**CHAPTER 14: INDEXING STRUCTURES FOR FILES**

**Answers to Selected Exercises**

14.14 Consider a disk with block size B=512 bytes. A block pointer is P=6 bytes long,

and a record pointer is P R =7 bytes long. A file has r=30,000 EMPLOYEE records

of fixed-length. Each record has the following fields: NAME (30 bytes), SSN (9

bytes), DEPARTMENTCODE (9 bytes), ADDRESS (40 bytes), PHONE (9 bytes),

BIRTHDATE (8 bytes), SEX (1 byte), JOBCODE (4 bytes), SALARY (4 bytes, real

number). An additional byte is used as a deletion marker.

(a) Calculate the record size R in bytes.

(b) Calculate the blocking factor bfr and the number of file blocks b assuming an

unspanned organization.

(c) Suppose the file is ordered by the key field SSN and we want to construct a primary

index on SSN. Calculate (i) the index blocking factor bfr i (which is also the index

fan-out fo); (ii) the number of first-level index entries and the number of first-level

index blocks; (iii) the number of levels needed if we make it into a multi-level

index; (iv) the total number of blocks required by the multi-level index; and

(v) the number of block accesses needed to search for and retrieve a record from

the file--given its SSN value--using the primary index.

(d) Suppose the file is not ordered by the key field SSN and we want to construct a

secondary index on SSN. Repeat the previous exercise (part c) for the secondary

index and compare with the primary index.

(e) Suppose the file is not ordered by the non-key field DEPARTMENTCODE and we want

to construct a secondary index on SSN using Option 3 of Section 14.1.3, with an extra

level of indirection that stores record pointers. Assume there are 1000 distinct

values of DEPARTMENTCODE, and that the EMPLOYEE records are evenly distributed

among these values. Calculate (i) the index blocking factor bfr i (which is also the

index fan-out fo); (ii) the number of blocks needed by the level of indirection that

stores record pointers; (iii) the number of first-level index entries and the

number of first-level index blocks; (iv) the number of levels needed if we make it a

multi-level index; (v) the total number of blocks required by the multi-level index

and the blocks used in the extra level of indirection; and (vi) the approximate

number of block accesses needed to search for and retrieve all records in the file

having a specific DEPARTMENTCODE value using the index.

(f) Suppose the file is ordered by the non-key field DEPARTMENTCODE and we want to

construct a clustering index on DEPARTMENTCODE that uses block anchors (every

new value of DEPARTMENTCODE starts at the beginning of a new block). Assume

there are 1000 distinct values of DEPARTMENTCODE, and that the EMPLOYEE

records are evenly distributed among these values. Calculate (i) the index blocking

factor bfr i (which is also the index fan-out fo); (ii) the number of first-level

index entries and the number of first-level index blocks; (iii) the number of levels

needed if we make it a multi-level index; (iv) the total number of blocks required

by the multi-level index; and (v) the number of block accesses needed to search for

and retrieve all records in the file having a specific DEPARTMENTCODE value using

the clustering index (assume that multiple blocks in a cluster are either contiguous

or linked by pointers).

(g) Suppose the file is not ordered by the key field Ssn and we want to construct a B + -

tree

access structure (index) on SSN. Calculate (i) the orders p and p leaf of the

B + -tree; (ii) the number of leaf-level blocks needed if blocks are approximately

69% full (rounded up for convenience); (iii) the number of levels needed if

internal nodes are also 69% full (rounded up for convenience); (iv) the total

number of blocks required by the B + -tree; and (v) the number of block accesses

needed to search for and retrieve a record from the file--given its SSN value--

using the B + -tree.

Answer:

(a) Record length R = (30 + 9 + 9 + 40 + 9 + 8 + 1 + 4 + 4) + 1 = 115 bytes

(b) Blocking factor bfr = floor(B/R) = floor(512/115) = 4 records per block

Number of blocks needed for file = ceiling(r/bfr) = ceiling(30000/4) = 7500

(c) i. Index record size R i = (V SSN + P) = (9 + 6) = 15 bytes

Index blocking factor bfr i = fo = floor(B/R i ) = floor(512/15) = 34

ii. Number of first-level index entries r 1 = number of file blocks b = 7500 entries

Number of first-level index blocks b 1 = ceiling(r 1 /bfr i ) = ceiling(7500/34)

= 221 blocks

iii. We can calculate the number of levels as follows:

Number of second-level index entries r 2 = number of first-level blocks b 1

= 221 entries

Number of second-level index blocks b 2 = ceiling(r 2 /bfr i ) = ceiling(221/34)

= 7 blocks

Number of third-level index entries r 3 = number of second-level index blocks b 2

= 7 entries

Number of third-level index blocks b 3 = ceiling(r 3 /bfr i ) = ceiling(7/34) = 1

Since the third level has only one block, it is the top index level.

