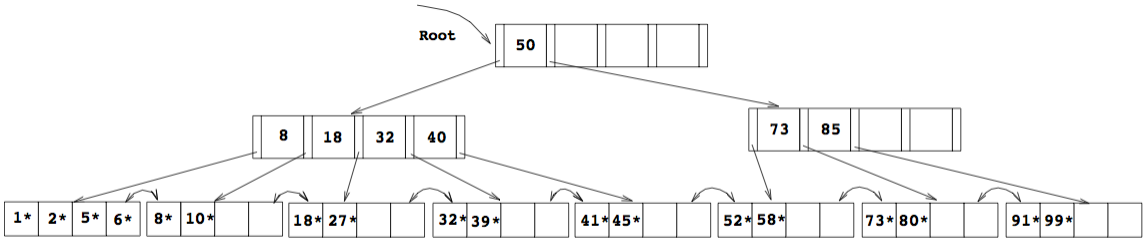

Figure 1: The original B+ tree.

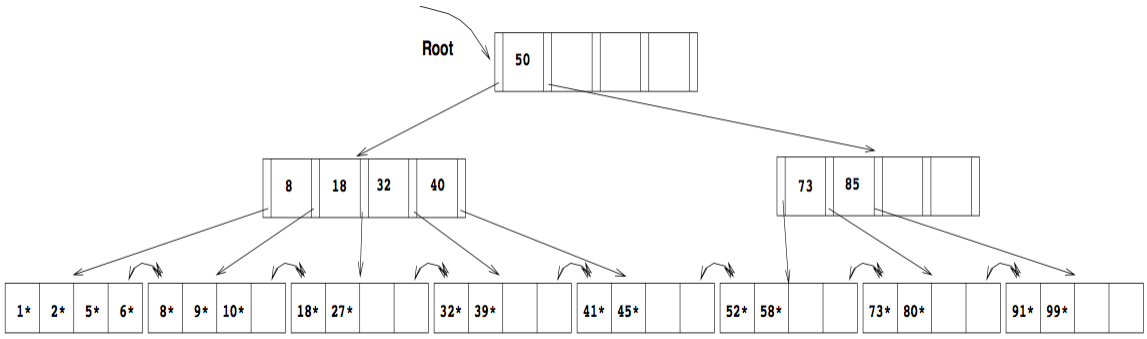

Figure 2: The B+ tree after the changes in (1).

1: Tree Indexing

Consider the B+ tree index of degree d = 2 shown in Figure 1. Instead of pointers to the data records in data files, the leaf nodes in this B+ tree contain data records. The insertion, deletion, and update algorithms for this B+ tree is the same as the ones discussed in the lecture.

1. Show the tree that would result from inserting a data entry with key 9 into this tree.

   Solution: The data entry with key 9 is inserted on the second leaf page. The resulting tree is shown in Figure 2.

2. Show the B+ tree that would result from inserting a data entry with key 3 into the original tree.

   Solution: 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 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.

   Solution: 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 4.

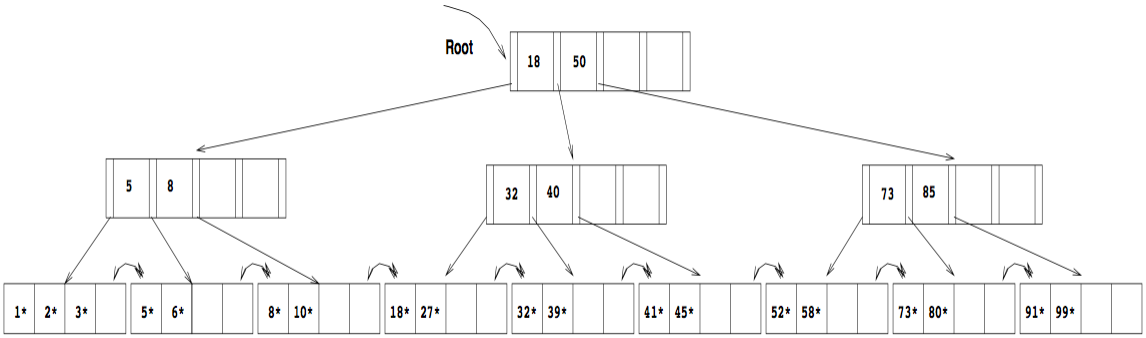

Figure 3: The B+ tree after the changes in (2).

