T1 矩形染色

图形是对称的，于是考虑该问题的 $\frac{1}{4}$ ，即只考虑第一象限，然后把把面积 $*4$ 。

首先将矩形按照 x 坐标排序，把整个图形根据 x 可以分为 n 份，发现每一份的高 y 即为后缀最大值。

考察知识点

模拟

T2 五彩斑斓

20%

$1 \leq n, m \leq 80$，直接 $\mathcal{O}(n^4)$ 枚举子矩形即可

40%

对于另外 20% 的数据，$c_{i,j} = 0$ 或 1，合法的状态很少，分别扫描统计即可。

20%~80%

$\mathcal{O}(n^4)$ 枚举子矩形时，先判断两个点已经异色的矩形，如果同色才进行后续的枚举。

80%

$1 \leq c_{i,j} \leq 400$

考虑反向统计，子矩阵一共有 $\frac{n(n+1)}{2} * \frac{m(m+1)}{2}$ 个，减掉四个角都相同的即可。

$\mathcal{O}(n^2)$ 枚举矩形上下边界为第 i 行和第 j 行，$\mathcal{O}(n)$ 扫描每一列。

假设当前第 k 列，若 $c_{i,k} \neq c_{j,k}$，那么第 k 列必定不能作为左右边界。

否则假设第 k 列是**右边界**，那么左边界必须也是同样的字母，并且上下两行对应相同。

因此我们维护一个对颜色的计数器，每次扫描到 $c_{i,k} = c_{j,k}$ 时，操作：

```
++cnt[c[i][k]];
ans += cnt[c[i][k]];
```

即可求出答案，复杂度是 $\mathcal{O}(n^3 + n^2c)$ 的，每次都要清空一下计数器数组，颜色太多清空计数器复杂度太大。

100%

对于全部数据，$1 \leq n, m \leq 400, 1 \leq c_{i,j} \leq 10^6$

颜色太多清空计数器复杂度太大怎么办？只需要清空这两行涉及到的颜色即可，复杂度还是 $\mathcal{O}(n^3)$ 的 。

考察知识点

枚举，容斥

T3 SOS字符串

20%

$n \leq 12$，可以本地打表或者手算打表。

仔细想想可以发现只有三类字符：S，O 与其余 24 种字符。依据三种字符枚举即可$O(3^n)$。

40%

考虑动态规划，$dp[i][j][k]$表示前 $1 \sim i$ 个字符已经出现了 j 个 SOS，现在匹配到 SOS 的第 k（$k \in 0,1,2$）个字符，从 $dp[i-1][j][0/1]$（非$'S'$和$'O'$结尾,$'S'$结尾)和$dp[i-1][j-1][2]$($'SO'$结尾)转移即可

时间复杂度$O(n^2)$

60-100%

发现先前的动态规划中我们并不在乎$j >= 2$的结果，这些转移都是非常简单的，因此我们不需要转移那么多，时间复杂度$O(n)$，根据写法的常数大小有概率 TLE 一些点。

100%

可以简化之前的转移，根据题意可知，本题其实关键可以是构成了几个“SOS”，以及构成到了第几个字母，因此可以长度为i的字符串构成到了第几个字母主要划分的阶段。

我们可以建立第几个字母对应的字符串类型

```
// 0: ''
// 1: '*S'
// 2: '*SO'
// 3: '*SOS*'
// 4: '*SOS*S'
// 5: '*SOS*SO'
// 6: '*SOS*SOS*'
// 7: '*SOS*SOS*S'
// 8: '*SOS*SOS*SO'
// 9: '*SOS*SOS*SOS*'
```

由此我们可以依据每次添加一个字符来实现 $0-9$ 之间状态的转移，可以发现只有三类字符，S，O 与其余 24 种字符，依据三种字符与这10 种状态的关系转移即可。

时间复杂度$O(n)$，空间复杂度$O(n)$

```
    dp[0][0]=1;
    for(int i=1;i<=n;++i){
        for(int j=0;j<10;++j){
            int nj;
            //填S
            nj=(j%3!=1)? j+1:j/3*3+1;
```

```
            nj=min(nj,9);
            dp[i][nj]=dp[i][nj]+dp[i-1][j];
            //填o
            nj=(j%3==1)? j+1:j/3*3;
            nj=min(nj,9);
            dp[i][nj]=dp[i][nj]+dp[i-1][j];
            //填其它
            nj=j/3*3;
            dp[i][nj]=dp[i][nj]+dp[i-1][j]*mint(24);
        }
    }
    cout<<dp[n][9]<<endl;
```

更优秀的做法?

