

Title: InvertedSearch

Brief:	The purpose of storing an index is to optimize speed and performance in finding relevant documents for a search query. Without an index, the search engine would scan every document in the corpus, which would require considerable time and computing power.
Prerequisite:	<ul style="list-style-type: none">• Hashing• Single linked list
Level:	Complex

Introduction:

An inverted index is an index data structure storing a mapping from content, such as words or numbers, to its locations in a database file, or in a document or a set of documents. The purpose of an inverted index is to allow fast full text searches, at the cost of increased processing when a document is added to the database. The inverted file may be the database file itself, rather than its index. It is the most popular data structure used in document retrieval systems, used on a large scale for example in search engines.

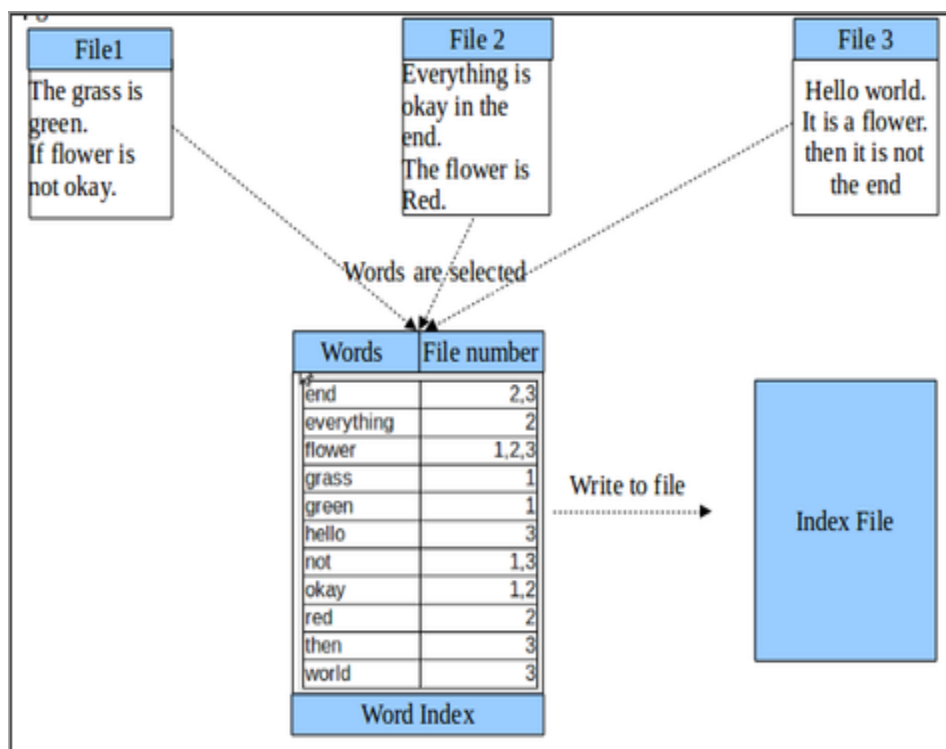
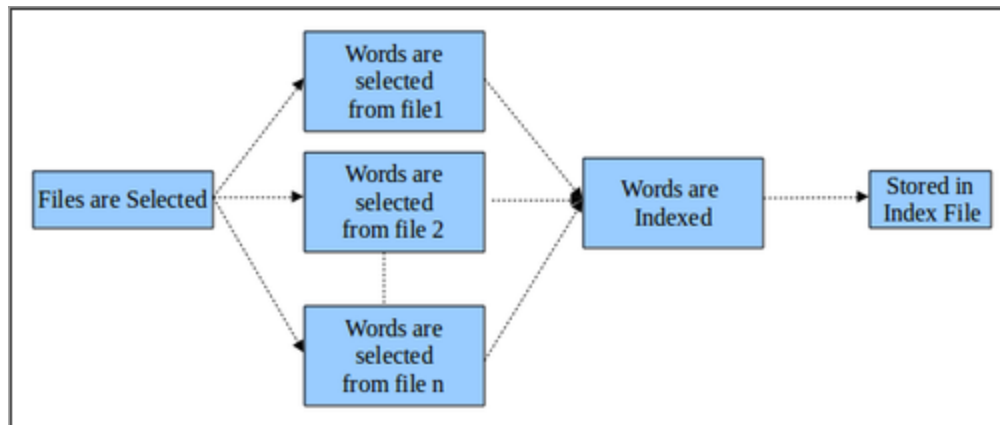
Requirement Specification

Implementing this search program mainly consists two important functions.

