Solution Homework 2

17.18 Consider a disk with block size B = 512 bytes. A block pointer is P = 6 bytes long, and a record pointer is PR = 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), Department_code (9 bytes), Address (40 bytes), Phone (10 bytes), Birth_date (8 bytes), Sex (1 byte), Job_code (4 bytes), and Salary (4 bytes, real number). An additional byte is used as a deletion marker.

a. Calculate the record size R in bytes.
   Record length R = (30 + 9 + 9 + 40 + 9 + 8 + 1 + 4 + 4) + 1 = 115 bytes

b. Calculate the blocking factor bfr and the number of file blocks b, assuming an unspanned organization.
   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. Suppose that 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 bfri (which is also the index fan-out fo)
   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) The number of first-level index entries and the number of first-level index blocks
   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) The number of levels needed if we make it into a multilevel index
   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) The total number of blocks required by the multilevel index
   Total number of blocks for the index b_i = b 1 + b 2 + b 3 = 221 + 7 + 1 = 229 blocks
   (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
   Number of block accesses to search for a record = x + 1 = 3 + 1 = 4

d. Suppose that 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.

(i) Index record size Ri = (V SSN + 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
(This has not changed from part (c) above)

(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(30000/34) = 883 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= 883 entries
Number of second-level index blocks b_2 = ceiling (r_2 / bfr $_i$) = ceiling (883/34) = 26 blocks
Number of third-level index entries r_3 = number of second-level index blocks b_2 = 26 entries
Number of third-level index blocks b_3 = ceiling (r_3 / bfr $_i$) = ceiling (26/34) = 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 = 883 + 26 + 1 = 910

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

e. Suppose that the file is not ordered by the nonkey field Department_code and we want to construct a secondary index on Department_code, using option 3 of Section 17.1.3, with an extra level of indirection that stores record pointers. Assume there are 1,000 distinct values of Department_code and that the EMPLOYEE records are evenly distributed among these values. Calculate

(i) the index blocking factor bfri (which is also the index fan-out fo)
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) The number of blocks needed by the level of indirection that stores record pointers
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) the number of first-level index entries and the number of first-level index blocks

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) The number of levels needed if we make it into a multilevel index
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) The total number of blocks required by the multilevel index and the blocks used in the extra level of indirection
Total number of blocks for the index b_i = b_1 + b_2 + b indirection = 30 + 1 + 1000 = 1031 blocks

(vi) The approximate number of block accesses needed to search for and retrieve all records in the file that have a specific Department_code value, using the index.
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. Suppose that the file is ordered by the nonkey field Department_code and we want to construct a clustering index on Department_code that uses block anchors (every new value of Department_code starts at the beginning of a new block). Assume there are 1,000 distinct values of Department_code 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)
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) The number of first-level index entries and the number of first-level index blocks
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) The number of levels needed if we make it into a multilevel index
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) The total number of blocks required by the multilevel index
Total number of blocks for the index $b_i = b_1 + b_2 = 30 + 1 = 31$ blocks

(v) The number of block accesses needed to search for and retrieve all records in the file that have a specific Department_code value, using the clustering index (assume that multiple blocks in a cluster are contiguous).
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. Suppose that 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
Branch nodes will contain the Ssn search key and block pointers to other nodes, so p = [(B − P) / (V + P)] = [(512 B − 6 B) / (9 B+6 B)] = 34. Leaf nodes will have a record pointer for each key value (because the file is not ordered on that field) plus one block pointer to the next leaf node, so p_{leaf} = [(B − P) / (V + P_R)] = [(512 B − 6 B) / (9 B+7 B)] = 31.

(ii) The number of leaf-level blocks needed if blocks are approximately 69% full (rounded up for convenience)
There will be [0.69· p_{leaf}] = [0.69·31] = 22 search keys per leaf node. Since the file is not ordered on that field, there will have to be a search record for each record in the file, or 30,000 of them. So the number of leaf blocks is [30,000 / 22] = 1364 blocks. (The ceiling because the last block will be partially full.)

(iii) the number of levels needed if internal nodes are also 69% full (rounded up for convenience)
By the same reasoning there will be [0.69·p] = [0.69·34] = 24 block pointers per branch node. Then, one way to compute this is iteratively: If there are 1364 blocks in the first level of the tree, there must be 1364 block pointers in the second level. So there are [1364 / 24] = 57 blocks in the second level. Similarly, [57 / 24] = 3 blocks in the third level, and clearly 1 block in the fourth level. So x = 4.
Alternatively, use the logarithmic-height formula: x = [$\log_{fo}$ (b_1)] + 1 = [($\log_{24}$ (1364)] + 1 = 4.

(iv) The total number of blocks required by the B+-tree
1364 at the first level, 57 at the second, 3 at the third and 1 at the fourth. The sum is 1425 blocks

(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.
Four to traverse the height, plus one to access the full record in the primary file, so 5.

h. Repeat part g, but for a B-tree rather than for a B+-tree. Compare your results for the B-tree and for the B+-tree.

(i) the orders p and p_{leaf} of the B-tree
Key field = 9 bytes.
Block pointer = 6 bytes.
Record pointer = 7 bytes.
Let the order of the tree be p.
Then we have (p-1)*16 + p*6 <= 512
22p<=528
p<=24
p=24
Assuming that each node of the B- tree is 69% full, each node on the average will have P*0.69 = 24*0.69 = 16 nodes.
Therefore, the order of the tree is 16.