> 可以发现每次是$dp[i][0-9]$和$dp[i-1][0-9]$之间的线性转移，因此可以构建矩阵刻画两者间的转移，时间复杂度$O(9^3 \log n)$

考察知识点

动态规划，计数问题

T4 一个真实的故事

20%

修改可以直接改，对于每一次询问$O(n^2)$枚举左端点判断最早合法的右端点，枚举区间$[i,j]$可以在$[i,j-1]$基础上判断是否合法求出答案，复杂度$O(mn^2)$

50%

发现对于每一次询问$O(n)$双指针可以求出答案，复杂度$O(mn)$

70%

发现只有三个数字，可以尝试记录左右两侧第一个 1，2，3 的位置

发现这样修改太多了，尝试分块均衡，记录每一个块内 123 ，最左和最右的位置，每次扫块内和块间查询和修改即可，每次修改完维护全局的答案，只有块内和块间的答案更新了，更新会控制在 $O(\min(B,\frac{N}{B}))=O(\sqrt{n})$。

复杂度$O(km\sqrt{n})$

100%

使用线段树解决，主要考察线段树上节点的 Msg 的合并操作。

可以求一个区间的答案可以考虑有这个区间的两个子区间的答案拼凑出来：

> - 拿左区间的答案
> - 拿右区间的答案
> - 拿左区间拥有的数字中靠右些的，右区间拿靠左的。

考虑线段树节点的 Msg 需要维护什么信息?

> - 区间的答案，即这个线段树节点所代表的区间包含 $1\sim k$ 的最短区间，不存在则为 ∞。
> - 这个线段树节点所代表的区间最左边的 $1\sim k$ 分别在哪个位置。
> - 这个线段树节点所代表的区间最右边的 $1\sim k$ 分别在哪个位置。

如何合并?

> 先取出左区间最右的 $1\sim k$ ，取出右区间最左的 $1\sim k$ 。
>
> 合并起来按位置编号归并（也可以直接 $sort$ ，会多一个 log ）。
>
> 然后 $O(k)$ 双指针即可求出合并后的答案。
>
> 同时我们可以很方便的维护出合并后该区间的最左的 $1\sim k$ 和最右的 $1\sim k$ 。

然后会发现好像类似线段树或者分块的数据结构都可以比较快的支持这个操作，复杂度$O(km\log n)$(如果使用 $sort$ 为 $O(km\log n\log k)$) 。

```
struct Msg{
    int ans;//该线段树节点所代表的区间答案
    vector<int >vl,vr;//这个线段树节点所代表的区间最左边和最右边的 1~k 所在位置。
    Msg():ans(1e9),vl(31,1e9),vr(31,0) {}//初始化
    Msg(ll MX):ans(MX) {}
    Msg operator + (const Msg &p)   const{
        Msg res;
        res.ans=min(ans,p.ans);//更新ans1：全部在左边区间或者全部在右边区间
        vector<ii >v;   v.clear();
        for(int i=1;i<=k;++i){//将左边区间右边的1~k和右边区间左边的1~k存入，待会双指针扫描用
            if(vr[i]>=1)    v.pub({vr[i],i});
            if(p.vl[i]<=n)  v.pub({p.vl[i],i});
        }
        sort(all(v));
        int tot=0;
        for(int i=0,j=-1;i<sz(v);++i){//在v上双指针扫描，同时更新答案
            while(j+1<sz(v)&&tot<k){
                j++;
                if(cnt[v[j].se]==0) tot++;
                cnt[v[j].se]++;
            }
            if(tot==k)  res.ans=min(res.ans,v[j].fi-v[i].fi+1);//更新ans2：横跨左右区间
得到的答案
            cnt[v[i].se]--;
            if(cnt[v[i].se]==0) tot--;
        }
        for(int i=1;i<=k;++i)   res.vl[i]=min(vl[i],p.vl[i]);//更新这区间最左边的 1~k
所在位置。
        for(int i=1;i<=k;++i)   res.vr[i]=max(vr[i],p.vr[i]);//更新这区间最右边的 1~k
所在位置。
        return res;
    }
};
```

考察知识点

双指针 线段树 分块 区间问题