# #837. 新21点【动态规划】【反序求解】

## 题目描述：



## 我的动态规划，一个找不出错误的错误解法--错在dp[k] [N]=dp[k-1] [N-1]+..

1. dp[k] [2],其中dp[k] [0]表示上限为k时的所有可能，而dp[k] [1]表示上限为k时成功的所有可能；故概率为dp[k] [1]/dp[k] [0];
2. 初始化dp[1] [0];
3. 依次遍历k从1到k；通过前面的dp[i]来初始化dp[k].
4. 可以看一下下面的思路；错在dp[k]起始和两个维度有关：既与K有关，也与N有关。
5. <https://leetcode-cn.com/problems/new-21-game/solution/ying-gai-he-ti-jie-qu-jue-da-bu-fen-fang-7ys5/>

## 题解区的大佬帮我解惑

假设进行了一次抽取，抽取出m，那么问题就转化成了N-m, K-m，对这两个维度进行 状态转移

dp[N] [K]=dp[N-1] [K-1]+dp[N-2] [K-2]......+dp[N-m] [K-m]的和；

class Solution {  
public:  
 unordered\_map<int, double> info;//建立一个哈希的映射  
 const int M = 10001;//由于N、K的取值小于10000；故后续建立哈希映射时不会整数不会越界  
  
 double new21Game(int N, int K, int W) {  
 if(info.find(M\*N + K) != info.end()) {  
 return info[M\*N + K];  
 }//查找是否存在  
   
 double res = 0;  
 if(K <= 0) {  
 return 1.0;  
 }  
   
 if(K == 1) {  
 // 此时只能抽一次，那么 就看W当中有多少个不超过N的值了  
 if(W <= N) {  
 return 1.0;  
 } else {  
 return 1.0 \* N / (W \* 1.0);  
 }  
 } else {  
 int upper = std::min(N, W);  
   
 for(int i = 1; i <= upper; i++) {  
 res += (1.0/W) \* new21Game(N-i, K-i, W);  
 }  
 }  
 info[M\*N + K] = res;//建立一个哈希的映射  
   
 return res;  
 }  
};  
  
作者：haithink  
链接：https://leetcode-cn.com/problems/new-21-game/solution/ying-gai-he-ti-jie-qu-jue-da-bu-fen-fang-7ys5/  
来源：力扣（LeetCode）  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

从两个维度上来解题的话，很容易超时！！！

**上面的代码中第一个参数的值 始终比第二个参数大 N-K，记为 delta；**

**通过参数之间的关系，将二维的参数降低为一维的参数**  
那么 只需要对第二个参数进行归纳，或者说状态转移即可， 设第二个参数为x

那么任何时候，第一个参数（现在隐藏起来了）的值为 x + delta

对于x比较普遍的情况，也就是x比W大的时候，状态转移方程为：  
f(x) = 1/w \* (f(x-1)+f(x-2) + ... + f(x-W))  
通过差分技巧，官方题解也有用，数学课程里面证明一些数列求和公式的时候也用这个技巧，可以化简为

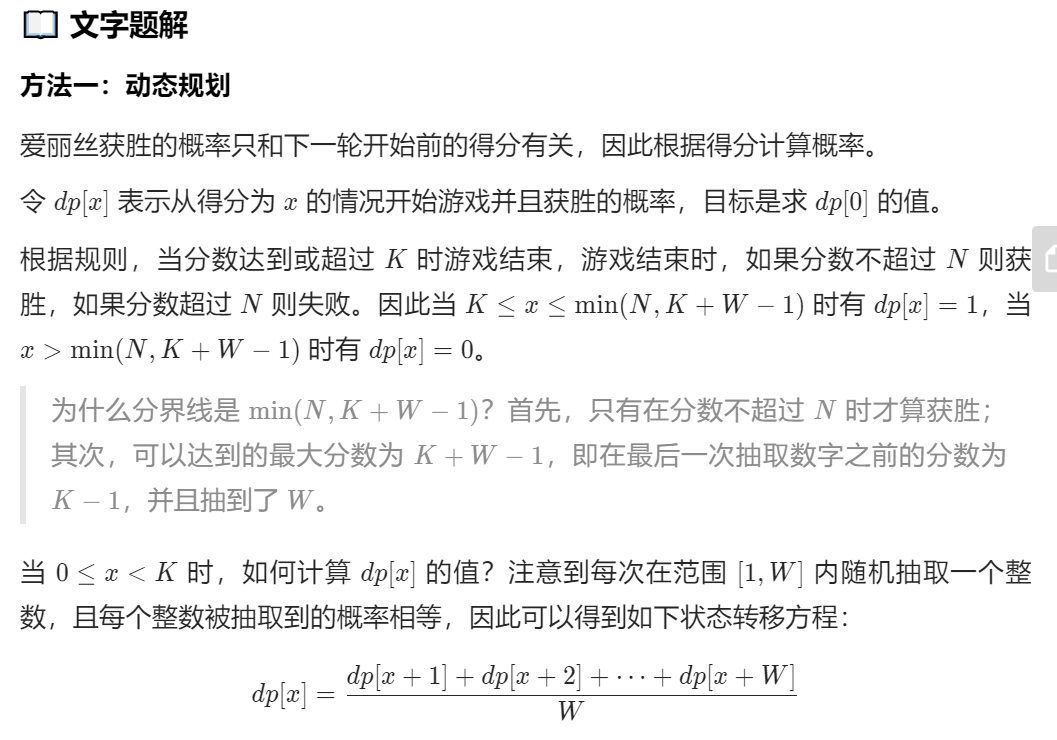
f(x) = f(x-1) + 1/w \* f(x-1) - 1/w \* f(x-1-w)

