The aim of the project is to build a Boolean Retrieval System which takes in a query as the input from the user and returns all the documents from the corpus which satisfy the query.

The design process for the project is as follows:

1. Pre-processing the documents of the corpus
2. Building an inverted index for the corpus
3. Processing the Boolean Query and returning the suitable documents

**Pre-processing the documents of the corpus:**

1. **Tokenization:**  
   -> Tokenization is the process of splitting the input document (given in the form of a string) into its constituent terms. We use the NLTK library for the same. It generates a list of tokens in the input text.
2. **Removal of Punctuations:**  
   -> For our purpose, punctuations include all tokens that aren’t in the form of alphabets of numerals.  
   -> From the generated tokens above, we identify the tokens that aren’t in the form of alphabets or numerals and remove them.
3. **Removal of Stopwords:**  
   -> Stopwords are the most commonly used words in a language whose purpose is to connect the parts of a sentence. They don’t convey any specific meaning on their own.  
   Eg: - a, the, for, are etc.  
   -> Removing these stopwords from our documents saves processing space and time and its affect on the final output is negligible.  
   -> Stopwords are removed from the list of tokens generated above with the help of the NLTK library. NLTK Library holds the list of stopwords for the English language which is used to perform the task at hand.
4. **Stemming:**  
   ->Stemming is the process of reducing inflected / derived words to their word stem by chopping off the prefixes of the words. This reduces the storage space required and helps in efficient analysis of the input query.  
   -> To stem all the tokens that were generated above, we take the help of the NLTK Library which provides us with the ‘PorterStemmer’ to achieve the task.

**Building an inverted index for the corpus:**

* Inverted Index is a data structure that stores all unique terms in the corpus and a posting list for each term.
* A posting list for a term is the list of documents that contain the particular term.
* We used the Python Dictionary to build the Inverted Index where the keys of the dictionary are all the unique terms in the corpus and the values are the posting lists for each term.

**Processing the Boolean Query and returning the suitable documents:**

* The boolean query is a combination of terms or wildcard queries joined using boolean words.  
  Eg: - Julius or cl\*ra and not calphurnia
* Before proceeding further, we first need to take of a couple of things:

1. **Spelling Correction:**  
   We need to correct the spellings of misspelled words (not in wildcard query form).  
   -> We use the Levenshtein / Edit Distance method to achieve the task.  
     
   **Levenshtein Distance Method:**-> It calculates the minimum number of operations to convert the source word to the destination word.  
   -> The only operations allowed are inserting, deleting, or replacing a character. Each operation is given equal weightage.  
   -> The make use of the principles of dynamic programming and matrix to store all the edit distance values of substrings to solve this problem.  
     
   **Spelling Correction:**-> We run the edit distance method on the query terms w.r.t all words in the dictionary.   
   -> If the input word is already correct, it returns the word itself.  
   -> If the input word has been misspelled, it returns the word from the dictionary that has the least edit distance when compared with the query term, which is most likely to be the correct spelling of the query term.
2. **Wildcard Query Processing:**  
   -> Wildcards are special characters that can stand in for unknown characters in a text. For boolean retrieval, the wildcard character used is ‘\*’ which stands for zero or more unknown characters.  
   -> When a wildcard character is used in a term, it becomes a wildcard term.  
   Eg: - cl\*p\*ra  
   -> The technique we implemented to process a wildcard term is the ‘Permuterm Method’  
     
   **Permuterm Method:**-> We append all the terms in the dictionary with a ‘$’ symbol to signify where the term ends.  
   -> The we take all possible permutations of the newly generated term.  
   -> We used a dictionary (Permuterm Index) to achieve the same where they keys are the possible permuterms for all the terms in the dictionary and the values are the actual terms those permuterms correspond to.  
     
   -> Then we rotate the wildcard query terms (to the right) given as input as follows:  
   a) X\* => X\*$  
   => $X\*  
   b) \*X => \*X$  
   c) X\*Y => X\*Y$  
   => Y$X\*  
   d) \*X\*$ => $\*X\*  
   e) X\*Y\*Z   
   => Z$X\* and $\*Y\*  
   -> We process this as two separate wildcard query terms and then take the intersection of the results obtained.  
   f) \*Y\*Z   
   => Z$\* and $\*Y\*  
   -> We process this as two separate wildcard query terms and then take the intersection of the results obtained.  
   g) X\*Y\*   
   => $X\* and $\*Y\*  
   -> We process this as two separate wildcard query terms and then take the intersection of the results obtained.  
     
   -> We find all the permuterms generated that satisfy our wildcard query terms requirements and return all the terms that correspond to these permuterms. These terms are all the words that satisfy the given wildcard query.
3. **Boolean Query Processing:**-> For processing the boolean query, we follow a priority order for the connecting boolean words. The order is as follows:  
   NOT > AND > OR  
   -> We retrieve the posting lists of all terms in the query(in the same order as of their occurrence in the query) and then store them all, one by one, in another combined posting list.  
   -> If the query term is a simple term, we first check for its spelling, correct if necessary and then return its posting list.  
   -> If the query term is in wildcard form, we use the wildcard query processing method discussed above to retrieve the combined posting list of all terms returned by the wildcard term.  
   -> On these posting lists, we apply the operations NOT, AND, OR in the same order.  
   -> The output is a list of documents that satisfy the query.
4. **Output:**-> The user gets to input a query of their choice. The output is a .txt file containing the names of all the files in the corpus that satisfy the query.

**RUNTIMES :**

**Preprocessing -** 39 Seconds

**Building inverted index -** 2 seconds

**Search or retrieval -** 7 seconds

**Central Data Structures :**

Inverted Index , Permuterm Index, Dictionaries, Matrix (2-D List)