Storage and Indexing

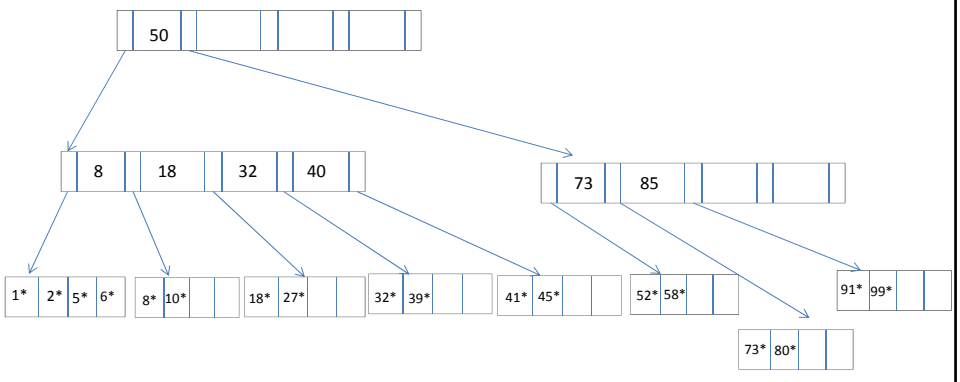

Figure 10.1

Exercise 10.1

- Consider the B+ tree index of order d=2 shown in figure 10.1
  1. Show the tree that would result from inserting a data entry with key 9 into this tree.

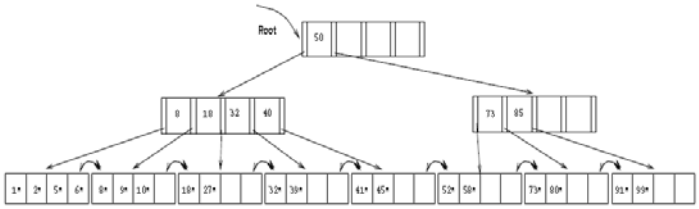

Figure 10.2

2. 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?

Ans:

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 Insertion requires 5 page writes, 4 page reads and allocation of 2 new pages.

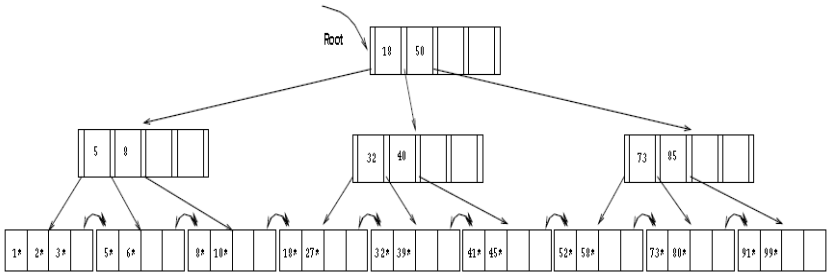

Figure 10.3

3. 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

Ans:

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.

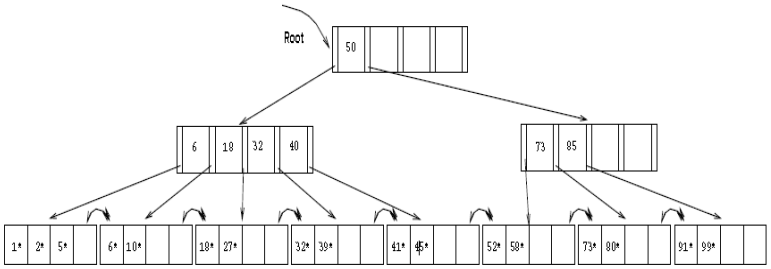

Figure 10.4

4. 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.

Ans:

As in Q3, the data entry with 8 is deleted from the leaf page N. N's right sibling R is checked for redistribution, but R has minimum number of keys. Therefore the two siblings merge. The key is the ancestor which distinguished between the newly merged leaves deleted.

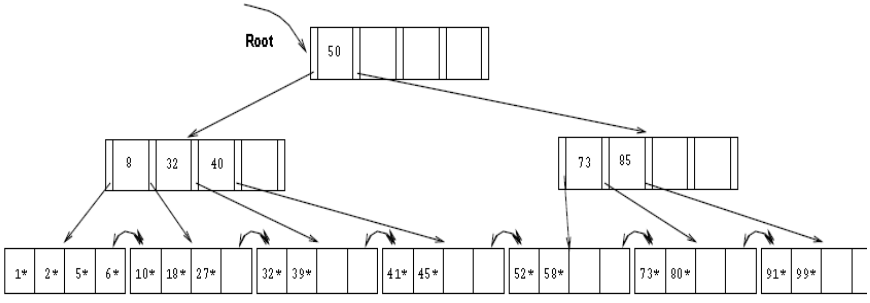

Figure 10.5

5. 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

Ans:

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 leaves 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.

