ΕΠΛ646 Προχωρημένα Θέματα Βάσεων Δεδομένων

Τμήμα Πληροφορικής
Πανεπιστήμιο Κύπρου

ΕΡΓΑΣΤΗΡΙΟ #4

ΜΕΡΟΣ Α΄: Tree Structured Indexing

1. **(Exercise 10.1)** Consider the B+ tree index of order $d = 2$ shown in Figure 10.1.
   i. Show the tree that would result from inserting a data entry with key 9 into this tree.
   ii. Show the B+ tree that would result from inserting a data entry with key 3 into the original tree. How many page reads and page writes does the insertion require?
   iii. Show the B+ tree that would result from deleting the data entry with key 8 from the original tree, assuming that the left sibling is checked for possible redistribution.
   iv. Show the B+ tree that would result from deleting the data entry with key 8 from the original tree, assuming that the right sibling is checked for possible redistribution.
   v. Show the B+ tree that would result from starting with the original tree, inserting a data entry with key 46 and then deleting the data entry with key 52.
   vi. Show the B+ tree that would result from deleting the data entry with key 91 from the original tree.
   vii. Show the B+ tree that would result from starting with the original tree, inserting a data entry with key 59, and then deleting the data entry with key 91.
   viii. Show the B+ tree that would result from successively deleting the data entries with keys 32, 39, 41, 45, and 73 from the original tree.

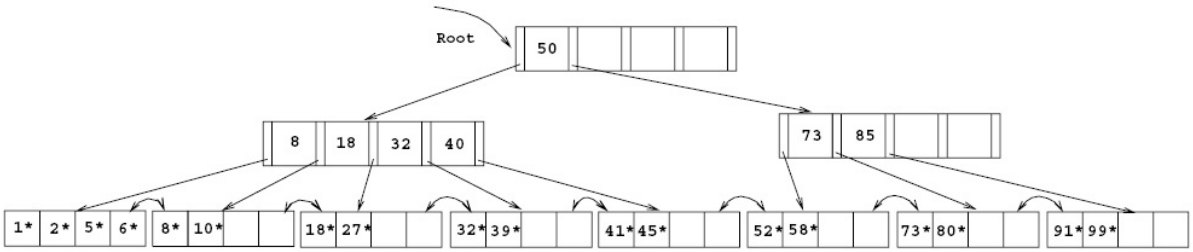

Figure 10.1 Tree for Exercise 10.1

***Answer**:*

i. H The data entry with key 9 is inserted on the second leaf page. The resulting tree is shown in figure 10.2

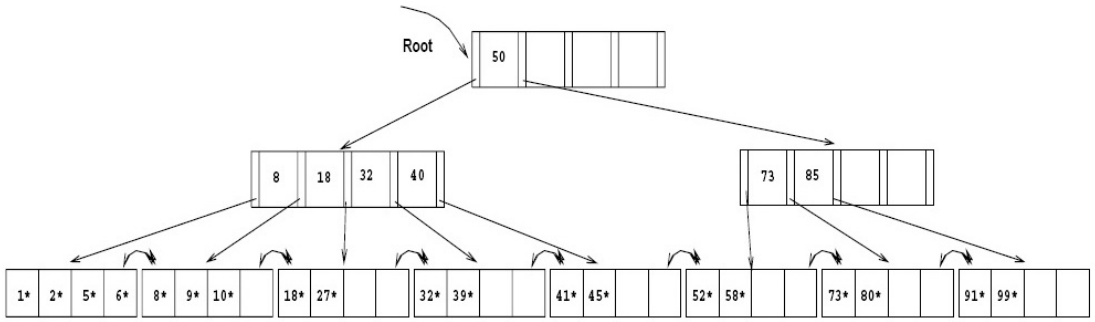

Figure 10.2

ii. The data entry with key 3 goes on the first leaf page F. Since F can accommodate at most four data entries (d = 2), F splits. The lowest data entry of the new leaf is given up to the ancestor which also splits. The result can be seen in figure 10.3. The insertion will require 5 page writes, 4 page reads and allocation of 2 new pages.

