1 Final Exam: Data Structures, Optimization, Computational Models

This section contains problems that may appear on the **Final Exam**.

1. (*, HW2) Give a data structure that maintains the product of a set of n non-negative integers mod M under modifications in $O(\log n)$ time per update, and only performs multiplications modulo M on non-negative integers of value at most $O(n^{10}M^{10})$.

   Note that M may not be prime, and the intended solution does not use any number theory beyond the fact that $(a \cdot b) \equiv ((a \mod M) \cdot b) \equiv ((a \mod M) \cdot (b \mod M)) \pmod M$.

2. (*, HW2) Give an algorithm that takes a length n array of 2-tuples (a_i, b_i) and computes for **each** i whether there is some $j < i$ such that i is larger than j in both attributes, aka. $a_i > a_j$ and $b_i > b_j$, in a total time of $O(n \log n)$.

3. (**, Oral HW2) Give a data structure that tracks the sum of all positive weights in a nonnegative array undergoing range additions/subtractions in $O(\log n)$ time. As input, you are given an array $w[1], \ldots, w[n]$ of *weights* and an initial array $a[1], \ldots, a[n]$ of all 0's. We want to support the following updates to a.

   (a) RangeAdd(ℓ, r, x): for all $\ell \le i \le r$, set $a[i] \leftarrow a[i] + x$. Here, x is a (possibly negative) integer, but **it is guaranteed that $a[i] \ge 0$ at all times.**

   (b) PositiveWeight(): Find the sum of $w[i]$ over all indices i where $a[i] > 0$.

   Build a data structure that supports the above two operations in $O(\log n)$ time per operation.

4. (**, Oral HW2) Give an algorithm that reports all ways of k-mismatching a pattern string s in a text string t in $O(|t| \cdot k \log |s|)$ time. The inputs are strings s and t of lengths n and m respectively, where $n \le m$, and a positive integer k. Find the number of indices $i \in \{1, 2, \ldots, m - n + 1\}$ such that the number of mismatches between the strings s and $t[i, i + 1, \ldots, i + n - 1]$ is at most k. The desired runtime is $O(mk \log n)$.

5. (* HW 3 p1, with DP removed) Given an array $a[1 \ldots n]$ of length n, as well as two monotone sequences of left and right indices $1 \le l[1] \le l[2] \le \ldots \le l[n]$, $1 \le r[1] \le r[2] \le \ldots \le r[n]$, such that $l[i] \le r[i]$. Compute, for each i, the range minimum/maximum in $a[l[i] \ldots r[i]]$ in $O(n)$ time.

6. (** HW 3, upgraded) In an undirected graph with colors on the n vertices undergoing m edge insertions, track the number of pairs of vertices of the same color that can reach each other in time $O((m + n) \log^2 n)$ (or better). Note that there may be up to n colors: Quiz 3/HW 3 asked for the version with 2 colors.

7. (** HW4, with story involving foxes removed) Given sets A, B, C, each of size n, as well as some admissible pairs $E \subseteq A \times B$ and $F \subseteq B \times C$, compute, using a single maximum-flow call, the maximum number of pairwise disjoint triples a_i, b_i, c_i such that $(a_i, b_i) \in E$ and $(b_i, c_i) \in F$ for all i.

8. (***, Recitation 7) Give an $O(m \log n)$-time algorithm to decide whether an undirected graph with edges weighted either 0 or 1 has a spanning tree with weight exactly k. Justify the correctness of this algorithm.

9. (* HW5) Let G be a directed graph with m edges whose edges have positive lengths. Let s be a source vertex, and let v_1, v_2, ..., v_k be other vertices given as input. Your goal is to return edge-disjoint paths P_1, P_2, ..., P_k, where P_i is from s to v_i, and minimize the total sum of their lengths. Formulate this as a linear program in standard form.

10. (* Oral HW3) Consider the following flow-based view of Markov Decision Processes:

    (a) Flows still need to be in the direction of the edge, and must respect capacities.

    (b) There are now two types of vertices. The ones from max-flow / min-cost flow are now "conservation vertices," where the incoming flow can be distributed arbitrarily among the outgoing edges.

    (c) There are also "random walk" vertices, which have out-degree 2: any flow that arrives at such a vertex gets distributed evenly among the two outgoing edges.

    Again, we want to route one unit of flow from s to t, satisfying conservation at all other vertices, while minimizing the total cost on the edges, subject to capacity constraints on edges.

    Formulate this as a linear program in standard form.

