题解

T2：排列计数

P15

枚举全排列

P30

DP记录已经决策了前几个位置和那些点。

P70

考虑动态规划

定义状态$dp[i][j]$表示已经决策了前i个数有j个数是稳定的。然后决策第$i+1$位。

如果第$i+1$位就放$i+1$这个数，那么就是$dp[i+1][j+1]+=dp[i][j]$

如果第$i+1$位和前面一个不稳定的位置的数交换，就是$dp[i+1][j]+=dp[i][j]*(i-j)$

如果第$i+1$位和前面一个稳定的位置的数交换，就是$dp[i+1][j-1]+=dp[i][j]*j$

正解

对于1~n的全排列，其中m个位置是稳定的的方案数为C_n^m,然后剩下$n-m$个数都不在自己对应的位置上的方案数就是$n-m$个数错排的方案数。两者相乘就是答案。

T5：赌约

P30

dfs枚举每一关的状态来计算答案。复杂度$O(2^n)$

P60

按照期望DP的套路，我们倒序（按照关卡）决策。定义状态$dp[i][j][k]$表示已经决策完`i+1~n`个关卡的状态，当前如果通关可以获得的分数为j,当前剩余血量为k。那么转移方程就是

$$dp[i][j][k]=(dp[i+1][min(j+1,R)][min(k+1,Q)]+j)*p+(dp[i+1][1][k-1])*(1-p)$$

终止状态就是i为n+1或是k为0。

（但貌似有点后面数据修改后导致这里有点卡常，我的锅）

P70

可以看出这个转移方程可以矩乘，但是这里有三维的状态，我们可以把后面两维压成一维。这样就可以用矩阵乘法优化了。可是这里还有一个问题，就是那个通关时获得的k怎么加。

假设我们的答案是A*Mul(B,n)（A,B都是矩阵）

85分就是把这些k都放在A数组的后面。

初始矩阵就变成了

> 0 0 0 0 0 1 2 3 4 5

注意B数组要保证让这后R个数不会变化。

但这样会增加矩阵大小从而增加复杂度。只能得到70分。

P85

由于我们观察到矩阵内的数中0的个数很多，于是我们一乘到0就可以continue。就像这样

```
Matrix operator *(const Matrix &_)const{
        Matrix res;
        res.resize(n,_.m);
        res.clear();
        for(int k=1;k<=m;k++){//枚举顺序改一下
            for(int i=1;i<=res.n;i++){
                if(num[i][k]<eps)continue;//**
                for(int j=1;j<=res.m;j++){
                    if(_.num[k][j]<eps)continue;//**
                    res.num[i][j]+=num[i][k]*_.num[k][j];
                }
            }
        }
        return res;
    }
```

P100

P70的方法是将矩阵的边长增加了R。但是实际上只用增加1就可以了。我们增加一个恒为1的数。（也可以把他当成一个val恒为1的状态）然后把他当状态一样转移就行了。

附上网上题解的一张图

> 看这两个矩阵 （初始矩阵） 0 0 0 0 0 1 （转移矩阵） 0 0 0 0 0 0 //考虑把不同状态之间的概率转移写进左上角5*5的矩阵里。 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 3 4 5 1

复杂度$O((R\cdot Q)^3\cdot log^2(n))$

还有一个优化，就是有很多状态其实是不合法的。（像你连过的关数已经大于Q但是你的血量却还没满）删去这些状态在进行矩乘。

```
#include<cstdio>
#define P 998244353
#include<algorithm>
#include<cstring>
#define M 10005
#define eps 1e-15
using namespace std;
int n,lim,m,p;
struct Matrix{
    int n,m;
    long long num[180][180];
    //Matrix(){memset(num,0,sizeof(num));}
    void resize(int n,int m){this->n=n;this->m=m;}
    void clear(){for(int i=0;i<=n;i++)for(int j=0;j<=m;j++)num[i][j]=0;}
    void Init(){for(int i=1;i<=n;i++)for(int j=1;j<=m;j++)num[i][j]=i==j;}
    Matrix operator *(const Matrix &_)const{
        Matrix res;
        res.resize(n,_.m);
        res.clear();
```

```
        for(int k=1;k<=m;k++){
            for(int i=1;i<=res.n;i++){
                if(num[i][k]==0)continue;
                for(int j=1;j<=res.m;j++){
                    if(_.num[k][j]==0)continue;
                    res.num[i][j]=(res.num[i][j]+1ll*num[i][k]*_.num[k][j]%P)%P;
                }
            }
        }
        return res;
    }
    void Print(){
        for(int i=1;i<=n;i++){
            for(int j=1;j<=m;j++)printf("%lld ",num[i][j]);
            puts("");
        }
    }
};
int ID[45][45];
int sz;
long long Get_ans(){
    Matrix A,B;
    A.resize(1,sz);B.resize(sz,sz);
    A.clear();B.clear();
    A.num[1][sz]=1;B.num[sz][sz]=1;
    for(int res=1;res<=m;res++){
        for(int K=1;K<=lim;K++){
            B.num[ID[min(res+1,m)][min(K+1,lim)]][ID[res][K]]=p;
            B.num[sz][ID[res][K]]=1ll*p*K%P;
            if(res>1)B.num[ID[res-1][1]][ID[res][K]]=(1-p+P)%P;
        }
    }
    Matrix res;res.resize(B.n,B.m);res.Init();
    int b=n;
    while(b){
        if(b&1)res=res*B;
        B=B*B;
        b>>=1;
    }
    A=A*res;
    return (A.num[1][ID[m][1]]+P)%P;
}
void Solve(){
    sz=0;
    for(int res=1;res<=m;res++)for(int K=1;K<=lim;K++)ID[res][K]=++sz;
    sz++;
    printf("%lld\n",Get_ans());
}
int main(){
    int T;
    scanf("%d",&T);
    while(T--){
        scanf("%d%d%d%d",&n,&lim,&m,&p);
        Solve();
    }
    return 0;
}
```