一类算法复合的方法

江苏省扬州中学 张煜承

摘要

本文讲了一类算法复合的方法。这种方法是指将一个问题的若干种算法，分别使用于这个问题中若干个互补的部分。

本文对两个有意思的问题作了详细的分析，使用了这种算法复合的方法成功解决了这两个问题。问题一中我们将一个$O(N^2)$和一个$O(NR)$的算法复合，分别使用于问题中的两部分询问，得到了一个$O(NR^{0.5})$的算法。问题二中，我们将两个$O(N^2)$的算法使用于原问题分割得到的三部分，得到了一个$O(N^{1.5})$的算法。

本文最后对这类方法进行了总结。每个算法都可能有各自的优势和劣势。而将它们复合，使用于问题中的不同的部分，就有可能会将它们的优势结合起来，取长补短，得出一个总体更优的算法。这种思想是极为重要的。

关键字

算法复合 方法

一、问题一[1]

1.1 问题描述

维护一个集合 S，初始时为空。对这个集合有两种操作：

1、 B X 在集合 S 中插入一个整数 X，保证当前集合中 X 还不存在

2、 A Y 询问集合 S 中，被 Y 除余数最小的数是多少。如果有多个数余数相

[1] 题目来源：The 2006 ACM Asia Programming Contest - Shanghai

等，取任意一个

有 N 个操作需要依次处理。计算所有询问的答案。允许离线算法。

其中$1 \le N \le 40000$，$1 \le X, Y \le 500000$

1.2 初步分析

这道题让我们设计算法维护一个集合S。我们先考虑一些容易想到的算法。

最容易想到的算法是直接模拟问题中规定的操作，我们称其为*算法 1.0*。每当遇到一个询问操作“A Y”时，我们枚举当前集合S中的每个数，从中找出被Y除余数最小的。算法的时间复杂度为$\mathrm{O}(\text{插入操作个数} \times \text{询问操作个数})$，最坏情况下显然会超时。但当插入操作很少或询问操作很少时，这个算法会很快。

另一个略优一些的算法也很容易想到（*算法 1.1*）。设$p(y)$表示当前集合 S 中使得 $x \bmod y$ 最小的数 x，也就是询问“A y”的答案。因为允许离线算法，我们可以事先整理出询问中所有不同的 Y 组成的集合 T，然后我们对每个$y \in T$维护$p(y)$的值。每当插入一个数的时候，我们用$\mathrm{O}(|T|)$的时间逐个更新这些值。算法的时间复杂度为$\mathrm{O}(\text{插入操作个数} \times |T|)$。$|T|$同样是$\mathrm{O}(N)$级别的，所以也不能完全解决问题。其实，这里的集合 T 可以理解为我们想维护的询问。我们可以只维护一部分询问中出现的 Y，维护需要的时间就会减少，但是将会有一些询问得不到回答。

1.3 抓住问题的特征得出另一个算法（*算法 1.2*）

为了解决这个问题，我们抓住问题的特征，深入思考。

当遇到一个询问“A Y”的时候，我们要在当前集合 S 中寻找使得 $x \bmod Y$ 最小的数 x。我们把这里的 x 写成$kY + r$，其中$0 \le r < Y$。那么$x \bmod Y = (kY + r) \bmod Y = r$。这就是说，我们要在集合 S 中，寻找使得 r 最小的数$kY + r$。

如果把 k 确定，那么我们就是要在集合 S 中找区间$[kY, (k+1)Y)$中的最小值。所以我们不难想到一个算法：枚举 k，寻找它对应的区间$[kY, (k+1)Y)$中的最小值，最后在这些最小值中取最优的。图 1.1 形象地描述了这个过程。

0	1	2	…	*Y*-1	*Y*	*Y*+1	*Y*+2	…	2*Y*-1	2*Y*	2*Y*+1	2*Y*+2	…	3*Y*-1	3*Y*	…

最小值为 2　　　最小值为 *Y*　　　最小值为 2*Y*+1　　　…

图 1.1：图中用方格表示所有的自然数，其中集合 *S* 中的元素用阴影表示
我们从每个区间$[kY,(k+1)Y)$中找出最小值，最后再取其中的最优值。这里的最优值为 Y，$Y \bmod Y = 0$

