国际大学生程序设计竞赛例题解（三）

图论、动态规划算法、综合题专集

郭嵩山　关沛勇　蔡文志　梁　锋　编著

含光盘1张

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

.1-44
:3

国际大学生程序设计竞赛例题解

（三）

图论、动态规划算法、综合题专集

郭嵩山　关沛勇　蔡文志　梁　锋　编著

電子工業出版社

Publishing House of Electronics Industry

北京 • BEIJING

内 容 简 介

本书以图论、动态规划算法、综合题的形式介绍了ACM国际大学生程序设计竞赛(ACM/ICPC)中所用到的典型算法，并结合例题，对如何灵活地运用这些算法进行比较详细的分析和深入浅出的讲解。本书以精讲多练为教学宗旨，并在每一个专题论述后用一章的篇幅选出一批有代表性的竞赛例题，对每道例题都有详细的解题分析、基本的测试数据及答案，以便同学们能在了解基本算法后作为学习、训练之用。随书附带的光盘里存放了所有例题中完整的测试数据，以便于有更高、更严格要求的同学能利用规模更大的测试数据进行训练和学习。

本书可以作为高等院校有关专业的研究生和本科生参加国际大学生程序设计竞赛的辅导教材，也可作为高等院校有关专业相关课程的教材和教学参考书，还可作为中学青少年信息学奥林匹克竞赛省级及省级以上优秀选手备战信息学奥林匹克竞赛的培训教材及训练题集。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有，侵权必究。

图书在版编目(CIP)数据

国际大学生程序设计竞赛例题解.3，图论、动态规划算法、综合题专集/郭嵩山等编著. —北京：电子工业出版社，2007.7

ISBN 978-7-121-04643-8

I. 国… II.郭… III.①程序设计—竞赛—高等学校—解题②图论算法—高等学校—解题 IV.TP311.1-44

中国版本图书馆CIP数据核字（2007）第096274号

责任编辑：龚立堇
印　　刷：
装　　订：北京市李史山胶印厂
出版发行：电子工业出版社
　　北京市海淀区万寿路173信箱　邮编 100036
开　　本：787×1 092　1/16　印张：18.5　字数：472千字
印　　次：2007年7月第1次印刷
印　　数：5 000册　　定价：32.00元　(含光盘1张)

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。

服务热线：（010）88258888。

前　言

ACM 国际大学生程序设计竞赛（ACM International Collegiate Programming Contest，简称 ACM/ICPC）是由国际计算机界历史悠久、颇具权威性的组织 ACM 学会（Association for Computer Machinery）主办，世界上公认的规模最大、水平最高的国际大学生程序设计竞赛，其目的旨在使大学生运用计算机来充分展示自己分析问题和解决问题的能力。该项竞赛从 1970 年举办至今已历 31 届，因历届竞赛都荟萃了世界各大洲的精英，云集了计算机界的“希望之星”，而受到国际各知名大学的重视，并受到全世界各著名计算机公司的高度关注，成为世界各国大学生最具影响力的国际级计算机类的赛事。ACM 所颁发的获奖证书也为世界各著名计算机公司、各知名大学所认可。该项竞赛分区域预赛和世界决赛两个阶段进行，各预赛区第 1 名获得参加世界决赛的资格，世界决赛安排在每年的 3－4 月举行，而区域预赛安排在上一年的 9－12 月在各大洲举行。ACM/ICPC 的区域预赛是规模很大、范围很广的赛事，以 2006 年为例，全世界有超过 1700 所大学的 6099 支参赛队，在六大洲的 82 个国家（或地区）的 35 个赛站中争夺全球总决赛的 88 个名额，其激烈程度可想而知。

