矩阵及其应用

雍高鹏

矩阵乘法及其优化

```
///矩阵乘法的优化
Matrix f_mat_mul(Matrix a,Matrix b){
    Matrix ans;
    memset(ans.a,0,sizeof(ans.a));
    for(int i=0;i<n;i++){
        for(int j=0;j<n;j++){
            if(a.a[i][j]){
                for(int k=0;k<n;k++){
                    ans.a[i][k]+=a.a[i][j]*b.a[j][k];
                }
            }
        }
    }
    return ans;
}
```

快速幂

对于整形数字 b=a^k%m ， 当 k 很大的时候（ 1e9 ）， 可以用二分的思想快速求出 b 的值。

```
LL fastmod(LL a,LL k,LL m){
    LL b=1;
    while(k){
        if(k&1) b=a*b%m;
        a=(a%m)*(a%m)%m;
        k=k>>1;
    }
    return b;
}
```

那么，类似的，对于矩阵 s ， s^k%mod 也可以快速求出。

复杂度分析，在算法竞赛中，矩阵的规模不会太大，乘法的 **O(N^3)** 当做常数处理。

```
Matrix fastmod(struct Matrix s,int k){
    Matrix ans;
    ans.init();
    while(k){
        if(k&1) ans=mat_mul(ans,s);
        s=mat_mul(s,s);
        k=k>>1;
    }
    return ans;
}
```

快速幂

S^k 可以用二分求解，如果是 **S^1+S^2+….+S^k** 呢？答案就是，二分，再二分。

```
Matrix dfs(struct Matrix s,int k){
    if(k==1) return s;
    Matrix ans;
    ans.init();
    ans=mat_plu(ans,fastmod(s,k>>1));
    ans=mat_mul(ans,dfs(s,k>>1));
    if(k&1) ans=mat_plu(ans,fastmod(s,k));
    return ans;
}
```

上面 mat_plu() 是矩阵相加的函数，对应位置相加。
以 **k=16** 为例：
s^1+…s^16=(s^1+…+s^8)+s^8(s^1+…+s^8)
s^1+…+s^8=(s^1+…+s^4)+s^4(s^1+…+s^4)
…
规模依次折半。

关于构造

给定 **n** 个点，对所有点同时进行 **m** 个操作（操作包括平移、缩放、翻转和旋转），试构造 **O(m+n)** 的算法输出 **m** 个操作后各点的位置。

其中翻转是以坐标轴为对称轴进行翻转（两种情况），旋转则以原点为中心。

$$\begin{pmatrix} 1 & 0 & p \\ 0 & 1 & q \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x+p \\ y+q \\ 1 \end{pmatrix}$$

平移

$$\begin{pmatrix} L & 0 & 0 \\ 0 & L & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x*L \\ y*L \\ 1 \end{pmatrix}$$

缩放

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ -y \\ 1 \end{pmatrix}$$

上下翻转

$$\begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} -x \\ y \\ 1 \end{pmatrix}$$

左右翻转

$$\begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\alpha\, x - \sin\alpha\, y \\ \sin\alpha\, x + \cos\alpha\, y \\ 1 \end{pmatrix}$$

绕原点旋转

关于构造

求第 **n** 个 **Fibonacci** 数 **mod p** 的值。

Fibonacci 数的递推关系， **f(n)=f(n-1)+f(n-2)** 。我们就要试图找到一个矩阵 **s** ， 使得 **s** 与 **(a,b)** （ **a** 、 **b** 为两个连续 **Fibonacci** 数）相乘后， 会得到 **(b,a+b)** 这种形式。因此矩阵可以构造为 $\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} b \\ a+b \end{pmatrix}$

更一般的： 找到 **f(n) = 4f(n-1) - 3f(n-2) + 2f(n-4)** 的第 **k** 项

其对应矩阵的构造方法为： 在右上角的 **(n-1)*(n-1)** 的小矩阵中的主对角线上填 **1** ， 矩阵第 **n** 行填对应的系数， 其它地方都填 **0** 。

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 2 & 0 & -3 & 4 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} b \\ c \\ d \\ 2a-3c+4d \end{pmatrix}$$

图的邻接矩阵的应用

一张 N 个点的无向无权图。如果点 a 和点 b 之间有边，那么邻接矩阵 G[a][b] = G[b][a] = 1， 否则等于 0 。

那么，考虑邻接矩阵自乘，即 G^2， 会有什么意义。

若 **G^2[a][b]** ！ **=0**， 就说明存在点 **i**， 使得 **a i** 间有边并且 **i b** 间也有边。 **G^2[a][b]** 的值，即为从点 **a** 到点 **b** 经过 **1** 个中间点的路径条数。