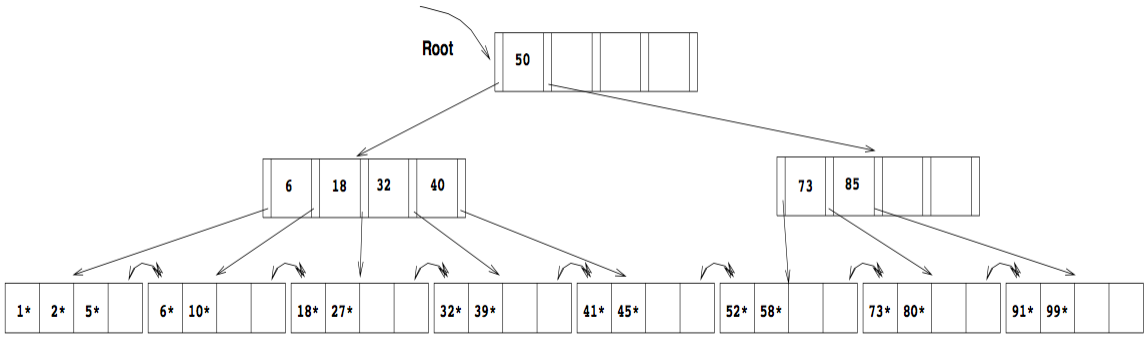

Figure 4: The B+ tree after the changes in (3).

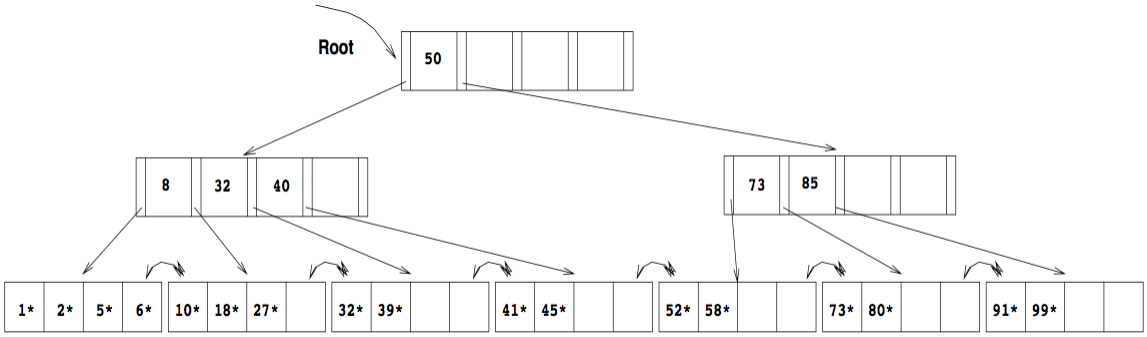

Figure 5: The B+ tree after the changes in (4).

