回到起点——一种突破性思维

南京市外国语学校　朱泽园

[关键字]　起点　类比　分离集合森林

[摘要]

高层次的信息学竞赛已经不是单纯一个算法、一种数据结构间的较量，而是对思维方法、创新能力的考验。本文旨在深刻剖析一种突破性思维——回到起点，亦即敢于放弃当前成果，回到初步分析处展开理性思考的可贵品质。

本文第一章提出一个已被经典算法解决的普通题。第二章对这道题展开讨论，简述了经典的解决方案，而后模拟本文主线描绘的思维方式进行思考，提出一个被遗忘的简单算法，并进行优化和类比，精确计算复杂度后问题被完美解决。

第三章对本思维方式在两个例题中的应用作了对比和升华，辩证地以前进性和曲折性的统一阐述了这类思想的重要性。

[目录]

[正文]

§1 问题的提出

§1.1 问题描述 [USACO 2.1 Shaping Regions 改编]

N 个不同颜色的不透明长方形（1≤N≤3000）被放置在一张长宽分别为 A、B 的白纸上。这些长方形被放置时，保证了它们的边与白纸的边缘平行。所有的长方形都放置在白纸内，所以我们会看到不同形状的各种颜色。坐标系统的原点（0,0），设在这张白纸的左下角，而坐标轴则平行于纸边缘。

输入：

第 1 行：A ， B 和 N, 由空格分开。

第 2~N+1 行：按照从下往上的顺序，每行输入的是一个长方形的放置。为五个数 llx, lly, urx, ury, color 这是长方形的左下角坐标，右上角坐标和颜色。其中 color 为整数。0<=llx,urx<=A，0<=lly,ury<=B。所有坐标（包括 A、B）都是 0 至 10^9 的实数。

颜色 1 和底部白纸的颜色相同。

输出：

输出文件应该包含一个所有能被看到颜色连同该颜色的面积（保留小数点后三位）的列表（即使颜色的编号不是连续的）。按 color 的增序顺序输出，不要输出面积为 0 的颜色。

样例输入：

20 20 3
2 2 18 18 2
0 8 19 19 3
8 0 10 19 4

样例输出

1 91.000
2 84.000
3 187.000
4 38.000

§1.2 问题的初步分析——离散化

自 IOI1998 的 Picture 问题始，和平面放置矩形有关的问题已在信息学竞赛中屡见不鲜，其最基本的预处理操作是将平面离散化。如图 1

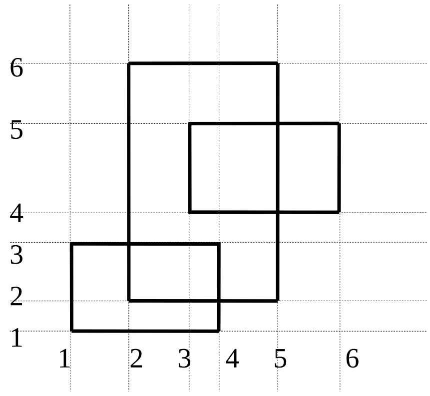


图 1

将平面理解成网格，也就是说将所有在矩形坐标中出现过的 x 坐标（y 坐标）提出，排序后删除重复，并按顺序编号。任意两个相邻格点连成的线段，称之为“元线段”；两个相邻离散过的 x 坐标（y 坐标）之间的区域，称之为“离散列”（“离散行”）；任意一个离散行和离散列的公共部分称之为“离散格”。对每个矩形坐标相应地存在一个格点坐标与之对应。

这有助于处理坐标为实数，或者范围超过 longint 的问题。离散化之后，任意矩形顶点坐标均可表示为 1 到 2N 之间的整数：利用 hash 策略可以将实数坐标在 O(1)时间内转化为对应的整点坐标，更显然地，直接读表可以在 O(1)时间内将格点坐标转化为原矩形坐标。因此后文只考虑坐标为整数，且范围在 1 到 2N 之间的 N 个格点矩形的问题。

