ACM/ICPC at Wuhan University

Xioumu STL(文字)

Created by xioumu, for OpenShield, FreeWaiting

目录

Graph

混合图欧拉路径

当一个有向图所有的度为 0 时，这个有向图为欧拉图。
所以混合图的判断欧拉图只要判断原图的无向边往哪个方向即可，所以可以先以任意形态连边，然后有流量的为正向边，无流量的为反向边。
入度 - 出度 >0 的，和源点连一条(入度 - 出度）/ 2 的边，< 0 的， 和汇点连边。
最后如果最大流等于所有与源点连的流量，即存在一种方法使原图变为欧拉图。

生成树的计数（matrix-Tree）

***Matrix-Tree* 定理是解决生成树计数问题最有力的武器之一。它首先于 1847 年被 Kirchhoff 证明。在介绍定理之前，我们首先明确几个概念：**

1. ***G* 的度数矩阵 *D*[*G*]是一个 *n***n* 的矩阵，并且满足：当 *i*≠*j* 时,d_{ij}=0；当 *i*=*j* 时，d_{ij} 等于 v_i 的度数。**
2. ***G* 的邻接矩阵 *A*[*G*]也是一个 *n***n* 的矩阵， 并且满足：如果 v_i、v_j 之间有边直接相连，则 a_{ij}=1，否则为 0。**

我们定义 *G* 的 Kirchhoff 矩阵(也称为拉普拉斯算子)*C*[*G*]为 C[*G*]=*D*[*G*]-*A*[*G*]，则 *Matrix-Tree* 定理可以描述为：*G* 的所有不同的生成树的个数等于其 Kirchhoff 矩阵 *C*[*G*]任何一个 *n*-1 阶主子式的行列式的绝对值。所谓 *n*-1 阶主子式，就是对于 *r*(1≤*r*≤*n*)，将 *C*[*G*]的第 *r* 行、第 *r* 列同时去掉后得到的新矩阵，用 C_r[*G*]表示。

下面我们举一个例子来解释 *Matrix-Tree* 定理。如图 *a* 所示，*G* 是一个由 5 个点组成的无向图。

图 *a*

根据定义，它的 Kirchhoff 矩阵 *C*[*G*]为

我们取 *r*=2，则有：

$|C_2[G]|$

$$= \begin{vmatrix} 2 & -1 & 0 & 0 \\ -1 & 3 & 0 & -1 \\ 0 & 0 & 2 & -1 \\ 0 & -1 & -1 & 2 \end{vmatrix}$$

$$= 2 \cdot \begin{vmatrix} 3 & 0 & -1 \\ 0 & 2 & -1 \\ -1 & -1 & 2 \end{vmatrix} - (-1) \cdot \begin{vmatrix} -1 & 0 & -1 \\ 0 & 2 & -1 \\ 0 & -1 & 2 \end{vmatrix} + 0 \cdot \begin{vmatrix} -1 & 3 & -1 \\ 0 & 0 & -1 \\ 0 & -1 & 2 \end{vmatrix} - 0 \cdot \begin{vmatrix} -1 & 3 & 0 \\ 0 & 0 & 2 \\ 0 & -1 & -1 \end{vmatrix}$$

$$= 2\left(3 \cdot \begin{vmatrix} 2 & -1 \\ -1 & 2 \end{vmatrix} - 0 \cdot \begin{vmatrix} \quad & \quad \\ \quad & \quad \end{vmatrix} + (-1) \cdot \begin{vmatrix} 0 & 2 \\ -1 & -1 \end{vmatrix}\right) - (-1)\left((-1) \cdot \begin{vmatrix} 2 & -1 \\ -1 & 2 \end{vmatrix} - 0 \cdot \begin{vmatrix} \quad & \quad \\ \quad & \quad \end{vmatrix} + (-1) \cdot \begin{vmatrix} 0 & 2 \\ 0 & -1 \end{vmatrix}\right)$$

$$= 2\left(3(4-1) - 0 + (-1)(0-(-2))\right) - (-1)\left((-1)(4-1) - 0 + (-1)(0-0)\right) + 0 - 0$$