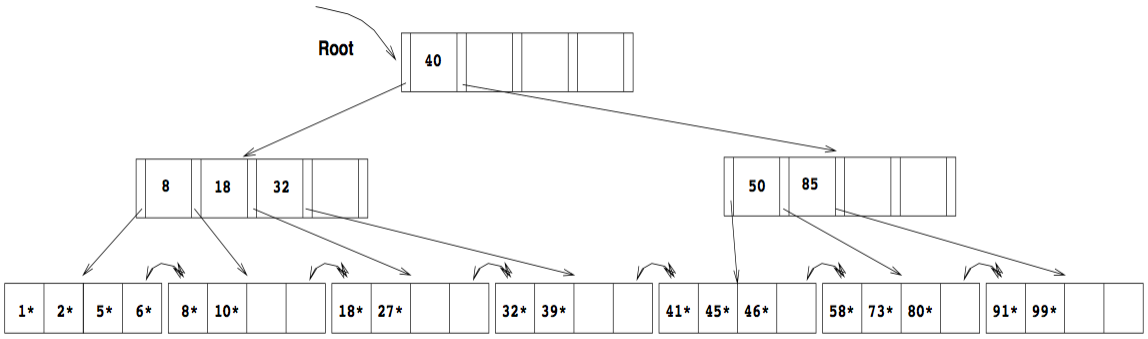

Figure 6: The B+ tree after the changes in (5).

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.

   Solution: 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 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.

   Solution: 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 6.

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

   Solution: Deleting the data entry with key 91 causes a scenario similar to part 5. The result can be seen in Figure 7.

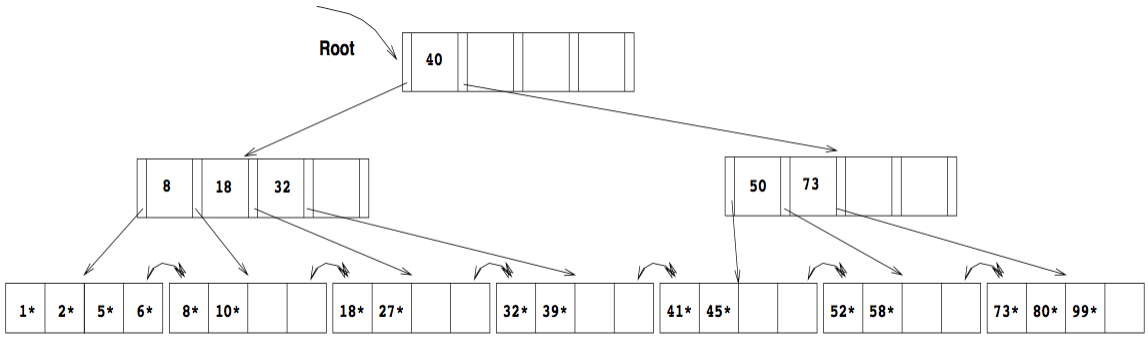

Figure 7: The B+ tree after the changes in (6).

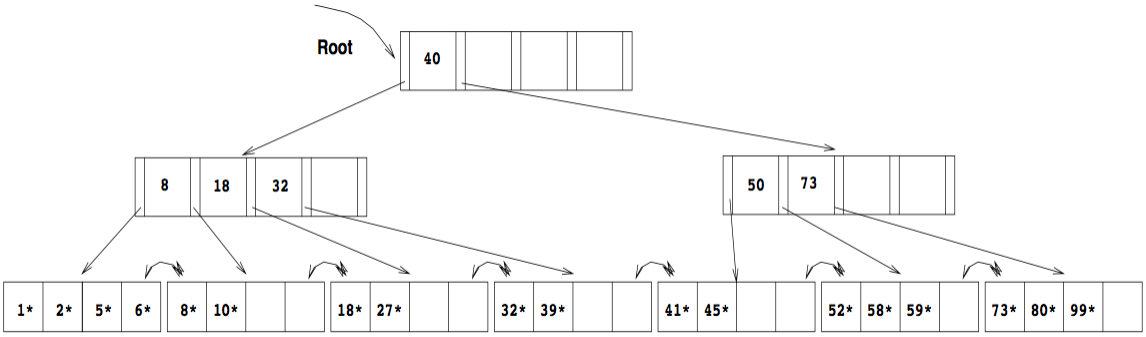

Figure 8: The B+ tree after the changes in (7).

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.

   Solution: 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 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.

   Solution: 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 9.

