ABOUT THE SEARCH SYSTEM:

This project implements the **Boolean Retrieval Model** using Python. It covers basic AND, OR, NOT queries, Wildcard queries, Phrase and Proximity Queries. Further, it outputs the documents on the basis of their ranks, and the user can also explicitly input the number of documents he/she wants to retrieve.

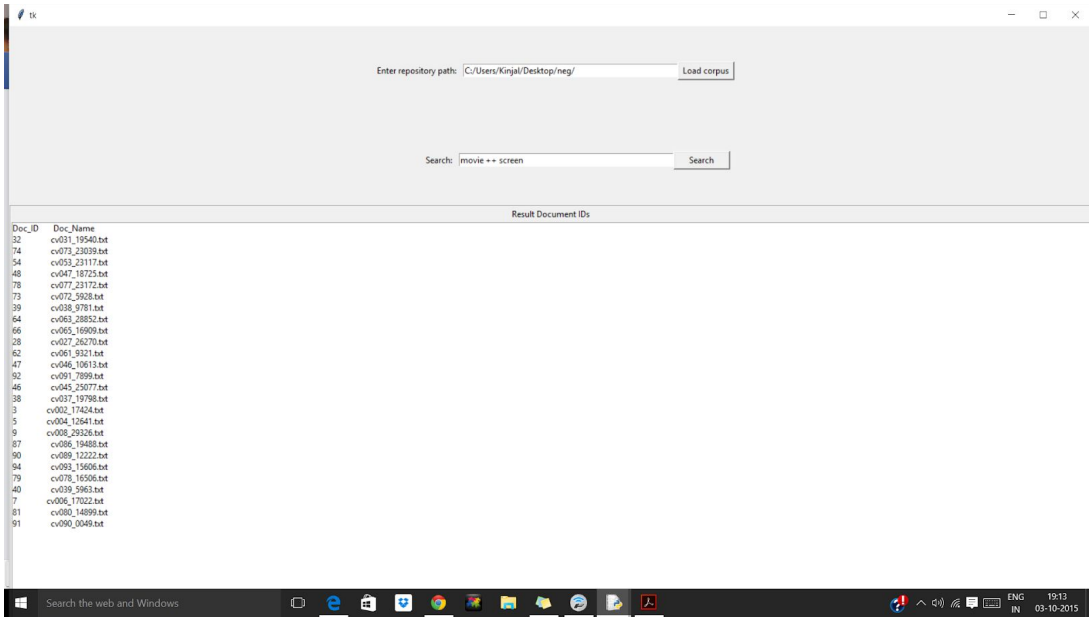
SYMBOLS FOR ADVANCED USE:

In the code, the following operators have been used to denote the type of retrieval:

1. '++' for and AND operation
2. '//' for an OR operation
3. '--' for a NOT operation
4. ‘^^n’ for PHRASE AND PROXIMITY queries
5. ‘..n’ for printing only the first n documents ordered by rank

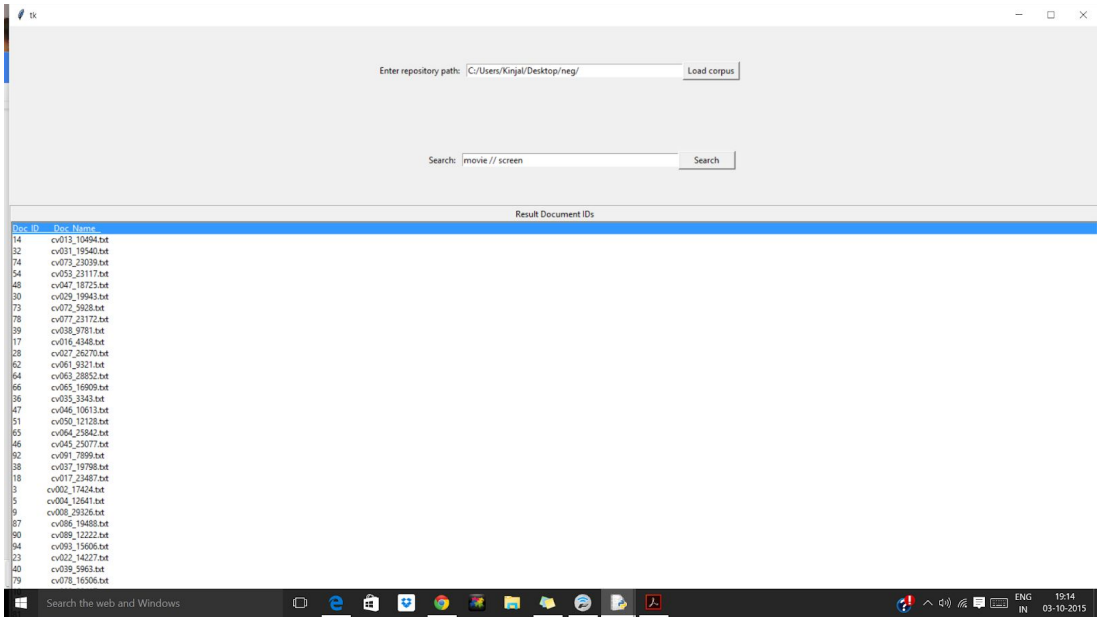
AND:

The AND operation involves appending the Document ID (docID) to the final list if it matches with the retrieved docIDs of all the query terms present the query separated by '++'.



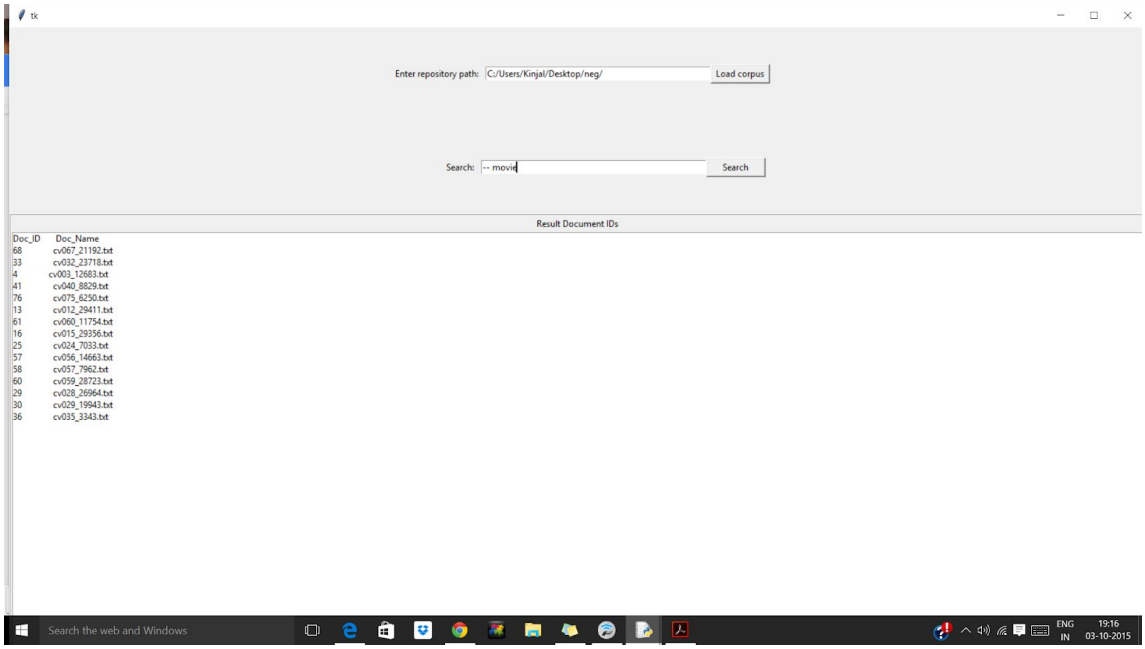
OR:

The OR operation appends all the docIDs relevant to the query terms on either side of the '//'.



