Homework 4 – COT5405 *

Kobee Raveendran

November 2021

1. **Peter is the owner of a Japanese sushi restaurant and today he invites you for dinner at his restaurant. In front of you are n sushi dishes that are arranged in a line. Each dish is different and they have different costs. Peter hopes that you can select the dishes you want, starting from the left to the right. However, there is a further restriction: when you select a dish, say A, the next dish you can select must cost higher than A. Design an $O(n^2)$-time algorithm to select the dishes so as to maximize the total costs.**

2. **There is a stair with n stages, and on each stage, there is a coupon to your favorite restaurant. Each coupon has an associated value which may be different. You can climb up 1, 2, or 3 stages in a step. Since you are in a hurry, you need to reach the top of the staircase in at most L steps, where $L \leq n$. When you visit a particular stage, you can collect the coupon on that stage. Note that L is a part of the input (means you cannot fix $L = n$ or so). Find a way to climb the stair within L steps so that you can collect coupons with the maximum total value. Your algorithm should run in $O(n^2)$ time.**

   Note: For this problem, I'm assuming coupon values cannot be negative (i.e. taking a step can ONLY increase your "score"). This means that I will always try to take up to L steps. If negative coupons are allowed, then the solution would have to change, as it may be optimal to take much less than L steps while still reaching the top (i.e. we may need to skip some stairs with negative values).

   This problem can be solved with dynamic programming using a 2D dp matrix. The matrix will by $L \times n$, where L is the max. number of steps we can take and n is the total number of stairs. Our objective function (i.e. what each cell in the dp matrix represents) is simply the maximum cumulative coupon value we've gotten by our L^{th} step at the n^{th} stair. With this in mind, the main idea behind the algorithm is to build the matrix from the top left to the bottom right as we go along, and our answer will be somewhere in the final column of the matrix. If we take L steps, the final answer will be in the bottom right corner (otherwise, if we took less than L steps, we'd have to do a linear scan of the final column). This is because the final column represents our "score" by the time we've reached the top of the staircase (the n^{th} step), and the score we have by the time we've taken our L^{th} step should be the maximum possible.

   Initially, all matrix cells will be initialized to 0, after which we'll fill in our base cases. Since we can take 1, 2, or 3 steps, we can fill in the values for stairs 1, 2, and 3 before starting (these are our base cases). Thus, `dp[1][1]`, `dp[1][2]`, and `dp[1][3]` will be set to the coupon values of stairs 1, 2, and 3 respectively, since those are the stairs we can reach within 1 step. From there, we fill in the dp matrix row by row, by filling a cell with the max value of the previous 3 stairs in the previous step (i.e. the cells up 1 row and left 3 columns) plus the current stair's coupon value. See the example below for a walkthrough (assume the table continues past the bounds shown here, where the rows continue until an arbitrary L and the columns continue until an arbitrary n):

   Assume we have a staircase with $L = 3$ and the following coupon values (in order of stair number): $[3, 7, 9, 5, 1]$

*For some questions, I received guidance from or collaborated with: TA (office hours)

3	7	9	0	0
0	0	0	0	0
0	0	0	0	0

Table 1: Base cases

3	7	9	0	0
0	10	16	14	0
0	0	0	0	0

Table 2: After first outer loop iteration

3	7	9	0	0
0	10	16	14	0
0	0	19	21	17

Table 3: Final iteration (we've reached taken our last possible step)

Our final answer will be in `dp[L][n]`, and we can confirm this is correct (start from the ground (step "0"), go to stair 2 and collect 7, then go to step 3 and collect 9, then go to the final (n^{th}) step and collect 1 for a total of 17).