Handson 3

Competitive Programming and Contests

Silvio Martinico

December 14, 2022

Abstract

In this report we present the resolution of two problems: *Holiday Planning* and *Xmas lights*.

Both the solutions rely on *Dynamic Programming*, the first one exploits a Dynamic Programming matrix.

The solutions are implemented in `Rust` and have respectively a time complexity of $\Theta(nD^2)$ (where n is the number of arrays and D is both the length of these arrays and the total number of elements we want to pick from them) and $\Theta(n)$ (where n is the length of the input string).

IN SUPREMÆ DIGNITATIS
1343

Contents

1 Holyday Planning problem

Problem: Given n arrays (cities) of positive integers (attractions) with length D (days), find the maximum value given by the sum of the elements of the prefix of each array (we can also take 0 elements from an array) such that, the sum of the lengths of the chosen prefixes is D.

1.1 Implementation

We fix an arbitrary order for the cities (the input order is fine). The cities were stored in a $n \times (D+1)$ matrix C such that $C[i][j]$ represents the (cumulative) number of attractions a tourist can visit in j days in the i-th city. Note that $j \in \{0, 1, \dots, D\}$ and $C[i][0] = 0 \;\forall\; i = 0, 1, \dots, n-1$.

The answer of the problem is computed via a $n \times (D+1)$ dynamic programming matrix M, were $M[i][j]$ represents the maximum number of attractions a tourist can visit in j days in the first $i+1$ cities. At this point we can define the recurrence:

$$M[i][j] \;:=\; \begin{cases} 0 & \text{if } \; j = 0 \\ C[0][j] & \text{if } \; i = 0 \\ \max\limits_{k \le j} \Big\{ C[i][j] + M[i-1][j-k] \Big\} & \text{if } \; 0 < i < n \,,\; 0 < j \le D \end{cases} \quad .$$

Clearly, we can find the answer of the problem at the position $M[n-1][D]$, which, as we said before, represents the number of attraction a tourist can visit in D (all) days in the first n (all) cities.

1.2 Complexity analysis

In order to assess the space complexity it is sufficient to notice that the input data requires $\Theta(nD)$ space and the only additional space is that of the dynamic programming matrix, which is again $\Theta(nD)$. However, each row of the matrix M only depends on the previous row and, at the end of the algorithm, we only need the last cell of the last row. So, it is sufficient to just store two rows at a time, reducing the **additional** required space to $\Theta(D)$.

Regarding the time complexity, what we do is to load the matrix M one cell at a time, so we have $\Theta(nD)$ iterations. Let's estimate the time complexity of a single iteration.

In a single iteration (i, j) we compare for every $k \le j$ the numbers $C[i][j] + M[i-1][j-k]$s. Since each comparison requires constant time and for a cell in the j-th column we have to do j comparisons, the overall time complexity for filling the cell $M[i][j]$ is $\Theta(j)$. Hence, for computing a single row, the time complexity is:

$$\sum_{j=1}^{D} \Theta(j) \;=\; \Theta\left(\frac{D(D+1)}{2}\right) \;=\; \Theta(D^2)$$

and, since we have n rows, the final asymptotic time complexity is $\Theta(nD^2)$.

2 Xmas lights problem

Problem: For a string S of length n in the alphabet $\Sigma = \{R, W, G, X\}$, we call the occurrences of X *wildcard characters*, as we can substitute them with one of the three characters $\{R, W, G\}$. Given a string $S \in \Sigma^n$, count the total number of *patriotic selections* (ordered triplets (R, W, G)), for each possible (independent) choice of the the wildcards X.

2.1 Resolution of the problem

The algorithm relies on dynamic programming. For each prefix $S[1, i]$ of the given string S, we want to compute the number of its patriotic selections exploiting that of the previous prefix $S[1, i-1]$.

