ISSN (e): 2250 – 3005 || Vol, 04 || Issue, 10 || October – 2014 ||
International Journal of Computational Engineering Research (IJCER)

Furnish an Index Using the Works of Tree Structures

[1]Chiranjib Mukherjee [2]Dr.Gyan Mukherjee
[1]Research Scholar in Computer Science, Magadh University (INDIA)
[2]Department of Mathematics, S.P.S. College, B.R.A.B.University (INDIA)

ABSTRACT
We consider two tree-based indexing schemes that are widely used in practical systems as the basis for both primary and secondary key indexing. We define B-tree and its features, advantages, disadvantages of B-tree. The difference between B^+-tree and B-tree has also been discussed. We show the algorithm, examples and figures in the context of B^+-tree.

KEYWORDS: *Indexing Schemes, Primary key, Secondary key, B-tree, B^+-tree.*

I. INTRODUCTION

In a tree based indexing scheme the search generally starts at the root node. Depending on the conditions that are satisfied at the node under examination, a branch is made to one of several nodes, and the procedure is repeated until we find a match or encounter a leaf node.

Tree Schemes: Each node of the tree except the leaf nodes can be considered to consist of the following information:

$$[n, T_{i1}, k_{i1}, T_{i2}, k_{i2}, \ldots\ldots, T_{in}, k_{in}, T_{i(n+1)}]$$

Where the k_{ij}'s are key values and the T_{ij}'s are pointers. For an m-order tree the following conditions are true:

- $n < m$
- $k_{i1} \leq k_{i2} \leq \ldots. \leq k_{in}$
- each of the pointers, T_{ij}, $1 \leq j \leq (n+1)$, points to a sub tree containing values less than k_{ij} and greater than or equal to $k_{i(j-1)}$.

The leaf nodes of the B^+-tree are quite similar to the non leaf nodes, except that the pointers in the leaf nodes do not point to subtrees. The pointers T_{Lj}, $1 \leq j \leq n$, in the leaf nodes point to storage areas containing either records having a key value k_{ij}, or pointers to records, each of which has a key value k_{ij}. The number of key values in each leaf node is at least $[(m – 1)/2]$ and at most m-1.

The pointer $T_{L(n+1)}$ is used to chain the leaf nodes in a sequential order. This allows for sequential processing of the underlying file of records. The following conditions are satisfied by the nodes of a B^+-tree :-

- The height of the tree is ≥ 1.
- The root has at least two children.
- All nodes other than the root node and the leaf nodes have at least [m/2] children, where m is the order of the tree.
- All leaf nodes are at the same level.

Operations: All operations of B^+-tree require access to the leaf nodes.

Search: The number of nodes accessed is equal to the height of the tree. Once the required leaf node is reached, we can retrieve the pointer for the storage location containing the records; knowing the storage location, we can retrieve the required records.

Insertion: We assume that the records themselves would be inserted in the pertinent storage locations. Insertion and deletion that violates the conditions on the number of keys in a node requires the redistribution of keys among a node, its sibling and their parent. If after insertion of the key, the node has more than m-1 keys, the node is said to overflow. Overflow cam be handled by redistribution if the number of entries in the left or right sibling of the node is less than the maximum.

Deletion: The leaf node containing the key to be deleted is found and the key entry in the node deleted. If the resultant node is empty or has fewer than [(m-1)/2] keys,

- The data from the sibling nodes could be redistributed, i.e., the sibling has more than the minimum number of keys and one of these keys is enough to bring the number of keys in node TD to be equal to [(m-1)/2]
- The node TD is merged with the sibling to become a single node. This is possible if the sibling has only the minimum number of keys. The merger of the two nodes would still make the number of keys in the new nodes less than the maximum.

Capacity: The upper and lower limits of the capacity of a B^+-tree of order m may be calculated by considering each node of the tree to be maximally (m-1) keys or minimally full ([m/2]-1)keys. Let the height of the tree is h. As every key must occur in the leaf node and the leaf nodes may also contain a minimum of [(m-1)/2] and a maximum of (m-1) keys we have

$$2 * [(m-1)/2 * [m/2]^{h-2} < N < (m-1) * m^{h-1}$$

B-Tree : (Balanced Tree) The basic B-tree structure has growth to become one of the most popular techniques for organizing an index structure while accessing the records using such structure, several conditions of the tree must be true, to reduce disk access:

- The Height of the tree must be kept to a minimum.
- These must be no empty sub trees above the leaves of the tree.
- The leaves of the tree must be at the same level.
- All nodes except the leaves must have as few as two and as many as the maximum number of children.

The Features of a B-tree:-

- There is no redundant storage of search key values i.e., B-tree stores each search key value in only one node, which may contain other search key values.
- The B-tree is inherently balanced and is ordered by only one type of search key.
- The insertion and deletion operations are complex with the time complexity $O(\log_2 n)$.
- The number of keys in the nodes is not always the same. The storage management is only complicated if you choose to create more space for pointers keys otherwise the size of a node is fixed.
- The B-tree grows at the node as opposed to the Binary tree, BST and AVL trees.
- For a B-tree of order N with n nodes, the height is log n. The height of B-tree increases only because of a split at the root node.

