7月17日 概率与期望

前言：因为晚上还有较高难度的讲解，所以我们选择的内容较为简单。

概率与期望回顾

概念呈现（复习）

1. 基于集合的事件表述与运算
2. 概率：（感性理解）某种事件发生的可能性的量化表述（例如：某事件发生的概率为p，意味在x次独立重复事件中，该事件发生xp次）。
3. 伯努利实验：在同样条件下重复的独立的进行的一种随机试验，其特点是，该随机试验只有两种可能结果，发生或不发生。把伯努利实验做n次称为n重伯努利实验。（ $p_{发生}+p_{不发生}=1$ ）
4. n重伯努利实验中 $期望=\frac{1}{概率}$
5. 二项分布：
6. 几何分布：在实验过程中，前面k-1次全部失败，在第次k取得首次成功的概率
7. 等其他古典概率、几何概率模型
8. 决策树
9. 条件概率：贝叶斯公式$P(A|B)=\frac{P(B|A)\times P(A)}{P(B)}$ 即已知B发生了，想要求出A发生的概率。公式意义：换一个情况思考。
10. 条件概率：全概率公式（条件概率扩展，将事件AB划分为较简单的事件AB_1 AB_2 $...AB_n$ 从而方便分开求后单独合并答案。）

11. 期望：（感性理解）某事件发生的结果的长期平均表现（例如：
12. 期望的线性性质
13. 全期望公式

知识规律

为了很好的完成各种概率与期望题目，你至少应该具备如下知识：

1. 计数：组合数学与基础数论（统计答案、推公式）、基本数据结构等（统计贡献）
2. 概率与期望：全概率、贝叶斯、各种模型等 （推公式）
3. DP（期望DP、图上DP、树上DP、DP优化等）
4. 后效性处理：高斯消元或直接算（技巧之一：注意精度）
5. 迭代与递推（计算方法）

方法归纳

1. **直接计算**：这种方法是通过推导出一个数学公式，然后通过程序来计算这个式子的值。

   > 例如，在百事世界杯之旅的例子中，假设有n个不同球星的名字，每个名字出现的概率相同，平均需要买几瓶饮料才能凑齐所有的名字呢?
   >
   > 古典概率：组合计数推公式
   >
   > 几何概率：数形结合算“面积”
   >
   > 条件概率、贝叶斯定理

2. **动态规划**：动态规划是一种应用范围很广的方法，由于概率和期望具有前面提到过的一些性质（特别是**期望的定义**$E(X)=\sum xP(X=x)$以及**期望的“线性”性质**$E(X+Y)=E(X)+E(Y)$），使我们可以**在概率和期望之间建立一定的递推关系**，这样就可以通过动态规划来解决一些概率问题。

   > 例如，在多米诺骨牌的例子中，我们可以通过动态规划来得到最优的摆放方案。设Ei 是摆放 i块骨牌所需要的最少期望次数。

3. **概率——期望系统**：（基于迭代法的高斯消元方法）这是一个利用方程的思想在概率和期望问题中的一个应用。概率——期望系统是一个由一组概率和期望的等式组成的系统，这些等式是由题目的条件以及概率和期望的性质得出的。通过解这个系统，我们就可以得到题目的答案。

   > 应用场景：绝大多数以期望作为状态的期望问题的模型（方程组）

4. **迭代法求期望**： 这指2种不同的操作。
   - 其一是：当问题精度和概率范围比较确定的时候，一定次数后的迭代对结果的影响将可以忽略不计。

> 限制 要求问题有收敛性而且收敛的速度要足够快，而且要求的结果精度不要太高，对于同一规模的不同的输入，迭代法的效率可能会有很大的改变，所以这种方法有可能因为遇到比较坏的数据而失效。

- 其二是：用期望迭代定律（全期望公式）去把联合分布转成单变量分布。这是一种基础技巧，让题目变得可做。（基础）

5. **其他技巧**
    - 模拟法（蒙特卡洛法）：通过计算机模拟随机实验，估算小样本概率值。用于验证做题时的某些猜想，问题特殊合适时，也可以作为正解。
    - 对称性分析：将问题划分为若干对称的子问题，然后计算一个子问题的概率，最后将各个子问题的概率加起来得到总概率。
6. **前人总结的规律**
    1. 概率正着推，期望倒着来
    2. 概率 | 期望 的 公式 | 思想 解决问题
    3. dp转移方程(组)中有环？高斯消元，long double
    4. 树上dp转移，可用dfs对方程迭代求解方程
    5. 当公式或计算时有除法时， 特别要注意分母是否为零
    6. 各种算法相结合，关键是学会分析思路和过程

