国际大学生程序设计竞赛辅导教程

郭嵩山 崔昊 吴汉荣 陈明睿 著

北京大学出版社

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最悠久、颇具权威性的组织 ACM 学会（Association for Computer Machinery）主办，是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已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 25 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，各预赛区第一名自动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 月~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，近几年，全世界有 1000 多所大学，近 2000 支参赛队在六大洲的 28~30 个赛站中争夺世界决赛的 60 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，而且必需要以最佳的方式解决指定的命题；它涉及知识面广，与大学计算机系本科以及研究生如程序设计、离散数学、数据结构、人工智能、算法分析与设计等相关课程直接关联，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、共用一台电脑，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，没有任何水份，是其实际能力的真实表露，其成绩可信度甚高；但 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（但不能带软盘），不需要去死背算法，而强调的是算法的灵活运用；与其它计算机竞赛（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，ACM/ICPC 对成绩的争议较少，大家比较心服口服。

综观近三年（1998~2000）中山大学共参加了 6 次地区预赛，成绩全部在三甲之列：连续三年夺得上海站季军（1999~2000 年还连续两年夺得上海站第四名）、夺得台北站冠军、

香港站亚军和日本站亚军；并连续三年争得了世界决赛权。并在第 24 届（2000 年）在美国佛罗里达州奥兰多市举行世界决赛中夺得了第 11 名的好成绩，在第 25 届（2001）在加拿大温哥华市举行的世界决赛中首获铜牌（世界第 14 名）。

为了帮助高等院校的大学生们备战国际大学生程序设计竞赛，帮助他们提高程序设计水平和培养更强的分析问题和解决问题的能力，我们编写了这本辅导教材。本书所用的语言是 Pascal（Delphi）。全书共分六章，第 1 章先介绍 Delphi 的运行环境，以便于读者能更好地读通后面各章的程序；第 2 章采用精讲的方式，简明扼要、深入浅出地介绍了在国际大学生程序设计竞赛中经常用到的各种典型算法；而在第 3 章中，我们着重介绍了寻找最优解的算法，诸如图论中的搜索算法和如何运用动态规划的思维来解决实际问题等方法；在第 4 章中，我们从 ACM/ICPC 世界决赛和区域预赛试题中精选了有代表性的 10 道例题，通过对例题的详细分析，力图让读者能更深刻地理解第 2、3 章中所介绍的基本算法；在第 5 章中，我们精选了一批有代表性的试题作为习题，并为这些习题设计了严格的、有梯度的测试数据，以便于读者检验自己编程的正确性；而在第 6 章中，我们给出了这些习题的详细分析和解答。为便于读者们学习和理解，本书的全部例题和习题都给出了我们自己编写的参考程序，而所有参考程序都有详细的注释。

参加编写本书的 4 位作者，第一位是国际大学生程序设计竞赛中山大学队的教练，其余 3 位都是参加过多次世界决赛和亚洲多个赛站区域预赛的中山大学队的主力队员。他们都是在校的研究生，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试题集，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，也可以作为省级及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。

中山大学计算机科学系 97 级孔颖同学（也是参加过两次世界决赛和亚洲多个赛站区域预赛的中山大学队的主力队员）曾参与过本书的一些程序的编写工作，在此表示衷心的感谢。

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

2001 年 10 月

目录

前言……1
第一章 Delphi 简介……1
第一节 Delphi 的运行环境……1
第二节 Delphi 常量……5
第三节 Delphi 变量……5
第四节 Delphi 类型……6
第五节 Delphi 的基本语句……11
第六节 Delphi 函数与过程……15
第七节 函数的递归……17
第八节 面向对象 Object Pascal……18
第九节 Delphi 中使用嵌入汇编……19
第二章 基本算法介绍……21
第一节 概述……21
第二节 常用数据结构在 Delphi 中的实现……21
第三节 枚举算法……32
第四节 回溯算法……33
第五节 贪心算法……35
第六节 分治算法……38
第七节 数值计算……39
第八节 计算几何……47
第九节 模拟题解法……51
第三章 寻找最优解的算法……53
第一节 动态规划……53
第二节 最短路问题……61
第三节 搜索算法……70
第四章 国际大学生程序设计竞赛（ACM/ICPC）试题及分析……89
第一节 生成字符串……89
第二节 模式识别的“中心”问题……95
第三节 划分凸多边形……99
第四节 防卫导弹……101
第五节 邮票问题……105
第六节 骨牌矩阵……108
第七节 师生树……114
第八节 旅游预算……120
第九节 正整数竖式除法……128
第十节 移棋子……132
第五章 习题……142
第一节 习题……142
第二节 部分习题测试数据及参考答案……152

第六章 习题解答.....174
第一节 电子表格 (Table) 174
第二节 DEL 命令 (DEL).....177
第三节 分割方格 (Divide) 182
第四节 信息编码 (Decode).....190
第五节 海上交通控制 (Lane).....192
第六节 投递最佳路线(Best Deliver) 200
第七节 计算机网络连接(computer network).....207
第八节 联系圈(Circle).....211
第九节 球钟(Ball Clock).....214
第十节 建筑物(Buildings).....217
附录： 1997-2000 年（第 22~25 届）ACM 国际大学生程序设计竞赛（ACM/ICPC）亚洲区预赛成绩 227
参考资料.....230

第一章 Delphi 简介

第一节 Delphi 的运行环境

1.1.1 Delphi 简介

Delphi 是著名的 Borland 公司开发的可视化软件开发工具。“真正的程序员用 c，聪明的程序员用 Delphi”，这句话是对 Delphi 最经典、最实在的描述。Delphi 被称为第四代编程语言，它具有简单、高效、功能强大的特点。和 VC 相比，Delphi 更简单、更易于掌握，而在功能上却丝毫不逊色；和 VB 相比，Delphi 则功能更强大、更实用。可以说 Delphi 同时兼备了 VC 强大的功能和 VB 简单易学的特点。因此，它一直是程序员至爱的编程工具。

Delphi 具有以下的特性：基于窗体和面向对象的方法，高速的编译器，强大的数据库支持，与 Windows 编程紧密结合，强大而成熟的组件技术。但最重要的还是 Object Pascal 语言，它才是一切的根本。Object Pascal 语言是在 Pascal 语言的基础上发展起来的，简单易学。

Delphi 提供了各种开发工具，包括集成环境、图像编辑（Image Editor），以及各种开发数据库的应用程序，如 Desktop DataBase Expert 等。除此之外，还允许用户挂接其它的应用程序开发工具，如 Borland 公司的资源编辑器（Resourse Workshop）。

在 Delphi 众多的优势当中，它在数据库方面的特长显得尤为突出：适应于多种数据库结构，从客户机 / 服务机模式到多层数据结构模式；包括高效率的数据库管理系统和新一代更先进的数据库引擎；具备最新的数据分析手段和提供大量的企业组件。

Delphi 发展至今，从 Delphi 1 、Delphi 2 到现在的 Delphi6，不断添加和改进各种特性，功能越来越强大。本书将以 Delphi6 为基础，介绍 Delphi 的开发环境、基本概念，让读者能在 Delphi 环境下编写竞赛程序。

1.1.2 Delphi 的 IDE 环境

启动 Delphi 后，屏幕上会出现如图 1.1 所示的界面，这是使用 Delphi 开发 Windows 应用程序和 Internet 应用程序常见的界面。

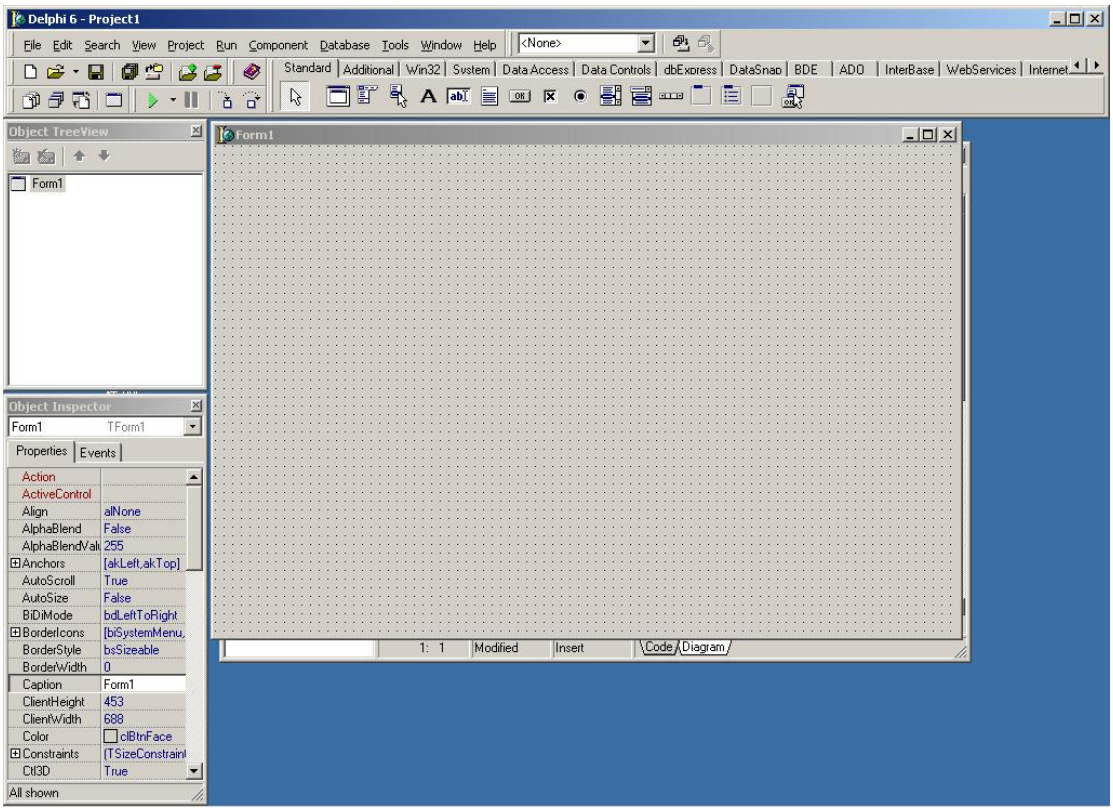


图 1.1

与上面两种应用不用，编写竞赛程序常常是使用控制台应用程序（Console Application），创建一个新的控制台应用程序可以使用如下的步骤：

（1）使用菜单 File -> New -> Other，如图 1. 2 所示

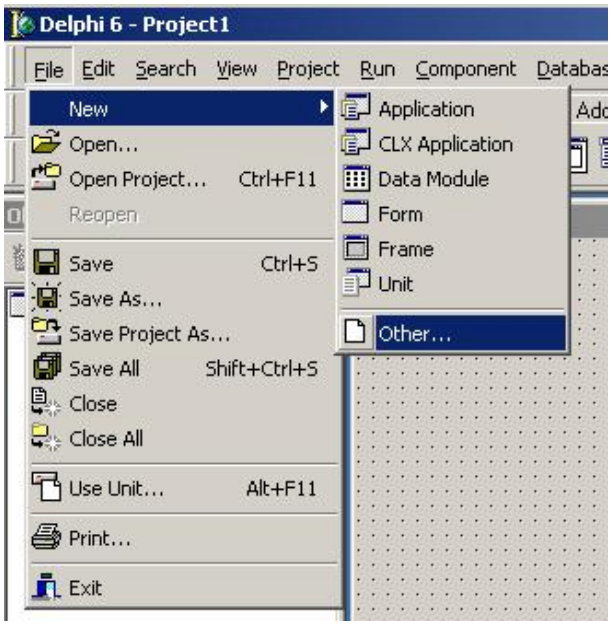


图 1. 2

（2）在 New Items 对话框中选择新建“Console Application”（ 图 1. 3）

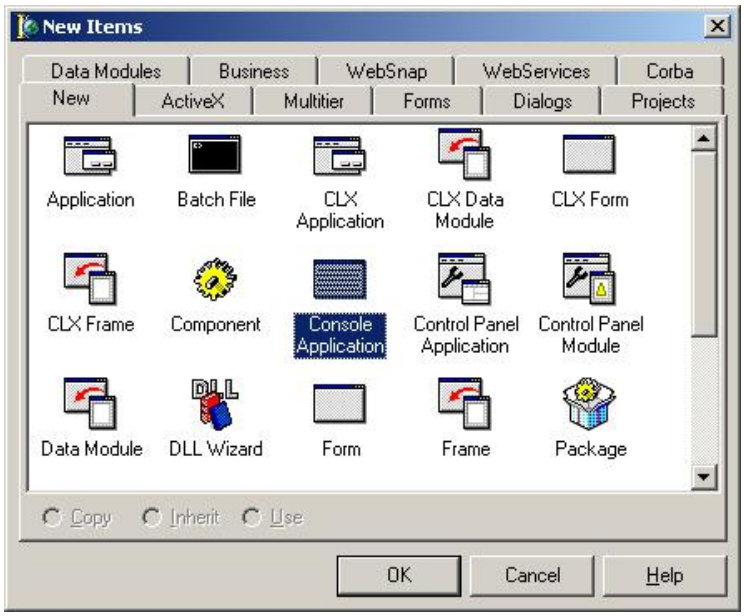


图 1. 3

新建的控制台应用程序界面如图 1. 4 所示：

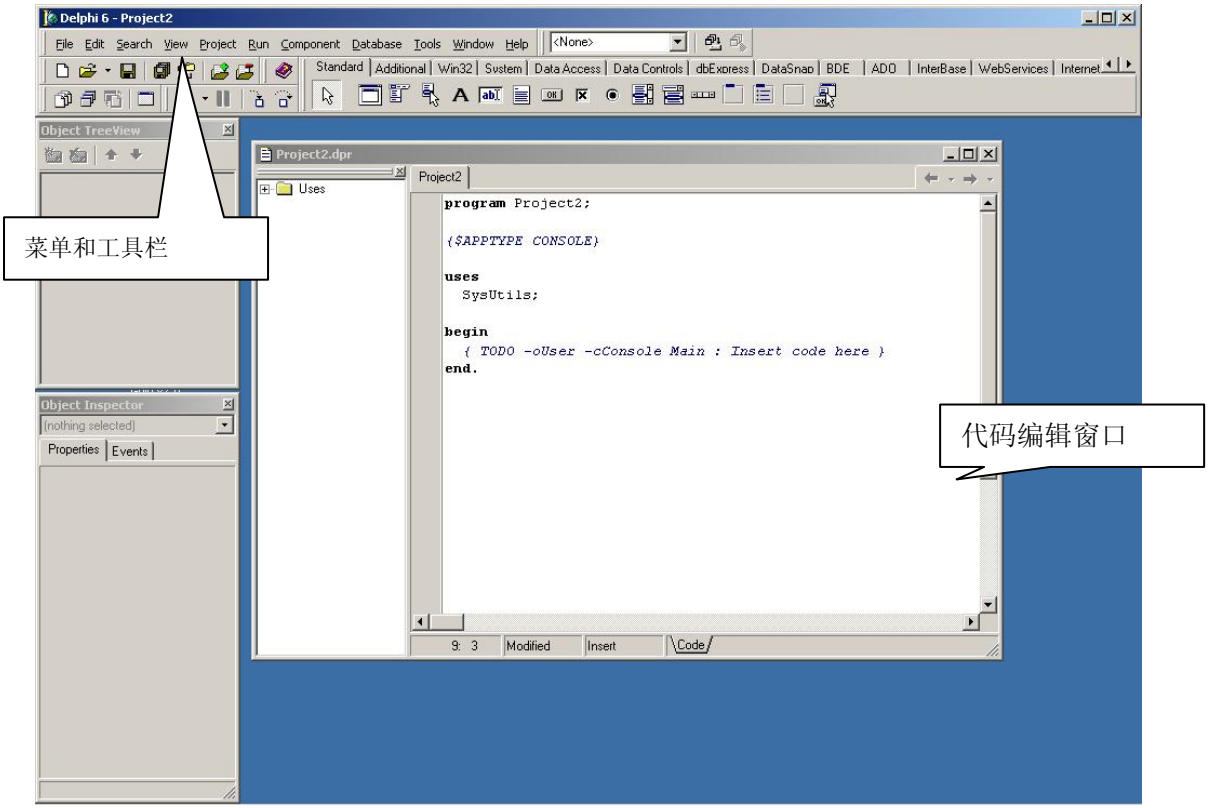


图 1. 4

下面将开始我们的第一个程序：Hello 程序，并且在 Delphi 环境中运行这个程序。程序在代码编辑窗口中编辑，例如可以在代码编辑窗口中输入 Hello 程序：

```
program Hello;

{$APPTYPE CONSOLE}
```

```
uses
  SysUtils;

begin
  { TODO -oUser -cConsole Main : Insert code here }
  WriteLn('Hello World.');
  ReadLn;
end.
```

其中，只有下划线部分是手工输入的，其他部分是 Delphi 产生的代码。语句“WriteLn('Hello World.')”是输出“Hello World.”的字样；语句“ReadLn”是等待输入一个回车符号，以便观察程序输出结果。

然后选择菜单”Run->Run”运行程序。

这时，屏幕中会出现“Hello World.”的字样，如图 1.5 所示。

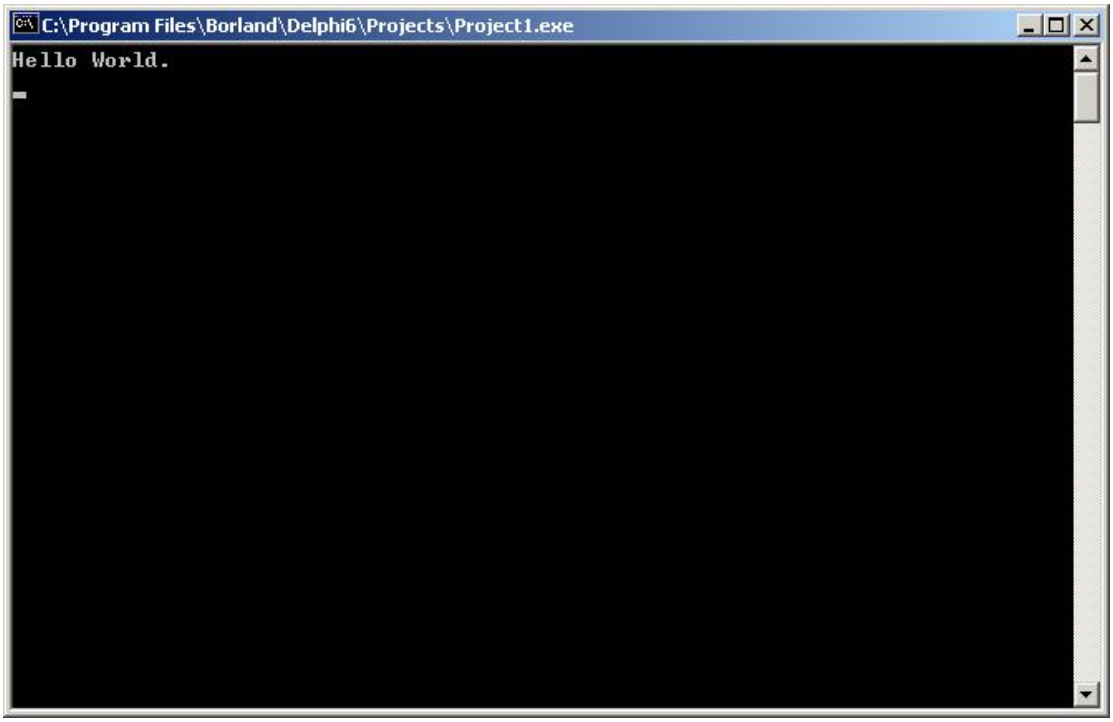


图 1.5

1.1.3 Delphi 程序的编译、运行和调试

下表列出了编译、运行和调试 Delphi 程序的常用命令：

功能	菜单命令	快捷方式
编译（Compile）	Project -> Complite	Ctrl + F9
运行（Run）	Run -> Run	F9
单步跟踪（Trace Into）	Run -> Trace Into	F7
单步执行（Step Over）	Run -> Step Over	F8
执行到光标所在位置（Run to Cursor）	Run -> Run to Cursor	F4
设置（Breakpoint）	Run -> Add Breakpoint	鼠标单击代码行左边的蓝点
观察窗口（Watch）	Run -> Add Watch	Ctrl + F5

更多的命令请参考 Delphi 的联机帮助。

第二节 Delphi 常量

标识符是 Delphi 应用程序中一些量的名称，这些量包括变量(var)、常量(const)、类型(type)、过程(procedure)、方法(Method)及其他，Object Pascal 在应用标识符时，必须首先说明它们。Object Pascal 是强类型语言，它的编译器可以检查确保赋给变量或属性的值是正确的类型，以便于您改正错误。由于 Object Pascal 是编译语言，所以 Delphi 的执行速度要比使用解释语言快得多。在使用标识符前说明它们，可以减少程序错误并增加代码的效率。有一点需要特别说明的是 Object Pascal 中的标识符是不区分大小写的，这和 C/C++语言有很大的区别。例如：Delphi 和 delphi 被看成是同一个标识符。

常量说明是为一个标识符赋予一个值，在程序执行过程中是不可改变的。常量说明使用保留字“const”作为开头。格式为：
const 常量名=常量值;
下面的例子声明了三个常量：

```
const
Pi = 3.14159;
Age = 34;
ProductName = ‘Delphi’;
```

上文的三个常量分别是实型、整型和字符串型常量。常量用“=" 表示两边的值是相等的。和 C/C++语言不同，Object Pascal 的字符串常量是使用单引号（’）作为定界符的，例如上面的’Delphi’，而不是使用双引号（”）。

第三节 Delphi 变量

变量是程序代码中代表一个内存地址的标识符，而该地址的内存内容在程序代码执行时可以被改变。在使用变量前必须对它进行说明，即对它进行命名，并说明它的类型。在所有变量说明以前加上保留字 var。变量说明左边是变量的名称，右边则是该变量的类型，中间用(:)隔开。即格式为：
var 变量名:变量类型;
例如：

```
var
Value ,Sum : Integer;
Name : String;
```

在 Delphi 中，可以在变量声明同时赋初始值，格式为：
const 变量名:变量类型=变量初始值;
注意，此时是使用 const 保留字开头，而不是 var。
例如：

```
const
```

```
Value: Integer:=10;
Name : String='Gates';
```

第四节 Delphi 类型

Delphi 有两大数据类型，一类是系统已经预定义的，另一类是用户自定义的。 Object Pascal 有一些系统预定义的数据类型，这些类型包括有：整型、实型、布尔型、字符型、指针型；而用户自定义的数据类型有枚举型、子界型、数组型、集合型、记录型、对象型等，您可以利用这些用户预定义的数据类型来构造新的数据类型以满足程序的特定需要。

1.4.1 Delphi 预定义类型

Object Pascal 有多个预定义的数据类型，您可以说明任何这些类型的变量：

1．整型：

与 CPU 和操作系统相关的整型包括 Integer 和 Cardinal，在当前 32 位编译器下，取值范围如下：

类型	范围	格式
Integer	-2147483648..2147483647	signed 32-bit
Cardinal	0..4294967295	unsigned 32-bit

与 CPU 和操作系统无关的整型如下：

类型	范围	格式
Shortint	-128..127	signed 8-bit
Smallint	-32768..32767	signed 16-bit
Longint	-2147483648..2147483647	signed 32-bit
Int64	-2^63..2^63-1	signed 64-bit
Byte	0..255	unsigned 8-bit
Word	0..65535	unsigned 16-bit
Longword	0..4294967295	unsigned 32-bit

2．实型：

下表列出了实型的范围和存储格式：

类型	范围	有效位	占用字节(bytes)
Real48	2.9 x 10^-39..1.7 x 10^38	11-12	6
Single	1.5 x 10^-45..3.4 x 10^38	7-8	4
Double	5.0 x 10^-324..1.7 x 10^308	15-16	8
Extended	3.6 x 10^-4951..1.1 x 10^4932	19-20	10
Comp	-2^63+1 .. 2^63-1	19-20	8
Currency	-922337203685477.5808..922337203685477.5807	19-20	8

通用类型 Rea 在当前的解释下，等价于 Double。

类型	范围	有效位	占用字节(bytes)
Real	5.0 x 10^-324..1.7 x 10^308	15-16	8

3．布尔型：

Boolean，只包含 true 或 False 两个值，占用 1 字节内存。

4．字符型：

Char，一个 ASCII 字符；字符型的常量形式上和字符串型常量一样，都是使用单引号（’）作为定界符，例如：’A’。

5．字符串类型：

String 一串最长可达 2G 个 ASCII 字符。

6．指针型：

Pointer，可以指向任何特定类型。相当与 C/C++中的“void*”。

类型的兼容性：

整型类别和实型类别都各有五种类型，同一类别中，所有的类型与其他同类别的都相容，您可以将一种类型的值赋给相同类别中不同类型的变量或属性，而只需要这个值的范围在被赋值的变量或属性的可能值范围内。例如，对于一个 Shortint 型的变量，可以接受在-128 到 127 范围内的任意整数，例如，对于 Shortint 类型，您不能将 128 赋给它，因为 128 已经超出了 Shortint 的范围了；此时，如将范围检查功能打开(选用 Options|Project，并在 Compiler Options Page 中选择 Range Checking)，将会检查出一个范围错误；如果 Range Checking 没有被打开，那么程序代码将可以执行，但被赋值的值将不是您期望的值。

在一些情况下，您可以进行不同类型的变量或属性的赋值。一般来说，可以将一个较小范围的值赋给一个较大范围的值。例如，您可以将整型值 10 赋给一个能接受实型值的 Double 类型的变量，而使其值成为 10.0，但如果将一个 Double 类型的值赋给整型变量，则会出现类型错误。如果您不清楚类型的兼容性，可以参阅 Delphi 的在线帮助中“Type Compatibility and Assignment Compatibility”主题。

1.4.2 枚举类型

一个枚举型的声明列出了所有这种类型可以包括的值，枚举型的声明格式如下：

type typeName = (val1, ..., valn);

其中，typeName,val1,...,valn 都是合法的标识符。例如：

```
type
  TSound = (tsClick, tsClack, tsClock);
  TMyColor = (mcRed, mcBlue, mcGreen, mcYellow, mcOrange);
```

可以定义上述枚举类型的变量和对该变量赋值如下：

```
var MyColor:TMyColor;
MyColor:=mcBlue;
```

在枚举型中，括号中的每一个值都有一个由说明它的位置决定的整型值。例如 mcRed 有整型值 0，mcBlue 有整型值 1 等。您可以把 MyColor 说明为一个整型变量，并将每种颜

色赋一个整型值以达到相同的效果，但用枚举型会使得程序可读性好，编写容易。当您在枚举型中列出值时，您同时说明了这个值是一个标识符。例如您的程序中如果已经含有 TMyColor 类型且说明了 MyColor 变量，则程序中便不能使用 mcRed 变量，因为它已经被说明为标识符了。

1.4.3 子界类型

子界型是下列这些类型中某范围内的值：整型、布尔型、字符型或枚举型。任何形如 Low..High 的构造都表示一个子界类型，其取值范围是从 Low 到 High。例如：

```
type TColors = (Red, Blue, Green, Yellow, Orange, Purple, White, Black);
//（枚举类型）
type TMyColors = Green..White;
type
  SomeNumbers = -128..127;
  Caps = 'A'..'Z';
```

子界型限定了变量的可能取值范围。当范围检查打开时，(在库单元的 Implementation 后面有{$R*.DFM}字样表示范围检查打开，否则您可以在 Options|Project|Complier Options 中选择 Range Cheking 来打开范围检查)，如果变量取到子界以外的值，会出现一个范围检查错误。

1.4.4 数组类型

数组是某种数据类型的有序组合，其中每一个元素的值由其相对位置来指定，您可以在数组的某个位置上放置数据，并在需要时使用这些数据。数组的声明形式为：

array [indexType1, ..., indexTypen] **of** baseType

其中，indexType 是基本类型，它的范围应该在 2G 以内，baseType 是数组的基类型。一般来说，indexType 是子界类型。请看下面的例子：

```
//声明数组
var MyArray: array[1..100] of Char;            //Example 1
type TMatrix = array[1..10] of array[1..50] of Real; //Example 2
type TMatrix = array[1..10, 1..50] of Real;    //Example 3
var MyMatrix:TMatrix;                          //Example 4
//Usage of Arrays 使用数组元素的范例
MyArray[1]:='A';
MyMatrix[1][50]:=0.0;
MyMatrix[10,50]:=1.0;
WriteLn(MyArray[10]);
```

Example 1 中，声明了一维数组 MyArray，包含 100 个 Char；Example2 和 Example3 都可以定义了二维数组的类型，这两种写法都是等价的。Example4 声明上述二维数组类型的变量。

您也可以在声明数组时，给数组赋初始值，例如：

```
//给数组赋初始值
```

```
const MyArray:array [1..3,1..3] of Integer
=((1,2,3),(4,5,6),(7,8,9));
const MyChars:array [1..2] of Char=('A','B');
```

1.4.5 字符串类型

字符串类型本质上是一个一维的字符数组。当您说明一个字符串型的变量时，您可以指明这个字符串的大小，下面是说明字符串类型的例子：

type MyString: string[15];

var MyName: MyString;

var tmpString:string;

则变量 MyName 被说明成为最多可以包含 15 个字符。如果您没有说明字符串的大小，Delphi 会认为字符串最多包含最大值 2G 个字符（如变量 tmpString）。给字符串赋值可以直接使用单引号括起的字串赋值：

MyName := 'Bill Gates';

因为 MyName 是一个可以包含 15 个字符的 MyString 型变量。当您给字符串型变量赋的值多于定义数值时，例如将 MyName 赋为‘FrankSmith.Franklin’，则 Delphi 只会接受前 15 个字符‘FrankSmith.Fran’。您可以使用索引值来访问字符串的字符，例如，用 MyName[1] 可以得到 MyName 的第一个字符'B'，也可以直接地修改字符串中的字符，例如：MyName[1]=’F’。

您可以充分利用 Delphi 丰富的运算符、过程和函数来处理字符串型的变量和属性。详细的请参阅“Delphi 函数与过程”中关于常用的字符串函数部分的内容。

1.4.6 集合类型

集合类型是一群相同类型元素的组合，这些类型必须是有限类型如整型、布尔型、字符型、枚举型和子界型，并且该类型的元素个数不得多于 256 个。在检查一个值是否属于一个特定集合时，集合类型非常有用。声明集合使用“**set of**”来声明，请看例子：

```
type                                        //定义集合类型
  TSomeInts = 1..250;
  TIntSet = set of TSomeInts;
type TIntSet2 = set of 1..250;              //和上面的方式等价
var Set1, Set2: TIntSet;                    //声明集合变量
Set1 := [1, 3, 5, 7, 9, 100..200];          //给集合赋值
Set2 := [2, 4, 6, 8, 10];
```

集合使用方括号为定界符，里面的元素使用逗号作为分隔。

集合的操作也比较丰富，常用的操作见下表：

操作符	操作说明	参数类型	结果类型	例子
+	合集	集合,集合	集合	Set1 + Set2
-	差集	集合,集合	集合	S - T
*	交集	集合,集合	集合	S * T
<=	子集	集合,集合	布尔	Q <= MySet

>=	超集	集合,集合	布尔	S1 >= S2
=	等于	集合,集合	布尔	S2 = MySet
<>	不等	集合,集合	布尔	MySet <> S1
in	成员	基类型,集合	布尔	A in Set1

1.4.7 记录类型

记录是您的程序可以成组访问的一组数据的集合。下面的例程说明了一个记录类型的用法：

```
type
TEmployee=record
    Name : string[20];
    YearHired:1990..2000;
    Salsry: Double;
    Position: string[20];
end;
```

记录包含可以保存数据的域，每一个域有一个数据类型。上文的记录 TEmployee 类型就含有四个域。您可以用以下的方式说明记录型的变量：

```
var NewEmployee,PromotedEmployee:TEmployee;
```

用如下的方法可以访问记录的单域：

```
NewEmployee.Salary := 1000;
```

编写如下的语句可以给整个记录赋值：

```
with PromotedEmployee do
begin
  Name :='';
  YearHired := 1993;
  Salary := 2000.00
  Position := 'editor';
end;
```

您的程序可以将记录当成单一实体来操作：

```
PromptEmployee := NewEmployee;
```

以上介绍了用户常用的自定义类型。在 Delphi 的编程中，对象是非常重要的用户自定义数据类型。象记录一样，对象是结构化的数据类型，它包含数据的域(Field)，也包含作为方法的过程和函数。关于对象的详细内容请参考“面向对象 Object Pascal”一节。

1.4.8 指针类型

您可以通过下面的语法声明任何类型的指针：
type pointerTypeName = ^BaseType
当您定义了一个记录或者是其他数据类型时，通常都会定义一个指向这种类型的指针，这样使用这种数据类型的实例时，可以避免大量重复的内存数据拷贝。
下面是定义指针的例子：

```
type
TEmployee=record
    Name : string[20];
    YearHired:1990..2000;
    Salsry: Double;
    Position: string[20];
end;
type PEmployee=^TEmployee;
```

创建和释放指针，分别使用 New 和 Dispose 过程，下面是使用指针的例子：

```
var aEmployee:PEmployee;
begin
  New(aEmployee);                  //创建指针
  aEmployee^.Name:='Gates';        //使用指针
  Dispose(aEmployee);              //释放指针
end.
```

注意：在使用指针时，“^”的位置与 C/C++语言不同。

第五节 Delphi 的基本语句

1.5.1 赋值语句

赋值语句用作把一个表达式的值赋给一个变量，例如，计算圆面积：

```
S:=Pi*R*R;
```

在 Delphi 中，赋值号为“:=”。赋值时，要求表达式值的类型和变量的类型兼容，关于兼容性的问题，请参考“Delphi 类型”一节。

1.5.2 复合语句

在 Delphi 中，一个语句是以分号“**;**”作为语句结束符的。复合语句的作用是把若干个语句组合起来，在逻辑上看成是一个语句。复合语句的格式是使用“**begin**”和“**end**”把一组语句组合起来，例如：

```
if J <> 0 then
begin
  Result := I/J;
  Count := Count + 1;
end
else if Count = Last then
  Done := True
else
  Exit;
```

1.5.3 分支语句

Delphi 的跳转语句有 if 和 case 两个，下面就分别介绍这两个语句。

if 语句

if 语句通过计算其后的表达式，并根据计算结果来决定程序执行的流程。当表达式为 True 时，执行 then 后的语句。否则执行 else 后的代码，if 语句也可以不含 else 部分，表达式为 False 时自动跳到下一行程序。

if 语句可以嵌套，当使用复合语句表达时，复合语句前后需加上 begin…end。else 保留字前不能加“;”，而且，编译器会将 else 语句视为属于最靠近的 if 语句。必要时，须使用 begin…end 保留字来强迫 else 部分属于某一级的 if 语句。

if 语句的例子可以参考前述的“复合语句”中的例子。

case 语句

case 语句适用于多分支的条件判断，用 case 语句进行逻辑跳转比编写复杂的 if 语句容易阅读。case 的语法格式如下：

```
case selectorExpression of
  caseList1: statement1;
  ...
  caseListn: statementn;
else
    elsestatement;
end;
```

其中，selectorExpression 是序数类型的表达式（如整型，字符型，枚举型和子界类型），例如：

```
case I of
  1..5: Caption := 'Low';
  6..9: Caption := 'High';
  0, 10..99: Caption := 'Out of range';
else
  Caption := '';
end;
```

1.5.4 循环语句

Delphi 的循环语句有三种：repeat、while 和 for 语句。

repeat 语句

repeat 语句会重复执行一行或一段语句直到某一状态为真。语句以 repeat 开始，以 until 结束，其后跟随被判断的布尔表达式。参阅以下的例程：

```
i := 0;
repeat
i := i+1;
Writen(i);
until i=10;
```

程序运行的结果是：1 到 10 的 10 个数字。布尔表达式 i=10 (注意，与 C/C++语言不同的是，“=”是关系运算符，而不能进行赋值操作)直到 repeat..until 程序段的结尾才会被计算，这意味着 repeat 语句至少会被执行一次。

while 语句

while 语句和 repeat 语句的不同之处是，它的布尔表达式在循环的开头进行判断。while 保留字后面必须跟一个布尔表达式。如果该表达式的结果为真，循环被执行，否则会退出循环，执行 while 语句后面的程序。

下面的例程达到和上面的 repeat 例程达到同样的效果：

```
i := 0;
while i<10 do
begin
  i := i+1;
  writeln(i);
end;
```

for 语句

for 语句的程序代码会执行一定的次数。它需要一个循环变量来控制循环次数。您需要说明一个变量，它的类型可以是整型、布尔型、字符型、枚举型或子界型。

下面的程序段达到和上面的 repeat 和 while 例程相同的效果。

```
var I:Integer
For I:=1 to 10 do
  WriteLn(I);
```