class Solution {  
public:  
 double new21Game(int N, int K, int W) {  
 double res = 0;  
 if(K <= 0) {  
 return 1.0;  
 }  
   
 std::vector<double> info(K+1);  
   
 info[0] = 1.0;  
   
 int delta = N-K;//参数1和参数2之间的差值  
 int bound = std::max(0, W - delta);  
   
 if(W <= delta + 1) {  
 info[1] = 1.0;  
 } else {  
 info[1] = 1.0 \* (delta+1) / (W \* 1.0);  
 }  
   
 for(int i = 2; i <= std::min(K,bound); i++) {  
 info[i] = (1.0 / (W \* 1.0) + 1.0) \* info[i-1];  
 }  
 for(int i = bound+1; i <= std::min(K,W); i++) {  
 info[i] = info[i-1] + 1.0 / (W \* 1.0) \* info[i-1] - 1.0 / (W \* 1.0);  
 }  
 // f(x) = f(x-1) + 1/w \* f(x-1) - 1/w \* f(x-1-w)  
 for(int i = W+1; i <= K; i++) {  
 info[i] = info[i-1] + 1.0 / (W \* 1.0) \* info[i-1] - 1.0 / (W \* 1.0) \* info[i-1-W];  
 }  
  
 return info[K];  
 }  
};  
  
作者：haithink  
链接：https://leetcode-cn.com/problems/new-21-game/solution/ying-gai-he-ti-jie-qu-jue-da-bu-fen-fang-7ys5/  
来源：力扣（LeetCode）  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

## 我的看不出问题的解法---已经找到问题

class Solution {  
public:  
 double new21Game(int N, int K, int W) {  
 //动态规划  
 vector<vector<int>>dp(K+1,vector<int>(2,0));  
 //初始化  
 dp[1][0]=W;  
 if(N<=W)  
 dp[1][1]=N;  
 else dp[1][1]=W;  
  
 //遍历  
 for(int i=2;i<=K;i++){  
 for(int j=1;j<i;j++){  
 dp[i][0]+=dp[j][0];  
 dp[i][1]+=dp[j][1];  
 }  
 if(i<=W){  
 dp[i][0]+=(W-i+1);  
 if(N<=W)  
 dp[i][1]+=(N-i+1);  
 else dp[i][1]+=(W-i+1);   
 }  
 }  
  
   
  
 double res=(double)dp[K][1]/(double)dp[K][0];  
  
 return res;  
 }  
};

## 官方解法【反序求解】



1. 这个问题虽然由点概率论的味道，但也别被吓到。
2. 反序求解：有点像概率论中的条件分布，由结果反求条件。
3. dp[x]表示的是从得分x的情况下开始游戏并且获胜的概率。
4. 状态转移方程：dp[x]=（dp[x+1]+dp[x+2]......+dp[x+w]）/W；
5. 这个问题在初始化时要注意空间的分配，以访出现越界的情况。

## 官方代码实现：

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; i++) {  
 dp[i] = 1.0;  
 }  
 for (int i = K - 1; i >= 0; i--) {  
 for (int j = 1; j <= W; j++) {  
 dp[i] += dp[i + j] / W;  
 }  
 }  
 return dp[0];  
 }  
};  
  
作者：LeetCode-Solution  
链接：https://leetcode-cn.com/problems/new-21-game/solution/xin-21dian-by-leetcode-solution/  
来源：力扣（LeetCode）  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。