设 *R* 为最大可能会插入的数，根据题目，$R = 500000$。容易得到这里有$\mathrm{O}\left(\frac{R}{Y}\right)$个不同的 k。

我们这样做就把一个找被 *Y* 除余数最小的数的问题，转化成了若干个找给定区间内最小数的问题。

而这个问题我们很熟悉，可以用线段树解决。[2]建一棵$[0,R]$的线段树。在线段树的每个结点$[l,r]$上记录集合 *S* 内 $[l,r]$中的最小数。因为我们事先知道所有要插入的数，我们可以把$[0,R]$离散化，只保留要插入的$\mathrm{O}(N)$个点，这样每次操作就只需$\mathrm{O}(\lg N)$的时间。

但在同样的时间内，线段树可以实现更多种操作。而我们用到的操作只有询问一段区间内的最小值和插入一个数，并且插入时只会在位置 *pos* 插入 *pos* 这个数。所以使用线段树就显得比较“浪费”，我们或许可以找到一个支持的操作较少，但效率更高的算法。

询问集合 *S* 中区间$[a,b]$内的最小数，可以看成是询问大于或等于a的最小数$q(a)$。如果没有大于或等于 *a* 的数，我们称$q(a) = +\infty$。显然，如果$q(a) > b$，那么说明区间$[a,b]$中没有在 *S* 中的数，否则$q(a)$就是区间$[a,b]$内的最小数。图 1.2 给出了一个具体的例子。

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	…
2	2	2	7	7	7	7	7	8	12	12	12	12	+∞	+∞	+∞	…

图 1.2：这里，*S* 当前为{2,7,8,12}，*Y*=5
$q(n)$在方格的下方表示出来

很容易观察到，对很多连续的 *a*，$q(a)$是相等的。如果 *S* 为空，则对于任意的自然数 *a*，$q(a) = +\infty$。否则我们把集合 *S* 中的数排序，得到$p_1 < p_2 < p_3 < \ldots <$

2 关于线段树，可以参见算法导论和国家集训队的相关论文

$p_n(n \geq 1)$（因为插入的数保证不会重复）。那么当$a \in [0, p_1]$时$q(a) = p_1$，当$a \in [p_1 + 1, p_2]$时$q(a) = p_2$，以此类推，直到当$a \in [p_n, +\infty)$时$q(a) = +\infty$。

当我们插入一个数 *X* 的时候，假设 *X* 所在的区间为$[s, t]$。如图 1.3，插入后，当$a \in [s, X]$时，$q(a) = X$；当$a \in [X + 1, t]$时，$q(a) = t$。也就是区间$[s, t]$被拆分成了两个区间$[s, X]$和$[X + 1, t]$。

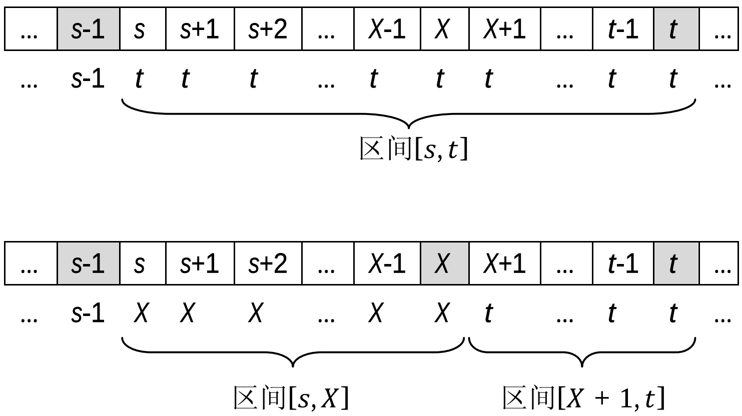

图 1.3：图中带阴影的方格表示在 S 中的元素。
当插入数 *X* 后，区间$[s, t]$被拆分成了$[s, X]$和$[X + 1, t]$。

因为只有插入操作，所以我们一直在拆分区间，而不合并区间。如果我们让时间倒流，把所有的操作按照从后往前的顺序处理，那么区间就一直都在被合并了。询问时，我们只需要找一个数在哪个区间。这让我们想起了并查集。[3]

