《Graduate》设计报告

刘汝佳

【题目背景】

本题是 NOI2004 第二试的最后一题，也是最不易把握的一题。本题来源于生活，是命题者经常头疼的一个问题。本题是开放的，虽然命题者采取的方法并不总是表现很好，但是对于各种常见数据都比较有效。这些数据并非针对算法的优势所出，它们本身具有一定典型性。本题的目的是让选手在比赛规定的时间里想一个虽不完美，但是实际效果并不错的方法，不管是写程序、手算或是二者结合。事实上，checker 程序被设计成具有丰富信息（如拼接图），就是为了方便选手使用，让解题的途径多样化。

【基本思路】

约定：一个玩具的面积为它占用的实心格子数，闭包为包含它的最小矩形。

本题的方法很多，其中不少很直观，但是却难以在短时间内编程实现，而且效果也不一定好。由于评分方法比较松，只要和标准答案相差不是太远都可以得不少分数，因此一个相当实用的方法为：

1. 编程对各个数据进行必要的统计工作，如玩具面积的分布、闭包面积分布、所有玩具的总面积。有了这些统计值，加上人眼对玩具形状的大概情况（是规则的还是不规则的，轮廓光滑吗？实心还是空心？…），可以初步估算出一个比较理想的结果。比如数据 5，有一个玩具的闭包为 18*121，而所有玩具的总面积恰好为 2178=18*121，那么有理由猜想最优解就是 18*121。
2. 有了对解的估计，下一步就是把边界固定下来。例如数据 8 的总大小是 10000，而且玩具大都是 1*1 和 1*2，因此可以把边界固定为 100*100，放置时不许超出此边界，如果所有玩具都成功放置，则一定找到了最优解。如果找不到解，需要把边界稍微扩大点。扩大得太厉害会得到很差的解，而太微小可能仍然得不到解，这就需要选手根据数据特点和对自己算法的把握进行决策了。

现在，问题的关键是要设计一个程序 graduate(i, h, w)，即把第 i 个点的所有玩具放在长为 h 宽为 w 的矩形中。如果成功则应产生文件 graduatei.out。

【核心程序】

算法思想很简单。先按照面积从大到小把各个玩具进行排序，然后依次放置各个玩具。放置玩具的方式是从左到右从上到下试各个位置是否能放入。在每个位置上试验 4 种旋转方案，如果有多种方案都可以放下，取闭包最小的一种。为了加快速度，可以让玩具面积不变时接着上次找到位置继续寻找，而不是从头找（很可能无法填下）。这个思想的实现方式不唯一，对于细节的不同处理会导致不同的结果，读者不妨一试。

按照这个思路写的参考程序可以得到 91%的分数，已经相当优秀了。如果用手算来调整数据 3 和数据 10，可以得到 95%的分数，而数据 7 剩下的 5 分需要比较大的耐心、灵活性，或者一个新的程序。

命题者试过好几种不同的算法，唯有这种实现简单，效果又好。这个方法成功的关键是设置好边界。这一步通常需要有一定的人工分析。例如数据 4，5 就是看闭包最大的玩具，而数据 6、7、8、9、10 更主要是看总面积。由于算法的特点所限，可能一个大的边界出不来，小的反而出来，无法用二分法来确定边界。谨慎一点的选手可以用枚举的方法，但是如果程序实现不好可能会导致时间开销无法忍受。

【数据分析】

graduate1.in

手工生成的数据。三个玩具的图案分别是 N，O，I 三个字母。难度并不大，观察即可。

graduate2.in

手工生成的数据。很容易观察出来，一个套一个即可。

graduate3.in

手工生成的数据。由于只有 9 块，而且都是四连的“标准图形”，也并不难手算出来。

之所以出这样一个数据还有一个意图，见数据 10。

总面积是 4*9=36，拼出一个 6*6 方阵。

graduate4.in

手工生成的数据。该数据的特点是第一块的纵横跨度达到了 7*36，这是解的下界。

猜想其他玩具都可以插到缝隙里从而达到这个下界。事实上这的确可以办到，步骤如下：

1. 注意到其他 9 块中只有第二块的跨度很大，所以应先把它嵌入。用 checker 不难发现这种嵌入是唯一的。

2. 用 checker 不难试出：第四块也只有一种放置方法。

3. 用 checker 不难发现：第五块贴近第四块几乎不会造成什么浪费，暂时把它也固定下来。

4. 其他几块放置方法并不唯一，但是由于空位已经不多，多试几次也不难摆好。

graduate5.in

手工生成的数据。该数据和 graduate4 很类似，第一块的纵横跨度达到了 18*121，这是解的下界。这是最难的一个手算数据。不过虽然有 26 个块，但是只有 4 个是需要精心策划的，其他的影响要小得多。