(ii) The number of leaf-level blocks needed if blocks are approximately 69% full (rounded up for convenience)

Root	1 node	15 entries	16 pointers
Level 1	16 nodes	16*15=240 entries	16*16= 256 pointers
Level 2	256 nodes	256*15 = 3840 entries	256*16 = 4096 pointers
Level 3	4096 nodes	4096*15 = 61440 entries	

Hence the number of leaf – level blocks needed = 4096

(iii) the number of levels needed if internal nodes are also 69% full (rounded up for convenience)
Using the logarithmic-height formula: $x = [\log_{fo} (b_1)] + 1 = [(\log_{16} (4096)] + 1 = 4$ (including the root).

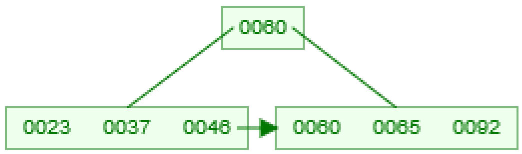
(iv) The total number of blocks required by the B-tree
1+ 16 + 256 + 4096 = 4369.

(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.
The number of block accesses needed = No. of levels of the tree + 1 (To access access the full record)
4 + 1 = 5

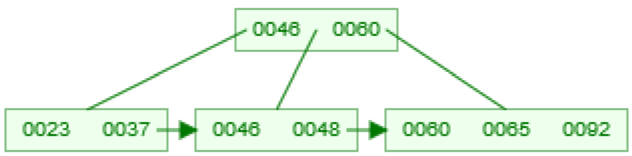
17.19 A PARTS file with Part# as the key field includes records with the following Part# values: 23, 65, 37, 60, 46, 92, 48, 71, 56, 59, 18, 21, 10, 74, 78, 15, 16, 20, 24, 28, 39, 43, 47, 50, 69, 75, 8, 49, 33, 38. Suppose that the search field values are inserted in the given order in a B+-tree of order p = 4 and pleaf = 3; show how the tree will expand and what the final tree will look like.

