CS 170 Homework 5

Due **3/1/2022, 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 Copper Pipes

Bubbles has a copper pipe of length n inches and an array of nonnegative integers that contains prices of all pieces of size smaller than n. He wants to find the maximum value he can make by cutting up the pipe and selling the pieces. For example, if length of the pipe is 8 and the values of different pieces are given as following, then the maximum obtainable value is 22 (by cutting in two pieces of lengths 2 and 6).

length	1	2	3	4	5	6	7	8
price	1	5	8	9	10	17	17	20

Give a dynamic programming algorithm so Bubbles can find the maximum obtainable value given any pipe length and set of prices. Clearly describe your algorithm, prove its correctness and runtime.

Solution:

Main idea: We create a recursive formula, where for each subproblem of length k we choose the cut-length i such that $Price(i) + Value(k-i)$ is maximized. Here $Price(i)$ is the price of selling the full pipe of length i and $Value(k-i)$ is the amount obtained after optimally cutting the pipe of length $k-i$.

```
def cutPipe(price, n):
    val = [0 for x in range(n+1)]
    val[0] = 0

    # Build the table val[] in bottom up manner and return
    # the last entry from the table
    for i from 1 to n:
        max_val = 0
        for j from 0 to i-1:
             max_val = max(max_val, price[j] + val[i-j-1])
        val[i] = max_val

    return val[n]
```

Proof: An inductive proof on the length of the pipe will show that our solution is correct. We let $cutPipe(n)$ represent the optimal solution for a pipe of length n. Base case: If the pipe is length 1, $cutPipe(1) = Price(1) = Val(1)$. Inductive: Assume the optimal price is

found for all pipes of length less than or equal to k. If the first cut the algorithm makes x_1 is not optimal, then there is an x_1' such that $Val((k+1)-x_1)+Price(x_1) < Val((k+1)-x_1')+Price(x_1')$. By the induction hypothesis, this implies that $cutPipe((k+1)-x_1)+Price(x_1) < cutPipe((k+1)-x_1')+Price(x_1')$. So the algorithm must have chosen x_1' instead of x_1, by construction (a contradiction). Therefore, by induction $cutPipe(n) = Val(n)$ for all $n > 0$.

Run-time: The algorithm contains two nested for-loops resulting in a run-time of $O(n^2)$.

3 Egg Drop

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

(a) Find $f(1, k)$, $f(0, k)$, $f(n, 1)$, and $f(n, 0)$.

(b) Find a recurrence relation for $f(n, k)$. *Hint: Whenever you drop an egg, call whichever of the egg breaking/not breaking leads to more drops the "worst-case event". Since we need to find ℓ regardless of its value, you should assume the worst-case event always happens.*

Solution:

(a) We have that:

- $f(1, k) = 1$, since we can drop the egg from the single floor to determine if it breaks on that floor or not.
- $f(0, k) = 0$, since there is only one possible value for ℓ.
- $f(n, 1) = n$,since we only have one egg, so the only strategy is to drop it from every floor, starting from floor 1 and going up, until it breaks.
- $f(n, 0) = \infty$ for $n > 0$, since the problem is unsolvable if we have no eggs to drop.

(b) The recurrence relation is

$$f(n, k) = 1 + \min_{x \in \{1 \ldots n\}} \max\{f(x-1, k-1), f(n-x, k)\}.$$

Consider dropping an egg floor x when there are n floors and k eggs left. If the egg breaks, we only need to consider floors 1 to $x-1$, and we have $k-1$ eggs left since an egg broke, in which case we need $f(x-1, k-1)$ more drops. If the egg doesn't break, we only need to consider floors $x+1$ to n, and there are k eggs left, so we need $f(n-x, k)$ more drops. So in the worst case, we need $\max\{f(x-1, k-1), f(n-x, k)\}$ drops if we drop from floor x. Then, the optimal strategy will choose the best of the n floors, so we need $\min_{x \in \{1 \ldots n\}} \max\{f(x-1, k-1), f(n-x, k)\}$ more drops.