目录

Chapter 0

二维殷集程序实现文档

本篇文档是二维平面上的布尔运算的程序实现的设计文档。

0.1 作业题目

实现张老师的论文《BOOLEAN ALGEBRA OF TWO-DIMENSIONAL CONTINUA WITH ARBITRARILY COMPLEX TOPOLOGY》中定义的殷集和殷集集合上的布尔运算。

0.2 数学定义

0.2.1 殷集的表示

连通区域的边界 Jordan Curve

Jordan Curve 是首尾相连并且没有自相交的简单闭合曲线，详细见论文 **Theorem 3.5**，**Definition 3.6** 和 **Definition 3.7**.

殷集边界 Spadjor

Spadjor 是 Jordan Curve 的集合，并且 Jordan Curves 之间包好关系可以表示成哈斯图。详见论文 **Definition 3.15 - 3.17**. 并且由 **Corollary 3.22.** 殷集与殷集边界一一对应，即每个 Spadjor 可以唯一表示一个殷集。

0.2.2 殷集集合上的布尔运算

切运算

从 Jordan Curve 之间的恰当交点和不恰当交点将所有 Jordan Curves 切成有向曲线(折线)集合。**Definition 4.1.**

粘运算

将有向折线集合沿着端点恰当的粘合得到两两之间没有恰当交点的 Jordan Curve 集合。**Lemma 4.2.**

判断点和 Spadjor 之间的包含关系

Spadjor 将平面划分为殷集的内部和外部，输出点在 Spadjor 表示的殷集的内部或外部。

补运算

若殷集边界 Spadjor s 非空，

- 计算 s 中的 Jordan Curve 之间的交点。
- 表示殷集的 s 和交点调用切运算.
- 调换所有有向线段的方向.
- 使用粘运算得到新的 Spadjor 表示补运算的殷集结果的 Spadjor。

Definition 4.4.

交运算

若殷集边界 Spadjor s1, s2 非空，

- 计算 s1, s2 之间的所有交点。
- 调用切运算得到有向折线集合 l1, l2。
- 移除 l1 中内部点在 s2 表示的殷集外部的折线，l2 中同理。
- 将 l1,l2 中剩余的折线粘合得到表示交运算的殷集结果的 Spadjor。

Definition 4.8.

并运算

并运算可以通过补运算和交运算实现。**Definition 4.10.**

0.3 数学概念的 c++ 包装

这个程序主体有 5 个类：

0.3.1 点 class Point

固有属性：

- 点的坐标：coord[2]：double
- 每个点的唯一身份标识：identity：int (到 Sweepline::points 中 O(1) 的时间找出指定的点。)
- 是哪些线段的端点 (以角度排序)：nearline：AVLTree<pair<double, vector<int> > >(int 唯一标识线段，double 是这条线段与从此点开始的与 x 轴平行，方向是 x 轴正向的射线之间的逆时针方向的角度值)

固有操作：

- 读取修改点的坐标：operator[](const int)：double
- 计算相对位置：operator-(const Point)：Point
- 判断是否同一个点：operator==(const Point)：bool
- 判断两个点的大小顺序：operator<(const Point)：bool, operator>(const Point)：bool

- 点乘和叉乘用于计算有向面积大小：dot(const Point)：double, cross(Point)：double
- 用于 nearline 中的线段按角度大小排序：friend template<> operator<(const pair<double, vector<int> >,const pair<double, vector<int> >)：bool
- 输出与给定线段之间逆时针方向角度最小的线段：nextline(const Line)：Line (要求给定线段必须以该点为端点)
- 添加，删除以此点为端点的线段在 nearline 中的记录：addnearline(const Line)： void delnearline(const Line)：void

0.3.2 线段 class Line

固有属性：

- 线段的端点：lp[2]：int (到 Sweepline::points 中 O(1) 的时间找出指定点，方向为 lp[0]->lp[1]。)
- 每条线的唯一身份标识：identity：int (Sweepline::lines 中找出指定线段)
- 所在的 jordanline： injordanline： int （jordanlines 中找出制定 jordan 线）
- 纵坐标较小的端点所在的 x 轴平行线 l，l 穿过的所有线段按交点 x 坐标大小排序 orderline，orderline 中该线段左边相邻的线段：leftline： int
- lp[0] 处该线段与 x 轴正向之间逆时针方向的角的弧度值 [0，2pi)： angle：double
- 是否在另一个殷集内部：IfInOtherYinset： bool

固有操作：

- 输出端点：operator[](const int)：Point
- 在确定 y 坐标时进行线段之间的比较，以交点的 x 轴坐标比较，使 orderline 中线段能进行排序：operator<(const Line)：bool, operator>(const Line)：bool
- 计算点是否在另一个直线的上：ifintheline(const Point)：int
- 判断两条线段是否相交：ifintersection(const Line)：bool
- 计算两条线段所在直线的交点：intersection(const Line)：Point
- 输出角度：angle()：double
- 输出在哪条 jordan 线内：injordanline()：int
- 输出是否在另一个殷集内：IfInOtherYinset()：bool
- static 函数从已有 line 集合 lines 内生成 Jordanlines 并更新相应数据：generator()：vector<Jordanline>

0.3.3 Jordan 线 class Jordanline

固有属性:

- 包含的线段: line : vector<int> (到 Sweepline::lines 中查找)
- 身份标识: identity : int
- 以这个 Jordan 线的最左下的端点为终点的在这个 Jordan 线内的线段: leftmostline : int
- 从最左下端点出发向左的射线第一个相交的 Jordan 线: leftJordanline : int
- 内边界或者外边界，即方向顺时针或逆时针,type 当 jl 是外边界时为 1，内边界时为 0 : type : int

固有操作:

- 返回身份标识: identity() : int
- 更新 leftmostline，leftjordanline 等数据，为 generator() 生成 spadjor 提供数据: update() : void
- static 函数从已有的 Jordan 线集合 Jordanlines 生成 spadjor : generator() : spadjor

0.3.4 spadjor 与殷集一一对应 class spadjor

固有属性:

- 殷集的边界集合，是一组 Jordan 线: jordanline : vector<int> (到 Jordanline 的 static 成员 jordanlines 中取出相应的 Jordan 线集合)
- 树结构的哈斯图，表现 jordanline 中的 jordan 线的包含关系: hassmap : HassNode*
- spadjor 的类型，若无穷远处属于它，type 为 true，否则为 false : type : bool
- 身份标识: identity : int

固有操作:

- 布尔运算求补: complement() : spadjor
- 布尔运算求交: meet(const spadjor) : spadjor
- 布尔运算求并: join(const spadjor) : spadjor
- 输出输入数据, 依次输出 Jordan 线, 按线段的方向依次输出点的坐标: friend operator«(ostream&, spadjor) : ostream& friend operator»(istream&, spadjor)

0.3.5 哈斯图存储结构 HassNode

固有属性:

- 身份标识: identity : int
- 父节点: int
- 子节点: vector<int>

0.3.6 数据存储结构平衡二叉树 template<class T> class AVLTree

固有属性:

- – 树节点结构 template<class T> class AVLTreeNode
  – 数据: data : T
  – 深度: height : int
  – 左，右，父节点: Left，Right，father : AVLTreeNode<T>*
- 根节点: Root : AVLTreeNode<T>*

固有操作:

- 加入数据: add(T) : AVLTreeNode<T>*
- 可以删除指针指定的树节点或数据。: del(T) : bool; del(AVLTreeNode<T>*) : bool
- 查找某个数据和查找前一个或后一个数据:find(T) : AVLTreeNode<T>*, findprevnode(AVLTreeNode<T>*) : AVLTreeNode<T>*, findnextnode(AVLTreeNode<T>*) : AVLTreeNode<T>*
- 查找第一个数据和最后一个数据:findfirstnode() : AVLTreeNode<T>*, findlastnode() : AVLTreeNode<T>*

0.3.7 扫描线 class sweepline

固有属性:

- static 成员，水平扫描线的 y 轴坐标，用于线段排序。y : double
- static 成员，用平衡二叉树存储扫描线法的事件点 eventpoint : AVLTree<Point>
- static 成员,用平衡二叉树存储扫描线法过程中的有序线段 orderline,orderline2,orderline3 :AVLTree<Line>
- static 成员，扫描线算法，存储每条线在 orderline,orderline2,orderline3 中的树节点指针 orderlinemark, orderlinemark2, orderlinemark3 : AVLTree<pair<int, AVLTreeNode<Line>*»
- static 成员，存储所有点。points : vector<Point>
- static 成员，存储所有线。lines : vecotr<Line>
- static 成员，存储所有还在图上的点和线的 identity。existpoints : AVLTree<int> ; existlines : AVLTree<int>;
- static 成员，存储所有 Jordanline。jordanlines : vector<Jordanline>
- static 成员，存储所有还在图形上的 Jordanline 的 identity。existjordanlines : vector<int>
- static 成员，存储所有 spadjor。spadjors : vector<spadjor>

固有操作:

- static 函数，扫描线法计算所有交点，和更新 Line::leftline,Line::IfInOtherYinset。intersection() : void
- static 函数，在扫描算法中用来计算两条线是否有交点，若有，更新 eventpoint，lines 等数据。: inter(Line) : void

0.4 类之间的关系，UML 类图

之前的类互相依赖，继承关系如图 1 所示

- ⇠ 虚线箭头是依赖关系，即类的实现需要箭头指向的另一个类的协助
- ◇– 空心菱形直线是聚合关系，即菱形所指的类是另一个类的组合。
- ♦– 实心菱形直线是组合关系，和聚合的区别是部分不能单独存在。

0.5 算法设计与证明

0.5.1 Point

Point::operator[](const int)

契约

input 输入一个 int 和一个 Point 类；

output 输出一个 double 或者 Point 类的坐标引用用于修改坐标，通过重载函数实现，具体调用取决于需要；

precondition 输入的 int 是 0 或 1，Point 类已经初始化；

postcondition 对应于 int 的值为 0 或 1，分别输出这个 Point 的 x 轴坐标和 y 轴坐标。

算法实现 直接输出输入的 Point 类的成员 coord[i] 的右值或左值。

证明 直接可得正确性。

Point::norm()

契约

input 输入一个表示向量的 Point 类 p。

output 输出 double 值。

precondition 输入的 Point 类已经初始化。

postcondition 输出的 double 值是输入的 Point 到 0 点之间的距离，也是表示的向量的长度，结合 operator-可以计算点之间的距离。

算法实现

```
1         double d0 = coord[0], d1 = coord[1];
2         return squrt(d1 * d1 + d2 * d2);
```

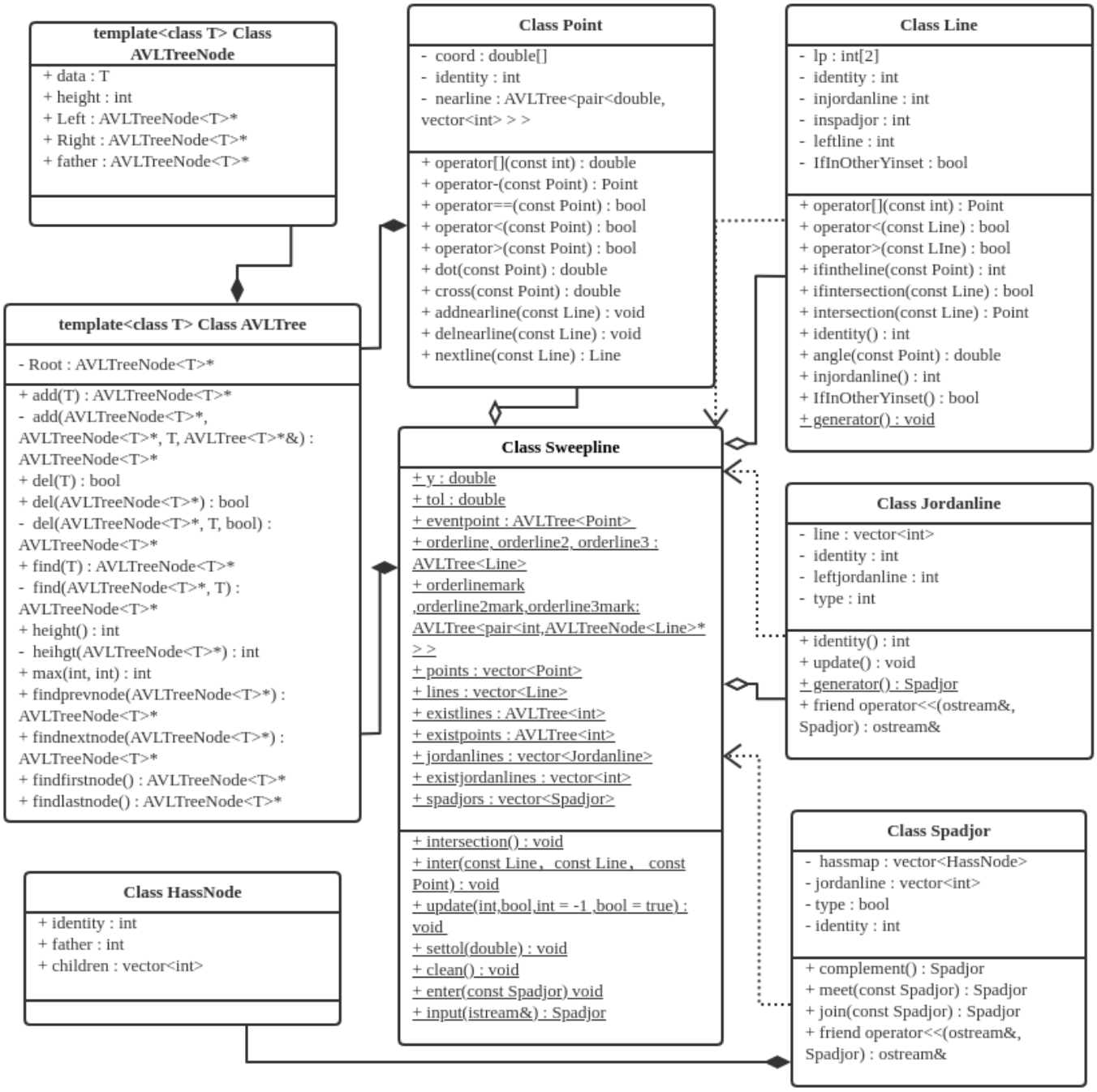

图 1: 以上所有类的关系的 UML 图

证明 由距离和模长的定义，算法符合要求。

Point::operator-(const Point)

契约

input 输入两个 Point 类 p1，p2。

output 输出一个表示两个点相对位置的 Point 类，也是表示 p1 到 p2 的向量的 Point 类。

precondition 输入的类 p1，p2 已初始化。

postcondition 输出的点，x，y 轴坐标分别等于 p1 的 x，y 轴坐标减 p2 的 x，y 轴坐标.

算法实现

```
1         return Point(coord[0] - p2.coord[0], coord[1] - p2.coord[1]);
```

证明 x，y 轴坐标减法计算得到输出的点的横纵坐标，之后调用构造函数。
Sbo

Point::operator<(const Point)

契约

input 输入两个 Point 类 p1，p2。

output 输出一个 bool 值。

precondition 输入的类片，p2 已初始化。

postcondition a 和 b 在容忍度为 tol 的情况下的字典排序比大小结果。即 a 的 y 轴坐标比 b 的 y 轴坐标小 tol，或者当 a 的 y 轴坐标 - b 的 y 轴坐标绝对值小于 tol 时，a 的 x 轴坐标比 b 的 x 轴坐标小 tol，这两种情况返回 true，其他情况返回 false。

算法实现

```
1         if(coord[1] < p2.coord[1] - tol){
2                 return true;
3         }
4         else if((coord[1] > p2.coord[1] - tol) && (coord[1] < p2.coord[1] + tol)
              ){
5                 if(coord[0] < p2.coord[0] - tol){
6                 return true;
7                 }
8         }
9         return false;
```

证明 由字典排序定义，符合要求。

Point::operator>(const Point)

契约与 operator<() 类似，输出结果相反。因此交换参数顺序，调用 operator<(), 返回小于操作符返回的 bool 值。

Point::dot(const Point)

契约

input 输入两个 Point 类 p1，p2。

output 输出 double 值

precondition 输入的 p1,p2 要是 operator-() 输出的表示向量的 Point 类。

postcondition 输出的 double 值是两个向量的点乘结果。

算法实现

```
1         return (coord[0] * p2.coord[0] + coord[1] * p2.coord[1]);
```

证明 正确性由点乘定义得。

Point::cross(const Point)

契约

input 输入两个 Point 类 p1，p2。

output 输出 double 值

precondition 输入的 p1,p2 要是 operator-() 输出的表示向量的 Point 类。

postcondition 输出的 double 值是两个向量的叉乘结果。

算法实现

```
1 return (coord[0] * p2.coord[1] - coord[1] * p2.coord[0]);
```

证明 正确性由叉乘定义得

friend template<> operator<(const pair<double, int>,const pair<double,int>)

用于 Point 类的以此点为端点的线段集合 nearline 排序，double 是 Line 与 x 轴正向之间的逆时针方向角的弧度值 angle，int 是 Line 类的身份标识 identity。

契约

input 输入两个 Pair<double,int> pa1,pa2。

output 输出 bool 值。

precondition 仅用于泛型算法 sort() 将 nearline 排序中。

postcondition 输出按 double 值的大小比较结果。由于是 double 值比较，这里设置的容忍度是 tol/100.

算法实现

```
1         if(pa1.first < pa2.first - tol/100) return true;
2         else return false;
```

证明 由实现显然。

Point::nextline(const Line)

输入 Line l2 在已排序的 nearline 成员中查找，Line l1，使得（l2 - l1）在 mod（2 ）意义下最小。

契约

input 输入一个 Line 类 l2 和一个 nearline vector<pair<double,vector<int»>。

output 输出一个 Line 类 l1，不会在 nearline 中删除 l1，l2。

precondition l2 必须在 nearline 集合中。由每次调用 nearline 的点是 l2 的终点来保证。要求 nearline 中在容忍度 tol/100 的情况下有相等的弧度, 并且没有存储在一个 vector<int> 内。nearline 必须已排序。这个点的出度等于入度。

postcondition 输出的 Line 类 l1 的起点 p 是输入的 l2 的终点。并且它们之间不存在以 p 点为起点的 Line 类 l 使得 $\angle l2 \to l < \angle l2 \to l1$。

算法实现

```
1     Line Point::nextline(const Line& l2){
2     bool t = false, t1 = false;
3     Line l1;
4     vector<int> l2v;
5     l2v.push_back(l2.identity());
6     AVLTreeNode<pair<double, vector<int>>* k = nearline.find(make_pair(l2.angle
          (), l2v));
7     if(k == NULL) cout << "nextline() k out of range";
8         k = nearline.findprevnode(k);
9         if(k == NULL) k = nearline.findlastnode();
```

```
10          if(k != NULL){
11              for(auto i = 0; i < k->data.second.size(); i++){
12              if(t == true && t1 == false){
13                              if(k->data.second[i][0] == identity){
14                                      t1 = true;
15                                      l1 = lines[k->data.second[i]];
16                              }
17                      }
18                      if(k->data.second[i] == l2.identity()){
19                                      t = true;
20                      }
21              }
22      }
23      if(t1 != true){
24              k = nearline.find(make_pair(l2.angle(), l2v));
25              if(k != NULL){
26                      for(auto i = 0; i < k->data.second.size(); i++){
27                          if(t == true && t1 == false){
28                                  if(k->data.second[i][0] == identity){
29                                          t1 = true;
30                                                  l1 = lines[k->data.second[i]];
31                                          }
32                                  }
33                                  if(t == false && k->data.second[i] == l2.
                                       identity()){
34                              t = true;
35                                  }
36                      }
37              }
38      }
39      bool test = false;
40      while(t1 != true){
41              k = nearline.findnextnode(k);
42              if(k == NULL){
43                      k = nearline.findfirstnode();
44                      if(test == false)
45                          test = true;
46                      else {
47                          cout << "nextnearline() wrong test";
48                          exit(0);
49                      }
50                  }
51                  if(k != NULL){
```

```
                    for(auto i = 0; i < k->data.second.size(); i++){
                            if(t == true && t1 == false){
                                    if(k->data.second[i][0] == identity){
                                            t1 = true;
                                            l1 = lines[k->data.second[i]];
                                    }
                            }
                            if(t == false && k->data.second[i] == l2.identity())
                               {
                                t = true;
                            }
                    }
            }
            if(t1 == true) break;
    }
    return l1;
    }
```

证明 nearline 是按 double 值排序，几乎重合的线段存储在一个 double 和 angle 的差小于 tol/100 的 pair<double, vector<int> > 中的 vector 中。因此首先按 l2 的 angle 值搜索必会搜索到 l2 所在的 pair pa1。若 pa1 中的 vector 有以 l2 的终点 p 为起点的线段 l1，那么 l1 闭符合条件，因为 $\angle l2 \to l1 = 0$, 不会有更小的逆时针角。若 pa1 的 vector 中没有符合要求的 l1，那么按 angle 值继续往前搜索，若指针是 nearline 的第一个则将指针移到最后一点。由出度等于入度，必将搜索到符合要求的 l1。

Point::addnearline(const Line)

契约

input 输入一个 Line 类 l，和一个 Point 类 p 的 nearline。

output 在 p 的 nearline 中添加 l.identity。

precondition p 是 l 的端点之一。

postcondition 若 nearline 中已有 pair 的 double 数据与 l.angle 的差小于 tol/100。则将 l.identity 加入到这个 pair 的 vector<int> 中。其余情况插入一个新 pair<double l.angle, vector<int> lv(l.identiy())>。

算法实现

```
        vector<int> lv;
        lv.push_back(l.identity());
        pair<double, vector<int> pa = make_pair(l.angle(), lv);
        AVLTreeNode<pair<double, vector<int>>>* k = nearline.find(pa);
        if(k == NULL)
                nearline.add(pa);
```

```
        else
                k->data.second.insert(l.identity());
```

证明 首先构造一个 pair，使用 AVLTree 的接口函数 find() 在 nearline 中搜寻是否已有 pair 满足 double 值足够接近。如果有，find() 返回这个 pair，将 l 的身份标识 identity 插入到 pair 的第二项 vector 中。如果没有，find() 返回 NULL，将以 l 的数据生成的 pair 插入到 nearline 中。

Point::delnearline(const Line)

契约

input 输入一个 Line 类 l，和一个 Point 类 p 的 nearline。

output 在 p 的 nearline 中删除 l.identity。

precondition p 是 l 的端点，并且 l 的 nearline 中有保存 l.identity。

postcondition 删除 nealine 中保存的 l.identity。若此时存在 pair 的 vector 为空，将存储这个 pair 的节点从 AVLTree 中删除。

算法实现

```
        void Point::delnearline(const Line& l2){
                bool t = false;
                vector<int> l2v;
                l2v.push_back(l2.identity());
                AVLTreeNode<pair<double, vector<int>>* k = nearline.find(
                    make_pair(l2.angle(), l2v));
                if(k == NULL) cout << "nextline() k out of range";
                k = nearline.findprevnode(k);
                if(k == NULL) k = nearline.findlastnode();
                if(k != NULL){
                        auto i = k->data.second.begin();
                        while(i != k->data.second.end()){
                                if(t == false && (*i) == l2.identity()){
                                        t = true;
                                        if (k->data.second.size() == 1){
                                                nearline.del(k);
                                                break;
                                        }
                                        else{
                                                k->data.second.erase(i);
                                                break;
                                        }
                                }
```

```
23                              i++;
24                      }
25              }
26              else cout << "delnearline() wrong";
27              if(t != true){
28                      k = nearline.find(make_pair(l2.angle(), l2v));
29                      if(k != NULL){
30                              auto i = k->data.second.begin();
31                              while(i != k->data.second.end()){
32                                      if(t == false && (*i) == l2.identity()){
33                                              t = true;
34                                              if(k->data.second.size() == 1){
35                                                      nearline.del(k);
36                                                      break;
37                                              }
38                                              else{
39                                                      k->data.second.erase(i);
40                                                      break;
41                                              }
42                                      }
43                                      i++;
44                              }
45                      }
46              }
47              bool test = false;
48              while(t != true){
49                      k = nearline.findnextnode(k);
50                      if(k == NULL){
51                              k = nearline.findfirstnode();
52                              if(test == false)
53                                      test = true;
54                              else {
55                                      cout << "nextnearline() wrong test";
56                                      exit(0);
57                              }
58                      }
59                      if(k != NULL){
60                              auto i = k->data.second.begin();
61                              while(i != k->data.second.end()){
62                                      if(t == false && (*i) == l2.identity()){
63                                              t = true;
64                                              if(k->data.second.size() == 1){
65                                                      nearline.del(k);
```

```
                                                    break;
                                                }
                                                else{
                                                    k->data.second.erase(i);
                                                    break;
                                                }
                                            }
                                            i++;
                                        }
                                    }
                                    if(t1 == true) break;
                                }
                            }
```

证明 由 l 的数据构造一个 pair，使用 find() 在 nearline 中搜索充分接近的 double 值的 pair。l 只可能存储在搜索到的节点的前一个或者后一个，只有这 3 个节点的 pair.first 的 double 值可能与 l.angle 值的差小于 tol/100，因为每个 pair.first 值的差大于 tol/100。在搜索到的 vector 中，如果只有一个值直接删除，还有其他值时只删除 l.identity。

0.5.2 Line

Line::operator[](const int)

契约

input 输入 Line l，int i,point 集合 Sweepline::points。

output 输出 point p。

precondition

i 等于 0 或 1。l 已经初始化。

postcondition

i 等于 0 时输出 l 的起点，i 等于 1 时输出终点。

算法实现

```
        return Sweepline::points[l.lp[i]];
```

证明 访问类数据成员借助 static 成员 Sweepline::points 输出需要的点。

Line::operator<(const Line)

用于在扫描线上对线段排序

契约

input 输入两个 Line l1,ll2; 一个扫描线 y 轴坐标 double sweepline::y;

output 输出 bool 值反映大小关系

precondition l1，l2 与 y 轴坐标为 sweepline::y 的 x 轴平行线 l 有交点，

postcondition 输出 l1 和 l2 与扫描线 l 的交点的 x 轴坐标的大小比较。

算法实现

```
1         vector<Point>& points = Sweepline::points;
2         vector<Line>& lines = Sweepline::lines;
3         Line l1 = lines[this->identity];
4         Line l2 = lines[ll2.identity()];
5         l1 = lines[l1.identity()];
6         double y = sweepline::y;
7         double x1 = min(min(points[l1[0]].[0],points[l1[1]].[0]), min(points[l2
              [0]].[0],points[l2[1]].[0]));
8         double x2 = max(max(points[l1[0]].[0],points[l1[1]].[0]), max(points[l2
              [0]].[0],points[l2[1]].[0]));
9         Point a1(x1, y), a2(x2, y);
10        Line l3(a1, a2);
11        Point p1, p2;
12        Point l1p0 = points[l1[0]], l1p1 = points[l1[1]], l2p0 = points[l2[0]],
              l2p1 = points[l2[1]];
13        if(l1p0 > l1p1) { l1p0 = points[l1[1]]; l1p1 = points[l1[0]];}
14        if(l2p0 > l2p1) { l2p0 = points[l2[1]]; l2p1 = points[l2[0]];}
15        if(l3.ifintheline(l1p1)) p1 = l1p1;
16        if(l3.ifintheline(l1p0)) p1 = l1p0;
17        if(l3.ifintheline(l2p1)) p2 = l2p1;
18        if(l3.ifintheline(l2p0)) p2 = l2p0;
19        if(l3.ifintheline(l1p1) == false && l3.ifintheline(l1p0) == false)
20                p1 = l1.intersection(l3);
21        if(l3.ifintheline(l2p1) == false && l3.ifintheline(l2p0) == false)
22                p2 = l2.intersection(l3);
23        return p1[0] < p2[0];
```

证明 先判断 l1 与扫描线是否重合 coincide, 若重合输出左端点。不重合调用 intersection() 计算交点。最后通过判断交点的顺序可得扫描线与线段相交的顺序，因为重合时左端点即是第一次相交。

Line::operator>(const Line)

交换参数顺序之后，调用 operator<()，输出 < 输出的 bool 值。

Line::ifintheline(const Point)

契约

input 输入 Line l，Point p，double tol,Sweepline::points。

output 输出 bool 值。

precondition l，p 已初始化。

postcondition 若在容忍度 tol 下，与 p 点调用 Point::operator== 返回 true 的所有点都不在 l 上时返回 0，否则返回 1。

算法实现

```
1         Point lp1 = Sweepline::points[l[0]], lp2 = Sweepline::points[l[1]];
2         if(p == lp1 || p == lp2) reutrn 1;
3         if((p[0] < min(lp1[0],lp2[0]) - Sweepline::tol) || (p[0] > max(lp1[0],
              lp2[0]) + Sweepline::tol)
4          ||(p[1] < min(lp1[1],lp2[1]) - tol) || (p[1] > max(lp1[0],lp2[1]) + tol
               ))
5                 return 0;
6         Point p1 = Point(p[0] - tol, p[1] - tol);
7         Point p2 = Point(p[0] + tol, p[1] - tol);
8         Point p3 = Point(p[0] - tol, p[1] + tol);
9         Point p4 = Point(p[0] + tol, p[1] + tol);
10        if(((p1 - lp1).cross(lp1 - lp2) * (p - lp1).cross(lp1 - lp2)) > 0 &&
11           ((p2 - lp1).cross(lp1 - lp2) * (p - lp1).cross(lp1 - lp2)) > 0 &&
12           ((p3 - lp1).cross(lp1 - lp2) * (p - lp1).cross(lp1 - lp2)) > 0 &&
13           ((p4 - lp1).cross(lp1 - lp2) * (p - lp1).cross(lp1 - lp2)) > 0)
14           return 0;
15         return 1;
```

证明 首先判断 p 是否与 l 的端点充分接近，然后判断 p 是否在线段 l 附近。再将与 p 点在容忍度 tol 的情况下等价的所有点所在的正方形的四个顶点 p1,p2,p3,p4 取出，若正方形内存在点在 l 所在直线上，那么这四个顶点也必然在 l 所在直线的两边或者上面。对任一顶点都计算 p1 和 l 两个顶点之间的有向面积乘 p 和 l 两个顶点的有向面积，如果是正数说明在同一边，为 0 说明有一点在 l 所在直线上，为负数说明在两边。又因为前面已经判断 p 在 l 附近，结合得 l 是否包含与 p 等价的点。

Line::coincide(const Line)

契约

input 输入 Line l1，l2。

output 输出 int 值

precondition l1,l2 已初始化。

postcondition 判断 l1，l2 是否重合，即 l1，l2 上的两个端点是同一个点。

算法实现

```
        if(l1[0] == l2[0] || l1[0] == l2[1]){
                if(l1[1] == l2[0] || l1[1] == l2[1])
                        return 1;
        }
        return 0;
```

证明 直接比较 Line 类中存储的端点身份标识进行比较是否是同一个点。如果都是返回 1，否则返回 0。

Line::ifintersection(const Line)

契约

input 输入 Line l1，l2。

output 输出 bool 值。

precondition l1,l2 已初始化。l1 的端点不在 l2 上，l2 端点不在 l1 上。

postcondition 输出 l1，l2 是否有交点，若有输出 true，没有输出 false。

算法实现

```
        vector<Point>& points = Sweepline::points
        Point p10 = points[l1[0]], p11 = points[l1[1]],p20 = points[l2[0]], p21
           = points[l2[1]];
        if(((p10 - p20).cross(p20 - p21)) * ((p11 - p20).cross(p20 - p21)) < 0
           &&
           ((p20 - p10).cross(p10 - p11)) * ((p21 - p10).cross(p10 - p11)) < 0)
                return true;
        return false;
```

证明 同上计算有向面积，两条线段相交等价于一条线段的两个端点分别在另一条线段的所在直线的两端，并且由于没有一条线段端点在另一条线段上，所以它们必交与内部。

Line::intersection(const Line)

契约

input 输入两条 Line l1,l2。

output 输出一个 Point p。

precondition l1，l2 有内部交点，线段端点不在另一条线段上，l1，l2 不重合。

postcondition 输出的 Point 是 l1，l2 的内部交点，在容忍度 tol 下交点和端点不等价。

算法实现

```
vector<Point>& points = Sweepline::points
Point p1 = points[l1[0]], p2 = points[l1[1]],p3 = points[l2[0]], p4 =
    points[l2[1]];
double s1 = fabs((p1 - p2).cross(p2 - p3)), s2 = fabs((p1 - p2).cross(p2
    - p4)));
return Point((p4[0] * s1 + p3[0] * s2)/(s1 + s2), (p4[1] * s1 + p3[1] *
    s2)/(s1 + s2));
```

证明 计算 $\triangle p1p2p3$ 和 $\triangle p1p2p4$ 的面积，面积的比值等于交点到线段端点的距离比值，所以交点的坐标可以通过面积的比值和端点坐标算出。

Line::angle(const Point)

契约

input Line l,Point p。Sweepline::points。

output double

precondition l 已初始化,p 是 l 的端点，points 中存储了它的端点。

postcondition 输出与在起点为 p 的 x 轴正向射线之间的弧度值。范围 [0 - ε,2PI - ε)。ε 是 tol 导致的。

算法实现

```
Point p0,p1;
if(l.lp[0] == p.identity()) { p0 = points[lp[0]]; p1 = points[lp[1]];}
else { p1 = points[lp[0]]; p0 = points[lp[1]];}
Point p = p1 - p0;
double d;
if(p[0] > 0) {
        d = atan(p[1]/p[0]);
        if(d < 0) d = d + 2 * PI;
}
else if(p[0] < 0) {
        d = atan(-p[1]/p[0]); d = PI - d;
}
else {
        if(p[1] > 0) d = PI / 2;
        else if(p[1] < 0) d = PI * 3 / 2;
```

```
16                  else exit(0);
17          }
18          return d;
```

证明

Line 类中还有一些访问和修改私有成员：identity，leftline，IfinOtherYinset 的成员函数，例如 identity() 和 setidentity() 等。

契约

input 输入一个 Line l。

output 输出私有成员的数据或数据引用。

precondition l 已初始化。

postcondition 根据需要输出数据或数据的引用。

算法实现

```
1           return identity;
```

证明

static Line::generator()

契约

input Line 类集合 Sweepline::lines，Point 集合 Sweepline::points。现存所有 Line 类的 identity 集合 Sweepline::existlines

output 更新 Sweepline::jordanlines 的数据。

precondition Sweepline::lines 内的每个 Line l 的端点 Point 的 nearline 集合已全部更新，并且出度等于入度。

postcondition 输出的是还没有更新 leftjordanline，leftmostline，type 默认值的 Jordanline 类集合 vector<vector<Jordanline>>。

算法实现

```
1           void Line::generator() {
2                   vector<Point> points = Sweepline::points;
3                   vector<Line>& lines = Sweepline::lines;
4                   AVLTree<int> existl = Sweepline::existlines;
5                   vector<Jordanline>& jordanlines = Sweepline::jordanlines;
```

```
6                AVLTree<int>& existjordanlines = Sweepline::existjordanlines
7                vector<int> jordanline;
8                int l, p, k;
9                int k = Sweepline::jordanlines.size();
10               AVLTree<pair<int, int>> jvl;
11               while(true){
12                       if(existl.findfirstnode() == NULL) break;
13                       l = (existl.findfirstnode())->data;
14                       p = l[0];
15                       jvl.push_back(make_pair(p, l));
16                       while(true){
17                               int l1 = l;
18                               int p2 = lines[l1][1];
19                               int l2 = (points[p2].nextline(lines[l1]).
                                     identity());
20                               p = p2;
21                               l = l2;
22                               while(jvl.find(make_pair(p, l)) != NULL){
23                                       int pp1 = (jvl.find(make_pair(p, l))->
                                             data).first;
24                                       int ll1 = (jvl.find(make_pair(p, l))->
                                             data).second;
25                                       int pp2 = lines[ll1][1];
26                                       p = pp2;
27                                       if(p == p2){
28                                               points[pp1].delnearline(ll1);
29                                               points[pp2].delnearline(ll2);
30                                               jvl.del(make_pair(pp1, ll1));
31                                               existl.del(ll1);
32                                               lines[ll1].setinjordanline(k);
33                                               jordanline.add(ll1);
34                                               Jordanline jordan(jordanline, k)
                                                     ;
35                                               jordanlines.push_back(jordan);
36                                               jordanline.erase(jordanline.
                                                     begin, jordanline.end());
37                                               existjordanlines.push_back(k);
38                                               k++;
39                                               break;
40                                       }
41                                       int ll2 = (points[pp1].nextline(lines[
                                             ll1])).identity();
42                                       points[pp1].delnearline(ll1);
```

```
                                        points[pp2].delnearline(ll1);
                                        jvl.del(make_pair(pp1, ll1));
                                        existl.del(ll1);
                                        lines[ll1].setinjordanline(k);
                                        jordanline.add(ll1);
                                        l == ll2;
                                }
                                if(jvl.empty()) break;
                                p = p2;
                                l = l2;
                                jvl.add(make_pair(p, l));
                        }
                }
        }
```

证明 首先由每个点出度等于入度，由定理 2.12 知可以生成一些圈恰好遍历所有边。又因为遍历时寻找 nextline 时取逆时针角最小的边，结合不存在 Point 类以外的交点，可得产生的圈没有不合适的交点。还有算法中的遍历的路径上若出现重复的点，将重复的点之间的路径截下产生一个新的圈，因此每个圈不会出现自相交。综上产生的圈都是 jordan 线的 vector 集合。

0.5.3 Jordanline

Jordanline 的私有数据：identity，leftjordanline，type 的更新函数 Jordanline::update()

契约

input 输入 Jordanline jl，和 jl.vector<int> 中对应的 Sweepline::lines 中的 Line 类。

output 更新 jl.leftjordanline,jl.leftmostline,jl.type。

precondition 图上所有的 line 类已经划分到一个个 Jordanline 类中而且更新了 leftline 和 injordanline 数据，vector<int> 中代指的线段是按 jordan 线的方向依次排列的。

postcondition 更新后的 leftmostline 是 jl 以最左下角的点为终点的边的 identity，leftjordanline 是 jl 最左下角的 Point 出发的向左射线第一个相交的 Line 所在的 Jordanline 的 identity。type 当 jl 是外边界时输出 1，内边界时输出 0。

算法实现

```
        vector<Line>& lines = Sweepline::lines;
        vector<Point>& points = Sweepline::points;
        Point p = lines[line[0]].[1];int j = 0;
        for(auto i = 1; i < line.size(); i++){
                if(lines[line[i]].[1] < p) {p = lines[line[i]].[1]; j = i;}
        }
```

```
        leftmostline = j;
        if(lines[line[j]].leftline() == -1) leftjordanline = -1;
        else {
                leftjordanline = lines[lines[line[j]].leftline()].injordanline()
                    ;
                if(leftjordanline == identity && lines[line[j + 1]].leftline()
                    != -1)
                leftjordanline = lines[lines[line[j + 1]].leftline()].
                    injordanline();
                else leftjordanline = -1;
        }
        Point p1 = points[lines[line[j]].[0]],p2 = points[lines[line[j]].[1]],p3
            = points[lines[line[j + 1]].[1]];
        if((p1 - p2).cross(p2 - p3) < 0) type = 0;
        else { type = 1;}
```

证明 首先在 jl 的边集合中按顺序遍历找到最左下的 Point p，设置 leftmostline 为以这条线段为终点的 Line l。由于 p 是字典排序最小的点，所以从 p 出发的向左的射线相交的顺序可以在扫描线算法中 orderline 的线段顺序得到记录在 l.leftline 中，除了与左边的 jordan 线相交，有可能先与 jl 中另一条以 p 为端点的边交于点 p，所以先判断一次是否等于 identity 如过等于，取另一条边的 leftline 才是左下的 jordan 线上的边。因为 p 是最左下点，顺时针或逆时针直接通过计算与 p 相邻的三个点组成依 jl 的方向的顺序围成的三角形的有向面积的正负性可得。

static Jordanline::generator()

契约

input 输入所有 Jordanline 的集合 jordanlines。

output 输出一个 spadjor。

precondition jordanlines 内的所有 Jordanline jl 都已经调用 update() 更新过数据。

postcondition 输出的 spadjor 能一一对应到殷集，殷集的边界 Jordan 线集合由 jordanlines 确定，需要在 spadjor 中完成哈斯图体现 jordan 线之间的包含关系。

算法实现

```
        vector<int> existjordanlines = Sweepline::existjordanlines;
        AVLTree<pair<int,int>> mark;
        vector<Jordanline>& jordanlines = Sweepline::jordanlines;
        vector<spadjor>& spadjors = Sweepline::spadjors;
        Jordanline jl;
        vector<HassNode> vhn(existjordanlines.size() + 1);
        for(int i = 0; i < existjordanlines.size() + 1; i++){
```

```
                int j = existjordanlines[i];
                mark.add(make_pair(existjordanlines[i],i));
                HassNode hn(j);
                vhn.push_back(hn);
        }
        HassNode hn(existjordanlines.size());
        vhn.push_back(hn);
        for(int i = 0; i < existjordanlines.size(); i++){
                jl = jordanlines[existjordanlines[i]];
                while(jl.type == (jordanlines[existjordanlines[i]].type())){
                        if(jl.leftjordanline == -1) break;
                        jl = jordanlines[jl.leftjordanline];
                }
                if(jl.leftjordanline == -1) {
                        vhn[i].father = existjordanlines.size();
                        vhn[existjordanlines.size()].children.push_back(i);
                }
                else {
                        int j = mark.find(make_pair(jl.identity(),0)).second;
                        vhn[i].father = j;
                        vhn[j].children.push_back(i);
                }
        }
        int k = spadjors.size();
        jl = jordanlines[vhn[*(vhn.end()--).children[0]].identity];
        bool b;
        if(jl.type == 1) b = false;
        else b = true;
        return spadjor(vhn, existjordanlines, b, k);
```

证明 主要是更新 jordan 线的包含关系，定义最后一个 hassnode 对应无穷远处依情况定方向的 Jordan 线。对每一个 jordan 线 l，因为每次调用 leftjordanline 转移到下一个 Jordan 曲线都会使得左下点在字典排序上更小所以不会循环，又因为 Jordan 线数量有限。所以要么是找到合适的和 l 方向相反的 jordan 线 l2 包含 l，此时更新 l 对应的 hassnode 的 father 为 l2 的 identity，并且添加 l.identity 到 l2 对应的 hassnode 的 children 中。要么不存在 l2 符合要求，则更新 l 对应的 hassnode 的 father 为无穷远处 jordan 线，并且加入到最后一个 hassnode 的 children 中添加 l.identity。由于每个 hassnode(除了无穷远处 jordan 线对应的 hassnode) 只有唯一的子父关系对应于被包含关系，并且每个 hassnode 都会更新一次 father，所以 spadjor 所有的包含关系都保存到 hassnode 中了。

0.5.4 spadjor

spadjor::complement()

契约

input 输入 Sparjor。Sweepline::points，Sweepline::lines，Sweepline::jordanlines。

output 输出 Sparjor。更新 Sweepline::points，Sweepline::lines，Sweepline::jordanlines。

precondition 输入的 Sparjor 是殷集 A。A 的边界 Line 和 Point 都以经存储在 Sweepline::points,Sweepline::lines 中。

postcondition 输出的 Sparjor 表示的是 A 的补集。

算法实现

```
        vector<Line>& lines = Sweepline::lines;
        for(auto i = 0; i < jordanline.size(); i++){
                Jordanline jl = jordanline[i];
                int j = jl.size();
                vector<int> jl1(j);
                for(auto i = 0; i < jl.size(); i++){
                        jl1[j - i - 1] = jl[i];
                        int m = lines[jl[i]].[0];
                        lines[jl[i]].[0] = lines[jl[i]].[1];
                        lines[jl[i]].[1] = j;
                }
        }
```

证明 首先调换 A 边界上所有 lines 中存储的所有线段的方向，即遍历所有边并交换端点顺序。其次，还要调换殷集边界的边的存储顺序，因为 jordan 线是按线段方向存储的，所以要随着边的变向一起变向。通过对 Sweeplines::jordanlines 中所有 Jordanline::line 中的 Line 类 identity 的顺序换向，同时改变这些边的方向。

spadjor::meet(spadjor)

契约

input 输入两个 spadjor A 和 B。Sweepline::points，Sweepline::lines，Sweepline::jordanlines。

output 输出一个 spadjor C，并更新 Sweepline::points,Sweepline::lines，Sweepline::jordanlines 中的 Point 类,Line 类和 Jordanline 类的数据。

precondition A 和 B 的边界上的 Point 类,Line 类和 Jordanline 类都已经初始化并输入到 Sweepline::points，Sweepline::lines，Sweepline::jordanlines 中。

postcondition C 是 A,B 的交，Sweepline::points 添加新的交点，Sweepline::lines 添加新的线段 (不删除原有线段，只是删除所有调用方式，比如 Point::nearline 和原来的 Jordanline 类)，Sweepline::jordanlines 删除原有的 Jordanline 输入新的 Jordanline(因为原有的 jordan 线已经不是 Jordan 线了)。

算法实现

```
1  调用Sweepline::intersection()计算所有交点并更新points和lines内类的数据(
       例如Point::nearline,Line::leftline,Line::angle)。
2  遍历lines中所有Line l，若l.IfInOtherYinset为false将l.lp[]对应的Point p
       中删除p.nearline中代表l的数据，当IfInOtherYinset为true时跳过。
3  调用Line::generator()生成新的vector<Jordanline>替换原jordanlines。
4  遍历jordanlines中所有Jordanline类分别调用Jordanline::update()更新内部数
       据。
5  调用Jordanline::gennerator()生成一个spadjor类C，输出C。
```

证明 intersection() 算出了所有交点，并将新产生的有向线段都插入到 lines 中。因此 A 和 B 中所有边还在 lines 中或被划分为多条 Line(原 Line 的 Point 中的访问路径已被删除约等于懒惰删除，只保留了可能是 C 的边的所有 Line 类)。

遍历并利用 IfInOtherYinset 筛选边，由于 Line l 属于 C 的边当且仅当 l.IfInOtherYinset 为 true，所以删除所有 false 的边的 Point 中的访问路径后，当且仅当 Line l 是 C 的边界时才能从 Point 中访问 l。

Line::generator() 从现存的 Line 中生成一组只更新了 Jordanline::line 的 Jordanline 类的 vector<Jordanline>，每个 Jordanline 类代表一个 jordan 线而且两两之间没有不合适的交点。因此这组 jordan 线就是 C 的边界集合。

最后生成 spadjor 还需要 jordan 线之间的包含关系，因此从 line 中的 leftline 信息调用 Jordanline::update() 更新 leftjordanline 数据。调用 Jordanline::generator() 生成唯一的 spadjor 由 spadjor 和 Jordanline 的一一对应关系知所得的 spadjor 即是 C。

spadjor::join(spadjor)

契约

input 同 meet();

output 同 meet();

precondition 同 meet();

postcondition C 是 A 和 B 的交

算法实现

```
1  调用A.complement(),B.complement;
2  调用C = A.meet(B);
3  调用C.complement();
4  输出C。
```

证明 由布尔运算的交并补关系得。

friend operator«(ostream&, spadjor)

契约

input spadjor C。

output 点坐标输入到 ostream 中。

precondition C 是 Jordanline::generator() 生成的。

postcondition 首先输出 spadjor::jordanline.size(), 再依次输出 C 边界上的 jordan 线，即 jordanline 中存储的 Jordanline 类。输出 Jordanline 类时先输出 Jordanline::line.size() 再依次输出 line 中的 Line 类。输出 line 类时只输出起点 Point 的坐标。

算法实现

```
        for(auto i = 0; i < jordanline.size(); i++){
                Jordanline jl = jordanlines[i];
                os << jl;
        }
```

证明 依次调用符合条件的重载 operator« 操作符。

friend operator»(istream&, spadjor)

契约

input 首先输入读入殷集 Y 边界的 Jordan 线的数量，然后以 Jordan 线的方向依次读入 jordan 线上的点。

output spadjor 类，lines，points，jordanlines 中加入新的类。

precondition 输入的数据是一组 jordan 线上按 jordan 线方向输入的点的坐标，这些 jordan 线是 Y 的所有边界。

postcondition 输出的 spadjor 内的 hassmap 体现了所有 jordan 线的包含关系，jordanline 是 Y 的所有 jordan 线边界的 identity 的集合。

算法实现

```
        int i;
        is >> i;
        vector<int> jordanline(i);
        int k = 0;
        while(k < i){
                生成Point, Line, Jordanline并插入到Sweepline::points,Sweepline::
                    lines,Sweepline::jordanlines中。
                k++;
        }
```

证明

0.5.5 Sweepline

Sweepline::intersection()

契约

input vector<Point> points,vector<Line> lines,AVLTree<Point> eventpoint, AVLTree<Line> orderline1,orderline2,orderline3 ,AVLTree<int> existlines,AVLTree<int> existpoints。

output vector<Point> points,vector<Line> lines, AVLTree<int> existlines, AVLTree<int> existpoints。

precondition eventpoint,orderline1,orderline2,orderline3 都为空，existlines 和 existpoints 存储所有还存在于图上的 Line 和 Point。points，lines 存储所有从程序开始就产生的 Point 和 Line。

postcondition 计算出所有的交点，构造 Point 并插入到 points 中。交点划分线段，在 existlines 中删除被划分的 Line l0.identity 并插入新的两个 Line l1.identity, l2.identity。交点中 nearline，加入 l1 l2，原来的线段端点的 nearline 都删除 l0，分别插入 l1 或 l2。若出现两个点充分接近，合并成字典排序中较小的点，并且合并 nearline，更新 nearline 中所有线段的端点，从 existpoints 中删除被合并的 Point 的 identity。

算法实现

```
        for(auto i = 0; i < existpoints.size(); i++){
                eventpoint.add(points[existpoints[i]]);
        }
        AVLTreeNode<Line>* orderlinemarkp;
        AVLTreeNode<Line>* prev = NULL, next = NULL;
        AVLTreeNode<Point>* atnp = eventpoint.findfirstnode();
        while(true){
                prev = NULL;
                next = NULL;
                Point p = atnp->data;
                y = p[1];
                AVLTree<Point>* m = eventpoint.findnextnode(atnp),mt = NULL;
                while(m->data == p){
                        Point q = m->data;
                        AVLTree<pair<double,vector<int> >* z = q.nearline.
                            findfirstnode();
                        while(z != NULL){
                                vector<int> vi = ((z->data).second;
                                for(auto k = 0; k < vi.size(); k++){
                                        if(lines[vi[k]].[0] == q.identity())
                                                lines[vi[k]].[0] = p.identity();
                                        if(lines[vi[k]].[1] == q.identity())
                                                lines[vi[k]].[1] = p.identity();
```

```
23                                  points[p.identity()].addnearline(lines[
                                        vi[k]]);
24                              }
25                              z = q.nearline.findnextnode(z);
26                          }
27                          mt = m;
28                          existpoints.del(q.identity());
29                          m = eventpoint.findnextnode(m);
30                          eventpoint.del(mt->data);
31                      }
32                      AVLTree<pair<double,vector<int>>* atndv = p.nearline.
                            findfirstnode();
33                      while(true){
34                          vector<int> v = (atndv->data).second;
35                          for(auto i = 0; i < v.size(); i++){
36                              Line l = lines[v[i]];
37                              if(points[l[0]] < p || points[l[1]] < p){
38                                  if(orderlinemark.find(make_pair(l.
                                        identity(), NULL)) == NULL) cout << "
                                        orderlinemark wrong\n";
39                                  orderlinemarkp = orderlinemark.find(
                                        make_pair(l.identity(),NULL))->data.
                                        second;
40                                  prev = orderline.findprevnode(
                                        orderlinemarkp);
41                                  next = orderline.findnextnode(
                                        orderlinemarkp);
42                                  orderline.del(orderlinemarkp);
43                                  orderlinemark.del(make_pair(l.identity()
                                        ,orderlinemarkp));
44                              }
45                          }
46                          atndv = p.nearline.findnextnode(atndv);
47                          if(atndv == NULL) break;
48                          if((atndv->data).first < PI + tol/100) break;
49                      }
50                      if(atndv != NULL){
51                          vector<int> v = (atndv->data).second;
52                          Line l = lines[v[0]];
53                          if(prev == NULL && next == NULL){
54                              orderlinemarkp = orderline.add(l);
55                              prev = orderline.findprevnode(orderlinemarkp);
56                              next = orderline.findnextnode(orderlinemarkp);
```

```
57                              orderline.del(orderlinemarkp);
58                      }
59              }
60              while(next != NULL && (next->data).ifintheline(p)){
61                      Line nextline = lines[(next->data).identity()];
62                      existlines.del(nextline.identity());
63                      orderline.del(orderlinemark.find(make_pair(nextline.
                            identity(),NULL))->data.second);
64                      orderlinemark.del(make_pair(nextline.identity(),NULL));
65                      int k = lines.size();
66                      Line l1(nextline[0], p.identity(), k, nextline.inspadjor
                            ());
67                      lines.push_back(l1);
68                      k++;
69                      Line l2(p.identity(), nextline[1], k, nextline.inspadjor
                            ());
70                      lines.push_back(l2);
71                      points[p.identity()].addnearline(l1);
72                      points[p.identity()].addnearline(l2);
73                      points[nextline[0]].delnearline(nextline);
74                      points[nextline[0]].addnearline(l1);
75                      points[nextline[1]].delnearline(nextline);
76                      points[nextline[1]].addnearline(l2);
77                      next = orderline.findnextnode(next);
78              }
79              while(prev != NULL || (prev->data).ifintheline(p)){
80                      Line prevline = lines[(prev->data).identity()];
81                      existlines.del((prevline.identity());
82                      orderline.del(orderlinemark.find(make_pair(prevline.
                            identity(),NULL))->data.second);
83                      orderlinemark.del(make_pair(prevline.identity(),NULL));
84                      int k = lines.size();
85                      Line l1(prevline[0], p.identity(), k, prevline.inspadjor
                            ());
86                      lines.push_back(l1);
87                      k++;
88                      Line l2(p.identity(), prevline[1], k, prevline.inspadjor
                            ());
89                      lines.push_back(l2);
90                      points[p.identity()].addnearline(l1);
91                      points[p.identity()].addnearline(l2);
92                      points[prevline[0]].delnearline(prevline);
93                      points[prevline[0]].addnearline(l1);
```

```
                        points[prevline[1]].delnearline(prevline);
                        points[prevline[1]].addnearline(l2);
                        prev = orderline.findprevnode(prev);
                }
                atndv = p.nearline.findfirstnode();
                Line leftl,rightl;
                int g = 0;
                Line l;
                while(true){
                        vector<int> v = (atndv->data).second;
                        for(auto i = 0; i < v.size(); i++){
                                l = lines[v[i]];
                                if(points[l[0]] > p || points[l[1]] > p){
                                        if(g == 0){ leftl = l; g = 1;}
                                        orderlinemarkp = orderline.add(l);
                                        orderlinemark.add(make_pair(l.identity()
                                            ,orderlinemarkp));
                                }
                        }
                        atndv = p.nearline.findnextnode(atndv);
                        if(atndv == NULL) {rightl = l;break;}
                }
                if(g == 1){
                        if(prev != NULL) inter(prev->data,leftl,p);
                        if(next != NULL) inter(rightl,next->data,p);
                }
                else {
                        if(prev != NULL && next != NULL)
                        inter(prev->data,next->data,p);
                }
                atnp = eventpoint.findnextnode(atnp);
                if(atnp == NULL) break;
        }
```

证明

Sweepline::inter()

契约

input 输入两个 Line l1 l2 和当前 eventpoint p1 判断交点是否已处理。

output 根据 l1，l2 是否相交，更新 eventpoint ，orderline，lines，points，existpoints，existlines 内 Point.nearline 和 Line::lp 的数据等

precondition Line l1，l2 初始化并已在 lines，existlines 等集合中。

postcondition 若没有交点无变化，若有交点，更新相应的信息。

算法实现

```
        AVLTreeNode<Line>* orderlinemarkp;
        Line prevl = lines[l1.identity()], l = lines[l2.identity()];
        Point lp0,lp1,prevlp0,prevlp1;
        if(points[l[0]] < points[l[1]]) {
                lp0 = points[l[0]];lp1 = points[l[1]];
        }
        else {
                lp0 = points[l[1]];lp1 = points[l[0]];
        }
        if(points[prevl[0]] < points[prevl[1]]) {
                prevlp0 = points[prevl[0]];prevlp1 = points[prevl[1]];
        }
        else {
                prevlp0 = points[prevl[1]];prevlp1 = points[prevl[0]];
        }
        if(lp0 == prevlp0){
                AVLTree<pair<double,vector<int> >* z = lp0.nearline.
                   findfirstnode();
                while(z != NULL){
                        vector<int> vi = ((z->data).second;
                        for(auto k = 0; k < vi.size(); k++){
                                if(lines[vi[k]].[0] == lp0.identity())
                                lines[vi[k]].[0] = prevl0.identity();
                                if(lines[vi[k]].[1] == lp0.identity())
                                lines[vi[k]].[1] = prevl0.identity();
                                points[prevlp0.identity()].addnearline(lines[vi[
                                   k]]);
                        }
                        z = lp0.nearline.findnextnode(z);
                }
                existpoints.del(lp0.identity());
                eventpoint.del(lp0);
        }
        else if(prevl.ifintheline(lp0)){
                int k = lines.size();
                Line l1(prevl[0], lp0.identity(), k, prevl.inspadjor());
                lines.push_back(l1);
                k++;
```

```
37                  Line l2(lp0.identity(), prevl[1], k, prevl.inspadjor());
38                  lines.push_back(l2);
39                  existlines.del(prevl.identity());
40                  existlines.add(l1.identity());
41                  existlines.add(l2.identity());
42                  orderlinemarkp = orderlinemark.find(make_pair(prevl.identity(),
                        NULL))->data.second;
43                  if(prevl[0] < lp0) {
44                          orderlinemarkp->data = l1;
45                          orderlinemark.del(make_pair(prevl.identity(),NULL));
46                          orderlinemark.add(make_pair(l1.identity(),orderlinemarkp
                                ));
47                  }
48                  else {
49                          orderlinemarkp->data = l2;
50                          orderlinemark.del(make_pair(prevl.identity(),NULL));
51                          orderlinemark.add(make_pair(l2.identity(),orderlinemarkp
                                ));
52                  }
53                  points[lp0.identity()].addnearline(l1);
54                  points[lp0.identity()].addnearline(l2);
55                  points[prevl[0]].delnearline(prevl);
56                  points[prevl[0]].addnearline(l1);
57                  points[prevl[1]].delnearline(prevl);
58                  points[prevl[1]].addnearline(l2)
59          }
60          else if(l.ifintheline(prevlp0)){
61                  int k = lines.size();
62                  Line l1(l[0], prevlp0.identity(), k, l.inspadjor());
63                  lines.push_back(l1);
64                  k++;
65                  Line l2(prevlp0.identity(), l[1], k, l.inspadjor());
66                  lines.push_back(l2);
67                  existlines.del(l.identity());
68                  existlines.add(l1.identity());
69                  existlines.add(l2.identity());
70                  orderlinemarkp = orderlinemark.find(make_pair(l.identity(),NULL)
                        )->data.second;
71                  if(l[0] < prevlp0) {
72                          orderlinemarkp->data = l1;
73                          orderlinemark.del(make_pair(l.identity(),NULL));
74                          orderlinemark.add(make_pair(l1.identity(),orderlinemarkp
                                ));
```

```
75                  }
76                  else {
77                          orderlinemarkp->data = l2;
78                          orderlinemark.del(make_pair(l.identity(),NULL));
79                          orderlinemark.add(make_pair(l2.identity(),orderlinemarkp
                                ));
80                  }
81                  points[prevlp0.identity()].addnearline(l1);
82                  points[prevlp0.identity()].addnearline(l2);
83                  points[l[0]].delnearline(l);
84                  points[l[0]].addnearline(l1);
85                  points[l[1]].delnearline(l);
86                  points[l[1]].addnearline(l2);
87          }
88          else {
89                  if(l.ifintersection(prevl)) {
90                          Point p = prevl.intersectoin(l);
91                          if(p < p1) return;
92                          int k = points.size();
93                          p.setidentity(k);
94                          points.push_back(p);
95                          existpoints.add(k);
96                          k = lines.size();
97                          Line l1(p.identity(), prevl[1], k, prevl.inspadjor());k
                                ++;
98                          Line l2(prevl[0], p.identity(), k, prevl.inspadjor());k
                                ++;
99                          Line l3(l[0], p.identity(), k, l.inspadjor());k++;
100                         Line l4(p.identity(), l[1], k, l.inspadjor());
101                         lines.push_back(l1);lines.push_back(l2);lines.push_back(
                                l3);lines.push_back(l4);
102                         existlines.del(prevl.identity());
103                         existlines.del(l.identity());
104                         existlines.add(l1.identity());
105                         existlines.add(l2.identity());
106                         existlines.add(l3.identity());
107                         existliens.add(l4.identity());
108                         points[p.identity()].addnearline(l1);
109                         points[p.identity()].addnearline(l2);
110                         points[p.identity()].addnearline(l3);
111                         points[p.identity()].addnearline(l4);
112                         eventpoint.add(points[p.identity()]);
```

```
113                         orderlinemarkp = orderlinemark.find(make_pair(prevl.
                                identity(),NULL))->data.second;
114                         orderlinemark.del(make_pair(prevl.identity(),NULL));
115                         if(p > prevl[1]) {
116                                 orderlinemarkp->data = l1;
117                                 orderlinemark.add(make_pair(l1.identity(),
                                        orderlinemarkp));
118                         }
119                         else {
120                                 orderlinemarkp->data = l2;
121                                 orderlinemark.add(make_pair(l2.identity(),
                                        orderlinemarkp));
122                         }
123                         orderlinemarkp = orderlinemark.find(make_pair(l.identity
                                (),NULL))->data.second;
124                         orderlinemark.del(make_pair(l.identity(),NULL));
125                         if(p > l[0]) {
126                                 orderlinemarkp->data = l3;
127                                 orderlinemark.add(make_pair(l3.identity(),
                                        orderlinemarkp));
128                         }
129                         else {
130                                 orderlinemarkp->data = l4;
131                                 orderlinemark.add(make_pair(l4.identity(),
                                        orderlinemarkp));
132                         }
133                         points[l[0].identity()].delnearline(l);
134                         points[l[0].identity()].addnearline(l3);
135                         points[l[1].identity()].delnearline(l);
136                         points[l[1].identity()].addnearline(l4);
137                         points[prevl[0].identity()].delnearline(prevl);
138                         points[prevl[0].identity()].addnearline(l2);
139                         points[prevl[1].identity()].delnearline(prevl);
140                         points[prevl[1].identity()].addnearline(l1);
141                 }
142         }
```

证明

Sweepline::update(int, bool, int = -1, bool = true)

契约

input 输入 lines, points 等。算两个交点的 spadjor::identity int spadjor2,spadjor3, 和 type, spadjortype2, spadjortype3。还有一个标记 m, 是 meet 或 complement 决定是否更新 IfInOtherYinset。

output 更新 lines 中 Line 的 Line::leftline,Line::IfInOtherYinset。

precondition intersection() 函数调用后调用。

postcondition 设置 leftline 为线段下节点是在 orderline 中前一个 Line 的 identity。IfInOtherYinset 设置为它是否在另一个殷集内部。

算法实现

```
        for(auto i = 0; i < existpoints.size(); i++){
                eventpoint.add(points[existpoints[i]]);
        }
        int m = 0;
        if(spadjor3 != -1) m = 1;
        AVLTreeNode<Line>* orderlinemarkp;
        AVLTreeNode<Line>* prev = NULL;
        AVLTreeNode<Point>* atnp = eventpoint.findfirstnode();
        while(true){
                AVLTree<pair<double,vector<int>>* atndv = p.nearline.
                    findfirstnode();
                while(true){
                        vector<int> v = (atndv->data).second;
                        for(auto i = 0; i < v.size(); i++){
                                Line l = lines[v[i]];
                                if(points[l[0]] < p || points[l[1]] < p){
                                        if(orderlinemark.find(make_pair(l.
                                            identity(), NULL)) == NULL) cout << "
                                            orderlinemark wrong\n";
                                        orderlinemarkp = orderlinemark.find(
                                            make_pair(l.identity(),NULL))->data.
                                            second;
                                        prev = orderline.findprevnode(
                                            orderlinemarkp);
                                        if(prev != NULL) lines[l.identity()].
                                            setleftline(prev->data.identity());
                                        else lines[l.identity()].setleftline(-1)
                                            ;
                                        orderline.del(orderlinemarkp);
                                        orderlinemark.del(make_pair(l.identity()
                                            ,NULL));
                                        if(m == 1){
                                                if(l.inspadjor() != spadjor2) {
```

```
25                                                  orderline3.del(
                                                        orderlinemark3.find(
                                                        make_pair(l.identity
                                                        (),NULL))->data.
                                                        second);
26                                                  orderlinemark3.del(
                                                        make_pair(l.identity
                                                        (),NULL));
27                                                  orderlinemarkp =
                                                        orderline2.add(l);
28                                                  prev = orderline2.
                                                        findprevnode(
                                                        orderlinemarkp);
29                                                  if(prev == NULL) {
30                                                          lines[l.identity
                                                              ()].
                                                              setIfInOtherYinset
                                                              (spadjortype2
                                                              );
31                                                  }
32                                                  else {
33                                                          Point p0 =
                                                              points[(prev
                                                              ->data)[0]],
                                                              p1 = points[(
                                                              prev->data)
                                                              [1]];
34                                                          if(p0 < p1)
                                                              lines[l.
                                                              identity()].
                                                              setIfInOtherYinset
                                                              (true);
35                                                          else lines[l.
                                                              identity()].
                                                              setIfInOtherYinset
                                                              (false);
36                                                  }
37                                                  orderline2.del(
                                                        orderlinemarkp);
38                                          }
39                                          else {
40                                                  orderline2.del(
                                                        orderlinemark2.find(
```

```
                                                    make_pair(l.identity
                                                    (),NULL))->data.
                                                    second);
41                                                  orderlinemark2.del(
                                                    make_pair(l.identity
                                                    (),NULL));
42                                                  orderlinemarkp =
                                                    orderline3.add(l);
43                                                  prev = orderline3.
                                                    findprevnode(
                                                    orderlinemarkp);
44                                                  if(prev == NULL) {
45                                                          lines[l.identity
                                                              ()].
                                                              setIfInOtherYinset
                                                              (spadjortype3
                                                              );
46                                                  }
47                                                  else {
48                                                          Point p0 =
                                                              points[(prev
                                                              ->data)[0]],
                                                              p1 = points[(
                                                              prev->data)
                                                              [1]];
49                                                          if(p0 < p1)
                                                              lines[l.
                                                              identity()].
                                                              setIfInOtherYinset
                                                              (true);
50                                                          else lines[l.
                                                              identity()].
                                                              setIfInOtherYinset
                                                              (false);
51                                                  }
52                                                  orderline3.del(
                                                    orderlinemarkp);
53                                          }
54                                  }
55                          }
56                          if(points[l[0]] > p || points[l[1]] > p){
57                                  if(orderlinemark.find(make_pair(l.
                                      identity(), NULL)) != NULL) cout << "
```

```
                                        orderlinemark wrong\n";
58                                  orderlinemarkp = orderline.add(l);
59                                  orderlinemark.add(make_pair(l.identity()
                                        ,orderlinemarkp));
60                                  if(l.inspadjor() == spadjor2){
61                                          orderlinemarkp = orderline2.add(
                                                l);
62                                          if(m == 1) orderlinemark2.add(
                                                make_pair(l.identity(),
                                                orderlinemarkp));
63                                  }
64                                  else {
65                                          orderlinemarkp = orderline3.add(
                                                l);
66                                          if(m == 1) orderlinemark3.add(
                                                make_pair(l.identity(),
                                                orderlinemarkp));
67                                  }
68                              }
69                      }
70                      atndv = p.nearline.findnextnode(atndv);
71                      if(atndv == NULL) break;
72              }
73              atnp = eventpoint.findnextnode(atnp);
74              if(atnp == NULL) break;
75      }
76      if(m == 1){
77              AVLTreeNode<int>* pt = existlines.findfirstnode()
78              while(true){
79                      int t = pt->data;
80                      Line l = lines[t];
81                      AVLTreeNode<int>* markpt = existlines.findnextnode(pt);
82                      if(l.IfInOtherYinset() == false){
83                              existlines.del(t);
84                              Point p0 = points[l[0]], p1 = points[l[1]];
85                              p0.delnearline(l);
86                              p1.delnearline(l);
87                      }
88                      pt = markpt;
89                      if(pt == NULL) break;
90              }
91      }
```

证明

0.5.6 template<class T> Class AVLTree

以下的 n 都是指树节点的数量。即数据的数量

AVLTree::add(T)

契约

input 输入 T 类型数据，AVLTree<T> 树 a。

output 以 O(lgn) 的时间复杂度存入数据到 a。返回插入的树节点的指针。

precondition T 类型数据定义了重载 operator<, operator>。出现非全排序情况可能会内存泄露并且结果未定义。

postcondition 可以重复存储，以插入顺序排序。以 O(lgn) 的时间复杂度。

算法实现

证明

AVLTree::del(T)

契约

input T 类型数据 t。AVLTree<T> a。

output 以 O(lgn) 的时间复杂度删除二叉树 a 中值为 t 的树节点。

precondition T 类型。

postcondition 若二叉树 a 中存储了 t，删除这个树节点；若有多个 t，删除的是其中一个，不一定是先插入的或后插入的；若没有存储 t，返回 false。以 O(lgn) 的时间复杂度。

算法实现

证明

AVLTree::del(AVLTreeNode<T>*)

契约

input 一个树节点指针 p，和一棵树 tree

output tree 中删除 p。

precondition p 是 tree 中节点。

postcondition O(lgn) 时间完成，并且不破坏平衡二叉树和数据的排序。

算法实现

证明

AVLTree::find(T)

契约

input T 类型数据 t。AVLTree<T> a。

output AVLTreeNode<T>* pn,pn 指向 a 中一个 data 值等于 t 的树节点。

precondition

postcondition 若二叉树 a 中存储了 t，返回这个树节点；若有多个 t，返回的是其中一个，不一定是先插入的或后插入的；若没有存储 t，返回 NULL。以 O(lgn) 的时间复杂度。

算法实现

证明

AVLTree::findprevnode(AVLTreeNode<T>*)

契约

input 输入一个树节点指针 pn0，AVLTree<T> a。

output 返回一个树节点指针 pn1。

precondition 这个树节点指针是指向这棵树 a 中的节点

postcondition 输出 a 中所有数据按 operaotr< 排序中，存储着 pn0 代表的数据前一个数据的树节点 pn1。以 O(lgn) 的时间复杂度。若 pn0 存储的数据是最前面的数据，pn1 是 NULL。

算法实现

证明

AVLTree::findnextnode(AVLTreeNode<T>*)

与 findprevnode() 相反，这个是返回下一个数据，若已经是最后一个，返回 NULL。以 O(lgn) 的时间复杂度。

AVLTree::findfirstnode()

契约

input 一个 AVLTree<T> a。

output 一个 AVLTreeNode<T>* pn。

precondition

postcondition 指针 pn 是存储着 a 中所有数据中最小的数据的树节点的指针，若 a 中没有数据，返回 NULL。以 O(lgn) 的时间复杂度。

AVLTree::findlastnode()

与 findfirstnode() 相反，返回最后一个数据的树节点的指针或 NULL。