Analysis of Spatial Database Index Technology

Wang Wei
Software College,Chuangchun University
wangweitea@163.com

***Abstract*-as the access engine of a spatial database, spatial database indexing technology is an important mechanism related to how to improve the spatial database information processing and information management performance, is also at an important stage of research and discussion. This paper starts with the mainstream spatial database index technology which is generally accepted by the current industry, to describe the basic principles and main methods of the various spatial database indexes, and analyze the applicability, advantages and disadvantages of them.**

Keywords-database; spatial index; information management

I. INTRODUCTION

Binary search tree is a basic index data structure, which has outstanding performance in the linear sequence data item index. In the random case, the average search length is 1+41ogn; the average search length of the balanced binary search tree is logn, with a very good search performance. Therefore, the binary search tree in practice is extended to spatial data indexing, through nested decomposition for the spatial region to establish efficient binary index tree, reduce the queried spatial extent and improve the spatial search performance. The following will respectively describe the typical spatial index structures based on ambiguous tree.

II. KD-TREE

A. Kd-tree definition

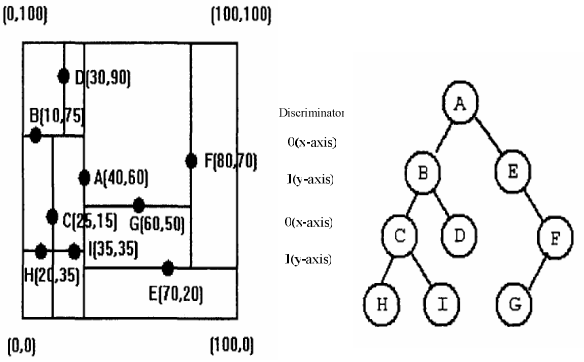
The kd-tree Bentley proposed in 1975 is k (k ≥ 2)-dimensional binary search tree (BST), which is mainly used in multi-attribute data or multi-dimensional point data index. Each node of the kd-tree denotes a point in k-dimensional space, and each layer of the tree makes a branch decision according to this layer discriminator. The discriminator of the kd-tree first i-layer is defined as: i mod k (the tree root node in 0 layers …in ascending order).

In short, kd-tree is an empty tree, or a binary tree with the following properties:

1) If its left sub-tree is not empty, then the first d-dimensional values of all the nodes on it are less than the root node d-dimensional value (d as the root node discriminator);

2) If its right sub-tree is not empty, then the first d-dimensional values of all the nodes on it are more than or equal to the root node d-dimensional value (d as the root node discriminator);

3) Its left and right sub-tree is separately kd-tree.



(a) Plan (b) kd-tree structure chart

Figure 1. F kd-tree sketch map

Figure 1 is a kd-tree example in one-dimensional space. As shown in it, assuming that the discriminator value of any one node P is i (corresponding to d-dimension), then the d-dimensional values of all the data points (all of the descendant node)in the left sub-tree are less than of P, the other hand, the values of the right sub-tree are more than of P.

B. Kd-tree search

For the given point exact match and search, kd-tree search process is similar to BST, the difference is that, when selecting branch down the query it should consider the node discriminator value. For example, supposing to search point O (25, 15) from the kd-tree in figure 1, first we should compare the 0 coordinates with the root node A (40, 60), if matched the search is successful. If the matching is not successful, then to compare with the x-dimension (x dimension is discriminator of A) of O and A, because 25<40, so to enter the left A sub-tree to search; comparison of 0 and B (10, 75) coordinates, if not matched, to further compare the O and B-point Y dimension value (Y dimension is discriminator of B), because 15<75, so to enter the left B sub-tree to search; comparison of O and C coordinates, matched, the search is successful.

Regional search and point matching search process is similar, except that the paths are often many.

C. Kd-tree insertion

To insert a new node to the kd-tree is similar to BST insertion process. Insert principle is: if the kd-tree is empty, then the insertion node to be root node; otherwise, to compare with the discriminator values of the current node, select its left or right sub-tree to continue the search until a leaf node left or right sub-tree is empty. At this time, the

inserted node as a new leaf node, becomes the left or right child of the leaf node.

D. Kd-tree deletion

Kd-tree delete node algorithm is similar to BST, but more complex. Because the kd-tree was introduced to discriminator, so to delete the node must consider the discriminator restriction to avoid violation of the principle of kd-tree.

Deletion process is as follows: first find the node to be removed (set to Q, and let Q discriminator value is d), and then process with three situations:

1) If Q is a leaf node, then to empty the pointer field corresponding to Q in its parent nodes (if Q's parent node exists) and delete the node;

2) If the node Q has the right son, to find the node (set to Q1) with the smallest d-dimensional value from its right sub-tree to assign node Q, and then call the delete function to delete j node Q1;

3) If the node Q has only left son, then to find the node (set to Q2) with the largest d-dimensional value from its left sub-tree to assign node Q, and then call the delete function to delete j node Q2.

Taking into account in the Q left sub-tree, for the first d-dimensional, there may be multiple nodes have the same value with Q2. In the third case, using the node Q2 with the largest d-dimensional value in the left sub-tree to replace Q may lead to errors, which disobeys the kd-tree sorting rules (the first d-dimensional values of all the nodes in the left sub-tree are less than the root node).

For example, in figure 1, to find the node with the largest Y-dimensional value from the left sub-tree when deleting node B (either F or G) to replace B all will be contrary to kd-tree rules. Therefore, for this situation requires another solution. One solution is to exchange the left and right pointers of Q node, so that make the Q left sub-tree become the right sub-tree (that is, to translate the third case into the second case). And then for normal removal process, the new index tree would not violate the kd-tree rules.

E. Kd-tree analysis

Kd-tree is based on the binary search tree, to be extended to multi-dimensional space. The point search not only inherits the advantages of binary search tree, that is, good query performance (average search length is 1+41ogn), but also inherits its shortcomings, that is, the operation when deleting the kd-tree middle node is complex and large time expense. In addition, how to maintain a balance of kd-tree is also a very complex issue; for the region query, the spatial data to be queried can be reflected as a space approximate objective, and then continue to transform into the M-dimensional space index points to index. But this will undermine the adjacency relationship between the spatial data, resulting in inefficient search.

III. K-D-B TREE

Because of the massive nature of spatial data, spatial database index file is generally stored in external memory. External memory space allocation, release and other scheduling all use page as a unit, so to store the kd-tree in external memory, a certain paging strategy must be adopted (such as bottom-up approach), which undoubtedly adds to the system overhead. For this, Robinson proposed K-D-B tree.

K-D-B tree is a combination of kd-tree and B-tree. It is composed by two basic structures- region page and point page and point, the latter is to store point spatial objects, and the former is to store index sub-space description and a page pointer pointing to the next layer. In the non-isomorphic kd-tree, each node is implicitly associated with an index space: root node is associated with the entire space, the middle node is associated with the sub-space obtained by the division of the parent node, and each page node is associated with a sub-space without division. But in K-D-B tree, the regional pages explicitly store the sub-spatial information.

A. K-D-B tree basic operations

For the point-matching search, it should begin from the tree root node to traverse the tree branch until a certain point page, that is, to access to all index sub-spaces, including the search page up to a certain point, finally to extract the page points to one by one judge; for the range query, it needs to access all the region and point pages of the index space intersected with the search region, so the search path is not unique.

When you insert the data point, we must first find the point page it should be inserted, if the page is unfilled, simply insert the point to the page; if the page has been full, you must split the page. The splitting method is to divide it into two parts and make them contain almost as same many points. What is to be noted is that after the point page split, you must increase a page in the higher level of regional page. Therefore, the point-page split may lead to the parent region page divisions. And the point-page split also may continuously transmit up to the root node. Therefore, K-D-B tree is similar to B tree, always maintain a high balance.

When a region page is splitting, all the items of it (including additional items) will be divided into two groups with the same number. A hyper-plane is used to divide the index space of it into two sub-index spaces. It is worth noting that this hyper-plane may cut some sub-spaces of the region page, leading to splitting of the sub-space intersected with it, so that the emerging sub-space will be completely included in the final regional page. Therefore, split operations may also be down transmitted.

If the limit of dividing the region page into two-page with almost the same number of items, then split down transmission can be avoided. Upward transmission does not cause page underflow, but the down transmission may cause page underflow (namely, the page utilization of storage space below the set threshold). In order to avoid low space utilization, the tree structure can be partially reorganized.

B. K-D-B tree analysis

K-D-B tree is highly balanced, all the leaf nodes are located in the same layer, but this has a cost of low storage efficiency. In addition, the construction of the tree also will

be complex because the upward or downward spread caused by the page splitting operations. Finally, the same as the kd-tree, K-D-B tree was also proposed for indexing multi-attribute data or multi-dimensional space points, to be used for other spatial objects index also needs target approximation and mapping, efficiency is poor.

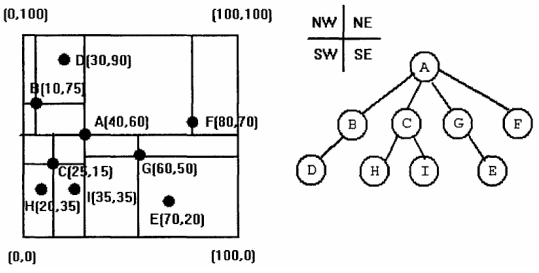
Binary tree-based index structure is mainly suitable for spatial point storage and index, if used for indexing non-point-like spatial objects, only to use goal approximation and mapping or target repeated storage two ways. If using goal approximation and mapping technology, the spatial relations (such as proximity) between the objectives is no longer maintained, for the regional search, efficiency is poor; if using target repeated storage strategy, the storage overhead will be increased. In addition, to store a binary tree in external memory, page schedule is also a complex issue.

IV. QUAD-TREE-BASED SPATIAL DATABASE INDEX TECHNOLOGY

Quad-tree is a common spatial index, which was established based on the region cycle decomposition, is a clear hierarchical index technology, and thus has ability to gather the spatial objects. It is to carry out nested quad decomposition for the overall spatial objects, to divide it into four equal sub-spaces, and then continue to decompose each or several sub-spaces, until the required granularity, thereby forming a spatial decomposition based on quad-tree.

A. Point quad-tree

R.A.Finkel and J.L.Bentley proposed a point quad-tree for spatial point storage and index in 1974. Corresponding to k-dimensional data space, each node of point quad-tree is implicitly corresponding to an index space, which stores not only the current node's point spatial information, but also the pointer pointed to 2k son nodes (the point for the decomposition point to decompose the corresponding index space into 2k sub-spaces disjoint each other) to ordinal corresponding to the sub-space and establish index sub-tree on it. Figure 2 is a two-dimensional point quad-tree example.



(a) Plan (b) Point quad-tree structure chart

Figure 2. Point quad-tree sketch map

Point quad-tree construction process is very simple, that is, to begin from the root node, to reach a node, divide the corresponding index space into four sub-spaces, determine which sub-space the node fell into, if the corresponding sub-nodes exist, then continue down the query; otherwise, directly insert into the current node; but to delete a space point, all the descendant nodes corresponding to the point must be re-inserted, resulting in inefficiencies. For the exactly matched point search, the path is unique and efficient; for the region query, then the performance is poor.

Point quad-tree has simple structure, but the established index structure dynamic is poor, and the balance of the index tree is difficult to control; at the same time, the space utilization is not high, each node must store 2k pointers, although containing a large number of null pointers. When the k value is larger, the overhead of the spatial storage is exponentially increased, which significantly reduces the space utilization.

B. Region quad-tree

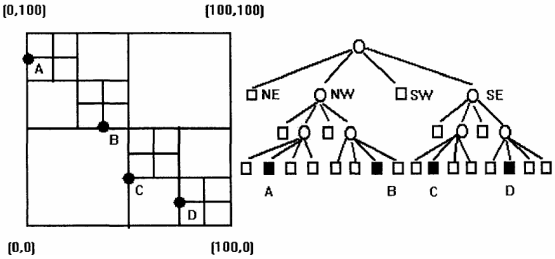
For the point quad-tree deficiencies of poor dynamic nature, determining structure by insertion sequence, Samet and Orenstein separately used MX and PR quad-tree to index the multidimensional space points.

Each node of the regional quad-tree implicitly corresponds to an index space, the root node corresponds to the whole space, and then to carry out nested decomposition, each node sub-node corresponds to the decomposed sub-space. It is worth mentioning is that the middle node of the quad-tree does not store space point information, but merely stores a pointer to all decomposed sub-spaces.

C. MX quad-tree

The index space corresponding to the MX quad-tree root node is the whole index region. For a k-dimensional space, MX index tree begins to repeat 2k decomposition for the entire index region from the root node, and each decomposed sub-space is called as quadrant. MX index tree is repeatedly decomposed until all the spatial points are located in one of the smallest particle size of the lower left quadrant. As a result, every point in space belongs to a particular quadrant, that is, a particular is associated with a spatial point.

Figure 3 is a MX-tree example.



(a) Plan (b) MX tree structure chart

Figure 3. MX quad-tree sketch map

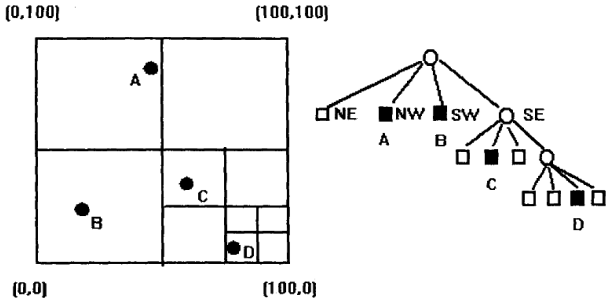
As shown in the figure, all the spatial nodes of the MX quad-tree are located in the leaf nodes, has a strong balance. The use of a linear storage structure, then relative to the binary search tree, it can avoid null pointer domain storage, improve space utilization. But it will affect the depth of the tree when inserting and deleting the spatial data points, so part of the leaf nodes need to re-positioning; at the same

time, because of its nature, the MX quad-tree depth is larger, which reduces the efficiency of the query.

D. PR quad-tree

PR quad-tree principle is basically the same as the MX quad-tree. The difference is that the spatial data points of the PR quad-tree are contained in a specific quadrant, rather than located in the lower-left corner of a quadrant.

Figure 4 is a PR quad-tree example, from it we can see that its leaf nodes are not necessarily at the same level, PR quad-tree is to begin from the root node to nested decompose the whole index space, until a certain quad sub-space contains the spatial points. This makes its leaf node and tree depth obviously smaller than MX quad-tree, while improving the efficiency of the index.



(a)Plan (b) PR tree structure chart

Figure 4. PR quad-tree sketch map

V. TECHNICAL ANALYSIS AND CONCLUSION OF SPATIAL DATABASE INDEX BASED ON QUAD-TREE

Quad-tree was established based on the regional cycle decomposition, is a clear hierarchical index technology, and thus has the ability to aggregate spatial objects. However, the spatial data items of large region in the quad-tree has repeated storage problem. The decomposition granularity of the quad-tree to index space can decide its depth (namely, level). When indexing a large number of spatial data, if the quad-tree depth is too small, the search performance will be reduced; if too large, the repeated storage of spatial data will be deteriorated, thus reducing the efficiency of the index.

This paper described several popular spatial data index technologies. And for each spatial data index technology this paper separately introduced its organizational structure and related algorithms, after its performance analysis, gave its advantages and disadvantages.

REFERENCES

[1]. Don Murray and Dale Lutz, ESRI' s Spatial Database Engine, ESRI's Research Report, 1995.

[2]. J.T.Robinson, "The K-D-B-tree: A search structure for large multidimensional dynamic indexes", Proc.ACM SIGMOD Int. Conf. On Management of Data, 1981:10-18.

[3]. Volker Gaede, Oliver Gunther, “Multidimensional Access Methods”, ACM Comput Surv, 1998, 30 (2): 170-231.

[4]. J .Nievergelt, H.Hinterberger, K.C.Sevcik, “The gridfile: An adaptable symmetric multikey file structure”, ACM Transactions on Database Systems, 1984, 9(1):3 8-71.

[5]. MICHAEL FREESTON, “The BANG File: A New Kind of Grid File”, SIGMOD Conference, 1987: 260-269.