Problem Set 5

Assigned: June 26
Due: July 3

Problem 1.

Consider a B-tree with a branching factor of B implementing a set of size N. Suppose that, at a particular time while running the database, you have loaded into memory one particular path of blocks from the room to one leaf, plus all the sibling blocks to nodes on that path. Assume that each node is implemented in memory as a sorted array of keys and pointers to descendant nodes. Assume that B-trees are of the form discussed in class (and in Siegel) where all the values are in the leaves, and are repeated as needed in internal nodes. Assume for simplicity that the same value B is both the maximal number of children of an internal node and the maximum number of values in a leaf node.

A. What is the worst-case order-of-magnitude running time of searching for an element as a function of N and B? How about adding an element? Explain your answers.

B. Consider the following scenario. There is a rule, to guard against crashes, that whenever an in-memory image of a block of the B-tree is modified, that block must be written out to disk. In some particular situation, a set is created by carrying out a sequence of N Adds (no Deletes). After adding N elements, the B-tree is still actually small enough to fit in main memory, so the only disk accesses that have occured are the writes required by the above rule.

   Compute the number of disk writes carried out in total as a order-of-magnitude function of N and B.

   Hint: Splitting a node or transferring children between siblings requires two writes to disk. Adding an element to a leaf node or adding an arc to an internal node without other restructuring requires one write to disk. Therefore when an element is added to the set, that requires one or two leaf nodes to be writen out. When an arc is added to the tree, that requires one or two internal nodes to be written out. When a new root is created, two arcs are created but only one node (the root) is written out.

C. Suppose that in (B), after adding N elements to the tree, the tree is a fully branching tree of height K, where every internal node has B children and every leaf has B values. Give an *exact* count of the total number of disk writes performed.

Problem 2

Suppose that we have a hash table of size 19 and we insert the keys 10, 39, 4, 24, 42, 1, 62, 11, 48. Show the final state of the hash table, assuming we use:

A. Chaining, with the hash function $h(k) = k \bmod 19$.

B. Linear probing, with hash function $h(k, i) = (k + i) \bmod 19$

C. Double hashing, with hash function $(k + i * (1 + k \bmod 15)) \bmod 19$

Problem 3

Suppose that you have a set of n large, orderable, objects, each of size q, so that it requires time $\Theta(q)$ to time to compute a hash function $h(x)$ for any object. Describe a compound data structure, built out of a heap and a hash table, that supports the following operations with the specified run times.

- `elt(x)` — Is `x` an element of the set? Expected run time $O(q)$.
- `add(x)` — Add `x` to the set. Expected run time $O(q + \log n)$.
- `delete(x)` — Delete `x` from the set. Expected run time $O(q + \log n)$. Here, note that, in a heap, when you replace `x` by the last element `y`, `x` may be either greater than `y` or less than `y` and your algorithm has to consider both cases.
- `min(x)` — Return the minimum element in the set. Worst case run time $O(1)$.

Problem 4: Extra credit (quite difficult)

In problem 3, suppose that you pick an element at random and delete it. Assume also that the heap has been constructed by doing `adds` in random order, and that therefore, in any row, the order left to right among elements is random. Prove that the expected running time for the delete is $O(q)$ (independent of n).