CS 336 Assignment 3

February 14th 2025

Problem 1.

Alice plays the game $2 * 3 * 5$. In this game, she moves a piece on a $1 \times n$ board consisting of n squares (see below). There is a number (any integer, not necessarily positive) written in every square of the board. We denote the number in square i by r_i. In the beginning of the game, the piece is located on square 1. At every turn, Alice advances her piece by 2, 3, or 5 squares to the right, and gets a reward for every move she makes. The amount of the reward depends on (1) the number written in the square where the piece stops and (2) the distance the piece travels during the move. Specifically:

- if Alice moves the piece by 2 squares, and the piece lands on the square i, she receives $2r_i$ dollars (if r_i is negative, she loses money);
- if Alice moves the piece by three squares, she receives $3r_i$ dollars;
- if Alice moves the piece by five squares, she receives $5r_i$ dollars.

The game ends when Alice cannot move the piece. Design a dynamic programming algorithm that finds the maximum reward Alice can get. The input of the algorithm is the array $r_1, \ldots, r_n$.

Example: Let $r = [1, 2, 3, 4, 5, 6]$, these are the following possible routes:

- $1 \to 6$ with reward $5 \cdot 6 = 30$ (optimal).
- $1 \to 3 \to 5$ with reward $2 \cdot 3 + 2 \cdot 5 = 16$.
- $1 \to 3 \to 6$ with reward $2 \cdot 3 + 3 \cdot 6 = 24$.
- $1 \to 4 \to 6$ with reward $3 \cdot 4 + 2 \cdot 6 = 24$.

Please do the following:

- Formulate the subproblem. Please state it as precisely as possible.
- Design a dynamic programming algorithm for solving this problem:
  - State the base case.
  - State the recurrence relation.
  - Explain why the recurrence relation is correct (from your explanation, one should understand how to get your recurrence relation).
  - Please provide the **pseudocode**. Please use the bottom-up approach.
  - Explain:
    - What is the running time of your algorithm.
    - How to recover the maximum reward.

Problem 2.

There are n gifts, and you want to buy each gift exactly once. You can buy gift i at cost $cost[i]$. However, you have another option: for any i, you can buy gift i together with gift $i+1$. In this case, to buy both gifts, you pay $bundle[i]$. You can use any number of bundles, as long as you buy every gift exactly once.

Please, implement the following function:

```
vector<int> Purchases(const vector<int>& cost, const vector<int>& bundle)
```

The function should return the vector $result$ consisting of 0, 1 and -1. For each i:

- If you buy element i by itself, then $result[i] = 0$.
- If you buy element i together with $i+1$, then $result[i] = 1$.
- If you buy element i together with $i-1$, then $result[i] = -1$.

For all test cases, it is guaranteed that the optimal solution is unique.

Example If $cost = [1, 3, 4]$ and $bundle = [2, 6]$, then there are 3 possible solutions:

- Buy each gift separately: $1 + 3 + 4 = 8$
- Buy the first two gifts together: $2 + 4 = 6$
- Buy the last two gifts together: $1 + 6 = 7$

The second choice is optimal, so the output is $[1, -1, 0]$.

Time limit The instructions are similar to the previous programming assignments. Your program should pass each tests in no more than 1 second. You can assume that $1 \leq n \leq 2 \cdot 10^5$, and all prices are between 0 and 10^9.