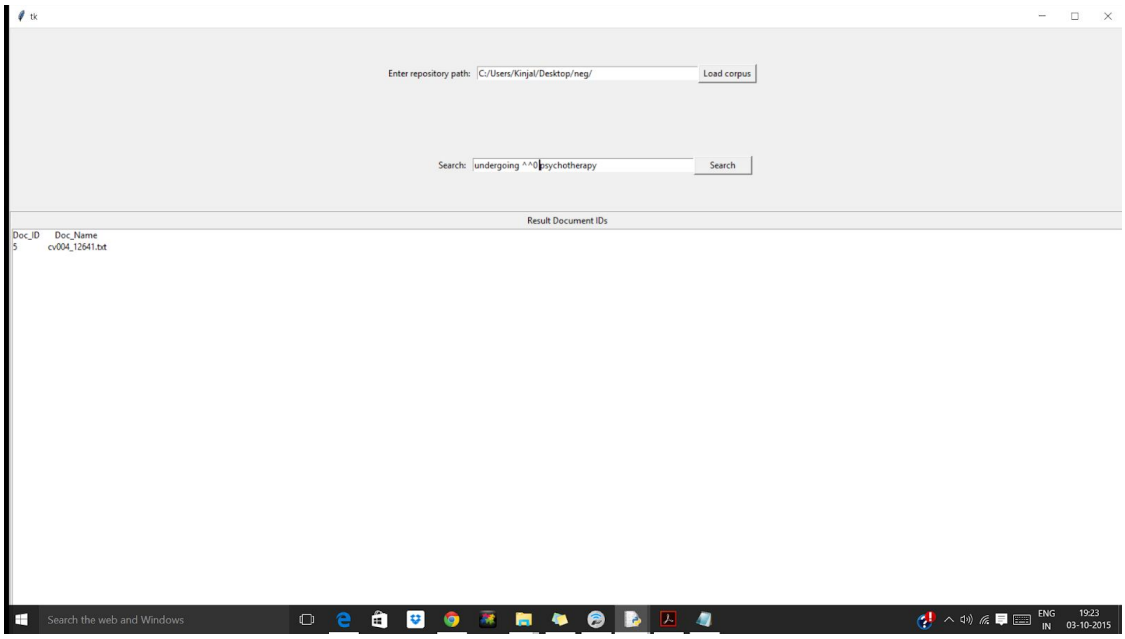
NOT:

The NOT operation simply removes any docIDs from the final list that match with the docIDs retrieved for the query term following '--' operator.

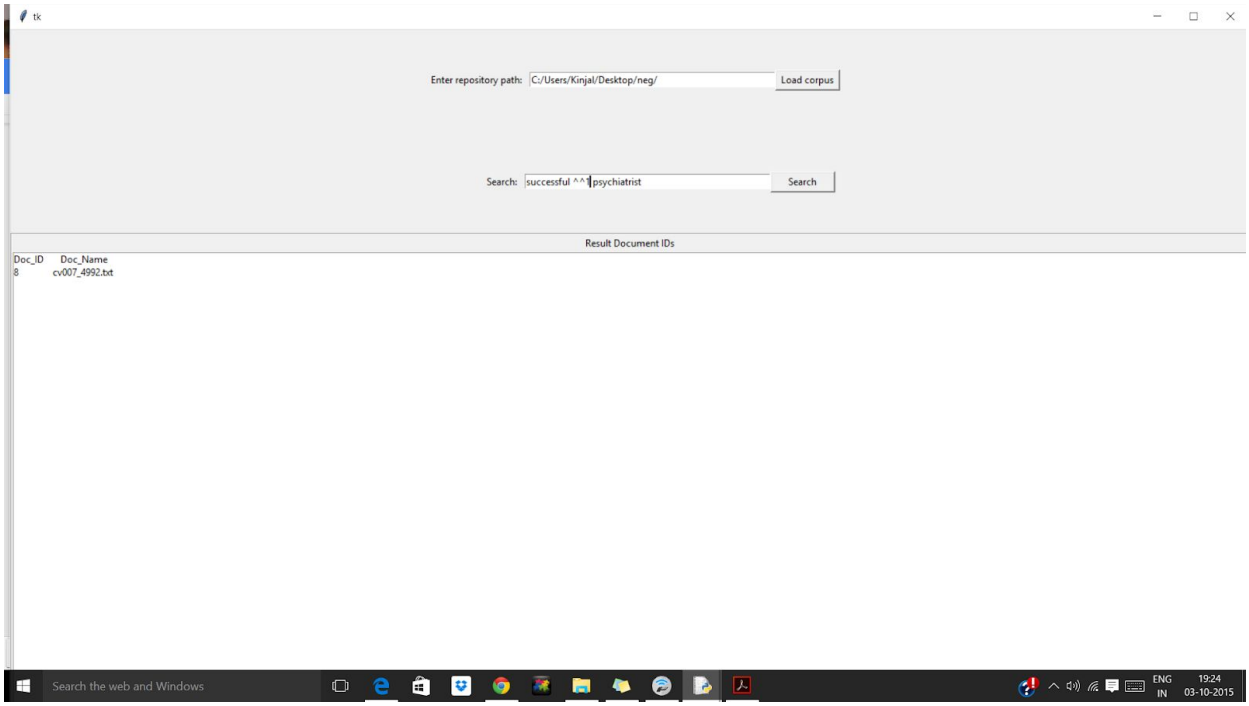


PHRASE QUERY:

The PHRASE query checks for the presence of query terms in documents with a proximity distance of "0" giving the docIDs which abide by it.