Hence, the index has x = 3 levels

iv. Total number of blocks for the index b i = b 1 + b 2 + b 3 = 221 + 7 + 1

= 229 blocks

v. Number of block accesses to search for a record = x + 1 = 3 + 1 = 4

(d) i. the index record size would be V SSN + P R = 9 + 7 = 16 bytes.

Leaf-level ndex blocking factor bfr i = floor(B/R i ) = floor(512/16)

= 32 index records per block

However, for internal nodes, block pointers are always used so the fan-out for

internal nodes fo would still be 34 ( V SSN + P ).

ii. Number of first-level index entries r 1 = number of file records r = 30000

Number of first-level index blocks b 1 = ceiling(r 1 /bfr i ) = ceiling(30000/32)

= 938 blocks

iii. We can calculate the number of levels as follows:

Number of second-level index entries r 2 = number of first-level index blocks b 1

= 938 entries

Number of second-level index blocks b 2 = ceiling(r 2 /bfr i ) = ceiling(938/34)

= 28 blocks

Number of third-level index entries r 3 = number of second-level index blocks b 2

= 28 entries

Number of third-level index blocks b 3 = ceiling(r 3 /bfr i ) = ceiling(28/34) = 1

Since the third level has only one block, it is the top index level.

Hence, the index has x = 3 levels

iv. Total number of blocks for the index b i = b 1 + b 2 + b 3 = 938 + 28 + 1 = 987

v. Number of block accesses to search for a record = x + 1 = 3 + 1 = 4

(e) i. Index record size R i = (V DEPARTMENTCODE + P) = (9 + 6) = 15 bytes

Index blocking factor bfr i = (fan-out) fo = floor(B/R i ) = floor(512/15)

= 34 index records per block

ii. There are 1000 distinct values of DEPARTMENTCODE, so the average number of

records for each value is (r/1000) = (30000/1000) = 30

Since a record pointer size P R = 7 bytes, the number of bytes needed at the level

of indirection for each value of DEPARTMENTCODE is 7 \* 30 =210 bytes, which

fits in one block. Hence, 1000 blocks are needed for the level of indirection.

iii. Number of first-level index entries r 1

= number of distinct values of DEPARTMENTCODE = 1000 entries

Number of first-level index blocks b 1 = ceiling(r 1 /bfr i ) = ceiling(1000/34)

= 30 blocks

iv. We can calculate the number of levels as follows:

Number of second-level index entries r 2 = number of first-level index blocks b 1

= 30 entries

Number of second-level index blocks b 2 = ceiling(r 2 /bfr i ) = ceiling(30/34) = 1

Hence, the index has x = 2 levels

v. total number of blocks for the index b i = b 1 + b 2 + b indirection

= 30 + 1 + 1000 = 1031 blocks

vi. Number of block accesses to search for and retrieve the block containing the

record pointers at the level of indirection = x + 1 = 2 + 1 = 3 block accesses

If we assume that the 30 records are distributed over 30 distinct blocks, we need

an additional 30 block accesses to retrieve all 30 records. Hence, total block

accesses needed on average to retrieve all the records with a given value for

DEPARTMENTCODE = x + 1 + 30 = 33

(f) i. Index record size R i = (V DEPARTMENTCODE + P) = (9 + 6) = 15 bytes

Index blocking factor bfr i = (fan-out) fo = floor(B/R i ) = floor(512/15)

= 34 index records per block

ii. Number of first-level index entries r 1

= number of distinct DEPARTMENTCODE values= 1000 entries

Number of first-level index blocks b 1 = ceiling(r 1 /bfr i )

= ceiling(1000/34) = 30 blocks

iii. We can calculate the number of levels as follows:

Number of second-level index entries r 2 = number of first-level index blocks b 1

= 30 entries

Number of second-level index blocks b 2 = ceiling(r 2 /bfr i ) = ceiling(30/34) = 1

Since the second level has one block, it is the top index level.

Hence, the index has x = 2 levels

iv. Total number of blocks for the index b i = b 1 + b 2 = 30 + 1 = 31 blocks

v. Number of block accesses to search for the first block in the cluster of blocks

= x + 1 = 2 + 1 = 3

The 30 records are clustered in ceiling(30/bfr) = ceiling(30/4) = 8 blocks.