$$= 2(9-2) - (-1)(-1)(3) = 11$$

这 11 棵生成树如图 *b* 所示。

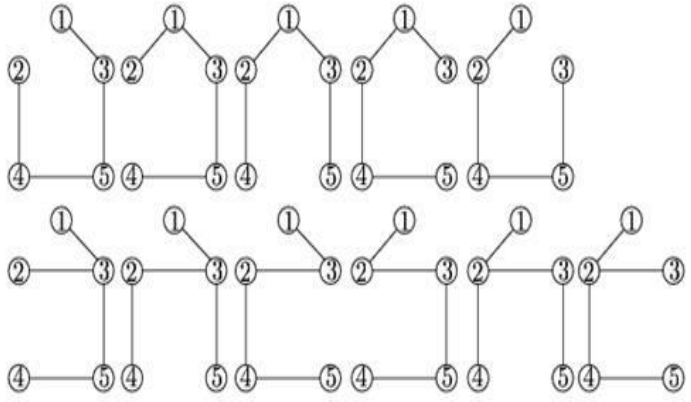


图 *b*

关于中国邮递员问题和欧拉图应用

中国邮递员问题：

1962 年有管梅谷先生提出中国邮递员问题（简称 CPP）。一个邮递员从邮局出发，要走完他所管辖的每一条街道，可重复走一条街道，然后返回邮局。任何选择一条尽可能短的路线。

这个问题可以转化为：给定一个具有非负权的赋权图 G，

（1）用添加重复边的方法求 G 的一个 Euler 赋权母图 G*，使得尽可能小。

（2）求 G*的 Euler 环游。

人们也开始关注另一类似问题，旅行商问题（简称 TSP）。TSP 是点路优化问题，它是 NPC 的。而 CPP 是弧路优化问题，该问题有几种变形，与加权图奇点的最小完全匹配或网络流等价，有多项式算法。[1]

欧拉图：

图 G 中经过每条边一次并且仅一次的回路称作欧拉回路。存在欧拉回路的图称为欧拉图。

无向图欧拉图判定：

无向图 G 为欧拉图，当且仅当 G 为连通图且所有顶点的度为偶数。

有向图欧拉图判定：

有向图 G 为欧拉图，当且仅当 G 的基图[2]连通，且所有顶点的入度等于出度。

欧拉回路性质：

性质 1 设 C 是欧拉图 G 中的一个简单回路，将 C 中的边从图 G 中删去得到一个新的图 G’，则 G’的每一个极大连通子图都有一条欧拉回路。

性质 2 设 C1、C2 是图 G 的两个没有公共边，但有至少一个公共顶点的简单回路，我们可以将它们合并成一个新的简单回路 C’。

欧拉回路算法：

1 在图 G 中任意找一个回路 C；

2 将图 G 中属于回路 C 的边删除；

3 在残留图的各极大连通子图中分别寻找欧拉回路；

4 将各极大连通子图的欧拉回路合并到 C 中得到图 G 的欧拉回路。

由于该算法执行过程中每条边最多访问两次，因此该算法的时间复杂度为 O(|E|)。

如果使用递归形式，得注意|E|的问题。使用非递归形式防止栈溢出。

如果图 是有向图，我们仍然可以使用以上算法。

http://acm.hdu.edu.cn/showproblem.php?pid=1116 有向图欧拉图和半欧拉图判定

http://acm.pku.edu.cn/JudgeOnline/problem?id=2337 输出路径

中国邮递员问题①：

一个邮递员从邮局出发，要走完他所管辖的每一条街道，可重复走一条街道，然后返回邮局。所有街道都是双向通行的，且每条街道都有一个长度值。任何选择一条尽可能短的路线。

分析：

双向连通，即给定无向图 G。

如果 G 不连通，则无解。

如果 G 是欧拉图，则显然欧拉回路就是最优路线。

如果 G 连通，但不是欧拉图，说明图中有奇点[3]。奇点都是成对出现的，证明从略。

对于最简单情况，即 2 个奇点，设（u，v）。我们可以在 G 中对（u，v）求最短路径 R，构造出新图 G’ = G ∪ R。此时 G’就是欧拉图。

证明：u 和 v 加上了一条边，度加一，改变了奇偶性。而 R 中其他点度加二，奇偶性不变。

由此可知，加一次 R，能够减少两个奇点。推广到 k 个奇点的情况，加 k/2 个 R 就能使度全为偶数。

接下的问题是求一个 k 个奇点的配对方案，使得 k/2 个路径总长度最小。

这个就是无向完全图最小权匹配问题。有一种 Edmonds 算法，时间复杂度 O（N^3）。[4]