类型1-可以求概率

DP 求概率：这类题目采用顺推，也就是从初始状态推向结果。

同一般的 DP 类似的，难点依然是对状态转移方程的刻画，需要应用概率论知识设计递推关系。

T1 [CF148D] Bag of mice

> 1400

【题意】

袋子里有 w 只白鼠和 b 只黑鼠$0 <= w, b <= 1000$，公主和龙轮流从袋子里抓老鼠。谁先抓到白色老鼠谁就赢，如果袋子里没有老鼠，且没有谁抓到白色老鼠，那么算龙赢。公主每次抓一只老鼠，龙每次抓完一只老鼠之后会有一只老鼠跑出来。每次Å抓的老鼠和跑出来的老鼠都是随机的。公主先抓。问公主赢的概率。

【题解】

定义状态$f_{i,j}$为还剩i个白老鼠，j 个黑老鼠时公主的胜率

因为还有老鼠会跑，所以这里把**公主抓+恶龙抓**视为一轮操作。

很明显操作前后会有状态转移$f_{\text{某个局面}} \rightarrow \text{操作一下} \rightarrow f_{\text{当前局面}}$

考虑一些胜负情况：

1. 先抓了白老鼠，公主胜率 = $\frac{i}{i+1}$
2. 先黑，后白，胜率 =å 0

3. 先黑，后黑，跑白 = $\frac{j}{i+j}\times\frac{j-1}{i+j-1}\times\frac{i}{i+j-2}\times f_{i-1,j-2}$
4. 先黑，后黑，跑黑 = $\frac{j}{i+j}\times\frac{j-1}{i+j-1}\times\frac{j-2}{i+j-2}\times f_{i,j-3}$

merge 一下

P(公主胜) = $\frac{i}{i+1}+0+\frac{j}{i+j}\times\frac{j-1}{i+j-1}\times\frac{i}{i+j-2}\times f_{i-1,j-2}+\frac{j}{i+j}\times\frac{j-1}{i+j-1}\times\frac{j-2}{i+j-2}\times f_{i,j-3}$

考虑一些边界情况：

1. 全白老鼠，先手必胜$f_{i,0}=1$，全黑老鼠，龙必胜 $f_{0,i}=0$
2. 一只黑老鼠，先手必胜 = $1-$ 先手黑老鼠 = $f_{i,1}=1-\frac{1}{i+1}=\frac{i}{i+1}$

T2 HDU 5985 Lucky-Coins

给n种$(1\le n\le 10)$ 硬币,每种硬币有若干种，并给你每种硬币的个数和正面朝上的概率p $(0.4\le p\le 0.6$，硬币总数小于10^6)。每次将所有硬币扔一下，然后去掉背面朝上的硬币，再重复扔到只剩下一种硬币或者没有硬币。最后剩下的那种硬币叫幸运硬币，问每种硬币成为幸运硬币的概率。要求精度10^{-6} 。

> 这道题可以帮助大家进一步理解概率的一些特点。
>
> 按照常见做题套路，可以考虑一种特例，如果我只有1种100个硬币，0.5的概率上翻，第一轮期望剩下50个，再来一轮，期望剩下25个，再来一轮期望剩下12.5个，.... 可以发现，其实翻几轮后，100个硬币就没剩下几个了。这启发我们，可以考虑从轮次入手。考虑极端情况"对于题目要求总量不超过 10^6 假设概率为 0.6 那么也就是大概20轮后全部GG " 发现也可做。
>
> 那么为了计算某种硬币成为Lucky-Coin的概率,可以有：
>
> - Part 1 可以先算某种硬币恰好在某一天成为幸运硬币的概率，然后在把前x天的概率全部加起来（第x天之后的概率因为非常小，所以不影响大橘，直接略过）
> - Part 2 第i种硬币在第j天的成为Lucky-Coin的概率 = 其他硬币在第j 天全GG * 第i 种硬币在前面活的好好的在第$j+1$天突然全部暴毙的概率。（如果不暴毙其意义为在第$k\sim\infty$ 中的任意一轮成为幸运硬币的概率。）
>
> Part1 更具体一点，定义$f_{i,j}$ 表示第i种硬币 在j 轮后全部GG的概率，p_i 表示第i种硬币正面向上的概率，n_i 表示第i种硬币的个数，则
>
> $$f_{i,j}=(1-p_i^j)^{n_i} \tag{1}$$
>
> 则第i种硬币 在j 轮后至少还活1个的概率：
>
> $$g_{i,j}=1-(1-p_i^j)^{n_i}=1-f_{i,j} \tag{2}$$
>
> 根据Part2的思想，定义ans_i 为第i 种硬币成为lucky-coin的概率，则有：
>
> $$\text{ans}_i=\sum_{j=0}^{\infty}(g_{i,j}-g_{i,j+1})\prod_{k=1,k\neq i}^{n}f_{k,j} \tag{3}$$
>
> 根据经验，要算到第100天才行。
>
> ```
>
> double f[11][1010];
> double qpow(double a, int b) {
> ```

```cpp
    double res = 1;
    while (b) {
        if (b & 1) res = res * a;
        b >>= 1;
        a *= a;
    }
    return res;
}

int main() {
    int T;
    scanf("%d", &T);
    while (T--) {
        int n, ni, mxk = 1000;
        double pi;
        scanf("%d", &n);
        for (int i = 0; i < n; i++) {
            scanf("%d %lf", &ni, &pi);
            double pik = 1;
            for (int k = 0; k < mxk; k++, pik *= pi) {
                f[i][k] = qpow(1 - pik, ni);
                //某种硬币在第j天全部死亡的概率
            }
        }
        for (int i = 0; i < n; i++) {
            double ans = 0;
            for (int k = 0; k < mxk - 1; k++) {
                double tmp = 1;
                for (int j = 0; j < n; j++) {
                    if (j == i) continue;
                    tmp *= f[j][k];
                }
                ans += (f[i][k+1] - f[i][k]) * tmp;
                //当天存活下一天死亡的概率*其他所有硬币死亡的概率
            }
            printf("%.6f%c", ans, " \n"[i==n-1]);
        }
    }
    return 0;
}
```

