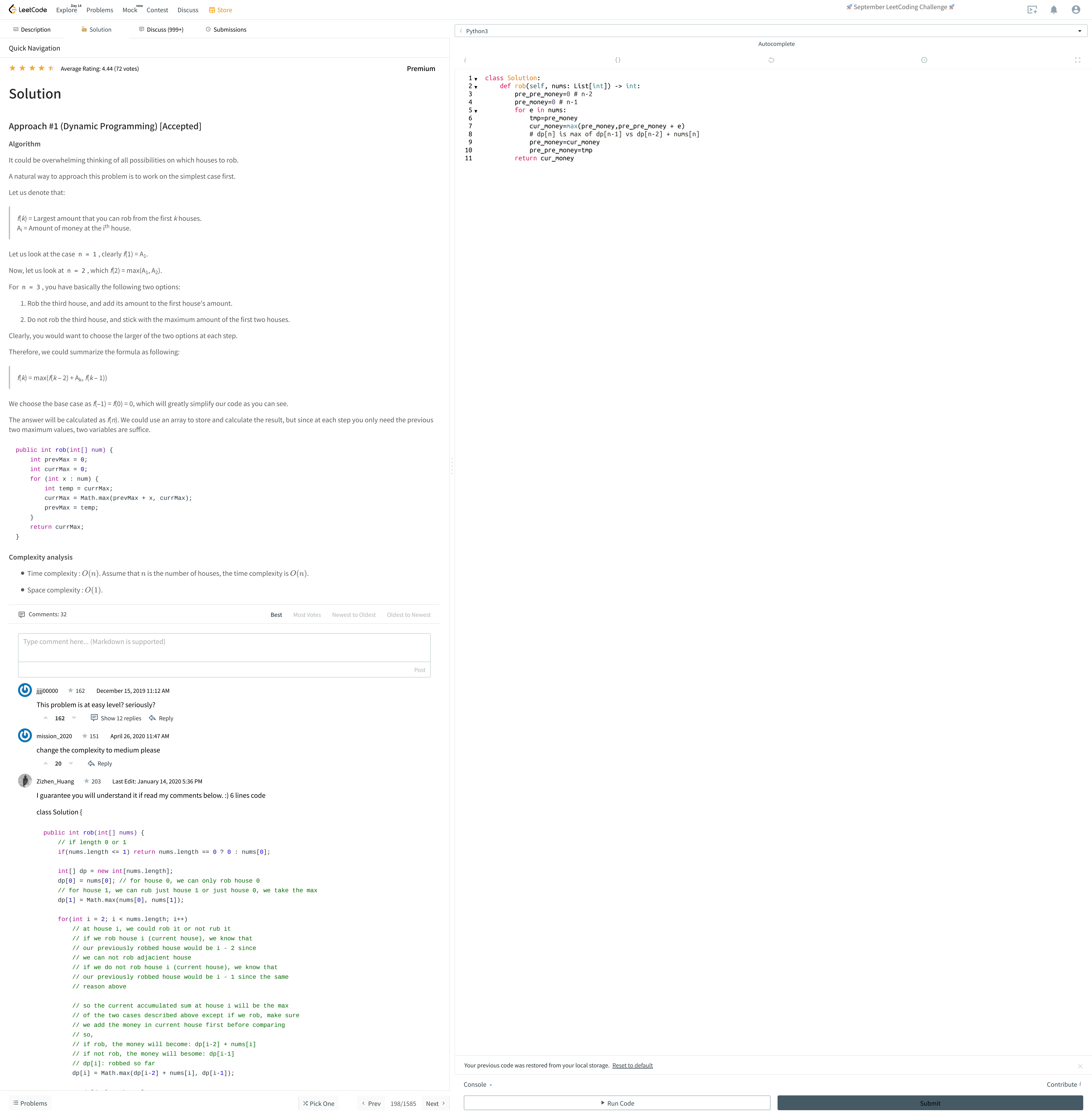

Solution

Approach #1 (Dynamic Programming) [Accepted]

Algorithm

It could be overwhelming thinking of all possibilities on which houses to rob.

A natural way to approach this problem is to work on the simplest case first.

Let us denote that:

> $f(k)$ = Largest amount that you can rob from the first k houses.
> A_i = Amount of money at the i^{th} house.

Let us look at the case `n = 1`, clearly $f(1) = A_1$.

Now, let us look at `n = 2`, which $f(2) = \max(A_1, A_2)$.

For `n = 3`, you have basically the following two options:

1. Rob the third house, and add its amount to the first house's amount.
2. Do not rob the third house, and stick with the maximum amount of the first two houses.

Clearly, you would want to choose the larger of the two options at each step.

Therefore, we could summarize the formula as following:

> $f(k) = \max(f(k-2) + A_k, f(k-1))$

We choose the base case as $f(-1) = f(0) = 0$, which will greatly simplify our code as you can see.

The answer will be calculated as $f(n)$. We could use an array to store and calculate the result, but since at each step you only need the previous two maximum values, two variables are suffice.

```
public int rob(int[] num) {
    int prevMax = 0;
    int currMax = 0;
    for (int x : num) {
        int temp = currMax;
        currMax = Math.max(prevMax + x, currMax);
        prevMax = temp;
    }
    return currMax;
}
```

Complexity analysis

- Time complexity : $O(n)$. Assume that n is the number of houses, the time complexity is $O(n)$.
- Space complexity : $O(1)$.

```
class Solution:
    def rob(self, nums: List[int]) -> int:
        pre_pre_money=0 # n-2
        pre_money=0 # n-1
        for e in nums:
            tmp=pre_money
            cur_money=max(pre_money,pre_pre_money + e)
            # dp[n] is max of dp[n-1] vs dp[n-2] + nums[n]
            pre_money=cur_money
            pre_pre_money=tmp
        return cur_money
```

Comments: 32

jjjj00000 ★ 162 December 15, 2019 11:12 AM

This problem is at easy level? seriously?

162 Show 12 replies Reply

mission_2020 ★ 151 April 26, 2020 11:47 AM

change the complexity to medium please

20 Reply

Zizhen_Huang ★ 203 Last Edit: January 14, 2020 5:36 PM

I guarantee you will understand it if read my comments below. :) 6 lines code

class Solution {

```
  public int rob(int[] nums) {
      // if length 0 or 1
      if(nums.length <= 1) return nums.length == 0 ? 0 : nums[0];

      int[] dp = new int[nums.length];
      dp[0] = nums[0]; // for house 0, we can only rob house 0
      // for house 1, we can rub just house 1 or just house 0, we take the max
      dp[1] = Math.max(nums[0], nums[1]);

      for(int i = 2; i < nums.length; i++)
          // at house i, we could rob it or not rub it
          // if we rob house i (current house), we know that
          // our previously robbed house would be i - 2 since
          // we can not rob adjacient house
          // if we do not rob house i (current house), we know that
          // our previously robbed house would be i - 1 since the same
          // reason above

          // so the current accumulated sum at house i will be the max
          // of the two cases described above except if we rob, make sure
          // we add the money in current house first before comparing
          // so,
          // if rob, the money will become: dp[i-2] + nums[i]
          // if not rob, the money will besome: dp[i-1]
          // dp[i]: robbed so far
          dp[i] = Math.max(dp[i-2] + nums[i], dp[i-1]);
```

Your previous code was restored from your local storage. Reset to default