GEO-TEXTUAL INDEX STRUCTURE FOR APPROXIMATE KEYWORD SEARCH WITHIN GIVEN RANGE ON SPATIAL DATABASE

[1]SU NANDAR AUNG, [2]MYINT MYINT SEIN

University of Computer Studies, Yangon (Myanmar), Research and Development Department
(University of Computer Studies, Yangon, Myanmar)
E-mail: sunandaraung@ucsy.edu.mm, myint@ucsy.edu.mm

Abstract- There is more and more commercial and research interest in nearest neighbor objects for location-based search from spatial database. Spatial keyword search has been well studied for years due to its importance to commercial search engines. Specially, a spatial keyword query takes a user location and user-supplied keywords as arguments and returns objects that is nearest k objects from user current location and textually relevant to the user required keyword. In these systems, inconsistencies and errors can exist in the user's typed queries. To bridge the gap between queries and data, it is important to support approximate keyword search on spatial database. In this paper we study how to answer such queries efficiently. A number of geo-textual indices have been proposed in recent years which mainly combine the R-tree and its variants and the inverted file. This paper proposes new index structure that combines K-d tree and inverted file. The proposed index structure supports a wide range of query predicates efficiently. We also discuss approximate keyword search for the correct use's mistyped keywords.

Keywords- Combination Scheme, Problem Statement, Proposed System, Range Keyword Search Algorithm, System Architecture.

I. INTRODUCTION

Spatial database systems manage large collections of spatial data, which apart from spatial attributes contain non spatial information. Spatial data are data that have a location (spatial) and mainly required for Geographic Information Systems (GIS) whose information is related to geographic locations. GIS model supports spatial data types, such as point, line and polygon. A geospatial collections increase in size, the demand of efficient processing of spatial queries with text constraints becomes more prevalent. Due to the popularity of keyword search, particularly on the Internet, many of these applications allow the user to provide a list of keywords that the spatial objects should contain, in their name or description or categories. Spatial keyword search is an important tool in exploring useful information from a spatial database and has been studied for years. In reality, for many scenarios, keyword search for retrieving approximate string matches is required. The spatial keyword query consists of a spatial location, a set of keywords and the answer is a list of objects ranked according to a combination of their distance to the query point and the relevance of their text description to the query keyword. Keyword search over a large amount of data is an important operation in a wide range of domains. Approximate string search could be necessary when users have a fuzzy search condition or simply a spelling error when submitting the query. In the context of spatial databases approximate string search could be combined with any type of spatial queries, including range and nearest neighbor queries. For example, "find all objects within a spatial range r that have a description that is similar to shopping center". Similar example could be constructed for k nearest neighbor (kNN) queries. A key issue in understanding the semantics of these queries is to define the similarity between two strings. The edit distance metric is often adopted for such approximate string queries.

During the design of a spatial index, issues that need to be minimized are:
(a) The area of covering rectangles maintained in internal nodes.
(b) The overlaps between covering rectangles for indexes developed based on the overlapping native space indexing approach.
(c) The number of objects being duplicated for indexes developed based on the non-overlapping native space indexing approach.
(d) The index shows effective storage utilization.
(e) The index answers queries efficiently.

Many index structures that have been proposed in recent years mainly use R-tree and then combine with inverted file, namely the families of IR-tree [4, 5, 6, 7, 8, 9, and 10]. All use R-tree for spatial (latitude/longitude) index and inverted file for textual index and they only support efficient keyword search with exact matches. They all created hybrid index structure according to the combination schemes: (1) Text first loose combination scheme, employs the inverted as the top-level index and then arrange the postings in each inverted list in a spatial structure. (2) Spatial-first loose combination scheme employs the spatial index as the top-level index and its leaf nodes

contain inverted files or bitmaps for the text information of objects contained in the nodes. (3) Tight combination index combines a spatial and a text index tightly such that both types of information can be used to prune the search space simultaneously during query processing.

The construction of an efficient index structure should take into account overlaps between nodes and coverage of a node. Minimization of a node coverage leads to more precise searching within the tree and minimization of the overlap between nodes reduces the number of paths tested in the tree during a search that can reduce search time. As the data objects in the R-tree can be overlapping and covering each other, the search process in the R-tree might suffer from unnecessary node visits and higher IO cost. Moreover, the IR-trees suffer from high update cost. Each node has to maintain an inverted index for all the keywords of documents associated with this node's MBR. When a node is full and split into two new nodes, all the textual information in the node has to be re-organized. As the R-tree need to reorganized, it suffers from higher CUP costs. This paper intends to reduce IO costs, CUP costs and searching time for kNN keyword search.

