Web-based Hpa-an City Telephone Directory by implementing B-tree Database

Nyo Lay Myint, Soe Hay Mar
Computer University, Hpa-an, Myanmar
nyolaymyint2010@gmail.com

Abstract

A directory is an essential part of people's lives today. By accessing static directories like the paper phone books, people can easily find out phone numbers of other people, shops, organizations etc. But, all information in the paper directories is static and does not change in real time. So, conventional applications and paper works were instituting with web-based application. Online directory is necessary for modern lifestyle and have the capacity to be much more up-to-date. And, more amount of information can be added to an online directory like the organization, department in which that person works or the location of his/her office etc.

In computer science, b-tree is a tree data structure that keeps data sorted, allows search, insertion, deletion and sequential access in logarithmic amortized time and the special thing about B-trees is that they are designed for really huge indexed databases. The system build web-based telephone directory of Hpa-an (local city of Myanmar) by implementing B-tree database. B-tree results are shown as according to b-tree's operations (searching, inserting, and deleting) with user's wanted information. There will be b-tree's operation time, b-tree's traveled nodes, and size of b-tree in b-tree results. And, apache tomcat 6.0.26 server is used as web service in the system.

1. Introduction

B-tree is a balanced multi-way search tree designed to work well on disks. The B-tree's creators, Rudolf Bayer and Ed McCreight, have not explained what the B-stands for. The most common belief is that B-stands for balanced, as all the leaf nodes are at the same level in the tree. B may also stand for Bayer, or for Boeing, because they are working for Boeing Scientific Research Labs at this time [8, 9]. B-Trees are not binary trees: each node can have many children. Each node of a B-tree potentially contains several keys, not just one. When doing searches, we decide which child link to follow by finding the correct interval of our search key in the key set of the current node [1, 5]. It can give efficient insert and delete at the expense of some space overhead. It must be ensured that searching and processing are efficient in terms of memory space and computation time [3]. B-trees are the basis for many commercial database systems and optimized for using minimum number of disk operations for large data structures. But in general, anytime an application needs an index, B-trees offer an efficient data structure to use [9, 10].

There is no web based Hpa-an city telephone directory in my city. So, the system will construct the telephone directory that can provide both of phone number and related information for Hpa-an by implementing B-tree database.

2. Data Structure of B-tree

B-trees are trees like data structures most commonly used in databases and file systems. An index is a feature in a database that allows quick access to the rows in a table. An index is created using one or more columns of the table. Indexes are often optimized for quick searches, usually via balanced tree.

A B-tree index is usually created on columns which contain mostly unique values. A B-tree index is most effective when a query retrieves less than twenty percent of rows in a table. B-tree contains following different types of nodes [6, 9]:

- One root node: A node that contains node pointed to branch nodes.
- Two or more branch nodes: Branch node contain pointer to leaf nodes or other branch nodes.
- Many leaf nodes: A leaf node contains index items and horizontal pointers to other leaf nodes.

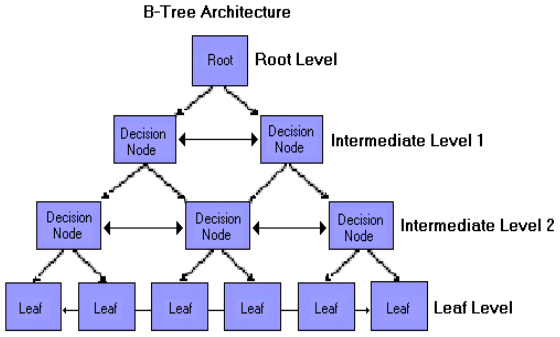

Figure 1.Architecture of B-tree

A database is a collection of data organized in a fashion that facilitates updating, retrieving, and maintaining the data. Database products like Microsoft SQL Server, Sybase Adaptive Server, IBM DB2, and Oracle serve as a foundation for accounting systems, inventory systems, medical record keeping systems, airline reservation systems, and countless other important aspects of modern business [5].

It is not uncommon for a database to contain millions of records requiring many gigabytes of storage. For examples, TELSTRA, an Australian telecommunication company, maintains a customer billing database with 51 billion rows, and 4.2 terabytes of data. In order for a database to be useful and usable, it must support the desired operation, such as retrieval and storage quickly. B-tree is a good way to do this [2, 5].