§1.3 一个朴素的想法

最朴素算法的实现取自于简单的灌水（Floodfill）算法。就是用 2Nx2N 的数组，分别记录每个离散格的颜色，初始时全部无色。自顶至下处理每一个矩形，将其覆盖之下无色的部分全部填上颜色。

考虑到空间可能无法承受，改进的措施是每次处理一个离散行（2N 的一维数组），自顶向下处理每个矩形，如果这个矩形与当前离散行有公共部分，那么就和前面类似地，将无色部分涂色，如图 2：

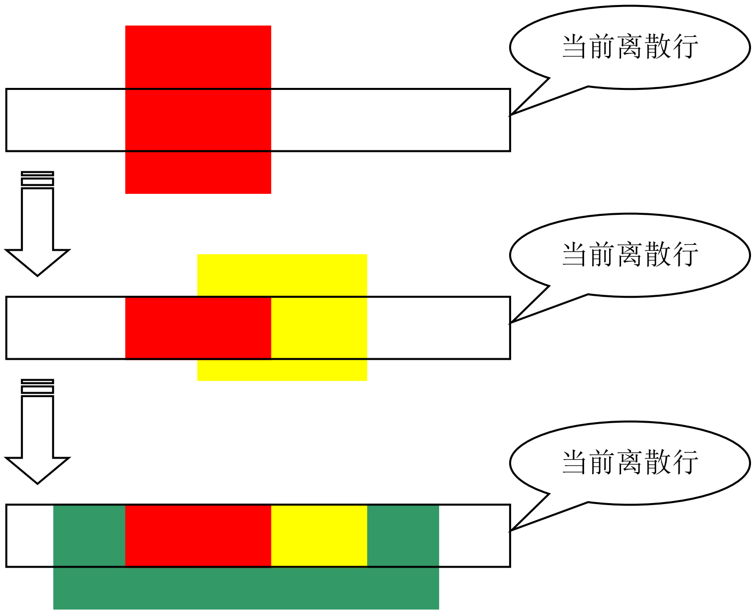


图 2

该方法基本不会增加程序运行时间，只是时间复杂度系数稍微增加了些，并不影响大局。

§2 问题的解决

§2.1 经典算法

解决这类问题的经典算法莫过于线段树。同样是对每个离散行做处理，该算法利用了 O(N)空间的树型数据结构。如图 3，构建一个线段树：

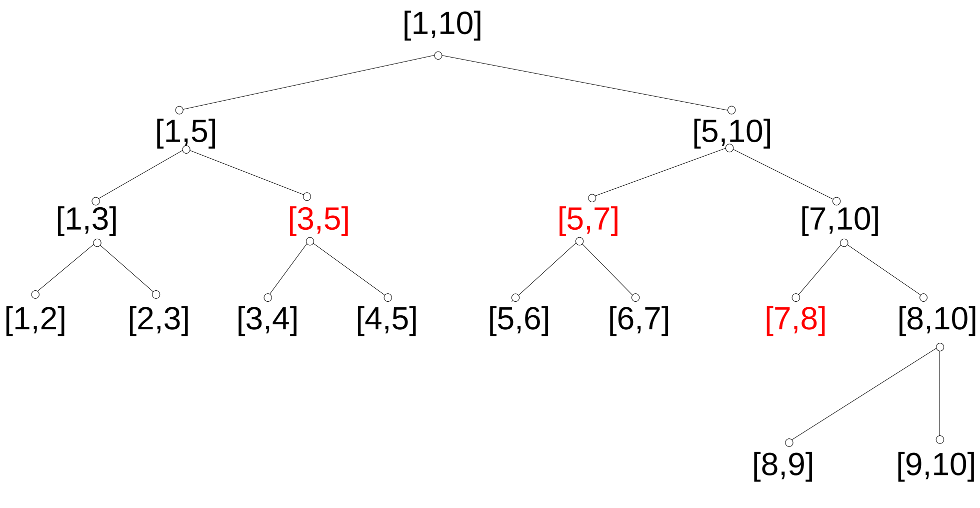


图 3

该数据结构主要支持的操作是，给定线段[l,r]和属性 X 组成的操作对(l,r,X)，意思是将区间[l,r]赋予属性 X。称“在 P 节点上进行(l,r,X)操作”为 P(l,r,X)，具体操作如下：

```
Operation P(l,r,X)
  If [l,r]与当前节点 P 对应的区间交集为空 Then Return
  If [l,r]可以完全覆盖当前结点 P 对应的区间
    Then 将属性 X 记录在该结点
    Else 依次对 P 的左右子结点操作 P.leftchild(l,r,X), P.rightchild(l,r,X)
 End If
End Operation
```

