Text File Search Engine

Group 5

April 17, 2015

Group members :	Raghav Somani	130123029
	Prakhar Shukla	130123047
	Vaibhav Adlakha	130123041
	Madhur Bhattad	130123011
	Akshit Jain	130123005
	Sithal Nimmagadda	130123023
	Shivanshu Chouhan	130103084
	Ravi Teja Reddy	130123017
	Saurabh Agrawal	130123035
Instructor :	Gautam K Das	

1 Objective

- The objective of this program is to make a text file search engine which sorts the names of the text files in descending order according to their occurrences.
- The main idea of this project is to hunt for the best algorithm and data structure to serve the purpose. An analysis of 6 different data structure and techniques have been done to make out a satisfactory relation between asymptotic theoretical analysis and experimental data of space and time complexities.
- The program takes the help of efficient inbuilt basic data structures and algorithms provided by C++ in their Standard Template Library which is inbuilt in the recent versions of C++11.

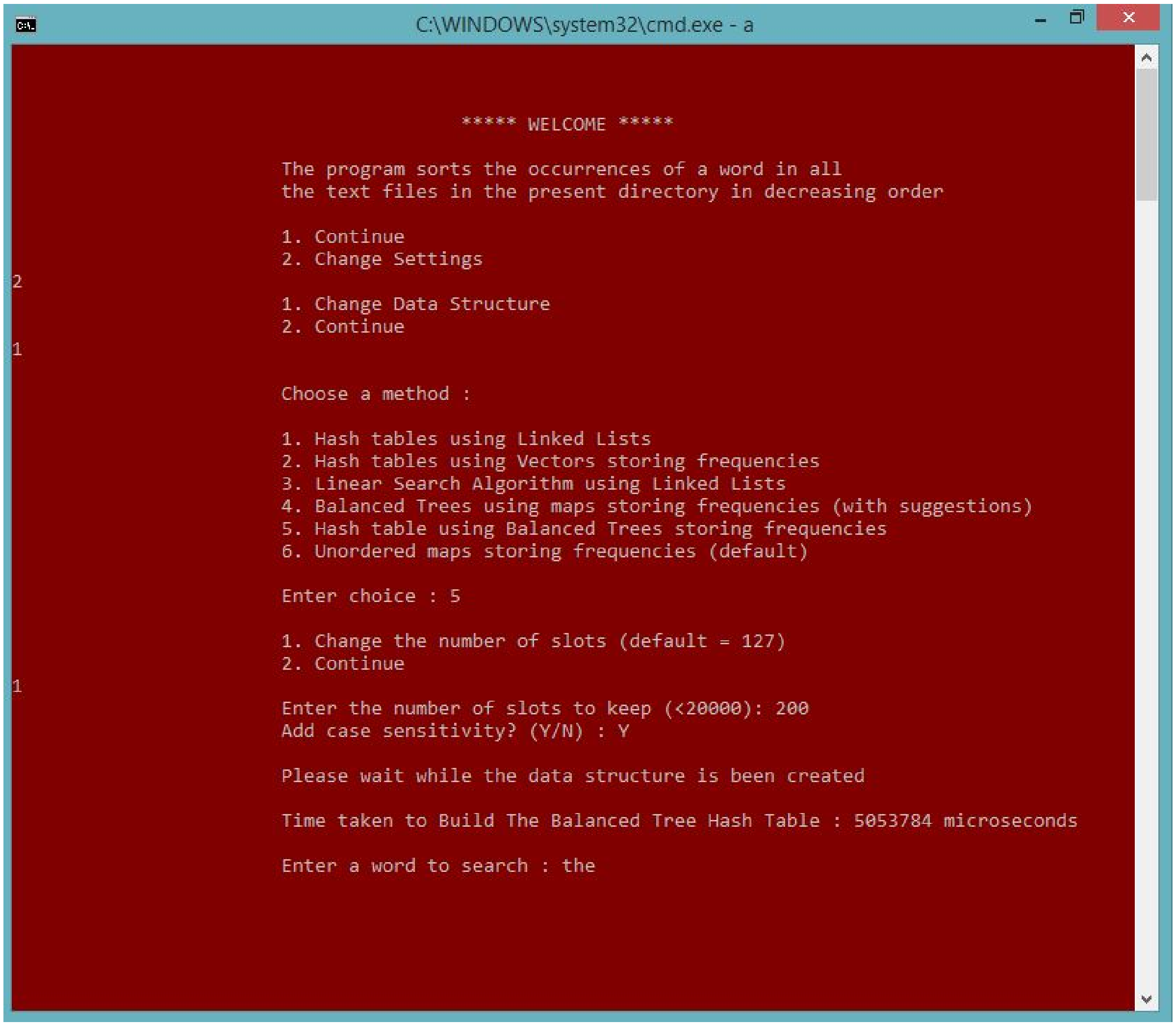
2 How to run

The program code uses the features of C++11. To compile the code use

```
g++ -std=c++11 main.cpp
```

