题目地址(837. 新 21 点)

https://leetcode-cn.com/problems/new-21-game

入选理由

1. dp + 滑动窗口的结合。让大家理解滑动窗口是如何和其他技巧联系的。 之前我也出了 二分 + dfs 这种联动性题目， 这种题目对于你学习知识很有帮助。

标签

- 二分
- 滑动窗口

难度

- 中等

题目描述

爱丽丝参与一个大致基于纸牌游戏 “21点” 规则的游戏，描述如下：

爱丽丝以 0 分开始，并在她的得分少于 K 分时抽取数字。 抽取时，她从 [1, W] 的范围中随机获得一个整数作为分数进行累计，其中 W 是整数。 每次抽取都

当爱丽丝获得不少于 K 分时，她就停止抽取数字。 爱丽丝的分数不超过 N 的概率是多少？

示例 1：

输入：N = 10, K = 1, W = 10
输出：1.00000
说明：爱丽丝得到一张卡，然后停止。
示例 2：

输入：N = 6, K = 1, W = 10
输出：0.60000
说明：爱丽丝得到一张卡，然后停止。
在 W = 10 的 6 种可能下，她的得分不超过 N = 6 分。
示例 3：

输入：N = 21, K = 17, W = 10

输出：0.73278

提示：

0 <= K <= N <= 10000
1 <= W <= 10000
如果答案与正确答案的误差不超过 10^-5，则该答案将被视为正确答案通过。
此问题的判断限制时间已经减少。

前置知识

- 动态规划[1]
- 滑动窗口[2]

思路

我们倒着往前思考，由于每次选择的数字范围都是 [1, W]，并且如果当前分数大于 K 就会停止，因此**最后一次**抽取的时候分数一定是小于 K 的，并且抽取的数字 + 当前的分数要大于等于 K。

而我们要求的就是**分数大于等于 K 的这些情况中不大于 N 的概率**，即满足 K<= x <=N 中的概率，其中 x 为分数。也就是求满足 K<= x <=N 的总个数再除以 W，因为一次抽取有 W 种可能，分别是 [1,W]，且概率均相等，都为 1 /W。

如果用 dp[i] 表示当前分数为 i 的情况下，爱丽丝的分数不超过 N 的概率。那么：

```
dp[i] = sum(dp[i + j] for j in range(1, W + 1)) / W
```

其实就是说 dp[i] = (dp[i+1] + dp[i+2] + ... + dp[i+W]) / W，这就是动态转移方程。

由于我们的转移方程的 dp[i] 依赖 dp[i + x]，其中 x 属于 [1,W]，也就是说我们需要从后往前遍历，这样才能保证结果的正确性。

另外，我们需要初始化 [K, min(K+W-1, N)] 范围内的 dp 为 1，这是我们的边界条件，有了它们的存在，才能推动算法**动态**运算下去，他们的作用就像是给一个在高山上的雪球一个推力，帮助雪球滚落。

基于此，我们可以写出下面的代码。

Python3 代码：

```
class Solution:
    def new21Game(self, N: int, K: int, W: int) -> float:
        dp = [0] * (K + W)
        for i in range(K, K + W):
            if i <= N:
                dp[i] = 1
```

```
        for i in range(K - 1, -1, -1):
            dp[i] = sum(dp[i + j] for j in range(1, W + 1)) / W
        return dp[0]
```

复杂度分析

- 时间复杂度：$O(KW)$
- 空间复杂度：$O(K + W)$

时间复杂度是 $O(KW)$，代入题目的限制条件 0 <= K <= N <= 10000，1 <= W <= 10000，得到总的计算次数大约是 10^8，大于 10^7，因此会超时。

> 不明白为什么是 10^7 ？ 力扣加加公众号看《来和大家聊聊我是如何刷题的（第三弹）》

通过观察发现，每次 sum 我们变化的其实仅仅是左右两侧的数，中间的是不会变的。这不就是**我们讲义中的滑动窗口使用场景**么？

也就是说，我们只需要预先计算窗口的总和，之后更新窗口的时候都减去窗口右侧过期的数，再加上窗口左侧刚进去的数就行了。这种算法的时间复杂度是 $O(W + K)$

代码

代码支持：Python3, JS, Java， CPP

```
class Solution:
    def new21Game(self, N: int, K: int, W: int) -> float:
        # 滑动窗口优化（固定窗口大小为 W 的滑动窗口）
        dp = [0] * (K + W)
        win_sum = 0
        for i in range(K, K + W):
            if i <= N:
                dp[i] = 1
            win_sum += dp[i]

        for i in range(K - 1, -1, -1):
            dp[i] = win_sum / W
            win_sum += dp[i] - dp[i + W]
        return dp[0]
```

JS Code:

```
var new21Game = function (n, k, maxPts) {
  const dp = new Array(k + maxPts + 2).fill(0);

  let windowSum = 0;
  for (let i = k; i < k + maxPts; i++) {
    if (i <= n) dp[i] = 1;
    windowSum += dp[i];
  }

  for (let i = k - 1; i >= 0; i--) {
    dp[i] = windowSum / maxPts;
    windowSum -= dp[i + maxPts];
    windowSum += dp[i];
  }

  return dp[0];
};
```

Java Code:

```
class Solution {
     public double new21Game(int N, int K, int W) {
        if (K == 0 || N >= K + W) return 1;
        double dp[] = new double[N + 1],  Wsum = 1, res = 0;
        dp[0] = 1;
        for (int i = 1; i <= N; ++i) {
            dp[i] = Wsum / W;
            if (i < K) Wsum += dp[i]; else res += dp[i];
            if (i - W >= 0) Wsum -= dp[i - W];
        }
        return res;
    }
}
```

CPP Code:

```
class Solution {
public:
    double new21Game(int N, int K, int W) {
        if (K == 0) return 1.0;
        vector<double> dp(K + W);
        for (int i = K; i <= N && i <= K + W - 1; i++)  dp[i] = 1.0;
        dp[K - 1] = 1.0 * min(N - K + 1, W) / W;
        for (int i = K - 2; i >= 0; i--) dp[i] = dp[i + 1] - (dp[i + W + 1] - dp[i + 1]) / W;
        return dp[0];
```

```
    }
};
```

复杂度分析

- 时间复杂度：$O(K + W)$
- 空间复杂度：$O(K + W)$

参考资料

[1] 动态规划: *https://github.com/azl397985856/leetcode/blob/master/thinkings/dynamic-programming.md*

[2] 滑动窗口: *https://github.com/azl397985856/leetcode/blob/master/thinkings/slide-window.md*

上一页 下一页