上述的程序段中，for 语句也可以把循环变量（I）是递增的，在实际应用中，有时需要循环变量是递减的，这时，只需要把 **to** 改成 **downto** 就可以了。

break 和 **continue**

这两个语句的作用和在 C/C++语言中的作用一样，起到控制循环的流程的作用，break 语句的作用是跳出循环，而 continue 语句的作用是转入下一轮的循环。

以上介绍了三种循环语句。如果您知道循环要执行多少次的话，可以使用 for 语句。for 循环执行速度快，效率比较高。如果您不知道循环要执行多少次，但至少会执行一次的话，选用 repeat..until 语句比较合适；当您认为程序可能一次都不执行的话，最好选用 while..do 语句。

1.5.5 输入输出语句

Delphi 提供了简单的输入输出语句（过程）实现从文本文件或者是标准设备输入和输出文本流。相对于 C/C++的输入输出而言，Delphi 的输入输出会显得更加简单。

输入语句

procedure **Readln**([var F: Text;] V1 [, V2, ...,Vn]);

procedure **Read**([var F: Text;] V1 [, V2,...,Vn]);

如果变量 F 省略的话，表示从标准设备（键盘）中读入输入，保存在变量 V1，V2，...，Vn 中。Vk 可以是整型、实型、字符型或者字符串型变量，如果是字符串型变量，系统会读到回车符为止（不是空格）。

ReadLn 和 Read 的区别在于：ReadLn 把读入的值保留在变量 V1，V2，...，Vn 后，会自动换行，而 Read 不会。

请看下面的例子：

程序 ex1_5_5.dpr

```
program ex1_5_5;
{$APPTYPE CONSOLE}
uses  SysUtils;
Var F:Text;
Var a,b:Integer;
begin
  Assign(F,'ex1_5_5.dat');          //指定读入文件
  ReSet(F);                         //以读的方式打开文件
  Read(F,a);
  Read(F,b);
  WriteLn(a,' ',b)
  Close(F)                          //关闭文件
end.
```

文件 ex1_5_5.dat

```
1   2
3
```

运行程序后，结果为“1 2”，而如果把程序中的 Read 改为 ReadLn，程序运行结果会变为“1 3”。

输出语句：

procedure **Write**([var F: Text;] P1 [, P2, ...,Pn]);

procedure **Writeln**([var F: Text;] P1 [, P2, ...,Pn]);

把表达式 P1，P2，...，Pn 输出到文本文件 F 中，如果省略变量 F，则表示输出到标准设备（屏幕）中。表达式 P1，P2，...，Pn 的类型可以是整型、实型、字符型和字符串型。

Write 和 Writeln 的区别在于：Writeln 在输出完表达式的之后，紧接着输出一个回车符。

下面是一个使用文件输出的例子：

程序 ex1_5_5_2.dpr

```
program ex1_5_5_2;
{$APPTYPE CONSOLE}
uses  SysUtils;
var F:Text;
begin
  Assign(F,'ex1_5_5_2.dat');          //指定输出文件
  ReWrite(F);                         //以写(覆盖)的方式打开文件
  WriteLn(F,1,' ',2);
  WriteLn(F,1+2);
  Close(F);                           //关闭文件
end.
```

程序的输出文件和前面的 ex1_5_5.dat 相同。

需要特别说明的是，如果使用编译指令{$I+}，当遇到 I/O 错误时，系统会报错；如果我们需要自行检查错误，可以使用编译指令{$I-}，然后调用函数 IOResult 来检测是否有 I/O 错误。

例如，程序 ex1_5_5.dpr 中的“ReSet(F);”改成“{$I-}ReSet(F);{$I+}”，然后下面判断 IOResult 的值是否为 0，就可以判断文件是否被正常打开，并作出相应的处理。

第六节 Delphi 函数与过程

1.6.1 函数和过程的调用

在 Delphi 中，过程和函数的主要区别在于函数有返回值，而过程没有。

过程和函数的调用例子如下：

```
var X:array [1..100] of Integer
Fillchar(X,sizeof(X),0);    //在这里调用了 FillChar 过程，对数组 X 清零

var L:Integer;S:String;
S:='Delphi 6.0';
L:=Length(S);               //在这里调用了求字符串的长度的函数
```

1.6.2 常用的函数

Delphi 提供的常用函数和过程有：

算术函数：

function Abs(X);	返回 X 的绝对值
function Ceil(const X: Extended):Integer;	返回不小于 X 的最小整数
function Exp(X: Real): Real;	返回 e 的 X 次方

function Floor(const X: Extended): Integer; 返回不大于 X 的最大整数
function Frac(X: Extended): Extended; 返回 X 的小数部分
function Int(X: Extended): Extended; 返回 X 的整数部分（结果是实型）
function Ln(X: Real): Real; 返回 X 关于 e 的对数
function Pi: Extended; 返回圆周率
function Power(const Base, Exponent: Extended): Extended;
返回 Base 的 Exponent 次方
function Round(X: Extended): Int64; 返回 X 四舍五入的取整值
function Sqr(X: Extended): Extended;
function Sqr(X: Integer): Integer;
返回 X 的平方
function Sqrt(X: Extended): Extended; 返回 X 的平方根
function Trunc(X: Extended): Int64; 返回 X 的整数部分（结果是整型）

字符串函数（过程）：

function Concat(s1 [, s2,..., sn]: string): string; 连接两个（或者多个）字符串，相当于'+'
function Copy(S; Index, Count: Integer): string; 提取字符串中的一个子串
procedure Delete(var S: string; Index, Count:Integer);
删除字符串中的一个子串
procedure Insert(Source: string; var S: string; Index: Integer);
插入一个子串
function Length(S): Integer; 求字符串的长度
function LowerCase(const S: string): string; 返回字符串的小写形式
function Pos(Substr: string; S: string): Integer; 求子串位置
function Trim(const S: string): string; 删除首尾空格
function UpperCase(const S: string): string; 返回字符串的大写形式
procedure Val(S; var V; var Code: Integer); 把字符串转成数值类型

其他函数（过程）：

procedure Inc(var X [; N: Longint]); 变量 X 增加 N。（N 缺省值为 1）
procedure Dec(var X [; N: Longint]); 变量 X 减少 N。（N 缺省值为 1）
function Ord(X); 求字符的 ASCII 编码
function Chr(X: Byte): Char; 求 ASCII 码为 X 的字符
procedure FillChar(var X; Count: Integer; Value: Byte);
从 X 开始的地址连续填充 Count 个值

1.6.3 自定义函数和过程

自定义过程和函数的格式如下：

```
//声明过程
procedure procedureName(parameterList); directives;
localDeclarations;
begin
  statements
end;
//声明函数
```

```
function functionName(parameterList): returnType; directives;
localDeclarations;
begin
  statements
end;
```

其中，procedureName 和 functionName 分别是过程名和函数名，必须是合法的标识符；parameterList 是参数列表，参数之间使用分号“;”分隔；directives 是指示字，一般在竞赛中使用不多；localDeclareations 是函数内部的说明部分，说明部分可以包括类型说明、变量说明、常量说明等。statements 是过程或函数的语句部分。当函数需要一个返回值时，returnType 作为返回值的类型。具体执行时，可以将返回值赋给函数名称，或将返回值赋给 Result 变量。

```
function CalculateInterest(Principal,InterestRate: Double):Double;
begin
CalculateInterest := Principal * InterestRate;
//以下的写法也是合法的
//Result := Principal*InterestRate;
end;

//下面是这个函数的调用方法：
InterestEarned :=CalculateInterest(2000,0.012);
```

在 Delphi 中，参数的传递可以有支持传值调用和引用调用两种形式，在默认的情况下，是使用传值调用；只有在参数前面加上 var，该参数传递才是引用调用。请看下面的例程：

```
//传值调用
procedure IncByValue(X:Integer)
begin
  X:=X+1;
end;
//引用调用
procedure IncBeRef(var X:Integer)
begin
  X:=X+1;
end;

var Y:integer;
Y:=0;
IncByValue(Y);          //Y=0 now
IncByRef(Y);            //Y=1 now
```

第七节 函数的递归

函数递归调用有两种形式，一种是直接调用，即自己调用自己；另一种是间接调用，即 A 调用 B，同时，B 也可调用 A。

1．直接递归调用

例如：下面是求 n 的阶乘的程序：

```
function f(n:Integer):Integer;
begin
  if (n=0) then Result:=1
  else Result:=n*f(n-1);
end;
```

在函数 f 中，调用了函数 f。

2．间接递归调用

在 Object Pascal 中，过程或函数必须先说明再调用。以上规则在递归调用时是例外情况。在递归调用中，函数要进行前置，即在函数或过程的标题部分最后加上保留字 forword。

```
var alpha:Integer;
procedure Test2(var A:Integer):forword;
//{Test2 被说明为前置过程}
procedure Test1(var A:Integer);
begin
A :=A-1;
  if A>0 then
  test2(A); //{经前置说明，调用未执行的过程 Test2}
  writeln(A);
end;
procedure Test2(var A:Integer);//{经前置说明的 Test2 的执行部分}
begin
  A :=A div 2;
  if A>0 rhen
  test1(A); //{在 Test2 中调用已执行的过程 Test1}
end;
//主程序
begin
  Alpha := 15; //{给 Alpha 赋初值}
  Test1(Alpha); //{ 第一次调用 Test1,递归开始}
end.
```

Alpha 赋初值后，调用 Test1，实现先减 1 再除 2 的循环递归调用，直到 Alpha 小于 0 为止。

第八节 面向对象 Object Pascal

Delphi 是基于面向对象编程的先进开发环境。面向对象的程序设计(OOP)是结构化语言的自然延伸。OOP 的先进编程方法，会产生一个清晰而又容易扩展及维护的程序。一旦您为您的程序建立了一个对象，您和其他的程序员可以在其他的程序中使用这个对象，完全不必重新编制繁复的代码。对象的重复使用可以大大地节省开发时间，切实地提高工作效率。

但是就竞赛而言，竞赛是富于创造性的活动，代码的重用率不高，因此对象在实际中使用的频率不高。

下面主要讨论使用 Delphi 中的对象，至于 Delphi 如何实现对象机制，请参考 Delphi 的联机帮助或者相关参考书。

使用对象前，需要对对象实例化，一般是通过调用 Create 方法来完成，释放对象的实例，一般通过调用 Free 方法来完成。使用对象中的属性和方法，通过“对象名.属性”和“对象名.方法”，例如：（TStringList 是字符串列表类）

程序 ex1_8.dpr

```
program ex1_8;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  Classes;
var
  T:TStringList;                        //声明对象
begin
       T := TStringList.Create;         //对象实例化
       T.Add('Delphi 6.0');             //调用对象的方法
       WriteLn(T.Count);                //引用对象的属性
       T.Free;                          //释放对象
       ReadLn;
end.
```

第九节 Delphi 中使用嵌入汇编

在 Delphi 中可以直接嵌入汇编，使用汇编的好处是可以提高程序运行的效率，但是，另一方面，使用汇编会加大编程的难度和调试的难度。因此在竞赛中往往只要一些关键而又简单的小程序段才使用汇编来编写。

Delphi 目前是使用 32 位的汇编，请先看下面的例子：

```
program ex1_9;
{$APPTYPE CONSOLE}
uses
  SysUtils;
function Counter(var A;Count:Integer):Integer;
label lbegin,lloop;
begin
       asm
              Mov ESI,&A;
              Mov ECX,&Count;
              Mov EAX,0;
              Mov BL,0;
       lbegin: CMP [ESI],BL;
              JNE lloop;
              Inc EAX;
       lloop:  Inc ESI;
              loop lbegin;
              Mov &Result,EAX;
       end;
end;
var X:array [1..10] of Byte;
begin
       Fillchar(X,sizeof(X),0);
       X[5]:=1;
       WriteLn(Counter(X,10));
       ReadLn;
end.
```

函数 Counter 的功能是统计长度为 Count 的 Byte 类型数组 A 中“0”的个数。

使用汇编应该注意：

（1）嵌入汇编是以”asm”开头，并以”end”结尾的部分，进入和退出汇编是不需要保存和回复各个寄存器的值，因为这项工作 Delphi 会自动完成。

（2）汇编中使用的标号，被认为是 Delphi 的标号，需要使用 label 来声明。

（3）在汇编中，对 Delphi 原有变量的引用，需要使用“&”符号，当然在没有歧异的情况下，“&”可以省略。有歧异是指，例如：“CH”，“&CH”是不同的两个概念，前者是寄存器的引用，而后者是对 Delphi 的变量“CH”的引用。而象上面例子中的 A,Count 等，前面的“&”都可以省略。

关于嵌入汇编的进一步说明，请参考 Delphi 联机帮助中“Inline Assembler Code”部分。

第二章 基本算法介绍

第一节 概述

从 60 年代末开始，程序设计已经从技巧发展成一门科学。与程序设计联系最密切的课题有数据结构、算法分析与设计和程序设计方法。著名的计算机科学家沃斯（N. Wirth）提出了“算法+数据结构＝程序”的著名观点。

数据是客观事物表示的一种抽象结果，而数据结构是研究如何把客观世界待处理的信息逐层抽象成计算机可以接受的某种形式。

所谓的算法是指解题方法的精确描述，它是有穷处理动作的序列。数据结构与算法有着密切的联系，数据结构是建立在算法的基础之上的。

在这章里，首先在第二节介绍常用的数据结构及其在 Delphi 中的实现，为介绍算法做好准备。然后在第三至第九节，依次介绍一些基本的算法，如枚举算法、贪心算法、分治算法、数值计算和计算几何里的算法，以及模拟题的解法。

第二节 常用数据结构在 Delphi 中的实现

2.2.1 线性表

1．概述

线性表是由有限具有相同的数据类型个元素组成的有序集合，线性表可以表示成：

（$a_0,a_1,a_2,...,a_{i-1},a_i,a_{i+1},...,a_n$）

对线性表常见的操作有：

（1）求线性表的长度。

（2）求线性表中第 k 个位置的元素。

（3）插入元素

（4）删除元素

（5）查找元素

线性表实现的方法，一般来说有两种，一种是采用数组的方式实现，另一种是采用链表的方式实现。

数组实现方式的优点在于：实现简单，易于编程和调试；对（1）（2）的操作速度快（时间复杂度为 O（1））。

而使用链表的方式的优点在于：对（3）（4）的操作速度快（时间复杂度为 O（1））；可以容纳无限个元素（当然还会受实际内存的限制）。

2．线性表的数组方式实现

线性表可以采用如下方式定义：

```
type
```

```
TLinearList=record
  len:Integer;
  element:array [0..max] of Pointer;
end;
```

其中，max 是最多容纳元素的个数。
实现上述的五种操作的程序段如下：

```
unit LinearList;

......

//求线性表的长度
function LLength(var alist:TLinearList):Integer;
begin
       Result:=alist.len;
end;
//求线性表中第 n 个位置的元素。
function LGet(var alist:TLinearList;n:Integer):Pointer;
begin
       if (n<0) or (n>=alist.len) then Result:=nil
       else Result:=alist.element[n];
end;
//位置 n 前插入元素
procedure LInsert(var alist:TLinearList;n:Integer;Value:Pointer);
begin
       if n>=alist.len then
              alist.element[alist.len] := Value
       else
       begin
              Move(alist.element[n],alist.element[n+1],
                SizeOf(Pointer)*(alist.len-n));
              alist.element[n] := Value
       end;
       Inc(alist.len);
end;
//删除位置为 n 的元素
procedure LDelete(var alist:TLinearList;n:Integer);
begin
       if (n>=0) and (n<alist.len) then
       begin
              Move(alist.element[n+1],alist.element[n],
                SizeOf(Pointer)*(alist.len-n-1));
              Dec(alist.len);
       end;
end;
//查找元素
function LFind(var alist:TLinearList;Value:Pointer):Integer;
var I:Integer;
begin
       for I:=0 to alist.len -1 do
       if alist.element[I]=Value then
       begin
              Result:=I;
              exit;
       end;
       Result:=-1;
```

```
end;
```

说明，（1）Pointer 是 Delphi 中比较通用的类型，可与其他类型转换。（2）在实现中，用到了 Move 函数，在运行速度方面要比使用循环逐个元素移动快得多。（3）程序中没有对元素个数多于 EMAX 进行判断，这是因为在实际的竞赛中，我们总是开出一个足够大的数组，不需要考虑溢出问题。

下面是使用上面函数的例子，主要用意是演示如何在 Pointer 之间 Integer 的转换。

程序 ex2_2_1.dpr

```
program ex2_2_1;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  LinearList in 'LinearList.pas';
var a:TLinearList;
begin
      LEmpty(a);
      LInsert(a,0,Pointer(2));
      LInsert(a,0,Pointer(1));
      WriteLn(Integer(LGet(a,0)),' ',Integer(LGet(a,1)));
      LDelete(a,0);
      WriteLn(Integer(LGet(a,0)),' ',Integer(LGet(a,1)));
      LInsert(a,0,Pointer(1));
      WriteLn(Integer(LGet(a,0)),' ',Integer(LGet(a,1)));
      WriteLn(Integer(LFind(a,Pointer(2))));
      ReadLn;
end.
//屏幕输出为：
//1 2
//2 0
//1 2
//1
```

3．线性表在 Delphi 中的实现

线性表的链表实现，在 Delphi 中被封装成 TList 类，您可以直接使用，而不需要自己重写。Delphi 会自动对 TList 占用的内存进行管理，您可以认为 TList 可以容纳无限多个元素。如果您对如何用链表实现线性表感兴趣，请参考关于数据结构方面的书籍。

下面简单介绍一些 TList 类中的主要属性和方法。

property Count: Integer; 线性表中元素个数

property Items[Index: Integer]: Pointer; 取出或者设置表中的元素，是**基 0** 数组

function Add(Item: Pointer): Integer; 增加元素

procedure Insert(Index: Integer; Item: Pointer); 插入元素

procedure Delete(Index: Integer); 删除元素

function IndexOf(Item: Pointer): Integer; 查找元素

说明，在删除元素时，仅仅是把该元素从列表中移走，并不释放该元素相关的内存。TList 在 Classes 单元中，在使用之前，需要引用该单元：“uses Classes”。

请看下面关于 TList 的使用例子：

```
{$APPTYPE CONSOLE}
```

```
uses Classes;
type
  PMyList = ^AList;
  AList = record
    I: Integer;
    C: Char;
  end;

var

  MyList: TList;
  ARecord: PMyList;
  B: Byte;
begin
  MyList := TList.Create; //创建线性表
  New(ARecord);
  ARecord^.I := 100;
  ARecord^.C := 'Z';
  MyList.Add(ARecord); //{把元素（100,’Z’）增加到列表中 }
  New(ARecord);
  ARecord^.I := 200;
  ARecord^.C := 'X';
  MyList.Add(ARecord); //{把元素（200,’X’）增加到列表中 }

  //输出列表内容
  for B := 0 to (MyList.Count - 1) do
  begin
    ARecord := MyList.Items[B];
    Writeln(ARecord^.I:10, ARecord^.C:10)
  end;

  //注意必须自行释放线性表中的元素占用空间
  for B := 0 to (MyList.Count - 1) do
    begin
      ARecord := MyList.Items[B];
      Dispose(ARecord);
    end;
  MyList.Free;    //清除线性表
end;
```

2.2.2 栈

1．概述

栈又称为后进先出（last in first out, LIFO）表，栈可以表示为：

（$a_1,a_2,a_3,...,a_{i-1},a_i,a_{i+1},...,a_n$）

其中，首元素 a1 称为栈底，最后一个元素 an 称为栈顶，栈的插入和删除操作都在栈顶进行。

对栈的操作主要有：

（1） 压栈，push(x)，把元素 x 放入栈顶。

（2） 出栈，pop()，取出栈顶元素（栈顶元素从栈中删除）。

和线性表类似，栈实现一般来说有两种，一种是采用数组的方式实现，另一种直接使用

Delphi 封装的类 TStack。

2．用数组实现栈

栈可以采用如下方式定义：

```
type TAStack=record
       stack:array [1..EMAX] of Pointer;
       top:Integer;
end;
```

下面的程序段实现上述对栈的操作：

```
//清空栈
procedure SEmpty(var astack:TAStack);
begin
       astack.top:=0;
end;
//压栈
procedure SPush(var astack:TAStack;X:Pointer);
begin
       Inc(astack.top);
       astack.stack[astack.top]:=X;
end;
//出栈
function SPop(var astack:TAStack):Pointer;
begin
       if astack.top > 0 then
       begin
              Result:= astack.stack [astack.top];
              Dec(astack.top);
       end
       else Result:= nil;
end;
```

3．栈在 Delphi 中的实现

Delphi 把栈封装成类 TStack。和线性表类似，从竞赛的角度看，您可以认为 TStack 可以容纳无限多个元素。

TStack 的主要属性和方法如下：

function Count: Integer; 栈中元素的个数

function Peek: Pointer; 不改变栈的状态，返回栈顶元素

function Pop: Pointer; 出栈

function Push(AItem: Pointer): Pointer; 压栈

TStack 在 Contnrs 单元中，如果您要使用 TStack 类，必须先使用“uses Contnrs”，引用该单元。

2.2.3 队列

1．概述

队列又称为先进先出（last in first out, LIFO）表，队列可以表示为：

（$a_1,a_2,a_3,\ldots,a_{i-1},a_i,a_{i+1},\ldots,a_n$）

其中，首元素 a1 称为队首，最后一个元素 an 称为队尾，队列的插入在队尾中进行，而删除操作在队首中进行。

对队列的操作主要有：

（1） 入队列，push(x)，把元素 x 插入队尾。

（2） 出队列，pop()，取出队首元素（队首元素从队列中删除）。

和线性表类似，队列实现一般来说有两种，一种是采用数组的方式实现，另一种直接使用 Delphi 封装的类 TQueue。

2．用数组实现队列

队列可以采用如下方式定义：

```
type TAQueue=record
       queue:array [1..EMAX] of Pointer;
       head,tail:Integer;
end;
```

下面的程序段实现上述对队列的操作：

```
//清空队列
procedure QEmpty(var aqueue:TAQueue);
begin
       with aqueue do
       begin
              head:=0;tail:=0;
       end;
end;
//入队列
procedure QPush(var aqueue:TAQueue;x:Pointer);
begin
       with aqueue do
       if tail<EMAX then
       begin
              Inc(tail);
              queue[tail]:=x;
       end;
end;
//出队列
function QPop(var aqueue:TAQueue):Pointer;
begin
       with aqueue do
       if head<tail then
       begin
              Inc(head);
              Result:=queue[head];
       end
       else Result:=nil;
end;
```

3．队列在 Delphi 中的实现

Delphi 把队列封装成类 TQueue。和线性表类似，从竞赛的角度看，您可以认为 TQueue 可以容纳无限多个元素。

TQueue 的主要属性和方法如下：

function Count: Integer; 队列中元素的个数

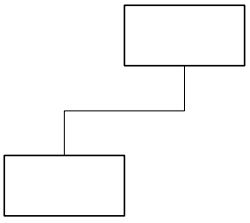
function Peek: Pointer;　不改变队列的状态，返回队首元素
function Pop: Pointer;　出队列
function Push(AItem: Pointer): Pointer; 入队列
说明，TQueue 和 TStack 在接口上非常相似，这是因为它们都是从 TOrderedList 中继承下来的，它们的区别只是在接口方法的实现上不同而已。

2.2.4 二叉树

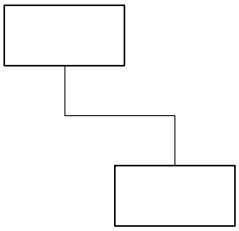
1．二叉树的概念
二叉树是由 n（n>=0）个结点的有限集合，这个集合或者是空集，或者是由根结点加上两个分别称为左子树和右子树的互不相交的二叉树组成。二叉树是使用递归方式定义的。
如图，下图中的都是二叉树：

(1)空二叉树

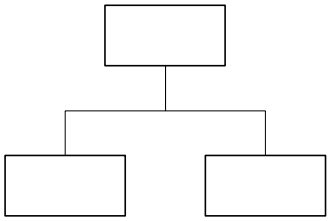
(2)仅有根的二叉树



(3)缺少右儿子的二叉树



(4)缺少左儿子的二叉树



(5)完全二叉树

图 2.1 各种二叉树

有 2^K-1 个结点的 K 层二叉树称为满二叉树，满二叉树是一种特殊的二叉树，如下图所示，下图是 4 层的满二叉树。

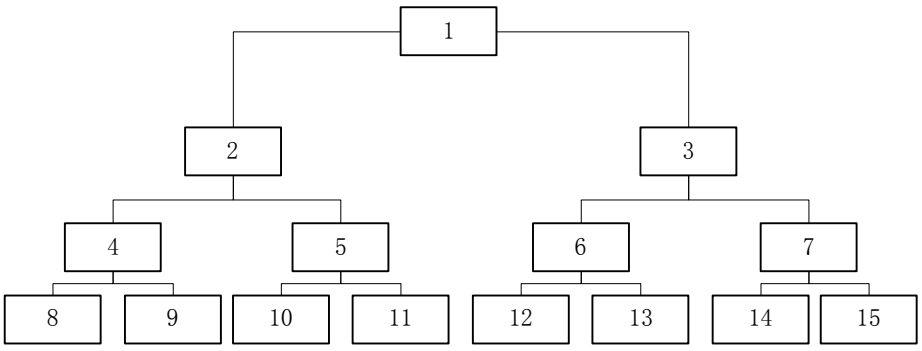


图 2.2 满二叉树

2．二叉树的存储
二叉树的存储主要需要考虑的问题是“如何表示二叉树父子之间的联系”，解决这个问题大致有两种方式，一种是直接的算术映射，采用数组来存储，另一种是使用指针链接的方式，采用动态存储。
第一种存储方式主要参考满二叉树的模型，如上图，可容易得到父子结点之间的编号关系：

编号为 n 的结点的左儿子编号为 2n，而右儿子的编号为 2n+1；编号为 n 的结点，其父结点的编号为[n/2]。

因此，二叉树可以定义如下：

```
Type TBinTree=array [1..EMAX] of Pointer
```

数组中下标为 n 的元素与满二叉树中编号为 n 的结点对应。

上述获得父结点，左子树和右子树的操作如下：

```
// 初始化二叉树（设为空树）
procedure BEmpty(var bt:TBinTree);
begin
       FillChar(bt,SizeOf(bt),0);
end;
//返回父结点，n 是二叉树编号，函数完成后，n 保存父结点的编号
function BParent(var bt:TBinTree;var n:Integer):Pointer;
begin
       if n>1 then
       begin
              n:=n div 2;
              Result:=bt[n];
       end
       else Result:=nil;
end;
//返回左儿子，n 是二叉树编号，函数完成后，n 保存左儿子的编号
function BLeft(var bt:TBinTree;var n:Integer):Pointer;
begin
       n:=n*2;
       Result:=bt[n];
end;
//返回右儿子，n 是二叉树编号，函数完成后，n 保存右儿子的编号
function BRight(var bt:TBinTree;var n:Integer):Pointer;
begin
       n:=n*2+1;
       Result:=bt[n+1];
end;
```

对于稀疏的二叉树，上述的存储方式缺点在于浪费存储空间，对于 K 层的二叉树，用户保存的数组的容量（大小）至少为 2^K-1，对于规模较大的问题，这种方式不可取，可以参考第二种存储方式。这种存储方式的优点在于结点之间的关系简单，对于满二叉树和完全二叉树，这是一个不错的选择。

第二种存储方式采用指针来维持结点之间的关系，定义如下：

```
type
PBinTreeNode=^TBinTreeNode;
TBinTreeNode=record
  Data:Pointer;
  Left:PBinTreeNode;
  Right:PBinTreeNode;
  Parent:PBinTreeNode;
end;
```

寻找父结点、左儿子和右儿子的操作如下：

```
// 初始化二叉树（设为空树）
procedure BEmpty(var tree:PBinTreeNode);
begin
```

```
        tree=nil;
end;
//返回父结点
function BParent(node:PBinTreeNode):PBinTreeNode;
begin
        Result:=node^.Parent;
end;
//返回左儿子
function BLeft(node:PBinTreeNode):PBinTreeNode;
begin
        Result:=node^.Left;
end;
//返回右儿子
function BRight(node:PBinTreeNode):PBinTreeNode;
begin
        Result:=node^.Right;
end;
```

3. 二叉树的遍历

遍历是指循着某条搜索路线巡查某数据结构的结点，而且每个结点只能被访问一次。二叉树的遍历常用有三种遍历方式：（1）先序遍历：即先访问根结点，再先序遍历左子树，最后先序遍历右子树；（2）中序遍历：即先中序遍历左子树，再访问根结点，最后中序遍历右子树；（3）后序遍历：即先后序遍历左子树，再后序遍历右子树，最后访问根结点。

二叉树的遍历是递归定义的。

下面以中序遍历为例，以指针的方式作为二叉树存储结构，描述中序遍历算法：

```
procedure Mid_Traversing(node:PBinTreeNode);
begin
  if node=nil then exit;
  Mid_Traversing(BLeft(node));      //中序遍历左子树
  //Access node                     //访问根结点
  Mid_Traversing(BRight(node));     //中序遍历右子树
end;
//中序遍历以 root 为根的子树，调用过程如下：
//Mid_Traversing(root)
```

2.2.5 图

图是由点集 V 和边集 E 组成的二元组（V，E）。根据边的性质，图可以分为无向图和有向图。无向图中的边是没有方向的，即边（Vi,Vj）和（Vj,Vi）（i≠j）被认为是同一条边，而有向图中的边是有方向的，即边（Vi,Vj）和（Vj,Vi）（i≠j）被认为是两条不同的边。

图的存储方式

图的存储需要考虑如何刻画点集和边集的关系。常用的存储结构有：邻接矩阵，邻接表和边目录方式。

邻接矩阵是指使用二维矩阵 Aij 来表示图，如果存在边（Vi,Vj），则 Aij=1，否则，Aij=0。

邻接表是指使用图 G 的各个顶点及其相邻顶点的方式来表示图。

边目录方式是指通过列出所有边的的起点和终点的方式表示图。
例如：下面是一个图，这三种方式见下表。

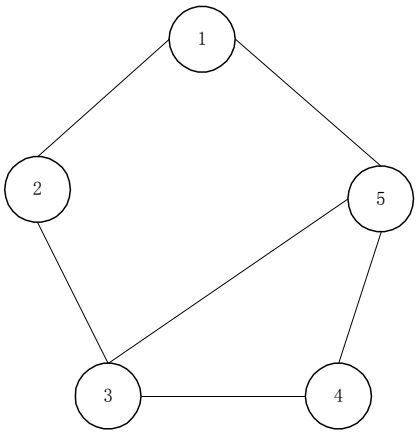


图 2.3 图的示例

邻接矩阵：

	1	*2*	*3*	*4*	*5*
1	0	1	0	0	1
2	1	0	1	0	0
3	0	1	0	1	1
4	0	0	1	0	0
5	0	0	1	0	0

邻接表：

顶点	*相邻顶点*		
1	2	5	
2	1	3	
3	2	4	5
4	3	5	
5	1	3	4

边目录：

边起点	1	2	3	3	4
边终点	2	3	4	5	5

对于一个顶点数为 n，边数为 e 的图，邻接矩阵需要使用 n*n 的二维数组作为存储空间，邻接表需要 n*(n-1)的二维数组作为存储空间，而边目录需要 2*e 的二维数组作为存储空间。因此，对于边比较稀疏的图，采用边目录的方式存储比较有利；而对于边比较稠密的图，采用邻接矩阵或者邻接表的方式储存比较合适。

图的遍历

在二叉树的遍历中，提到了遍历的概念，图的遍历也类似。所谓图遍历是指循着某条搜索路线巡查图的结点，而且每个结点只能被访问一次。图的遍历分为深度优先搜索（DFS）遍历和广度优先搜索遍历（BFS）两种。

深度优先搜索的基本思想是：“往前走，碰壁回头”。具体来说，首先从图 G 的一个顶点 V_0 出发，访问任意一个和 V_0 邻接的结点 V_1，再从 V_1 出发，访问和 V_1 相邻但又没有访问过的结点 V_2（往前走）。重复上述过程，直至某个结点的相邻结点都被访问过（碰壁），这时，沿着原来前进路线，逆向回退到含义未被访问的相邻结点的结点（回头），重复上述往前走的过程。

深度优先搜索是个递归定义的过程，算法性描述如下：

```
procedure DFS(v:TNode);
begin
  Visit (v);
  for each w，w与v相邻
      if w is not visited then DFS(w);
end;
```

深度优先搜索可借助“栈”的数据结构来实现（请参考前面“栈”一节），需要用到的栈操作有：置空栈：Empty(s)，判空：IsEmpty(s)，进栈：push(s,value)和出栈 pop(s)。与上面等价的深度优先搜索的算法性描述为：

```
procedure DFS(v);
begin
  Empty(s);
  push(s,v);
  while not IsEmpty(s) do
  begin
    v=pop(s);
    if v is not visited then
    begin
      Visit(v);
      for each w,w与v相邻 do push(s,w);
    end;
  end;
end;
```

图的深度搜索有许多应用，读者可以参考例题（第四章）第十节“移棋子”以及习题（第五章）第 3、5 和 7 题。

广度优先搜索是“按层来访问”的，具体来说，从图 G 的某一顶点 V_0 出发，访问与之相邻的顶点 $V_1,V_2,...,V_n$，然后再访问与 $V_1,V_2,...,V_n$ 相邻且未访问过的顶点，如此重复，直至所有顶点被访问。

广度优先搜索的实现需要借助“队列”的数据结构（请参考前面“队列”一节），需要用到的队列操作有：置空队列：Empty(q)，判空：IsEmpty(q)，进队列：push(q,value)和出队列 pop(q)。广度优先搜索的算法性描述如下：

```
procedure BFS(v);
begin
  Empty(q);
  push(q,v);
  while not IsEmpty(q) do
  begin
    v=pop(q);
    if v is not visited then
    begin
      Visit(v);
      for each w,w与v相邻 do push(q,w);
    end;
  end;
end;
```

对比广度优先搜索和深度优先搜索（采用“栈”数据结构描述），可以发现深度和广度优先搜索是很统一的，算法过程是大体一致，只是所用的数据结构不同而已。
广度搜索的应用，请参考例题（第四章）第七节“师生树”。

第三节 枚举算法

所谓枚举算法，是指从可能的集合中一一枚举各元素，用题目给定的检验条件判定哪些是无用的，哪些是有用的。能使命题成立者，即为问题的解。
采用枚举算法解题的基本思路如下：
（1）建立问题的数学模型，确定问题的可能解的集合（可能解的空间）。
（2）逐一枚举可能解集合中的元素，验证是否是问题的解。
使用类 Pascal 伪代码可以描述为：

```
for each s in S   //S 是问题所有可能解的集合
  if s is a solution then
    begin
      Write(s);
      exit the program;
    end;
```

例题：百鸡问题：有一个人有一百块钱，打算买一百只鸡。到市场一看，公鸡三块钱一只，母鸡两块钱一个，小鸡一块钱三只。现在，请你编一程序，帮他计划一下，怎么样买法，才能刚好用一百块钱买一百只鸡？
按照枚举算法的思路，首先应该构造可能解的集合：S={(x,y,z)|0≤x,y,z≤100}，其中三元组(x,y,z)表示买公鸡 x 只，母鸡 y 只和小鸡 z 只。因为一共需要买 100 只鸡，因此，买公鸡、母鸡和小鸡的数量都不会超过 100。然后确定验证解的条件：x+y+z=100 and 3x+2y+z/3=100。
下面是解这百鸡问题的程序：

```
program ex2_3_1;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var x,y,z:Integer;
begin
       //枚举可能解空间的元素
       for x:=0 to 100 do
       for y:=0 to 100 do
       for z:=0 to 100 do
       //验证可能解
       if (x+y+z=100) and (x*3+y*2+z div 3=100) and (z mod 3=0) then
       WriteLn(Format('(x,y,z)=(%3d,%3d,%3d)',[x,y,z]));
       ReadLn;
end.
```

程序输出结果为：

```
(x,y,z)=(  0,  40,  60)
(x,y,z)=(  5,  32,  63)
```

```
(x,y,z)=( 10, 24, 66)
(x,y,z)=( 15, 16, 69)
(x,y,z)=( 20,  8, 72)
(x,y,z)=( 25,  0, 75)
```