3 Input

The program takes the text files kept in the current directory as an input



```
                                  ***** WELCOME *****

                        The program sorts the occurrences of a word in all
                        the text files in the present directory in decreasing order

                        1. Continue
                        2. Change Settings
2
                        1. Change Data Structure
                        2. Continue
1

                        Choose a method :

                        1. Hash tables using Linked Lists
                        2. Hash tables using Vectors storing frequencies
                        3. Linear Search Algorithm using Linked Lists
                        4. Balanced Trees using maps storing frequencies (with suggestions)
                        5. Hash table using Balanced Trees storing frequencies
                        6. Unordered maps storing frequencies (default)

                        Enter choice : 5

                        1. Change the number of slots (default = 127)
                        2. Continue
1
                        Enter the number of slots to keep (<20000): 200
                        Add case sensitivity? (Y/N) : Y

                        Please wait while the data structure is been created

                        Time taken to Build The Balanced Tree Hash Table : 5053784 microseconds

                        Enter a word to search : the
```

4 Features

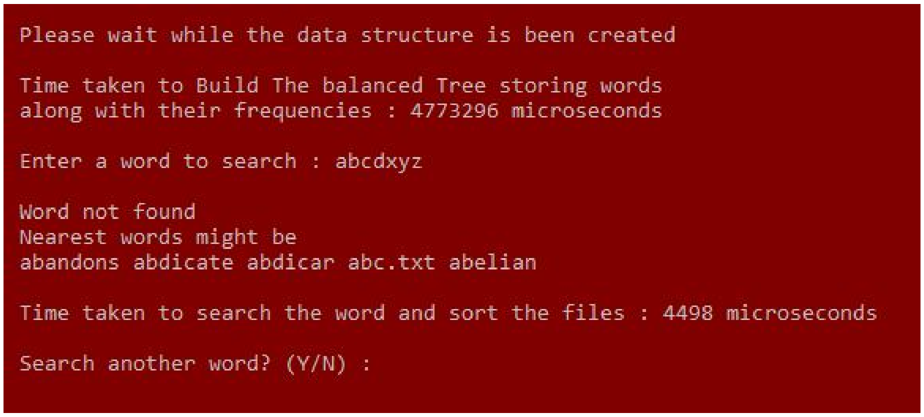
The program is build using STL which is inbuilt in C++11.
6 different data structures were used to analyze different possible methods to achieve the result.

- Hash table using linked list
- Hash table using vector storing frequency
- Linear search algorithm using linked list
- Balanced Trees using maps storing frequency
- Hash Table using Balanced Trees storing frequency
- Unordered maps storing frequency

The program allows th user to be more flexible with these data structures to analyze them further.

The user can

- Add or remove case sensitivity
- Change the number of slots in data structures which use Hash Tables
- Reuse the same data structure built to search more words at a time
- The code can be modified to perform searching for other types of files like .c, .cpp, .dat, etc.
- Give suggestions when a word is not found (applied in Case 4)
- Show the data structure building time and word searching time in microseconds.



