Lecture 17: Indexed Sequential Files

Last Day: Dynamic Indexed File

- Binary Trees, Multiway Trees
- B-Trees
- Insertions

Today: B-Trees

- Sequential Retrieval
- Deletions

Folk & Zoellick, ch. 9.

Retrieval

- Single Records: Given a key, a B-tree can quickly retrieve the record with that key

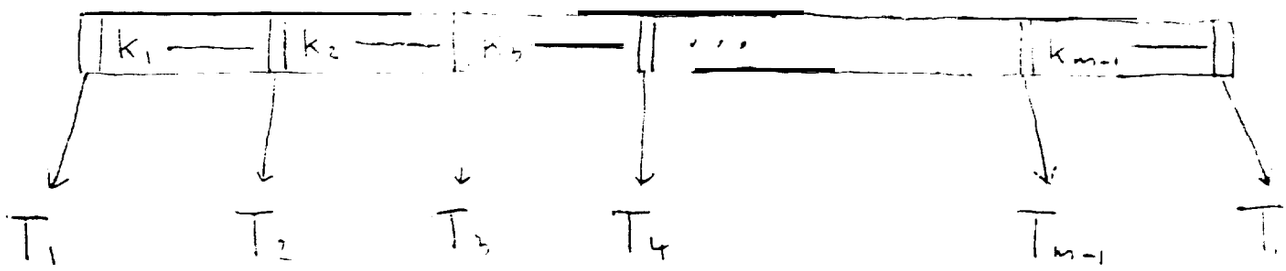
- Range Queries: Given key_1 and key_2, a B-tree can quickly retrieve all records such that $key_1 \leq key \leq key_2$ (& in sorted order)

- Sequential Retrieval: A B-tree can efficiently retrieve all records in sorted order (sorted by key).

Sequential Retrieval

Problem: From a given B-tree node (eg, the root), retrieve all records at or beneath this node in sorted order.

Node:



Recursive Algorithm:

First, print the records in subtree T_1 in sorted order.

Then, print record k_1

Then, print the records in subtree T_2 in sorted order.

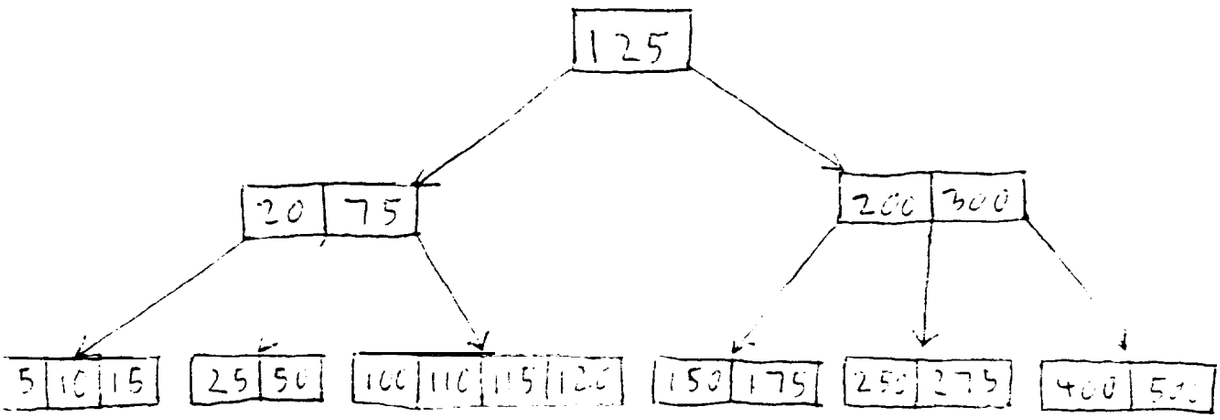
Then, print record k_2.

...

