Architecture and Implementation of Database Systems

Exercise Sheet 3 - Solution

ETH Zurich
08./09.10.2009

Exercise 3.1 – True or False

Respond with “true” or “false” to the following statements. If your response is “false”, briefly explain why.

(a) Sorted files are a valid option if the data to store is relatively small and does not change.

True.

(b) The asymptotic runtime complexity of lookups in a sorted file of *n* records is O(log(*n*)). Updates and deletes have the same runtime complexity. Insertion runtime complexity is O(*n*).

True.

(c) The ISAM index structure ensures that overflow pages are allocated besides the overflowing data page.

False. Overflow pages are allocated on-demand, wherever there is space.

(d) A clustered B+-tree is is faster at answering key-matching range queries than an unclustered B+-tree.

True.

(e) A clustered B+-tree stores its data pages in order of their keys.

True.

(f) There can only be one clustered index on a table.

False. There may be multiple attributes with an isomorphic order relation.

(g) Every index organized table is a clustered index.

True.

(h) Prefix truncation can potentially decrease the height of a B+-tree.

True (because prefix truncation increases the fan-out, which in turn decreases the height).

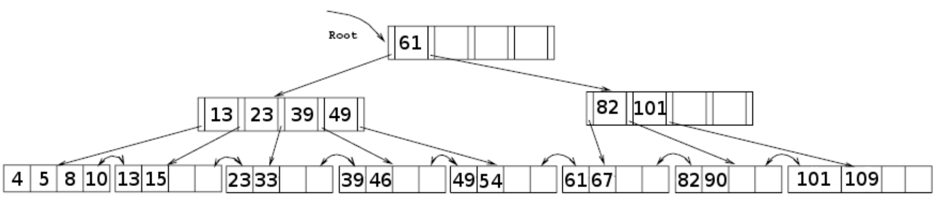
(i) Two separate B+-trees are *always* better than a single, partitioned B+-tree over a composite key.

False. If, for example, a query contains conjunctive equality predicates on each attribute of the composite key, the partitioned B+-tree will return fewer record IDs than either B+-tree on a single attribute.

(j) A hash index is *always* a good alternative to a B+-tree.

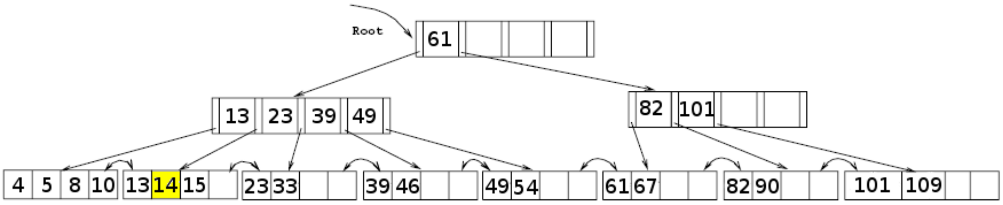
False. Hash indexes do not support range queries (except for really tiny ranges over discrete attribute domains).

Exercise 3.2 – B+-Trees

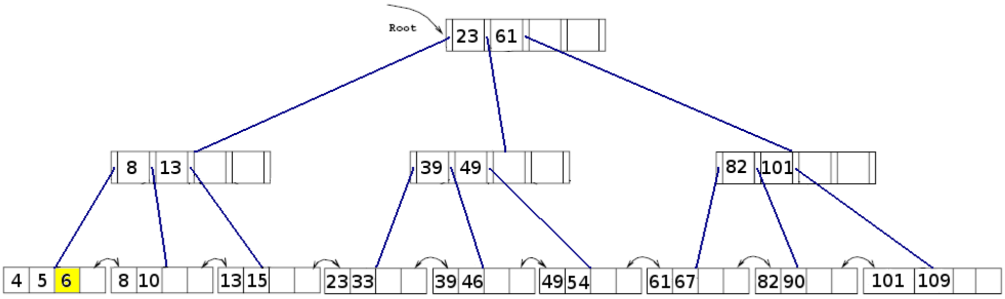


Consider the B+-tree shown above. For each exercise (a) to (h), always start with the original tree shown above. Apply the following operations and show the resulting trees.

(a) Insert an entry with key 14.

