扫描线

算法思想

问题引入：给 n 个矩形的左下角和右上角坐标 求并集覆盖的总面积

复杂度 $O(nlogn)$

比如按照横坐标从左到右扫，横坐标小的为 1，横坐标大的为 -1，面积就是相邻横坐标的差乘以当前区间有多少覆盖是正数的（毕竟不可能是负数）。这个线段树维护一下就行了。遇到 1 或者 -1 都交给线段树 $update$ 就可以了。

周长笨方法就是作两边扫描线，横纵各一次，周长是相邻两个阶段覆盖区域的绝对值之差，需要画个图看一下。

但我觉得笨方法看起来更好想不是吗 (x)

代码练习

1.求矩形面积并 hdu1542

```
#include <algorithm>
#include <cstdio>
#include <cstring>
#define maxn 300
using namespace std;

int lazy[maxn << 3];  //标记了这条线段出现的次数
double s[maxn << 3];

struct node1 {
  double l, r;
  double sum;
} cl[maxn << 3];  //线段树

struct node2 {
  double x, y1, y2;
  int flag;
} p[maxn << 3];  //坐标

//定义sort比较
bool cmp(node2 a, node2 b) { return a.x < b.x; }

//上传
void pushup(int rt) {
  if (lazy[rt] > 0)
    cl[rt].sum = cl[rt].r - cl[rt].l;
```

```
  else
    cl[rt].sum = cl[rt * 2].sum + cl[rt * 2 + 1].sum;
}

//建树
void build(int rt, int l, int r) {
  if (r - l > 1) {
    cl[rt].l = s[l];
    cl[rt].r = s[r];
    build(rt * 2, l, (l + r) / 2);
    build(rt * 2 + 1, (l + r) / 2, r);
    pushup(rt);
  } else {
    cl[rt].l = s[l];
    cl[rt].r = s[r];
    cl[rt].sum = 0;
  }
  return;
}

//更新
void update(int rt, double y1, double y2, int flag) {
  if (cl[rt].l == y1 && cl[rt].r == y2) {
    lazy[rt] += flag;
    pushup(rt);
    return;
  } else {
    if (cl[rt * 2].r > y1) update(rt * 2, y1, min(cl[rt * 2].r, y2), flag);
    if (cl[rt * 2 + 1].l < y2)
      update(rt * 2 + 1, max(cl[rt * 2 + 1].l, y1), y2, flag);
    pushup(rt);
  }
}

int main() {
  int temp = 1, n;
  double x1, y1, x2, y2, ans;
  while (scanf("%d", &n) && n) {
    ans = 0;
    for (int i = 0; i < n; i++) {
      scanf("%lf %lf %lf %lf", &x1, &y1, &x2, &y2);
      p[i].x = x1;
      p[i].y1 = y1;
      p[i].y2 = y2;
      p[i].flag = 1;
      p[i + n].x = x2;
      p[i + n].y1 = y1;
      p[i + n].y2 = y2;
      p[i + n].flag = -1;
      s[i + 1] = y1;
```

```
      s[i + n + 1] = y2;
    }
    sort(s + 1, s + (2 * n + 1));  //离散化
    sort(p, p + 2 * n, cmp);  //把矩形的边的横坐标从小到大排序
    build(1, 1, 2 * n);       //按纵坐标建树
    memset(lazy, 0, sizeof(lazy));
    update(1, p[0].y1, p[0].y2, p[0].flag);
    for (int i = 1; i < 2 * n; i++) {
      ans += (p[i].x - p[i - 1].x) * cl[1].sum;
      update(1, p[i].y1, p[i].y2, p[i].flag);
    }
    printf("Test case #%d\nTotal explored area: %.2lf\n\n", temp++, ans);
  }
  return 0;
}
```