This paper includes the following contributions:

1) The main contribution is to create index structure that combine K-d tree and inverted file for efficiently process spatial keyword queries within minimum time.

2) Range keyword search algorithm is developed using the proposed index structure to efficiently answer range queries and to explore useful information that contains user required keywords using approximate string matching method.

3) The own dataset is created for Yangon (Myanmar) region which contains latitude, longitude, name, description and category type of each object.

II. SPATIAL KEYWORD QUERIES AND PROBLEM STATEMENTS

Standard spatial keyword queries involve different conditions on the spatial and textual aspects of places. In spatial databases, the arguably most fundamental queries are range queries and k nearest neighbor queries. In text retrieval, queries may be Boolean, requiring results to contain the query keywords, or ranking-based, returning the k places that rank the highest according to a text similarity function.

Three types of spatial keyword queries are receiving particular attention. The Boolean range query $q = (\rho, \psi)$ where ρ is a spatial region and ψ is a set of keywords, returns all places that are located in region ρ and that contain all the keywords in ψ. Variations of this query may rank the qualifying places. The Boolean kNN query $q=(\lambda, \psi, k)$ takes three arguments, where λ is a point location, ψ is as above, and k is the number of places to return. The result consists of up to k places, each of which contains all the keywords in ψ, ranked in increasing spatial distance from λ. Next, the top-k range query $q = (\rho, \psi, k)$ where ρ, ψ, and k are as above, returns up to k places that are located in the query region ρ, now ranked according to their text relevance to ψ. Finally, the top-k kNN query takes the same arguments as the Boolean kNN query. It retrieves k objects ranked according to a score that takes into consideration spatial proximity and text relevance. Among these queries, the latter two ones that perform textual ranking are the most similar to standard web querying, and the last one is the one that is most interesting and novel, as it integrates the spatial and textual aspects in the ranking. This query, also called the top-k spatial keyword query. The propose system intends to anwer the range queries with the user's required keyword that can have typing error called range approximate keyword query. Let D is a spatial database that contains D= {o1,o2,o3…..,on} such that every object o in D has many attributes <oid, ol, od> where oid is an identifier of an object, ol is a spatial location that contain latitude and longitude and od is an text document of each object for keyword querying. Let q=<qk, qr> be a range keyword query where qk is user required keywords w1…..wm that can have inconsistencies and error and qr is the user desired range (km). A query q return all objects in D that contain all similar keywords qk={w1, w2,…., wm} and belong to the range qr .

III. APPROXIMATE STRING MATCHING

Approximate string search could be necessary when users have a spelling error when submitting the query or the strings in the inverted file contain some degree of uncertainty or errors. Approximate string matching is the technique of finding strings that match a pattern approximately. The closeness of a match is measured in terms of the number of primitive operations necessary to convert the string into an exact match. This number is called the edit distance between the string and the pattern. Edit distance is a way of quantifying how dissimilar two strings (e.g., words) are to one another by counting the minimum number of operations required to transform one string into the other. Given a query string 'sq' and a set of strings 's' from DB, estimate the size of the answer set {s | ed(sq,s) <= t and s Є DB}, where t is the edit distance threshold. One of the simplest sets of edit operations is Levenshtein Distance that was defined in 1966. The Levenshtein distance is a string metric for measuring the difference between two sequences. Informally, the Levenshtein distance between two words is the

minimum number of single-character edits (i.e. insertions, deletions or substitutions) required to change one word into the other.

Mathematically, the Levenshtein distance between two strings a,b is given by $lev_{a,b}(a|b)$ where

$$lev_{a,b}(i,j) = \begin{cases} \max(i,j) & if\ \min(i,j) = 0, \\ \min \begin{cases} lev_{a,b}(i-1,j)+1 \\ lev_{a,b}(i,j-1)+1 \\ lev_{a,b}(i-1,j-1)+1_{(a_i \neq b_i)} \end{cases} & otherwise \end{cases}$$

Where 1(ai≠bj) is the indicator function equal to 0 when ai=bj and equal to 1 otherwise.