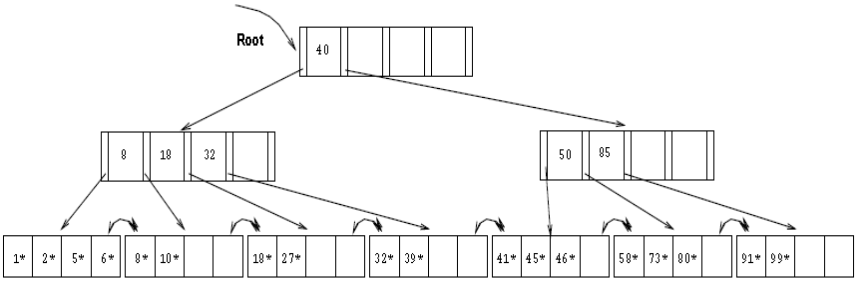

Figure 10.6

6. Show the B+ tree that would result from deleting the data entry with key 91 from the original tree

Ans:

Deleting the data entry with key 91 causes a scenario similar to Q5.

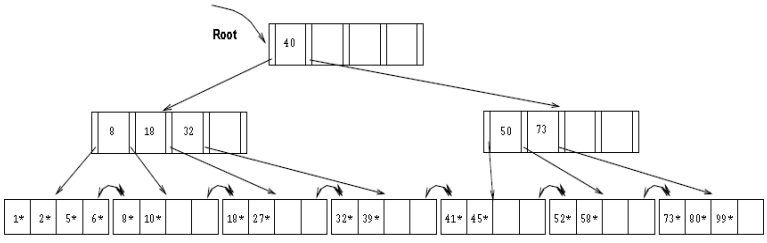

Figure 10.7

7. 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

Ans:

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 similar to part 6.

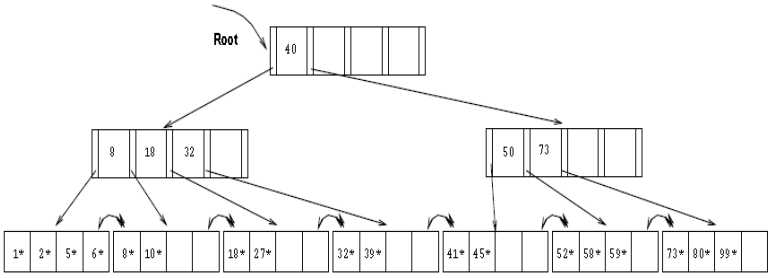

Figure 10.8

8. 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

Ans:

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 below

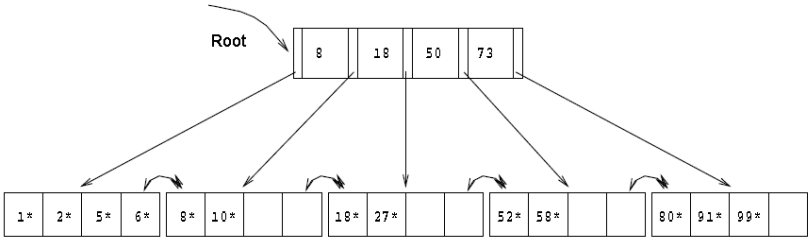

Figure 10.9

Exercise 10.10

- Consider the instance of the students relation shown in figure 10.22. Show a B+ tree of order 2 in each of these cases, assuming the duplicates are handled using overflow pages. Clearly indicate what the data entries are (i.e., do not use k* convention).

sid	*name*	*login*	*age*	*gpa*
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	3.8
53666	Jones	jones@cs	18	3.4
53901	Jones	jones@toy	18	3.4
53902	Jones	jones@physics	18	3.4
53903	Jones	jones@english	18	3.4
53904	Jones	jones@genetics	18	3.4
53905	Jones	jones@astro	18	3.4
53906	Jones	jones@chem	18	3.4
53902	Jones	jones@sanitation	18	3.8
53688	Smith	smith@ee	19	3.2
53650	Smith	smith@math	19	3.8
54001	Smith	smith@ee	19	3.5
54005	Smith	smith@cs	19	3.8
54009	Smith	smith@astro	19	2.2

Figure 10.22 An Instance of the Students Relation

1. A B+ tree index on age using Alternative (1) for data entries

Ans:

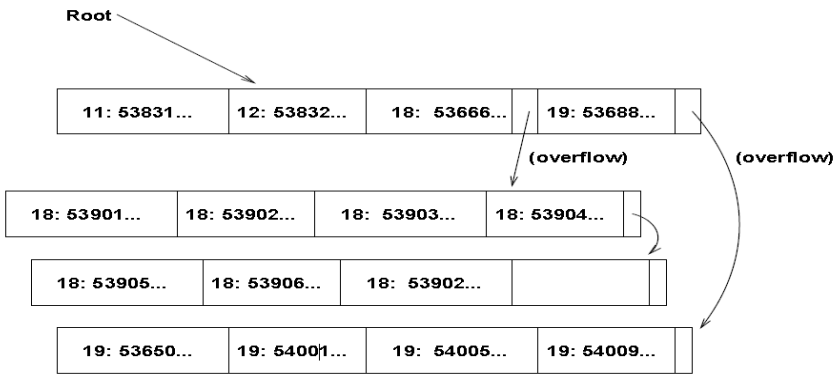

Figure 10.23

2. A dense B+ tree index on gpa using Alternative(2) for data entries. For this question, assume that these tuples are stored in a sorted file in the order shown in Figure 10.22. The first tuple is in page 1, slot 1; the second tuple in page 1, slot 2; and so on. Each page can store up to three data records. You can use <page-id, slot> to identify a tuple.

Ans:

See fig. 10.24. Note that the data entries are not necessarily stored in the same order as the data records, reflecting the fact that they may have been inserted in a different order. We assume a simple insertion algorithm that locates a leaf in the usual way, and if the leaf already contains a data entry with the given key value, put the new data entry into the overflow chain associated with the leaf. Thus, the data entries in a leaf have distinct key values. An obvious problem that arise here is that when the leaf splits (because a data entry with a new key value is inserted into the leaf when the leaf is full), the overflow chain must be scanned to ensure that when a data entry is moved to the new leaf node, all data entries with that key value are moved. An alternative is to maintain a separate overflow chain for each key value with duplicates, but considering the capacity of a page, and the likely number of duplicates for a given key value (probably low), this may lead to poor space utilization

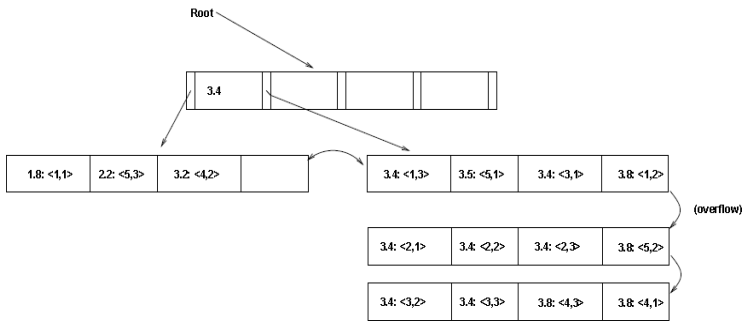

Figure 10.24

Exercise 11.1

- Consider the Extensible Hashing index shown in Figure 11.1. Answer the following questions about this index:

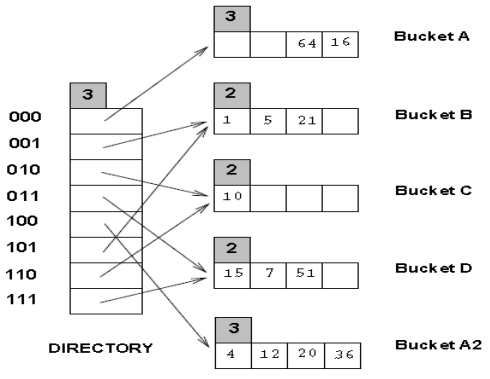

Figure 11.1 Figure for Exercise 11.1

1. What can you say about the last entry that was inserted into the index?

Ans:

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

2. 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

Ans:

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

3. Suppose you are told that there have been deletions from this index so far what can you say about the last entry whose insertion into the index caused a split

Ans:

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 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.

