2-3 Trees and B-Trees

1 Recap

1.1 Balanced Binary Search Trees

Binary Search Trees that guarantee $O(\log(n))$ height by rebalancing after Insert/Delete operations.

1.2 Example Insertion and Rotation

![Insertion and Rotation Diagram]

2 2-3 Trees

2.1 Properties

2-3 Trees are balanced search trees. Every node with children (non-leaf) has either two children (2-node) and consists of one piece of data, or has three children (3-node) and consists of 2 pieces of data.

- Every non-leaf is a 2-node or 3-node
- All leaves are at the same level
Recitation 2: 2-3 Trees and B-Trees

- All non-leaves branch
- All data is kept in sorted order
- Every leaf node will contain one or two fields
- Height $\leq \log n$ (More dense than BST)

2.2 Example 2-3 Tree

![2-3 Tree Diagram]

2.3 Search

Very similar to Binary Search. Start at the root node, and traverse down tree in order. Tree is sorted so only need to look at one node for each level of tree. Because of this Search runtime is $O(\log(n))$
2.4 Insert

Insert(X) Steps:

- Search for element X for where it would go in a leaf of the tree.
- Insert element X into where it would go.
- While there is overflow, a node has more than 3 elements, Split node into left half, median, and right half Then promote median up a level. If there is a parent, add it to the node. If there is no parent, create a new node of just the Median node as the new root.
- Runtime is $O(lg(n))$
2.5 Delete

Delete(X) Steps:

- Swap item to delete with inorder successor if item is not already a leaf.

- Redistribute and merge nodes if there is underflowing in order to get back to a correct 2-3 tree

- Runtime is $O(lg(n))$
2.5.1 Redistribute Example

Delete 38

Delete 38

Delete 48

Delete 48
2.5.2 Merge Example

3 B-Trees

One parameter B. Trees have between $\lceil \frac{B}{2} \rceil$ and B children. Let C denote number of children. Each node will have C-1 keys. As with 2-3 Trees, each node has one more child than the number of elements in that node.

- Search time is $O(lg(n))$
- Insert/Delete time is $O(lg(n))$ if $B = O(1)$

Why B-Trees

- Caches read whole blocks of data, and want entire block useful
- Set parameter B equal to block size
- $O(log_b(n))$ block reads per Search, Insert, Delete operations.

B-Trees are used by most databases and filesystems:
- Databases: Sleepycat/BerkeleyDB, MySQL, SQLite
- Filesystems: MacOS HFS/HFS+, ReiserFS, Windows NTFS, Linux ext3, shmfs