OI 计算几何 Part 2 二维凸包与单调性优化

Computational Geometry in OI

daklqw

Zhenhai High School

July 31, 2024

Outline

1 凸包

2 凸包上操作

3 凸包与数据结构

凸包

Definition (Convex Hull)

给定 n 个点 $S = \{x_1, x_2, \dots, x_n\}$，其构成的凸包为

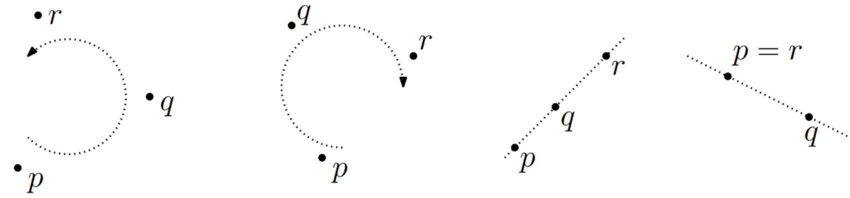
$$\mathrm{Conv}(S) = \{\alpha_1 x_1 + \alpha_2 x_2 + \cdots + \alpha_n x_n : \alpha_1 + \alpha_2 + \dots \alpha_n = 1, \alpha_i \geq 0\}.$$

也就是包含点集的最小凸多边形。一般按逆时针顺序存下凸包边界上的顶点。

定向

逆时针为正向，对应着叉积为正。

$\mathrm{orient}(p, q, r) > 0$ $\mathrm{orient}(p, q, r) < 0$ $\mathrm{orient}(p, q, r) = 0$



Graham 扫描法

Algorithm (Graham's Scan)

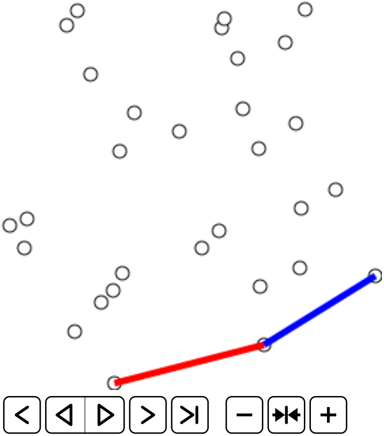
选择一个必在凸包上的点 v，此时存在一条直线，使得剩下点都在这条直线的同一侧，这样好做极角排序。

令 $S = \{v\}$，按照极角序将点一个个加入集合 S，同时实时维护 S 的凸包。

将凸包的顶点按极角序存储，那么每当新点 w 加入 S，w 会导致凸包顶点序列的一段后缀从凸包中移除。

即这个凸包是个单调栈：一旦在序列末尾追加点 w 会导致最后三个点不是逆时针的（即凸的），说明序列末尾的点再也不会出现在后来的凸包里。

Graham 扫描法



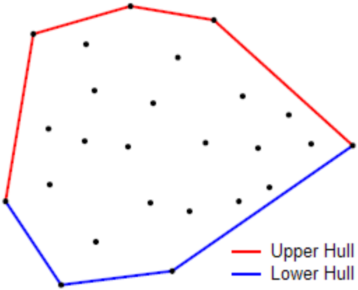
Graham 扫描法

```cpp
// Assumptions:
// p :: std::vector<vec2d> // points
// p[0] is the bottommost and leftmost point
// c :: std::vector<vec2d> // convex hull
std::sort(p.begin() + 1, p.end(), polar_cmp);
for (vec2d w : p) {
  while (c.size() >= 2 && \
         cross(c[c.size() - 2], c.back(), w) <= 0) {
    c.pop_back();
  }
  c.push_back(w);
}
```

Andrew 算法

Algorithm (Andrew)

