Assignment 3
UE15CS311
Mohammed Salamuddin
01FB15ECS175

B Trees

ABSTRACT

In computer science, a **B-tree** is a self-balancing tree data structure that keeps data sorted and allows searches, sequential access, insertions, and deletions in logarithmic time. The B-tree is a generalization of a binary search tree in that a node can have more than two children. Unlike self-balancing binary search trees, the B-tree is optimized for systems that read and write large blocks of data. B-trees are a good example of a data structure for external memory. It is commonly used in databases and filesystems.

INTRODUCTION

The program to read a data set of 1 million records.insert the keys into the B tree and search for the record efficiently by its key is implemented in the C language.

The program has 2 methods of implementation:

- Implemented as array: all records are brought into the main memory.
- Implemented as a file: records exist on the disk (secondary memory) only,and are accessed and read to the main memory on search.

OBJECTIVES

The main objective of this exercise is to implement the following:

- Create a B tree (as array and file) for the data set given.
- Search of a record in the B tree.
- Delete a record from the B tree.
- With different values of t-minimum degree of each node.

APPROACH

1. **Introduction**

The program to read a data set of 1 million records.insert the keys into the B tree and search for the record efficiently by its key is implemented in the C language.
Implemented as array: all records are brought into the main memory. And Implemented as a file: records exist on the disk (secondary memory) only,and are accessed and read to the main memory on search.

2. Methodology

B tree contains a structure for each node,containing:

1. The number of keys in the node.
2. A leaf variable to say if the node is leaf or not
3. An array of keys
4. An array of the references to the childs of the node
5. And a root which points to the root of the B tree

A B-tree T is a rooted tree (whose root is T:root) having the following properties:

1. Every node x has the following attributes:
   a. x:n, the number of keys currently stored in node x,
   b. the x:n keys themselves, x:key 1 ; x:key 2 ; : : : ; x:key x: n , stored in nondecreas-
   ing order, so that x:key 1 x:key 2   x:key x: n ,
   c. x:leaf , a boolean value that is TRUE if x is a leaf and FALSE if x is an internal
   Node.
2. Each internal node x also contains x:n C 1 pointers x:c 1 ; x:c 2 ; : : : ; x:c x: nC1 to
   its children. Leaf nodes have no children, and so their c i attributes are unde-
   Fined.
3. The keys x:key i separate the ranges of keys stored in each subtree: if k i is any
   key stored in the subtree with root x:c i , then
   k 1 x:key 1 k 2 x:key 2   x:key x: n k x: nC1 :
4. All leaves have the same depth, which is the tree's height h.
5. Nodes have lower and upper bounds on the number of keys they can contain.
   We express these bounds in terms of a fixed integer t 2 called the minimum
   degree of the B-tree:
   a. Every node other than the root must have at least t 1 keys. Every internal
   node other than the root thus has at least t children. If the tree is nonempty,
   the root must have at least one key.
   b. Every node may contain at most 2t 1 keys. Therefore, an internal node
   may have at most 2t children. We say that a node is full if it contains exactly
   2t 1 keys. 2
   The simplest B-tree occurs when t D 2. Every internal node then has either 2,
   3, or 4 children, and we have a 2-3-4 tree. In practice, however, much larger values
   of t yield B-trees with smaller height.

3. Structure

Each node contains the following structure.

1. n : gives the number of the keys in the node.
2. x[2*t-1] : is the array which stores the records.
3. c[2*t] : is the array which stores the indices of the child nodes.
4. Leaf : is a variable to indicate a leaf.

And each of x array is a structure which contains the input records.here t is the min degree of node .
For the file implementation,the records are not brought into the main memory.the offset of the record in the file is stored as the value in the array of the B tree.this offset is used to read the disk and search.

4. Insertion

The insertion function follows a single pass algorithm.
Pre-emptive splitting of the node is used to achieve the single pass nature of the algorithm.Insertions can only take place at the leaf node of the B tree.

The split function is called on a node whose child is full.This function creates a new child and adds this child to the node.the new child will borrow values from its sibling.after the split this function passes the key to be inserted recursively to the non-full function.

5. Search

The search function searches by the key in the tree and displays the record.and prints search failed if not found.

In array implementation ,the required record is stored in the global array,this array is accessed from the array using the index in O(1) time.

In the file implementation ,the record resides in the file in the disk. This record has to be read and brought to the primary memory to be displayed. The record is accessed by using fseek() function to set the offset of the file pointer to the required position in the file,which exists as the value for the key in the node. The file access and display happens in O(1) time.

