目录

8.19 练习题.....2
1. Max GEQ Sum CF1691D.....2
2. Counting Rectangles CF1722E.....4
3. New Year Tree CF379F.....6
4. Fox And Names CF510C.....8
5. LIS of Sequence CF486E.....9
6. Watto and Mechanism CF514C.....10
7. Bottles CF730J.....12
8. 无尽的生命 洛谷 P2448.....13

8.19 练习题

1. Max GEQ Sum CF1691D

【问题描述】

现在你有一个由 n 个整数组成的数组 a，(i,j) 满足条件当且仅当 max(ai，ai+1,…,aj)≥ai+ai+1+…+aj，其中 1≤i≤j≤n。问是否所有(i,j) 都满足要求。

【输入格式】

第一行包括一个整数 t，表示数据的组数。

每组数据的第一行包括一个整数 n，表示数组的长度。

每组数据的第二行为 n 个整数，即为所给出的数组。

保证所有数据的 n 之和不超过 2×10^5

【输出格式】

所有（i，j）符合条件输出 YES 否则输出 NO。

【样例输入】

```
3
4
-1 1 -1 2
5
-1 2 -3 2 -1
3
2 3 -1
```

【样例输出】

```
YES
YES
NO
```

题目链接：https://www.luogu.com.cn/problem/CF1691D

【题目分析】

解题思路分为两部分，所有区间最值+区间最大子段和。

对于“所有区间最值”的问题，有一个很套路的做法:用单调栈维护每个值作为最大值所管的区间，即：左边最近的比当前数大的数的位置到右边最近的比当前数大的数的位置。所以只需开一个单调递减的栈，正着扫一遍找出左边最后一个比当前数大的数，然后倒着扫一遍找出右边的。

对于第 x 个数管理的区间[l[x],r[x]]。我们找到当前数作为最大值的区间时，i，j 的范围为 l[x]<=i<=x<=j<=r[x]。其中区间最大值就是 a[x]。

剩下的任务是判断区间的最大子段和与 a[x]的关系，若符合题目要求，则最大子段和应小于等于 a[x]，即维护 sum[j]-sum[i-1]<=a[x]。此时可以利用 st 表或者线段树分别维护前缀和的 max 和 min，用 max 的 sum[j]和 min 的 sum[i-1]作为最大值，判断此时 sum[j]-sum[i-1]与 a[x]的关系即可。

【参考代码】

```
#include<bits/stdc++.h>
using namespace std;
#define ll long long
const int maxn=2e5+10;
ll t,n,nn,a[maxn],sum[maxn],st1[maxn][50],st2[maxn][50];
ll s[maxn],top,l[maxn],r[maxn];
/*
st1 维护前缀和的区间最大值，st2 则维护最小值。
s 是单调栈，top 为栈顶。
[l[i],r[i]]为以 a[i]作为最大值的最大区间。
*/
void init()
{
```

```
    cin>>n;
    nn=log2(n)+1;
    for(int j=1;j<=nn;j++)
        for(int i=0;i<=n;i++)
            st2[i][j]=1e9;
    for(int i=1;i<=n;i++)
    {
        cin>>a[i];
        sum[i]=sum[i-1]+a[i];//前缀和
    }
    for(int i=0;i<=n;i++)
        st1[i][0]=st2[i][0]=sum[i];
}
ll query1(int x,int y)//查询 sum 的区间最大值
{
    int z=log2(y-x+1);
    return max(st1[x][z],st1[y-(1<<z)+1][z]);
}
ll query2(int x,int y)//查询 sum 的区间最小值
{
    int z=log2(y-x+1);
    return min(st2[x][z],st2[y-(1<<z)+1][z]);
}
void ST()
{
    for(int j=1;j<=nn;j++)
    {
        for(int i=0;i+(1<<(j-1))<=n;i++)
        {
            st1[i][j]=max(st1[i][j-1],st1[i+(1<<(j-1))][j-1]);
            st2[i][j]=min(st2[i][j-1],st2[i+(1<<(j-1))][j-1]);
        }
    }
}
void work1()
{
    top=0;
    for(int i=1;i<=n;i++)//左边最近的比 a[i]大的数
    {
        while(top && a[s[top]]<=a[i]) top--;
        l[i]=s[top]+1;//+1 是因为[l[i],r[i]]为闭区间
        s[++top]=i;
    }
    top=0;
    for(int i=n;i;i--)//右边最近的比 a[i]大的数
    {
        while(top && a[s[top]]<=a[i]) top--;
        int x=s[top];
        if(x==0) x=n+1;
        r[i]=x-1;
        s[++top]=i;
```

```
    }
}
bool work2()
{
    for(int i=1;i<=n;i++)
    {
        ll MAX=query1(i,r[i]);
        ll MIN=query2(l[i]-1,i-1);
        if(MAX-MIN>a[i]) return 0;
        //只要有一个不满足条件就 return 0;
    }
    return 1;
}
int main()
{
    cin>>t;
    while(t--)
    {
        init();
        ST();
        work1();//处理 l[i],r[i]
        if(work2()) puts("YES");
        else puts("NO");
    }
    return 0;
}
```

