A Modified and Memory Saving Approach to B+ Tree Index for Search of an Image Database based on Chain Codes

Sabina Priyadarshini
Department of Information Technology
Birla Institute of Technology
Mesra, Ranchi

Gadadhar Sahoo
Department of Information Technology
Birla Institute of Technology
Mesra, Ranchi

ABSTRACT

Space savings is a demand of these days. Almost all applications are trying to represent their data in the least possible amount of space. It is the goal of all algorithms to consume as less memory as possible in the computer. Many techniques have been developed to compress large bulk of data so that large amount of information can be accommodated in lesser amount of space. The proposed approach called Coded B+ tree is a modified approach to the conventional B+ Tree index and can be applied only to chain codes of images and to a search of an image database based on chain codes of images as its search key value. The proposed method brings about considerable space savings of more than a half at the cost of 4 to 5 more comparisons. It is a useful technique of indexing for systems that require saving memory spaces.

General Terms

Image data base indexing, Digital Image Processing, Content Based Image Retrieval.

Keywords

chain codes, indexing technique, space savings, efficient search, coding of index.

1. INTRODUCTION

A B+ Tree index is a type of tree that represents sorted data in a way that allows efficient insertion, retrieval and removal of records, each of which is identified by a key. It is a balanced tree in which every path from the root of the tree to the leaf of the tree is of the same length. It provides good performance but has space overhead too[1].

In this paper, a modified approach to B+ Tree index has been proposed for indexing an image database based on chain codes as its search key field. This new approach reduces the space overhead of the B+ Tree index to a large extent at the cost of only three to five more comparisons.

The paper is organized as follows. Section 2 gives the Literature Review. Section 3 discusses the chain code of images. Section 4 describes the B+ tree index. The proposed approach has been given in Section 5. Section 6 gives a comparison of the conventional B+ tree index and the modified approach used in terms of space overhead and total number of comparisons. Section 7 gives advantage of the Coded B+ tree indices. Section 8 gives a conclusion.

2. LITERATURE REVIEW

Many researchers have come up with different techniques to search an image database. Different types of indices have been built and different search strategies have been proposed. The works done by the different researchers are discussed below.

Guttman[2] has suggested an R-Tree as a height balanced tree with index records in its leaf nodes containing pointers to data objects. This enables a spatial search to visit only a small number of nodes. The index is dynamic in nature. The structure performs very well and is useful for database systems in spatial applications.

Bayer et. al [3] discuss simple prefix B-trees and prefix B-Trees. Both the methods store only parts of keys called prefixes in the index part of a B*-tree. The prefixes are selected carefully to minimize their length. Prefixes are not fully stored but are reconstructed as the tree is searched. It combines the advantages of B-trees, digital search trees and key compression techniques.

Lin et al. [4] have proposed a new index structure based on R-Trees that has been designed to improve query efficiency with the strategy of increasing space to reduce time. Interior nodes have been made to contain data entries also. Data entry contains both Minimum Bounding Rectangle and Maximum Enclosed Circle of spatial data object. It performs better than R-Trees.

Wei et. al [5] propose a methodology to control the access of B+ Treee indexed data in a batch and real time fashion. The number of disk I/O operations required is reduced and system performance is improved without introducing priority inversion. For situations where timing constraints are important, a data reservation mechanism has been described.

Bumbulis [6] has presented an improved index creation method based on a path-compressed binary tree in a database system comprising of database tables and indexes on those tables. A path compressed binary tree for the given index is determined. It is made up of internal nodes and leaf nodes. An index is created comprising a first array of internal nodes encountered during the traversal and a second array of leaf nodes encountered during traversal. The database system employs said first and second arrays for a given key value.

3. CHAIN CODE

A chain code is a form of encoding a boundary. The boundary is converted into a connected sequence of straight line segments of specified length and direction. Each segment's direction is coded by a numbering scheme as shown in Figure 1. The sequence of numbers so generated is called a Freeman chain code. There are two types of chain codes:

1. 4-directional chain code
2. 8-directional chain code

A 4-directional chain code has four directions and provides the numbers 0,1,2 and 3 for the four directions as shown in Figure 1 (a). An 8-directional chain code has eight directions and provides the numbers 0,1,2,3,4,5,6 and 7 for the eight directions as shown in Fig ure1(b).