Because databases cannot typically be maintained entirely in memory, b-trees are often used to index the data and to provide fast access. If the same data is indexed with a b-tree of minimum degree 10, 114 comparisons will be required in the worst case. Clearly, indexing large amount of data can significantly improve search performance. Although other balanced tree structures can be used, a b tree also optimizes costly disk accesses that are of concern when dealing with large data sets. B-tree offer database implementation for indexes. For instance Oracle, Berkeley and MYSQL, use B trees for indexes.

3. Background Theory

If data is added or deleted in a tree structure, leaves randomly develop and the operation efficiency will decrease. A tree structure capable of reorganizing itself is called a balanced multi-way search tree (B-tree). B tree is a further developed version of a binary tree. Unlike a binary tree, each node of a b-tree may have a variable number of keys and children. The keys are stored in non-decreasing order. Each key has an associated child that is the root of a subtree containing all nodes with keys less than or equal to the key but greater than the preceding key. A node also has an additional rightmost child that is the root for a subtree containing all keys greater than any keys in the node [5, 10].

A b-tree has a minimum number of allowable children for each node known as minimization factor. With a large branching factor m, the height of a B-tree is low resulting in fewer disk accesses. The branching factor can be chosen such that a node reference corresponds to a block of secondary memory. B-trees usually keep related records on the same block. Again this results in fewer disk accesses.

A B-tree of order *m* is an *m*-way tree (i.e., a tree where each node may have up to *m* children) in which [2]:

1. The number of keys in each non-leaf node is one less than the number of its children and these keys partition the keys in the children in the fashion of a search tree.
2. All leaves are on the same level.
3. All non-leaf nodes except the root have at least $\lceil m/2 \rceil - 1$ keys.
4. The root is either a leaf node, or it has from two to *m* children.
5. A leaf node contains no more than $m - 1$ keys, the number *m* should always be odd.

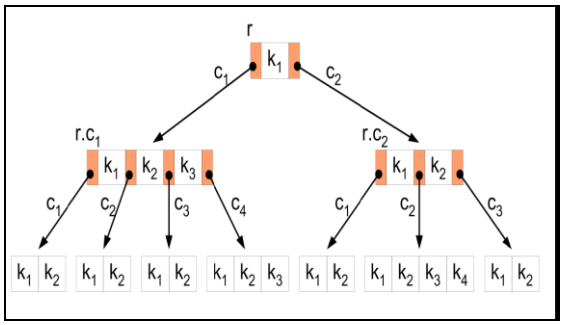

Figure 2.Schema showing the B-tree Structure

4. B-tree's Algorithms

B-Tree-Search(x, k)
i <- 1
while i <= n[x] and k > key_i[x]
do i <- i + 1
if i <= n[x] and k = key_i[x]
then return (x, i)
if leaf[x]
then return NIL
else Disk-Read(c_i[x])
return B-Tree-Search (c_i[x], k)

The search operation on a b-tree is analogous to a search on a binary tree. Instead of choosing

between a left and a right child as in a binary tree, a b-tree search must make an n-way choice. The correct child is chosen by performing a linear search of the values in the node. After finding the value greater than or equal to the desired value, the child pointer to the immediate left of that value is followed. If all values are less than the desired value, the rightmost child pointer is followed. Of course, the search can be terminated as soon as the desired node is found.

```
B-Tree-Insert (T, k)
r <- root[T]
if n[r] = 2t - 1
    then s <- Allocate-Node()
        root[T] <- s
            leaf[s] <- FALSE
            n[s] <- 0
            c1 <- r
            B-Tree-Split-Child(s, 1, r)
            B-Tree-Insert-Nonfull(s, k)
  else B-Tree-Insert-Nonfull(r, k)
```

All insertions start at a leaf node. To insert a new element, search the tree to find the leaf node where the new element should be added. Insert the new element into that node with the following steps:

(1) If the node contains fewer than the maximum legal number of elements, then there is room for the new element. Insert the new element in the node, keeping the node's elements ordered. Otherwise the node is full, so evenly split it into two nodes.

(2) A single median is chosen from among the leaf's elements and the new element.

(3)Values less than the median are put in the new left node and values greater than the median are put in the new right node, with the median acting as a separation value.

