COSC2406/2407 Database Systems

Tree Index Structures

Xiangmin (Emily) Zhou

Thursdays

Email : xiangmin.zhou@rm

Lecture 5

References: Ramakrishnan & Gehrke Chapter 10
Garcia-Molina et al. Chapter 13
Elmasri & Navathe Chapter 5
Diagrams courtesy Ramakrishnan & Gehrke

Overview

In this lect

Specific

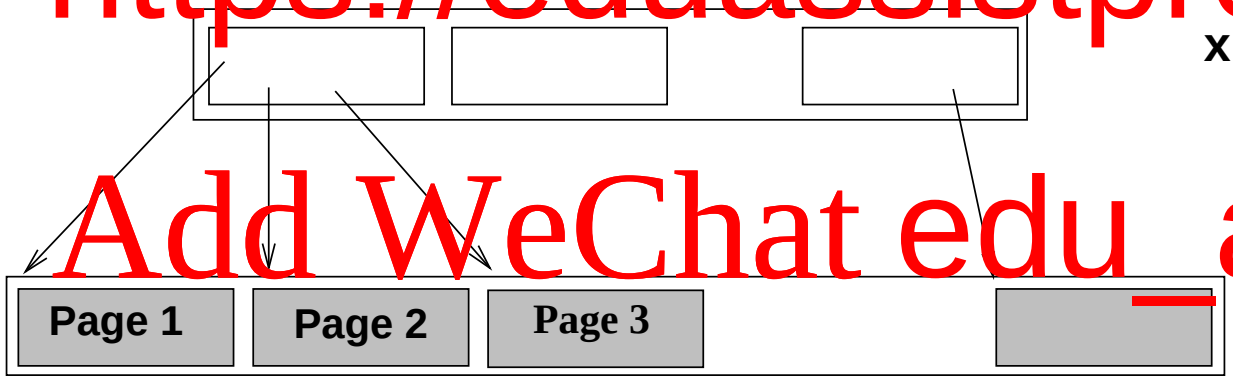
- The
- Dynamic B+-trees

Tree-structured Indexes

- As for any index, 3 alternatives for index data entries $k*$:
  1. Data record with search key value k
  2. (k, `rid` of data record with search key value k)
  3.
  – Ch
- Tre hes and equality searches
- ISAM (Indexed Sequential Access Meth structure
- B+-tree is a dynamic structure that adjusts w deletions

Motivation for Tree Structures: Range Searches

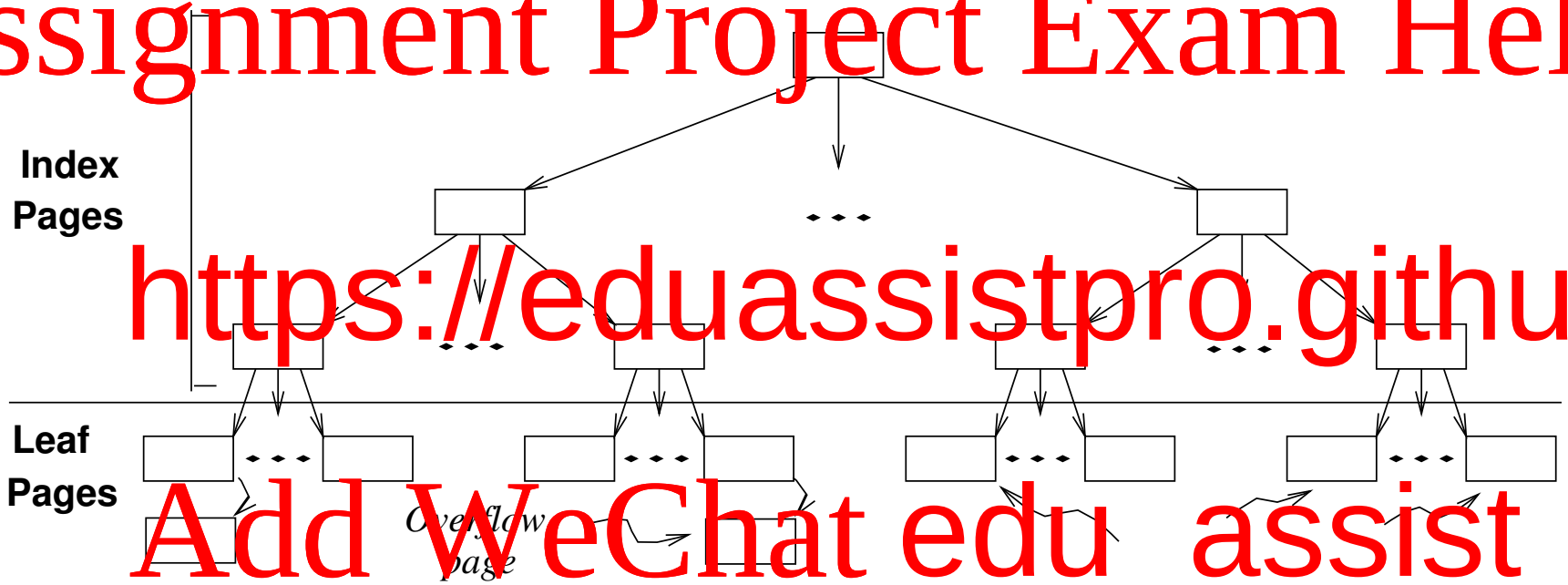
- *"Find all students older than 22."*
  - If data is in sorted file, we can binary search to find first such student, then scan to find other matches
    - The cost of binary search can be high
- Sim



- Now we can binary search on the (smaller) index file!

