# EXERCICE 9.6

**Answer 3.6** 1. $1024/100 = 10$. We can have at most 10 records in a block.

2. There are 100,000 records all together, and each block holds 10 records. Thus, we need 10,000 blocks to store the file. One track has 25 blocks, one cylinder has 250 blocks. we need 10,000 blocks to store this file. So we will use more than one cylinders, that is, need 10 surfaces to store this file.

3. The capacity of the disk is 500,000K, which has 500,000 blocks. Each block has 10 records. Therefore, the disk can store no more than 5,000,000 records.

4. There are 25K bytes, or we can say, 25 blocks in each track. It is block 26 on block 1 of track 1 on the next disk surface.

   If the disk were capable of reading/writing from all heads in parallel, we can put the first 10 pages on the block 1 of track1 of 10 surfaces. Therefore, it is block 2 on block 1 of track 1 on the next disk surface.

5. A file containing 100,000 records of 100 bytes needs 40 cylinders or 400 tracks in this disk. The transfer time of one track of data is 0.011 seconds. Then it takes $400 \times 0.011 = 4.4 seconds$ to transfer 400 tracks.

   This access seeks the track 40 times. The seek time is $40 \times 0.01 = 0.4 seconds$. Therefore, total access time is $4.4 + 0.4 = 4.8 seconds$.

   If the disk were capable of reading/writing from all heads in parallel, the disk can read 10 racks at a time. The transfer time is 10 times less, which is 0.44 seconds. Thus total access time is $0.44 + 0.4 = 0.84 seconds$

6. For any block of data, $averageaccesstime = seektime + rotationaldelay + transfertime$.

$$seektime = 10msec$$

$$rotationaldelay = 6msec$$

$$transfertime = \frac{1K}{2,250K/sec} = 0.44msec$$

   The average access time for a block of data would be 16.44 msec. For a file containing 100,000 records of 100 bytes, the total access time would be 164.4 seconds.

# EX. 10.2

1. I1, I2, and everything in the range [L2..L8].

2. See Figure 5.11.

3. See Figure 5.12.

4. There are many search keys $X$ such that inserting $X$ would increase the height of the tree. Any search key in the range [65..79] would suffice. A key in this range would go in L5 if there were room for it, but since L5 is full already and since it can't redistribute any data entries over to L4 (L4 is full also), it must split; this in turn causes I2 to split, which causes I1 to split, and assuming I1 is the root node, a new root is created and the tree becomes taller.

5. We can infer several things about subtrees A, B, and C. First of all, they each must have height one, since their "sibling" trees (those rooted at I2 and I3) have height one. Also, we know the ranges of these trees (assuming duplicates fit on the same leaf): subtree A holds search keys less than 10, B contains keys $\geq 10$ and $< 20$, and C has keys $\geq 20$ and $< 30$. In addition, each intermediate node has at least 2 key values and 3 pointers.

6. The answers for the questions above would change as follows if we were dealing with ISAM trees instead of B+ trees.

   (a) This is only a search, so the answer is the same. (The tree structure is not modified.)

   (b) Because we can never split a node in ISAM, we must create an overflow page to hold inserted key 109.

   (c) Search key 81 would simply be erased from L6; no redistribution would occur (ISAM has no minimum occupation requirements).

(d) Being a *static* tree structure, an ISAM tree will never change height in normal operation, so there are no search keys which when inserted will increase the tree's height. (If we inserted an $X$ in [65..79] we would have to create an overflow page for L5.)

(e) We can infer several things about subtrees A, B, and C. First of all, they each must have height one, since their "sibling" trees (those rooted at I2 and I3) have height one. Here we suppose that we create a balanced ISAM tree. Also, we know the ranges of these trees (assuming duplicates fit on the same leaf): subtree A holds search keys less than 10, B contains keys $\geq 10$ and $< 20$, and C has keys $\geq 20$ and $< 30$. By the way, each of A, B, and C contain five leaf nodes (which may be of arbitrary fullness), and these nodes are the 15 consecutive pages prior to L1.

7. If this is an ISAM tree, we would have to insert at least nine search keys in order to develop an overflow chain of length three. These keys could be any that would map to L4, L5, L7, or L8, all of which are full and thus would need overflow pages on the next insertion. (Presumably four data entries would fit on a page, so nine data entries would be the minimum needed for three pages.)

