HW 8 Solution

Problem [100pts]

Assume that you have just built a dense B+ tree index using Alternative (2) on a heap file containing 20,000 records. Alternative (2) means that it contains the search key value being indexed and the pointer, but no other values of the data records. The key field for this B+ tree index is a 40-byte string, and it is a candidate key. Pointers (i.e., record ids and page ids) are (at most) 20-byte values.

The size of one disk page is 1,000 bytes. The index was built in a bottom-up fashion using the bulk-loading algorithm, and the nodes at each level were filled up as much as possible.

Rubric:

If your final answer is correct — you don't have to write down all the points of this solution. As long as you explain your answer reasonably, you can get full marks.

Else: give your points based on this rubric.

(1) How many levels does the resulting tree have? [40pts]

Since the index is a dense index, there are as many data entries in the B+tree as records in the data file. An index page consists of at most 2*d keys and (2*d + 1) pointers. So we have to maximize d under the condition that [4 pts]

2*d*40 + (2*d+1)*20 <= 1000 bytes (size of disk page). [4 pts]

For d = 8, that is, we get 16 keys and 17 pointers on an index page. [12 pts]

A record on a leaf page consists of the key field and a pointer each, as we are working with alternative (2). Its size thus is 40+20 = 60 bytes each. [4 pts] Therefore a leaf page has space for (1000/60) = 16 data entries. If we wanted to be even more accurate, we would also need to account for the extra space to hold one extra pointer linking the leaf records.

Then the page could only hold 16 data entries. Either calculation is fine.

As consequence, the resulting tree has [log_17 (20,000/16) + 1] = 4 levels.

(2) For each level of the tree, how many nodes are at that level? [40pts]

Since the nodes at each level are filled as much as possible, there are [20,000/16] = 1250 leaf nodes again (on level 4). An internal full index node has 2d+1 = 17 children. Therefore there are [1250/17] = 74 index pages on level 3. Therefore there are [74/17] = 5 index pages on level 2. There is of course one index page on level 1, namely, the root.

(3) How many levels would the resulting tree have if key compression is used and it reduces the average size of each key in an entry to 10 bytes? [20pts]

This answer should be very similar to the answer (1) above.

Again, we have to maximize d with :

2*d*10 + (2*d+1)*20 <= 1000 bytes (size of disk page).

20*d + 40*d + 20   <= 1000 bytes

60*d + 20   <= 1000 bytes

d <= 980/60 ~ 16.3 = 16

For d = 16, that is, we get 32 keys and 33 pointers on an index page.

A leaf node has space roughly for (1000/(10+20)) = 33 data entries. If we wanted to be even more accurate, we would also need to account for the extra space to hold one extra pointer linking the leaf records. Then the page could only hold 32 instead of 33 data entries.

The resulting tree has [log_33 (20,000/32) + 1] = 3.