2. Counting Rectangles CF1722E

【问题描述】

给定 n 个矩阵，q 次询问。

每次给定四个整数 hs,ws,hb,wb，长在(hs,hb)，宽在(ws,wb)之间的所有矩阵的面积之和。

【样例输入】

```
3
2 1
2 3
3 2
1 1 3 4
5 5
1 1
2 2
3 3
4 4
5 5
3 3 6 6
2 1 4 5
1 1 2 10
1 1 100 100
1 1 3 3
3 1
```

```
999 999
999 999
999 998
1 1 1000 1000
```

【样例输出】

```
6
41
9
0
54
4
2993004
```

题目链接：https://www.luogu.com.cn/problem/CF1722E

【题目分析】

非常典型的一个二维前缀和和差分求平面上一个矩形内权值之和的题。

常用公式（公式均用于整点上的前缀和和差分）：

- 二维前缀和通用公式:

  d[i][j]=d[i−1][j]+d[i][j−1]−d[i−1][j−1]+w[i][j] 其中 w[i][j]表示点(i,j)上的权值。

- 二维差分通用公式：

  ans=d[x1][y1]−d[x1][y0−1]−d[x0−1]y[1]+d[x0−1][y0−1]

其中(x0,y0)表示矩形上横纵坐标均最小的顶点，(x1,y1)表示矩形上横纵坐标均最大的顶点。

【参考代码】

```
#include<bits/stdc++.h>
#include<bits/stdc++.h>
#define int long long
using namespace std;
int n,q,d[1010][1010],t;
signed main(){
    // freopen("1.out","w",stdout);
    cin>>t;
    while(t--){
        cin>>n>>q;
        for(int i=1;i<=1000;i++){
            for(int j=1;j<=1000;j++){
                d[i][j]=0;
            }
        }
        int x,y;
        for(int i=1;i<=n;i++){
            cin>>x>>y;
            d[x][y]+=x*y;//记录每个位置的权值之和
        }
        for(int i=1;i<=1000;i++){
            for(int j=1;j<=1000;j++){
```

```
            d[i][j]=d[i][j]+d[i-1][j]+d[i][j-1]-d[i-1][j-1];
        }
    }//统计前缀和
    int xs,xt,ys,yt;
    while(q--){
        cin>>xs>>ys>>xt>>yt;
        cout<<d[xt-1][yt-1]-d[xt-1][ys]-d[xs][yt-1]+d[xs][ys]<<endl;//计算矩形内权值的差分结果

    }
  }
  return 0;
}
```

3. New Year Tree CF379F

【问题描述】

有一颗 4 个节点的树，2,3,4 号节点的父亲节点都是 1。

m 次操作。操作有 1 种：

u：设现在这颗树有 n 个节点，则您要新建两个节点 n+1,n+2 并让它们变为 u 的儿子节点（连一条 u 到 n+1 的边和 u 到 n+2 的边）。保证 u 是叶子节点。进行完操作后，您要输出此时树的直径。

1≤m≤5×10^5。

【样例输入】

```
5
2
3
4
8
5
```

【样例输出】

```
3
4
4
5
6
```

题目链接：https://www.luogu.com.cn/problem/CF379F

【题目分析】

题目要求每次向一个叶节点增加两个节点，但两个节点到叶节点的长度都是 1，对于树的直径是否变化的贡献是一样的，所以只需任选一个判断即可。

新加入的节点，如果能对直径产生影响，仅会更新当前直径的一个端点，那么，每次加入节点后，分别求出新节点到当前直径两端点的距离是否大于直径，更新最大值同时更新端点即可。由于初始树已给，除根节点外任选两节点成为直径端点，直径长度为 2 。

