北京亦庄实验中学

研究性学习课程
结题报告

课题名称 对凸包与半平面交问题的研究

课题组成员　蔡越同 尹玉文东 李灏冬 张钰晨

课题组组长　蔡越同

指 导 教 师　苏虹宇

管 理 教 师　王 千

研 究 领 域　数学 信息学

2022 年 1 月 21 日

对凸包与半平面交问题的研究

尹玉文东，蔡越同，李灏冬，张钰晨

【摘要】 方法 使用 C++语言编写计算几何工具库，利用其研究凸包和半平面交问题的解法。之后借助洛谷网等算法竞赛[1]相关资源网站，总结这两类问题的应用场景。 **结果** 介绍了 Andrew 算法和 Sort-and-Incremental 算法，简述了两种算法的思路与时间复杂度。通过对算法竞赛中相关典例的分析，指出了两类问题在算法竞赛和生活中的应用。
【关键词】 凸包；半平面交；计算几何

Convex Hull and Half-plane Intersection

[Abstract] Methods Using C++ language to program computational geometry tool library, using it to study convex hull and half-plane intersection problem. Then, with the help of resource websites related to algorithm competitions such as Luogu.com, the application scenarios of these two types of problems are summarized. **Results** The Andrew algorithm and the Sort-and-Incremental algorithm are introduced, and the ideas and time complexity of the two algorithms are briefly described. Through the analysis of the relevant examples in algorithm competition, the application of the two types of problems in algorithm competition and life is pointed out.
[Key words] Convex hull; Half-plane intersection; Computational geometry

1. 引言

随着信息技术的大范围普及，算法竞赛的影响力与日俱增。计算几何在算法竞赛中有着重要应用，凸包和半平面交是计算几何中的经典问题。

计算几何是基于向量运算的体系。相较于解析几何，计算几何能有效控制计算机运算中的浮点数精度问题，更适用于算法竞赛。

作为算法竞赛中较为重要的模块，相关资料整理却不甚完全。我组决定借此机会整合资料，并对此类算法和问题进行学习研究，便于后续使用。

2. 计算几何库及有关说明

本课题组使用 C++语言编写了计算几何工具库，并利用其研究了这两类问题的解法，随后分析并提取出部分典型模型。

本文的研究均在平面体系下进行，若无特殊说明，两向量叉乘默认代表结果向量的模长。本文的参考代码可能直接调用库中函数。

在本文的**附件一：计算几何库**部分，包含了计算几何库的代码。

下面是工具库的功能及部分原理解释。

2.1. 浮点数相关函数

为了避免浮点数运算造成的精度问题，编写了相关函数用于控制精度。

[1] 洛谷网：https://www.luogu.com.cn；其它资源网站包括：http://poj.org，https://www.spoj.com.

包含功能：判断数的正负、比较两个数的大小。

2.2. 向量类

向量类用于解决二维向量的相关运算问题，包含下列功能。

a) 向量的加法、减法、数乘、点积、叉积（不需要坐标表示，只考虑模长及方向）；
b) 求向量的模长；
c) 求两向量之间的夹角；
d) 对一组向量按照字典序（坐标序）或极角排序。

2.3. 直线类

对于直线上任意两点 $A(x_a, y_a)$，$B(x_b, y_b)$ 构造坐标对应的向量 $\vec{s} = (x_a, y_a)$，$\vec{t} = (x_b, y_b)$，直线类利用向量类，记录这两个向量 $\vec{s}$ 和 $\vec{t}$。

包含下列功能：

a) 利用直线上两点构造直线；
b) 判断两直线是否平行；
c) 求两直线交点。

2.4. 极角排序的原理

取笛卡尔坐标系中的 x 轴作为极轴，以逆时针方向为正。

利用 C++语言中的 std::atan2(double y, double x)进行极角排序，函数的返回值为 (x, y) 与 x 轴的极角 $\theta \in (-\pi, \pi]$.

2.5. 求两直线交点的原理

首先判断直线是否平行，若平行则不存在交点，否则：

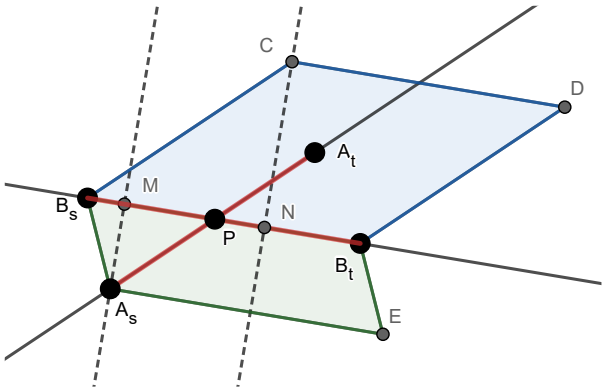

图 1

如**图 1**，直线 A 为 (A_s, A_t)，直线 B 为 (B_s, B_t)，求 A 与 B 的交点。

函数只需求出 $t = \frac{|A_sP|}{|A_sA_t|}$，则交点可以表示为 $\vec{P} = \overrightarrow{A_s} + t \cdot \overrightarrow{A_sA_t}$，下面求 t 的值。

先给出结论，$t = \frac{|A_sP|}{|A_sA_t|} = \frac{|\overrightarrow{B_sB_t} \times \overrightarrow{B_sA_s}|}{|\overrightarrow{B_sB_t} \times \overrightarrow{A_sA_t}|}$，下面给出证明。

把 $\overrightarrow{A_sA_t}$ 平移到 $\overrightarrow{B_sC}$ 的位置，则可以把叉乘的模看作面积，即 $t = \frac{\text{Area}(B_sB_tEA_s)}{\text{Area}(B_sB_tDC)}$。

记 $S_1 = |\overrightarrow{B_sB_t} \times \overrightarrow{B_sA_s}|$，$S_2 = |\overrightarrow{B_sB_t} \times \overrightarrow{A_sA_t}|$，两个平行四边形同底，所以有 $\frac{S_1}{S_2} = \frac{|A_sM|}{|CN|}$.

又因为 $\triangle A_SMP \sim \triangle CNB_S$，所以 $t = \frac{S_1}{S_2} = \frac{|A_SM|}{|CN|} = \frac{|A_SP|}{|CB_S|} = \frac{|A_SP|}{|A_SA_t|}$.

3. 凸包

3.1. 凸包

一个子集 S 被称为凸的，当且仅当对于任意的两点 $p,q \in S$，线段 $\overline{pq}$ 都完全属于 S。（如**图 2**[1]）

集合 S 的凸包 $\mathcal{CH}(S)$，是包含 S 的最小凸集，也就是包含 S 的所有凸集的交。

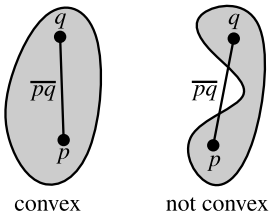

图 2[1]

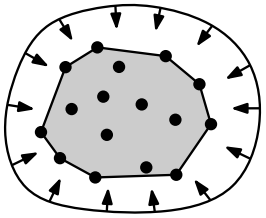

图 3[1]

如**图 3**[1]，不严谨地表示，凸包是平面上能包含所有给定点的最小凸多边形。

3.2. 求解方式 1——基础算法

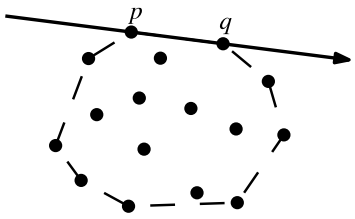

图 4[1]

注意到，任取一条线段 $\overline{pq}$，若其在凸包上，则点集 P 中的点均位于直线 $\overline{pq}$ 的同一侧（如**图 4**[1]）。

若我们钦定 $p \to q$ 按顺时针方向，则有更强的限制，需要 P 中的点都在直线的右侧。

考虑向量的叉积，点 t 在 $\overline{pq}$ 右侧 $\Leftrightarrow \overrightarrow{pt} \times \overrightarrow{pq} > 0$。于是可以枚举所有有序点对 $(p,q) \in P \times P$，若 P 中的点都在有向线段 $\overline{pq}$ 的右侧，则 $\overline{pq}$ 是 $\mathcal{CH}(P)$ 中的一条边。

此算法时间复杂度为 $\mathcal{O}(n^3)$。其中 n 为点数。

3.3. 求解方式 2——Andrew 算法

首先把所有点排序，以横坐标为第一关键字，纵坐标为第二关键字。排序后，第一个点和末尾的点，一定在凸包上，容易通过反证法证明。

从左往右看，上下凸壳斜率的单调性相反，即所旋转的方向不同，所以要分开求解。升序枚举求出下凸壳，然后降序枚举求出上凸壳，这样凸包的每条边都是向逆时针方向旋转的。

设当前枚举到点 P，即将把其加入凸包；当前栈顶的点为 S_1，栈中第二个点为 S_2。

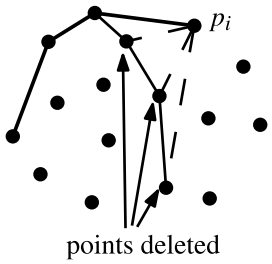

图 5[1]

如**图 5**[1]，求凸包时，若 P 与 S_1 构成的新线段是顺时针旋转的，即叉积满足：$\overrightarrow{S_2S_1} \times \overrightarrow{S_1P} < 0$，则弹出栈顶，继续检查，直到 $\overrightarrow{S_2S_1} \times \overrightarrow{S_1P} \geq 0$ 或者栈内仅剩一个元素为止。

记 $n = |P|$，则时间复杂度为 $\mathcal{O}(n\log n)$，瓶颈在排序部分。

相较于基础算法，Andrew 算法的效率大幅提升。

3.4. 代码实现

Convex_hull_2d 是求平面凸包的函数，需要传入三个参数。

其中 int n 代表点集大小，Point p[]是传入的点集，Point ret[]是用于记录凸包的点集。

```
inline bool check(Point s1, Point s2, Point p) {
    return Vec(s2, s1) * Vec(s1, p) > 0;
}
int Convex_hull_2d(int n, Point p[], Point ret[]) {
    sort(p, p + n, cmp1);
    int top = -1;
    for (int i = 0; i < n; i++) {
        while (top > 0 && !check(ret[top], ret[top - 1], p[i]))
            top--;
        ret[++top] = p[i];
    }
    int k = top;
    for (int i = n - 2; i >= 0; i--) {
        while (top > k && !check(ret[top], ret[top - 1], p[i]))
            top--;
        ret[++top] = p[i];
    }
    return top;
}
```

3.5. 在算法竞赛中的应用

3.5.1. 求覆盖点集的最小凸多边形周长

此类问题即可转化为求凸包周长。

先求出按照顺时针排序的，构成凸包的点集 p，记 $n = |p|$。再把相邻两点组成的向量的模长求和，即：

$$l = \sum_{i=1}^{n} |\overrightarrow{p_i p_{i+1}}| + |\overrightarrow{p_1 p_n}|$$

代码实现如下：

```
double dis(Point a, Point b) {
    return (a - b).len();
}
double Convex_hull_2d_L(int n, Point *p) {
    Point convex[N];
    int siz = Convex_hull_2d(n, p, convex);
    double ans = dis(convex[0], convex[siz - 1]);
    for (int i = 1; i < siz; i++)
        ans += dis(convex[i - 1], convex[i]);
    return ans;
}
```

3.5.2. 求覆盖点集的最小凸多边形面积

此类问题即可转化为求凸包面积。

先求出按照顺时针排序的，构成凸包的点集 p，记 $n = |p|$。任取凸包内一点（一般取 p_1），则有：

$$s = \sum_{i=2}^{n-1} \text{Area}\,(p_1, p_i, p_i + 1) = \sum_{i=2}^{n-1} \frac{|(p_i - p_1) \times (p_{i+1} - p_1)|}{2}$$

代码实现如下：

```
double area(Point a, Point b, Point c) {
    return (b - a) * (c - a) / 2.0;
}
double Convex_hull_2d_S(int n, Point *p) {
    Point convex[N];
    int siz = Convex_hull_2d(n, p, convex);
    double ans = 0;
    for (int i = 2; i < siz; i++)
        ans += area(convex[0], convex[i - 1], convex[i]);
    return ans;
}
```

3.5.3. 动态维护凸包

动态支持操作：加入一个顶点并更新凸包、查询一个坐标是否位于凸包内部。

对于加入操作，一个点维护凸包内，当且仅当它在上凸包的“下面”，且在下凸包的“上面”，可以利用叉积判断。

对于查询操作，可以先找到插入点上（下）凸包中的位置，向左右分别删点，直到满足凸性之后，再插入这个点到凸包中。

利用 C++中的 std::set，则每次插入和查询操作的时间复杂度均为 $\mathcal{O}(n\log n)$，代码见**附件二：动态维护凸包的 C++语言实现**。

3.6. 在生活中的应用

3.6.1. 包围盒算法

其广泛应用于与游戏物理引擎的制作中。

游戏过程中，游戏中的元素有时会发生碰撞，且这些元素的形状通常不规则。如果严谨地对不规则元素进行碰撞检测，会使得游戏效率低下。由此产生了一个近似算法，即包围盒算法。

其中包围盒指能完整覆盖元素的矩形。在游戏应用中为了使图形拟合更准确以及更简单地对游戏元素进行操作，通常会将以最小包围盒代替游戏中的不规则元素。

此问题可转换为凸包问题进行求解。

3.6.2. 碰撞检测及避免

其广泛应用于 3D 游戏中。这可以使人物在场景中可以平滑移动、遇到一定高度的障碍物可以越过而遇到过高的障碍物则停下、在各种前进方向被挡住的情况下都要尽可能地让人物沿合理的方向滑动而不是被迫停下。在满足这些要求的同时还可以做到足够精确和稳定。

在这个算法中，三维凸包可以用于找出人物轮廓，并简化后续计算及判断过程。

4. 半平面交

4.1. 相关定义

4.1.1. 半平面

半平面是一条直线和直线的一侧，是一个点集。

当点集包含直线时，称为闭半平面；当不包含直线时，称为开半平面。

4.1.2. 半平面交

半平面交是多个半平面的交集。

半平面交是一个点集，并且是一个凸集。在直角坐标系上是一个区域。

半平面交在代数意义下，是若干个线性约束条件，每个约束条件形如：

$$a_i x + b_i y \leq c_i$$

其中 a_i, b_i, c_i 为常数，且 a_i 和 b_i 不都为零。

如**图 6**[1]，半平面交有下列五种情况：（用灰色阴影表示直线所代表的半平面）

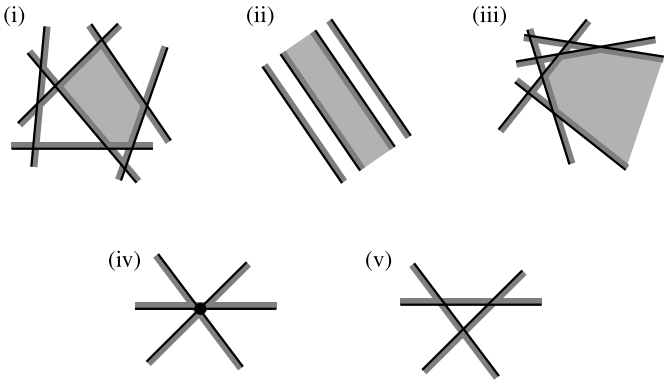

图 6[1]

4.2. 求解方式——Sort and Incremental 算法

4.2.1. 算法思路

S&I 算法利用了性质：半平面交是凸集。

因为半平面交是凸集，所以我们维护凸壳。

若我们把半平面按极角排序，那么在过程中，只可能删除队首或队尾的元素，因此使用双端队列维护。

4.2.2. 算法流程

a) 把半平面按照极角序排序。这需要 $O(n\log n)$ 的时间复杂度。

b) 对每个半平面，执行一次增量过程，每次根据需要弹出双端队列的头部或尾部元素。这是线性的，因为每个半平面只会被增加或删除一次。

c) 通过双端队列中的半平面，在线性时间内求出半平面交（一个凸多边形，用若干顶点描述）。

这样，我们得到了一个时间复杂度为 $O(n\log n)$ 的算法，瓶颈在于排序。若题目按照极角序给出半平面限制，则我们可以在线性的时间内求出半平面交。

4.3. 代码实现

实现过程中，用直线类表示半平面。直线 (S,T) 表示 ST 左侧的半平面。

形式化地，若点 P 位于直线 (S,T) 表示的半平面中，则 $\overrightarrow{ST}\times\overrightarrow{SP}>0$.

```
bool Halfplane_intersection(int n, Line *hp, Point *p) {
    if(n < 3) return false;
    sort(hp, hp + n, cmp2);
    Halfplane_unique(n, hp);
    st = 0; ed = 1;
    que[0] = 0; que[1] = 1;
    if(parallel(hp[0], hp[1])) return false;
    Calc_intersection(hp[0], hp[1], p[1]);
    for(int i = 2; i < n; i++) {
        while(st < ed &&
              sgn((hp[i].t - hp[i].s) * (p[ed] - hp[i].s)) < 0)
            ed--;
        while(st < ed &&
              sgn((hp[i].t - hp[i].s) * (p[st + 1] - hp[i].s)) < 0)
            st++;
        que[++ed] = i;
        assert(ed >= 1);
        if(parallel(hp[i], hp[que[ed - 1]])) return false;
        Calc_intersection(hp[i], hp[que[ed - 1]], p[ed]);
    }
    while(st < ed &&
          sgn((hp[que[st]].t - hp[que[st]].s)*(p[ed] - hp[que[st]].s)) < 0)
        ed--;
    while(st < ed &&
```

```
            sgn((hp[que[ed]].t - hp[que[ed]].s)*(p[st + 1] - hp[que[ed]].s)) < 0)
            st++;
        if(st + 1 >= ed) return false;
        return true;
    }
```

4.4. 在生活中的应用

4.4.1. 赛车问题

有一种赛车比赛，若一辆车在比赛过程中存在一个时刻处于第一位上，即可获奖。

为简化问题，视每辆赛车都在无限长的直线赛道上匀速行驶，则每辆车的 $x-t$ 图像都是一条直线。

若通过求每两条直线的交点，进而解决此问题，时间复杂度为 $O(n^2)$，耗时费力。

由图像的几何意义，若取每条直线的左侧半平面，再与 x 轴以上、y 轴以右一起求交，则所有凸多边形上的直线可以“获奖”。

问题转化为半平面交问题，时间复杂度为 $O(n\log n)$。

4.4.2. 光的传播问题

此类问题指给定若干障碍物，并规定光的传播方向，求哪些障碍物是可见的。

最基础的模型为所有障碍物均是直线，即：若在 x-y 直角坐标平面上有 n 条直线 $L_1, L_2, \dots L_n$，若在 y 值为正无穷大处往下看，求出所有可见的直线每条直线 L_i 均可表示为 $y=kx+b$ 的形式。

这个模型可以转化为半平面交问题，利用凸性求解。

5. 结语

本文对凸包与半平面交两类问题在算法竞赛及生活中的应用进行了研究。除此之外，这两类问题还在其他多个领域有所应用，特别是图像处理相关行业。前人对于此类问题多集中于学术化的研究，本研究组附加了在生活中的实际应用作为补充。

对凸包和半平面交问题的典例整理，可以作为他人学习时的参考资料，使人们在应对算法竞赛中的相关问题时，有更灵活的方法。

6. 参考资料

[1] Berg M , Kreveld M , Overmars M H . Computational Geometry: Algorithms and Applications[M]. 2008.
Berg M , Kreveld M , Overmars M H . Computational Geometry: Algorithms and Applications[M]. 2008.
[2]Thomas,H.Cormen,Charles,E.Leiserson,Ronald,L.Rivest,Clifford,Stein, 殷建平,徐云,王刚,刘晓光,苏明,邹恒明,王宏志. 算法导论(原书第 3 版)[J]. 计算机教育, 2013(10):1.
[3]林厚从.信息学奥赛之数学一本通[M].南京:东南大学出版 社,2016:203-297.
[4]人民教育出版社 课程教材研究所 中学数学教材实验研究组.普通 高中教科书 数学（B 版）必修 第二册[M].北京:人民教育出版社,2019:131-172.