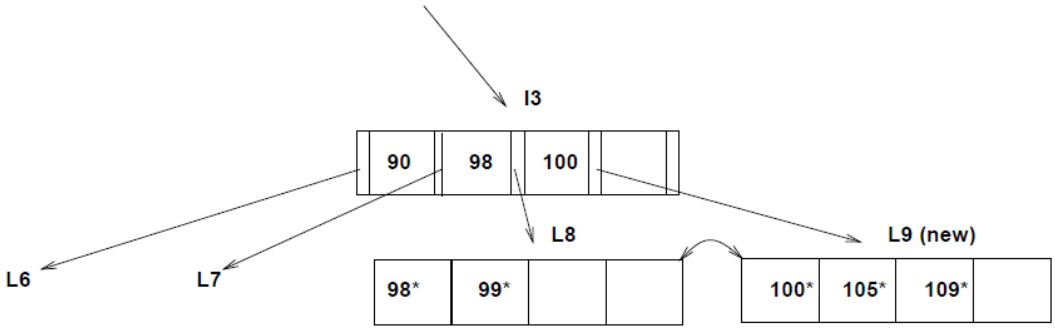


**Figure 5.11**

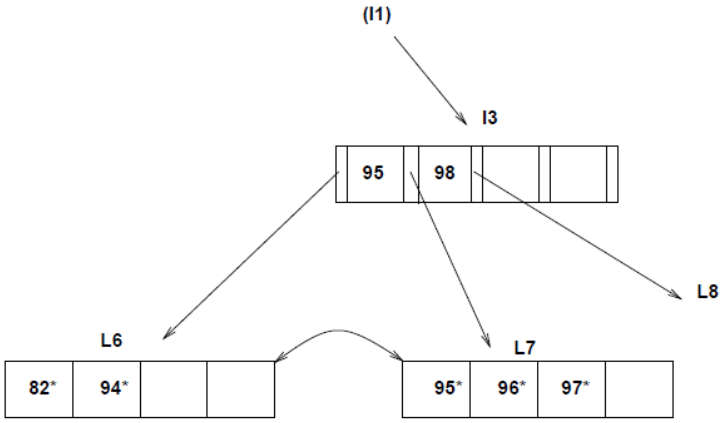


**Figure 5.12**

# EX 11.1

**Answer 6.1** 1. 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 $56_D$ $24_D$ $8_D$
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 6.1.

2. The last insertion could not have caused a split because the total number of data entries in the buckets $A$ and $A_2$ is 6. If the last entry caused a split the total would have been 5.

3. 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 atleast 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 $A_2$. 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 $A_2$.

