**Project Summary**

In Project 6, we practiced the implementation of BSTMap and use it to organize large data files. The goal of the project is to use BSTMap to identify word frequencies in of raddit comments from 2008 through 2016, which allows us to analyze how culture and language changes over time. The implementation of BSTMap ensures that the large set of data is handled in an efficient manner, compared to using other data structure.

BST is a data structure that keep the data in sorted order, and allows fast lookup, input and removal of elements in an order of logN, where N is the total number of elements in the tree.

**Task Solutions**

KeyValuePair

In Lab 6, we created the KeyValuePair class, which holds generic pair of key and value. In the word frequency test, the key was used to store the content of words and the value was used to store the frequency that it appears in the files.

BSTMap

The BSTMap constructs the BST tree and it implements the interface MapSet, meaning that it will have all the functions that exist in MapSet.

The class contains methods (get, put, containsKey) that traverse over the tree and get the information or do operations to the tree.

In order to test the BST tree, I wrote the method printInOrder, which is a recursive in TNode that prints the tree in order.

WordCounter

The WordCounter class takes in a file and creates a BSTMap with String as key and integer as value, so it has a field of a BSTMap and an int field of totalCount.

The analyze method takes in a filename and interprets it line by line with a while loop. Then I created a String array by calling line.split(), which will split the lines into words. Then it will check if the map also contains the key, and if so, it will increment the count of this word by calling the put method of BSTMap. If the word is not in the map, it will call the put function with 1 as the value.

PIC

getTotalWordCount returns the totalCount whereas getUniqueWordCount returns the size of the BSTMap.

Then I wrote the toPreOrderString method in BSTMap, which generates a string that gives a basic structure of the tree. The strategy is similar to the method printInOrder, where a recursive method is called the TNode class. toPreOrderString method of TNode class first adds the KVP of the root to the string, and then adds its left and right, provided that they exist. Since the method is called recursively, it will add tabs to the child of a TNode. Below is a snippet of toPreOrderString method in TNode class.

PIC

And the result printed by toPreOrderString.

PIC

The method writeWordCountFile uses FileWriter to create a new file and first writes out the total word count and then each KeyValuePair representing a word and the number of its appearance. I used the entrySet method of BSTMap to generate an arraylist of entires in pre-order and writes out each element.

Below is the result of writing the wordCountFile of conunttest.txt.

PIC

readWordCountFile rebuilds a BSTMap from a WordCountFile. It handles the first one separately as it contains the total word count. For the rest of the lines, a while loop is used to split the lines into words and put the words as key and counts and values into the BST.

PIC

To test these methods, I created the WCTester class, which has only a main method that does the following:

* Use analyze to load counttest.txt.
* Use writeWordCount to write the results to counts\_ct.txt.
* Use readWordCount to load counts\_ct.txt.
* Use writeWordCount to write counts\_ct\_v2.txt.

If the methods are working properly, counts\_ct.txt and counts\_ct\_v2.txt should be exactly the same. When I checked if they are identical using Unix utility diff, it shows nothing (as shown below) meaning that the documents are identical.

PIC

Analyzing the Reddit files

I first modified the main function of WordCounter to enable it process multiple files at the same time. I used a for loop to loop through each argument in the command line and call analyze on them, and keep track of the time taken for analyzing each file. The for loop also allows me to use the same file-naming scheme for each file as shown below.

PIC

At the end of the for loop, I cleared the WordCount object so that the next file will create a new BST tree. Below is the statistics from analyzing the files.

FORM

Below are the three graphs showing the total word count relative to year, unique word count relative to year, and total word count relative to processing time.

GRAPHS

It appears that the change of total word count and unique word count have similar trend across the eight years, and there’s a positive relationship between total word count and processing time. According to the concept of BST, the time it takes to process the data should be a factor of logN, where N in this context should be the total word count. The scatter plot graph of processing time against total word count does show a trend of such relationship.

**Extensions**

Using an Array to implement a BST

I wrote an array-based representation of a BST in ArrayBST.java. The basic concept is to use an array to hold the root/leaves of a BST in level, and each Node can be indexed using a function.

* The index of the left child of a Node at position n is (2n+1)
* The index of the right child of a Node at position n is (2n+2)

Different from the BST, the array-based representation needs to have space for every possible leaves in the tree. i.e. if a leaf is missing at a level in the middle of the tree, the array still needs to leave a blank position for it.

The class has an object array field, and int field size to keep track of the number of elements in the array, and a comparator field. Its constructor initialized an array of size 10. For the methods put, get, and containsKey, there’s a helper function that handles special cases, such as when the array is empty, and a recursive function that is able to traverse over the array.

For instance, the helper function of put takes in a key and a value, and it first checks if the array is empty, and if so, it will directly create a new KeyValuePair object and add it to the first position of the array.

PIC

The recursive function of put first compare the key with the first element in the array (the root), and similar to the put method of BSTMap, it either updates the value or recursively searches for the left/right of the tree. For example, if the new key is smaller than the key of the first element, a leftIndex will be calculated suing (2n+1), where n is the index of the root. Because the available space of an array is set, the program will then check if the leftIndex is out of bound of the array and double the size if needed.

PIC

If the position indexed by the leftIndex is empty (null), the new KVP will be inserted, otherwise the function will recursively return itself with leftIndex as rootIndex.

PIC

The methods get and containsKey use follows the same algorithm.

An array representation of BST is efficient only if the BST is a complete tree, meaning that there will be no empty position in the array, otherwise there will be a large amount of wasted space because the index will be increased by a factor of 2n to continue searching for a leftIndex. I tried to use the array-based presentation to process the Reddit file of 2008, which is the smallest of the eight files, and the program run into error very quickly because of the lack of space.

toLevelString method that prints tree by level

To print out the tree by level, I used a Queue and first offer it the root of the tree, if it exists. Then the while loop will poll the first element from the queue and offer its right and left children to the Queue if they exist, and the while loop will break when the Queue is empty. In this way, the children of a Node *p* will always be printed after the rest of the Nodes in the same level of *p.* Below is snippet of the while loop.

PIC

remove method for BSTMap

To maintain the structure of the rest of the tree, removing a node from a BST should replace the node to remove with the maximum node on its left subtree (or the minimum node on its right subtree). The method also needs a helper function in BSTMap class, and several recursive functions in TNode class. The helper function checks for empty case and return the recursive remove function in TNode class.

There are several functions in TNode class that helps the final remove function. getMax() returns the largest node that is below the current node. It will be called upon the left child of the node to remove.

removeMax() returns the root of the new tree with maximum node below the current node removed. If the current node as no right child, meaning that the current node is the largest of the brunch, then its left child will be the new root.

Otherwise, the function will call removeMax() on its right child and return the current node.

The actual remove function takes in a key, a TNode *r*, and a comparator. If *r* in is the node to remove, the function handles three cases:

1. If *r* has no child, the function will simply return null (setting *r* to null)
2. If *r* only has one child, the child will be returned and set to the new root in the helper function
3. If *r* has two children, the function will get and remove the largest node of its left subtree and set it to the new root.

PIC

If *r* is not the node to remove, the function will recursively search the left or the right subtree of *r* until it finds the node to remove.

PIC

I tested the remove method in the main function and below is the result printed after removing the node with key “ten”.

PIC

**Conclusion**

This project helps me better understand the concept and implementation of BST and its efficiency in handling a large amount of data. I’ve created a basic structure of BST that can be used in subsequent projects. The remove method I wrote in extension created a challenge that tested and improved my understanding of recursion.

Credits:

Prof. Maxwell’s lecture notes

TA Prashant

Roujia Zhong

Ruize Li