2: Hash Indexing

In the extendible hash table shown in Figure 10, the search key is either a letter or a decimal number. If it is a letter, the hash function returns the ASCII binary representation of that letter and if it is a decimal number, the hash function returns the binary representation of that number. For example, if the input to the hash function is letter 'A', it returns "01000001" and if the input is 128, it returns "10000000". Show the final hash table

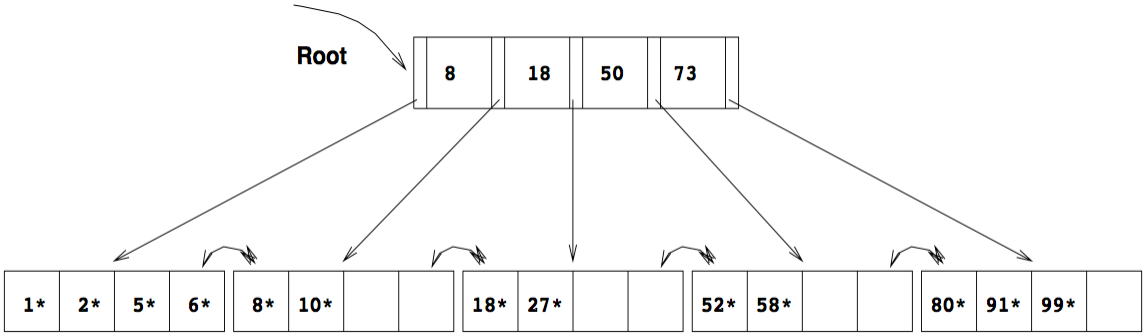

Figure 9: The B+ tree after the changes in (8).

resulting from the insertion of values *'P'* and *208*, *192*, and *255* into the aforementioned extendible hash table. **(solution)** (Shown in the figure 11)

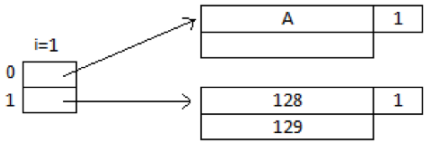

Figure 10: The hash index used in question 2.

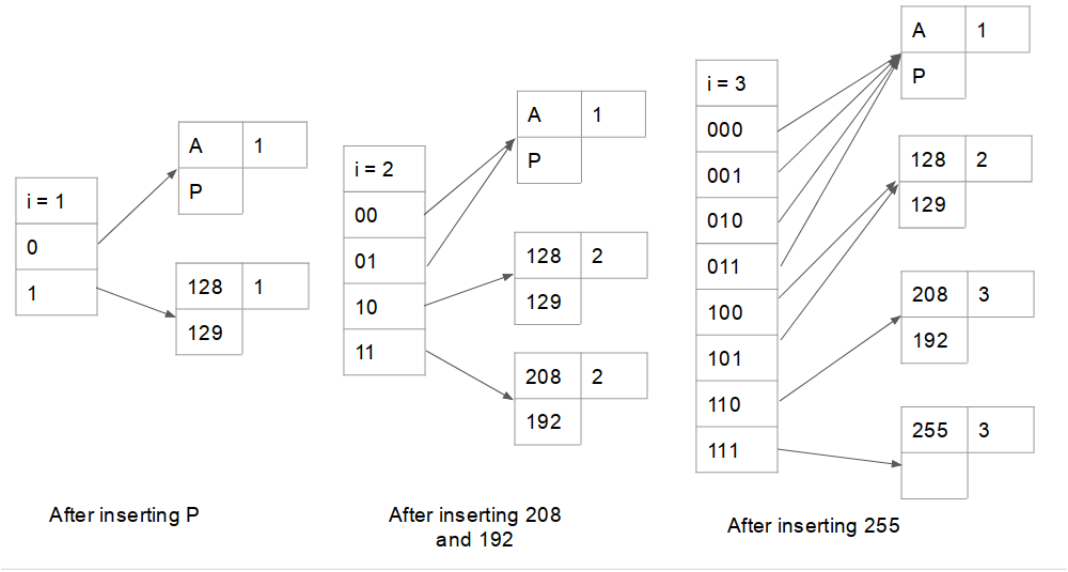

Figure 11: Steps for inserting the data items into the index.