Joshua Baunach

November 30, 2017

CS 405G Fall 2017

Homework 5

Problem 1

(a) The record size R is $30 + 9 + 9 + 40 + 9 + 8 + 1 + 4 + 4 + 1 = 115\ bytes$.

(b) The blocking factor bfr, or the number of records per block, is $Floor\left(\frac{B}{R}\right) = Floor\left(\frac{1024}{115}\right) = 8\ records/block$. The number of blocks need for the full file is $Ceil\left(\frac{r}{bfr}\right) = Ceil\left(\frac{3{,}000{,}000}{8}\right) = 375000$

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

(i) The index blocking factor, bfri, is given by the index record size $Ri = (V(SSN) + P) = 9 + 7 = 16\ bytes$, and from there, the index blocking factor bfri is $floor\left(\frac{B}{Ri}\right) = floor\left(\frac{512}{16}\right) = 32$

(ii) The number of first-level index entries and the number of first-level index blocks is the following:

Number of first-level index entries $r1 = file\ blocks = 375000$

Number of first-level index blocks $b1 = Ceil\left(\frac{r1}{bfri}\right) = Ceil\left(\frac{375000}{32}\right) = 11719$

(iii) The number of levels that would be needed to make it a multi-level index are shown below.

To apply a second-level index…

Number of second-level index entries $r2 = first\ level\ blocks = 11719$

Number of second-level index blocks $b2 = Ceil\left(\frac{r2}{bfri}\right) = Ceil\left(\frac{11719}{32}\right) = 367$

To apply a third-level index…

Number of third-level index entries $r3 = second\ level\ blocks = 367$

Number of third-level index blocks $b3 = Ceil\left(\frac{r3}{bfri}\right) = Ceil\left(\frac{367}{32}\right) = 12$

To apply a fourth-level index…

Number of fourth-level index entries $r4 = third\ level\ blocks = 12$

Number of fourth-level index blocks $b4 = Ceil\left(\frac{r4}{bfri}\right) = Ceil\left(\frac{12}{32}\right) = 1$

Since the fourth level has only one block in it, it is the top index level. So, the index has four levels.

(iv) The number of blocks for the multi-level index is $b1 + b2 + b3 + b4 = 11719 + 367 + 12 + 1 = 12099$

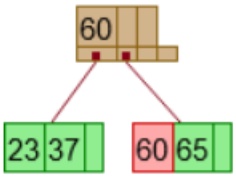
(v) The number of block accesses needed to search for and retrieve a record from the file, given its SSN value, using the primary index is the number of index levels + 1, or 5.

Problem 2

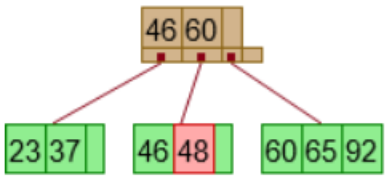
If we are to build a B+ tree with a max fan-out of 4, the tree will expand as follows.

When the first three values 23, 65, and 37 are added, they are added to the first node of the tree. It is shown below:

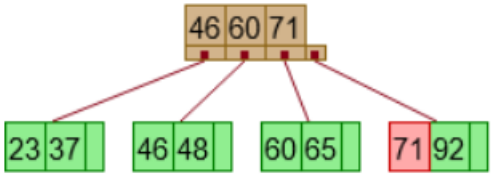
When 60 is added, the root node must split to satisfy the fan-out constraint of 4. The tree will look like this:



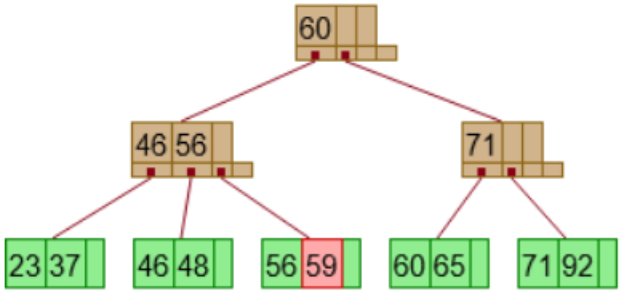
46 is added to the leaf node on the left and 92 is added to the leaf node on the right. When 48 is added, the left node must split, resulting in the tree to look like this:



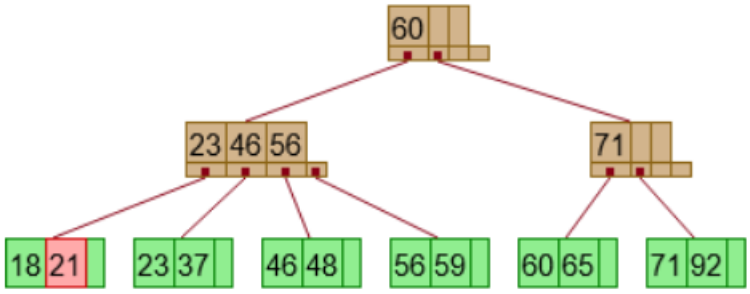
Similarly, when 71 is added, the rightmost leaf node must split, causing an extra key to be added to the root node as well. The tree will look like this:



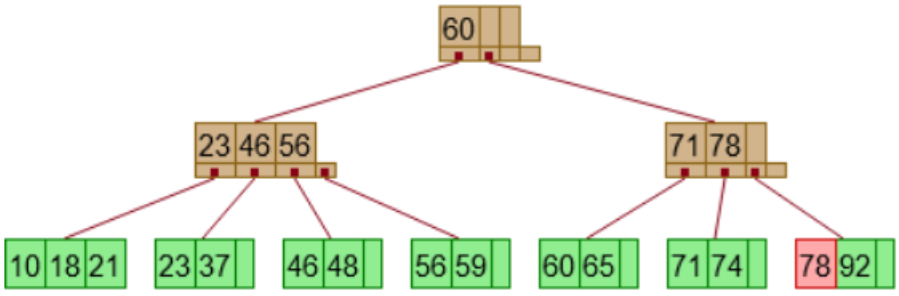
56 is added to the 2nd leaf node. When 59 is added, the 2nd leaf node must split. However, the root node is full, so the root node must split as well. The new tree looks like this:



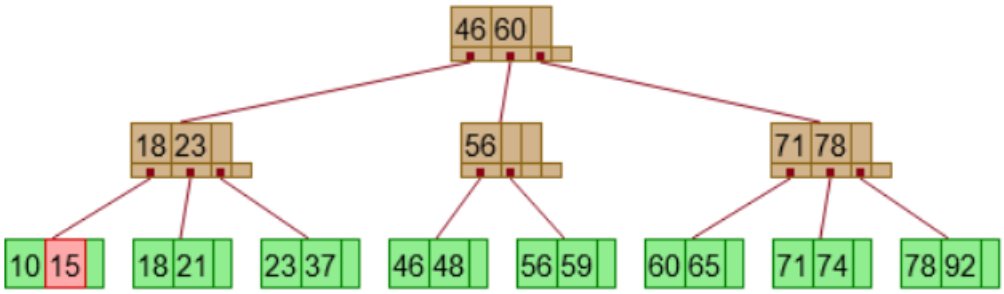
18 is added to the first leaf node. When 21 is added, the left leaf node will split, causing an extra value to be added to its parent node. The tree looks like this:



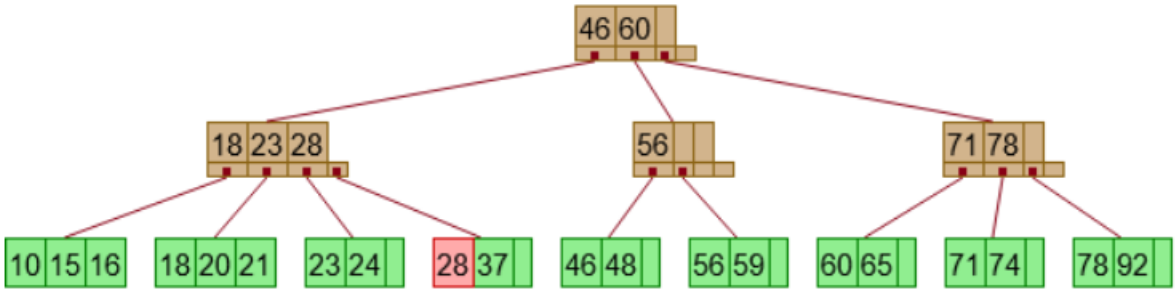
10 is added to the first leaf node and 74 is added to the last leaf node. When 78 is added, the last leaf node will split and a new value will be added to its parent node. The tree will now look like this:



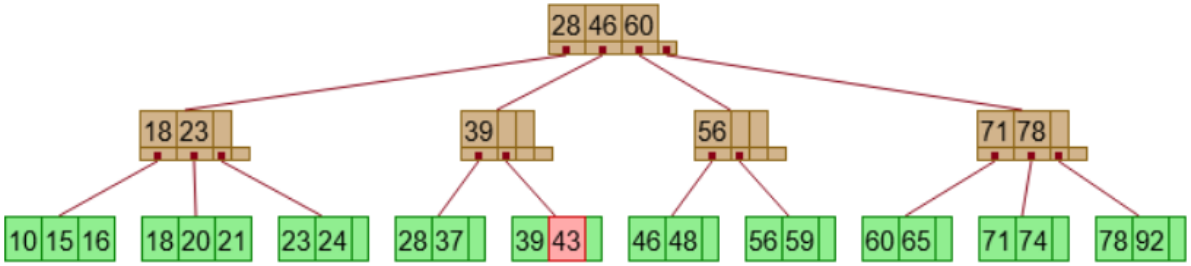
Similarly, when 15 is added, the first leaf node will split. The tree will look like this:



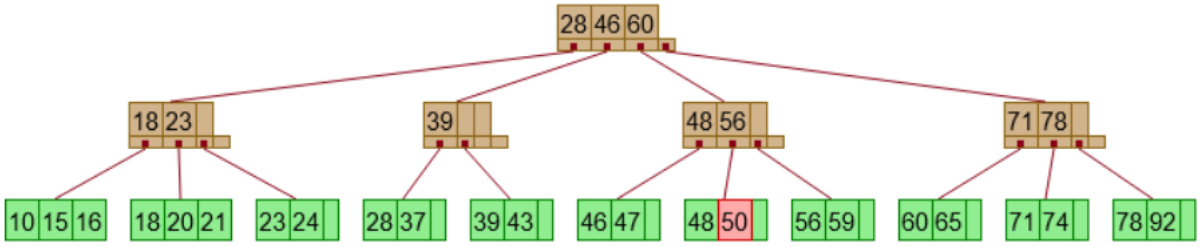
16 is added to the first leaf node, 20 is added to the second leaf node, and 24 is added to the third leaf node. When 28 is added, the third leaf node must split and a new key will be added to its parent. The tree now looks like this:



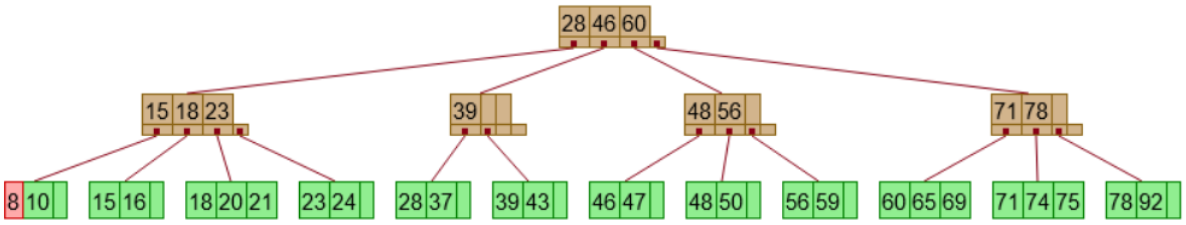
39 is added to the fourth leaf node. When 43 is added, the fourth leaf node must split, but since its parent node is full, that node must split as well, adding a new value to the root node. The tree will look like this:



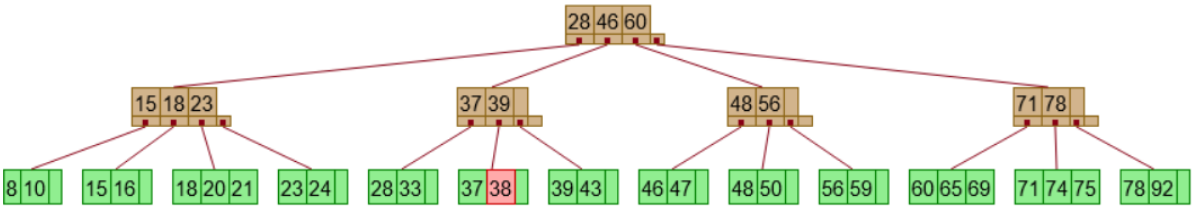
47 is added to the sixth leaf node. When 50 is added, that node will split, adding a new key to the parent node. The tree now looks like this:



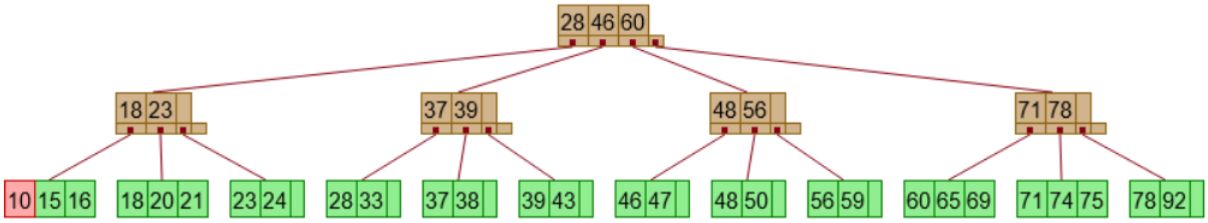
69 will be added to the third-to-last leaf node and 75 will be added to the second-to-last leaf node. When 8 is added to the first node, the node will split and a new value will be added to the parent node. The tree now looks like this:



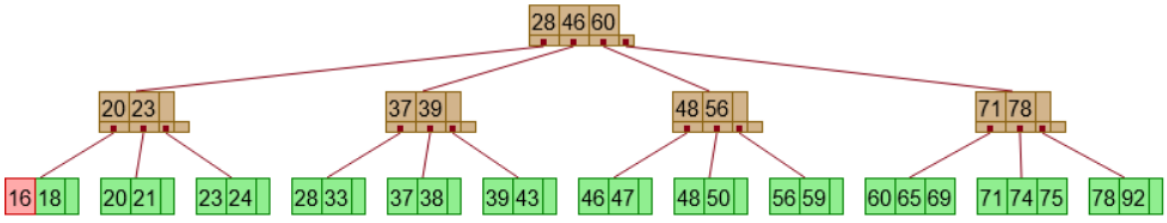
When 47 is added (again), it will be added to the same node the current 47 is in. 33 is added to the fifth leaf node. When 38 is added, it will be added to the node 33 was just added to, but it must be split. A new value will be added to the parent node. The final tree looks like this:



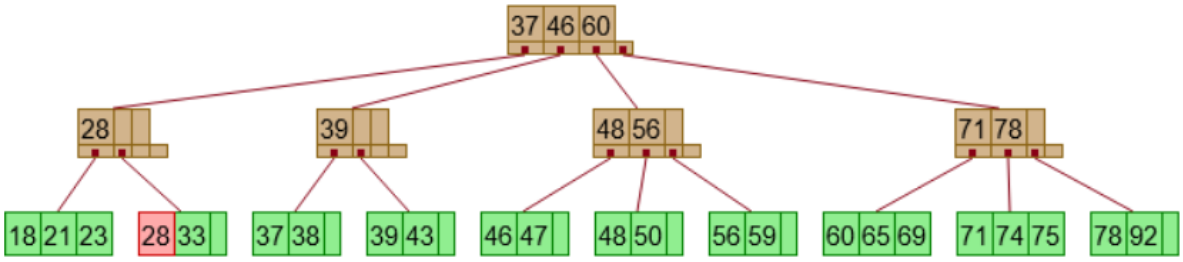
When 8 is removed, the first leaf node will be too small, so it must borrow an element from the second leaf node. However, that node will become too small, so it will merge with the second leaf node. The tree now looks like this:



10 is removed without any problems. When 15 is removed, the first node must borrow an element from the second node, causing the tree to look like this:



When 16 is removed, the 18 will merge with the second leaf node and one value will be removed from the parent node. 20 is removed without any issues. Finally, when 24 is removed, the 23 will merge with the first leaf node. The final tree looks like this:


37 46 60
28
39
48 56
71 78
18 21 23
28 33
37 38
39 43
46 47
48 50
56 59
60 65 69
71 74 75
78 92