Indexer Implementation

Professor Russel
CS 214
Group Members: Krishna Yellayi & Arnesh Sahay

For the Indexer assignment me and my partner decided to implement it using a hash table. The hash table has 36 bins each bin corresponding to a letter or number the “word” begins with. Each bucket is a linked list with each node containing the word and a pointer to another type of node which we called a filenode. This filenode stores the name of the file and the number of occurences that the word appears in the file. This makes the process of extracting the inverted index much easier and adds a level of organization to the code. For the tokenization process we decided to create a function that would create a string of separators given the the string input. In the project direction a delimeter is any character that is not alphanumeric. What we did was parse through the string and store any non-alphanumeric characters into it. This would become our separators string we pass into the TKCreate function. InsertToTable function takes the value of the word that is being returned by the tokenizer and the value of the filename. It then gets the index of the word by calling the getindex function and places the word in the following bucket. If the word already exists in the hashtable it then checks the file node's linked list to see if the filenames are different. If they are different it will create a new file node that will append itself to the linked list. If both filenames are the same, it will increment the value of the occurrences. Writetofile creates a new file pointer to write. The function then parses through the hash table to find all the words, filenames, and number of occurrences. It then writes to the file in its specific format given to by the instructions. The dir_traversal method acts as a file scanner to distinguish between directories and files. It feeds the filenames of legitimate files to the Fparse function. If a directory is detected, the dir_traversal method is recursively called to traverse the directory. The directory time analysis is O(n + (m-1)), considering there are n files and m directories. m-1 because we assume that we are already in the first directory. Files are parsed once and directories are traversed once individually. The worst case for insertToTable function would be O(n + m) when given n files & m words in each file. Appending a node to the end of the linked list for a particular letter would have O(m) since there are m words in each file. However, if the word compared with another word at the end of the linked list is the same, it would have to traverse at most m file nodes in the file node list. Therefore, worst case scenario is O(n+m).