CS127 Homework #6

Due: November 2nd, 2017 2:59 P.M.

Handing In

Upload your homework to Gradescope.
Please write your Banner ID on your submission. Do not write your name on the submission.

Warmup #1

Explain why the time complexity of B-trees (insertion/deletion/search) is $O(\log_B(N))$ where $N$ is the number of records and $B$ is the branching factor (= number of pointers).

Solution:
Always concerned with part of the tree; since all values are sorted we can always eliminate $2^B - 1$ choices as we move through the tree. It is like a binary tree search where at every level we are limiting the branch we are concerned with.

Warmup #2

Explain the steps you take when an insertion into a B-tree leads to a leaf node overflowing? Why can’t the inserted value fit in the node?

Solution:
You must split at half and move up; this is in order to maintain B tree property of maximum amount of pointers per node; the splitting at halfway also maintains the property of the left side being values less than, and the right side being values greater than.

Warmup #3

What are the differences between B+-tree and B-tree? What are the advantages of a B-tree over a B+-tree? B+-tree over a B-tree?

Solution:
B+-tree also has linked value pairs at the leaf level making it easier to do range queries. B-tree better for searches of key,value pairs that are closest to the root. B+-tree is more memory efficient on queries, because you only have to read in keys as you traverse the tree, as opposed to keys and tuples in a B-tree.

Warmup #4

When is it preferable to use a dense index rather than a sparse index? Explain your answer.

Solution:
It is preferable to use a dense index instead of a sparse index when the file is not sorted on the indexed field (such as when the index is a secondary index) or when the index file is small compared to the size of memory.
Problem 5 (To Be Graded)

Construct a B+-tree for the following set of key values:
(4, 5, 6, 7, 10, 12, 14, 19, 20, 21, 23)
Assume that the tree is initially empty and values are added in ascending order. Construct B+-trees for the cases where the number of pointers that will fit one node is as follows:

1. Four
2. Five
3. Seven

Using the b+ tree created in part a, delete 4, 5 and 6. Show the stops along the way.
Figure 1: 4 PTR
Figure 2: 5 PTR

Figure 3: 7 PTR
Figure 4: Delete