浅谈数据结构题的几个非经典解法 ——《Claymore》命题报告

许昊然

南京外国语学校

题目大意

要求维护一组半平面，并支持：

- 插入一个半平面
- 删除某次插入的半平面
- 查询一个点是否在所有的半平面内

数据规模

各测试点数据规模如下。时间限制为3秒。

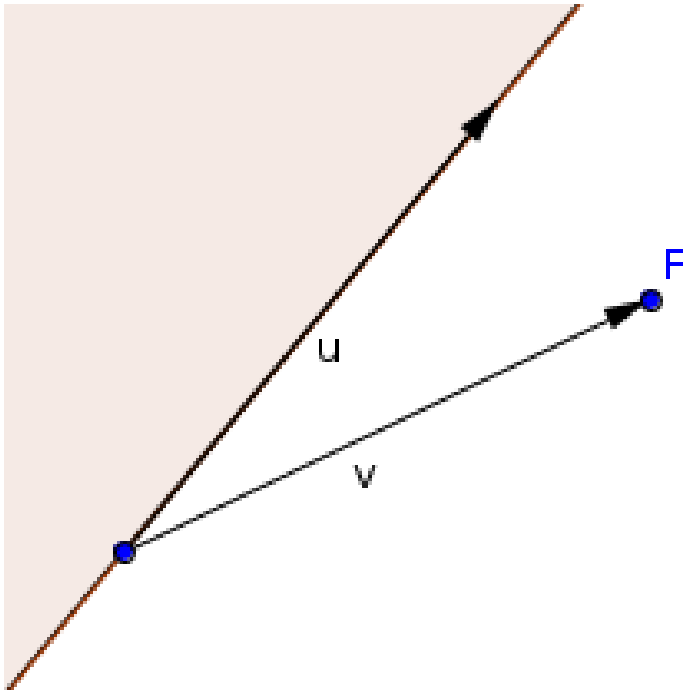
编号	操作数目	特殊性质
1	≤ 2000	没有删除操作且第一个询问后无插入操作
2	≤ 80000	
3	≤ 60000	没有删除操作
4	≤ 80000	
5	≤ 90000	
6	≤ 100000	
7	≤ 60000	无
8	≤ 80000	
9	≤ 90000	
10	≤ 100000	

10%的数据

由于数据规模非常小，当遇到询问时直接暴力判断该点是否在所有半平面内即可。

10%的数据

由于数据规模非常小，当遇到询问时直接暴力判断该点是否在所有半平面内即可。



只需判定向量u叉乘向量v是否为负即可。

20%的数据

前20%的数据中，保证没有删除操作，并且第一个询问之后就再也没有修改。

20%的数据

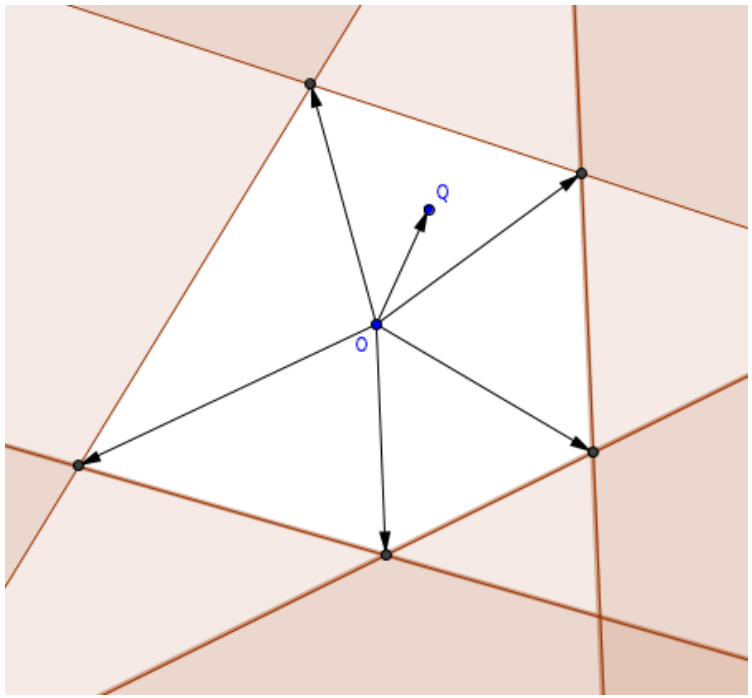
前20%的数据中，保证没有删除操作，并且第一个询问之后就再也没有修改。

我们很容易想到，用半平面交算法求出所有半平面的交集后，选择其中一点作为中心，并对交集顶点进行极角排序，查询时，二分找到查询点所位于的极角序区间，然后判定查询点是否属于这个三角形即可。