有 6 种可选的方案。

按照上面的程序，程序需要循环 100^3 次，即$|S|=100^3$。我们通过条件 x+y+z=100 来约束求解空间，缩小可能解的集合的规模，请看下面的程序：

```
program ex2_3_2;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var x,y,z:Integer;
begin
       //枚举可能解空间的元素
       for x:=0 to 100 do
       for y:=0 to 100-x do
       begin
              z:=100-x-y;
              //验证可能解
              if (x+y+z=100) and (x*3+y*2+z div 3=100)
                and (z mod 3=0) then
              WriteLn(Format('(x,y,z)=(%3d,%3d,%3d)',[x,y,z]));
       end;
       ReadLn;
end.
```

程序 ex2_3_2 的运行结果和程序 ex2_3_1 相同，但是循环次数为（100*101/2），是程序 ex2_3_1 循环次数的 1/200 左右。

从上面的对比可以看出，对于枚举算法，程序优化的主要考虑方向是：通过加强约束条件，缩小可能解的集合的规模。

第四节 回溯算法

我们在日常生活中经常会遇到一些要求最优解的问题，比如说从一个城市到另一个城市怎么走才最快，怎么走才最省钱。但是在处理一些最优解的问题方面，没有任何的理论也无法采用精确的数学公式来帮助我们找到最优解，我们只能求助于穷举搜索方法。在这里我们介绍一种系统化的穷举搜索技术，称为回溯技术。

所谓的回溯技术就是像人走迷宫一样，先选择一个前进方向尝试，一步步往前试探，在遇到死胡同不能再往前的时候就回退到上一个分叉点，选另一个方向尝试，而在前进和回撤的路上都设置一些标记，以便能正确返回，直到达到目标或者所有的可行方案都已经尝试完为止。在通常的情况下，我们使用递归方式来实现回溯技术，也就是在每一个分叉点进行递归尝试。在回溯时通常采用栈来记录回溯过程，使用栈可使穷举过程能回溯到所要位置，并继续在指定层次上往下穷举所有可能的解。回溯算法可以用伪码描述如下：

```
Proc Search(当前状态);
begin
  If  当前状态等于目标状态  then exit;
  for 对所有可能的新状态
    Search(新状态);
```

end;

回溯算法是一种十分常用的算法，象一些经典问题如八皇后问题、骑士周游问题、地图着色问题都可以采用回溯算法来解。

例题：求马的不同走法总数

问题描述：在一个 4*5 的棋盘上，马的起始位置坐标（纵，横）位置由键盘输入，求马能返回初始位置的所有不同走法的总数(马走过的位置不能重复，马走“日”字)。

算法分析：

由于棋盘的大小只有 4*5，所以只需使用回溯算法，搜索马能返回初始位置的所有不同走法，效率基本上能达到要求。

递归的回溯算法可描述为：

```
procedure search(now:position); {now 是当前位置}
  begin
    for 马从当前位置 now 出发走一步到位置 next 的每一种走法 do begin
      if next 在棋盘内 and next 位置没有走过 then
        if next=出发点 then 不同走法总数加 1
        else begin
          标记 next 已经走过了;
          search(next);
          取消位置 next 的标记;
        end;
    end;
  end;
```

下面讨论算法的具体实现。

棋盘用坐标表示，点 P(x,y)表示棋盘上任一个点，x,y 的范围是：1<=x<=4，1<=y<=5。

从 P(x,y)出发，下一步最多有 8 个位置，记为 P_1，P_2，……,P_8，若用 k 表示这 8 个方向，则 k=1,2,…,8。即马从 P 点出发，首先沿 k=1 的方向行进，当在此方向走完所有的不同走法后，就进行回溯，改变 k=2 方向继续行进……

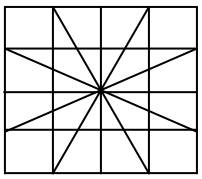


图 2.4　马的八种不同走法

各点坐标的计算。设 P 点坐标为(x,y)，则能到达点的坐标分别为 P_1(x+1,y-2)，P_2(x+2,y-1)，…，P_7(x-2,y-1)，P_8(x-1,y-2)。为简化坐标的计算，引入增量数组：

```
direction:array[1..8] of position=
   ((x:1;y:-2),(x:2;y:-1),(x:2;y:1),(x:1;y:2),
    (x:-1;y:2),(x:-2;y:1),(x:-2;y:-1),(x:-1;y:-2));
```

则按方向 k 能到达点的坐标是：

P_k(x+direction[k].x,y+direction[k].y)。

程序如下：

```
program ex2_4_1;

{$APPTYPE CONSOLE}

uses
  SysUtils;
```

```
type position=record  x,y:integer; end;
const direction:array[1..8] of position=
      ((x:1;y:-2),(x:2;y:-1),(x:2;y:1),(x:1;y:2),
   (x:-1;y:2),(x:-2;y:1),(x:-2;y:-1),(x:-1;y:-2));

var pass:array [1..4,1..5] of integer;
    start:position;
    total:integer;

procedure search(now:position); {now 是当前位置}
  var i:integer;
      next:position;
  begin
    for i:=1 to 8 do begin
      next.x:=now.x + direction[i].x;
      next.y:=now.y + direction[i].y;
      if (next.x>=1) and (next.x<=4) and (next.y>=1) and (next.y <= 5)
      and (pass[next.x,next.y]=0) then
        if (next.x=start.x) and (next.y=start.y) then inc(total)
        else begin
          pass[next.x,next.y]:=1;
          search(next);
          pass[next.x,next.y]:=0;
        end;
    end;
  end;



begin
       total:=0;
       fillchar(pass,sizeof(pass),0);
       write('Start position:');
       readln(start.x,start.y);
       search(start);
       writeln('Total=',total);
       readln;
end.
```

回溯算法的应用，请参考例题（第四章）第六节“骨牌矩阵”。

第五节 贪心算法

所谓贪心算法是指，在对问题求解时，总是作出在当前看来是最好的选择。也就是说，不从整体最优上加以考虑，它所做出的仅是在某种意义上的局部最优解。

贪心算法不是对所有问题都能得到整体最优解，但对范围相当广泛的许多问题它能产生整体最优解或者是整体最优解的近似解。

采用贪心算法的基本思路如下：

（1）建立数学模型来描述问题。

（2）把求解的问题分成若干个子问题。

（3）对每一子问题求解，得到子问题的局部最优解。

（4）把子问题的解局部最优解合成原求解问题的一个解。

例题：

无向图的最小生成树问题。

设 G=[V,E]是一个无向图，如果 T=[V,E]是由 G 的全部顶点及其一部分边组成的子图，T 是树，则称 T 是 G 的一个生成树。记 L（T）为 T 的长度，即树 T 的各边之和。求 G 的所有生成树中 L（T）最小的生成树。

如图所示：

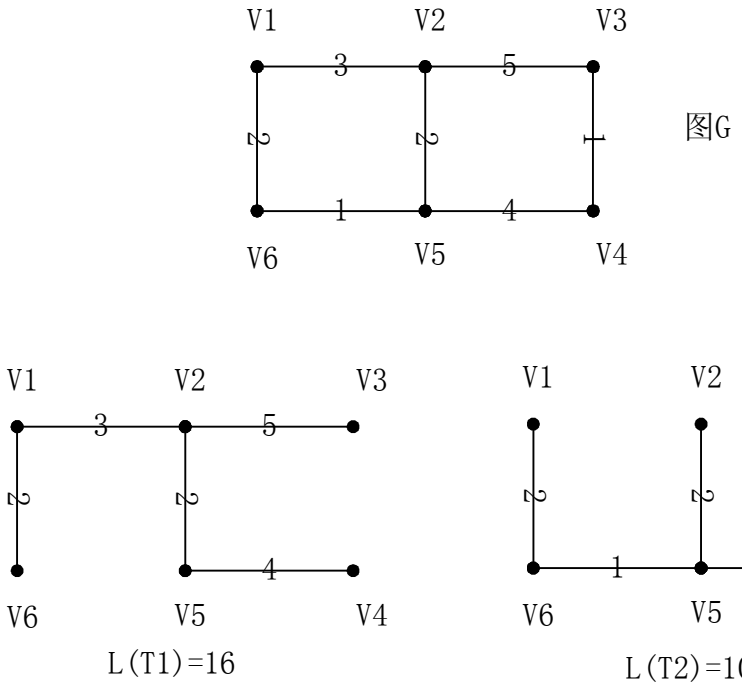


图 2.5 无向图 G 及其生成树

下面的两棵树都是图 G 的生成树，其中 T2 是所有图 G 的最小的生成树。

最小生成树的算法思路是：由于 n 个顶点的图，其最小生成树共有 n-1 条边，因此寻找最小生成树的问题就是选这 n-1 条边的过程，我们可以把这个过程分解为 n-1 次的选择，每次选择都选一条边。在每次选边的时候，我们采用贪心的原则：选择一条权值最小而未被选过，且和已选定的边不会构成圈的边。

最小生成树的算法如下：

T=空;

for i:=1 to n-1 do

begin

在图 G 中选取权值最小，不在 T 中，而且与 T 中的边不构成圈的边 e_i;

把 e_i 加入 T 中;

end;

T 就是图 G 的最小生成树。

最小生成树程序实现如下：

```
program ex2_5_1;

{$APPTYPE CONSOLE}
```

```
uses
  SysUtils;

var     F:Text;
        //图 G 的点数 N 和边数 M
        N,M,i,j:Integer;
        //对已选择的边 ei,Selected[i]为 1,否则为 0
        Selected:Array [1..100] of Integer;
        //边的起点和终点
        E:Array [1..100,1..2] of Integer;
        //边的权值
        Value:Array [1..100] of Integer;
        //若 Vi 是生成中的结点,T[i]=1,否则为 0
        T:Array [1..100] of Integer;
        //分别是当前选边的权值、选边的编号和树的长度
        Min,MinE,ValueT:Integer;
begin
        //读入图 G，图 G 采用边目录表示法。
        Assign(F,'ex2_4_1.in');
        Reset(F);
        ReadLn(F,N,M);
        for i:=1 to M do ReadLn(F,E[i,1],E[i,2],Value[i]);
        //初始化
        fillchar(Selected,sizeof(Selected),0);
        fillchar(T,sizeof(T),0);
        ValueT:=0;
        //n-1 次选边过程
        for i:=1 to N-1 do
        begin
                Min:=Maxint;
                MinE:=0;
                for j:=1 to m do
                //未被选中
                if Selected[j]=0 then
                //不构成圈
                if ((T[E[j,1]]=0) xor (T[E[j,2]]=0)) or (i=1) then
                //权值最小
                if Value[j]<Min then
                begin
                        Min:=Value[j];
                        MinE:=j;
                end;
                //做选中的标记
                Selected[MinE]:=1;
                T[E[MinE,1]]:=1;
                T[E[MinE,2]]:=1;
                ValueT:=ValueT+Min;
        end;
        WriteLn('T:','':10,'Length=',ValueT);
        for i:=1 to m do
        if Selected[i]=1 then
        begin
                WriteLn('(',E[i,1],',',E[i,2],')');
        end;
        Close(F);
        ReadLn;
end.
```

测试数据如下：

```
6 7
1 2 3
1 6 2
2 3 5
2 5 2
3 4 1
4 5 4
5 6 1
```

程序运行结果如下：

```
T:        Length=10
(1,6)
(2,5)
(3,4)
(4,5)
(5,6)
```

贪心算法的应用请参考第五章习题的第 7 题“计算机网络连接”。

第六节 分治算法

所谓分治算法，是指将一个规模较大的问题分解为若干个规模较小的部分（这些小问题的难度应该比原问题小），求出各部分的解，然后再把各部分的解组合成整个问题的解。

采用分治算法解题的基本思路如下：

（1）对求解建立数学模型和问题规模描述。

（2）建立把一个规模较大的问题划分为规模较小问题的途径。

（3）定义可以立即解决（规模最小）的问题的解决方法。

（4）建立把若干个小问题的解合成大问题的方法。

其中，算法中最难的地方是如何分（第 2 步）和如何合（第 4 步），这两部分应该统一进行考虑，与解题经验有密切关系。

例题：

求正整数集合（$a_1,a_2,...,a_n$）的最大值和最小值。

采用分治的方法解决

建立数学模型和问题规模的描述：题目本身有很强的数学背景，数学模型应该是该问题的一般数学解释。我们可以定义问题（f，t）表示求集合（$a_f,a_{f+1},...,a_t$）中的最大值和最小值。我们需要解决的问题是（1，n）

当需要求解问题（f，t）（共 t-f+1 个元素），我们可以把这个集合（$a_f,a_{f+1},...,a_t$）分成两半，即设 m=f+(f-t)/2，集合分为（$a_f,...,a_m$）和（$a_{m+1},...,a_t$）两个集合，这两个集合中只含有(f-t)/2 或者(f-t)/2+1 个元素。这样，就可以把问题（f，t）划分成两个规模较小的问题（f，m）和（m+1，t）。

显然，当集合中只有一个元素时，问题立刻有解，集合的最大值和最小值都是集合中唯一的元素。

建立把若干个小问题的解合成大问题的方法：显然，问题（f，m）的最大值和问题（m+1，t）的最大值中的大者就是问题（f，t）的最大值；问题（f，m）的最小值和问题（m+1，t）的最小值中的小者就是问题（f，t）的最小值。

实现程序如下：

```
program ex2_6_1;

{$APPTYPE CONSOLE}

uses
  SysUtils,
  Math;

var a:array [1..10000] of Integer;

procedure MaxMin(f,t:Integer;var rMax,rMin:Integer);
var    m:Integer;
       Max1,Max2,Min1,Min2:Integer;
begin
       if f=t then
       begin
              rMax:=a[f];
              rMin:=a[t];
       end
       else
       begin
              m:=(t-f) div 2 + f;
              MaxMin(f,m,Max1,Min1);
              MaxMin(m+1,t,Max2,Min2);
              rMax:=Max(Max1,Max2);
              rMin:=Min(Min1,Min2);
       end;
end;

var Fin:Text;
var i,n,rMax,rMin:Integer;
begin
       Assign(Fin,'ex2_5_1.txt');
       Reset(Fin);
       ReadLn(Fin,n);
       For I:=1 to n do Read(Fin,a[I]);
       Close(Fin);
       MaxMin(1,n,rMax,rMin);
       WriteLn(Format('Max=%d,Min=%d',[rMax,rMin]));
       Readln;
end.
```

其中测试用例，ex_2_6_1.txt 如下：

```
7
2 34 32 19 1 39 4
```

程序运行结果如下：

```
Max=39,Min=1
```

分治算法的应用请参考例题（第四章）第三节“划分凸多边形”

第七节 数值计算

数值计算是计算机最早的应用之一，早期的计算机其主要使命就是进行数值计算。

在这一节里，主要讨论数值计算中一些最常用最基础的部分。

2.7.1 高精度计算

在“Delphi 的基本类型“一节中介绍过，Delphi 可以精确计算的范围是有限的，精确度最高的整型是 Int64，值域是[-2^{63}－2^{63}-1]。如果需要精度更高的计算，就需要编程实现。在这一部分中，主要介绍正整数的高精度计算（加，减，乘和除），至于整数，有理数乃至实数的高精度运算，这一部分所介绍的方法和技巧还是很有参考价值的。

1．正整数高精度加法

加法和减法都是最基本的运算，乘除法运算是建立在加减法之上的。

加法算法如下：

步骤	说明	举例（S1+S2） 表示参与运算或被改变的数字	
		S1=82	S2=74
1	对位。在被加数和加数前面适当补 0，使他们的包含相同的位数。	82	74
2	前面再补一个 0，确定和的最多位数。	082	074
3	从低位开始，对应位相加，结果写入被加数中。如果有进位，直接给被加数前一位加 1。	086 156 156	074 074 074
4	删除和前面多余的 0	156	074

2．正整数高精度减法

减法和加法的最大区别在于：减法是从高位开始相减，而加法是从低位开始相加。

减法算法如下：

步骤	说明	举例（S1+S2） 表示参与运算或被改变的数字	
		S1=82	S2=74
1	对位。在被减数和减数前面适当补 0，使他们的包含相同的位数。	82	74
2	从高位开始，对应位相减，结果写入被减数中。如果需要借位，直接给被减数前一位减 1。	12 08	74 74
3	删除和前面多余的 0	8	74

3．正整数高精度乘法

乘法的主要思想是把乘法转化为加法进行运算。请先看下面的等式：

12345*4=12345+12345+12345+12345

12345*20=123450*2

12345*24=12345*20+12345*4

等式（1）说明，多位数乘一位数，可以直接使用加法完成。

等式（2）说明，多位数乘形如 $d*10^n$ 的数，可以转换成多位数乘一位数来处理。

等式（3）说明，多位数乘多位数，可以转换为若干个“多位数乘形如 $d*10^n$ 的数”之和。

因此，多位数乘多位数最终可以全部用加法来实现。

算法形式化描述如下：

```
function multiply(s1,s2:string):string;
var i:Integer;C:Char;
```

```
begin
        Result:='0';
//多位数乘多位数，可以转换为若干个“多位数乘形如 d*10^n 的数”之和
        for i:=Length(s2) downto 1 do
        begin
                //多位数乘一位数，可以直接相加
                for C:='1' to S2[i] do
                Result:=addition(Result,S1);
                //多位数乘形如 d*10^n 的数转换成多位数乘一位数来处理
                S1:=S1+'0';
        end;
        Result:=Clear(Result);
end;
```

例如 12*12，算法执行过程如下（多位数乘一位数看成是一步完成）：

执行步数	S1	S2	Result
0	12	12	0
1	12	12	24
2	120	12	144

4．正整数高精度除法

高精度除法是基于精度减法的，主要的思想是“试商”，模仿笔算除法的方法。

算法形式化描述为：

```
function division(s1,s2:string):string;
begin
        //Result 中存放的是商
        Result:='';
        //S 中存放的是余数，S1 是被除数，S2 是除数
        S:='';
        //逐位试商
        for i:=1 to Length(S1) do
        begin
                S:=S+S1[i];
                //先商 0
                Result:=Result+'0';
                //然后从 1 开始不断往上试
                While Bigger(S,S2) do
                begin
                        Inc(Result[Length(Result)]);
                        S:=Subtraction(S,S2);
                end;
        end;
        //删除多余的 0
        Result:=Clear(Result);
end;
```

例如，144/12 算法执行如下：

执行步数	S1	S2	S	Result	说明
0	144	12			开始状态
1	144	12	1		S 从 S1 中取得第 1 位
2	144	12	1	0	试商 0，然后由于 1<12，试商结束
3	144	12	14	0	S 从 S1 中取得第 2 位，与上面剩下的余数组合
4	144	12	14	00	试商 0
5	144	12	2	01	试商 1，然后 S=S-S2，此时 S

6	144	12	24	01	已经小于 S2 了，试商结束 S 从 S1 中取得第 3 位，与上面剩下的余数组合
7	144	12	24	010	试商 0
8	144	12	12	011	试商 1 然后 S=S-S2
9	144	12	0	012	试商 2，然后 S=S-S2，此时 S 已经小于 S2 了，试商结束
10	144	12	0	12	清除多余的 0

5. 实现正整数高精度运算的样例程序如下：

```
program ex2_7_1;

{$APPTYPE CONSOLE}

uses
  SysUtils,Math;
//清除整数前面多余的 0
function clear(s:string):string;
begin
       if s='' then s:='0';
       while (length(s)>0) and (s[1]='0') do delete(s,1,1);
       if s='' then s:='0';
       clear:=s;
end;
//比较两个数的大小，如果 S1>=S2，结果返回为真。
function bigger(s1,s2:string):boolean;
begin
       bigger:=true;
       if length(s1)>length(s2) then exit;
       if (length(s1)=length(s2)) and (s1>=s2) then exit;
       bigger:=false;
end;
//加法
function addition(s1,s2:string):string;
var i:integer;
begin
       //对位
       while Length(S1)<Length(S2) do S1:='0'+S1;
       while Length(S2)<Length(S1) do S2:='0'+S2;
       //考虑到首位可能进位，在首位中增加一个 0
       S1:='0'+S1;S2:='0'+S2;
       for i:=Length(S1) downto 1 do
       begin
       //逐位相加
              Inc(S1[i],Ord(S2[i])-Ord('0'));
       //处理进位
              if S1[i]>'9' then
              begin
                     Dec(S1[i],10);
                     Inc(S1[i-1]);
              end;
       end;
       Result:=Clear(S1);
```

```
end;
//减法
//这里假设 S1>=S2，如果不满足这个条件，函数返回结果不正确
function Subtraction(s1,s2:string):string;
var    i:integer;
begin
       //对位
       while Length(S1)<Length(S2) do S1:='0'+S1;
       while Length(S2)<Length(S1) do S2:='0'+S2;
       //逐位相减
       For i:=Length(S1) downto 1 do
       If S1[i]>=S2[i] then Dec(S1[i],Ord(S2[i])-Ord('0'))
       else       //有借位的情况
       begin
              Inc(S1[i],10);
              Dec(S1[i-1]);
              Dec(S1[i],Ord(S2[i])-Ord('0'))
       end;
       Result:=Clear(S1);
end;
//乘法
function multiply(s1,s2:string):string;
var i:Integer;C:Char;
begin
       Result:='0';
       for i:=Length(s2) downto 1 do
       begin
              for C:='1' to S2[i] do
              Result:=addition(Result,S1);
              S1:=S1+'0';
       end;
       Result:=Clear(Result);
end;
//除法
//这里假设除数不为 0，如果除数为 0，函数进入死循环。
function division(s1,s2:string):string;
var i:integer;
   s:String;
begin
       Result:='';
       S:='';
       for i:=1 to Length(S1) do
       begin
              S:=S+S1[i];
              Result:=Result+'0';
              While Bigger(S,S2) do
              begin
                     Inc(Result[Length(Result)]);
                     S:=Subtraction(S,S2);
              end;
       end;
       Result:=Clear(Result);
end;

begin
       WriteLn(Addition('999','1'));
       WriteLn(Subtraction('890','99'));
       WriteLn(Multiply('99','99'));
```

```
        WriteLn(Division('9899','99'));
        ReadLn;
end.
```

程序运行结果：

```
1000
791
9801
99
```

特别说明，上面的程序中的减法和除法没有判断是否够减和除数为 0，如果对于一般问题，调用减法和除法之前应该做检查，如下例所示：

```
//判断够减（结果不出现负数）
if Bigger(s1,s2) then Result:=Subtraction(s1,s2);
//判断除数不为 0
if Clear(s2)<>'0' then Result:=Division(s1,s2);
```

高精度的应用请参考例题（第四章）第九节“正整数竖式除法 高精度计算”。

2.7.2 求解线性方程组

解线性方程组是一类常用的基础问题。例如，在计算几何中，需要求直线的交点，在列出直线方程后，就直接转化为解线性方程组问题了；又如线性规划问题，解线性方程组也是其中的一个基础子问题。

解线性方程组的方法很多，例如 Gauss 消元法，LU 法，求逆阵法，Gauss-Seidel(迭代)法等等，其中 Gauss 消元法需要的数学背景知识最少，而且比较容易实现，下面主要介绍使用 Gauss 消元法求解线性方程组问题。

下面简单介绍一下关于线性方程组的一些背景知识：

$a_1x_1+...+a_nx_n=b$ **线性方程式**(Linearequation)，其中 $a_1,...,a_n$ 为系数，$x_1,...,x_n$ 为未知数或变量(unknowns)，b 为常数。

n 元 m 次联立方程式：

$$\begin{cases} a_{11}x_1+\cdots+a_{1n}x_n=b_1 \\ a_{21}x_1+\cdots+a_{2n}x_n=b_2 \\ \qquad\vdots \\ a_{m1}x_1+\cdots+a_{2m}x_n=b_m \end{cases}$$

上式有 m 个方程式及 n 个未知元所形成的**线性方程組**。

可以表为

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}。$$

其中称 $\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$ 为**系数矩阵**，$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ 为**未知向量**，$\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$ 为**常数向量**。

上面的矩阵也可以表示为增广的形式：

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} & b_m \end{array}\right] \cdots\cdots$$

Gauss 消元法主要是基于增广矩阵的变换。算法如下：

（1） 设所有行都为“未处理行”，然后从第 1 列到第 n 列依次扫描矩阵 a。

（2） 当扫描到第 k 列时，在矩阵 a 所有未处理过的行中找出第 k 列中的最大值，和最大值所在行 p。

（3） 如果最大值为 0，说明方程组无解或者有无穷多组解。否则，继续（4）。

（4） 设置行 p 为已处理行，同时对其他的每一行，找到一个适当的数，然后把行 p 乘以这个适当的数加到该行中，使得通过上述变换后，该行第 k 列的元素为 0。显然，对第 i 行，这个适当的数为-a[i,k]/a[p,k]。

（5） 经过上面的从第 1 列到第 n 列的扫描后，矩阵 a 中每行仅又一个非 0 值，记第 i 行的非 0 值为 a[i,j]，则有 x[j]:=b[i]/a[i,j]。

算法举例：

求解方程：

$$\begin{cases} x_1 + 2x_2 + x_3 = 3 \\ 3x_1 - x_2 - 3x_3 = -1 \\ 2x_1 + 3x_2 + x_3 = 4 \end{cases}$$

增广矩阵表示如下：

```
 1.00    2.00    1.00 |    3.00
 3.00   -1.00   -3.00 |   -1.00
 2.00    3.00    1.00 |    4.00
```

在第 1 列中，最大值为 a[2,1]，把第 2 行乘以-1/3 后加到第一行，同时把第 2 行乘以-2/3 后加到第三行

```
 0.00    2.33    2.00 |    3.33
 3.00   -1.00   -3.00 |   -1.00
 0.00    3.67    3.00 |    4.67
```

在第 2 列中，最大值为 a[3,2]，把第 3 行乘以-2.33/3.67 后加到第一行，同时把第 3 行乘以 1/3.67 后加到第二行

```
-0.00   -0.00    0.09 |    0.36
 3.00    0.00   -2.18 |    0.27
```

```
  0.00    3.67    3.00 |    4.67
```

在第 3 列中，最大值为 a[1,3]（因为此时未处理行只有第一行），类似上面的做法处理。

```
 -0.00   -0.00    0.09 |    0.36
  3.00   -0.00    0.00 |    9.00
  0.00    3.67    0.00 |   -7.33
```

最后的结果为：

```
x[1]= 3.000000    x[2]=-2.000000    x[3]=4.000000
```

实现 Gauss 消元法样例程序如下：

```
program ex2_7_2;
//定义一维向量和二维向量
const
       maxn=100;
type
       tmatrix=array[1..maxn,1..maxn] of real;
       tvector=array[1..maxn] of real;

//用 Gauss 消元法求解线性方程组 ax=b
//Input: 数组 a[1..n,1..n]，b[1..n]
//Output: 解 x[1..n]
function GaussElimination
     (n:integer;a:tmatrix;b:tvector;var x:tvector):Boolean;
var q:array [1..maxn] of integer;
    i,j,k,p:Integer;
    max,l:real;
begin
    fillchar(q,sizeof(q),0);
    for k:=1 to n do
    begin
        //选出第 k 列中，绝对值最大值对应的行（p）
        p:=0;max:=0;
        for i:=1 to n do
        if (q[i]=0) and (max+1e-10<abs(a[i,k])) then
        begin
            max:=abs(a[i,k]);
            p:=i;
        end;
        //如果绝对值最大为 0，说明方程组无解或者有无穷多组解。
        if p=0 then
        begin
            Result:=False;
            exit;
        end
        //做上处理过的标记
        else q[p]:=1;
        //把 p 行乘上一个适当的系数后加到其他行上，使第 k 列中只有一个非 0 值
        for i:=1 to n do if i<>p then
        begin
            l:=a[i,k]/a[p,k];
            for j:=1 to n do a[i,j]:=a[i,j]-l*a[p,j];
            b[i]:=b[i]-l*b[p];
        end;
    end;
    //求解 x
    for i:=1 to n do for j:=1 to n do
```

```
            if abs(a[i,j])>1e-10 then x[j]:=b[i]/a[i,j];
            Result:=True;
        end;
```

数值计算的范围很广，有不少相关的著作和资料详细讨论这方面的内容。如果您对此有兴趣，想做进一步的了解，您可以参考这方面相关的书籍。

第八节 计算几何

计算几何是计算机科学中新的分支，它的形成和发展与计算机图形学、超大规模集成电路设计、计算机辅助设计和机器人等学科的发展都有密切的联系。

2.8.1 线段问题

在本节中将结合两条线段的交点问题，介绍处理线段的一些常用方法。

首先讨论的是关于平面上的线段的若干性质。

已知 A（x_1，y_1）和 B（x_2，y_2）是平面上任意两点，则线段 AB 上的任意一点 P（x，y），存在实数 a($0 \le a \le 1$)，满足：$x=ax_1+(1-a)x_2$，$y=ay_1+(1-a)y_2$。

A 和 B 两点可以看成是以 O（0，0）点为起点的向量 $\vec{A}$ 和 $\vec{B}$，$\vec{A}$ 和 $\vec{B}$ 的向量的积可以定义为：$\vec{A}\times\vec{B}=\begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix}=(x_1y_2-x_2y_1)=-\vec{B}\times\vec{A}$

如图所示：

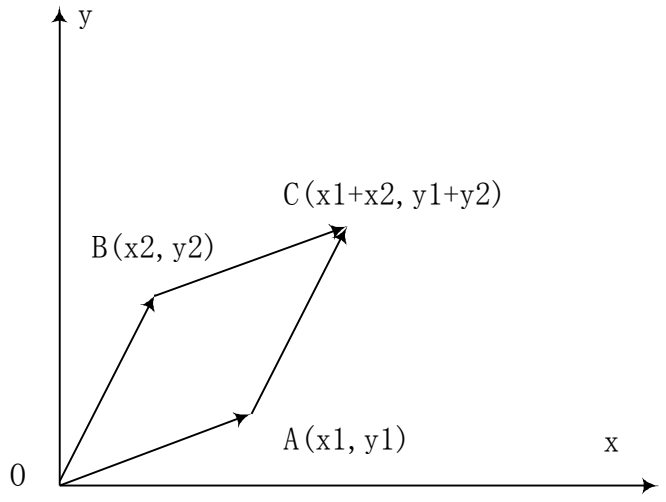


图 2。6 向量 $\vec{A}$ 和 $\vec{B}$

$\vec{A}\times\vec{B}$ 的绝对值表示平行四边形 OACB 的面积。若 $\vec{A}\times\vec{B}$ 为正，则 $\vec{A}$ 是反时针旋转到 $\vec{B}$；若 $\vec{A}\times\vec{B}$ 为负，则 $\vec{A}$ 是顺时针旋转到 $\vec{B}$；若 $\vec{A}\times\vec{B}$ 为 0，则 $\vec{A}$ 和 $\vec{B}$ 是同向或者反向共线。

上面的结论可以推广到以某点 A 作为起点向量的 $\overrightarrow{AB}$ 和 $\overrightarrow{AC}$。这时，$\overrightarrow{AB}\times\overrightarrow{AC}$ = （x_B-x_A）（y_C-y_A）-（x_C-x_A）（y_B-y_A）。

下面就正式转入如何判断两条线段 AB 和 CD 是否相交和求线段的交点。

假设 AB 和 CD 交于 P 点，那么下面的方程组：

$x=d_1x_A+(1-d_1)x_B$

$y=d_1y_A+(1-d_1)y_B$

$x=d_2x_C+(1-d_2)x_D$

$y=d_2y_C+(1-d_2)y_D$

如果线段相交当且仅当方程组有解，而且 $0\le d_1,d_2\le 1$。接下来的工作就是解上面的四元齐次方程组了（解方程组的算法可以参考上节“求解线性方程组”部分，当然也可以手工先求解方程组）。实现上面算法的程序如下：

```
program ex2_8_1;
type TPoint=record x,y:real end;
//by rei
//判断线段[s1,e1]和[s2,e2]是否相交
//Input: 线段[s1,e1]和[s2,e2]
//Output: Intersect = false 线段[s1,e1]和[s2,e2]不相交
//               true  线段[s1,e1]和[s2,e2]相交，交点用全局参数 d1,d2 给出
//                       交点坐标(x,y)的计算: x=s1.x+d1*(s2.x-s1.x)
//                                            y=s1.y+d1*(s2.y-s1.y)
function Intersect(s1,e1,s2,e2:TPoint):boolean;
var a1,a2,b1,b2,c1,c2:real;
begin
  Intersect:=false;
  a1:=e1.x-s1.x;
  a2:=e1.y-s1.y;
  b1:=s2.x-e2.x;
  b2:=s2.y-e2.y;
  c1:=s2.x-s1.x;
  c2:=s2.y-s1.y;
  if Same(a1*b2,a2*b1) then exit;
  d1:=(c1*b2-c2*b1)/(a1*b2-a2*b1);
  d2:=(c1*a2-c2*a1)/(b1*a2-b2*a1);
  Intersect:=true;
end;
```

2.8.2 凸包问题

设平面上有一点集 S，存在一最小的凸多边形 P，使得 S 中的每一点或在 P 的边界上或是 P 的内点。这个多边形 P 便称为 S 的凸包。

求凸包是一个有趣的问题，而且许多计算几何的问题都是从求凸包开始，所以它是计算几何的基础。

下面介绍 Grabam 扫描法求解凸包问题。

Graham 扫描法是从凸包的一个角点开始，通常选取 $P_i(x_i,y_i)(i=1,2,...,n)$中 yi 达到最小的点。不失一般性，令这一点为 $P_1(x_1,y_1)$，而且使得闭 n 边形 $P_1P_2...P_nP_1$ 满足 P_1P_i 的线段和 x 轴正向的夹角是一单调递增序列，上面的要求可以通过对倾斜角的排序来实现。

然后依次多边形中的顶点，删去不必要的顶点，使得最后留下来的是所求的凸包。
什么是不必要的点呢？如下图所示，

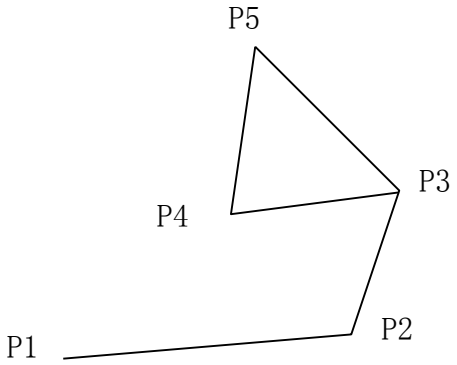


图 2。7 凸包问题中的不必要的点

P4 点是一个不必要的点，因为 $\overrightarrow{P_3P_4}$ 到 $\overrightarrow{P_4P_5}$ 是顺时针的，而其他都是逆时针的。

Graham 扫描法算法描述如下：
对点集进行重新排序，满足，P_1 的 y 座标最小，而且 PiP1 的对 x 轴正向的夹角是单调递增的。
把前 3 个点压入堆栈中。

对点 Pi(i>3)依次检查。如果当前栈中出现了 $\overrightarrow{P_{stack(k)}P_{stack(k+1)}}$ 到 $\overrightarrow{P_{stack(k+1)}P_i}$ 是顺时针的，把点 $P_{stack(k+1)}$从栈中弹出，直至检查完整个栈。其中 $P_{stack(k)}$表示存放在栈中的第 k 个点。
把 Pi 压入栈中，重复（3），直至所有点都检查完。
最后留在栈中的点就是凸包的顶点。
实现 Graham 扫描法的程序如下：

```
program ex2_8_2;
// by splutter
// 扫描法求凸包
// Input: List=点集数组 Len=点的总数
// Output: List=以逆时针方向给出的凸多边形顶点数组 Len=顶点数目

{$apptype console}

uses
  sysutils;

const
  maxn=1000;

type
  tpoint=record
    x,y:integer
  end;
  tarrpoint=array[1..maxn] of tpoint;

var
  top:integer;
  stack:array[1..maxn] of integer;
```

```
            //求线段内积，从而可以判断顺时针或者逆时针
            function multi(var p1,p2,p0:tpoint):integer;
            begin
              multi:=(p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
            end;
            //交换两个点
            procedure swap(var a,b:TPoint);
            var
              t:tpoint;
            begin
              t:=a; a:=b; b:=t
            end;

            procedure scan(var list:tarrpoint; var len:integer);
              //比较两个点的位置关系
              function comp(var p1,p2:tpoint):boolean;
              var
                t:integer;
              begin
                t:=multi(p1,p2,list[1]);
                comp:=(t>0) or ((t=0) and (sqr(p1.x-list[1].x)+sqr(p1.y-list[1].y)
<sqr(p2.x-list[1].x) +sqr(p2.y-list[1].y)));
              end;
              //排序（快速排序）
              procedure sort(p,r:integer);
              var
                i,j:integer;
                x:tpoint;
              begin
                if r-p+1<5 then
                begin
                  for j:=p+1 to r do
                  begin
                    i:=j;
                    while (i>1) and (comp(list[i],list[i-1])) do
                    begin
                      swap(list[i],list[i-1]);
                      dec(i);
                    end;
                  end;
                end
                else begin
                  x:=list[p+random(r-p+1)];
                  i:=p;j:=r;
                  repeat
                    while comp(list[i],x) do inc(i);
                    while comp(x,list[j]) do dec(j);
                    if i<j then swap(list[i],list[j]);
                  until i>=j;
                  sort(p,j);
                  sort(j+1,r);
                end;
              end;

            var
              i:integer;
              t:tarrpoint;
            begin
```

```
  //(1)对点集重新排序
  for i:=1 to len do
  begin
    if (list[i].y<list[1].y) or ((list[i].y=list[1].y)
       and (list[i].x<list[1].x))
     then swap(List[1],List[i]);
  end;
  sort(2,len);
  //(2)把前 3 个点压入堆栈中
  for i:=1 to 3 do stack[i]:=i;
  top:=3;
  //(3)对点 Pi(i>3)依次检查。
  for i:=4 to len do
  begin
    //弹出不必要的点
    while multi(list[i],list[stack[top]],list[stack[top-1]])>0
       do dec(top);
    inc(top);
    //(4)把 Pi 压入栈中
    stack[top]:=i;
  end;
  //(5)最后留在栈中的点就是凸包的顶点
  t:=list;
  for i:=1 to top do list[i]:=t[stack[i]];
  len:=top;
end;

begin
end.
```

