Principles of Data Management

Alvin Lin

August 2018 - December 2018

Indexes

Indexes serve as search keys for content. There are ordered indices and hashed indices, and since hashed indices function similar to a hash table, we will simply cover ordered indices.

Evaluation Metrics

- Access types supported efficiently (records with attributes, range of values)
- Access time
- Insertion time
- Deletion time
- Space overhead

Types of Indices

With an ordered index, search keys are sorted. With a **primary index**, records are sorted in sequential order of the file. This is also called a clustering index and is specified by the primary key. With a **secondary index**, an order is specified other than that of the primary. This is also known as a non-clustering index. With **dense index files**, there exists an index entry for every search key. In contrast, a **sparse index file** only has index entries for some of the values in the table.

A secondary index will typically point to a primary index which has pointers to values in the table. Thus, it must be a dense index because the values are not

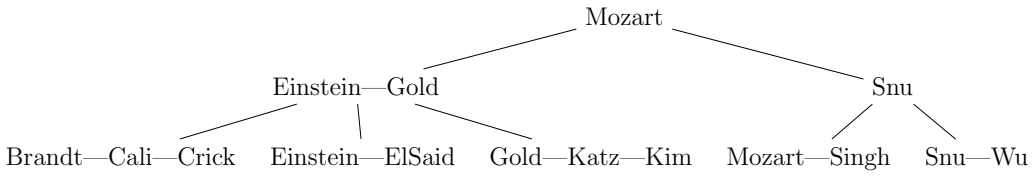
ordered. There is some overhead on insertions and deletions since both primary and secondary indices must both be updated.

Disks are slow, so what often happens in a database is that multilevel indices are used. There is data stored in blocks on disk that we need to read, for which access time is slow. An inner index will store information about different blocks on disk. There may be one or more inner indices. A single outer index will point to the inner indices to determine where data should be accessed. The outer index would be stored in memory while the inner indices are stored on disk like the data.

Indices managed on linear files generally degrade in performance as the index grows due to the constant reordering needed as data is inserted. B+ trees support local reorganization and have a fixed height and are much better for index organization.

B+ Trees

B+ trees have extra delection time, insertion time, and space overhead. Generally for databases though, the advantages of a B+ tree far outweigh the disadvantages.



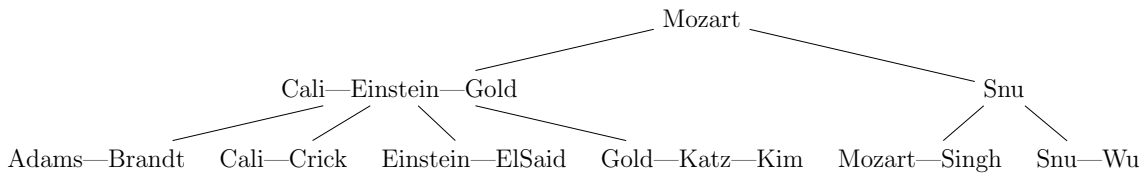
Each of the elements in the leaf nodes point to sections of the data. They also contain linear pointers to the next leaf node on the same level. The height of the B+ tree is fixed. All paths from root to leaf are the same length.

- Each internal node has $\lceil \frac{n}{2} \rceil$ to n children.
- Each leaf node has $\lceil \frac{n-1}{2} \rceil$ to $n-1$ values.
- The root, if it is not a leaf, has at least 2 children.
- The root, if it is a leaf, has 0 to $n-1$ values.

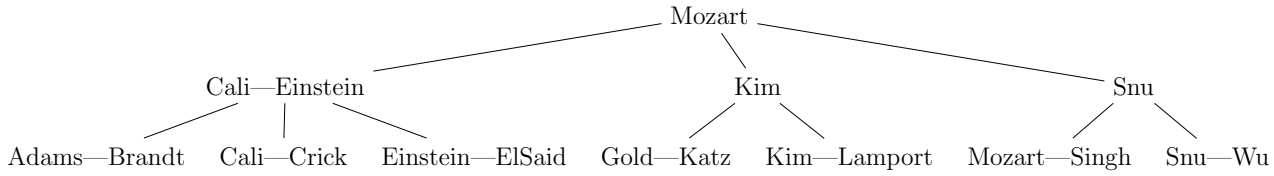
Given k search keys, the tree height is $\lceil \log_{\lceil \frac{n}{2} \rceil}(k) \rceil$.

Node Insertion

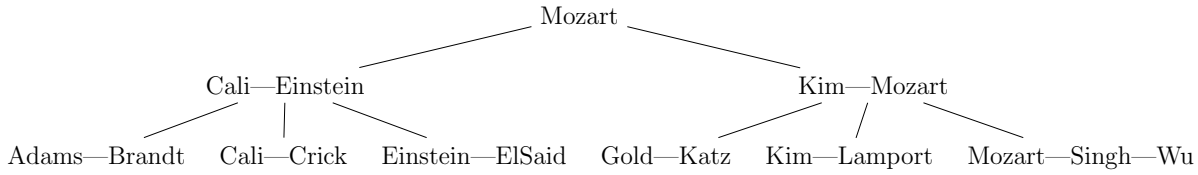
An insertion of the value `Adams` adds a new leaf node and propagates up the tree.



An insertion of the value `Lamport` adds a new leaf node, which adds a new internal node.



Deleting `Snu` from the rightmost leaf causes the leaf node to be merged left. This makes the right internal node invalid, so it will merge left as well.



You can find all my notes at `http://omgimanerd.tech/notes`. If you have any questions, comments, or concerns, please contact me at alvin@omgimanerd.tech