ECS 165B: Database System Implementation
Lecture 10

UC Davis
April 19, 2010

Acknowledgements: portions based on slides by Raghu Ramakrishnan and Johannes Gehrke.

Class Agenda

- Last time:
  - A taste of database theory, Part 2: containment, equivalence, and minimization of conjunctive queries
- Today:
  - Overview of DavisDB project, Part 2: indexing
  - Short lecture
- Reading
  - Chapter 10 of Ramakrishnan and Gehrke (or Chapter 12 of Silberschatz et al)

Announcements

Thanks for your hard work on Part 1!!! Hopefully, it will get easier from here.

Part 2 of project out tonight, due Sunday, 5/2 @ 11:59pm

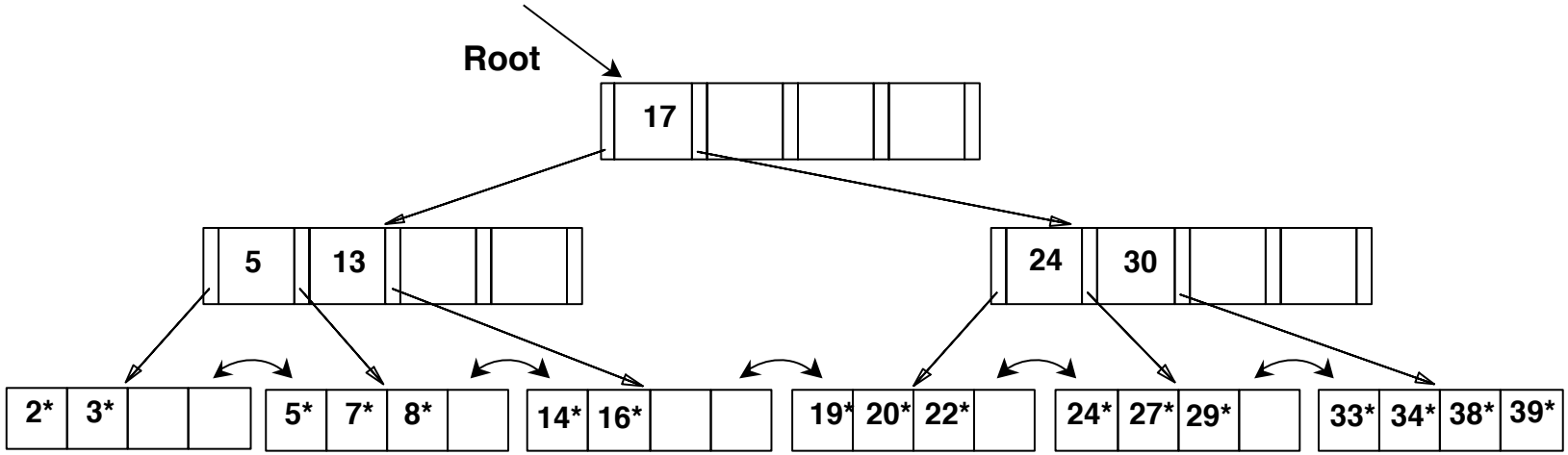
Quiz #1 in class next Wednesday

DavisDB, Part 2: Indexing

- Second part of project: **indexing** component
- Provides classes and methods for managing persisting indices on data in unordered heap files (i.e., record files)
- Like RecordManager, uses page files underneath
- Sits side-by-side with RecordManager on top of PageFileManager
- Indexing structure we'll use: **B+ tree** (with some simplifications)

