1st March

[LeetCode] 312. Burst Balloons

Find Other LeetCode Python Answers: https://github.com/wqr619/leetcode [https://github.com/wqr619/leetcode]

Given `n` balloons, indexed from `0` to `n-1`. Each balloon is painted with a number on it represented by array `nums`. You are asked to burst all the balloons. If the you burst balloon `i` you will get `nums[left] * nums[i] * nums[right]` coins. Here `left` and `right` are adjacent indices of `i`. After the burst, the `left` and `right` then becomes adjacent.

Find the maximum coins you can collect by bursting the balloons wisely.

Example:

Given `[3, 1, 5, 8]`

Return `167`

This is an optimization problem aiming to get maximum coins. The key point here is that every time you burst a balloon, the number of coin you gain depends on your previous steps of bursting coins. If you did not burst your adjacent balloons of ballon i, the number of coins you gain

```
nums[i] * nums[i-1] * nums[i+1]
```

This is the kind of problem we use dynamic programming. WHY? it's very challenging to figure out what's the pattern of optimal burst order. In fact, there's no clear rule that makes sense. Shall we burst the balloon with maximum coins? Or shall we burst the one with least. This is the time we introduce Dynamic Programming, as we want to solve the big problem from small subproblem. It is clear that the amount of coins you gain relies on your previous steps. This is a clear signal of using DP.

The hard part is to define the subproblem. Think out what is clear in this problem? Let's scale this problem down. What is the fact you know for sure? Say if the array has only 1 balloon. The maximum coin would be the coin inside this ballon. This is the starting point! So let's move on to array with 2 balloons. Here, we have 2 cases, which of the balloon is the last one. The last one times the coins in boundary is the gain we get in the end. That is to say, last balloon is the key. Since we don't know the pattern of optimal. We just blindly iterate each balloon and check what's total gain if it's the last ballon.

Let's use dp[i][j] to denote maximum gain from balloon range i to j. We try out each balloon as last burst in this range. Then the subproblem relation would be:

```
foreach k in i to j:
 dp[j][i] = max(array[j-1]*array[k]*array[i+1] + dp[j][k-1] + dp[k+1][i], dp[j][i]);
```

So the final Java code is:

```
public class Solution {
    public int maxCoins(int[] nums) {
        // Extend list with head and tail (both are 1), index starts at 1
        int array[] = new int[nums.length + 2];
        array[0] = 1;
        array[array.length-1] = 1;
        for (int i = 0; i < nums.length; i++) {
            array[i+1] = nums[i];
        }

        // Initialize DP arrays, 1 index based
        int dp[][] = new int[array.length][array.length]; //dp[i][j] stands for max coins at i step, burs
        for (int i =0; i < array.length; i++) {
            for (int j = 0; j < array.length; j++) {
                dp[i][j] = 0;
            }
        }

        for (int i=1; i< array.length-1; i++) {
            for (int j=i; j >=1; j--) {
                // k as last
                for (int k=j; k <= i; k++) {
                    dp[j][i] = Math.max(array[j-1]*array[k]*array[i+1] + dp[j][k-1] + dp[k+1][i], dp[j][i
                }
            }
        }

        return dp[1][array.length-2];
    }
}
```

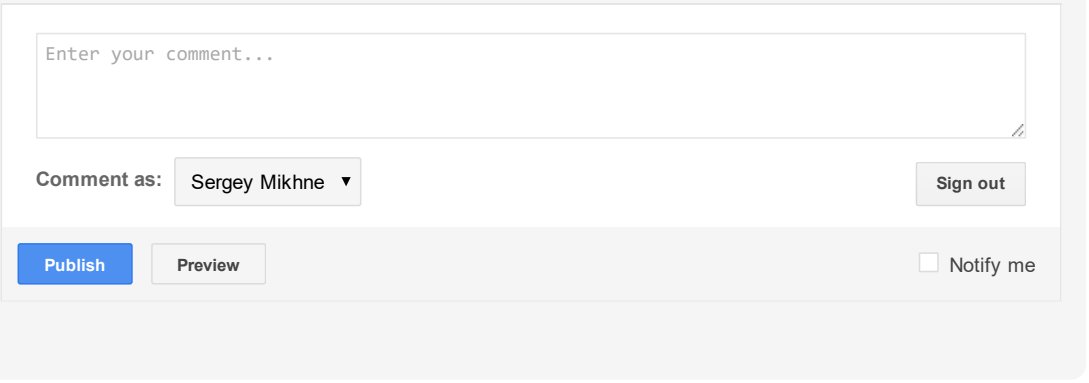
Posted 1st March by Qirong Wang

1 View comments

ZengRui Wang March 12, 2016 at 11:08 PM

Thank you, have been trying to understand what this equation means and searching online for quite a long time, but your explanation is really detailed and made my life easier!

Reply


Enter your comment...
Comment as:
Sergey Mikhne
Sign out
Publish
Preview
Notify me