Volume 5, Issue 12, December 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

B-Tree Data Structure - Storing Spatial Data and Efficient Search of Geographical Locations

Ajay Sood
Research Scholar
Punjab Technical University
Punjab, India

Er. Charanjeet Singh
Asso. Professor
RIMT Institute of Engineering and Technology
Address of College

Abstract: Spatial database systems offer the underlying database technology for geographic information systems and other applications. Finding the correction location of user based on latitude and longitude is big challenge in spatial database.There is an urgent need for effective and efficient methods to extract unknown and unexpected information from spatial data sets of unprecedentedly large size, high dimensionality, and complexity. To address these challenges, spatial data mining and geographic knowledge discovery has emerged as an active research field, focusing on the development of theory, methodology, and practice for the extraction of useful information and knowledge from massive and complex spatial databases. In this paper we are developing an algorithm which take the input of latitude and longitude based on that it get the location much faster with accurate way.

Keywords—spatial data mining,data structure

I. DATA STRUCTURE

In computer science, a data structure is a particular way of organizing data in a computer so that it can be used efficiently. To use efficiently, data structure is a method to organize the data.Data structure can implement one or more particular abstract data types (ADT), which are the means of specifying the contract of operations and their complexity. In comparison, a data structure is a concrete implementation of the contract provided by an ADT.

Different kinds of data structures are suited to different kinds of applications, and some are highly specialized to specific tasks. For example, databases use B-tree indexes for small percentages of data retrieval and compilers and databases use dynamic hash tables as look up tables.

Data structures provide a means to manage large amounts of data efficiently for uses such as large databases and internet indexing services. Usually, efficient data structures are key to designing efficient algorithms. Some formal design methods and programming languages emphasize data structures, rather than algorithms, as the key organizing factor in software design. Storing and retrieving can be carried out on data stored in both main memory and in secondary memory.

Data structures are generally based on the ability of a computer to fetch and store data at any place in its memory, specified by a pointer – a bit string, representing a memory address that can be itself stored in memory and manipulated by the program. Thus, the array and record data structures are based on computing the addresses of data items with arithmetic operations; while the linked data structures are based on storing addresses of data items within the structure itself. Many data structures use both principles, sometimes combined in non-trivial ways (as in XOR linking).

The implementation of a data structure usually requires writing a set of procedures that create and manipulate instances of that structure. The efficiency of a data structure cannot be analyzed separately from those operations. This observation motivates the theoretical concept of an abstract data type, a data structure that is defined indirectly by the operations that may be performed on it, and the mathematical properties of those operations (including their space and time cost).

A. Spatial data mining

Mining spatial co-location patterns [is an important spatial data mining task. A spatial co-location pattern is a set of spatial features that are frequently located together in spatial proximity. To illustrate the idea of spatial co-location patterns, let us consider a sample spatial data set, as shown in Fig. 1. In the figure, there are various spatial instances with different spatial features that are denoted by different symbols. As can be seen, spatial feature + and × tend to be located together because their instances are frequently located in spatial proximity. The problem of mining spatial co-location patterns can be related to various application domains. For example, in location based services, different services are requested by service subscribers from their mobile PDA's equipped with locating devices such as GPS. Some types of services may be requested in proximate geographic area, such as finding the nearest Italian restaurant and the nearest parking place. Location based service providers are very interested in finding what services are requested frequently together and located in spatial proximity. This information can help them improve the effectiveness of their location based recommendation systems where a user requested a service in a location will be recommended a service in a nearby location. Knowing co-location patterns in location based services may also enable the use of pre-fetching to speed up service delivery. In ecology, scientists are interested in finding frequent co-occurrences among spatial features, such as

drought, EI Nino, substantial increase/drop in vegetation, and extremely high precipitation. The previous studies on co-location pattern mining emphasize frequent cooccurrences of all the features involved. This marks off some valuable patterns involving rare spatial features. We say a spatial feature is rare if its instances are substantially less than those of the other features in a co-location. This definition of "rareness" is relative with respect to other features in a co-location. A feature could be rare in one co-location but not rare in another. For example, if the spatial feature A has 10 instances, the spatial feature B has 20 instances, and the spatial feature C has 10,000 instances. A is not considered a rare feature in the co-location {A, B} but it is considered a rare feature in co-location {A, C}. Of course, a feature with very small number of instances are often rare in many co-location patterns.

