数位dp+状压dp

数位DP模板

```
typedef long long ll;
int a[20];
ll dp[20][state];//不同题目状态不同
ll dfs(int pos,/*state变量*/,bool lead/*前导零*/,bool limit/*数位上界变量*/)//不是每个题都要判断前导零
{
    //递归边界，既然是按位枚举，最低位是0，那么pos==-1说明这个数我枚举完了
    if(pos==-1) return 1;/*这里一般返回1，表示你枚举的这个数是合法的，那么这里就需要你在枚举时必须每一位都要满足题目条件，也就是说当前枚举到pos位，一定要保证前面已经枚举的数位是合法的。不过具体题目不同或者写法不同的话不一定要返回1 */
    //第二个就是记忆化(在此前可能不同题目还能有一些剪枝)
    if(!limit && !lead && dp[pos][state]!=-1) return dp[pos][state];
    /*常规写法都是在没有限制的条件记忆化，这里与下面记录状态是对应，具体为什么是有条件的记忆化后面会讲*/
    int up=limit?a[pos]:9;//根据limit判断枚举的上界up;这个的例子前面用213讲过了
    ll ans=0;
    //开始计数
    for(int i=0;i<=up;i++)//枚举，然后把不同情况的个数加到ans就可以了
    {
        if() ...
        else if()...
        ans+=dfs(pos-1,/*状态转移*/,lead && i==0,limit && i==a[pos]) //最后两个变量传参都是这样写的
        /*这里还算比较灵活，不过做几个题就觉得这里也是套路了
        大概就是说，我当前数位枚举的数是i，然后根据题目的约束条件分类讨论
        去计算不同情况下的个数，还有要根据state变量来保证i的合法性，比如题目
        要求数位上不能有62连续出现,那么就是state就是要保存前一位pre,然后分类，
        前一位如果是6那么这意味就不能是2，这里一定要保存枚举的这个数是合法*/
    }
    //计算完，记录状态
    if(!limit && !lead) dp[pos][state]=ans;
    /*这里对应上面的记忆化，在一定条件下时记录，保证一致性，当然如果约束条件不需要考虑lead，这里就是lead就完全不用考虑了*/
    return ans;
}
ll solve(ll x)
{
    int pos=0;
    while(x)//把数位都分解出来
    {
        a[pos++]=x%10;//个人老是喜欢编号为[0,pos),看不惯的就按自己习惯来，反正注意数位边界就行
        x/=10;
    }
    return dfs(pos-1/*从最高位开始枚举*/,/*一系列状态 */,true,true);//刚开始最高位都是有限制并且有前导零的，显然比最高位还要高的一位视为0嘛
}
int main()
```

```
{
    ll le,ri;
    while(~scanf("%lld%lld",&le,&ri))
    {
        //初始化dp数组为-1,这里还有更加优美的优化,后面讲
        printf("%lld\n",solve(ri)-solve(le-1));
    }
}
```

A-不要62

杭州人称那些傻乎乎粘嗒嗒的人为62（音：laoer）。

杭州交通管理局经常会扩充一些的士车牌照，新近出来一个好消息，以后上牌照，不再含有不吉利的数字了，这样一来，就可以消除个别的士司机和乘客的心理障碍，更安全地服务大众。

不吉利的数字为所有含有4或62的号码。例如：

62315 73418 88914

都属于不吉利号码。但是，61152虽然含有6和2，但不是62连号，所以不属于不吉利数字之列。

你的任务是，对于每次给出的一个牌照区间号，推断出交管局今次又要实际上给多少辆新的士车上牌照了。

输入

输入的都是整数对n、m（0<n≤m<1000000），如果遇到都是0的整数对，则输入结束。

输出

对于每个整数对，输出一个不含有不吉利数字的统计个数，该数值占一行位置。

样例输入

1 100

0 0

样例输出

80

```
/*解题思路:
就是数位上不能有4也不能有连续的62，没有4的话在枚举的时候判断一下，不枚举4就可以保证状态合法了，所以这个约束没有记忆化的必要，而对于62的话，涉及到两位，当前一位是6或者不是6这两种不同情况我计数是不相同的，所以要用状态来记录不同的方案数。
dp[pos][state]表示当前第pos位，前一位是否是6的状态，这里state只需要去0和1两种状态就可以了，不是6的情况可视为同种，不会影响计数。*/


LL dp[25][25],a[25];

LL dfs(LL pos,LL state,LL limit){//pos表示位数，state表示前一位是否为6，limit表示是否有数位上界
    if(pos==0) return 1;//到第0位，结束，返回1
```

```
    if(!limit&&dp[pos][state]!=-1) return dp[pos][state];//没有数位上界，同时改状态访问过
    LL up=limit?a[pos]:9;//该位数字最高能到达的位数
    LL ans=0;//记录该位数量
    for(int i=0;i<=up;i++){
        if(state&&i==2) continue;//前位是6，该位是2则continue
        if(i==4)    continue;//该位是4 则continue
        ans+=dfs(pos-1,i==6,limit&&i==up);//寻找下一位，判断该位是否为6，是否仍维持上界
    }
    if(!limit) dp[pos][state]=ans;//如果该位的前位没有限制，则表示通用可以记录该状态下的数量
    return ans;//返回数量
}

LL solve(LL num){//预处理出每位数字，辅助确定上界
    memset(dp,-1,sizeof(dp));
    LL top=0;
    while(num){
        a[++top]=num%10;
        num/=10;
    }
    return dfs(top,0,1);//高位向低位寻找
}
```

Mondriaan's Dream （POJ2411）

```
#include<iostream>
#include<stdio.h>
using namespace std;
#define FAST_IO std::ios::sync_with_stdio(false), std::cin.tie(0), std::cout.tie(0)
#define ll long long
const ll INF = 0x3f3f3f3f3f3f3f3f;
int n,m;
ll f[12][1<<12];//f[i][j]表示第i行，j状态（转为二进制状态）
bool in_s[1<<12];//判断该状态是否可行。二进制第k位为1表示是一个竖着的长方形上面一部分，0表示其他情况。则不能有连续奇数个0;

int main()
{
    FAST_IO;
    while(cin>>n>>m&&n){//n*m的棋盘
        for(int i=0;i<(1<<m);i++){
            bool cnt=0,has_odd=0;//cnt辅助记录连续0的奇偶，has_odd记录改二进制串是否出现奇长度的0
            for(int j=0;j<m;j++){
                if(i>>j&1)has_odd|=cnt,cnt=0;
                else cnt^=1;
                in_s[i]=has_odd|cnt?0:1;//均是偶长度的连续0，为可行的二进制状态，设为1
            }
        }
            f[0][0]=1;
            for(int i=1;i<=n;i++){
                for(int j=0;j<(1<<m);j++){//j表示第i行状态
                    f[i][j]=0;
```

```
                    for(int k=0;k<(1<<m);k++){//k表示第i-1行状态
                        if((j&k)==0&&in_s[j|k])//(j&k)==0表示上下两行没有同为1的位置，in_s[j|k]表示上下状态结合没有连续0为奇数的状态
                            f[i][j]+=f[i-1][k];//该状态加上上一个状态的个数
                    }

                }
            }
        cout<<f[n][0]<<endl;//最后一行不能向下延申，则每一位都是平放着的都是0，输出f[n][0]
    }
    return 0;
}
```