Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X
IMPACT FACTOR: 5.258

IJCSMC, Vol. 5, Issue. 6, June 2016, pg.331 – 336

Inverted Index for Fast Nearest Neighbour

Sayali Borse[1], Prof. P. M. Chawan[2]

[1]P.G. Student, Department of Computer Engineering & IT, V.J.T.I., Mumbai, Maharashtra, India

[2]Associate Professor, Department of Computer Engineering & IT, V.J.T.I., Mumbai, Maharashtra, India

Abstract - ***Users may search for different things from anywhere and anytime. But Search results depend on user entered query which has to satisfy their searched predicates that is stored in the spatial database. Many applications are developed for finding closest objects to a specified location containing a set of keywords entered by users. For example, nearest neighbor restaurants search allow users to enter an address and a set of keywords. In return, the user obtains a list of restaurants whose menu list contains these keywords, ordered by their distance from the entered address. Nearest neighbor search problems have been broadly studied separately on spatial database and keyword search on text data .An efficient algorithm for spatial databases, the various researchers delivered various approaches for finding the result based on the keywords. However, to the best of our knowledge there is no efficient method to answer spatial keyword queries, that specify both a location and a set of keywords. Motivated by this, we arrive at a new method known as spatial inverted index which extends the conventional inverted index to manage multidimensional spatial data and comes with algorithms that can create inverted index for searching nearest neighbor restaurants with keywords in real time.***

Keywords - Spatial data, Spatial queries, Spatial keyword search, Spatial inverted index, Nearest Neighbour Search

I. INTRODUCTION

The World-Wide Web has outstretch a size where it is becoming very challenging to satisfy certain information needs. While current search engines are still capable of indexing a reasonable subset of the web, the pages a user is really looking for are often hidden in hundreds of thousands of less interesting results. An increasing number of applications need the effective execution of nearest neighbor (NN) queries which satisfy the predicates on the spatial objects in spatial databases. Particularly on the Internet, due to the popularity of keyword search, many of these applications permit the user to provide a list of keywords that the spatial objects should contain, in their description or other attribute. For example, nearest neighbor restaurants search allow users to enter an address and a set of keywords. In return, the user obtains a list of restaurants whose menu list contains these keywords, ordered by their distance from the entered address.

Spatial databases are the databases used to hold geographical information. They contain data which are multidimensional in nature. A spatial database manages the multidimensional objects such as points and rectangles, etc. Such databases can be used for obtaining information based on queries and provides fast access to which objects based on the different the selection criteria. Today, the extensive use of search engines has made it reasonable to write spatial queries in a new way.

Traditionally, queries focuses on only geometric properties of objects, like whether a point is in a rectangle or in specified areas or how close two points are from each other. We have seen some modern applications that demand the capability to select objects which satisfies both of their geometric coordinates and their associated texts. Unfortunately there is no efficient support for top-k spatial keyword queries, where a prefix of the results list is required. Instead, combinations of keyword search techniques and nearest neighbor (NN) are used by current systems to tackle the problem. Similarly, an R-Tree is used to find the nearest neighbors and for each neighbor an inverted index is used to check if the query keywords are included. We show that how to create inverted index for fast searching in spatial databases.

Users use keyword search to retrieve restaurants which contains entered menu items by simply typing in keywords (menu items) as queries. Current keyword search systems normally use an inverted index, a data structure that maps each word in the dataset to a list of IDs of restaurants in which the keyword appears to efficiently retrieve restaurants. The inverted index for a restaurants collection consists of a set of so-called inverted lists, known as posting lists. Each inverted list corresponds to a menu item, which stores all the IDs of restaurants where this menu items appears. In practice, real world datasets are so large that keyword search systems normally use various compression techniques to reduce the space cost of storing inverted indexes. Compression of inverted index not

only reduces the space cost, but also leads to less disk I/O time during query processing.