利用 LCA+st 表求节点间长度，由于从根节点走到两个节点时会重复经过两节点最近公共祖先的深度，则两节点间长度为 a 节点深度与 b 节点深度和减去两倍最近公共祖先深度。

【参考代码】

```
#include<bits/stdc++.h>
```

```
using namespace std;
const int maxn = 2e6 + 5;
int f[maxn][22], d[maxn], n, q, x;
int LCA(int u, int v) {    //求两点的最近公共祖先
    if(d[u] > d[v]) swap(u, v);   //默认 u 的深度比 v 小
    int h = d[v] - d[u];
    for(int i = 20; i >= 0; --i) {   //将 v 升到 u 的同层
        if(h & (1 << i)) v = f[v][i];
    }
    if(u == v);
    for(int i = 20; i >= 0; --i) {
        if(f[v][i] != f[u][i]) {    //运用倍增，如果没有超过，就往上升
            v = f[v][i];
            u = f[u][i];
        }
    }
    return f[u][0];
}
int main() {
    n = 4;
    f[2][0] = f[3][0] = f[4][0] = 1;
    int len = 2, a = 2, b = 3;
    d[1] = 0; d[2] = d[3] = d[4] = 1;
    cin >> q;
    for(int i = 1; i <= q; ++i) {
        int u = n + 1;
        int v = n + 2;
        n += 2;
        scanf("%d", &x);
        f[u][0] = f[v][0] = x;
        d[u] = d[v] = d[x] + 1;
        for(int j = 1; (1 << j) <= n; j++) {  //更新 ST 表
            f[u][j] = f[f[u][j - 1]][j - 1];
            f[v][j] = f[f[v][j - 1]][j - 1];
        }

        int lca = LCA(a, u);
        if(d[u] + d[a] - d[lca] - d[lca] > len) {  //如果已 u 为起点的直径大于原直径，更新 len
            b = u;
            len++;
        }
        else {
            lca = LCA(b, u);
            if(d[u] + d[b] - d[lca] - d[lca] > len) { //反之亦然
                a = u;
                len++;
            }
        }
        printf("%d\n", len);
    }
    return 0;
```

```
}
```

4. Fox And Names CF510C

【问题描述】

记得熟悉的字典序吗？ 现在有一些按照某个被改变的字典序（26 个字母的顺序不再是 abcdefg...xyz）排序后的字符串，请求出其字典序。

（存在多组解，请输出任意一组）否则输出 Impossible（请注意大小写）

数据范围 字符串很开心自己有至多 100 个字符

【样例输入】

```
3
rivest
shamir
adleman
```

【样例输出】

```
bcdefghijklmnopqrsatuvwxyz
```

题目链接：https://www.luogu.com.cn/problem/CF510C

【题目分析】

首先简化题意，现在有 n 个按照某个被改变的字典序从小到大排序后的字符串，请求出改变后 26 个字母的字典序。有关先后顺序，易想到拓扑排序，考虑如何建图。模拟字符串的排序过程，对于相邻的两个字符串，第一个出现不同的两个字符一定有顺序关系，由先来的向后到的连一条边，即由字典序在前的向在后的连一条边。

在扫字符串的过程中还需要判断后一个字符串是否为前一个字符串的前缀字符串（但不能等于前一个字符串。），因为一个字符串的前缀字符串的字典序一定小于等于这个字符串。如果后一个字符串为前一个字符串的前缀字符串的话就直接输出 "Impossible" 就行了。最后拓扑排序一遍，有环就输出 "Impossible"，否则把更新的字典序输出。

【参考代码】

```
#include<bits/stdc++.h>
using namespace std;
int n,in[1145],cnt,ans[1145];
string s1,s2;
vector<int>e[1145];
queue<int>q;
int main(){
    cin>>n>>s1;
    for(int i=1;i<n;i++){
        cin>>s2;
        int m=min(s1.size(),s2.size()),j;
        for(j=0;j<m;j++){
            if(s1[j]!=s2[j]){
                int x=s1[j]-96,y=s2[j]-96;//这里给存进去的字符减去('a'-1)
                e[x].push_back(y);
                in[y]++;
                break;
            }
        }
        if(j>=m&&s2.size()<s1.size()){//特判
            printf("Impossible");
            return 0;
        }
```

```
        s1=s2;
    }
    for(int i=1;i<=26;i++) if(!in[i]) q.push(i);
    while(!q.empty()){//拓扑
        int x=q.front();
        q.pop();
        ans[++cnt]=x;
        for(auto a:e[x]){
            in[a]--;
            if(!in[a]) q.push(a);
        }
    }
    if(cnt<26) printf("Impossible");
    else for(int i=1;i<=26;i++) printf("%c",ans[i]+96);//到最后记得把减掉的加回来
    return 0;
}
```