我们把这些区间每个区间看作是一个集合，并对每个集合维护这个区间对应的 q。集合的合并和查找使用经典的算法，可以做到每次操作需要均摊近似O(1)的时间。为了方便，下文中近似地认为每次操作需要O(1)的时间。

回到原问题。每个插入操作只需O(1)的时间。对一个询问“A *Y*”，需要询问$O\left(\frac{R}{Y}\right)$个区间。因为$Y \geq 1$，所以也就是需要询问$O(R)$个区间。虽然每个区间我们可以做到只需O(1)的时间，一次询问的时间复杂度仍然高达$O(R)$。*算法 1.2* 的总时间复杂度为$O(NR)$。

显然和前面提到的*算法 1.0* 和*算法 1.1* 和一样，*算法 1.2* 也不能解决问题，甚至看上去比它们更慢。

[3]关于并查集，可以参见算法导论和国家集训队的相关论文

1.4 同时使用*算法 1.1* 和*算法 1.2*

分析一下*算法 1.2* 的瓶颈。瓶颈在于处理询问“A *Y*”时，需要处理的区间可能非常多，会发生在当 *Y* 比较小的时候。而这是在算法的一开始就决定好的，不同的 *k* 的个数有$\mathrm{O}\left(\frac{R}{Y}\right)$个，要减少询问的区间数非常困难。

但是我们注意到，瓶颈只会在 *Y* 比较小的时候出现。而大多数的情况下，需要处理的区间是比较少的。当$Y > \sqrt{R}$的时候，$\frac{R}{Y} < \sqrt{R}$。这时 *N* 次询问的时间复杂度为$\mathrm{O}(NR^{0.5})$。对于本题的规模，这个时间复杂度已经可以接受了。

因此我们可以按照 *Y* 的大小把询问分成两部分，设两部分的分界值为 *K*，当$Y > K$的时候，我们继续使用*算法 1.2*；当$Y \le K$的时候我们使用另一种算法。

算法 1.2 可以解决大多数 *Y* 的询问，剩下的 *Y* 会比较少。回想我们前面提出的*算法 1.1*，当需要维护的 *Y* 很少时很好。所以当$Y \le K$的时候，我们正好可以使用*算法 1.1*。此时我们令*算法 1.1* 中的集合 *T* 为$[1, K]$，也就是对$1 \le Y \le K$的询问维护答案。

因此我们得出*算法 1.3*：

首先顺序地处理操作，回答$Y \le K$的询问。每次插入对每个$1 \le i \le K$更新$p(i)$，需要$\mathrm{O}(K)$的时间。回答每个$Y \le K$的询问显然只需$\mathrm{O}(1)$的时间。

然后倒序地处理操作，回答$Y > K$的询问。每次插入操作要把两个集合合并，需要$\mathrm{O}(1)$的时间。询问 A *Y* 时，我们找$\mathrm{O}\left(\frac{R}{Y}\right)$个区间中的最小数，对每个区间$[a, b]$，我们查找 *a* 所在的集合，需要$\mathrm{O}(1)$的时间。因为$Y > K$，询问的时间复杂度为$\mathrm{O}\left(\frac{R}{K}\right)$。

算法 1.3 的总时间复杂度为$\mathrm{O}\left(NK + \frac{NR}{K}\right)$，其中 *K* 是一个我们设的边界值。将 *N* 和 *R* 看作常数，容易得出当$K = \sqrt{R}$时总时间复杂度最小，为$\mathrm{O}\left(N\sqrt{R}\right)$。本题中 *N* 最大 40000，*R*=500000，$N\sqrt{R}$最大约为 28284271，本算法可以完全解决本题。