7. 附件一：计算几何库

```
const double pi = 3.1415926535898;
const double eps = 1e-8;
inline int sgn(double x) {
    if(fabs(x) < eps) return 0;
    else return x > 0 ? 1 : -1;
}
inline int fcmp(double x, double y) {
    if(fabs(x - y) < eps) return 0;
    else return x > y ? 1 : -1;
}
struct Point{
    double x, y;
    Point(){};
    Point(double a, double b): x(a), y(b) {}
    Point(Point a, Point b): x(b.x - a.x), y(b.y - a.y) {}
    friend Point operator + (const Point &a, const Point &b) {
        return Point(a.x + b.x, a.y + b.y);
    }
    friend Point operator - (const Point &a, const Point &b) {
        return Point(a.x - b.x, a.y - b.y);
    }
    friend bool operator == (const Point &a, const Point &b) {
        return fcmp(a.x, b.x) == 0 && fcmp(a.y, b.y == 0);
    }
    friend Point operator * (const double &a, const Point &b) {
        return Point(a * b.x, a * b.y);
    }
    friend Point operator * (const Point &a, const double &b) {
        return Point(a.x * b, a.y * b);
    }
    friend double operator * (const Point &a, const Point &b) {
        return a.x * b.y - a.y * b.x;
    }
    friend double operator & (const Point &a, const Point &b) {
        return a.x * b.x + a.y * b.y;
    }
    friend Point operator / (const Point &a, const double &b) {
        return Point(a.x / b, a.y / b);
    }
    friend bool operator < (const Point &a, const Point &b) {
        return (a.x < b.x) || (a.x == b.x && a.y < b.y);
    }
```

```
    inline double len() {
        return sqrt(1.0 * x * x + 1.0 * y * y);
    }
    friend double area(Point &a, Point &b, Point &c) {
        return (b - a) * (c - a) / 2.0;
    }
};
typedef Point Vec;

inline bool cmp1(Vec a, Vec b) {
    return (fcmp(a.x, b.x) == -1) ||
           (!fcmp(a.x, b.x) && fcmp(a.y, b.y) == -1);
}
bool cmp2(Point a, Point b) {
   if(atan2(a.y, a.x) - atan2(b.y, b.x) == 0)
      return a.x < b.x;
   return atan2(a.y, a.x) < atan2(b.y, b.x);
}
struct Line{
    Point s, t;
    Line() {};
    Line(Point a, Point b) : s(a), t(b) {}
    double ang() { return atan2((t - s).y, (t - s).x); }
    friend bool parallel(const Line &A, const Line &B) {
        return sgn((A.s - A.t) * (B.s - B.t)) == 0;
    }
    friend bool Calc_intersection(const Line &A, const Line &B, Point &res) {
        if(parallel(A, B)) return false;
        double s1 = (B.t - B.s) * (B.s - A.s);
        double s2 = (B.t - B.s) * (A.t - A.s);
        res = A.s + (A.t - A.s) * (s1 / s2);
        return true;
    }
};
```

8. 附件二：动态维护凸包的 C++语言实现

```
bool check_top(int x, int y) {
    auto k = top.lower_bound(x);
    if(k == top.end())
        return false;
    if(k -> first == x)
        return y <= k->second;
    if(k == top.begin()) return false;
    auto j = k; j--;
    return Point(k->first - x, k->second - y)
           Point(j->first - x, j->second - y) >= 0;
}
bool check_down(int x, int y) {
    auto k = down.lower_bound(x);
    if(k == down.end())
        return false;
    if(k -> first == x)
        return y >= k->second;
    if(k == down.begin()) return false;
    auto j = k; j--;
    return Point(k->first - x, k->second - y) *
           Point(j->first - x, j->second - y) <= 0;
}
void insert_top(int x, int y) {
    if(check_top(x, y)) return;
    top[x] = y;
    auto it = top.find(x);
    auto jt = it;
    if(it != top.begin()) { //remove left
        jt--;
        while(remove_top(jt++)) jt--;
    }
    if(++jt != top.end()) { //remove right
        while(remove_top(jt--)) jt++;
    }
}
void insert_down(int x, int y) {
    if(check_down(x, y)) return;
    down[x] = y;
    auto it = down.find(x);
    auto jt = it;
    if(it != down.begin()) { //remove left
        jt--;
```

```
            while(remove_down(jt++)) jt--;
        }
        if(++jt != down.end()) { //remove right
            while(remove_down(jt--)) jt++;
        }
    }
    void insert_top(int x, int y) {
        if(check_top(x, y)) return;
        top[x] = y;
        auto it = top.find(x);
        auto jt = it;
        if(it != top.begin()) { //remove left
            jt--;
            while(remove_top(jt++)) jt--;
        }
        if(++jt != top.end()) { //remove right
            while(remove_top(jt--)) jt++;
        }
    }
    void insert_down(int x, int y) {
        if(check_down(x, y)) return;
        down[x] = y;
        auto it = down.find(x);
        auto jt = it;
        if(it != down.begin()) { //remove left
            jt--;
            while(remove_down(jt++)) jt--;
        }
        if(++jt != down.end()) { //remove right
            while(remove_down(jt--)) jt++;
        }
    }
```