B. Common spatial data-mining tasks

Spatial data mining is a growing research field that is still at a very early stage. During the last decade, due to the widespread applications of GPS technology, web-based spatial data sharing and mapping, high-resolution remote sensing, and location-based services, more and more research domains have created or gained access to high-quality geographic data to incorporate spatial information and analysis in various studies, such as social analysis (Spielman & Thill, 2008) and business applications (Brimicombe, 2007). Besides the research domain, private industries and the general public also have enormous interest in both contributing geographic data and using the vast data resources for various application needs. Therefore, it is well anticipated that more and more new uses of spatial data and novel spatial data mining approaches will be developed in the coming years. Although we attempt to present an overview of common spatial data mining methods in this section, readers should be aware that spatial data mining is a new and exciting field that its bounds and potentials are yet to be defined.Spatial data mining encompasses various tasks and, for each task, a number of different methods are often available, whether computational, statistical, visual, or some combination of them. Here we only briefly introduce a selected set of tasks and related methods, including classification (supervised classification), association rule mining, clustering (unsupervised classification), and multivariate geovisualization.

C. Spatial association rule mining

Association rule mining was originally intended to discover regularities between items in large transaction databases (Agrawal, Imielinski, & Swami, 1993). Let I = {i1, i2, ..., im} be a set of items (i.e., items purchased in transactions such as computer, milk, bike, etc.). Let D be a set of transactions, where each transaction T is a set of items such that T # I. Let X be a set of items and a transaction T is said to contain X if and only if X # T. An association rule is in the form: X) Y, where X I; Y I and X \ Y ¼ £. The rule X) Y holds in the transaction set D with confidence c if c% of all transactions in D that contain X also contain Y. The rule X) Y has support s in the transaction set D if s% of transactions in D contain X [Y. Confidence denotes the strength and support indicates the fre- quencies of the rule. It is often desirable to pay attention to those rules that have reasonably large support (Agrawal et al., 1993). Similar to the mining of association rules in transactional or relational databases, spatial association rules can be mined in spatial databases by considering spatial properties and predicates (Appice, Ceci, Lanza, Lisi, & Malerba, 2003; Han & Kamber, 2001; Koperski & Han, 1995; Mennis & Liu, 2005). A spatial association rule is expressed in the form A) B [s%, c%], where A and B are sets of spatial or non-spatial predicates, s% is the support of the rule, and c% is the confidence of the rule. Obviously, many possible spatial predicates (e.g., close_to, far_- away, intersect, overlap, etc.) can be used in spatial association rules. It is computationally expensive to consider various spatial predicates in deriving association rules from a large spatial datasets. Another potential problem with spatial association rule mining is that a large number of rules may be generated and many of them are obvious or common knowledge. Domain knowledge is needed to filter out trivial rules and focus only on new and interesting findings. Spatial co-location pattern mining is spiritually similar to, but technically very different from, association rule mining (Shekhar & Huang, 2001). Given a dataset of spatial features and their locations, a co-location pattern represents subsets of features frequently located together, such as a certain species of bird tend to habitat with a certain type of trees. Of course a location is not a transaction and two features rarely exist at exactly the same location. Therefore, a user-specified neighborhood is needed as a container to check which features co-locate in the same neighborhood. Measures and algorithms for mining spatial co-location patterns have been proposed (Huang, Pei, & Xiong, 2006; Lu & Thill, 2008; Shekhar & Huang, 2001).

D. B Tree Algorithm

In computer science, a B-tree is a tree data structure that keeps data sorted and allows searches, sequential access, insertions, and deletions in logarithmic time. The B-tree is a generalization of a binary search tree in that a node can have more than two children. Unlike self-balancing binary search trees, the B-tree is optimized for systems that read and write large blocks of data. B-trees are a good example of a data structure for external memory. It is commonly used in databases and file systems.

