CS 161: Section 4

1 A Different Universal Hash Family

Let's say we have u-bit long binary keys (so your universe is of size 2^u), and we want to insert our keys into a hash table of size $M = 2^b$. We can number our buckets with b-bit long bit strings. For example, we might want to hash the key 10010 (so $u = 5$) and your hash table has positions $000, 001, 010, ..., 111$ (so $b = 3$). We talked in class about using tricks of modular arithmetic; here's a different way to form a universal hash family using matrix multiplication!

Because our keys and buckets are bit strings, we can think of them as vectors of bits. So our example key is now the length-5 vector $(1, 0, 0, 1, 0)$. How can we transform this length-5 vector into a length-3 vector? One natural thing to do is to multiply by a 3×5 matrix of 1's and 0's, using binary (i.e., modulo 2) arithmetic. So to pick a hash function h, we pick a matrix A and then let $h(x) = Ax$. Let's say we happen to pick the matrix:

$$A = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Using our example A and our example vector as x, this gives us:

$$h(x) = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

1. Prove that this gives a universal hash family.
2. What is the size of this hash family? How does that compare with the family discussed in class? What might be the pros and cons of each hash family in a practical application?

2 A Randomly Built BST

In this problem, we prove that the average depth of a node in a randomly built binary search tree with n nodes is $O(\log n)$. A *randomly built binary search tree* with n nodes is one that arises from inserting the n keys in random order into an initially empty tree, where each of the $n!$ permutations of the input keys is equally likely.

Let $d(x, T)$ be the depth of node x in a binary tree T (the depth of the root is 0). Then, the average depth of a node in a binary tree T with n nodes is

$$\frac{1}{n} \sum_{x \in T} d(x, T) \ .$$

1. Let the *total path length* $P(T)$ of a binary tree T be defined as the sum, over all nodes x in T, of the depth of node x. We note that the average depth of a node in T with n nodes is equal to $\frac{1}{n} P(T)$. Show that $P(T) = P(T_L) + P(T_R) + n - 1$, where T_L and T_R are the left and right subtrees of T, respectively.

2. Let $P(n)$ be the expected total path length of a randomly built binary search tree with n nodes. Show that $P(n) = \frac{1}{n}\sum_{i=0}^{n-1}(P(i) + P(n-i-1) + n - 1)$.

3. Show that $P(n) = O(n \log n)$. You may cite a result previously proven in the context of other topics covered in class.

4. Design a sorting algorithm based on randomly building a binary search tree. Show that its (expected) running time is $O(n \log n)$. Assume that a random permutation of n keys can be generated in time $O(n)$