1. Indexing
2. Querying.

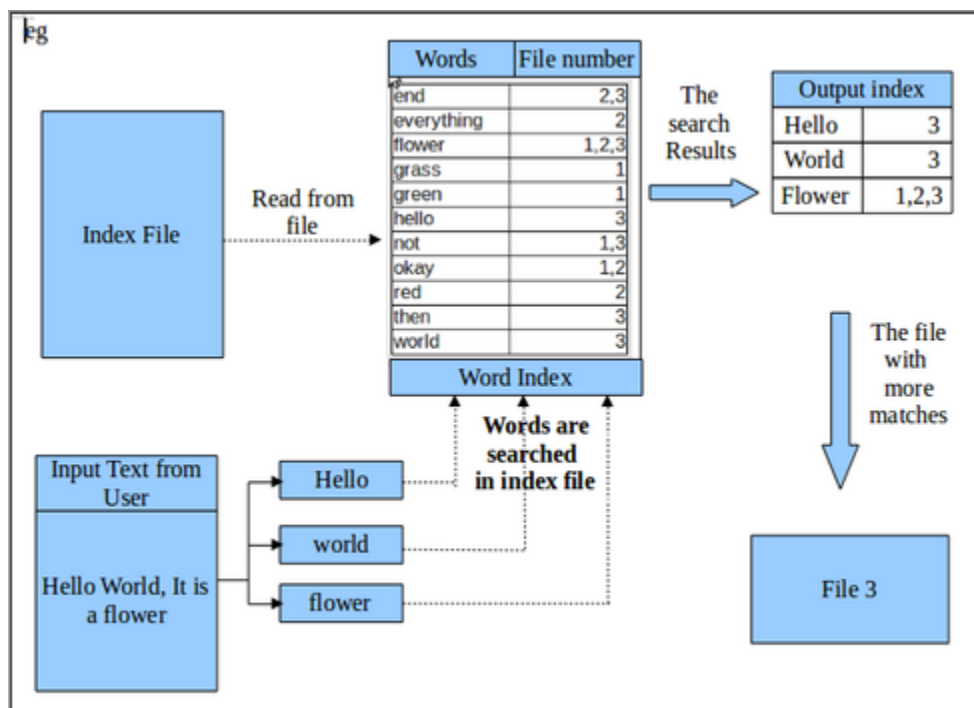
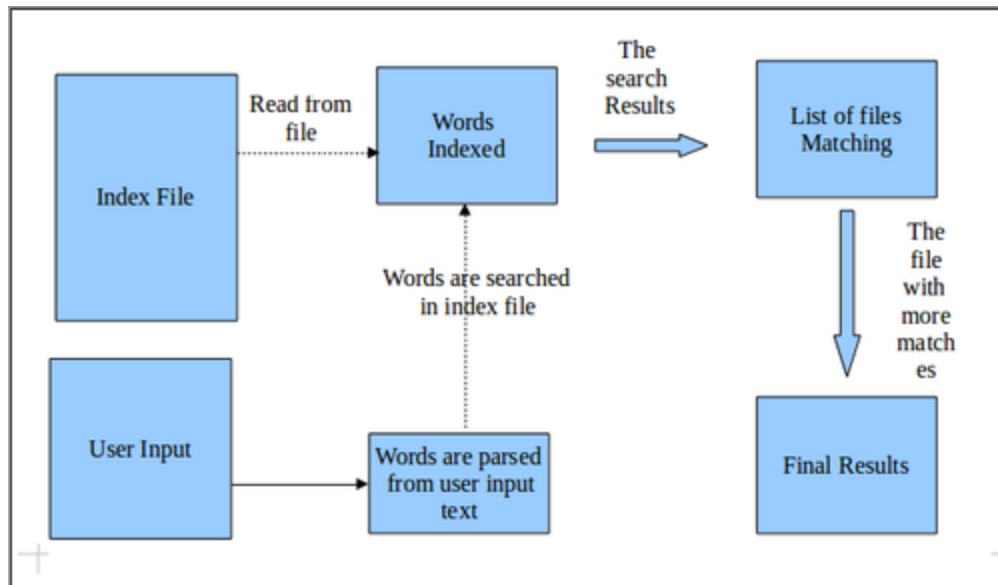
Indexing

By Indexing, we are creating a database file which contains the index of all words. So this can be termed as Database Creation also. All the files whose index are to be created are selected and inputted to this function. All the files are parsed and words are separated and indexed. They are arranged in sorted order. For this a sorted Linked List or Hashing is used which will store the words and the related file details. The index thus created is then stored in the file as database. This file is later used in Querying. While the files are removed or added this index file is updated.



Searching:

Once the Indexing is over we have the Querying or Searching. The text to be searched is inputted which is parsed into words and those words are searched in the index file. To avoid the overhead of reading the file again, the file is converted back to a linkedList or hashing program, in which the words are searched. The information about the files which contain the words are collected. The ones with more matches are filtered and produced as the result.