例如图 3 中红色的部分是处理 Root(3,8,X)操作后被赋予 X 属性的结点。可以证明对每个操作 Root(l,r,X)，其时间复杂度为 O(logN)（N 是叶结点个数，也就是区间范围）。

观察本题需要这个数据结构支持怎样的特殊操作：

1、插入一条颜色为 X 的线段[l,r]，该区间[l,r]上原有颜色不被替换，其余部分染上颜色 X。

2、返回所有颜色当前的覆盖量。（该操作只将在该离散行处理完毕后一次性调用）

操作 2 只需将线段树全部遍历一遍即可求得，时间复杂度 O(N)。至于操作 1，对线段树上的每条线段标记颜色不是难事儿，只需要将前文所述的属性 X 定义其为 color flag，用来标记颜色。因此主要难点在于不破坏区间上的已有颜色。

其实这一点也不难实现，类似地我们给出这个操作的具体过程：

```
Operation P(l,r,X)
  If [l,r]与当前节点 P 对应的区间交集为空 Then Return
  If 节点 P 已经被某种颜色覆盖 Then Return
  If [l,r]可以完全覆盖当前结点 P 对应的区间
    Then 将颜色 X 记录在该结点
    Else 依次对 P 的左右子结点操作 P.leftchild(l,r,X), P.rightchild(l,r,X)
 End If
End Operation
```

至此该算法的空间复杂度为 O(N)，时间复杂度为 $O(N^2\log N)$，在某种程度上是一个可以通过本题数据的优秀算法。

§2.2 另类算法

避开繁文缛节，我们回到起点，分析一下原始算法，就是“§1.3 一个朴素的想法”内的算法。有效解决问题往往有两条途径，一个是使用高级算法或数据结构，另一个就是分析原有的算法或数据结构，找出冗余进行针对性改进。

