CSE 548 Fall 2012: Homework #3

Due: Nov. 26, 2012

First, write up solutions to the mid-term questions. Then do the following additional problems.

Problem 1

Write an algorithm to merge two skip lists into a single skip list. If it helps, you may assume the following data structures are used to represent the skip lists:

```c
struct skip_list_node {
    struct skip_list_node *next, *child;
    void *data;
};

struct skip_list {
    int nlevels;
    struct skip_list_node *levels[nlevels];
};
```

(Note: it looks like C, but it’s not C – you only need to write pseudo-code for this assignment.)

Problem 2

Design a variant of skip lists for external storage. Analyze your skips lists in the DAM model. How do they compare to B-trees? B’-trees?

Problem 3

In many real disks, reading a sequence of k consecutive blocks at once is not much more expensive than reading a single block. Suppose we revise the model so that a single I/O can read up to B blocks. How would you modify the external memory merge sort algorithm to take advantage of this feature? What would be its new running time?

Problem 4

Consider a B-tree augmented with pointers connecting the leaves into a linked list, so that we can iterate over all the leaves in order. Deleting a single element
Problem 5
Prove that AVL trees always have height $O(\log n)$.

Problem 6
Show that, if we insert $n$ elements into a splay tree in increasing order, then the resulting tree will have height $\Theta(n)$.

Problem 7
This question considers a few tweaks to the $B^\epsilon$-tree. Recall: leaves can hold $B$ elements, internal nodes (other than the root) have $B^\epsilon$ children, buffers hold $B^{1-\epsilon}$ elements. You may also assume $B^\epsilon \geq 2$.

- The amortized analysis of inserts that we did in class did not consider the I/Os required during node splits. Show that the amount of I/O required for node splits when inserting $N$ items is $O(N/B)$, so that this does not affect the amortized insert time.
- As described in class, we do not flush buffers during lookups in a $B^\epsilon$-tree. Consider an alternative approach that flushes all buffers along a lookup path. Show that this is a bad idea.
- As described in class, a $B^\epsilon$-tree node had one buffer for each child. Consider an alternative where we have one buffer per node. For a given node $n$, let the buffer contents be $S = S_1 \cup S_2 \cup \cdots \cup S_{B^\epsilon}$, where $S_i$ is the set of elements in the buffer destined for $n$'s $i$th child. Suppose that when we flush a buffer, we only flush the largest $S_i$. Prove that this offers the same asymptotic insert performance as the standard $B^\epsilon$-tree.
- Suppose we have a single buffer per node, as above. However, when we flush the buffer in a node, we flush all its contents, which may involve updating more than one child. Prove that this also has the same asymptotic insert performance as the standard $B^\epsilon$-tree.
- One possible concern with upserts is that the tree could become filled with upsert messages even though it only contains a small number of actual keys, so that the work required to perform a lookup will be $\frac{\log_B (N+U)}{2}$, where $N$ is the number of real keys, and $U$ is the number of upsert messages. Show that the number of upsert messages in the tree will always be $O(N/B)$.