Motivation for Tree Structures: Range Searches ...

Index file may still be quite large. Apply idea repeatedly.

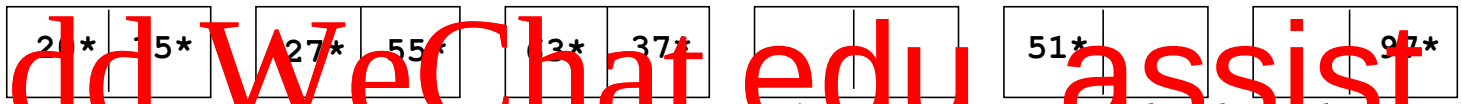


Leaf pages contain data entries.

Tree Indexes

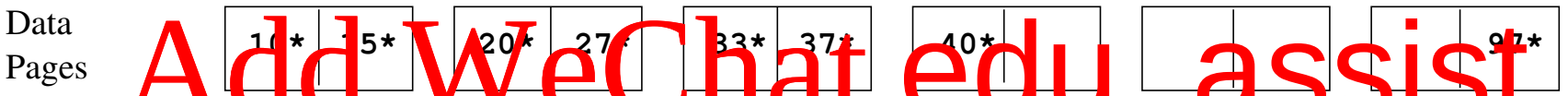
Consider a heap file, with 2 records per page:

Data Pages



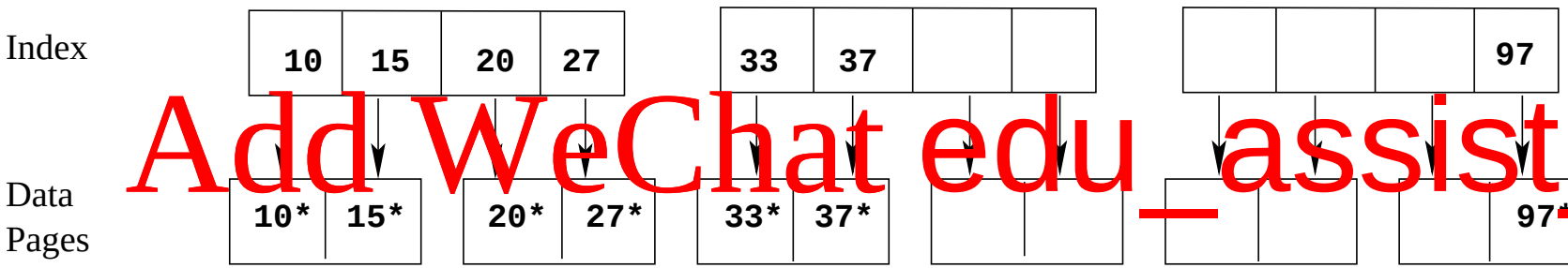
Tree Indexes...

Sort file on search key:



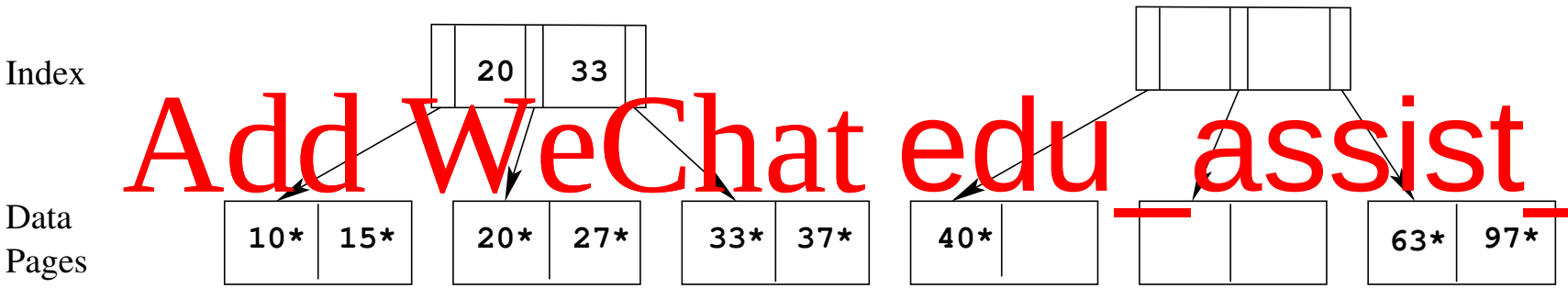
Tree Indexes...

Build an index on file:



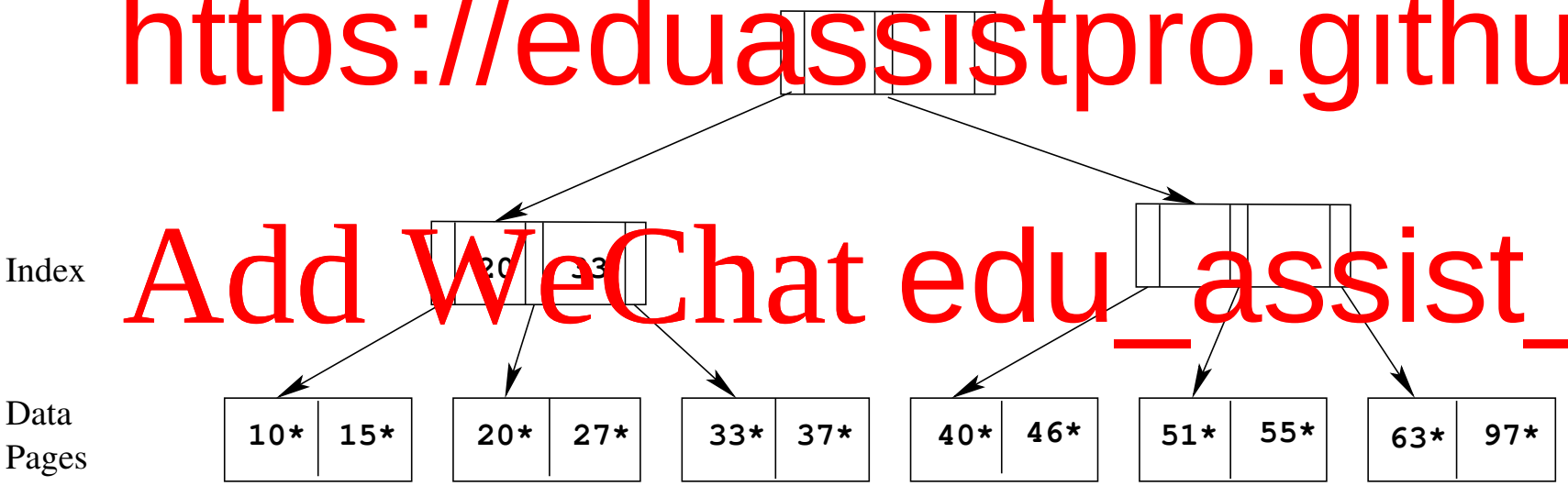
Tree Indexes...

Build a sparse index on file:



Tree Indexes...

Build an index on the index:



ISAM

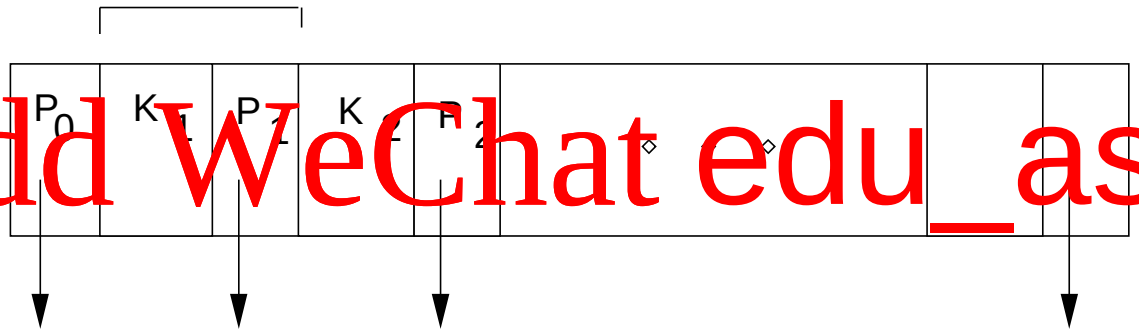
The tree we have constructed is a so-called ISAM (Indexed Sequential Access Method) structure. (See Section 10.2)

Each index page has *index data entries* of the form (key, pointer).

