15-451/651 Algorithm Design & Analysis, Spring 2026

Homework #2

Homework #2, Due: Sunday, Feb 1 at 3:00 pm

Guidelines:

1. This homework has three problems, all graded on completion.
2. In all cases the write-up you submit must be written by you alone. You may not share your written solutions with each other.
3. You can discuss the problem as a group, or consult the internet (in fact, references are provided directly when possible). However, you then must write up your final solutions independently in your own words.
4. When asked to "give an algorithm" or "show a runtime", you should:
   (a) describe your algorithm in English **OR** pseudocode,
   (b) provide a short argument for correctness and running time,
   (c) work with arbitrary, unknown, inputs, instead of on a single instance/example.
5. For all dynamic programming problems, you should clearly state the states, transitions, and base cases.

Submission: Submit to `gradescope` using join code that you received by email: please contact us if you did not find this code, or have trouble joining the course.

Your solutions **must** be typeset, **not handwritten**, and submitted as a PDF file. You should use US letter-size paper, a font size no smaller than 10pt, margins no smaller than 1in.

1. (*, dynamic product maintenance mod arbitrary M)
   (a) Show that given a tree where each node p is associated with a non-negative integer x_p, and a fixed modulus M, the product of each subtree mod M,
   $$\prod_{q \in \textsc{SubTree}(p)} x_q,$$
   for all nodes p, can be computed in $O(n)$ time.
   (b) Using the DP state from the above part, or some other method of your choice, 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. (** weighted k-independent set) Give an algorithm that takes a tree with weights on the vertices, returns the maximum weight of a subset of exactly k vertices such that no two vertices in the set are adjacent, in time $O(n^{10}k^{10})$.
   Note 1: In an undirected graph, two vertices are adjacent if they are the end points of some edge. A subset of nodes of a rooted tree, S, have no pairs adjacent iff for all $p \in S$, $\textsc{Parent}(p) \notin S$.
   Note 2: the degrees of the tree nodes here can be arbitrary: up to $n-1$.
   Note 3: One possible simplification is to start with a rooted tree where each node has at most two children. Another possible simplification is to remove the restriction of having k vertices.

3. (** pareto optimum points) 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 $O(n \log n)$.