LEARNING

- More about structures, pointers.
- using files etc

CONCLUSION

B trees are balanced search tree structures the are used to store large quantities of data in an efficient manner.B trees are mostly used with disk storage devices.

B trees are used to access multiple records at the same time,which usually corresponds to the size of a page in the file system. This leads to optimality of the utilization of space and time. All records do not have to be in the primary memory.

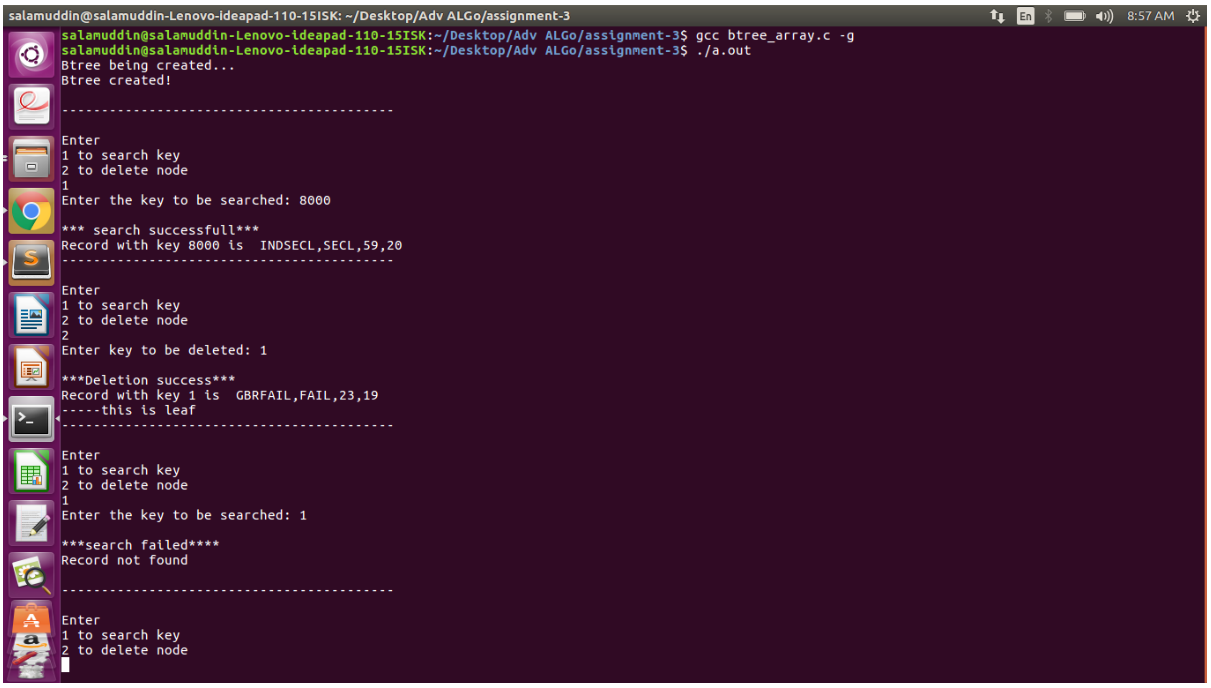
B tree implementation and a million records are stored into the B tree.
The records are stored in an array(main memory) in the first implementation.
The records are stored in the secondary memory in the file implementation.

REFERENCES

Introduction to algorithms - CLRS

www.cs.usfca.edu - B trees visualization

ILLUSTRATION


salamuddin@salamuddin-Lenovo-ideapad-110-15ISK: ~/Desktop/Adv ALGo/assignment-3
En
8:57 AM

```
salamuddin@salamuddin-Lenovo-ideapad-110-15ISK:~/Desktop/Adv ALGo/assignment-3$ gcc btree_array.c -g
salamuddin@salamuddin-Lenovo-ideapad-110-15ISK:~/Desktop/Adv ALGo/assignment-3$ ./a.out
Btree being created...
Btree created!

-----------------------------------------

Enter
1 to search key
2 to delete node
1
Enter the key to be searched: 8000

*** search successfull***
Record with key 8000 is  INDSECL,SECL,59,20
-----------------------------------------

Enter
1 to search key
2 to delete node
2
Enter key to be deleted: 1

***Deletion success***
Record with key 1 is  GBRFAIL,FAIL,23,19
-----this is leaf
-----------------------------------------

Enter
1 to search key
2 to delete node
1
Enter the key to be searched: 1

***search failed****
Record not found

-----------------------------------------

Enter
1 to search key
2 to delete node
```