DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY (BHU)
VARANASI

SEARCH ENGINE FOR GIVEN SET OF DOCUMENTS

by-Animesh Anand

Roll No.- 15075005

PROJECT GUIDE: Dr. Ravindranath Chowdary C

ABSTRACT

The aim of this project is to develop a web tool which produces a generic search result from a given set of documents. Our approach to get faster search result is purely based on building inverted index of given documents. We have ranked the search results produced of any query asked, by the assigning score to documents of search results. Higher the score,the document will be displayed on higher place of search result.The indexing is done by assigning a document ID to each document.Our index will have the record of any word and in which documents it is present.While ranking is done by simply number of maximum match of words (from the query asked) from a document.

ACKNOWLEDGEMENT

This document is prepared by the guidance received from Assistant Professor Dr. Ravindranath Chowdary C, Department of Computer Science and Engineering, Indian Institute of Technology, Varanasi. His contributions are gratefully acknowledged.

-Animesh Anand

CERTIFICATE

It is certified that work contained in the project report titled **'Search Engine in Given Set of Documents'** by **Animesh Anand** has been carried out under my supervision and that this work is not submitted elsewhere for a degree.

Signature

Dr. Ravindranath Chowdary C

Computer Science & Engineering

IIT(BHU), Varanasi

TABLE OF CONTENTS

1 INTRODUCTION

In our world, the information overload is challenging. Today we have huge amount of data on World Wide Web. To sort out and retrieve information for recent and future use is not an easy task.To get information from the billions of documents in efficient time and efficient manner is not so easy. Therefore, the need of a Search Engine is obvious. In simplest terms, a search engine is any software program that searches for sites based on the words users have keyed in, that have to do with the subject of their interest.Search Engines essentially act as filters for the wealth of information available on the Internet. They allow users to quickly and easily find information that is of genuine interest or value to them, without the need to wade through numerous irrelevant web pages. There is a lot of filtering to do - In 2004 the number of pages in Google's index exceeded the number of people of the planet, reaching the staggering figure of over 8 billion. With that much content out there, the Internet would be essentially unworkable without the Search Engines, with Internet users drowning in sea of irrelevant information and shrill marketing messages.

The goal of the Search Engines is to provide users with search results that lead to relevant information on high-quality websites. The operative word here is "relevant". To attain and retain market share in online searches, Search Engines need to make sure they deliver results that are relevant to what their users search for. They do this by maintaining databases of web pages, which they develop by using automated programs known as "spiders" or "robots" to collect information. The Search Engines use complex algorithms to assess websites and web pages and assign them a ranking for relevant search phrases. These algorithms are jealously guarded and frequently updated. Google looks at over 200 different metrics when assessing websites, including copy, in-bound links, website usability and information architecture.

Since we have lots of data to look for a query the user have keyed in. If we look for the query user have asked ,the time taken to traverse through each document and find the information will be very high. So to reduce the time to retrieve information here we have built inverted index of the documents available.The index will have the words and list of document ID in which it is present. Each document will have assigned their unique document ID corresponding to it.At first we split the query into words, then we look for each word of query into the index we have built before. For each word we will look in which *docIDs* it is present. Finally we will sum up which *docID* have maximum number of words of query. According to matches of words we

will rank the results. Higher the number of number of words in which *docID* is present that document will be ranked higher.

In this way we find the results for the query asked by the user by the search engine in the faster way. The algorithm for the building index and ranking the result of search is explained in later section.
The Modern Search engines use many more complex algorithms to retrieve the information from the data available on World Wide Web and rank them according to user relevance to the information. But the basic thing they do to build index of documents.

2 BUILDING INVERTED-INDEX OF DOCUMENTS

To gain the speed benefits of indexing at retrieval time, we have to build the index in advance. The major steps in this are:

1. Collect the documents to be indexed:
Collect all the documents in which we will loo for the queries asked by the user.

2. Tokenize the text, turning each document into a list of tokens:
The documents we have collected,we assign a *docID* to each document.

3. Do linguistic preprocessing, producing a list of normalized tokens, which are the indexing terms:
We will make the dictionary of documents name as a key and a unique ID corresponding to it,i.e., we serialise the documents. We list the documents in a list also and the the document name will be present in list at the index of its respective serial no (*docID*)

4.Index the documents that each term occurs in by creating an inverted index, consisting of a dictionary and postings.