Advantages of B-tree indexes:-

- There is no overflow problem inherent with the type of organization it is good for dynamic table- those that suffer a great deal of insert / update / delete activity.
- Because it is so large extent self-maintaining, it is good in supporting 24 hours operation.
- As data is retrieved by the index, it is always presented in order.
- 'Get next' queries are efficient because of the inherent ordering of rows within the index blocks.
- B-tree indexes are good for every large tables because they will need minimal reorganization.
- There is predictable access time for any retrieval because the B-tree structure keeps itself balanced, so that there is always the same number of index levels does increases both with the number of records and the length of the key value.

Disadvantages of B-tree indexes:-

- For static tables, there are better organizations that require fewer I/Os. ISAM indexes are preferable to B-tree in this type of environment.
- B-tree is not really appropriate for every small table because index look-up becomes a significant part of the overall access time.
- The index can use considerable disk space, especially in products which allow different users to create separate indexes on the same table/ column combinations.
- Because the indexed themselves are subject to modification when rows are updated, deleted or inserted, they are also subject to locking which can inhabit concurrency.

Difference between B^+-tree and B-tree:-

- Retrieval of the next record is relatively easy in the B^+-tree, this is not the case in the B-tree unless the internal nodes of the B-tree are linked in a sequential order.
- The deletions in a B^+-tree are always made in the leaf nodes. In a B-tree, a value can be deleted from any node, making deletions more complicated than in a B^+-tree.

- Insertions in a B^+-tree are always made in the leaf nodes. In the B-tree, insertions are made at the lowest non leaf node. Insertions (or deletions) may cause node splits and thereby affect the height of the tree in both cases.
- The capacity of the B-tree can be calculated in a manner similar to that used for the B^+-tree. That the order of the tree is dictated by physical storage availability, among other factors. For the same buffer size, the order of the B-tree would be less than that of the B^+-tree.

II. ALGORITHM – SEARCHING B^+-TREE

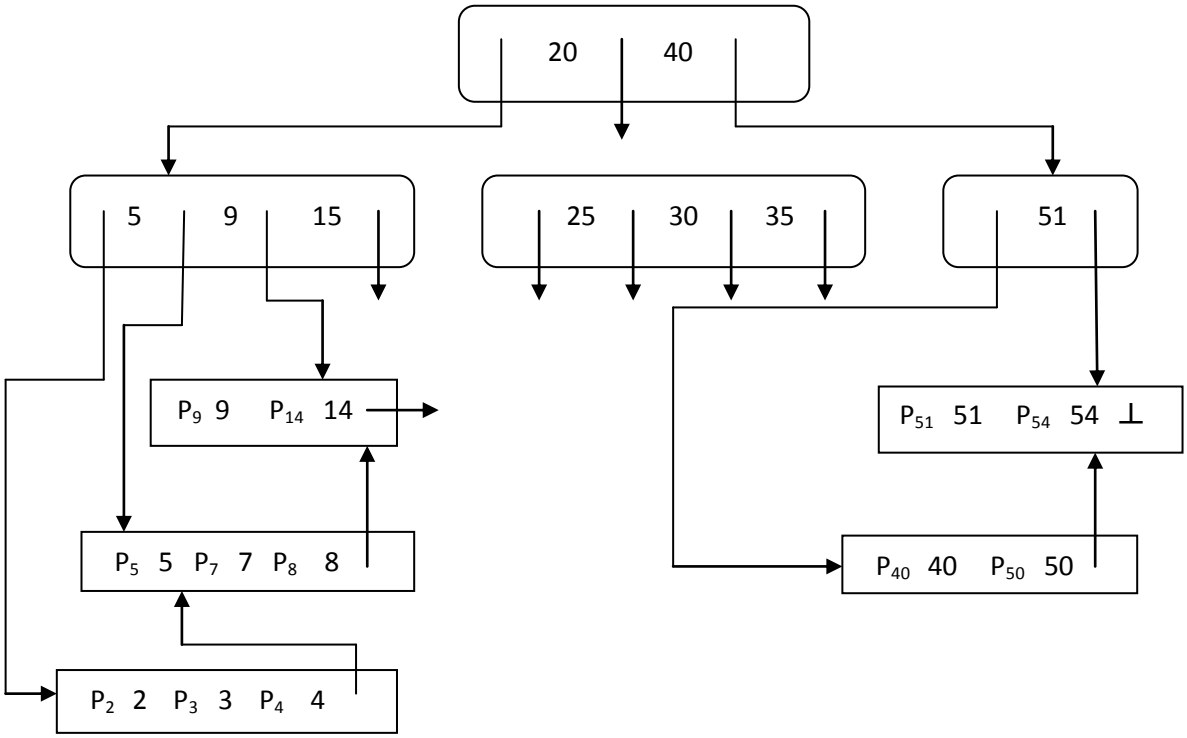
K_s, the search key
Found,(a Boolean value), and
A, the address of record if found
{nodes content : [n, T_1, k_1, T_2, k_2 ……, T_n , k_n, T_{n+1}]$k_{n+1} = \infty$ is assumed}
get root_node
while not leaf_node do
begin
i : = 1
while not (i > n or $K_s < k_i$) do
i : = i + 1
{T_i points to the subtree that may contain K_s}
get subtree T_i
end {while not leaf_node}
{search leaf node for key K_s}
{content of leaf node : [n, P_1, k_1, P_2, k_2 ……, P_n , k_n, P_{n+1}]}
i : = 1
found : = false
while not (found or i > n) do
begin
found : = $K_s = K_i$
if found then
A : = P_i
else i : = i + 1
end {while not (found or i > n)}

