Description | Editorial | Solutions (408) | Submissions

808. Soup Servings

Medium 771 2.1K

Companies

There are two types of soup: **type A** and **type B**. Initially, we have `n` ml of each type of soup. There are four kinds of operations:

1. Serve `100` ml of **soup A** and `0` ml of **soup B**,
2. Serve `75` ml of **soup A** and `25` ml of **soup B**,
3. Serve `50` ml of **soup A** and `50` ml of **soup B**, and
4. Serve `25` ml of **soup A** and `75` ml of **soup B**.

When we serve some soup, we give it to someone, and we no longer have it. Each turn, we will choose from the four operations with an equal probability `0.25`. If the remaining volume of soup is not enough to complete the operation, we will serve as much as possible. We stop once we no longer have some quantity of both types of soup.

Note that we do not have an operation where all `100` ml's of **soup B** are used first.

Return *the probability that **soup A** will be empty first, plus half the probability that **A** and **B** become empty at the same time.* Answers within 10^{-5} of the actual answer will be accepted.

Example 1:

```
Input: n = 50
Output: 0.62500
Explanation: If we choose the first two operations, A will
become empty first.
```

Java | Auto

```java
class Solution {
    private HashMap<Pair<Integer, Integer>, Double> memo = new HashMap<>();

    public double soupServings(int N) {
        if (N > 4451) {
            return 1.0;
        }
        N = (N + 24) / 25;

        return dp(N, N);
    }

    private double dp(int a, int b) {
        if (a <= 0 && b <= 0) {
            return 0.5;
        }
        if (a <= 0) {
            return 1.0;
```

Testcase | Result

Accepted Runtime: 0 ms

Case 1 | Case 2

Input

n =
50

Output

0.62500

Console | Run | Submit

Description | Editorial | Solutions (4.7K) | Submissions

746. Min Cost Climbing Stairs

Hint

Easy 9.9K 1.5K

Companies

You are given an integer array `cost` where `cost[i]` is the cost of i^{th} step on a staircase. Once you pay the cost, you can either climb one or two steps.

You can either start from the step with index `0`, or the step with index `1`.

Return *the minimum cost to reach the top of the floor.*

Example 1:

```
Input: cost = [10,15,20]
Output: 15
Explanation: You will start at index 1.
- Pay 15 and climb two steps to reach the top.
The total cost is 15.
```

Example 2:

```
Input: cost = [1,100,1,1,1,100,1,1,100,1]
Output: 6
Explanation: You will start at index 0.
- Pay 1 and climb two steps to reach index 2.
- Pay 1 and climb two steps to reach index 4.
- Pay 1 and climb two steps to reach index 6.
- Pay 1 and climb one step to reach index 7.
- Pay 1 and climb two steps to reach index 9.
- Pay 1 and climb one step to reach the top.
The total cost is 6.
```

Java | Auto

```java
class Solution {
    public int minCostClimbingStairs(int[] cost) {
        int[] dp = new int[cost.length];
        return Math.min(helper(cost, 0, dp), helper(cost, 1, dp));
    }

    private int helper(int[] cost, int i, int[] dp) {
        if(i >= cost.length) return 0;
        if(dp[i] != 0) return dp[i];

        int oneStep = cost[i] + helper(cost, i + 1, dp);
        int twoStep = cost[i] + helper(cost, i + 2, dp);

        return dp[i] = Math.min(oneStep, twoStep);
    }
}
```

Testcase | Result

Accepted Runtime: 0 ms

Case 1 | Case 2

Input

cost =
[10,15,20]

Output

15

Console | Run | Submit