Finally, print the record in subtree T_m in sorted order

Example

Sequential retrieval of all records in a B-tree



Execution Trace of the Recursive Algorithm:

↓↓ 5, 10, 15 ↑ 20 ↓ 25, 50 ↑ 75 ↓ 100, 110, 115, 120 ↑↑ 125

↓↓ 150, 175 ↑ 200 ↓ 250, 275 ↑ 300 ↓ 400, 500 ↑↑

Note: Each node is read once.

∴ Total cost = 9 file accesses

No-2

- Although the records of a B-tree can be retrieved in sorted order (i.e. sequentially) they are not stored in sorted order.

- In fact the nodes of a B-tree may be scattered randomly in a file.

Deleting Records from a B-Tree

Problem: Nodes may become too small.
(Recall that B-tree nodes must be at least half full.)

Solutions:

- Borrow records from another node.
- Merge two small nodes

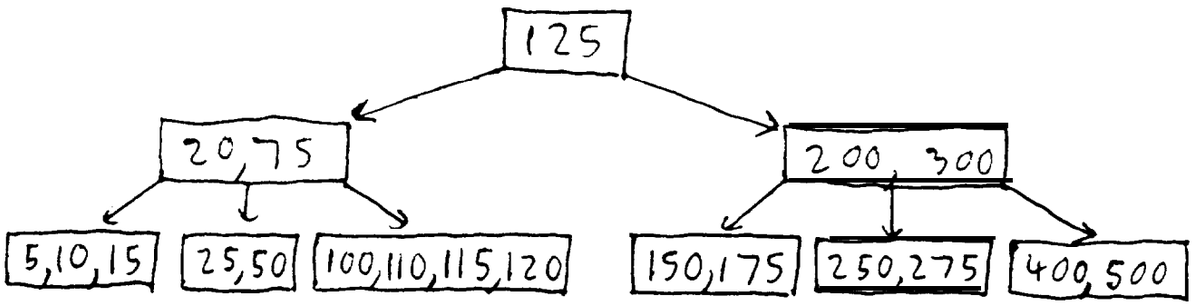
Two Cases

(1) Deleting a record from an internal node

(2) Deleting a record from a leaf node.

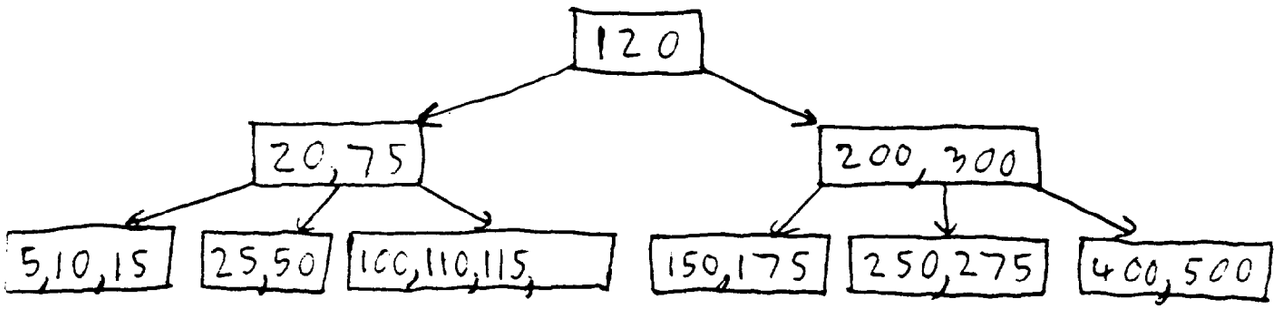
Case (1) reduces to case (2), as we shall see