计算几何的应用请参考第五章习题第 10 题“建筑物“。

第九节 模拟题解法

模拟题是从实际工程应用中提取出来的一些核心问题，或者本身就是某个工程的简化模型。解答模拟题特别需要有良好的理解能力、分析能力和规划能力。模拟题的算法一般都不太复杂，关键是所有条件都不能遗漏并要把条件分析清楚，求解模拟题一般都比较繁，编程前一定要预先规划好各个模块的功能和结构，以免具体编程时注意了局部而遗漏了某些重要的方面。解答模拟题通常的步骤可以是：

1. 认真仔细地读懂题目。模拟题的描述通常都比较详细，篇幅一般都比较长，你应该边阅读边将有关的条件一条条地记录下来，阅读完后要反复核对，绝对不能有错漏；
2. 建立各个条件之间的关系，最好用一些简明的表格列出，例如，可以用下表建立各项条件之间的关系：

	条件 1	条件 2	条件 3 •••	条件 N
条件 1		关系 1	关系 2	
条件 2				

条件 3　　关系 2　　关系 3

-
-
-

条件 N　　关系 4

3. 认真分析这些关系，并建立这些关系的数学模型；
4. 规划各个模块的结构，用相应的语言采用逐步求精的方法描述具体的算法；
5. 用你所熟悉的程序设计语言编写程序，调试并运行；
6. 检查题目给出的样例能否通过。ACM/ICPC 这类竞赛题目中都会给出输入输出样例，让你检查程序的输入输出格式是否正确，但这些样例往往会比竞赛时评判所用的测试数据简单，所以你一定不能满足通过这些样例，还要自拟更复杂、更全面的测试数据来检查程序的正确性。经过反复的调试、检查，才算完成该题。

有关模拟题的例子可参考第四章第二节的模式识别的“中心”问题。

第三章 寻找最优解的算法

根据问题对解的性质要求的不同，我们可以把问题的求解分成两类：求任意解和求最优解。从计算复杂度的角度看，两类问题的性质是不同的：求任意解的问题可以属于 P 或 NP 问题，而求最优解的问题通常不属于 NP 问题，因为最优解往往不是确定性图灵机能在多项式时间内验证的；举例来说，图论中著名的 Hamilton 圈问题是 NP 问题，而加权图的最短 Hamilton 圈问题不是 NP 问题，因为我们无法在多项式时间内验证一条 Hamilton 圈是否所有 Hamilton 圈中最短的。

一般来说，寻找最优解的算法都具有指数时间的复杂度，因此算法的优化显得十分重要；但是有一类特殊的最优化问题，我们通常可以找到具有多项式时间的复杂度的算法，这种方法就是我们下面要介绍的动态规划。

第一节 动态规划

动态规划是研究一类最优化问题的方法，在经济、工程技术、企业管理、工农业生产及军事等领域中都有广泛的应用。近年来，在 ACM/ICPC 中，使用动态规划（或部分应用动态规划思维）求解的题不仅常见，而且形式也多种多样。而在与此相近的各类信息学竞赛中，应用动态规划解题已经成为一种趋势，这和动态规划的优势不无关系。

3.1.1 动态规划的常用名词

在学习动态规划之前，先得对下面的名词有所了解。本书将标准名词作了一些简化，便于大家更好的理解。

(1) 状态(state)

对于一个问题，所有可能到达的情况（包括初始情况和目标情况）都称为这个问题的一个状态。

(2) 状态变量(s_k)

对每个状态 k 关联一个状态变量 s_k，它的值表示状态 k 所对应的问题的当前解值。

(3) 决策(decision)

决策是一种选择，对于每一个状态而言，你都可以选择某一种路线或方法，从而到达下一个状态。

(4) 决策变量(d_k)

在状态 k 下的决策变量 d_k 的值表示对状态 k 当前所做出的决策。

(5) 策略

策略是一个决策的集合，在我们解决问题的时候，我们将一系列决策记录下来，就是一个策略，其中满足某些最优条件的策略称之为最优策略。

(6) 状态转移函数(t)

从一个状态到另一个状态，可以依据一定的规则来前进。我们用一个函数 t 来描述这样的规则，它将状态 i 和决策变量 d_i 映射到另一个状态 j，记为 $t(i,d_i)=j$。

(7) 状态转移方程(f)

状态转移方程 f 描述了状态变量之间的数学关系。一般来说，与最优化问题相应，状态转移方程表示 s_i 的值最优化的条件，或者说是状态 i 所对应问题的最优解值的计算公式，用代数式表示就是：

$s_i=f(\{(s_j,d_j)|i=t(j,d_j)$，对决策变量 d_j 所有可行的取值$\})$

3.1.2 最优化原理

1951 年美国数学家 R. Bellman 等人，根据一类多阶段问题的特点，把多阶段决策问题变换为一系列互相联系的单阶段问题，然后逐个加以解决。一些静态模型，只要人为地引进“时间”因素，分成时段，就可以转化成多阶段的动态模型，用动态规划方法去处理。与此同时，他提出了解决这类问题的“**最优化原理**”(Principle of optimality)：

“一个过程的最优决策具有这样的性质：即无论其初始状态和初始决策如何，其今后诸策略对以第一个决策所形成的状态作为初始状态的过程而言，必须构成最优策略”。简言之，一个最优策略的子策略，对于它的初态和终态而言也必是最优的。

这个“最优化原理”如果用数学化一点的语言来描述的话，就是：假设为了解决某一优化问题，需要依次作出 n 个决策 D_1，D_2，…，D_n，如若这个决策序列是最优的，对于任何一个整数 k，$1<k<n$，不论前面 k 个决策是怎样的，以后的最优决策只取决于由前面决策所确定的当前状态，即以后的决策 D_{k+1}，D_{k+2}，…，D_n 也是最优的。

最优化原理是动态规划的基础。任何一个问题，如果失去了这个最优化原理的支持，就不可能用动态规划方法计算。

3.1.3 什么是动态规划

动态规划是运筹学的一个分支。与其说动态规划是一种算法，不如说是一种思维方法来得更贴切。因为动态规划没有固定的框架，即便是应用到同一道题上，也可以建立多种形式的求解算法。许多隐式图上的算法，例如求单源最短路径的 Dijkstra 算法、广度优先搜索算法，都渗透着动态规划的思想。还有许多数学问题，表面上看起来与动态规划风马牛不相及，但是其求解思想与动态规划是完全一致的。

因此,动态规划不像下一节讨论的深度或广度优先那样可以提供一套模式,需要的时候,取来就可以使用；它必须对具体问题进行具体分析处理，需要丰富的想象力去建立模型，需要创造性的思想去求解。

3.1.4 动态规划适于解决什么样的问题

准确地说，动态规划不是万能的，它只适于解决一定条件的最优策略问题。

或许，大家听到这个结论会很失望；其实，这个结论并没有削减动态规划的光辉，因为属于上面范围内的问题极多，还有许多看似不是这个范围中的问题都可以转化成这类问题。

上面所说的“满足一定条件”主要指下面两点：

(1) 状态必须满足最优化原理；

(2) 状态必须满足无后效性。

所谓的**无后效性**是指：“过去的决策只能通过当前状态影响未来的发展，当前的状态是对以往决策的总结”。

这条特征说明什么呢？它说明动态规划适于解决当前决策和过去状态无关的问题。状态，出现在策略的任何一个位置，它的地位都是相同的，都可以实施同样的决策。这就是无后效性的内涵。

3.1.5 用动态规划解题的好处

说了这么多的动态规划，它到底给我们解题能带来什么好处呢？

其实动态规划的最大优势在于它具有极高的效率，而且动态规划还有其他的优势，例如：动态规划可以得出一系列解，算法清晰简便，程序易编易调等等。

3.1.6 动态规划的逆向思维法

动态规划是一种思维方法，没有统一的、具体的模式。动态规划可以从多方面去考察，不同的方面对动态规划有不同的表述。我们不打算强加一种统一的表述，而是从多个角度对动态规划的思维方法进行讨论，希望大家在思维具体问题时，也能够从多个角度展开，这样收获会更大。

逆向思维法是指从问题目标状态出发倒推回初始状态或边界状态的思维方法。如果原问题可以分解成几个本质相同、规模较小的问题，很自然就会联想到从逆向思维的角度寻求问题的解决。

你也许会想，这种将大问题分解成小问题的思维不就是分治法吗？动态规划是不是分而治之呢？其实，虽然我们在运用动态规划的逆向思维法和分治法分析问题时，都使用了这种将问题实例归纳为更小的、相似的子问题，并通过求解子问题产生一个全局最优值的思路，但动态规划不是分治法：关键在于分解出来的各个子问题的性质不同。

分治法要求各个子问题是独立的（即不包含公共的子子问题），因此一旦递归地求出各个子问题的解后，便可自下而上地将子问题的解合并成原问题的解。如果各子问题是不独立的，那么分治法就要做许多不必要的工作，重复地解公共的子子问题。

动态规划与分治法的不同之处在于动态规划允许这些子问题不独立(即各子问题可包含公共的子子问题)，它对每个子问题只解一次，并将结果保存起来，避免每次碰到时都要重复计算。这就是动态规划高效的一个原因。

动态规划的逆向思维法的要点可归纳为以下三个步骤：

(1) 分析最优值的结构，刻划其结构特征；

(2) 递归地定义最优值；

(3) 按自底向上或自顶向下记忆化的方式计算最优值。

例题 3.1

背包问题描述：

有一个负重能力为 m 的背包和 n 种物品，第 i 种物品的价值为 v[i]，重量为 w[i]。在不超过背包负重能力的前提下选择若个物品装入背包，使这些的物品的价值之和最大。每种物品可以不选，也可以选择多个。假设每种物品都有足够的数量。

分析：

从算法的角度看，解决背包问题一种最简单的方法是枚举所有可能的物品的组合方案并计算这个组合方案的价值之和，从中找出价值之和最大的方案。显然，这种靠穷举所有可能方案的方法不是一种有效的算法。

但是这个问题可以使用动态规划加以解决。下面我们用动态规划的逆向思维法来分析这个问题。

(1) 背包问题最优值的结构

动态规划的逆向思维法的第一步是刻划一个最优值的结构，如果我们能分析出一个问题的最优值包含其子问题的最优值，问题的这种性质称为最优子结构。一个问题的最优子结构性质是该问题可以使用动态规划的显著特征。

对一个负重能力为 m 的背包，如果我们选择装入一个第 i 种物品，那么原背包问题就转化为负重能力为 m-w[i]的子背包问题。原背包问题的最优值包含这个子背包问题的最优值。若我们用背包的负重能力来划分状态，令状态变量 s[k]表示负重能力为 k 的背包和 n 种物品的背包问题中选择物品价值之和的最大值，那么 s[m]的值只取决于 s[k](k≤m)的值。因此背包问题具有最优子结构。

(2) 递归地定义最优值

动态规划的逆向思维法的第二步是根据各个子问题的最优值来递归地定义原问题的最优值。对背包问题而言，有状态转移方程：

$$s[k]=\begin{cases}\max\{s[k-w[i]]+v[i]\}\ (\text{其中 } 1\leqslant i\leqslant n,\ \text{且 } k-w[i]\geqslant 0) & \text{若 } k>0 \text{ 且存在 } 1\leqslant i\leqslant n \text{ 使 } k-w[i]\geqslant 0,\\ 0 & \text{否则。}\end{cases}$$

有了计算各个子问题的最优值的递归式，我们就可以直接编写对应的程序。下述的函数 knapsack 是输入背包的负重能力 k，返回对应的子背包问题的最优值 s[k]：

```
function knapsack(k:integer):integer;
begin
  knapsack←0;
  for i←1 to n do
    if k-w[i]≥0 then
    begin
      t←knapsack(k-w[i])+v[i];
      if knapsack<t then knapsack←t;
    end;
end;
```

上述递归算法在求解过程中反复出现了一个子问题，且对每次重复出现的子问题都要重新解一次，这需要多花费不少时间。下面先考虑一个具体的背包问题。例如，当

m=3，n=2，v[1]=1，w[1]=1，v[2]=2，w[2]=2，

图 3.1.1 示出了由调用 knapsack(3)所产生的递归树，每一个结点上标有参数 k 的值，请注意某些数出现了多次。

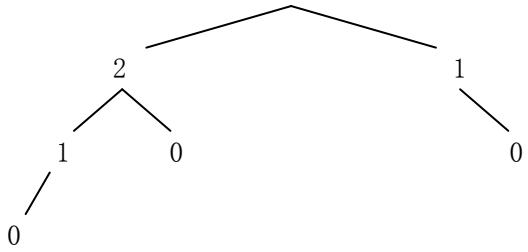


图 3.1.1 由调用 knapsack(3)所产生的递归树

例如，knapsack(1)被引用了两次：在计算 knapsack(3)和 knapsack(2)中分别被引用；而 knapsack(0)更是被引用了三次。如果 knapsack(1)和 knapsack(0)每次都要被重新计算，则增加的运行时间相当可观。

下面，我们来看看动态规划是如何解决这个问题的。

(3) 按自顶向下记忆化或自底向上的方式求最优解

一般地，如果一个解最优化问题的递归算法经常反复地解重复的子问题，而不是总在产生新的子问题时，我们说该最优化问题包含重迭子问题。这类问题不宜用分治法求解，因为分治法递归的每一步要求产生相异的子问题。在这种情况下，采用动态规划是很合适的，因为该方法对每一个子问题只解一次，然后把解存放在一个表中，以便在解同样的子问题时查阅，充分利用了重迭子问题。一般来说，解决重迭子问题的方式有两种。

(a) 自顶向下的记忆化方式

该方式的程序流程基本按照原问题的递归定义，不同的是，它专门设置了一张表，以记忆在求解过程中得出的所有子问题的解。一个记忆的递归算法为每个子问题的解在表中记录一个表项。初始时每个表项都包含一个特殊值，以示该表项的解有待填入。例如背包问题中

s[i]=-1; (0≤i≤m)

当在递归算法的执行中第一次遇到一个子问题时（即 s[i]=-1），计算其解并填入表中，以后每遇到该子问题，只要查看表中先前填入的值即可。

下面，我们按照自上而下的记忆化方式，写出求背包问题的递归算法。

函数 memorized_knapsack 输入背包的负重能力 k，返回背包问题的解 s[k]。s 表应设为全局变量，使得函数执行后，传出所有子问题的解 s[i](0≤i≤m)。

```
function memorized_knapsack(k:integer):integer;
begin
  if s[k]=-1 then
  begin
    s[k]←0;
    for i←1 to n do
      if k-w[i]≥0 then
      begin
        t←memorized_knapsack(k-w[i])+v[i];
        if s[k]<t then s[k]←t;
      end;
  end;
  memorized_knapsack←s[k];
end;
```

我们在主程序通过调用 memorized_knapsack(m)即可获得背包问题的解。

显然这种自顶向下记忆化的算法效率比重复解重迭子问题的 knapsack 算法要强一些。

此外，在程序中加入判别哪些子问题需要求解的语句，只解那些肯定要解的子问题，从而节省了时间。

(b) 自底向上的方式

假设我们要解决在分析函数 knapsack 当中提出的那个具体的背包问题。观察一下在调用 memorized_knapsack(4)的过程中，s[k]被计算出来的顺序。我们发现，首先是 s[0]被计算出来，然后是 s[1]，s[2]，最后 s[3]被计算出来，这与调用的次序正好相反。

动态规划的执行方式是自底向上，所有的子问题计算一次，充分利用重迭子问题。因此，我们很自然就想到与这种自底向上的执行方式相应的，采用自底向上的方式递推所有子问题的最优值。

我们知道，计算负重能力为 k 的背包问题的解仅依赖于计算负重能力小于 k 的背包问题的解，因此填 s 表的方式与解决负重能力递增的背包问题相对应：

最初设定负重能力为 0 的背包问题的解 s[0]=0。然后依次考虑负重能力为 1,2,…,m 的背包问题的解。每填入一个 s[k](0≤k≤m)时，充分利用所有重迭子问题的解：s[k-w[i]](0≤i≤n，k-w[i]≥0)

下面是按照自底向上方式计算背包问题的算法流程。过程 knapsack_order 输入背包的负重能力 k。s 表设为全局变量。过程执行后所有子问题的解通过 s 表传给主程序。

```
procedure knapsack_order(k:integer);
begin
  for i←1 to k do s[i]←0;
  for j←1 to k do
  for i←1 to n do
    if j-w[i]≥0 then
    begin
      t←s[j-w[i]]+v[i];
      if s[j]<t then s[j]←t;
    end;
end;
```

我们在主程序调用 knapsack_order(m)，过程执行后 s[m]即为背包问题的解。

最后，我们分析一下背包问题的动态规划解法的复杂度。所需的存储开销主要在 s 表上，因此空间复杂度是 O(m)。而整个 s 表用两重循环求出，循环内语句的执行只需常数时间，因此时间复杂度是 O(m*n)。

自顶向下的记忆化是动态规划的一种变形。动态规划的执行方式是自底向上，因此自底向上的计算方式是与动态规划的执行方式相一致的。它无需递归代价，且维护记忆表的开销也要小些，因此其效率通常好于自顶向下的记忆法。

但是，在动态规划的执行过程中，并不是所有的子问题都要用到的。因此，对某个具体问题而言，可能有大部分子问题的最优值是不必计算的。自底向上的计算方式无法判断那些子问题是需要求解的，所以它将一视同仁地处理所有的子问题，可能会把大量的时间都花在计算不必解决的子问题上；而自顶向下的记忆法可以判断那些子问题是需要求解的，只解那些肯定要解的子问题，在这个意义上，自顶向下的算法效率又好于自底向上。所以到底那种方式效率更高，我们要具体问题具体分析。

最后，给出求解背包问题的程序如下：

程序名：ex3_1_1.dpr

```
//背包问题程序
program knapsack;

{$APPTYPE console}

const
  maxn=100;
  maxm=1000;

var
  m,n:integer;
  s:array[0..maxm] of integer;
  v,w:array[1..maxn] of integer;
//输入数据
procedure read_data;
var
  i:integer;
begin
  read(m,n);
  for i:=1 to n do read(v[i],w[i]);
end;
//采用自底向上的方式求解背包问题
procedure knapsack_order;
var
  i,j,t:integer;
begin
  for i:=1 to m do s[i]:=0;
  for j:=1 to m do
  for i:=1 to n do
    if j-w[i]>=0 then
    begin
      t:=s[j-w[i]]+v[i];
      if s[j]<t then s[j]:=t;
    end;
end;
//输出结果
procedure output_solution;
begin
  writeln(s[m]);
end;

begin
  read_data;
  knapsack_order;
  output_solution;
end.
```

3.1.7 动态规划的正向思维法

正向思维法是指从初始状态或边界状态出发，利用某种规则不断到达新的状态，直到问题目标状态的方法。动态规划的正向思维法，正是从已知最优值的初始状态或边界状态开始，

按照一定的次序遍历整个状态空间，递推出每个状态所对应问题的最优值。

提出动态规划的正向思维法的根本原因，是为了摆脱逆向思维法当中那种将大问题转化为子问题的思维框框，提供一种新的思维方式。在正向思维法中，我们不再区分原问题和子问题，将动态规划的过程看成是从状态到状态的转移。我们将所有的状态构造出一个状态空间，并在状态空间中设想一个状态网络，若对两个状态 i,j，存在决策变量 d_i 使 $t(i,d_i)=j$，则向状态网络添加有向边<i,j>。给定已知最优值的初始状态或边界状态，我们可以沿着有向边推广到未知最优值的新状态，利用状态转移方程得到新状态的状态变量的最优值。我们可以用这种方式遍历整个状态空间，得到每个状态的状态变量的最优值。

因为正向思维法中不再区分原问题、子问题、子子问题，所以我们不再按照问题被递归调用求解的相反次序的方法确定状态最优值的计算次序。从上面我们知道可以按照状态的拓扑序列来计算每个状态的最优值，于是我们用一个新的名词“阶段”来描述在状态空间遍历的过程中，各个状态最优值的计算次序。我们将每个状态和一个阶段挂钩，前一个阶段的状态先计算，后一个阶段的状态后计算。有的时候我们甚至将一组状态和一个阶段挂钩，前一个阶段的那组状态先计算，后一个阶段的那组状态后计算，而在同一个阶段内，那些状态的计算次序可以是任意的。

动态规划的正向思维法的要点可归纳为以下三个步骤：

(1) 构造状态网络；

(2) 根据状态转移关系和状态转移方程建立最优值的递推计算式；

(3) 按阶段的先后次序计算每个状态的最优值。

在下一节“最短路问题”当中，我们将结合最短路问题来示范动态规划的正向思维法。

动态规划的正向思维法带给我们什么启示呢？动态规划需要按阶段遍历整个状态空间，因此动态规划的效率取决于状态空间的大小和计算每个状态最优值的开销：如果状态空间的大小是多项式的，那么应用动态规划的算法就是多项式时间的；如果状态空间的大小是指数的，那么应用动态规划的算法也是指数时间的。因此，找一个好的状态划分对动态规划的效率是至关重要的。

将这个结论应用到逆向思维上，我们得出如下结果：一个问题的最优子结构常常暗示了动态规划的状态空间。典型的情况是，某一个特定问题可有几类“自然”的子问题，不同的子问题往往意味着不同的最优子结构，从而我们得到不同的状态划分方案。但是由这些不同的状态得到的算法的效率可能差异极大。例如，某个问题的输入是一个有序序列，有两类“自然”的子问题，一类子问题的输入是原问题输入序列的所有子序列，另一类子问题的输入是原问题输入序列的任意元素构成的新序列。显然若我们采用后者的最优子结构来建立状态，它的状态空间必然远远大于基于前者的状态空间。那末基于后者的状态空间的动态规划则要解许多不必要的问题，使得算法的效率骤然下降。

从动态规划的正向思维法的分析可以看出，我们从已知最优值的初始状态和边界状态出发，利用最优化原理，一步一步向未知的目标状态推进，直到目标状态的最优值解决。这种“从已知推广到未知”的思维，正是动态规划正向思维法的精髓。大家在运用动态规划的正向思维法时如果能掌握这一点，那么就能达到应用自如的境界。

在第四章当中有三道应用动态规划求解的例题（第四节“防卫导弹”、第五节“邮票问题”和第八节“旅游预算”），供大家分析和学习。希望读者能够从题目的分析中领会这一点：应用动态规划解题是富于技巧和创造性的，没有固定的模式可套；题目出现的形式多种多样，而且大部分表面上与动态规划看不出直接的联系。只有在充分把握其思想精髓的前提下大胆联想，才能达到得心应手，灵活运用的境界。

第二节 最短路问题

最短路问题是一类比较简单但又很重要的最优化问题。所谓最短路问题就是指给定起点及终点，并知道由起点到终点的各种可能的路径，问题是要找一条由起点到终点的最短的路，即长度最短的路。需要指出的是，最短路问题中的“长度”可以是通常意义下的距离，也可以是运输的时间或者运输费用等等。而且，有些与运输根本没有关系的问题也可以化为求最短路的模型，例如求关键路径（实际上是 AOE 网中的最长路）问题等。甚至有些问题从表面上看根本和最短路问题扯不上关系，却也可以归结为最短路问题。

下面我们给最短路问题下一个抽象的定义：

设 G=(V,E)是一个（有向）图，它的每一条边(u,v)∈E（对有向图，就是<u,v>∈E）都有一个权 w[u,v]。在 G 中指定一个顶点 v_s，要求把从 v_s 到 G 的每一个顶点 v_j(v_j∈E)的最短（有向）路找出来（或者指出不存在从 v_s 到 v_j 的（有向）路，即 v_s 不可达 v_j)。

在上述定义中，由于源顶点 v_s 是给定的，因此我们又形象地称为单源最短路径问题。

当然最短路问题远不止上述一种，但它们一般都可用单源最短路径问题的算法来解决。例如：

(1) 单目标最短路径问题：找出每一个顶点 v_j(v_j∈E)到某指定顶点 v_s 的每条最短路径。把图中的每条边的方向反向，我们就可以把这一问题变成单源最短路径问题；

(2) 单对顶点间的最短路径问题：对于给定顶点 u 和 v，找出一条从 u 到 v 的最短路径。如果我们解决了源顶点为 u 的单源最短路径问题，则这问题自然获得了解决。目前还没有一种比单源算法更快的算法来解决这一问题；

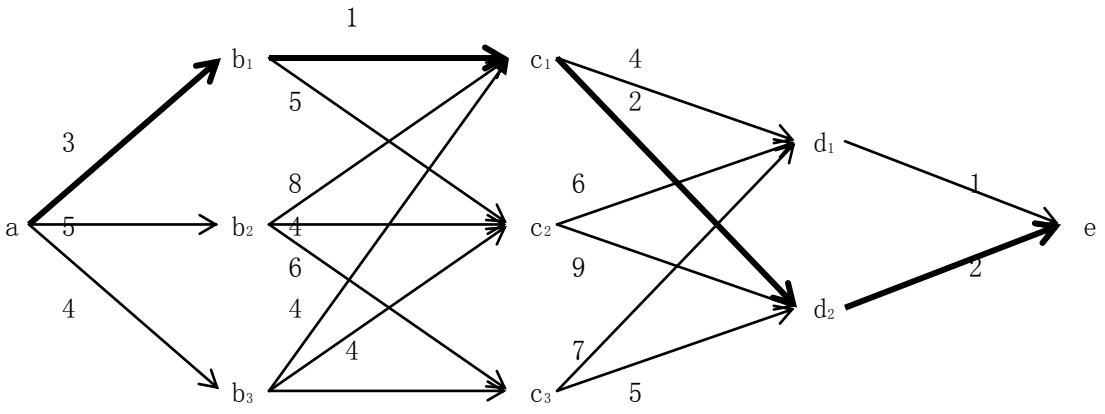
(3) 每对顶点间的最短路径问题：对于每对顶点 u 和 v 找出从 u 到 v 的最短路径。我们可以用单源算法将每个顶点作为源点运行一次就可以解决这一问题。但是还有运行效率更高的算法，我们将在这一节中介绍有关算法。

首先我们从单对顶点间的最短路径问题引出单源最短路径算法。

3.2.1 有向无环图上的单对顶点间的最短路径问题

例题 3.2.1：

由起点 a 到终点 e 的路线图如图 3.2.1 所示，连接两点之间的路的长度用连线上的数字表示，求由起点 a 到终点 e 的最短路。



2

图 3.2.1　求由起点 a 到终点 e 的最短路示意图

一般地，求一个具有 n 个结点的图的最短路，如果使用穷举法，其运算量是 n 的指数函数。当 n 比较大，同时图的深度较深时，这个算法在时间效率上是不可取的。如果用最优化原理来思考这个问题，我们可以注意到最短路有这样一个特性，即最短路径的子路径也必然是最短路径，而且最短路径的长度只和各条子路径的最短长度有关。这提示我们：最短路径问题可以用动态规划解决。

以图中的顶点划分状态。令状态量 s[k]表示顶点 a 到顶点 k 的最短距离，则有状态转移方程：

$$s[k]=\begin{cases}0 & \text{若 } k=a,\\ \min\{s[i]+w[i,k]\}\ (\text{其中 } 1\leqslant i\leqslant n,\ \text{且}\langle i,k\rangle\in E) & \text{若 } k\neq a \text{ 且存在 } 1\leqslant i\leqslant n \text{ 使}\langle k,i\rangle\in E,\\ +\infty & \text{否则。}\end{cases}$$

容易验证这一状态划分满足动态规划的两个条件：最优化原理和无后效性。所以我们可以运用动态规划得到相应的算法。下面给出一个按自上而下记忆化方式求单对顶点间的最短路径问题的递归算法。

函数 memorized_ shortest_distance 以顶点 k 作为输入，返回顶点 a 到顶点 k 的最短距离 s[k]，s 表应设为全局变量。初始时 s[a]=0，s[i]=-1(1≤i≤n,i≠a)，以示该表项的解有待填入。

```
function memorized_shortest_distance(k:integer):integer;
begin
  if s[k]=-1 then
  begin
    s[k]←+∞;
    for i←1 to n do
      if <i,k>∈E then
      begin
        t←memorized_shortest_distance(i)+w[i,k];
        if t<s[k] then s[k]←t;
      end;
  end;
  memorized_shortest_distance←s[k];
end;
```

主程序通过调用 memorized_ shortest_distance(e)获得顶点 a 到顶点 e 的最短距离是 8。求出的最短路为 a→b_1→c_1→d_2→e，如图中粗线所示。

如果我们采用自底向上的计算方式，首先要解决子问题计算次序的问题，但是从最优值的递归计算式中难以直接观察出一个合理的子问题计算次序。如何确定这个次序呢？注意到上图具有有向无环图的性质，于是我们得到一个递推的算法：

(1) 求出 G 的一个拓扑排序，将各顶点按照其在拓扑顺序中的位置号来编号，即若顶点 u 在拓扑序列中的位置号是 k，则 u 的编号就是 k，而顶点 u 记作 v_k。并且设 a,e 两个顶点记为 v_s, v_t；

(2) 令状态量 S[k]表示从顶点 v_s 到顶点 v_k 的最短距离，则有

(a) 初始状态：S[v_s]=0，S[v_i]=+∞(1≤i≤n,i≠s)；

(b) 状态转移方程：S[v_k]=min{S[v_j]+w[v_j,v_k]}，若 s<k 且<v_j,v_k>∈E。

我们知道，动态规划不仅求出了原问题的最优值，还顺便带来了所有子问题的最优值。换句话说，当这个递推算法执行完之后，我们得到的不仅仅是由起点 a 出发到终点 e 的最短路，而且还得到了从顶点 a 到所有各中间顶点的最短路。我们本来求的是单对顶点间的最短路径，但是现在我们得到了单源最短路径问题的解答。因此，上述算法不仅是有向无环图上单对顶点间的最短路径算法，也是同一个图上的单源最短路径算法。

3.2.2 任意图上的单源最短路径问题

现在的问题是，对任意给定的路线图，如何定出这个求解子问题顺序的序列呢？有向无环图上的单对顶点间的最短路径算法采用的方法是将图中的顶点进行拓扑排序，但是这只能算是其中的一种考虑。不要忽视以下三点：

(1) 有向无环图上的单对顶点间的最短路径算法不适用于有向带环图和无向图，因为这些图的顶点不能进行拓扑排序；

(2) 即使图中含有长度为正数的回路，问题还是有解的；

(3) 如果图中含有起点可达的负权回路，算法应该宣告问题无解。

有向无环图上的单对顶点间的最短路径算法无法解决这三个问题。

要从这个困局之中摆脱出来，我们采用动态规划的正向思维法，从另一个角度来看这个问题。

在最短路问题里，状态就是各个顶点，所以给定的图就构成了状态网络，省去了我们建立状态网络的麻烦。动态规划的正向思维法的精髓，就是从已知推广到未知，将已知逐步扩展到整个状态空间。这种从已知最短路的顶点推广到未知最短路的顶点的思路，就是求任意图的单源最短路径算法的核心。

假设图中没有起点 a 可达的负权回路，令 s[v]表示从顶点 a 到顶点 v 的最短距离。我们来描述一下正向思维法求最短路的过程。开始时只知道起点 a 到 a 的最短距离 s[a]=0，其他的都是未知；然后我们用 a 到其所有邻接点 v 的距离 w[a,v]去更新那些顶点 v 的最短距离 s[v]。在这些顶点当中，必定有一个距离 a 最近的顶点 u（否则没有顶点与 a 相邻，不存在 a 到其他顶点的最短路），它与 a 的最短距离就是 s[u]（因为图中没有 a 可达的负权回路，所以 s[u]的值不可能再小了），这样我们从已知推广到未知，令已知的信息多了一点；接着我们从 u 出发重复上面的过程，这时所有顶点除 a 和 u 之外 s[v]最小的顶点 w 就是下一个已知最短距离的顶点，已知的信息又多了一点……以此类推，直至已知扩展到整个状态空间，这样我们就得到了顶点 a 到所有顶点的最短距离。

将以上的思路化为算法，我们就得到著名的 dijkstra 算法。它是求单源最短路径问题的最经典的算法。

```
procedure dijkstra;
begin
  s[a]←0,s[v]←+∞(v=1..n,v≠a);
  S←{};
  for i←1 to n-1 do
  begin
    令 u∈V－S 使 s[u]≤s[v](v∈V－S);
    对所有 u 的邻接顶点 v do
    begin
```

```
      t←s[u]+w[u,v];
      if t<s[v] then s[v]←t;
    end;
    S←S∪{u};
  end;
end;
```

集合 S 当中的顶点是已知达到最优值的顶点；在每次循环当中，u 就是新的已知最优值的顶点。

需要特别指出的是，在单源最短路径问题的实例中，可能存在权为负的边。如果图不包括从起点 a 可达的负权回路，dijkstra 算法求出的最短路径的长度依然正确，即便它是一个负值也是如此。

如果图中存在一条起点可达的负权回路，最短路的长度就无从计算了，因为我们总可以顺着找出的“最短”路径再穿过负权回路从而获得一条权值更小的路径，最短路的长度实际上成了-∞，此时 dijkstra 算法便爱莫能助了。但是，如果我们能预先检测出图中包含起点可达的负权回路，就可以补救这一问题。

对有向图来说，一种简单高效的检测负权回路的方法是拓扑排序。因此一个完整的求有向图最短路的算法是：预先对有向图进行一次拓扑排序。若图中存在起点可达的负权回路，则宣告该问题无解；否则用 dijkstra 算法求最短路及其权。

对无向图来说，可以通过将每条边改成方向相反的有向边的方法将无向图改造成有向图，再调用上面的算法求最短路。

我们来简单分析一下 dijkstra 算法的时间复杂度。

dijkstra 算法的外层循环的次数是 n-1；

如果数据结构组织得当（例如，采用堆来存放 s 表），在 V-S 里的顶点 v 当中找 s[v]最小的顶点 u 那一步只需 $O(\log_2 n)$时间；

对于普通图来说，对顶点 u 的所有邻接点 v 进行松弛操作的时间复杂度是 O(n)；

其它操作只需常数时间 O(1)；

即使要给 Dijkstra 算法增加负权回路的检测，求有向图的拓扑排序的时间复杂度也仅为 O(n)。

因此，在一般情况下，dijkstra 算法的时间复杂度是 $O(n+(n-1)*(\log_2 n+n))=O(n^2)$。

但是当图 G 是一个稀疏图时，特别地，当图中所有顶点的度数都有上界 C（C 是常数），如果数据结构组织得当（例如，采用邻接表存储各个顶点的邻接顶点，且采用堆来存放 s 表），可以将 Dijkstra 算法的时间复杂度降为 $O(n\log_2 n)$。当 n 特别大的时候，这一点就显得至关重要了。

下面我们举例说明稀疏图上的 dijkstra 算法的应用。

例题 3.2.2：

迷宫镜子问题。（本题取材于 ACM/ICPC 题库第 258 题）

在一个遥远的银河系里，有一群科学家发明了一种装置可以杀死所有的计算机病毒。这种装置可以在整个宇宙中运行，因为它使用了激光。这种装置有一个特别的组成部分：一个含有黑洞和镜子的二维迷宫，它是一个关键部分。

这个迷宫有两个出口。迷宫中所有的镜子都具有两个反射面，这些镜子总是与激光成四十五度摆放，但它们可以作九十度旋转，所以每一个镜子有两种可能状态。激光会完全被黑洞吸收。激光能够成九十度地穿过自身。