The distance is the number of deletions, insertions, or substitutions required to transform s into t. The greater the Levenshtein distance, the more different the strings are.

IV. PROPOSED SYSTEM

The proposed system creates hybrid geo-textual index structure that integrates spatial index and text index to process spatial keyword queries efficiently.

In this proposed system K-d tree loosely combined with inverted file. K-d tree is used for spatial queries and inverted file is used for keywords information that is the most efficient index for text information retrieval. For each node of K-d tree, an inverted file is created for indexing the text components of objects contained in the node.

As K-d trees represent a disjoint partition, the proposed system can't cause more IO costs and also K-d trees don't need to rebalance the textual information so the proposed can reduce update cost (CPU Costs).

Table1. Example Dataset

id	Latitude	Longitude	Keywords
Obj1	16.779568	96.152687	Mobile, Shopping, Mall, Telecommunication, Electronics, Tools
Obj2	16.779533	96.15269	May, Shopping, Center, Super Market
Obj3	16.813517	96.08475	Cat, Walk, Foot, Wear
Obj4	16.779565	96.135581	NorthPoint, Shopping, Center, Super, Market, Food, Drink
Obj5	16.881351	96.152549	Gamonpwint, Shopping, Center, Super, Market, Food, Drink
Obj6	16.779581	96.169647	Gamonpwint, Shopping, Center, Super, Market, Food, Drink
Obj7	16.779568	96.152719	Asia, Shopping, Center, Super, Market
Obj8	16.830324	96.186432	Moon, Bakery, Food, Drink

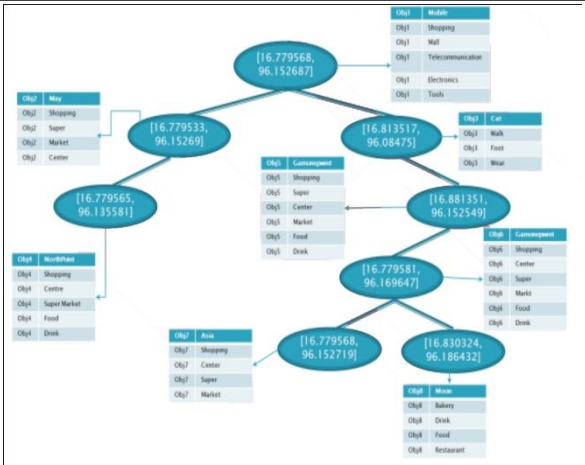


Fig1. Proposed Index Structure for Dataset of Table 1

Most geo-textual indices use the inverted file for text indexing. An inverted file has a vocabulary of terms, and each term is associated with an inverted file. K-d tree structure is known as point indexing structures as it is designed to index data objects which are points in a multi-dimensional space. It can be used efficiently for nearest neighbor queries and range queries. K-d tree provide convenient tool for range search query in databases with more than one key. The search might go down the root in both directions (left and right), but can be limited by strict inequality on key value at each tree level. K-d tree is the only data structure that allows easy multi-key search. This paper proposes range keyword search algorithm using K-d tree and inverted file. And also the propose system use Levenshtein Distance for approximate string search when the user's required keyword contains inconsistencies and errors.

V. RANGE APPROXIMATE KEYWORD SEARCH ALGORITHM

Algorithm1. Range Keyword Search in Hybrid Index Structure

Input: user's required keyword, K-d tree, query point, range, Max/Min BB

```
pq: priority queue
RANGEKEYWORDSEARCH (keyword,T,BB,q,r)
    if T=leaf then return
    p<−T.key;   i<−T.discr;
    distance <− DISTANCE(q,p);
edit_distance<−LEVENSHTEIN(p.keyword,keyword)
    if distance ≤ r and edit_distance ≤ α then
      pq.PUSH (p,distance);
    COMPUTEBOUNDINGBOXES (lBB,rBB,p[i],i)
    if INTERSECTS (lBB,q,r) then
          RANGEKEYWORDSEARCH (keyword, T.left,
          lBB, c, radius)
    if INTERSECTS (rBB,q,r) then
          RANGEKEYWORDSEARCH(keyword,
          T.right, rBB, c, radius)
```