In B-trees, internal (non-leaf) nodes can have a variable number of child nodes within some pre-defined range. When data is inserted or removed from a node, its number of child nodes changes. In order to maintain the pre-defined range, internal nodes may be joined or split. Because a range of child nodes is permitted, B-trees do not need re-balancing as frequently as other self-balancing search trees, but may waste some space, since nodes are not entirely full. The lower and upper bounds on the number of child nodes are typically fixed for a particular implementation. For example, in a 2-3 B-tree (often simply referred to as a 2-3 tree), each internal node may have only 2 or 3 child nodes.

Each internal node of a B-tree will contain a number of keys. The keys act as separation values which divide its sub trees. For example, if an internal node has 3 child nodes (or sub trees) then it must have 2 keys: a_1 and a_2. All values in the

leftmost sub tree will be less than a_1, all values in the middle sub tree will be between a_1 and a_2, and all values in the rightmost sub tree will be greater than a_2.

Usually, the number of keys is chosen to vary between d and 2d, where d is the minimum number of keys, and d+1 is the minimum degree or branching factor of the tree. In practice, the keys take up the most space in a node. The factor of 2 will guarantee that nodes can be split or combined. If an internal node has 2d keys, then adding a key to that node can be accomplished by splitting the 2d key node into two d key nodes and adding the key to the parent node. Each split node has the required minimum number of keys. Similarly, if an internal node and its neighbor each have d keys, then a key may be deleted from the internal node by combining with its neighbor. Deleting the key would make the internal node have d-1 keys; joining the neighbor would add d keys plus one more key brought down from the neighbor's parent. The result is an entirely full node of 2d keys.

The number of branches (or child nodes) from a node will be one more than the number of keys stored in the node. In a 2-3 B-tree, the internal nodes will store either one key (with two child nodes) or two keys (with three child nodes). A B-tree is sometimes described with the parameters (d+1) _ (2d+1) or simply with the highest branching order, (2d+1).

A B-tree is kept balanced by requiring that all leaf nodes be at the same depth. This depth will increase slowly as elements are added to the tree, but an increase in the overall depth is infrequent, and results in all leaf nodes being one more node farther away from the root.

B-trees have substantial advantages over alternative implementations when the time to access the data of a node greatly exceeds the time spent processing that data, because then the cost of accessing the node may be amortized over multiple operations within the node. This usually occurs when the node data are in secondary storage such as disk drives. By maximizing the number of keys within each internalnode, the height of the tree decreases and the number of expensive node accesses is reduced. In addition, rebalancing of the tree occurs less often. The maximum number of child nodes depends on the information that must be stored for each child node and the size of a full disk block or an analogous size in secondary storage. While 2-3 B-trees are easier to explain, practical B-trees using secondary storage need a large number of child nodes to improve performance.

B-Tree is a self-balancing search tree. In most of the other self-balancing search trees (like AVL and Red Black Trees), it is assumed that everything is in main memory. To understand use of B-Trees, we must think of huge amount of data that cannot fit in main memory. When the number of keys is high, the data is read from disk in the form of blocks. Disk access time is very high compared to main memory access time. The main idea of using B-Trees is to reduce the number of disk accesses. Most of the tree operations (search, insert, delete, max, min, ..etc) require O(h) disk accesses where h is height of the tree. B-tree is a fat tree. Height of B-Trees is kept low by putting maximum possible keys in a B-Tree node. Generally, a B-Tree node size is kept equal to the disk block size. Since h is low for B-Tree, total disk accesses for most of the operations are reduced significantly compared to balanced Binary Search Trees like AVL Tree, Red Black Tree, ..etc.

II. PROBLEM FORMATION

Spatial database is big database which is combination of latitude and longitude based on this index we can get the particular location information. But for searching in the spatial database is a very tedious process and time consuming process. For solving this problem we will develop an efficient B-tree algorithm base on that we will search any location inside spatial database much faster than existing algorithm that is skip list. Problem statement in our research we will use Getty database to search any location, add any new entry delete any new entry in B tree. B tree occupy less memory as compare to skip linked list because one node contain parent value with four child nodes in B tree height get increase not exceed memory it's an advantage than skip list insertion is easy, searching is much faster simple by finding the parent value if search value is less than parent value then we can find the value in their child of parent value

III. CONCLUSION

Proposed solution provide an fast and accurate data for the user in fraction of milliseconds . In our research we also covered up the memory space utilization because B tree memory size does not increase instead its height increase. Our research can further extend to show correct position of user if user change the angle of position.