4. See Fig 6.2.

5. See Fig 6.3.

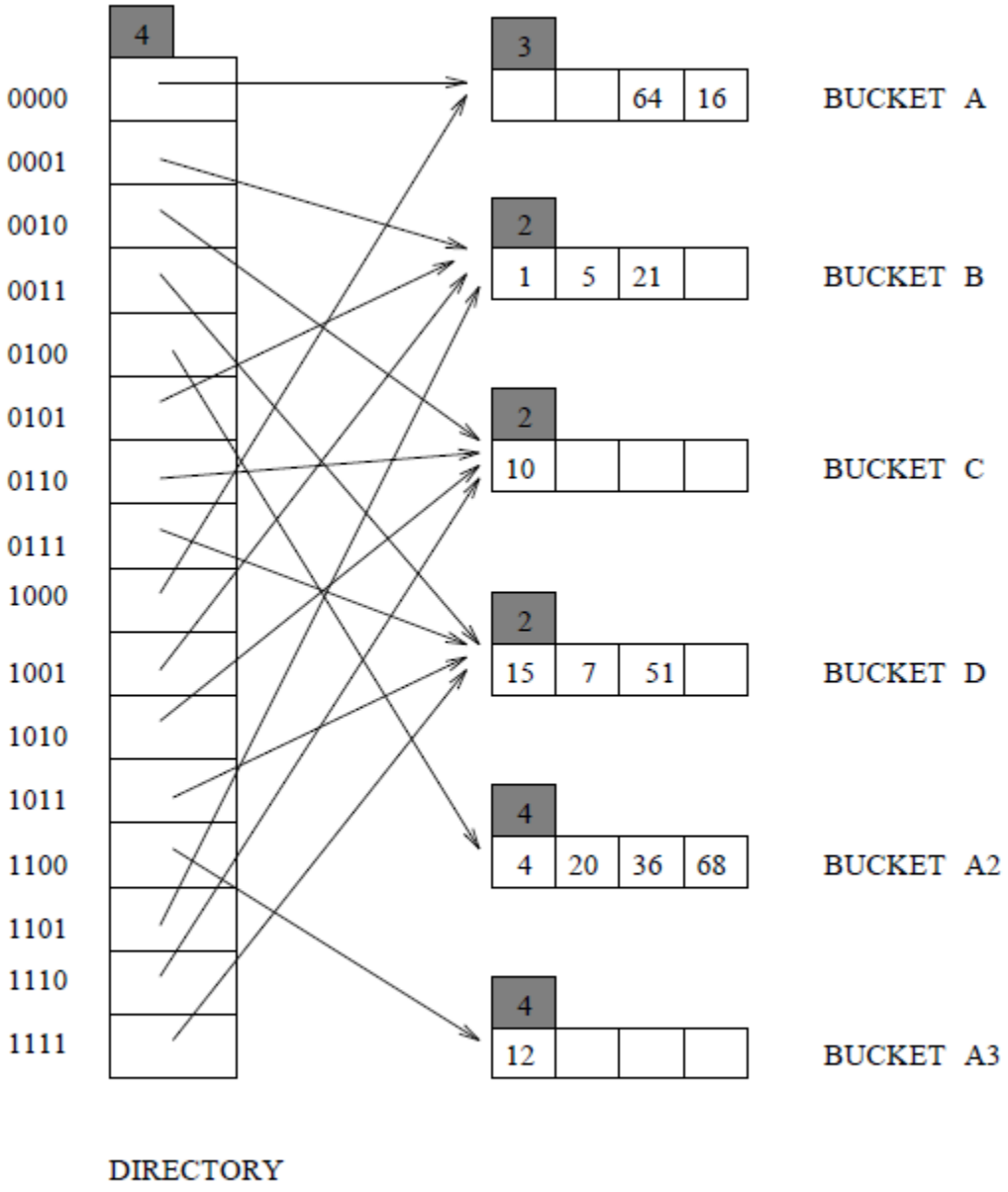
4
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
DIRECTORY
3
64
16
BUCKET A
2
1
5
21
BUCKET B
2
10
BUCKET C
2
15
7
51
BUCKET D
4
4
20
36
68
BUCKET A2
4
12
BUCKET A3