Example: B-tree of order 5



delete 125

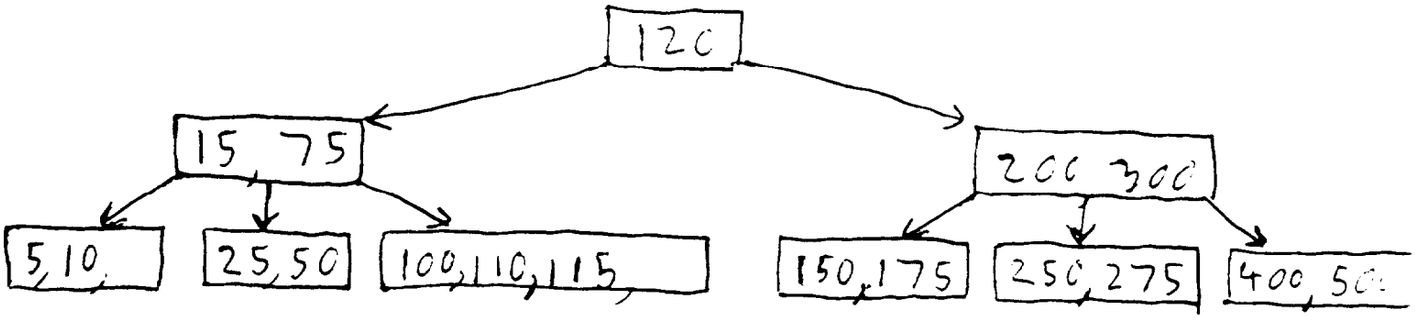
- swap 125 with the largest key in its left subtree (120).
- 125 is now a leaf node.
- remove 125 from this leaf, to give



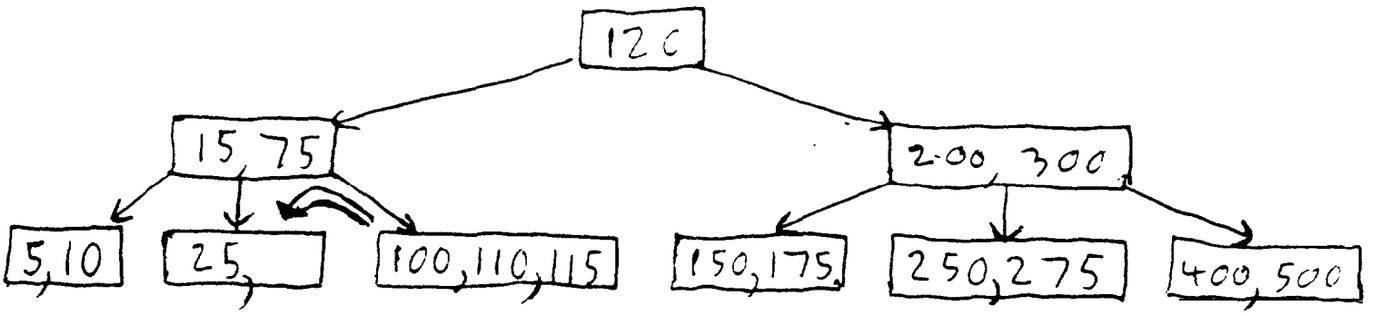
Note: the result is still a valid B-tree

Delete 20:

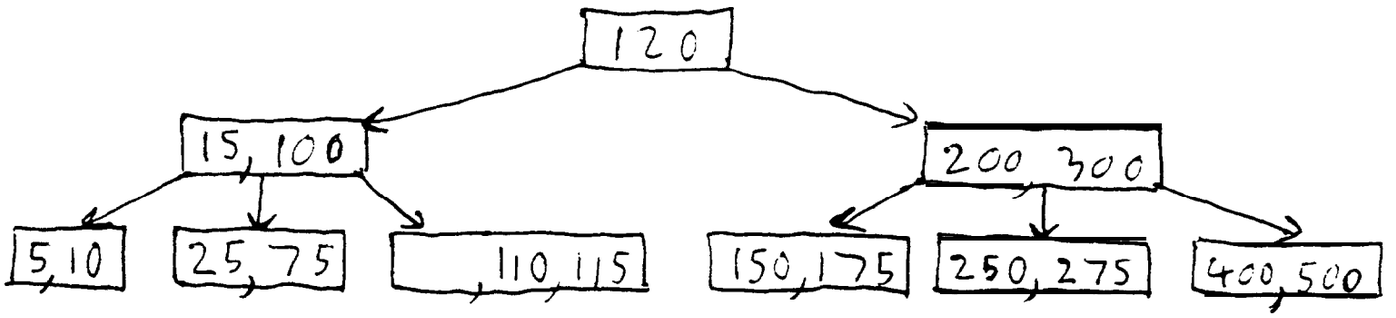
- swap 20 with the largest key in its left subtree (15).
- 20 is now a leaf.
- Delete this leaf, to give:



Delete 50:

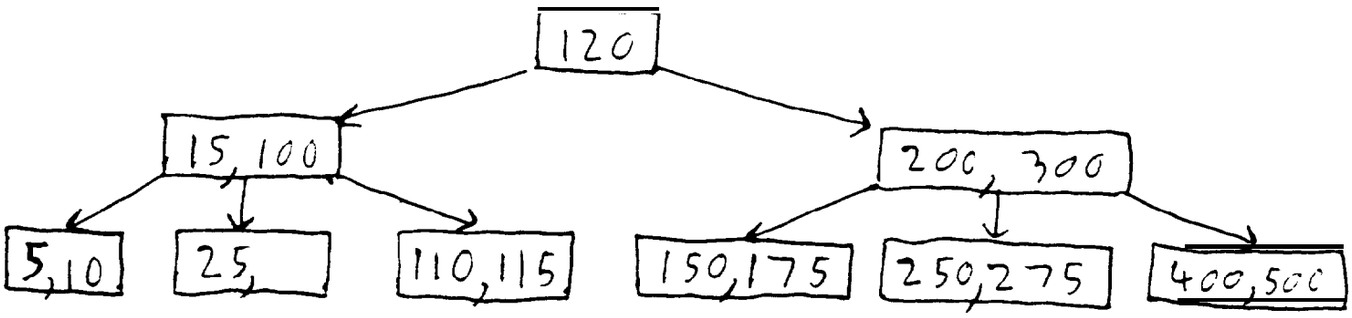


- Node is too small! (fewer than 2 records)
- Borrow a record, by "rotating" it from a larger sibling node via the parent node.

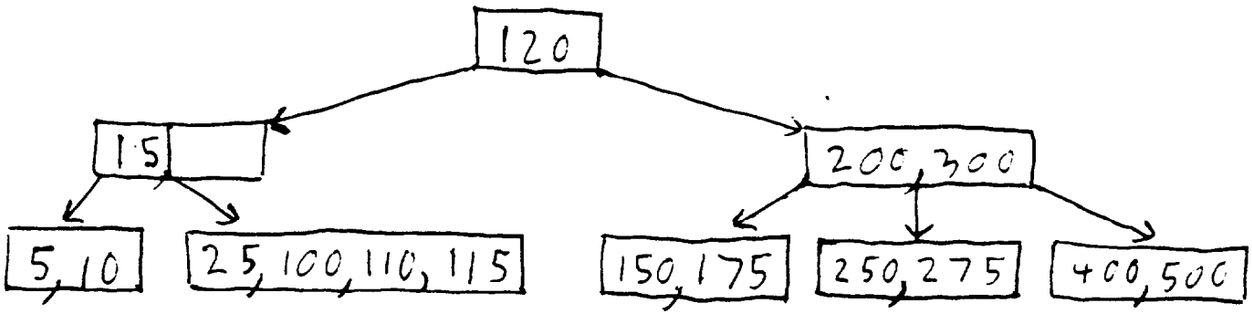


Note: all nodes are now (at least) half full.

