Eric Freire Antunes

1 Overview

In this solution, the problem was modeled as a set of subproblems and the following recurrence equation was devised:

$$Q_{n,b_{acc}} = \min_{m \epsilon M} \left(\max\left(0, P_{n,m} - b_{acc}\right) + Q_{n+1,B_{n,m}}\right) \tag{1}$$

$$Q_{n,b_{acc}} = 0 \forall n >= N \tag{2}$$

And the desired solution is:

$$Q_{0,0} \tag{3}$$

Where

- $Q_{n,b_{acc}}$ is the amount of start ammunition needed to clear the game starting in level n with b_{acc} bullets accumulated from the level n-1.
- $P_{n,m}$ is the power of enemy m from level n
- $B_{n,m}$ is the amount of bullets of enemy m from level n

This modeling was possible only because of the property that the bullets acquired during a given level can only be used in the very next one.

The key idea of this solution is that, if the amount of bullets accumulated from the last level is known, it is easy to determine how many start bullets is needed to defeat a given enemy. The idea is that the player should use as many accumulated bullets as possible, saving the start bullets. Hence, the number of start bullets needed to defeat enemy m from level n is:

$$P_{n,m} - b_{acc} \tag{4}$$

Notice that if the Power of the enemy is lower than the accumulated bullets, the player will have bullets to spare, represented by a negative number, meaning that no start bullets is needed. Once we do not want this negative number, the function max was used.

$$\max\left(0, P_{n,m}\right) \tag{5}$$

Behind the scenes, this problem consists of choosing which enemy to defeat at each level. Choosing a given enemy impacts the very next level directly because of the number of bullets acquired. Hence, if the player chooses to defeat the enemy m from level n, the amount of start bullets needed would be the number of start bullets needed to defeat enemy m ($\max\left(0, P_{n,m} - b_{acc}\right)$) plus the number of starting bullets to clear the game starting from level n+1 with $B_{n,m}$ bullets accumulated to next level ($Q_{n+1,B_{n,m}}$).

Hence, one way to do this is to iterate though all enemies of a level and pick the one that needs the minimum start bullets.

Notice that in a given level, there may be more than one enemy with the same amount of bullets (but different power). Once $Q_{n,b_{acc}}$ is determined only by the level and amount of bullets accumulated, there are many cases in which we would have to calculate the same problem. In that case, dynamic programming plays its role. Using a memoization matrix, the solution to subproblems are stored to be used later.

2 The code

Link to repository: `https://github.com/efreirea/desafioms.git`

3 Time Complexity

Concepts of Dynamic Programming were used to solve this problem. The memoization matrix used has dimensions $N \times B$. To solve each of the subproblems, iterating through all enemies M in a given level was necessary. In the worst case scenario, this solution fills the entire matrix in order to reach the answer. Hence, the time complexity of this solution is $O(N \times B \times M)$

3.1 Implementation ways of saving processing time

There are some logical facts about the nature of this solution that can be used to lower the amount of processing needed. The bottleneck of this solution is the number of enemies. M is the largest possible value between N,B and M. Then, finding cases in which it would not be necessary to iterate through all enemies can save processing time. Three such cases were found:

- The lowest possible number of starting bullets needed is zero. Hence, during the iteration though all enemies, if one enemy that needs 0 extra bullets to be defeated is found, then it is not necessary to find another.
- Suppose we are calculating $Q_{n,b_{acc}}$. If $Q_{n,b_{acc}-1}$ is known, a solution for level n is already known. This means there is a upper bound to the value of starting bullets needed. If a solution for $Q_{n,b_{acc}}$ is found such that $Q_{n,b_{acc}} > Q_{n,b_{acc}-1}$, this solution would not be optimal and should not be considered. Hence, enemies that would make $Q_{n,b_{acc}} > Q_{n,b_{acc}-1}$ should not be considered.
- While iterating though enemies to compute the minimum, if $\max(0, P_{n,m} - b_{acc})$ is higher than any previous enemy m_i already computed previously, it is logical that $\max(0, P_{n,m} - b_{acc}) + Q_{n+1,B_{n,m}}$ will be higher than m_i because $Q_{n+1,B_{n,m}} >= 0$. Hence, such enemy should not be considered.

Notice that the given facts above do not change the time complexity in the worst case scenario. Although they speed up computing time in specific cases, there may be cases in which neither of them applies.

4 How can you improve your existing solution? If that is possible, what would your new solution's O() complexity be?

Honestly, I do not know how to improve this solution. I have done my best to devise this one. If I could think of a better one, I would have done it.

But, in terms of implementation, I could have used less abstract structures or even used another programming language that worked in lower levels. This could reduce the overhead of calling methods and performing abstract operations. Another thing that could have been done is to implement the dynamic programming iteratively. But neither of these alternatives would change the time complexity.

5 What is the complexity class (P, NP, NP-complete, etc) of this problem ?

This problem in NP. Given a solution, it is possible to check its validity in polynomial time.

6 Provide us a diagram containing the state machine automata of your algorithm

I do not know how to build an automaton for algorithms.