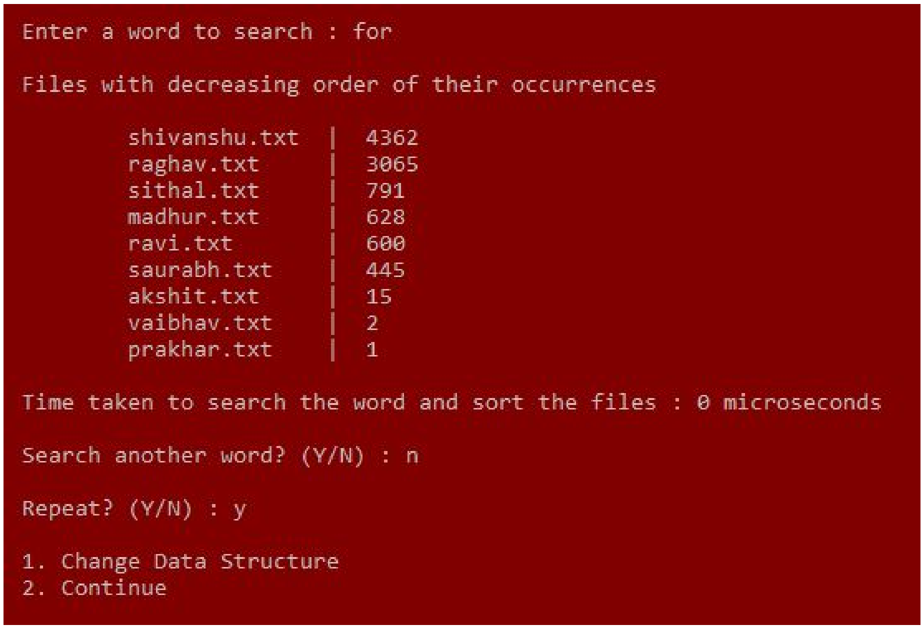
```
Please wait while the data structure is been created

Time taken to Build The balanced Tree storing words
along with their frequencies : 4773296 microseconds

Enter a word to search : abcdxyz

Word not found
Nearest words might be
abandons abdicate abdicar abc.txt abelian

Time taken to search the word and sort the files : 4498 microseconds

Search another word? (Y/N) :
```

5 Basic Code Structure

STL containers like pair, vector, list, map, unordered_map are used in the program.
The code first looks for the Operating System upon which it is running. System commands are used to take the names of the text files. A switch case is used to make 6 cases for 6 data structures techniques. In each case, every file is opened and the string is taken one by one from each file. Each string is processed such that the intra-sentential punctuations are removed. The intra-word punctuations are preserved. The processed string is inserted into the data structure. The user enters the word to search and the word is searched in the data structure. The frequency thus obtained is stored and is sorted in decreasing order and printed. Case 4 has a special feature to provide with suggestions which uses the lower_bound function to find the closest lexicographical word in the Map. A do while loop allows user to re-use the data structure again and again for searching more than 1 word. Another do while loop allows user to change the data structure while the program is running.

6 Output

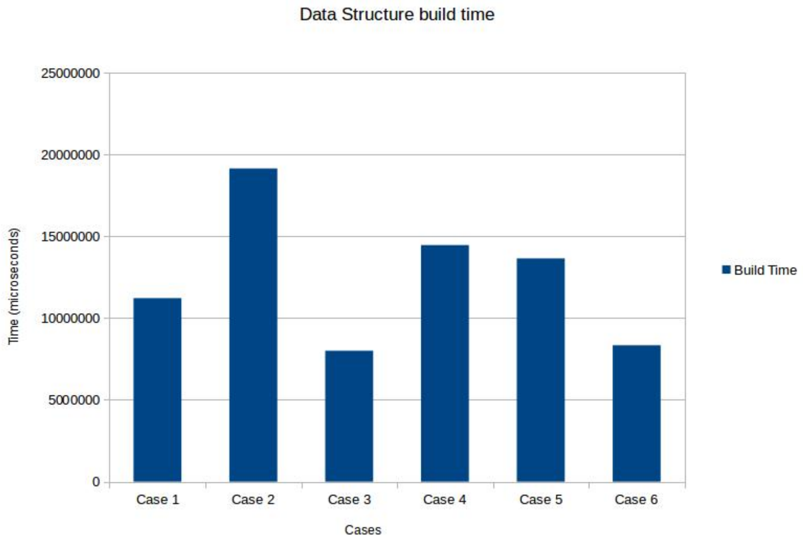
The program prints the list of files in decreasing order according the occurrence of the searched word.

