CSCI-GA 3520: Honors Analysis of Algorithms

Final Exam: Fri, Dec 18 2015, Room WWH-102, 11:00-3:00pm.

- This is a four hour exam. There are six questions, worth 10 points each. Answer all questions and all their subparts.
- This is a closed book exam. No books, notes, reference material, either hard-copy, soft-copy or online, is allowed.
- Please print your name and SID on the front of the envelope only (not on the exam booklets). Please answer each question in a separate booklet, and number each booklet according to the question.
- Read the questions carefully. Keep your answers legible, and brief but precise.
- Assume standard algorithms and results.
- You must prove correctness of your algorithm and prove its time bound unless stated otherwise. The algorithm can be written in plain English (preferred) or as a pseudo-code.

Best of luck!

Problem 1

Recall that a binary counter is an array A where $A[i] \in \{0, 1\}$ for $i = 0, 1, \ldots$. Such an array represents the number $X = \sum_i 2^i A[i]$.

Initially, all array entries are zero. An increment operation updates the array, so that if A represents X before the operation, it represents $X + 1$ after the operation. We can implement the increment operation as follows:

$i \leftarrow 0$
while $A[i] = 1$ do
    $A[i] \leftarrow 0$
    $i \leftarrow i + 1$
$A[i] = 1$

We can measure the *cost* of a single increment operation as the number of entries in A whose value changes during the execution of the above algorithm. It can be shown that performing a sequence of n increment operations has a cost of at most $2n$ — you are NOT being asked to show this here.

Now suppose we want to implement a counter that supports both increment and decrement operations. Assuming that A represents $X > 0$ before the decrement operation, it should represent $X - 1$ after the operation. The following algorithm would work:

$i \leftarrow 0$
while $A[i] = 0$ do
    $A[i] \leftarrow 1$
    $i \leftarrow i + 1$
$A[i] = 0$

(a) Just as for increment, the cost of a decrement operation is the number of entries in A that change. Suppose we allow arbitrary sequences of interleaved increment and decrement operations, starting from $X = 0$ (but requiring that $X \geq 0$ throughout).. Show that in the worst case, such sequences cost $\Omega(n \log n)$, where n is the number of increment/decrement operations.

(b) To fix the problem identified in part (a), we may use a *redundant binary counter*. Such a counter is an array A where $A[i] \in \{-1, 0, 1\}$ for $i = 0, 1, \ldots$. As before, such an array represents the number $X = \sum_i 2^i A[i]$. Note that nonzero numbers may have more than one representation.

Show how to implement increment and decrement in such a way that any sequence of n interleaved increment and decrement operations, starting from $X = 0$, has a cost that is $O(n)$. Again, the cost for an operation is the number of entries in A that change.

HINT: Use a credit argument, or some other type of amortized analysis.

Problem 2

Let $h : \mathcal{U} \to \{0, \ldots, m-1\}$ be a hash function, mapping from some universe $\mathcal{U}$ of data items to a set of slots $\{0, \ldots, m-1\}$. For a finite set $S \subseteq \mathcal{U}$ and an element $a \in S$, we say that h *isolates* a *in* S if the only element of S that hashes to the slot $h(a)$ is a itself, i.e.,

$$\text{for all } b \in S: \quad h(a) = h(b) \implies a = b.$$

Now recall the notion of a *perfect* hash function. Using the above terminology, we can say that h is a perfect hash function for S if h isolates every element of S. Consider the following, weaker property: let us say that h is a *pretty good hash function for* S if h isolates at least $|S|/2$ elements of S.

Your task is to design an efficient, probabilistic algorithm that takes as input a set $S = \{a_1, \ldots, a_n\}$ of n distinct items, and finds a hash function that is pretty good for S.

To this end, assume that for every positive integer m, there exists a universal family $\mathcal{H}_m$ of hash functions mapping from $\mathcal{U}$ to $\{0, \ldots, m-1\}$. You may assume that you can choose $h \in \mathcal{H}_m$ uniformly at random in time $O(1)$, and that you can evaluate h at any point $a \in \mathcal{U}$ in time $O(1)$.

On input S as above, your algorithm should find m and $h \in \mathcal{H}_m$ such that h is pretty good for S. The value m and the expected running time of your algorithm should be $O(n)$.

NOTE: recall that "$\mathcal{H}_m$ is universal" means that for every $a, b \in \mathcal{U}$ such that $a \neq b$, if we choose $h \in \mathcal{H}_m$ uniformly at random, then $h(a) = h(b)$ with probability at most $1/m$.