- Insert 23, 65, 37

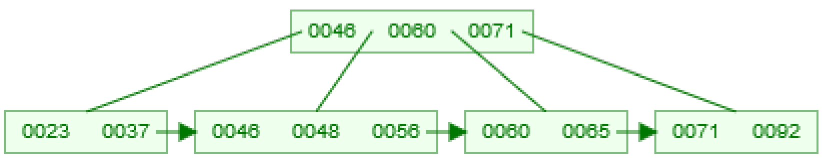
- Insert 60, 46, 92



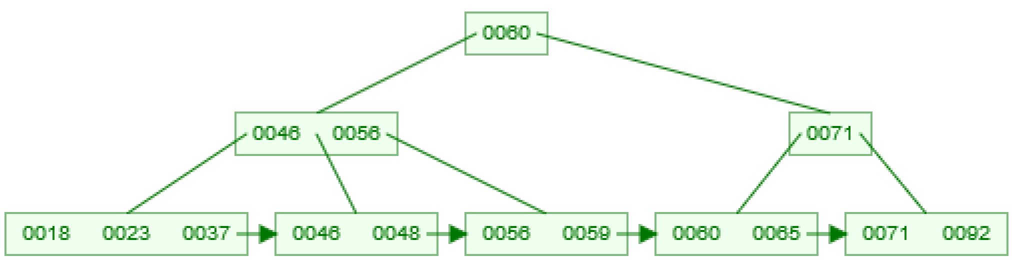
- Insert 48



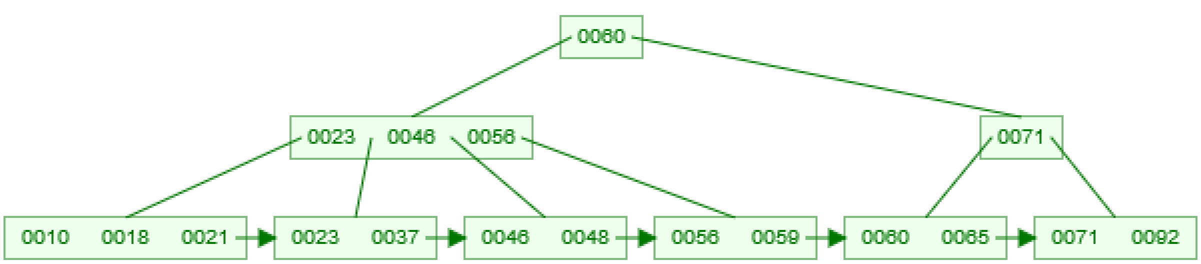
- Insert 71, 56



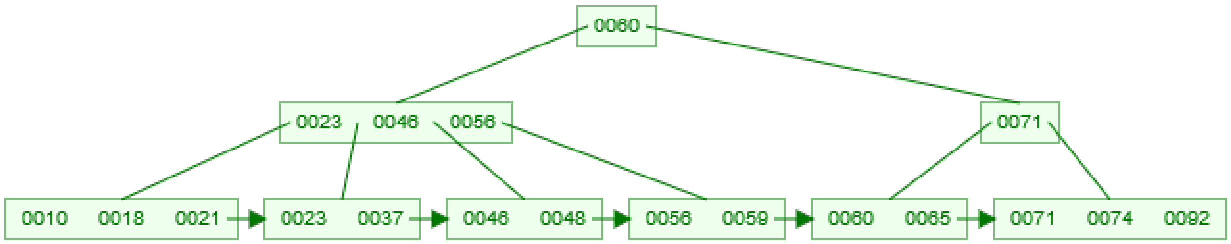
- Insert 59, 18



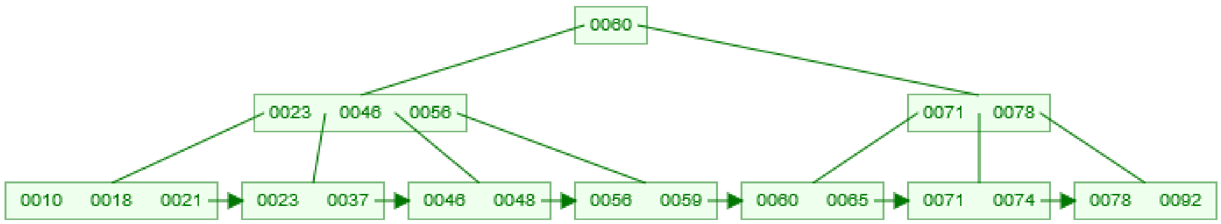
- Insert 21, 10



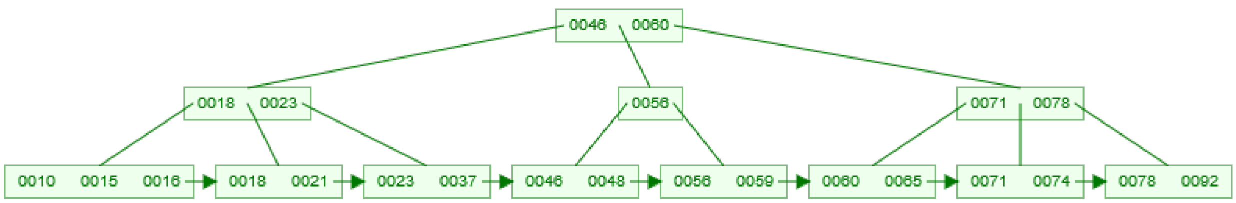