1.5 小结

对这道题，我们先经过初步思考，得出了两个朴素算法：*算法 1.0* 和*算法 1.1*。它们在某些输入下会有很好的表现，但最坏情况下都太慢了，不能完全解决问题。

需要注意的是，其中*算法 1.1* 当$|T|$很小，也就是需要维护的询问很少时，会有很好的表现。

然后我们抓住问题的特征，由使被一个数除余数最小入手，得出了*算法 1.2*。*算法 1.2* 当询问中的 *Y* 比较大的时候比较快，但仍然不能完全解决问题。

算法 1.1 和*算法 1.2* 单独使用都不能完全解决问题，但是我们注意到它们可以解决这个问题中两个互补的部分。我们根据 *Y* 的大小，把询问分成两部分处理。对$Y \le \sqrt{R}$的询问使用*算法 1.1*，对$Y > \sqrt{R}$的询问使用*算法 1.2*。这样做完全解决了问题。

可见，我们解决本题的重点是，不使用统一的算法，而是同时使用这个问题的两种算法，分别解决问题中的两个互补的部分。

二、问题二[4]

2.1 问题描述

在一个平面上给定 *N* 个点。求以这 *N* 个点中的任意 4 个点为顶点，可以组成多少个边和坐标轴平行的矩形。

其中$1 \le N \le 250000$。每个点的时限最多 30s。

2.2 初步分析

虽然这道题的时限非常长，但 *N* 最大为 250000。为了解决问题，我们预期要设计出一个时间复杂度低于$\mathrm{O}(N^2)$的算法。

因为组成矩形要求边和坐标轴平行，所以只是需要点的坐标相等，我们只关心坐标的相对关系。所以我们可以把点的坐标离散化。如图 2.1，这样我们会得到一个最坏情况大小N^2的网格，输入给定的点分布在网格的格点上。

[4] 题目来源：MIT Individual Contest 2007，SPOJ RECTANGL

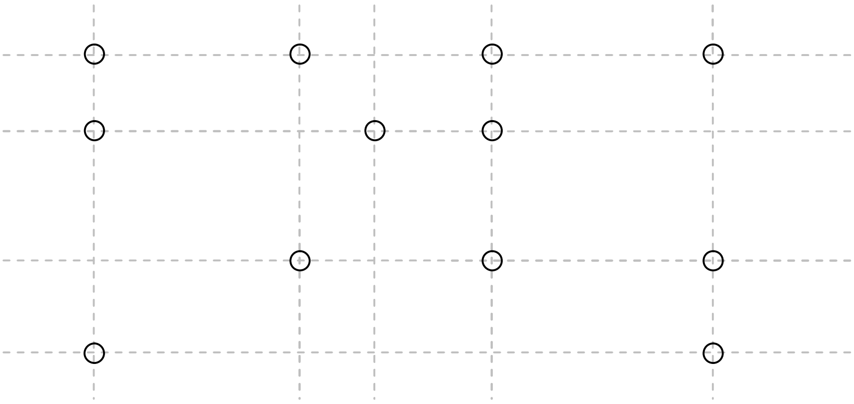

图 2.1：对 12 个点进行离散化，得到了一个4×5的网格

显然，组成矩形的 4 个点会有 2 个点在网格的一行，2 个点在网格的另一行上。因此，我们可以算出网格上每两行点组成的矩形的个数，最后把它们相加即为答案。

2.3 一个不难想到的算法

一个$O(N^2)$的算法（*算法 2.1*）不难想到。我们分别以网格的每两行$i, j (i < j)$作为矩形的上边界和下边界，计算可以组成多少个矩形。计算时，我们先枚举一行 *i*，把这一行元素的列号放进 hash。然后再去枚举另一行 *j*，统计行 *j* 中有多少点的列号在 hash 里。这样做也就是算出这两行中有多少对列号相等的点。显然，如果有 *s* 对列号相等的点，这就意味着以这两行中的点组成矩形，可以组成C_s^2个矩形。最后将每两行的矩形个数累加即是答案。