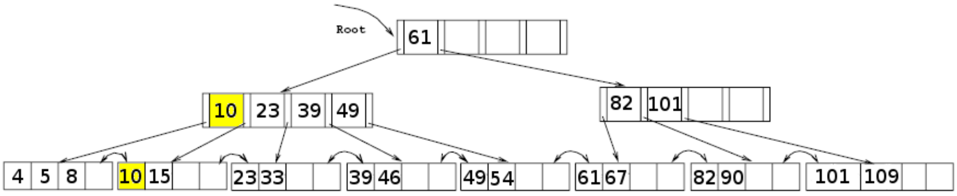


(b) Insert an entry with key 6. How many page reads and writes are required for this insertion?

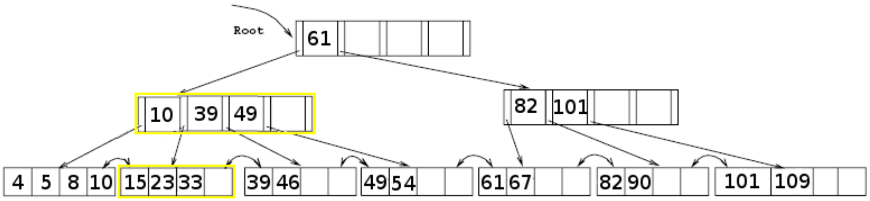


3 page reads, 5 page writes.

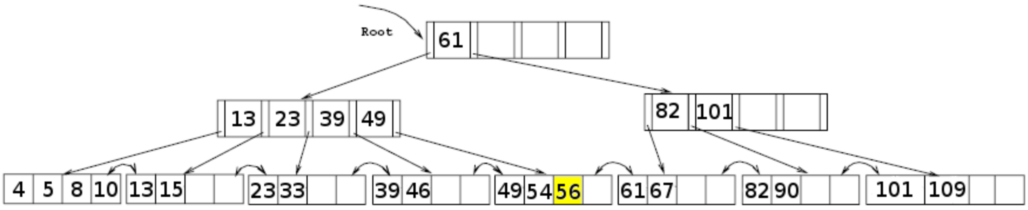
(c) Delete the entry with key 13. Assume that the left sibling is checked for possible redistribution.

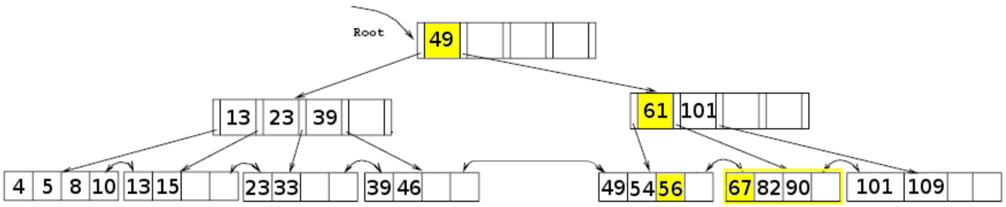


(d) Repeat (c), assuming that right sibling is checked for possible redistribution.



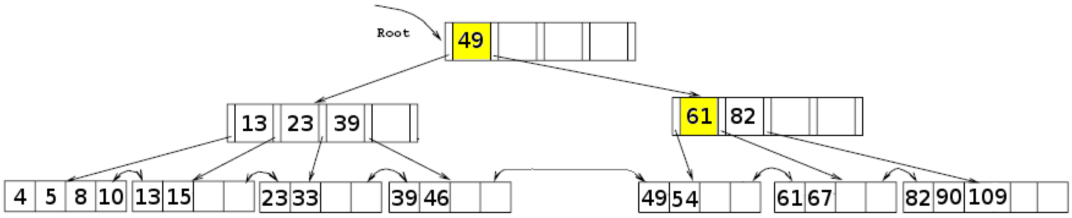
(e) Insert an entry with key 56 and then delete the entry with key 61.



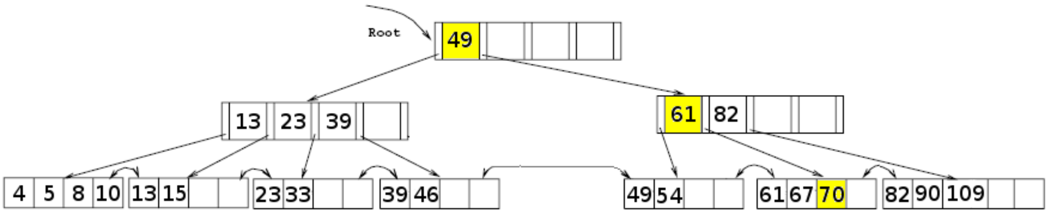


Note that when deleting 61, the leaf node *has to* be redistributed with its right sibling, since the left sibling belongs to a different parent node.