- Insert 74



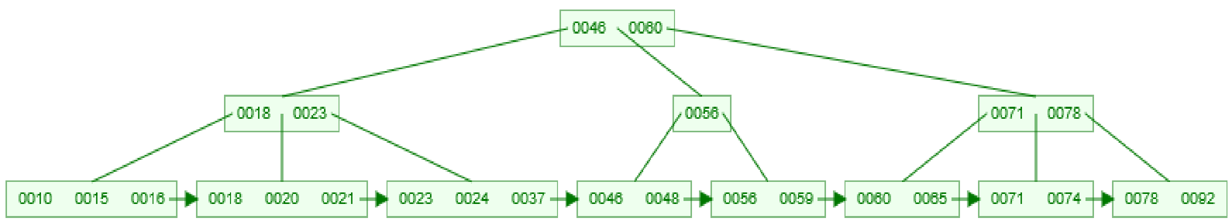
- Insert 78



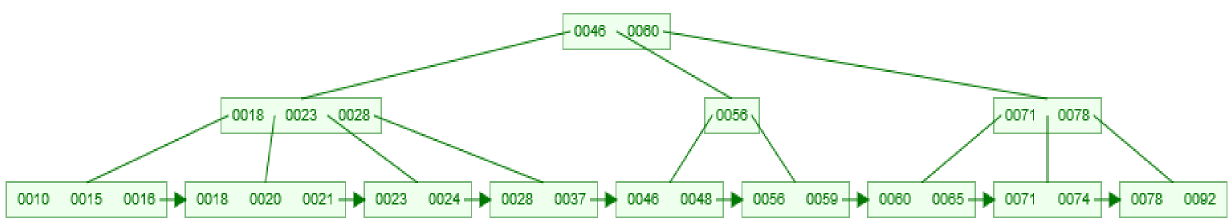
- Insert 15, 16



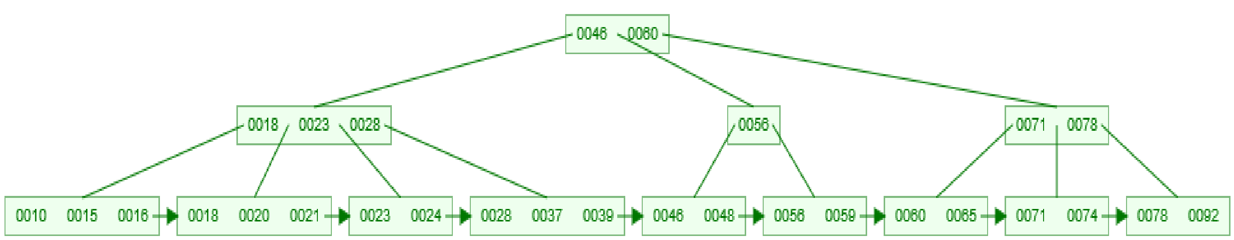
- Insert 20, 24



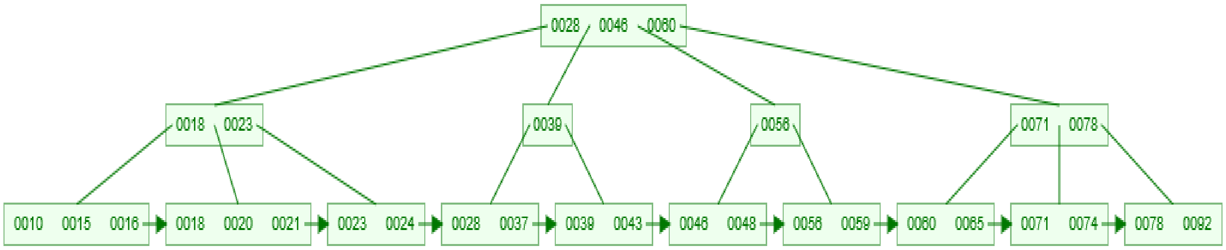
- Insert 28



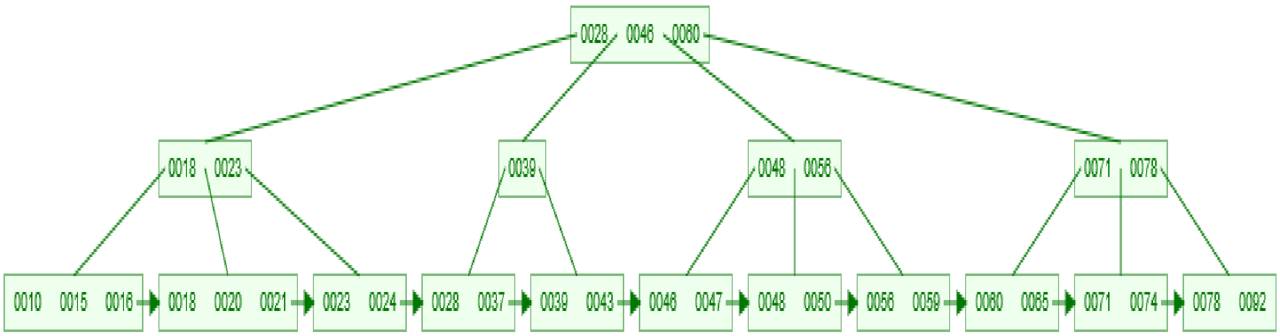
- Insert 39



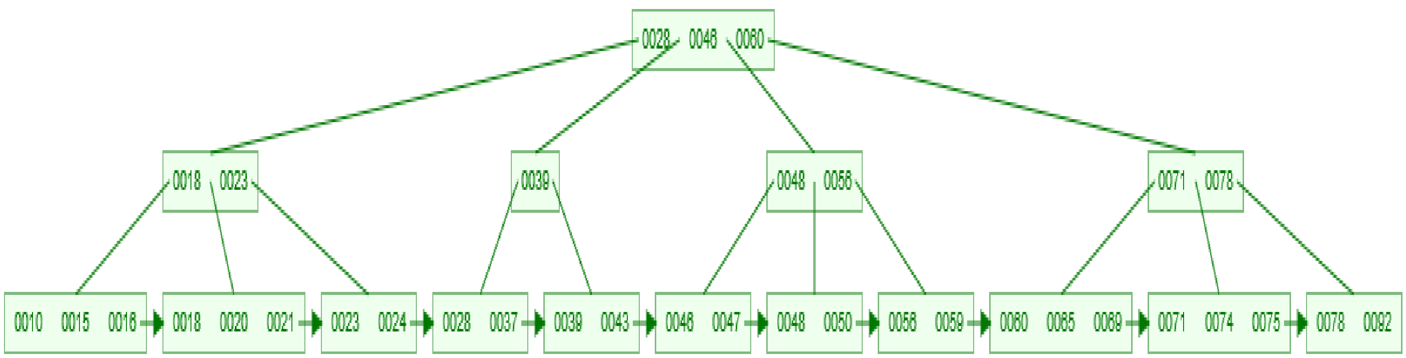