T3 [CF601C] Kleofáš and the n-thlon

> 2300

m 个人参加 n 项比赛。每场比赛有一个分值, 在 $[1,m]$ 之间 (每轮比赛中m种分有且仅有一人取得), 且同一场比赛得分两两不同。总分为每场比赛得分之和。给出某个人A每场比赛的得分 c_i ，求这个人总分排名（第几小）的期望值。$1 \le n \le 100, 1 \le m \le 1000$

因为 期望 $= \sum$ 概率 $*$ 状态值

所以$E(A\text{的期望总分排名}) = P(\text{比完}n\text{场后总分} < A\text{的总分}) * (m-1)$

又因为$P(\text{比完}n\text{场后总分} < A\text{的总分}) = \sum^{i<A\text{的总分}} P(i)$

所以我们可以先定义$f_{i,j}$ 表示前i场，获得总分为j的概率

$$f_{i,j} = \sum_{k=1,k\neq c_i}^{min(m,j)} \frac{f_{i-1,j-k}}{m-1} \tag{4}$$

上面这个公式的意思是：每种得分被获得概率均为$\frac{1}{m-1}$ 且c_i 已经被指定给A了

记这个人得分为 $S = \sum_{i=1}^{n} c_i$ ，得分比这个人小的概率$P(A)$为

$$P(A) = \sum_{j=1}^{S-1} f_{n,j} \tag{5}$$

则根据期望定义，可计算出期望排名$E(A)$为:

$$E(A) = 1 + (m-1) \times P(A) = 1 + (m-1) \times \sum_{j=1}^{S-1} f_{n,j} \tag{6}$$

当然你也可以用期望DP做。

算一下复杂度

DP算$f_{i,j}$ 开销在于n轮比赛+至多 nm 分（每轮）都要处理，所以需要 $n * nm$ 的开销，即

$O(n^2m) = (1e2)^2 \times (1e3) = 1e7$

一个实现细节:

1. 在求解$f_{i,j}$时考虑利用前缀和优化 DP 内层循环对于 $f_{i,j}$ 的求解(需要在O(1)内求出来)由此会带来一些细节的改变
2. 具体来说：如果想不清楚请在课后思考这样一个问题：在DP 的i,j 动态变化的过程中第i 场c_i是否对于所有i,j 都会被计算？”
3. 如果还是想不清楚，请直接搜索一篇代码。

类型2- 可以求期望

T4 [CF908D]New Year and Arbitrary Arrangement

> 2200

题目大意: 给出 $\mathrm{k}, \mathrm{pa}, \mathrm{pb}$, 一开始有一个空序列, 每次有 $\frac{\mathrm{pa}}{\mathrm{pa}+\mathrm{pb}}$ 的概率在末尾放一个 a ，有 $\frac{\mathrm{pb}}{\mathrm{pa}+\mathrm{pb}}$ 的概率在末尾放一个 b, 当存在 k 个形如 ab 的子序列时停止, 问此时形如子序列 "ab" 的期望个数是多少？（注意aab算两个子序列ab）$(1 <= k <= 1000，1 <= p_a, p_b <= 1000000)$

一些思维过程：

问题的约束条件是字符串的子串中有k个ab，即达到某种状态（子串中有k个ab）时的期望长度

草稿纸大法整点小数据帮助思考：aaab = 3个ab， aaaba = 3个ab， aaabab = 3+4个ab

显然可以思考出一些转移的核心操作（通过在末尾添加一个字符a或者字符b）

但是状态不定义，是无法思考出转移过程的。

考虑这个状态需要什么记录什么信息？

根据添加过程，最显然需要统计的信息是子串ab的个数。因为这告诉我们什么时候停止。除此之外，还要考虑转移过程中，添加b后会增加多少个ab子串，即需要统计前面a的个数。