对于 **G^3,** 即 **G[a][i]G[i][j]G[j][b] = 1** 当且仅当 **G[a][i] = G[i][j] = G[j][b] =1**， 也就是说 **a<->i<->j<->b** 组成一条路经。那么 **G^3[a][b]** 的值就是从点 **a** 到点 **b** 经过 **2** 个中间点的路径条数。

因此，运用数学归纳法不难证明， **G^k[a][b]** 就等于 **a** 到 **b** 经过 **k-1** 个中间点也就是长度为 **k** 的路径条数。

图的邻接矩阵的应用

一些推广及处理：

1 、前面的结论同样适用于有向图。

2 、对于有重边的图，只需要把 G[a][b] 改为表示点 a, b 之间重边的数目 。

3 、对于有向图的自环，直接加边即可 。

4 、对于无向图的自环，通常是把 G[a][a] 加上 2 ，这样在计算 G^k 的时候这条边就被算成 2 条不同的边，如果只加上 1 ，又会不满足矩阵里所有元素和等于边数两倍的性质 (入度和出度) 。

5 、忽略前面所说的，具体问题具体分析。

图的邻接矩阵的应用

给定一张 **M** 条边的无向带权图，求从起点 **S** 到终点 **E** 恰好经过 **K** 条边的最短路径。 **2<=M<=100,2<=K<=1000000** 。

思考：考虑无向无权图，经过 **k** 条边是通过 **k** 次矩阵的相乘得到的，结果得到的是路径数。因为矩阵相乘的实质是固定两端、枚举中间节点，计算结果全部加和。 **g[i] [j] = ∑ g[i] [k]*G[k][j]**

现在要求的是最短路． **f[i] [j] =min(f[i][k] + G[k][j])**

从方程的比对中，可以发现我们只需重定义矩阵乘法。每经过一条路做一次 **floyd** 即可。

优化 DP

有 K 种珍珠，每种 N 颗，求这些珍珠组成的长度在 1~N 之间包含 K 种珍珠的项链有多少种（答案模 1234567891）。

其中，1<=K<=30,1<=N<=1000,000,000

状态转移方程： dp[i][j] = j*dp[i-1][j]+(k-j+1)*dp[i-1][j-1]. 其中 dp[i][j] 表示包含 j 种珍珠且长度为 i 的项链的种数。 ans= dp[1][k]+…+dp[n][k]

直接开 dp[][] 数组会 MLE，用滚动数组 O(nk) 递推会 TLE.

注意到，dp[i][] 的状态只与 dp[i-1][] 相关，那么从状态 dp[i-1][] 转移到状态 dp[i][]，相当于一个 k*k 的线性变化。即 F[i]=A*F[i-1]，这里 A 是转移矩阵，即 $F[i]=A^{i-1}*F[1]$，所以 $ans=F[1]+\ldots+F[n]=A^{0}*F[1]+\ldots+A^{n-1}*F[1]=(E+A+A^{2}+\ldots+A^{n-1})*F[1]$。（注意这里有矩阵，自己构造自己想）

$$
\begin{matrix}
0 & k & 0 & 0 & \ldots\ldots & 0 & 0 & 0 \\
0 & 1 & k-1 & 0 & \ldots\ldots & 0 & 0 & 0 \\
0 & 0 & 2 & 0 & \ldots\ldots & 0 & 0 & 0 \\
\ldots\ldots & & & & & & & \\
0 & 0 & 0 & 0 & \ldots\ldots & 0 & 0 & k
\end{matrix}
$$

优化 DP

使用矩阵优化的条件:

1 、状态必须是一维或者两维,如果状态本身有超过两维要状态压缩。

2 、每一个状态 dp[i] [] 必须满足只和 dp[i-1] [] 有关,并且只能是线性关系

3 、转移矩阵都相同或者至少是循环出现才能通过快速幂加速。

4 、矩阵规模较小,转移次数较大的时候才运用,否则很可能增加复杂度

题目

POJ 3233
POJ 3070
POJ 3735
HDU 3306
HDU 1757
HUD 2294
ZOJ 2974
POJ 3613
POJ 3420

参考资料

2008 国家集训队论文 day1 《矩阵乘法在信息学中的应用》

Matrix67 的博客

离散数学 可达性矩阵