Hence, total block accesses needed on average to retrieve all the records with a given

DEPARTMENTCODE = x + 8 = 2 + 8 = 10 block accesses

(g) i. For a B + -tree of order p, the following inequality must be satisfied for each

internal tree node: (p \* P) + ((p - 1) \* V SSN ) < B, or

(p \* 6) + ((p - 1) \* 9) < 512, which gives 15p < 521, so p=34

For leaf nodes, assuming that record pointers are included in the leaf nodes, the

following inequality must be satisfied: (p leaf \* (V SSN +P R )) + P < B, or

(p leaf \* (9+7)) + 6 < 512, which gives 16p leaf < 506, so p leaf =31

ii. Assuming that nodes are 69% full on the average, the average number of key

values in a leaf node is 0.69\*p leaf = 0.69\*31 = 21.39. If we round this up for

convenience, we get 22 key values (and 22 record pointers) per leaf node. Since the

file has 30000 records and hence 30000 values of SSN, the number of leaf-level

nodes (blocks) needed is b 1 = ceiling(30000/22) = 1364 blocks

iii. We can calculate the number of levels as follows:

The average fan-out for the internal nodes (rounded up for convenience) is

fo = ceiling(0.69\*p) = ceiling(0.69\*34) = ceiling(23.46) = 24

number of second-level tree blocks b 2 = ceiling(b 1 /fo) = ceiling(1364/24)

= 57 blocks

number of third-level tree blocks b 3 = ceiling(b 2 /fo) = ceiling(57/24)= 3

number of fourth-level tree blocks b 4 = ceiling(b 3 /fo) = ceiling(3/24) = 1

Since the fourth level has only one block, the tree has x = 4 levels (counting the

leaf level). Note: We could use the formula:

x = ceiling(log fo (b 1 )) + 1 = ceiling(log 24 1364) + 1 = 3 + 1 = 4 levels

iv. total number of blocks for the tree b i = b 1 + b 2 + b 3 + b 4

= 1364 + 57 + 3 + 1 = 1425 blocks

v. number of block accesses to search for a record = x + 1 = 4 + 1 = 5

14.15 A PARTS file with Part# as key field includes records with the following Part#

values: 23, 65, 37, 60, 46, 92, 48, 71, 56, 59, 18, 21, 10, 74, 78, 15, 16,

20, 24, 28, 39, 43, 47, 50, 69, 75, 8, 49, 33, 38. Suppose the search field

values are inserted in the given order in a B + -tree of order p=4 and p leaf =3;

show how the tree will expand and what the final tree looks like.

Answer:

A B + -tree of order p=4 implies that each internal node in the tree (except possibly the

root) should have at least 2 keys (3 pointers) and at most 4 pointers. For p leaf =3, leaf

nodes must have at least 2 keys and at most 3 keys. The figure on page 50 shows how the

tree progresses as the keys are inserted. We will only show a new tree when insertion

causes a split of one of the leaf nodes, and then show how the split propagates up the tree.

Hence, step 1 below shows the tree after insertion of the first 3 keys 23, 65, and 37,

and before inserting 60 which causes overflow and splitting. The trees given below show

how the keys are inserted in order. Below, we give the keys inserted for each tree:

1 :23, 65, 37; 2:60; 3:46; 4:92; 5:48, 71; 6:56; 7:59, 18; 8:21; 9:10; 10:7 4 ;

11:78; 12:15; 13:16; 14:20; 15:24; 16:28, 39; 17:43, 47; 18:50, 69; 19:7 5 ;

20:8, 49, 33, 38;

14.17 Suppose the following search field values are deleted in the given order from the

B + -tree of Exercise 14.15, show how the tree will shrink and show the final tree.

The deleted values are: 65, 75, 43, 18, 20, 92, 59, 37.