题目将给出一个迷宫，在这个迷宫当中，镜子必须恰当地放置以使激光能在一个出口射

入，另一个出口射出。迷宫的边缘除了两个出口以外都是黑洞。由于给出的迷宫中镜子的角度可能不正确，所以需修改镜子的角度使激光能通过迷宫。请设计找出一个适当放置镜子的方案的算法，使需要修改角度的镜子数目最少。

输入：迷宫用 M 列 N 行字符表示(3≤M,N≤100)，镜子用“\”和“/”表示，黑洞用“*”表示，空白用点“.”表示，例如：

```
**** (M=4,N=5)        或   *.** (M=4,N=4)
*\/*                       *.\*
*./.                       *\\*
*..*                       **.*
*.**
```

输出：请输出需要修改角度的镜子最少数目。如果不存在放置镜子的方案使激光从一个出口到达另一个出口，则输出-1。

分析：

用迷宫里的每面镜子的坐标 p 和激光的入射方向 d 作为状态(p,d)并构造状态网络。若对某面镜子 p_1 和特定的入射角度 d_1，激光可以以某个入射角度 d_2 射到另一面镜子 p_2，则在对应的两个状态之间连一条边：对于状态(p_1,d_1)，如果无需改变镜子的摆放方式就可以使激光射到(p_2,d_2)，则相应边的权值是 0；否则相应边的权值是 1。设激光从一个出口可以射到(p_s,d_s)，而从(p_t,d_t)可以射到另一个出口，则迷宫镜子问题转化为状态网络上状态(p_s,d_s)到状态(p_t,d_t)的最短路问题。

因为迷宫最大 100×100，但是只有迷宫内部才能放置镜子；激光的入射方向有四个，所以不同的状态数最多有 98*98*4=38416，构造的状态网络具有 38416 个顶点，规模可谓庞大。但是每个顶点最多只和两个顶点有边相连（因为镜子只有两个不同的放置角度），因此这个状态网络是典型的每个顶点度有常数上界的稀疏图，可以应用稀疏图上的 dijkstra 算法解决。

下面给出迷宫镜子问题的程序解答。

程序名：ex3_2_1.dpr

```
//迷宫镜子问题程序
program mirror;

{$APPTYPE console}

uses SysUtils;

type
  //状态结点
  TNode=record
    p: Word; //坐标
    d: Byte  //激光入射角度
  end;
const
  //迷宫维数最大值
  MaxN = 100;
  //四个方向在两个坐标轴的增量
  DX: array[0 .. 3] of ShortInt = (0, -1, 0, 1);
  DY: array[0 .. 3] of ShortInt = (1, 0, -1, 0);
var
  N, M: Byte; //迷宫的长和宽
  Maze: array[1 .. MaxN, 1 .. MaxN] of Char; //迷宫
  //Dist[i]=从入口到下标 i(由坐标和入射角度计算)的最短距离
```

```
  //A[i]=第 i 小的距离的下标
  //B[i]=A 的反查表
  Dist, A, B: array[1 .. MaxN * MaxN * 4] of Integer;
  Opening: array[0 .. 1] of TNode; //入口和出口
//将坐标(i,j)和激光入射角度 d 转化为下标
function Index(i, j, d: Word): Word;
begin
  Index:=((i - 1) * M + (j - 1)) * 4 + d + 1
end;
//输入数据
procedure InputData;
var
  i, j, tot: Integer;
begin
  Readln(N, M);
  tot := 0;
  for i := 1 to N do
  begin
    for j := 1 to M do
    begin
      Read(Maze[i, j]);
      if (Maze[i, j] = '.') and ((i = 1) or (i = N) or (j = 1) or (j
= M)) then
        with Opening[tot] do
        begin
          if j = 1 then d := 0
          else if i = N then d := 1
          else if j = M then d := 2
          else d := 3;
          if tot = 1 then d := (d + 2) mod 4;
          p := Index(i, j, d);
          Inc(tot);
        end;
    end;
    Readln;
  end;
end;
//交换下标为 i 和 j 的元素
procedure Swap(i, j: Word);
var
  k: Word;
begin
  k := A[i]; A[i] := A[j]; A[j] := k;
  k := B[A[i]]; B[A[i]] := B[A[j]]; B[A[j]] := k;
end;
//将下标为 i 的元素下沉到合适位置，表的长度是 last
procedure GoDown(i, last: Word);
var
  m: Word;
begin
  m := i;
  repeat
    i := m;
    if (i * 2 <= last) and (Dist[A[i*2]] < Dist[A[m]]) then m := i *
2;
    if (i * 2 + 1 <= last) and (Dist[A[i*2+1]] < Dist[A[m]]) then m :=
i * 2 + 1;
    if i <> m then Swap(i, m);
```

```
    until i = m;
  end;
  //将下标为 i 的元素上浮到合适位置
  procedure GoUp(i: Word);
  begin
    while (i > 1) and (Dist[A[i]] < Dist[A[i div 2]]) do
    begin
      Swap(i, i div 2);
      i := i div 2;
    end
  end;
  //从下标 p 和激光入射角度 d 找下一个到达的镜子的下标
  function FindMirror(var p: Word; d: Byte): Boolean;
  var
    x, y: Byte;
  begin
    FindMirror := False;
    p := (p - 1) div 4;
    x := (p div M) + 1;
    y := (p mod M) + 1;

    repeat
      Inc(x, DX[d]);
      Inc(y, DY[d]);
      if (x < 1) or (x > N) or (y < 1) or (y > M) or (Maze[x, y]='*') then
Exit;
      p:=Index(x, y, d);
    until (Maze[x, y] in ['\', '/']) or (p = Opening[1].p);

    if p <> Opening[1].p then
      if Maze[x, y]='\' then d := 3 - d else d := (5 - d) mod 4;
    p := Index(x, y, d);
    FindMirror := True;
  end;
  //稀疏图上的 Dijkstra 算法
  procedure Dijkstra;
  var
    i, j, k: Word;
  begin
    for i := 1 to N * M * 4 do
    begin
      A[i] := i; B[i] := i; Dist[i] := MaxInt
    end;

    j := Opening[0].p;
    if FindMirror(j, (j - 1) mod 4) then Dist[j] := 0 else Exit;
    for i:=N * M * 4 div 2 downto 1 do GoDown(i, N * M * 4);

    for i := N * M * 4 downto 1 do
    begin
      k := A[1];
      if (Dist[k] = MaxInt) or (k = Opening[1].p) then Exit;
      Swap(1, i);
      GoDown(1, i - 1);

      j := k;
      if FindMirror(j, (j - 1) mod 4) then
        if Dist[j] > Dist[k] then
```

```
      begin
        Dist[j] := Dist[k];
        GoUp(B[j])
      end;

    j := k;
    if FindMirror(j, (j + 1) mod 4) then
      if Dist[j] > Dist[k] + 1 then
      begin
        Dist[j] := Dist[k] + 1;
        GoUp(B[j])
      end;
  end;
end;
//输出需要改变的镜子最少数目
procedure Print;
begin
  if Dist[Opening[1].p] = MaxInt then
    Writeln(-1)
  else
    Writeln(Dist[Opening[1].p]);
end;

begin
  Assign(Input, 'mirror.in');
  Reset(Input);
  Assign(Output, 'mirror.out');
  Rewrite(Output);

  InputData;
  Dijkstra;
  Print;

  Close(Input);
  Close(Output);
end.
```

3.2.3 任意图上每对顶点间的最短路径问题

现在我们再来讨论图中每对顶点间的最短路问题。当然，可以把单源最短路径算法运行 n 次来求每对顶点间的最短距离，每次依次把图中的每个顶点作为源点。如果所有边非负，则对每个顶点运行一次 dijkstra 算法。下面我们将介绍一种直接求任意图上每对顶点间的最短路径问题的算法。

求任意图上每对顶点间的最短路径问题的算法仍然基于动态规划的思想。用起点、终点和已经检查的点数划分状态。令状态量 s[k,i,j]表示从顶点 i 出发，中途经过以图中前 k 个顶点的集合{1,2,…,k}为中间点到顶点 j 的最短距离，则有状态转移方程：

s[0,i,j]=w[i,j] i,j=1,2,…,n。

s[k,i,j]=min(s[k-1,i,j],s[k-1,i,k]+s[k-1,k,j]) k>0， i,j=1,2,…,n。

其中 s[n,i,j]即为 i,j 之间的最短距离。

将已经检查的点数作为阶段，并注意到第 k 个阶段的状态只和第 k-1 个阶段的状态有关，

于是我们可以令多阶段共用两个阶段的存储空间；又由于对所有顶点 i，w[i,i]=0，因此有 s[k-1,i,k]=s[k,i,k]且 s[k-1,k,j]=s[k,k,j]，于是我们又可以将两个阶段的存储空间需求降低为一个阶段的存储空间需求，得到求任意图上每对顶点间的最短路径问题的 floyd 算法：

```
procedure floyd;
begin
  s←w;
  for k←1 to n do
  for i←1 to n do
  for j←1 to n do
    if s[i,k]+s[k,j]<s[i,j] then s[i,j]←s[i,k]+s[k,j];
end;
```

算法的形式十分简洁，主要部分只有 i,j,k 三重循环，故其时间复杂度为$\Theta(n^3)$。

floyd 算法不仅可以求解任意图上每对顶点间的最短路问题，而且它还可以衍生出许多有用的算法。下面即是一例。

例题 3.2.3：

求图的传递闭包。

输入图 G，输出矩阵 T=[t(i,j)]。若 t[i,j]=true，则 i 到 j 有通路；若 t[i,j]=false，则 i 到 j 无通路。

分析：

显然这个问题比图上每对顶点间的最短路问题简单多了：对于任何顶点对 i,j∈V，只要判断是否存在一条从 i 到 j 的路径，而不需要了解最短路径的权和构成。大家可以马上得到一个算法：对图中的每条边赋权为 1，然后对该图运行 floyd 算法。如果从顶点 i 到顶点 j 存在一条通路，我们将得到 s[i,j]<n；否则，将得到 s[i,j]=+∞。但对于这么简单的问题而言，这种算法如同“杀鸡用牛刀”，得不偿失。下面我们来介绍一种类似的方法，这种方法在实际应用中可以节省时间和空间。

我们仍然运用动态规划思维，采用与 floyd 算法相同的状态划分，令状态量 s[k,i,j]表示是否存在从顶点 i 出发，中途经过以图中前 k 个顶点的集合{1,2,…,k}为中间点到顶点 j 的通路，则有状态转移方程：

$$s[0,i,j]=\begin{cases}\text{true} & \text{若 } i=j \text{ 或}(i,j)\in E,\ i,j=1,2,\cdots,n,\\ \text{false} & \text{若 } i\neq j \text{ 且}(i,j)\text{不属于 E。}\end{cases}$$

s[k,i,j]=s[k-1,i,j] or (s[k-1,i,k] and s[k-1,k,j]) k>0，i,j=1,2,…,n。

其中 s[n,i,j]即为 t[i,j]。

和 floyd 算法的状态转移方程比较，求图的传递闭包的算法将 floyd 算法中的算术运算操作 min 和+用相应的逻辑运算 and 和 or 代替。对于 i,j 和 k=1,…,n，如果图 G 中从顶点 i 到顶点 j 存在一条通路且所有中间顶点均属于集合{1,2,…,k}，则定义 s[k,i,j]=true，即(i,j)边加入传递闭包；否则 s[k,i,j]=false。

与 flody 算法类似，运用动态规划的优化手段，我们直接用 t[i,j]存储 s[k,i,j]。这时 t 的递推式可表示为：

在第 k 次迭代，t[i,j]=t[i,j] or (t[i,k] and t[k,j]) i,j=1,2,…,n。

由于布尔值的存储量少于整数值，位逻辑操作的执行速度快于对整数字长的数据的算术运算操作，因此采用专门的传递闭包算法的方案无论在时间上和空间上都优于采用 floyd 算法的方案。

按上述说明，得到求图的传递闭包的算法。我们把它称为 warshall 算法。

```
procedure warshall;
begin
  for i←1 to n do
  for j←1 to n do
    t[i,j]←(i=j) or (i,j)∈E;
  for k←1 to n do
  for i←1 to n do
  for j←1 to n do
    t[i,j]←t[i,j] or (t[i,k] and t[k,j]);
end;
```

在第四章第七节“师生树”当中，我们将用到类似的算法判断回路。

关于最短路问题的算法就介绍到这里。和距离有关的最优化问题还有一些，例如最长路问题，其实它们的求解思路和最短路问题都是大同小异的。最短（长）路问题无疑是动态规划最重要的应用，因为很多能用动态规划解决的最优化问题，通过适当的转化，都可以归结为最短（长）路问题，从而可以套用现成的算法进行解决；而且最短路问题的所有求解的思路都是基于动态规划的，因此掌握好最短路问题的整个思路，对理解和把握动态规划的思维方式有莫大的帮助。

第三节 搜索算法

在第一节“动态规划”当中我们有一个结论：即如果状态空间呈指数复杂度，则动态规划的时间复杂度也是呈指数的，其效率与穷举法相比并没有明显优势。如果找不到多项式复杂度的状态表示，或者状态的数目虽然多项式复杂度的，但是状态数×每个状态最优值的计算复杂度很大（例如 n^{10}），是否就无计可施了呢？

在这一章的开头我们说过，很多的最优化问题至今只找到指数时间的算法，因此算法的优化显得十分重要；在茫茫状态空间当中寻找最优解不能靠瞎猜瞎碰，不能靠简单地枚举，必须有一种系统的方法帮我们尽快找到问题的解答。

在这节中我们年介绍的搜索算法，是一种在状态空间中寻找特定的目标状态及到达目标状态的途径的系统方法。搜索是计算机求解问题的最基本方法，适用范围很广，没有像动态规划那样对状态有最优化原理和无后效性的约束。当然，对具体的算法，特别是运用了某种智能化的优化手段，也许会带来某些具体的约束。

首先，让我们从搜索的基本概念入手。

3.3.1 搜索的基本概念

当所给定的问题用状态网络表示时，则求解过程可归结为对状态空间的遍历。搜索空间是搜索达到的状态空间，它是状态空间的一部分。从图 3.3.1 表示出的搜索过程可以看出，

当问题有解时，使用不同的搜索策略，找到解的搜索空间范围是有区别的。一般来说，对大空间问题，搜索策略是要解决组合爆炸的问题。

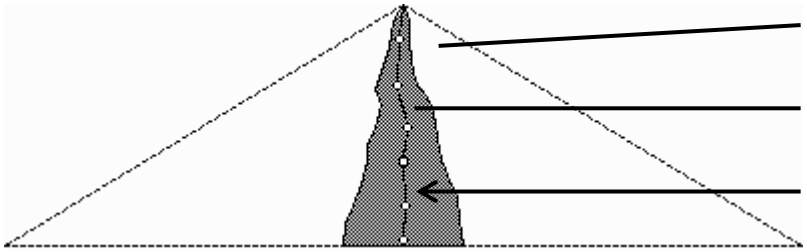


图 3.3.1 问题的状态空间、搜索空间和解路径的示意图

状态是客观存在的数学抽象，它是不会被创造或消灭的；但是搜索的过程是一个从已知推广到未知的过程，我们需要有一个新的概念“结点”去描述这种推进，它是表示状态特征及其关联方式的基本信息单元。结点表示已知的范围，结点及其相互联系构成了搜索空间。在搜索开始之间，我们已经有表示初始状态的结点；搜索的过程就表现为结点的不断扩展，即新结点的产生。从结点 u 应用规则产生新结点 v，我们说扩展结点 u，生成新结点 v。

在一般的情况下，问题可能有多个解，但有时会要求得到有某些附加约束条件的解，例如要求步数等于特定值、距离最短等。这个约束条件通常是用耗散或代价这一概念来概括。在本章当中我们的研究对象是最优化问题，这时问题可提为寻找具有最小耗散的解。常见的搜索算法，例如深度优先、广度优先、广度优先双向搜索和最小耗散优先等算法都可以用来求解最优化问题。

3.3.2 搜索算法的一般模式

一般说来，搜索算法要求记录完整的搜索空间，即保留所有生成的结点。这样做有两个目的：状态判重（复）和输出解路径。因为结点是表示状态特征及其关联方式的基本信息单元，所以每个结点都包含了状态的信息，可以用来表明那些状态已经到达，避免重复生成、扩展同一状态的结点。另一方面，结点也包含了应用规则的信息，搜索结束的时候，只要在这些信息的引导下“顺藤摸瓜”，就可以从目标结点逆推到初始结点，并得到从初始状态到目标状态的规则序列，从而构造解路径。

我们知道，如果图是连通的，那么图的遍历将产生一棵由图中的顶点构成的遍历生成树（例如深度优先生成树或广度优先生成树）；类似的，因为状态网络一般是连通的，如果只有一个初始结点，那么图搜索算法将产生一棵搜索树，即结点之间的联系是树结构的。

但是，为了方便描述选择待扩展的结点和生成新结点的过程，我们同时将结点组织成两张表（即结点之间存在两种联系方式：树和表），称为 CLOSED 表和 OPEN 表。CLOSED 表中存放已经扩展的结点；OPEN 表中存在已经生成但尚未扩展的结点。搜索开始之前，CLOSED 表是空的，而 OPEN 表中只有初始结点。

有了 CLOSED 表和 OPEN 表的概念，搜索的过程就可以描述为：从 OPEN 表中选择一个结点 u，将结点 u 从 OPEN 表删除，加入到 CLOSED 表，然后扩展结点 u 得到的新结点 $v_1, \cdots, v_k$（这些顶点的状态应当与已经生成的结点的状态是不同的），将这些结点加入 OPEN 表。

由第二章“图的遍历”，我们知道图的遍历方式可以有一种统一的表述，因此我们也可以仿照图的遍历方式得到图搜索算法的一般表述。因为结点本身已有状态判重的功能，所以我们不再设置数组 visited。

```
procedure graph_search;
begin
```

```
  CLOSED 表初始化为空;
  OPEN 表初始化为只含初始结点;
  while OPEN 表非空 do
  begin
    取 OPEN 表中的一个结点 u;
    从 OPEN 表删除 u;
    u 进入 CLOSED 表;
    对扩展结点 u 得到的每个新结点 vi do
      if vi的状态与 CLOSED 表和 OPEN 表中的结点的状态都不相同 then
        vi进入 OPEN 表;
  end;
end;
```

3.3.3 两种基本的搜索算法

与图的两种遍历方式相对应，有两种基本的搜索算法：深度优先搜索和广度优先搜索。有了搜索树的概念，我们就可以定义结点的深度为结点在搜索树中的层次。深度优先搜索，就是优先扩展尚未扩展且具有最大深度的结点；广度优先搜索，就是在扩展完第 k 层的结点之后，才扩展第 k+1 层的结点。

我们知道，图的深度优先遍历与广度优先遍历的差别仅在于存储访问过但尚未检查的顶点的数据结构；类似地，深度优先搜索和广度优先搜索的差别仅在于 OPEN 表的组织：如果 OPEN 表是后进先出的，则上面的算法就是深度优先搜索算法；如果 OPEN 表是先进先出的，那么上面的算法就成了广度优先搜索算法。

在第二章我们介绍过回溯算法。大家研究一下回溯算法就会发现，如果将回溯到达的状态用规则联系起来，并用图的方式表示，实际上形成了一棵状态树；按回溯的每一步到达状态的次序访问这棵树的状态，就是这棵树的深度优先遍历。换句话说，回溯实际上也生成了一棵深度优先遍历树。那么，回溯与深度优先搜索有什么联系呢？

从本质上说，回溯和深度优先搜索几乎就是一回事。不过仔细分析一下，两者还是有点差别的。

首先，回溯方式不保留完整的树结构，只记住当前工作的一条路径，回溯就是对这条路径进行修正；而深度优先搜索则记下完整的搜索树，搜索树同时起到记录解路径和状态判重的作用。因此，回溯方式明显比深度优先搜索节省空间，但同时也失去了状态判重的依据，对某一类问题（例如走迷宫问题，如果迷宫存在起点可达的回路），会重复到达同一状态，从而陷入死循环，在这种情况下，一般要人为设置前进的最大深度，一旦超过这个深度就强行回溯；而深度优先搜索虽然多占用空间，但是较适合于这类问题的情况。对另一类问题（例如八皇后问题），不同的规则序列决定不同的状态，于是不存在重复到达同一状态的问题，用回溯算法求解就具有很大的优势了。

其次，回溯方式要求能够按照一种事先确定好的某种固定排序依次调用规则。即对于每一层 i 事先都确定好一个规则的序列 $Ri_1, Ri_2, \cdots, Ri_n$，假设第 i 层的当前规则是 Ri_k，如果到第 i+1 层发现所有的规则都不合适，就回溯到第 i 层，若 $k<n$，则继续尝试规则 Ri_{k+1}，否则再回溯到第 i-1 层……。这种对规则的有序性的要求的原因是，回溯方式不保留完整的树结构，即每一层所用过的规则，如果放弃了对规则的有序性的要求，就无从判断那些规则用过，那些规则没用过，那些层次的规则已经全部尝试过了。深度优先搜索则没有对规则有序性的

要求，因为深度优先搜索保留全部生成的结点，而结点就包含用过的规则的信息。

于是我们看到，回溯和深度优先搜索非常相似，其区别的根源仅在于是否保留搜索树。其实纯粹的回溯和深度优先搜索各有利弊，因此在实际程序设计当中，是很少有人使用纯粹的回溯和深度优先搜索算法的；通常的做法是把两者的优点结合起来，形成一种新的算法模式。这种新的算法模式吸收了回溯算法节省空间和深度优先搜索算法可以状态判重的优点，一方面，它不保留完整的树结构，只记住当前工作的一条路径，同时在搜索的每一层确定调用规则的序列；另一方面，它采用类似深度优先遍历算法的方式，用标志的方法记录访问过的状态（当然，如果不同的规则序列决定不同的状态，就可以省去这个标志了）。我们通常将这种新的算法模式也称为深度优先搜索算法，并以此取代旧的深度优先搜索算法。以后我们再提到深度优先搜索算法，指的是新的算法。所以常说“深度优先搜索比广度优先搜索节省空间”，就是指这个意思。

3.3.4 搜索算法的优化手段

对大部分的问题，搜索树都是非常庞大的，结点的数目高居在指数级。如果解路径很长，计算机便会承受不住“组合爆炸”的压力。为了缓解这一压力，搜索算法的一种基本的优化是“剪枝”。在生成一个结点的时候，如果能够通过一些信息，说明继续扩展该结点不可能到达目标结点，则可以将搜索树在该结点以下的分枝全部剪去，从而达到减少搜索范围、提高搜索速度的目的。

带剪枝优化的搜索算法的一般描述是：

```
procedure graph_search_with_branch_cutting;
begin
  CLOSED 表初始化为空;
  OPEN 表初始化为只含初始结点;
  while OPEN 表非空 do
  begin
    取 OPEN 表中的一个结点 u;
    从 OPEN 表删除 u;
    u 进入 CLOSED 表;
    对扩展结点 u 得到的每个新结点 vi do
      if vi 不满足剪枝条件 then
        if vi 的状态与 CLOSED 表和 OPEN 表中的结点的状态都不相同 then
          vi 进入 OPEN 表;
  end;
end;
```

关于搜索剪枝的例题，请参考第四章第十节“移棋子”。

另一种基本的优化手段是“双向搜索”。双向搜索，顾名思义就是从初始状态和目标状态同时出发，寻找结果，一旦双方接上头，解路径就找到了。于是我们定义：

正向搜索：从初始结点向目标结点方向搜索；

反向搜索：从目标结点向初始结点方向搜索。

使用双向搜索有一个条件：每一条规则都是可逆的（即若状态 s_1 应用规则 r_1 产生状态 s_2，则必定存在规则 r_2 使 s_2 应用 r_2 产生 s_1），至少解路径上的规则都是可逆的。

一般来说，双向搜索的一个方向能对另一个方向可以使用的规则施加一定的限制，从而起到抑制搜索树的增长速度，提高搜索效率的作用。

至于双向搜索的具体算法模式，对不同的目标和具体问题是有所不同的。后面我们将介绍广度优先双向搜索，从中大家可以对此有所体会。

3.3.5 应用搜索算法求解最优化问题

从搜索的角度看，求解最优化问题就是寻找具有最小耗散（即代价）的解序列。当对解加上最小耗散限制之后，搜索就产生了一些新的概念，同时搜索的过程也发生了一些变化。

状态网络的边的耗散值相当于使用规则的代价（是一个正数）。路径的耗散值等于连接这条路径的各结点间所有边的耗散值总和。结点的耗散值是指从初始结点到该结点的当前路径的耗散值。对任意一个图，当从初始结点到目标结点有一条路径存在时，如果搜索算法总是在找到一条从初始结点到目标结点的最优路径上结束，则称该算法是可接纳的。

同时搜索算法也发生了一些变化。最明显的就是在生成新结点时，还要计算新结点的耗散值；再者，因为每个结点都附上了一个耗散值，有可能出现反复生成同一结点的情况（当然，每次生成时该结点的耗散值都是不同的），因此在生成新结点的同时，也可能发现在 CLOSED 表中的某个结点的耗散值不够优，要从 CLOSED 表中删除以待重新生成。

因此应用搜索方式求解最优化问题的一般算法描述是：

```
procedure graph_search_of_optimality;
begin
  CLOSED 表初始化为空;
  OPEN 表初始化为只含初始结点;
  while OPEN 表非空 do
  begin
    取 OPEN 表中的一个结点 u;
    从 OPEN 表删除 u;
    u 进入 CLOSED 表;
    对扩展结点 u 得到的每个新结点 vi do
    begin
      计算 vi 的耗散值;
      do case
        case vi 的状态与 CLOSED 表和 OPEN 表中的结点的状态都不相同:
          vi 进入 OPEN 表;
        case vi 的状态与 CLOSED 表中的某个结点 w 的状态相同但耗散值更优:
          从 CLOSED 表中删除 w 且 vi 进入 OPEN 表;
        case vi 的状态与 OPEN 表中的某个结点 w 的状态相同但耗散值更优:
          修改 w 的耗散值和到达 w 的路径为 vi 的耗散值和到达 vi 的路径;
      end;
    end;
  end;
end;
```

一般来说，应用深度优先搜索和广度优先搜索都要搜索完整个状态空间，才能确定目标结点的最小耗散。搜索开始前，设最优解的最小耗散是+∞；在搜索的过程中，每次生成目

标结点时，都将目标结点的耗散值与最优解的最小耗散比较，如果目标结点的耗散值更优，则更新最优解的最小耗散和解路径。在搜索完整个状态空间后，我们就获得了最优解的最小耗散和相应的解路径。

3.3.6 广度优先搜索求单位耗散问题最优解的算法

在单位耗散，即结点耗散值等于结点深度的情况下，广度优先搜索第一次生成目标结点就达到目标结点的最小耗散。因此广度优先搜索可以直接用于求解单位耗散的最优化问题，下面给出其算法模式：

```
procedure breadth_first_search_of_optimality;
begin
  CLOSED 表初始化为空;
  OPEN 队列初始化为只含初始结点;
  while OPEN 队列非空 do
  begin
    取 OPEN 队头结点 u;
    u 出队 OPEN;
    u 进入 CLOSED 表;
    对扩展结点 u 得到的每个新结点 vi do
    begin
      计算 vi 的耗散值;
      if vi 的状态=目标状态 then
      begin
        输出结点 vi 的解路径;
        exit;
      end;
      if vi 的状态与 CLOSED 表和 OPEN 队列中的结点的状态都不相同 then
        vi 入队 OPEN;
    end;
  end;
  宣告问题无解;
end;
```

例题 3.3.1：八数码问题

在 3×3 的九宫格棋盘上摆有八个棋子，每个棋子都刻有 1-8 中的某一个数码。棋盘中留有一个空格，允许其周围的某一个棋子向空格移动，这样通过移动棋子解可以不断改变棋盘的格局。给定一种初始棋盘格局和一个目标棋盘格局（例如下图所示），求一个移动棋子的序列，实现从初始格局到目标格局的转变，并使移动棋子的步数最少。

初始格局

8	6	7
2	5	4
3		1

→

1	2	3
4	5	6
7	8	

目标格局

分析：

定义结点的耗散值为移动棋子的步数。因为应用规则一次移动棋子的步数就加一，因此

结点的耗散值和结点的深度相同，可以运用广度优先搜索求解。

需要注意的要点是，像八数码这样的问题，如果采用广度优先搜索，搜索树的增长速度是很快的。为了减少判重的工作量，必须寻找一种较好的判重方法。

考虑全排列的地址映射公式，我们可以将数字 0 到 8 的排列和 0 到 362779 之间的数字一一对应起来。因此可以建立一个大数组，其下标就是排列对应的数字，如果某个排列已经存在了，数组中对应下标的值就为 1，不存在就为 0。这样新结点判重时只要查一下数组就可以了，但是从存储开销来看，要存储全部结点的状态似乎太大（共 9!=362780 种）。怎么办呢？

其实，结点只有两个状态，“存在”和“不存在”，我们用一个字节(byte)保存状态是很浪费的，实际上，用一位(bit)存储一个结点的状态就可以了，这样计算下来，全部结点状态只要用 9!/8=40360 就可以保存下来了。这样判重的速度就大大加快了。

下面是广度优先搜索求解八数码问题的程序。

程序名：ex3_3_1.dpr

```
//广度优先搜索求解八数码问题
program breadth_first_search_of_eight_puzzle;

{$APPTYPE console}

const
  grid=9; //棋盘格子数
  maxnode=45360; //=9!/8(每字节 bit 数)
  bits:array[0..7] of byte=(1,2,4,8,16,32,64,128); //判重时的辅助数组
  power:array[1..grid-1] of integer= //计算排列序号时的辅助数组
    (40320,5040,720,120,24,6,2,1);
  way:array[1..grid,0..4] of byte= //每个位置可以移动的方案
    ((2,2,4,0,0),(3,1,3,5,0),(2,2,6,0,0),
     (3,1,5,7,0),(4,2,4,6,8),(3,3,5,9,0),
     (2,4,8,0,0),(3,5,7,9,0),(2,6,8,0,0));
type
  dtype=string[grid]; //状态的字符串存储方式
  link=^datatype; //结点指针类型
  datatype=record //结点的类型
    d:integer; //棋盘状态
    n:link; //指向下一个结点的指针
    l:link; //指向父结点的指针
  end;
  checktype=array[0..maxnode-1] of byte; //判重数组类型
var
  root:link; //根结点
  closed,open:link; //CLOSED 表和 OPEN 表指针
  check:checktype; //判重数组
  goal:integer; //目标状态
//将状态的字符串表示 data 转换成整数
function datatonum(var data:dtype):integer;
var
  i,j:byte;
  x:integer;
begin
  x:=0;
  for i:=1 to grid-1 do
  for j:=i+1 to grid do
    if data[i]>data[j] then x:=x+power[i];
  datatonum:=x;
end;
```

```
//将状态的整数表示 num 转换成字符串
function numtodata(num:integer):dtype;
var
  data:array[0..grid] of byte;
  i,j:byte;
begin
  data[grid]:=0;
  data[0]:=grid;
  for i:=1 to grid-1 do
  begin
    data[i]:=num div power[i];
    num:=num mod power[i];
  end;
  for i:=grid downto 1 do
  for j:=i-1 downto 1 do
    if data[i]>=data[j] then inc(data[i]);
  numtodata:=dtype(data);
end;
//输入数据和初始化
procedure init;
var
  i:byte;
  first,second:dtype;
begin
  readln(first);
  readln(second);
  for i:=1 to grid do
  begin
    first[i]:=chr(ord(first[i])-48);
    second[i]:=chr(ord(second[i])-48);
  end;

  fillchar(check,sizeof(check),false);
  new(root);
  with root^ do
  begin
    d:=datatonum(first);
    n:=nil;
    l:=nil;
    check[d div 8]:=check[d div 8] or bits[d mod 8];
  end;
  open:=root;
  closed:=root;
  goal:=datatonum(second);
end;
//输出移动的步数和方案
procedure print;
var
  po:link;
  result:array[1..100] of dtype;
  step,i,j:byte;
  te:dtype;
begin
  po:=open;
  step:=0;
  repeat
    inc(step); result[step]:=numtodata(po^.d);
    po:=po^.l;
```

```
  until po=nil;
  for i:=1 to step div 2 do
  begin
    te:=result[i];
    result[i]:=result[step-i+1];
    result[step-i+1]:=te;
  end;

  writeln('total step=',step-1);
  for i:=1 to step do
  begin
    writeln('step ',i-1:2,':');
    for j:=1 to 3 do write(ord(result[i][j]));
    writeln;
    for j:=4 to 6 do write(ord(result[i][j]));
    writeln;
    for j:=7 to 9 do write(ord(result[i][j]));
    writeln;
  end;
  readln;
  halt;
end;
//判断状态 num 是否重复
function checkit(num:integer):boolean;
var
  x:integer;
  y:byte;
begin
  x:=num div 8;
  y:=num mod 8;
  if check[x] and bits[y]=0 then
  begin
    check[x]:=check[x] or bits[y];
    checkit:=true;
  end
  else
    checkit:=false;
end;
//广度优先搜索
procedure search;
var
  i,zero:byte;
  num:integer;
  data:dtype;
begin
  if closed^.d=goal then print;
  repeat
    data:=numtodata(closed^.d);
    zero:=pos(#0,data);
    for i:=1 to way[zero,0] do
    begin
      data[zero]:=data[way[zero,i]];
      data[way[zero,i]]:=#0;
      num:=datatonum(data);
      if checkit(num) then
      begin
        if open^.n=nil then
        begin
```

```
            new(open^.n);
            open^.n^.n:=nil
          end;
          open:=open^.n;
          with open^ do
          begin
            d:=num;
            l:=closed
          end;
          if num=goal then print;
        end;
        data[way[zero,i]]:=data[zero];
        data[zero]:=#0;
      end;
      closed:=closed^.n;
    until closed=nil;
  end;

  begin
    init;
    search;
  end.
```

3.3.7 广度优先双向搜索

搜索算法的一种基本的优化是双向搜索。将双向搜索的思想应用到广度优先搜索，我们得到广度优先双向搜索，它是对广度优先搜索的一种改进，同样可以直接用于求解单位耗散的最优化问题。使用广度优先双向搜索的条件除了单位耗散，还有每一条规则都是可逆的，至少最优解要用到的规则都是可逆的。

由于广度优先双向搜索是广度优先搜索的改进，因此我们必须保证找到的第一个解就是最优解，否则在搜索最优解的算法上，广度优先双向搜索就不比广度优先搜索有优势了。

首先，如何扩展结点是第一个问题。我们可以考虑的无非下面几种方式：

(1) 两边同步扩展；

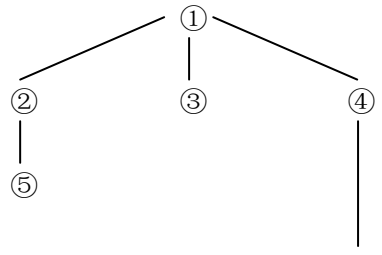
(2) 扩展完一边，再扩展另一边；

(3) 一边扩展一层结点，另一边再扩展一层；

(4) 扩展完一层后，选择结点个数较少的那个方向再扩展一层。

四种方式，明显可以看出(2)就是单向搜索，剩下的三个，(1)比较容易，但是不知是否可接纳的。

我们发现方式(1)对于某些特殊情况可能会找不着最优解。例如下图所示的搜索树，要求出结点①到结点⑨之间的最短路径。按照方式(1)，正方向扩展②得到新结点⑤，反方向扩展⑥也得到⑤，这样算法就会认为最优解是①-②-⑤-⑥-⑨。但实际上，解①-④-⑧-⑨比①-②-⑤-⑥-⑨更优，方式(1)不正确。



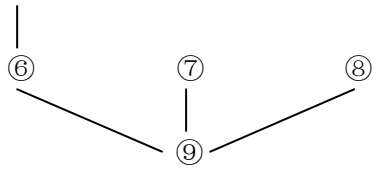


图 3.3.2 广度优先双向搜索示意图

因此只剩下(3)和(4)两种可能了。事实上，两者都是可接纳的，因为每边都扩展了整层的搜索树，所以从初始结点到目标结点的每一条最优路径都不会放过。那么哪种方式更好一些呢？因为我们提出广度优先双向搜索的初衷是尽可能抑制搜索树增长的速度，假设正反两个方向结点的生成速度是不平衡的，如果机械地按照“正方向扩展一层，再反方向扩展一层”，两个方向依层交替扩展的方法的话，就会造成某个方向上的搜索树增长过快，导致广度优先双向搜索的效率下降。因此，(4)更有利于克服两个方向结点的生成速度不平衡的状况，效率一般比(3)要高。

如何判断两个方向的搜索接上头呢？只要我们在生成结点的同时，判断该结点是否出现在相反方向的搜索树即可。如果我们使用标志数组 visited 来进行结点判重的话，那么这个工作就可以方便高效地完成：在结点所在的方向出现重复，新结点就该被去掉；在另一个方向上出现重复，就找到了最优解。

