ASPECTS CONCERNING THE IMPLEMENTAZION OF A MULTIDIMENSIONAL INDEX STRUCTURE – THE BANG FILE

Marius Zaharia

POLITEHNICA University of Bucharest

Computers Department

Splaiul Independentei, 313, Bucuresti – 77206, Romania

E-mail: zaharia@cs.pub.ro

***Abstract*: This paper describes and evaluates a method to implement the BANG file (Bakanced and Nested Grid File) a structure used to store and index disk-resident multidimensional point data. This structure has similarities with hierarchical data structures and grid based indexing techniques (array based directories)**

***Index terms*: spatial index, k-d PR tree, multidimensional point data, database**

1. Introduction

Many applications, especially those in the field of data bases used for graphics processing (Geographic Information Systems, CAD systems or computer vision applications) require efficient coding and retrieval of information in multidimensional point data sets. A correlation can usually be made between points in a multidimensional data space and tuples (multiattribute records) of a database relationship. That is why the multidimensional point data indexing techniques based on geometric (spatial) considerations could also be used in non-spatial database applications.

There are a lot of different modalities ro represent multidimensional point data. The programmer may choose the suitable one according to the most heavily used operations which are expected to be performed on the data. The issues distinguishing those representations are:

-the data space organized by the representation (object space, image space or both – in case of hybrid methods, some attributes are represented in object space others in image space)

-the context in which data will be used (static or dynamic)

-the quantity of data (if there is a sufficiently small amount of data these could be stored only in core, otherwise auxiliary memory would be used)

The data points are grouped in pages (buckets). Each page includes the points from a hiper-rectangular spatial region. The method presented and evaluated below, builds a directory file containing binary codes. Each code is related with a spatial region corresponding to a data page. To retrieve a data point (P), a discrimination binary code © must be formed. Subsequently th directory is searched for a region (R) whose code is similar with (c). Finally (P) will be searched in the data bucket associated with (R).

2. The method of spatial decomposition

In the following text, a two-dimensional rectangular data space will be considered. Let q be the capacity of a data page. If the insertion of a data point produce the overflow of a data page (pag) capacity, the bucket will be splitted in two pages (i.e. a new data bucket (pag1) will be created). The split is made by dividing the spatial region corresponding to (pag), in two equal sizes and distributing the q+1 points from (pag) between (pag) and (pag1). The spatial division is accomplished across a (hiper)line parallel with one of the cartesian axes of the data space. Sometimes, even after a split process, one data page still overflows, in that case the spatial division will be made across other axis. The splitting axes are cyclically selected.

The result of the above mentioned algorithm which accomplish the data space splitting process can be represented by a binary tree. The leaf nodes correspond to data (point) pages and the internal nodes contain split coordinates values. The depth of an internal node represents the space coordinates across which the division is produced. As an example, the root (level 0 node) corresponds to a split line orthogonal to Ox (i.e. axis number 0).

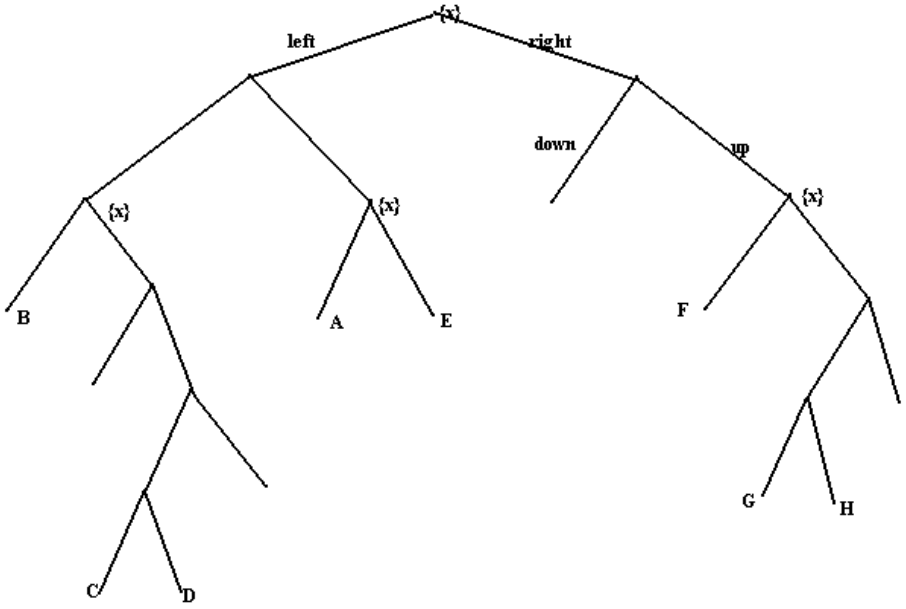

Figure 1

In a k-dimensional data space, an internal node placed on level i will correspond to a division across the axis numbered (i mod k). This binary tree is called k-d PR tree (k-dimensional Point Region). Figure 1 shows the k-d PR tree describing the set of data points from Figure 2 (data page capacity is 1 point).

The k-d PR tree can be compared with a binary search tree in which rhe keys from internal nodes placed on different levels are associated with different spatial split coordinates. In case of nonuniformly distributed data points it is possible to obtain heavily unbalanced trees. (The shape of the tree is unfluenced by the insertion order of the data points). These trees could include internal nodes having only one (nonempty) son. This type of nodes can be eliminated in two steps, without losing the spatial split information.

1)Associate to every internal node (N) of the k-d PR tree a binary code representing the results of the tests needed to traverse the path from root to the left son of (N).

2)Remove from the resulting tree all nodes having only one son

Doing the path compression of the k-d PR tree from Figure 1 results the tree from Figure 2.

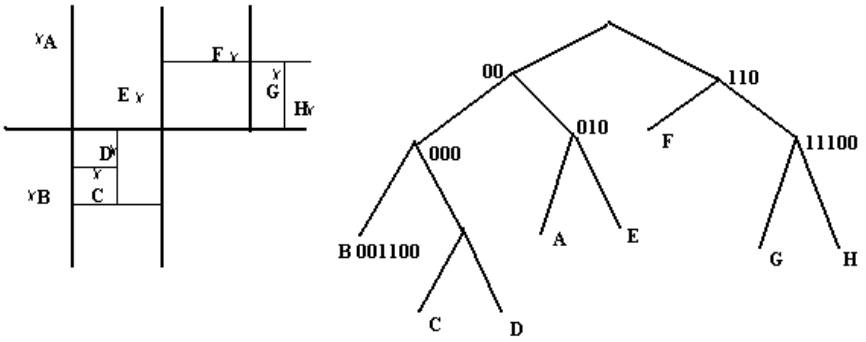

Figure 2

The way to remove the "one son internal nodes" can be deduced from Figure 1 and Figure 2. The compression algorithm is solving situations as those from Figure 3.

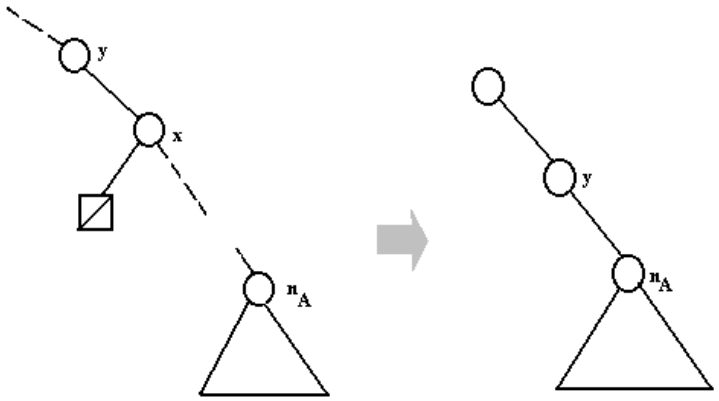

Figure 3

It is assumed that all the nodes placed on the path from x to n_A have only one nonempty son and x is the first internal node (on the path from root to n_A) having only one son. The path compression process of the k-d PR tree will produce a binary tree in which all internal nodes have two sons (*). This compression process will improve the efficiency of searching information in the tree (the tree height is normally reduced) if that tree would be used as an index structure.

Proposition: In a binary tree with property (*) the difference between the number of leaf nodes and the number of internal nodes is 1.

The proposition will be demonstrated applying induction over the number of internal nodes. Obviously a tree with n=0 internal nodes will have one leaf node and for n=1 the tree will be made up from one internal node and two nonempty leaves.

Let's suppose that, given an arbitrary value for n, any tree A_n (with n internal nodes and the property (*)) has n+1 leaves. A tree A_{n+1} (with n+1 internal nodes) can be built from a tree A_n by adding a new internal node (r) on the path between two nodes p, q (which are father and son) from A_n. Following this operation, (r) will be p's son and q will become r's son. The property (*) for A_{n+1} will remain true if and only if r will get a new son (leaf node). Though the leaves number of A_{n+1} tree will be n+2 and the proof is complete.

The proposition above could be a validation criterion for the correctitude of the k-d PR tree compression algorithm.

The organization of the data space in case of splitting after a k-d PR policy can also be represented as a kind of grid file. The binary discrimination codes are specifying the geometrical information necessary for the data space partitioning. This type of information can also be stored on auxiliary memory, in a file called BANG directory. Every record of that directory includes information about one data page from the BANG data file [1]. The binary discrimination code corresponding to the internal nodes of the compressed k-d PR tree are obtained by bit interleaving of the binary signatures associated to every side of the node's rectangular region.