也可转换为二分图，用松弛优化的 KM 算法，时间复杂度也是 O（N^3）。

完整的算法流程如下：

1 如果 G 是连通图，转 2，否则返回无解并结束；

2 检查 G 中的奇点，构成图 H 的顶点集；

3 求出 G 中每对奇点之间的最短路径长度，作为图 H 对应顶点间的边权；

4 对 H 进行最小权匹配；

5 把最小权匹配里的每一条匹配边代表的路径，加入到图 G 中得到图 G’；

6 在 G’中求欧拉回路，即所求的最优路线。

中国邮递员问题②：

和①相似，只是所有街道都是单向通行的。

分析：

单向连通，即给定有向图 G。

和①的分析一样，我们来讨论如何从 G 转换为欧拉图 G’。

首先计算每个顶点 v 的入度与出度之差 d’（v）。如果 G 中所有的 v 都有 d’（v）=0，那么 G 中已经存在欧拉回路。

d’（v）>0 说明得加上出度。d’（v）<0 说明得加上入度。

而当 d’（v）=0，则不能做任何新增路径的端点。

可以看出这个模型很像网络流模型。

顶点 d’（v）>0 对应于网络流模型中的源点，它发出 d’（v）个单位的流；顶点 d’（v）<0 对应于网络流模型中的汇点，它接收-d’（v）个单位的流；而 d’（v）=0 的顶点，则对应于网络流模型中的中间结点，它接收的流量等于发出的流量。在原问题中还要求增加的路径总长度最小，我们可以给网络中每条边的费用值 设为图 中对应边的长度。这样，在网络中求最小费用最大流，即可使总费用最小。

这样构造网络 N：

1 其顶点集为图 G 的所有顶点，以及附加的超级源 和超级汇 ；

2 对于图 G 中每一条边(u,v)，在 N 中连边(u,v)，容量为∞，费用为该边的长度；

3 从源点 向所有 d’(v)>0 的顶点 v 连边(s,v)，容量为 d’(v)，费用为 0；

4 从所有 d’(v)<0 的顶点 向汇点 t 连边(u,t)，容量为-d’(v)，费用为 0。

完整的算法流程如下：

1 如果 G 的基图连通且所有顶点的入、出度均不为 0，转 2，否则返回无解并结束；

2 计算所有顶点 v 的 d’(v)值；

3 构造网络 N；

4 在网络 N 中求最小费用最大流；

5 对 N 中每一条流量 f(u,v)的边(u,v)，在图 G 中增加 f(u,v)次得到 G’；

6 在 G’中求欧拉回路，即为所求的最优路线。

NPC 问题：

如果部分街道能够双向通行，部分街道只能单向通行。这个问题已被证明是 NPC 的。[5]

差分约束

求解差分约束系统，可以转化成图论的单源最短路径问题。观察 xj-xi<=bk，会发现它类似最短路中的三角不等式 d[v] <=d[u]+w[u,v]，即 d[v]-d[u]<=w[u,v]。因此，以每个变量 xi 为结点，对于约束条件 xj-xi<=bk，连接一条边(i,j)，边权为 bk。再增加一个原点(s,s)与所有定点相连，边权均为 0。对这个图以 s 为原点运行 Bellman-ford 算法（或 SPFA 算法），最终{d[i]}即为一组可行解。

DataStructure

扫描线（面积并）

注意线段树中的每个点要代表一个左闭右开的区间！

Lazy 函数

Lazy[t] = true 表示需要向下传值，但节点 t 的值已经更新完。
也就是说 t 节点的值与 lazy[t]无关。

离散

假如离散线段的成一个点的话，记得两边端点和左边端点左边的点和右边端点右边的点各离散一个点， 一条线段总共离散成 4 个点。否则会出问题。

Computational Geometry

一些要注意的地方

1.判大小等于要用 Sgn()

```
Int sgn(double x) {
        Return (x > esp) - (x < -esp);
}
```

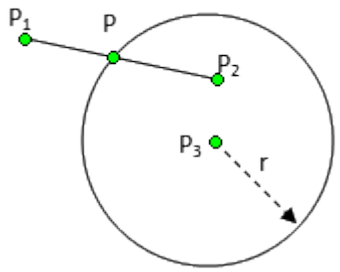
2.Sqrt：y = sqrt(x) x < 0 的时候会出现混乱，所以:

```
if (d < 0)
      res = 0.0;
else
      res = sqrt(d);
```