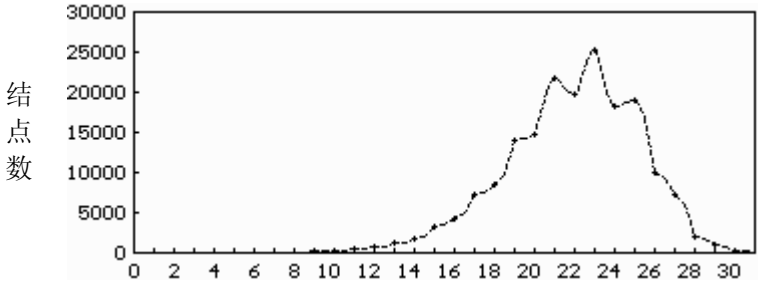
广度优先双向搜索的算法要求两张 CLOSED 表和两张 OPEN 表，我们扩展表的下标表示搜索方向。用 0 表示正方向，用 1 表示反方向，例如 CLOSED[0]表示正方向的 CLOSED 表。另外，我们用 node[0..1]表示两个方向搜索树最底层的结点数，用 new_node[0..1]表示搜索过程中两个方向最新扩展的一层的结点数。则广度优先双向搜索的算法可描述为：

```
procedure two_way_breadth_first_search_of_optimality;
begin
  CLOSED[0]、CLOSED[1]表初始化为空;
  OPEN[0]队列初始化为只含初始结点，OPEN[1]队列初始化为只含目标结点;
  node[0]←初始结点数，node[1]←目标结点数;
  while node[0]>0 or node[1]>0 do
  begin
    if node[0]<node[1] then d←0 else d←1;
    if node[d]=0 then d←1-d;
    new_node←0;
    for i←1 to node[d] do
    begin
      取 OPEN[d]队头结点 u;
      u 出队 OPEN[d];
      u 进入 CLOSED[d]表;
      对扩展结点 u 得到的每个新结点 vi do
      begin
        计算 vi 的耗散值;
        if vi 的状态与 CLOSED[1-d]表或 OPEN[1-d]队列中的某个结点的状态相同 then
        begin
          顺序输出正方向从起始结点到 vi 的路径;
          逆序输出反方向从 vi 到目标结点的路径;
          exit;
        end;
```

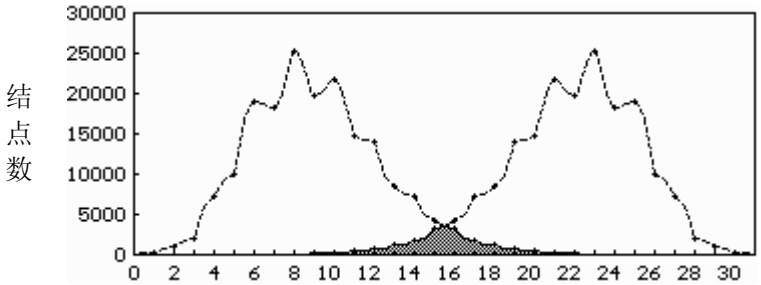
```
        if vi的状态与CLOSED[d]表和OPEN[d]队列中的结点的状态都不相同 then
        begin
          vi入队OPEN[d];
          inc(new_node);
        end;
      end;
    end;
    node[d]←new_node;
  end;
  宣告问题无解;
end;
```

广度优先双向搜索与广度优先搜索相比，优势有多大呢？从理论上说，如果每一结点可扩展的子结点数为 M，不计约束条件，以完全 M 叉树计算，广度优先搜索必须在搜索树上扩展完 L-1 层的所有结点，扩展的结点数为 $M(M^L-1)/(M-1)$。对于广度优先双向搜索来说，正向搜索扩展的结点数为 $M(M^Y-1)/(M-1)$，反向搜索扩展的结点数为 $M(M^{L-Y}-1)/(M-1)$，广度优先双向搜索扩展的结点总数为 $M(M^Y+M^{L-Y}-1)/(M-1)$（其中 L 是最优解路径长度，Y=(M+1) div 2，两个方向的搜索在 Y 层生成同一子结点）。设 L 为偶数(L=2*Y)，广度优先双向搜索扩展的结点数约是广度优先搜索的 $2/(M^{L/2}+1)*100\%$，相对减少 $(M^{L/2}-1)/(M^{L/2}+1)*100\%$。

那么，广度优先双向搜索实际上比广度优先搜索好多少呢？我们用数据来说明。用一个九位数码来简记棋盘格局，则例题当中的图所示的初始格局可表示为 867254301，目标格局可表示为 123456780。要实现这个初始格局到目标格局的转变需要 31 步。我们分别用广度优先搜索和广度优先双向搜索求解这个问题的实例，下面是两种算法的搜索量图：



广度优先搜索的搜索量



广度优先双向搜索的搜索量（图中阴影部分所示）

图 3.3.3 两种搜索算法的搜索量

从中可以清楚看出，在八数码问题这样的结点数增长很快的问题上，广度优先双向搜索的效率还是很高的。

下面给出广度优先双向搜索求解八数码问题的程序：

程序名：ex3_3_2.dpr

```
//广度优先双向搜索求解八数码问题
program two_way_breadth_first_search_of_eight_puzzle;

{$APPTYPE console}

const
  grid=9; //棋盘格子数
  maxnode=45360; //=9!/8(每字节bit数)
  bits:array[0..7] of byte=(1,2,4,8,16,32,64,128); //判重时的辅助数组
  power:array[1..grid-1] of integer= //计算排列序号时的辅助数组
    (40320,5040,720,120,24,6,2,1);
  way:array[1..grid,0..4] of byte= //每个位置可以移动的方案
    ((2,2,4,0,0),(3,1,3,5,0),(2,2,6,0,0),
     (3,1,5,7,0),(4,2,4,6,8),(3,3,5,9,0),
     (2,4,8,0,0),(3,5,7,9,0),(2,6,8,0,0));
type
  dtype=string[grid]; //状态的字符串存储方式
  link=^datatype; //结点指针类型
  datatype=record //结点的类型
    d:array[0..1] of integer; //棋盘状态
    n:link; //指向下一个结点的指针
    l:array[0..1] of link; //指向父结点的指针
  end;
  checktype=array[0..maxnode-1] of byte; //判重数组类型
var
  root:link; //根结点
  closed,open:array[0..1] of link; //双向CLOSED表和OPEN表指针
  check:array[0..1] of checktype; //双向判重数组
//将状态的字符串表示data转换成整数
function datatonum(var data:dtype):integer;
var
  i,j:byte;
  x:integer;
begin
  x:=0;
  for i:=1 to grid-1 do
  for j:=i+1 to grid do
    if data[i]>data[j] then x:=x+power[i];
  datatonum:=x;
end;
//将状态的整数表示num转换成字符串
function numtodata(num:integer):dtype;
var
  data:array[0..grid] of byte;
  i,j:byte;
begin
  data[grid]:=0;
  data[0]:=grid;
  for i:=1 to grid-1 do
  begin
    data[i]:=num div power[i];
    num:=num mod power[i];
  end;
  for i:=grid downto 1 do
  for j:=i-1 downto 1 do
```

```
    if data[i]>=data[j] then inc(data[i]);
  numtodata:=dtype(data);
end;
//输入数据和初始化
procedure init;
var
  i:byte;
  start,goal:dtype;
begin
  readln(start);
  readln(goal);
  for i:=1 to grid do
  begin
    start[i]:=chr(ord(start[i])-48);
    goal[i]:=chr(ord(goal[i])-48);
  end;

  fillchar(check[0],sizeof(check[0]),false);
  fillchar(check[1],sizeof(check[1]),false);
  new(root);
  with root^ do
  begin
    d[0]:=datatonum(start); d[1]:=datatonum(goal);
    n:=nil;
    l[0]:=nil;
    l[1]:=nil;
    check[0,d[0] div 8]:=check[0,d[0] div 8] or bits[d[0] mod 8];
    check[1,d[1] div 8]:=check[1,d[1] div 8] or bits[d[1] mod 8];
  end;
  open[0]:=root;
  open[1]:=root;
  closed[0]:=root;
  closed[1]:=root;
end;
//从方向 which 的状态 num 输出移动的步数和方案
procedure print(which:byte;num:integer);
var
  po:array[0..1] of link;
  result:array[1..100] of dtype;
  step,i,j:byte;
  te:dtype;
begin
  po[0]:=closed[0];
  po[1]:=closed[1];
  repeat
    if po[1-which]^.d[1-which]=num then break;
    po[1-which]:=po[1-which]^.n;
  until po[1-which]=nil;
  step:=0;
  repeat
    inc(step);
    result[step]:=numtodata(po[0]^.d[0]);
    po[0]:=po[0]^.l[0];
  until po[0]=nil;
  for i:=1 to step div 2 do
  begin
    te:=result[i];
    result[i]:=result[step-i+1];
```

```
    result[step-i+1]:=te;
  end;
  repeat
    inc(step);
    result[step]:=numtodata(po[1]^.d[1]);
    po[1]:=po[1]^.l[1];
  until po[1]=nil;

  writeln('total step=',step-1);
  for i:=1 to step do
  begin
    writeln('step ',i-1:2,':');
    for j:=1 to 3 do write(ord(result[i][j]));
    writeln;
    for j:=4 to 6 do write(ord(result[i][j]));
    writeln;
    for j:=7 to 9 do write(ord(result[i][j]));
    writeln;
  end;
  readln;
  halt;
end;
//判断状态 num 在方向 which 是否重复
function checkit(which:byte;num:integer):boolean;
var
  x:integer;
  y:byte;
begin
  x:=num div 8;
  y:=num mod 8;
  if check[which,x] and bits[y]=0 then
  begin
    check[which,x]:=check[which,x] or bits[y];
    checkit:=true;
    if check[1-which,x] and bits[y]<>0 then print(which,num);
  end
  else
    checkit:=false;
end;
//广度优先双向搜索
procedure search;
var
  which,i,zero:byte;
  num,j:integer;
  node:array[0..1] of integer;
  newnode:integer;
  data:dtype;
begin
  node[0]:=1;
  node[1]:=1;
  repeat
    if node[0]<=node[1] then which:=0 else which:=1;
    if node[which]=0 then which:=1-which;
    newnode:=0;
    for j:=1 to node[which] do
    begin
      data:=numtodata(closed[which]^.d[which]);
      zero:=pos(#0,data);
```

```
        for i:=1 to way[zero,0] do
        begin
          data[zero]:=data[way[zero,i]];
          data[way[zero,i]]:=#0;
          num:=datatonum(data);
          if checkit(which,num) then
          begin
            inc(newnode);
            if open[which]^.n=nil then
            begin
              new(open[which]^.n);
              open[which]^.n^.n:=nil
            end;
            open[which]:=open[which]^.n;
            with open[which]^ do
            begin
              d[which]:=num;
              l[which]:=closed[which]
            end;
          end;
          data[way[zero,i]]:=data[zero];
          data[zero]:=#0;
        end;
        closed[which]:=closed[which]^.n;
      end;
      node[which]:=newnode;
    until (node[0]=0) and (node[1]=0);
  end;

  begin
    init;
    search;
  end.
```

3.3.8 最小耗散优先

广度优先搜索和广度优先双向搜索都只能求解单位耗散的最优化问题。对于非单位耗散的最优化问题，就需要借助于最小耗散优先算法。这个算法在选择待扩展的结点时，总是选取耗散值最小的结点。如果被选中的结点就是目标结点，则目标结点的耗散值一定达到最小，即求出问题的最优解。

最小耗散优先算法可以描述为：

```
procedure minimum_cost_first_search_of_optimality;
begin
  CLOSED 表初始化为空;
  OPEN 表初始化为只含初始结点;
  while OPEN 表非空 do
  begin
    取 OPEN 表中 g 值最小的结点 u;
    if u 的状态=目标状态 then
```

```
    begin
      输出结点 u 的解路径;
      exit;
    end;
    从 OPEN 表删除 u;
    u 进入 CLOSED 表;
    对扩展结点 u 得到的每个新结点 vi do
    begin
      计算 vi 的耗散值;
      do case
        case vi 的状态与 CLOSED 表和 OPEN 表中的结点的状态都不相同:
          vi 进入 OPEN 表;
        case vi 的状态与 OPEN 表中的某个结点 w 的状态相同但耗散值更优:
          修改 w 的耗散值和到达 w 的路径为 vi 的耗散值和到达 vi 的路径;
      end;
    end;
  end;
  宣告问题无解;
end;
```

在单位耗散的情况下，最小耗散优先算法就成了广度优先搜索算法。因此广度优先搜索算法可以看成是最小耗散优先算法的特例。

如果大家把最小耗散优先算法和求最短路的 dijkstra 算法进行比较，就会发现两者惊人地相似。若我们将距离的概念就定义成耗散值，则两个算法都是求从初始结点（状态）到目标结点（状态）最小距离；每次都是选择到初始状态的距离最短的结点（状态）进行扩展（松弛操作）。

如果再推广到动态规划的正向思维法，我们会发现更多的相似点：它们都是求最优解的方法（算法）；都是从一些已知的状态出发，一步一步向未知的状态空间推进；动态规划的状态和最小耗散优先算法的结点几乎是同一样东西;它们都是用一定的状态转移方程(规则）架起状态（结点）之间的桥梁；动态规划的正向思维法当中对状态最优值的计算次序，与最小耗散优先算法扩展结点的次序是一样的;动态规划的正向思维法对状态空间的遍历也生成了一棵状态树，与最小耗散优先算法生成的搜索树几乎一模一样。这些，甚至使我们怀疑，最小耗散优先和动态规划是不是同一样东西。

仔细琢磨，它们之间还是有许多不同的。首先，状态（结点）的排列顺序不同：动态规划是按状态顺序排列（数组方式）；最小耗散优先算法是按结点的耗散值大小排序（线性表方式，一般是链表形式）。其次，它们的状态转移（结点扩展）方式也不同：动态规划是由很多状态推出一个状态；最小耗散优先算法是从一个结点推出好多个结点。

尽管最小耗散优先和动态规划有着种种的区别，但是无可否认，搜索和动态规划本身是紧密联系的；许多最优化问题既可以使用搜索也可以使用动态规划来求解。最小耗散优先算法（广度优先搜索）的思想本质与动态规划是一致的。动态规划思维的应用是极为广泛的，可以渗透到很多领域、很多层面和很多具体的算法。

例题 3.3.3：

单源最短路径问题（见第二章“有向无环图上的单对顶点间的最短路径问题”）。

分析：

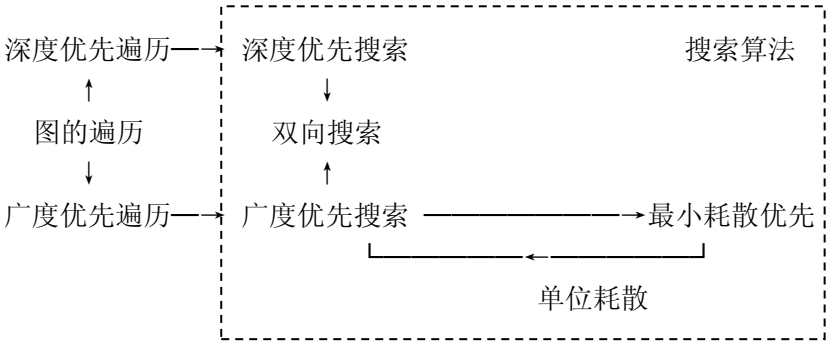
在第二章当中我们已经用动态规划解决了单源最短路径问题。现在我们换一个角度，从图的搜索来重新审视这个算法，重写求解单源最短路径问题的算法，以加深大家对动态规划的认识。

与第二章的表述类似，令 s[v]表示从顶点 a 到顶点 v 的最短距离。

```
procedure minimum_cost_first_search_of_shortest_distance;
begin
  s[a]←0,s[v]←+∞(v=1..n,v≠a);
  CLOSED 表初始化为空;
  OPEN 表初始化为只含初始顶点 a;
  while OPEN 表非空 do
  begin
    取 OPEN 表中 s[u]值最小的顶点 u;
    从 OPEN 表删除 u;
    u 进入 CLOSED 表;
    对 u 的每个邻接顶点 vi do
    begin
      if vi与 CLOSED 表和 OPEN 表中的顶点都不相同 then vi进入 OPEN 表;
      t←s[u]+w[u,vi];
      if t<s[vi] then s[vi]←t;
    end;
  end;
end;
```

3.3.9 搜索算法的总结

作为本节的最后，我们总结一下各种图搜索算法之间的关系。从最初的深度优先搜索、广度优先搜索，通过改进、融合产生了各种搜索算法，可以说是搜索算法的鼻祖，各种搜索算法都和它有深刻的联系。



我们来总结一下各种搜索算法的要点：

(1) 深度优先搜索（回溯）：对解无特殊要求，对空间要求不大。

(2) 广度优先搜索：适用于求单位耗散的最优化问题，对空间要求很大。

(3) 广度优先双向搜索：适用于求单位耗散的最优化问题，是广度优先搜索的改进版本。部分地克服了广度优先搜索的缺点（空间要求大）。

(4) 最小耗散优先：对解无特殊要求。广度优先搜索的推广，融合动态规划的思想，解决了

广度优先搜索对单位耗散的依赖。

对于不同的最优化问题，我们应该如何选择算法呢？这里提供一些可行的策略：

(1) 根据问题的要求和各种算法的适用范围，排除掉那些不可行的算法；

(2) 根据问题的规模，排除掉某些不适用于大规模问题的算法；

(3) 分析问题，看是否可用某些适用于特定性质问题的算法解决；

(4) 试图找出问题中隐藏的信息，设计剪枝条件；

(5) 如果进一步找到更多关于问题的知识，也许可以使用更精妙的算法（动态规划或数学方法）解决它。

第四章 国际大学生程序设计竞赛（ACM/ICPC）试题及分析

第一节 生成字符串

本题取材于 NWERC（西北欧）89C 题。

问题描述:

假设字符串只由字符‘0’,‘1’,‘*’组成，其中字符‘*’表示该字符可由字符‘0’或‘1’替代。

现有一些字符串，根据这些字符串生成所有可生成的字符串。

如：{10，*1，0* }可生成{10，01，11，00 }

{101，001，*01}可生成{ 101，001}

注意后一个例子中‘*01’并没有生成新的字符串。

输入格式:

从当前目录下的文本文件“STRINGS.DAT”读入数据。该文件的第一行是两个整数 m,n。（1≤m≤15,1≤n≤2500）m 表示字符串的长度，n 表示字符串的个数。两个整数之间由一个空格隔开。以下 n 行每行各有一个字符串。文件中各行的行首、行末没有多余的空格。

输出格式:

答案输出到当前目录下的文本文件“STRINGS.OUT”中，该文件只有一个整数 total，表示所能生成的字符串的个数。

输入输出举例:

输入文件：STRINGS.DAT

```
2 3
10
*1
0*
```

输出文件：STRINGS.OUT

```
4
```

测试数据:

strings1.dat

```
3 5
00*
*00
1**
0*0
```

111

strings2.dat
5 15
11*10
1100*
01*01
1001*
*1110
00100
000*0
11**0
01***
*0111
***0*
0*0*1
1**00
**10*
10*01

参考输出:

序号	测试文件名	参考输出	限时（秒）
1	STRINGS1.DAT	7	5”
2	STRINGS2.DAT	28	5”

算法分析:

本题是一道简单的生成字符串题目。由题目可知：字符串仅由字符‘0’,‘1’,‘*’组成；字符‘*’表示的字符可由字符‘0’或‘1’替代。

而生成字符串的过程可以分解为：（1）把输入的字符串按规则展开生成新的字符串；（2）统计所有生成的字符串的个数；（3）两个不同的字符串生成同一字符串时，统计时只作一个。

例如，输入{‘001’,‘00*’,‘1**’}，则生成{‘001’,‘000’,‘100’,‘101’,‘110’,‘111’}，共生成 6 个字符串。

为此，我们必须处理好两个问题：把各字符串展开；设置一个集合储存所生成的字符串。

1．把字符串展开

此时我们只需要把字符串中的字符‘*’依次转化为‘0’和‘1’即可。我们可利用递归实现：

```
procedure s_produced(s:string);
begin
    若 s 中不存在字符‘*’，则把 s 插入集合中，否则
    查找出字符‘*’的位置 T，
    把 s[T]置为‘0’，s_produced(s);
```

```
        把 s[T]置为 '1', s_produced(s);
    end;
```

2．设置集合存储字符串

显然，Pascal 语言中的集合类型的基数不超过 256，不能满足本题需要，我们只能另设集合类型。由于每个生成的字符串只由字符 '0'，'1' 组成，且长度相同，因此我们可以把每个字符串看成一个二进制数。我们可以用向量数组的形式存储字符串，，每个字符串的下标为其所对应的二进制数。

为此，我们需要把一个二进制数转化为十进制数：

```
function oct(s:string):integer; {s 仅由 '0','1' 组成}
begin
    t:=0;
    for I:=1 to length(s) do
        若 s[I]= '1' 则 t:=t+2 的 length(s)-I 次幂;
    ct:=t;
end;
```

如，若输入{ '001','00*','1**' },

```
    置 e[0..7]:=false;
则， '001' 生成 '001' =1,   置 e[1]:=true;
     '00*' 生成 '000' =0,   置 e[0]:=true;
           生成 '001' =1; {以有 e[1]=true}
     '1**' 生成 '100' =4,   置 e[4]:=true;
           生成 '101' =5,   置 e[5]:=true;
           生成 '110' =6,   置 e[6]:=true;
           生成 '111' =7,   置 e[7]:=true;
```

此时 e=（true,true,false,false,true,true,true,true），统计 'true' 的个数，可知共生成 6 个字符串。

程序分析：

```
 {$R-,V-,S-}
Program GDSOI98_1_Strings;

const
 nmax=16000;

type
 existtype=array[0..nmax] of boolean;

var
 finame,foname:string;
 fi,fo:text;
 m,n:integer;
 s:string;
```

```
 exist:^existtype; {   判断各数是否存在的向量数组  }
{$R-,V-,S-}
Program GDSOI98_1_Strings;

const
 nmax=16000;

type
 existtype=array[0..nmax] of boolean;

var
 finame,foname:string;
 fi,fo:text;
 m,n:integer;
 s:string;
 exist:^existtype; {   判断各数是否存在的向量数组  }
 p2:array[0..30] of longint;   {   2 的各次幂  }
 i,j:integer;


function oct_for_bin(s:string):longint; {   把一个二进制数转化为十进制数}
  var
   t:longint;
   i:integer;
 begin
  t:=0;
  for i:=length(s) downto 1 do
   if s[i]='1' then t:=t+p2[length(s)-i];
  oct_for_bin:=t;
 end;

procedure v_inserted(v:longint); { 把数 v 置为“存在”  }
 begin
  if exist^[v]=false then exist^[v]:=true;
 end;

procedure s_inserted(s:string);{   把字符串 s 所生成的数置为“存在”}
 var
  bin:boolean;
  value:longint;
  t:integer;
  i:integer;
 begin
  bin:=true;
```

```
  for i:=1 to length(s) do
   if bin then
    if s[i]='*' then
     begin bin:=false; t:=i; end;
  if bin then
    begin
     value:=oct_for_bin(s);
     v_inserted(value);
    end
   else
    begin
     s[t]:='0'; s_inserted(s);
     s[t]:='1'; s_inserted(s);
    end;
 end;
procedure total_output;                 {   输出总数    }
 var
  i:longint;
  total:longint;
 begin
  total:=0;
  for i:=0 to nmax do
   if exist^[i]=true then inc(total);
  write(fo,total);
 end;


begin
 new(exist);
 p2[0]:=1;
 for i:=1 to 30 do
  p2[i]:=p2[i-1]*2;
 for i:=0 to nmax do
  exist^[i]:=false;

 finame:='strings.dat';
 assign(fi,finame);
 foname:='strings.out';
 assign(fo,foname);

 reset(fi);
 readln(fi,m,n);
 for i:=1 to n do
  begin
```

```
    readln(fi,s);
    s_inserted(s);
   end;
  close(fi);

  rewrite(fo);
  total_output;
  close(fo);
  dispose(exist);
 end.
```

第二节 模式识别的“中心”问题

本题取材于 ICPC（世界赛）96E 题。

问题描述：

模式识别的一个关键问题是判别图形的“中心”，当图形经过扫描仪扫描后，得到一个实数矩阵，我们首先要找到该图形的“中心”，然后才能开始识别。设实数矩阵由 m 行 n 列组成（n≤100，m≤100），所谓的中心（i，j）是使第 i 行上边元素（不包括第 i 行）的总和与第 i 行下边元素（不包括第 i 行）的总和之差的绝对值最小，而且第 j 列左边元素（不包括第 j 列）的总和与第 j 列右边元素（不包括第 j 列）的总和之差的绝对值最小。

现已知一扫描所得的实数矩阵，求其“中心”。若有多个“中心”，给出任意一个“中心”即可。

输入格式：

从键盘输入一个文本文件的文件名。该文件第一行有两个数 m、n，以下 m 行是实数矩阵，每行各有 n 个实数。在每一行中，数据之间只有一个空格。每行的行首、行末无多余空格。

输出格式：

结果输出到屏幕上：

Center=(xxx,yyy)　　xxx,yyy 分别表示中心的行和列

输入输出举例：

SAMPLE1.DAT

```
5 5
0.2 0.3 0.2 0.3 0.2
0.2 0.3 0.4 0.2 0.2
0.3 0.4 0.2 0.2 0.4
0.5 0.2 0.2 0.2 0.3
0.3 0.3 0.4 0.4 0.2
```

在屏幕上输出

```
Center=(3,3)
```

测试数据：

T101.dat

```
5 5
0.34 0.80 0.61 0.42 0.55
0.80 0.41 0.56 0.95 0.55
0.26 0.49 0.06 0.93 0.58
0.19 0.08 0.31 0.84 0.79
0.12 0.43 0.33 0.50 0.86
```

T102.dat

```
10 20
0.24 0.42 0.79 0.90 0.54 0.33 0.48 0.35 0.54 0.60 0.75 0.30 0.09 0.60 0.33 0.25 0.75
0.52 0.15 0.73
```

0.40 0.26 0.71 0.50 0.62 0.83 0.84 0.32 0.13 0.58 0.21 0.80 0.20 0.55 0.49 0.44 0.68 0.54 0.21 0.04
0.02 0.82 0.57 0.97 0.59 0.47 0.81 0.07 0.05 0.85 0.80 0.12 0.63 0.69 0.12 0.10 0.49 0.78 0.73 0.82
0.72 0.22 0.84 0.55 0.36 0.74 0.82 0.68 0.74 0.57 0.15 0.63 0.94 0.85 0.70 0.51 0.08 0.55 0.93 0.28
0.00 0.40 0.78 0.36 0.99 0.25 0.27 0.87 0.65 0.01 0.19 0.20 0.81 0.92 0.31 0.31 0.63 0.86 0.02 0.09
0.07 0.04 0.38 0.78 0.48 0.05 0.88 0.08 0.78 0.42 0.48 0.08 0.33 0.92 0.74 1.00 0.10 0.34 0.01 0.45
0.03 0.51 0.13 0.35 0.76 0.76 0.02 0.06 0.34 0.06 0.63 0.79 0.82 0.62 0.03 0.55 0.81 0.58 0.67 0.60
0.64 0.16 0.78 0.56 0.50 0.43 0.32 0.22 0.76 0.35 0.27 0.01 0.49 0.06 0.27 0.31 0.60 0.34 0.55 0.29
0.32 0.73 0.52 0.52 0.95 0.96 0.30 0.39 0.03 0.33 0.46 0.73 0.31 0.39 0.77 0.70 0.25 0.21 0.73 0.85
0.99 0.73 0.05 0.37 0.89 0.20 0.18 0.25 0.09 0.70 0.67 0.39 0.81 0.55 0.90 0.70 0.23 0.76 0.50 0.14

参考输出:

序号	测试文件名	中心
1	T101.DAT	Center=(3,4)
2	T102.DAT	Center=(5,11)

算法分析:

求矩阵的中心，即确定矩阵中心的行和列坐标，考虑到对称性，行坐标和列坐标的求法是类同的。下面是求行坐标的算法，求列坐标的算法就不再重复。

求行坐标采用枚举法，枚举出所有可能的行坐标 line，计算出 line 行上边元素和与下边元素和之差的绝对值 difference，difference 最小的行即为中心所在行。

枚举过程可以描述为：

```
min:=+∞;
for line:=1 to m do
begin
  求出 line 行上面元素与下面元素绝对值之差 difference;
  if min>difference then
    begin
      min:=difference;
      保存 line 作为矩阵中心所在行;
    end;
end;
```

例如，原题例子给出矩阵如下：

5 5
0.2 0.3 0.2 0.3 0.2
0.2 0.3 0.4 0.2 0.2

0.3 0.4 0.2 0.2 0.4
0.5 0.2 0.2 0.2 0.3
0.3 0.3 0.4 0.4 0.2
对该矩阵进行枚举可得下表：

line	1	2	3	4	5
difference	5.8	3.3	0.5	2.4	5.4

difference 最小的第 3 行即为中心所在行。

程序分析：

```
{$N+}
program GDKOI 98_1_ pattern_center;
const error=-(1e-6);{允许误差}
var n,m:integer;{原实数矩阵的行和列}
      center_of_line,center_of_column:integer;{所求中心的行和列}
      i,j,line,column:integer;
      sum_of_a_line,sum_of_a_column:array [1..100] of extended;{记录各行（列）的实数总和}
      InputFile,OutputFile:text;
      FileName:string;
      r,min,difference:extended;

begin

{从文件读入数据}
write('Input File Name:');
readln(FileName);
assign(InputFile,FileName);
reset(InputFile);
readln(InputFile,m,n);
fillchar(sum_of_a_line,sizeof(sum_of_a_line),0);
fillchar(sum_of_a_column,sizeof(sum_of_a_column),0);
for i:=1 to m do for j:=1 to n do
begin
   read(InputFile,r);
   sum_of_a_line[i]:=sum_of_a_line[i]+r;
   sum_of_a_column[j]:=sum_of_a_column[j]+r;
end;
close(InputFile);

{确定中心的行坐标}
min:=1e10;
for line:=1 to m do
begin
   {求出 line 行上面元素与下面元素绝对值之差}
   difference:=0;
   for j:=1 to line-1 do difference:=difference+sum_of_a_line[j];
```

```
    for j:=line+1 to m do difference:=difference-sum_of_a_line[j];
    difference:=abs(difference);
    {判断 line 行是否为中心所在行}
    if min>difference+error then
       begin
          min:=difference;
          center_of_line:=line;
       end;
end;

{确定中心的列坐标}
min:=1e10;
for column:=1 to n do
begin
    {求出 column 列左边元素与右边元素绝对值之差}
    difference:=0;
    for j:=1 to column-1 do difference:=difference+sum_of_a_column[j];
    for j:=column+1 to n do difference:=difference-sum_of_a_column[j];
    difference:=abs(difference);
    {判断 column 列是否为中心所在列}
    if min>difference+error then
       begin
          min:=difference;
          center_of_column:=column;
       end;
end;

{输出中心坐标}
writeln('Center=(',center_of_line,',',center_of_column,')');
end.
```

第三节 划分凸多边形

问题描述：

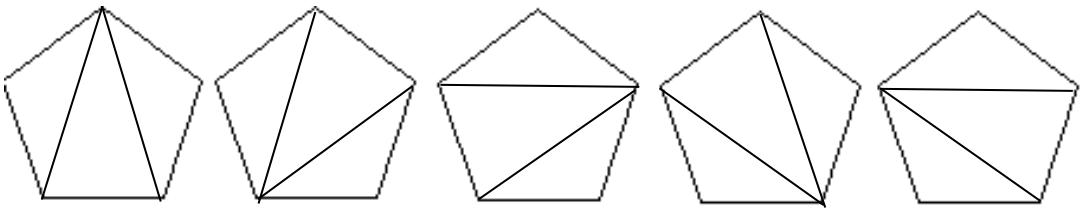
一个正凸 N 边形，可以用 N-3 条互不相交的对角线将正 N 边形分成 N-2 个三角形。

任务：从键盘输入 N（N<=20），在显示器上输出不同分法的总数（不需要输出各种分法）。

输入格式：N=5

输出格式：TOTAL=5

例如：当 N=5 时，共有 5 种分法。



测试数据和参考输出：

序号	N	总数
1	2	No answer
2	3	No answer
3	6	14
4	8	132

算法分析：

题目所求的是分法总数,不要求具体的分法.而 N 可以大到 20。因此，如用简单的搜索来求解,会耗时较多，而应想方设法找出 N 为不同值时,分法总数的变化规律.