HINT: Markov's inequality may be helpful.

Problem 3

We say that a sequence of real numbers $a_1, \ldots, a_k$ is a *track* if consecutive numbers differ by at most 1, i.e., $|a_i - a_{i+1}| \leq 1$ for all $i = 1, \ldots, k-1$. For example,

$$9,\ 8,\ 8.9,\ 9.5,\ 10.4,\ 10,\ 10$$

is a track.

(a) Give an algorithm that takes as input a sequence $x_1, \ldots, x_n$ of n real numbers and outputs the length of the longest subsequence $x_{i_1}, \ldots, x_{i_k}$ (where $1 \leq i_1 < i_2 < \cdots < i_k \leq n$) that forms a track. Your algorithm should run in time $O(n^2)$.

What is the space requirement of your algorithm?

(b) Give an algorithm for the same task with the improved running time of $O(n)$, assuming that $x_i \in \{0, \ldots, 42n\}$ for $i = 1, \ldots, n$.

Problem 4

You are given a directed acyclic graph $G = (V, E)$ with n vertices and m edges, along with two distinguished vertices $s, t \in V$. At each vertex $v \in V$, there are a number of stones $q(v)$ (so $q(v)$ is a *nonnegative integer*). Your goal is to find a path from s to t, picking up stones along the way, and arrive at t with as many stones as possible. At each node v along the path, you are allowed to pick up at most $q(v)$ stones (and for simplicity, assume that $q(t) = 0$). However, there is a complication: each edge $e \in E$ has a capacity $c(e)$ (which is a positive integer), and you are not allowed to carry more than $c(e)$ stones across that edge.

Design an efficient algorithm to solve this problem. The input is G (in sparse representation) and vectors q and c representing the stone counts and capacities as above. The output should be an optimal path from s to t, and for every vertex along the path, your output should include the number of stones to be picked up at that vertex. Include an analysis of the running time. For full credit, your algorithm should run in time $O(n + m)$.

NOTE: You may want to first solve a variant of the above problem in which at each vertex, in addition to picking up stones, you may also throw some stones away. Partial credit will be given for a solution to this variant.

Problem 5

The city of Brombus has n subway stations $v_1, \ldots, v_n$ and k subway systems $B_1, \ldots, B_k$. Each system B_j has the following structure: there is a fixed entrance fee p_j to enter B_j (from any station v_i), and there is a set of fares $w_j(v_i, v_t)$ to go from station v_i to station v_t on system B_j. We assume that each entrance fee p_j is a positive number, and that each fare $w_j(v_i, v_t)$ is a positive number or ∞.

When traveling from v_i to v_t from the street, one can enter any subway system B_j, pay admission p_j, travel to some intermediate city v_ℓ by paying the fare $w_j(v_i, v_\ell)$, and then (if necessary) repeat the same process, using either a different subway system (and paying its entrance fee), or using the same subway system (without needing to pay the entrance fee again), until one reaches the desired destination v_t.

Design an algorithm that takes as input the entrance fee and fare data as above, and outputs a table of values M_{it} for $i, t = 1, \ldots, n$, where M_{it} is the minimum cost required to get from v_i to v_t. Your algorithm *does not* need to compute the corresponding routes. State its complexity as a function of n and k.

Note: *You will given significant partial credit for any solution that runs in time polynomial in n and k, but for full credit, your algorithm should run in time cubic in n and linear in k.*

Problem 6

A 3CNF formula is a conjunction of clauses where each clause is the disjunction of *exactly* three distinct literals. We say that a 3CNF formula over variables $x_1, \ldots, x_n$ is *satisfiable* if there exists an assignment of values to $x_1, \ldots, x_n$ from $\{0, 1\}$ that satisfies all clauses in the formula. We define 3SAT as the language of all satisfiable 3CNF formulas. You can assume that 3SAT is NP-complete.

Say that a 3CNF formula is *joker-satisfiable* if there exists an assignment of values to $x_1, \ldots, x_n$ from $\{0, 1, *\}$ (with $*$ denoting a "joker" or "wildcard"), with at most one variable being assigned a joker, that satisfies all clauses in the formula. Here, a clause is satisfied if either

- at least one of its variables is assigned a joker, or
- all of its variables are assigned 0 or 1, and the clause is satisfied in the usual sense.

Let J3SAT be the language of all joker-satisfiable 3CNF formulas. Prove that J3SAT is NP-complete.