CS/COE 1501

www.cs.pitt.edu/~nlf4/cs1501/

B-trees

The problem

- We’ve discussed several approaches to search through a set of keys and retrieve a value
    - Several implementations of a symbol table
- All of them assumed we were storing the keys/values (the symbol table) in memory
- What if data needs to be stored on disk?
    - What should we do differently?

Consider the following example

- You’re writing software that will be used to store records of online store transactions, each with a unique ID
  - E.g., vinyl album sales
- You’ll want to store these transaction records on disk
  - You expect a large volume of transaction records
  - You want the transaction records stored in non-volatile memory
- How can you still efficiently search for a given transaction by its ID?

Disk storage

- Data stored on disk is grouped into *blocks*
  - Typically of size 4KB
- I/O to the disk is performed at the block level
- To read a file from disk, the OS will fetch all of the blocks that store some portion of that file, and read the data from each block

B-trees

- Operates similarly to a binary search tree, but not limited to a branching factor of 2
- The order of a B-tree determines the max branching factor
  - Invariants for an order M B-tree:
    - Nodes have a max of M children
    - Interior nodes have at min of $\lceil M/2 \rceil$ children
      - Nodes that are not the root or leaves
      - Corollary: all interior nodes must be at least half full
    - Root has at least two children if it is not a leaf node
    - Non-leaf nodes with k children have k-1 keys stored
    - All leaves appear on the same level

Inserting into a B-tree

- Start with a single node
- Add keys until the node fills
  - I.e., contains M-1 keys, has M children
- In adding the Mth key, split the node in two
  - Pull one key up to the parent node
    - Potentially creating a new parent node

OK, so how does this help us store transaction records?

- See how to store IDs as keys, but what about full records of a sale transaction
  - ID, customer info, price, item purchased, how many purchased, etc.

B Tree analysis

- Runtime
    - Search?
    - Insert?

- To maintain invariants, tree must be *self-balancing*

Deleting from a B-tree

- Find and delete the key
  - If the key is not in a leaf node, you need to find a replacement…
- Rebalance the tree
  - Is there a sibling node with more than minimum keys?
    - If so *rotate* right/left accordingly
  - If not, need to *merge* with the left or right sibling

Wait, what does this have to do with disks??

What if we want to read all records?

- How long will it take us to find all the disk blocks containing records?
- Is there a better way?

B+trees

- Maintain a copy of all keys in the leaves of the tree
- Create a linked-list out of the leaf nodes of the tree

B-/+tree discrepancies

- Defining order
  - Here M is the max number of children
  - Elsewhere, could be the min number of keys
    - Min was the original notation, but is ambiguous
- Where to go to follow = keys
  - Some implementations have left link point to keys <=, and right point to keys strictly >
  - Others have left point to keys strictly <, and right point to keys >=

Note:

- The variant of B-trees presented here differs slightly from that presented in the book
- B+trees are not discussed in the book

Realistic application of this solution

- Typically, you'll store such records in a database
  - But how does the database store records?
    - IBM DB2, Informix, Microsoft SQL Server, Oracle 8, Sybase ASE, and SQLite all use B+trees to store tables indexes
- Other applications?
  - NTFS, ReiserFS, NSS, XFS, JFS, ReFS, and BFS all use B+trees for metadata indexing