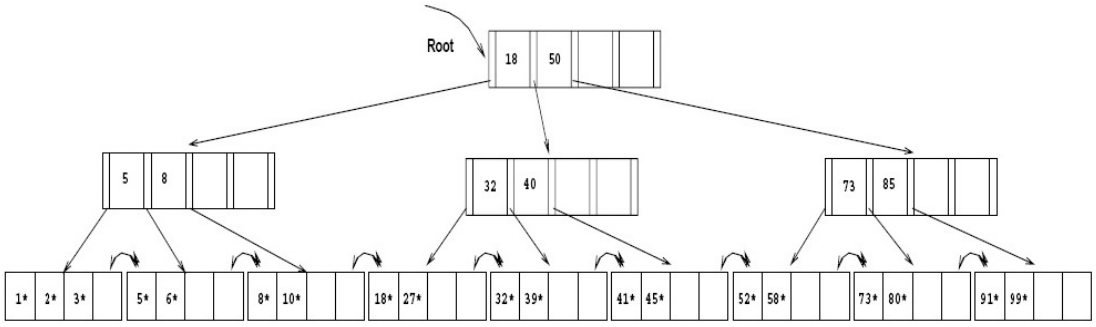

Figure 10.3

iii. The data entry with key 8 is deleted, resulting in a leaf page N with less than two data entries. The left sibling L is checked for redistribution. Since L has more than two data entries, the remaining keys are redistributed between L and N, resulting in the tree in figure 10.4.

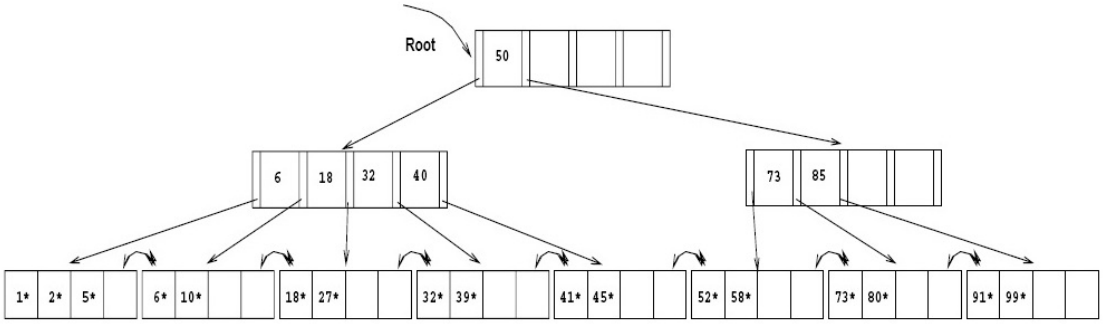

Figure 10.4

iv. As is part 3, the data entry with key 8 is deleted from the leaf page N. N's right sibling R is checked for redistribution, but R has the minimum number of keys. Therefore the two siblings merge. The key in the ancestor which distinguished between the newly merged leaves is deleted. The resulting tree is shown in figure 10.5.