Each ind

A key serv

pointed t

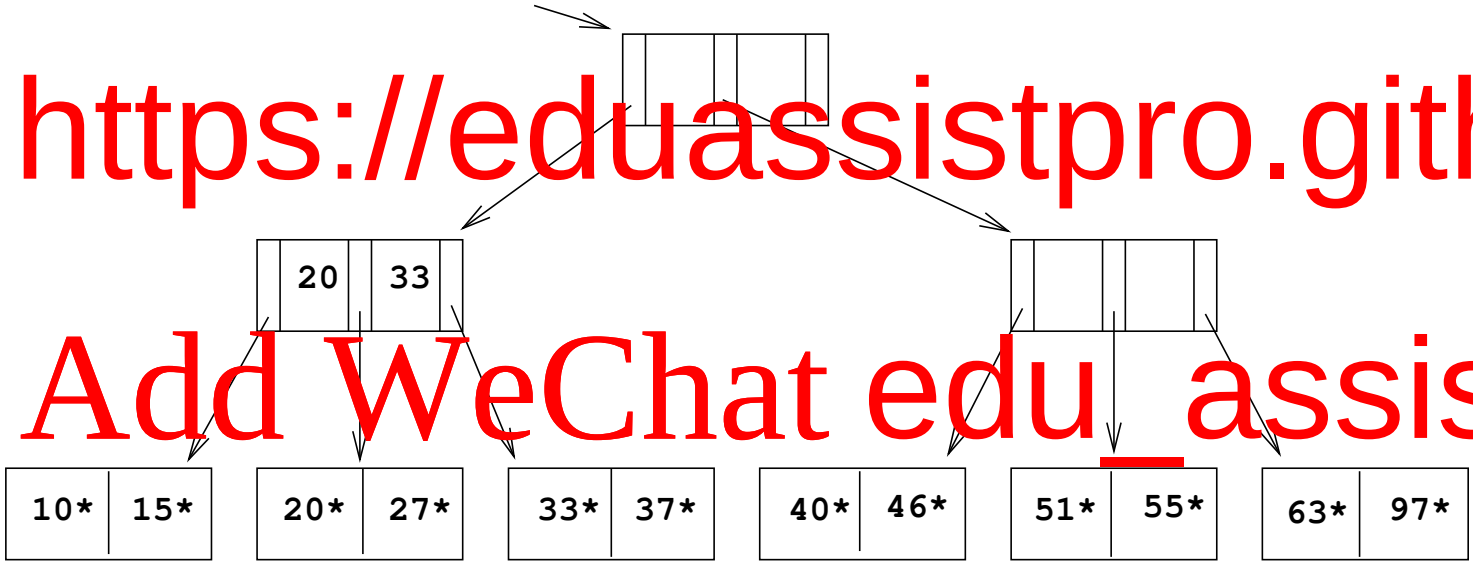


Comments on ISAM

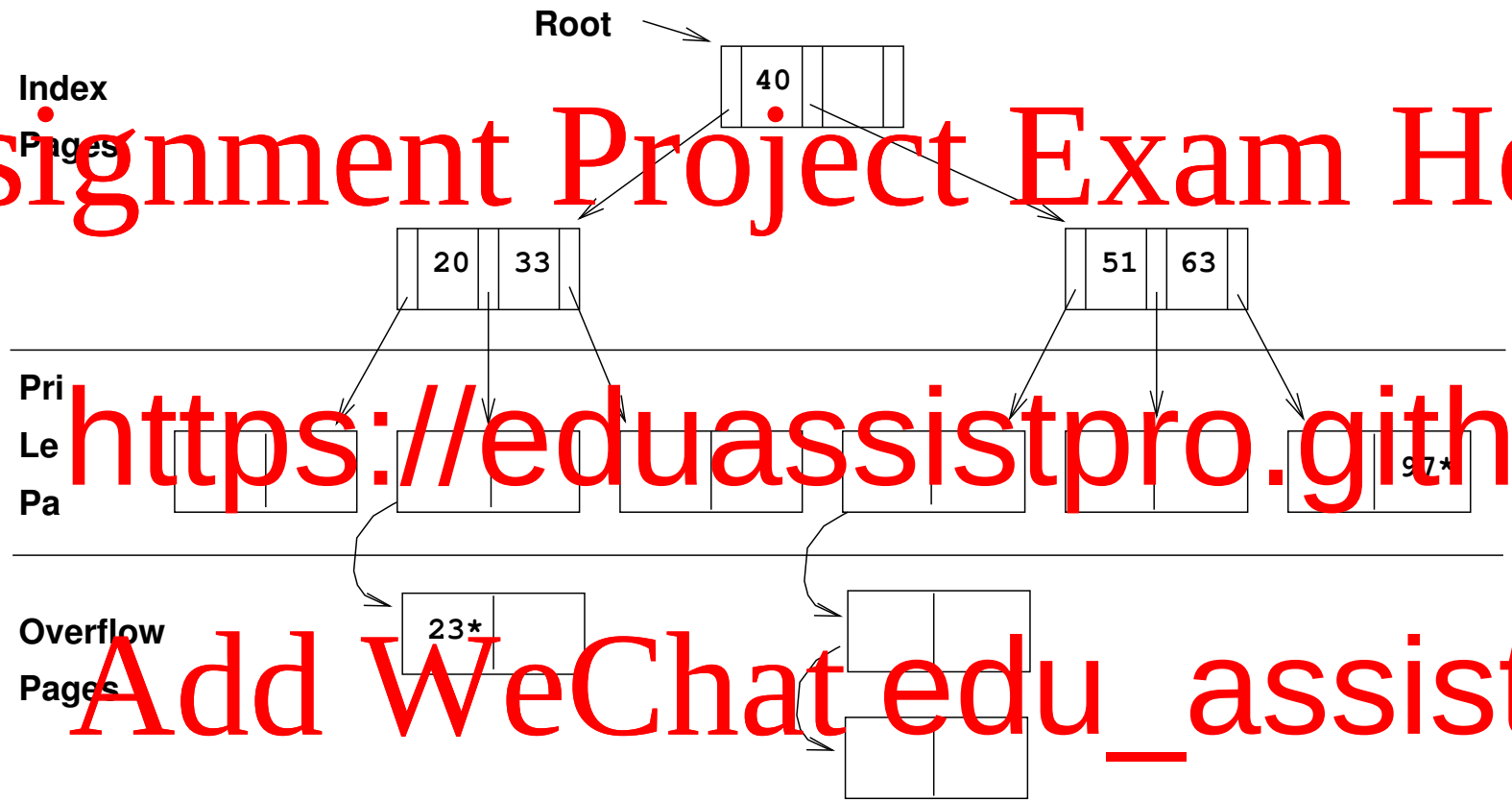
- File creation: Leaf (data) pages allocated sequentially, sorted by search key; then index pages allocated, then space for overflow pages.
- Ind
  sea
- Se
  Co $\log_F$ N is
  number of leaf pages.
- Insert: Find the leaf page where the data entr
  it there. Create overflow pages if necessar
- Delete: Find and remove the entry from the leaf; if this empties an overflow page, de-allocate the page.

Example ISAM Tree

Each node can hold 2 entries; no need for 'next-leaf-page' pointers (because of the sequential allocation of leaf pages).