回顾一下之前的算法

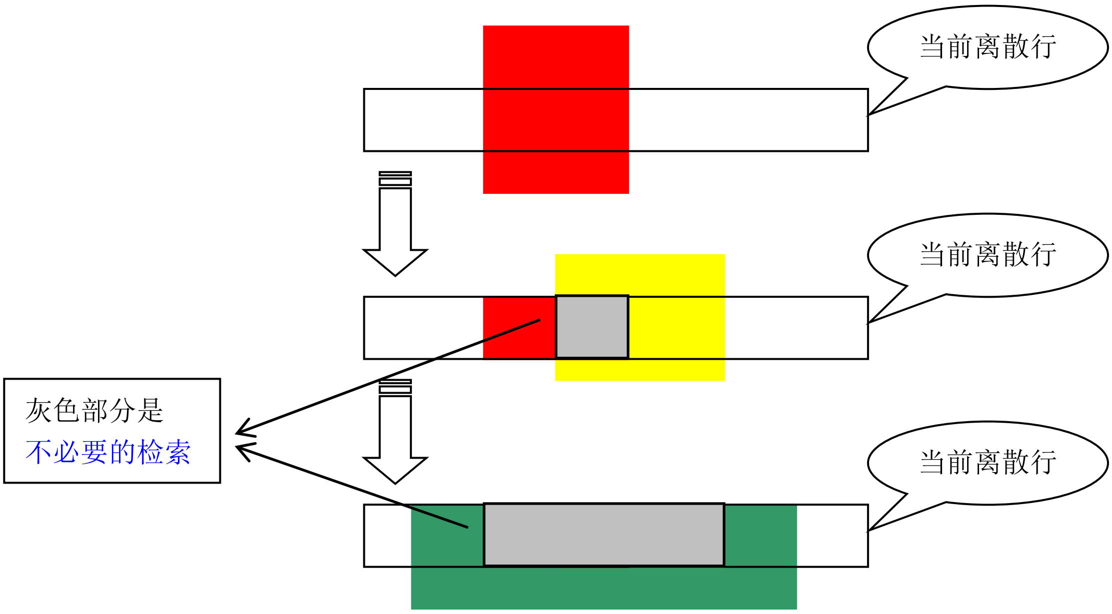


图 4

迅速找到冗余：处理当前线段（矩形与当前离散行的交集）时，对已经覆盖的线段，不能迅速避开，需要无畏的比较。改进措施如下：

设立 next 数组，初始时 next[i]=i+1；另还有 color 数组，color[i]记录该离散行的第 i 个离散格当前的颜色。设当前插入的线段为[*l*,r]，即占用编号为 *l* 至 r-1 的离散格：

1. i←*l*
2. 查看 color[i]，如果它未被着色，那么将其着色。
3. 移动指针 i←next[i]，并记录指针经过的位置。
4. 如果 i 指针没有越过边界 r-1，跳转 2，否则跳转 5。
5. 将所有指针经过位置 x 的 next[x]置为 next[r-1]。

下面的图 5 将给出一组覆盖的例子：（插入的线段与图 4 稍有类似）

第一次用颜色 1 的线段覆盖[2,6)，因为 next 数组完全是初始值，因此 i 指针从 *l*=2 一直移动到 i=r，color[2..5]被置为颜色 1，next[2..5]被置为 next[r-1]=r=6。

第二次用颜色 2 的线段覆盖[4,7)，i 指针来到 *l*=4，color[4]不被修改，i←next[i] = next[4] = 6。接下来 color[6]被修改，指针移动到 next[6]=7=r，移动完毕。指针经过了 4，因此 next[4]被置为 next[r-1]=r=7。

第三次用颜色 3 的线段覆盖[1,9)，i 指针先来到 1，修改 color[1]，i←next[i]来到 2，观察到 color[2]已经有颜色以后先后转战 i←next[2]=6，i←next[6]=7。至此可以修改 color[7]以及 color[8]，并且最终在 i=9 的时候停止。过程中 next[1]、next[2]、next[6]、next[7]都需要被修改为 next[8]=9。

i	1	2	3	4	5	6	7	8
next[i]	2	3	4	5	6	7	8	9
color[i]	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL

颜色 1 覆盖 [2,6)

i	1	2	3	4	5	6	7	8
next[i]	2	6	6	6	6	7	8	9
color[i]	NIL	1	1	1	1	NIL	NIL	NIL

颜色 2 覆盖 [4,7)

i	1	2	3	4	5	6	7	8
next[i]	2	6	6	7	6	7	8	9
color[i]	NIL	1	1	1	1	2	NIL	NIL

颜色 3 覆盖 [1,9)

i	1	2	3	4	5	6	7	8
next[i]	9	9	6	7	6	9	9	9
color[i]	3	1	1	1	1	2	3	3

图 5

观察上述算法，其精髓在于将已染色的相邻离散格合并，或者对其“预定”合并，也就是做上标记，使其在以后的处理中自动合并——平摊思想。这一想法源于“带路径压缩按秩合并的分离集合森林”，也就是高效的俗称 “并查集”算法。注意到当前算法并未完全进行路径压缩，而且没有按秩合并，因此该算法有待完善。（其实可以证明本算法在大部分情况下时间复杂度维持在 N^2 左右，不易退化）

§2.3 通过完整的路径压缩完善算法

将相邻的已染色线段看成一个集合，这样在[l,r)的检索过程中，遇到已染色线段可以大跨步跨越。我们的目标是让这个跨越在平摊 O(1)的时间内完成。算法改进如下：

设立 next 数组，初始时 next[i]=i+1；同时设立 p 数组，初始时 p[i]=i（p 数组类似并查集算法中的父节点指针）；另还有 color 数组，color[i]记录该离散行的第 i 个离散格当前的颜色。设当前插入的线段为[l,r]，即占用编号为 l 至 r-1 的离散格：