11. (*, Recitation 8) Show that in a directed graph with capacity *cap*, the fractional s-t minimum-cut linear program can be written as

$$\min_{\substack{x \in \mathbb{R}^V, x_s = 0, x_t = 1 \\ y \in \mathbb{R}^E, y \geq 0 \\ y \geq Bx}} \text{cap}^\top y \tag{1}$$

    where recall $B \in \mathbb{R}^{m \times n}$ is the edge-vertex incidence matrix, with an edge $e = (u \to v)$ corresponding to a row with $B_{e,u} = -1$, $B_{e,v} = 1$, and 0 everywhere else.

12. (**, HW5) Let G be a directed graph, and let (s_1, t_1), ..., (s_k, t_k) be pairs of sources and sinks. We want to send one unit of flow for each pair $s_i \to t_i$ (**Important: the flows need not be integral.**) in a way that minimizes the maximum congestion

on any edge. Formally, let $f^{(1)}, \ldots, f^{(k)}$ be the flows. Then the congestion of edge e is defined as

$$\text{cong}_e = \sum_{i=1}^{k} f_e^{(i)},$$

where $f_e^{(i)}$ denotes the amount of flow on edge e in $f^{(i)}$. You want to find flows $f^{(i)}$, each routing one unit from s_i to t_i, that minimize $\max_e \text{cong}_e$.

13. (*, HW6) Show that the weighted majority algorithm works when the outcome space is no longer binary, with the same bounds as in the binary case. More precisely, the number of errors the algorithm makes is at most $O(M + \log n)$, where M is the number of errors that the best expert made. That is, there is now a space of k outcomes, and you (or an expert) are correct only if the outcome picked is the true one.

    note: although this was acceptable on homework 6, the intended constants did not depend on k. Refer to the solution for how this is done.

14. (**, Recitation 19) Consider the following algorithm, starting with a directed graph with source s, sink t, and edges with capacity 1.

    (a) Initialize weights on edges to 1.

    (b) Repeatedly:

        i. Send 1 unit of flow along shortest $s \to t$ path (in terms of weights)

        ii. Double all weights on the path found.

    Repeat this for $T = 10F \log_2 n$ iterations. Prove that the amount of flow sent on any edge is at most $O(\log n)$.

15. (**, HW6) In this problem we consider caching as an online algorithm. There are n items (numbered $1 \ldots n$ for convenience), and you have a cache of size k, with which you can keep k of them. The process then proceeds in rounds (similar to the 'days' in rent-or-buy): every turn a number comes up, if it's not in the cache, you need to bring it into the cache at cost 1. Also, if the cache is now full, you need to remove something form cache (could be the item you just brought in).

    Prove that when $k = n - 1$, no algorithm can achieve a competitive ratio less than $(n - 1)$.

16. (**, Oral HW4) Consider the even more limited query oracle of $\textsc{Same}(i, j)$, which returns true if $A[i] = A[j]$, false otherwise. Show that:

    (a) one needs at least $\Omega(n^2)$ calls to $\textsc{Same}$ to decide whether n numbers are all distinct. (Hint: it's much easier to build an adversary that constructs a ground truth based on the queries asked)

(b) one can decide whether there is a majority element, i.e., one that appears more than $n/2$ times, using $O(n)$ queries to SAME.

17. (**, Oral HW4) Recall that an increasing subsequence of an array $A[1 \dots n]$ is a subsequences of indices $i_1 < i_2 < \dots < i_l$ such that $A[i_1] < A[i_2] < \dots < A[i_l]$. Show that the number of comparisons needed to sort a length n array whose elements are all distinct, but with longest increasing subsequence at most t is $\Theta(n(1 + \log t))$ (both asymptotic upper and lower bounds).

    You may assume the algorithm has access to t, and the output of the sort is defined to be a permutation under which the input is in non-decreasing order.

    Note that you only need to optimize the number of comparisons, not the runtime of your algorithm.

18. (*, Lecture 20) Understand why:

    (a) Sorting in the comparison model requires $\Omega(n \log n)$ queries (using decision trees).

    (b) Finding the k-th smallest element can be done deterministically in $O(n)$ queries using the median-of-medians algorithm for selecting a pivot.

19. (Oral HW4, ***) There is an $n \times n$ grid with all cells set to 0, except non-touching bars of 1s, one horizontal and one vertical. Formally, the two bars some continuous group of at least two 1s in some row, and some continuous group of at least two 1s in some column, and the non-touching condition means that no 1s in either bar touch (share a side, or a corner).

    You can query the grid using queries of 'does this sub-rectangle of the grid contain some 1', and need to find the precise locations of the two bars.

    Show upper/lower bounds of $6 \log_2 n \pm \Theta(1)$ queries.