Algorithm1 is the proposed range keyword search procedure. The procedure RANGEKEYWORDSEARCH returns all points 'p' such that d (q,p) ≤ r and Levenshtein distance (p.keyword, q.keyword) ≤ t (thresholds t=length of q.keyword /2). The procedure COMPUTEBOUNDINGBOXES ( ) calculates the bounding boxes lBB and rBB for the left and for the right sub tree, respectively. The procedure INTERSECTS (…) tells if the bounding box BB intersects with the region that satisfies the distance constraints. If the intersection is non-empty, the sub tree to be explored. The DISTANCE (…) procedure calculates the distance between two points using Euclidean distance $d(q,p) = \sqrt{(q_{lat} - p_{lat})^2 + (q_{lon} - p_{lon})^2}$. LEVENSHTEIN (…) procedure calculates the edit distance between input keyword and each point keyword.

VI. USER INTERFACE OF PROPOSED SYSTEM

The propose system adopts the browser-server model for desktop and laptop computer. Fig2 shows the user interface for the proposed system. Users can input their queries through the web browser and the queries are sent to the server for processing. After the queries are processed, the results are sent back and displayed using Google Maps in the users' browser. Queries are sent from the browser to the server by the HTTP post operation.

The browser side use Google Map API to provide interfaces to users for generating queries and viewing the returned spatial web objects. Users can specify the current's location by clicking a location in Google Map to get the latitude and longitude of that location and can type the required keywords. And then choose the specified range and then sort by type. The query is sent to the server and then relevant objects are retrieved by the server within given range and specified keyword as shown in Fig3. The results are sorted by the distance or keyword and then are displayed on Google Maps in the browser.

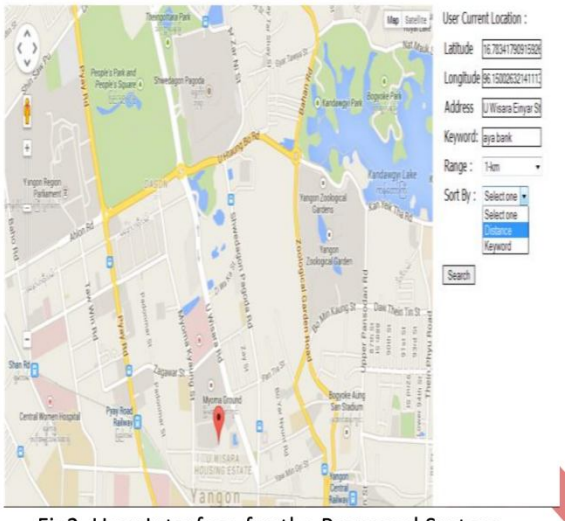


Fig2. User Interface for the Proposed System

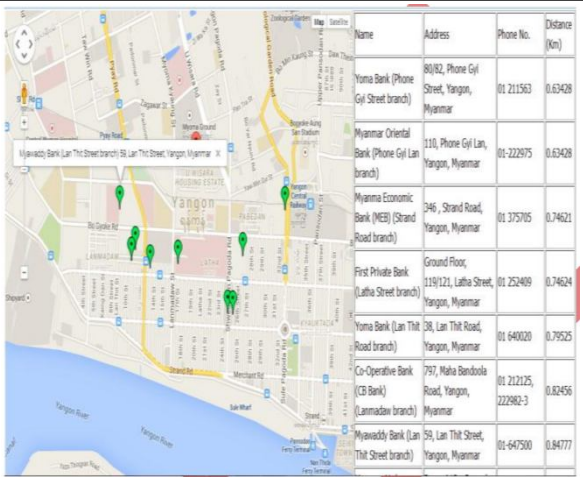


Fig3. User Interface for the Result

VII. EXPERIMENTAL RESULTS

Fig4 compare the searching time (second) between using proposed index structure and without using index. Depending on the desired range (km), searching time is varied. Searching time using index structure is faster than without using index about 100-times in second. Fig5 shows the index construction time (second) depending on the size of datasets. Also Fig6 shows the searing time with varying number of keywords within the same range.