把一个正凸 N 边形的各顶点按顺时针分别编上 1,2,……,N.顶点 1、顶点 N 和顶点 I（I∈[2,N-1]）能够构成一个三角形 S.这样,凸 N 边形被分成三角形 S 和一个 I 边形和一个 N+1-I 边形(I,N+1-I∈[2,N-1],我们假设 2,3 边形的分法总数都为 1).因此,凸 N 边形被分成 N-2 个三角形的分法总数 TOTAL[N]等于 I 分别取 2,3……N-1 时 I 边形的分法总数乘以 N+1-I 边形的分法总数的总和，其递推公式如下：

$$TOTAL[N]=\sum_{I=2}^{N-1} TOTAL[I]*TOTAL[N+1-I] \quad (N>=4, TOTAL[2]=1, TOTAL[3]=1)$$

根据这个式子,可以依次求出 TOTAL[4..N]的值.另外，由于没有一或二边形，三边形也不能被分割，所以程序要给出无解的信息。

根据上面分析，程序只需两重循环,其时间复杂度为 O(N^2)。

该算法可以简单描述为：

1. 读入 N;
2. 若 N<=3,则输出"NO ANSWER !"并退出,否则继续 3;
3. 置 TOTAL[2]:=1,TOTAL[3]:=1;

4. I 从 4 到 N 循环:
   4.1. 置 TOTAL[I]初值为 0;
   4.2. J 从 2 到 I-1 循环:
      4.2.1. inc(TOTAL[I],TOTAL[J]*TOTAL[I+1-J]);
5. 打印 TOTAL[N];

程序分析:

```
Program GDOI97_3;
var
  i,j,n:integer;
  total:array[2..20] of longint;
begin
{$i-}
  repeat                                              {读入 N}
    write('N=');
    readln(n);
  until (ioresult=0) and (n>=1) and (n<=20);
  if n<=3 then writeln('NO ANSWER !')      {若 N<=3,则打印"NO ANSWER !"}
  else
    begin
      total[2]:=1;                          {设 TOTAL[2],TOTAL[3]为 1}
      total[3]:=1;
      for i:=4 to n do                      {根据以上式子计算 TOTAL[4..N]}
        begin
          total[i]:=0;
          for j:=2 to i-1 do
            inc(total[i],total[j]*total[i+1-j]);
        end;
      writeln('TOTAL=',total[n]);           {打印 TOTAL[N]}
    end;
end.
```

第四节 防卫导弹

本题取材于 ICPC（世界赛）94B 题。

问题描述：

一种新型的防卫导弹可截击多个攻击导弹。它可以向前飞行，也可以用很快的速度向下飞行，可以毫无损伤地截击进攻导弹，但不可以向后或向上飞行。但有一个缺点，尽管它发射时可以达到任意高度，但它只能截击比它上次截击导弹时所处高度低或者高度相同的导弹。

现对这种新型防卫导弹进行测试，在每一次测试中，发射一系列的进攻导弹（这些导弹发射的间隔时间固定，飞行速度相同），该防卫导弹所能获得的信息包括个进攻导弹的高度，以及它们发射次序。

现要求编一程序，求在每次测试中，该防卫导弹最多能截击的进攻导弹数量，一个导弹能被截击应满足下列两个条件之一：

1. 它是该次测试中第一个被防卫导弹截击的导弹。
2. 它是在上一次被截击导弹的发射后发射，且高度不大于上一次被截击导弹的高度的导弹。

输入格式：

从当前目录下的文本文件“CATCHER.DAT”读入数据。该文件的第一行是一个整数 n（$1 \le n \le 4000$），表示本次测试中，发射的进攻导弹数，以下 n 行每行各有一个整数 h_i（$0 \le h_i \le 32767$），表示第 i 枚进攻导弹的高度。文件中各行的行首、行末无多余空格。输入文件中给出的导弹是按发射顺序排列的。

输出格式：

答案输出到当前目录下的文本文件“CATCHER.OUT”中，该文件第一行是一个整数 max，表示最多能截击的进攻导弹数，以下的 max 行每行各有一个整数，表示各个被截击的进攻导弹的编号（按被截击的先后顺序排列）。输出的答案可能不唯一，只要输出其中任一解即可。

输入输出举例：

输入文件：CATCHER.DAT

```
3
25
36
23
```

输出文件：CATCHER.OUT

```
2
1
3
```

测试数据：

CATCHER1.DAT

```
5
14
11
4
6
31
```

CATCHER2.DAT

```
10
15522
```

13267
7072
23658
4780
5093
954
21401
30560
5332

参考输出:

CATCHER1.OUT
3
1
2
3

CATCHER2.OUT
5
1
2
3
5
7

算法分析:

本题可采用搜索算法，但缺点是不能满足规模较大的测试数据。而另一较优的算法是动态规划法。我们注意到，若定义 max[I]表示当第 I 个导弹为首攻时所能击落的最大导弹数，则 max[I]=max{max[j]}+1}，(j=I+1,I+2,...,n，且导弹 I 的高度>=导弹 j 的高度)。由此可写出求 max 数组的算法：

```
for I:=n downto 1 do
  begin
    max[I]:=1;  next[I]:=0;
    for j:=I+1 to n do
      if 导弹 I 的高度>=导弹 j 的高度 then
        if max[I]<max[j]+1 then
          begin
             max[I]:=max[j]+1;
             next[I]:=j;
          end;
      if max[I]>maxnum then
        begin
           maxnum:=max[I];
           first:=I;
```

```
                    end;
          end
```

如，当 n=5，h=5，2，4，1，3 时，

有 max[5]=1, next[5]=0;

```
    max[4]=1,    next[4]=0;
    max[3]=max{max[j]}+1=2,  j=4,5,且 h[3]>=h[j];
        next[3]=5;
    max[2]=max{max[j]}+1=2,  j=3,4,5,且 h[2]>=h[j];
        next[2]=4;
    max[1]=max{max[j]}+1=3,  j=2,3,4,5,且 h[1]>=h[j];
        next[1]=3;
```

则 首攻为第 1 个导弹，依次为第 3，5 个导弹，最大击落导弹数为 3。

程序分析:

```
program GDOI98_6_catcher;

const
 maxn=4000;

var
 finame,foname:string;
 fi,fo:text;
 n:integer;                  {   导弹个数        }
 h,max,next:array[1..maxn] of integer; {   导弹的高度,最大数,下一导弹   }
 maxnum:integer;               {   最大攻击数          }
 head:integer;               {   首攻导弹        }
 i,j:integer;

begin
 finame:='catcher.dat';
 assign(fi,finame);
 foname:='catcher.out';
 assign(fo,foname);

{   读入数据    }
 reset(fi);
 readln(fi,n);
 for i:=1 to n do
  readln(fi,h[i]);
 close(fi);

{   求出 max 数组    }
 maxnum:=0;
 for i:=n downto 1 do
```

```
   begin
    max[i]:=1; next[i]:=0;
    for j:=i+1 to n do
     if h[i]>=h[j] then
      if max[i]<max[j]+1 then
       begin
        max[i]:=max[j]+1;
        next[i]:=j;
       end;
    if max[i]>maxnum then
     begin
      maxnum:=max[i];
      head:=i;
     end;
   end;

{   结果输出    }
  rewrite(fo);
  writeln(fo,maxnum);
  while head<>0 do
   begin
    writeln(fo,head);
    head:=next[head];
   end;
  close(fo);
 end.
```

第五节 邮票问题

本题取材于 ICPC（世界赛）95E 题。

问题描述：

给定一个信封，最多只允许粘贴 N（N≤100）张邮票，我们现在有 m（m≤100）种邮票，面值分别为：x_1，x_2，……，x_m 分，（x_i≤255，为正整数），并假设各种邮票都有足够多张。

要求计算所能获得的邮资最大范围。即求最大值 MAX，使在 1—MAX 之间的每一个邮资值都能得到。

例如，N=4，有 2 种邮票，面值分别为 1 分、4 分，于是可以得到 1—10 分和 12 分、13 分、16 分邮资，由于得不到 11 分和 15 分，所以邮资的最大范围 MAX=10。

输入格式：

从键盘输入一个文本文件的文件名。该文件第 1 行为最多粘贴的邮票张数 N；第 2 行为邮票种数 m；以下 m 行各有一个数字，表示邮票的面值 x_i。

输出格式：

1. 若最大范围为空，则在屏幕上输出 MAX=0
2. 若最大范围不为空，则把结果输出到屏幕上：

   MAX=xxx　　xxx 表示邮资最大范围

输入输出举例：

输入	输出
4 2 1 4	MAX=10

测试数据：

T701.dat

```
10
5
2
4
6
8
10
```

T702.dat

```
5
3
1
3
9
```

T703.dat
20
10
1
14
101
116
144
168
178
228
242
247

参考输出:

序号	测试文件名	MAX
1	T701.DAT	0
2	T702.DAT	25
3	T703.DAT	4868

算法分析:

本题可以看成是一个集合问题，即求在贴的邮票不多于 n 张的可满足条件的邮资集。集合问题的关键在于判定元素是否在集合中，对本题而言，是判断某个邮资是否在贴不多于 n 张邮票可满足。这个判断问题可以用递归的方法来解决。设当前考虑的邮资值为 max，最多允许贴 n 张邮票，记为(max,n)。如果首先贴一张面值为 x_i 的邮票，那么剩下的问题是(max-x_i,n-1)的问题。如果(max-x_i,n-1)可解，那么(max,n)问题也可解。

进一步，这个递归的算法可以转化为递推的算法来解决。设 pieces[value]表示邮资为 value 时所需最少的邮票数，pieces[value]=min{pieces[value-x_i]+1}。当 pieces[value]≤n 时，问题(max,n)可解，否则无解。

递推算法可以描述为：

```
for i:=1 to m do if value-x[i]>=0 then
  begin
    if pieces[value]=0 then pieces[value]:=pieces[value-x[i]]+1;
    if pieces[value]>pieces[value-x[i]]+1 then pieces[value]:=pieces[value-x[i]]+1;
  end;
```

例如：最多可贴 4 张邮票，邮票有两种，面值分别为 1 和 4。按照上述算法可得下表：

value	pieces[value]
0	**0**
1	**min{pieces[1-1]+1}=1**
2	**min{pieces[2-1]+1}=2**
3	**min{pieces[3-1]+1}=3**
4	**min{pieces[4-1]+1,pieces[4-4]+1}=1**
5	**min{pieces[5-1]+1,pieces[5-4]+1}=2**
6	**min{pieces[6-1]+1,pieces[6-4]+1}=3**
7	**min{pieces[7-1]+1,pieces[7-4]+1}=4**

8	**min{pieces[8-1]+1,pieces[8-4]+1}=2**
9	**min{pieces[9-1]+1,pieces[9-4]+1}=3**
10	**min{pieces[10-1]+1,pieces[10-4]+1}=4**
11	**min{pieces[11-1]+1,pieces[11-4]+1}=5(>4)**

程序分析：

```
{$R-,V-,S-}
program gdkoi98_7_ Stamps;
var x:array [1..255] of byte;              {各种邮票的面值}
      pieces:array [0..30000] of byte;{pieces[value]表示邮资为 value 时最少需要的邮票数}
      max,m,n,i,j:integer;
      filename:string;
      f:text;

begin
  {读入数据}
  write('Input File:');
  readln(filename);
  assign(f,filename);
  reset(f);
  readln(f,n);
  readln(f,m);
  for i:=1 to m do readln(f,x[i]);
  close(f);

  fillchar(pieces,sizeof(pieces),0);
  max:=0;
  repeat
    max:=max+1;
    {计算邮资为 max 最少需要的邮票张数}
    for i:=1 to m do if max-x[i]>=0 then
      begin
        if pieces[max]=0 then pieces[max]:=pieces[max-x[i]]+1;
        if pieces[max]>pieces[max-x[i]]+1 then pieces[max]:=pieces[max-x[i]]+1;
      end;
    {判断所需最少邮票张数是否超过 n 张}
    if (pieces[max]=0) or (pieces[max]>n) then
      begin
        writeln('MAX=',max-1);
        exit;
      end;
  until false;
end.
```

第六节 骨牌矩阵

本题取材于 ICPC（世界赛）91D 题。

问题描述：

多米诺骨牌是一个小正方形方块，每个骨牌都标有一个数字（0～6），现在有 28 组骨牌，每组两个，各组编号为 1～28，每组编号对应的两个骨牌数值如下。

骨牌组编号	骨牌	骨牌组编号	骨牌	骨牌组编号	骨牌	骨牌组编号	骨牌
1	0|0	8	1|1	15	2|3	22	3|6
2	0|1	9	1|2	16	2|4	23	4|4
3	0|2	10	1|3	17	2|5	24	4|5
4	0|3	11	1|4	18	2|6	25	4|6
5	0|4	12	1|5	19	3|3	26	5|5
6	0|5	13	1|6	20	3|4	27	5|6
7	0|6	14	2|2	21	3|5	28	6|6

现将这 28 组骨牌排成一个 7×8 矩阵，此时只能看到每个骨牌上的数字（0～6），而不能知道每组的组号。如左下图所示。请编程将每组骨牌分辨出来（见下图。图中数字为对应上图每组骨牌的编号）。骨牌摆放可旋转，例如第 9 组骨牌经旋转可得以下 4 种放法：

1|2、2|1、$\frac{1}{2}$、$\frac{2}{1}$。

7×8 骨牌矩阵

```
6 6 2 6 5 2 4 1
1 3 2 0 1 0 3 4
1 3 2 4 6 6 5 4
1 0 4 3 2 1 1 2
5 1 3 6 0 4 5 5
5 5 4 0 2 6 0 3
6 0 5 3 4 2 0 3
```

骨牌组编号矩阵

```
28 28 14  7 17 17 11 11
10 10 14  7  2  2 21 23
 8  4 16 25 25 13 21 23
 8  4 16 15 15 13  9  9
12 12 22 22  5  5 26 26
27 24 24  3  3 18  1 19
27  6  6 20 20 18  1 19
```

输入格式：

从键盘输入一个文本文件的文件名。该文件包含了一个 7 行×8 列的骨牌矩阵，每行有 8 个 0～6 的整数，每个整数之间用空格分开。每行的行首、行末无多余空格。

输出格式：

答案输出到一个文本文件中，文件名由键盘输入。

1. 若问题无解，则输出“-1”；

2. 若问题有解，则将所有可能的解输出，每个解之间用一个空行分开，最后输出解的总数。数字间用空格分开。

输入输出举例：

输入：SAMPLE2.DAT　　输出：SAMPLE2.OUT

```
5 4 3 6 5 3 4 6
0 6 0 1 2 3 1 1
3 2 6 5 0 4 2 0
5 3 6 2 3 2 0 6
4 0 4 1 0 0 4 1
5 2 2 4 4 1 6 5
5 5 3 6 1 2 3 1
```

```
6 20 20 27 27 19 25 25
6 18 2 2 3 19 8 8
21 18 28 17 3 16 16 7
21 4 28 17 15 15 5 7
24 4 11 11 1 1 5 12
24 14 14 23 23 13 13 12
26 26 22 22 9 9 10 10

1
```

测试数据：

T201.dat

```
4 1 2 0 6 3 0 4
4 1 3 0 1 3 1 4
3 2 5 4 2 4 3 5
5 3 4 1 1 5 2 2
6 1 2 5 5 1 4 6
2 0 6 0 3 5 2 3
0 5 0 0 6 6 6 6
```

T202.dat

```
4 6 1 6 3 4 3 0
5 0 1 4 2 1 2 6
3 6 0 2 4 3 4 4
2 6 0 5 2 6 5 0
1 4 4 5 6 3 3 0
5 0 2 0 1 2 2 6
1 1 3 1 3 5 5 5
```

T203.dat

```
6 0 1 4 4 6 6 0
1 0 2 3 1 5 1 2
3 5 3 2 3 5 0 3
4 4 3 1 1 3 1 1
2 4 2 6 4 6 3 0
5 4 0 4 0 0 2 5
5 5 2 2 6 5 6 6
```

参考输出：

T201.OUT

```
23 9 9 1 13 19 5 5
23 10 10 1 13 19 11 11
15 15 24 24 16 16 21 21
27 20 20 8 8 12 14 14
```

```
27 2 18 26 26 12 25 25
3 2 18 4 4 17 17 22
3 6 6 7 7 28 28 22

1

T202.OUT
25 25 13 13 20 20 4 4
6 6 11 11 9 9 18 18
15 28 1 17 16 22 24 5
15 28 1 17 16 22 24 5
12 23 23 27 27 19 19 7
12 3 3 2 2 14 14 7
8 8 10 10 21 21 26 26

25 25 13 13 20 20 4 4
6 6 11 11 9 9 18 18
15 28 1 16 16 22 24 5
15 28 1 17 17 22 24 5
12 23 23 27 27 19 19 7
12 3 3 2 2 14 14 7
8 8 10 10 21 21 26 26

2

T203.OUT
13 2 2 20 25 25 7 7
13 3 3 20 10 12 12 15
21 21 19 9 10 6 6 15
16 23 19 9 11 22 8 8
16 23 18 18 11 22 4 4
26 24 5 5 1 1 17 17
26 24 14 14 27 27 28 28

13 2 2 20 25 25 7 7
13 3 3 20 10 12 12 15
21 21 19 9 10 6 6 15
16 23 19 9 11 22 8 8
16 23 18 18 11 22 4 4
24 24 5 5 1 1 17 17
26 26 14 14 27 27 28 28

13 2 2 20 25 25 7 7
13 3 3 20 10 12 12 15
```

```
21 21 19 9 10 6 6 15
23 23 19 9 11 22 8 8
16 16 18 18 11 22 4 4
26 24 5 5 1 1 17 17
26 24 14 14 27 27 28 28

13 2 2 20 25 25 7 7
13 3 3 20 10 12 12 15
21 21 19 9 10 6 6 15
23 23 19 9 11 22 8 8
16 16 18 18 11 22 4 4
24 24 5 5 1 1 17 17
26 26 14 14 27 27 28 28

4
```

算法分析：

这题的算法比较简单，只需使用标准的回溯算法，而且由于本题的规模比较小，不需要对算法进行优化，便可满足时间上的要求。

在骨牌矩阵中，每组骨牌既可以横放也可以竖放。我们可以按从左到右从上到下的顺序对骨牌矩阵中的每个位置进行回溯，判断相应的骨牌组应该是横放还是竖放。最终目标就是要求二十八组骨牌组都能够不重复的排成一个 7*8 的矩阵而且每个骨牌上的数字和输入的骨牌矩阵一一对应，还要求每组骨牌不互相重叠。

在核心算法中的 find 过程就是用于回溯的递归过程，它先查找下一个还没放置骨牌的位置，较如已经超过矩阵的范围，而且所有的骨牌组已经放置好，则表示已经找到一个解，将它输出。否则，按四种方式尝试放置 1 个骨牌组，假如成功则尝试放置下一个骨牌组，假如不成功就回溯。

核心算法是递归放置一个骨牌组：

```
procedure find;
    begin
  查找下一个还没放置骨牌的位置(x,y);
     若没有，则表示已经找到一个解，输出，返回;
  把在(x,y)处的骨牌作为当前骨牌组的一个骨牌;
  把在(x+1,y)处的骨牌作为当前骨牌组的另一个骨牌;
  判断当前骨牌组是否未被使用，如果未被使用则递归放置下一个骨牌组;
  把在(x,y+1)处的骨牌作为当前骨牌组的另一个骨牌;
  判断当前骨牌组是否未被使用，如果未被使用则递归放置下一个骨牌组;
  两种尝试都失败，进行回溯;
 end;
```

程序分析：

```
Program GDKOI 98_2_Domino;{骨牌矩阵}
type tmap=array[1..7,1..8] of integer;
var pip,map:tmap;
    {pip 存放骨牌矩阵，map 存放骨牌对编号}
```

```
      i,j,count:integer;
      DominoPairs:array[0..6,0..6] of integer;
      {存放两个骨牌相对应的骨牌对编号}
      infile,outfile:text;
      CanUse:array[1..28] of boolean;
      {表示相应的骨牌对能否被使用}
      inname,outname:string;
{------------------------------------------------------------}
procedure find(a,b:integer);
{递归过程查找从坐标(a,b)开始的多米诺骨牌}
 var i,j:integer;
 begin
  i:=a;j:=b;
  while(i<=7)and(j<=8)and(map[i,j]>0) do
   begin
    inc(j);
    if j>8 then begin j:=1;inc(i);end;
   end;{查找下一个可以放的位置}
  if i>7 then
   {已经超过 7 行，输出已找到的骨牌对矩阵，回溯}
   begin
    inc(count);
    for i:=1 to 7 do
     begin
      for j:=1 to 7 do write(outfile,map[i,j],' ');
      writeln(outfile,map[i,8]);
     end;
     writeln(outfile);
    exit;
   end;
  if (i<7)and(map[i+1,j]=-1)and(CanUse[DominoPairs[pip[i,j],pip[i+1,j]]]) then
   {假设该骨牌竖放}
   begin
    map[i,j]:=DominoPairs[pip[i,j],pip[i+1,j]];
    map[i+1,j]:=DominoPairs[pip[i,j],pip[i+1,j]];
    CanUse[DominoPairs[pip[i,j],pip[i+1,j]]]:=false;
    find(i,j);
    CanUse[DominoPairs[pip[i,j],pip[i+1,j]]]:=true;
    map[i,j]:=-1;map[i+1,j]:=-1;
   end;
  if (j<8)and(map[i,j+1]=-1)and(CanUse[DominoPairs[pip[i,j],pip[i,j+1]]]) then
   {假设该骨牌横放}
   begin
    map[i,j]:=DominoPairs[pip[i,j],pip[i,j+1]];
```

```
        map[i,j+1]:=DominoPairs[pip[i,j],pip[i,j+1]];
        CanUse[DominoPairs[pip[i,j],pip[i,j+1]]]:=false;
        find(i,j);
        CanUse[DominoPairs[pip[i,j],pip[i,j+1]]]:=true;
        map[i,j]:=-1;map[i,j+1]:=-1;
      end;
  end;
{-------------------------------------------------------}
begin
  count:=1;
  for i:=0 to 6 do
    for j:=i to 6 do
      begin
        DominoPairs[i,j]:=count;
        DominoPairs[j,i]:=count;
        inc(count);
      end;{生成多米诺骨牌对}
  write('Input file name:');readln(inname);
  write('Output file name:');readln(outname);
  assign(infile,inname);
  reset(infile);
  assign(outfile,outname);
  rewrite(outfile);
  for i:=1 to 7 do
    begin
      for j:=1 to 8 do read(infile,pip[i,j]);
      readln(infile);
    end;
  for i:=1 to 7 do
    for j:=1 to 8 do
      map[i,j]:=-1;{将所有位置设为未被占用}
  count:=0;
  for i:=1 to 28 do
    CanUse[i]:=true;{所有骨牌未被使用}
  find(1,1);{递归搜索所有可能的解}
  writeln(outfile,count);
  close(outfile);
end.
```

第七节 师生树

问题描述：

假设用〈A，B〉表示字符A，B有师生关系且B是A的1代学生（字符指A～Z，0～9共36个)。若给出〈A，B〉，〈B，C〉则C是A的2代学生。若给出〈A，B〉，〈B，C〉，〈C，D〉，〈D，E〉，〈B，E〉则E是A的2代学生，如果无最后一个关系〈B，E〉，则E是A的4代学生。如果某人没有老师，则称为师祖。所有具有师生关系的人组成一个师生树。

任务：从数据文件1中输入一组关系，求出师生树总数并分别输出各师生树的成员。输出各师生树的成员时，首先输出师祖，再依次输出各代学生，各代学生间用“,”分隔，同代学生中按ASCII码由小到大顺序输出。

输入格式：

从键盘输入数据文件名

输入数据文件格式如下：

5 ——— 表示有N 组关系

〈A，B〉 ——— 每行有一组关系，共N行

〈B，C〉

〈A，E〉

〈B，E〉

〈D，E〉

输出格式：在显示器上输出

1：A，BE，C ——— 表示该师生树成员表，

2：D，E

TOTAL=2 ——— 表示师生树总数

测试数据：

T201.dat

4

<A,B>

<B,C>

<C,D>

<B,D>

T202.dat

6

<A,B>

<B,C>

<C,D>

<B,D>

<D,E>
<E,B>

T203.dat
14
<8,9>
<5,6>
<1,0>
<3,0>
<7,3>
<5,1>
<9,0>
<4,9>
<1,9>
<3,2>
<4,2>
<2,6>
<1,3>
<9,3>

参考输出:

T201.out
1:A,B,CD
total=1

T202.out
total=0（存在圈）

T203.out
1:4,29,036
2:5,16,039,2
3:7,3,02,6
4:8,9,03,2,6
total=4

算法分析：

把师生关系看作有向图，使用邻接矩阵记录。首先要判断数据逻辑的合法性，即不存在 A 是 B 师祖同时 B 又是 A 的师祖，从图论的观点看不能有“圈”，实质就是在有向图中找回路：采用一种三重循环的算法

```
FOR I:=1 TO N DO
  FOR J:=1 TO N DO
    FOR K:=1 TO N DO
      IF (G[J,I]=1)AND(G[I,K]=1) THEN G[J,K]:=1;
```

这样，若 G[I,I]=1 则存在一条通过 I 的回路，即可判断合法性。此算法虽然编程十分简单，但是时间复杂度为 O（ n^3 ）是最大缺点，不过就本题而言，n<=36，因此用此算法并没有太大问题。

既然把师生关系看作有向图，那么求师生树就可用图的广度搜索算法。由于相同字符不会在队列里重复，所以这个算法实际很快，可认为是 O （ n ）。

在数据结构的设计中，采用了集合类型，这样可以简化了判断师祖、判图和标记结点已访问过的操作，从而使程序更加直观。

该算法可以描述为：

```
program 主程序;
begin
  初始化;
  判断逻辑合法性;
  寻找师生树 find
end.

procedure find(root:char);
begin
  初始化队列;
  repeat
    队首指针+1;
    if 当前结点的学生未扩展 then
    begin
      队尾指针+1;
      新结点按顺序进队列;
    end;
  until 首指针=尾指针;
  打印一棵师生树;
end;
```

程序分析：

主要变量说明：

link[i,j]：邻接矩阵，link['A','B']=1 表示 B 是 A 的学生；
roots：师祖的集合；
names：所有师、生的名字的集合；
total：师生树的总数。

过程说明：

```
procedure init;
    初始化，读入数据，计算 roots 和 names;
procedure judge;
    判断师生关系中是否有圈;
procedure find(root:char);
    寻找以 root 为师祖的一棵师生树;
procedure main;
```

总控过程，寻找所有师生树。

程序清单：

```
{$I-,R-}
Program GDOI97_2;
var f:text;
    fn:string;
    test,link:array['0'..'Z','0'..'Z']of 0..1;{<A,B>-->link[a,b]=1}
    roots,names:set of char;
    total:word;


procedure init;  {初始化过程}
var i,n:byte;
    s:string;
    te,st:char;
begin
  names:=[];roots:=[#0..#255];{师祖集合初始为所有人}
  fillchar(link,sizeof(link),0);total:=0;
  writeln;write('Enter file''s name:');
  readln(fn);assign(f,fn);reset(F);
  if ioresult<>0 then
    begin writeln('No the file '+fn);halt;end;
  readln(f,n);
  for i:=1 to n do
    begin
      readln(f,s);te:=s[2];st:=s[4];{读入 teacher 和 student}
      names:=names+[te]+[st];
      roots:=roots-[st];{在师祖集合排除学生}
      link[te,st]:=1;
    end;
  roots:=roots*names;{取交集}
  close(f);
end;
procedure judge;  {判图}
var i,j,k:char;
begin
  test:=link;
  for i:='0' to 'Z' do if i in names then
    for j:='0' to 'Z' do if j in names then
      for k:='0' to 'Z' do if k in names then
        if (test[j,i]=1)and(test[i,k]=1)then test[j,k]:=1;
  for i:='0' to 'Z' do
    if test[i,i]=1 then                          {找圈}
```

```
        begin writeln('No answer !');halt;end;
end;
procedure find(root:char);  {找师祖}
var qu:array[1..100]of          {队列}
          record
            ch:char;
            level:byte;{第几代学生}
          end;
    processed:set of char;{已经进入队列的师、生集合}
    ft,re,j:word;
    i:char;

begin
  processed:=[root];ft:=0;re:=1;
  fillchar(qu,sizeof(qu),0);
  qu[1].ch:=root;
  repeat
    inc(ft);
    with qu[ft] do
      for i:='0' to 'Z' do
        if (link[ch,i]=1)and(not(i in processed)) then
          begin
            processed:=processed+[i];
            inc(re);
            qu[re].ch:=i;qu[re].level:=qu[ft]. level+1;       {新结点进队尾}
            {---------进队列时按 ASCII 码排序---------------}
            j:=re-1;
            while (qu[j].level=qu[re].level)and(qu[j].ch>i) do dec(j);
            inc(j);
            move(qu[j],qu[j+1],sizeof(qu[j])*(re-j+1)); {后移}
            qu[j]:=qu[re+1];                            {插入}
          end;
  until ft=re;
  {------------输出一棵师生树----------------}
  inc(total);
  write(total,': ');j:=0;ft:=0;
  repeat
    inc(ft);
    if qu[ft].level>j then
      begin write(',');inc(j);end;
    write(qu[ft].ch);
  until ft=re;
  writeln;
end;
```

```
procedure main;  {寻找所有师生树}
var i:char;
begin
  for i:='0' to 'Z' do
    if i in roots then find(i);
  writeln('Total=',total);
end;
begin  {主控程序}
  init;
  judge;
  main;
end.
```

第八节 旅游预算

本题取材于 ICPC（世界赛）93A 题。

问题描述：

一个旅行社需要估算乘汽车从某城市到另一城市的最小费用，沿路有若干加油站，每个加油站收费不一定相同。

旅游预算有如下规则：

若油箱的油过半，不停车加油，除非油箱中的油不可支持到下一站；

每次加油时都加满；

在一个加油站加油时，司机要花费 2 元买东西吃；

司机不必为其他意情况而准备额外的油；

汽车开出时在起点加满油箱；

计算精确到分（1 元=100 分）。

编写程序估计实际行驶在某路线所需的最小费用。

输入格式：

从当前目录下的文本文件“route.dat”读入数据。

按以下格式输入若干旅行路线的情况：

第一行为起点到终点的距离（实数）

第二行为三个的实数，后跟一个整数，每两个数据间用一个空格分隔。

其中第一个数为汽车油箱的容量（升），第二个数是每升汽油行驶的公里数，第三个数是在起点加满油箱所需的费用，第四个数是加油站的数量（≤50）、

接下去的每行包括两个实数，每个数据之间用一个空格分隔，其中第一个数是该加油站离起点的距离，第二个数是该加油站每升汽油的价格（元/升）。加油站按它们与起点的距离升序排列。所有的输入都一定有解。

输出格式：

答案输出到当前目录下的文本文件“route.out”中。

该文件包含两行。第一行为一个实数和一个整数，实数为旅行的最小费用，以元为单位，精确到分，整数表示途中加油的站的总数 N。第二行是 N 个整数，表示 N 个加油的站的编号，按升序排列。数据间用一个空格分隔，此外没有多余的空格。

输入输出举例：

输入文件：route.dat

```
516.3
15.7 22.1 20.87 3
125.4 1.259
297.9 1.129
345.2 0.999
```

输出文件：route.out

```
38.09 1
2
```

测试数据：

ROUTE1.DAT

475.6

11.9 27.4 14.98 6

102.0 0.999

220.0 1.329

256.3 1.479
275.0 1.029
277.6 1.129
381.8 1.009

ROUTE2.DAT
500
30 25.4 10 20
102.0 0.999
130.0 1.000
144.1 1.111
150.2 1.100
166.5 1.050
180.0 1.142
188.0 1.205
190.1 1.300
220.0 1.329
234.1 1.299
244.2 1.123
256.3 1.479
275.0 1.029
277.6 1.129
299.0 1.022
320.0 1.002
360.0 2.000
381.8 1.009
421.4 1.562
470.3 2.100

ROUTE3.DAT
320
10 10 10 30
10 1
20 1
30 1
40 1
50 1
60 1
70 1
80 1
90 1
100 1
110 1
120 1

```
130 1
140 1
150 1
160 1
170 1
180 1
190 1
200 1
210 1
220 1
230 1
240 1
250 1
260 1
270 1
280 1
290 1
300 1
```

```
ROUTE4.DAT
2717.1
15.2 12.5 13.3 45
36.2 1.186
122.1 1.777
220.0 0.906
243.9 1.505
276.4 1.008
366.9 1.182
430.2 1.178
461.6 1.583
495.1 1.166
525.5 1.674
547.5 1.493
575.2 1.706
673.6 1.283
720.1 1.270
778.2 0.937
837.8 1.664
921.7 1.650
952.2 1.177
1049.1 1.003
1113.1 1.167
1171.8 1.104
1230.6 1.013
```

1303.0 1.019
1338.3 1.339
1360.6 1.487
1455.0 1.081
1493.6 1.127
1592.1 1.281
1641.7 1.657
1713.0 1.714
1779.1 1.825
1866.3 1.692
1926.3 1.064
1972.2 1.679
2056.2 1.082
2122.6 1.554
2214.2 1.890
2305.9 1.644
2372.5 0.998
2413.4 1.883
2453.7 1.748
2473.9 1.706
2559.0 1.045
2606.7 1.537
2671.0 0.947

参考输出:

ROUTE1.OUT
27.31 1
4

ROUTE2.OUT
10.00 0

ROUTE3.OUT
38.00 3
6 12 22

ROUTE4.OUT
301.04 17
2 3 6 9 13 15 18 20 23 26 28 31 33 35 37 39 43

算法分析:

本题的求解与求有向图的最短路径比较相似,也是选取两点间的直接连线费用和经过某个中间结点进行转折费用中的最小者。因此,求总费用的最小值可以分解为求若干分段的费用最小值,它满足优化原则,所以本题可以采用动态规划的方法进行求解。其核心算法是:

```
  for interval:=1 to n-1 do
   for start:=0 to n-1 do
    if start+interval<=n then
     begin
      stop:=interval+start;
      if 从start能直接到stop then
        begin
          lest[start,stop]:=从start能直接到stop的费用;
          nextstop[start,stop]:=stop;
        end;{求出直接从start到stop的费用}
      for next:=start+1 to stop-1 do
      if 从start能直接到next then
        begin
          cost:=从start到next的费用+从next到stop的费用;
          if cost<lest[start,stop] then
            begin
              lest[start,stop]:=cost;
              nextstop[start,stop]:=next;
            end;
        end; {中间经过next是否能减少费用}
    end; {动态规划的求解过程}
```

然后，从各油站中找到能直接到终点，而且从起点到该站的lest值最小的油站Stop。则该旅程的最小费用为该lest值加上在起点的费用。然后利用下面的迭代式就可以求出，中途所停的油站：

start：=0；start：=nextstop[start，stop]；(直到start=stop)

程序分析：

```
program GDOI98_3_Route;{旅游预算}
const max=50;
      error=1e-10;
var min,full,firstcost,km,cost,distance:extended;
    n,interval,start,stop,next,i,m:integer;
    d,price:array [0..max] of extended;
    result:array [1..max] of integer;
    lest:array [0..max,0..max] of extended;
     {两个油站间的最小费用}
    nextstop:array [0..max,0..max] of integer;
     {两个油站间中途停的站}
    fin,fout:text;

procedure initialize;
  {初始化}
  var i,j:integer;
```

```
  begin
    assign(fin,'route.dat');
    reset(fin);
    readln(fin,distance);
    readln(fin,full,km,firstcost,n);
    for i:=1 to n do
      begin
        readln(fin,d[i],price[i]);
      end;
    close(fin);
    {读入油站信息}
    d[0]:=0;price[0]:=0;
    for i:=0 to n do for j:=0 to n do lest[i,j]:=1e10;
    {两个油站间的最小费用置无穷大}
  end;

function feasible(src,des:integer):boolean;
{在油箱满的时候能否直接从 src 到 des}
  begin
    if d[des]-d[src]>full*km+error then
      begin
        feasible:=false;
        exit;
      end;
    if (d[des+1]-d[src])>full*km+error then
      begin
        feasible:=true;
        exit;
      end;
    if d[des]-d[src]>full*km/2+error then
      begin
        feasible:=true;
        exit;
      end;
    feasible:=false;
  end;

function achieve(from:integer):boolean;
{在油箱满的时候能否直接从 from 到目的地}
  begin
    if distance-d[from]<full*km+error then achieve:=true
    else achieve:=false;
  end;
```

```
begin
  initialize;

  for interval:=1 to n-1 do
   for start:=0 to n-1 do
    if start+interval<=n then
     begin
      stop:=interval+start;
      if feasible(start,stop) then
        begin
          lest[start,stop]:=2+price[stop]*(d[stop]-d[start])/km;
          nextstop[start,stop]:=stop;
        end;{求出直接从 start 到 stop 的费用}
      for next:=start+1 to stop-1 do
      if feasible(start,next) then
        begin
          cost:=2+price[next]*(d[next]-d[start])/km+lest[next,stop];
          if cost<lest[start,stop] then
            begin
              lest[start,stop]:=cost;
              nextstop[start,stop]:=next;
            end;
        end;{中间经过 next 是否能减少费用}
    end;{动态规划的求解过程}

  assign(fout,'route.out');
  rewrite(fout);
  min:=1e10;
  if distance<full*km+error then
    begin
      writeln(fout,firstcost:0:2,' ',0);
      {可以直接从起点到终点}
    end
  else
    begin
      for i:=1 to n do if achieve(i) then
        begin
          if lest[0,i]<min then
            begin
              min:=lest[0,i];
              stop:=i;
            end;
        end;
      min:=min+firstcost;
```

```
      write(fout,min:0:2);
      m:=0;start:=nextstop[0,stop];
      while start<>stop do
      begin
        inc(m);
        result[m]:=start;
        start:=nextstop[start,stop];
      end;
      inc(m);
      result[m]:=stop;
      writeln(fout,' ',m);
      for i:=1 to m do writeln(fout,result[i]);
      {输出旅行的最小费用，和所停的站}
    end;
  close(fout);
end.
```

第九节 正整数竖式除法

问题描述：

编程序实现计算两个正整数的除法。

任务：被除数和除数的位数长度均小于256位，用除法竖式的形式计算并输出商和余数。

输入格式：

从键盘输入数据文件的文件名

输入数据文件格式如下：

10000000020 ——— 表示被除数的值

1000000000 ——— 表示除数的值

输出格式：

在显示器上输出：

```
                10      ———  商
 1000000000 /10000000020
             1000000000
                   20
                    0
                   20      ———  余数
Quotient=10              ———  商=10
Remainder=20             ———  余数=20
```

测试数据：

T701.Dat

```
20000
255
```

T702.Dat

```
37403
331
```

T703.Dat

```
2432453254323740343243254
543543231
```

T704.Dat

```
24324543254323740343243254
54354325435432543254325431
```

T705.Dat

2432454325432374034324325 4
25435432543254325431

参考输出:

```
T701.Out
        78
      -----
255 / 20000
      1785
      -----
       2150
       2040
      -----
        110


T702.Out
       113
      -----
331 / 37403
      331
      -----
       430
       331
      -----
        993
        993
      -----
          0
```

T703.Out
Quotient=4475179002502820
Remainder=473831834

T704.Out
Quotient=0
Remainder=2432454325432374034324325 4

T705.Out
Quotient=956325
Remainder=3227396047575442179

算法分析:

由于被除数、除数长度范围是 2 5 6 以内，因此必须采用高精度除法；
可以用计算机模拟手算除法，把除法试商转化为连减，具体做法如下：

1 输入被除数 dio、除数 di（判断除数是否为零，是则结束），转 2
2 length(di)→j， Q 为空串，转 3
3 截取被除数 dio 前 j 位，赋给 dd，转 4
4 从 dd 中逐次减去 di，直到减了 K 次后 dd 小于 di 为止，转 5
5 把 K 加到商数 Q 末尾，转 6
6 如果还未除完，则 j+1→j，dd＋dio[j]，转 4 ，否则转 7
7 去除商数 Q 前多余的 0 ，输出商数 Q 及余数 dd