(4)Insert the separation value in the node's parent, which may cause it to be split, and so on. If the node has no parent (i.e., the node was the root), create a new root above this node (increasing the height of the tree).

```
B-Tree-Delete(x, k)
if x is a leaf then
if k is in x then
delete k from x and return true
else return false
else
if k is in x then
y = the child of x that precedes k
if y has at least t keys then
k' = the predecessor of k (use B-Tree-FindLargest)
Copy k' over k
B-Tree-Delete(y, k')
else
z = the child of x that follows k
if z has at least t keys then
k' = the successor of k
Copy k' over k
B-Tree-Delete (z, k')
else
merge k and all of z into y
B-Tree-Delete(y, k)
else //k is not in internal node x.
ci[x] points to the root, c, of the subtree that could
contain k.
if c has t-1 keys then
if c has an immediate left/right sibling, z, with t or
more keys then
Let k1 be the key in x that precedes/follows c.
Move k1 into c as the first/last key in c.
Let k2 be the last/first key in the immediate left/right
sibling, z.
Replace k1 in x with k2 from z (i.e., move k2 up into
x).
Move the last/first child subtree of z to be the
first/last child subtree of c.
else
merge c with one of its immediate siblings and
make the appropriate key of x the middle key of the
new node, c.
B-Tree-Delete(c, k)
```

To delete a key, first perform the usual search operation to locate the node containing the key. (If the key isn't found, it isn't in the tree and can't be deleted.). If the value is in a leaf node, it can simply be deleted from the node. If underflow happens, check siblings to either transfer a key or fuse the siblings together. If deletion happened from right child retrieve the max value of left child if there is no underflow in left child in vice-versa situation retrieves the min element from right.

5. Implementation of Web-based Hpa-an Telephone Directory

This is applying web services from online telephone directory web site. In the system, the b-tree database stores about 1536; total records of Hpa-an telephone information. The related information (phone number, name, business category, and address) can be searched as according to user's searching key (by phone number, by name, or by business category) and b-tree's operation results are shown with these related information in the system. There will be b-tree's traveled nodes, b-tree's operation time and size of b-tree in operation results.

Administrator is responsible for updating information that he or she wanted. In this system, administrator is authorized person who is permitted to hold this telephone directory database. And more city phone code of our country can be found in Hpa-an web-based telephone directory.

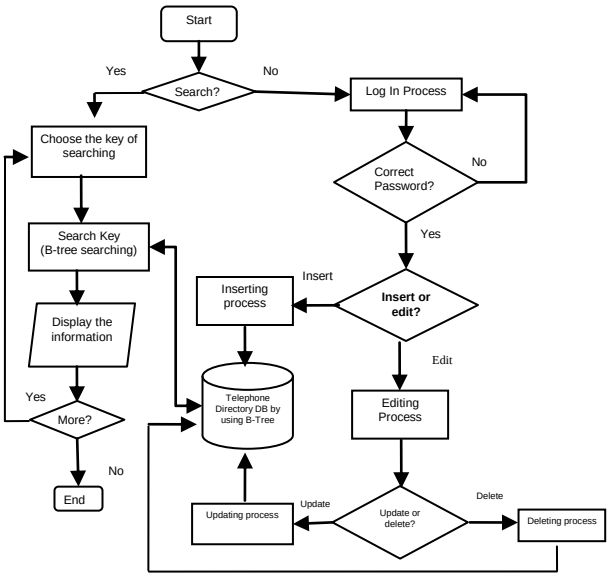

Figure 3.System Flow Diagram

6. Experimental Results

The number of nodes we need to access during a search for a record with a given key value is equal to the height of the tree plus one. Time efficiency depends on the height (h) of the tree.

The root is either a leaf or has between 2 and m children. A leaf has between 1 and m-1 entries or keys. Each node, except the root and the leaves, has between $\lceil m/2 \rceil$ and m children and keys between $\lceil m/2 \rceil$-1and m-1. The root of the tree will contain, at least, one key. So, finding the smallest number of keys a B-tree of order m and height h can have:

Level (1): It will have at least two nodes with at least $\lceil m/2 \rceil$– 1 keys in each of them, for the total minimum number of keys 2($\lceil m/2 \rceil$ -1).

Level (2): It will have at least $2\lceil m/2 \rceil$ nodes (the children of the nodes at level 1) with at least $\lceil m/2 \rceil$– 1 in each of them, for the total minimum number of keys $2\lceil m/2 \rceil$ ($\lceil m/2 \rceil$-1).