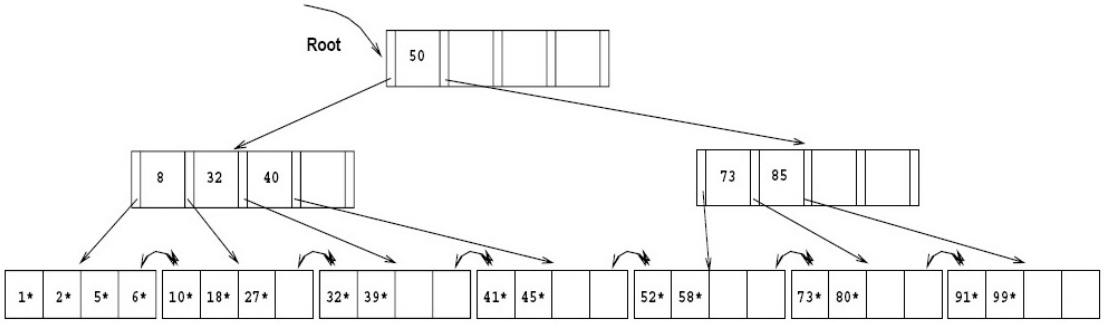

Figure 10.5

v. *The data entry with key 46 can be inserted without any structural changes in the tree. But the removal of the data entry with key 52 causes its leaf page L to merge with a sibling (we chose the right sibling). This results in the removal of a key in the ancestor A of L and thereby lowering the number of keys on A below the minimum number of keys. Since the left sibling B of A has more than the minimum number of keys, redistribution between A and B takes place. The final tree is depicted in figure 10.6.*

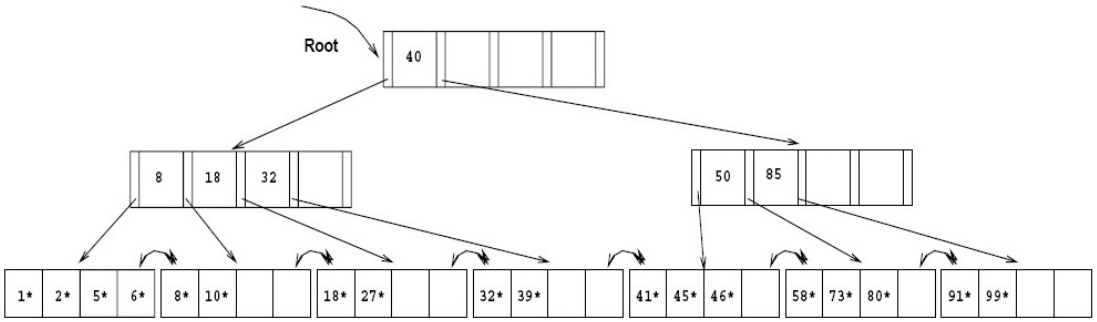

Figure 10.6

vi. *Deleting the data entry with key 91 causes a scenario similar to part 5. The result can be seen in figure 10.7.*

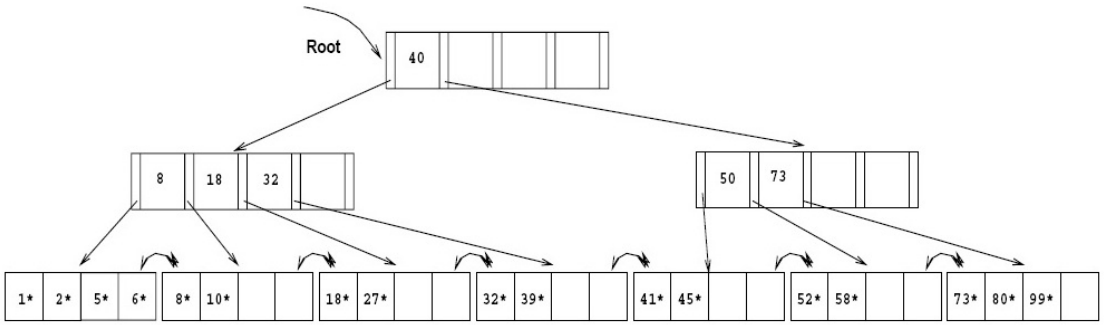

Figure 10.7

vii. *The data entry with key 59 can be inserted without any structural changes in the tree. No sibling of the leaf page with the data entry with key 91 is affected by the insert. Therefore deleting the data entry with key 91 changes the tree in a way very similar to part 6. The result is depicted in figure 10.8.*

