Lecture Notes: B-Trees

Miao Qiao
Computer Science Department
University of Auckland
miao.qiao@auckland.ac.nz

Sorting arranges data elements in a way that searching can be efficiently done. It can be easily proved that under comparison model, searching one element in a set of N elements takes $\Omega(\log_2 N)$ comparisons in the worst case (think about why). However, datasets in real world are not static. Obviously, to resort the dataset upon each element update — element insertion or element deletion — is not economical. Such a requirement naturally derives the formulation of Problem 1 – the indexing problem.

Problem 1. *Create a data structure in maintaining the sorted order of N data elements. Examine:*

- *Space complexity: the total number of storage unit occupied by the structure,*
- *Query time: given a range query $[q_1, q_2]$, the worst-case time complexity in reporting all the data elements that fall in range $[q_1, q_2]$, and*
- *Update time: the worst-case time complexity in element insertion / deletion while maintaining the order of the dataset.*

Problem 1 triggered the invention of a large number of data structures in which balanced binary search trees play an important role [1]. This lecture first introduce a balanced binary search tree called (a,b)-Tree in RAM model, and then demonstrate B-trees and Weight Balanced B-Trees in EM model.

1 (a,b)-Tree

Definition 1. *A tree $\mathcal{T}$ is an (a,b)-tree if the following conditions hold:*

1. *All leaves of $\mathcal{T}$ are at the same level; all data elements are in the leaves of $\mathcal{T}$; each leaf contains a to b data elements.*
2. *Each internal node (except for the root) has a to b children separated by (a-1) to (b-1) routing elements.*
3. *The root can either be an internal node with 2 to b children or a leaf with 1 to b data elements.*

Normally, the N data elements are stored in sorted order in leaves of $\mathcal{T}$; the elements in the internal nodes of $\mathcal{T}$ are only used to guide searches. In this way, $\mathcal{T}$ uses linear space and has a height of $O(\log_a N)$.

Query. To answer a range query $[q_1, q_2]$, we first search the leaves of $\mathcal{T}$ for q_1 and q_2, respectively, and then report all data elements in leaves in between the two leaves.

Insertion. To insert an element x in an (a,b)-tree $\mathcal{T}$,

1. We first search down $\mathcal{T}$ for the relevant leaf u and insert x in u;
2. If u now contains $b+1$ elements, we split it into two leaves u' and u'', containing $\lfloor \frac{b+1}{2} \rfloor$ and $\lceil \frac{b+1}{2} \rceil$ elements, respectively; then we remove the reference to u in $parent(u)$ and insert references to u' and u'' instead — we also insert a new routing element in $parent(u)$;
3. If $parent(u)$ now has $b+1$ children, we recursively split it.

The split may propagate up through $O(\log_a N)$ nodes with a new root produced who has two children.

Deletion. To delete an element x from an (a,b)-tree $\mathcal{T}$,

1. We first find and remove x from the relevant leaf u;
2. If u now contains less than a elements, we *fuse* it with any of its siblings u', that is, we delete u' and insert its elements in u;
3. If this results u containing more than b elements we split it into two leaves; as before, we update $parent(u)$ appropriately;

The fuse operation may propagate up through $O(\log_a N)$ nodes which may reduce the height of the tree by one.

Generally, the two parameters, integers a and b, must satisfy the following conditions:

1. $2 \leq a < b$ — to ensure that the height of the tree is sublinear.
2. $a \leq \lfloor \frac{b+1}{2} \rfloor$ — to ensure that each split produces two legal nodes.

2 B-tree

A B-tree is an (a,b)-tree with a, b both $\Theta(B)$. Assume that each internal node takes one block to store. Let b be the maximum number of children an internal node can have. Let a be $\lfloor \frac{b}{4} \rfloor$. Let k be the maximum number of data elements that can be stored in a block.

