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 a 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. The purpose of this project is to implement the inverted search using Hash Algorithms.

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 a 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.

Forward Indexing:

Creating the initial word list requires several operations. First, the individual words must be recognized from the text. Then these words can be stored using a dynamic Linked List or Hashing. Along with the words the information about the corresponding document or file should also be stored . Like that

all documents in the database are indexed. Words which are repeated in different or same files are not indexed separately, but the same word is updated with a list of those file names, in which they occur. So each word in the list is mapped with all the files in which it contains. The detail of the files can be stored as URL or as names etc.

Inverted Indices:

The forward index is sorted to transform it to an inverted index. This can be done with the help of the best sorting method. The list is arranged in sorted order of words. This will help to search the words easily in the list and produce the information about the documents in which they are present.

Most computers cannot sort the very large disk files needed to hold the initial word list within a reasonable time frame, and do not have the amount of storage necessary to hold a sorted and unsorted version of that word list, plus the intermediate files involved in the internal sort.

This can be avoided by implementing Hashing or sorted Linked list or Binary Search tree at the time of Indexing. While Indexing, before storing,the words are compared and arranged in the sorting order. This helps to reduce the time complexity in searching for a word through the list. The efficiency can be increased by implementing Hashing, which stores words based on unique indices. So while retrieval of words also the efficiency can be obtained. This index can only determine whether a word

exists within a particular document, since it stores no information regarding the frequency and position of the word; it is therefore considered to be a boolean index.

Such an index determines which documents match a query but does not rank matched documents. In some designs the index includes additional information such as the frequency of each word in each document or the positions of a word in each document. Position information enables the search algorithm to identify word proximity to support searching for phrases; frequency can be used to help in ranking the relevance of documents to the query. Such topics are the central research focus of information retrieval.

Requirement Details

Implementing this project mainly consists two important functions:

1. Indexing
2. Querying

Let us take a look into them in detail.

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 input 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 a database. This file is later used in Querying. While the files are removed or added this index file is updated.

Once the Indexing is over we have the Querying or Searching. The text to be searched is input 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 linked list 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.

Sample Output

Here are the sample output expected by the end of project execution.

```
user@emertxe] gcc Inverted_Search.c -o Inverted_Search
user@emertxe] ./Inverted_Search
1.Create DATABASE
2.Display Database
3.Update DATABASE
4.Search
5.Save DATABASE
Enter your choice: 1
Enter the file name: input.txt
Do you want to continue??? Enter Y/y to continue: y
1.Create DATABASE
2.Display Database
3.Update DATABASE
4.Search
5.Save DATABASE
Enter your choice: 1
Enter the file name: input_2.txt
Do you want to continue??? Enter Y/y to continue: y
1.Create DATABASE
2.Display Database
3.Update DATABASE
4.Search
5.Save DATABASE
Enter your choice: 2
-------------------------------------------------------------------------------------------
[4]  [Emertxe] 2 file/s:File:input.txt 1 timesFile:input_2.txt 1 times
[Embedded] 2 file/s:File:input.txt 1 timesFile:input_2.txt 1 times

[8]  [is] 1 file/s:File:input.txt 1 times[India's] 1 file/s:File:input.txt 1 times

[15]  [pioneer] 1 file/s:File:input.txt 1 times

[18]  [Systems.] 1 file/s:File:input.txt 1 times[Systems] 1 file/s:File:input_2.txt 1 times

-------------------------------------------------------------------------------------------
Do you want to continue??? Enter Y/y to continue: y
1.Create DATABASE
2.Display Database
3.Update DATABASE
4.Search
5.Save DATABASE
Enter your choice: 4
Enter the word you want search: Emertxe
String Emertxe is present in 2 file(s):
In File: input.txt 1 time(s).
In File: input_2.txt 1 time(s).
Do you want to continue??? Enter Y/y to continue: y
1.Create DATABASE
2.Display Database
3.Update DATABASE
4.Search
5.Save DATABASE
Enter your choice: 5
Enter the file name to save database: backup.txt
Database is saved
Do you want to continue??? Enter Y/y to continue: n
user@emertxe]
```

Fig1: Create and Search the Database

```
user@emertxe]
cat backup.txt
#4;
Emertxe;2;input.txt;1;input_2.txt;1;#
Embedded;2;input.txt;1;input_2.txt;1;#
#8;
is;1;input.txt;1;#
India's;1;input.txt;1;#
#15;
pioneer;1;input.txt;1;#
#18;
Systems.;1;input.txt;1;#
Systems;1;input_2.txt;1;#
user@emertxe]
```

Fig2: Validating the created Database

```
user@emertxe] gcc Inverted_Search.c -o Inverted_Search
user@emertxe] ./Inverted_Search
1.Create DATABASE
2.Display Database
3.Update DATABASE
4.Search
5.Save DATABASE
Enter your choice: 3
Enter the file name: backup.txt
Database is updated
Do you want to continue??? Enter Y/y to continue: y
1.Create DATABASE
2.Display Database
3.Update DATABASE
4.Search
5.Save DATABASE
Enter your choice: 2
-------------------------------------------------------------------------------------------
[4]  [Emertxe] 2 file/s:File:input.txt 1 timesFile:input_2.txt 1 times
[Embedded] 2 file/s:File:input.txt 1 timesFile:input_2.txt 1 times

[8]  [is] 1 file/s:File:input.txt 1 times[India's] 1 file/s:File:input.txt 1 times

[15]  [pioneer] 1 file/s:File:input.txt 1 times

[18]  [Systems.] 1 file/s:File:input.txt 1 times[Systems] 1 file/s:File:input_2.txt 1 times

-------------------------------------------------------------------------------------------
Do you want to continue??? Enter Y/y to continue: y
1.Create DATABASE
2.Display Database
3.Update DATABASE
4.Search
5.Save DATABASE
Enter your choice: 1
Enter the file name: input_3.txt
Do you want to continue??? Enter Y/y to continue: y
1.Create DATABASE
2.Display Database
3.Update DATABASE
4.Search
5.Save DATABASE
Enter your choice: 2
-------------------------------------------------------------------------------------------
[2]  [C] 1 file/s:File:input_3.txt 1 times

[4]  [Emertxe] 2 file/s:File:input.txt 1 timesFile:input_2.txt 1 times
[Embedded] 3 file/s:File:input.txt 1 timesFile:input_2.txt 1 timesFile:input_3.txt 1 times

[8]  [is] 1 file/s:File:input.txt 1 times[India's] 1 file/s:File:input.txt 1 times

[15]  [pioneer] 1 file/s:File:input.txt 1 times[program] 1 file/s:File:input_3.txt 1 times

[18]  [Systems.] 1 file/s:File:input.txt 1 times[Systems] 1 file/s:File:input_2.txt 1 times

-------------------------------------------------------------------------------------------
Do you want to continue??? Enter Y/y to continue: n
user@emertxe]
```

Fig3: Updating the Database