Search

Program Usage:

```
./search <file name>
sa <term> …: “logical AND” query.
so <term> …: “logical OR” query.
          q : exit program.
```

Inverted Index Format:

- <list> term
  path1 wc1 path2 wc2 path3 wc3
  <\list>

  OR (the end tag)

- <list> term
  path1 wc1 path2 wc2 path3 wc3
  </list>

- The program also takes care of any number of spaces and new lines between any of the terms, filenames, or wordcount by tokenizing everything
- It is also set to catch illegal formats (nonvalid tags, missing words, any of the paths, or any of the wordcounts)

Program Description:

- Given a valid inverted index file, *search* will load the file into memory and use it to answer users’ search queries.
- Query command descriptions:
  - sa <term> …: If there are more than 1 term, the search tool returns only files that contain all terms in the query.
  - so <term> …: If there are more than 1 term, the search tool returns all files that contains any of the terms in the query.

Program Analysis:

- *Search* reads an inverted index file into memory, storing the unique words into a hash table.
- A file is stored into the hash table by storeKeys() and depends on the file size [O(n) time].

- Once all the tokens are processed, it continuously polls for user queries only until **q** is entered, if an illegal entry is given it assumes it's a mistake and keeps prompting the user for input for a new query.
- A master list is used to keep track of the index list for the given query, it is dynamically allocated so it does not alter the hash table list. The master list is freed at the end of every query and prepared for the next query.
- For a "logical AND" query, getValue() searches the hash table for the given first term, if found, the inverted index list is copied to the master list. If any of the given terms to search for are not found in the hash table, the program prints an error and exits. If there are multiple terms the master list, which contains the indexes from the first term, are checked with the index list from the other terms, if not found, that index is removed from the master list. After all terms are searched and compared, the remaining indexes are those that ALL of the given terms have in common. If the master list is empty, the given terms have no indexes in which all of them appear. This query takes O(n^2) time. And also O(n) memory depending on the size of the index list of the first term.
- For a "logical OR" query, getValue() searches the hash table for the given term, if found, the inverted index is searched for that term. Next, the indexes associated with that term are added to a master list and for each remaining term given, it compares the indexes of that term with the indexes in the master list. If an index for the current term is not found in the master list, it is added. After all of the terms given are searched and compared, the master list consists of all the indexes in which any term appears. If the master list is empty, then none of the terms were found in any file with in the inverted index. This query takes O(n^2) time. And also O(n) memory depending on the size of all the unique indexes of all the lists.
- Lastly, freeing memory allocated for the hash table and all its nodes. [O(n^2)]

Data Structures:

- Hash map used to store a list of unique tokens – O(1) to store, O(1) to find key and get list
- Linked list is used to store a list of indexes – O(n) to insert, O(n) to find and remove indexes

Error Handling (listed in test plan):

- Illegal Number of arguments -> Print illegal number of args error, prints usage and exits
- Invalid/non-existing file (if FILE * is null when trying to open it)-> prints error returned by fopen and exits
- Empty inverted index file (caught by the sizeOfFile function)-> prints file is empty error and exits
- Non empty file that contained only delimiters (check if hash was null) -> prints hash is empty error and exits
- Inverted Index file in the wrong format (didn't recognize one of the tags, missing wordcount, pathname, or missing words (empty space after tag)). This is caught while tokenizing the file, if not in the proper order atoi will attempt to convert a string to an integer and return 0, which is not a valid wordcount -> Prints invalid format error an exits
- Invalid query entry -> prints invalid option, prints usage and prompts the user for another query
- Query larger than buffer size (check if query length is equal to MAXQUERYSIZE-1)-> warns user that list might not be accurate (tokens may have been cutoff) tells him where buffer size could be increased and asks user if he would like to continue.
- SA query with a word that is not in the hash -> terminates query with a message that word didn't exist, says list is empty and asks for new query

Outside Contributions:

- UTHash – Hash table for C structures
- Author: Troy Hanson
  Source: https://github.com/troydhanson/uthash

Authors:

- Mauricio Trajano
- Paul McNeil