Mohit Mori
Matt Platoff
Systems Programming
Asst2: Recursive Indexing

Description:

Our recursive indexing program takes two arguments, the first one being the name of the file to be outputted, and the second one being either the name of a file or directory to index. Our program will check if the output file already exists, and if it does, the user is given the option to overwrite it or not. Additionally, for the second argument, our program will take either a relative or absolute path for the directory/file.

After all of the input is validated, the program will send the file/directory name to a recursive function to handle directories. This function goes through all directories and files within the directory input by the user. If the directory handler reaches a file, the filename is passed to a tokenize function, which tokenizes all of the words that fit the specs as stated in Asst2.pdf.

As each word is tokenized, it is passed to a function called addEntry which adds the token to a linked list that has another linked list in each node and keeps track of token, filename, and occurrences. addEntry handles the various situations that can occur when a token is being added. (word does not exist, word exists but not for that file, both word and file already exist, etc.) The tokens and files are added in alphabetical order (numbers after letters).

Once all files in all directories have been tokenized and added to the linked list structure, our program goes through each file linked list for each token and uses a stable variant of selection sort to order the file names under each token by occurrence, while maintaining the alphabetical order for files that have the same number of occurrences of the token. Finally, a function, iterate(), iterates through our data structure and prints the results in the specified XML format to the file indicated by the user in the first argument.

Efficiency:

The time complexity for the main functions of the program (indexing function, tokenizing function, and insertion function) is O((kn) + (n^2)) where n is the total number of words across all files, and k is the number of files. This is because when inserting, the program has to check through all words that already exist in the linked list n times. Also, for every word inserted, the program will check through all of the files that also have that word, looking for a match. The time complexity of the final sort function, however is O(nk^2) because for every word (n words), the program has to do selection sort on the file linked list, which has a time complexity in the worst case of O(k^2).

The space complexity of the program is O(nk) where n is the number of words, and k is the length of each word. This is because every time a word is tokenized, space is malloced to fit the length of the word(k). And this occurs n times, thus n*k.