CS 170 Homework 8

Due **10/25/2023, at 10:00 pm (grace period until 11:59pm)**

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, you must explicitly write "none".

2 Egg Drop Revisited

Recall the Egg Drop problem from Homework 7:

You are given m identical eggs and an n story building. You need to figure out the highest floor $\ell \in \{0, 1, 2, \ldots n\}$ that you can drop an egg from without breaking it. Each egg will never break when dropped from floor ℓ or lower, and always breaks if dropped from floor $\ell + 1$ or higher. ($\ell = 0$ means the egg always breaks). Once an egg breaks, you cannot use it any more. However, if an egg does not break, you can reuse it.

Let $f(n, m)$ be the minimum number of egg drops that are needed to find ℓ (regardless of the value of ℓ).

Instead of solving for $f(n, m)$ directly, we define a new subproblem $M(x, m)$ to be the maximum number of floors for which we can always find ℓ in at most x drops using m eggs.

For example, $M(2, 2) = 3$ because a 3-story building is the tallest building such that we can always find ℓ in at most 2 egg drops using 2 eggs.

(a) Find a recurrence relation for $M(x, m)$ that can be computed in constant time given the previous subproblems. Briefly justify your recurrence.

Hint: As a starting point, what is the highest floor that we can drop the first egg from and still be guaranteed to solve the problem with the remaining $x - 1$ drops and $m - 1$ eggs if the egg breaks?

(b) Give an algorithm to compute $M(x, m)$ given x and m and analyze its runtime.

(c) Modify your algorithm from (b) to compute $f(n, m)$ given n and m.

Hint: If we can find ℓ when there are more than n floors, we can also find ℓ when there are n floors.

(d) Show that the runtime of the algorithm from part (c) is $O(nm)$. Compare this to the runtime you found in last week's homework.

(e) Show that we can implement the algorithm from part (c) to use only $O(m)$ space.

(f) Suppose that we are given a special machine that is able to "revive" an egg after the first time it breaks so that it is reusable again. In other words, each egg has 2 lives. Based on this modification to the problem, write a new recurrence relation for M.

Hint: you may need to add an additional parameter to your subproblem.

3 Knightmare

Give a dynamic programming algorithm to find the number of ways you can place knights on an L by H ($L < H$) chessboard such that no two knights can attack each other (there can be any number of knights on the board, including zero knights). Knights can move in a 2×1 shape pattern in any direction.

Provide a 4-part solution. Your algorithm's runtime should be $O(2^{3L}LH)$, and return your answer mod 1337.

Hint: if a knight is on row i, what rows on the chessboard can it affect?

4 Max Independent Set Again

You are given a connected tree T with n nodes and a designated root r, where every vertex v has a weight $A[v]$. A set of nodes S is a k-independent set of T if $|S| = k$ and no two nodes in S have an edge between them in T. The weight of such a set is given by adding up the weights of all the nodes in S, i.e.

$$w(S) = \sum_{v \in S} A[v].$$

Given an integer $k \leq n$, your task is to find the maximum possible weight of any k-independent set of T. We will first tackle the problem in the special case that T is a binary tree, and then generalize our solution to a general tree T.

(a) Assume that T is a binary tree, i.e. every node has at most 2 children. Describe an $O(nk^2)$ algorithm that solves this special case, and analyze its runtime. Proof of correctness and space complexity analysis are not required.

(b) Now, consider any arbitrary tree T, with no restrictions on the number of children per node. Describe how we can add up to $O(n)$ "dummy" nodes (i.e. nodes with weight 0) to T to convert it into a binary tree T_b.

(c) Using your responses to parts (a) and (b), describe an $O(nk^2)$ algorithm to solve the general case (i.e. when T is any arbitrary tree), and analyze its runtime. Proof of correctness and space complexity analysis are not required.

5 Coding Questions

For this week's coding questions, we'll implement dynamic programming algorithms to solve two classic problems: **Weighted Independent Set in a Tree** and **TSP**. There are two ways that you can access the notebook and complete the problems:

1. **On Local Machine**: `git clone` (or if you already cloned it, `git pull`) from the coding homework repo,

   https://github.com/Berkeley-CS170/cs170-fa23-coding

   and navigate to the `hw08` folder. Refer to the `README.md` for local setup instructions.

2. **On Datahub**: Click here and navigate to the `hw08` folder if you prefer to complete this question on Berkeley DataHub.

Notes:

- *Submission Instructions:* Please download your completed submission `.zip` file and submit it to the Gradescope assignment titled "Homework 8 Coding Portion".
- *OH/HWP Instructions:* Designated coding course staff will provide conceptual and debugging help during office hours and homework parties.
- *Edstem Instructions:* Conceptual questions are always welcome on the public thread. If you need debugging help first try asking on the public threads. To ensure others can help you, make sure to:

  1. Describe the steps you've taken to debug the issue prior to posting on Ed.
  2. Describe the specific error you're running into.
  3. Include a few small test cases, alongside both the output you expected to receive and your function's actual output.

  If staff tells you to make a private Ed post, make sure to include *all of the above items* plus your full function implementation. If you don't provide them, we will ask you to provide them.

- *Academic Honesty Guideline:* We realize that code for some of the algorithms we ask you to implement may be readily available online, but we strongly encourage you to not directly copy code from these sources. Instead, try to refer to the resources mentioned in the notebook and come up with code yourself. That being said, we **do acknowledge** that there may not be many different ways to code up particular algorithms and that your solution may be similar to other solutions available online.