework/Algorithms/HW_2/Alg_hw2$ make
g++ -I./include/ -std=c++11 src/linked_list.cpp src/graph.cpp src/queue.cpp src/stack.cpp src/bfs.cpp src/bst.cpp src/main.cpp  -o bin/main
./bin/main

Perform unit test on your BST implementation
Insert (key, value) = (0, 1) into BST.
        Find (key, value) = (0, 1): Found! Correct!
        Find (key, value) = (1, 2): Not Found! Correct!
Insert (key, value) = (1, 2) into BST.
        Find (key, value) = (1, 2): Found! Correct!
        Find (key, value) = (2, 3): Not Found! Correct!
Insert (key, value) = (2, 3) into BST.
        Find (key, value) = (2, 3): Found! Correct!
        Find (key, value) = (3, 4): Not Found! Correct!
Insert (key, value) = (3, 4) into BST.
        Find (key, value) = (3, 4): Found! Correct!
        Find (key, value) = (4, 5): Not Found! Correct!
Insert (key, value) = (4, 5) into BST.
        Find (key, value) = (4, 5): Found! Correct!
        Find (key, value) = (5, 6): Not Found! Correct!
Insert (key, value) = (5, 6) into BST.
        Find (key, value) = (5, 6): Found! Correct!
        Find (key, value) = (6, 7): Not Found! Correct!
Insert (key, value) = (6, 7) into BST.
        Find (key, value) = (6, 7): Found! Correct!
        Find (key, value) = (7, 8): Not Found! Correct!
Insert (key, value) = (7, 8) into BST.
        Find (key, value) = (7, 8): Found! Correct!
        Find (key, value) = (8, 9): Not Found! Correct!
Insert (key, value) = (8, 9) into BST.
        Find (key, value) = (8, 9): Found! Correct!
        Find (key, value) = (9, 10): Not Found! Correct!
Insert (key, value) = (9, 10) into BST.
        Find (key, value) = (9, 10): Found! Correct!
        Find (key, value) = (10, 11): Not Found! Correct!
Insert (key, value) = (9, 10) into BST.
```

```
Maximum key in BST: 9
        Find (key, value) = (9, 10): Found! Correct! We have two nodes of (key, value) = (9, 10) in BST.
Minimum key in BST: 0
        Find (key, value) = (0, 1): Not Found! Correct!
Sum of all values: 54. Correct!
Remove (key, value) = (1, 2) out of BST.
        Find (key, value) = (1, 2): Not Found! Correct!
        Find (key, value) = (2, 3): Found! Correct!
Remove (key, value) = (2, 3) out of BST.
        Find (key, value) = (2, 3): Not Found! Correct!
        Find (key, value) = (3, 4): Found! Correct!
Remove (key, value) = (3, 4) out of BST.
        Find (key, value) = (3, 4): Not Found! Correct!
        Find (key, value) = (4, 5): Found! Correct!
Remove (key, value) = (4, 5) out of BST.
        Find (key, value) = (4, 5): Not Found! Correct!
        Find (key, value) = (5, 6): Found! Correct!
Remove (key, value) = (5, 6) out of BST.
        Find (key, value) = (5, 6): Not Found! Correct!
        Find (key, value) = (6, 7): Found! Correct!
Remove (key, value) = (6, 7) out of BST.
        Find (key, value) = (6, 7): Not Found! Correct!
        Find (key, value) = (7, 8): Found! Correct!
Remove (key, value) = (7, 8) out of BST.
        Find (key, value) = (7, 8): Not Found! Correct!
        Find (key, value) = (8, 9): Found! Correct!
Remove (key, value) = (8, 9) out of BST.
        Find (key, value) = (8, 9): Not Found! Correct!
        Find (key, value) = (9, 10): Found! Correct!
Remove (key, value) = (9, 10) out of BST.
        Find (key, value) = (9, 10): Not Found! Correct!
Your BST implementation is correct!

Perform unit test on your implementation with graph
Shortest path from 0 to 5 by : 0 1 2 4 5
Total Distance: 13

Shorest path from RSWE to  detination: RSWE -> LMK123 -> LMK124 -> LMK125 -> LMK126 -> LMK127 -> LMK101 -> LMK100 -> PDTF -> GREG -> FSDI -> GIBS -> KIMP -> JBHT -> HILL -> MEEG -> HEAT -> PHYS -> SCEN -> FERR -
> CEAR
Total Distance: 1360
You have to use OpenCV to visualize your map road
```