We store four variables (or even an array of length 4) where we save useful information. More precisely, we want to store one data (a positive integer) for each of the steps which leads to a patriotic combination and one for the wildcard characters:

- $\mathcal{R}_i$ indicates, for a fixed index i, the number of R characters in every possible combination (each possible assignment of the wildcard characters) of $S[1, i-1]$;
- $\mathcal{W}_i$ indicates, for a fixed index i, the number of (R, W) pairs in every possible combination (each possible assignment of the wildcard characters) of $S[1, i-1]$;
- $\mathcal{G}_i$ indicates, for a fixed index i, the number of (R, W, G) triplets (i.e. the number of patriotic selections) in every possible combination (each possible assignment of the wildcard characters) of $S[1, i-1]$;
- $\mathcal{X}_i$ indicates, for a fixed index i, the number of X characters in $S[1, i-1]$

What we want to do is to compute the number of patriotic selections of $S[i]$ exploiting the four information above, depending on the character $S[i]$. First, we have to define the base case, i.e. the values $\mathcal{R}_0, \mathcal{W}_0, \mathcal{G}_0, \mathcal{X}_0$. It is sufficient to set all these values to 0.

For $i > 0$, we describe the algorithm by cases, one for each possible value of $S[i]$:

- $S[i] == R \implies \mathcal{R}_i = \mathcal{R}_{i-1} + 1$

  It is sufficient to increment by 1 the value of $\mathcal{R}$ since adding one R to the end of the sequence does not create new pairs (R, W) or triples (R, W, G).

- $S[i] == W \implies \mathcal{W}_i = \mathcal{W}_{i-1} + \mathcal{R}_{i-1}$

  In this case, for each R in $S[1, i-1]$, we can create a new pair (R, W) with the new W.

- $S[i] == G \implies \mathcal{G}_i = \mathcal{G}_{i-1} + \mathcal{W}_{i-1}$

  As before, for each already present pair (R, W), we can make a new patriotic selection using the new G.

- $S[i] == X$
$$\begin{aligned} S[i] == X \implies \mathcal{G}_i &= 3 \cdot \mathcal{G}_{i-1} \\ \implies \mathcal{G}_i &= \mathcal{G}_i + \mathcal{W}_{i-1} \\ \implies \mathcal{W}_i &= 3 \cdot \mathcal{W}_{i-1} \\ \implies \mathcal{W}_i &= \mathcal{W}_i + \mathcal{R}_{i-1} \\ \implies \mathcal{R}_i &= 3 \cdot \mathcal{R}_{i-1} \\ \implies \mathcal{R}_i &= \mathcal{R}_i + 3^{\mathcal{X}_{i-1}} \\ \implies \mathcal{X}_i &= \mathcal{X}_{i-1} + 1 \end{aligned}$$

This case is quite tricky. For each already present combination $(R, (R, W)$ or $(R, W, G))$, we have to multiply the respective counter by 3 because for each possible value of the new wildcard character we can count that combination one time. Furthermore, the new character X could be considered as one of the previous cases, thus we do the same exact updates for each of the first two cases (G and W). Regarding the third case, i.e. when we consider the new X as the character R, we have to consider that, for each X in $S[1, i-1]$ we can choose its value in 3 different ways while counting the new one as a new R (this happens only for the R because this combination is independent of other characters, unlike the combinations (R, W) and (R, W, G)). This gives us a total of $3^{\mathcal{X}_{i-1}}$ new R combinations.
Finally, we increment $\mathcal{X}$ by one. Note that the order of the operations is important and it can not be changed.

2.2 Complexity analysis

The complexity analysis is the easy part of this algorithm. In fact, the only things we have to store are the four counter variables. Since for updating them we just need to look at the previous step, we don't need arrays or other structures; just 4 variables.

Therefore, the **additional** required space is $O(1)$, for a total space complexity of $\Theta(n)$ (we have to store the string of colors, i.e. the input).

Regarding the time complexity, we just need to scan the string, one character at a time. For each character we do a constant number of operations, each of which requires clearly constant time. So, the overall asymptotic time complexity is $\Theta(n)$, i.e. the algorithm works in linear time.