所以有状态定义：

设$f_{i,j}$表示给定前缀中形如'a'的子序列有i个，形如'ab'的子序列有j个时期望ab的个数的答案。

同时因为终止状态略多，而初始状态固定单一，所以考虑倒着算期望：

$f_{i,j} = \frac{p_a}{(p_a+p_b)} * f_{i+1,j} + \frac{p_b}{(p_a+p_b)} * f_{i,i+j}$

显然倒推到$f_{0,0}$ 就是答案。

考虑边界条件初始化的问题，这是一个难点，因为字符串可以无限长。

通过思考，其实这个无限长也是有限的：

因为如果前面一群a，后面突然添一个b，那么当a足够多时$(\geq k)$ 再添一个b就会满足条件，但是如果不添加b而是加任意多个a，则不能满足，所以我们可以将$(i+j \geq k)$ 作为一个边界条件。

那么当$(i+j \geq k)$ 达到边界条件的时候，$f_{i,j}$ 的边界值是?

> 提示1 $f_{i,j}$的意思为串中有i个字符a，有j个ab子串的的时候, ab 子序列个数的期望值为 $f_{i,j}$
>
> 提示2 先根据串的生成操作推一推$f_{i,j}$

$$\begin{aligned} f_{i,j} = & \frac{p_a}{p_a+p_b} f_{i+1,j} + \frac{p_b}{p_a+p_b} f_{i,j+i} \\ = & \frac{p_a}{p_a+p_b} f_{i+1,j} + \frac{p_b}{p_a+p_b} (j+i) \end{aligned} \tag{7}$$

现在获得了$f_{i,j}$ 与 $f_{i+1,j}$ 的转移关系。

如果我再根据定义在在其中找一些关系，那么我就可以建立方程组联立求解。

根据定义，从$f_{i,j}$ 到 $f_{i+1,j}$ 串差异是差个`a`，由于j是相同的，所以`a`一定是放在串的后面，考虑到我们最终得给他插入一个`b` 就结束了，那么现在多增加的这个`a` 在后面对答案的贡献为+1 所以可以得到他们之间的一个更具体的关系：

$$f_{i,j} + 1 = f_{i+1,j} \tag{8}$$

联立公式 7 和公式 8 可得

$$f_{i,j} = \frac{p_a}{p_b} + i + j \tag{9}$$

好，坐到这里，最难的边界条件就被我们解决了。

同时进一步考虑$f_{0,0}$的转移过程，发现根据公式算出来竟然是 $f_{0,0}=p_a\times f_{1,0}+p_b\times f_{0,0}$，

但考虑第一个a出现之前的情况，无论有多少个b，都不能构成ab，所以都可以无视掉，所以答案可以转化为统计$f_{1,0}$ 的期望长度。

然后就是一些细节，注意整体是在 模p 意义下运算。

由于逆推i,j且最大不超过k，所以复杂度保证到 $O(k^2)$

T5 CF1540B Tree Array

> 2300

一棵 n 个结点的无根树（$2\le n\le 200$），开始时等概率选一个点标记，然后每一步在和已经选择的点集的相邻点的集合中，等概率选择一个未被标记过的点进行标记。把点编号按照选择顺序排序，求排序后点编号的期望逆序对数，答案对 10^9+7 取模。

分析：根据题目描述，显然选择哪个节点作为root会对结果有影响，按照决策树的思路，我们应枚举root，对于每一个root统计期望值，最后/n求选择该root时对总体期望的贡献。

在确定root后，进一步思考关于逆序对的期望计算?

为什么会产生逆序对,因为对于任意两节点 $a<b$, (a,b) b 有一定概率比a 先被访问，如下图所示

在指定root的情况下，我们想要求出这个概率，那就可以进一步枚举数对(a,b) 中b比a先访问的概率。

因为是在树上随机拓展相邻点集，当到达lca(a,b)后，可以感性理解为在a~lca(a,b) 和 b～lca(a,b) 两条链上或其他链上随机摸索，有一定概率先摸到b后摸到a，那么(a,b)构成逆序对，很显然这个概率只与$lca(a,b)$到a，b的深度有关系,他们的深度可以表示为，$dist(lca(a,b),a),dist(lca(a,b),b)$

为了求出这个概率，我们可以将这个问题转化为，有两个大小为$\mathrm{dist}(lca(a,b),a)$和$\mathrm{dist}(lca(a,b),b)$的栈，有一个任意的$p$从两个栈之一中移除一个节点（和$1-2p$什么都不做），并找出$\mathrm{dist}(lca(a,b),b)$在$\mathrm{dist}(lca(a,b),a)$之前达到零的概率。

