CS 105 (Winter 2005)
Homework 1 Solutions

Problem 1

We have a function family $h_A : \{0,1\}^m \to \{0,1\}^n$, parametrized by a Boolean matrix $A \in \{0,1\}^{(m+1)\times n}$, and defined thus:

$$h_A(x) \;=\; x^{(1)}A\,,$$

where all arithmetic is done modulo 2 and where $x^{(1)}$ denotes the vector x with a 1 appended at the end. We shall prove straightaway that the family $\{h_A\}$ is strongly 2-universal. By a result in Problem 2, this implies that it is also 2-universal.

Let $x, y \in \{0,1\}^m$ be two distinct keys (i.e., $x \neq y$) and let $r, s \in \{0,1\}^n$ be two (not necessarily distinct) hash values. Our goal is to show that $\Pr_A[h_A(x) = r \;\wedge\; h_A(y) = s] = 2^{-2n}$. Let x_i denote the i^{th} entry of x, etc. and let a_{ij} denote entry (i,j) in A. Since $x \neq y$ there must exist a $k \in [1,m]$ such that $x_k \neq y_k$.

For $j \in [1,n]$, let $\mathcal{E}_j$ denote the event "$(h_A(x))_j = r_j \;\wedge\; (h_A(y))_j = s_j$". Since the columns of A are chosen independently and since the j^{th} entries of $h_A(x), h_A(y)$ depend only on the j^{th} column of A, it follows that the n events $\mathcal{E}_1, \ldots, \mathcal{E}_n$ are pairwise independent. Therefore

$$\Pr[h_A(x) = r \;\wedge\; h_A(y) = s] \;=\; \Pr[\mathcal{E}_1 \wedge \cdots \wedge \mathcal{E}_n] \;=\; \prod_{j=1}^{n} \Pr[\mathcal{E}_j]\,. \tag{1}$$

Fix a column $j \in [1,n]$ and fix any arbitrary assignment of values to a_{ij} for $i \in [1,m], i \neq k$. The event $\mathcal{E}_j$ is equivalent to the following system of two linear equations (remember, all arithmetic is modulo 2):

$$\begin{aligned} \sum_{i=1}^{m} a_{ij}x_i + a_{(m+1),j} &= r_j\,, \\ \sum_{i=1}^{m} a_{ij}y_i + a_{(m+1),j} &= s_j\,. \end{aligned}$$

Since all values above except for a_{kj} and $a_{(m+1),j}$ are fixed, we can think of this as a system of equations in those two unknowns:

$$\begin{bmatrix} x_k & 1 \\ y_k & 1 \end{bmatrix} \begin{bmatrix} a_{kj} \\ a_{(m+1),j} \end{bmatrix} \;=\; \begin{bmatrix} r' \\ s' \end{bmatrix}, \tag{2}$$

where r' and s' are some fixed values in $\{0,1\}$. The determinant

$$\begin{vmatrix} x_k & 1 \\ y_k & 1 \end{vmatrix} \;=\; x_k - y_k \;\neq\; 0$$

by assumption, so this system has a unique solution. Since the two random bits $(a_{kj}, a_{(m+1),j})$ together have four equally likely values in all, the probability that they take on exactly the values given by this unique solution is $1/4$. Thus, for every fixed setting of a_{ij} for $i \in [1,m], i \neq k$, the conditional probability of $\mathcal{E}_j$ is $1/4$.

Since this is true for *every* fixed setting, the *unconditional* probability $\Pr[\mathcal{E}_j] = 1/4$ as well.

Using this in (1), we reach our goal:

$$\Pr[h_A(x) = r \;\wedge\; h_A(y) = s] \;=\; \prod_{j=1}^{n} \frac{1}{4} \;=\; 2^{-2n}\,.$$

It's clear that our argument made use of the padding of x, y to $x^{(1)}, y^{(1)}$ when we considered equation (2). Indeed, if we did not pad, we would have always hashed the zero vector 0^m to 0^n for every A and the family $\{h_A\}$ would not have been strongly 2-universal. It would still have been 2-universal, though. (So what does this say about the definition of 2-universal?)

