CS 361A - Advanced Data Structures and Algorithms

Autumn Quarter, 2005

Homework #1 (Due: 10/19/05)

1. This question deals with the notion of perfect or near-perfect hash functions. We are interested in hash function families $H = \{h : M \to N\}$, where $|M| = m$ and $|N| = n$.

   (a) The family H is said to be perfect if for each $S \subseteq M$ with $|S| = n$, there exists a hash function $h \in H$ which is one-to-one when restricted to S. Prove the best lower bound that you can on the size of a perfect hash function family. For what range of values of m can you find a perfect family H of size at most m?

   (b) In a like manner, we say that h is b-perfect for S if it maps at most b elements of S onto any particular element of N. Show that for all $m \geq n$ there exists a $\log m$-perfect hash family of size at most m.

2. Recall the definition of a *near-2-universal* hash function family $H = \{h : M \to N\}$. It has the property that for any $x, y \in M$, and a random $h \in H$, the probability that $h(x) = h(y)$ is at most $2/n$, unless of course $x = y$. Assume that m is a prime number and define the function $h_k(x) = (kx \pmod m) \pmod n$ for each $k \in \mathcal{Z}_m$.

   (a) Show that the family $H = \{h_k \mid k \in Z_m\}$ is near-2-universal.

   (b) Prove a lower bound on the collision probability for this family of hash functions.

3. Recall the elementary data types STACK and QUEUE. You are given a black-box implementation of STACK which performs the PUSH or POP operations in $O(1)$ worst-case time. Show how you can implement the QUEUE data structure with *two* such STACK data structures such that the *amortized* cost of each QUEUE operation is $O(1)$. Also, determine the worst-case cost of each operation under your implementation.

4. We have seen that searching a sorted array takes logarithmic time. It is also easy to see that any update (insert or delete) will take linear time in the worst-case. In this problem we will develop a scheme for improving the update time by partitioning the array into sorted sub-arrays.

   Suppose that we are implementing a dynamic dictionary whose current size is n. Let the binary representation of n be $b_{k-1}b_{k-2}\ldots b_0$, where $k = \lceil \log(n+1) \rceil$. The idea is to have k sorted arrays $A_0, \ldots, A_{k-1}$, where the length of A_i is 2^i. The n elements of the dictionary are (arbitrarily) partitioned among the various arrays in such a way that: if $b_i = 0$ then A_i is completely empty, and if $b_i = 1$ then A_i is completely full. (Notice that $\sum_{i=0}^{k-1} b_i 2^i = n$.)

   (a) Describe how to perform a search on this data structure and analyze its worst-case cost.

   (b) Describe how to insert an element into this data structure and analyze both the worst-case and amortized costs of your implementation.

   (c) Discuss the issue of implementing the delete operation.