Craig Perkins
Alex Tang

README Project 3 – Indexer

To implement the indexer we ended up using the code we wrote for sorted-list to create a sorted-list of sorted-lists. From the analysis of sorted-list project we concluded that insertion time was O(N) for the number of nodes in the list. We also said that it also takes up O(N) memory and that each node only took a total of 16 Bytes, n 8 Byte void pointer and an 8 Byte node pointer. In our project we have 2 structures. We have a struct called Term and a struct called Record. Our term struct contains a char* for the word being indexer and a SortedListPtr to the sorted-list of the filenames and frequencies. In total one term will take up N+8 where N is the size of the string of characters and 8 is the size of a SortedListPtr. A Record struct contains a string for the filename of where a character occurs and an integer for its frequency. The memory required for a Record will be N+1 Bytes, where N is the length of the filename. Also know that a Node in a sorted list takes up 16 Bytes, 8 for the void* to data and 8 for the NodePtr to next. The program also makes use of an iterator when writing to the output file, but the iterator only takes up 16 Bytes of memory for the entire program.

Let's assume that the program has one file for input with the file containing N words. Let's also say that the average length of a word is M and that the filename contains L characters. In the worst case we'll have all unique words, but all of the records will be able to be updated in constant time, O(1), since only 1 file is being analyzed. With 1 file the most costly operation for the program will be the insertion of each word, or Term, into the main sorted-lists of words. The program in this scenario will run in O(N) time for the worst-case because only one file is analyzed and will take N*(M+8+8+sizeof(void*))+N*(L+1+8+sizeof(void*)) Bytes of memory.

Now let's analyze the program for a FileSystem (Directory + Subdirectories). Let's assume that there are N files on the FileSystem, with on average M words per file and each file having an average word length of P. Let's also assume that the length of the average file name is Q. In this scenario insertion into the sorted-list of records will not be constant time, but instead will be O(N) for the number of files in the filesystem. In the worst case, the creation of the sorted-list of sorted-lists will take O(N*M) time. The memory analysis is like that of above, but now we need to take into account all of the files on the filesystem. The program will take up N*(M*(P+8+8+sizeof(void*))+M*(Q+1+8+sizeof(void*))).

One thing I think is creative about our solution to the project, is the we actually have an iterative solution for going through all the files of the filesystem instead of a recursive solution. We make use of the filetreewalk in linux to get a list of the files as a tree view from the parent, which is passed to the program. At first we had problems with the recursion because S_ISDIR was returning the wrong value if the relative path from the processes working directory wasn’t passed in correctly. We ended up resolving the issue, but ultimately decided that we thought the iterative solution was creative and decided to keep it. In the process of resolving the issue we learned a lot more about C and the different functions already available within the language.