

Test 20221114

依旧是爆零的一天呢，虽然没有那么爆

T1 substr

一眼 AC 自动机（不是说好的省选算法不考吗），然后不会。

题意

给定 n 个字符串 S_1, S_2, \dots, S_n ，要求找到一个最短的字符串 T ，使得这 n 个字符串都是 T 的子串。

分析

好像可以不用 AC 自动机，直接硬状压都可以？毕竟这题只有十二个字符串，但是我状压也贼弱啊，没有网是在没有办法搜题解了，只能去复习 AC 自动机了。

然而现在还没写完

Code

```
null
```

Code

T2 flower

题意

简单来说，给出一个序列 a_i ，每次询问给出 $[l, r], k$ 询问区间 $[l, r]$ 内在模 k 意义下最大的是多少。

分析

这个题一看到我就想到了感觉是主席树加一个线段树取模，但是主席树好像忘了怎么打了，理所应当地放弃，然后打了一个暴力，注意在区间最大值都没有 k 大的时候，就可以直接输出区间最大值了，然后这样就有 40 分。

正解果然是分块啊，这数据范围不就是为 $O(n\sqrt{n})$ 设计的吗。

分块还没改出来，如果说八点之前要交题解的话，那就暂时只能这样了，但是我一测的话本地 lemon 测出来有 70，不知道为什么上面掉了 30，可能是因为没有开 O2 的原因吧

Code

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<queue>
#define Hanggoash
using namespace std;
```

```

template<typename T>inline void re(T &x)
{
    x=0;
    int f=1;char c=getchar();
    for(;!isdigit(c);c=getchar())if(c=='-')f=-1;
    for(;isdigit(c);c=getchar())x=(x<<1)+(x<<3)+(c^48);
    x*=f;
}

template<typename T>inline void wr(T x)
{
    if(x<0)putchar('-'),x=-x;
    if(x>9)wr(x/10);
    putchar(x%10^48);
}

const int maxn=1e5+10;
int a[maxn];
int n,m,k;
struct Segtree{int l,r,val;}rt[maxn<<2];
inline void pushup(int x){rt[x].val=max(rt[x<<1].val,rt[x<<1|1].val);}
inline void build(int x,int l,int r)
{
    rt[x].l=l,rt[x].r=r;
    if(l==r){rt[x].val=a[l];return ;};
    int mid=l+r>>1;
    build(x<<1,l,mid),build(x<<1|1,mid+1,r);
    pushup(x);
}

inline int query(int x,int l,int r)
{
    if(l<=rt[x].l&&rt[x].r<=r)return rt[x].val;
    int mid=rt[x].l+rt[x].r>>1;
    int tmp=-1;
    if(l<=mid)tmp=max(tmp,query(x<<1,l,r));
    if(r>mid)tmp=max(tmp,query(x<<1|1,l,r));
    return tmp;
}

inline void solve()
{
    re(n),re(m);
    for(register int i=1;i<=n;++i)re(a[i]);
    build(1,1,n);
    for(register int i=1,l,r;i<=m;++i)
    {
        re(l),re(r),re(k);
        int ans=-1,tmp;
        if(k>(tmp=query(1,l,r))){ans=tmp;goto A;}
        for(register int j=1;j<=r;++j)
        {
            ans=max(ans,a[j]%k);
            if(ans==k-1)break;
        }
        A:wr(ans),putchar('\n');
    }
}

```