1. 第四块非常“蜿蜒曲折”，经实验，它只有一种放置方法。

2. 第 5、6 块的“齿”吻合得相当好，而且合并以后能放在第 7 块里，在不少地方卡得比较死，因此这三块以后一起考虑。

3. 第 2 块可以装在第 10 块里，以后一起处理

4. 从 K 开始都是一些字母形状，很难相互卡得很死

5. 只要 1~3 做好，其他即使随意放置效果也不会很差。稍微留心一下也不难得到 18*121 的方案。

graduate6.in

标准数据生成器生成。总规模为 80*20, 共 4 个玩具，grow_factor 均为 1，占满所有区域。

这个数据的直观特点：玩具都是“实心”的，不可能套在中间或者很蜿蜒的“相互卡住”，而应该着眼于外轮廓上。虽然初看起来比较复杂，但实际上这个数据比数据 4 和 5 都要容易，只要每次尽量卡住就可以了具体方法：2 和 4 对接，会留

下一个狭长的通道，然后把 1 和 3 嵌在里面可以得到一个 80*20 的方案。只要思路对了，加一点耐心都是满分，否则根本拼不在一块。

graduate7.in

标准数据生成器生成。规模为 100*100，共 50 个玩具，grow_factor 均为 1，占满所有区域。

由于最大的跨度是 100，总面积又是 10000，所以猜想可以铺成 100*100 的。这是整道题目最难的数据。和数据 6 一样，玩具是实心的，猜想可以通过“吻合外轮廓”来得到下界，但是由于有多达 50 块，需要一个自动化工具来进行。前面介绍的算法在这里效果不大好，目前只能得到一个 100*220 的解。一个可行的方法是从 S 入手，借助核心程序找最合适的块和它拼接在一起。这个过程需要选手耐心的尝试，手工控制程序中玩具的排列顺序（而不是简单的按照面积或者闭包面积排序），每次加入一个或者两个块，让“中洞”最少。这个“中洞”的多少可以用人眼观察，但是最好用程序进行预判（种子填充是一个比较好的途径）。当排列顺序安排好之后（需要安排前 35 个玩具，如果比这个少，需要加入随机化），核心程序仍然可以得到最优解。

graduate8.in

标准数据生成器生成。规模为 100*100，共 5000 个玩具，grow factor 均为 1，占满所有区域。

虽然玩具很多，但是每个都很小（约有 2200 个 1*1 的），应该先出来大的，再铺小的。1*1 可以随时插空。

graduate9.in

标准数据生成器生成。规模为 100*100，共 5000 个玩具，grow factor 除了中间一个为 1 外其余为 0，占满所有区域。可以看到前面和后面的大部分都是 1*1，但是中间有一些东西。其实这些东西根本就不需要卡死。只有不到 200 个块的面积超过 5。

graduate10.in

特殊数据生成器生成。数据 3 的每个玩具个数扩大 400 倍。20*20 个 6*6 拼一块儿就可以了。需要写一个程序来计算各块的旋转方式和位置。

【一般数据生成方法】

- 输入参数：拼接后规模（长 h、宽 w），玩具个数 n，玩具总面积（不超过 h*w），每个玩具的生长因子 grow_factor
- 算法：随机撒 n 个点，作为每个玩具的“starting square”，然后每次随机选择一个玩具进行“生长”，即增加一个格子。玩具被选择生长的概率和 grow_factor 成正比，而在生长的时候保证玩具 4-连通。如果有多个候选的新增格子，尽量选择让玩具跨度比较大或者比较“崎岖”的形状。具体评价函数可以有多种。

【文件说明】

附件中有以下文件：

- 数据：

graduate*.in	输入
graduate*.ans	参考答案
graduate*.out	参考方案

- 程序：

graduate_check.cpp	提供给选手的 check 程序。
graduate_e.cpp	评测系统用的 evaluator
graduate_data_gen_letter.cpp	一般数据生成器
graduate_data_gen_number.cpp	一般数据生成器
graduate_data_convertor_letter.cpp	转换器（根据拼接方案生成输入。随机旋转每个块）
graduate_data_convertor_number.cpp	转换器
graduate_data_gen_10.cpp	第 10 个数据生成器
graduate.cpp	可以解决所有数据的程序，参数已在程序中设定

- 数据生成

1-2	手工输入*.in

3-5	手工输入*.txt，运行 graduate_data_convertor_letter，得到.in
6-7	运行 graduate_data_gen_letter 得到*.txt，graduate_data_convertor_letter，得到.in
8-9	运行 graduate_data_gen_number 得到*.txt，graduate_data_convertor_number，得到.in
10	运行 graduate_data_gen_10

注意：程序的随机数种子没有经过初始化，因此在 VC6 下每次运行结果相同。用此法可以分析数据。