Alexio Mota
Hua Yang
CS 214 Assign 4
ReadMe

We use our own indexer for this program.

A normal hash function is implemented in hash.c and hash.h. In search.c, a tokenizer is used to token the file stream after it is read into memory.

The output file from our indexer is modified so that the first token will be the number of files and the second token will be the number of words. Afterwards, the names of each file are listed and are enclosed by <name></name>. After that, the words are listed with each word followed by 1 or more pairs of numbers. Each number represents the place file in order from how they appeared above the list of words. (EX: 1 is the first file that is read into the filekeys in search.c)

The names of the files are stored in char **filekeys to be used to output the names of which files contains a given terms later on where the user is asked for input.

In the hash structure, the node contains an integer array field which is the size of the number of files and each index in the array is either a 1 or 0. A 1 means the word in the node is present in the file and a 0 means the word is not in the file.

We didn't free all of the memory because it was causing difficulties and due to already being tardy, we decided to just leave it as is.

The searchor method is a max(n, k) runtime where n is the number of words and k is the number of files. Searchand is relatively the same but is slightly faster. They're about linear time.

The method buildFileKeys has a runtime of n where n is the number of files and the buildHash method has technically a runtime of m insertion where m is the number of words being stored but since it's hash it's always constant.

The file result.txt is an example of the output from our indexer so you have a picture of what it looks like.

For our search program, we used a hash map to store the words, and each word points to a sorted list of file names. What used the most memory is the hash map, since there can be more words within a file than there are files.