CS 3510 - Honors Algorithms
Homework 5
Assigned March 1
Due Tuesday, March 14

1. Consider the following family H of hash functions mapping $\{1, 2, 3, 4\}$ into $\{0, 1\}$. The family H contains the three functions h_1, h_2, h_3 defined below. Is H a family of 2-universal hash functions?

$$h_1(1) = 0, \quad h_2(1) = 1, \quad h_3(1) = 1,$$
$$h_1(2) = 1, \quad h_2(2) = 0, \quad h_3(2) = 1,$$
$$h_1(3) = 1, \quad h_2(3) = 1, \quad h_3(3) = 0,$$
$$h_1(4) = 0, \quad h_2(4) = 0, \quad h_3(4) = 0.$$

2. Let $[m]$ denote the set $\{0, 1, \ldots, m-1\}$. For each of the following families of hash functions, say whether it is 2-universal or not, specify how many random bits are needed to sample from the family, and then justify your answer.

   a) $H_a = \{h(x_1, x_2) = a_1 x_1 + a_2 x_2 (\pmod m) | a_1, a_2 \in [m]\}$, where m is some fixed prime. (Note that each $h \in H_a$ has signature $h : [m]^2 \to [m]$, i.e., it maps a pair of integers in $[m]$ to a single integer in $[m]$.)

   b) $H_b = \{h(x_1, x_2) = a_1, x_1 + a_2 x_2 (\pmod m)\} : a_1, a_2 \in [m]\}$, where $m = 2^k$ is some fixed power of two.

   c) the set of all functions $f : [m] \to [m-1]$.

3. Here we explore problems with hashing by being an adversary who controls what the input looks like. Our analysis that universal families of hash functions is good relies on the fact that the input and the choice of hash function from our family are independent of each other.

   a) Suppose you want to slow down an application's performance. You know the application has a hash table with m buckets and you know the hash function h. How many insertions of elements of your choice do you need to degrade the lookup performance to $O(m)$ on queries of your choice? Assume that you can efficiently find hash collisions, i.e., x and y such that $x \neq y$ but $h(x) = h(y)$. Also assume that your universe U is very large compared to m.

   b) The application designer has come out with a new version of her application. This one uses a Bloom filter instead of a hash table. Unfortunately, one of her

friends told you the hash functions $h_1, ..., h_k$ that the Bloom filter uses, and these functions are static for all copies of the application.

This time you want to increase the false positive probability of the Bloom filter. If there are m buckets in the Bloom filter array, how many insertions of your choice do you need before the probability of a false positive is 1? Assume the Bloom filter has just been initialized and no legitimate elements have been inserted. You may also assume that for any hash function h_i you can efficiently find collisions *on targeted elements* for all hashes simultaneously, i.e., for all i, given a value t, you can find an x such that $h_i(x) = t$. Also assume that your universe U is very large compared to m.

c) Why would you want to perform this kind of adversarial hashing? (Hint: Google's "algorithmic complexity for denial of service.") Give a short (1 paragraph or less) description of a plausible scenario.

4. Given an array A of n integers $a_1, a_2, \ldots, a_n$ in unsorted order, we say that $a_{j_1}, \ldots, a_{j_t}$ is an *increasing subsequence* of A of length t if the following conditions are met:

   (a) $j_1 < j_2 < \ldots < j_t$

   (b) $a_{j_1} \leq a_{j_2} \leq \ldots \leq a_{j_t}$

   For example, given the array $[1, 4, 7, 3, 11, 2, 5, 13, 6]$, the sequence $[3, 5]$ is one possible increasing subsequence of length 2, while $[1, 4, 7, 11, 13]$ is an increasing subsequence of length 5.

   We want to find an algorithm that, given A, finds in polynomial time the longest increasing subsequence of A.

   a) Consider the following attempt at a dynamic programming solution. We use a vector $M[1 \ldots n]$, with the intended meaning that $M[k]$ contains the length of the longest increasing subsequence of $a_1, \ldots, a_k$. After that vector is filled, the solution to our problem is contained in $M[n]$. We give the following recursive formula for M. We set the base case to be $M[1] = 1$. Then $M[i] = M[i-1]$ if $a_{i-1} > a_i$ and $M[i] = M[i-1] + 1$ otherwise.

   Prove that this algorithm is wrong by describing a counter-example. That is, describe an input instance A such that the above recursive rules construct a vector M that does not contain the right entries and such that the last entry of M is not the length of the longest increasing subsequence of A.

   (Hint: it is possible to find such a counter-example where A has 4 entries, the longest subsequence has length 2 but the output of the incorrect program gives 3.)

b) Describe a dynamic programming algorithm that correctly computes the length of the longest increasing subsequence of A. Find a polynomial time algorithm (running time $O(n^2)$ will do).

Specify: what the dynamic programming matrix (or vector) contains, how big it is, how to retrieve the final answer, what one fills in first, and finally give a recursive formula for the other entries. The total time should be the number of entries times the time it takes to fill in each entry in the table.

5. Suppose you want to drive from Atlanta to Los Angeles on I-40. Your car holds C gallons of gas and gets m miles to the gallon. You are handed a list of the n gas stations that are on I-40 and the price for gas at each. Let d_i be the distance of the ith gas station from Atlanta, and let c_i be the cost of gasoline at the ith station. You man assume that for any two stations i and j, the distance $|d_i - d_j|$ between them is divisible by m. You start out with an empty tank at station 1. Your final destination is gas station n. You may not run out of gas between stations, but you need not fill up when you stop at a station. (For example, you might decide to purchase only 1 gallon at a given station.)

   Find a polynomial-time dynamic programming algorithm to output the minimum gas bill to cross the country. Analyze the running time of your algorithm in terms of n and C. You do not need to find the most efficient algorithm.

6. Consider the following card game. A dealer produces a sequence $s_1, \ldots, s_n$ of cards, face up, where s_i has a positive value v_i and n is even. Then two players take turns taking a card from the sequence, but each player can only take the first or last card of the remaining sequence of cards. The goal is to collect cards of maximal total value.

   a) Find a sequence of cards such that it is not optimal for the first player to start by choosing the available card of largest value.

   b) Give an $O(n^2)$ algorithm that computes an optimal strategy for the first player. Given the initial sequence, your algorithm should pre-compute in $O(n^2)$ time some information, and then the first player should be able to make each move optimally in $O(1)$ time by looking up the pre-computed information.

   c) Show that the first player always has a non-losing strategy. Notice that just because he has an optimal means of play (part (b)) does not mean that he can always win. I want you to give an argument, not based on dynamic programming, that convinces me that player 1 has a strategy which will always tie or win. The optimal strategy has to be at least this good, so the optimal strategy will also guarantee that player 1 cannot lose.