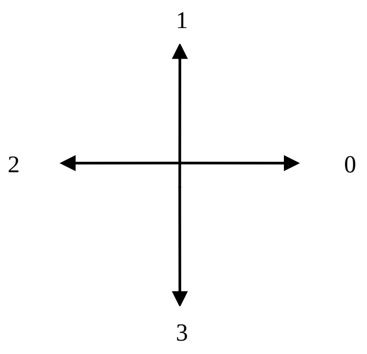

Figure 1(a) 4-directional chain code

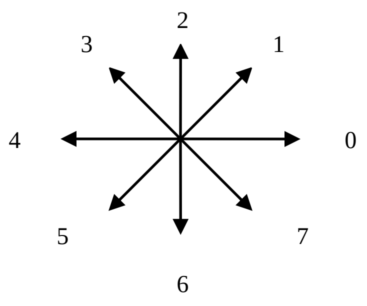

Figure 1(b) 8-directional chain code

The chain code of any image region boundary is computed by taking a starting pixel and moving in counterclockwise or clockwise direction, traversing the region edges and coding each direction using the Freeman chain code direction codes. For example, consider the rectangle drawn in Figure 2. It will be coded starting from the point (1,1) using 4-direction chain code. The rectangle is coded by traversing through it in counterclockwise direction going to point (2,1) , followed by (3,1) and so on, and coding each direction as we move from one pixel to the next one. Therefore, the chain code obtained is 000011222233.

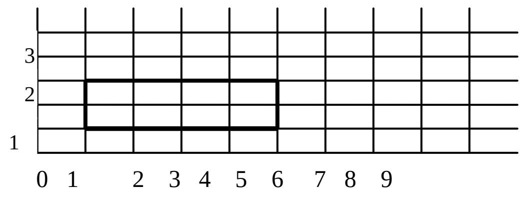

Figure 2. Rectangle shape on grid space

4. B+ TREE

A B+ tree index structure is the most widely used index structure. It is very efficient. It is represented by a balanced tree in which every path from the root of the tree to a leaf of the tree is of the same length. A node of a B+ tree contains upto n-1 search key values and n pointers. The search key values within a node are kept in sorted order. For example, if K_1, K_2, K_3, ….K_n are the search key values and n pointers P_1, P_2, P_3, ……..P_n are there, then $K_i < K_j$ and the node looks like as shown below in Figure. 3.

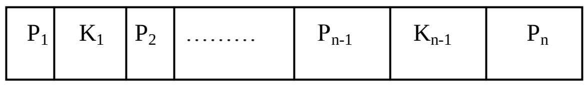

Figure 3. B+ Tree node

In the leaf node, pointer P_i points to either a file record with search key value K_i or to a bucket of pointers, each of which points to a file record with search key value K_i. Non leaf nodes do not contain data pointers. They only contain tree pointers and key values [1].

4.1 Space Overhead of a B+Tree and Number of Comparisons

The space taken by a B+ tree is of O(n) where n is calculated as follows:

$$n= b^h-b^{h-1}$$

where b is the order of the tree and h is the height of the tree.

The order of the tree is computed as follows:

Index node size = floor (b*(tree and data pointer size) + (b-1) *search key size)

where b is the order of the tree.

The height of the tree is calculated as follows:

$h > (1+\log n-\log(b-1))/ \log b$.

where n is the total number of records , b is the order of the tree and h is the height of the tree.

The total number of comparisons is calculated as follows:

$\text{Log}_{(s/2)}$ (total number of records in the database) where s is calculated as follows:

s=disk block size/(search key size+ pointer size).