1. i←l
2. 查看 color[i]，如果它未被着色，那么将其着色。
3. 将当前指针位置记录
4. 如果 i 指针已经越过边界 r-1，跳转 7
5. 如果 p[i]!=i，i←p[i]；否则 i←next[i]。
6. 跳转 2
7. 将所有指针经过位置 x 的 p[x]置为 p[r-1]，next[x]置为 next[r-1]。

为了建立分离集合森林，我们作如下定义：

1、用 i 代表编号为 i 的节点

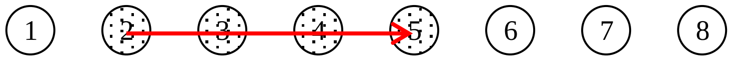
2、将 p[i]定义为节点 i 的父节点

3、父节点为自身的节点定义为根节点。

注意：对于 p[i]!=i 的节点，也就是分离集合森林中的非根节点，其 next[i] 已经失去意义。而对于根节点，必然有 next[i] = i+1。这一点将为 2.4 节服务。

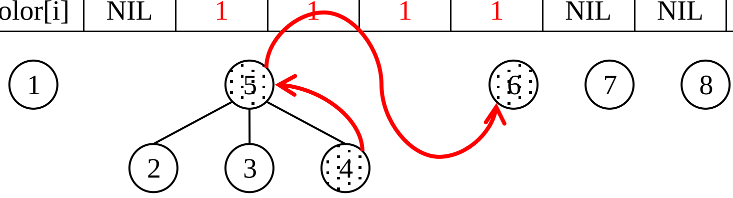
与图 5 相同，这里仍对上文数据给出图示及文字解释（因为 next/color 的定义与 §2.2 类似，因此对其改变我们不多赘述，将重点放在分离集合森林的构建）：

i	1	2	3	4	5	6	7	8
p	1	2	3	4	5	6	7	8
next[i]	2	3	4	5	6	7	8	9
color[i]	NIL	NIL	NIL	NIL	NIL	NIL	NIL	NIL



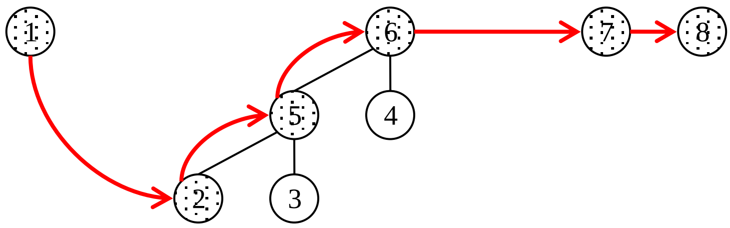
颜色 1
覆盖
[2,6)

i	1	2	3	4	5	6	7	8
p	1	5	5	5	5	6	7	8
next[i]	2	~~6~~	~~6~~	~~6~~	6	7	8	9
color[i]	NIL	1	1	1	1	NIL	NIL	NIL



颜色 2
覆盖
[4,7)

i	1	2	3	4	5	6	7	8
p	1	5	5	6	6	6	7	8
next[i]	2	~~6~~	~~6~~	~~7~~	~~7~~	7	8	9
color[i]	NIL	1	1	1	1	2	NIL	NIL



颜色 3
覆盖
[1,9)

i	1	2	3	4	5	6	7	8
p	8	8	5	6	8	8	8	8
next[i]	~~9~~	~~9~~	~~6~~	~~7~~	~~9~~	~~9~~	~~9~~	9
color[i]	3	1	1	1	1	2	3	3