```
int main()
{
    #ifdef Hanggoash
    freopen("flower.in", "r", stdin);
    freopen("flower.out", "w", stdout);
    #endif
    solve();
    return 0;
}
```

T3 reflect

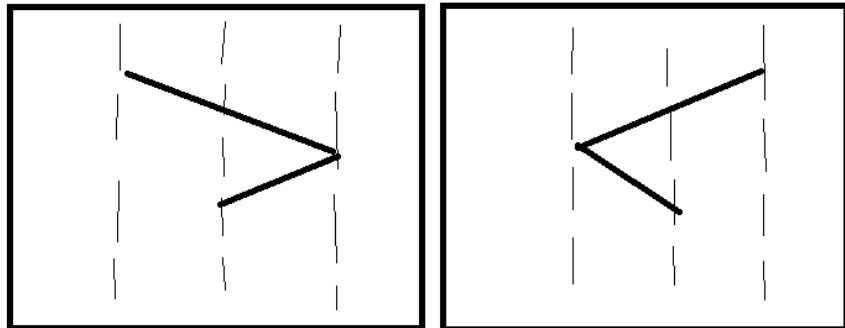
强烈怀疑是银牌佬 LOL 打多了不会写英文了

题意

给出 n 个 (x, y) 二元组 作为平面上的镜子，然后给出一种折射方式 (x_i, x_j 和 y_i, y_j 的大小关系)，问一共能构成多少光线。

分析

首先，题目里面提到的折射方式只能是这样：（为什么有点像揣手手？）



错解

（或者不能叫做解）

这个题，一看就知道是 dp 了，包括考场上面也是打的 dp，比较容易想到的一种方向是把所有点按照 y 降序排序，然后定义 $dp[i][j][k][2]$ 为第 i 个点，当前两个点的端点为 j, k ，方向为 0, 1（左、右）时的方案数，然后就很容易写出状态转移方程了，当然这是错解，时间空间都是错误的，就不多说了，只要知道这种方法的瓶颈在于必须记录左右端点作为状态。

正解

不妨换一个角度，按照 x 升序排序，然后定义 $dp[i][2]$ 为在第 i 个点，射出的光线方向是 $1, 0$ （左、右）的方案数，发现我们如果排序后枚举 x ，那么当前枚举到的这个 x 一定只能是三元光线的第一个点或者第二个点（详情见上图）。

然后我们就要开始奇奇怪怪地同时使用刷表法和填表法了？？？

虽然说 dp 的状态就是一张 DAG，但是不借助关系图来理解，实在是不好懂题解的 dp 操作，画出来那个 DAG 就发现这样算貌似是不会算重的。

然后状态转移很简单，当前从当前这个点往前面枚举：

如果前面的点的 y 小于当前，那么当前和前面的那个点就可以分别构成三元光线的第一、第二个点，所以 $dp[i][1] += dp[j][0]$

（填表）

如果前面的点的 y 大于当前，那么当前和前面的那个点就可以分别构成三元光线的第二、第一个点，所以 $dp[j][0] += dp[i][1]$

（刷表）

最后的答案就是 $(\sum_{i=1}^n dp[i][1] + dp[i][0]) - n$

小细节

讲一下为什么要减去 n ，因为这个题默认一个点也算一种光线，我们转移的时候需要把一个点的 $dp[x][1]/[0]$ 都设置成 1，但是统计答案的时候势必会把单点的情况多统计一次，所以需要减去。

感觉和数位 dp 前导零的处理异曲同工了。

Code

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<queue>
#define int long long
#define Hanggoash
using namespace std;
const int MOD=1e9+7;
template<typename T>inline void re(T &x)
{
    x=0;
    int f=1;char c=getchar();
    for(;;!isdigit(c);c=getchar())if(c=='-')f=-1;
    for(;;isdigit(c);c=getchar())x=(x<<1)+(x<<3)+(c^48);
    x*=f;
}
template<typename T>inline void wr(T x)
{
    if(x<0)putchar('-'),x=-x;
    if(x>9)wr(x/10);
    putchar(x%10^48);
}
```

```

const int maxn=6010;
struct point{int x,y;inline void read(){re(x),re(y);}}p[maxn];
bool cmp(point a,point b){return a.x<b.x;}
int n;
inline void pre()
{
    re(n);
    for(register int i=1;i<=n;++i)p[i].read();
    sort(p+1,p+n+1,cmp);
}
int dp[maxn][2];
signed main()
{
    #ifdef Hanggoash
    freopen("refract.in","r",stdin);
    freopen("refract.out","w",stdout);
    #endif
    pre();
    for(register int i=1;i<=n;++i)
    {
        dp[i][1]=dp[i][0]=1;
        for(register int j=i-1;j>=1;--j)
        {
            if(p[j].y>p[i].y)dp[j][0]=(dp[j][0]+dp[i][1])%MOD;
            else dp[i][1]=(dp[i][1]+dp[j][0])%MOD;
        }
    }
    // for(register int i=1;i<=n;++i)
    //     printf("dp[%d][0]:%d dp[%d][1]:%d\n",i,dp[i][0],i,dp[i][1]);
    int ans=0;
    for(register int i=1;i<=n;++i)ans=(ans+dp[i][1]+dp[i][0])%MOD;
    wr((ans-n+MOD)%MOD);

    return 0;
}

