1. 题目
2. 题干

把n个骰子扔在地上，所有骰子朝上一面的点数之和为s。输入n，打印出s的所有可能的值出现的概率。

你需要用一个浮点数数组返回答案，其中第 i 个元素代表这 n 个骰子所能掷出的点数集合中第 i 小的那个的概率。

（二） 示例

示例 1:

输入: 1

输出: [0.16667,0.16667,0.16667,0.16667,0.16667,0.16667]

示例 2:

输入: 2

输出: [0.02778,0.05556,0.08333,0.11111,0.13889,0.16667,0.13889,0.11111,0.08333,0.05556,0.02778]

1. 题解
2. 思路

竟然又是DP的题目，只要题目中有递增，递减关系的都要想一想DP能不能做。所有骰子的点数和就是一个个骰子叠加上去的，因此下一次骰子的点数和等于前一次骰子的点数和 + 1到6之间的数即可，这就形成了DP的状态转移方程。因此，各个阶段之间的转换关系已经找出来，现在来分析最后一个阶段是什么？题目要求的是当投掷完n枚骰子后，各个点数出现的次数。因此阶段的状态表示就是第n枚骰子，点数j出现的次数，因此设定一个二维DP数组来表示，思路形成了，具体的代码就不需要讲解了。

可以进一步优化空间，仅使用一维数组就可以实现，但是如果按照升序来叠加，就一定会叠加覆盖之前的数，如果被覆盖的数还要被其他的数所使用，就会无法实现DP。因此，我们使用倒序来叠加，这样即使要覆盖之前的数，他也已经被其他的数使用完毕，不用担心了。

1. 代码

未优化：

class Solution {

    public double[] dicesProbability(int n) {

        // s是n个骰子的点数之和，n个骰子共有？种点数和种类

        // 骰子的结果是对称的，

        // dp，每次骰子的点数都是增加的，注意，dp的题目的关键字是递增关系或者递减关系

        // 下一次某个点数等于上6个点数之和

        int[][] dp = new int[n+1][70];

        // 第一次

        for(int i = 1; i <= 6; ++i){

            dp[1][i] = 1;

        }

        // 从第二个骰子开始

        for(int i = 2; i <= n; ++i){

            // 下一个骰子的可能点数和

            for(int j = i; j <= i\*6; ++j){

                // 前一个骰子 再加上1- 6 等于下一个骰子可能的骰子总和

                for(int z = 1; z <= 6; ++z){

                    if((j-z) < (i-1)){

                        break;

                    }

                    dp[i][j] += dp[i-1][j-z];

                }

            }

        }

        double[] out = new double[5\*n+1];

        int i = 0;

        for(int times : dp[n]){

            if(times == 0) continue;

            out[i++] = (double)(times / Math.pow(6, n));

        }

        return out;

    }

}

优化：

class Solution {

public double[] twoSum(int n) {

int dp[]=new int[70];

for (int i=1;i<=6 ;i++ ) {

dp[i]=1;

}

for (int i=2;i<=n ;i++ ) {

for (int j=6\*i;j>=i ;j-- ) {

dp[j]=0;

for (int cur=1;cur<=6 ;cur++ ) {

if (j-cur<i-1) {

break;

}

dp[j]+=dp[j-cur];

}

}

}

double all=Math.pow(6,n);

double[] res=new double[5\*n+1];

for (int i=n;i<=6\*n ;i++ ) {

res[i-n]=(dp[i]\*1.0/all);

}

return res;

}

}