REFERENCES

[1] Jeong-hoon Park, Chin-Wan Chung and U Kang. June 2015. Reverse nearest neighbor search with a non-spatial aspect Information Systems Reverse nearest neighbor Spatial Non-spatial RNN LBS journal Elsevier. "Multipath Routing for Wireless Mesh Networks",

[2] *IMartin Kuhmaier, Christian Kanzian and Karl Stampfer March 2014. Identification of potential energy wood terminal locations using a spatial multicriteria decision analysis Bio mass and Bio energy journal Elsevier XXX.*

[3] Madhvi Sharma and Shaila Chugh August 2014. Advanced Fast Nearest Neighbor Search with Keywords over spatial database. IEEE International Conference on Advances in Engineering & Technology Research 978-1-4799-6393-5.

[4] Sneha Arjun Dhargalkar and A.D. Bapat 2014. Determining Missing Values in Dimension Incomplete Databases using Spatial-Temporal Correlation Techniques. In IEEE International Conference on Advanced Communication Control and Computing Technologies(ICACCCT), 978-1-4799-3914-5.

[5] *YAmol Barewar, Mansi A. Radke and U. A. Deshpande 2014. Geo Skip List Data Structure -storing spatial data*

and efficient search of geographical locations International Conference on Advances in Computing, Communications and Informatics (ICACCI) 978-1-4799-3080-7.

[6] Deepak Paramashivan Kaundinya , P. Balachandra, N.H. Ravindranath and Veilumuthu Ashok. March 2013. A GIS (geographical information system)-based spatial data miningapproach for optimal location and capacity planning of distributed biomass power generation facilities: A case study of Tumkur district,India Bioenergy GIS (geographical information system)Spatial clustering Rural energy planning journal Elsevier.

[7] Yue Cui 2013.A systematic approach to evaluate and validate the spatial accuracy of farmers market locations using multi-geocoding services Farmers market directory User-generated content Geocoding service Web-based GIS journal Elsevier Applied Geography 41 (2013) 87-95.

[8] Amgad Madkour, Walid G. Aref and Saleh Basalamah 2013. Knowledge Cubes - A Proposal for Scalable and Semantically-Guided Management of Big Data. This research was partially supported by the National Science Foundation under Grants III-1117766, and IIS-0964639,and IIS-0916614.

[9] Wendy Osborn,and Annika Hinze 2013. TIP-tree: A spatial index for traversing locations in context-aware mobile access to digital libraries Pervasive and Mobile Computing journal Elsevier 1574-1192.

[10] Li, Zhisheng; Lee, Ken C.K.; ZHENG, Baihua; Lee, Wang-chien; Lee, DikLun; and Wang, Xufa. IR-tree: An Efficient Index for Geographic Document Search. (2011). IEEE Transactions on Knowledge and Data Engineering (TKDE). , 23(4), 585-599.

[11] *I.D.Felipe, V.Hristidis, and N.Rishe. Keyword search on spatial databases Published in: Proceeding ICDE '08 Proceedings of the 2008 IEEE conference, 2008, pages 656–665.*

[12] Ramaswamy Hariharan, Bijit Hore, Chen Li, Sharad Mehrotra, Processing Spatial-Keyword (SK) Queries in Geographic Information Retrieval (GIR) System. SSDBMv 2007, Banff, Canada.

[13] Enviromental Systems Research institute, inc., 1999, Modeling our world the ESRI Guide to Geodatabase design. ESRI Press.

[14] *SEnvironmental Systems Research Institute, Inc., 1999, Managing ArcSDE™ Services, ESRI Press*

[15] Abraham, T. and Roddick, J.F. 1997. 'Discovering meta-rules in mining temporal and spatio-temporal data'. In Proc. Eighth International Database Workshop, Data Mining, Data Warehousing and Client/Server Databases (IDW'97),