```

T4 paint

题意

初始有一个 $r \times c$ 的全 0 矩阵，然后一次可以把一个连通块（不能斜对角联通）内所有的元素变成 0/1

分析

这个题考试的时候看到真的没什么思路（还以为又是网络流呢），但是想到了应该存在一种操作使得我们能够从外向内一层一层地染色使得答案成立，并且这种操作一定是最优的，这只是考试的时候猜的结论，并没有想到怎么写。

考完以后看了正解，发现就是在这种结论上面隐图最短路，然后这里有 0, 1 两种边权，典型的 0/1bfs 了，感觉最近银牌佬出这种题出的好多，0 的优先级高于 1。

发现 c, r 都很小，于是我们尝试枚举每一个点，我们假设这个点的所在的连通块是我们最后一次操作的连通块，那么我们从这个点往外开始 bfs，遇到和当前点颜色相同的且没有被遍历过的点：那么同色就 dis 不变， $push_front()$ ；异色就 $dis+1$ ， $push_back()$ ；最后在所有目标图上为 1 的点的 dis 中取一个最大值，然后最大值+1，就是当前点作为最后一个被操作的连通块的情况下的答案。

然后我们在所有点的答案中取一个最小值就是最后要输出的答案了。

小细节

注意最后一个被操作的有可能是 0，也有可能是 1

至于为什么最后最大值要 +1 呢？因为我们第一次操作一定是把一个集合全部变成 1，然而我们的程序并不会考虑这第一步操作（随便写一个样例就知道了），所以最后答案要 +1。

Code

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<queue>
#define Hanggoash
using namespace std;
const int maxn=100;
template<typename T>inline void re(T &x)
{
    x=0;
    int f=1;char c=getchar();
    for(;!isdigit(c);c=getchar())if(c=='-')f=-1;
    for(;isdigit(c);c=getchar())x=(x<<1)+(x<<3)+(c^48);
    x*=f;
}
template<typename T>inline void wr(T x)
{
    if(x<0)putchar('-'),x=-x;
    if(x>9)wr(x/10);
    putchar(x%10^48);
}
int n,m,map[maxn][maxn];
int dis[maxn][maxn];
int c1[]={0,-1,0,0,1},c2[]={0,0,-1,1,0};
int ans=0x3f3f3f3f;
deque< pair<int,int> >q;
inline bool valid(int x,int y){return 1<=x&&x<=n&&1<=y&&y<=m;}
inline void pre()
{
    re(n),re(m);
    for(register int i=1;i<=n;++i)
        for(register int j=1;j<=m;++j)
            scanf("%1d",&map[i][j]);
}
inline void bfs(int i,int j)
{
    memset(dis,-1,sizeof dis);
    int tmp=-1;
```

```

dis[i][j]=0,q.push_back(make_pair(i,j));
while(q.size())
{
    int x=q.front().first,y=q.front().second;
    q.pop_front();
    if(map[x][y])tmp=max(tmp,dis[x][y]);
    for(int i=1;i<=4;++i)
    {
        int x1=x+c1[i],y1=y+c2[i];
        if(!valid(x1,y1)||dis[x1][y1]==-1)continue;
        if(map[x1][y1]==map[x][y])
        {
            dis[x1][y1]=dis[x][y];
            q.push_front(make_pair(x1,y1));
        }
        else
        {
            dis[x1][y1]=dis[x][y]+1;
            q.push_back(make_pair(x1,y1));
        }
    }
}
ans=min(ans,tmp+1);//最开始把所有变成 1 的那一步还要加上去
}

int main()
{
    #ifdef Hanggoash
    freopen("paint.in","r",stdin);
    freopen("paint.out","w",stdout);
    #endif
    pre();
    for(register int i=1;i<=n;++i)
        for(register int j=1;j<=m;++j)
            bfs(i,j);
    wr(ans);
    return 0;
}

```

总结

还是不想说什么

觉得自己还是太菜了

传统艺能：

如果硬要说点什么的话，那就是会的不够多，比如 T2 想到了分块写不来，那就跟没想到是一样的，所以考完试不要吼自己马上就写出来正解了，it's cruel，判题机又不是根据你脑子里想的做法是什么来评分，写得出来才是硬道理，像我，就是太菜了，要学的还是太多了。

记得以前有一个题叫做电路维修，本来可以用 $0/1bfs$ 来做，但是当时偷懒就直接写了个 dj 水过了，不想去学，但是真正要用的时候又不会了，所以说逃避迟早会付出代价，能多学就要多学，不要留下遗憾。