3. asin(x), acos(x) x ->[-1, 1];

```
Double trim(double d, double l = 1.0) {
        Return d > l ? l : (d < -1 ? -1 : d);
}
```

计算线段和圆的交点



设线段的两个端点分别是 P1(x1, y1)和 P2(x2, y2)，圆的圆心在 P3(x3, y3)，半径为 r，那么如果有交点 P(x, y)的话

$$\vec{P} = \vec{P_1} + u(\vec{P_2} - \vec{P_1})$$

其中，u 在 0 到 1 之间，转换成各个坐标

$$\begin{cases} x = x_1 + u(x_2 - x_1) \\ y = y_1 + u(y_2 - y_1) \end{cases}$$

由于 P 也在圆上，所以

$$(x - x_3)^2 + (y - y_3)^2 = r^2$$

联立上面的公式，可以得到

$$Au^2 + Bu + C = 0$$

其中

$$A = (x_2 - x_1)^2 + (y_2 - y_1)^2$$
$$B = 2[(x_2 - x_1)(x_1 - x_3) + (y_2 - y_1)(y_1 - y_3)]$$
$$C = {x_3}^2 + {y_3}^2 + {x_1}^2 + {y_1}^2 - 2[x_3 x_1 + y_3 y_1] - r^2$$

解一元二次方程，可以得到

$$u = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

根据

$$B^2 - 4AC$$

的结果，可以判断**线段所在直线**和圆的相交情况

- 如果小于 0，表示没有交点
- 如果等于 0，表示相切，只有一个交点
- 如果大于 0，表示有两个交点

针对 P1 和 P2 之间的线段，根据计算出的 u 值，有 5 种结果

- 如果线段和圆没有交点，而且都在圆的外面的话，则 u 的两个解都是小于 0 或者大于 1 的
- 如果线段和圆没有交点，而且都在圆的里面的话，u 的两个解符号相反，一个小于 0，一个大于 1
- 如果线段和圆只有一个交点，则 u 值中有一个是在 0 和 1 之间，另一个不是
- 如果线段和圆有两个交点，则 u 值得两个解都在 0 和 1 之间
- 如果线段和圆相切，则 u 值只有 1 个解，且在 0 和 1 之间

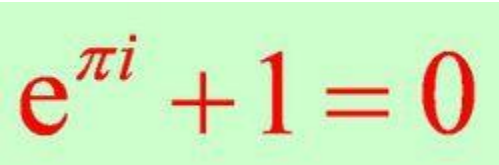
数论

欧拉公式

（欧拉公式）如果一个连通的平面图有 *n* 个点，*m* 条边和 *f* 个面，那么 *f*=*m*-*n*+2

欧拉函数

e^ix=cosx+isinx



自然对数的底 e，圆周率 π，两个单位：虚数单位 i 和自然数的单位 1，以及被称为人类伟大发现之一的 0。数学家们评价它是“上帝创造的公式”

设$p_1^{e_1}, p_2^{e_2}, p_3^{e_3}, p_4^{e_4}, \ldots\ldots, p_k^{e_k}$是$m$的所有互异素因数，那么

$$\phi(m) = m\prod_{i=1}^{k}(1-\frac{1}{p_i})$$

欧拉 φ 函数：φ(n)是所有小于 n 的正整数里，和 n 互素的整数的个数。n 是一个正整数。

欧拉证明了下面这个式子：	

如果 n 的标准素因子分解式是 p1^a1*p2^a2*……*pm^am，其中众 pj(j=1,2,……,m)都是素数，而且两两不等。则有

φ(n)=n(1-1/p1)(1-1/p2)……(1-1/pm)

利用容斥原理可以证明它。

欧拉定论

欧拉定理的内容是：如果 a 和 n 互质，那么 $a^{\phi(n)}=1 \pmod n$；对于任意 a，n 和较大的 b，有 $a^b=a^{\phi(n)+b \bmod \phi(n)} \pmod n$

欧拉公式（面，点，边的关系）

χ=V-E+F 来计算，V-vertices 表示顶点数，E-edges 表示边数，F-faces 表示围成的面数。于是面的个数可以这样计算：对于几个相连的圆，其内有 F=E-V+1 个面，其中 1 是平面图情况的欧拉欧拉示性数(Euler characteristic)；

