Christopher Diehl Sandeep C M

The program takes 2 arguments as per specification. Error checking is done to ensure that the inputs are proper.

If the argument is a directory then the function directory_handle is called and else the file handler .

Within the directory_handle function , if the current dir contains another directory , then the new directory is opened recursively. If it is a file , then we call the file handler using the file name.

We divided the program into three different programs: frequencyList.c, fileTokenizer.c, and jsonWriter.c. FrequencyList.c is used to store the tokens, token frequency count, filepaths, and format options. FileTokenizer.c is used to tokenize the file. JsonWriter.c is used to write the frequencyList to the file the user selects.

We use a global array of cstrings to handle the file paths . So when a new file is processed by the program , then the file path is appended to the array . The array is expanded as required within the file_handler func.

Within the file_handler we do error checking and obtain the tokenizer from the file path.
Then we keep on extracting the tokens from the file , using the GetToken function and we add it to the frequency List using the addToList program. The loop ends when we have no more tokens to extract by returning a 0.

The addToList function is from frequencyList , we have a struct called Node which maintains the filename , frequency, token and if the token is trailing, ie the same token is found in another filename.

Within the addtoList func , if the same token is seen multiple times within the file , we increase the frequency on the token. If an existing token is found in a different file , then a node is created to put the filenames in descending order.

A boolean called isTrailing is used for formatting in jsonwriter.c. If isTrailing == 1 then the node is trailing and should have a trailing comma at the end of the record.
If a new token is found , then we add it in a sorted manner in the addToList method call in frequencyList.c

We create the inverted list in memory , then print it out using the jsonWrite function found in jsonWriter.c.
The jsonWrite func ensures that the output file does not already exists and if it does two choices are given to either append to the file or to overwrite the file.

We use fwrite to overwrite the exisiting file by opening with ‘a’, or append to existing file by opening with ‘w’.

RUNTIME AND COMPARISON AND SPACE

-In order to tokenize each file, we iterate through ever character in the file. So O(n) where n is the number of characters in the file. In terms of space used, we only store valid characters, so we use only as much space as the token needs.

-In order to add to the frequency list, at worst we must iterate through every existing token node. So O(n) where n is the number of nodes in the frequency list. The frequency list mallocs space for the node solely, it does not copy the tokens to make its own copy.

-In order to write to the file, we must again iterate through every node in the frequency list. So O(n) where n is the number of nodes in the frequency list. We require a space for every line in the file, but free the space as soon as the line is written into the file.

-The main program, index.c, requires enough space to store the names and paths of the file. We store the paths of the file in index.c rather than the frequency list so we only have to worry about mallocing and freeing a unique filepath rather than having the filepath stored in every node in frequencylist.

-To summarize. The entire program requires enough space to store the filepaths once ,and every token once. We compare every char in every input file with the tokenizer, every token with the frequency list, and then pull from the frequency list to write to the file.