CS F469 : IR

Assignment 1

Date : 21th March,2022

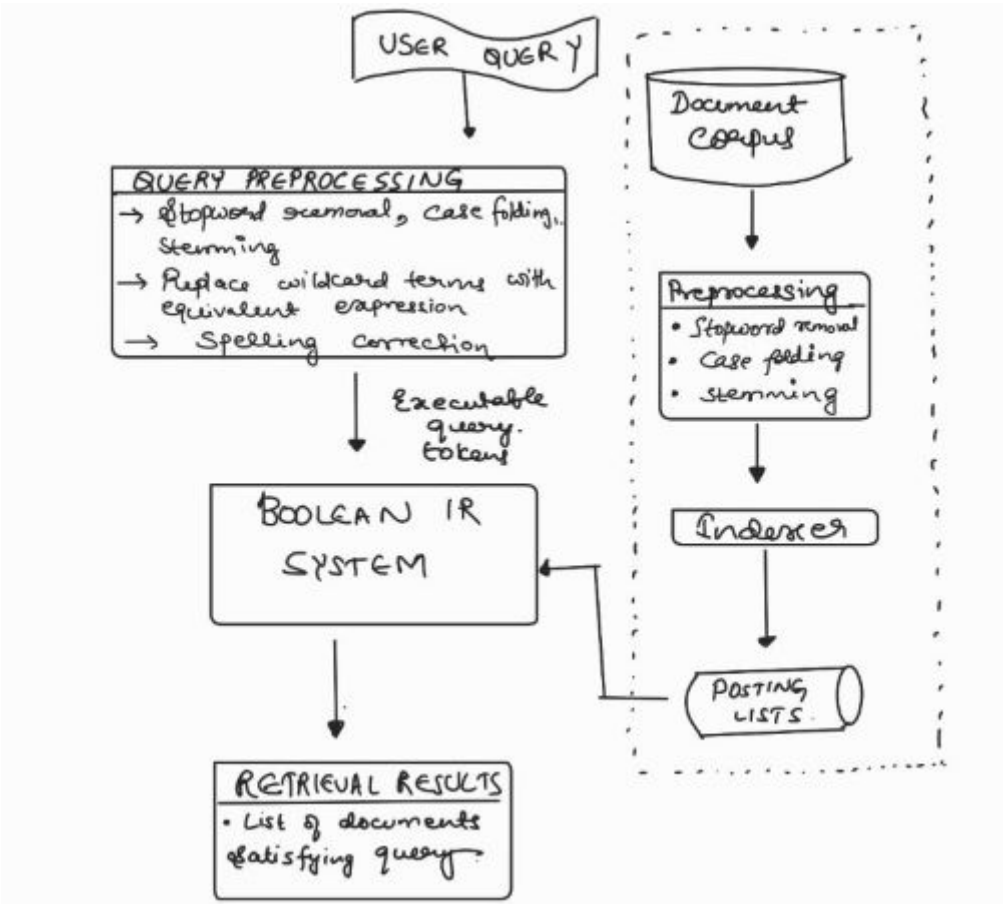
Team Members

R Adarsh	2019A7PS0230H
Sahaj Gupta	2019A7PS0148H

Problem 1 Boolean IR System

1.1. Model Description and implementation

ARCHITECTURE



PROCEDURE

Note:
N = number of documents in the corpus
|W| = total number of words/terms in text in a particular document
|V| = number of words in vocabulary set
|w| = number of words/tokens in a query string
|t| = number of letters in a term

PRE-PROCESSING

Document preprocessing :
1) Involves tokenizing, removing stopwords, case folding, stemming (PorterStemmer used) and building the inverted index structure.
⇒ Traversing the corpus of documents and parsing each document to get text content of files: O(N) time complexity
⇒ Pre-processing on text and creating vocabulary : O(|W|)
2) The function preprocess() takes path of folder where documents are stored as input and builds
a) The inverted index data structure which maps each term with list of documents in which the term appears
⇒ **Inverted index data structure**
- Uses dictionary data structure of python
- Maps each term to list of documents in which that document appears
- Building the inverted index structure requires traversing through the list of terms and generating posting list for each term : O(|W|)
- The above step is done for each document, so worst case time complexity to build the inverted index is O(N*|W|)

b) Doc_map which stores doc_id ==> doc_name
⇒ **Doc_map data structure**
- Uses dictionary data structure of python
- Maps doc id to doc name
- Time complexity to build doc_map : O(N)

c) Set of vocabulary terms
⇒ Uses the set data structure of python to store words. Created from the keys of posting lists. Worst case complexity : O(|W|)

Thus, Overall time complexity in document pre-processing : O(N*|W|)

Query preprocessing :
1) Tokenizes the query terms
2) Replaces wildcard terms with equivalent boolean expression of terms
⇒ **Permuterm index data structure**
- Permuterm index is used to handle wild-card entries
- Uses dictionary data structure of python
- For each key in the inverted index structure, generates all rotation of the term and maps them to original word : O(|V|*|t|)
- For matching regex expression to correct term, have to traverse the permuterm index
- So, overall complexity for wildcard matching using permuterm : O(|V|*|t|)

3) Spelling correction of terms using edit-distance method
⇒ To calculate the edit distance between two words w1 and w2, the classical dynamic programming approach is used : O(|t1|*|t2|)

Overall worst case time complexity in query-preprocessing : O(|w|*|V|) , considering the number of letters in a term as constant

QUERYING & DOCUMENT-MATCHING

⇒Expects a *well formed query* to be given as input

***Well-formed query:**

- Every word/symbol must be space separated
- The following symbols represent boolean operators : **AND = & , OR = | , NOT = ~**
- If parentheses are used, it must be ensured that they are properly balanced
- In absence of parenthesis, the precedence order followed by operators is ~ > & > |
- Supports wildcard entries/terms of following formats: A* , *A , A*B , A*B*C

⇒ The major task in querying operations is to evaluate the given boolean query expression to retrieve match documents.

⇒ **Evaluating expression & document matching**

- Uses two stacks : operand-stack and operator-stack to evaluate expression based on parenthesis balancing & operator precedence
- Stack is implemented using the deque data structure available in python
- Based on operator, document retrieval involves the following set of operations between the term posting lists containing docIds:
    a) Intersection for AND
    b) Union for OR
    c) Negation for NOT

Each of the above operation takes linear time in number of documents to run : O(N) for each operation

- Overall complexity of this step is O(N*|w|)

⇒**Document Retrieval**

Based on the list of docIds generated as result of the above step, the ids are matched to doc_names using the *doc_map data structure* previously created. Worst case complexity : O(N)

Overall complexity of querying and doc matching : O(N*|w|)

CONCLUSION

Retrieval of documents based on boolean query and wild-card matching can be efficiently carried out by using data structures like hash-maps and dictionaries. The major advantage of these data structures is that they have constant look-up time and can efficiently store the data. Creating an inverted index and then carrying out boolean operations on these lists saves a lot of time as compared to checking each document in corpus.