Pick 公式

定理一：Pick 定理，1899 年

设 Γ 为平面上以格子点为顶点之单纯多边形，则其面积为

$$A = \frac{b}{2} + i - 1 \qquad (8)$$

其中 b 为边界上的格子点数，i 为内部的格子点数。(8)式叫做 Pick 公式。

定理二：（棱线定理，Edge Theorem）

对于任意连通的三角形化的平面图枝，其棱线的个数恒为

$$E = 2b + 3i - 3 \qquad (12)$$

其中 b 与 i 分别表示图枝边界上的顶点数与内部的顶点数。

卡特兰数

卡特兰数 ：1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670, 129644790, 477638700, 1767263190, 6564120420, 24466267020, 91482563640, 343059613650, 1289904147324, 4861946401452,

令 h(1)=1,h(0)=1，catalan 数满足递归式：

h(n)= h(0)*h(n-1)+h(1)*h(n-2) + ... + h(n-1)h(0) (其中 n>=2)

另类递归式：

h(n)=h(n-1)*(4*n-2)/(n+1);

该递推关系的解为：

h(n)=C(2n,n)/(n+1) (n=1,2,3,...)

二维旋转

看下面的矩阵

$$\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

这个是二维向量的旋转矩阵，它可以将一个向量逆时针旋转一个角度。

将其变形，变会得到二维向量的顺时针旋转的形式：

$$\begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$

这里要注意一种特殊的情况，就是当角度为 90 度的时候，sin 和 cos 的结果只有 1, 0, -1 三种可能性。所以这个矩阵可以改写成特殊的形式，其意义在于用这样的旋转操作，不会产生精度问题。

sin 和 cos 的运算的精度是比较低的，能少用则尽量少用。在计算几何中的向量旋转操作，大部分都可以通过变形，只用到旋转 90 度，从而避免精度问题。

===3 维====

围绕轴 u = (ux, uy, uz)来旋转的矩阵代码如下：

```
double a[3][3] = {
    {SQR(ux) + (1 – SQR(ux)) * c, ux * uy * (1 – c) – uz * s, ux * uz * (1 – c) + uy * s},
    {ux * uy * (1 – c) + uz * s, SQR(uy) + (1 – SQR(uy)) * c, uy * uz * (1 – c) – ux * s},
    {ux * uz * (1 – c) – uy * s, uy * uz * (1 – c) + ux * s, SQR(uz) + (1 – SQR(uz)) * c}
  };
```

要注意这个 ux * ux + uy * uy + uz * uz = 1

有一个比较有意思的问题就是，知道旋转矩阵之后，怎么来确定向量 u 呢？

我们做如下变形之后，可以得到：

Ru = Iu –> (R – I)u = 0

也就是说我们要找到一个非 0 的向量，使得他和一个矩阵乘起来得到一个空矩阵。这个问题可以用高斯消元来解决。实际上我们就是要解一个线性方程组。

逆元

定义 如果 $ab \equiv 1 \pmod m$，则称 b 是 a 的模 m 逆,记作 a 的模 m 逆是方程 $ax \equiv 1 \pmod m$的解.

即 gcd(a, m) == 1，即(a, m)互质

例：求 5 的模 7 逆

做辗转相除法，求得整数 b,k 使得 $5b+7k=1$，则 b 是 5 的模 7 逆.

计算如下:

$7=5+2,\quad 5=2\times 2+1.$

回代 $1=5-2\times 2=5-2\times(7-5)=3\times 5-2\times 7,$

得 $5^{-1}\equiv 3 \pmod 7$.

例：求 21 的模 73 逆

做辗转相除法，求得整数 b,k 使得 $21b+73k=1$，则 b 是 21 的模 73 逆.

计算如下:

$73=21*3+10$

21=10*2+1

回代 **1=21-10*2**

1=21-（73-21*3）*2

=21-73*2+6*21

=7*21-73*2

得 **21 $^{-1}$≡7(mod73).**

求和公式

1^4+2^4+3^4+4^4+……+n^4 = n(n+1)(2n+1)(3n^2+3n-1)/30

1^3+2^3+3^3+4^3+……+n^3=[n(n+1)/2]^2

1^2+2^2+3^2+4^4+……+n^2=n(n+1)(2n+1)/6

1+2+3+4+……+n=n(n+1)/2

三维旋转

平移：

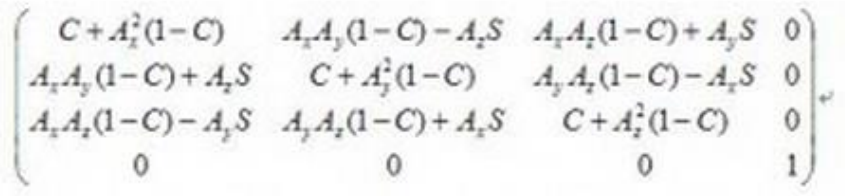
$$\begin{matrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ tx & ty & tz & 1 \end{matrix}$$