实际上，概率p并不重要。是因为减小x或减小y的概率总是相同的，所以无论p如何，只要确定了两个栈的深度，则最终状态的概率就确定了。

我们提出一个函数$f_{x,y}$ ，定义一个大小为y的栈在一个大小为x的栈之前变为0的概率 ,很显然，我们在最多只有200个节点的树上，我们可以与处理出所有情况下的概率 $f_{x,y}$

$$f_{x,y}=\frac{f_{x-1,y}+f_{x,y-1}}{2} \tag{10}$$

> 提示：边界条件 $f_{0,j}=1$

来总结一下刚才的思路过程：

1. 预处理$F_{x,y}$
2. 枚举$root$节点
3. 在每次指定的$root$节点上处理出$lca(a,b)$
4. 枚举$a<b$,(a,b)构成逆序对的概率并求和 即$ans+=F[\mathrm{dist}(lca(a,b),a)][\mathrm{dist}(lca(a,b),b)]$

5. 因为是期望（某种事件的长时间平均表现）枚举的n个$root$节点相当于重复统计n次，则$/n$即可。

时间复杂度是$O\left(N^3\log N\right)$ 或预处理LCA $O\left(N^3\right)$ 。$N\leq 200$ 均可过。

一份参考代码：忘记来源了，如果找到了可以给czc说一下。

```
#include<iostream>
#include<cstdio>
using namespace std;
const int N = 505, MOD = 1e9 + 7;
typedef long long ll;

int head[N], ver[N], net[N], idx;
int fa[N][11], dep[N], f[205][205];

void add(int a, int b){
    net[++idx] = head[a], ver[idx] = b, head[a] = idx;
}

void dfs(int u, int f){//f 上一层来源点，避免又搜回去了。
    dep[u] = dep[f] + 1, fa[u][0] = f;
    for (int i = 1; i <= 9; i++) fa[u][i]=fa[fa[u][i - 1]][i - 1];
    for (int i = head[u]; i; i = net[i]){
        int v = ver[i];
        if (v == f)continue;
        dfs(v, u);
    }
}

int lca(int x, int y){
    if (dep[x] < dep[y])swap(x, y);
    for (int i = 9; i >= 0; i--)
        if (dep[fa[x][i]] >= dep[y])
            x = fa[x][i];
    if (x == y)return x;
    for (int i = 9; i >= 0; i--)
        if (fa[x][i] != fa[y][i])
            x = fa[x][i], y = fa[y][i];
    return fa[x][0];
}

int ksm(int a, int b){
    int res = 1;
    while (b){
        if (b & 1)
            res = (ll)res * a % MOD;
        a = (ll)a * a % MOD;
        b >>= 1;
    }
    return res;
```

```
}

int main(){
    int n;
    scanf("%d", &n);
    for (int i = 1; i < n; i++){
        int u, v;
        scanf("%d%d", &u, &v);
        add(u, v), add(v, u);
    }
    int inv2 = 500000004; //2 在mod 意义下的inv
    //初始化f
    for (int i = 1; i <= n; i++) f[0][i] = 1; //f边界条件,
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            f[i][j] = (ll)(f[i - 1][j] + f[i][j - 1]) * inv2 % MOD;

    ll ans = 0;
    for (int i = 1; i <= n; i++){
        dfs(i, 0);//枚举根，以i为root开始处理dfn等信息方便后面lca
        for (int j = 1; j <= n; j++){ //枚举(a,b)
            for (int k = 1; k < j; k++){
                int u = lca(j, k);
                (ans += f[dep[j] - dep[u]][dep[k] - dep[u]]) %= MOD;
            }
        }
    }
    printf("%lld", ans * ksm(n, MOD - 2) % MOD);//注意最后要/n (即*inv(n))
    return 0;
}
```

T6 [CF605E] Intergalaxy Trips

> 2700

n 个点的有向完全图。

$i \to j$ 的边每天可通过的概率均为 $p_{i,j}$, 若 $i = j$, 有 $p_{i,j} = 1$ 。

每天选择一条存在的出边走过去(也可以呆在原地)。

求最优策略下从 1 到 n 的期望天数。(误差不超过10^{-6})

$n \le 10^3$

思路：模拟过程：首先能想到，每个节点到节点n都应该有个期望天数，那就先定义一个 E_x 表示结点 x 到节点n 的期望天数。

然后可以想象出最优策略下的期望在图上计算“传播”的过程中的一些规律：

1. 点x 接下来转移的点必须是 `当前能走的` 出边中E值最小的点。
2. 点x 不会做蠢事，即若下一个到达点的期望 $< E_x$ 那就干脆不更新了（如果让答案变得更差，宁愿原地等一天，也不转移）

