BTrees and B+Trees

September 28, 2023

Generalised Search Tree

Each node has format:

$$P_1, K_1, P_2, K_2, ...P_{n1}, K_{n1}, P_n$$

where P_i are tree values and K_i are search values.
In a database context, a node corresponds to a disk block.
Hence, the number of values per node depends on the size of the key field, block size and block pointer size.

Constraints

The following constraints hold:

- $K_1 < K_2 < ... < K_{n1} < K_n$
- For all values x in a subtree pointed to P_i, $K_{i1} < x < K_i$
- The number of tree pointers per node is known as the order $/rho$ of the tree

Efficiency

- For a generalised search tree:

$$T(N) = 1 + T(N/) = ... = O(log(N))$$

- This assumes a balanced tree
- In order to guarantee this efficiency in searching and in other operations, we need techniques to ensure a balanced tree

B Trees

- A B Tree is a balanced generalised search tree
- Can be viewed as a dynamic multi-level index
- The properties of a search tree still hold.
- The algorithms for insertion and deletion of values are modified in order

B Trees: Node structure

- The node structure contains a record pointer for each key value.
- Node structure is as follows:

$$P_1, < K_1, Pr_1 > P_2, < K_2, Pr_2 >, ...P_{n1}, < K_{n1}, Pr_{n1} >, P_n$$

Example

- Consider a B tree of order 3 (two values and 3 tree pointers per node/block).
- Insert records with key values: 10, 6, 8, 14, 4, 16, 19, 11, 21

Algorithm to insert value into B Tree

1. Find appropriate leaf level node to insert value
2. If space remains in leaf-level node, then insert the new value in correct location.
3. If no space remains, we need to deal with collision.

Dealing with collisions

1. split node into left and right nodes
2. propagate middle value up a level and place value in node there (*)
3. place values less than middle value in the left node
4. place values greater than the middle value in the right node

*

Note: this propagation may cause further propagations and even the creation of a new root node

- This maintains the balanced nature of the tree.
- $\Rightarrow O(log_{\rho}(N))$ for search, insertion and deletion
- However, there is always potential for unused space in the tree.
- Note: Empirical analysis has shown that B-trees remain 69% full given random insertions and deletions.

Exercises

- Can you define an algorithm for deletion (at a high level)?
- How much work is needed in the various cases (best, average, worst)?

B+ tree

- The most commonly used index type is the B+-tree - a dynamic, multi-level index.
- Differs from a B Tree in terms of structure.
- Insertion and deletion algorithms slightly more complicated.
- Offers increased efficiency over B Tree. Ensures a higher order ρ.
- Two different node structures in B+ Trees:
  - internal nodes
  - leaf-level nodes

Node structure

- All record pointers are maintained at the leaf level in a B+tree.
- internal node structure
- $$P_1, K_1, P_2, K_2, \ldots P_{n1}, K_{n1}, P_n$$
- No record pointers.
- Less information per record; hence more search values per node

Node structure

- leaf level node structure
- One tree pointer is maintained at each leaf-level node. This points to the next leaf-level node.
- Each node's structure $K_1, Pr_1, K_2, Pr_2, ...K_m, Pr_m, P_{next}$
- The P_{next} pointer facilitates range queries.
- Note only one tree pointer per node at the leaf-level.

Example

- Let $B = 512, P = 6, Pr = 7, K = 10$
- Assume 30000 records as before.
- Assume tree is 69% full.
- How many blocks will the tree require? How many block accesses will a search require?

Example

- A tree of order ρ has at most $\rho - 1$ search values per node
- For a B+ Tree, there are two types of tree nodes; hence there are 2 different orders ρ and ρ_{leaf}
- To calculate ρ:
- $$\rho(|P|) + (\rho - 1)(|K|) \leq B$$
- $$\Rightarrow 16\rho < 522$$
- $$\Rightarrow \rho = 32$$

- To calculate ρ_{leaf}
- $$|P| + (\rho_{leaf})(|K| + |Pr|) \leq B$$
- $$\Rightarrow 17(\rho_{leaf}) \leq 506$$
- $$\rho_{leaf} = 29$$

Given fill factor = 69%:

- Each internal node will have, on average, 22 pointers
- Each leaf level node will have, on average, 20 pointers

- Root: 1 node 21 entries 22 pointers
- level1: 22 nodes 462 entries 484 pointers
- level2: 484 nodes .. etc.
- leaf level: ....

- Hence, 4 levels are sufficient
- Number of block accesses = 4 + 1
- Number of block 1 + 22 + 484 + ..

Recap

- Looked at structure of a BTree;
- Looked at insertion algorithm for BTrees.
- Introduced a B+Tree and looked at some calculations in order to illustrate how to work out the required size and number of accesses.