Hello World ! ---期望DP

/\*

期望DP 概率DP 好题

题意：NOI2005 一个无向图。老鼠步会选择去一个相邻点或者呆在原点，而却这些地方的概率是相同的；猫每一步选择一个最靠近老鼠的相邻点，如果有多个，则选择标号最小的那一个，如果走完第一步后仍然没有吃到老鼠，猫还可以在在本段时间内再向老鼠走近一步（也就是猫一步能走两个点）。没次都是猫先走，老鼠后走。某时刻猫和老鼠在同一点，则猫把老鼠吃掉。给出猫和老鼠的初始位置，问老鼠被吃掉的期望步数。

思路：设(i,j)为猫在i点老鼠在j点，目标状态为(i,i)，e(i,i) = 0。 这跟我之前做的有固定初始状态和目标状态不一样，这里只有固定的初始状态，目标状态为(i,i).

先n次bfs预处理出p[i,j]表i->j的最短路上与i相邻且编号最小的顶点，即猫在i鼠在j时，猫下一步会走到的点。感觉代码还是挺简单巧妙的。

对于当前状态，设deg[i]为点的度数。可以确定，猫在下一步所在的顶点p[p[i,j], j]，老鼠在点v(v==i或者v与i相邻)，概率1/(deg[i]+1).

e(p[p[i,j],j] , j) + Σe(p[p[i,j], j] , v)

e(i,j) = 1 + ---------------------------------------------

deg[i] + 1

初始化e(i,i) = 0, 若p[p[i,j], j] = j或p[i,j] = j，则e(i,j) = 1

注意：这里老鼠虽然能留在原地，，但是猫是一直在跑的，所以该图没有自环！(i,j)没有自环！我一开始用带自环的公式就错了。

细节还是挺多的，但不烦，所以我觉得是道好题。

\*/

//bzoj 1415

void init(int s, int son, int u) //其实我觉得预处理出p[i][j] 是比较简单巧妙的~

{

for(int Size = adj[u].size(), i = 0; i < Size; i++) {

int v = adj[u][i];

if(dis[v] > dis[u] + 1) {

dis[v] = dis[u] + 1;

p[s][v] = son;

init(s, son, v);

}

}

}

void dfs(int i, int j) //当前状态(i,j) cat\_and\_mouse

{

if(f[i][j] + 1 > ZERO) return ;

f[i][j] = 0;

int v = p[i][j];

for(int Size = adj[j].size(), k = 0; k < Size; k++) {

int v = adj[j][k];

dfs(p[p[i][j]][j], v);

f[i][j] += f[ p[p[i][j]][j] ][v];

}

dfs(p[p[i][j]][j], j);

f[i][j] += f[p[p[i][j]][j]][j];

f[i][j] /= (adj[j].size() + 1);

f[i][j] += 1;

}

int main()

{

int Cat, Mouse, x, y;

scanf("%d%d%d%d", &n, &m, &Cat, &Mouse);

for(int i = 0; i < m; i++) {

scanf("%d%d", &x, &y);

adj[x].push\_back(y);

adj[y].push\_back(x);

}

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

sort(adj[i].begin(), adj[i].end()); //猫会挑最小编号走

memset(dis, 127, sizeof(dis));

dis[i] = 0;

for(int Size = adj[i].size(), j = 0; j < Size; j++) { //预处理出p[i,j]

int v = adj[i][j];

p[i][v] = v;

dis[v] = 1;

init(i, v, v);

}

}

for(int i = 0; i <= n; i++) for(int j = 0; j <= n; j++) f[i][j] = -1;

for(int i = 1; i <= n; i++)

for(int j = 1; j <= n; j++) {

if(p[i][j] == j || p[p[i][j]][j] == j) f[i][j] = 1;

}

for(int i = 1; i <= n; i++) f[i][i] = 0;

dfs(Cat, Mouse);

printf("%.3f\n", f[Cat][Mouse]);

return 0;

}

/\*

期望DP 概率DP

题意：一个bug仅属于一个系统和一个分类，现有n个系统和s个分类，一天可以找出一个bug，问找到的bug覆盖所有系统和分类所期望的天数。

思路：可以把该题建成一个有向图：状态(i,j)表已找出的bug覆盖i个系统j个分类，e(i,j)在图里作为一个节点，在数学上表示(i,j)状态到(n,s)的期望，所以e(n,s) = 0, 目标是求e(0,0)；概率关系看作有向边，易知这是个有自环的图。

