Marcellus Hunt

3/21/18

362 Project 2

Project Report

For planning the project, I decided to have three classes. I have a class called Cross Reference that contains the main methods in order to parse the text file and insert it into the tree and print from it. The next class is the tree class where I construct the tree node by using an insert function and this class also holds the function to print out the words in alphabetical order. Then my driver class just calls the functions from the CrossReference class that parses the file and uses the Binary Tree class to implement the tree. First, I’ll go through the CrossReference class.

I created an object from the BinaryTree class to build up the tree from insert but I’ll come back to this a couple times. I created a function called ‘printTable()’ that calls my sortTree function from the BinaryTree class to print out the words in order.

My next function is void ‘parseWords()’ that will read in the file and tokenize each line to get every word and then insert the words into the binary tree. I have an int lineNum that will keep track of the lineNumber the words are on. I declare std::string line that will represent a line being read and std::string word that represents a word being read. I create ifstream inputFile(“example.txt”) which is the file that’s going to be read. While file is open, get a line that can be read in and create stringstream ssWord(line) that will help read in file in next conditional statement. I also increment lineNum because a new line is read every time it’s called. Next conditional statement is while the stringsteram object is read in each string separated by a space, it will insert the words into the the binary tree using the insert function from the BinaryTree class using the tree object I mentioned earlier. In this I also check for punctuations at the end of the strings and erase them if there are any.

Next I move to the BinaryTree class. When constructing a node, it has the elements: string key, vector<int> num, and node \*left and node\*right. It’s public methdos include an insert method and a sortTree function that prints out the elements of the tree in order. In the insert method, I create a new node that holds the values passed to the parameters when its called. If the current node string equals the temporary node string, then it will add that line number to the vector of that node if the numbers aren’t equal. If current string greater than temp string, node point to right holds the insertion of the root pointing to right as long as its not null. If it is then root pointing to right just holds the temporary node. It’s a similar process if the current node string greater than temp string, than node point to left holds insertions of root->left as long as its not null. Else root pointing left just holds temp. At the end of each function call I return the pointer to that node.

For the sortTree function, it prints out the strings in a lexicographic order. If the root of the tree isn’t null recursively call the left side of the tree for values smaller then root node and print it out and then print out the root node and then recursively call the right side of the tree for the larger values. Its algorithm time is O(n) because it goes through every element in the tree. Then to run the program, the Driver file has the int main where it creates the CrossReference object to call parseWords() and then call printTable().