20%的数据

前20%的数据中，保证没有删除操作，并且第一个询问之后就再也没有修改。

我们很容易想到，用半平面交算法求出所有半平面的交集后，选择其中一点作为中心，并对交集顶点进行极角排序，查询时，二分找到查询点所位于的极角序区间，然后判定查询点是否属于这个三角形即可。



60%的数据

前60%的数据中，保证没有删除操作。因此，我们实际需要支持的操作是：

增加一个半平面

查询一个点是否在所有的半平面内

60%的数据

前60%的数据中，保证没有删除操作。因此，我们实际需要支持的操作是：

增加一个半平面

查询一个点是否在所有的半平面内

这个问题我们有很多种解法。

60%数据解法A

在上个做法中，我们发现每次预处理的复杂度是$O(n \log n)$，而查询的复杂度是$O(\log n)$，这个复杂度很不平衡。

60%数据解法A

在上个做法中，我们发现每次预处理的复杂度是$O(n \log n)$，而查询的复杂度是$O(\log n)$，这个复杂度很不平衡。

于是，最直观的想法是，把这些半平面（不妨设有n个）任意分成$O(\sqrt{n})$组，每组有$O(\sqrt{n})$个半平面。当修改时，只需暴力修改其所属组的半平面交，$O(\sqrt{n} \log n)$；查询时，只需判定该点是否属于各个组的半平面交即可，$O(\sqrt{n} \log n)$。

60%数据解法A

在上个做法中，我们发现每次预处理的复杂度是$O(n \log n)$，而查询的复杂度是$O(\log n)$，这个复杂度很不平衡。

于是，最直观的想法是，把这些半平面（不妨设有n个）任意分成$O(\sqrt{n})$组，每组有$O(\sqrt{n})$个半平面。当修改时，只需暴力修改其所属组的半平面交，$O(\sqrt{n} \log n)$；查询时，只需判定该点是否属于各个组的半平面交即可，$O(\sqrt{n} \log n)$。

时间复杂度$O(n\sqrt{n} \log n)$，期望得分30分。

60%数据解法A

在上个做法中，我们发现每次预处理的复杂度是$O(n \log n)$，而查询的复杂度是$O(\log n)$，这个复杂度很不平衡。

于是，最直观的想法是，把这些半平面（不妨设有n个）任意分成$O(\sqrt{n})$组，每组有$O(\sqrt{n})$个半平面。当修改时，只需暴力修改其所属组的半平面交，$O(\sqrt{n} \log n)$；查询时，只需判定该点是否属于各个组的半平面交即可，$O(\sqrt{n} \log n)$。

时间复杂度$O(n\sqrt{n} \log n)$，期望得分30分。

顺带一提，如果使用S&I算法进行半平面交，并且插入时维护每组内半平面极角的有序的话，可以通过设置分块大小为$\sqrt{n \log n}$而做到时间复杂度$O(n\sqrt{n \log n})$，只是一个理论上的优化。

60%数据解法B

我们发现，实际我们需要做的是以极角序为关键字维护半平面交的顶点。

60%数据解法B

我们发现，实际我们需要做的是以极角序为关键字维护半平面交的顶点。

我们可以使用splay树来维护它。

插入时，向平衡树插入元素，删掉不再属于半平面交的元素

查询时在平衡树上二分即可找到查询点所在的极角区间

60%数据解法B

我们发现，实际我们需要做的是以极角序为关键字维护半平面交的顶点。

我们可以使用splay树来维护它。

插入时，向平衡树插入元素，删掉不再属于半平面交的元素

查询时在平衡树上二分即可找到查询点所在的极角区间

时间复杂度$O(n \log n)$，期望得分60分，但代码很复杂。

60%数据解法C

有没有更简便的解决方法呢?