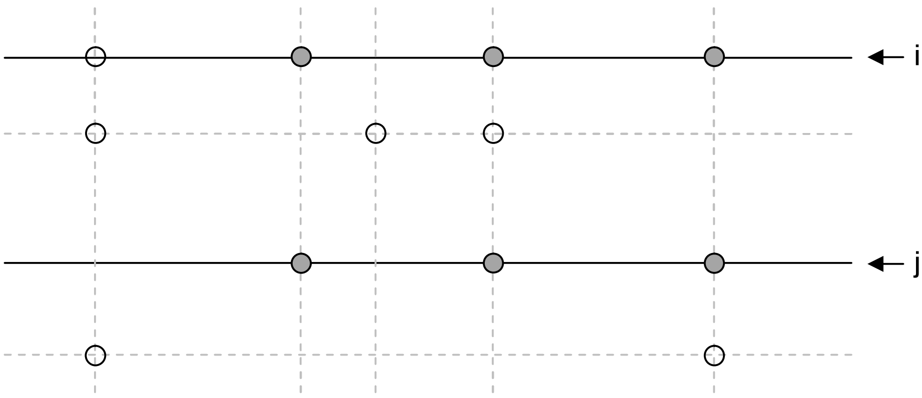

图 2.2：指针 *i* 当前为 1，*j* 当前为 3。这两行有 3 对列号相同的点，即图中用阴影标出的。这两行可以组成$C_3^2 = 3$个矩形。

这样做，我们需要在$O(N)$行中枚举一行，对每一行要处理$O(N)$个点。这$O(N)$

个点每个点最多放进 hash 一次，或者检查一次 *y* 坐标是否在 hash 中。因此，时间复杂度为$O(N^2)$。本算法没有达到我们的预期，但是当网格的行数较少的时候，它是很好的。

2.4 将问题分割成 3 部分

可以注意到当网格中每行的点数都比较多的时候，因为总点数的限制，网格的行数会很小。所以我们按每行中的点数把行分为两类，设 *K* 为分界值，当行 *x* 中的点数$> K$时，我们称行 *x* 为 A 类，否则为 B 类。显然问题就被分成了三部分：

1、 以两个 A 类行中的点组成矩形，共有多少个矩形

2、 以两个 B 类行中的点组成矩形，共有多少个矩形

3、 以一个 A 类行和一个 B 类行组成矩形，共有多少个矩形

很容易注意到，A 类行的个数必然$< \frac{N}{K}$。证明很容易。假设 A 类行的个数$\geq \frac{N}{K}$。因为每行的点数$> K$，所以 A 类行中的总点数$> \frac{N}{K} \times K = N$。而事实上 A 类行中点的个数必然$\leq N$，矛盾。

有了 A 类行的个数比较少这个限制，我们就可以对部分 1 使用*算法 2.1*。

注意到*算法 2.1* 中，我们只要先枚举一行，另一行的枚举在时间复杂度上相当于把所有的点都扫描一遍。这就允许我们在处理部分 1 时，“顺便”处理部分 3，并且不影响时间复杂度。

而对部分 2，我们有了一个新的限制，即每行中的点数$\leq K$，也就是说，每行中的点数会比较少。我们抓住问题的特征，也就是这个限制，设计一个针对每行中点数较少时比较优的算法。

2.5 对部分 2 设计另一个算法（*算法 2.2*）

部分 2 中每行中的点数较少也就意味着，以每行中任意两个不同的点为端点，组成的线段的个数也较少。以一个点作为线段 *l* 的右端点，因为一行最多有 *K* 个点，线段 *l* 的左端点可以有$O(K)$个选择。那么以所有的$O(N)$个点作为右端点，会有$O(KN)$条线段。

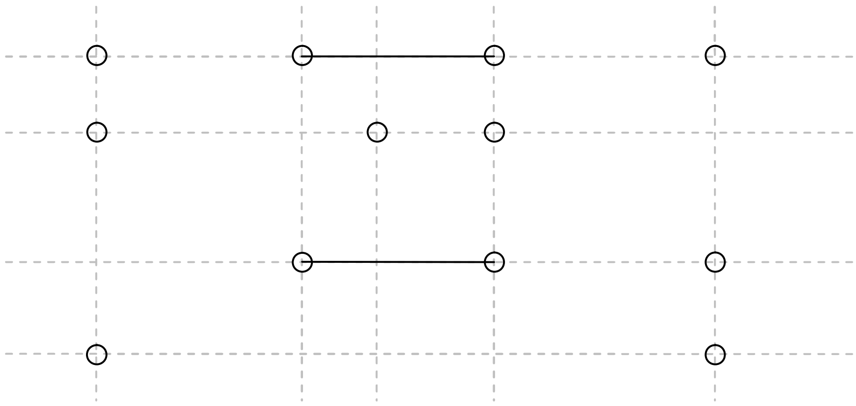

图 2.3：线段[2,4]出现了 2 次，即图中的两条黑线

显然，将所有的行上的线段放在一起，对每种线段$[a,b]$统计它出现的次数 s。这里的 *a* 和 *b* 是指同一行上两个点的列号。容易得出答案就是$\sum C_s^2$。统计这样的线段出现的次数，很容易想到可以使用 hash，时间复杂度为$O(KN)$。但注意到空间复杂度同样为$O(KN)$。

