CSE-4411

Tree-based Indexes

Okay, there are plenty of data-structures for balanced search trees: e.g., AVL trees, red-black trees, skew trees.
So what is the difficulty? Let's just use one.

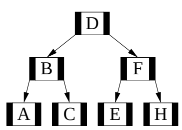
Difficulties:

- Tree may be too big for main memory.
  - Need to be careful about #I/O's for *searches*, *inserts*, and *deletes*.
  - Must group information by page.
- Tree may be too deep to be efficient.

Can any techniques we know be adapted?

Balanced Binary Trees

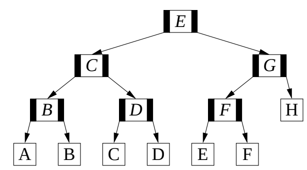
Consider we have N records to store.



What is the *height* (depth) H of the shallowest possible, balanced binary tree, if we keep data / records in leaf *and* non-leaf nodes?

$$\lceil \log_2(N+1) \rceil$$

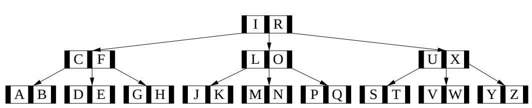
Balanced Binary Trees



What is the *height* (depth) H of the shallowest possible, balanced binary tree, if we keep data / records *only* in leaf nodes?

$$\lceil \log_2 N \rceil + 1$$

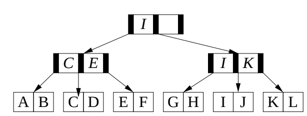
Balanced K-ary Trees



What is the *height* (depth) H of the shallowest possible, balanced K-ary tree, if we keep data / records in leaf *and* non-leaf nodes?

$$\lceil \log_K(N+1) \rceil$$

Balanced K-ary Trees



What is the *height* (depth) H of the shallowest possible, balanced binary tree, if we keep data / records *only* in leaf nodes?

$$\lceil \log_K N \rceil + 1$$

K-ary Trees!

- If K is large, the height of the tree will be small! We shall rename K as F—as the textbook uses—for *fan-out*.
  - Make a *node* the size of a page.
- So does F really save us anything?
  - Not on #comparisons, but *yes* on #I/O's:
    - $(\log_2 F)(\log_F N/R) = \log_2 N/R$ comparisons still,
    - but just $\log_F N/R$ I/O's.

    ($N =$ #records in all, $R =$ #records per page.)
- Design choice: Okay, should we keep records in non-leaves or not?
- How do we keep an F-ary tree balanced on *inserts* and *deletes*?

B Trees & B+ Trees

- The **B tree** and, subsequently, the **B+ tree** models are K-ary tree models with a particular balance rule.
  - For B trees, records (or data entries) are kept in non-leaf nodes *and* leaf nodes.
  - For B+ trees, records (or data entries) are kept just in leaf nodes (*data pages*). Non-leaf nodes are called *index pages*.
- We *trade off* space for the balance strategy: Each node (equal to a page) can be up to half empty.
- All leaves are at the same depth.
  - Tree grows when a leaf page is split, and when the index pages up to the root must consequently be split.

B Trees versus B+ Trees

Advantages of B trees:

- May use fewer tree nodes than a corresponding B+ tree.
- Sometimes possible to find the search-key value *before* reaching a leaf node.

Advantages of B+ trees:

- Only a *small* fraction of searches are found *before* a leaf page.
- Non-leaf pages can contain more index keys than the B tree's non-leaf page can contain records (or entries), so the fan-out (F) is more.
- Insertion and deletion is easier.
- implementation is easier.