题目大意

2.7.4 Millionaire

Millionaire（2008 APAC local onsites C）

你被邀请到某个电视节目中去玩下面这个游戏。一开始你有 x 元钱，接着进 M 轮赌博。每一轮，可以将所持的任意一部分钱作为赌注。赌注不光可以是整数，也可以是小数。一分钱不押或全押都没有关系。每一轮都有 P 的概率可以赢，赢了赌注就会翻倍，输了赌注就没了。如果你最后持有 1000000 元以上的钱的话，就可以把这些钱带回家。请计算当你采取最优策略时，获得 1000000 元以上的钱并带回家的概率。

⚠限制条件

- $0 \leqslant P \leqslant 1.0$
- $1 \leqslant X \leqslant 1000000$

Small

- $1 \leqslant M \leqslant 5$

Large

- $1 \leqslant M \leqslant 15$

样例 1

输入

```
M = 1, P = 0.5, X = 500000
```

输出

```
0.500000(一开始便全押)
```


样例 2

输入

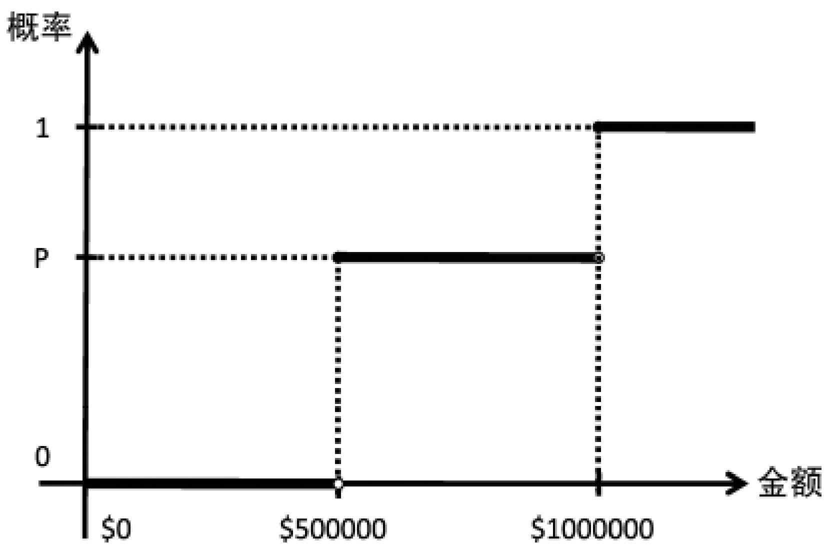
```
M = 3, P = 0.75, X = 600000
```

输出

```
0.843750
```

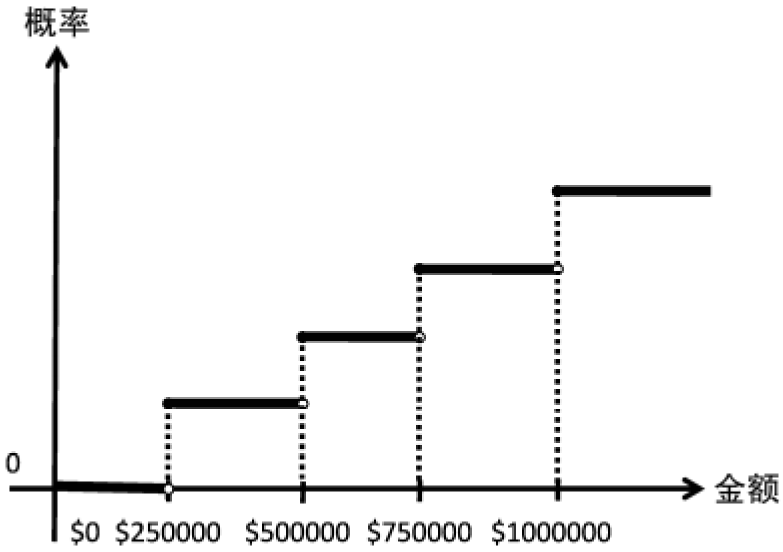

解题思路

离散化思想: 首先对金钱分阶段进行离散化，如果只有一轮的情况下，容易知道有三个阶段:



最后一轮带钱回家的概率

当有两轮的时候，也可以知道，有五个阶段:



最后两轮带钱回家的概率

当轮数为M时，有2^M+1个状态

- 证明:当有M轮时，对M轮的输赢，共有2^M中情况。 我们对每一种情况分配一个金钱阶段，在这种情况下，处于这个金钱阶段里的金钱最终能够到达1000000。比如对全输的情况，那么对应的金钱阶段就是大于等于1000000元。还有一种情况就是无论如何都到达不了1000000元，因此共有2^M+1种状态。又因为每次赌博是翻倍的，因此每个阶段的覆盖的金钱区间长度是一样的。

动态规划过程:

- 定义$dp[i][j]$: 第i轮赌博时，拥有的钱在阶段j能走人的最大概率。
- 目标:$dp[1][stage(X)]$:第1轮赌博时，拥有的钱为X, 其阶段为$stage(X)$.此时获胜的最大概率。
- 状态转移: 对状态$dp[i][j]$来说，(1)赌徒可以拿出横跨k个阶段的钱来赌博。(2)可能赌输也可能赌赢。
  - 对(1) 有$1<=k<=min(2^M+1-j,j)$, 这里不考虑$k>=2^M+1-j$,是因为没必要花更多的钱去达到更少的钱能够达到的阶段。
  - 对(2): 1、赌赢: 概率为P, 会转移到$dp[i+1][j+k]$状态。2、赌输: 概率为$1-P$, 会转移到$dp[i+1][j-k]$阶段。
  - 由于赌赢赌输互斥，因此由全概率公式，有转态转移方程:

$$dp[i][j]=max\{P*dp[i+1][j+k]+(1-P)*dp[i+1][j-k]\quad|1<=k<=min(2^M+1-j,j)\}$$

  前一项$dp[i+1][j+k]$是因为拿覆盖k阶段出来赌，赢了翻倍即多出k阶段钱，即到了$j+k$阶段。
- 更新策略： i从大到小更新。(实现时采用滚动数组交替更新)
- 初始化: 考虑最后一轮：
  - 当钱$0<=money<500000$时，概率为0, 即阶段$1<=k<=2^{M-1}$.
  - 当$500000<=money<1000000$时, 概率为P，即阶段$2^{M-1}<k<=2^M$.
  - 概率为P, 当$money>=1000000$时，概率为1，即阶段2^M+1.
- 复杂度: $O(M2^M)$

代码

```
#include<iostream>
#include<stdio.h>
using namespace std;

const double thread = 1000000;
const int MAXM = 16;
double dp[2][1 << MAXM];
double X;
int M;
double P;

int main()
{
    while(cin >> M >> P >> X)
    {
```

```
        int m = (1 << M) + 1; // 1 - 2^M + 1共 2^M+1个状态
        double per_range = thread / (m-1);
        for(int i=1; i<=m/2; i++)
            dp[M%2][i] = 0;
        for(int i=m/2; i<=m-1; i++)
            dp[M%2][i] = 0.5;
        dp[M%2][m] = 1.0;

        for(int i=M-1; i>=1; i--)
        {
            for(int j=1; j<=m; j++)
            {
                for(int k=1; k<=min(j, m-j); k++)
                {
                    dp[i%2][j]= max(dp[i%2][j], P*dp[1-i%2][j+k]+(1-P)*dp[1-i%2][j-k]);
                }
            }
        }
        printf("%.6f\n", dp[1%2][(long long)X*m/1000000+1]);
    }
    return 0;
}
```