Delete 75

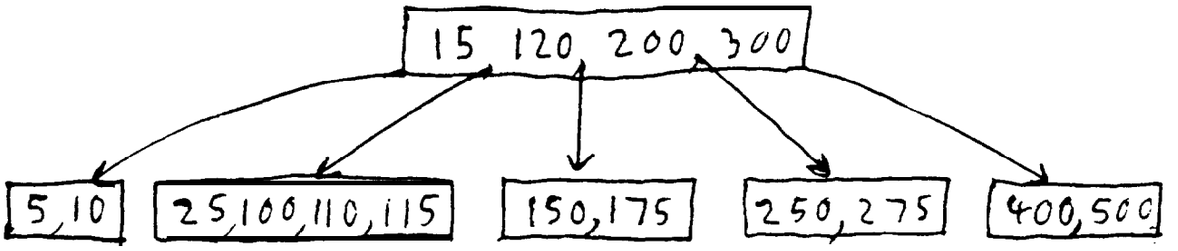


- Node is too small!
- Can't borrow from either sibling.
- Instead merge with a sibling, taking a record from the parent.



Problem. Parent is now too small.

Solution: Merge parent with its sibling, taking a record from the root.



Note: We now have a new root. (ie, the old root is now empty, so we discard it.)

Delete: 150 (borrow)
175 (borrow)
115 (merge)
⋮
Continue until tree is empty

} exercise.

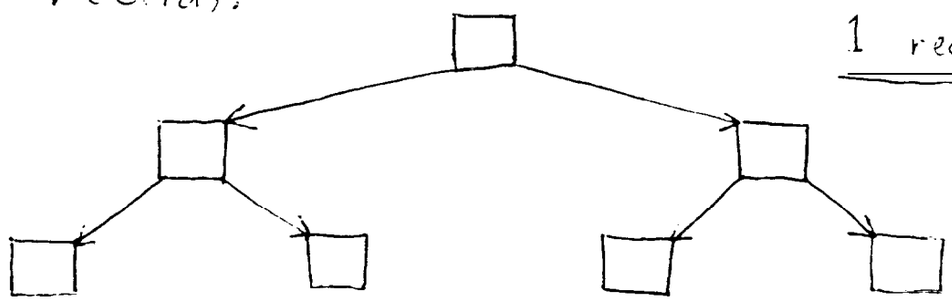
Observation

- We prefer B-trees of higher order.
  ie, Trees with more children per node.

- Such trees can have fewer levels, and so fewer file accesses are needed to reach the leaves (where most records are stored)

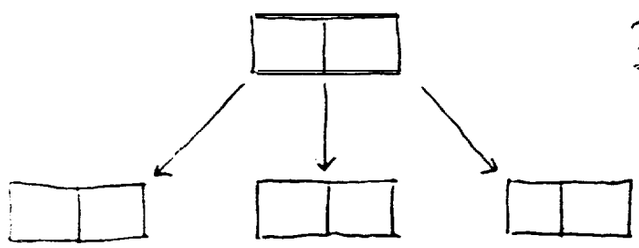
Example

- A tree of order 2 needs 3 levels to store 7 records:



1 record per node

- A tree of order 3 needs only 2 levels to store 8 records:

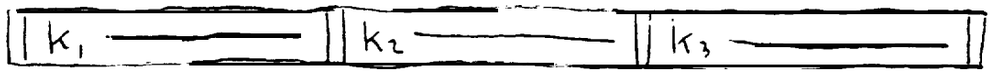


2 records per node

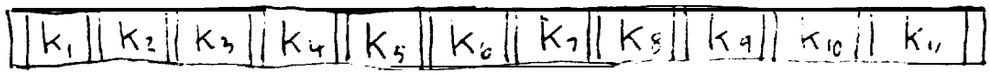
This tree has more records and faster access.

Moral: Pack as many keys as possible into a node (as long as each node fits in one disk block)

Observation: A B-tree is inefficient in this regard since it stores entire records in a node, not just keys

eg. 

Improvement: If we stored just the keys (& no other data) in a node, then we could pack many more keys into a node (& onto a disk block)



The data itself can be stored elsewhere (eg, in another file). This is the basis of B+ Trees

Next Day