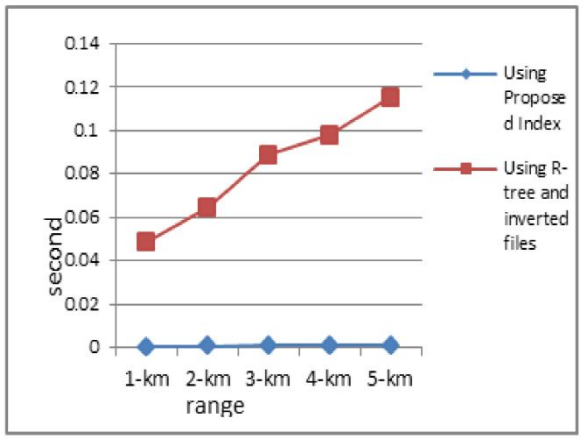


Fig4. Searching Time in range keyword search

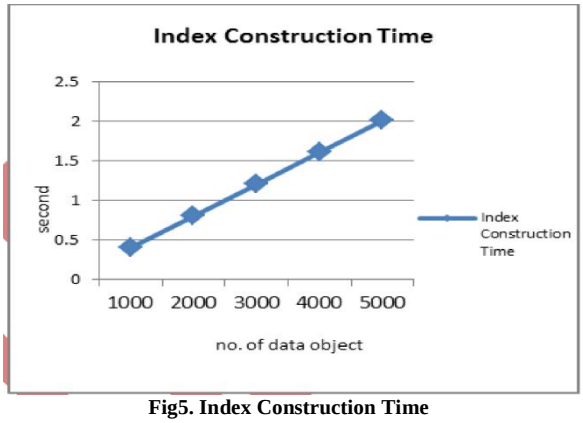


Fig5. Index Construction Time

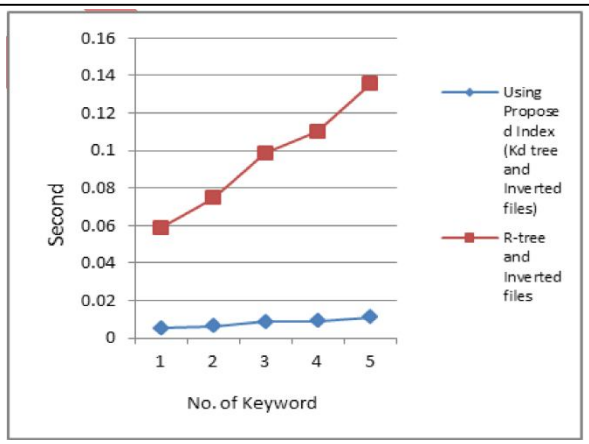


Fig 6: Result for Varying Number of Keywords

VIII. RELATED WORKS

There has been lot of interest in building geographic information retrieval system. Spatial Keyword search has been well studied for years due to its importance to commercial search engines. Various types of spatial keyword queries have been proposed. For spatial keyword search, the index structure is created for both spatial and textual relevance. Most index structures [10, 5, 6, 8, 9] use R-tree and its variants as spatial index and inverted file for text index. They all combine both indices depending on the combination schemes [15]. Among them , Felipe et al. Reference [8] integrates signature file instead of inverted file into each node of the R-tree. Inverted file-R*tree (IF-R*) and R*-tree-inverted file (R*-IF) [10] are two geo-textual indices that loosely combine the R*-tree and inverted file. Hariharan et al. Reference [8] proposed the KR*-tree. This paper proposed a framework for GIR systems and focus on indexing strategies. Reference[9] uses R*-tree for spatial index and inverted file for text index. Cary et al, proposed SKI that combines and R-tree with an inverted index by the inclusion of spatial references in posting lists. In [5] the posting list of term contains all its term bitmaps rather than documents. The IR tree [6] creates each nodes of the R-tree with a summary of the text content of the objects in the corresponding subtree. Li et al. proposed an index structure, which is also called IR tree that stores one integrated inverted file for all the nodes. Reference [3] proposed S2I index structure based on R-tree and inverted file. The objects in [3] are stored differently according to the document frequency and infrequency of the term. [ND6] proposed I3 (Integrated Inverted Index), which adopts the Quad tree structures to hierarchically partition the data space into cells. The basis unit of I3 is the keyword cell, which captures the spatial locality of a keyword.

