Codeforces #319E: Ping-Pong Solution

这道题给的一个条件十分奇怪：区间(a,b)能转移到区间(c,d)当且仅当c<a<d or c<b<d，想一下有两种情况符合这个定义

1. (a,b)与(c,d)互相不包含且有交集
2. (a,b) ⊂ (c,d)

这两种情况又有区别：第一种情况下(a,b)和(c,d)互相联通，而第二种情况只能(a,b)-->(c,d). 如果说第一种情况可以用并查集维护的话，第二种情况就十分难处理

另外题目有一个特殊的限制：区间是按照长度从小到大严格递增给出的，这个条件不知道该怎么用

从简单想起 如果只考虑两种情况中的第一种情况，显然可以想到并查集

注意到这样一种关系是可以传递的，所以(a,b)和(c,d)联通后，这个区间相当于变成了一个长区间(min{a,b},max{c,d}),所以我们对并查集的每个节点再维护一个l,r表示最远延伸的左右端点

对于当前区间(ai,bi)，如何知道他与哪些区间有相交的部分呢？可以想到线段树，对于之前的每个独立的区间（即并查集中的一棵棵树）在线段树的对应区间上打一个标记，然后查询时将ai和bi分别带入线段树，从根查询到叶子的这条路径上，如果某个节点有标记，说明这个区间包含了ai/bi，则(ai,bi)与原区间有交错

这样处理忽略了一种情况，当(ai,bi) ⊂ 原某区间时，也会被误查找到，但注意到题目的条件：区间按照长度递增给出，说明之前的区间中不可能有包含当前区间的区间，于是这个问题就解决了

尝试分析这个问题的复杂度，首先这个标记是不会下放的，每个区间打的标记数是logn级别，所以标记的总数是nlogn级别的

再考虑每个标记会被访问多少次，由于一个标记被访问一定会同时进行一个union的操作，所以访问一次之后这个标记就没有价值了，因为之后我们一定会用它的祖先标记。所以每个标记最多被访问一次。这样复杂度有了保证

'于是这里有一个细节：每次处理完标记以后一定要将标记数组清空，否则复杂度会炸

第一段中的第二种情况还没有考虑，我们尝试分析这样的情况有什么性质

我们设第一种情况的区间移动为A移动，第二种为B移动，那么一个区间到另一个区间的route会是个由A,B构成的字符串

其中全A的部分相当于在并查集中的一棵树中游走

然后我们发现了一个重要的性质：假设有一个集合S，里面的元素是并查集中祖先相同的区间，左右最远端点为l,r;假设存在一个$a \notin S$且a包含S中的某一个元素，那么a一定包含(l,r)

这个结论很好证明，考虑反证法：如果$a \notin S$包含S中的某一个元素b却不包含(l,r),那么要么$a_{left} > l$,要么$a_{right} < r$,那么a与(l,r)一定相交,即$a \in S$,矛盾

既然a包含了(l,r),那我可以直接从某个区间跳到a，而不需要在之前做若干次A移动，所以，所有的由A,B构成的移动序列，都可以转化成只有一个B加若干个A的移动序列，或是只有A的移动序列

所以最终判断区间(a,b)是否能转移到区间(c,d)只要分两种情况：

1. $findanc(a,b) = findanc(c,d)$
2. $a \in (L[findanc(c,d)], R[findanc(c,d)])$或$b \in (L[findanc(c,d)], R[findanc(c,d)])$

至此这道题全部做完

还有一些小细节

比如说如果两个区间的端点正好重合，它们是不相交的

这个问题有两种解决办法：一个是让线段树的每个叶子维护一个单位长度的线段而不是一个点；另一个是向线段树里添加区间(l,r)的时候，转而添加(l+1,r-1)