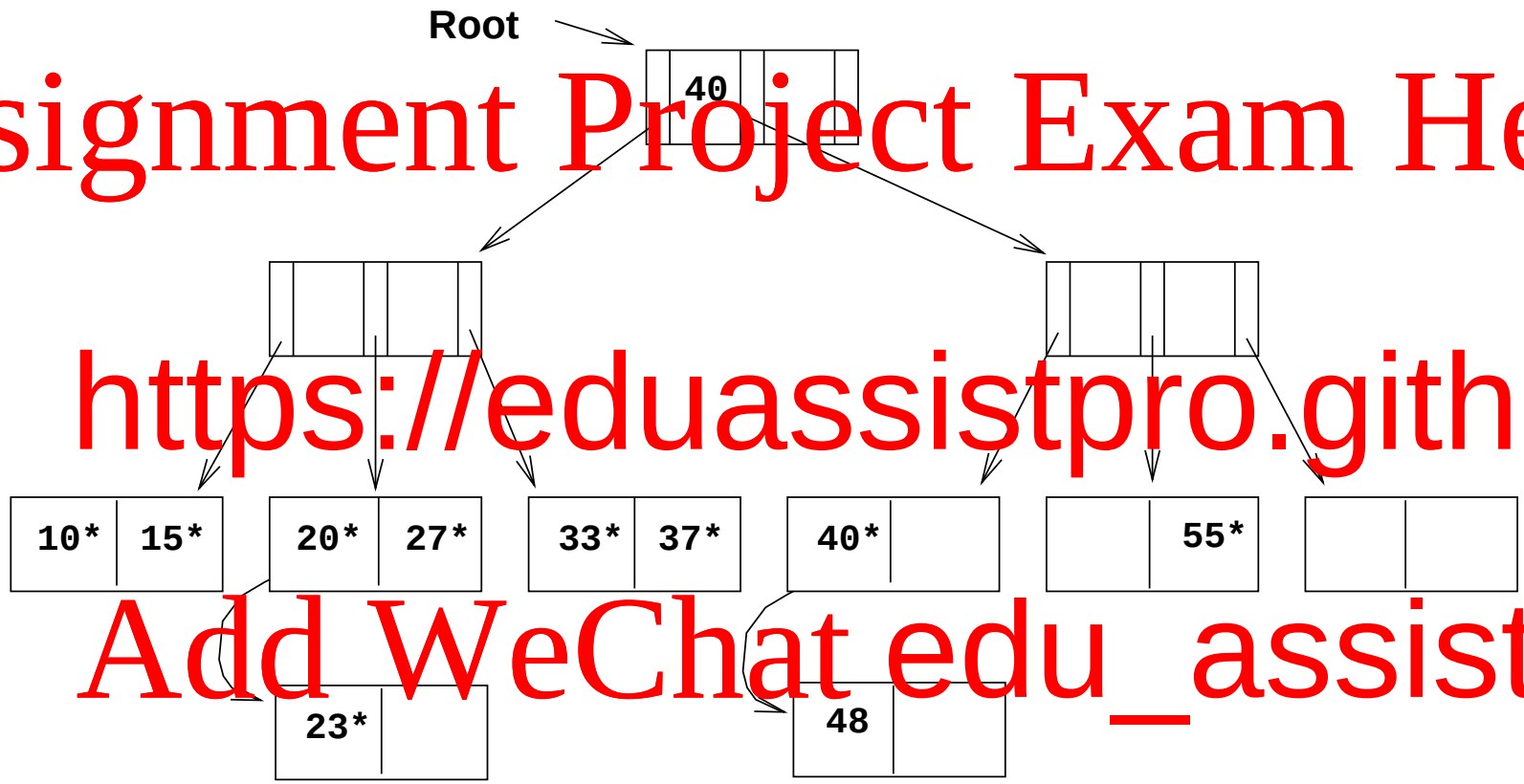


After Inserting 23*, 48*, 41*, 42* ...



Advantage: Less locking problems in ISAM than in other structures, since no index page is changed.

... Then Deleting 42*, 51*, 97*



Note that 51* appears in index levels, but not in a leaf!

Overflow Chains

Once an ISAM file is created, inserts and deletes affect only the leaf page content.

As a result required t well.

To reduc (20%, say) for future insertions.

Otherwise, the only way to get rid of overflow chain reorganisation of the whole file structure.

ISAM and Locking

The fact that only leaf pages are adjusted gives a big advantage for *concurrent* access:

When a pa ot
concurr
of users w
significa
the root of a tree.

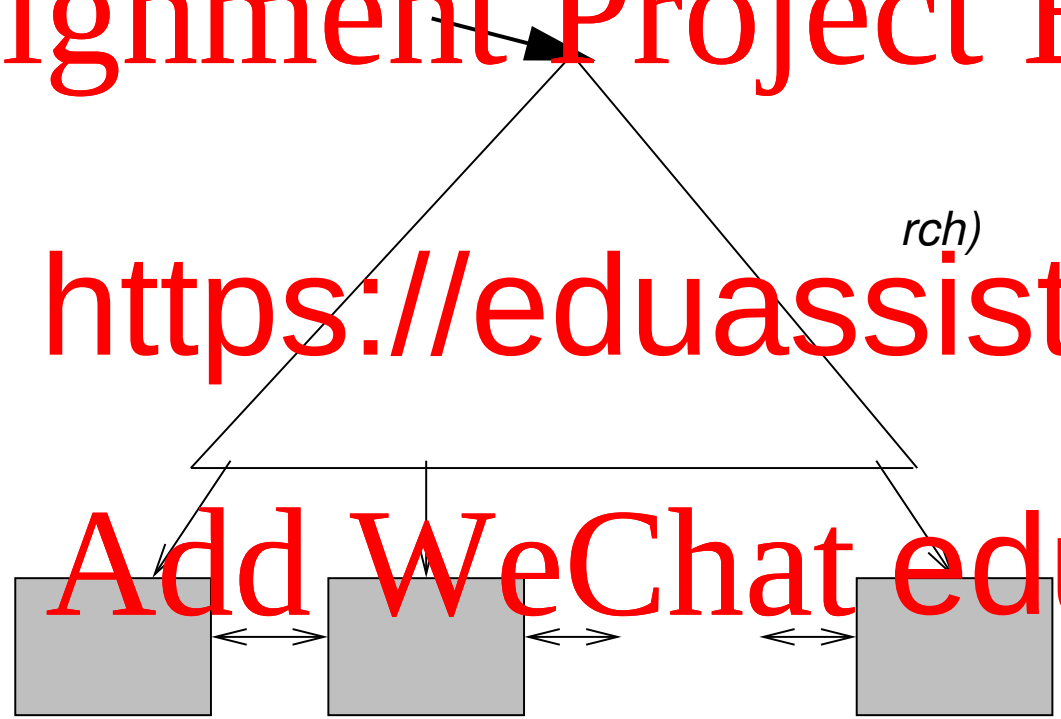
Since ISAM index pages are never modified, the
locked.