5. LIS of Sequence CF486E

【问题描述】

给你一个长度为 n 的序列 a1,a2,...,an，你需要把这 n 个元素分成三类：1，2，3：

1:所有的最长上升子序列都不包含这个元素

2:有但非所有的最长上升子序列包含这个元素

3:所有的最长上升子序列都包含这个元素

【输入格式】

第一行包含一个正整数 n，表示序列的长度。第二行包含 n 个正整数 a1,a2,...,an，表示序列中的元素。

【输出格式】

一行，包含一个长度为 n 的、由 1,2,3 三种数字组成的字符串，第 i 个数字表示 ai 所属类别。

【样例输入】

```
4
1 3 2 5
```

【样例输出】

```
3223
```

题目链接：https://www.luogu.com.cn/problem/CF486E

【题目分析】

判断元素属于 LIS 序列的哪一类别的模版题，对于一个元素，根据题意有三种 LIS 的类别，对于这一类问题，首先需要求出从前往后以第 i 位为结尾的最长上升子序列，可以记为 len1[i]，同时可以从后往前求出以第 i 位为开头的最长上升子序列记为 len2[i]，len 为最长上升子序列的长度，而类别就由 len1[i],len2[i]以及 len 的关系决定。

当 a[i]在某一最长上升子序列里时，必定满足 len1[i]+len2[i]=len+1,所以当 len1[i]+len2[i]＜len+1 时，为第一类。

第二类与第三类的判断方法是与元素之间的相互关系有关的，当存在 len1[i] = len1[j] ,len2[i] = len2[j]（i!=j）时，说明当前两个元素是可以在 LIS 序列中被替换的，所以属于第二类，反之属于第三类。

求 LIS 可以用二分或树状数组，判断是否存在相同 len 值可以用 map。

【参考代码】

```
#include<bits/stdc++.h>
using namespace std;
#define read(x) scanf("%d",&x)
const int M=1e5+5;
int a[M],b[M],dp[M],len1[M],len2[M],ty[M],vis[M];
int main(){
   int n;read(n);
```

```
    for(int i=1;i<=n;i++)read(a[i]);

    //求 len1
    memset(dp,0x3f,sizeof(dp));dp[0]=0;
    for(int i=1;i<=n;i++){
        int k=lower_bound(dp+1,dp+n+1,a[i])-dp;
        dp[k]=a[i];
        len1[i]=k;//以 a[i]结尾的 LIS 长度
    }
    int m=lower_bound(dp+1,dp+n+1,0x3f3f3f3f)-dp-1;//LIS 长度

    //把数组反转 求 len2
    for(int i=1;i<=n;i++)b[i]=-a[n-i+1];//负号：需要求反转后的最长下降子序列，添负号就变为最长上升子序列，然后就和上面一样
    memset(dp,0x3f,sizeof(dp));dp[0]=-1e9;
    for(int i=1;i<=n;i++){
        int k=lower_bound(dp+1,dp+n+1,b[i])-dp;
        dp[k]=b[i];
        len2[i]=k;
    }
    for(int i=1;i<=n;i++)if(len1[i]+len2[i]==m+1)ty[i]=2;else ty[i]=1;

    //求 len1 是否唯一，即 ty[i]=3
    for(int i=1;i<=n;i++)if(ty[i]==2)vis[len1[i]]++;
    for(int i=1;i<=n;i++)if(vis[len1[i]]==1&&ty[i]==2)ty[i]=3;

    for(int i=1;i<=n;i++)printf("%d",ty[i]);
    return 0;
}
```

6. Watto and Mechanism CF514C

【问题描述】

给出 n 个已知字符串，m 次询问，每次询问给出一个字符串，问上面 n 个字符串中是否有一个字符串满足恰好有一个字母不同于询问的字符串。（字符串的字符集为{'a','b','c'}）n,m<=3e5,输入总长度不超过 6e5。

【样例输入】

```
2 3
aaaaa
acacaca
aabaa
ccacacc
caaac
```

【样例输出】

```
YES
NO
NO
```

题目链接：https://www.luogu.com.cn/problem/CF514C

【题目分析】

Trie 树上的 dfs，观察本题中的字符集，可以发现字符集非常的小，所以单词查询的时间复杂度为 O（3Sn）,S 为本次插叙的字符串的长度，这使得字典树上的 dfs 在本题中可用。