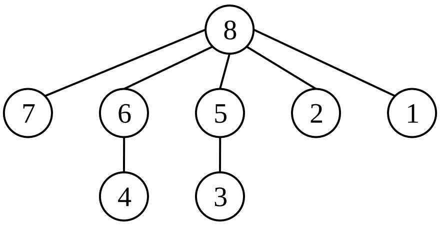


图 6

第一次用颜色 1 的线段覆盖[2,6)，因为所有节点自成集合，故本次覆盖操作

将 2..5 节点合并，***5 为他们的根节点（代表元）***，显然 next[2..4]失去其意义（在表格中用 X 表示）。

第二次用颜色 2 的线段覆盖[4,7)，i 指针来到 l=4，根据分离集合森林的定义，需要找到其所在集合的根节点（该集合的代表元），也就是节点 p[4]=5。紧接跳转下一个集合，也就是节点 6。指针 i 移动到 6，并且修改 color[6]。最后压缩路径——前面所经过的所有点 x 的 p[x]都指向 6——5 和 6 所在的集合合并，***6 成为新的根节点***。

第三次用颜色 3 的线段覆盖[1,9)，i 指针先来到 1，修改 color[1]后跨入 next[1]=2 所在的集合，发现其已染色，通过 i←p[p[2]]=6 以及 i←next[i]=7 跨越该集合，并对 7、8 两个独立的集合染色。最后 1、2、7、8 所在集合进行总合并。***8 为新的集合的根节点（代表元）***。

§2.4 秩的建立

所谓秩，就是一个集合(树)所含的节点个数。对分离集合的合并，采取将节点数多的集合(树)，附着在节点数少的集合(树)上的策略，可以将集合的合并复杂度降至平摊的 O(1)。这一个过程需要在上文蓝色粗斜体部分，即确定合并后新根节点的部分稍加修改，选取秩最大的树根作为新的根节点（代表元）即可。注意到这时对根节点不一定有 next[i] = i+1。

一共有最多 2N 个集合，合并次数不超过 2N-1，对当前离散行的处理时间复杂度仅为 O(N)，整个算法的时间复杂度为 $O(N^2)$。

§2.5 小结

摒弃当今主流的线段树算法，回到起点，对朴素算法进行改进的思想实属不易。但其后算法的完全建立，思维过程极其缜密：首先找到冗余进行修改，旨在**量变**——某种意义上提高算法对部分数据的运行效率；发现算法的优越之处后，展开**类比**，将类似数据结构、算法的自身优点吸取，并进行改进；最后大步伐迈向让算法复杂度**质变**的目标。

§3 总结

希腊传说弗里吉亚国国王打了一个戈尔迪之结，预言称能解开它的人将统治整个亚洲。亚历山大大帝认为它在预言中找到了自己的命运归宿，拿起长剑，径直将结劈开。有人说亚历山大将绳劈开是“作弊”的行为，但或许奥林匹斯山诸神寻找的就是如此少说多做的决断力？

面对**多元化**的世界，我们总会有这种“**答案意识过强**”的状态，就是这种倾向：对答案进行快速的设想，迅速沿那条道兴高采烈地出发，却并没在意先前设想的正确性、全面性。被自己创造的无形障碍蒙蔽实乃可惜，因此我们不能因片面的认识，或对放弃眼前利益的恐惧而裹足不前，偶尔我们也需要此类**果敢与魄力**。这正是**前进性和曲折性的统一**。

问题的表示往往比答案更重要，答案不过乃数学或实验。要提出新的问题、新的可能性、从某个新的角度考虑一个旧问题，都要求创造性的想象力，**回到起点**对问题重新定义，这才是真正的科学进步之所在。

[参考资料]

i. 《Introduction to Algorithms》
by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein
ii. 《The Art of Computer Programming》
by Donald E. Knuth
iii. 《The Computer Science and Engineering Handbook》
by Allen B. Tucker, etc.
iv. 《实用算法的分析和程序设计》
by 吴文虎 王建德
v. http://get-me.to/oi 辛韬同学提供的 BalticOI2004 解答

[感谢]

1、感谢辽宁省的辛韬同学对我的大力帮助（包括部分算法和证明）。

2、感谢江苏省青少年信息学（计算机）奥赛委员会教练组全体老师对我的指导。

3、感谢我的母校江苏省南京市外国语学校的老师给我的支持。

4、感谢我的家人给我的关怀。

5、感谢所有的信息学好友！