60%数据解法C

我们发现，我们通过数据结构优化复杂度的实质是预处理一些东西，而询问时可以利用预处理的东西来快速得到答案。

60%数据解法C

我们发现，我们通过数据结构优化复杂度的实质是预处理一些东西，而询问时可以利用预处理的东西来快速得到答案。

但这题中，我们还需要支持插入操作，因此，我们预处理的东西不仅要“对询问有利”，还要能“跟上时代的步伐”，也就是面对修改，能快速更新我们的预处理内容。

60%数据解法C

借鉴各类树形数据结构，我们很容易想到利用二进制进行预处理。

60%数据解法C

借鉴各类树形数据结构，我们很容易想到利用二进制进行预处理。

我们将一个数拆成2的幂次从大到小的和的形式。示例：

$$
\begin{aligned}
21 &= 16+4+1 \\
22 &= 16+4+2 \\
23 &= 16+4+2+1 \\
24 &= 16+8
\end{aligned}
$$

60%数据解法C

这有什么意义呢?
我们不妨用上述拆分方法对半平面分组，用20%数据的解法来维护半平面（求出半平面交后，对顶点排序，查询时二分查找）。

60%数据解法C

这有什么意义呢?
我们不妨用上述拆分方法对半平面分组，用20%数据的解法来维护半平面（求出半平面交后，对顶点排序，查询时二分查找）。

比如说，对前21个半平面，我们会把前16个半平面分为一组，第17～20个半平面分为第二组，第21个半平面作为第三组（21=16+4+1）。按照上述的分组方法，显然最多只有$O(\log n)$组半平面，查询时在各组内都查询一遍即可，时间复杂度$O(\log^2 n)$。

60%数据解法C

这有什么意义呢?
我们不妨用上述拆分方法对半平面分组，用20%数据的解法来维护半平面（求出半平面交后，对顶点排序，查询时二分查找）。

比如说，对前21个半平面，我们会把前16个半平面分为一组，第17～20个半平面分为第二组，第21个半平面作为第三组（21=16+4+1）。按照上述的分组方法，显然最多只有$O(\log n)$组半平面，查询时在各组内都查询一遍即可，时间复杂度$O(\log^2 n)$。