II. RELATED WORK

Suffix trees and suffix arrays are efficient data structures which can be used to index a text and support searching for any arbitrary pattern. These data structures can be maintained in linear space and can report all the occurrence of a pattern P in optimal (or nearly optimal) time. The space efficient versions of suffix trees and suffix arrays are called compressed suffix trees and compressed suffix arrays, respectively, which take space close to the size of the indexed text. From a collection D of |D| documents {d1, d2, ..., d|D|} of total length n, the problem of reporting documents containing a query pattern P is called the "document listing" problem. This problem was first studied by Matias, where they proposed a linear space index with O(p log n+|output|) query time; here, p denotes the length of the input pattern P and |output| denotes the number of the qualified documents in the output. An index with optimal O(p+|output|) query time was later achieved in. Sadakane showed how to solve the document listing problem using succinct data structures, which take space very close to that of the compressed text. He also showed how to compute the tfidf of each document with the proposed data structures. Similar work was also done by Valimaki and Makinen where they derived alternative succinct data structures for the problem.

In many practical situations, we may be interested in only a few documents which are highly relevant to the query. Relevance ranking refers to the ranking of the documents in some order, so that the result returned first is what the user is most interested. This can be the document where the given query pattern occurs most number of times (frequency). The relevance can also be defined by a similarity metric, such as the proximity of the query pattern to a certain word or to another pattern. This problem is modeled as top-k document retrieval, where the task is to retrieve the k highest scoring documents based on some score function.

A. Prior Work on Doc ID Assignment The compressed size of an inverted list, and thus the entire inverted index, is a function of the d-gaps being compressed, which itself depends on how we assign docIDs to documents (or columns to documents in the matrix). usual integer compression algorithm want fewer bits to represent ta smaller integer than a larger one, but the number of bits required is typically less than linear in the value. This means that if we assign docIDs to documents such that we get many small d-gaps, and a few larger d-gaps, the resulting inverted list will be more compressible than another list with the same average value but more uniform gaps. This is the insight that has motivated all the recent work on optimizing docID assignment [4]-[5] .Note that this work is related in large part to the more general topic of sparse matrix compression [6], with parallel lines of work often existing between the two fields.

III. INVERTED INDEX

An inverted index of file is an index data structure used for storing content of original file in compressed form, such as digits, to its place in a database file, or in a file or a collection of documents. The main motive of an inverted index structure is to allow fast searching of text with increased processing speed when a document is added to the original database. It is possible that the inverted data may be the database file itself, rather than index of that file. It is the most popular data structure used for document storing and accessing the systems, used on a big scale for example in google.

In our example restaurants are normally stored as lists of menu items, but inverted indexes invert this by storing for each menu item (keyword) the list of restaurants that the menu item (keyword) appears in, hence the name "Inverted index". There are several variations on inverted indexes. At a minimum, you need to store for each menu item (keyword) the list of restaurants that the menu item (keyword) appears in.
Example :

rest_id	famous_for
1	Pure Veg, South Indian
2	Pure Veg, South Indian
3	South Indian, Beverages
4	Pure Veg, South Indian
5	North Indian, Mughlai, Chinese
. . . . .	. . . . .
144	Deserts,Ice Cream
145	Chinese,Fast food
146	Ice Cream,Deserts
147	North Indian, South Indian,Chinese
148	Fast food, Caf
149	North Indian, Chinese, Manglorean, Malwani,Seafood
150	Fast food, Tex-mex

I. Dataset

word	word_present
Pure Veg	1, 2, 4, 11, 20, 25, 40, 57, 95, 99, 143
South Indian	1, 2, 3, 4, 9, 11, 12, 18, 20, 22, 23, 25, 26, 29,124, 141, 147
North Indian	5, 7, 8, 11, 13, 14, 15, 16, 112, 113, 115, 118, 119, 124, 125, 129, 133, 135, 147, 149
Mughlai	5, 7, 13, 14, 15, 16, 38, 58, 62, 65, 82, 93
. . .	. . .
Goan	112, 113, 117, 132
Konkan	117, 132
Finger food	125
Gujarati	126
Rajasthani	126
Snack Bar	134
Manglorean	149

II. Inverted index for keyword search (from table I)

Given the texts
T [0] = "Pure Veg, South Indian"
T [1] = "South Indian, Beverages, Chinese"
T [2] = "North Indian, South Indian, Chinese"
T [3] = "North Indian, Chinese"