In general, the nodes at the **level *i***, will contain at least $2\lceil m/2 \rceil^{i-1}(\lceil m/2 \rceil - 1)$keys. Finally, level ***h***, the leaf level, will have at least $2\lceil m/2 \rceil^{h-1}$. Thus, the minimum number of nodes and height are following:

$$n \geq 1 + \sum_{}^{i-1} 2\lceil m/2 \rceil \quad (\lceil m/2 \rceil - 1) + 2\lceil m/2 \rceil^{h-1}$$

$$n \geq 4 \lceil m/2 \rceil^{h-1} - 1$$

$$h \geq \left\lfloor \log_{\lceil m/2 \rceil} \frac{n+1}{4} \right\rfloor + 1$$

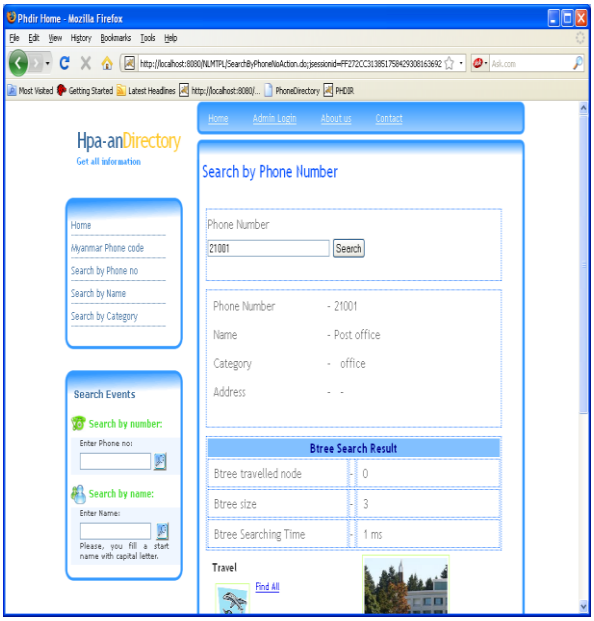

Figure 4.Result of searching time, traveled nodes and size of B-tree

In the figure, b-tree results are shown as according to b-tree's operations (searching, inserting, and deleting) with user's wanted information (phone number, name, business category and address). There will be b-tree's operation time, b-tree's traveled nodes, and size of b-tree in b-tree results. B-tree's traveled nodes show that how many nodes are traveled to find a target key.

7. Conclusion

The payoff of the B-tree insert and delete rules are that B-trees are always "balanced". So, B-tree data representation methods are very useful for storing and searching huge amounts of data. It needs comparatively small memory space and less computation time because of B-tree disk-based storage structure. In many database and file systems, b-tree is an efficient, useful and versatile. Moreover, our system will also provide the web-based user interface to be quick exploit the local Hpa-an telephone number and related information.

In the system, the database stores about 1536; total records of Hpa-an telephone information. As limitation, the system implements telephone directory database only Hpa-an record, not include Kayin state. So, as further extension, the system can be extensible to Kayin State phone directory.

8. References

[1] A. Kaltenbrunner, L. Kellis & D. Mart´ı 1, B-trees.

[2] Bayer, R., M. Schkolnick, "*Concurrency of Operations on B-Trees. In Readings in Database Systems"*, (ed. Michael Stonebraker), pages 216-226, 1994.

[3] Benoit Maréchal, RAQUEL B-tree File Stack – B-tree Introduction, 21 May 2007.

[4] D.Comer, "The Ubiquitous B-Tree", Computer Science Department, Purdue University, West Lafayette, Indiana 47907. Computing Surveys, Vol 11, No2, June 1979.

[5] Peter Neubauer, B-trees: Balanced Tree Data Structures.

[6] Ruchir Babbar, Database Management, B-Tree (Balanced Tree), Research Paper.

[7] The Perl Journal, Bricolage: B-Trees.

[8] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001.

[9] http://en.wikipedia.org/wiki/Btree

[10] http://en.wikipedia.org/wiki/Data_structure

[11] http://mattfleming.com/node/192

[12] http://topic.asp.htm

[13] http://www.forums.sum.com/thread.jspa.htm

[14] http://www.articlesOf-Online-White-Pages-hone-directory/1425901#ixzz0xsD87e18 under Creative Common License.