Competitive Programming and Contests
Hands-on 3

LEONARDO DA POZZO

A.A. 2022/2023

1 Introduction

The purpose of this hands-on is to solve two different problems with the use of dynamic programming. Both solutions were implemented in the Rust programming language.

2 Holiday Planning

2.1 Approach

We start by defining subproblems: let n be the number of cities and d the number of days, we consider the best holiday we can plan with a subset of $i < n$ cities and $j < d$ days. First of all, we build a $n \cdot d$ table containing each city's itinerary (a prefix sum over the city's attractions) and call it I. We then build a table M of size $(n+1) \cdot (d+1)$ with the first row and the first column filled with 0. We store the best out of the following three values in each cell $M[i, j]$:

- the best holiday of j days computed without considering city i. This result is stored in $M[i-1, j]$;
- the holiday in which we only visit city i for j days. This is already stored by $itineraries[i][j]$;
- the best holiday of j days total in which we consider the current city as well as the previous ones. This is computed as

  $$\max_{0 \leq k < j} matrix[i-1][j-k] + itineraries[i][k]$$

  Basically we compare all the best ways to spend $j-k$ days without city i (this was already computed in the previous iterations) extended with k days in city i and take the best combination.

Once the table is built, the solution is in M[n][d].

2.2 Complexity

The matrices are $\Theta(n \cdot d)$, but for each one of M's cell we perform $O(d)$ iterations, so that gives us a total time complexity of $O(n \cdot d^2)$ and a space complexity of $\Theta(n \cdot d)$.

3 Xmas Lights

3.1 Preliminary Problem

In order to count the number of reverse flags when no X's are involved, we can do the following: keep three variables

- r: number of R's encountered up to now. Incremented by one every time we find a R;
- rw: number of RW substrings encountered up to now. Every time we find a W, we increment rw by r since we can extend every previous R with the newfound W;
- rwg: number of full flags found. Every time we find a G we increment rwg by rw since we can extend every previous RW substring with the newfound G.

Once the full string has been scanned, the result is in rwg.

3.2 Approach

For the actual problem we have to deal with X's as well, so we keep one more variable, x, and we need to decide what we need to do when we encounter this character. We will use dynamic programming: our subproblems are the strings of length $n' < n$, where n is the total length. We scan the string left to right and treat the R, W and G letters as before, but every time we find a new X we update the four variables as follows (order is important):

1. rwg: we use as base the previous value of rwg multiplied by three, since any previous string can be extended in three different ways with this new X. To this, we need to sum all the flags that end in this X, which is given by the current value of rw;
2. rw: like before, we multiply the old value by three to account for all the new extensions and then consider all the RW strings which end in the current X, which is given by the current value of r;
3. r: again, we multiply the previous value by three, but then we also sum it the value of 3^x. This is because if we fix the current X to be R, the total number of strings which end in R depend on the number of previously encountered X's, since those can vary among three values each;

4. x: incremented by one.

At the end, the result is stored in rwg.

3.3 Complexity

Time complexity is $\Theta(n)$ since we scan the input string once. As far as space complexity goes, we only store four variables in addition to the input data, so we only take up $\Theta(n)$ space.