Albert-Ludwigs-Universität, Inst. für Informatik
Prof. Dr. Fabian Kuhn
Hamid Ghodselahi

November 28, 2012

Algorithm Theory, Winter Term 2012/13
Problem Set 4

hand in by Wednesday, December 12, 2012

Exercise 1: Amortized Analysis [12 Points]

You plan to implement a hash table. Because you don't know how many keys will be inserted into the hash table and because the number of keys stored in the hash table might change over time, you want to be able to adapt the size of the hash table to the number of keys stored in the table. Your implementation should allow two operations *insert* and *delete*.

Assume that you already have an efficient implementation for fixed table size s. Your implementation allows new keys to be inserted efficiently as long as the load of the hash table is less than 3/4. That is, as long as the number of elements in the table is less than $\frac{3}{4} \cdot s$, your implementation guarantees that an *insert* operation (which inserts one new key) costs $O(1)$. A *delete* operation (which deletes one key) can always be done in $O(1)$ time. To adjust the table size based on the number of keys stored in the table, you use the available implementation as follows.

Initially, the hash table is empty. When the first element x is inserted into the table, you build an initial table of fixed size s_0 and insert x. Assume that the cost for doing this is $O(1)$. For simplicity, assume that $s_0 = 8$. Throughout, we will always work with one instance of the available hash table implementation for a table of sufficiently large size s (using $s_0 = 8$ will guarantee that s will always be divisible by 8). Operations *insert* and *delete* are implemented as follows. Assume that s is the current table size and n is the current number of keys stored in the table.

***insert*(*x*):**

- If $n < \frac{3}{4}s$, x is inserted into the current table. Recall that this can be done in $O(1)$ time.
- If $n = \frac{3}{4}s$, we set up a new hash table of size $2s$ and move all items from the old table to the new larger table. Then, x is inserted into the new table. We assume that the time to do this is $O(s)$.

***delete*(*x*):**

- If $n > \frac{1}{8}s$, x is deleted from the current table. Recall that this can be done in $O(1)$ time.
- If $n = \frac{1}{8}s$, we first delete x. If $s > 8$, we then set up a new hash table of size $s/2$ and move all items from the old table to the new smaller table. We assume that the time to do this is $O(s)$.

For simplicity, we normalize time units such that all the above operations that can be done in $O(1)$ time need time at most 1 and the operations that take $O(s)$ time need time at most s.

a) Use the accounting method (the "bank account method") to show that the amortized running times of *insert* and *delete* are $O(1)$.

b) Use the potential function method to show that the amortized running times of *insert* and *delete* are $O(1)$.

Exercise 2: Union-Find [10 Points]

(a) Show that when implementing a union-find data structure by using disjoint-set forests with the union-by-size heuristic, the height of each tree is at most $O(\log n)$.

Hint: *Show that any subtree with k nodes has height at most $\lfloor \log_2 k \rfloor$.*

(b) Demonstrate that the above analysis is tight by giving an example execution that creates a tree of height $\Theta(\log n)$. Can you even get a tree of height $\lfloor \log_2 n \rfloor$?

Exercise 3: Maximum Flow [8 Points]

Consider a set of mobile computing clients in a certain town who each need to be connected to one of several possible *base stations*. We'll suppose there are n clients, with the position of each client specified by its (x, y) coordinates in the plane. There are also k base stations; the position of each of these is specified by (x, y) coordinates as well.

For each client, we wish to connect it to exactly one of the base stations. Our choice of connections is constrained in the following ways. There is a *range parameter* r: a client can only be connected to a base station that is within distance r. There is also a *load parameter* L: no more than L clients can be connected to any single base station.

Design a polynomial-time algorithm for the following problem. Given the positions of a set of clients and a set of base stations, as well as the range and load parameters, decide whether every client can be connected simultaneously to a base station, subject to the described range and load conditions.