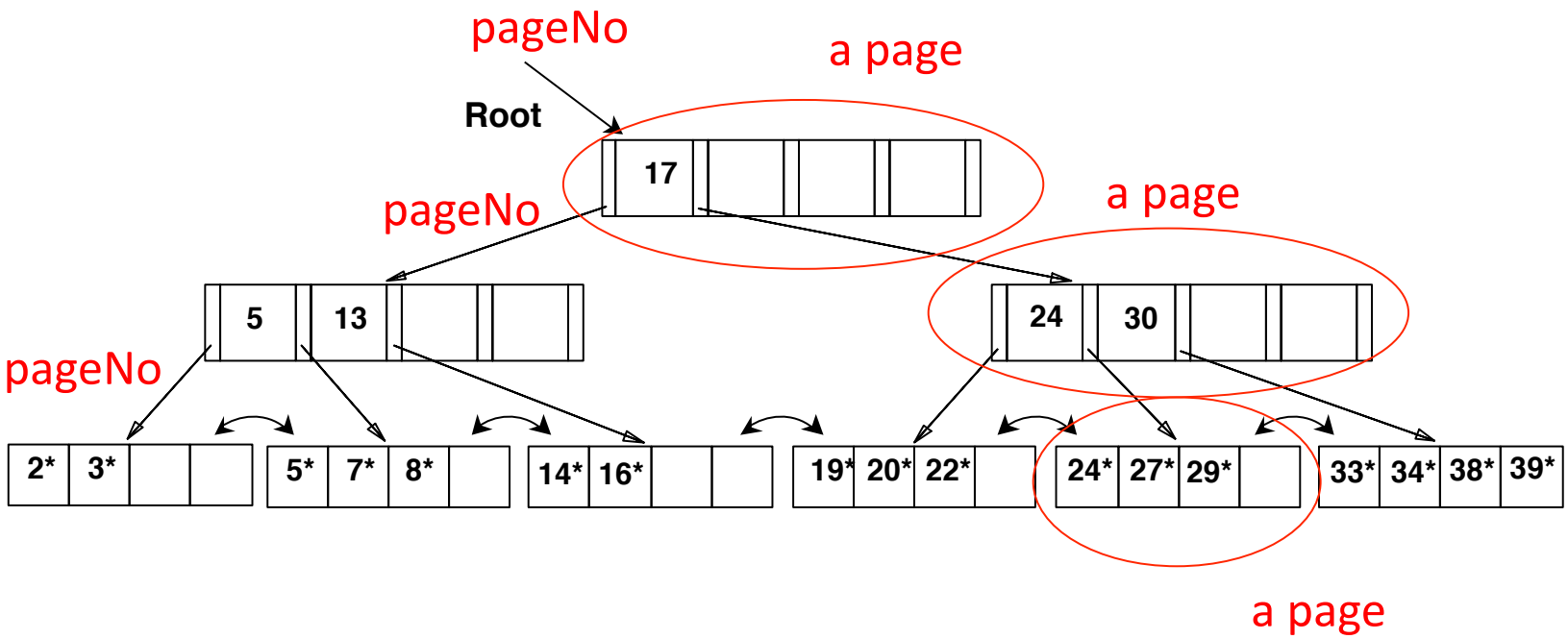
Recall: B+ Trees

- We already covered B+ trees in depth in Lectures 4 and 6



- 3 alternatives for storing records (records themselves, ids, or lists of ids)
- Insertions: need to **split** nodes when they become full
- Deletions: need to **merge** nodes when they become less than half-full

B+ Trees and Page Files



Will also need a **header page**, just as in Record Manager (with pageNo of root node, perhaps some statistics, etc)

Simplifications

- Only need to support-attribute index (recall that B+ tree may in general index several attributes)
- Deletions: you may use **tombstones** instead of **merging/redistribution**
  - When an entry is deleted, it is replaced by a special marker indicating an empty slot (which may be reused later)
  - Tree nodes are never deleted or merged
- Extra credit for implementing full textbook deletion algorithm (with merging/redistribution) --- tricky!
  - May help reduce I/Os when index is used subsequently for answering queries
- No special support for bulk loading

Which of Three Alternatives?

- Alternative 1 (keep record itself in tree): don't do this; you should keep record ids, not records

- Alternative 2 (<key,rid>) or Alternative 3 (<key, list of rids>): either is OK; think about the tradeoffs before coding

- Note, cannot assume all rids for a given key will fit on one page

Handling Duplicates

- If using Alternative 2 (<key, rid>), may have duplicate key entries in internal nodes

- If using Alternative 3 (<key, list of rids>), have to worry about variable-length list of rids and page overflow

- Allowed simplication (described in R&G): include record id in the key – no duplicates, by construction!
  - hence <key, rid> becomes the key
  - downside: index uses more space (=> more I/Os)
  - upside: deletions are faster (don't have to scan the duplicates)

How it All Fits Together

- Three main classes: **IndexManager**, **IndexHandle**, and **IndexScan**
- **IndexManager**: create/delete/open/close B+ tree indices
- **IndexHandle**: insert/delete records
- **IndexScan**: perform comparison-based scans
- Headers and skeleton classes will be added to your repositories tonight

Coding Tips

- "Use the debugger, not printf!" true in general; but every rule has an exception
  - You may find it helpful to write an IndexHandle::dumpTree() method to output a human-readable picture of your B+ tree
- For deletions, can assume at most one scan is open; deletions **can** occur during the scan, of a certain form: you need to allow deletions of some/all entries satisfying a condition
- Get insertions working first, then scans, then deletions, then deletions during scans
- Keep it simple

Let's Go to the Doxygen Docs!

Already online at the usual place:

http://www.cs.ucdavis.edu/~green/courses/ecs165b/docs