Definition 2. *A tree $\mathcal{T}$ is a B-tree with branching parameter b and leaf parameter k if the following conditions hold:*

1. *All leaves of $\mathcal{T}$ are at the same level; all data elements are in the leaves of $\mathcal{T}$; each leaf contains $\frac{k}{4}$ to k data elements;*
2. *Each internal node (except for the root) has $\frac{b}{4}$ to b children;*
3. *The root can either be an internal node with 2 to b children or a leaf with 1 to k data elements.*

Theorem 1. *An N-element B-tree $\mathcal{T}$ with branching parameter $b > B^c$ for a constant $c \in (0,1)$ and leaf parameter $k = \Omega(B)$*

- *uses $O(N/B)$ space,*
- *supports an update in $O(\log_{B^c} \frac{N}{B}) = O(\log_B \frac{N}{B})$ I/Os and*
- *supports a range query in $O(\log_B \frac{N}{B} + \frac{|\#\ of\ resulting\ elements|}{B})$ I/Os.*

Theorem 2. *Assume that $\Omega(B)$ elements can fit in one block. Constructing a B-tree on N elements costs $\Theta(\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B})$ I/Os.*

3 Buffered Tree

The complexity of B-tree can be further improved by dropping an assumption of Problem 1 — this leads to the definition of batched dynamic problem and a new data structure called buffered tree.

Problem 2 (Batched Dynamic Problem)**.** *Given a sequence of updates and queries, create a data structure in maintaining the sorted order of N data elements in supporting the range queries and updates (insertions and deletions). The queries must be eventually answered.*

Unlike online problems where queries must be answered immediately, batched dynamic problems may have a long query delay. On the other hand, a batched dynamic problem is also different from an offline problem such as sorting since queries are posed on different "snapshots" of the dataset — each data element has two timestamps to represent their "lifespan".

Since the height of a B-Tree is related to the update cost and query cost, a buffered tree was designed to reduce the height of the tree from $\log_B N$ to $\log_{\frac{M}{B}} \frac{N}{B}$ by increasing the fanout of each node from $O(B)$ to $O(\frac{M}{B})$. However, in increasing the fanout of a tree node, loading a node becomes rather clumsy. In reducing the access of a tree node while keeping the dataset updated when a query comes, a buffered tree, as the name suggests, equips each B-Tree node with a buffer such that updates are logged and reflected in the dataset in a lazy manner. The buffer size of a node v is set to $O(M)$ such that all the updates can be redistributed to the children of v in a memory load and $O(\frac{M}{B})$ I/Os (now think about why the fanout of a node is set to $O(M/B)$).

Definition 3. *A basic buffer tree $\mathcal{T}$ is a B-tree with branching parameter $\frac{M}{B}$ and leaf parameter B where each internal node has a buffer of size M.*

Theorem 3. *The total cost of a sequence of N update operation on an initially empty buffer tree is $O(\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B})$.*

For details of query, insertion and deletion, please refer to [1].

4 Weight-balanced B-trees

In a B-tree of height $h(\mathcal{T})$, each internal node has a fanout of $\frac{1}{4}b$ to b while a leaf has $\frac{1}{4}k$ to k data elements. Therefore, an internal node may have $4^{h(\mathcal{T})-1}$ times more data elements than another internal node in the same level. To avoid this exponential gap, we introduce a technique called weight balancing. Each node v in the tree has a weight $w(v)$ which is defined as the total number of data elements stored in leaves of the subtree rooted at v. This technique is especially useful when there are secondary structures associated with each node — reconstructions on heavy nodes must be strictly restricted to ensure the overall complexity while light nodes have large freedom in reconstruction.

Definition 4. *A tree $\mathcal{T}$ is a weight balanced B-tree with branching parameter b and leaf parameter k, $b, k \geq 8$, if the following conditions hold:*

- *All leaves of $\mathcal{T}$ are on the same level (level 0) and each leaf u has weight $\frac{1}{4}k \leq w(u) \leq k$.*
- *Except for the root, each internal node v on level l has weight $w(v) \in [\frac{1}{4}b^l k, b^l k]$.*
- *The root has a weight no greater than $b^h k$ where h is the level of the root.*

Query. The same as B-Tree.

Insertion. It is interesting to consider the following questions on Weighted Balanced B-Tree (WBB-tree) $\mathcal{T}$ with leaf parameter k and branching parameter b ($k, b \geq 8$) and then progressively develop the solution for WBB-Tree:

1. In inserting an element x into $\mathcal{T}$, we first find the leaf u and insert x into u. Obviously, if $w(u) = k+1$, we split u into two leaves such that all leaves are balanced. Consider the parent p of u. If $w(p) = kb+1$ after the insertion, how to rebalance the node p?
2. Let v be an internal node of $\mathcal{T}$. If all subtrees rooted at the children of v are WBB-Trees while $w(v) = kb^l + 1$ with l denoting the level of v, how to rebalance the subtree rooted at v?

We leave the deletion to the readers who have interest.

References

[1] L. Arge. External memory data structures. In *ESA*, pages 1–29, 2001.