这和dij中的松弛思想非常类似。

考虑到松弛，那我们完全可以进一步思考：对这个松弛思想进行优化：先设置一个点集A={n},$E_n = 0$，然后每一轮从点集A中选一个期望最小的旧点（优先队列实现）向外拓展新点，并计算新点的期望，然后再将该点加入点集A中，然后再进行重复操作，直至点1被加入到点集中。

这个做法可以保证到目标点的时候期望是最小的。

然后接下来开始算期望：如果要从节点i 转移到节点j，那么需要保证对于所有$< E_j$ 的期望的节点的对外通道都是关闭的(因为需要在当前打开的通道中选取一条最小的边）。且$i \to j$ 通道需要打开。

若 E_i 表示尚未拓展的节点（改节点与已拓展集合直接相连）的期望天数，E_j 表示当前已经拓展到的节点的到达期望天数，$P_{i,j}$ 为点j到点 i 通道打开的概率，可以有个求E_i 的公式：

$$E_i = \sum E_j \times P_{i,j} \prod_{E_k < E_j} (1 - P_{i,k}) \tag{11}$$

发现公式 11 只计算了至少开一条通道的概率。还有罚站的概率（一条通道都不开）也要考虑到期望中，即还需要处理至少开一条通道的概率：

$$E_i' = \frac{\sum E_j \times P_{i,j} \prod_{E_k<E_j}(1 - P_{i,k})}{1 - \prod_{k \neq i}(1 - P_{i,k})} \tag{12}$$

剩下的大家应该自己可做了，略。

T7 CF1267G Game Relics

> 3000

给定 n 张卡，购买第 i 张卡的价格为 c_i 你有两种方式获得卡牌：

- 随机抽一张卡，花费为 x，如果得到了一个已经有的卡，那么返还 $\frac{x}{2}$
- 直接购买一张卡，花费为 c_i

求期望最少花多少集齐所有卡。精度要求 10^{-9}

$n \le 100, \sum c_i \le 10^4$

题解: 策略一定是先抽卡，然后再全买。

思考一下策略细节：

Think1 : 假设我抽到某个局面，已经抽出了i种不同的卡，再想抽出$i+1$张不同的卡的期望?

先考虑抽卡期望E_i 表示抽到i张不同的卡的期望次数。

- 抽了卡，有 $\frac{i}{n}$ 的概率还是之前的卡（所以还是得继续抽）
- 抽了卡,有$\frac{n-i}{n}$ 的概率是一张新卡（不用抽了）

$$E_i = \frac{i}{n}\left(E_i + \frac{x}{2}\right) + \frac{n-i}{n} \times (x + E_{i+1}) \tag{13}$$

则

$$E_i - E_{i+1} = (\frac{n}{n-i} + 1) \times \frac{x}{2} \tag{14}$$

所以每次整出一张新卡（从$i \to i+1$）期望的花费为：$(\frac{n}{n-i}+1)\times \frac{x}{2} $

> 或者我们可以根据一些概念和规律直接把这个期望公式给写出来，$n-i$ 表示当前还有多少张卡没抽出来，n 表示卡的总数，因为抽卡事件是伯努利实验，抽出新卡的概率为 $\frac{n-i}{n}$ 所以需要抽的期望次数是 $\frac{n}{n-i}$ ，+1表示最后一次把卡抽出来了，并且支付了x的代价。

Think2 : 什么时候直接购买?

刚刚已经算出来，当($i \to i+1$)的时候，抽卡的代价为$(\frac{n}{n-i}+1)\times \frac{x}{2} $

因为如果一买肯定要剩下的全买，那么对于这张卡的平均代价为 $\frac{\sum_{i\in Rest-card} c_i}{n-i}$ 若抽卡的代价比买卡的平均代价小，那肯定继续抽，反之，则肯定直接全买剩下。

> 提示：随着操作的进行，i在不断增大， 抽卡的代价在不断增大，同时剩余卡的平均代价是在减少（分子减少>1,分母减少1）

Think3: 如何求答案?

因为$\sum c_i \le 10^4$ 所以设置$dp_{i,j}$ 表示得到了i张卡，代价和为j的方案数。

整个背包：$dp_{i,j} = \sum dp_{i-1,j-c_k}$

$dp_{i,j}$ 这个局面出现的概率为$\frac{dp_{i,j}}{C_n^i}$ 转移代价为$\min((\frac{n}{n-i}+1)\times\frac{x}{2}, \frac{\sum_{i\in Rest-card} c_i}{n-i})$

由于期望的线性性，所以我们可以分别计算出第n行的期望，然后再相加即可。

复杂度 $\mathcal{O}(n^2\sum c)$

具体实现细节想不清楚的，需要详细参考代码

类型3- DP后效性处理

T8 CF24D Broken robot

> 2400

题目大意: 给出一个 $n*m$ $(1 \leq n,m \leq 1000)$的矩阵区域, 一个机器人初始在第 x 行第 y 列, 每一步机器人会等概率地选择停在原地, 左移一步, 右移一步, 下移一步, 如果机器人在边界则不会往区域外移动, 问机器人到达最后一行的期望步数。