与其他编程竞赛相比，ACM/ICPC 题目难度更大，更强调算法的高效性，不仅要解决一个指定的命题，而且必须要以最佳的方式解决指定的命题；它涉及的知识面广，与大学计算机系本科及研究生的课程直接关联，如程序设计、离散数学、数据结构、人工智能、算法分析与设计等，对数学要求更高；由于采用英文命题，对英语要求较高，ACM/ICPC 采用 3 人合作，公用一台电脑，所以它更强调团队协作精神；由于许多题目并无现成的算法，需要具备创新的精神；ACM/ICPC 不仅强调学科的基础，更强调全面素质和能力的培养。由于 ACM/ICPC 是采用 5 小时全封闭式竞赛，参赛队员与外界完全隔离，完全独立完成，是参赛队员实际能力的真实表露，其成绩可信度甚高。ACM/ICPC 又是一种“开卷考试”，可以带任何书籍、资料，甚至源程序代码清单（但不能带电子媒体），不需要死背算法，而强调的是算法的灵活运用。与其他计算机竞赛（如软件设计、网站设计等）相比，ACM/ICPC 有严谨而客观的评判规则（严格的数据测试），排除了因评委的主观因素而造成评审不公平的现象。所以，对 ACM/ICPC 成绩的争议较少。

中山大学自 1997 年首次参加 ACM/ICPC 亚洲区预赛以来的 10 年中，每年都派出多支队共参加过 31 次亚洲区预赛，成绩有 28 次在前 6 名之列（有 3 次在前 10 名），其中有 18 次进入三甲，夺得 3 次冠军（1999 年台北，2002、2003 年高雄）、5 次亚军（2000 年香港、筑波，2003 年北京、广州，2006 年河内）、10 次季军（1998－2000 年上海，2001 年达卡，2002 年北京，2003 年高雄，2004 年马尼拉，2005 年台北、北京，2006 年首尔）。中山大学的参赛队曾 8 次进入全球总决赛（1999－2001 年、2003－2007 年）：2000 年在美国佛罗里达州奥兰多市举行的第 24 届全球总决赛中夺得了第 11 名的好成绩；2001 年在加拿大温哥华市举行的第 25 届全球总决赛中首获铜牌（世界第 14 名）；2003 年在美国洛杉矶市好莱坞举行的第 27 届

全球总决赛中获得世界第 8 名并首获银牌的好成绩，挤身世界八强之列； 2004 年在捷克布拉格市举行的第 28 届全球总决赛中获得世界第 11 名并再获铜牌，且在中国内地高校中排名第一；2005 年在上海举行的第 29 届全球总决赛中获得世界第 17 名；2006 年在美国得克萨斯州圣安东尼奥市举行的第 30 届全球总决赛中获得世界第 19 名；2007 年在日本东京市举行的第 31 届全球总决赛中获得世界第 26 名。

为了帮助备战国际大学生程序设计竞赛的高等院校学生们提高程序设计水平，培养他们更强的分析问题与解决问题的能力，我们编写了这套《国际大学生程序设计竞赛例题解》。本书是作为这套例题解的第三册，编程所用的语言是 Pascal 和 C++。全书共分 5 章，第 1 章介绍图论相关知识及基本算法，第 2 章为图论解题样例，第 3 章介绍动态规划的相关知识及基本算法，第 4 章为动态规划的解题样例，第 5 章介绍了综合题的解题样例。对于在大学学习 C++的同学，可以将书中用 Pascal 描述的算法用 C++来实现，以加深对所学到的算法知识的理解。

为了方便读者学习，本书对每个题目作了详尽的题目分析并详细地讲解其算法实现的原理，同时提供了完善的参考程序及其程序分析供读者参考，书中还提供了基本测试数据以方便读者测试自行完成上述题目的结果。随书还附带光盘，存放所有例题中完整的测试数据，以便于有更高、更严格要求的同学能利用规模更大的测试数据进行训练和学习之用。

本书目录中带*号的例题都是原创题，参加命题的有李志业、金涛、陈明睿、李伟星、郑凤梅、张磊、莫瑜、关沛勇、黎俊瑜、陈实、蔡文志、王颖、林祺颖、杭啸、张子臻等，他们大部分是硕士研究生，都是参加过世界决赛或亚洲多个赛站区域预赛并取得很好成绩的中山大学队的主力队员。本书所提供的测试数据全部由中山大学历届 ACM/ICPC 队队员设计。

本书由中山大学郭嵩山教授和他指导的 3 位硕士研究生编写，郭嵩山老师是国际大学生程序设计竞赛中山大学队的主教练，关沛勇、蔡文志、梁锋是参加过世界决赛或亚洲多个赛站区域预赛并取得很好成绩的中山大学队的主力队员。我们期望能将自己的知识、经验、心得和体会，奉献给广大的程序设计爱好者，以便与大家共同探讨和交流。

本书题目构思新颖，所涉及到的算法知识面广，基本上覆盖大学计算机类本科专业所学到的基本算法。本书可以作为高等院校大学生和研究生们准备参加各级国际大学生程序设计竞赛活动的辅导教材和训练题集，也可以作为高等院校研究生和本科高年级学生学习相关课程的参考书，还可以作为中学省级及省级以上信息学奥林匹克优秀选手准备高层次程序设计竞赛的参考用书。

由于我们水平所限，书中难免有不足之处，欢迎读者批评指正，谢谢！

作者

2007 年 3 月

此为试读，需要完整PDF请访问：www.ertongbook.com

目　录

此为试读，需要完整PDF请访问：www.ertongbook.com

第 1 章　图论相关知识和基本算法

本章讨论图论的基本内容。图论是一门有趣而又实用的科学，广泛应用在电路分析、规划调度、遗传学等方面。图论的计算有很多很好的结论和算法，不过规模比较大的图非常复杂，靠手工计算是不可能的事情。计算机具有强大的存储和计算能力，把它应用在解决图论问题上具有得天独厚的优势。在很多有名的程序设计竞赛中，图论的题目通常都会占有一席之地。要解决这类题目要求选手具备很扎实的编程能力和对图论知识的透彻理解，两者缺一不可。

1.1　图的基本概念

图是由一个顶点的集合 V 及顶点间的关系集合 E 组成的，即

$$G=(V,\ E)$$

其中，V 是顶点的一个有穷非空集合；E 是顶点之间二元关系的一个有穷集合，称为边集，它的元素称为一条边。

下面介绍图的基本概念。

有向图：顶点对有序的图，即边$\langle u,\ v\rangle$和边$\langle v,\ u\rangle$是不同的两条边。

无向图：顶点对无序的图，边$\langle u,\ v\rangle$和边$\langle v,\ u\rangle$都是指同一条边，都用（u，v）表示。

权：如果图的边被赋予一个与它相关的数，这个数称为该边的权。

顶点的度：一个顶点的度是与它之间相关联的边的条数。在有向图中，顶点的度还可以细分为入度和出度，入度等于以顶点为终点的有向边的数目，出度等于从该顶点出发的有向边的数目。

路径：如果从顶点 v_1 出发，沿着一些边依次经过一些顶点 v_2，v_3，…，v_n，则称顶点序列$(v_1,\ v_2,\ v_3,\ \cdots,\ v_n)$为从顶点 v_1 到顶点 v_n 的路径。

路径长度：如果边是不带权的，路径长度等于路径上边的条数；如果边是带权的，路径长度等于路径上边的权值之和。

回路：如果一条路径上第一个顶点和最后一个顶点是相同的，则称这样的路径为回路或环。

连通：在无向图中，如果从顶点 v_1 到顶点 v_2 有路径存在，则称 v_1 和 v_2 是连通的。

连通图：如果图中任意一对顶点都是连通的，则称该图为连通图。

子图：如果有两个图 G 和 G'，G'的顶点集是 G 的顶点集的子集，且 G'的边集是 G 的边集的子集，那么称 G'为 G 的子图。

连通分量：图的极大连通子图称为一个连通分量。

1.2 图的邻接矩阵表示和邻接表表示

图是由顶点和边组成的，而边是反映顶点和顶点之间关系的。在计算机中存储和表示图，最根本的就是记录下顶点及顶点之间的关系，经常使用的方法有邻接矩阵和邻接表。

邻接矩阵就是使用一个矩阵存储顶点和顶点之间是否直接有边相连这个二元关系。假设邻接矩阵为 A，$A[i][j]$ 可以表示顶点 i 到顶点 j 之间是否直接相连，在边具有权值的图中，还可以存储边的权值。邻接矩阵的空间复杂度为 $O(n^2)$，好处是可以随机的访问任意两个顶点间的关系。但是如果需要遍历整个图的所有边，需要耗时 $O(n^2)$ 而不是 $O(e)$，e 为图的边数。图 1.2.1 所示为一个无向图的例子，而图 1.2.2 是图 1.2.1 的邻接矩阵表示。

