

# 错题集 3

## 1 The Escape Plan of Groundhog

[链接](#)

### 题意

给定一个  $n \times m$  的  $01$  矩形，求满足下列条件的子矩阵数目：

1. 该子矩形的边界上没有  $0$
2. 该子矩形中内部(即不含边界)的  $0$  个数差不超过  $1$
3. 该子矩形的长宽大于  $1$

### 题解

考虑  $O(n^2)$  枚举所有的矩阵上下边界，然后  $O(n)$  扫描所有的列。

对与约束条件一，不妨依次处理上下边界均为连续  $1$  的区间，然后只考虑在该区间中全是  $1$  的列的贡献。

对约束条件二，不妨将  $0$  设为  $-1$ ，于是约束变为子矩阵内部的数值和绝对值不超过  $1$ 。

考虑桶维护区间内所有合法列的内部数值和，总时间复杂度  $O(n^3)$

```

const int MAXN=505,MAXM=MAXN*MAXN;
int a[MAXN][MAXN],b[MAXN][MAXN],s[MAXN],c[MAXM<<1];
int main()
{
    int n=read_int(),m=read_int();
    LL ans=0;
    _rep(i,1,n)_rep(j,1,m){
        a[i][j]=read_int();
        if(!a[i][j])a[i][j]=-1;
        b[i][j]=b[i-1][j]+a[i][j];
    }
    _rep(i,1,n)_rep(j,i+1,n){
        int last=0;
        s[last]=MAXM;
        _rep(k,1,m){
            if(a[i][k]==-1||a[j][k]==-1){
                _rep(t,last+1,k){
                    if(b[j][t]-b[i-1][t]==j-i+1)
                        c[s[t]]--;
                }
                s[last=k]=MAXM;
            }
        }
    }
}

```

```
    }  
    else{  
        s[k]=s[k-1]+b[j-1][k]-b[i][k];  
        if(b[j][k]-b[i-1][k]==j-i+1)  
            ans+=c[s[k-1]-1]+c[s[k-1]]+c[s[k-1]+1],c[s[k]]++;  
    }  
}  
_rep(k,last+1,m){  
    if(b[j][k]-b[i-1][k]==j-i+1)  
        c[s[k]]--;  
}  
}  
enter(ans);  
return 0;  
}
```

## 2 Groundhog and Gaming Time

[链接](#)

### 题意

给定  $n$  个区间，每个区间都有  $\frac{1}{2}$  的概率被选中，问所有选中区间的交区间长度的平方的期望值。

### 题解

设离散化后  $\text{X}$  轴被分割为  $[x_1, x_2 \cdots x_{m+1}]$  记  $[x_i, x_{i+1}]$  的长度为  $y_i$

易知某个选择方案的贡献为  $\frac{(y_l + y_{l+1} + \cdots + y_r)(y_l + y_{l+1} + \cdots + y_r)}{2^n}$

上式可以化为  $y_l \frac{y_l + y_{l+1} + \cdots + y_r}{2^n} + y_{l+1} \frac{y_l + y_{l+1} + \cdots + y_r}{2^n} + \cdots + y_r \frac{y_l + y_{l+1} + \cdots + y_r}{2^n}$

于是对每个交区间包含  $[x_i, x_{i+1}]$  的方案  $[x_i, x_{i+1}]$  的贡献为该方案的交区间总长度乘以  $y_i$  除以方案数。

于是  $[x_i, x_{i+1}]$  的总贡献为所有包含  $[x_i, x_{i+1}]$  的方案的交区间长度的期望值乘以  $y_i$  考虑扫描过程中维护所有包含  $[x_i, x_{i+1}]$  的线段。

设某个区间  $[x_j, x_{j+1}]$  假设它被  $d$  条线段覆盖，则它对交区间长度的期望值的贡献为  $\frac{2^{d-1}}{2^n}$

考虑先对每个区间贡献乘以  $2^n$  然后加一，于是只需要建立一棵支持区间乘法的线段树即可维护交区间的期望长度。

最后处理先前操作的影响即可，时间复杂度  $O(n \log n)$

```

const int MAXN=1e6+5,Mod=998244353;
struct Node{
    int l,r;
}node1[MAXN>>1],node2[MAXN>>1];
int x[MAXN],lef[MAXN<<2],rig[MAXN<<2],s[MAXN<<2],lazy[MAXN<<2];
bool cmp1(const Node &a,const Node &b){
    return a.l<b.l;
}
bool cmp2(const Node &a,const Node &b){
    return a.r<b.r;
}
void build(int k,int L,int R){
    lef[k]=L,rig[k]=R;
    s[k]=x[R+1]-x[L],lazy[k]=1;
    if(L==R)return;
    int M=L+R>>1;
    build(k<<1,L,M);build(k<<1|1,M+1,R);
}
void update(int k,int L,int R,int v){
    if(L<=lef[k]&&rig[k]<=R){
        s[k]=1LL*s[k]*v%Mod;
        lazy[k]=1LL*lazy[k]*v%Mod;
        return;
    }
    int mid=lef[k]+rig[k]>>1;
    if(mid>=L)update(k<<1,L,R,v);
    if(mid<R)update(k<<1|1,L,R,v);
    s[k]=1LL*(s[k<<1]+s[k<<1|1])*lazy[k]%Mod;
}
int quick_pow(int a,int b){
    int ans=1;
    while(b){
        if(b&1)ans=1LL*ans*a%Mod;
        a=1LL*a*a%Mod;
        b>>=1;
    }
    return ans;
}
int main()
{
    int n=read_int(),m=0;
    _for(i,0,n){
        node1[i].l=read_int(),node1[i].r=read_int()+1;
        x[++m]=node1[i].l,x[++m]=node1[i].r;
    }
    sort(x+1,x+m+1);
    m=unique(x+1,x+m+1)-x;
    _for(i,0,n){
        node1[i].l=lower_bound(x+1,x+m,node1[i].l)-x;
        node1[i].r=lower_bound(x+1,x+m,node1[i].r)-x-1;
    }
}