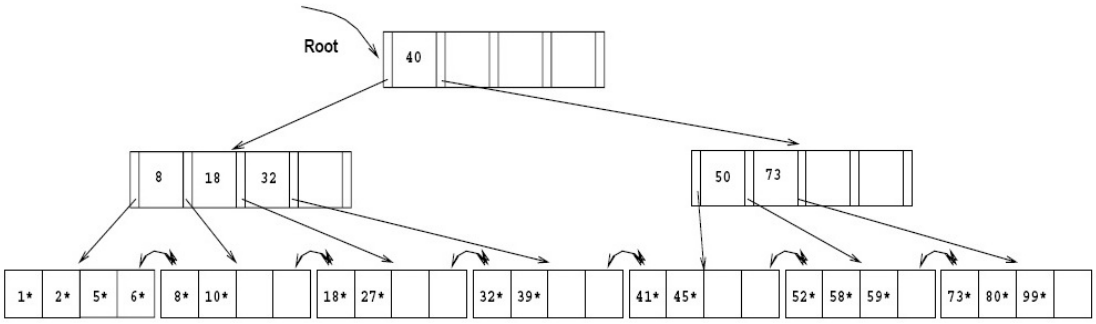

Figure 10.8

viii. Considering checking the right sibling for possible merging first, the successive deletion of the data entries with keys 32, 39, 41, 45 and 73 results in the tree shown in figure 10.9.

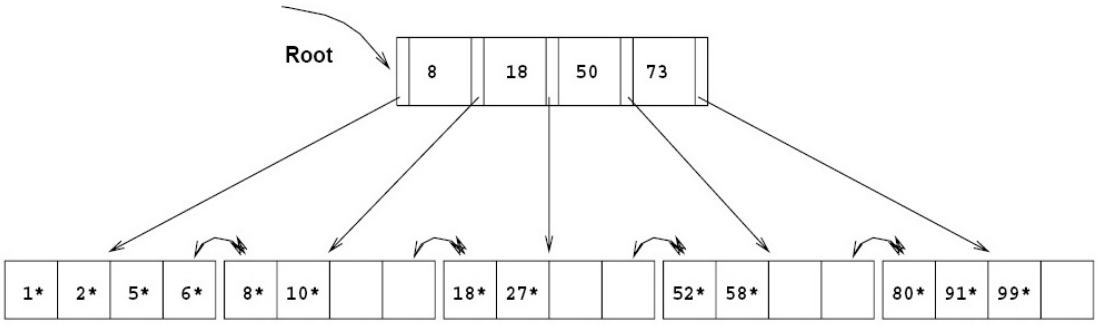

Figure 10.9

ΜΕΡΟΣ Β΄: Extendible Hashing

2. **(Exercise 11.1)** Consider the Extendible Hashing index shown in Figure 11.1. Answer the following questions about this index:
   i. What can you say about the last entry that was inserted into the index?
   ii. What can you say about the last entry that was inserted into the index if you know that there have been no deletions from this index so far?
   iii. Suppose you are told that there have been no deletions from this index so far. What can you say about the last entry whose insertion into the index caused a split?
   iv. Show the index after inserting an entry with hash value 68.
   v. Show the index after inserting entries with hash values 17 and 69 into the original tree.
   vi. Show the index after deleting the entry with hash value 21 into the original tree. (Assume that the full deletion algorithm is used.)
   vii. Show the index after deleting the entry with hash value 10 into the original tree. Is a merge triggered by this deletion? If not, explain why. (Assume that the full deletion algorithm is used.)