采用更加直接的排序：以 x 为第一关键字，y 为第二关键字排序。
此时如果使用单调栈可以分别得出凸包的上凸壳和下凸壳。



Codeforces 某一道题，但是我找不到了

Example

给定 n 个点，对于每个点询问，如果删去这个点，剩下的点集凸包会有多少点。

$n \le 2 \times 10^5$。

Algorithm

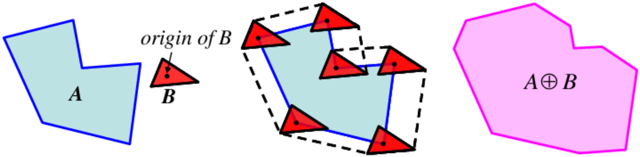
发现删除凸包里的点完全没有影响。删除凸包的顶点后，其相邻顶点仍在凸包上。
因此对凸包的定点进行奇偶标号：建两个凸包，一个是点集删去所有奇数点的，一个是删去所有偶数点的。
然后就可以根据两个凸包的结果分别查询出来答案。

Minkowski 和

Definition (Minkowski Sum)

给定位置向量（即原点到点 P 的向量）的集合 A, B，则 A 和 B 的 Minkowski 和定义为：

$$A + B = \{a + b : a \in A, b \in B\}.$$



(a) (b) (c)

凸包的 Minkowski 和

Example

给定两个凸包 A, B，求 $A + B$。

Proposition

两个凸集的 *Minkowski* 和是凸的。

Proof.

$\forall x, y \in A + B, \lambda \in [0, 1]$，假设 $x = a_x + b_x, y = a_y + b_y$，则
$\lambda x + (1 - \lambda)y = (\lambda a_x + (1 - \lambda)a_y) + (\lambda b_x + (1 - \lambda)b_y) \in A + B$。 □

凸包的 Minkowski 和

Proposition

凸包的 *Minkowski* 和为凸包顶点 *Minkowski* 和的凸包。即 $\mathrm{Conv}(A+B)=\mathrm{Conv}(A)+\mathrm{Conv}(B)$。

Proof.

凸包顶点 Minkowski 和为 A_i+B_j 的形式，所以 $\mathrm{Conv}(A+B)$ $\subseteq \mathrm{Conv}(A)+\mathrm{Conv}(B)$。
对于凸包的 Minkowski 和，其内部点为 $\lambda_1 A_1+\lambda_2 A_2+\cdots+\lambda_n A_n$ $+\gamma_1 B_1+\gamma_2 B_2+\cdots+\gamma_m B_m$ 的形式。

凸包的 Minkowski 和

Proof (Cont.)

构造如下点集 C_i：维护 $i \in [n], j \in [m]$，起初 $i = j = 1$。将 $p = A_i + B_j \in A + B$ 加入点集，并将 λ_i 和 γ_j 共同减去 $w_p = \min(\lambda_i, \gamma_j)$。如果 $\lambda_i = 0$ 则令 i 加一，如果 $\gamma_j = 0$ 则令 j 加一。
因为 $\sum \lambda_i = \sum \gamma_j = 1$，如此构造得到的点集是一组 $A + B$ 点的线性组合，并且 $\sum w_p = 1$。
所以有 $\mathrm{Conv}(A) + \mathrm{Conv}(B) \subseteq \mathrm{Conv}(A + B)$。 □

凸包的 Minkowski 和

Proposition

假设凸包的边集按顺序为 $a_1, a_2, \ldots, a_n$ 以及 $b_1, b_2, \ldots, b_m$，并且 A, B 的顶点 $A_i = A_1 + a_1 + a_2 + \cdots + a_{i-1}, B_i = B_1 + b_1 + b_2 + \ldots b_{i-1}$，则 $A + B = C$ 的顶点满足：

$$\begin{cases} C_1 = A_1 + B_1, \\ C_i = C_1 + c_1 + c_2 + \cdots + c_{i-1}. \end{cases}$$

其中 $c_1, c_2, \ldots, c_{n+m}$ 为 $a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_m$ 经过极角排序后的结果。

凸包的 Minkowski 和

Proof.

对于凸包 A, B，$A+B$ 的顶点一定是 A_i+B_j 的形式，其中 A_i 和 B_j 都是对应凸包上的顶点。

$A+B$ 上的相邻点一定是 A_i+B_j 和 $A_i+B_{j'}$ 或者 $A_{i'}+B_j$ 的形式。

这两点使用反证法都可以得到。因此 $A+B$ 只可能有 A 和 B 上的边。

同时因为 $A+B$ 的边斜率单调，因此只会有 $n+m$ 条边。 □

凸包的 Minkowski 和

Proposition

凸包的 *Minkowski* 和可以在 $O(n)$ 内完成。

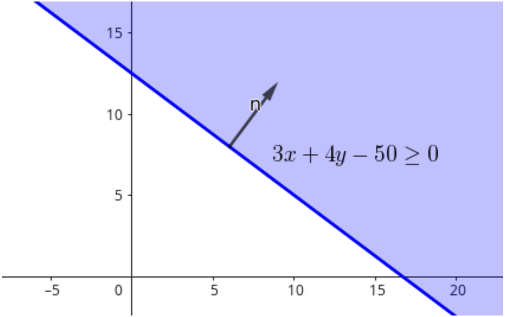
对凸包 A, B 的边进行归并排序，即可得到 $A + B$。

半平面

Definition

直线法向量一侧的区域为半平面。半平面的式子为:

$$Ax + By + C \geq 0.$$



半平面交

Proposition

有限个半平面的交是凸集。

Proof.

一个点 u 在半平面交内当且仅当 $\forall i, \langle u, n_i \rangle + c_i \geq 0$。
此时，若 u, v 都在半平面交内 $\lambda u + (1-\lambda)v$ 也在半平面交内，因为 $\langle \lambda u + (1-\lambda)v, n_i \rangle + c_i \geq 0$。所以半平面交为凸集。 $\square$

半平面交

Algorithm

类似凸包，将半平面按照极角排序并一个个加入，然后使用双端队列维护在半平面交里的直线。同时维护半平面交带来的凸包。
改变交集的半平面减小了凸包的大小。想象将凸包切了一刀，则若干边会被完全舍去，至多两条边会被砍掉半个。即加入这个半平面后，将队列首尾的若干半平面弹出，对应着被完全砍去的边。
实际为了方便实现，一般忽略最新加入的半平面是否真正切到了这个凸包，选择在最后处理这种情况。

半平面交

```cpp
bool check(hp a, hp b, hp c) {
  vec2d p = intersection(b, c);
  return dot(p, a.n) + a.C >= 0;
}
std::sort(half_planes.begin(), half_planes.end(), polar_cmp);
for (hp v : half_planes) {
  while (dq.size() >= 2 && !check(v, dq[-1], dq[-2])) dq.pop_back();
  while (dq.size() >= 2 && !check(v, dq[0], dq[1])) dq.pop_front();
}
while (dq.size() >= 2 && !check(dq[0], dq[-1], dq[-2])) dq.pop_back();
while (dq.size() >= 2 && !check(dq[-1], dq[0], dq[1])) dq.pop_front();
```

注：我没写过双端队列的半平面交，所以建议大家再去网上找找板子，我这个不保证细节正确。

点线对偶

Proposition (Point-Line Duality)

点	线
$P:(A,B)$	$\ell: Ax+By+1=0$
• 两点确定一条直线	• 两条直线交于一点
• 三点共线	• 三线共点
• 凸包	• 半平面交

凸包半平面交对偶

通过点线对偶，可以通过凸包算法求半平面交。即 $Ax + By + 1 \geq 0 \Leftrightarrow (A, B)$。

对所有半平面做对偶，但是不要忘记加入全平面对应的点 $(0, 0)$。

如果存在半平面 $C = 0$，可以通过给所有半平面平移来避免。

然后就可以用凸包求半平面交了。

线性规划

Definition (Linear Programming)

线性规划是一类问题，具有以下形式：

$$
\begin{aligned}
\textbf{Find}\ & x \in \mathbb{R}^n \\
\textbf{Maximize}\ & c^T x \\
\textbf{Subject to}\ & Ax \leq b \\
\textbf{And}\ & x \geq 0.
\end{aligned}
$$

二维线性规划

Example

二维线性规划就是向量 x 只有两维。
对应着半平面 $Ax_1 + Bx_2 \leq C$。因此可以用单纯形法解决。
有解对应着半平面交为一个凸包。
无解对应着半平面交为空。
解无界对应着半平面交有无界区域。

凸包 DP

Example

给定 n 个点和 m 条线段，线段连接这这些点。问这些点能构成多少种凸多边形。$n \leq 500$。

凸包 DP

Example

给定 n 个点和 m 条线段，线段连接这这些点。问这些点能构成多少种凸多边形。$n \leq 500$。

Algorithm

枚举凸包起点 S，记 *DP* 状态为 f_i，表示上个点是 i 的情况下凸壳的方案数。

按照极角序将线段排序，并按极角序枚举边 (u, v)，进行转移：

$$f_v {+}{=} f_u.$$

由于枚举了极角序，不需要判断凸性。

[JSOI2007] 合金

Example

有三种元素。有 n 个原材料，每个材料的三种元素比例由 a_i, b_i, c_i 给出。原材料可以合成，合成后的材料元素比例是线性组合。
现在给出 m 个要求材料的元素比例 d_i, e_i, f_i，问最少要多少种不同的原材料能够合成出所有要求的材料。
保证 $a_i + b_i + c_i = d_i + e_i + f_i = 1$，$n, m \leq 500$。

Algorithm

可以发现，因为 $a+b+c=1$，所以可以把材料看作平面上的点 (a,b)。
材料的合成即是线性组合，因此原材料能合成的材料为一个凸包。
所以此题的任务是找出一个最小的点集，使得其凸包包含所有点。
找出所有可能在凸包上的边，求一个最小环即可。

旋转卡壳

Example (凸包直径)

给定凸包，在线性时间内求它的直径（内部最远点对）。

旋转卡壳

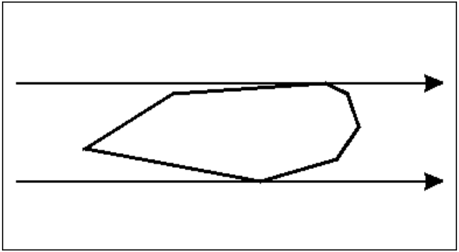
Example (凸包直径)

给定凸包，在线性时间内求它的直径（内部最远点对）。

Algorithm

最远的两点肯定在凸包的顶点上。按斜率枚举一对平行线（对应着枚举最远两点连线的斜率），使其刚好卡住整个凸包。
此时平行线切到的点是不断在旋转的。旋转卡壳做的就是，枚举这个旋转的过程，找出所有被切到点对改变的时刻。
此时，我们能得到一段区间斜率的区间，使得切到的点是不变的。在这个区间内求极值是容易的。

旋转卡壳



旋转卡壳

Algorithm

假设逆时针旋转平行线，目前两点为 u, v，则下一个切到的点和 u, v 逆时针方向的边有关系。
比较两条边哪条更先被转到（使用叉积），来选择哪一个点被替换掉。
实际实现其实可以使用类似于双指针的方法，即从小到大枚举一个点，可以直接得到切到另一个点的区间。

最小矩形覆盖

Example

给定点集，求一个矩形，使得点集被矩形完全覆盖，且矩形面积最小。

最小矩形覆盖

Example

给定点集，求一个矩形，使得点集被矩形完全覆盖，且矩形面积最小。

Algorithm

因为矩形的边是垂直的，因此直接枚举矩形一条边的斜率，可以直接枚举到这条平行线和垂线切到的四个点。

[SCOI2007] 最大土地面积

Example

给定点集，从其中选四个点使其构成的多边形面积最大化。
点个数 $n \leq 2000$。

[SCOI2007] 最大土地面积

Algorithm

最优的点一定都在凸包上。可以发现，一旦枚举了对角线，剩下两个点便是与对角线法向量内积最大与最小的点。而这个点，是随着对角线斜率单调变化的。
所以枚举其中一个点，用类似旋转卡壳的方法，枚举对角线另一个点，用平行于这个对角线的直线去卡凸包得到剩下两个点。

Outline

1 凸包

2 凸包上操作

3 凸包与数据结构

凸包上三分/二分

对于凸函数，寻找其极值可以使用三分。如果能方便求得其导数，便可以用二分。

在凸包上由于其斜率容易计算，因此很好二分。

凸包上三分/二分

对于凸函数，寻找其极值可以使用三分。如果能方便求得其导数，便可以用二分。

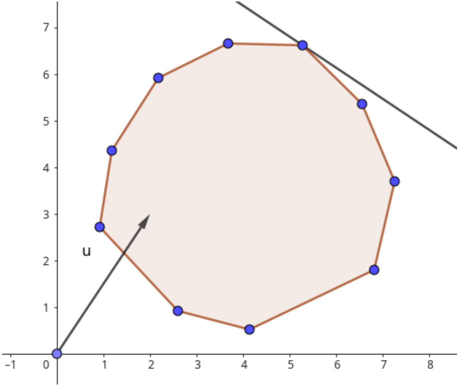
在凸包上由于其斜率容易计算，因此很好二分。

Example

给定向量 x 和凸包 C，求 C 上的点 p，使得内积 $\langle x, p\rangle$ 被最大化。

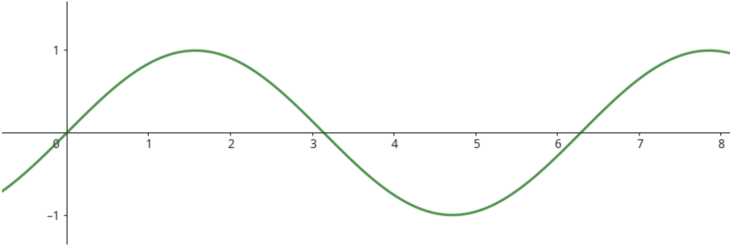
相当于找一个方向上的最远点。

凸包上三分/二分



凸包上三分/二分

凸包的形状类似圆，想象 $\sin$ 函数：如果随意截取一个周期，那么这段区间内函数很可能不是凸的，因为其导数在一个周期内产生了两次符号变化：



凸包上三分/二分

为了解决这个问题，截取半个周期，分两次二分可以保证函数是上凸/下凸的。

因此，凸包二分可以对着凸壳做。使用 Andrew 算法直接获取凸包的上下凸壳，并根据给定向量的 y 方向分量大小决定选择二分的凸壳。

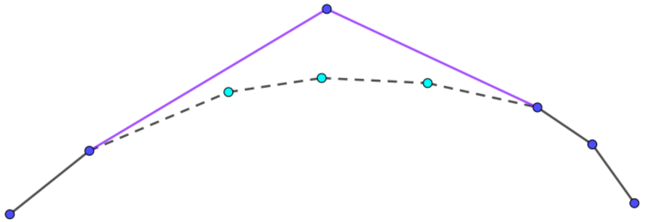
上下凸壳加点

Example

维护一个上凸壳，支持动态加点，支持查询使内积最大化的点。

Algorithm

使用平衡树维护上凸壳，每次加点时使用二分找到插入位置，然后检查是否需要删除其左右的点。查询时使用二分找到使内积最大化的点。



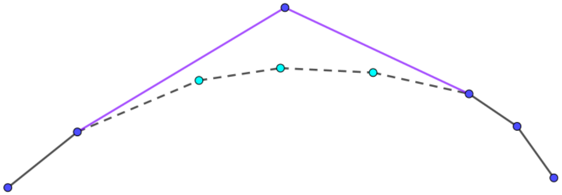
凸包切线

Example

给定凸包，问凸包外一个点到凸包的切线。

Algorithm

等价于将这个点加入点集，产生的新边。因此在平衡树上二分切线的位置即可。



凸包合并

Example

维护一堆点的集合 S_i，支持以下操作：

- 合并两个集合成一个新的集合；
- 对一个集合 S 查询，给定向量 x，最大化 $\langle x, p\rangle$ 的点 $p \in S$。

凸包合并

Example

维护一堆点的集合 S_i，支持以下操作：

- 合并两个集合成一个新的集合；
- 对一个集合 S 查询，给定向量 x，最大化 $\langle x, p\rangle$ 的点 $p \in S$。

Algorithm

使用平衡树维护集合的上下凸壳，此时问题在合并。直接使用启发式合并的复杂度是 $O(\log^2 n)$。

凸包合并

Algorithm

使用空间更多的做法：使用线段树合并，并且在边界处判断是否有点要删除。因为删点总次数是 $O(n)$，所以这么做仍然是 $O(\log n)$ 的。
使用平衡树启发式合并：可以做到线性空间。例如 *Splay* 这种同样可以在合并时判断删点条件：当插入一个点，被旋到根时，可以根据区间信息（每个点维护区间最前面和最后面两个点）来判断删点。

SOJ418.【SHR 1】忌蒜挤核

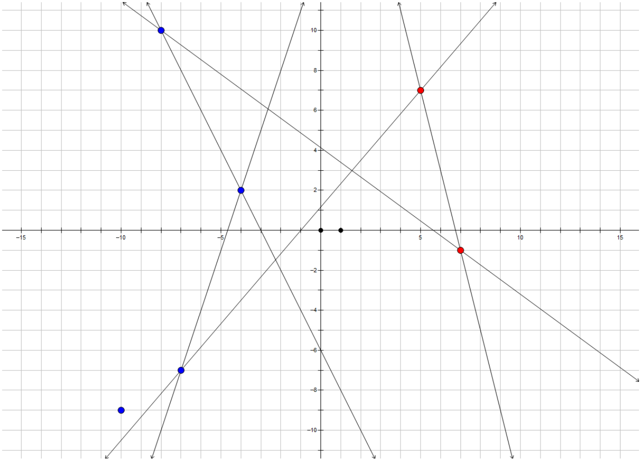
Example

维护一个点集，支持往里面加点。支持查询满足以下条件本质不同直线的数量：

- 经过点集里至少两个点。
- 将点分成两侧，一侧是 $x < 0$ 的，另一侧是 $x > 0$ 的。

点数 $n \leq 3 \times 10^5$。

SOJ418.【SHR 1】忌蒜挤核



SOJ418.【SHR 1】忌蒜挤核

Algorithm

需要建立凸壳。容易发现只有两条内公切线以及公切线中间的凸壳边是可用的。

因为凸壳是单调增长的，因此可以对凸壳中改变里的位置重新求内公切线。但是实现较阴间。

SOJ418.【SHR 1】忌蒜挤核

实际上做射影变换 $(x, y) \to (1/x, y/x)$ 后，发现答案就变成了所有点构成的凸壳的边。

原因是，题目本质上等价于，求点 i, j，使得对于所有 k，行列式 $\begin{vmatrix} x_i & y_i & 1 \\ x_j & y_j & 1 \\ x_k & y_k & 1 \end{vmatrix}$ 的符号与 x_k 相同。

根据行列式性质，相当于判断 $\begin{vmatrix} x_j/x_j & y_j/x_j & 1 \\ x_i/x_i & y_i/x_i & 1 \\ x_k/x_k & y_k/x_k & 1 \end{vmatrix}$ 的符号。

Outline

1. 凸包
2. 凸包上操作
3. 凸包与数据结构

Example

要求维护一个关于二维点的栈，支持：

- 往序列尾加元素

Algorithm (势能线段树)

凸壳可以作为区间信息被维护。其合并即归并。但是其合并复杂度为线性。

实际上是一种基于势能的二进制分组。由于合并操作较慢，因此要求达到一定势能再合并。

李超线段树

Example

维护平面上线段的集合，支持：

- 动态插入线段。
- 给定横坐标 x_0，求直线 $x = x_0$ 与所有线段交点的最高点。

Algorithm

使用线段树，每个节点 $[l, r]$ 上维护线段的集合 S，并保证最高的线段只有一条。同样，每条线段分布在线段树的若干节点上，用类似于 *Lazy Tag* 的方法维护。

李超线段树

Algorithm

插入线段是一个区间修改，当在区间 $[l, r]$ 插入线段 x 的时候，和原有的线段 y 相比会有三种情况：

- x 完全比 y 高，则把 y 换成 x。
- y 完全比 x 高，则不用管。
- x 和 y 有交点：
  - 如果 x 在上方的部分比 y 多，则交换 x 和 y。
  - 否则递归处理交点所在的那半边。

这样插入复杂度为 $O(\log^2 n)$。查询需要查从 $[x, x]$ 到根的所有节点所以复杂度为 $O(\log n)$。

斜率优化

在一类 DP 问题中，DP 的转移实际上是在点集查询内积最大的点。此时问题就可以转化为维护一个凸壳。

并且有时候用来查询的向量斜率也是单调的，因此可以单调地选取凸壳上的点。

斜率优化相关题目详解

见 单调性优化课件 （中的 P21 到 P58，或第 50 帧到第 173 帧）。