Information Retrieval Assignment - 1
Design Document

Group Members:

1. Navam Shrivastav 2019A7PS0092H
2. Sukrit Kumar 2019AAPS0231H
3. Shrikrishna Lolla 2019AAPS0345H

Model Architecture:

1. **Preprocessing - Stemming, Tokenization, Lowercase conversion, Removing stop words.**
   We use the SnowballStemmer built into NLTK to perform stemming on our corpus. We also use the built-in list of English stop words in NLTK to remove the stopwords. Tokenisation was performed using the word tokenise function in NLTK.

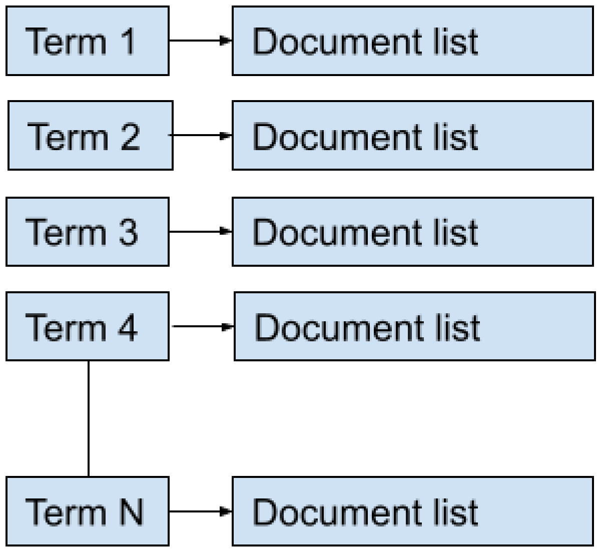
2. **Indexing Data Structure - Inverted Index**
   An inverted index is an index data structure storing a mapping from content such as words or numbers to its locations in a document or a set of documents. There are two kinds of Inverted Index: record-level inverted index and word-level inverted index. A record level inverted index contains a list of references to documents for each word. A word-level inverted index also includes each word's positions within a document.

   In this implementation, we use record level inverted index to map words to the documents in which they are present (with the help of document ids). If a word is present in a document, the document id is appended to the list of the concerned word. Note that the document id is appended only once, even if the word is present multiple times in the document. During the preprocessing, the doc ids are appended in increasing order. This enables us to implement AND operation in `O(n+m)` time complexity, where n is the length of the first list and m is the length of the second list. Instead of using a list, one can directly store the bitset corresponding to the binary no of the values wherein zero would imply the term doesn’t exist in the document and one the document exists. This would allow us to perform boolean queries directly using the bitwise operations.

   The InvertedIndex class consists of 2 helper functions which are Search and Find. The find function is used to check if a particular term exists in the index; if it does, it returns the document list and otherwise returns None. This operation takes a constant O(1) time. The Search function is used in the case for wildcard queries; the queries are rotated till * is the last value, and then we check all the keys of the indexes to check whether the key starts with the query or not. This operation takes O(d) times, where d is the size of the vocabulary.

The insert function is used to insert a new word in the InvertedIndex; this operation takes O(1) time to insert a new word and append its doc id in the corresponding list.

Permuterms was used to work with wildcard queries.



Time taken to preprocess the data and build the index was 15 seconds.

3. Querying

We first took in the query, tokenised it using word_tokenize and then performed stemming. We then looped through the entire query, retrieved the lists for each term, and performed the appropriate boolean operations based on the connectors. Edit distance was used to correct errors in the queries, and we decided to use the term with the lowest edit distance as the correct query terms.

We created three functions, one each for AND, OR and NOT. The AND operation performed the intersection of the list in O(m+n) time; the OR operation found the combined list in O(n+m) time where n is the length of the first list and m is the length of the second list. The NOT operation took O(N) time, where N was the total number of documents in the corpus.

Then depending on the type of query, wildcard or regular, we retrieved the list from the Index by using the previously mentioned search and find functions. The rotate string function was used extensively to make a permuterm index of words to use in wildcard queries, and they were rotated so that * was the last character and then searched in the index. The query takes less than 1 second to run in most cases, with wildcard queries generally taking more time.

Runtime Analysis:

Sl No	**Function**	**Run Time**
1.	**Preprocessing (Stemming, Tokenization, Lowercase conversion, Removing stop words)**	**1-2 seconds**
2.	**Building Index**	**15 - 16 seconds (first time) 0.75 seconds (reading from pickle file)**
3.	**Boolean Query**	**1-2 seconds**
4.	**Spelling Correction**	**1-1.5 seconds**
5.	**Wild Card Queries**	**4-5 seconds**