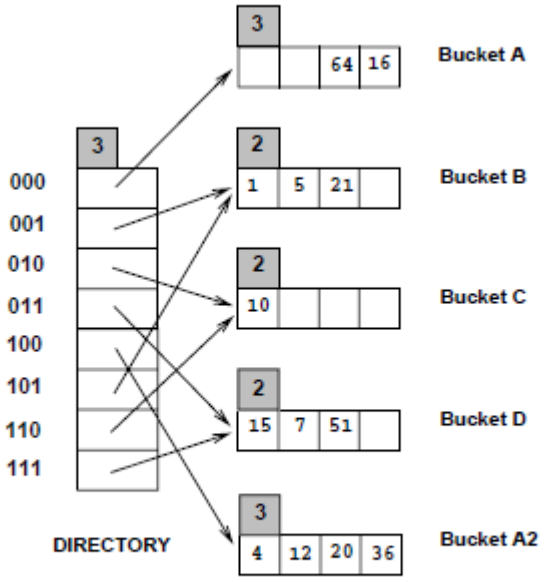

Figure 11.1 Figure for Exercise 11.1

***Answer**:*

i. *It could be any one of the data entries in the index. We can always find a sequence of insertions and deletions with a particular key value, among the key values shown in the index as the last insertion. For example, consider the data entry 16 and the following sequence:*

   1 5 21 10 15 7 51 4 12 36 64 8 24 56 16 56D 24D 8D

   The last insertion is the data entry 16 and it also causes a split. But the sequence of deletions following this insertion cause a merge leading to the index structure shown in Fig 11.1.

ii. *The last insertion could not have caused a split because the total number of data entries in the buckets A and A2 is 6. If the last entry caused a split the total would have been 5.*

iii. *The last insertion which caused a split cannot be in bucket C. Buckets B and C or C and D could have made a possible bucket-split image combination but the total number of data entries in these combinations is 4 and the absence of deletions demands a sum of at least 5 data entries for such combinations. Buckets B and D can form a possible bucket-split image combination because they have a total of 6 data entries between themselves. So do A and A2. But for the B and D to be split images the starting global depth should have been 1. If the starting global depth is 2, then the last insertion causing a split would be in A or A2*

