Algorithms and Data Structures

Chapter 18

Catherine Durso

cdurso@cs.du.edu

Chapter 18

Our model to this point in course has essentially assumed that all constant time operations are equivalent. This is a reasonable model for computation with data in RAM. When a program requires disk reads, we should model that fact that these operations can be much slower than accessing data in main memory, by a factor of more than 10^5. B-trees are designed to take advantage of the fact that a single disk read accesses a page of data ($2^{11} - 2^{14}$ bytes).

B-trees typically have a large branching factor at each node, allowing location of the data after few disk reads of nodes.

B-tree Node Properties

A B-tree is as rooted tree in which every node has the following:

1. $x.n$: the number of keys in the nodes
2. $x.n$ keys, $x.key1 \leq x.key2 \leq ...x.keyn$

Every internal node has $x.n + 1$ child pointers, $x.c_1$, $x.c_2$,..$x.c_{n+1}$.
In this implementation, pointers to satellite data are stored with the keys.

Alternatively, the tree may store satellite data only in the leaves, with the keys repeated in the internal nodes for navigation.

Search Property

The keys bound the contents of the subtrees rooted at the children. If k_i is any key stored in the subtree rooted at the child $x.c_i$, then
$k_1 \leq x.key1 \leq k_2 \leq x.key2 \leq ...k_n \leq x.keyn \leq k_{n+1}$.

Balance Conditions

Every leaf has the same depth, the tree's height h.

There is a fixed integer t, the ***minimum degree*** of the B-tree. Every node other than the root must have at least $t - 1$ keys. Thus internal nodes have at least t children.

Every node contains at most $2t - 1$ keys. Thus internal nodes have at most $2t$ children. A node is ***full*** if it has $2t - 1$ keys.

Height

If $N \geq 1$ then any $N-$key B-tree of height h and minimum degree $t \geq 2$ satisfies $h \leq \log_t \left(\frac{n+1}{2}\right)$.

First, note lower bounds on the numbers of keys at each level: the root has at least one key, hence two children. The next level has at least two nodes with at least $t-1$ keys each. The next level has at least $2t$ nodes with at least with at least $t-1$ keys each. In general, the k^{th} level has at least $2t^{k-1}(t-1)$ keys.

Proof, cont.

$$N \geq 1 + 2\,(t-1) \sum_{k=1}^{h} t^{k-1}$$
$$= 1 + 2\,(t-1) \left(\frac{t^h - 1}{t-1} \right)$$
$$= 2t^h - 1$$

so
$\frac{N+1}{2} \geq t^h$
and $h \leq \log_t \left(\frac{N+1}{2} \right)$, as required.

Impact

Typically, t is large, so h grows very slowly.

Ex. $t = 500$, $N = 500,000 - 1$:
$\frac{N+1}{2} = 250,000 = t^2$, so $h \leq \log_{500} 250,000 = 2$

Search

Search generalizes the binary search algorithm. If the query key k is not found at an internal node, descend to the child that could contain the node according to the rule
$k_1 \leq x.key1 \leq k_2 \leq x.key2 \leq ...k_n \leq x.keyn \leq k_{n+1}$.

Balance Operations

Inserting into the tree may require splitting a full node that is a child of a non-full node into two nodes.

Deletion may require modifying a node with exactly $t - 1$ keys. To add a key to a node with $t - 1$ keys, the node may be assigned a sibling's key if a sibling has a spare key. Sibling nodes with $t - 1$ keys may be merged with a key from the parent to make a node with $2t - 1$ keys

Insertion

Insertion into B-tree descends the B-tree search path, splitting nodes as necessary to avoid descending to a full node. The data is inserted in the appropriate leaf.

Deletion

Deletion occurs only from leaves.

Deletion of a value z from a B-tree descends the search path for the key $z.k$, using adoption and merging to prevent descending to a node with $t - 1$ keys. If the $z.k$ is found in an internal node, it is replaced by its successor k' from its child, and k' is recursively deleted from the child.