Search Engine Based On Permuterm Index

J.S.Sunil
(S20160010033)

K.Sai Prasanna Kumar
(S20160010047)

N.Prudhvi Krishna
(*S20160010057)*

Upendra Sainath Reddy
(S20160010038)

***Abstract*—The main area in spotlight is to implement a search engine based on permuterm index to handle wildcard query. Google's search with wildcard queries doesn't give us proper results. This will help us get familiar with its importance in practical scenarios.**

Keywords—Permuterm Index, Information, wild card query, Dictionary, Lucene.

I. INTRODUCTION

Generally Wild card queries are used in situations like when the user is uncertain of spelling of the query term. (e.g., Sydney vs. Sidney, which leads to wild card query S*dney). when user is aware of multiple variants of spelling a term and seeks documents containing variants of a term (e.g., colour vs. color);. when the user seeks documents containing variants of a term that would be caught by stemming but is unsure whether the search engine performs stemming (e.g., judicial vs. judiciary, leading to the wildcard query judicia*); the user is uncertain of the correct rendition of a foreign word or phrase (e.g., the query Universit* Stuttgart). We'll express given wild card query as Boolean on special constructed index to answer to user's query and matching them with indexes created and discarding vocabulary terms that do not match query from specially constructed index.

A Permuterm index is one of the efficient ways of handling these wild card queries. A general wild card query can have any number of * at any position. A Permuterm index uses a special symbol $ to mark the end of the term. It contains various rotations of each term augmented with $ all linked to the original vocabulary term. It can also be referred to as term mapping or Permuterm tree. To create Permuterm index we look at every term that goes into the standard inverted index. We then need to create rotations of that term.

II. METHODOLOGY

Rotation Description

These are all the possible permutations of the term. We mark the end of the term by a '$' sign. This helps us in identifying where the original term ended. For example, Rotations of term hello are:

hello$ ->ello$h -> llo$he -> lo$h\el -> o$hell -> $hello

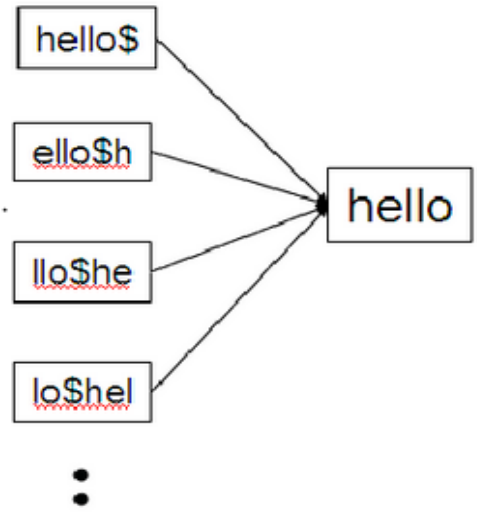


Figure 3.3: A portion of a permuterm index.

The Posting List of all rotations points to the original term (hello). All the rotations and their posting list are stored in a Permuterm Dictionary.

The idea behind Permuterm Index is to rotate wildcard query such that ***** goes to the end. Thus, you convert vague query to comparable query. As you wrote, lookup **$X*** for query **X*** because * is uncertain but the start part X is deterministic.

When it comes to **X**, we have two stars. The problem is which star should we r012otate.

1. Rotate 1st star

   You regard **X*** as one-part **Y**, then we get ***Y**. So, we should lookup **Y$***, which is **X*$***, equivalent to **X***.

2. Rotate 2nd star

   You regard ***X** as one-part **Y**, then we get **Y***. So we should lookup ***$Y**, which is ***$*. This** is not easy to handle with.

Based on that, we can know why we lookup **X*** when we have queries like ***X***. The reason is not **$** is that **$** means the end of the word whereas our query doesn't contain information about the end.

Query Look-Up

When a user enters a wild card query say X*Y, We need to first perform a look-up of Y$X* in Permuterm dictionary i.e. we want to rotate first so as to push wild card character towards the end and then perform look-up. For example for query = hel*o We will look up in Permuterm dictionary o$hel* which will lead us to different terms in posting list some of which can be hello, helio, helmo. Once these terms are available we lookup these in the inverted index dictionary to bring up the required documents.

It is important to note that Permuterm index usually quadruples the size of the dictionary. So it is required to consider efficient data structure like B-Tree for avoiding problems (heap space, large time complexity).