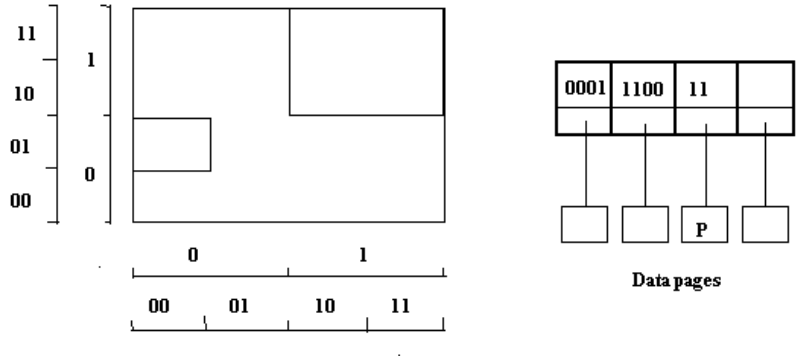

Figure 4

An entry in the BANG directory is a triplet (c, lg, pag) where:

-c represents the discrimination binary code associated with the leaf of the compressed k-d PR tree which refers the data page

-lg length (no. of bits) of c

-pag offset of the data page in the data file.

The search of the data point (P) is accomplished by the sequential traversal of the BANG directory until a discrimination binary code, which corresponds to a spatial region including (P) is found. In the directory file, the entries are sorted in the decreasing order of lg field (see Figure s).

3. Results evaluation

The k-d PR tree building method, its compression and the technique to get the resulting BANG directory were implemented in C (the evaluation was made on an INTEL 486SX/33 processor). The information was stored on a 128K RAMDISK. From the tables below can be observed that in case of sufficiently large page capacities (DIM_PAG=20) the resulting k-d PR tree is identical to the compressed tree (i.e. k-d PR tree satisfies the condition: no. of leaves = no. of internal nodes + 1). This is valid even for relatively large data sets (10000 data points).

The biggest compression rate is acquired in case of short data nages (DIM_PAG=2). In that case, the reduction rate of the tree's height (a measure of the information search time – worst case) is of maximum 18.2% (the 10000 points data set) and minimum 11.2% (the 5000 point data set).

The percentage diminishing of the internal nodes number is 11.2% (DIM_PAG=2, 10000 data points), 10% (DIM_PAG=2, 5000 data points) and 10.5% (DIM_PAG=2, 1000 data points) respectively. The compression time is not significant comparing with the k-d PR time.

The comparison of two discrimination binary codes (useful for information retrieval in the BANG file) is given in the below specified algorithm. The first node (d1) will have the maximum possible binary length (depending on the geometric length size of the data space rectangle) and will be associated with the point to be searched (P). The second code will be read from the BANG directory. The function will return 1 in case of two compatible codes.

```
function cmp_code(long d1, int l1, long d2, int l2) returns {0,1} is
        if l1<l2 then
                lmin ← l1 & lmax← l2 & dmin← d1 & dmax ← d2
        else
                lmin ← l2 & lmax← l1 & dmin← d2 & dmax ← d1
        
        if lmax ≠ lmin then
                dmax ← dmax >> (lmax – lmin)
        
        return dmax = dmin
end
```

DIM_PAG=20

No. Pnts.	Leaves k-d PR	Int nodes k-d PR	H k-d PR	Insert time	H compr. tree	Compr. time	Search time cmp.tr	Search time BANG
100	8	7	4	60	4	0	5	6
1000	70	69	8	192	8	0	33	346
5000	351	350	10	807	10	0	330	8535
10000	721	720	12	1719	12	0	829	36295

DIM_PAG=10

No. Pnts.	Leaves k-d PR	Int nodes k-d PR	H k-d PR	Insert time	H compr. tree	Compr. time	Search time cmp.tr	Search time BANG
100	15	14	6	68	6	0	0	11
1000	142	141	10	182	10	0	33	626
5000	717	716	12	807	12	5	319	16044
10000	1458	1457	14	1720	14	5	824	74369

Figure 5

Figure 5 shows the time for building the k-d PR tree spatial decomposition, in dependence of the data set size. The other part of the figure shows the total search time in the BANG file (a retrieval query for every point in the data set) in dependence of the data set size. The tables show that te medium search time for one point varies aüüroximately linearly in function of the data set size. Indeed, the search process consists in the sequential (linear) sons of the BANG directory.

In case of internal search using the compressed k-d PR tree as indexing structure, the search times have little variation depending on the data page size, longer times are registered for longer pages. That is true because in case of sufficiently large data pages the sequential search of a point in a data bucket has bigger weight comparing with the compressed tree traversal time.

References

[1] T.A. Mueck, Optimizing Sort Order Query Execution in Balanced and Nested Grid Files, *IEEE Transactions on Knowledge and Data Engineering*, vol. 7, No. 2, pag. 246-259, (april 1995).

[2] M.H. Overmars, Geometric Data Structures for Computer Graphics. An Overview. *Theoretical Foundations of Computer Graphics and CAD*, NATO ASI series, vol. F40, pag. 51-68, (1988)

[3] H.J. Samet, An Overview of Quadtrees, Octrees and Related Data Structures, *Theoretical Foundations of Computer Graphics and CAD*, NATO ASI series, vol F40, pag. 21-49, (1988)