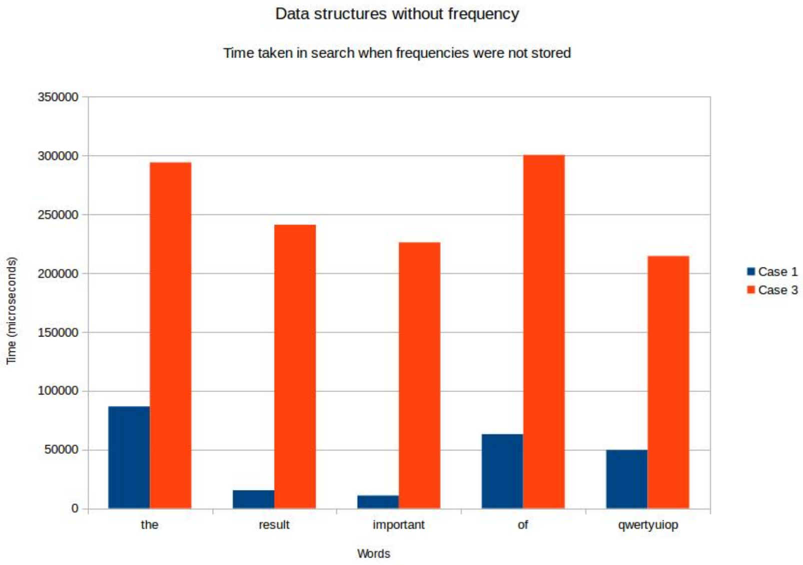


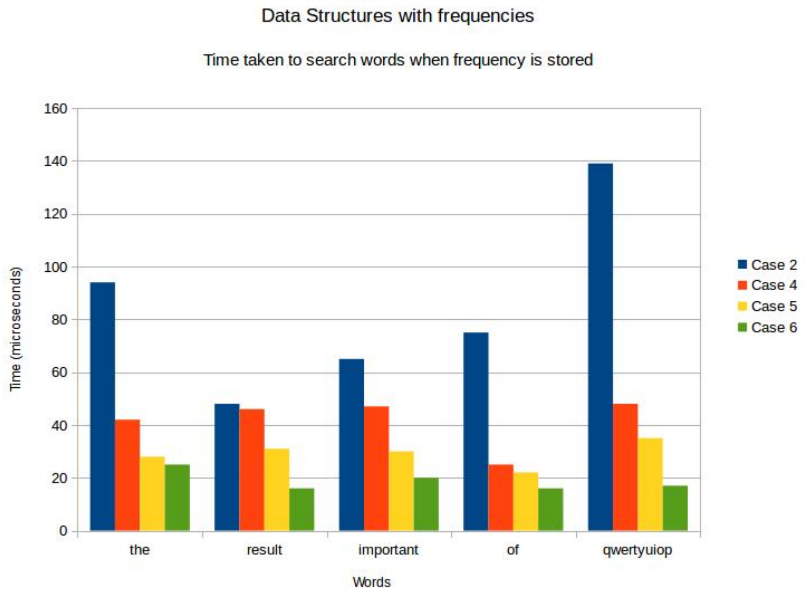
```
Enter a word to search : for

Files with decreasing order of their occurrences

        shivanshu.txt  |  4362
        raghav.txt     |  3065
        sithal.txt     |  791
        madhur.txt     |  628
        ravi.txt       |  600
        saurabh.txt    |  445
        akshit.txt     |  15
        vaibhav.txt    |  2
        prakhar.txt    |  1

Time taken to search the word and sort the files : 0 microseconds

Search another word? (Y/N) : n

Repeat? (Y/N) : y

1. Change Data Structure
2. Continue
```

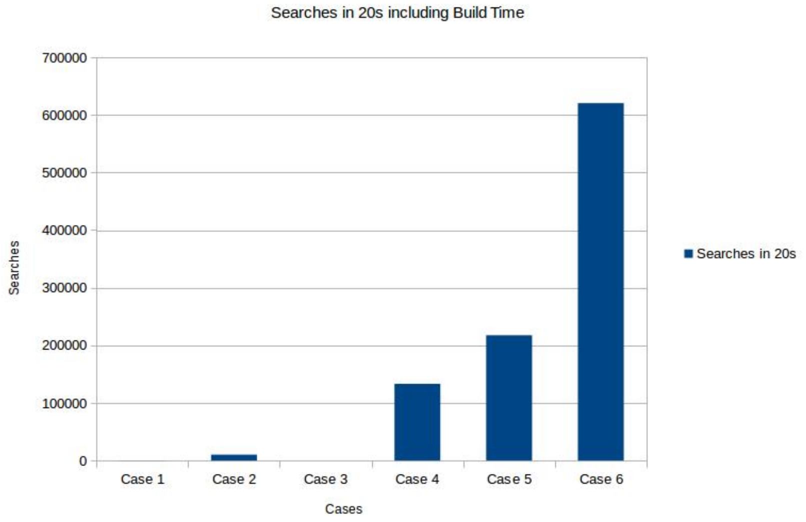
7 Results

The 6 data structures were thoroughly analyzed on the basis of build time, search time, space taken in RAM.
The following data were compiled in charts as below.









From the above statistical Data we clearly observe that Case 6 i.e., Unordered maps are the best amongst the 6 different Data Structures used in the program. So the default Case is set to 6.