```

```
node2[i]=node1[i];
}
m-=2;
build(1,1,m);
sort(node1,node1+n,cmp1);sort(node2,node2+n,cmp2);
int pos1=0,pos2=0,inv2=(1+Mod)/2,ans=-1LL*(x[m+1]-x[1])*(x[m+1]-x[1])%Mod;
_rep(i,1,m){
while(pos1<n&&node1[pos1].l<=i)update(1,node1[pos1].l,node1[pos1].r,2),pos1++;
while(pos2<n&&node2[pos2].r<i)update(1,node2[pos2].l,node2[pos2].r,inv2),pos2++;
ans=(ans+1LL*(x[i+1]-x[i])*s[1])%Mod;
}
ans=1LL*ans*quick_pow(inv2,n)%Mod;
enter((ans+Mod)%Mod);
return 0;
}
```

### 3 Decrement on the Tree

[链接](#)

#### 题意

给定一棵边权树。每次可以选择一条所有边的边权为正的路径，将路径上的所有边权减一。问至少需要多少次操作才能使树上所有边权为零。

接下来若干次修改，每次可以选择一条边修改边权，每次修改后再次询问至少需要多少次操作才能使树上所有边权为零。

#### 题解

可以将上述问题理解为最小路径覆盖，于是答案为起点数  $\$s\%$  终点数除以  $\$2\%$ 。

先考虑答案下界。如果每个点的来自某个方向的路径数  $\$v\%$  大于所有路径和  $\$s\%$  的一半，则最优方案为将其他所有  $\$s-v\%$  条路径与该方向路径配对。

则无法配对的路径还有  $\$2v-s\%$  条，他们只能作为起点或终点。

如果每个点的来自某个方向的路径数  $\$v\%$  不大于所有路径和  $\$s\%$  的一半且路径和为奇数，则会留下一个端点，如果为偶数则可以恰好配对。

接下来考虑答案下界的构造方案，可以每次选择度数为  $\$1\%$  的节点，将它删除同时将未配对的路径沿伸到与它相邻的节点，不断重复上述操作即可。

```
const int MAXN=1e5+5;
```

```

struct Edge{
    int u,v,w;
}edge[MAXN];
LL s[MAXN],ans;
multiset<int,greater<int> > g[MAXN];
void update(int i,int k){
    int v=*g[i].begin();
    if(v*2>s[i])ans+=(v*2-s[i])*k;
    else if(s[i]&1)ans+=k;
}
int main()
{
    int n=read_int(),q=read_int(),t,w;
    _for(i,1,n){
        edge[i].u=read_int(),edge[i].v=read_int(),edge[i].w=read_int();
        g[edge[i].u].insert(edge[i].w);g[edge[i].v].insert(edge[i].w);
        s[edge[i].u]+=edge[i].w,s[edge[i].v]+=edge[i].w;
    }
    _rep(i,1,n)update(i,1);
    enter(ans/2);
    while(q--){
        t=read_int(),w=read_int();
        update(edge[t].u,-1);update(edge[t].v,-1);
        g[edge[t].u].erase(g[edge[t].u].find(edge[t].w));g[edge[t].v].erase(g[edge[
t].v].find(edge[t].w));
        s[edge[t].u]-=edge[t].w,s[edge[t].v]-=edge[t].w;
        edge[t].w=w;
        g[edge[t].u].insert(edge[t].w);g[edge[t].v].insert(edge[t].w);
        s[edge[t].u]+=edge[t].w,s[edge[t].v]+=edge[t].w;
        update(edge[t].u,1);update(edge[t].v,1);
        enter(ans/2);
    }
    return 0;
}

```

From:

<https://wiki.cvbbacm.com/> - CVBB ACM Team

Permanent link:

[https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal\\_string:jxm2001:other:%E9%94%99%E9%A2%98%E9%9B%86\\_3&rev=1597629878](https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string:jxm2001:other:%E9%94%99%E9%A2%98%E9%9B%86_3&rev=1597629878)

Last update: 2020/08/17 10:04