

# IR System

BASED ON BOOLEAN RETRIEVAL MODEL

GROUP ID: 10 – Q3

PRITHVI NITHIN - 150911062

SUSHANT PRASAD – 150911130

SAUMYA BAHUKHANDI - 150911254

SHRADDHA - 150911100

## FUNCTIONALITY IMPLEMENTED:

- Can read the content of the documents
- Generates an inverted index
- Uses Postings list intersection algorithm and Multiple posting list intersection uses query optimization
- Query can consist of at AND, OR Operators

## Objective:

To Develop a simple Information retrieval system using Boolean Retrieval Model.

## Introduction:

In the Boolean retrieval model the documents to be searched and the user query are both treated as sets of terms in a Boolean expression i.e. the sets of terms are separated by Boolean operators like AND, OR, NOT etc.

## Implementation:

The IR System is built using Python exclusively and requires the Python Compiler to run. The Documents to be searched are stored in the "Documents" sub-directory in the application folder. The file names of the documents stored in this folder are obtained using the `listdir()` method in OS module of Python. The contents of these files are tokenized and processed to obtain the Inverted index. The variable "postings" in the code represents the Inverted Index. It is an `OrderedDict` object type from the `Collections` module of Python. The key-value pair is defined as :

Key : Term

Value : Document Frequency ( $f_d$ ), Posting List

The Inverted Index is sorted in alphabetical order. The function "`intersect()`" is used to find the intersection of 2 posting lists.

The function "`multintersect()`" is used to find the intersection of multiple posting lists. This function calls the `intersect()` function

The query is taken as a user input and the query is split into subqueries combined by OR. The terms are extracted from each sub-query. Now for each of these posting lists each query term is extracted from "postings" and stored in "qlists" which is then sorted according to document frequency (ascending order) for query optimisation purposes. Now the result is found for each sub query and combined to give the final output.

The final answer i.e. the list of documents with the required query terms is stored in "fanswer".

Appropriate output is then printed on the screen for the user.

## Code and Screenshots:

*Tokenizing data from files and building inverted index*

```
from collections import OrderedDict
import os

doclist = os.listdir(".\\Documents")          #List of Document File Names Obtained
text = {}
index = []
for file in doclist:                          #Tokenising and storing terms
    fhandle = open('.\\Documents\\'+file,"r")
    str1 = fhandle.read().lower()
    if str1[len(str1)-1] == '.':
        str1 = str1[:len(str1)-1]
    str1 = str1.replace(".", " ")
    str1 = str1.replace(", ", " ")
    lst = str1.split()
    text[file] = lst
    for terms in lst:
        index.append(terms)
```

*Intersection Function and Multiple-Intersection using Intersect Function*

```
def intersect(p1, p2):
    answer = []
    for doc in p1:
        for doc1 in p2:
            if doc1 == doc:
                answer.append(doc)
                p2.remove(doc)
                break
    return answer

fanswer = []

def multintersect(qlst):
    fanswer = qlst[0][1]
    qlst = qlst[1:]
    while (len(qlst) != 0 and len(fanswer) != 0):
        fanswer = intersect(fanswer, qlst[0][1])
        qlst = qlst[1:]
    return(fanswer)
```

### Main Function

```
query = input("Input your Query (You can use AND, OR operators): ")
queryOR=query.split(" OR ")
answer = [0,[]]
finalAnswer=[]

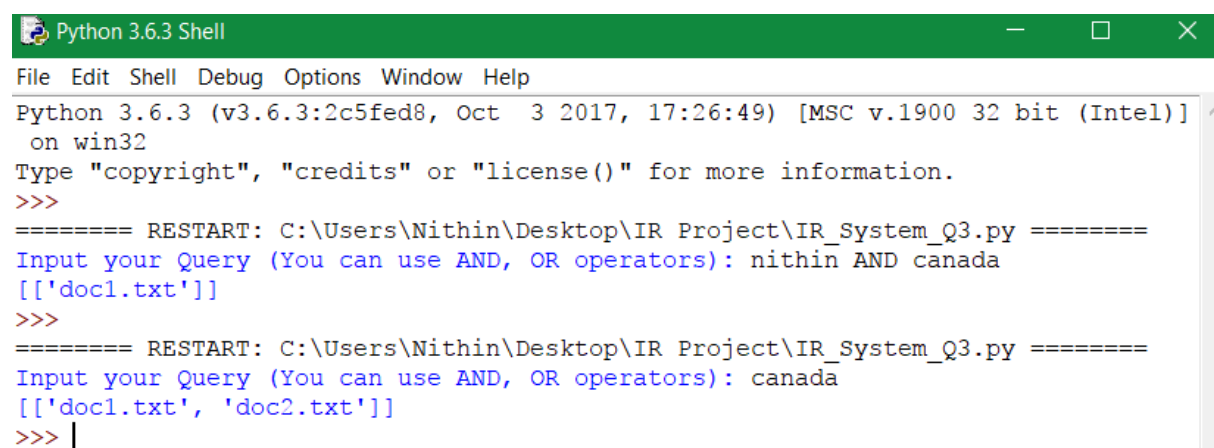
for ORterm in queryOR:
    query = ORterm.split(" AND ")
    qlists = []
    for term in query:
        if term.lower() in postings:
            qlists.append(postings[term.lower()])

    for i in range(len(qlists)-1):
        for j in range(len(qlists)-1):
            if qlists[j][0]>qlists[j+1][0]:
                temp = qlists[j]
                qlists[j] = qlists[j+1]
                qlists[j+1] = temp

    finalAnswer.append(multintersect(qlists))
print(finalAnswer)
```

As seen above the qlists are sorted in descending order of document frequency to optimize query search speeds by drastically cutting down comparisons

### Execution



```
Python 3.6.3 Shell
File Edit Shell Debug Options Window Help
Python 3.6.3 (v3.6.3:2c5fed8, Oct 3 2017, 17:26:49) [MSC v.1900 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\Nithin\Desktop\IR Project\IR_System_Q3.py =====
Input your Query (You can use AND, OR operators): nithin AND canada
[['doc1.txt']]
>>>
===== RESTART: C:\Users\Nithin\Desktop\IR Project\IR_System_Q3.py =====
Input your Query (You can use AND, OR operators): canada
[['doc1.txt', 'doc2.txt']]
>>> |
```

## **Conclusion**

Thus, a simple IR system has been built based on the Boolean Model. There are a couple of small limitations,

- The files to be tested should be put in the "documents" directory
- The query should be entered in a proper format ("term1" AND "term2" OR ....) i.e. Free queries are not allowed

Nevertheless, for case where high precision of results is favored over high recall, this application is suitable