4. Show the index after inserting an entry with hash value 68

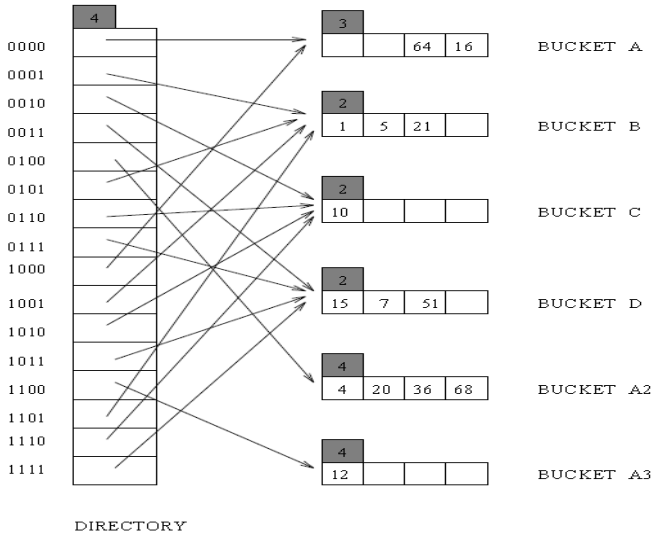

Figure 11.2

5. Show the index after inserting entries with hash value 17 and 69 into the original tree

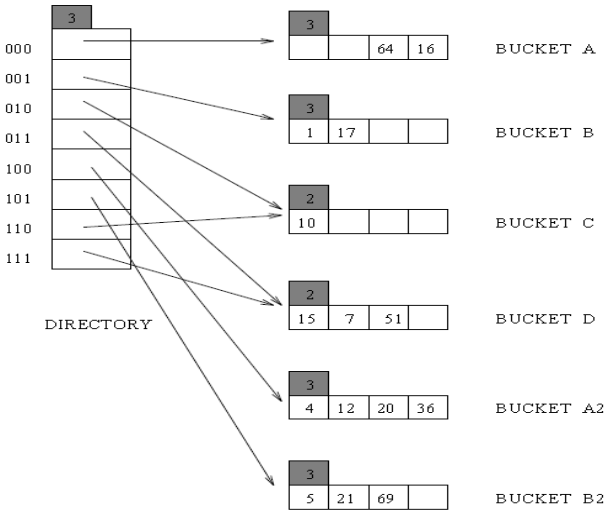

Figure 11.3

6. Show the index after deleting the entry with hash value 21 from the original tree. (Assume that the full deletion algorithm is used)

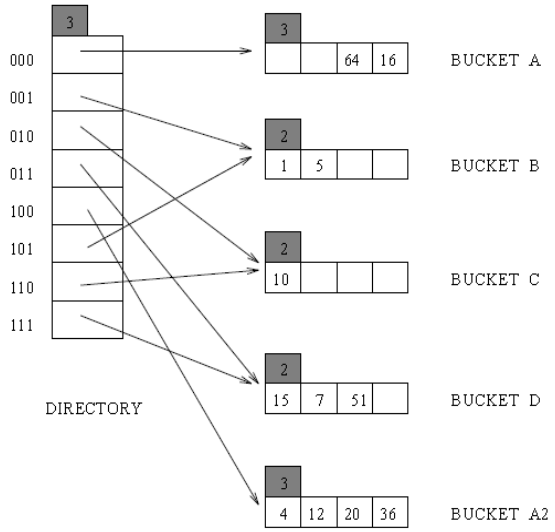

Figure 11.4

7. Show the index after deleting the entry with hash value 10 from the original tree. Is a merge triggered by this deletion? If not, explain why. (Assume that the full deletion algorithm is used.)

Ans:

The deletion of the data entry 10 which is the only 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.

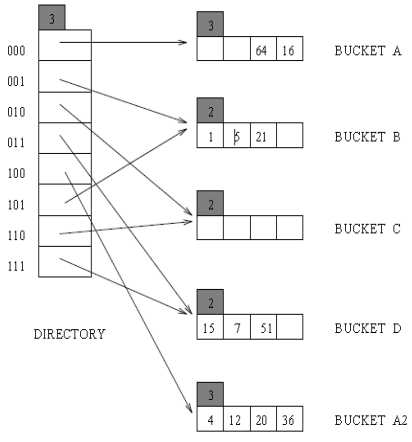

Figure 11.5

Exercise 11.7

- Consider a relation R(a, b, c, d) containing I million records, where each page of the relation holds 10 records. R is organized as a heap file with unclustered indexes, and the records in R are randomly ordered. Assume that attribute a is candidate key for R, with values lying in the range 0 to 999,999. For each of the following queries, name the approach that would most likely require the fewest I/Os for processing the query. The approaches to consider follow:
  - Scanning through the whole heap file for R.
  - Using a B+ tree index on attribute R.a
  - Using a hash index on attribute R.a

The queries are:

1. Find all R tuples.
2. Find all R tuples such that α<50
3. Find all R tuples such that α=50
4. Find all R tuples such α>50 and α>100

Ans:

Let h be the height of the B+ tree (usually 2 or 3) and M be the number of data entries per page (M>10). Let us assume that after accessing the data entry it takes one more disk access to get the actual record. Let c be the occupancy factor in hash indexing.

Consider the table shown below (disk accesses):

Problem	Heap File	B+ Tree	Hash Index
1. All tuples	10^5	$h + \frac{10^5}{M} + 10^6$	$\frac{10^5}{cM} + 10^6$
2. $a < 50$	10^5	$h + \frac{50}{M} + 50$	100
3. $a = 50$	10^5	$h + 1$	2
4. $a > 50$ and $a < 100$	10^5	$h + \frac{50}{M} + 49$	98

1. From the first row of the table, we see that heap file organization is the best (has the fewest disk accesses).
2. From the second row of the table, with typical value for h and M, the B+ Tree has the fewest disk accesses.
3. From the third row of the table, again we see that B+ Tree is the best.
4. From the fourth row or the table, again we see that B+ Tree is the best.