不难想到一个减小空间复杂度的方法是：先确定线段的右端点 *b*，然后将所有右端点为 *b* 的线段放在一起考虑，把它们的左端点放进 hash。

因为现在只要 hash 一个端点，所以空间被降到了$O(N)$。而每个点和原来一样都只被当作右端点考虑了一次，因此时间复杂度不变，为$O(KN)$。

2.6 问题的解决

至此 3 个部分都得到了较好的解决，我们将它们合并起来考虑。对部分 1 和部分 3 我们使用算法 2.1，时间复杂度为$O\left(\frac{N^2}{K}\right)$。对部分 2 我们使用*算法 2.2*，时间复杂度为$O(KN)$。总时间复杂度为$O\left(\frac{N^2}{K}+KN\right)$。当 K 取$N^{0.5}$时，时间复杂度达到最小，为$O(N^{1.5})$，可以解决本题。

2.7 小结

我们经过简单的初步分析后，很轻松地得出了*算法 2.1*。它不能完全解决问题，但当行数比较少的时候会很好。我们根据*算法 2.1* 的这个优势，把问题按每行点数的多少分成了 3 部分。对部分 1 和部分 3 我们是使用*算法 2.1*。而对部分

2，我们根据它的特征，设计出了一种针对这部分很快的*算法 2.2*。然后我们同时使用算法 2.1 和算法 2.2，得到了一个总时间复杂度$O(N^{1.5})$的算法，解决了问题。

而假如我们单独使用算法 2.1 或算法 2.2，都将得到最坏情况下$O(N^2)$的算法，不能完全解决问题。可见这种算法复合的方法在本题的解决中的重要性。

本题和问题一一样，都是将两种相对简单的算法进行了复合，使用于问题的不同部分，但部分的划分没有上一题那么明显。能这样将问题进行划分，需要我们敏锐的观察力和扎实的基本功。

三、总结

一个问题往往可以被看作是由若干个部分组成起来的。注意这里所说的部分是相对并列的。我们通常对这些部分使用统一的算法。而有时这个问题可以使用多种算法解决，并且当这些算法应用在问题中不同特征的部分时，会有不同的效果。这时我们就可以将这些算法复合，对问题的不同部分，根据它们的特征分别选择使用对这个部分较优的算法。这就是本文所讲的算法复合的方法。

对本文中的两个问题，我们都使用了这种方法。问题一中我们得出了两个最坏情况分别是$O(N^2)$和$O(NR)$的算法。它们都不能解决问题，但它们分别针对问题的两个部分会有很好的效果。于是我们对问题的两部分分别使用这两种算法，最终得到了$O\left(N\sqrt{R}\right)$的算法，使问题得到了较好的解决。问题二与之类似，我们将两个最坏情况下$O(N^2)$的算法复合起来，得到了一个$O(N^{1.5})$的算法。

我们注意到两个算法合并起来后，我们很“神奇”地得到了一个更优的算法。这是因为这两种算法具有互补的优势，而我们把问题分成了若干部分，对每一部分根据其特征使用较优的算法，就使得两种算法的优势得到了结合。

每个算法都有各自的优势和劣势。如果我们取长补短，充分利用它们的优势，也许就将会得出总体更优的算法。这种取长补短的思想是非常重要的。

本文讲的是一类算法复合的方法。作为一种方法，我们在解题时可以选择使用。同时，在解题时不断总结，形成一般性的方法是很重要的。

参考文献

[1] Introduction to Algorithms 作者：Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein
[2] 2005-2007 年国家集训队论文及作业