B+-tree: The Most Widely Used Index

The B+-tree is the most widely used index structure (see Section 10.3). It has the following characteristics:

- Internal nodes direct the search, index data entries are in the leaf
- The
- Ins *N* the nu
- Min
- Each node contains $d <= m <= 2$ ter d is called the *order* of the tree.
- Supports equality and range-searches ef
- Leaf pages are organised as a double linked list for fast traversal and reorganisation

B+-tree Index File



Node Format

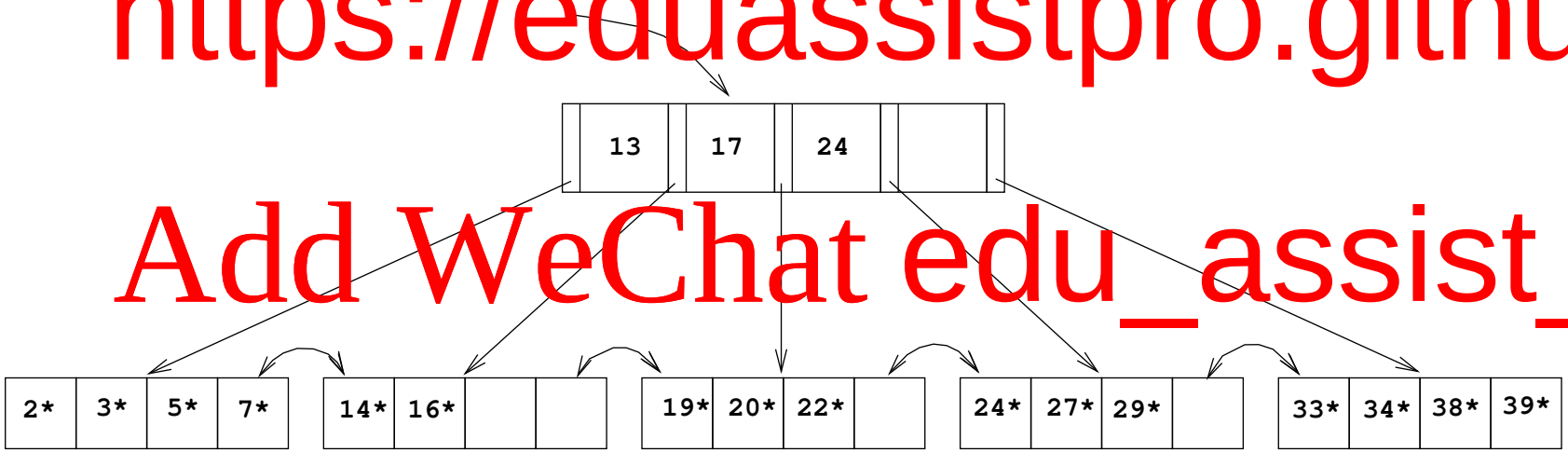
Non-leaf nodes contain m index entries, with $m + 1$ pointers to children (like ISAM structure).
Leaf nodes contain data entries, either:

- act *egrated* ind dat
- pointers to data records elsewhere on disk (the B+-tree is an *index file*, distinct fro records

If the indexed field is of fixed-length, so is the index e
the index entries are variable-length.

Example B+-tree

- A search begins at the root, and key comparisons direct the search to a leaf (as in ISAM)
- (Tr