因为想用递推直接搞，而递推的条件是图无环，我们可以通过一些数学上的方法把自环消掉。直接上公式了。

E(T) = c + Σ(p\*E(S)) 无环图,c表示一步的代价，p表示S->T的概率

(1-p')E(T) = c + Σ(p\*E(S)) 有自环图，p'表示自己到自己的概率

我是直接上递归的，当然DP的比较快。

e(i,j) = p(i,j)\*e(i,j) + p(i+1,j)\*e(i+1,j) + p(i,j+1)\*e(i,j+1) + p(i+1,j+1)\*e(i+1,j+1)

\*/

//poj 2096

double calc(int i, int j)

{

if(e[i][j] + 1 > ZERO) return e[i][j];

e[i][j] = 1;

if(i + 1 <= n && j + 1 <= s) e[i][j] += calc(i+1, j+1) \* ((1-i\*1.0/n)\*(1-j\*1.0/s));

if(i + 1 <= n) e[i][j] += calc(i+1, j) \* ((1-i\*1.0/n)\*(j\*1.0/s));

if(j + 1 <= s) e[i][j] += calc(i, j+1) \* ((i\*1.0/n)\*(1-j\*1.0/s));

e[i][j] /= (1 - 1.0\*i/n\*j\*1.0/s);

return e[i][j];

}

int main()

{

scanf("%d%d", &n, &s);

for(int i = 0; i <= n; i++) for(int j = 0; j <= s; j++) e[i][j] = -1;

e[n][s] = 0;

printf("%.4f\n", calc(0, 0));

return 0;

}

/\*

概率DP

题意：在ACM比赛种出题，共M道题，T个队，要求冠军出的题数不少于N，问每队至少过一题且冠军至少N题的概率。

思路：易知每个队至少出一题的概率是相互独立的。p[i][j]表示队i出j题的概率，g[i][j][k]表示队i对于前j题而言作出k题的概率。

g[i][j][k] = g[i][j-1][k-1] \* p[i][j] + g[i][j-1][k] \* (1 - p[i][j])

则对于队i，至少作出1题的概率为 1 - g[i][M][0].

则每个队至少作出1题的概率为 ans = ∏(1-g[i][M][0])

接着考虑至少有一队过N题的概率 = ans - 所有队都没过N题的概率

= ans - ∏( Σ g[i][M][j] ) , j = 1,2,...,N-1

\*/

//poj 2151

#include <stdio.h>

double get[1005][35][35], p[1005][35];

int main()

{

int Team, Topic, MaxTopic;

while(scanf("%d%d%d", &Topic, &Team, &MaxTopic), Topic+Team+MaxTopic) {

for(int i = 1; i <= Team; i++)

for(int j = 1; j <= Topic; j++) scanf("%lf", &p[i][j]);

double ans = 1.0;

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

get[i][0][0] = 1;

for(int j = 1; j <= Topic; j++) {

get[i][j][0] = get[i][j-1][0] \* (1 - p[i][j]);

for(int k = 1; k <= j; k++)

get[i][j][k] = get[i][j-1][k-1] \* p[i][j] + get[i][j-1][k] \* (1 - p[i][j]);

}

ans \*= (1 - get[i][Topic][0]); //每队至少过一题的概率

}

double tot = 1.0;

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

double tmp = 0;

for(int j = 1; j < MaxTopic; j++) tmp += get[i][Topic][j];

tot \*= tmp; //没有一个队过N题的概率

}

printf("%.3f\n", ans - tot);

}

return 0;

}

/\*

概率DP

题目：有2^n个队，相邻的两两打淘汰赛，求最后哪个队夺冠的概率最大。假设a1,a2,a3,..ak(i<k, ai<ak)为前一轮胜利的参赛者序列，则当前轮由a1,a2对手，a3,a4对手...

思路：dp[i][j]表示第i轮时队j胜利，那么dp[i][j]前提就是i-1轮的时候j胜利，而且在第i轮赢了对手。

那么接下来就是枚举j在第i轮时可能的对手k了，j要在i轮胜，就要在i轮把这些可能的对手k都打败。

通过二进制发现规律，j与k的高位都相等，到了第i-1位刚好相反，就用这个性质判断是不是j在i轮的对手。

dp[i][j] = Σ(dp[i-1][j] \* dp[i-1][k] \* p[j][k]), (k >> (i-1) ^ 1) == (j >> (i-1))

收获：原来这个规则下有这么个规律呀，j,k能在第i轮相遇的条件是(k >> (i-1) ^ 1) == (j >> (i-1)) ! “高位相同，i-1位相反”

\*/

//poj 3071

int main()

