猜数游戏

一. 考察内容：

线段树 二分答案

二. 题目分析：

[题目大意]

给出一些约束条件如l r a表示区间l到r，最小值为a，区间任意两个数都不相同，问这约束条件从第几个开始有互相矛盾的地方。

[写题思路]

经过仔细思考，可以发现如果有条件互相矛盾，一定是出现了以下两种情况：

1.最小值相同的区间交集为∅

2.某个区间包含一个比该区间最小值还小的区间（或是某个区间包含一个比该区间最小值还小的同权区间交集）

于是我们想到了用线段树同时处理这两种情况，由于需要离线处理，所以考虑二分答案，对于二分出的这些询问，我们按照权值从小到大排序，然后在线段树上覆盖即可。

对于第一种情况，我们在线段树上标记一个覆盖次数，我们想最小值相同的约束条件加入线段树，如果最后整个区间的最大覆盖次数小于加入的约束条件个数，则说明这些约束条件交集为∅。

对于第二种情况，我们让线段树支持区间赋值操作，对于一个约束条件l r a，我们将区间l—r赋值为a，最后依次查询每个约束条件的区间最小值是否为a，但是这里有点小bug，就是有可能出现情况2括号里的特殊情况，这时我们只需要将权值为x的一个约束条件的区间改为权值为x的所有约束条件的交集即可。

三. 代码实现：

```
#define _CRT_SECURE_NO_DEPRECATE
/************************
*创建时间：2018 09 09
*文件类型：源代码文件
*题目来源：BZOJ
*当前状态：已通过
*备忘录：线段树 二分答案
*作者：HtBest
************************/
#include <stdio.h>
#include <iostream>
#include <string>
#include <stdlib.h>
#include <string.h>
#include <algorithm>
#include <queue>
#include <bitset>
// #include <sys/wait.h>
// #include <sys/types.h>
// #include <unistd.h>
using namespace std;
#define MAXN 1000001
#define MAXQ 25001
int n,q;
struct QUERY
{
    int l,r,v;
    bool operator < (QUERY x) const
```

```
    {
        return v<x.v;
    }
}a[MAXQ],b[MAXQ];
struct NODE
{
    int lson,rson,v,cover,lazyc,lazyv,lazy_reset_c;
    void clear(){lson=rson=v=cover=lazyc=lazyv=0;}
    void merge(NODE &a,NODE &b)
    {
        v=min(a.v,b.v);
        cover=max(a.cover,b.cover);
    }
    void addl(int lc,int lv,int set)
    {
        if(lv)v=lazyv=lv;
        if(set)cover=lazyc=0,lazy_reset_c=1;
        lazyc+=lc,cover+=lc;

    }
}o[2*MAXN];
struct segmentTree
{
    int root,cnt;
    void downl(NODE &x)
    {
        if(x.lazyc||x.lazyv||x.lazy_reset_c)
        {
            o[x.lson].addl(x.lazyc,x.lazyv,x.lazy_reset_c);
            o[x.rson].addl(x.lazyc,x.lazyv,x.lazy_reset_c);
        }
        x.lazyc=x.lazyv=x.lazy_reset_c=0;
    }
    void build(int &x,int l,int r)
    {
        x=++cnt;
        o[x].clear();
        if(l!=r)
        {
            int m=(l+r)>>1;
            build(o[x].lson,l,m);
            build(o[x].rson,m+1,r);
            o[x].merge(o[o[x].lson],o[o[x].rson]);
        }
    }
    void update(int x,int l,int r,int ul,int ur,int v)
    {
        if(l>=ul&&r<=ur)o[x].addl(1,v,0);//,printf("覆盖%d---%d\n",l,r);
        else
        {
            int m=(l+r)>>1;
            downl(o[x]);
            if(ul<=m)update(o[x].lson,l,m,ul,ur,v);
            if(ur>m)update(o[x].rson,m+1,r,ul,ur,v);
            o[x].merge(o[o[x].lson],o[o[x].rson]);
        }
    }
    void query(int x,int l,int r,int ql,int qr,NODE &q)
    {
        if(l>=ql&&r<=qr)q=o[x];
        else
        {
            int m=(l+r)>>1;
            downl(o[x]);
            if(qr<=m)query(o[x].lson,l,m,ql,qr,q);
            else if(ql>m)query(o[x].rson,m+1,r,ql,qr,q);
            else
            {
                NODE a,b;
                query(o[x].lson,l,m,ql,qr,a);
                query(o[x].rson,m+1,r,ql,qr,b);
                q.merge(a,b);
```

```
            }
        }
    }

    int query_l(int x,int l,int r)
    {
        if(l==r)return l;
        else
        {
            int m=(l+r)>>1;
            downl(o[x]);
            if(o[o[x].lson].cover>=o[o[x].rson].cover)return query_l(o[x].lson,l,m);
            return query_l(o[x].rson,m+1,r);
        }
    }

    int query_r(int x,int l,int r)
    {
        if(l==r)return l;
        else
        {
            int m=(l+r)>>1;
            downl(o[x]);
            if(o[o[x].rson].cover>=o[o[x].lson].cover)return query_r(o[x].rson,m+1,r);
            return query_r(o[x].lson,l,m);
        }
    }

    void build(){cnt=0,build(root,1,n);}

    void update(int l,int r,int v){update(root,1,n,l,r,v);}

    int q_cover()
    {
        NODE ans;
        query(root,1,n,1,n,ans);
        return ans.cover;
    }

    int q_min(int l,int r)
    {
        NODE ans;
        query(root,1,n,l,r,ans);
        return ans.v;
    }

    void cover_clear()
    {
        o[root].addl(0,0,1);
    }
}seg;
/* Variable explain:

*/
void read()
{
    scanf("%d%d",&n,&q);
    for(int i=1;i<=q;++i)scanf("%d%d%d",&a[i].l,&a[i].r,&a[i].v);
    return;
}
bool check(int x)
{
    seg.build();
    for(int i=1;i<=x;++i)b[i]=a[i];
    sort(b+1,b+1+x);
    int cnt=0;
    for(int i=1;i<=x;++i)
    {
        if(b[i].v!=b[i-1].v)
        {
            // printf("reset\n");
```

```
            if(seg.q_cover()!=cnt){/*printf("程序运行在x=%d,i=%d时出现错误：%d
%d\n",x,i,cnt,seg.q_cover());*/return false;}
            b[i-1].l=seg.query_l(seg.root,1,n);
            b[i-1].r=seg.query_r(seg.root,1,n);
            seg.cover_clear();//全部置零
            cnt=0;
        }
        seg.update(b[i].l,b[i].r,b[i].v);
        // printf("覆盖：%d---%d，当前覆盖数：%d\n",b[i].l,b[i].r,seg.q_cover());
        // NODE a;

        // for(int j=1;j<=n;++j)
        // {
        //  int ls1,ls2;
        //  seg.query(seg.root,1,n,j,j,a);
        //  if(ls1!=a.cover)ls1=a.cover,printf("%d ",ls1);
        // }
        // puts("");
        ++cnt;
    }
    // if(seg.q_cover()!=cnt)return false;
    for(int i=1;i<=x;++i)if(seg.q_min(b[i].l,b[i].r)!=b[i].v)return false;
    return true;
    //校验：1.每个区间的最小值是否为该值，2.每次区间求交集是否不为空
}
void solve()
{
    int l=0,r=q,ans=0;
    while(l<=r)
    {
        int m=(l+r)>>1;
        if(check(m))l=m+1,ans=m;
        else r=m-1;
    }
    printf("%d\n",ans==q?0:ans+1);
}
int main()
{
    // freopen(".in","r",stdin);
    // freopen(".out","w",stdout);
    read();
    solve();
    return 0;
}
```

<题目跳转> <查看代码>