Design:

Module Level

Module 1

File1- Main_Indexing

Operation: Main file

Functions: This Main program calls the Indexing function. It reads a text file for the input files and as an output it creates an Index file in the same directory. For updating the database also this program can be called anytime.

```
int main ( int argc, char** argv );
```

File2- Indexing

Operation: Create/Update Database(Index) file

Functions:

1. Select the Files from a text file

The list of the files can be provided by storing all the file names in another file, FileList. So the names of the files which are to be documented are provided by this file. When a file is added or removed, FileList is changed accordingly. So read the file names and start indexing.

```
create_database( FILE* FileList);
```

2. Read file one by one

Open the file "FileList" and read the names of the files. This is an iterative process which reads one file name from the FileList and open that file in read mode.

```
word* read_datafile ( FILE *file, Word *WordList, char* filename)
```

Now the process of parsing starts.

1. Select words : Character by character are read from the file. When a character other than alphabet is read it is considered as the delimiter of words. So the alphabetic characters form words. When one word is collected it now send for storing. For space reductions avoid prepositions (in, is, from, etc), conjunction (and, for etc), articles (a, an, the) etc. In text files the occurrence of these words will be more.
2. Store in the index list along with file name in sorted order: Index List can be stored using Linked List or Hashing. Here we are implementing with LinkedList. The word is now stored in the linked list along with file names. So the Linked List is a structure storing a) word b) list of filenames c) pointer to next list. The word is inserted in the list in sorted order.

For arranging in the sorted order the words in the list are string compared with the new word to be inserted. Before inserting the word, make sure the word is not repeated. If the word is already present, select the same word node and add the file name in the file list. For the samewords in the same file no need of adding the same file name again. (instead for advanced search, you can increment the count of that word in the same file. For this the node should contain a count member which reflects the weightage of each words in each file.)

- Once the file reaches the end, it can be closed. And the next file file in the file list can be opened in read mode and process continues. This is done for all the files in the list. So When the files are over a very big index (linked list) is formed.

```
word* store_word (char* word, char* filename, Word* WordList);
```



- Write the index list in the Output Index File : All the files are opened and read. Words are listed. Now this list is to be stored. For this a new file “DatabaseFile” is created and opened in write mode. Now the words along with filelist (each node contents) are written to the Database file. In the file also the word should be written in sorted order. After that the file is closed properly.
- ```
write_databasefile (Word* WordList, FILE* Databasefile);
```

## Module 2

### File1- Main-Searching

This program can be called whenever a search is needed. This program takes user input texts and produces file names in return.

### File2- Searching

- Write the Index file to a index list  
Before starting ‘search phase’, open the Database File and read the word structure and store it in a LinkedList. This helps to avoid the overhead of accessing files from file always will be reduced. So open Database file in read mode, read each word& related files info, and store in a LinkedList node. Read till the file is completely converted to LinkedList. Make sure the sorted order is stil maintained.  

```
Word* create_indexlist (FILE* Index_file, Word* Index_List); return Index_List
```
- Take input from user  
Now the search phase is going to begin. To search, collect the user input text. It should be stored in a character string.  

```
Char* input_from_user(); returns user input string.
```
- Divide it into words  
The character string can be tokenized into words . It can be done in the same way, the file is parsed. The non-alphabetic characters can be used as the delimiter.  

```
Char* string_token (char* input_string); return words.
```
- Search
  - Search each word in the list.  
Each word is now used to search in the linkedlist. The word has to be compared with each word in the linked list. When found, the file details can be retrieved. Since the linkedlist is in sorted order, the complexity of searching the word in the complete list can be avoided. While string comparison, if the word in linkedlist is greater than the word to be search, it shows the word doesnt exists in the list,
  - Create an output index, based on words matching.  
For the matched words an output index is need to be created. For this an array or

another temporary linked list can be created. In this, file name and a count is stored. For each word matched, the corresponding file's count is incremented.

```
Struct _file { char* filename; int count; struct _file next;}File;
```

```
Search (char *word, Word * IndexList, File* Output_list); returns Output_list
```

5. Select the files with maximum matching and print the results in the order.

When all the words are searched the output list is formed. In this the file with max count (representing maximum word match ) can be selected and displayed. For advanced searched, (weightage of words stored in database linkedlist) can also be used. So the results will be more accurate.

```
display_output (Output_list);
```

## References

1. [https://en.wikipedia.org/wiki/Inverted\\_index](https://en.wikipedia.org/wiki/Inverted_index)
2. <https://nlp.stanford.edu/IR-book/html/htmledition/a-first-take-at-building-an-inverted-index-1.html>
3. <https://www.ijcsi.org/papers/IJCSI-8-4-1-384-392.pdf>
4. <https://www.elastic.co/guide/en/elasticsearch/guide/current/inverted-index.html>
5. <https://www.quora.com/Information-Retrieval-What-is-inverted-index>