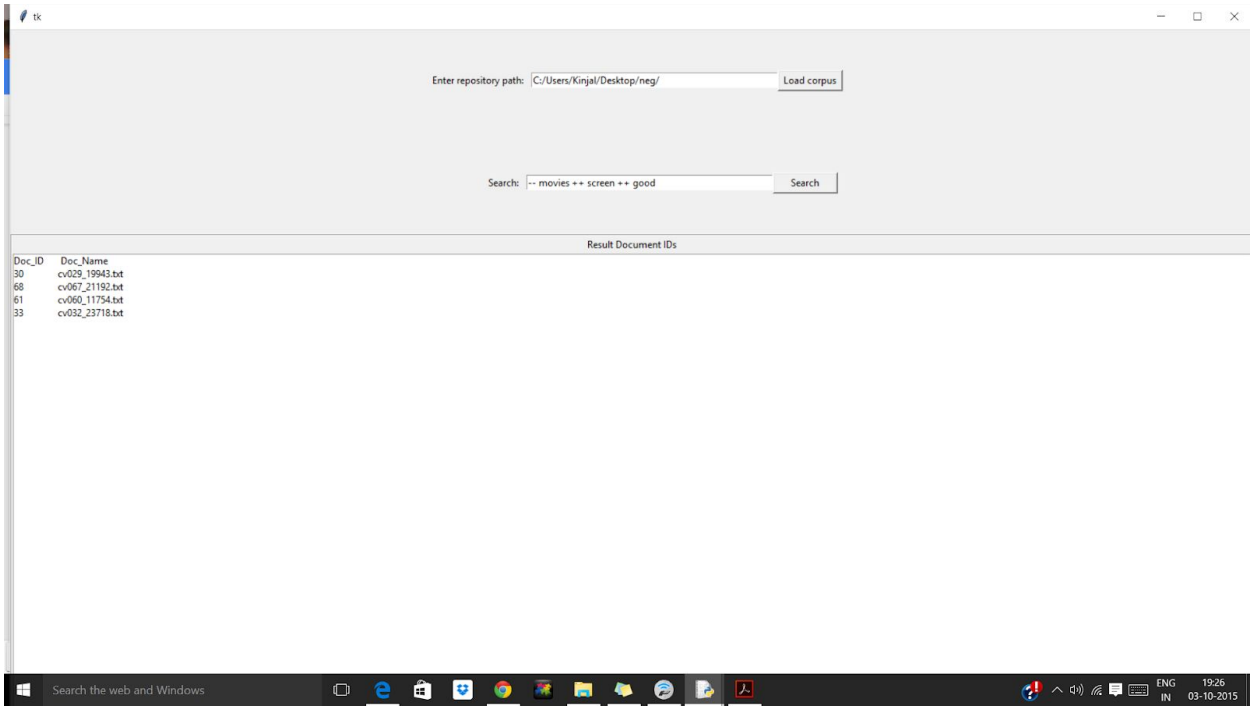


The user can give any particular “n” value to retrieve the docIDs which contain the query terms separated by ‘^^n’ at a distance of “n”.



COMBINATION OF SYMBOLS:

The user can also combine various operations to make a complex query.



DATA STRUCTURES:

The Data Structures used in the code are lists, list of lists, dictionaries and dictionary of dictionaries. Separate dictionaries are used to store the list of tokenized words, set of words after stemming, the stop words, and the contents of each file.

TOKENIZATION AND STEMMING:

Tokenization involves all words being converted to lowercase, whereas stemming is implemented via Porter Stemmer. Porter Stemmer removes all the plural forms, apostrophes and suffixes and converts the word into its root word.

POSTINGS LIST:

A postings list is created to hold the set of distinct terms, and the list is then sorted alphabetically. Postings list contains the word and the set of documents in which the word in present. It can be used to find the document frequency which is required for tf-idf weighting.

FINAL LIST:

Final list contains ranked documents as docIDs, with respect to their tf-idf weighting.

PICKLE:

Pre-processing large number of documents every time the search engine is used, is resource intensive and takes a long time to complete. To avoid this, Pickle has been implemented as a part of the search engine. This feature allows the user to store the pre-processed data like posting lists and term frequencies in serial form in a file. Whenever the user uses unaltered corpus, he/she can directly load data from the serial file, thus avoiding the need to pre-process data. This can be done by entering “pickle” into the repository text box. If the corpus is modified, the user can enter the path to the corpus, and the entire pre-processing is performed.

GUI:

A GUI has also been implemented, which uses the Python’s Tkinter module. It has a text field, wherein the user enters the path to access the repository of text documents and loads the corpus. The user can then enter his/her query in the Search text field, and the output is displayed in the listbox as ID and Name of the relevant documents ordered by rank.