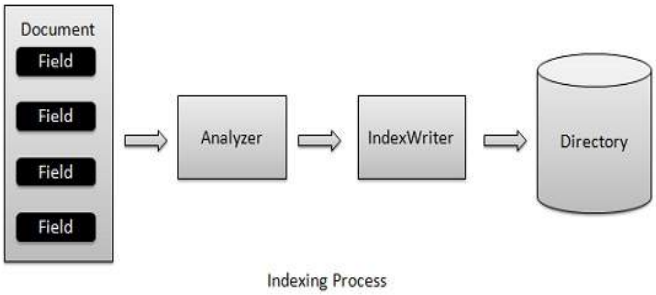
III. HOW DOES APACHE LUCENE WORKS

How does Apache Lucene work towards documents indexing and Searching?

Lucene Index:

A Lucene index is an inverted index manages an index over a dynamic collection of documents and provides very rapid updates to the index as documents are added to and deleted from the collection. Lucene indexes *terms*, which means that Lucene search searches over terms. A term combines a field name with a token. The terms created from the non-text fields in the document are pairs consisting of the field name and the field value. The terms created from text fields are pairs of field name and token.

The Lucene index provides a mapping from terms to documents. This is called an inverted index because it reverses the usual mapping of a document to the terms it contains. The inverted index provides the mechanism for scoring search results: if a number of search terms all map to the same document, then that document is likely to be relevant.



Indexing Process

1.Indexing Documents:

Document indexing consists of first constructing a document that contains the fields to be indexed or stored, then adding that document to the index. The key classes involved in indexing are , `oal.index.IndexWriter` which is responsible for adding documents to an index, and, `oal.store.Directory` which is the storage abstraction used for the index itself. Directories provide an interface that's similar to an operating system's file system. A `Directory` contains any number of sub-indexes called *segments*. Maintaining the index as a set of segments allows Lucene to rapidly update and delete documents from the index.

2.Document Search and Search Ranking:

The Lucene search API takes a search query and returns a set of documents ranked by relevancy with documents most similar to the query having the highest score. Lucene provides a highly configurable hybrid form of search that combines exact Boolean searches with softer, more relevance-ranking-oriented vector-space search methods. All searches are field specific because Lucene indexes terms and a term is composed of a field name and a token. The terms to generate the inverted index should be space tokenized terms from the documents

PROCEDURE:-

Given a set of documents create an inverted index of the set of terms in documents (excluding stop words) for each term in the inverted index you need to create a permute term index and its posting list.

1.Stop word removal
2.Generating permuterms for terms indexed.

Document Searching : -
1.Check whether query has one * or two *'s
2.
(i)If query has 1 '*':

a) Rotate term so as to push * character towards the end.

b) Perform look up in Permuterm dictionary.

(ii) If query has 2 '*'s:

E.g. : pr*s*n
a) Start with n$pr*.
b) Filter out all results not containing ‘s’ in the middle(exhaustive).
c) Look up the found terms in the standard inverted index.

IV. RESULTS

Obtained results from our Search Engine on Dataset
INPUT: - s*n*e
OUTPUT: - [snake, Snake, sense, single, someone, snakeThe]
Total Length: 13
Path : inputFiles\file3.txt, Score : 2.6777725
Path : inputFiles\file4.txt, Score : 1.9376903
Path : inputFiles\file4.txt, Score : 1.883633
Path : inputFiles\file2.txt, Score : 1.698161
Path : inputFiles\file1.txt, Score : 1.6672187
Path : inputFiles\file8.txt, Score : 1.640195
Path : inputFiles\file4.txt, Score : 1.6072798
Path : inputFiles\file6.txt, Score : 1.4801198
Path : inputFiles\file10.txt, Score : 1.179204
Path : inputFiles\file3.txt, Score : 1.0278574
Path : inputFiles\file9.txt, Score : 1.0052719
Path : inputFiles\file5.txt, Score : 0.89309055
Path : inputFiles\file6.txt, Score : 0.74049443

V. CONCLUSION

Thus we are able to build search engine based on Permuterm index obtained scores based on searching. Permuterm index is quite large since it contains all rotations of each term. On an average 10 times for English documents. Thus it is helpful for users who are uncertain of spelling of query and also useful to search for query which has multiple variants of spelling a term.

VI. REFERENCES

[1] https://stackoverflow.com/questions/50914724/how-does-a-permuterm-index-works.
[2] https://dzone.com/articles/apache-lucene-a-high-performance-and-full-featured.
[3] An Introduction to Information Retrieval. Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze.