HZ BOOKS
华章教育

本书用于ACM-ICPC等程序设计竞赛的训练

ACM-ICPC 世界总决赛试题解析 (2004~2011年)

Solutions and Analyses to ACM-ICPC World Finals(2004~2011)

吴永辉 王建德 等编著

机械工业出版社
China Machine Press

ACM-ICPC 世界总决赛试题解析 (2004~2011年)

Solutions and Analyses to ACM-ICPC World Finals(2004~2011)

吴永辉 王建德 杨溢 李明韫 等编著

机械工业出版社
China Machine Press

本书给出2004～2011年ACM国际大学生程序设计竞赛世界总决赛的所有试题的解析。本书将总决赛试题按年度划分，每一年度的总决赛试题为一章，而每一道试题作为一节。试题全部翻译成中文，试题解析以解题策略为主轴，给出详尽、细致的解析和带有详尽注解的程序代码。这样做使得本书可以面向各个阶层的广大读者，不仅要让编程高手从中受益，而且也要让刚入门的同学能轻松地学习，有效地提高通过编程解决问题的能力。

图书在版编目（CIP）数据

ACM-ICPC世界总决赛试题解析（2004～2011年）/吴永辉等编著. —北京：机械工业出版社，2012.8

ISBN 978-7-111-39094-7

Ⅰ.A…　Ⅱ.吴…　Ⅲ.程序设计—竞赛题—题解　Ⅳ.TP311.1-44

中国版本图书馆CIP数据核字（2012）第148658号

机械工业出版社（北京市西城区百万庄大街22号　邮政编码100037）
责任编辑：朱秀英
北京瑞德印刷有限公司印刷
2012年8月第1版第1次印刷
185mm×260mm • 23.75印张
标准书号：ISBN 978-7-111-39094-7
定价：55.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991；88361066
购书热线：（010）68326294；88379649；68995259
投稿热线：（010）88379604
读者信箱：hzjsj@hzbook.com

前 言

ACM 国际大学生程序设计竞赛（ACM International Collegiate Programming Contest，ACM-ICPC）是由国际计算机协会（Association for Computing Machinery，ACM）主办的国际性大学生计算机学科竞赛，旨在向全世界大学生提供一个展示团队创新精神和历练编程解题能力的机会。参赛对象为本科一年级到硕士研究生二年级的学生。竞赛以大学为单位，3 人一队，每年秋季先在各大洲进行各级预选赛，从中选拔优胜队参加第二年春季的世界总决赛。这项赛事受到全球大学的普遍重视。自 1977 年以来已经连续举办了 36 届，赛事规模增长迅速。在 1996～1997 赛季，ACM 在上海举办第一场在中国大陆的亚洲区预赛，而那一年在全球只有 560 所大学的 840 支队伍参加了比赛；而到了 2009～2010 赛季，全球 6 大洲共 82 个国家和地区，1931 所大学的 7319 支队伍参加了各级比赛。这些年来，ACM-ICPC 在中国大陆的比赛规模逐年扩大，参加的学校和参赛的队伍逐年增加，影响也越来越大。1996～2001 年上海大学连续主办了六届亚洲区域赛；2002～2004 年中国大陆增加到两个赛区；2005～2006 年增加到 3 个赛区；2007 年增加到 4 个赛区；而从 2008 年起，中国大陆每年秋季有 5 个赛区进行预赛。现在 ACM 国际大学生程序设计竞赛已经成为全世界范围内历时长久、规模宏大、影响广泛的权威性的大学生程序设计竞赛。

自 1996 年 ACM-ICPC 竞赛进入中国大陆时，复旦大学每年都参加这项赛事，自 2001 年每年入围 ACM-ICPC 世界总决赛；且 2006 年至今每年为 ACM-ICPC 亚洲区预赛命题。

为了调研 ACM-ICPC 在大学生中的影响，我们特意关注了网上大学生们对 ACM 活动的议论，顿感一股清风扑面而来：有的学生一投入 ACM 活动就遁入“痴迷”状态，说自己开始了一场与 ACM 的“精神恋爱”；有的学生动情地说：有一种感动，叫做 ACM，在这种感动中有坚持、有承受、有成长。甚至有的学生将 ACM 活动解读成“ACM 精神”。我们非常赞同这些学生的说法，ACM 活动确实培育了这样的一种精神：从寻求梦想、追随激情到精诚合作、永不言败的团队精神；从汇通中英文和相关知识到“真刀真枪做题”的实践精神；从浅尝辄止到深思熟虑、别出心裁、独具慧眼、研辨解题方法是非的创新精神。ACM 赛事受到各国大学生的青睐，在 ACM 活动中，他们表现出关注、活跃、热心、痴情等积极主动的情绪体验，我们认为至少有两个原因：

1）ACM 活动让大学生直面将来可能遇到的各种问题，对他们的未来生存能力是一种历练，需要他们动脑思考、动手操作，需要多感官参与、多种心理能力的投入，在活动中学，在做中学，主动地学，创造性地学。

2）用于培训和竞赛的试题，特别是世界总决赛的试题水平较高，这些试题知识基础面宽，涵盖了计算几何、数论、组合数学、搜索技术、动态规划、图论等方方面面。所有试题将知识“返璞归真”于问题背景，还原成现实生活中的事例，既有趣又新颖，要求解题者面对实际寻求一种合适的解决方法，而不在乎方法本身的难易程度如何。虽然解题需要依托一定的基础知识，但不能死套条条框框和现成模式。它要求解题者有思想和睿智、有方法和策略、有编程的经历和经验，能洞察其间隐藏着的规律，灵活机动地解题。各国、各地区的 ACM-ICPC 试题和测试

数据竞相登录 Internet，在全球各个角落，人们可以通过终端随时查询异国异地的竞赛资料，只要按动鼠标就可以置身于任何一场竞赛之中。众多试题在网络上共享，不仅为参与 ACM 活动的高校提供了取之不竭、用之不尽的培训信息，而且也可作为实例或考题供许多大学在进行程序设计课程教学时使用，极大地丰富了世界计算机教育的课程资源。

本书收录了连续八届全球总决赛（2004～2011 年）的 83 道试题及其解析。建议读者按下述步骤阅读：

首先是阅读试题，但不要急于看下面的算法分析，先把自己置于“没路”和“只有荆棘”的情境之中，自己思考一下解题方法，培养“路是从没路的地方踩踏出来的，从只有荆棘的地方开辟出来的”的探索与开拓精神。有了想法后，再与书中的方法对照一下，看看是否正确，效率怎样。这样做既可以客观地衡量自己的能力，也能真正地从书中得到更多启示，汲取更多养分。

看完试题和解析之后可以进入第二步：如果你觉得其中一些试题有用且对它们感兴趣，愿意付出时间，就不妨试着做一下，但最好不要依葫芦画瓢。在正确的思想指导下可以多阅读一些程序范例，多上机解题，特别是多做大题和难题，使自己解决实际问题的能力有所提高。

最后一步是温习。回过头来复习一下试题所涉及的知识，温故知新，可以使自己对这些知识的内涵、适用范围和应用方法多一点切身感受，少一点陌生和困惑。

我们之所以建议读者这样做，是因为问题的表示往往比答案更重要。书中的解析和程序只不过是数学或实验的一种结果形式。解题的途径绝非书中所述的一种，书上大多数试题没有固定的模式可套，全靠解题者应“题”而行，相机而动。在看完书中的解题方法之后，不妨回到起点对问题重新定义，从某个新的角度重新思考，激发解题的动力和突破“盒子”思考问题的创造思想，组合学过的算法和数据结构知识，通过语言和编程技术使想法变成计算机的系统实现，在试题和编程解题之间架设一座真正属于自己的桥梁。

著名科学家爱因斯坦曾说过，“想象力比知识更重要，因为知识是有限的，而想象力囊括着世界上的一切，推动着进步，并且是知识的源泉”。虽然这本书可以帮助读者拓展知识面，但这并不是终极目的。“纸上得来终觉浅，绝知此事要躬行”，本书的目的是要培养读者在不同问题情景下的编程实践能力，提高读者自主思考、提出问题、拓展思路、探索新知的创造力。因为这些知识和能力才是我们在未来知识经济时代中最能体现“自我”的价值所在，一旦与社会需求融合，将形成一种任何生产资料都无法取代，并且能产生巨大效益的智力资源。

需要说明的是，本书是在复旦大学 ACM 集训队长期活动的基础上积累而成的，每道程序都通过了严格的测试验证，其中一些程序经多次修改，精益求精。本书主要由吴永辉、王建德编著，程序编写工作分配为：杨溢、李明韫编写了 2004～2008 年 ACM-ICPC 世界总决赛试题程序，张一博、王禹、何羿宏、于竟成等编写了 2009～2011 年 ACM-ICPC 世界总决赛的试题程序。还有不少人为本书的出版付诸了辛勤的劳动，做出了不可或缺的贡献。在此，向他们表示由衷的感谢。

由于时间和水平所限，书中难免存在缺点和错误，表述不当和笔误也在所难免，恳切希望学术界同仁或读者指正。如果你在阅读中发现了问题，请通过书信或电子邮件告诉我们，以便我们及时整理成勘误表放在本书的专门网站上，供广大读者查询更正。我们更期望读者对本书提出建设性意见，以便再版时改进。我们的联系方式是：

通信地址：上海市邯郸路 220 号复旦大学计算机科学技术学院 吴永辉

邮编：200433

E-mail：yhwu@fudan.edu.cn

王建德　吴永辉

2012 年 4 月于上海

目 录

前言

第 1 章 2004 ACM-ICPC 世界总决赛试题解析 …… 1

试题 1-1 蚂蚁 Carl（Carl the Ant） …… 1
试题 1-2 直升机机场（Heliport） …… 5
试题 1-3 六面视图（Image Is Everything） …… 10
试题 1-4 危险的布拉格城（Insecure in Prague） …… 14
试题 1-5 相交的时间段（Intersecting Dates） …… 18
试题 1-6 拼接地图（Merging Maps） …… 21
试题 1-7 导航（Navigation） …… 26
试题 1-8 道路绿化（Tree-Lined Streets） …… 30
试题 1-9 悬吊！（Suspense!） …… 33
试题 1-10 地面飞行控制中心（Air Traffic Control） …… 37

第 2 章 2005 ACM-ICPC 世界总决赛试题解析 …… 43

试题 2-1 眼球弯曲（Eyeball Benders） …… 43
试题 2-2 GSM 网络的简化模型（Simplified GSM Network） …… 49
试题 2-3 裁判员的旅行问题（The Traveling Judges Problem） …… 53
试题 2-4 纸牌戏法（cNteSahruPfefrlefe） …… 57
试题 2-5 阳光普照（Lots of Sunlight） …… 60
试题 2-6 交叉的街道（Crossing Streets） …… 64
试题 2-7 铺满平面（Tiling the Plane） …… 68
试题 2-8 长城游戏（The Great Wall Game） …… 71
试题 2-9 讨论会（Workshops） …… 74
试题 2-10 通信服务区（Zones） …… 77

第 3 章 2006 ACM-ICPC 世界总决赛试题解析 …… 81

试题 3-1 最小费用的飞机旅行（Low Cost Air Travel） …… 81
试题 3-2 订购冰激凌薄饼片！（Remember the A La Mode!） …… 84
试题 3-3 稳态的雕塑（Ars Longa） …… 88
试题 3-4 二段数（Bipartite Numbers） …… 92
试题 3-5 压缩二进制消息（Bit Compressor） …… 94
试题 3-6 构造一个时钟（Building a Clock） …… 96
试题 3-7 朝圣（Pilgrimage） …… 102
试题 3-8 口袋数（Pockets） …… 106

试题 3-9 隔离度
（Degrees of Separation）……113
试题 3-10 通信路线（Routing）……115
第 4 章 2007 ACM-ICPC
世界总决赛试题解析……120
试题 4-1 基因计算（Consanguine
Calculations）……120
试题 4-2 集装箱（Containers）……124
试题 4-3 宏大的平面图
（Grand Pix）……126
试题 4-4 提花电路
（Jacquard Circuits）……130
试题 4-5 领取行李
（Collecting Luggage）……135
试题 4-6 小球游戏
（Marble Game）……140
试题 4-7 网络（Network）……147
试题 4-8 可视的屋顶部分
（Raising the Roof）……150
试题 4-9 水箱（Water Tanks）……156
试题 4-10 隧道（Tunnels）……161
第 5 章 2008 ACM-ICPC
世界总决赛试题解析……165
试题 5-1 空调机械公司（Air
Conditioning Machinery）…165
试题 5-2 都是整数解（Always an
Integer）……169
试题 5-3 传送带（Conveyor Belt）…172
试题 5-4 猎犬追兔游戏（The Hare
and the Hounds）……178
试题 5-5 哈夫曼编码
（Huffman Codes）……183
试题 5-6 Glenbow 博物馆
（Glenbow Museum）……188
试题 5-7 神经网络（Net Loss）……190
试题 5-8 画家（Painter）……195
试题 5-9 可疑的密码（Password
Suspects）……201
试题 5-10 天空是极限
（The Sky is the Limit）……206
试题 5-11 蒸汽压路机
（Steam Roller）……210
第 6 章 2009 ACM-ICPC
世界总决赛试题解析……216
试题 6-1 一个周全的调度
（A Careful Approach）……216
试题 6-2 判别电路故障
（My Bad）……218
试题 6-3 蚂蚁 Carl 又回来了
（The Return of Carl）……225
试题 6-4 管道内径
（Conduit Packing）……229
试题 6-5 运费稳定
（Fare and Balanced）……233
试题 6-6 防鹿围栏
（Deer-Proof Fence）……238
试题 6-7 纸牌的房屋
（House of Cards）……241
试题 6-8 多数部长的投票（The
Ministers' Major Mess）……248
试题 6-9 弹簧撑杆
（Struts and Springs）……252
试题 6-10 地铁的时间估算
（Subway Timing）……256
试题 6-11 后缀替换语法
（Suffix-Replacement
Grammars）……260
第 7 章 2010 ACM-ICPC
世界总决赛试题解析……263
试题 7-1 求值 apl 表达式！
（APL Lives!）……263
试题 7-2 条形码（Barcodes）……275
试题 7-3 生物机器人的轨迹
（Tracking Bio-bots）……281
试题 7-4 城堡（Castles）……284
试题 7-5 渠道（Channel）……288

试题 7-6 等高线地图
（Contour Mapping）……297
试题 7-7 岛屿（The Islands）……302
试题 7-8 下雨（Rain）……306
试题 7-9 冰上机器人
（Robots on Ice）……311
试题 7-10 分享巧克力
（Sharing Chocolate）……315
试题 7-11 镇纸（Paperweight）……318
第 8 章 2011 ACM-ICPC
世界总决赛试题解析……325
试题 8-1 加或乘
（To Add or to Multiply）……325
试题 8-2 仿射的混乱
（Affine Mess）……329
试题 8-3 古代的象形符号
（Ancient Messages）……334
试题 8-4 芯片的难题
（Chips Challenge）……338
试题 8-5 咖啡枢纽
（Coffee Central）……343
试题 8-6 机器公司
（Machine Works）……346
试题 8-7 魔杖（Magic Sticks）……351
试题 8-8 你心爱的采矿业（Mining
Your Own Business）……357
试题 8-9 疯狂木乃伊
（Mummy Madness）……360
试题 8-10 金字塔（Pyramids）……365
试题 8-11 垃圾迁移
（Trash Removal）……368

第 1 章

2004 ACM-ICPC 世界总决赛试题解析

试题 1-1　蚂蚁 Carl（Carl the Ant）

【问题描述】

蚂蚁在地面爬过的时候会留下微量的化学痕迹，以便于其他蚂蚁跟着自己的路径走。通常这些痕迹是直线形状的。但在一群蚂蚁中，有一只名叫 Carl 的不同寻常的蚂蚁。Carl 经常走“之”字形，有时候会无数次穿越自己所走过的路。而当其他蚂蚁到达一个交叉点时，它们总是沿着有着最强气味的路径，也就是沿着离开交叉点的、最新被走过的路径走。

蚂蚁长 1 厘米，行走或钻洞的速度是每秒 1 厘米，并且严格地沿着它们的路径走（在走到角落的时候转 90 度）。蚂蚁彼此间不能交叉地同时走过一个点，也不能一只蚂蚁叠在另一只蚂蚁的身体上。如果两只蚂蚁在同一时间到达同一交叉点，则在 Carl 的路径上走了较长距离的蚂蚁将优先走；否则，在交叉点等待时间较长的蚂蚁将优先走。

Carl 先挖洞爬到地面上，在原点于 0 时刻出发。它将沿着它的路径走到终点，再钻洞回到地下。其他的蚂蚁以特定的时间间隔跟着 Carl 出发。给出 Carl 所走的路径的描述以及其他蚂蚁的出发时间，请计算所有蚂蚁走完给定的路径，并钻洞返回地下所需要的时间，保证所有的蚂蚁都可以走完。注意：这里是由出题人确保，而不是要求解题人保证。

输入：

输入包含多组测试用例。输入的第一行是一个整数，表示测试用例的数目。

每组测试用例的第一行是 3 个正整数 n（$1 \leqslant n \leqslant 50$）、$m$（$1 \leqslant m \leqslant 100$）和 d（$1 \leqslant d \leqslant 100$），其中 n 表示 Carl 所走的路径中的线段个数，m 表示要走这条路径的蚂蚁的总数（包括 Carl），d 表示每两只连续出发的蚂蚁出发的时间间隔。Carl（编号为 0）在 0 时刻出发。下一只蚂蚁（编号为 1）在 d 时刻出发，第三只蚂蚁在 $2d$ 时刻出发，以此类推。如果蚂蚁爬出来的洞穴被堵住了，蚂蚁会按照正确的顺序尽可能快地从洞中爬出。

在每组测试用例的接下来的 n 行每行包括一对不重复的整数对 x，y（$-100 \leqslant x$，$y \leqslant 100$），这是按照 Carl 所走的路径顺序而给出的在路径上的线段的端点。第一条线段从原点（0,0）出发，后一条线段的起点就是前一条线段的终点。

为了方便起见，Carl 总是在平行于坐标轴的线段上行走，线段的端点不会与其他线段相交。

输出：

每组测试用例的输出如下：

```
Case C:
Carl finished the path at time t1
The ants finished in the following order:
a1 a2 a3 ... am
```

```
The last ant finished the path at time t2
```

这里 C 表示测试数据组编号（从 1 开始），$a_1, a_2, a_3, \cdots, a_m$ 表示按顺序到达目的地的蚂蚁，t_1 和 t_2 表示 Carl 和最后一只蚂蚁到达目的地的时刻（以秒表示）。每组测试用例之间输出一个空行。

样例输入	样例输出
2	Case 1:
4 7 4	Carl finished the path at time 13
0 4	The ants finished in the following order:
2 4	0 2 1 3 4 5 6
2 2	The last ant finished the path at time 29
-2 2	
4 7 2	Case 2:
0 4	Carl finished the path at time 13
2 4	The ants finished in the following order:
2 2	0 4 1 5 2 6 3
-2 2	The last ant finished the path at time 19

试题解析

看完试题后很自然地得出判断：这是一道模拟题，直接按照题意模拟每一时刻。

首先判断是否有新蚂蚁出现，然后判断每一只蚂蚁是否可以向前走。这里不能走的可能情况有两个：

1）在交叉口的蚂蚁优先级不够高。

2）前面的蚂蚁动不了。

把所有能向前走的蚂蚁都向前移一步，在此基础上判断是否所有蚂蚁都到达终点。但这仅是一个思维的粗浅轮廓。具体实现时，需要详细记录下面几点：

- 每一点最新的方向。
- 头在点（x,y）并朝向方向 k 的蚂蚁编号。
- 尾在点（x,y）并朝向方向 k 的蚂蚁编号。

在此基础上展开模拟运算：

若有新蚂蚁出现，就加入一只头和尾都在点（0,0）的蚂蚁 1。由于 Carl 的优先级最高且线段之间不能重合（每条线段的端点不与其他线段相交），Carl 一定不会被堵住，因此先假设蚂蚁 Carl 已经走过了（即更新某一点的最新方向），然后把它作为普通的蚂蚁处理。

对于每一只蚂蚁，如果它的头与多个蚂蚁的头在同一个交叉点，那么判断哪只蚂蚁的优先级较高，优先级较低的蚂蚁必须停留。对于剩下的可能前进的蚂蚁，还要判断在它前进的方向是否有蚂蚁。如果前面没有蚂蚁或前面的蚂蚁可以移动，那么可以向前走；否则必须停留。

当所有蚂蚁能否移动都确定了，就把其中可移动的蚂蚁向前移一步，并记录是否有蚂蚁到达终点。

若所有蚂蚁都到达终点了，则模拟结束并输出答案；否则继续下一时刻。

程序清单

```
#include <stdio.h>
#include <string>
struct info                    //蚂蚁的结构类型
```

```
{
    int x,y,len,wait,dir;        //坐标点（x,y），走过的距离为 len，等待的时间 wait，朝向 dir
};
const int MaxN=50;               //线段数上限
const int MaxM=100;              //蚂蚁数上限
const int MaxL=250;              //每条线段的最大长度
const int Add[4][2]={-1,0,  0,1,  1,0,  0,-1};    //上、右、下、左四个方向的单位向量
int N,cases,n,m,D,Rs,fin,sta,T,ex,ey;   //Rs 记录路线的长度,fin 记录到达终点的蚂蚁个数,sta
                                        //记录已经出发的蚂蚁个数，T 记录当前时刻
int route[MaxN * MaxL];                 //第 i 条单位线段的方向为 route[i]
int map[MaxL+1][MaxL+1];                //路线上（x,y）的最新方向
int ants[MaxL+1][MaxL+1][4];            //记录地图上头在点（x,y）并朝向方向 k 的蚂蚁编号为
                                        //ants[x][y][k]
int list[MaxM];                         //第 i 个到达目的地的蚂蚁编号为 list[i]
info ant[MaxM];                         //蚂蚁序列
void init()                             //输入当前测试组信息并计算 Carl 所走的路径
{
    int i,j,k,t,x1,y1,x2,y2;
    scanf("%d %d %d",&n,&m,&D);         //读入 Carl 所走的路径中的线段个数，要走这条路径的
                                        //蚂蚁的总数，蚂蚁相继出发的时间间隔
    x1=0; y1=0; Rs=0;
    for(i=0;i<n;i++)                    //计算 Carl 所走的路径
    {
        scanf("%d %d",&x2,&y2);         //读入当前线段的终点
        if(x1>x2) t=0;                  //判断线段的走向
        if(y1<y2) t=1;
        if(x1<x2) t=2;
        if(y1>y2) t=3;
        for(;x1!=x2 || y1!=y2;)         //计算当前线段的终点(x1,y1)，即下一条线段的起点
        {
            route[Rs]=t; Rs++;
            x1+=Add[t][0]; y1+=Add[t][1];
        }
    }
    fin=0; sta=0;                       //到达终点的蚂蚁数和已经出发的蚂蚁数初始化为零
    for(i=0;i<=MaxL;i++)                //清空地图上所有蚂蚁
        for(j=0;j<=MaxL;j++)
            for(k=0;k<4;k++) ants[i][j][k]= -1;
}
void work()                             //对当前时刻进行模拟计算
{
        bool ok;
    int i,j,x,y,d,p,tmp;
    int go[MaxM];                       //蚂蚁停留的标志
    memset(go,0,sizeof(go));            //1 表示不移动，0 表示可能可以移动
    ok=0;                               //累计到达终点的蚂蚁数
    for(i=0;i<sta;i++) if(ant[i].x<0) ok++;
    for(i=0;i<sta;i++) if(ant[i].x>=0)  //根据优先级计算每个蚂蚁能否移动的标志
    {
        x=ant[i].x; y=ant[i].y;         //（x,y）表示当前坐标
        for (j=0;j<4;j++)
        if(ants[x][y][j]>=0)
          {
              tmp=ants[x][y][j];        //计算头在（x,y）且方向为 j 的蚂蚁序号
```

```
                if (ant[tmp].wait>ant[i].wait || (ant[tmp].wait==ant[i].wait && ant[tmp].
len>ant[i].len))          //若蚂蚁 tmp 的优先级较高，则蚂蚁 i 就不能动了，累计不能移动的蚂蚁数
                {
                    p=1; go[i]=1; ok++;
                    break;
                    }
                }
        }
    do{                                                    //若前方有蚂蚁，则设停留标志
          ok=0;
          for(i=0;i<sta;i++)
          if (!go[i]&&ant[i].x>=0)
          {
                d=map[ant[i].x][ant[i].y];                 //计算蚂蚁 i 的前进方向
                x=ant[i].x + Add[d][0]; y=ant[i].y+Add[d][1];//计算移动后的新坐标
                if(ants[x][y][d]>=0&&go[ants[x][y][d]]==1)  //若新坐标上有停留的蚂蚁，则蚂
                                                           //蚁 i 不能动
                {
                     go[i]=1; ok=1;
                    continue;
                }
          }
       } while (ok);       //如果有新的蚂蚁被堵住，那么需要继续判断会不会导致更多的蚂蚁被堵住
       for(i=0;i<sta;i++)
          if(!go[i]&&ant[i].x>=0)
             {            //若蚂蚁 i 可移动，则移动距离+1，撤去等待时间和移前位置的蚂蚁编号
             ant[i].len++; ant[i].wait=0;
             ants[ant[i].x][ant[i].y][ant[i].dir]=-1;
          }else if(ant[i].x>=0) ant[i].wait++;             //否则等待时间+1
          for(i=0;i<sta;i++)
          if (!go[i]&&ant[i].x>=0)                         //加新标记
             {
                x=ant[i].x; y=ant[i].y;
          if(x==100&&y==100&&i!=sta-1) { ants[100][100][map[100][100]]=i+1; }
                                                           //若洞口被移开了，新出现一只蚂蚁
          d=map[x][y];                                     //蚂蚁 i 的前进方向
          ant[i].x+=Add[d][0]; ant[i].y+=Add[d][1]; ant[i].dir=d; //求出新坐标
          if(ant[i].x==ex&&ant[i].y==ey)                   //蚂蚁 i 到达目的地
          {
             list[fin]=i;
             fin++;
             ant[i].x=-1;
          }else ants[ant[i].x][ant[i].y][d]=i;             //记录蚂蚁 i 进入新位置
       }
}
void write()                                               //输出当前测试数据组的解
{
    int i;
    printf("Case %d:\n",cases);                            //输出测试数据组编号
    printf("Carl finished the path at time %d\n",ant[0].len+1);
                    //由于蚂蚁 Carl 的优先级最高，因此其移动时间即为移动长度
    printf("The ants finished in the following order:\n");
    printf("%d",list[0]);                                  //输出按顺序到达目的地的蚂蚁
    for(i=1;i<m;i++) printf(" %d",list[i]);
    printf("\n");
```

```
    printf("The last ant finished the path at time %d\n",T+1);
                                                    //输出最后一只蚂蚁到达目的地的时间
    if(cases<N) printf("\n");                       //若非最后一组测试组，则换行
}
int main()
{
    int X,Y;                                        //记录蚂蚁 Carl 在某时刻的坐标
    scanf("%d",&N);                                 //输入测试用例数
    for(cases=1;cases<=N;cases++)                   //依次处理每组测试用例
    {
       init();
       X=100; Y=100; ex=-1; ey=-1;                  //输入当前测试组信息并初始化(由于负数处理起来
                                                    //比较麻烦，因此将所有坐标统一加上 100)
       for(T=0;fin<m;T++)                           //顺序模拟每一时刻，直至所有蚂蚁到达终点
       {
          if(T<Rs)
          {
             map[X][Y]=route[T];                    //根据 Carl 的路线更新地图上的标记
             X+=Add[route[T]][0]; Y+=Add[route[T]][1];
             if(T==Rs-1){ ex=X; ey=Y;}              //记录 Carl 所走路线的终点
          }
          if(T%D==0 && sta<m)                       //若有新出发的蚂蚁
          {
             if(ants[100][100][map[100][100]]<0) ants[100][100][map[100][100]]=sta;
             ant[sta].x=100; ant[sta].y=100;
             ant[sta].len=0; ant[sta].wait=0; ant[sta].dir=map[100][100];
             sta++;
          }
          work();                                   //对当前时刻进行模拟计算
          }
          write();                                  //输出当前测试数据组的解
       }
}
```

试题 1-2　直升机机场（Heliport）

【问题描述】

当前，工作节奏很快，某公司投资建造了直升机机场，来减少其繁忙的管理人员的旅行时间。直升机机场通常是圆形的，建造在公司总部的楼顶上（见图 1.2-1）。

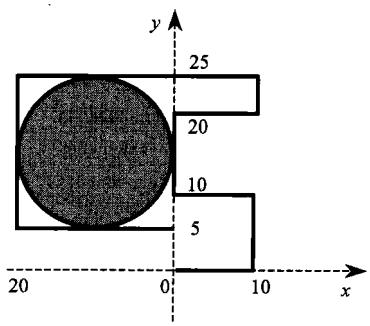

图　1.2-1

请编写一个程序，找到能建造在一个建筑物的平坦屋顶上的圆形直升机机场的最大半径。

屋顶的形状是一个简单多边形。由于这仅仅是施工的设计阶段的工作，因此该程序只需给出直升机机场的半径。图 1.2-1 中显示的直升机机场的最大半径是 10。

输入：

输入文件包含若干组测试用例，每组测试用例只有两行：第一行包含一个偶数 n（$4 \leqslant n \leqslant 20$），表述建筑物的边数；第二行包含 n 组形式为（m,d）的数据，其中 m 是一个整数（$1 \leqslant m \leqslant 50$），$d$ 是 U、R、D 或 L 中的一个字母。假定屋顶画在平面直角坐标系中，m 是屋顶的一条边的长度，d 是当你逆时针沿着屋顶边缘走时，经过这条边的方向。U、R、D 和 L 分别表示“上”、“右”、“下”和“左”。屋顶的边缘都是与 x 轴或 y 轴平行的，按逆时针顺序给出。起始位置是原点（0,0）。

最后一个测试用例后，给出一行，是一个数字 0。

输出：

对于每组测试用例，输出一行，包含测试用例编号（从 1 开始）和一个实数（四舍五入到小数点后两位），表示直升机机场的最大半径。在两个测试用例之间输出一个空行，如样例输出所示。

样例输入	样例输出
4 2 R 2 U 2 L 2 D 10 10 R 10 U 10 L 10 U 10 R 5 U 30 L 20 D 20 R 5 D 0	Case Number 1 radius is: 1.00 Case Number 2 radius is: 10.00

试题解析

1. 计算多边形的顶点坐标

建筑物是一个多边形，试题仅给出每条边的长度 m 和方向 d。我们从（$x[0]$,$y[0]$）=（0,0）出发，依据这些信息计算递推多边形 n 个顶点的坐标（$x[i]$,$y[i]$）（$0 \leqslant i \leqslant n-1$）：

当 d='U'时，（$x[i+1], y[i+1]$）=（$x[i], y[i]+m$）；

当 d='R'时，（$x[i+1], y[i+1]$）=（$x[i]+m, y[i]$）；

当 d='D'时，（$x[i+1], y[i+1]$）=（$x[i], y[i]-m$）；

当 d='L'时，（$x[i+1], y[i+1]$）=（$x[i]-m, y[i]$）。

2. 计算直升机机场的最大半径

采用二分的方法求直升机机场的最大半径。假设半径的可能区间为[ra,rb]，设 r=（ra+rb）/2，判断半径为 r 的直升机机场能否建造：如果能，则在右区间[r,rb]搜索最大半径；否则在左区间[ra,r]搜索最大半径。区间的初值设为[0,999]，当 $rb-ra<10^{-6}$ 即可停止二分过程，下面将展开对这个问题的分析。

3. 判断半径为 r 的直升机机场能否建造

假设半径为 r 的直升机机场可以建造在多边形内，我们可以移动这个直升机机场，使它满足以下 3 个条件中的一个，然后求出候选的圆心坐标，再逐一判断所求出的坐标是否合法。

1）直升机机场贴着两条互相垂直的边。

2）直升机机场贴着一条边和一个顶点。

3）直升机机场贴着两个顶点。

下面依次讨论这 3 种情况：

1）直升机机场贴着两条互相垂直的边（如图 1.2-2 所示）。

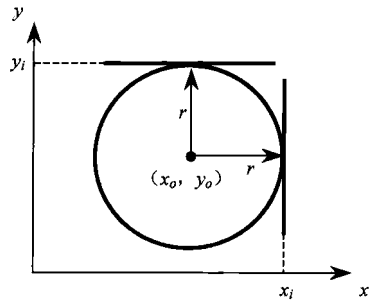

图　1.2-2

假设与 y 轴平行的边的 x 坐标为 x_i，与 x 轴平行的边的 y 坐标为 y_i，则候选的圆心坐标有 4 个：(x_i+r, y_i+r)、(x_i+r, y_i-r)、(x_i-r, y_i+r) 和 (x_i-r, y_i-r)。

2）直升机机场贴着一条边和一个顶点（如图 1.2-3 所示）。

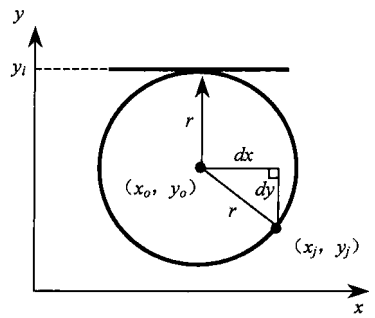

图　1.2-3

以与 x 轴平行的边为例，与 y 轴平行的边可以类似地处理。假设与 x 轴平行的边的 y 坐标为 y_i，顶点坐标为 (x_j, y_j)，则候选圆心的 y 坐标为 $y_o=y_i-r$ 或 $y_o=y_i+r$。如果 $y_j<y_i$，则取 $y_o=y_i-r$，否则取 $y_o=y_i+r$，求得 $dy=|y_o-y_j|$。如果 $dy>r$，则没有候选的方案；否则可以求得 $dx=\sqrt{r^2-dy^2}$，候选的圆心坐标为 (x_j+dx, y_o) 和 (x_j-dx, y_o)。

3）直升机机场贴着两个顶点（如图 1.2-4 所示）。

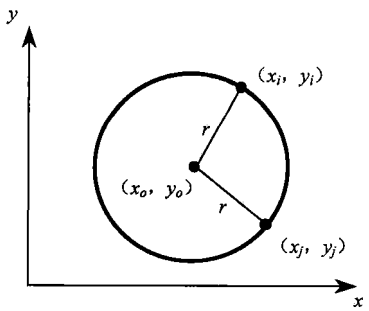

图　1.2-4

假设顶点坐标为 (x_i, y_i) 和 (x_j, y_j)，通过解方程组

$$(x_i-x_o)\times(x_i-x_o)+(y_i-y_o)\times(y_i-y_o)=r^2$$
$$(x_j-x_o)\times(x_j-x_o)+(y_j-y_o)\times(y_j-y_o)=r^2$$

即可求得圆心坐标（x_o,y_o）。

剩下的问题就是当圆心坐标和半径都确定之后，判断此方案是否合法。显然，只要满足以下两个条件，该方案就是合法的：

1）圆心在多边形内。

2）多边形的边不与圆相交。

于是，我们将候选的圆心坐标逐一判断，就可以知道半径为 r 的直升机机场能否建造，最终二分得到最大的可以建造的直升机机场半径。

其他细节见程序清单。

程序清单

```
#include <cstdio>
#include <math.h>
const double eps=1e-6;                  //精度

int x[25],y[25],n,i,len,px,py,cases;//第 i 条边的终点坐标为(x[i],y[i]); 当前边的长度为 len
                                        //终点坐标为 (px,py); 测试用例编号为 cases
double ra,rb,r;                         //直升机机场半径的可能区间范围为[ra,rb], 实际半径值为 r
char dr;                                //边的方向字符
bool check(double ox,double oy)         //判断圆心为(ox,oy)、半径为 r 的方案是否可行
{
   int i,s;
   s=0;                                 //射线与多边形边相交的次数初始化
   for(i=0;i<n;i++)                     //从(ox,oy)往 x 轴正方向射出一条射线
      if(x[i]>ox&&((y[i]>oy)^(y[i+1]>oy))) s++;//累计射线与多边形边相交的次数
      if(s%2==0)return false;           //如果是偶数，则说明圆心在多边形外，方案不可行
      for(i=0;i<n;i++)                  //逐一判断每条边和每个顶点
      {
         if((x[i]-ox)*(x[i]-ox)+(y[i]-oy)*(y[i]-oy)<(r-eps)*(r-eps))
                                        //若顶点在圆内，方案不可行
            return false;
         if(x[i]==x[i+1]&&((y[i]>oy)^(y[i+1]>oy))&&fabs(x[i]-ox)<r-eps) return false;
                                        //若与 y 轴平行的边在圆内，方案不可行
       if(y[i]==y[i+1]&&((x[i]>ox)^(x[i+1]>ox))&&fabs(y[i]-oy)<r-eps)return false;
                                        //若与 x 轴平行的边在圆内，方案不可行
   }
   return true;                         //否则方案可行
}
bool ok()                               //判断半径为 r 的机场能否建造
{
   int i,j;
   double di,dd,mx,my,dx,dy;
   for(i=0;i<n;i++)                     //机场贴着两条互相垂直的边，求圆心坐标
      if(x[i]==x[i+1])
         for(j=0;j<n;j++)
            if(y[j]==y[j+1])
            {
               if(check(x[i]+r,y[j]+r)) return true;
               if(check(x[i]+r,y[j]-r)) return true;
```

```
        if(check(x[i]-r,y[j]+r)) return true;
        if(check(x[i]-r,y[j]-r)) return true;
    }
    for(i=0;i<n;i++)                    //机场贴着一条边和一个顶点，求圆心坐标
    for(j=0;j<n;j++)
        if(x[i]==x[i+1])
            {
            di=fabs(x[j]-(x[i]+r));
            if(di<r)
            {
                dd=sqrt(r*r-di*di);
                if(check(x[i]+r,y[j]+dd)) return true;
                if(check(x[i]+r,y[j]-dd)) return true;
            }
            di=fabs(x[j]-(x[i]-r));
            if(di<r)
            {
                dd=sqrt(r*r-di*di);
                if(check(x[i]-r,y[j]+dd)) return true;
                if(check(x[i]-r,y[j]-dd)) return true;
            }
        }
        else
        {
            di=fabs(y[j]-(y[i]+r));
            if(di<r)
            {
                dd=sqrt(r*r-di*di);
                if(check(x[j]+dd,y[i]+r)) return true;
                if(check(x[j]-dd,y[i]+r)) return true;
            }
            di=fabs(y[j]-(y[i]-r));
            if(di<r)
            {
                dd=sqrt(r*r-di*di);
                if(check(x[j]+dd,y[i]-r)) return true;
                if(check(x[j]-dd,y[i]-r)) return true;
            }
    }
for(i=0;i<n-1;i++)                      //机场贴着两个顶点，求圆心坐标
    for(j=i+1;j<n;j++)
    {
        mx=(x[i]+x[j])/2.0;
        my=(y[i]+y[j])/2.0;
        di=sqrt((x[i]-mx)*(x[i]-mx)+(y[i]-my)*(y[i]-my));
        if(di>0&&di<r)
        {
            dd=sqrt(r*r-di*di);
            dx=(my-y[i])/di*dd;
            dy=(x[i]-mx)/di*dd;
            if(check(mx+dx,my+dy)) return true;
            if(check(mx-dx,my-dy)) return true;
        }
    }
    return false;
```