**Detailed Algorithm and Program Design:**

**Error Checking Class**

--create a bool function to check spacing of file

Check the first array of the string input from file, if the second element isn’t a space return false, else return true

**BSTree Class**

**Create Error Checking Function in Bst Class**

--create a new class in charge of making the bst

In the Header file:

Create a Node Struct:

--Make a struct for the node, the points to the left and right child

--Make a constructer for the node that takes data

--Set value of current node equal to the data

--make left and right nodes null pointers

--make a height int to check the height of the tree

Make a Node Constructer to set the values

Make a right rotating node to rotate a node to the right

Make a left rotating node to rotate a node to the left

Make a Balance number to keep track of each nodes balance

Set the current right and left node and return there difference in height, if right is a null pointer return 1, and if left is a null pointer return 0.

In the BST Cpp file:

Make a height and operation integer and set both to zero

Make a function to get maximum depth

Check if root is null

If not, traverse the tree from root to the left and then right, increment depth each time then return the maximum depth when null is reached

Make a Boolean function to see if tree is empty

If the root is null return true

If not return false

Make a void function to insert a node into the tree that takes a root and int

--If the root is null print the tree has no root to insert! Attempting to create tree root, then create a root and set it to the given value

If root is not empty:

--Compare the root and int, if int is bigger go left of root, else go right and repeat the comparing until we reach a null node, then set the value there

Make a function to print the tree

Check if tree is empty, and If it is tell user , if not, make a prefix for printing the tree that takes a bool to check if the node is on the left, a node, and a prefix, then if the node is on the left print the corresponding graph, if not, print the corresponding graph for nodes on the right, call print tree recursively and check if it is left in the argument, send in the node on the left, and a true, call another print tree but send the node on the right and false

Make a function to start the print tree function

Just call print tree , and send in a empty string, the root and a false

Make a function to search the tree that takes a node and a key int

If the node is false tell user and return a nullptr

Otherwise make a current node that points to the

**Main**

a) open file and check if it is open

i. if the file did not open, display error message and exit

b) get expression from file until the end of the file line is reached

c) call every error checking function from error checking class

i. if any of the function equal false check which

a. if the spacing function equals false, print : “Error: invalid spacing”

b. if the capitalization function equals false, print “Error: inaccurate capitalization”

c. if the parentheses function equals false, print “Error: mismatching parentheses”

d. if the formatting function equals false, print “Error: inaccurate formatting”

e. exit the program

ii. if all functions equal true

1. call the first function from the node class using a node object
2. call the function to build a tree from the binary tree class using a binary tree object. This function will call all the other functions in the class.
3. call the height function from the Binary tree class and set it equal to an integer variable
4. call the printing tree function and pass in the top node and the height.
5. call the numbers class and pass in the postfix expression to the first function.

Create a function to count the performed operation

Create a function to count height

Empty File:

Test with an empty file to ensure the program correctly identifies the file as empty and exits gracefully.

Invalid Spacing:

Input: "A B C"

Expected Output: "Error: invalid spacing"

This test ensures that the spacing check function works correctly.

Correct Format with Multiple Inserts:

Input: "I 5", "I 3", "I 7"

Build a BST with the root of value 5, a left child with value 3, and a right child with value 7. Then, print the structure of the tree.

Mismatched Parentheses in Input:

Input: "(5"

Output: "Error: mismatching parentheses"

Complete Functionality:

Input: Correctly formatted instructions for insertion and a valid expression. After building the BST, it should perform specified operations, outputting the tree and indicating how many operations were carried out in the entire process of balancing the binary search tree, and the height of the tree. This is to validate various aspects of the program right from error checks to the core functions of BST implementation.

Inaccurate Capitalization:

Input: "i 10", "I 20"

Expected Output: "Error: inaccurate capitalization"

Test that follows sets to ensure that the program detects a capitalization error in commands. The small letter 'i' in 'insert' is purposely retained to introduce an error, which would certainly make the program sensitive to cases in commands.

Inaccurate Formatting:

Input: "I10", "I 20"

Output: "Error: inaccurate formatting"

"First, what will come before the argument is an intentional lack of space; this will test if the program enforces space between command and argument. It would have been perfect if the absence of space in "I10" were a formatting error demonstrating the tight format in which the inputs to the commands were supposed to be entered."

Introduction

This report provides an analysis of various test cases designed to evaluate the robustness, functionality, and error handling of a program tasked with constructing and manipulating a binary search tree (BST) based on user input. The purpose of these test cases is to ensure comprehensive coverage of potential scenarios the program might encounter, from edge cases to core functionalities and syntax validation.

Test Case Overview

Empty File: Ensures the program gracefully handles empty inputs without crashing.

Invalid Spacing: Tests the program's ability to accurately parse input commands by identifying incorrect spacing.

Correct Format with Multiple Inserts: Assesses the core functionality of constructing a BST according to specified insertion commands.

Mismatched Parentheses in Input: Evaluates the program's syntax validation, particularly for parentheses pairing.

Complete Functionality: A comprehensive test for end-to-end functionality, including tree construction, operation execution, and reporting.

Inaccurate Capitalization: Checks case sensitivity in command recognition and execution.

Inaccurate Formatting: Tests the program's strict enforcement of input formatting, specifically command and parameter spacing.

Significance of Test Cases

Each test case targets specific aspects of the program, ensuring not only that the BST is constructed and manipulated correctly but also that the program can handle various input formats and syntaxes. These cases are crucial for validating the program’s reliability and usability, emphasizing error handling, input validation, and adherence to BST properties during operations.

Analysis

Edge Case Handling: The "Empty File" test case is vital for confirming the program's stability when faced with unexpected scenarios, such as empty input files.

Input Parsing and Validation: Both the "Invalid Spacing" and "Inaccurate Formatting" cases are essential for ensuring that the program can correctly interpret user commands, a fundamental requirement for accurate program operation.

Core Functionality: The "Correct Format with Multiple Inserts" case directly assesses the program's primary objective—BST construction—highlighting the importance of accurate data structure manipulation.

Syntax Validation: The "Mismatched Parentheses in Input" case emphasizes the program's ability to recognize and enforce correct syntax, crucial for preventing runtime errors.

Comprehensive Evaluation: The "Complete Functionality" case is critical for assessing the integration of different program components and the correct implementation of BST operations.

Input:

12

34

456

67

78

56

7

889

89

0

90

Output:

A black screen with white text

Description automatically generated