定义$f_{i,j}$表示从(i,j)走到最后一行的期望步数(机器人在最后一行的期望为0)

$$f_{i,j}=\begin{cases}1+\frac{1}{3}(f_{i,j}+f_{i,j+1}+f_{i+1,j}) & j=1\\ 1+\frac{1}{4}(f_{i,j}+f_{i,j-1}+f_{i,j+1}+f_{i+1,j}) & 1<j<m\\ 1+\frac{1}{3}(f_{i,j}+f_{i,j-1}+f_{i+1,j}) & j=m\end{cases} \tag{15}$$

由于可以原地呆，所以肯定有后效性。不能直接求。

但是因为不能向上走，即下去了就上不去了，且因为初始状态确定(x,y) 但是最终状态有多个，我们可以从第$i+1$ 行的信息反推第i 行，那么按行整理公式，可以得到:

$$\begin{aligned}-\frac{1}{3}f_{i+1,j}-1 &= -\frac{2}{3}f_{i,j}+\frac{1}{3}f_{i,j+1} & j=1\\ -\frac{1}{4}f_{i+1,j}-1 &= \frac{1}{4}f_{i,j-1}-\frac{3}{4}f_{i,j}+\frac{1}{4}f_{i,j+1} & 1<j<m\\ -\frac{1}{3}f_{i+1,j}-1 &= \frac{1}{3}f_{i,j-1}-\frac{2}{3}f_{i,j} & j=m\end{aligned} \tag{16}$$

注意1： 此时i+1的信息已经知道了，求i行的信息。

注意2： m=1比较特殊，不能向左或右移动，所以状态转移公式会有变化。

$$-\frac{1}{2}f_{i+1,j}-1=-\frac{1}{2}f_{i,j} \tag{17}$$

对于第i行的m列来说，我们发现公式16 中有m个方程，且有m个未知数。那么可以考虑高斯消元求解。例如下面是m=5时的矩阵

$$\begin{bmatrix}-\frac{2}{3} & \frac{1}{3} & 0 & 0 & 0\\ \frac{1}{4} & -\frac{3}{4} & \frac{1}{4} & 0 & 0\\ 0 & \frac{1}{4} & -\frac{3}{4} & \frac{1}{4} & 0\\ 0 & 0 & \frac{1}{4} & -\frac{3}{4} & \frac{1}{4}\\ 0 & 0 & 0 & \frac{1}{3} & -\frac{2}{3}\end{bmatrix}=\begin{bmatrix}-\frac{1}{3}f_{i+1,j}-1\\ -\frac{1}{4}f_{i+1,j}-1\\ -\frac{1}{4}f_{i+1,j}-1\\ -\frac{1}{4}f_{i+1,j}-1\\ -\frac{1}{3}f_{i+1,j}-1\end{bmatrix} \tag{18}$$

本来高斯消元的复杂度为$O(n^3)$ 但是我们愉快的发现，这个矩阵非常特殊。可以在$O(m)$ 内完成消元和回代。

找到一篇良心代码:

```
#include <bits/stdc++.h>
using namespace std;

template <class T>
inline void read(T &x) {
    x = 0;
    char c = getchar();
    bool f = 0;
```

```cpp
    for (; !isdigit(c); c = getchar()) f ^= c == '-';
    for (; isdigit(c); c = getchar()) x = x * 10 + (c ^ 48);
    x = f ? -x : x;
}

const int MAXN = 1000;
int n, m, x, y;
double a[MAXN + 5][MAXN + 5], f[MAXN + 5][MAXN + 5];

inline void Gauss() {
    for (int i = 1; i < m; ++i) {
        double t = a[i][i];
        a[i][i] = 1;
        a[i][i + 1] /= t;
        a[i][m + 1] /= t;
        t = a[i + 1][i];
        a[i + 1][i] = 0;
        a[i + 1][i + 1] -= t * a[i][i + 1];
        a[i + 1][m + 1] -= t * a[i][m + 1];
    }//消元，消去下一行方程开头的未知数
    a[m][m + 1] /= a[m][m];
    a[m][m] = 1;//求出最后一个未知数的解
    for (int i = m - 1; i; --i)
        a[i][m + 1] -= a[i + 1][m + 1] * a[i][i + 1];
    //回带，消去下一行方程末尾的未知数
}

int main() {
    read(n), read(m), read(x), read(y);
    for (int i = n - 1; i >= x; --i) {
        if (m == 1) {
            a[1][1] = -1.0 / 2;
            a[1][m + 1] = -f[i + 1][1] / 2.0 - 1;//特判 m = 1
        } else {
            a[1][1] = -2.0 / 3;
            a[1][2] = 1.0 / 3;
            a[1][m + 1] = -f[i + 1][1] / 3.0 - 1.0;
            for (int j = 2; j < m; ++j) {
                a[j][j] = -3.0 / 4;
                a[j][j - 1] = a[j][j + 1] = 1.0 / 4;
                a[j][m + 1] = -f[i + 1][j] / 4.0 - 1;
            }
            a[m][m] = -2.0 / 3;
            a[m][m - 1] = 1.0 / 3;
            a[m][m + 1] = -f[i + 1][m] / 3.0 - 1;
        }//构造矩阵
        Gauss();//高斯消元
        for (int j = 1; j <= m; ++j) f[i][j] = a[j][m + 1];
        //赋值求出的解
```

```
    }
    printf("%.10lf\n", f[x][y]);
    return 0;
}
```

