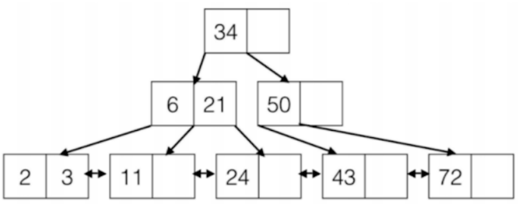
CS W186 Introduction to Database Systems
Spring 2020 Josh Hug, Michael Ball **DIS 3**

1 Indices (B+ Trees)

Assume we have the following B+ Tree of order 1. Each index node must have either 1 or 2 keys (2 or 3 pointers), and the leaf nodes can hold up to 2 entries.



(a) What is the maximum number of insertions we can do without changing the height of the tree?

Max number of entries: $2d * (2d+1)^h = 3^2 * 2 = 18$

18 - 6 = 12 insertions

(b) What is the minimum number of keys you could insert to change the height of the tree?

3. A possible insertion pattern is: 1, 4, 5

2 Indices

Two sets of terminology:

Clustered vs. unclustered

- In a clustered index the index field specifies the sequential order of the table file; in contrast, in an unclustered index the table file is not ordered by the index field. One consequence of this is that range queries on a unclustered index are much more inefficient than those on a clustered index.

Three alternatives for storing underlying data: alternatives 1, 2, and 3.

- (A1) By value: entire row in leaf
- (A2) By reference: (key, rid) pairs, with each key occurring once
- (A3) By reference: (key, [rid1, rid2, ...]) pairs

1. Is it possible to have two clustered indices on separate columns?

   Yes, but you would generally have to store two copies of the data (and keep both up to date).

2. Suppose we have an alternative 2 unclustered index on (assignment_id, student_id) with a depth of 3 (one must traverse 3 index pages to reach any leaf page).

   Here's the schema:

```
CREATE TABLE Submissions (
    record_id integer UNIQUE,
    assignment_id integer,
    student_id integer,
    time_submitted integer,
    grade_received byte,
    comment text,
    regrade_request text,
PRIMARY KEY(assignment_id, student_id));

CREATE INDEX SubmissionLookupIndex ON Submissions (
    assignment_id, student_id);
```

   Assume the table and its associated data takes up 12 MB on disk (1 MB = 1024 KB) and that page size is 64 KB. (This includes extra space allocated for future insertions.)

   (a) We want to scan all the records in Submissions. How many I/Os will this operation take?

   To do a full table scan, we read each page into memory once.

   There are 12 * 1024/64 = 192 pages in the table, so that's 192 I/Os (all page reads).

   (b) UPDATE Students SET grade_received=85 WHERE assignment_id=20 AND student_id=12345; How many I/Os will this operation take?

   Read the page into memory: 3 page reads for the index + 1 page read for the leaf page + 1 page read for the data page.

   Write the modification in memory and then flush the page back to disk: 1 page write for the data page.

   This costs us a total of 6 disk I/Os.

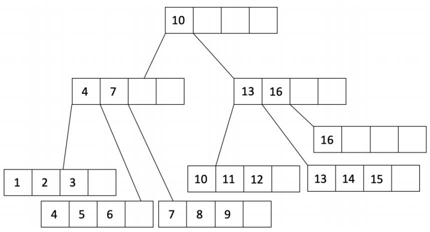
   (c) In the worst case, how many I/Os does it take to perform an equality search on grade_received?

   In the worst case, any record can match the grade_received predicate. Therefore, we must check every record of the table. This is equivalent to performing a table scan, and we would read every page, requiring 192 page reads.

3 Bulk-Loading

Suppose we were to create an order d=2 B+ tree via bulk-loading with a fill factor of 3/4. Here, fill factor specifies the fill factor for leaves only; inner nodes should be filled up to full and split in half exactly.

We insert keys with all integer values from 1-16 in order. Draw out the final B+ tree. What is its height?



The final height of the tree is 2.