- Insert 43



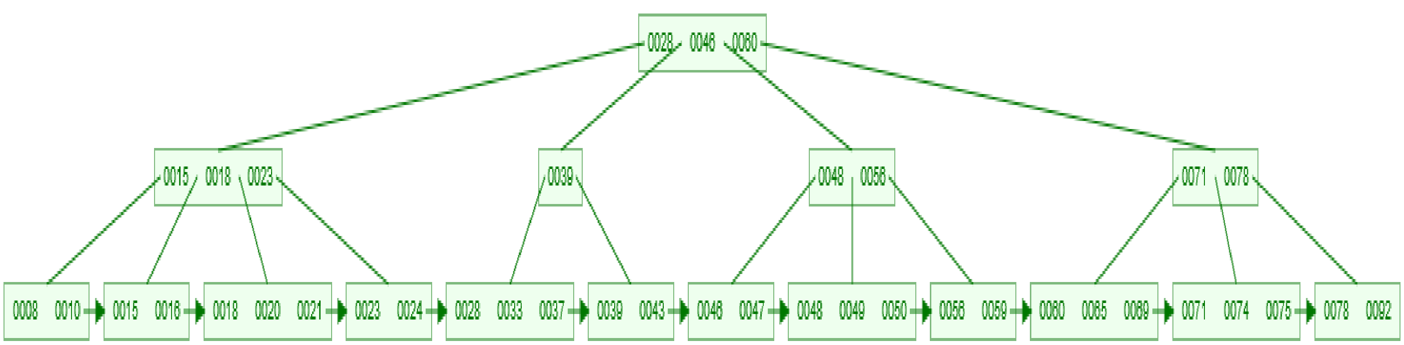
- Insert 47, 50



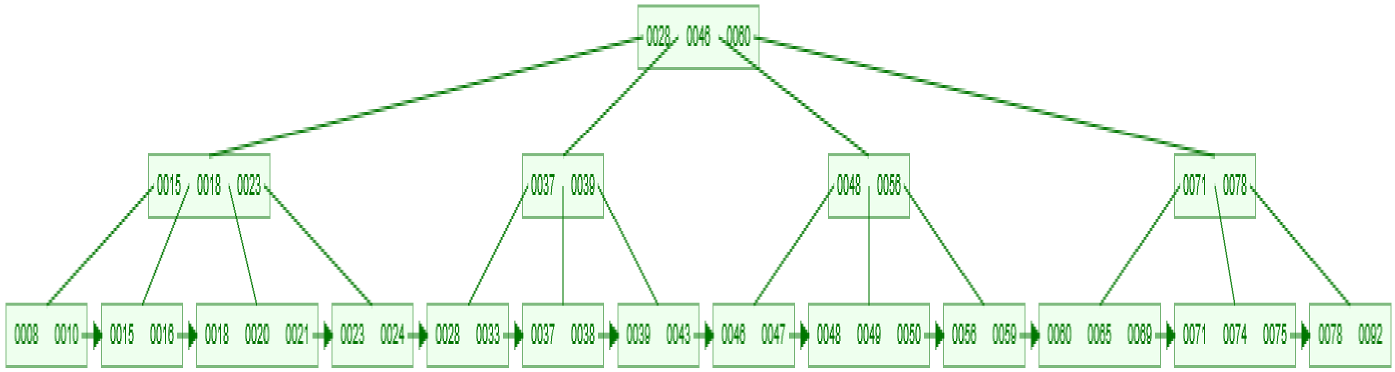
- Insert 69, 75



- Insert 8, 49, 33



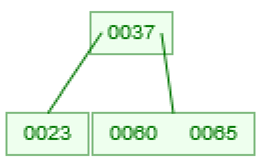
- Insert 38



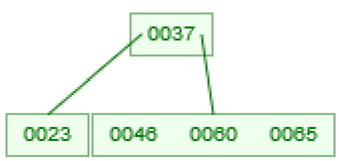
17.20 Repeat Exercise 17.19, but use a B-tree of order p = 4 instead of a B+-tree.

