**Project 2: B+Tree and Join Based on Hashing**

**Name: Nimoshika Jayaraman**

**UIN: 631000852**

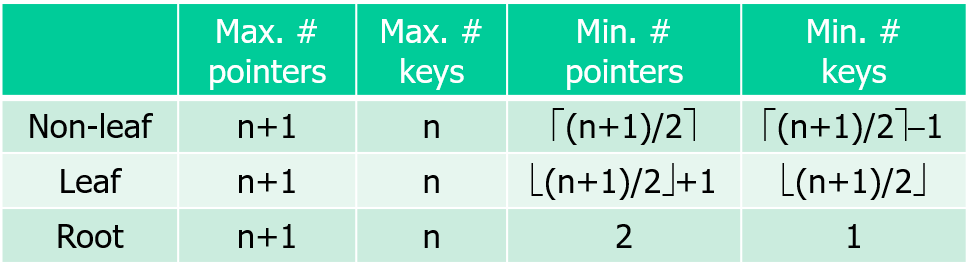
**Project Overview:**

In this project I have implemented the key algorithms B+trees and Hash-based algorithm which are widely used in Database management systems. This includes implementation of B+trees and implementation of hash based algorithms for relational algebraic operations. These algorithms play a major role in consuming less time and memory.

**B+Trees:**

A B+tree is constructed with few rules in mind.

* All leaves are at same lowest level (balanced tree).
* Pointers in leaves point to records except for “sequence pointer”
* Number of keys/pointers in nodes are given by rule



**Implementation:**

A B+tree is implemented to support some of the basic operations like search, delete, range search and insertion operations. In addition, implementation of procedure for a collection of records to construct a dense and sparse B+tree was also done.

**Data Generation:**

A search key as an integer is generated between 100000 and 20000. This generates 10000 different records which is used as a search key with no other attributes. An array of random numbers starting from the given range was generated using an increment operation until the maximum value that was mentioned. This generated output of values was used to build a dense and sparse B+tree based on the order value of the tree given.

**Dense Tree:**

Dense tree uses **pointers to all records.** Depending on the order of the tree, the keys and pointers are allocated. When we create a dense tree, we implement a pointer to all records. These pointers play a major role while inserting, searching and deletion in a record. In a dense tree, the nodes are full as much as possible. The way they divide or merge during basic operations vary from that of a sparse tree.

**Sparse Tree:**

Sparse tree uses **one pointer per block.** Depending on the order of the tree, the keys and pointers are allocated. When we create a sparse tree, we implement one pointer per block. These pointers play a major role while inserting, searching and deletion in a record. In a sparse tree, the nodes are as sparse as possible. The way they divide or merge during basic operations vary from that of a dense tree.

**Order of Tree:**

Two B+ trees are implemented. One with order 13 and one with order 24.

Operations:

**Search:**

In B+tree, the search starts from the root and goes down to the leaf block. Based on the search key provided which is rooted at the pointer the search operations happen. Based on whether the search or pointer is a leaf or non-leaf the search operation varies. If it’s in leaf, then it returns the value. If it’s a non-leaf then it find the pointer that points down and returns the value. In this implementation, as requested, the search key prints the results of the nodes that it passes through. The nodes in which the function is processing is also printed. Based on the search value, the search process continues in the left or right side. If the value is greater, it goes to right. If lesser value then it goes to the left.

**Insertion:**

In this B+tree implementation, if there is place in the leaf and the value is to be inserted, then it is inserted without violating the rules. But when it is full, it splits into two leaves. This causes an additional child for the Non leaf node or parent. If an no leaf node is full then it is split and an additional child node is added to the parent and the condition varies based on where exactly it is inserted. It could also lead to leaf or non-leaf overflow. This might also create a new root. When there is no space for a new child or when all pointers are in use, it splits the node into two nodes with both at least half full.

**Deletion:**

In this B+tree implementation, using search it reaches the pointer where the record must be deleted. If the leaf remains at least half-full after deleting the record, then the record is deleted. If not consider the sibling of the leaf. If it is more than half-full, then a record is moved from sibling to the leaf and return.If not, combine both into a single leaf. This operation varies for a dense and sparse tree. After coalescing, the pointer is deleted from its father.

**Testing:**

For the collection of 10,000 records generated the implementation is done with the operations mentioned. Randomly generated insertion/ deletion operations were performed and checked. It Prints all tree nodes involved in the operation. It also prints the nodes before and after the operation.

**Join Based on Hashing:**

It is one of the most efficient algorithms used in Database management system. Implementation is done based on two-pass join algorithm hashing. It is done based on assumption of having two relations R(A,B) and S(B,C) and their natural join operation is done using this algorithm. We have also made an assumption that each blocks holds upto 8 tuples of the relations and that we have a virtual main memory of 15 blocks and a virtual disk whose size is unlimited.

**Data Generation:**

A relation S with approx. 5000 tuples is generated where B is the key. C is taken as random not necessarily a key. B’s range is from 10,000 and 50000. The relation is stored in virtual disk. To differentiate between the values the a and b is used in the print screen.

**Virtual Disk I/O:**

Being a small-scale project implementing a virtual disk is done in a small way by specifying a unlimited size of memory as virtual and a fixed size of memory as main memory. When the operations are done, the tuples are being brought to the virtual memory as stated by the algorithm. Once done, it is stored in the fixed main memory. The operations are done in blocks as mentioned.

The number of disk I/Os involved in the operation are recorded and printed. Refer experiment section for the values.

**Join Algorithm:**

Two-pass natural join operation based hashing is done but when the operations are done in virtual memory it follows one-pass natural join algorithm.

Basic idea behind the two-pass algorithm is to break the relation into smaller pieces that fit in the main memory, make it more organized and store them back to disk. The movements between the virtual and main memory is recorded as disk I/Os. A counter is implemented to record this.

**Testing:**

A relation R(A,B) was generated in which the attributes were randomly picked from that in the relation S(B,C) with duplicates. The join is done and with the implemented counter the number of disk I/Os and size of natural join is calculated. Similarly for second experiment, the values of B not from the previous relation but the randomly picked ones are used and the disk I/Os and the size of natural joins are recorded.

**5.1 Number of I/Os = 2399, size of natural join = 1000**

**5.2 Number of I/Os = 2371, size of natural join = 159**

**Difficulties:**

Though I got the expected result and gained knowledge through this project, it would have been helpful if it were not given a day before final exam. I have another course’s exam also along with DB on the same day. It was very hectic to complete this as it was very time consuming. Please consider lenient grading.