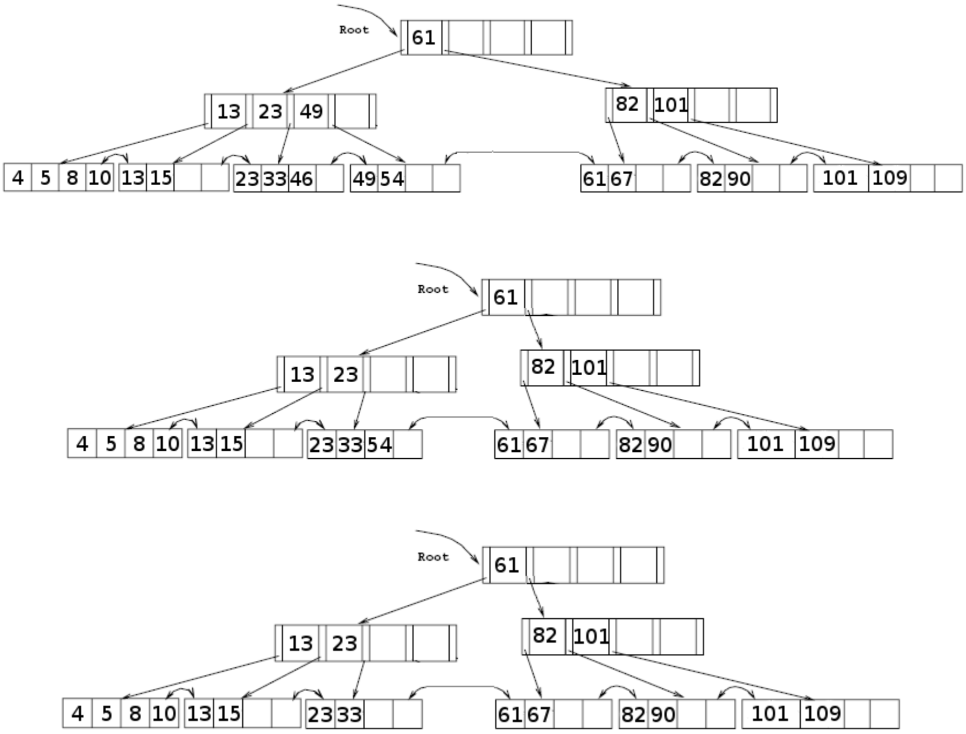
(f) Delete the entry with key 101.

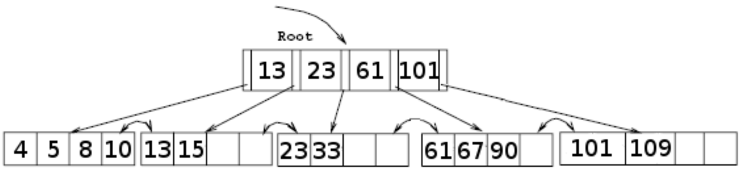


(g) Insert an entry with key 70 and then delete the entry with key 101.

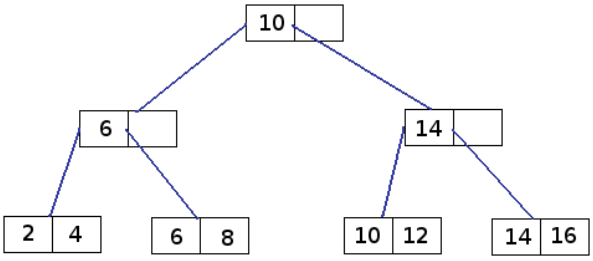


(h) Delete the entries with the following keys in that order: 39, 46, 49, 54, 82.





(i) Assume a B+-tree with order *d* = 1. Bulk load the tree with the following key values making sure that each leaf is full : 2, 4, 6, 8, 10, 12, 14, 16.


10
6
14
2
4
6
8
10
12
14
16