由于题目中要求恰好一个字符不同，考虑构造 dfs（r,u,num）表示当前在字符串第 r 位，在 trie 中节点 u 上，num 记录是否已经存在一位不同。在搜索中，若当前节点在 trie 上不存在并且已经修改过，返回 false，若处理到字符串末尾返回是否有字符串以此节点为结尾并且经历过一个不同的点。若已经修改过一次可以直接向下搜索，否则考虑下一个节点能否修改并继续搜索。

【参考代码】

```
#include <bits/stdc++.h>
using namespace std;

char s[600010];
int trie[400001][26],val[400001],cnt = 0;

void insert()//建树
{
    int u = 0,v,len = strlen(s);
    for (int i = 0; i < len; i++)
    {
        v = s[i] - 'a' +1;
        if (!trie[u][v])
        {
         trie[u][v] = ++cnt;
         memset(trie[cnt],0,sizeof(trie[cnt]));
        }
        u = trie[u][v];
        if (i == len-1) val[u] = 1;
    }
}

bool dfs(int r,int u,int num)
{
    if (s[r])
    {
     int v = s[r] - 'a' + 1;
     if (trie[u][v])
        {
            if (dfs(r+1,trie[u][v],num)) return true;
        }
        if (!num)
        {
            for(int i = 1; i <= 3; i++)
            {
                if (i != v && trie[u][i])
                    if (dfs(r+1,trie[u][i],num+1)) return true;
            }
        }
    }
    else if (num && val[u]) return true;
    return false;
}

int main()
```

```
{
    int n,m;
    scanf("%d%d",&n,&m);
    memset(val,0,sizeof(val));
    for (int i = 0; i < n; i++)
    {
        scanf("%s",s);
        insert();
    }
    for (int i = 0; i < m; i++)
    {
        scanf("%s",s);
        if (dfs(0,0,0)) printf("YES\n");
        else printf("NO\n");
    }
    return 0;
}
```

7. Bottles CF730J

【问题描述】

有 n 瓶水，第 i 瓶水的水量为 ai，容量为 bi。将 1 单位水从一个瓶子转移到另一个瓶子所消耗时间为 1 秒，且可以进行无限次转移。求储存所有水所需最小瓶子数 k 以及该情况下所用最小时间 t。

【输入格式】

第一行输入一个正整数 n（1≤n≤100）。
第二行输入 n 个正整数，第 i 个正整数表示 ai（1≤ai≤100）。
第三行输入 n 个正整数，第 i 个正整数表示 bi(1≤bi≤100)。
对于每一个 i，满足 ai≤bi。

【输出格式】

输出一行两个整数：k 和 t。

【样例输入】

```
4
3 3 4 3
4 7 6 5
```

【样例输出】

```
2 6
```

题目链接：https://www.luogu.com.cn/problem/CF730J

【题目分析】

所需最少瓶子数可以贪心地求出，优先装容量大的瓶子。
DP。最初想的状态是：dp[i][j][k]：前 i 个瓶子里挑 j 个，总水量是 k 时的最短移动时间。然后找不到转移方程。
换状态：因为移一个单位水消耗一单位时间，知道需要 k 个瓶子后，时间就是：“总水量 - 这 k 个瓶子原有的水量和”，那么就可以找 k 个总容量够装所有水，且已有水量总和最大的瓶子。
定义状态：dp[i][j][k]：前 i 个瓶子挑 j 个，它们总容量为 k 时最大的原有水量和。
转移方程：dp[i][j][k] = max { dp[i-1][j][k], dp[i-1][j-1][k-b[i]] + a[i] }
跑完 dp 在 dp[瓶子数][最少瓶子数][水量和 ~ 瓶子总容量] 中找最大值，再用总水量减掉此最大值，就是最小时间。