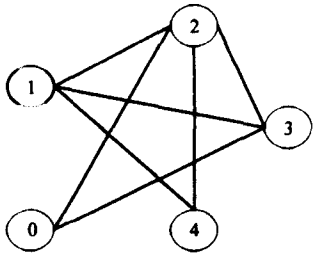

图 1.2.1　一个无向图

$$\begin{bmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

图 1.2.2　图 1.2.1 的邻接矩阵表示

当图比较稀疏的时候，也就是说图的边数比较少，使用邻接矩阵会浪费很多的存储空间，而且查找起来效率也不高。为了克服邻接矩阵的缺点，可以使用邻接表的方法。邻接表使用了链表对邻接矩阵进行改进，即每个顶点只保存所有直接与它相连的顶点，所有这些顶点通过链表来动态保存。邻接表的好处是使用最小限度的空间保存整个图，空间复杂度为 $O(e)$，没有保存任何多余的信息，当要遍历整个图的时候，只需要 $O(e)$ 的复杂度就行了。但是邻接表不能随机地访问任意两个顶点的关系，一定要通过顺序查找链表才能确定。其实邻接矩阵和邻接表的区别，类似于数组和链表这两种数据结构的区别。什么时候使用什么方法，要根据实际需要来决定。图 1.2.3 所示为一个图的邻接表表示。下面介绍用邻接表记录图的算法描述。

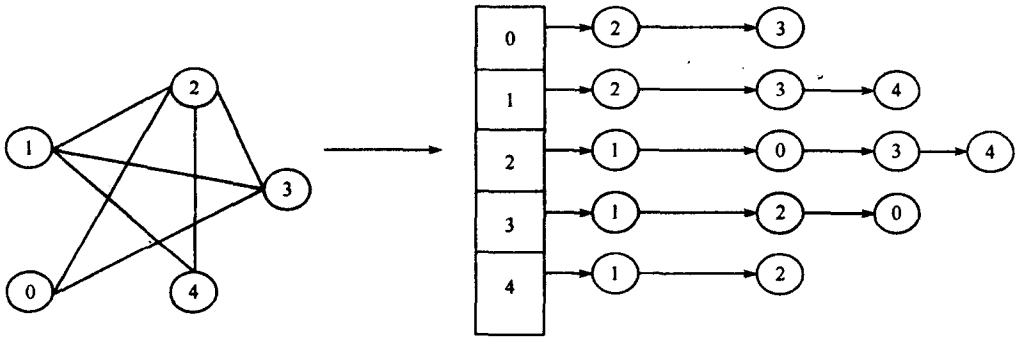

图 1.2.3　图的邻接表表示

基本算法描述

使用邻接表记录图

```
// n为图G顶点个数

// 定义邻接表的结构，一个结构记录一条边
struct Edge {
  int dest; // 记录目的地
  int value; // 边的权值
  Edge* link; // 记录链表的下一个元素
};

// 申请邻接表空间
Edge* edge = new Edge[n];
int i, u, v;

// 初始化邻接表
for(i = 0; i < n; i++) {
  edge[i] = null;
}

Edge* l;

// 接受输入，(u, v)代表一条边
while(cin>>u>>v) {
  // 新建一个邻接表结构
  l = new Edge;
  // 填写目的地为顶点v
  l->dest = v;
  // 把新创建的邻接表结构加入到顶点u的链表中去
  l->link = edge[u];
  edge[u] = l;
}

// 遍历所有边
for (i = 0; i < n; i++) {
  // 取顶点i的邻接表的链表入口元素
  l = edge[i];
  while(l) {
    // 输出从i出发可以直接到达的边
    cout<<i<<" "<<l->dest<<endl;
    // 取链表的下一个元素
    l = l->link;
  }
}
```

1.3 拓扑排序

给定一个有向图 G，对于任意一条有向边$<V_i, V_j>$，称 V_i 是 V_j 的直接前驱，V_j 是 V_i 的直接后继，这种前驱与后继的关系具有传递性。

