Solution of two dynamic programming problems: *holiday planner* and *xmas lights*

Giuseppe Muntoni

Contents

1 Introduction

In this report are discussed the solutions of two dynamic programming problems: *holiday planner* and *xmas lights*.

2 Holiday planner

2.1 Solution

The solution is the same as the Knapsack problem 0/1. Each city is considered with weight w from 1 to the number of days d. The value of a city, when considered for the first w days, is the prefix sum of the array of attractions for this city, until w.

For memoization, an array is sufficient. Just update it from right to left. The i_{th} element of the array contains the suboptimal number of attractions for i days and a subset of the cities, but, since it is not possible to consider a city multiple times with different weights, also a bool vector that indicates which cities are used, is contained.

2.2 Complexity

The time complexity is $O(n \cdot d^2)$ where n is the number of cities and d is the number of days.

This is so because there are $n \cdot d$ *objects* to consider, i.e., each city for each number of days that it can be visited. Each of these objects is considered for each capacity from 0 to d.

The space complexity is $O(n \cdot d)$ because the array used for memoization contains $d+1$ elements, and each element is an integer plus a vector of booleans of length n.

3 Xmas lights

3.1 Solution

Consider h as the list of houses in the street.

The subproblems of the original problem are suffixes of the original list. It is considered the last house, then the last two, and so on.

The recurrence of this problem considers two functions: g and w.

$$g(i) = \begin{cases} g(i+1) + 3^{x_{i+1}}, & \text{for } h(i) = G \\ 3 \cdot g(i+1) + 3^{x_{i+1}}, & \text{for } h(i) = X \\ g(i+1), & \text{otherwise} \end{cases}$$

where x_{i+1} are the number of x in the suffix from $i+1$.

This function counts the number of houses green for each possible combination of X in the suffix from i.

If the i_{th} house is green, then one multiplied by the number of all possible combinations of the suffix from $i+1$ is added to the total number of houses green in each combination of the suffix from $i+1$. If instead, the i_{th} house is not colored yet, i.e., X, it can be green, but also red or white, so the result of the suffix from $i+1$, in this case, is multiplied by three.

$$w(i) = \begin{cases} w(i+1) + g(i+1), & \text{for } h(i) = W \\ 3 \cdot w(i+1) + g(i+1), & \text{for } h(i) = X \\ w(i+1), & \text{otherwise} \end{cases}$$

This function counts the number of houses green associated with each possible house white in every combination of house white in the suffix from i. Consider the X at position j with $j > i$. Then, the function considers the case in which both i and j are white. This happens a number of times equal to 3 to the number of X between i and j, and this is multiplied by the number of houses green associated with the house white at position j.

Finally, the combine phase is done in this way:

$$patrioticSelections(i) = \begin{cases} patrioticSelections(i+1) + w(i+1), & \text{for } h(i) = R \\ 3 \cdot patrioticSelections(i+1) + w(i+1), & \text{for } h(i) = X \end{cases}$$

This considers the number of patriotic selections for the last i houses.

The final solution is simply $patrioticSelections(n)$ where n is the total number of houses.

The algorithm is an implementation of these recurrences but without recursive calls. Dynamic programming is used instead. Two arrays are maintained for calculating w and g. Then, an accumulator is used to calculate the total number of patriotic selections for every possible street. The arrays are filled from right to left.

3.2 Complexity

The time complexity is $\Theta(n)$ because each array is filled with a for loop from right to left. Inside the body of the loop are done simple operations that take constant time. The power of three in principle, if calculated each time, takes $O(\log(n))$ time, but this is calculated incrementally with an accumulator multiplied by three each time an X is encountered, so is constant for each iteration.

The space complexity is $\Theta(n)$ because two arrays of length n are maintained for memoization.