5. CODED B+TREE: THE PROPOSED APPROACH

The proposed approach suggests some added steps to B+ tree index for indexing an image database based on its chain codes. The approach works for chain codes of length of twenty characters. It can also work for chain codes that are longer than twenty characters. In such cases, the first twenty characters of the chain code are extracted from the entire chain code and then used. However, in this paper, twenty character chain codes are used for demonstrating the proposed work.

First the different twenty character chain codes present in the database are taken and converted into their corresponding index codes. Index codes are discussed in the subsection 5.1 below. As a second step, a B+ tree index structure is built on these index codes. The index codes reduce the space overhead needed by the B+ tree structure to quite a good extent.

5.1 Index Code

Index codes represent any four characters of the chain code in only two characters. All four-character combinations of the chain code are provided with a corresponding two character index code. The index code table has been depicted in Table 1 to show the index codes for the four-character chain codes.

Any character of the chain code can be in the range 0 to 7. Therefore, there can be 4096 types of four-character chain codes where each character can be a number in the range 0 to 7 (8 x 8 x 8 x 8 =4096). Each of these 4096 types is provided with a corresponding two character index code.

Table 1. The Index Code Table

Four Characters of the Chain Code	**Index Code**
0001	01
0002	02
0003	03
0004	04
0005	05 and so on upto 4096 rows.

The index codes are provided for the 4096 chain codes, each of four characters, in the following way:

The first 99 four-character chain codes are given the values from 00 to 99 which are two digits decimal numbers treated as two characters. The next 676 four-character chain codes are given the values in a sequence from AA to AZ, BA to BZ and so on upto ZA to ZZ. Then, the next 676 four-character chain codes are given values from aa to az, ba to bz, and so on upto za to zz. Then, the next 260 four-character chain codes are given values as A0 to A9, B0 to B9 and so on upto Z0 to Z9. The next 260 four-character chain codes are given values from 0A to 9Z, 0B to 9B, and so on upto 0Z to 9Z. Then, the next 1040 four-character chain codes are represented by two-character index code formed by an uppercase alphabet followed by a special character. Forty special characters are taken into consideration for the same. Then, the next 1040 four-character chain codes are represented by two-character index codes formed by a special character followed by an alphabet. Finally, the next 45 four-character chain codes are formed by taking lowercase alphabets followed by special characters. Therefore, 99+676+676+260+260+1040+1040+45=4096 four-character chain codes are assigned two character index codes each.

5.2 The B+ Tree Construction

The B+ Tree structure is built in the following steps:

Step 1:As the first step, all the different twenty character chain codes present in the image database are extracted.

Step 2: For each twenty-character chain code so obtained, the following steps are performed:

(a)Each twenty-character chain code is divided into five sequential sets of four-character chain codes.

(b)Index codes for all these five sets are fetched from the index code table and written in a sequence.

(c)The code so obtained is the coded version of the twenty-character chain code and is used as the search key value for the construction of the B+ tree. The size of the coded version of the chain code is ten characters.

Step 3: Similarly, for all chain codes in the database, the corresponding index codes are obtained by the Step 2 procedure.

Step 4: The coded version of the chain codes so obtained are used for the construction of the B+ tree structure. The tree so constructed is called as Coded B+ tree.

Step 5: The leaf nodes of the B+ tree contain pointers to matching records in the image database.

It is to be noted that a B+ tree index structure is built on the Index Code table. So, the search of index codes in Step 2 (b) is done using B+ tree index structure.

6. COMPARISON

A comparison of space overhead and total number of comparisons required by the coded B+ tree and the conventional B+ tree is given as follows :

6.1 Space Overhead

6.1.1 The Coded B+ tree:

The modified approach uses a search key value of size of 10 characters i.e.10 bytes. The pointers are 2 bytes and disk block size is 4KB or 4000 bytes. The order of the tree is calculated as follows:

b*2+(b-1)*20=4000 which means b=334.

Since the chain code length is 20 characters and it is assumed that chain code field in the database is a key field, so, the total number of records in the database is taken as 8^{20} .Height of the tree is calculated as follows:

h=1+log(8^{20})-log(333) /log(334)= 6.553 equivalent to 7.

