Vaggelis Spithas
sdi1500147

Compile with make. Run as ./minisearch -i input_file -k K. The arguments can be in any order but the input_file should always be after -i and K should always be after -k.

At first I check if the arguments are given properly and if not I terminate the program. If the arguments are right I proceed in counting the lines. For this I use getline and each time I increase a counter until the end of file is reached. Afterwards, I read the file again line by line and this time I split the doc in words and add them in the trie, as well as adding the whole line in the doc map. I also check if the ids are in order at this point. After that, all the structs are made and I read the input from the user to execute the given commands. Each command is implemented with a different method.

-Implementation of the commands
For the search command I get each word of the query and its respective posting list and for each document that the word appears I calcualte its part of the score and add it in the total score. After I do this for all the words I insert all the scores of the docs that are relevant to a heap and I get the top k elements of the heap to print the results.

For the df command with no arguments I traverse the trie. For each node I pass from, I store the character in a list. Each time I find a leaf node I print what is in the list so far and the frequency from the posting list. The traverse is done recursively in a dfs like order since first I go to the first child of each node until no more children are left and then I look the next child of the last one checked and so on. When I return in a previous call I remove the last character from the list. If an argument was given I search for it in the trie and then print the word and the frequency from the respective posting list or 0 if the list wasn't found.

For tf I just search for the word in the trie and if it is found I look for the respective document in the posting list found and print the results.

-Data Structures

Trie: Each trie node is implemented as a couple of pointers. One pointer points to the first child of the respective node while the other points to a next node suggesting the next child of its parent node. It also has a char field to keep the character and a pointer to the type of data it is storing. The trie has a pointer to a trie node which is the root. All of root's children are the starting letters of the words inserted.

Doc Map: The doc map is a consisted of 2 arrays. An array of char* that holds all the docs and an array of ints that holds the numbers of words that are in each doc. The arrays have the same size and in the same position of each array is the data for the same doc

Posting List: The posting list consists of an integer that counts in how many docs a word appears and a list of the documents. The lists consists of a couple of integers. One which is the id of the doc and one which is how many times this word appears in the respective doc. There is a cur filed in the posting list that indicates the current node of the list we are looking. This is used in on order to help us iterate all the nodes of the list.

Heap: I have also implemented a max pairing heap. The heap is consisted of nodes and each node is consisted of a couple of pointers. One that points to its child and one that points to a next node suggesting that it’s the next child of the parent node, same as the trie. Each node also has a cost field which is the field that is used to sort our elem fields. Some ideas for the heap code where taken from here:https://github.com/saadtaame/pairing-heap.

List: I also implemented a double linked list that is used in the df. More specifically when I traverse the trie I add each character in the end of the list and when a leaf is found I print eveything from the beginnig. It is double linked so it’s easier to remove an element from the end. It only holds char elements though since it was done in a rush.