- Add 23, 65, 37

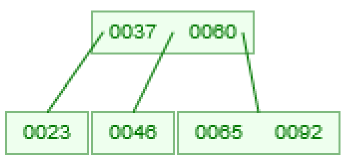
- Insert 60



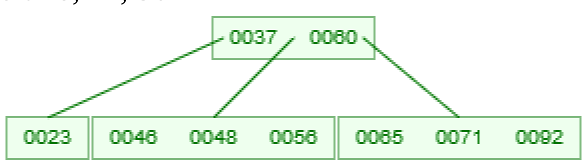
- Insert 46



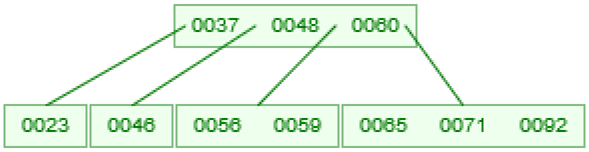
- Insert 92



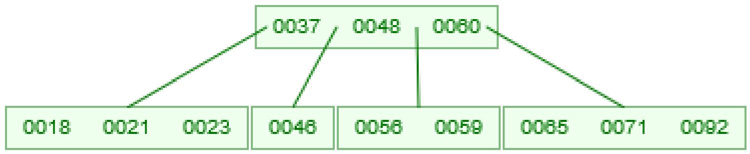
- Insert 48, 71, 56



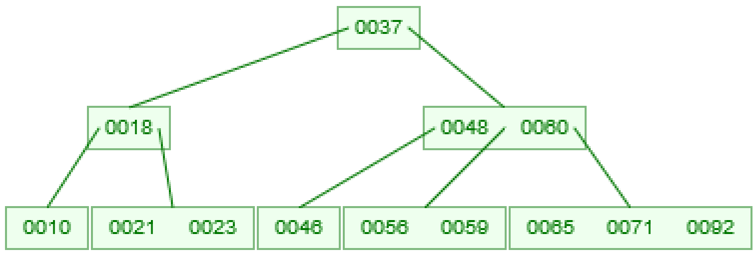
- Insert 59



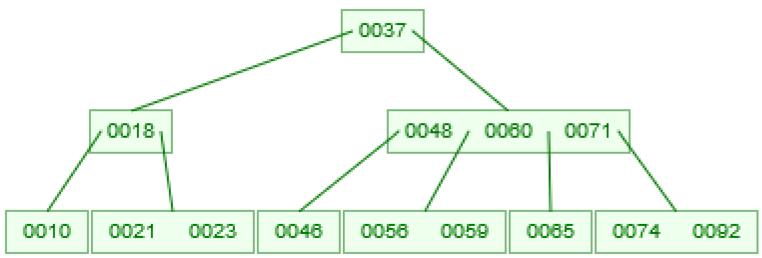
- Insert 18, 21



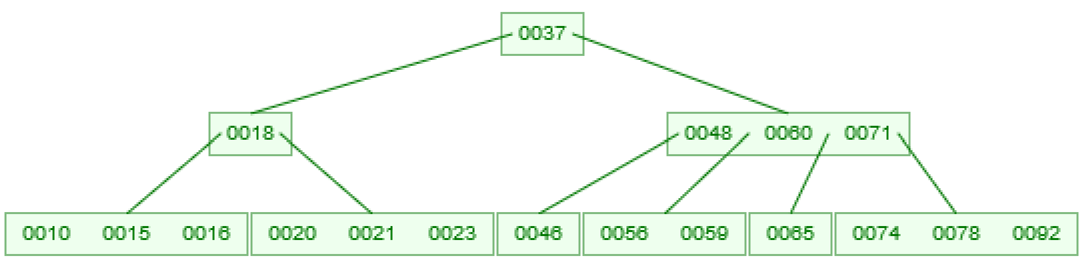
- Insert 10



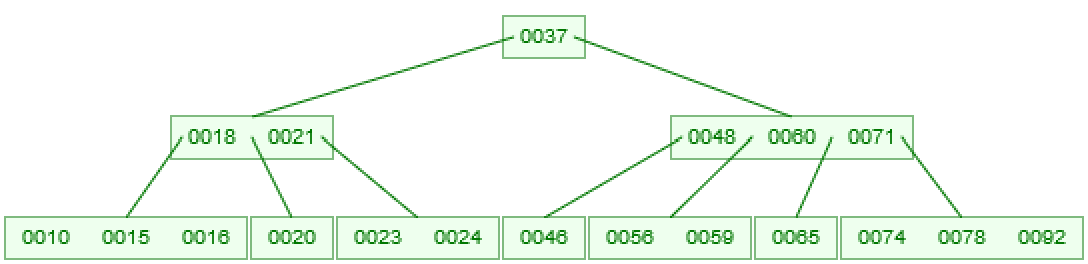
- Insert 74



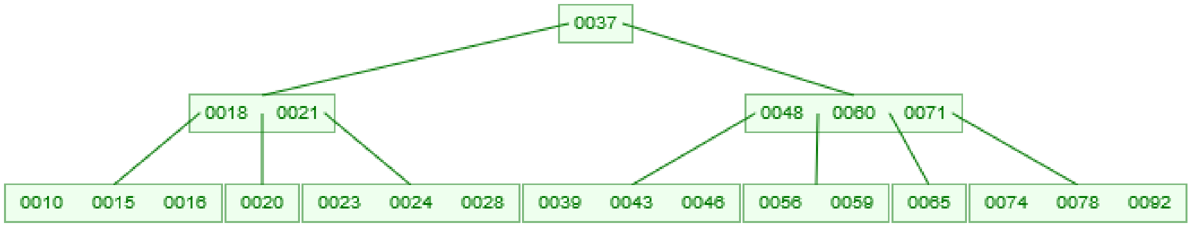
- Insert 78, 15, 16, 20



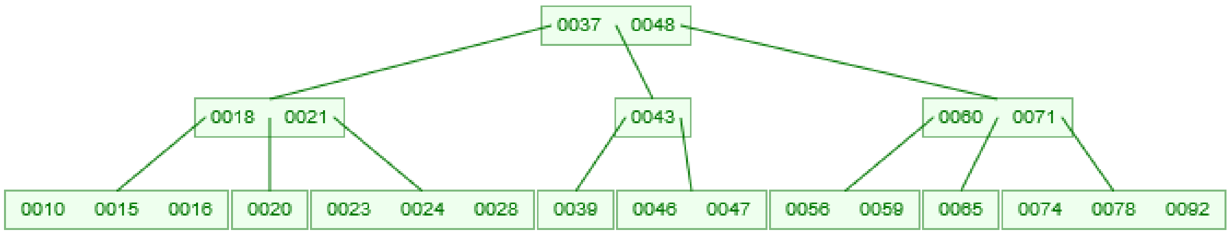
- Insert 24



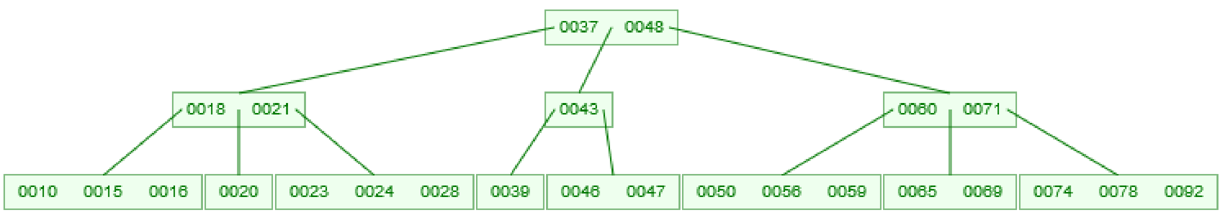
- Insert 28, 39, 43



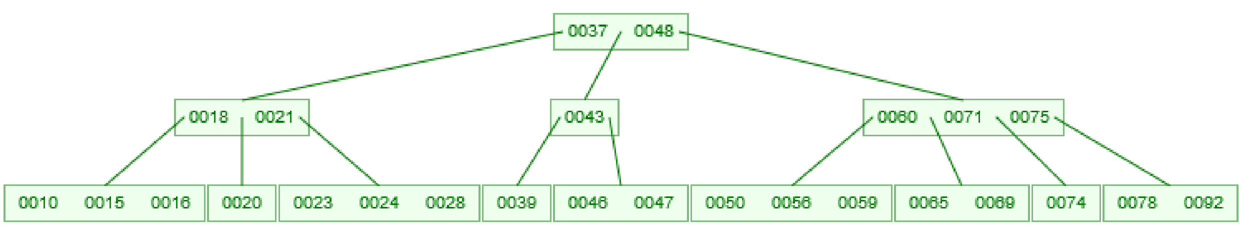
- Insert 47



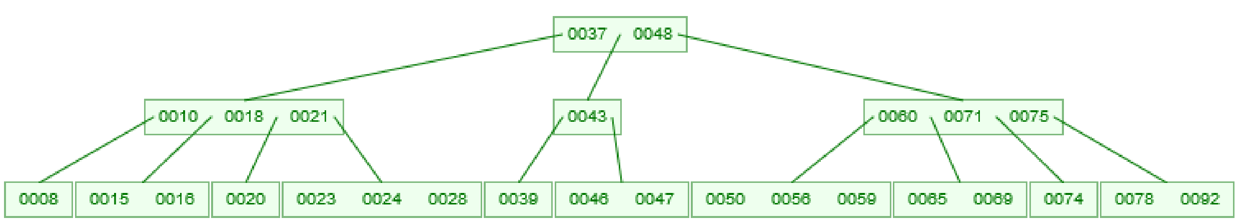
- Insert 50, 69



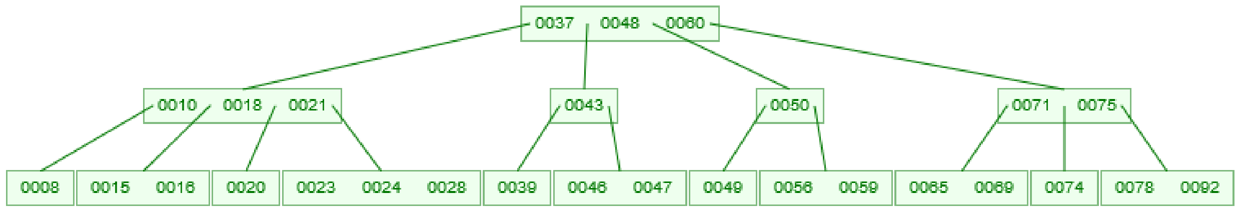
- Insert 75



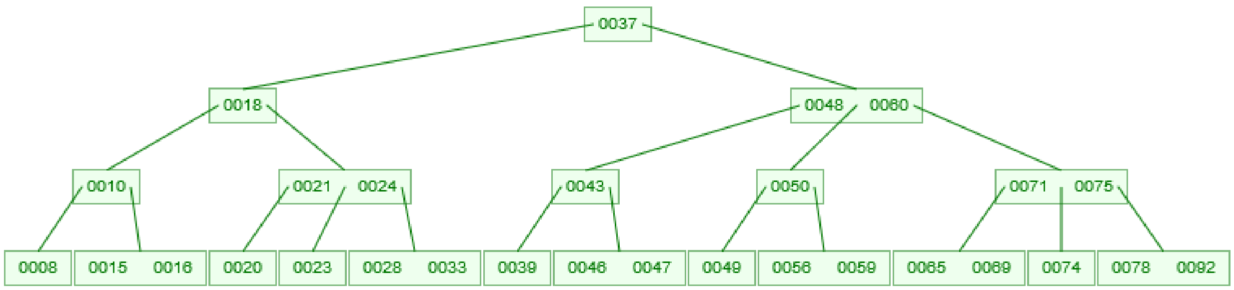
- Insert 8



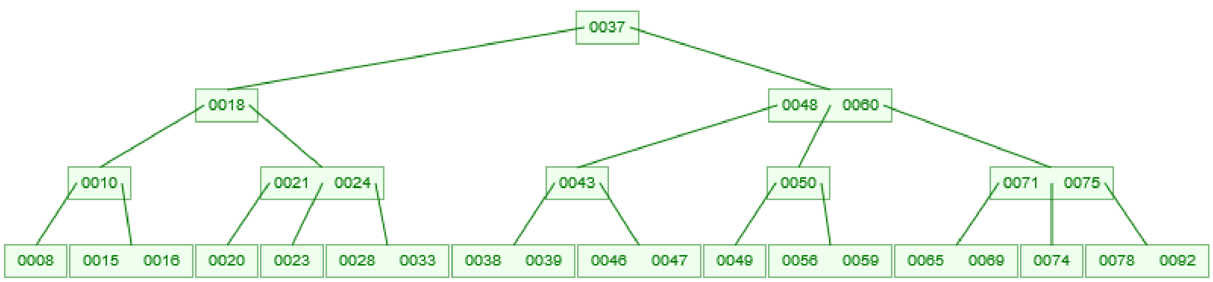