III. EXAMPLES AND FIGURES

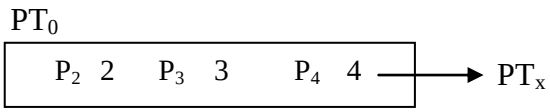
Example1: Given a file containing the following records-

Books	Subject Area
2	Files
3	Database
4	Artificial intelligence
5	Files
7	Discrete structures
8	Software engineering
9	Programming methodology
.	.
.	.
.	.
40	Operating system
50	Graphics
51	Database
52	Data structures

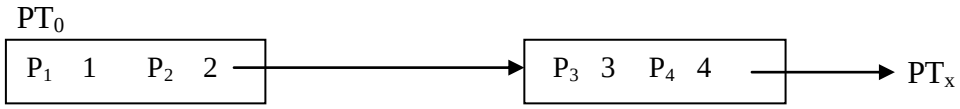
Figure1: A B^+- tree of order 4 on Book- Each P_i is a pointer to the storage area containing records for the key Books = i; ⊥ represents a null pointer.



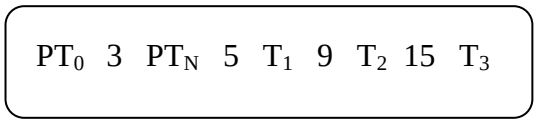
Example2: In the B^+-tree of example1, let us insert an entry for Books 1. The original contents of the leaf node (with the label PT_0) in which the key would be inserted are:



This nodes does not have a left sibling and the right sibling is already full. Hence, insertion of the key 1 would cause a split. Let the new node be PT_N. The contents of these nodes are below:

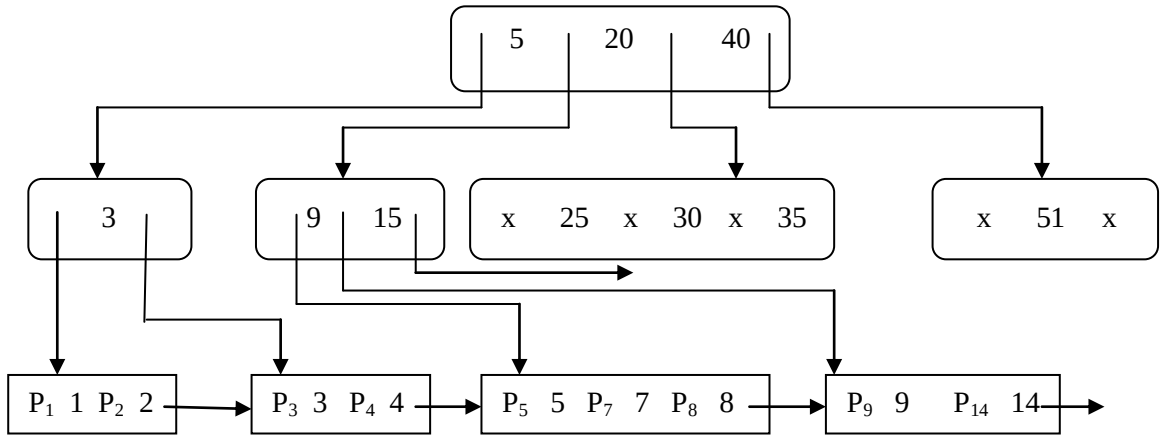


The pair < 3, PT_N > are passes to the parent node for insertion as indicated:

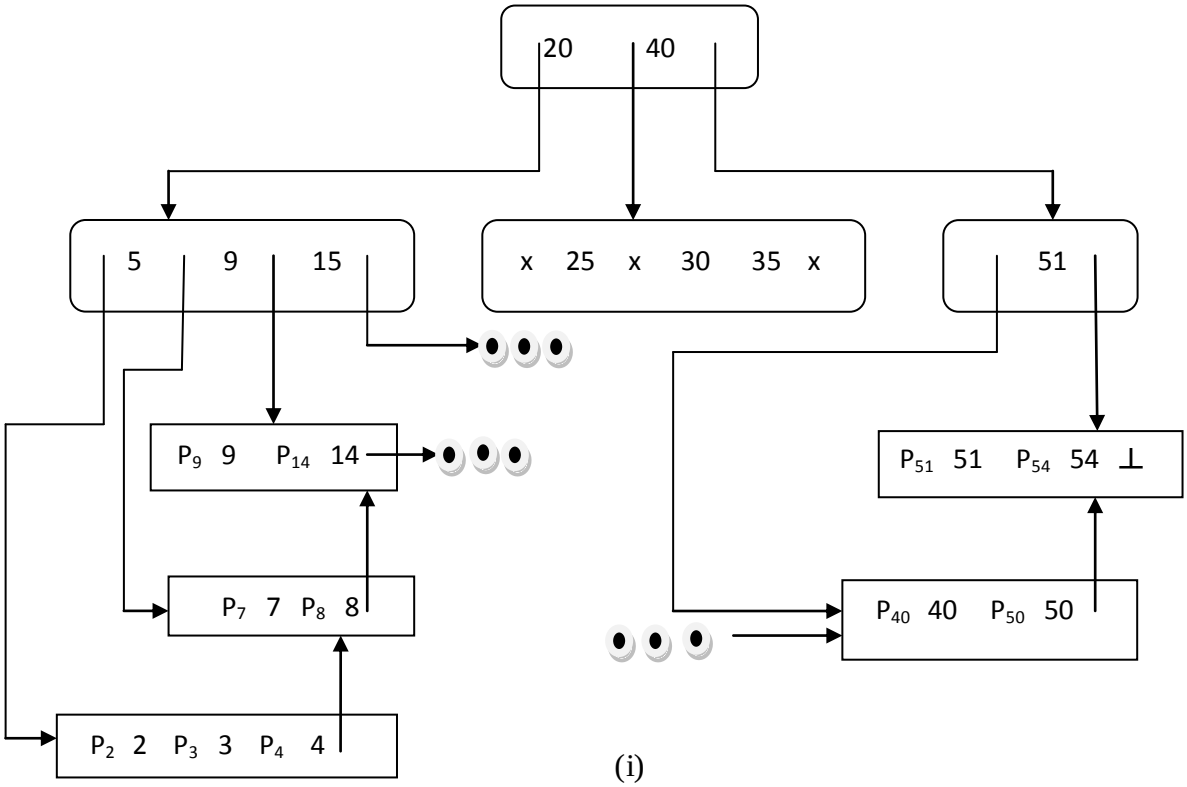


The insertion causes a split of this node into the following two nodes with the key value 5, along with a pointer passed to the parent of the node:

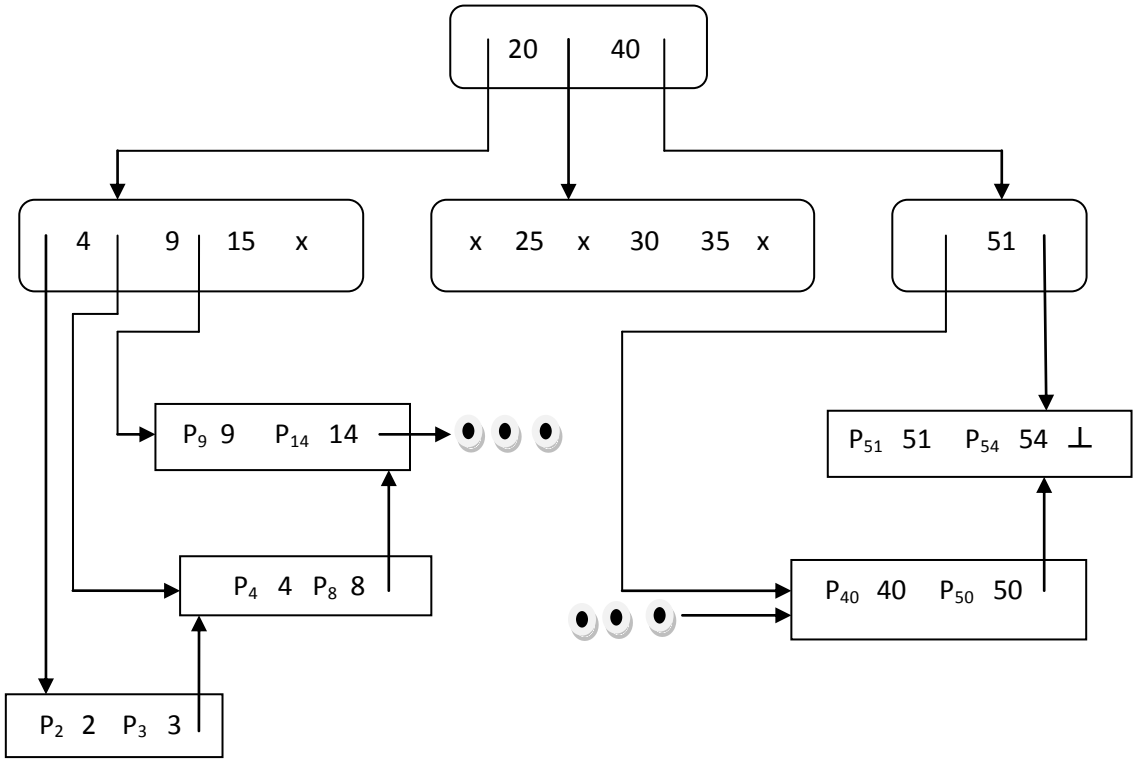
Let the address of the new node be P_Y . Then the pair < 5, P_Y > is passed to the parent node for insertion.
Figure2: The B^+- tree of example1 after insertion of the key for Books1.



Example3: Let delete the entry for Books5 from the tree shown in example1. The resultant tree is shown in part (i) of fogure3.
Figure3: (i) The B^+-tree that results after the deletion of key 5 from the tree of example1.
(ii) The B^+-tree after the deletion of key 7.



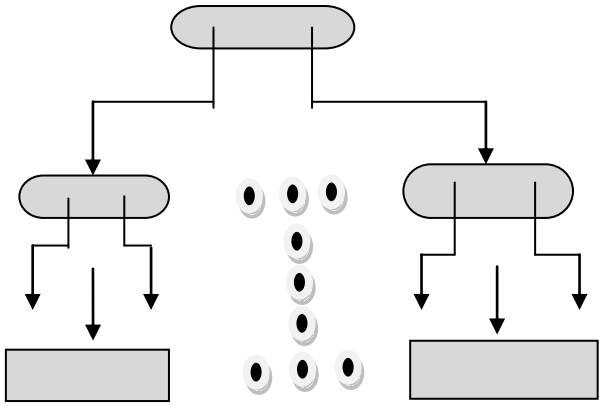
(i)



(ii)

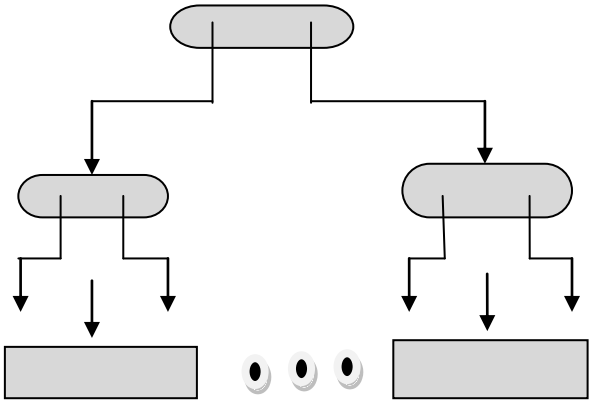
Figure 4: Capacity of a B^+-tree.

Level	Number of nodes at level
1	1
2	m
.	.
.	.
.	.
h	m^{h-1}



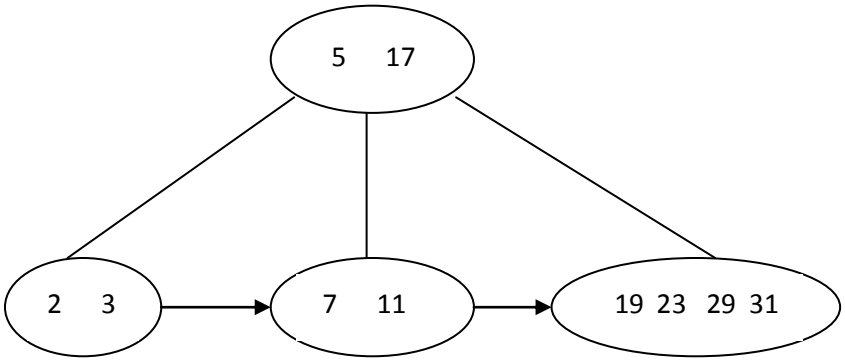
(a)

Level	Number of nodes at level
1	1
2	m
.	.
.	.
.	.
h	$2*[m/2]^{h-1}$

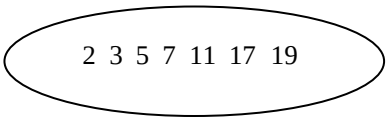


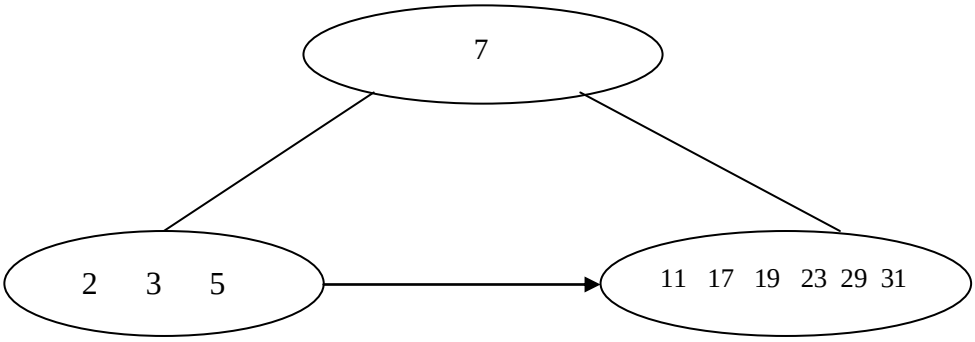
(b)

Example5: A B^+-tree for the following set of key values-(2,3,5,7,11,17,19,23,29,31).That the number of search key values that fit in one node is (a) 3 and (b) 7.
Figure: (a)



(b)



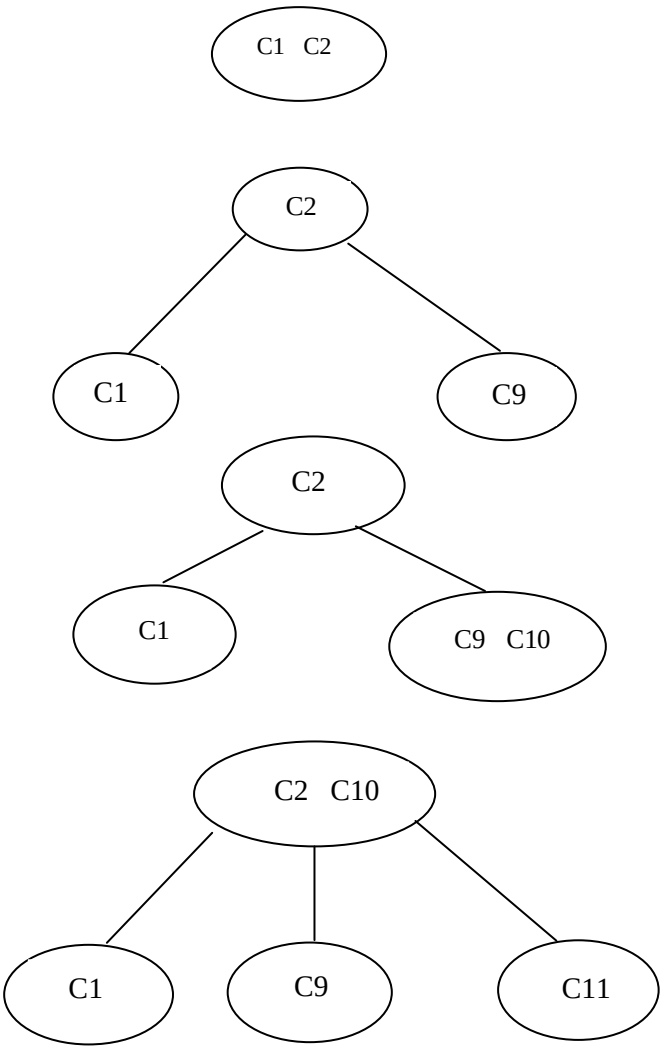


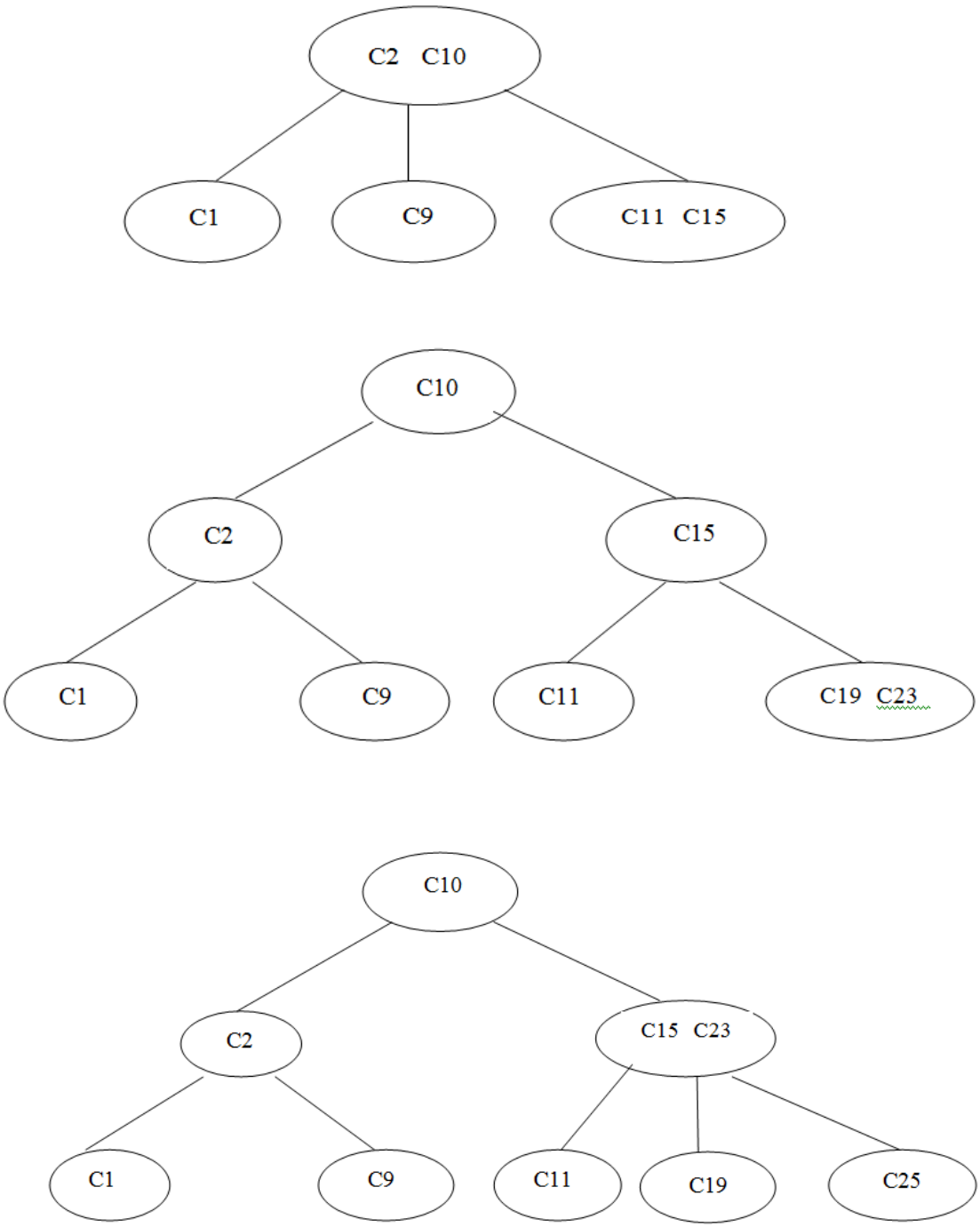
Example 6: Create a B-tree structure of the order 3 of the following relation.

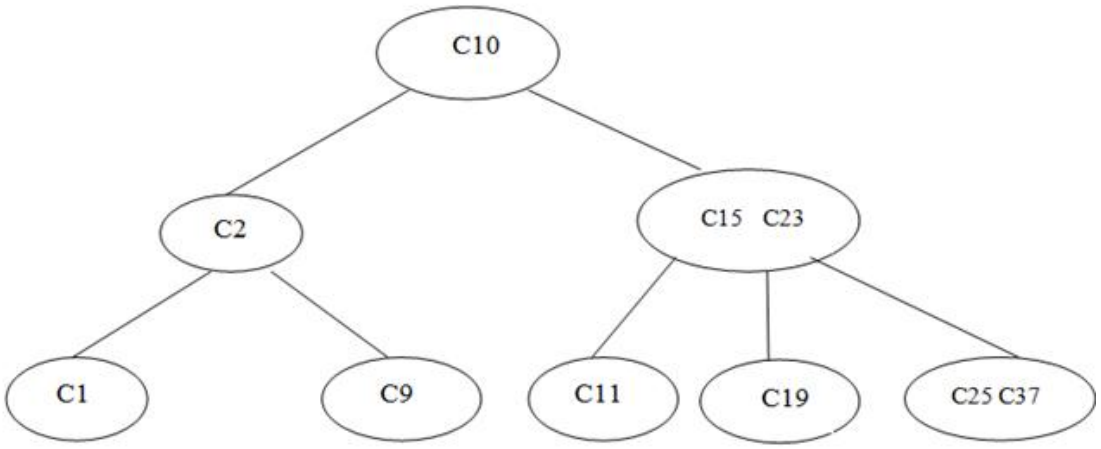
Customer

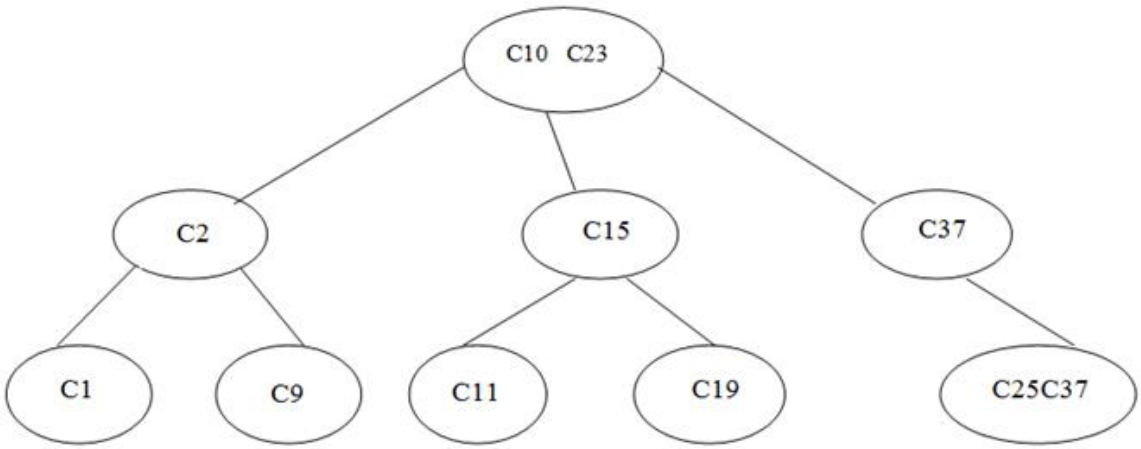
C_No.	Name	Location
C1	N1	L1
C2	N2	L2
C9	N3	L2
C10	N4	L3
C11	N5	L3
C15	N6	L3
C19	N7	L4
C23	N8	L3
C25	N9	L4
C37	N10	L2
C32	N11	L2
C34	N12	L1

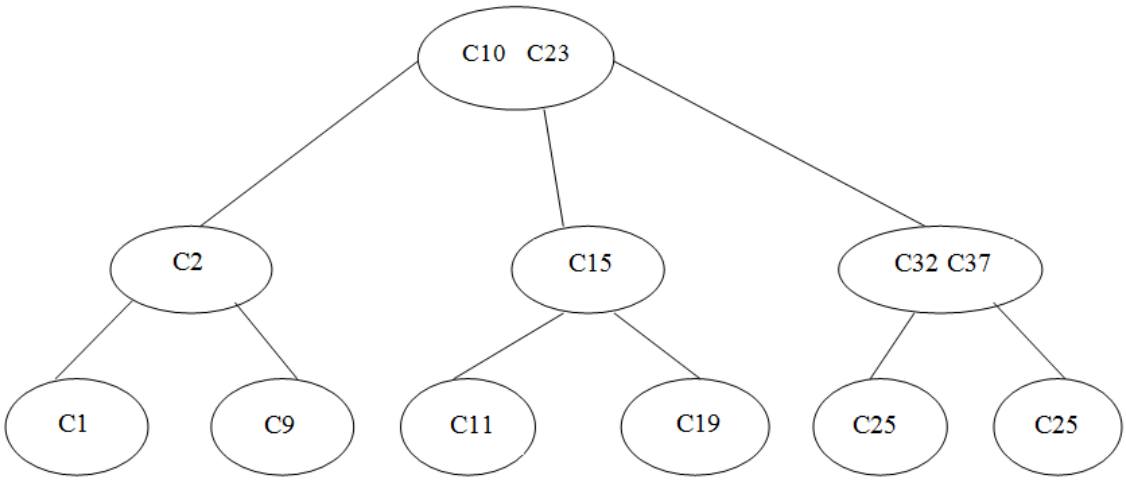
Figure 6:




C2 C10
C1
C9
C11 C15
C10
C2
C15
C1
C9
C11
C19 C23
C10
C2
C15 C23
C1
C9
C11
C19
C25


C10
C2
C15 C23
C1
C9
C11
C19
C25 C37


C10 C23
C2
C15
C37
C1
C9
C11
C19
C25C37


C10 C23
C2
C15
C32 C37
C1
C9
C11
C19
C25
C25

IV. CONCLUSION

Tree-based data organization schemes are used both for primary and secondary key retrieval. The B^+-tree scheme, each node of the tree except the leaf node contains a set of keys and pointers pointing to sub trees. The leaf nodes of the B^+-tree are similar to the non leaf or internal nodes except that the pointers in the leaf node point directly or indirectly to storage areas containing the required records. We also examined the method of performing the search and update operations using the B^+-tree and compared the B^+-tree with the B-tree.

REFERENCES

[1.] Avi Silberschatz, Henry F. Korth & S. Sudarshan; Database System Concept, 2010.
[2.] Alan L. Tharp; File Organization and Processing, John Wiley & Sons,2008.
[3.] Carolyn Begg , Thomas Connolly; Database System: A Practical Approach to Design, Implementation and Management, Addison-Wesley,2004.
[4.] Ramez Elmasri & Shamkant B. Navathe; Fundamental of Database Systen, Fourth Edition, Pearson Addison Wesley, NewYork,2003.
[5.] Hctor Garcia-Molina; Database System Implementation, Pearson Education,2000.
[6.] T. R. Harbon; File System Structures of Data Structures and Algorithms, Englewood Cliffs, NJ:Printice-Hall,1988.
[7.] P. Goyal; "File Organization" Computer Science report, Concordia University, Montreal,1987.
[8.] S. P. Ghosh; Database Organization for Data Management, Orlando, Academic Press, 1986.
[9.] E. Horowitz & S. Shani; Fundamentals of Data Structures, Rockville, Computer Science Press, 1982.