2.矩形周长 hdu1828

```
#include <algorithm>
#include <cstdio>
#include <cstring>
#define maxn 5010
using namespace std;
typedef long long ll;

int lazy[maxn << 3];  //标记了这条线段出现的次数
ll s[maxn << 3],s1[maxn << 3];

struct node1 {
    ll l, r;
    ll sum;
} cl[maxn << 3];  //线段树

struct node2 {
    ll x, y1, y2;
    int flag;
} p[maxn << 3];  //坐标
struct node3 {
    ll x1,x2,y;
    int flag;
}pp[maxn << 3];
//定义sort比较
bool cmp(node2 a, node2 b) {
    return a.x < b.x;
}
bool cmp1(node3 a,node3 b) {
    return a.y < b.y;
}
//上传
```

```
void pushup(int rt) {
    if (lazy[rt] > 0)
        cl[rt].sum = cl[rt].r - cl[rt].l;
    else
        cl[rt].sum = cl[rt * 2].sum + cl[rt * 2 + 1].sum;
}

//建树
void build(int rt, int l, int r,ll s[]) {
    if (r - l > 1) {
        cl[rt].l = s[l];
        cl[rt].r = s[r];
        build(rt * 2, l, (l + r) / 2,s);
        build(rt * 2 + 1, (l + r) / 2, r,s);
        pushup(rt);
    } else {
        cl[rt].l = s[l];
        cl[rt].r = s[r];
        cl[rt].sum = 0;
    }
    return;
}

//更新
void update(int rt, ll y1, ll y2, int flag) {
    if (cl[rt].l == y1 && cl[rt].r == y2) {
        lazy[rt] += flag;
        pushup(rt);
        return;
    } else {
        if (cl[rt * 2].r > y1) update(rt * 2, y1, min(cl[rt * 2].r, y2),
flag);
        if (cl[rt * 2 + 1].l < y2)
            update(rt * 2 + 1, max(cl[rt * 2 + 1].l, y1), y2, flag);
        pushup(rt);
    }
}

int main() {
    int temp = 1, n;
    ll x1, y1, x2, y2, ans;
    while (~scanf("%d", &n)) {
        ans = 0;
        for (int i = 0; i < n; i++) {
            scanf("%lld %lld %lld %lld", &x1, &y1, &x2, &y2);
            p[i].x = x1;
            p[i].y1 = y1;
            p[i].y2 = y2;
            p[i].flag = 1;
            p[i + n].x = x2;
```

```
            p[i + n].y1 = y1;
            p[i + n].y2 = y2;
            p[i + n].flag = -1;
            pp[i].y = y1;
            pp[i].x1 = x1;
            pp[i].x2 = x2;
            pp[i].flag = 1;
            pp[i + n].y = y2;
            pp[i + n].x1 = x1;
            pp[i + n].x2 = x2;
            pp[i + n].flag = -1;
            s[i + 1] = y1;
            s[i + n + 1] = y2;
            s1[i + 1] = x1;
            s1[i + n + 1] = x2;
        }
        sort(s + 1, s + (2 * n + 1));  //离散化
        sort(s1 + 1, s1 + 2 * n + 1);
        sort(p, p + 2 * n, cmp);  //把矩形的边的横坐标从小到大排序
        build(1, 1, 2 * n,s);        //按纵坐标建树
        memset(lazy, 0, sizeof(lazy));
        update(1, p[0].y1, p[0].y2, p[0].flag);
        ll dtmp=0;
        for (int i = 1; i < 2 * n; i++) {
            ans += abs(cl[1].sum-dtmp);
            dtmp=cl[1].sum;
            update(1, p[i].y1, p[i].y2, p[i].flag);
        }
        ans+=abs(cl[1].sum-dtmp);
//        printf("%lld\n",ans);
        memset(cl,0,sizeof(cl));
        sort(pp, pp + 2 * n, cmp1);  //把矩形的边的纵坐标从小到大排序
        build(1, 1, 2 * n,s1);        //按纵坐标建树
        memset(lazy, 0, sizeof(lazy));
        dtmp=0;
        update(1, pp[0].x1, pp[0].x2, pp[0].flag);
        for (int i = 1; i < 2 * n; i++) {
            ans += abs(cl[1].sum-dtmp);
            dtmp=cl[1].sum;
            update(1, pp[i].x1, pp[i].x2, pp[i].flag);
        }
        ans+=abs(cl[1].sum-dtmp);
        printf("%lld\n",ans);
    }
    return 0;
}
/*
7
-15 0 5 10
-5 8 20 25
15 -4 24 14
```

```
0 -6 16 4
2 15 10 22
30 10 36 20
34 0 40 16
*/
```