Hands-on 3 - Competitive Programming

Chiara Molinari

August 2023

1 Holiday planning

The solution uses dynamic programming. The involved data structure is a table 'plan' (vector of vectors, size $(n+1)(D+1)$) of integers.
The value $plan[j][i]$ is the max possible value of plans that include j days of holiday and visiting the first i cities. We have $0 \leq j \leq D$ and $0 \leq i \leq n$.
The first row and the first column are set to 0.
We iterate through the rest of the table and apply the following rule.

$$plan[j][i] = \max_{0 \leq r \leq j} plan[j-r][i-1] + sum(cities[i-1][0..r])$$

where $cities[a][b]$ is the value of the activity for day b in city a.
r stands for 'removed days' and this number can be interpreted as the number of days we can spend in the city that we are adding.
After computing the whole table, we are interested in the value of 'plan' for $j = D$ and $i = n$.

Space complexity: the dominant factor is the table size, which is $O(nD)$ space.

Time complexity: there are $O(nD)$ iterations, all consisting in looking for a max among a vector of size $O(D)$, giving a time complexity of $O(nD^2)$.

2 Xmas lights

This problem is also solved using dynamic programming. The involved data structure is a vector ('total', size $n+1$) of tuples of size 3.
Suppose $0 \leq i \leq n$ and $total[i] = (a, b, c)$. This means that the original vector, truncated at i, contains respectively 'a' R, 'b' couples RW and 'c' tuples RWG. Note that if Xs appear, we are considering the sum over the possible "light-scenarios".
The integer 'count' counts the number of scenarios involved. Indeed, it is firstly set to 1 and every time we encounter an X, we have 3 times the scenarios we had before.
We set total[0] to (0,0,0) and iterate through the Xmas lights vector as follows.
If we encounter an R, $total[i].0 = total[i-1].0 + count$, meaning that in all scenarios +1 R appears.
If we encounter a W, $total[i].1 = total[i-1].1 + total[i-1].0$, meaning that the sum over possible scenarios of RW is increased by $total[i-1].0$.
If we encounter a G, $total[i].2 = total[i-1].2 + total[i-1].1$, meaning that the sum over possible scenarios of RWG is increased by $total[i-1].1$.
Every non-mentioned entry of total[i] is set equal to the corresponding in total[i-1].
If we encounter an X, total[i] is the sum of the previously mentioned possible new values.
In other terms, we have that

$$total[i].0 = 3 * total[i-1].0 + count$$

$$total[i].1 = 3 * total[i-1].1 + total[i-1].0$$

$$total[i].2 = 3 * total[i-1].2 + total[i-1].1$$

Space complexity: the dominant term is the size of the vector 'total', giving a $O(n)$ space complexity.

Time complexity: the algorithm scans all the Xmas-light vector once and at each step performes $O(1)$ operations that require $O(1)$ time, giving a total time complexity of $O(n)$.