iv. *See figure 11.2*

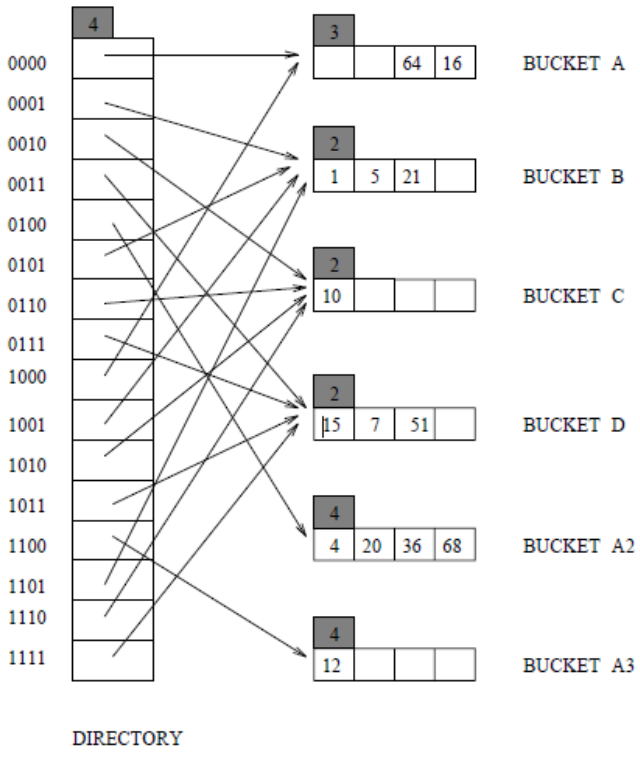

Figure 11.2

v. *See figure 11.3*

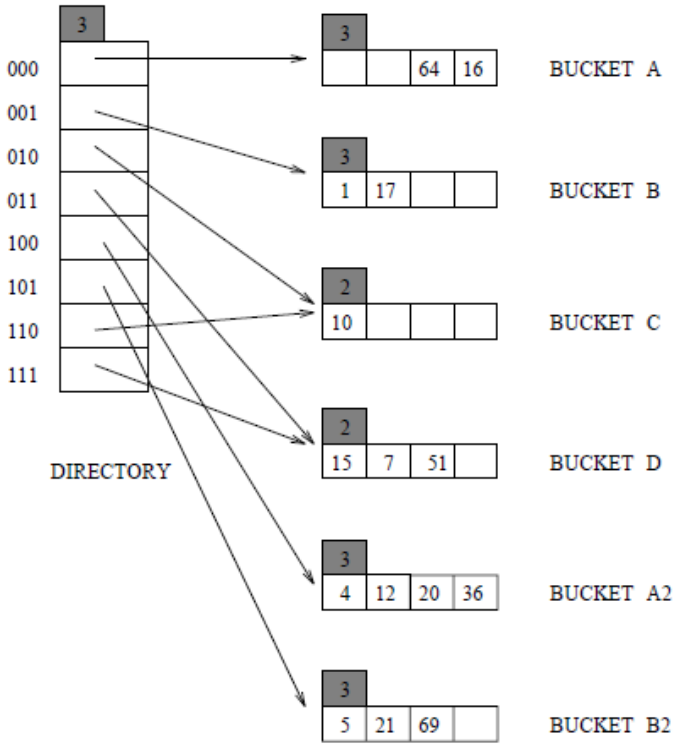

Figure 11.3

vi. *See figure 11.4*

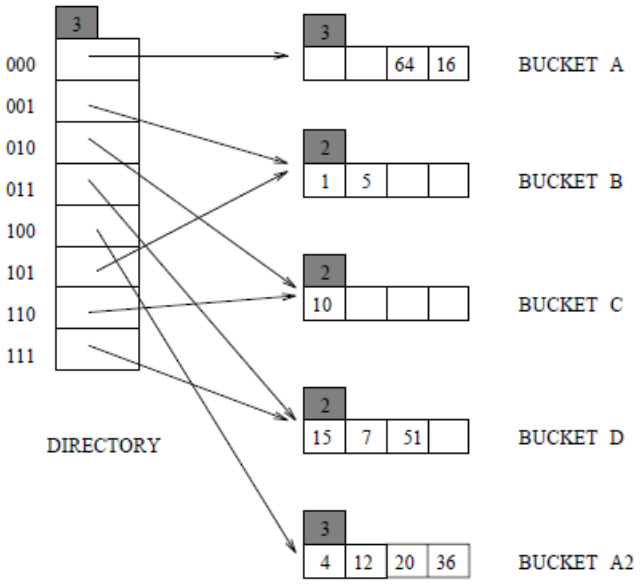

Figure 11.4

vii. The deletion of the data entry 10 which is the only data entry in bucket C doesn't trigger a merge because bucket C is a primary page and it is left as a place holder. Right now, directory element 010 and its split image 110 already point to the same bucket C. We can't do a further merge. See Fig 11.5.

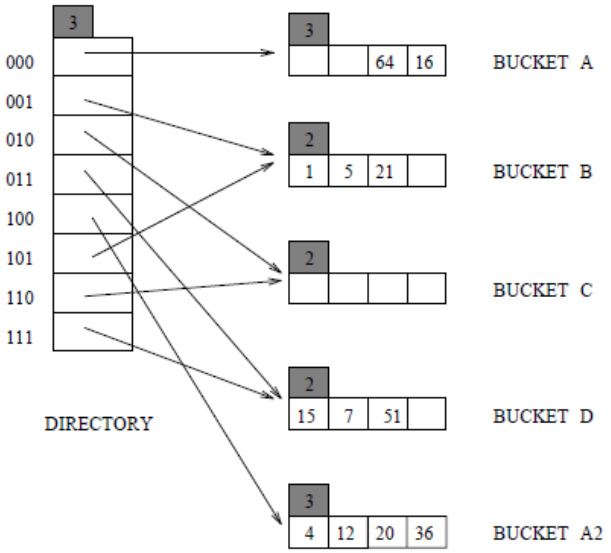

Figure 11.5