而增加半平面时怎么做？直接删掉不再存在的分组，暴力建出新增加的分组。

下面给出图形化的例子。

60%数据解法C

当前有21个半平面（第i个点代表第i个插入的半平面）

60%数据解法C

当前有21个半平面（第i个点代表第i个插入的半平面）

于是我们的分组是这样的（21=16+4+1）

60%数据解法C

当前有21个半平面（第i个点代表第i个插入的半平面）

于是我们的分组是这样的（21=16+4+1）

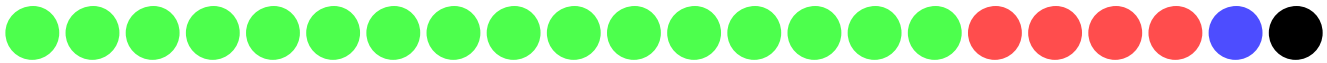
查询时，我们对每一组记录其半平面交顶点的极角序，查询时在各组内二分查找即可。

60%数据解法C

修改时，比如说，现在来了第22个半平面。（22=16+4+2）

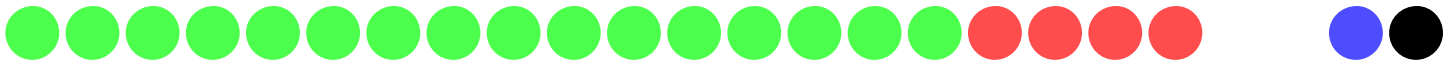
60%数据解法C

修改时，比如说，现在来了第22个半平面。（22=16+4+2）



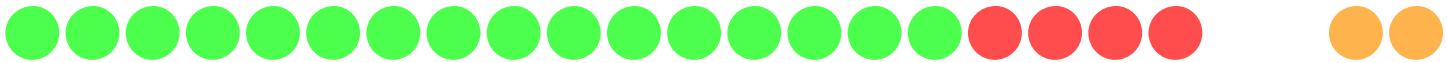
60%数据解法C

修改时，比如说，现在来了第22个半平面。（22=16+4+2）



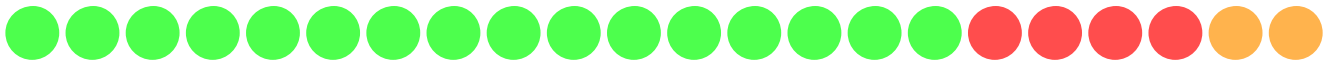
60%数据解法C

修改时，比如说，现在来了第22个半平面。（22=16+4+2）



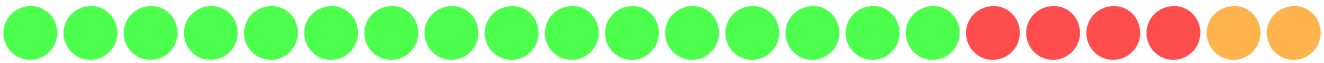
60%数据解法C

修改时，比如说，现在来了第22个半平面。（22=16+4+2）



60%数据解法C

比如说，又来了第23个半平面。（23=16+4+2+1）



60%数据解法C

比如说，又来了第23个半平面。（23=16+4+2+1）

60%数据解法C

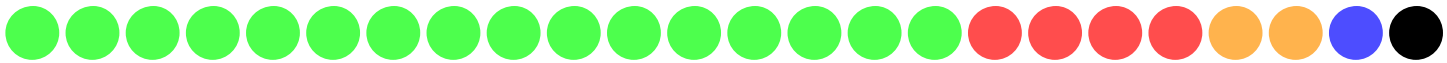
比如说，又来了第23个半平面。（23=16+4+2+1）

60%数据解法C

又比如说，又来了第24个半平面。（24=16+8）

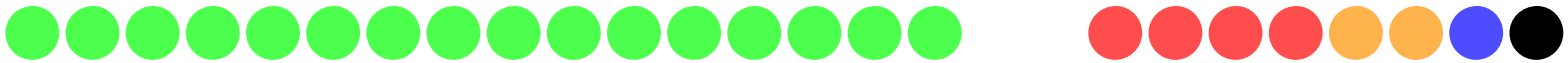
60%数据解法C

又比如说，又来了第24个半平面。（24=16+8）



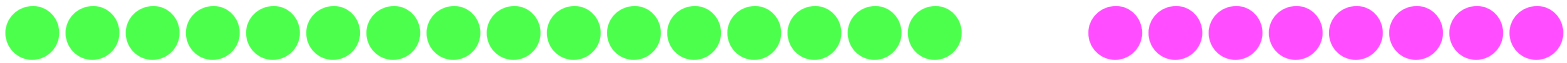
60%数据解法C

又比如说，又来了第24个半平面。（24=16+8）



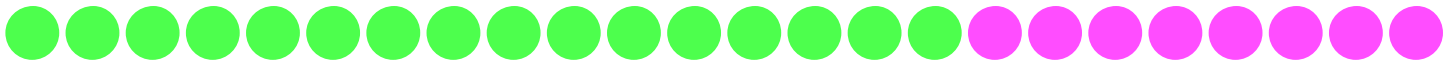
60%数据解法C

又比如说，又来了第24个半平面。（24=16+8）



60%数据解法C

又比如说，又来了第24个半平面。（24=16+8）



60%数据解法C

时间复杂度分析:
询问的复杂度前文已经说明，为$O(\log^2 n)$每次询问。下面我们证明修改的复杂度。

60%数据解法C

时间复杂度分析:
询问的复杂度前文已经说明，为$O(\log^2 n)$每次询问。下面我们证明修改的复杂度。

我们考虑我们重建的所有组的元素总数。实际上我们发现，当添加第k个半平面时，我们重建的组的元素个数是$lowbit(k)$，也就是k的二进制表示下最右侧的1所代表的权值。
比如$22 = (10110)_2$所以$lowbit(22) = (10)_2 = 2$。

考虑$lowbit(k)$的取值及其数量，我们很容易推出，总时间复杂度是

$$\begin{aligned} & \sum_{1\le k\le n} lowbit(k) * \log lowbit(k) \\ = & \sum_{1\le i\le \log n} \left\lfloor \frac{n}{2^{i+1}} + 0.5 \right\rfloor * 2^i * i \\ \le & \sum_{1\le i\le \log n} n * \log n = O(n \log^2 n) \end{aligned}$$

60%数据解法C

上述做法实现简单，期望得分60分。
但还能更简单吗?

60%数据解法D

我们发现，这道题有几个有用的性质：

这道题没有强制要求在线算法，因此我们可以使用往往更简单的离线算法。

这道题中，插入操作对询问的贡献是互相独立的。

60%数据解法D

我们发现，这道题有几个有用的性质：

这道题没有强制要求在线算法，因此我们可以使用往往更简单的离线算法。

这道题中，插入操作对询问的贡献是互相独立的。

对时间分治！

60%数据解法D

我们不妨将整个操作序列等分为两份，考虑前一半操作序列中的查询，这些查询和后一般操作序列中的插入操作完全无关。考虑后一半操作的查询，它们的答案只与两部分内容相关：

前一半操作序列中的所有插入操作

后一半操作序列中在该询问之前的插入操作

60%数据解法D

我们发现，因为之前提到的性质，插入操作对询问的贡献是互相独立的。

60%数据解法D

我们发现，因为之前提到的性质，插入操作对询问的贡献是互相独立的。

因此，我们可以单独把前一半操作序列中的插入操作拉出来，并算出它对后一半操作序列中的询问的贡献。

60%数据解法D

我们发现，因为之前提到的性质，插入操作对询问的贡献是互相独立的。

因此，我们可以单独把前一半操作序列中的插入操作拉出来，并算出它对后一半操作序列中的询问的贡献。

于是问题转化为：给定若干半平面，要求支持查询一个点是否在所有半平面内。
用最早我们提到的20%数据做法就可以了。$O(n \log n)$

60%数据解法D

我们发现，处理了前一半的插入操作对后一半的查询操作的贡献之后，原问题被分割为了两个完全相同的独立子问题：前一半操作序列内部的贡献和后一半操作序列内部的贡献，递归处理即可。

60%数据解法D

时间复杂度分析：假设当前待处理序列长度为n，那么时间复杂度是

$$T(n) = 2T(\tfrac{n}{2}) + O(n \log n)$$

使用主定理易解得

$$T(n) = O(n \log^2 n)$$

期望得分60分，代码非常简单。

60%数据解法D

其实可以通过归并排序、离线扫描等方法把时间复杂度优化到$O(n \log n)$

60%数据解法D

其实可以通过归并排序、离线扫描等方法把时间复杂度优化到$O(n \log n)$

但因为常数原因，速度优化不明显，且代码较复杂。这里不作阐述，有兴趣的同学请参考我的论文。

满分做法

现在，我们不仅要支持修改，还要支持删除。这下，先前的做法都失效了。

满分做法

现在，我们不仅要支持修改，还要支持删除。这下，先前的做法都失效了。

我们不妨继续考虑对时间分治的做法。现在，修改操作（我们不妨认为插入和删除都算修改）不完全独立了，因为现在一个插入操作可能会被后面的删除操作撤销掉。

满分做法

我们不妨依然把操作序列等分为两半。我们考虑前一半操作中的查询，它们的答案显然与后半部分的修改操作都完全无关，因此它是与原问题完全相同的子问题，可以递归解决。

满分做法

考虑后一半操作的查询，它们的答案只与两部分内容相关：

- 前一半操作序列中的所有插入操作中在该询问之前未被删除的部分
- 后一半操作序列中在该询问之前的且未被删除的插入操作

满分做法

我们发现，第二部分中，因为后一半序列中的插入操作显然不会在前一半序列被删除，因此第二部分与前一半操作序列完全无关。这一部分也是与原问题完全相同的子问题，可以递归解决。

满分做法

我们发现，第二部分中，因为后一半序列中的插入操作显然不会在前一半序列被删除，因此第二部分与前一半操作序列完全无关。这一部分也是与原问题完全相同的子问题，可以递归解决。

我们发现，因此实际我们要处理的内容是：
『前一半操作序列中的插入操作中未被删除的部分』对『后一半操作序列的询问』的贡献。

满分做法

因此，我们实际的任务是，一开始给定一组半平面，要求支持：

删除一个半平面

查询一个点是否在所有的半平面中

满分做法

因此，我们实际的任务是，一开始给定一组半平面，要求支持：

删除一个半平面

查询一个点是否在所有的半平面中

很不幸，这个问题想要直接解决是非常困难的。

满分做法

但我们发现这个问题中只有删除操作，没有插入操作。于是一个经典的解决方法派上用场了：

满分做法

但我们发现这个问题中只有删除操作，没有插入操作。于是一个经典的解决方法派上用场了：

时间倒流！

满分做法

因为我们使用的是离线算法，我们完全可以预知转化后的问题中『初始的半平面集』『删除和查询序列』等所有我们需要的信息。

满分做法

因为我们使用的是离线算法，我们完全可以预知转化后的问题中『初始的半平面集』『删除和查询序列』等所有我们需要的信息。

我们先求出最后没有被删除的半平面是哪些，然后逆序处理操作序列，这样，查询没有变，而删除半平面变成了插入半平面！

满分做法

于是我们再次把问题转化成了：要求支持

插入一个半平面

查询一个点是否在所有的半平面中

满分做法

于是我们再次把问题转化成了：要求支持

插入一个半平面

查询一个点是否在所有的半平面中

是不是很熟悉呢？就是60%数据的那个问题！
再次对时间分治即可！

满分做法

时间复杂度分析：假设当前待处理序列长度为n，那么时间复杂度是

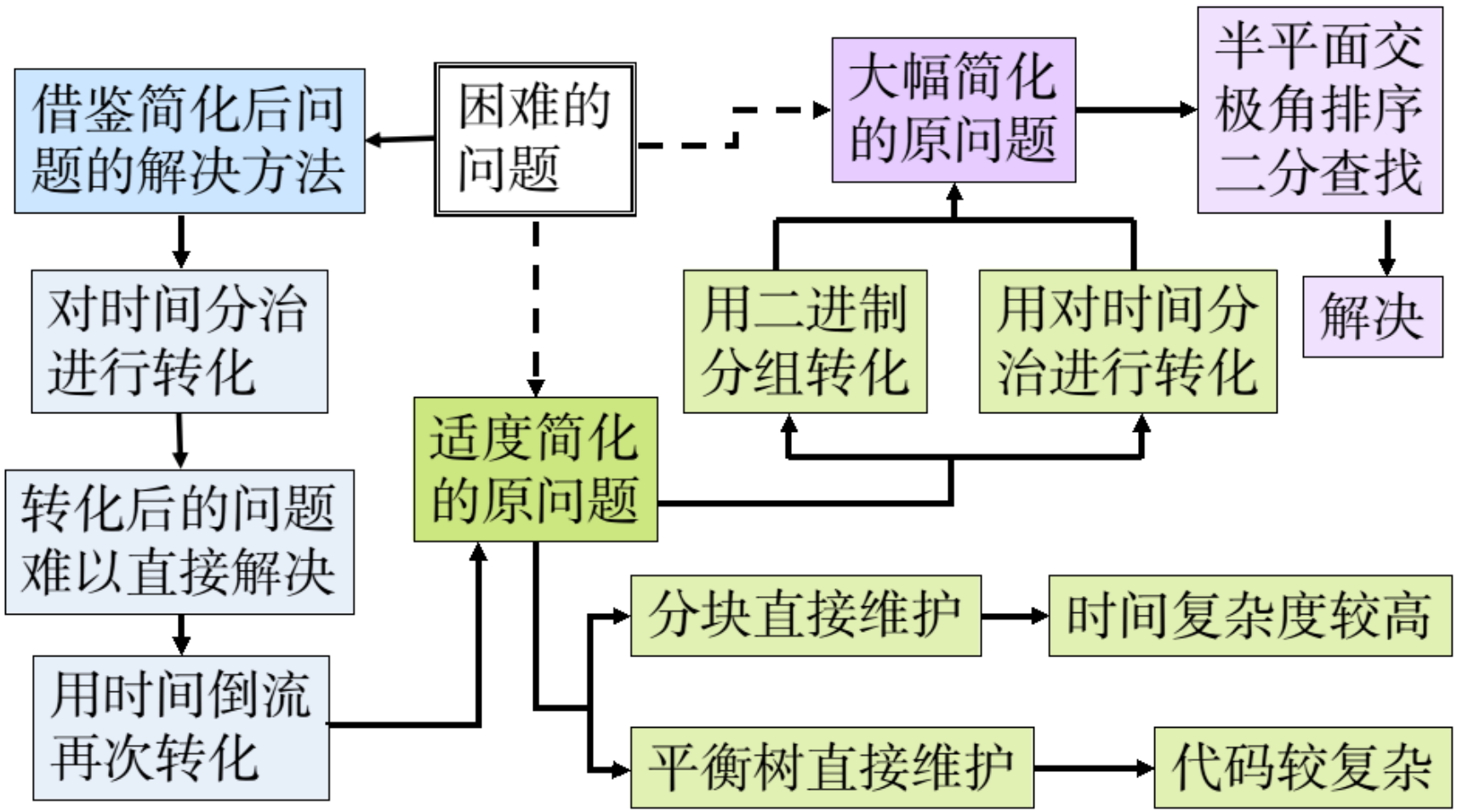
$$T(n) = 2T(\tfrac{n}{2}) + O(n \log^2 n)$$

使用主定理易解得

$$T(n) = O(n \log^3 n)$$

期望得分100分，代码很简单。当然，如果用我们利用归并优化，时间复杂度可以降低到$O(n \log^2 n)$，不过代码就会相对复杂了。

思维导图


借鉴简化后问题的解决方法
困难的问题
大幅简化的原问题
半平面交极角排序二分查找
对时间分治进行转化
用二进制分组转化
用对时间分治进行转化
解决
转化后的问题难以直接解决
适度简化的原问题
用时间倒流再次转化
分块直接维护
时间复杂度较高
平衡树直接维护
代码较复杂

扩展

对每个半平面赋予一个权值，题目变成：

插入一个带权值的半平面

删除某次插入的半平面

询问覆盖某个点的半平面中，权值最大的是多少?

扩展

对每个半平面赋予一个权值，题目变成：

插入一个带权值的半平面

删除某次插入的半平面

询问覆盖某个点的半平面中，权值最大的是多少?

与整体二分算法相结合。通过整体二分算法即可将这个问题转化为本文讲的问题。
关于整体二分算法的更多介绍请参见我的论文。

启发

对时间分治算法应用要求：修改独立，允许离线
二进制分组算法应用要求：修改独立，要求在线

启发

对时间分治算法应用要求：修改独立，允许离线
二进制分组算法应用要求：修改独立，要求在线

通过利用数据结构题的操作中隐藏的特殊性质，例如“修改独立”“贡献独立”“允许离线算法”等，来做到转化问题、简化问题。这启示我们，要充分挖掘题目特殊性、利用题目特殊性，从而得到更加优美高效的算法。

欢迎提问

感谢CCF提供了这个交流的平台
感谢父母、学校对我的培养
感谢与我交流讨论、给予我帮助和鼓励的同学们
感谢我的班主任和其他老师对我的支持
数据设计与得分期望请参见隐藏幻灯片

数据设计

本题没有删除操作的数据设计较简单，只需生成一个椭圆，然后在椭圆上选若干点，这些点将作为最终半平面交集的顶点。

插入操作只需随机选择两个相邻顶点连接起来即可

询问操作可以先随机决定这个询问操作的答案，再生成符合答案的点

数据设计

本题没有删除操作的数据设计较简单，只需生成一个椭圆，然后在椭圆上选若干点，这些点将作为最终半平面交集的顶点。

- 插入操作只需随机选择两个相邻顶点连接起来即可
- 询问操作可以先随机决定这个询问操作的答案，再生成符合答案的点

有删除操作的数据设计也并不困难。我们可以生成若干个椭圆，并在椭圆上随机选若干点。

- 插入和询问设计同上
- 通过删除操作将一个椭圆内的半平面交逐步转化为另一个椭圆的半平面交

数据设计

本题没有删除操作的数据设计较简单，只需生成一个椭圆，然后在椭圆上选若干点，这些点将作为最终半平面交集的顶点。

- 插入操作只需随机选择两个相邻顶点连接起来即可
- 询问操作可以先随机决定这个询问操作的答案，再生成符合答案的点

有删除操作的数据设计也并不困难。我们可以生成若干个椭圆，并在椭圆上随机选若干点。

- 插入和询问设计同上
- 通过删除操作将一个椭圆内的半平面交逐步转化为另一个椭圆的半平面交

本题数据设计有很强的梯度，目的在于鼓励选手想不出完美做法时使用各种复杂度的算法，来尝试获取更多分数。

得分期望

本题代码难度不大，主要侧重点是思维和分析能力的考察。预计如果在NOI赛场上，绝大多数选手都可以获得30分以上的成绩，大约三分之一的选手可以获得60分以上的成绩，少数优秀的选手能获得满分。