Problem 2

Part (a) We'll use [CLRS] terminology in this solution. Suppose $\mathcal{H}$ is 2-universal. Suppose x and y are two distinct keys in U. Let $\mathcal{E}_i$ denote the event "$h(x) = i \wedge h(y) = i$", where $i \in \{0, 1, \ldots, m-1\}$. By definition of 2-universality, for any such i, we have

$$\Pr_{h \leftarrow \mathcal{H}}[\mathcal{E}_i] \;=\; 1/m^2\,.$$

Note that the event "$h(x) = h(y)$" is simply $\mathcal{E}_0 \cup \cdots \cup \mathcal{E}_{m-1}$. Since the events $\{\mathcal{E}_i\}$ are mutually exclusive,

$$\Pr_{h \leftarrow \mathcal{H}}[h(x) = h(y)] \;=\; \Pr_{h}[\mathcal{E}_0 \cup \cdots \cup \mathcal{E}_{m-1}] \;=\; \sum_{j=0}^{m-1} \Pr_{h}[\mathcal{E}_j] \;=\; \sum_{j=0}^{m-1} 1/m^2 \;=\; 1/m\,.$$

Therefore $\mathcal{H}$ is universal.

Part (b) We'll continue to use [CLRS] terminology. We'll do arithmetic in the field $\mathbb{Z}_p$ for this solution, so we don't have to keep writing "mod p" repeatedly. Suppose $x = \langle x_0, \ldots, x_{n-1}\rangle$ and $y = \langle y_0, \ldots, y_{n-1}\rangle$ are two distinct keys. There must exist a $k \in [0, n-1]$ such that $x_k \neq y_k$. Consider any two (not necessarily distinct) hash values $r, s \in \mathbb{Z}_p$. The event $\mathcal{E} :=$ "$h_{a,b}(x) = r \wedge h_{a,b}(y) = s$" occurs iff the following equation holds:

$$\begin{bmatrix} x_k & 1 \\ y_k & 1 \end{bmatrix} \begin{bmatrix} a_k \\ b \end{bmatrix} \;=\; \begin{bmatrix} r - \sum_{i=0, i\neq k}^{n-1} a_i x_i \\ s - \sum_{i=0, i\neq k}^{n-1} a_i y_i \end{bmatrix}$$

Consider any fixed assignment of values in $\mathbb{Z}_p$ to the a_i's for $i \in [0, n-1], i \neq k$. Then, the above equation can be viewed as a system of two linear equations in the unknowns a_k and b. Since the determinant $= x_k - y_k \neq 0$, the system has a unique solution in $\mathbb{Z}_p$ (here, we are using the fact that $\mathbb{Z}_p$ is a field and so the nonzero quantity $(x_k - y_k)$ has a multiplicative inverse). Since a_k and b are chosen uniformly at random from $\mathbb{Z}_p$, they will attain the values given by the unique solution with probability $1/p^2$. Therefore, conditioned on the above fixed assignment, the probability of $\mathcal{E}$ is $1/p^2$.

Since this conditional probability is $1/p^2$ for *every* such assignment of values to a_i, $i \neq k$, the *unconditional* probability $\Pr[\mathcal{E}] = 1/p^2$ as well. This proves the 2-universality, as required.

Problem 3

Part 3.1 Let $d_1, \ldots, d_n$ be the depths of the n nodes in the binary search tree. The average access time under a uniform distribution is, by definition, $\sum_{i=1}^{n}(d_i/n)$. We are given that this average is $\Theta(\log n)$, so we have

$$\sum_{i=1}^{n} d_i \;=\; \Theta(n \log n)\,. \tag{3}$$

Since the cost of splaying at a node is a linear function of its depth, the worst case cost of splaying in this tree is $\Theta(h)$, where h is the depth of the deepest node. The path from the root to this deepest node consists of nodes at depth $1, 2, \ldots, h$. The sum of these depths is clearly $1 + 2 + \cdots + h = h(h+1)/2 \geq h^2/2$. Using this in (3), we see that

$$h^2/2 \;\leq\; \Theta(n \log n)\,,$$

whence $h = O(\sqrt{n \log n})$.

Part 3.2 Consider a tree T whose root's left child T_L is a left path of length $\ell := \lceil \sqrt{n \log n} \rceil$ and whose root's right child T_R is a balanced (or as close to balanced as possible) binary tree on $(n - 1 - \ell)$ nodes. Clearly, it costs $\Theta(\sqrt{n \log n})$ to splay at the deepest node in such a tree.

To analyze the average access time, let $d(x)$ denote the depth of a node x in this tree. Notice that any $x \in T_R$ satisfies $d(x) \leq 1 + \lfloor \log(n-1-\ell) \rfloor \leq \lceil \log n \rceil$. Therefore,

$$\begin{aligned}
\text{average access time} &= \sum_{x \in T} \frac{d(x)}{n} \\
&= \frac{1}{n} + \sum_{x \in T_L} \frac{d(x)}{n} + \sum_{x \in T_R} \frac{d(x)}{n} \\
&= \frac{(\ell+1)(\ell+2)}{2n} + \sum_{x \in T_R} \frac{d(x)}{n} \\
&\leq \frac{\ell^2}{n} + \sum_{x \in T_R} \frac{\lceil \log n \rceil}{n} \\
&= \frac{O(n \log n)}{n} + \frac{\lceil \log n \rceil}{n} \cdot |T_R| \\
&= O(\log n)\,,
\end{aligned}$$

because $|T_R| \leq n$. This proves the result.

Problem 4

Define an "n-gloop" to be a binary tree consisting of a single path alternating between left and right children and with the root's child being a left child. We would like to show that, for any odd $n \geq 5$, there is a sequence of accesses that turns any n-node splay tree into an n-gloop. We may assume, WLOG, that the keys stored in the tree are $\{1, 2, \ldots, n\}$.

We shall prove this by induction on n. For the base case, $n = 5$, we can easily work out that the access sequence $\langle 1, 2, 3, 4, 5, 2, 4, 1, 5 \rangle$ will turn any 5-node splay tree into a 5-gloop.

For the induction step, suppose n is odd and $n \geq 7$. Consider the access sequence $\sigma_i = \langle 1, 2, \ldots, i \rangle$, where $i \leq n$. We shall show that applying σ_{n-1} to any n-node splay tree T turns it into the tree T_1 shown in Figure 1. (You may skip the following claim and its proof if you find this obvious.)

> CLAIM: Applying σ_i to T turns it into a tree T' consisting of the nodes $i, i-1, \ldots, 1$ along a root-to-leaf left path and the remaining nodes in the right subtree of the root.
>
> PROOF: We proceed by induction on i. For the base case $i = 1$, the claim is clear because splaying moves 1 to the root and all other nodes, which have keys > 1, must fall into the right subtree of this new root. For $i > 1$, let $\hat{T}$ be the tree obtained by applying σ_{i-1} to T. By induction hypothesis, the node i is the *smallest* node in the right subtree of the root. Therefore, when we access i in $\hat{T}$ and splay at that node, it can never become the right child of the right child of the root, i.e., the final splaying step can never be a zig-zig. Now a simple study of the definitions of the zig and zig-zag operations shows that T' has the desired structure, which completes the proof.

Let T_1' be the subtree of T_1 consisting of nodes $\{2, 3, \ldots, n-1\}$. By induction hypothesis, there is an access sequence π that turns T_1' into an $(n-2)$-gloop. Let x be the root and y the left child of the root in this gloop, as shown in Figure 2. Let us apply π to T_1 to get a tree T_2. We ask ourselves: where do nodes 1 and n end up in T_2?

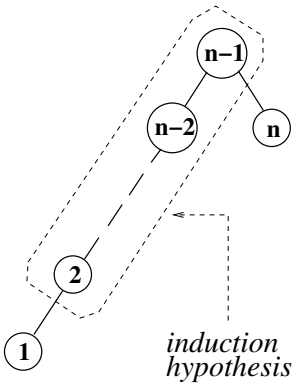

Figure 1

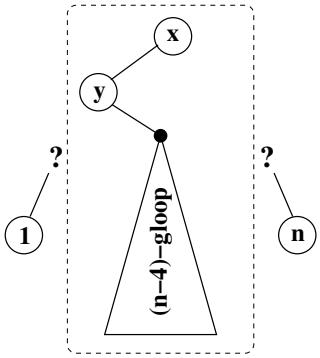

Figure 2

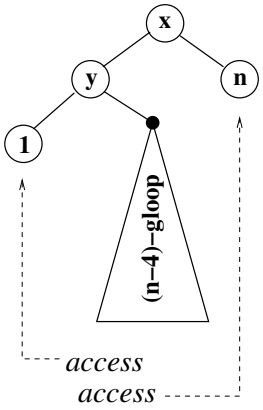

Figure 3

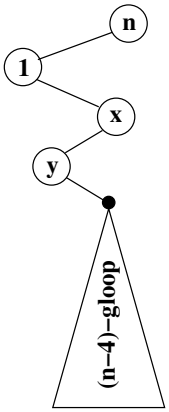

Figure 4

Since n and 1 are the largest and smallest keys, respectively, the only possible locations for them are as the right child of x and as the left child of y, respectively. No other locations are consistent with the binary search tree ordering rules and with the fact that 1 and n must be non-roots (why?). Therefore, our splay tree now looks like the tree in Figure 3.

