Homework 3

Course: Algorithm Design and Analysis **Semester**: Spring 2026

Instructor: Shi Li **Due Date**: **May 17, 2026**

Student Name: _____ Student ID: _____

Problems	1	2	3	4	5	6	Total
Max. Score	15	15	20	20	20	15	105
Your Score							

The sum of maximum scores over all problems is 105, but your score will be capped at 100. For the dynamic programming problems, it suffices for you to describe the cells (i.e., sub-problems), how to compute each cell, and give the running time.

Dynamic Programming

Problem 1. You are stress-testing glass jars to determine the highest rung on an n-rung ladder from which a jar can be dropped without breaking. We call this the *highest safe rung*. If a jar is dropped from a rung and does not break, it can be reused; if it breaks, it is discarded.

There is a clear trade-off between the number of jars you are willing to break and the number of drops required. If you have only one jar, you must start from the first rung and go up one by one (requiring up to n drops). If you have infinite jars, you can use binary search (requiring $\lceil \log_2 n \rceil$ drops).

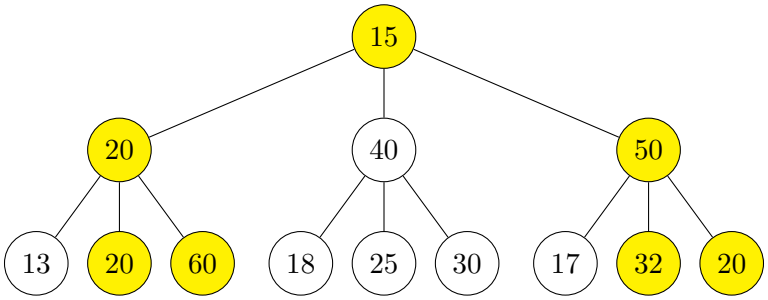
Design a dynamic programming algorithm to find the minimum number of drops required in the worst case to determine the highest safe rung, given n rungs and a budget of k jars. The highest safe rung can be any integer between 0 and n, inclusive, as it is possible that all rungs are safe or all are unsafe.

Example 1: Suppose $n = 10$ and $k = 1$. You only have one jar. To ensure you find the highest safe rung without running out of jars before the answer is found, you must drop the jar from rung 1, then 2, and so on. In the worst case, you will need 10 drops.

Example 2: Suppose $n = 14$ and $k = 2$. You can do better than sequential testing. If you drop the first jar from rung 5 and it breaks, you use the second jar to check rungs 1–4 (max 5 drops total). If it doesn't break at 5, you drop it again at rung 9 (5+4). If it breaks there, you check rungs 6–8 (max 5 drops total). If it doesn't break at 9, you move to rung 12 (9+3), and finally 14 (12+2). In all scenarios, you can find the answer in at most 5 drops.

Example 3: Suppose $n = 7$ and $k = 3$. With enough jars to perform binary search, you drop the first jar from rung 4. If it breaks, you have 2 jars left for 3 rungs. If it doesn't, you have 3 jars left for the upper 3 rungs. In this case, the result is $\lceil \log_2 8 \rceil = 3$ drops.

Problem 2. Let $T = (V, E)$ be a tree rooted at r and $w : V \to \mathbb{R}_{>0}$ be a weight function on vertices of T. Your goal is to compute the maximum-weight subgraph T' of T, such that T' is a tree rooted at r where every node has at most 2 children. Give a dynamic programming algorithm that computes the weight of T'. For example, for the following tree T, the maximum-weight subgraph T' satisfying the requirement has weight 227, as indicated by the yellow vertices.



For simplicity, you may assume $V = \{1, 2, 3, \cdots, n\}$ and a child i of a vertex j has $i < j$. (Thus, the root is n).

Problem 3. Given an array A of n numbers, we say that a 5-tuple $(i_1, i_2, i_3, i_4, i_5)$ of integers is inverted if $1 \le i_1 < i_2 < i_3 < i_4 < i_5 \le n$ and $A[i_1] > A[i_2] > A[i_3] > A[i_4] > A[i_5]$.

(3a) Give an $O(n^2)$-time algorithm to count the number of inverted 5-tuples w.r.t A.

(3b) Give an $O(n \log n)$-time algorithm to count the number of inverted 5-tuples w.r.t A.

Graph Algorithms

Problem 4. Solve the following two problems. You may use the algorithms you learned in class as a black-box, without repeating their details.

(4a) Given a graph $G = (V, E)$ with edge weights $c_e \in \mathbb{Z}_{\ge 0}, e \in E$, design an $O(n \log n + m)$-time algorithm to decide if a minimum spanning tree of G is unique or not.

(4b) Given a directed graph $G = (V, E)$ with two vertices $s, t \in V$ and edge weights $c_e \in \mathbb{Z}_{\ge 0}, e \in E$, design an $O(n \log n + m)$-time algorithm to decide if the shortest path from s to t in G is unique or not.

Problem 5. Suppose you are given a directed graph $G = (V, E)$ with costs on the edges c_e for $e \in E$ and a source s (costs may be negative). Assume that you also have finite values $d(v)$ for $v \in V$. Someone claims that, for each node $v \in V$, the quantity $d(v)$ is the cost of the minimum-cost path from the source s to the node v.

(5a) Give a linear-time algorithm (time $O(m)$ if the graph has m edges) that verifies whether this claim is correct.

(5b) Assume that the distances are correct, and $d(v)$ is finite for all $v \in V$. Now you need to compute distances from a different source s'. Give an $O(n \log n + m)$-time algorithm for computing distances $d'(v)$ from the source s' to all nodes $v \in V$. (Hint: it is useful to consider a new cost function.)

Problem 6. Consider the minimum cost arborescence problem on the directed graph $G = (V, E)$ with non-negative edge weights $(w_e)_{e \in E}$ and a specified root r. Let C be a 0-cost simple cycle in G that does not contain r. Prove the statement that we skipped in class: there exists a minimum cost arborescence T in G (rooted at r) that includes all but one edge of C.