【参考代码】

```
#include<bits/stdc++.h>
using namespace std;
const int N = 100, V = 100;
```

```
struct bottle
{
    int a, b;
    bool operator < (const bottle &bt) const
    {
        return b > bt.b;
    }
} bot[N+1];

int dp[N+1][N*V+1];

int main()
{
    int n, shui = 0, rong = 0;
    scanf("%d", &n);
    for(int i=1; i<=n; ++i)
    {
        scanf("%d", &bot[i].a);
        shui += bot[i].a;
    }
    for(int j=1; j<=n; ++j)
    {
        scanf("%d", &bot[j].b);
        rong += bot[j].b;
    }
    sort(bot + 1, bot + n + 1);
    int ping = 0;
    for(int i=1, w=shui; w>0; ++i)
    {
        w -= bot[i].b;
        ++ping;
    }
    memset(dp, -1, sizeof dp);
    dp[0][0] = 0;
    for(int i=1; i<=n; ++i)
        for(int j=ping; j; --j)
            for(int k=rong; k>=bot[i].b; --k)
                if(~dp[j-1][k-bot[i].b]) // 先判是否合法状态
                    dp[j][k] = max(dp[j][k], dp[j-1][k-bot[i].b] + bot[i].a);
    int mx = 0;
    for(int i=shui; i<=rong; ++i)
        mx = max(mx, dp[ping][i]);
    printf("%d %d\n", ping, shui-mx);
    return 0;
}
```

8. 无尽的生命 洛谷 P2448

【问题描述】

逝者如斯夫，不舍昼夜！

叶良辰认为，他的寿命是无限长的，而且每天都会进步。

叶良辰的生命的第一天，他有 1 点能力值。第二天，有 2 点。第 n 天，就有 n 点。也就是 Si=i

但是调皮的小 A 使用时光机，告诉他第 x 天和第 y 天，就可以任意交换某两天的能力值。即 Sx↔Sy。

小 A 玩啊玩，终于玩腻了。

叶良辰：小 A 你给我等着，我有 100 种办法让你生不如死。除非能在 1 秒钟之内告知有多少对“异常对”。也就是说，最后的能力值序列，有多少对的两天 x,y，其中 x<y，但是能力值 Sx>Sy 小 A：我好怕怕啊。

于是找到了你。

【输入格式】

第一行一个整数 k，表示小 A 玩了多少次时光机

接下来 k 行，xi,yi，表示将 Sxi 与 Syi 进行交换。

【输出格式】

输出共一行，表示有多少“异常对”。

【样例输入】

```
2
4 2
1 4
```

【样例输出】

```
4
```

题目链接：https://www.luogu.com.cn/problem/P2448

【题目分析】

利用树状数组求逆序对，但数据范围过大，无法一个个求出逆序对。

尽管数据范围很大，但题目中 k 范围不大，最多只涉及 2k 个数，所以可以将未涉及交换的一段区间内的数，也就是说一段连续的数，看作一个整体，记录下它代表的数值 id 和权值 t，然后再进行离散化。

这样将问题转化为了带有权值的逆序对个数，在求逆序对时，将前缀和*权值 t 就可以统计答案了。

【参考代码】

```
#include<bits/stdc++.h>
#define ll long long
using namespace std;
const int N=1e5+5;
int n,st,tot;
struct aaa{
    int x,y;
}Q[N];//记录需要交换的数
int t[2*N],id[2*N];//t:权值。id:代表元
int s[2*N],row[22*N];
ll ans,c[2*N];

int query(int val){//离散化的查询
    return lower_bound(row+1,row+1+tot,val)-row;
}

void adds(int pos,ll w){//
    for(;pos<=tot;pos+=pos&-pos) c[pos]+=w;
    return;
}

ll asks(int pos){
    ll sum=0;
    for(;pos;pos-=pos&-pos) sum+=c[pos];
    return sum;
}
```

```
int main(){
    scanf("%d",&n);
    for(int i=1;i<=n;i++){
        scanf("%d%d",&Q[i].x,&Q[i].y);
        s[i]=Q[i].x;s[i+n]=Q[i].y;
    }
    sort(s+1,s+1+2*n);
    st=unique(s+1,s+1+2*n)-(s+1);
    row[++tot]=s[1];t[tot]=1;
    for(int i=2;i<=st;i++){
        if(s[i]-s[i-1]>1){
            row[++tot]=s[i-1]+1;
            t[tot]=s[i]-s[i-1]-1;
        }
        row[++tot]=s[i];t[tot]=1;
    }
    for(int i=1;i<=tot;i++) id[i]=i;
    for(int i=1;i<=n;i++){
        int x=query(Q[i].x),y=query(Q[i].y);
        swap(t[x],t[y]);
        swap(id[x],id[y]);
    }
    for(int i=tot;i>=1;i--){
        ans+=asks(id[i]-1)*(ll)t[i];//注意，乘法
        adds(id[i],(ll)t[i]);
    }
    printf("%lld",ans);
    return 0;
}
```