Department of Computer science and Engineering

PES UNIVERSITY

UE19CS202: Data Structures and its Applications (4-0-0-4-4)

GRAPHS

Abstract

Case study-Indexing in databases (B-tree: K way tree)

Dr.Sandesh and Saritha

Sandesh_bj@pes.edu

Saritha.k@pes.edu

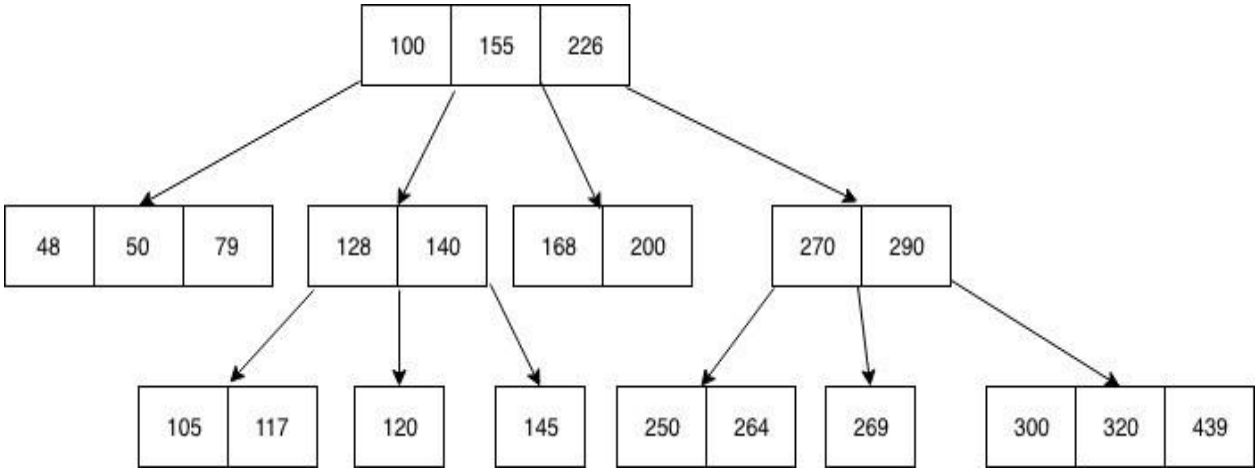
Case study-Indexing in databases (B-tree: K way tree)

Indexing:

Indexing is a technique used in data structure to access the records quickly from the database file. An Index is a small table containing two columns one for primary key and the second column contains a set of pointers for holding the address of disk block where the specific key value is stored.

What Is B-tree?

B-tree is a data structure that store data in its node in sorted order. A Simple B-tree is represented as shown below



Sample B-tree

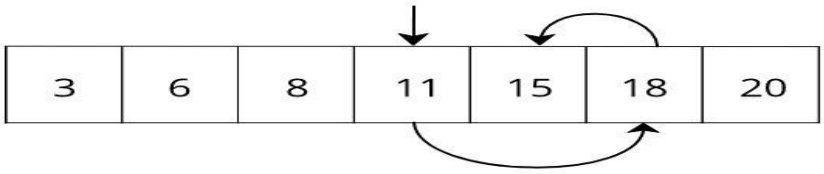
B-tree stores data such that each node contains keys in ascending order. Each of these keys has two references to another two child nodes. The left side child node keys are less than the current keys and the right side child node keys are more than the current keys. If a single node has “n” number of keys, then it can have maximum “n+1” child nodes.

Why Is Indexing Used in the Database?

Imagine a user needs to store a list of numbers in a file and search a given number on that list. The simplest solution is to store data in an array and append values when new values come. But to check if a given value exists in the array, then user need to search through the entire array elements one by one and check whether the given value exists. If the user finds the given value

in the first element then it is the best case. In the worst case, the value can be the last element in the array.

How could you improve this time? The solution is to sort the array and use binary search to find the value. Whenever you insert a value into the array, it should maintain order. Search starts by selecting a value from the middle of the array. Then compare the selected value with the search value. If the selected value is greater than search value, ignore the left side of the array and search the value on the right side and vice versa



Binary search

Here, we search key 15 from the array 3,6,8,11,15, and 18, which is already in sorted order. If you do a normal search, then it will take five units of time to search since the element is in the fifth position. But in the binary search, it will take only three searches.

Properties of B-tree:

- All the leaves created are at same level.
- B-Tree is determined by a number of degree m. The value of m depends upon the block size on the disk .
- The left subtree of the node has lesser values than the right side of the subtree.
- The maximum number of child nodes(a root node as well as its child nodes can contain) is calculated by :m – 1
- Every node except the root node must contain minimum keys of[m/2]-1
- The maximum number of child nodes a node can have is equal to its degree that is m.
- The minimum children a node can have is half of the order that is m/2

Why use B-Tree

- B-tree reduces the number of reads made on the disk
- B Trees can be easily optimized to adjust its size according to the disk size
- It is a specially designed technique for handling a bulky amount of data.
- It is a useful algorithm for databases and file systems.
- B-tree is efficient for reading and writing from large blocks of data

Insertion

- B-tree insertion takes place at leaf node.
- Locate the leaf node for the data being inserted.
- If the node is not full that is fewer than m-1 entries, the new data is simply inserted in the sequence of node.
- When the leaf node is full, we say overflow condition.
- Overflow requires that the leaf node be split into 2 nodes, each containing half of the data.
- To split the node, create a new node and copy the data from the end of the full node to the new node.
- After the data has been split, the new entry is inserted into either the original or the new node depending on its key value.
- Then the median data entry is inserted into parent node.

Insertion Algorithm

1. Traverse the B Tree in order to find the appropriate leaf node at which the node can be inserted.

2. If the leaf node contain less than m-1 keys then insert the element in the increasing order.

3. Else, if the leaf node contains m-1 keys, then follow the following steps.

a) Insert the new element in the increasing order of elements.

b) Split the node into the two nodes at the median.

c) Push the median element upto its parent node.

d) If the parent node also contain m-1 number of keys, then split it too by following the same steps.

B-tree Deletion

- When deleting a node from B-tree , there are three considerations
- 1. data to be deleted are actually present in the tree.
- 2. if the node does not have enough entries after the deletion, then correct the structural deficiency.
- A deletion that results in a node with fewer than minimum number of entries is an underflow
- 3. Deletion should takes place only from leaf node.
- If the data to be deleted are in internal node, find a data entry to take their place.

Deletion Algorithm

1.Locate the leaf node.

2.If there are more than m/2 keys in the leaf node then delete key

3.If the leaf node doesn't contain m/2 keys then

a)If the left sibling contains more than m/2 elements then push its largest element up to its parent and move the intervening element down to the node where the key is deleted.

b)If the right sibling contains more than m/2 elements then push its smallest element up to the parent and move intervening element down to the node where the key is deleted.

4.If neither of the sibling contain more than m/2 elements then create a new leaf node by joining two leaf nodes and the intervening element of the parent node.

5.If parent is left with less than m/2 nodes then, apply the above process on the parent too.