Inserting a Data Entry into a B+-Tree

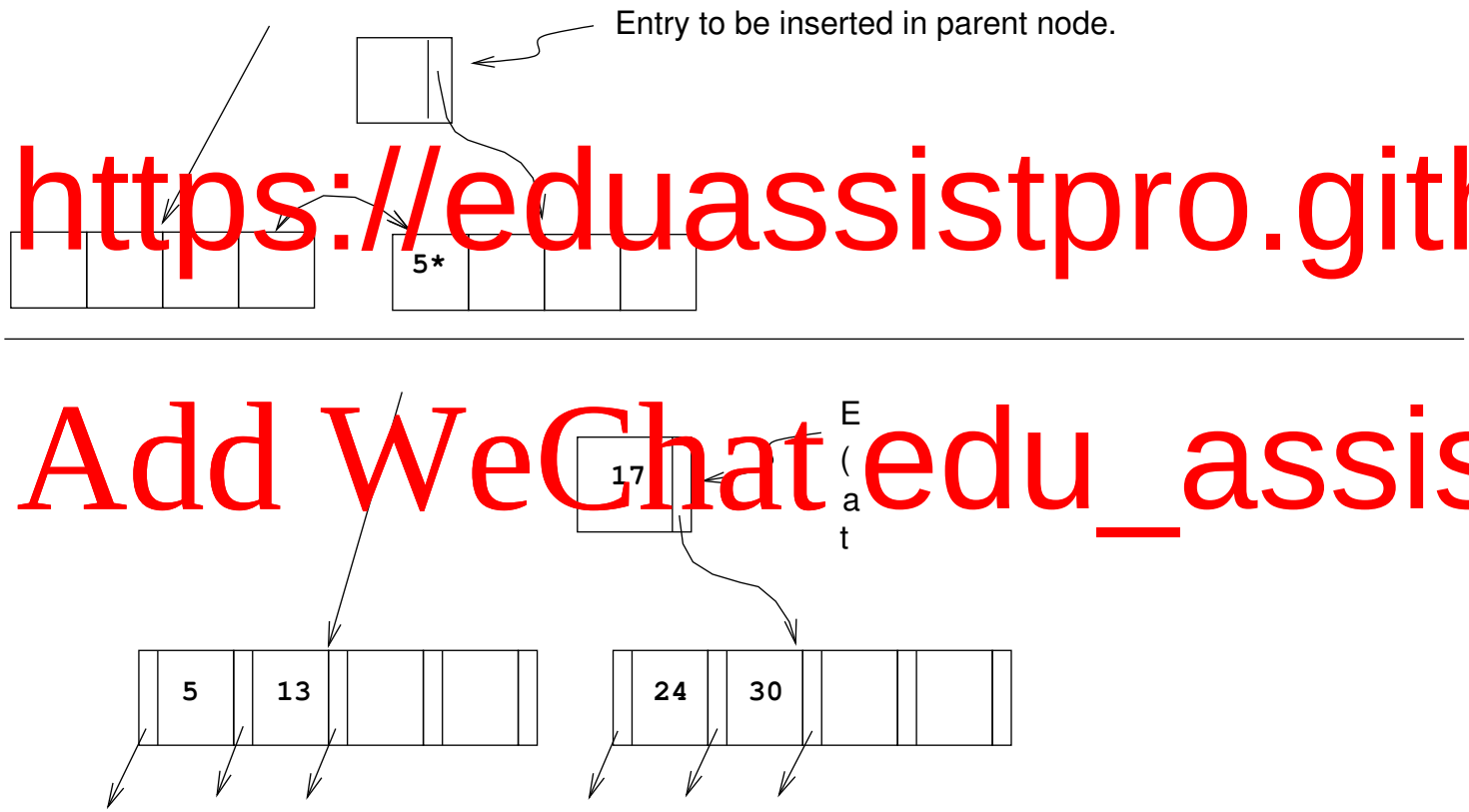
- Find correct leaf L.
- Put data entry into L.
  - If L has enough space, done!
  -

- This can happen recursively.
  - To split index node, redistribute entries e
    but *push-up* middle key. (Contrast with le

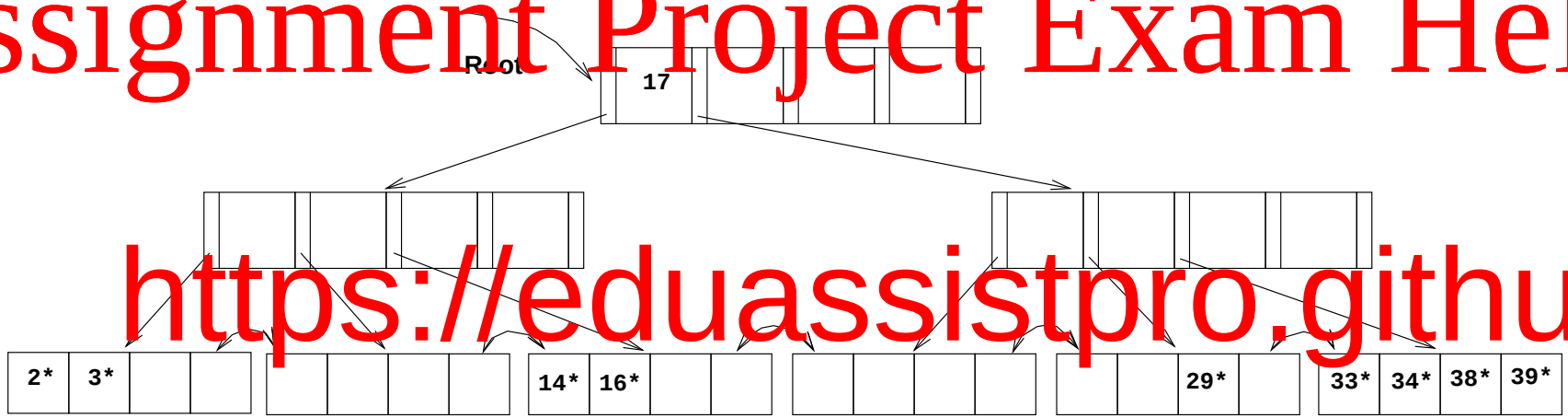
- Splits "grow" tree; root split increases heig
  - Tree growth: gets wider or one level taller at top.

Inserting 8* into Example B+-tree

- Observe how minimum occupancy level is guaranteed in both leaf and index pages splits.
- Note difference between *copy-up* and *push-up*!