拉伸：

$$\begin{matrix} a & & & \\ & b & & \\ & & c & \\ & & & 1 \end{matrix}$$

C = cos(angle), S = sin(angle).
绕(0, 0, 0) - (X, Y, Z) 向量顺时针旋转 angle (即从(x,y,z)向(0,0,0)点看,顺时针旋转)

旋转：



$$\begin{pmatrix} C+A_x^2(1-C) & A_xA_y(1-C)-A_zS & A_xA_z(1-C)+A_yS & 0 \\ A_xA_y(1-C)+A_zS & C+A_y^2(1-C) & A_yA_z(1-C)-A_xS & 0 \\ A_xA_z(1-C)-A_yS & A_yA_z(1-C)+A_xS & C+A_z^2(1-C) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

```
matrix get_rotate(double x, double y, double z, double d) {
	matrix now;
	now.set_one();
	d = -d / 180.0 * pi;
	double c = cos(d), s = sin(d);
	double l = sqrt(x * x + y * y + z * z);
	x /= l, y /= l, z /= l;
	now.ar[0][0] = c + x * x * (1 - c);
	now.ar[0][1] = x * y * (1 - c) - z * s;
	now.ar[0][2] = x * z * (1 - c) + y * s;

	now.ar[1][0] = x * y * (1 - c) + z * s;
	now.ar[1][1] = c + y * y * (1 - c);
	now.ar[1][2] = y * z * (1 - c) - x * s;

	now.ar[2][0] = x * z * (1 - c) - y * s;
	now.ar[2][1] = y * z * (1 - c) + x * s;
	now.ar[2][2] = c + z * z * (1 - c);
	now.ar[3][3] = 1;
	return now;
}
```

Pick 公式

定理一：Pick 定理，1899 年

设「为平面上以格子点为顶点之单纯多边形，则其面积为

$$A = \frac{b}{2} + i - 1 \qquad (8)$$

其中 b 为边界上的格子点数，i 为内部的格子点数。(8)式叫做 Pick 公式。

定理二：（棱线定理，Edge Theorem）

对于任意连通的三角形化的平面图枝，其棱线的个数恒为

$$E = 2b + 3i - 3 \qquad (12)$$

其中 b 与 i 分别表示图枝边界上的顶点数与内部的顶点数。

MOD 公式

A/B %M = A % (B * M) / B

一些公式

(abs(a + b) + abs(a - b)) / 2 = max(abs(a) , abs(b))

Pólya 计数

$$P = \frac{1}{|G|}\sum_{g\in G}\prod_{i=1}^{n} m^{\lambda_i(g)}$$

证明： 由Cauchy定理$P=\frac{1}{|G|}\sum_{g\in G} fix(g)$其中$fix(g)=\{\varphi|\ \varphi g=\varphi\}$这表明$\forall x\in V$有$\varphi g(x)=\varphi(x)$

所以$\varphi g^i(x)=\varphi(x)$（由$\varphi g^i(x)=\varphi g^{i-1}(x)...=\varphi(x)$知），即在<g>作用下相同的轨道有相同的赋值，而易知<g>作用下的轨道总数为：$\lambda_1+\lambda_2...+\lambda_n$(这是因为每个$x$只处于一个轮换中，且每个轮换构成一个轨道)。又每个轨道有m种取法，故$fix(g)=m^{\lambda_1}m^{\lambda_2}...m^{\lambda_n}$

即 $P=\frac{1}{|G|}\sum_{g\in G}\prod_{i=1}^{n}m^{\lambda_i(g)}$

DP

概率 DP

要倒着 DP，有环可用高斯消元，或者设 X 直接把 X 给化简到一边。

数位 DP

两种:

1.先预处理出来一些信息（如：f(10),f(100),f(1000)的值，一般用递推预处理），然后直接递归算。

2.只需预付处理少量信息，然后直接记忆化搜索。

优点:只需要想清楚各种转移情况即可，思路较清晰

缺点:情况多的话，较容易想乱。