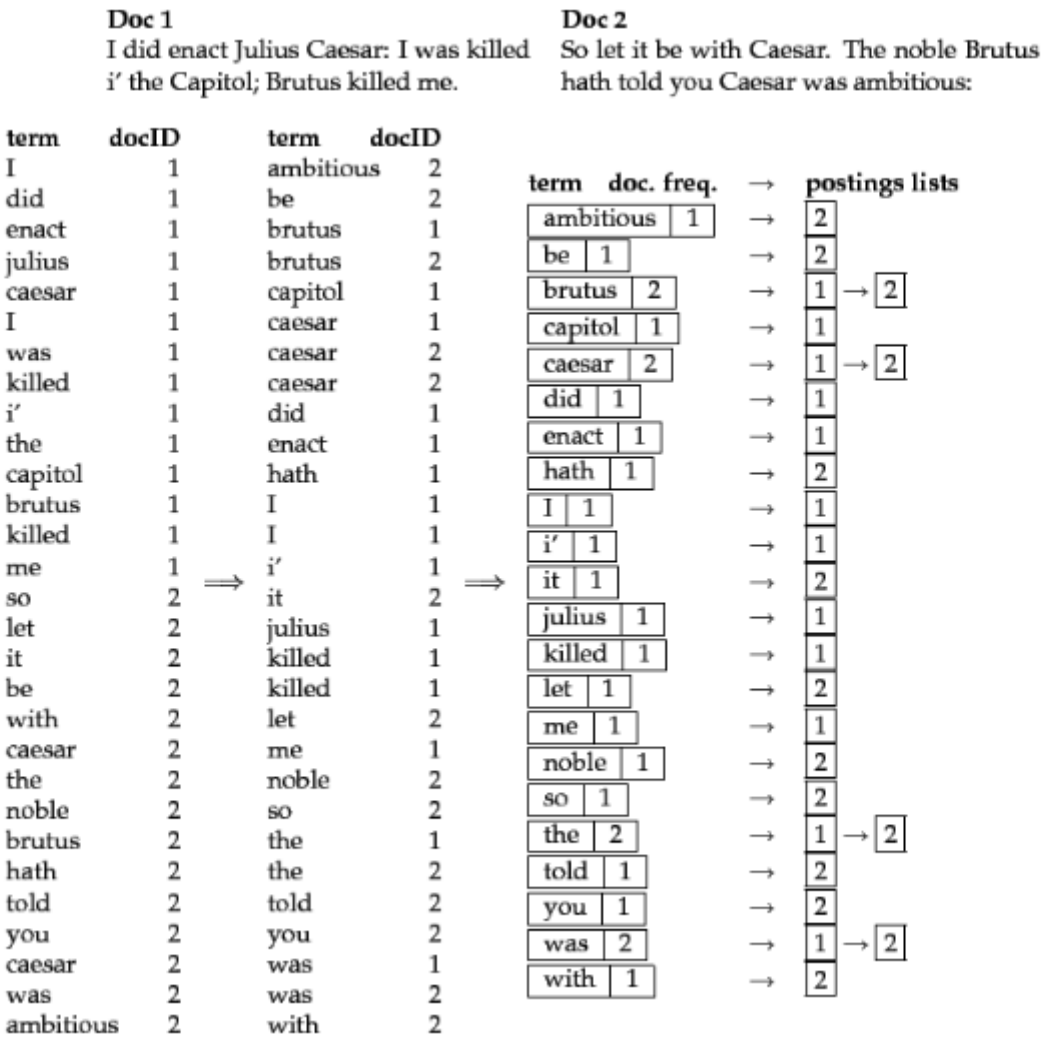
The pseudo code for the steps 1-3 is shown below.

```
import os
cur_dir=os.getcwd()
search_dir=cur_dir+"\search_files"
#search_dir::the directory in which the all documents are collected
Documents=[]
#list of wll have all the documents
docid={}
#Dictionary will have all the documents with a dcID
for files in os.walk(search_dir):
    Documents=files[2]
"""Now all the documents are listed"""
x=0
for i in l:
    docid[i]=x
    x+=1
"""
The dictioanary now contains all the documents with a ID
Notice the document is present in list at the same index as it's docID
"""
```

We will define and discuss the earlier stages of processing, that is, steps 1-3. Until then we can think of *tokens* and *normalized tokens* as also loosely equivalent to *words*.Since we have done first 3 steps already. And we examine building a basic inverted index by *sort-based indexing*.

Consider two Documents:

Doc 1 and Doc 2 as shown below and their corresponding index



Within a document collection, we assume that each document has a unique serial number, known as the document identifier *docID* . During index construction, we can simply assign successive integers to each new document when it is first encountered. The input to indexing is a list of normalized tokens for each document, which we can equally think of as a list of pairs of term and docID ,as shown above in figure. Since a term generally occurs in a number of documents, this data organization already reduces the storage

requirements of the index. The dictionary also records some statistics, such as the number of documents which contain each term (the *document frequency*, which is here also the length of each postings list). This information is not vital for a basic Boolean search engine, but it allows us to improve the efficiency of the search engine at query time, and it is a statistic later used in many ranked retrieval models. The postings are secondarily sorted by docID. This provides the basis for efficient query processing. This inverted index structure is essentially without rivals as the most efficient structure for supporting ad hoc text search.

3 SEARCHING FOR THE RESULTS AND RANKING IT

The way to avoid linearly scanning the texts for each query is to *index* the documents in advance.Since we have built the index before.
When user searches for a query then we split it into words.We look into index for each word. Since each word is hashed as key in the *index* and contains list in which *docIDs* it is present.The pseudo code below shows how it search and its ranking is implemented.

```
from sets import Set
ignore=['is','am','are','in','on','to'] # words to be ignored
display=[] #the list will contain address of documents in ranked way
query=raw_input("Enter the query to be searched\n") #the query by user
query=list(Set(query.split()))
doclist={}
""" dictionary will contain name of document as key and how many
number of words it contains from the query """
for word in query:
    if word in ignore:
        continue
    if word in di:
        for i in di[word]:
            if i in doclist:
                doclist[i]+=1
            else:
                doclist[i]=1
for w in sorted(doclist, key=doclist.get, reverse=True):
  display.append(cur_dir+'\\search_files\\'+w)
"""now display contains doc names in sorted order of number of words
containing from the query.If no match is there,list will be empty.
"""
if display==[]:
    print "No results found"
```

So the maximum numbers of words of the query the document will have, it will be ranked higher.

4 CONCLUSION

We have presented in the project a simple Search engine which searches in given set of text documents. Since traversing through each documents finding the query, it will take a long time.To reduce the time for searching we have built the *index* of the documents available in which searching operation is to take place. And we have displayed the documents in the order, in which document the number of words of query is present higher, ranked higher.

5 REFERENCES

1. Building the index of documents
https://nlp.stanford.edu/IR-book/html/htmledition/a-first-take-at-building-an-inverted-index-1.

2. Searching for the query from the index built
https://nlp.stanford.edu/IR-book/html/htmledition/an-example-information-retrieval-problem-1

3. https://docs.djangoproject.com/en/1.11/

4. https://docs.python.org/2/