Apply the access sequence $\langle 1, n\rangle$ to T_2. The first of these results in a zig-zig and the second in a zig-zig followed by a zig. It is straightforward to check that the final result is the tree T_3 shown in Figure 4. And T_3 is an n-gloop, which completes our proof!

Thanks to Rajendra Magar for this most elegant solution. Khanh Do Ba also wrote up a very rigorous proof, as did Anne Loomis. Most students found an access sequence that works but did not argue the correctness very rigorously.

Problem 5

Almost every submitter had a good solution for this problem, so we shall skip it here. The only part that might be considered tricky is the selection of a edge uniformly at random in $O(n)$ time. The way to do it is to maintain not just a weighted adjacency matrix of the graph but also the sum of the entries in each row of such a matrix (i.e., the degree of each vertex) and the sum of these sums (i.e., the total number of edges times two).

Let m be the current number of edges. To choose a random edge, first pick an integer $k \in [1, 2m]$ uniformly at random; this can be done using $\lceil \log(2m) \rceil$ random bits. Let d_i be the degree of the i^{th} vertex. Find the smallest i such that $d_1 + d_2 + \cdots + d_i \geq k$. Then, within the i^{th} row of the adjacency matrix, find the column j such that $d_1 + \cdots + d_{i-1} +$ the sum of the first j entries of this i^{th} row $\geq k$. The pair (i, j) is the next pair to be contracted.

Problem 6

We worked out in class that contraction to t vertices discards a min cut with probability at most $1 - t(t-1)/n(n-1) = 1 - \Theta(t^2/n^2)$. With k repetitions, the probability of discarding the min cut each time falls to

$$(1 - \Theta(t^2/n^2))^k \;\leq\; e^{-k \cdot \Theta(t^2/n^2)}\,.$$

To make this less than $1/2$, we need $k = \Theta(n^2/t^2)$ repetitions.

Each contraction can require up to $\Theta(n)$ time, so each series of $(n-t)$ contractions takes $\Theta(n(n-t))$ time. The cubic time algorithm applied to the remaining t-vertex graph takes a further $\Theta(t^3)$ time. Therefore, the k repetitions of the entire algorithm to get the failure probability below $1/2$ can make the overall running time as high as

$$T_t(n) \;:=\; \Theta\left((n(n-t) + t^3) \cdot n^2/t^2\right) \;=\; \Theta(f_t(n) + g_t(n))\,,$$

where $f_t(n) := n^3(n-t)/t^2$ and $g_t(n) = n^2 t$. We shall show that even if we choose the best t that minimizes $f_t(n) + g_t(n)$, we will still have $T_t(n) = \Omega(n^{8/3})$.

At this point, just about every submitted solution resorted to calculus to do the minimization. This is unnecessarily tedious and in fact hard to get right, because you have to compute two derivatives and "solve" a cubic equation. Life is a lot simpler than that: we only seek to prove an $\Omega()$-bound on the running time, so we don't need the precision that calculus give us.

First, we note that we may assume $t < n/2$. If not, $g_t(n) = \Omega(n^3) > \Omega(n^{8/3})$ already. So, with this assumption, $f_t(n) \geq h_t(n) := n^4/(2t^2)$. There is a point $t_0 \in [1, n/2]$ at which $h_{t_0}(n) = g_{t_0}(n)$: this t_0 is given by

$$\frac{n^4}{2t_0^2} \;=\; n^2 t_0\,, \quad \text{i.e., } t_0 = \Theta(n^{2/3})\,.$$

Now note that, as a function of t in the interval $[1, n/2]$, $h_t(n)$ is decreasing and $g_t(n)$ increasing. Therefore, at any point in the interval, one of these two quantities must be $\geq h_{t_0}(n) = g_{t_0}(n) = n^2 t_0 = \Theta(n^{8/3})$.

Thus, for $t \in [1, n]$, $T_t(n) \geq \Theta(n^{8/3})$, i.e., $T_t(n) = \Omega(n^{8/3})$.

Elizabeth Moseman was the only submitter to give this simple, calculus-free solution.