**Figure 6.2**

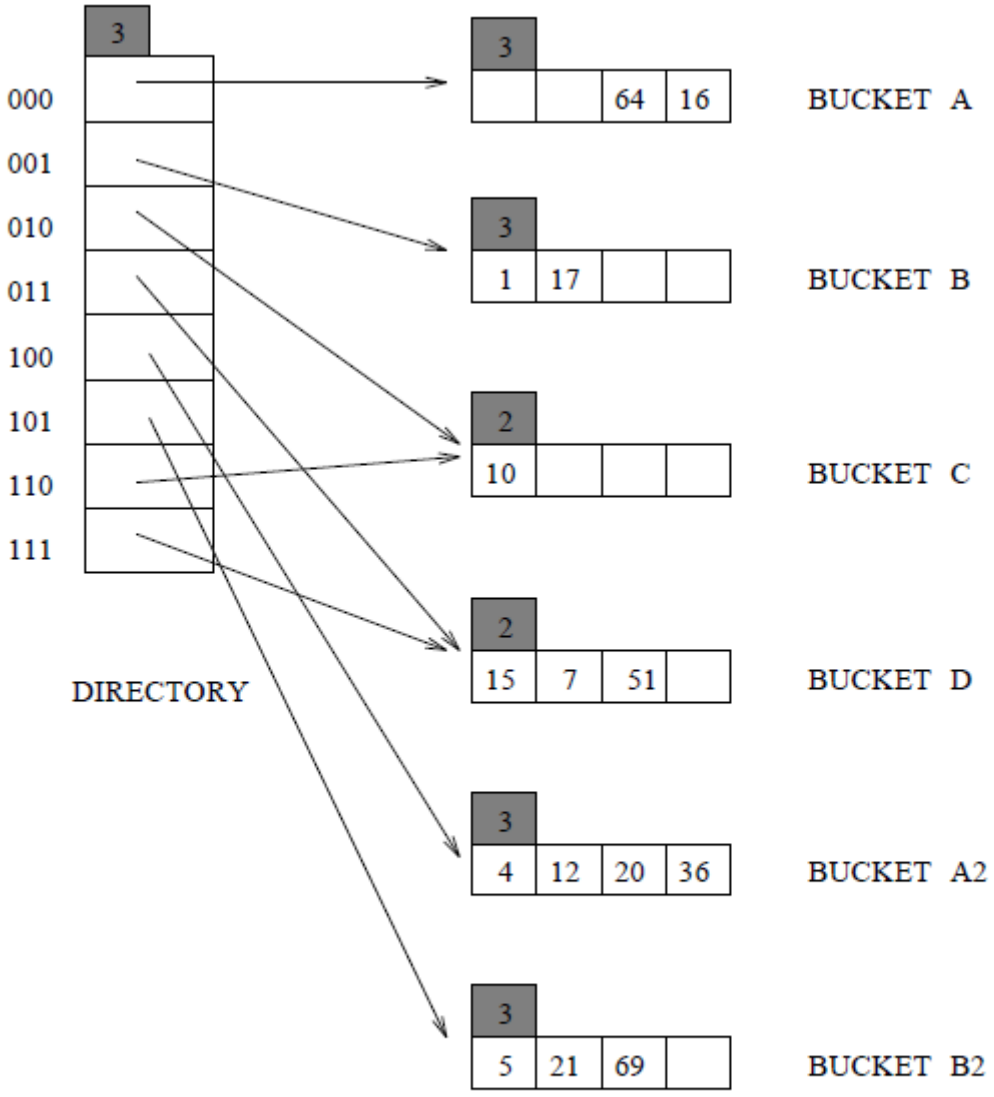
3
000
001
010
011
100
101
110
111
DIRECTORY
3
64
16
BUCKET A
3
1
17
BUCKET B
2
10
BUCKET C
2
15
7
51
BUCKET D
3
4
12
20
36
BUCKET A2
3
5
21
69
BUCKET B2

**Figure 6.3**

## EX. 19.10

1. Candidate key(s): BD. The decomposition into BC and AD is unsatisfactory because it is lossy (the join of BC and AD is the cartesian product which could be much bigger than ABCD)

2. Candidate key(s): AB, BC. The decomposition into ACD and BC is lossless since ACD $\cap$ BC (which is C) $\rightarrow$ACD. The projection of the FD's on ACD include C $\rightarrow$D, C $\rightarrow$A (so C is a key for ACD) and the projection of FD on BC produces no nontrivial dependencies. In particular this is a BCNF decomposition (check that R is not!). However, it is not dependency preserving since the dependency AB $\rightarrow$C is not preserved. So to enforce preservation of this dependency (if we do not want to use a join) we need to add ABC which introduces redundancy. So implicitly there is some redundancy across relations (although none inside ACD and BC).

3. Candidate key(s): A, C Since A and C are both candidate keys for R, it is already in BCNF. So from a normalization standpoint it makes no sense to decompose R further.

4. Candidate key(s): A The projection of the dependencies on AB are: A $\rightarrow$B and those on ACD are: A $\rightarrow$C and C $\rightarrow$D (rest follow from these). The scheme ACD is not even in 3NF, since C is not a superkey, and D is not part of a key. This is a lossless-join decomposition (since A is a key), but not dependency preserving, since B $\rightarrow$C is not preserved.

5. Candidate key(s): A (just as before) This is a lossless BCNF decomposition (easy to check!) This is, however, not dependency preserving (B consider $\rightarrow$C). So it is not free of (implied) redundancy. This is not the best decomposition ( the decomposition AB, BC, CD is better.)