B- Trees

Index: provides some information about the location of records with indicated key values.

B-tree: most important index orginization

↳ extends the idea of a 2-3 tree by permitting more than a single key in the same node of a search tree.

- All data records are stored at the leaves, in increasing order of the leaves.
- Parental nodes used for indexing.

In addition, a B-tree of order $m \geq 2$ must satisfy the following structural properties:

- The root is either a leaf or has between 2 and m children.
- Each node, except for the root and the leaves, has between $\lceil m/2 \rceil$ and m children (and hence between $\lceil m/2 \rceil - 1$ and $m - 1$ keys).
- The tree is (perfectly) balanced, i.e., all its leaves are at the same level.

- Searching similar to binary search tree.
- nodes of a B-tree normally correspond to the disk pages.

↳ # of disk accesses is the indicator of efficiency.

The most straightforward algorithm for inserting a new record into a B-tree is quite similar to the algorithm for insertion into a 2-3 tree outlined in Section 6.3. First, we apply the search procedure to the new record's key K to find the appropriate leaf for the new record. If there is room for the record in that leaf, we place it there (in an appropriate position so that the keys remain sorted) and we are done. If there is no room for the record, the leaf is split in half by sending the second half of the records to a new node. After that, the smallest key K' in the new node and the pointer to it are inserted into the old leaf's parent (immediately after the key and pointer to the old leaf). This recursive procedure may percolate up to the tree's root. If the root is already full too, a new root is created with the two halves of the old root's keys split between two children of the new root. As an example, Figure 7.9 shows the result of inserting 65 into the B-tree in Figure 7.8 under the restriction that the leaves cannot contain more than three items.

Order is max number of children per root.