Example B+-tree After Inserting of 8*



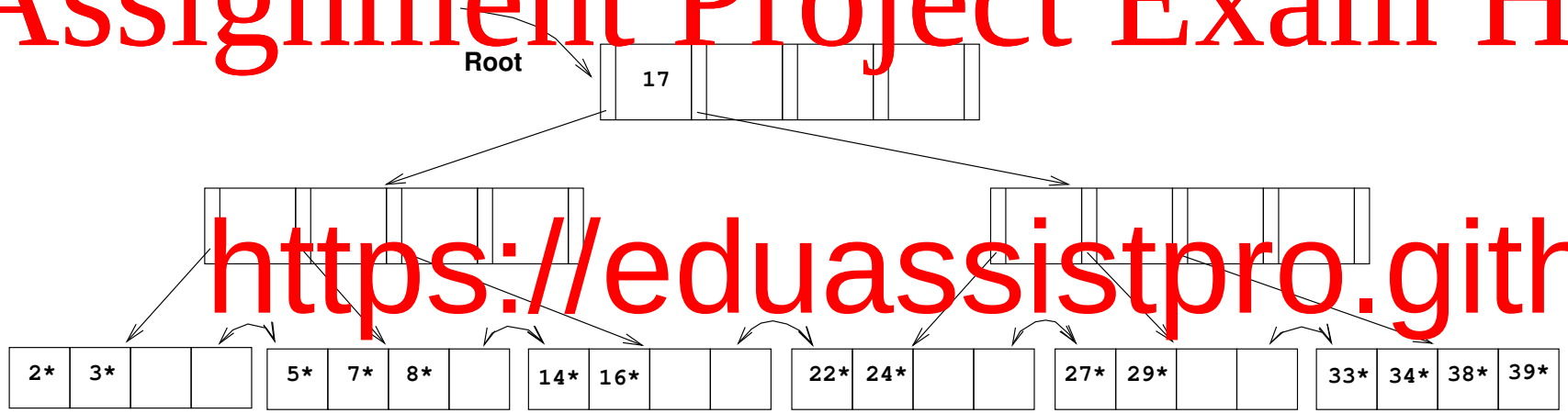
- Notice that root was split, leading to increas
- In this example, we can avoid split by re-distri
  however, this is usually not done in practice.

Deleting a Data Entry from a B+-Tree

- Start at root; find the leaf L where the entry belongs.
- Remove the entry.
  - If L is at least half full, done!
  -
- If merge occurred, must delete entry (pointi) from parent of L.
- Merge could propagate to root, decreasin

Refer to Section 10.6 for an example.
Garcia-Molina et al. illustrates a B-tree example in Section 13.3.6.

B+-tree After Deleting 19* and 20*

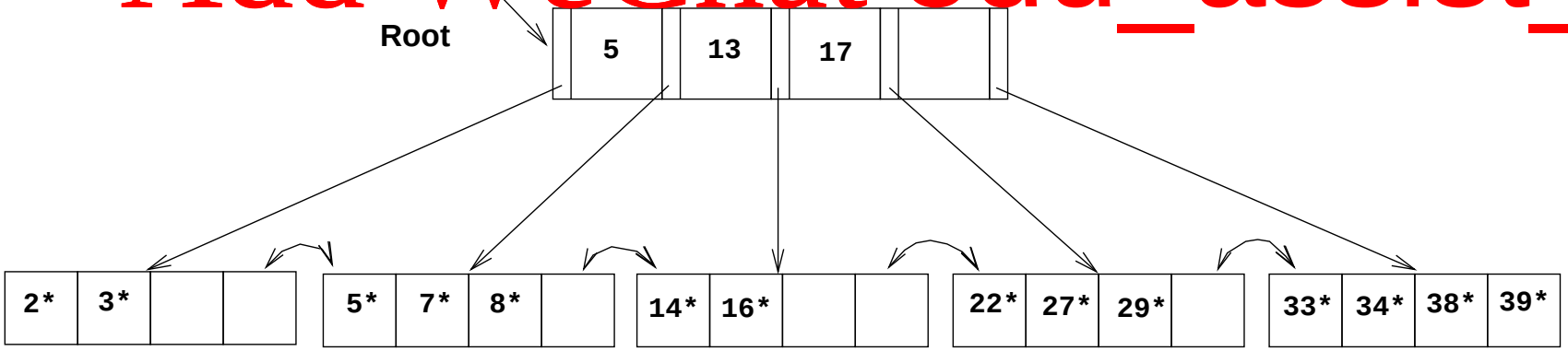


Deletion of 20* is solved by re-distribution from sib
the new splitting (middle) key is copied up.

Deleting 24*

Leaf nodes must be merged:

Then, index nodes need to be merged and entry 1



B+-trees in Practice

- Typical order (minimum node size): 100. Typical fill-factor: 67%.
  - Average fan-out = 133
- Ty
  - https://eduassistpro.github.i
- Can often hold top levels in the buffer manag
  - Add WeChat edu_assist_pr
  - Level 1 = 1 page = 8 Kbytes
  - Level 2 = 133 pages = 1 Mbyte
  - Level 3 = 17,689 pages = 133 Mbytes

B+-trees in MongoDB and Derby

Read more indexes in MongoDB (including use of B+-trees)

`https:/`

Read more about how B+-trees are implemented in Derby:

`http://db.apache.org/derby/papers/`

Summary: Tree Index Structures