The total space overhead is 334^7-334^6= 4.622990855 ^{17}

An index code table's B+ tree size is added to this figure. The total space overhead therefore, is as follows:

4.622990855 ^{17} +666=4.62299E+17

6.1.2 The conventional B+ tree:

The conventional B+ tree makes use of a search key size of 20 characters. The disk block size is 4000 bytes and pointer size is 2 bytes. The order of the tree is calculated as follows:

b*2+(b-1)*20=4000

b=183

Height of the tree is calculated as follows:

h=1+log(8^{20}) – log(182)/log(183)=7.426338818 equivalent to 8.

So, total space overhead is given as follows:

Total space= 183^8-183^7=1.25092E+18

Therefore, the total space savings achieved by Coded B+ tree is 1.25092+18 - 4.62299E+17 = 7.88621E+17

6.2 Total Number of Comparisions

6.2.1 Coded B+ Tree

The search key size is 10 bytes. The pointer size is 2 bytes and the disk block size is 4000 bytes. So, the total number of comparisons for the coded B+ tree is calculated as follows:

S=4000/12=334

$\text{Log}_{[334/2]}$](8^{20})= 7.156740971 equivalent to 7 comparisons.

The total number of comparisons needed to find 5 index codes for five sets of four character chain codes is 6 to 7.

Therefore, in all, the total number of comparisons that would be needed is 7+6=13 to 7+7=14, i.e. around 13 to 14 comparisons.

6.1.2 Conventional B+ Tree

S=4000/22=182

$\text{Log}_{[182/2]}$ (8^{20})=9.219713177 equivalent to 9 comparisons.

Therefore, there are 4 to 5 more comparisons required in case of Coded B+ Tree index structure when compared with the conventional B+ tree.

7. ADVANTAGE OF CODED B+TREE

These days, compression techniques are being applied widely to reduce the size of data to be stored so that the least amount of memory is needed to store a bulk of data. Although, the total number of comparisons required by Coded B+ tree is slightly increased, but, the space needed by the Coded B+ tree is much lesser and is therefore, a useful technique of indexing for systems where data needs to be stored in the least possible amount of space, consuming as less memory as possible. If compression techniques are applied to B+ trees, they also take added computation effort and time for compression and decompression.

8. CONCLUSION

The Coded B+ tree is a B+ tree that can be applied to chain codes of images to search the image database based on chain codes as its search key value. It reduces the space overhead to more than a half. The technique proves to be helpful and useful in memory space savings which is a great demand these days. The Coded B+ tree can be extended with better compression techniques and it can be reduced further to an even smaller size.

9. REFERENCES

[1] Silberschatz A., Korth H.F, Sudarshan S ,2006. Database system Concepts" ,Fifth Edition, Mc Graw Hill International Edition, pp 481-503.

[2] Guttman A., 1984 R-trees: A dynamic index structure for spatial searching, in Proc. of ACM SIGMOD, Boston, MA, pp. 47–57.

[3] Bayer R., Unterauer K. 1977. Prefix B- Trees, ACM Transactions on Database Systems, Volume 2, no. 1, March 1977, Page 11-26.

[4] Lin Weihua Huazhong, Wu Yonggang, Tan Xiajun, 2008. An improvement of Index Method and Structure based on R-Tree, Proceedings of International Conference on Computer Science and Software Engineering, 2008, Volume 4, pp 607-610, 2008, IEEE Computer Society, Washington DC,USA, ISBN:978-0-7695-3336-0

[5] Wei Chih-Hung, Kam-Yiu-Lam 2000. Real Time Data Access Control on B-Tree Index Structures, Sydney, Australia, March 23-26, ISBN-0-7695-0071-4. , Real Time Systems, 19,245-282 (2000), 2000 Kluwer Academic Publishers, Boston, Manufactured in Netherlands.

[6] Bumbulis P. 2003. System and methodology for providing compact B-Tree, US Patent Application Publication, Publication no.: US2003/0204513 A1, publication date: October 30,2003.