如果这个图的任何一个结点都具有反自反性，也就是说任何一个结点都不是自己的前驱或后继，那么这个有向图是拓扑有序的。

对于一个拓扑有序的有向图，我们可以把它的所有结点排成一个线性有序的序列，设为 $a_1, a_2, a_3, \cdots, a_n$，那么对于任意两个结点 a_i 和 a_j，如果 a_i 是 a_j 的前驱，那么必然有 $i > j$，即 a_i 必然在序列里排在 a_j 的前面，这样的序列是不惟一的。从有向图得到符合这样性质的序列的过程称为拓扑排序。经过拓扑排序后得到的序列，原有图的结点间前驱和后继关系可以在序列中的先后顺序中体现出来。

拓扑排序的一个应用是可以用来检测有向图中是否存在有向环。如果一个有向图存在环，那么必然存在结点不具有反自反性，因而相应的拓扑有序序列是不存在的。

从图 1.3.1 可以得到一个拓扑有序序列 C_0，C_1，C_2，C_3，C_4；同时也可以得到另外一个拓扑有序的序列 C_0，C_1，C_3，C_2，C_4。可以看出，拓扑有序的序列是不惟一的。

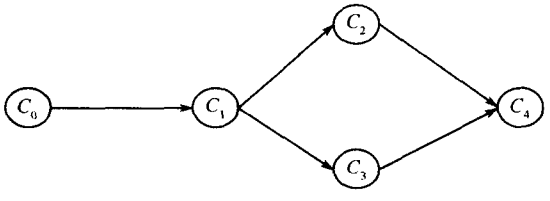

图 1.3.1　拓扑有序的有向图

下面给出一个拓扑排序的算法。其描述如下：假设有向图 G 有 n 个顶点，e 条边，在拓扑排序的过程中，搜索入度为 0 的顶点，把它所有的直接后继的入度数减 1，然后把该顶点输出。重复整个过程直到找不到入度为 0 的顶点就结束。如果所有 n 个结点都已经输出，那么证明 G 中不存在有向环，而且输出的序列就是拓扑有序的序列；否则 G 中存在有向环。在实现的过程中可以使用一个栈来保存当前搜索到的入度为 0 的点，每个顶点只会进一次栈和出一次栈，一共 n 次；顶点入度减 1 的运算共执行了 e 次，所以总时间复杂度为 $O(n+e)$。

基本算法描述

拓扑排序

```
// edge 是图 G 的邻接表，n 为图 G 顶点个数

void TopologicalSort() {
  // 统计每个顶点的入度数
  int* degree = new int[n];
  memset(degree, 0, n*sizeof(int));
  int i;
  Edge* l;
  for (i = 0; i < n; i++)
  {
    l = edge[i];
```

```
    while(l) {
      // i 的所有直接后继的入度数加 1
      degree[l->dest] ++;
      l = l->link;
    }
  }

  // top 是入度为 0 的顶点的栈的栈顶指针
  int top = -1;

  // 建立入度为 0 的顶点的栈。
  for (i = 0; i < n; i++)
  {
    if (degree[i] == 0) {
      degree[i] = top;
      top = i;
    }
  }

  // 一共有 n 个顶点进行拓扑排序，循环 n 次，每次输出一个顶点
  for (i = 0; i < n; i++) {
    if (top == -1) {
      // 栈为空，即不存在入度为 0 的顶点了，但是循环还没有结束，被输出的顶点还不够 n 个，
      // 证明当前剩下的顶点形成环
      cout << "A cycle exists in the graph!"<<endl;
      return;
    } else {
      // 取当前栈顶元素并出栈
      int j = top;
      top = degree[top];
      // 输出顶点 j
      cout<< j << endl;
      // 将 j 的所有直接后继的顶点的入度数减 1
      l = edge[j];
      while(l) {
        if (--degree[l->dest] == 0) {
          // 入度减至 0 的顶点，将其加入栈
          degree[l->dest] = top;
          top = l->dest;
        }
        // 取下一个直接后继
        l = l->link;
      }
    }
  }
  delete [] degree;
}
```


此为试读，需要完整PDF请访问：www.ertongbook.com