We have the following inverted file index (where the integers in the set notation brackets refer to the indexes (or keys) of the text symbols, T[0], T[1] etc.):

"Pure Veg" : {0}
"South Indian" : {0, 1, 2}
"North Indian" : {2, 3}
"Chinese" : {1, 2, 3}
"Beverages" : {1}

A term search for “South Indian” and “Chinese” would give the set

{0, 1, 2} ∩ {1, 2, 3} = {1, 2}

IV. SEARCH INDEX ALGORITHMS

Any keyword search method generally helps union and the intersection operations on inverted lists. The union operation is a basic operation to support OR query in which each and every data file that include at least one of the query keywords is returned as an output. The intersection operation is used to support AND query semantics, in which only those data files that include all the query keywords are returned.

A. Union operation

As in set theory, the union of a set of ID lists, denoted by S ={S1, S2, … , Sn}, is another ID list, in which each ID is contained in at least one ID list in S. For example, consider the following three ID lists:

Ice Cream : { 37, 41, 49, 54, 56, 59, 68, 87, 92, 102, 104, 123, 136, 144, 146 },

Deserts : { 37, 41, 48, 49, 54, 56, 59, 64, 66, 68, 77, 79, 87, 89, 92, 98, 100, 102, 104, 109, 116, 123, 130, 136, 144, 146 } and

Beverages : { 49, 54, 64, 66, 76, 77, 89, 100, 130 }

The union of these three ID lists is
{ 37, 41, 49, 54, 56, 59, 64, 66, 68, 76, 77, 79, 87, 89, 92, 98, 100, 102, 104, 109, 116, 123, 130, 136, 144, 146 }.

B. Intersection operation

The intersection operation, calculates the intersection list of a set of ordered lists. As with the definitions of the union of ID lists, the intersection of ID lists can be defined. Consider the three ID lists that we have used previously:

Ice Cream : { 37, 41, 49, 54, 56, 59, 68, 87, 92, 102, 104, 123, 136, 144, 146 },

Deserts : { 37, 41, 48, 49, 54, 56, 59, 64, 66, 68, 77, 79, 87, 89, 92, 98, 100, 102, 104, 109, 116, 123, 130, 136, 144, 146 } and

Beverages : { 49, 54, 64, 66, 76, 77, 89, 100, 130 }.

The intersection list of these ID lists is
{49, 54}.

V. CONCLUSION

To construct inverted index is an important issue in web search engines. This paper introduces the first practical version of inverted index for restaurants search. The idea is to store lists for a menu items (or keywords) in a conditionally sorted manner. Inverted Index include an effective index structure and algorithms to support keyword search. To enhance the search speed of Inverted Index, fast scalable methods are used by reordering restaurants in the database. Fewer intervals are introduced in Inverted Index, for fast searching by search algorithms. On the other hand, interval lists containing fewer intervals will require less storage space. Therefore, the search speed and the space cost are both improved by reducing the number of intervals in Inverted Index.

REFERENCES

[1]. X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi. Collective spatial keyword querying. In Proc. of ACM Management of Data (SIGMOD), pages 373–384, 2011.

[2]. X. Cao, G. Cong, and C. S. Jensen. Retrieving top-k prestige-based relevant spatial web objects. PVLDB, 3(1):373–384, 2010.

[3]. I. D. Felipe, V. Hristidis, and N. Rishe. Keyword search on spatial databases. In Proc. of International Conference on Data Engineering (ICDE), pages 656–665, 2008.

[4]. R. Blanco and A. Barreiro. Characterization of a simple case of the reassignment of document identifiers as a pattern sequencing problem. In Proc. of the 28th annual int. ACM SIGIR conference on Research and development in inf. retrieval, 2005.

[5]. R. Blanco and A. Barreiro. Tsp and cluster-based solutions to the reassignment of document identifiers. Inf. Retr., 9(4):499–517, 2006

[6]. D. Johnson, S. Krishnan, J. Chhugani, S. Kumar, and S. Venkatasubramanian. Compressing large boolean matrices using reordering techniques. In 30th Int. Conf. on Very Large Data Bases (VLDB 2004), August 2004.

