Question 1: (Compulsory)

You need to write a dynamic programming algorithm to play this game. There are three bowls in this game, each starting with \(X\), \(Y\) and \(Z\) number of balls respectively (\(X\), \(Y\), \(Z\) totally independent). A move constitutes exactly one of the following:

- Remove 1 ball from bowl 1
- Remove 1 or 2 ball(s) from bowl 2
- Remove 2 or 3 balls from bowl 3

Players take alternate moves, and the first player unable to make a move loses. i.e., if it’s a player’s turn to move but the playing board is either (0,0,0) or (0,0,1) balls, he/she loses.

Devise an algorithm that will, given numbers \((X, Y, Z)\), determine who (A or B) is the winner of the game, provided both players play optimally (A is player who makes the first move).

No pseudocode is required - just write the recurrence relation and justify its correctness. Argue that the running time is a polynomial function of \(X\), \(Y\) and \(Z\).

**An Example:** suppose the board starts with (0,1,4). The first player (say A) has a winning strategy here, i.e., she can force B to lose. A can remove two balls from bowl 3, resulting in (0,1,2). Now, B can either remove one ball from bowl 2 or two balls from bowl 3.

- If the board after B’s move is (0,0,2), A removes 2 balls from bowl 3; board is (0,0,0)
- If the board after B’s move is (0,1,0), A removes 1 ball from bowl 2; board is (0,0,0)

In both cases, B is forced to lose even if she plays optimally; which means A is the winner.
Question 2: (Compulsory)

You are given a wooden stick of length $X$ with $m$ markings on it at arbitrary places (integral), and the markings suggest where the cuts are to be made accordingly. For chopping a $L$-length stick into two pieces, the carpenter charges $L$ dollars (does not matter whether the two pieces are of equal length or not, i.e, the chopping cost is independent of where the chopping point is).

Design a dynamic programming algorithm that calculates the minimum overall cost (you do not need to give the order of chopping) and write the recurrence relation. Use tables (memoizations) to show how your algorithm obtains the optimal solution from the sub-problems in a bottom-up fashion and justify its complexity in terms of $m$.

Example: take an 8’’ stick with cut-marks at every inch.

A bad strategy would be to chop it at the 1’’ mark, then 2’’ mark, then 3’’ mark … which will cost $8+7+6+\ldots+2 = $35. The first two steps of this sub-optimal strategy would be like:

Optimal strategy for this case should be like (first three steps):
Question 3:

Given two strings \(X = x_1x_2x_3 \ldots x_m\) and \(Y = y_1y_2y_3 \ldots y_n\), the SCS (Shortest Common Supersequence) \(Z\) is defined by the string of minimum length which has got both \(X\) and \(Y\) as its subsequences (not necessarily substrings). Given strings \(X\) & \(Y\) of lengths \(m\) & \(n\), devise a \(O(mn)\) dynamic programming algorithm to generate the length of \(Z\) (you do not need to generate the actual \(Z\), just its length). Make sure you give the recurrence relation, and justify its correctness.

For example, if \(X = \langle PQRQPQP \rangle\) and \(Y = \langle QRPPQPQ \rangle\), \(Z = \langle PQRPPQPQP \rangle\).

Question 4:

You are given a sorted array of \(n\) distinct integers of sum \(S\) (\(S\) is even). Design a Dynamic programming algorithm to decide if the elements of the array could be partitioned into two groups of equal sum.

For example, \([1,3,4,6,8,14]\) can be partitioned into \((1,3,6,8) \& (4,14)\) while \([3,4,11,16]\) cannot be partitioned.

Question 5:

Optimal Coin-changing: Input is \(<\) a set of denomination of coins \(A[n]\) (coin of each denomination is available in infinite supply) and a target \(C\) \(>\). You need to change \(C\) using minimum number of coins.