Readme file

Common Assumptions:

1) All files have text based content, so we need to consider all files while processing.
2) Stopwords are not relevent while making inverted index or positional index.
3) Stemmed words paly imporatnt role in getting relevent search result.

Solution 1:

In this part for Unigram inverted indexing, we have used

nltk library.

We have written the code using googlecollab.

First, we have accessed the documents from google drive.

Now we have preprocessed the data files by two functions one is for stemming, and one is the main function by the name called process, which is our main function for preprocessing.

PREPROCESSING :

stemming () function uses the PorterStemmer() for stemming the words in the files.

The process() function, has seven steps, these are as follows :

Step 1: We are removing the punctuation by replace method

Step 2: Lowering the letters by lower() function

Step 3:We are removing the stop words.

Step 4:We are making the tokens by using the nltk.word_tokenize()

Step 5:Used the stemming function for stemming the words in each document

Step 6: Removed the blank spaces bu using the strip() function

Step 7: Removed the duplicates using the dict.fromkeys and make it a list and at the end, return this list

Now we have retrieved each document one, applying all the preprocessing on each word of all the documents and save it in the list called by the name, all_Terms[]

Unigram inverted Indexing Logic :

We have made a dictionary called my_Inverted_Dic = {} to save the documents ids of a specific word.

The dictionary has the key as the current word, and the value as another dictionary whose key is "found in these docs," The value is a list containing the document's id in which the current word is present.

Logic is that we have made a for loop on each specific word. If that specific current word is already present in our dictionary, we append the document id in its's list. If the word has come for the first time, we will make a new value pair.

Query Processing

We have made two functions

function1: userQuery()

It takes a string as the input to preprocess the words and return the list as the tokens.

function 2: take_Operators()

It takes a string to filter the operators from the user input, save the operators in the list, and return it.

Now the logic of OR, AND, OR NOT, AND NOT is written by the four functions

OR(), AND_Logic(), OR_NOT_Logic, AND_NOT_Logic

Let us say if the number of operators from take_Operators answer is my_op.

For loop for the query processing will be run for my_op times.

Each time it will take words and take an operator to run the function corresponding to four possible cases.

The query runs, and the Number of comparisons gets updated every time for different operators.

Three results will be printed at the end of the query as given in the rubrics.

1. The Minimum number of Total comparisons.

2. The Number of documents retrieved.

3. The list of documents names retrieved.

Solution 2:

In this Part the preProcessing is done as follows:

1) Punctuations are removed.
2) Text converted to lower case letters.
3) Stopwords are removed
4) Tokenise the text

5) White spaces are removed
6) Stemmimg is performed

Methodology:

In 2nd part of the question we have to make a positional index data structure using the dataset given to use .which is as follows:

Positional index structure:

1) It is in the form of a dictionary where we have a word token as a key and the value is itself a dictionary.
2) In this inner dictionary we have documents as the key and the value is a list containg positions at which that particular word is present in that document.

Positional index making:

3) This positional index is made by looping over all the documents .
4) Later after making the data structue we have saved it and read it again when we have to search the query.

Query Processing steps:

5) In searching procedure we have taken the query , process it in the same way that we have used before making the data structure.
6) After preprocessing our algorithm takes first two consecutive word token and read its dictionary where the information about its documents and positions are saved, and it iterate over the two dictionary and when it found that the words are in the same document then it search for there position ,such that whether they are consecutive or not.
7) So if it is find out that words are consecutive in same document it saves that documents and use it later.
8) In this way after processing on two words we take the 3rd word and so and the final documents id list is returned
9) After taking that document id list it print the name of the documents.

   Function **getDoc** is used to get the document id list where the words are present and function **getDocName** is used to get the name of the documents.

Tools used :

Dataset is loaded and saved in **google drive** .

Coding and analysis is performed over **google colab**.