另外一种有趣的思路：https://blog.csdn.net/Eric1561759334/article/details/105607180

T9 ZOJ 3329 One Person Game

有三个骰子Die_1, Die_2, Die_3，面数分别为k_1, k_2 , k_3 个面，每个面上分别写有数字$1 \sim k_1$，$1 \sim k_2$，$1 \sim k_3$ ，任意一个骰子的每个面朝上的概率相同。每次掷骰子，如果Die_1=`a`,Die_2=`b`,Die_3=`c` 则总分置0，否则总分加上三个骰子的本轮分数之和。当分数大于n时结束。求游戏的期望步数。初始总分数为0。，
$0 \leq n \leq 500, 1 < k_1, k_2, k_3 \leq 6$,$a \in [1, k_1]$,$b \in [1, k_2]$,$c \in [1, k_3]$ 多测，至多300组数据。

思路：

分数的变化过程存在明显的操作阶段。考虑大问题转化为小问题求解。

即，定义 f_i 为存到第 i 分时还需要走多少步（期望步）。

f_n = 0 , 答案存储在f_0 中。

由于存在分数归零的情况，显然f的状态转移是存在环的。我们可以先把方程组给列出来，然后高斯一下。

先预处理 p_k 表示出一轮扔出k分的概率,p_0 表示扔出a, b, c 喜提归零的概率。

$$f_i = \sum_{k=1}^{k_1+k_2+k_3} (f_{i+k} * p_k) + f_0 p_0 + 1 \tag{19}$$

其实可以不用算高斯，发现任意一f_i 都与 f_0 有关系，而 f_0 是我们的答案。所以考虑直接构造一个方程：

$$f_i = a_i * f_0 + b_i \tag{20}$$

并将上面的公式 19 带入构造的方程20中求解 (将公式 19 中的 f_{i+k} 用公式20 展开)。

$$f_i = \sum((a_{i+k} * f_0 + b_{i+k}) * p_k) + f_0 * p_0 + 1 \tag{21}$$

$$f_i = \sum(a_{i+k} * f_0 * p_k + b_{i+k} * p_k) + f_0 * p_0 + 1 \tag{22}$$

$$dp[i] = (\sum(a_{i+k} * p_k) + p_0) * f_0 + \sum(b_{i+k} * p_k) + 1 \tag{23}$$

因为系数对应等，所以 a_i, b_i 是一个递推公式

$a_i=\sum p_k*a_{i+k}+p_0$

$b_i=\sum p_k*b_{i+k}+1$

因为f_i参数略多，考虑求出$i=0$的情况，所以直接把a_0,b_0推出来，带入公式$f_0=a_0*f_0+b_0$求出f_0

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

double A[600],B[600];
double p[100];
int main(){
    int T;
    int k1,k2,k3,a,b,c;
    int n;
    scanf("%d",&T);
    while(T--){
        scanf("%d%d%d%d%d%d%d",&n,&k1,&k2,&k3,&a,&b,&c);
        double p0=1.0/k1/k2/k3;
        memset(p,0,sizeof(p));
        for(int i=1;i<=k1;i++)
          for(int j=1;j<=k2;j++)
            for(int k=1;k<=k3;k++)
              if(i!=a||j!=b||k!=c)
                p[i+j+k]+=p0;
        memset(A,0,sizeof(A));
        memset(B,0,sizeof(B));
        for(int i=n;i>=0;i--){
            A[i]=p0;B[i]=1;
            for(int j=1;j<=k1+k2+k3;j++){
                A[i]+=A[i+j]*p[j];
                B[i]+=B[i+j]*p[j];
            }
        }
        printf("%.16lf\n",B[0]/(1-A[0]));
    }
    return 0;
}
```

相关习题

[CF768D] [HNOI2015] 亚瑟王 [HNOI2013] 游走 [HDU 4418] Time travel 题解 [CF123E] Maze

 文档结束，第17页无内容