Answer:

An important note about a deletion algorithm for a B + -tree is that deletion of a key value

from a leaf node will result in a reorganization of the tree if: (i) The leaf node is less

than half full; in this case, we will combine it with the next leaf node (other algorithms

combine it with either the next or the previous leaf nodes, or both), (ii) If the key value

deleted is the rightmost (last) value in the leaf node, in which case its value will appear

in an internal node; in this case, the key value to the left of the deleted key in the left

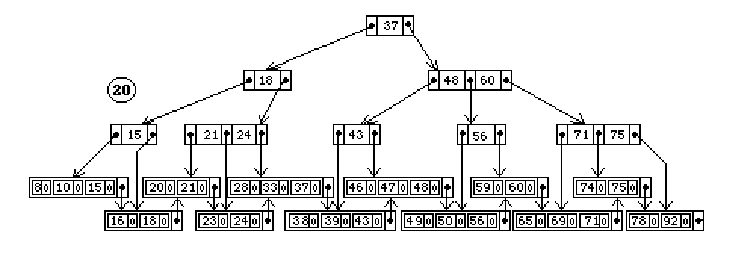
node replaces the deleted key value in the internal node. Following is what happens to the

tree number 19 after the specified deletions (not tree number 20):

Deleting 65 will only affect the leaf node. Deleting 75 will cause a leaf node to be less

than half full, so it is combined with the next node; also, 75 is removed from the

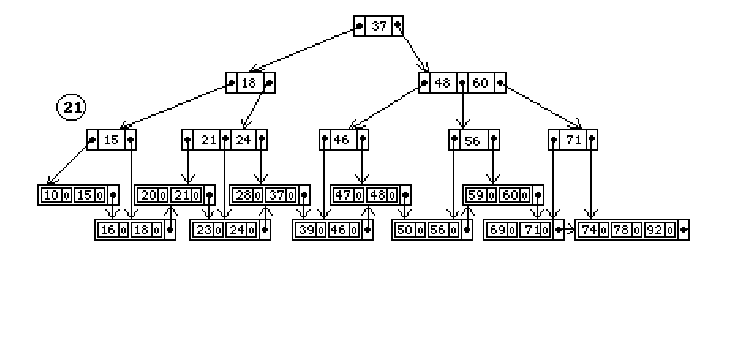
internal node leading to the following tree:



Deleting 43 causes a leaf node to be less than half full, and it is combined with the next

node. Since the next node has 3 entries, its rightmost (first) entry 46 can replace 43 in

both the leaf and internal nodes, leading to the following tree:



Next, we delete 18, which is a rightmost entry in a leaf node and hence appears in an

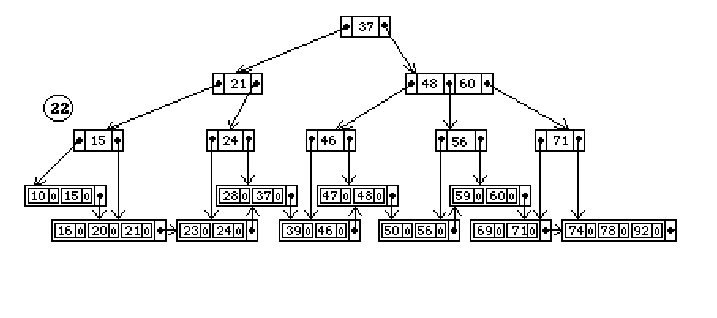
internal node of the B + -tree. The leaf node is now less than half full, and is combined

with the next node. The value 18 must also be removed from the internal node, causing

underflow in the internal node. One approach for dealing with underflow in internal

nodes is to reorganize the values of the underflow node with its child nodes, so 21 is

moved up into the underflow node leading to the following tree:



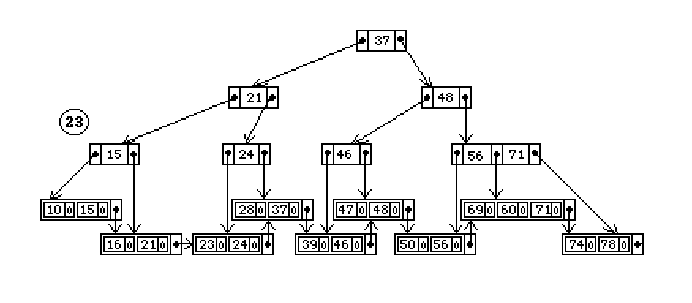
Deleting 20 and 92 will not cause underflow. Deleting 59 causes underflow, and the

remaining value 60 is combined with the next leaf node. Hence, 60 is no longer a

rightmost entry in a leaf node and must be removed from the internal node. This is

normally done by moving 56 up to replace 60 in the internal node, but since this leads to

underflow in the node that used to contain 56, the nodes can be reorganized as follows:



Finally, removing 37 causes serious underflow, leding to a reorganization of the whole

tree. One approach to deleting the value in the root node is to use the rightmost value in

the next leaf node (the first leaf node in the right subtree) to replace the root, and move

this leaf node to the left subtree. In this case, the resulting tree may look as follows:

