# 小 $q$ 的机器人解题报告

绍兴一中 张恒捷

## 1 试题来源

原创

## 2 试题大意

一条数轴上有 $n$ 个点，一开始每个点的速度都是 0，现在要按时间顺序执行 $m$ 个操作，每个操作分为两种：

1.重置某个点的速度，之后那个点将会以给定的速度在数轴上进行匀速运动（速度为负表示向数轴负方向运动）。

2.询问离原点最远的点的距离。

## 数据规模与约定

设第一种操作的个数为C，第二种操作的个数为Q。

所有测试数据的范围和特点如下表所示：

| 测试点编号 | 数据范围 | 特殊约定 |
|---|---|---|
| 1 | $n, m \leq 2000$ | 无 |
| 2 | | |
| 3 | $n, m \leq 10^5$ | $-1 \leq x_i \leq 1$ |
| 4 | $n \leq 10^5, C \leq 10^5, Q \leq 5 \times 10^5$ | 两个点发生碰面或者一个超越另一个的次数 $\leq 4 \times 10^5$ |
| 5 | | |
| 6 | $n, m \leq 10^5$ | 不会在$t_i > 0$时出现 *command* 操作 |
| 7 | | |
| 8 | | 无 |
| 9 | $n \leq 10^5, C \leq 10^5, Q \leq 5 \times 10^5$ | |
| 10 | | |

# 3 算法介绍

## 3.1 针对测试点1,2的做法

由于n和m都很小，所以可以维护每个点的位置与速度。在每次询问时算出每个点新的位置，加以判断即可。在每次修改操作时，也同样算出每个点新的位置，然后修改对应点的速度即可。

时间复杂度： $O(nm)$

## 3.2 针对测试点3的做法

由于 $x_i$ 只可能是-1,0,1，那么我们可以维护三个序列，维护 $x_i$ 分别为-1,0,1的点在数轴上的顺序。当询问时，只需要询问每个序列坐标最小的和坐标最大的点。当一个点的速度被改变时,只需要算出它的坐标，然后把它从原序列中删除后放入对应的新序列中即可。对于维护序列的方法，可以使用 $C++STL$ 中的 *set* 即可。

时间复杂度： $O(mlogn)$

## 3.3 针对测试点4,5的做法

这两个测试点有特殊条件，两个点发生碰面或者一个超越另一个的次数 $\leq 4 \times 10^5$。那么我们可以按照时间的顺序维护点的先后序列。随着时间的流逝最早的超越与碰撞一定由序列中相邻的点引发的。因此我们可以用一个优先队列存放所有相邻点的碰撞/超越时刻，每次选取一个最早的时刻更新一下序列即可。询问时只需要查询序列中第一个与最后一个的位置即可。

时间复杂度： $(4 \times 10^5 log\ n + Q)$

## 3.4 线段树套凸包

考虑建立二维笛卡尔坐标系，以时间为 $x$ 轴，以位置为 $y$ 轴，那么每一个点的行动路线都可以画成一条分段线性函数。询问最远点就相当于找 $x = t$ 这条直线与所有函数的交点中纵坐标最大与最小的点。

将每一个 *command* 时刻提出来，离散掉。那么一条分段线性函数就可以用一系列横坐标在某个范围内的直线来表示了。然后对离散过后的时间段建线段

树，线段树中一段区间里维护的是横坐标在该区间范围内的直线所组成的上凸壳与下凸壳。接下来做的就是把直线插入相应的线段树中的区间就行了。

可以事先将直线的斜率排序，以及离线询问，可以将复杂度减到一个 $log$。

时间复杂度： $O((C+Q+n)logC)$

期望得分：50~ 100分。

## 3.5 满分做法

由于题目并没有要求在线，所以我们使用离线做法。

考虑建立二维笛卡尔坐标系，以时间为 $x$ 轴，以位置为 $y$ 轴，那么每一个点的行动路线都可以画成一条分段线性函数，下文将会简称为折线。假设第 $i$ 个点速度被重置了 $a_i$ 次，那么这条折线就可以用 $a_i+1$ 个拐点来表示。由于题目要求某一时刻离原点最远的点的距离，所以只要求出所有折线取 $min$ 后的函数与所有折线取 $max$ 后的函数，询问便可轻松解决。由于求 $max$ 与求 $min$ 性质相同，所以下文将会以求 $min$ 为例分析。

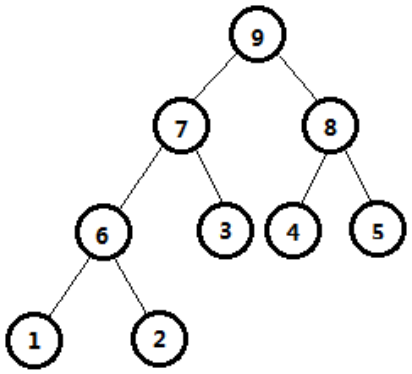
对于折线函数求 $min$ 这里提供一种做法：维护一个集合，这个集合包含了所有的折线。每次从集合中选取两条拐点数最少的折线。不妨设这两条折线分别有 $a,b$ 个拐点，使用 $O(a+b)$ 的复杂度合并这两条折线（对它们取 $min$ ），可以得到一条新的折线。然后将这条折线替换掉那两条折线。不断地递归这个过程直到整个集合只剩下了一条折线，就是我们所要求的折线。

**定理1.** *假如一条折线 $l$ 是由拐点分别为 $a_1,a_2,\ldots,a_m$ 的折线取 $min$ 而成的。则 $l$ 的拐点数是 $O(\sum a_i)$ 级别的。*

该定理的证明放在了本文末尾的附录上。

现在来计算上述算法的复杂度：

将每一条原来就存在的折线和合成出来的折线都抽象成一个节点，每一个合成出来的折线的节点向合成其所用的两条折线连边。这整个结构事实上就是一棵哈夫曼树。




考虑上图，其意义为：

原本只有1,2,3,4,5这5条折线，第一次操作选了拐点数最少的两条折线1,2，合成了折线6，第二次操作将折线6,3合成了折线7，第三次操作将折线4,5合成了折线8，最后将折线7,8合成了最终要求的折线9。

由于该算法使用了 $O(a+b)$ 的复杂度合并两条拐点数分别为 $a$ 与 $b$ 的折线，因此整个算法的复杂度等于所有折线的拐点之和。假设 $s_i$ 为 $i$ 点的子树中的叶子节点的拐点数之和。基于定理1 ，我们可以知道复杂度为:

$$\sum O(s_i) = O(\sum s_i) = O(\sum_{i\ is\ a\ leaf} s_i \times dep_i)$$

其中 $dep_i$ 为点 $i$ 的深度。

现在我们来求叶子节点深度的上界，以1号点为例，可以推出：

$$\begin{aligned} s_9 &= s_7 + s_8 && (1)\\ &\geq 2s_6 + s_3 && (2)\\ &\geq 3s_1 + 2s_2 && (3) \end{aligned}$$

(1)式推到(2)是因为6在8之前合并，所以可以得到 $s_8 \geq s_6$ ，(2)到(3)同理。

从推导中可以得知， $s_x$ 前的系数是斐波拉契数列。因此推广到任意点 $i$ 有：

$$s_{root} \geq Fib(dep_i) \times s_i + Fib(dep_i - 1) \times s_x$$

其中 $Fib(x)$ 为斐波拉契数列第 $x$ 项。 $s_{root}$ 即为拐点总数 $C$ 。因此可以得到 $dep_i \leq log\ n$

因此总复杂度为： $O((n+C)log\ C+Q)$

## 其他

**如何用 $O(a+b)$ 合并呢？**

考虑将拐点全部离散。对与两个横坐标相邻的拐点，折线在他们横坐标范围内的部分是一条卡住左右边界的线段。只要求出两条折线各自的线段，分相交与不相交两类讨论，再根据函数是求 *min* 还是求 *max* 选出正确的线段。最后将所有线段拼起来就得到了最终的线段。实现时可以将横坐标从小到大存，这样只要用两个指针扫一遍就行了。

**精度会有问题吗？**

为了防止精度误差，可以用分数来表示所有的拐点。由于拐点只可能是两条线段的交点，所以用64位整数存足够了。

**询问复杂度为什么是 $O(Q)$？** 由于我们求出了最后的函数，而且 *query* 的时间是按非递减顺序给出的，因此只要在函数上扫一遍就可以了。

# 4 附录

## 关于定理 1 的简易证明。

该定理看似显然但证起来复杂。以下给出本文作者较复杂的原创证明。

设拐点总共有 $m$ 个，则折线最多有 $m$ 条。

考虑扫描线的方法沿着 $x$ 轴扫过去，现在我们只关心对它们求 *minimum* 后的折线，不妨设其为 $l$，当 $l$ 在某个时刻不通过拐点出现转折的时候，可以知道有两条折线的上下关系发生了改变，并且在之后不出现拐点的情况下，它们的上下关系是不会发生变化的。我们需要搜集这些信息来确保定理的正确性。因此我们可以把每条折线抽象成一个点，发生上下关系后后来的较小者向之前的较小者连边，表示上下关系。而出现拐点时，一条折线的位置变得未知了，所以要将它连的边全部断掉，原来连向它的点向所有它连出去的点连边。这里要注意一个事实：由于我们只关心 $l$ 的变化，所以发生上下关系时一定是某条折线向最小的折线连边，然后这条折线变成了最小的折线。所以我们维护的这个结构其实是一棵森林!

现在重新整理一下这个问题，问题转化成了：

有 $m$ 个点的森林，其中有一个特殊点。有两种操作：

1.选择一棵不含有特殊点的树，假设树根为 $t$， $t$ 向特殊点连边，然后将 $t$ 变成特殊点。

2.选择一个点，若该点为树根，则将其子树全部拆开，否则它的父亲将会向其所有儿子连边。之后将这个点连的边全部拆开。

这里作一些解释：每个点代表每条折线，特殊点就是最小的折线。随着扫描线的移动，会有上述两种操作发生。1操作表示在 $l$ 上有两条折线的上下关系发生了变化，2操作表示出现了一个拐点。我们要做的就是用这个模型，在折线在任何时候都可以相交，拐点在任何时候都可以出现的情况下求出 $l$ 拐点数的上界。由于 $l$ 的拐点数小于 $m$ +操作1的个数。因此现在求出2操作固定为 $m$ 个的时候最多能有几个操作1。

由于每执行一次操作1，连通块个数就会减少1。如果连通块为1了就无法在执行了。而操作2作用在树根上会分裂出度数大小的连通块个数。因此操作1个数 $\le 2m$ +从树根上分裂出的连通块个数。

考虑最终操作序列中相邻两个操作2之间的操作1，其实是选择了一些树，

然后将它们的根用一条链连接起来。无视那些只有一个点的树（操作2作用在它们上面对答案贡献是 $O(1)$ 的），假设剩下的树为 $a_1, a_2 \ldots a_t$ ，很显然 $a_i$ 的深度 $\geq i-1$ ，而且每个树根的度数只加了1。对于拼合以后的树来说，它的树根度数只是加了1，而在这棵树中使用 $k$ 次2操作（不管是否作用在树根上）最多只会将最浅的 $a_1, ..., a_k$ 的度数贡献在答案上。注意： $a_1, ...a_k$ 也是用同样的方法合成的，所以有同样的性质："每次树根度数只是加了1"，所以能证明从树根上分裂出的连通块个数是 $O(m)$ 的。一直回到最初始的问题，定理得证。