Submitted by: **YASH BANSAL 2014A7PS0119H, SHIVAM AGRAWAL 2014B3A7PS0940H, AMAN GUPTA 2014A7PS0201H, VAIBHAV AGRAWAL 2014B3A7PS0501H**

DESIGN DOCUMENT

The proposed query search engine has been employed using Ranked Retrieval Vector Space model. It is an algebraic model for representing text documents (and any objects, in general) as vectors of identifiers, such as, for example, index terms. It is used in information filtering, information retrieval, indexing and relevancy rankings.

This project has been implemented purely by using Python 3. The code is executable on any PC having python3 libraries installed. The major data structures used in the project are:

- Dictionaries / HashMaps
- List
- String

The code consists of two files namely porter.py and indexer.py. Python's **nltk** library has been used to tokenize the words in the corpus. "Indexer.py" contains a list of functions together processing the corpus and producing the results.

The user is prompted to enter the query. After hitting the enter key, the qetquery() function normalize and tokenize the given query. Resultant list is searched in the corpus by generating the tf-idf of each token.

Obtained results are heapified using the inbuilt python library. This would help showing the results in decreasing order of relevance. Output is formatted as the value of document id.

The precision and recall values are calculated as follows:

Precision value = relevant documents retrieved / total retrieved documents.

Recall value = relevant documents retrieved / total relevant documents = 1

Precision > .5	Recall = 1

Novelty of the search engine

One of the most **important** ability of this search engine is the autocorrecting of the search query. It has been implemented by using the edit distance of various strings and comparing the results obtained. Furthermore, choice of optimal document is made on the basis of no. of query tokens contained within it. Furthermore, if only the edit distance approach is used, then it does not take care of the context of the query during the auto-correction phase. To counter the problem of not choosing an optimal or a good document, we reimbursed the cost of operation to find the document which

contains the maximum number of words having minimum edit distance from the incorrect query word.

Data Structure has been designed in such a way which ensures average case time complexity to be theoretically minimum possible value.

Time Complexity of Search Engine Functions:

Time to build index:

Average Case: O (words) + O (documents) + O (sum of lengths of all words)

Worst Case: O (words)^2 + O (documents)^2 + O (sum of lengths of all words)

Time spent per Query: If search query words lie in corpus:

Average Case: O (Documents Retrieved) + K*O(lg(Documents Retrieved)) + O(length of search query)+ Average Case Correction Time

K= No of results to be displayed at a time

Worst Case:

O(length of corpus) *O(documents Retrieved) + K*O(lg(Documents Retrieved)) + O(length of search query)*O(documents)+Worst case Correction Time

Time for correction:

- **If Query word not found:**
  Average Case: O(avg.length of a word)^2*O(words)+ O(1).
  Worst Case: O(Avg.length of a word)^2*O(words)+ O(1).

- **If Query word found:**
  Average Case: O(1)
  Worst Case: O(1)