我的代码

```
#include <cstdio>
#include <iostream>
#include <cstring>
#include <string>
#include <cstdlib>
#include <utility>
#include <cctype>
#include <algorithm>
#include <bitset>
#include <set>
#include <map>
#include <vector>
#include <queue>
#include <deque>
#include <stack>
#include <cmath>
#define LL long long
#define LB long double
#define x first
#define y second
#define Pair pair<int,int>
#define pb push_back
#define pf push_front
#define mp make_pair
#define LOWBIT(x) x & (-x)
using namespace std;

const int MOD=1e9+7;
const LL LINF=2e16;
const int INF=2e9;
const int magic=348;
```

```
const double eps=1e-10;
const double pi=3.14159265;

inline int getint()
{
    char ch;int res;bool f;
    while (!isdigit(ch=getchar()) && ch!='-') {}
    if (ch=='-') f=false,res=0; else f=true,res=ch-'0';
    while (isdigit(ch=getchar())) res=res*10+ch-'0';
    return f?res:-res;
}

int n;

struct Interval
{
    int left,right;
    int nl,nr;
    vector<int> qind;
}a[100048];int atot=0;

struct Point
{
    int pos,from;
    bool type;
    inline bool operator < (const Point &x) const {return pos<x.pos;}
}b[200048];int btot=0,Ind=0;

Pair q[100048];int qtot=0;

namespace DSU
{
    int pre[100048],L[100048],R[100048];
    inline void init()
    {
        int i;
        for (i=1;i<=atot;i++) pre[i]=i,L[i]=a[i].nl,R[i]=a[i].nr;
    }
    inline int find_anc(int x)
    {
        if (pre[x]!=x) pre[x]=find_anc(pre[x]);
        return pre[x];
    }
    inline void update(int x,int y)
    {
        x=find_anc(x);y=find_anc(y);
        pre[x]=y;
        L[y]=min(L[y],L[x]);R[y]=max(R[y],R[x]);
    }
}

namespace SegmentTree
{
    struct node
    {
        int left,right;
        vector<int> v;
    }tree[400048];
    inline void build(int cur,int left,int right)
    {
        tree[cur].left=left;tree[cur].right=right;tree[cur].v.clear();
        if (left!=right)
        {
            int mid=(left+right)>>1;
            build(cur<<1,left,mid);build(cur<<1|1,mid+1,right);
        }
    }
    inline void update(int cur,int pos,int ind)
    {
        int i;
        for (i=0;i<int(tree[cur].v.size());i++)
            DSU::update(tree[cur].v[i],ind);
        tree[cur].v.clear();
        if (tree[cur].left==tree[cur].right) return;
        int mid=(tree[cur].left+tree[cur].right)>>1;
        if (pos<=mid) update(cur<<1,pos,ind); else update(cur<<1|1,pos,ind);
    }
    inline void Insert(int cur,int left,int right,int ind)
    {
        if (left>tree[cur].right || right<tree[cur].left) return;
        if (left<=tree[cur].left && tree[cur].right<=right) {tree[cur].v.pb(ind);return;}
        Insert(cur<<1,left,right,ind);Insert(cur<<1|1,left,right,ind);
    }
}

int main ()
{
    n=getint();int i,j,type,x,y,l,r;
    for (i=1;i<=n;i++)
    {
        type=getint();
```

```
        if (type==1)
        {
            ++atot;a[atot].left=getint();a[atot].right=getint();
            b[++btot]=Point{a[atot].left,atot,false};b[++btot]=Point{a[atot].right,atot,true};
        }
        else
        {
            q[++qtot].x=getint();q[qtot].y=getint();
            a[atot].qind.pb(qtot);
        }
    }
    sort(b+1,b+btot+1);Ind=0;
    for (i=1;i<=btot;i++)
    {
        if (i==1 || b[i].pos!=b[i-1].pos) Ind++;
        if (!b[i].type) a[b[i].from].nl=Ind; else a[b[i].from].nr=Ind;
    }
    DSU::init();
    SegmentTree::build(1,1,Ind);
    for (i=1;i<=atot;i++)
    {
        SegmentTree::update(1,a[i].nl,i);SegmentTree::update(1,a[i].nr,i);
        l=DSU::L[i];r=DSU::R[i];
        if (r-l>=2) SegmentTree::Insert(1,l+1,r-1,i);
        for (j=0;j<int(a[i].qind.size());j++)
        {
            x=q[a[i].qind[j]].x;y=q[a[i].qind[j]].y;
            x=DSU::find_anc(x);y=DSU::find_anc(y);
            if (x==y) {printf("YES\n");continue;}
            l=DSU::L[y];r=DSU::R[y];
            int x1=a[q[a[i].qind[j]].x].nl,y1=a[q[a[i].qind[j]].x].nr;
            if ((l<x1 && x1<r) || (l<y1 && y1<r))
                printf("YES\n");
            else
                printf("NO\n");
        }
    }
    return 0;
}
```