Reference [15] proposes a Web Object Retrieval System (SWORS) that is capable of efficiently retrieving spatial web objects that satisfy spatial keyword queries. This system use IR tree and inverted file for index. It supports two types of queries that are location aware top-k text retrieval (Lkt) query and spatial keyword group (SKG) query. Reference [17] proposed the system that studies how to efficiently combine these two types of indexes, and how to search the resulting index to find answers. This paper use the IR tree index and try to bridge the gap between queries and data, it is important to support approximate keyword search on spatial database. Reference proposed the MHR-tree that supports a wide range of query including range and NN queries. This paper used the R-tree, the min-wise signature and the linear hashing technique for efficiently answering approximate string match queries in large spatial databases.

CONCLUSION AND FURTHER EXTENSION

This paper presented hybrid index structure for range keyword query searching with minimum IO costs and CPU costs and can also correct the inconsistencies of the user's input keyword. This index structure can avoid searching in overlapping area. So it can reduce searching time in overlap area. Moreover, it can't cause node overflow, so it doesn't need to re-organize the textual data and spatial data. Many Further extensions can be considered for efficient hybrid index structure for spatial database. As a further extension, we'll add an efficient nearest neighbour and approximate keyword search in this proposed index structure.

REFERENCES

[1] D. Zhang, K.L. Tan, Anthony K.H. Tung, "Scalable Top-K Spatial Keyword Search", EDBT/ICDT'13 March 18-22, 2013

[2] L. Chen, G. Cong, C. S. Jensen, and D. Wu, "Spatial Keyword Query Processing: An Experimental Evaluation", in proceddings of the VLDB Endowment, Vol.6, No.3, 2013.

[3] X. Cao, L. Chen, G. Cong, C. S. Jensen, Q. Qu, A. Skovsgaard, D. Wu, and M. L. Yiu, "Spatial keyword querying", in ER, pages 16–29, 2012.

[4] J. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nørv°ag, "Efficient processing of top-k spatial keyword queries", in SSTD, pages 205–222, 2011.

[5] Z. Li, K. C. K. Lee, B. Zheng, W.-C. Lee, D. L. Lee, and X. Wang, "Ir-tree: An efficient index for geographic document search", IEEE TKDE, 23(4):585–599, 2011.

[6] A. Cary, O. Wolfson, and N. Rishe, "Efficient and scalable method for processing top-k spatial Boolean queries", in SSDBM, pages 87–95, 2010.

[7] G. Cong, C. S. Jensen, and D. Wu, "Efficient retrieval of the top-k most relevant spatial web objects", PVLDB, 2(1):337–348, 2009.

[8] R. G¨obel, A. Henrich, R. Niemann, and D. Blank, (2009), "A hybrid index structure for geo-textual searches", in CIKM, pages 1625–1628, 2009.

[9] I. D. Felipe, V. Hristidis, and N. Rishe, "Keyword search on spatial databases", in ICDE, pages 656–665, 2008.

[10] R. Hariharan, B. Hore, C. Li, and S. Mehrotra, (2007), "Processing spatial-keyword (sk) queries in geographic information retrieval (gir) systems", in SSDBM, page 16, 2007.

[11] Y. Zhou, X. Xie, C. Wang, Y. Gong, and W.-Y. Ma, "Hybrid index structures for location-based web search", in CIKM, pages 155–162, 2005.

[12] H.M. Kakde, "Range Searching using Kd Tree", 2005.

[13] A. Guttman, "R-trees: A dynamic index structure for spatial searching", in SIGMOD, pages 47–57, 1984.

[14] B.C. Ooi, R. Sacks-Davis, J.Han, "Indexing in Spatial Databases".

[15] X.Cao, G.Cong, Christian S. Jensen, Jun.J. Ng, Beng C.Ooi, N.T. Phan, D. Wu, "SWROS: A System for the Efficient Retrieval of Relevant Spatial Web Objects".

[16] Y. Theodoridis, T. Sellis, "Optimization Issues in R-tree Construction", Technical Report KDBSLAB-TR-93-08.

[17] S. Alsubaiee, A.Behm, C.Li, " Supporting Loction-Based Approximate-Keyword Queries", ACM GIS' 10, November 2-5, 2010. San Jose, CA, USA.

[18] B.Yao, F.Li, M. Hadjieleftheriou, K. Hou, " Approximate String Search in Spatial Database".

[19] http://en.wikipedia.org/wiki/Levenshtein_distance

[20] http://en.wikibooks.org/ wiki/Algorithm_Implementation/Strings/ Levenshtein_distance

★★★