- Insert 49



- Insert 33

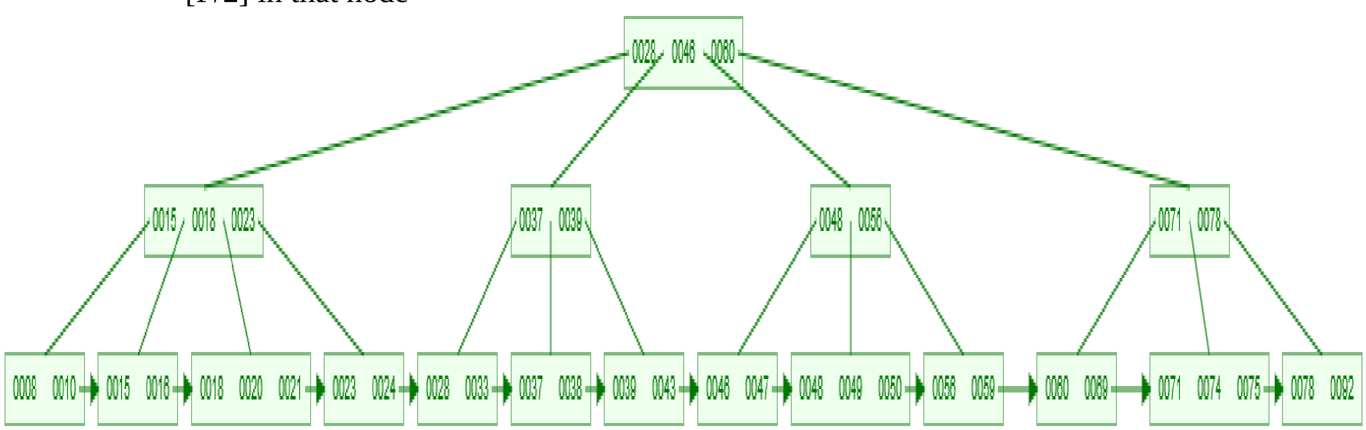


- Insert 38

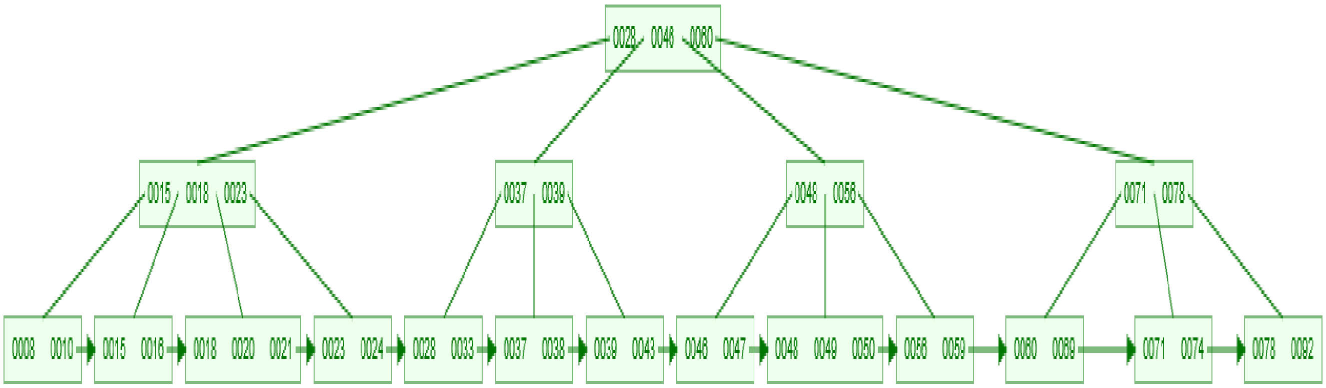


17.21 Suppose that the following search field values are deleted, in the given order, from the B+-tree of Exercise 17.19; show how the tree will shrink and show the final tree. The deleted values are 65, 75, 43, 18, 20, 92, 59, and 37.

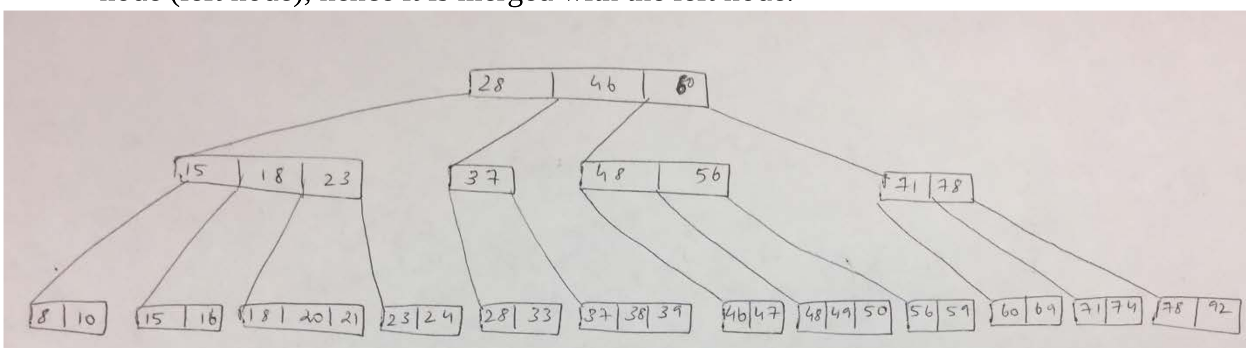
- Delete 65, it does not make much difference as the number of keys is not less than [P/2] in that node



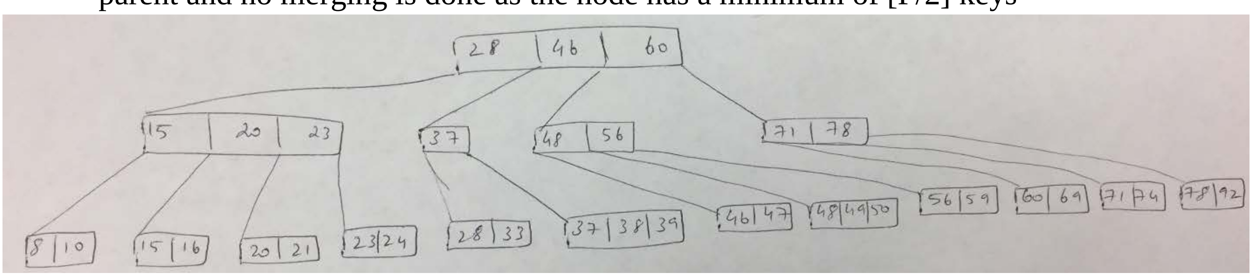
- Delete 75, it does not make much difference as the number of keys is not less than [P/2] in that node



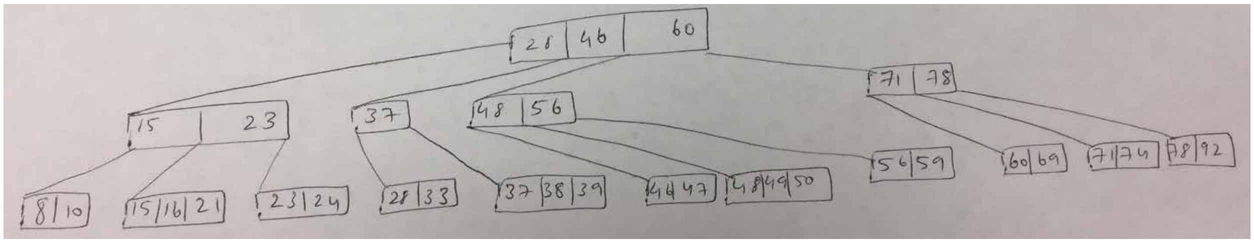
- Delete 43, this deletion results in key numbers less than [P/2] in that node and key cannot be borrowed from left node because it will lead to count < [P/2] in that node (left node), hence it is merged with the left node.



- Delete 18, this affects parent node too. Here, the next least value is updated as the parent and no merging is done as the node has a minimum of [P/2] keys



- Delete 20, this deletion results in key numbers less than [P/2] in that node and key cannot be borrowed from left node because it will lead to count < [P/2] in that node (left node), hence it is merged with the left node.



- Delete 92, this deletion results in key numbers less than [P/2] in that node and key cannot be borrowed from left node because it will lead to count < [P/2] in that node (left node), hence it is merged with the left node.
  Delete 59, this deletion results in key numbers less than [P/2] in that node and key can be borrowed from the left node as it has keys more than [P/2]
  Delete 37, this affects parent node too. Here, the next least value is updated as the parent and no merging is done as the node has a minimum of [P/2] keys
  The final tree is,