{

while(scanf("%d", &n), n != -1) {

int top = (1 << n);

for(int i = 0; i < top; i++)

for(int j = 0; j < top; j++) scanf("%lf", &p[i][j]);

for(int j = 0; j < top; j++) f[0][j] = 1;

for(int i = 1; i <= n; i++)

for(int j = 0; j < top; j++) {

f[i][j] = 0;

for(int k = 0; k < top; k++) if(((k >> (i-1)) ^ 1) == (j >> (i-1)) ) { //TODO

f[i][j] += f[i-1][j] \* f[i-1][k] \* p[j][k];

}

}

int index = 0;

for(int j = 0; j < top; j++) if(f[n][j] > f[n][index]) index = j;

printf("%d\n", index+1);

}

return 0;

}

/\*

概率DP 状压DP 条件期望DP 好题

题意：有n种卡片，一袋面里有一张卡片或者没有。给出每种卡片出现在一袋面里的概率，问要收集完n种卡片所需要买的袋面数目的期望(n <= 20).

思路：状压表当前已收集到的卡片的状态，位为1表收集到了，0表没有收集到。

对于状态T，考虑最近买的那袋面之前的状态S（也就是状态X在买了这袋面之后变成了状态T），

<1>这袋面没有卡片

<2>这袋面的卡片X已经收集过了

<3>这袋面的卡片X还没有收集过

方法1:

前两种情况新旧状态不变，即T = S；后面一种情况S = T ^ (1<<k)

则T数学期望E(T) = 1 + (1 - Σp[i])\*E(T) + Σ(p[j]\*E(T)) + Σ(p[k]\*E(S))

其中：i = 0, 1, 2,...n-1

j = 第j种卡片已经收集过了，S & (1<<j) == 1

k = 第k种卡片还没有收集过，S & (1<<j) == 0

移项可得：E(T) \* Σp[k] = 1 + Σ(p[k] \* E(S) ，这里的k=第k种卡片没收集到，S = T ^ (1<<k)

ps:我之前一直很奇怪为什么大家的代码没有对“袋面里可能没有卡片”这一点做处理，后来实在上面的移项过程种想明白的，移项的时候可以把情况<1>和<2>归到一起。我才想明白，其实袋子是否有可能为空并不影响做题，我们可以把空袋子当作一张“隐形卡”，每个状态都默认已经有了这张卡，所以情况<1>可以直接归为情况2，因为不存在空的情况了，隐形卡大家已经收集过了。

===================================================================================================================

方法2:

其实上述式子可以由条件期望的原理很容易地得出...

首先 E(X|Y=y) = Σxf(x|y)

E(E(X|Y)) = EX， 我们叫EX为全体加权平均，E(X|Y)为条件加权平均（局部加权平均），在这道题里X != Y

这样，我们就可以直接得出上述的公式了:

E(X|Y)/Σpy = EX

Σ(E(Y)\*py) / Σpy = EX

其实条件期望我也是晕晕忽忽的，详细看http://blog.csdn.net/henhen2002/article/details/5540039

\*/

//hdu 4336

int main()

{

while(scanf("%d", &n) != EOF) {

for(int i = 0; i < n; i++) scanf("%lf", &p[i]);

f[0] = 0;

int top = (1 << n);

for(int u = 1; u < top; u++) {

double sum = 0, tmp = 0;

sum += 1;

for(int i = 0; i < n; i++) if(u & (1 << i)) {

sum += f[u ^ (1 << i)] \* p[i];

tmp += p[i];

}

f[u] = sum/tmp;

}

printf("%f\n", f[top-1]);

}

return 0;

}

//tc srm420 div 500

double get\_profit(int R, int B) {

for(int i = 0; i <= R; i++) {

cur ^= 1;

for(int j = 0; j <= B; j++)

if(i == 0) e[i][j] = 0;

else if(j == 0) e[i][j] = e[i-1][j] + 1;

else e[i][j] = max((e[i-1][j]+1) \* (i\*1.0/(i+j)) + (e[i][j-1]-1)\*(j\*1.0/(i+j)), 0);

}

}

e(i,j)剩余i张红牌j张黑牌，还能赢钱的期望。

e(0,0) = 0;

(0,j) 状态只会输钱，计算的话期望为负数，所以我们取为0，即停止翻牌。

(i,0) 只会赢钱，所以e(i,0) = e(i-1,0) + 1;

(i,j) = max(0 , (1 + e(i-1,j)) \* i/(i+j) + (1 + e(i,j-1)) \* j/(i+j));

目标状态e(R,B)