**Introduction to Data Structure and Algorithm**

**Project : Generating Concordance Task**

Group Members: Yeung Tang, 20083706

Tsang Kwok Yiu, 20087857

Gurung Apan, 20092221

**Issues and their solutions while Programming**

For the Generating Concordance Task, we are asked to generate all the positions and frequency for all the words in the corpus, meaning the original text. We are then asked to save the result in a text file. To solve the sorting and counting problem, we decided to use a binary tree to record all the words in the file along with their position in the file. After which the binary tree would be sorted and displayed in alphabetical order.

The first issue we found while trying to sort the problem is the uneven number of new lines and spaces in the main file. It results in difficulties in recording the words from main file and the new lines waste memory space in the tree that was used. In addition, uneven number of spaced in between the words caused difficulties by abnormally increasing the memory needed.

This problem was solved by coding a function that takes every character from the main file and record it in another file. Here, function takes a character from the main file and checks if it is a blank space, new line or a character from a word. In case it is a character then the function records it to the sorted file and enters new line code for the next space or new line. After a new line the function ignores the remaining spaces and new line until a new character is inserted.

The second issue detected was knowing the initial position of the word after its been sorted.

This problem was solved by creating new elements in the structure of the tree itself. For us, we added pointers that record the locations that a word appears in along with the its total count which can be later printed when necessary.

The third issue during programming is the fact that the main file size too large. Because the file contains more than 2,000,000, it becomes a problem in inserting the words in a tree. It results in the tree containing too much branches which causes the program to close on its own due to memory exhaustion. In addition, a few words repeat too much thus resulting in them over crowding the tree thus leading to similar problem.

This problem was solved by using pointer instead of fixed value array and by using malloc() and realloc() features to allocate memory and increase its size.

**Outline of the Algorithm used in the Program**

1)Define the structure of the node

-> char \*value

-> int \*total

-> int \*location

->node \*left tree

->node \*right tree

2)Remove space and random new line

a) get char from main data

b) check if char is space, new line or character

c) input char to sorted data if character

d) input new line to sorted data if char == space or newline

e) stop new line input until a new character is inputted

3) read from sorted data and insert the characters to the tree

a) if root == NULL, insert data in the root

b) if root->word > word to be inputted, go to right leaf and repeat from step A

c) if root->word < word to be inputted, go to left leaf and repeat from step A

d)if root->word == word to be inputted, insert location in its location pointer

4) output the tree in order of left leaf, root and right tree (inorder)

a) print the word stored in the root and its total frequency into the output file

b) print all its location to the output file

c) go to next node

5) close all the files and print “processing complete”.

**Analysis**

The program directly stores the data with only minimum looping. The only looping used are to read through the files, display values from different nodes or to insert value in case the root is occupied, and the word does not equal to the stored word. As a result, the program can sort through all the words in the file in under 5 seconds.

**Reference**

* Sample codes form Dr. Lam Sun Sam

**Problems**

* Location value of words during middle and end segments miss by 1.