Cross Reference Index Report

Programming Assignment #2

Uladzimir Kasacheuski

3/27/16

Intro

The cross reference index project was a much simpler project than the knight's tour, but it contained some interesting elements, made use of the simple yet brilliantly powerful Binary Search Tree, and is a problem that is a lot more applicable to life: in other words; is more likely to be used than the Knight’s Tour. This problem did not have any significant hitches and was more simply just a rewarding exercise in implementing a binary search tree. Personally, I am very pleased I was presented with the opportunity to practice this implementation.

Background

Binary search trees are incredibly useful things. They were invented in 1960 and have been saving humanity loads of time ever since. The principle of binary search trees is simple. When adding an item to a tree, if the root does not exist : set it to be the root. If the root is smaller than the new node, make the new root the node to the right of the current root and try to put the new node there. If the root is bigger than the node, make the root the node to the left of the current root and try to put it there. The result of these rules is a data structure whose average case insert, find, and delete operations are O(nLogn). A more trivial, but still invaluable, result of these rules is that smallest value of the tree is the leftmost node, the largest is the rightest, and if you wanted to return the results in order from least to greatest (or vice versa) you simple transverse the tree in order (or backwards)!

Project Description

This project gives you a text file and designates a list of 5 assumptions about the text in the file. Given this text file with its certain conditions, we were to create a table that listed each word in the text file in alphabetical order and next to the word, on the same line, print out the line number of each occurrence of that word in the text file. The data structure to be used to store the data while processing it was kindly specified to be a Binary Search Tree. Each node in the

Binary Search Tree was to contain a value and also a pointer to the head of a list of values which would be used to store the line number that each instance of that word was found on.

Project Implementation

This project was implemented in a very simple, although tied together, format. In the end, to run the function all you do is specify the file path of the text file in the main.cpp file (it is set to 0_thefile.txt) as an argument of the constructor for the cInputParser class. The constructor then automatically parses the text file, stores all the data by building the tree and its lists, and then outputs all the data by the table specified in the instructions. Part of me wanted to seperate the cInputParser from the datastructure dsBinarySearchTree for code that was more independent, but it proved to make the code redundantly complex. Additionally, in this method the program is able to take multiple files (just create another cInputParser object with file path specified) and parse them all at the same time.

The cInputParser class is responsible for opening a stream of the text file and grabbing each word from it as well as the word's associated line number. When it grabs the word data it then passes it to it's dsBinarySearchTree object and tells the object to add it. The cInputParser locates each word by running through each line of the text file, and for each line going through the list looking for starting conditions of a word, continue conditions of a word, and ending conditions of a word. By these 3 conditions it is able to successfully and simply locate each word that starts with an alpha character and can contain alphanumeric characters, with a maximum size of 10 characters.

The dsBinarySearchTree class takes each word from the cInputParser and builds a tree from it. Using the rules defined above in the background section it successfully builds a tree and for each word creates a list containing the line number of each occurrence of the word. It is then called upon by the cInputParser class to output the table (done by traversing the tree in-order).

Conclusion

The main takeaways of this project are all about the utility and simplicity of the Binary Search Tree. Being able to implement a Binary Search Tree while we are learning about them will prove to be essential to retaining the knowledge in the future and to being confident in implementing Binary Search Tree’s as well.