程序分析:

```
{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q-,R-,S+,T-,V+,X+}
```

```
{$M 16384,0,655360}
Program GDOI97_7;
var
  dio,di,q,dd:string;  {被除数、除数、商数、余数}
  k,j:integer;

procedure init;   {输入}
var
  s:string;
  f:text;
begin
  write('File name='); readln(s);
  assign(f,s); reset(f);
  readln(f,dio); readln(f,di);
  close(f);
  while (dio<>'0')and(dio[1]='0')do delete(dio,1,1);
  while (di<>'0')and(di[1]='0')do delete(di,1,1);
  if di='0' then begin writeln('Error!'); halt; end;
  {↑判断除数是否为零}
end;

function than(s1,s2:string):integer; {比较两个数的大小}
begin
  if length(s1)>length(s2) then begin than:=1; exit; end;
  if length(s1)<length(s2) then begin than:=-1; exit; end;
  if s1>s2 then than:=1 else if s1<s2 then than:=-1 else than:=0;
end;

procedure sub(var s1:string;s2:string);  {高精度减法}
var
  i,j:integer;
  a:array[1..300]of integer;
begin
  j:=length(s2);
  for i:=length(s1)downto length(s1)-length(s2)+1 do
    begin
      dec(s1[i],ord(s2[j])-48);
      if s1[i]<'0' then
        begin
          inc(s1[i],10); dec(s1[i-1]);
        end;
      dec(j);
    end;
  while (s1<>'0')and(s1[1]='0')do delete(s1,1,1);
```

```
end;

procedure main;
var
  ft:boolean;
begin
  q:=''; ft:=false;
  dd:=copy(dio,1,length(di));
  j:=length(dd);
  repeat
    k:=0;
    {↓试商}
    while than(dd,di)<>-1 do
      begin
        inc(k);
        sub(dd,di);
      end;
    q:=q+chr(k+48);
    if j<length(dio)then
      begin
        inc(j); dd:=dd+dio[j];
      end else ft:=true;
  until ft;
  while (q<>'0')and(q[1]='0')do delete(q,1,1);
  {↓输出}
  writeln('Quotient=',q);
  writeln('Remainder=',dd);
end;

begin
  init;
  main;
end.
```

第十节 移棋子

本题取材于 ICPC（世界赛）93F 题。

问题描述：

有一个 5×5 的方格棋盘，棋盘上放着 24 粒不同的棋子，分别用英文大写字母 A、B、……、X 来表示；棋盘上还有一个方格空着，用空格表示。

游戏的每一步是将空格上方、下方、左方或右方的棋子移入空格，这四种操作分别用 1、2、3、4 来表示。

如果给出棋盘的初始状态和一定顺序的有限操作序列，就可以得到唯一的目标状态，例如：下图中的初态经过操作序列 144223"得到目标态"

T	R	G	S	J
X	D	O	K	I
M		V	L	N
W	P	A	B	E
U	Q	H	C	F

初态

T	R	G	S	J
X	O	K	L	I
M	D	V	B	N
W	P		A	E
U	Q	H	C	F

终态

但是，原来正确的操作序列的顺序给人打乱了，初态按照乱了操作序列并不能得到终态（仅仅是顺序打乱了，各类型操作的总数不变）。

现在的任务是求出原来的操作序列。（如果有多解，给出其中任意一种即可）

输入格式：

从当前目录下的文本文件“PUZZLE.DAT”中读入数据。该文件的第一行到第五行是棋盘的初态，每行有五个字符。第六行到第十行是棋盘的终态，每行有五个字符。第十一行是一个正整数，表示操作序列的长度 *1(1≤50)*。第十二行有 *1* 个字符，表示被人打乱后的操作序列。每行的各数据之间无空格。每行的行首、行末无多余空格。

输出格式：

答案输出到当前目录下的文本文件“PUZZLE.OUT”。如果问题有解，则输出一行 *1* 个字符，表示原来正确的操作序列；如果问题无解，则输出“0”。

输入输出举例：

输入文件：PUZZLE.DAT

```
TRGSJ
XDOKI
M VLN
WPABE
UQHCF
TRGSJ
XOKLI
MDVBN
WP AE
UQHCF
6
442231
```

输出文件：PUZZLE.OUT

```
144223
```

测试数据：

PUZZLE1.DAT

```
TRGSJ
XDOKI
M VLN
WPABE
UQHCF
TRGSJ
XDOKI
M VLN
WPABE
UQHCF
4
1122
```

PUZZLE2.DAT

```
TRGSJ
XDOKI
M VLN
WPABE
UQHCF
TR SJ
XDGOI
MVLKE
WPBNF
UQAHC
13
1111223334444
```

PUZZLE3.DAT

```
TRGSJ
XDOKI
M VLN
WPABE
UQHCF
TOKJI
DGS R
XMALN
WVPBE
UQHCF
21
111112222333334444444
```

PUZZLE4.DAT

TRGSJ
XDOKI
M VLN
WPABE
UQHCF
TROGS
XDIJ
MPLKN
BCVHE
AWUQF
28
11111222233333333444444444444

PUZZLE5.DAT
TRGSJ
XDOKI
M VLN
WPABE
UQHCF
RGOSJ
MTXAL
WDV I
UPKEN
QHCBF
36
111111111222222222333333334444444444

PUZZLE6.DAT
TRGSJ
XDOKI
M VLN
WPABE
UQHCF
XGVOS
RPTDI
W MJK
AHLEN
UQCBF
40
1111111111222222222233333333334444444444

参考输出:
PUZZLE1.OUT
1212

PUZZLE2.OUT
4442233141131

PUZZLE3.OUT
423131414231442314423

PUZZLE4.OUT
244233314442333141414133244

PUZZLE5.OUT
131442423132423413132224441314241132

PUZZLE6.OUT
242411133232444411132423143323113241423 2

算法分析:

这是一道典型的搜索题。根据题目的设定，可知：（1）在有限步内把原始状态移至目标状态，或确定无解；（2）移动的步数已知；（3）各个移动方向的步数已知，但序列不知；（4）解必须满足（2）和（3）的限制。

显然，题设的已知为搜索的优化提供了许多有益的条件，大大减少了搜索量。众所周知，搜索的优化一直是搜索题的关键，而本题优化条件使搜索的任务减轻了。

与一般的搜索题一样，本题的解主要由两部分组成：

（1） 搜索寻解；

（2） 确定剪枝条件，以优化搜索。

具体来说：

（1） 搜索寻解

这一部分比较简单，只须逐一检查各种可能的移动序列，输出满足题设的序列即可。

```
Procedure search;
    Begin
        检查所有移动序列
            if   序列把原始状态移为目标状态 then  输出移动序列;
    End;
```

显然，使用递归能很好地完成这一任务。

```
{   使用递归，寻找第 t 个移动方向      }
{   p 数组为移动序列，p[t]的值取 1，2，3，4，表示第 t 个移动方向      }
{   l 为题设的移动总步数，也即序列 p 的长度     }
Procedure search(position:integer);
Begin
    For I:=1 to 4 do
        Begin
            P[t]:=I;              {   p[t]分别取 1，2，3，4 的方向   }
            If t=l then
```

```
                    Begin
                        if   序列把原始状态移为目标状态
                             then 输出移动序列;
                    End;
                Else Search(t+1);
            End;
    End ;
```

（2） 确定剪枝条件，以优化搜索

显然，上述的全搜索在时间上是令人无法忍受的，我们应确定一些剪枝条件，避免一些无谓的搜索，以大规模地搜索量。

我们可以从以下几方面减少搜索：

1. 各移动方向的数目

由于各移动方向的数目是已知的，因此一旦某个移动方向的个数超出限制，则可不搜索此方向。

```
{   根据移动方向的数目决定继续与否   }
{   r 数组为各方向的目前数目   }
{   position 为当前的搜索深度，与上述过程的 position 一致 }
Function continue_by_direction_number:boolean;
Begin
    {   求出 r 数组          }
    置零 r 数组;
    for I:=1 to position do
        r[p[I]]:=r[p[I]]+1;
    {   判断可能性          }
    for I:=1 to 4do
        if r[I]>I 方向的总数目 then 返回 false;
    返回 true;
End;
```

2. 空格的位置和各方向的数目

由于目前状态和目标状态都已知，各方向的数目也已知，因此可确定目前状态的空格能否由此方向集合移至目标状态的位置。

```
{   各变量的定义如上 }
Procedure continue_by_space:boolean;
Begin
    求出 r 数组;
    if 目标状态的空格横坐标-当前状态的空格横坐标<>
            （4 方向的总数目-r[4]）-（3 方向的总数目-r[3]）
        then 返回 false;
    if 目标状态的空格纵坐标-当前状态的空格纵坐标<>
            （2 方向的总数目-r[2]）-（1 方向的总数目-r[1]）
        then 返回 false;
    返回 true;
```

End;

3. 各字符的位置和各方向的数目

同 2，我们可以根据目前状态和目标状态以及各方向的数目，确定目前状态的各字符能否由此方向集合移至目标状态的位置。

```
{    各变量的定义如上}
procedure continue_by_characters:boolean;
begin
    求出 r 数组;
    sum:=各字符的横坐标之和;
    if sum<>（4 方向的总数目-r[4]）-（3 方向的总数目-r[3]）
        then 返回 false;
    sum:=各字符的纵坐标之和;
    if sum<>（2 方向的总数目-r[2]）-（1 方向的总数目-r[1]）
        then 返回 false;
    返回 true;
end;
```

通过上述优化，搜索量大大减少，程序能在限定时间内完成搜索。

程序分析:

```
{$R-,V-,S-}

program GDOI98_8_puzzle;

const
 maxl=50;
 m=5;                {   格子的纵横个数   }
 x:array[1..4] of integer=(-1,1,0,0);        {   横坐标的位移量       }
 y:array[1..4] of integer=(0,0,-1,1);        {   纵坐标的位移量       }

var
 finame,foname:string;
 fi,fo:text;
 s,s1:array[1..m,1..m] of char;          {   原始状态和目标状态   }
 l:integer;                      {   移动步数     }
 p:array[1..maxl] of integer;            {   移动序列     }
 r:array[1..4] of integer;               {   各方向的数目     }
 answer:boolean;                     {   解的确定性       }
 sx,sy,sx1,sy1:integer;              {原始状态和目标状态的空格坐标}
{   数据输入     }

procedure data_input;
 const
```

```
  ord0=ord('0');
 var
  st:string;
  i,j:integer;
  ch:char;
 begin
  reset(fi);

  for i:=1 to m do
   begin
    readln(fi,st);
    for j:=1 to m do
     begin
      s[i,j]:=st[j];
      if s[i,j]=' ' then
       begin sx:=i; sy:=j; end;
     end;
   end;
  for i:=1 to m do
   begin
    readln(fi,st);
    for j:=1 to m do
     begin
      s1[i,j]:=st[j];
      if s1[i,j]=' ' then
       begin sx1:=i; sy1:=j; end;
     end;
   end;
  readln(fi,l);
  for i:=1 to 4 do r[i]:=0;
  for i:=1 to l do
   begin
    read(fi,ch);
    r[ord(ch)-ord0]:=r[ord(ch)-ord0]+1;
   end;

  close(fi);
 end;

{   根据空格的位置决定解的可能性，若不可能有解则返回 false，否则返回 true     }
 function space_check:boolean;
  var
   ok:boolean;
  begin
```

```
   ok:=true;
   if sx1-sx<>r[2]-r[1] then ok:=false;
   if ok then if sy1-sy<>r[4]-r[3] then ok:=false;
   space_check:=ok;
  end;

{   根据各格的位置判断解的可能性，返回值同上    }
 function go_on:boolean;
  var
   t:integer;
   r1:array[1..4] of integer;
   found:boolean;
   bb:boolean;
   i,j,k,l:integer;
  begin
   bb:=space_check;
   if bb then
    begin
     for i:=1 to 4 do r1[i]:=0;
     for i:=1 to m do  if bb then  for j:=1 to m do  if bb then
      if s[i,j]<>' ' then
       begin
        found:=false;
        for k:=1 to m do  if not(found) then
        for l:=1 to m do  if not(found) then
         if s[i,j]=s1[k,l] then
          begin
           found:=true;
           if i>k then begin inc(r1[2],i-k); if r1[2]>r[2] then bb:=false; end
            else begin inc(r1[1],k-i); if r1[1]>r[1] then bb:=false; end;
           if j>l then begin inc(r1[4],j-l); if r1[4]>r[4] then bb:=false; end
            else begin inc(r1[3],l-j); if r1[3]>r[3] then bb:=false; end;
          end;
       end;
     if r[1]-r1[1]<>r[2]-r1[2] then bb:=false;
     if r[3]-r1[3]<>r[4]-r1[4] then bb:=false;
    end;
   go_on:=bb;
  end;

{   解的序列输出    }
 procedure p_output;
  var
   i:integer;
```

```
  begin
   for i:=1 to l do
    write(fo,p[i]);
  end;

{   搜索第 t 步序列  }
 procedure p_found(t:integer);
  var
   sx0,sy0:integer;
   temp:char;
   i:integer;
  begin
   for i:=1 to 4 do
    if not(answer) then
     begin
      p[t]:=i;
      sx0:=sx; sy0:=sy;
      inc(sx,x[i]);
      inc(sy,y[i]);
      dec(r[i]);

      if (1<=sx) and (sx<=m) and (1<=sy) and (sy<=m) and (0<=r[i]) then
       begin
        temp:=s[sx0,sy0]; s[sx0,sy0]:=s[sx,sy]; s[sx,sy]:=temp;
        if go_on then
         if t=l then begin answer:=true; p_output; end
          else p_found(t+1);
        temp:=s[sx0,sy0]; s[sx0,sy0]:=s[sx,sy]; s[sx,sy]:=temp;
       end;

      inc(r[i]);
      dec(sx,x[i]);
      dec(sy,y[i]);
     end;
  end;

 begin
  finame:='puzzle.dat';
  assign(fi,finame);
  foname:='puzzle.out';
  assign(fo,foname);

  data_input;
```

```
  rewrite(fo);
  answer:=false;
  p_found(1);
  if answer=false then write(fo,'0');
  close(fo);
end.
```

第五章 习题

第一节 习题

1、电子表格（ Table ）

本题取材于 ICPC 92A 题。

问题描述：

在一个 M 行 N 列电子表格中共有 M*N 个表格单元，每个单元内容或是一个有符号整数，或是一个关于整数和引用其他单元的**加减表达式**，有符号整数由数字组成，为负数时在前面加上负号，表达式由有符号整数，加/减号和表格单元标号组成。请编一个程序计算该电子表格里各单元的内容。

输入格式：

从当前目录下的文本文件“table.dat”读入数据。

输入一个电子表格的描述文件，其描述如下：

第一行是电子表格的行数 M 和列数 N，行用字母 A-T 表示，列用 0-9 表示。

以下各行是各个单元的内容（一行表示一个单元内容，顺序是以行为主序、即每行内容分别是 A0,A1,A2,...,B0,B1,...的内容），每个单元是一个有符号整数或是一个表达式。若单元的内容首字符是一个“-”（负号）或数字表示它是一个有符号整数，否则是一个表达式。

输出格式：

答案输出到当前目录下的文本文件“table.out”中。

对给定的电子表格的描述，计算出每个单元的数值。若该电子表格中每个单元的值都能计算出来，则输出整个电子表格，若有单元计算不出来（即有单元循环引用），则只输出所有**不能**计算出来的内容（即行，列的标号（如 A1）和原来输入的表达式，标号与表达式之间用“:”分隔）。

例如，输入文件为：

```
3 2
A0
5
C1
7
A1+B1
B0+A1
```

输出应为：

```
A0:A0
B0:C1
C1:B0+A1
```

若全部能计算出来，则按表格的形式输出经计算后的电子表格（共 M 行，每行 N 个数据，数据间以一个空格分隔）。

输入输出举例：

输入文件：table.dat

```
2 2
A1+B1
5
3
B0-A1
```

输出文件：Table.out

```
3       5
3       -2
```

2、DEL 命令 (DEL)

本题取材于 ICPC 96D 题。

题目描述：

要求确定是否能够在 MS-DOS 内，用一个 DEL 命令把给定多个文件从 MS-DOS 的子目录中删除。子目录中没有嵌套的子目录。

DEL 命令的使用格式为：DEL　文件全名

文件全名由文件名和后缀名组成。文件名由 1 到 8 个字符构成，后缀名最多由 3 个字符构成，文件名和后缀名之间一句点“.”分隔。后缀名可以为空（这样文件的全名就以一个点号结尾）。在文件全名里可以使用通配符“？”和“*”。问号可以代替除点号外的任意一个字符；星号可以代替一串字符（不包括点号），甚至代替空串。星号只能出现在文件名或后缀名的最后一个字符。

MS-DOS 系统可能可以使用其他通配符，但它们在本任务中不能被使用。文件名和后缀名仅由大写英文字母和数字组成。

输入格式：

从当前目录下的文本文件“DEL.DAT”中读入数据。

该文件含有一组文件全名，其中没有空行或空格。在输入文件中每个文件名占一行，前面有一个控制用的符号：“-”（表示该文件要被删除），“+”（表示该文件要留下）。文件全名不会重复。文件中至少含一个文件全名，至少有一个文件要被删除。文件中最多有 3000 个文件。

输出格式：

结果输出到当前目录下的文本文件“DEL.OUT”中。

如果不能实现要求，输出“IMPOSSIBLE”；如能实现，则输出所要求的 DEL 命令行。“DEL”与文件全名间**必须**有一个空格。

输入输出举例：

输入：DEL.DAT

```
-BP.EXE
-BPC.EXE
+TURBO.EXE
```

输出：DEL.OUT

```
DEL ?P*.*
```

3、分割方格 (Divide)

本题取材于 NEERC 97H 题。

题目描述：

有一个 m×m（m ≤ 20）的方格纸，其中有 n^2 个（n≤m，n≤6）方格标有“*”的记号，标有“*”的方格是连通的，所谓连通是指：假设有一个“车”在一个标有“*”的方格上，每次移动只能移到与原方格垂直相邻或水平相邻的标有“*”的方格上，经过有限次移动，“车”可以到达任意另一个标有“*”的方格（如右图）。

				*			
		*		*			
		*	*	*			
		*	*	*	*		
		*			*		
		*	*	*	*		

现在要把这标有“*”的 n^2 个方格分成 A、B 两部分，每部分都是连通的，而且，这两部分经过旋转、翻转或平移后可以拼成 n×n 的正方形。（若有多解，请给出任意一个）

例如上例，可以分成如下两部分（图 A），并且这两部分可以拼成 4×4 的正方形（图 B）。

图 A

				B			
		B		B			
		B	B	B			
		A	A	B	A		
		A			A		
		A	A	A	A		

图 B

B	B	B	B
A	A	B	A
A	B	B	A
A	A	A	A

输入格式：

从当前目录下的文本文件“DIVIDE.DAT”中读入数据。

文件的第一行有一个整数 m。

以下的 m 行每行各有 m 个字符：“.”代表空格，“*”代表标有“*”记号的方格。

输出格式：

答案输出到当前目录下的文本文件“DIVIDE.OUT”中。

文件共 m 行每行有 m 个字符：“.”代表空格，“A”、“B”分别代表分割成的两部分的组成方格。

输入输出举例：

输入文件：DIVIDE.DAT

```
8
........
....*...
..*.*...
..***...
..****..
..*..*..
..****..
........
```

输出文件：DIVIDE.OUT

```
........
....B...
..B.B...
..BBB...
..AABA..
..A..A..
..AAAA..
........
```

4、信息编码 (Decode)

本题取材于 ICPC 91F 题。

问题描述：

下面介绍一种信息编码的方法。

编码后的信息分成两部分，第一部分为信息头，它实际上是信息中的字符的编码表；第二部分是用二进制表示的编码信息。这种编码方式的核心是由一系列由 0、1 组成的键，

如下所示：0，00，01，10，000，001，010，……，110，0000，0001，……，1110，00000，……其中第一个键长为 1，接下 3 个键长为 2，接下 7 个键长为 3，接下 15 个键长为 4，……以此类推（注意，没有全部为 1 的键）。这些键对应着信息头的字符，第 1 个键对应信息头中的第 1 个字符，第 2 个键对应信息头中的第 2 个字符，……

如信息头为 AB#TANCNRTXC

则 A 对应 0，B 对应 00，#对应 01，T 对应 10，A 对应 000

第二部分编码信息是由 0、1 组成的串（如中间有回车，则忽略它）。这部分分成若干段，每个段的头三个数字表示这段里的键的长度，例如，头三个数字为 010，则表示该段中剩余部分全部由长度为 2 的键（00、01 或 10）组成。段的结束标记是全为 1 的键（如 11）。若头三个数字为 000，则表示编码结束。

例如，信息头为：　　TNM□AEIOU　（□表示空格）

编码信息为：　0010101100011

1010001001110110011

11000

在信息头中可以知道如下对应关系：

字符	T	N	M	□	A	E	I	O	U
键	0	00	01	10	000	001	010	011	100

在编码信息中，忽略回车，按编码规则切分如下：

001/0/1/011/000/111/010/00/10/01/11/011/001/111/000

其含义见下表，所以该编码表示的信息为“TAN□ME”。

	键长	段内的内容			
第 1 段	001（键长为 1）	0（T）	1（段结束）		
第 2 段	011（键长为 3）	000（A）	111（段结束）		
第 3 段	010（键长为 2）	00（N）	10（□）	01（M）	11（段结束）
第 4 段	011（键长为 3）	001（E）	111（段结束）		
第 5 段	000（编码结束）				

现要求编一程序，实现对编码信息进行解码。

输入格式：

从当前目录下的文本文件“DECODE.DAT”中读入数据。该文件的第一行是信息头，在信息头中,可能出现相同的字母。余下的若干行是编码信息。各行行首、行末无多余空格。问题描述中“□”在文件中是空格。在输入文件中，每一行的长度不超过 255 个字符，行首、行末无多余空格。

输出格式：

解码所得的信息输出到当前目录下的文本文件“DECODE.OUT”。该文件就一行，为解码所得的信息，行首、行末无多余空格。

输入输出举例：

DECODE.DAT

```
TNM AEIOU
0010101100011
1010001001110110011
11000
```

DECODE.OUT

```
TAN ME
```

5、海上交通控制 （ Lane ）

本题取材于 ICPC 94G 题。

问题描述：

海上交通图可以用一个有向图来表示,顶点表示港口,边表示两个港口之间是否有航线可通。为保证海上交通安全和以尽量快的速度到达目的地，每艘船在出发前都将航行计划（包括出发时间、速度、出发与到达港口）提交给海上交通控制局，由海上交通控制局为它们制定航线。现给出一系列的船只航行计划（包括出发时间、速度、出发与到达港口），请你根据以下原则编程为它们制定航线：

1. 每艘船在出发的一瞬间提交航行计划（提交和出发的时间差可以忽略）；
2. 每艘船都严格按照出发时间出发，不能提前，也不能延迟；
3. 在任何时间一条航道（两港口间的直达航线）上只能有一艘船，因此，一艘船在出发的瞬间发现某航道将在未来的某段时间内会被在它之前出发的船占用,则它在那一段时间内将不会使用该航道，当然其余时间还是可以使用该航道；
4. 每个港口均可被无限艘船同时使用；
5. 在满足上述条件后，要使本船航行时间最短；
6. 假如某船不能到达目标港口，那么它将放弃这个航程。
7. 船在任何时候都不能停下来，即从出发后，要一直航行到目的地，中途不得在航道或港口中停留。

时间用 4 位数字表示如 2345 表示 23：45，速度单位用节（海里/小时）表示。

在计算时间时，中间结果应是精确的时间（即不要四舍五入到分钟），而航行时间的计算是以总距离除以速度为准,最终到目标地的时刻应是起航时刻加上航行时间的四舍五入到分钟的结果。

输入格式：

从当前目录下的文本文件“LANE.DAT”读入数据。输入的数据一定有解，且不会出现跨越 00：00 的情况，例如，一艘船在 23：55 出发，第二天 0：15 到达的情况是不会出现的。

输入文件开头是港口定义：

第一行是港口数 N（≤26）；

第二行是一个长度为 N 的大写字母串，每个字母表示一个港口名字；

第三行开始 N 行的 N×N 矩阵是一个邻接矩阵，每行有 N 个整数，其值为港口间距离（单位为海里），整数间以空格分隔（若为 0 表示两港口没有直达航线相连）；

接着的一行是一个整数 M（≤50），表示共有 M 艘船提交航行计划；

接下去的每 3 行表示一艘船的航行计划，其中第一行是船名，第二行是出发时间和航速，两者均为整数，以一个空格分隔，第三行是两个大写字母，之间没有任何分隔，第一个表示出发的港口，第二个表示目的港口；

输出格式：

答案输出到当前目录下的文本文件“LANE.OUT”中。

该文件的每 3 行表示一艘船的航线，其中第一行是船名，第二行是出发时间和到达时间，两者均为整数，以一个空格分隔，第三行是数个大写字母，之间没有任何分隔，表示该船经过的港口（包括出发和目的港口）。如果这艘船放弃航程时，到达时间用-1 表示，并留空第三行。

注意：在输入和输出中航行计划和航线均按出发时间排序，时间精确到分钟。

输入输出举例：

输入文件：LANE.DAT

```
5
ABCDE
0 10 0 50 10
10 0 20 70 0
0 20 0 20 0
50 70 20 0 10
10 0 0 10 0
4
Bluesky
0800 10
CB
Blackhorse
0900 5
AB
Greenforest
1000 20
DB
```

输出文件：LANE.OUT

```
Bluesky
800 1000
CB
Blackhorse
900 1100
AB
Greenforest
1000 1130
DEAB
Silverboat
1200 1300
DC
```

```
Silverboat
1200 20
DC
```

6、投递最佳路线

本题取材于 ICPC 97E 题。

问题描述：

某投递公司需要在各个车站间运送包裹。某位司机运送一个包裹到某站后，再将该站待运包裹送到下一站。一个可以运送的包裹是指可被某司机运送到目的地而不超过司机的工作时间范围的包裹。司机的工作时间由他开始运送第一个包裹算起，到运送完最后一个包裹为止。公司希望司机在工作时间内尽可能多送包裹。

程序设计要求为司机找出单个工作日内的最佳路线。司机最初的出发地都为 A 站，司机运送包裹有如下条件：

(1) 司机工作时间不超过 10 小时，但不限开始工作的时间。
(2) 每个包裹都有一个计划运送时间，在此时间之前则包裹尚未出现。
(3) 司机在 A 站开始运送的第一个包裹，应是 A 站未被运送包裹中的最早出现的一个。
(4) 一次只能运送一个包裹，途中不准放下，而且，包裹必须从指定的起点站运送到终点站，途中不得经过其它站。如果在某站中暂时没有可运送包裹，则他会去另一个有可运送包裹的站送包裹（空车行走时间不计入运送时间）。司机也可以等某车站的任一个未被运送包裹。
(5) 选择路线时，应选择累计运送时间最长的那条路线。若有多条路线累计最长运送时间相等，则选择使司机有最短工作时间的那条路线。
(6) A 站包裹一定要运送，且运完 A 站包裹即可，其它站的包裹则不一定要运送。

在 A 站运送第一个可运送包裹的司机的最佳路线，要在考虑下一个司机前确定，如此类推，直到 A 站所有包裹都被按计划运送。不能运送包裹要报告出来。

输入格式：

从键盘输入两个文本文件的文件名。第一个为输入数据文件名，第二个为输出结果文件名。所有文件每一行中的分隔符均为单个空格，行的分隔符均为单个回车。

第一行有一个整数 n （1≤ n≤500），表示包裹的数量。

接下来 n 行中每一行描述一个包裹的情况，格式为：

包裹标识号　起点站　终点站　时间

其中包裹标识号为自然数，站名为单个大写字母，时间指包裹计划运送时间，格式为 hhmm　　（hh 表示小时，范围从 00 到 23，mm 表示分钟，范围从 00 到 59）

接下来一行有一个整数 s（1≤ s≤26），表示站的总数。

接下来 s*(s-1)/2 行描述任意两站之间运送时间，格式为：

站 1　站 2　时间

其中时间表示行走此两站间所花费时间。

输出格式：

D　　（所需司机数）

Q　　（Q 表示司机 1 行走路段总数，以下 Q 行为这些路段的描述，再加两行有关时间）

n A B　（包裹#n 从 A 运到 B，若 n=0 则表示空车从 A 到 B）

……
hhmm　　（总运送时间 hh 小时 mm 分）
hhmm　　（总工作时间 hh 小时 mm 分）
……　　（其它司机运送路线和时间）
P　　（P 表示不能运送包裹总数，以下 P 行为这类包裹的描述）
n B　　（B 站包裹#n 不能被运送）

输入输出举例：

输入文件：SAMPLE3.DAT

```
7
1 A B 0800
3 A C 0850
2 B C 0700
6 B D 1250
5 B C 1400
7 C A 1600
8 D C 1130
4
A B 0400
A C 0135
A D 0320
B C 0345
B D 0120
C D 0200
```

输出文件：SAMPLE3.OUT

```
2          （需要 2 个司机）
3          （司机 1 的路段数为 3）
1 A B      （包裹#1 从 A 运到 B）
2 B C      （包裹#2 从 B 运到 C）
7 C A      （包裹#7 从 C 运到 A）
0920       （总运送时间 9 小时 20 分）
0935       （总工作时间 9 小时 35 分）
3          （司机 2 的路段数为 3）
3 A C      （包裹#3 从 A 运到 C）
0 C B      （司机从 C 到 B）
5 B C      （包裹#5 从 B 运到 C）
0520       （总运送时间 5 小时 20 分）
0905       （总工作时间 9 小时 05 分）
2
8 D        （D 站的#8 包裹不能被运送）
6 B        （B 站的#6 包裹不能被运送）
```

7、计算机网络连接

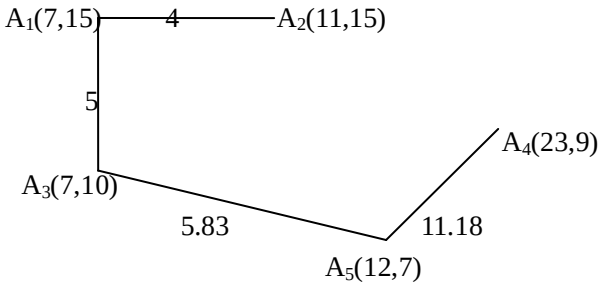
本题取材于 ICPC 92B 题。

问题描述：

要将 n(n≤30)台计算机连成链式网络，这 n 台计算机分别表示为 A_1，A_2，……，A_n。连接方法是除链的两端的两台计算机只与 1 台计算机相连外，其余各计算机都只与两台计算机相连。连接两台计算机的电缆长度即为这两台计算机之间的距离长度。

现求一种连接方式，使这 n 台计算机连成网络所需电缆总长度最短（保留两位小数）。

如图，有 5 台计算机，以下的连接方式使所需电缆总长度最短。



MIN=(4+5+5.83+11.18)=26.01

输入格式：

从键盘输入一个文本文件的文件名。该文件第一行有 1 个数 n，以下 n 行每行有两个实数，是计算机 Ai（1≤ i≤n）位置的坐标。在每一行中，数据之间只有一个空格。每行的行首、行

末无多余空格。

输出格式：

输出到文本文件中，文件名由键盘输入。文件的第一行是电缆的最短长度（保留两位小数），第二行是连接方案，由一个数字序列组成，数列中相邻数字 i、j 对应的计算机 Ai、Aj 有电缆直接相连。若有多种方案，只需输出其中任意一种。在每一行中，数据之间只有一个空格。每行的行首、行末无多余空格。

输入输出举例：

SAMPLE4.DAT

```
5
7 15
11 15
7 10
23 9
12 7
```

SAMPLE4.OUT

```
26.01
2 1 3 5 4
```

8、联系圈

本题取材于 ICPC 96b 题。

问题描述：

现在移动电话公司正在大打价格战。许多公司都竞相以低价推出新服务。一家公司推出了一项新服务，叫“联系圈”。你只须向电话公司提交一份你经常联系的人的名单。那么当你给名单中的人打电话时，电话费减半，而打电话给名单外的人时电话费照旧。该名单被称为“联系圈”。而其它公司指出，只有当你给名单内的人打电话时才会有优惠，如果你要改变你经常联系的人是最好把他们也加入联系圈。

重点电话公司是一家新成立的公司，他们希望能打垮其他竞争对手。重点电话公司也提供联系圈服务，但他们会自动为客户建立联系圈。他们见所有的电话联系记录下来，并根据这些记录来建立联系圈。对你而言，你的联系圈将包括曾打电话给你而你又找过他的人，无论是间接的还是直接的。例如，张三打电话给李四，李四打电话给王五，王五打电话给张三，则张三曾经间接找过王五，李四也间接找过张三，王五也间接找过李四。于是这三个人都在同一个联系圈中。如果王五打电话给刘六，刘六又曾给王五打电话，则刘六与王五就在同一个联系圈中。若王五给何七打电话，而何七没有打电话给张三、李四、王五、刘六，则何七就不在这个联系圈中。

假设你是重点电话公司的雇员，请你编程根据电话记录找出其中的所有联系圈。

输入格式：

从键盘输入一个文本文件的文件名。

该文本文件第一行为两个用空格分开的整数 n 和 m。其中 n（n≤25)表示电话记录中出现的人数，m(m≤150)表示该电话记录中有 m 次电话联系。接下来 m 行。每行表示一个电话联系。每个电话联系有两个人名组成，中间用空格分开，表示第一个人打电话给第二个人。

人名区分大小写，由 1～25 个字母或数字构成。例如：

Zhang3 Lisi4

表示 Zhang3 打电话给 Lisi4。

输出格式：

答案输出到一个文本文件中，文件名由键盘输入。

文件中每一行包含若干个人名（不少于一个），表示一个联系圈，人名之间用一个空格分开。

输入输出举例：

输入：SAMPLE5.DAT

```
5 6
Zhang3 Lisi4
Lisi4 Wang5
Wang5 Zhang3
Wang5 Liu6
Liu6 Wang5
Wang5 He7
```

输出：SAMPLE5.OUT

```
Zhang3 Lisi4 Wang5 Liu6
He7
```

9、球钟

本题取材于 ICPC 95B 题。

问题描述：

球钟是一个利用球的移动来记录时间的简单装置。它有三个可以容纳若干个球的指示器：分钟指示器，五分钟指示器，小时指示器。若分钟指示器中有 2 个球，五分钟指示器中有 6 个球，小时指示器中有 5 个球则时间为 5：32。

球钟的工作原理如下：分钟指示器最多可容纳 4 个球。每过一分钟，球钟就会从球队列的队首取出一个球放入分钟指示器，当放入第五个球时，在分钟指示器的 4 个球就会按照他们被放入时的相反顺序加入球队列的队尾。而第五个球就会进入五分钟指示器。按此类推，五分钟指示器最多可放 11 个球，小时指示器最多可放 11 个球。当小时指示器放入第 12 个球时，原来的 11 个球按照他们被放入时的相反顺序加入球队列的队尾，然后第 12 个球也回到队尾。这时，三个指示器均为空，回到初始状态，从而形成一个循环。因此，该球钟表示时间的范围是从 0：00 到 11：59。

现设初始时球队列的球数为 x（27 ≤ x ≤ 127),球钟的三个指示器初态均为空。问要经过多少天（每天 24 小时），球钟的球队列才能回复原来的顺序。

输入格式：

从键盘输入 x。

输出格式：

答案输出到屏幕。以整数形式输出所需天数。

输入输出举例：

输入：从键盘输入

```
30
```

输出：屏幕显示

```
15
```

10、建筑物

本题取材于 ICPC 96C 题。

问题描述：

在一个矩形地图里，有若干个矩形建筑物(矩形的边不一定与坐标轴平行)，现要求计算给定两点间的最短距离，计算时有如下限制：

(1) 不能穿过建筑物。

(2) 建筑物有两种：规则建筑物和不规则建筑物。规则建筑物是矩形建筑；不规则建筑物是由几个矩形建筑群重叠而成，如图 1 中，两个矩形建筑群重叠成一个不规则建筑物。

(3) 若两个建筑物的外墙相邻，容许人沿着这两建筑物之间的墙边穿过，如图 2，人可沿墙 AB 走。
(4) 起点及终点不会设在建筑物内部。
(5) 从起点到终点一定会有线路连接。
(6) 结果精确到小数点后两位。

图 1

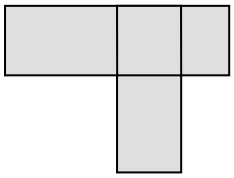
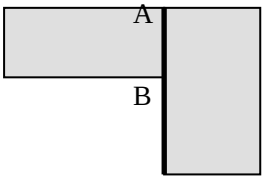


图 2



输入格式：

从键盘输入一个文本文件名。文件内容如下：
第一行为矩形建筑物的个数 n(0≤n≤20)。
第二行 X1 Y1 X2 Y2，其中(X1,Y1)为起点坐标，(X2,Y2)为终点坐标。
第三行开始的 n 行为 X1 Y1 X2 Y2 X3 Y3，表示一个矩形建筑物三个顶点的坐标。
每个坐标值为 0～1000 之间的一个实数。

输出格式：

route distance:xxx.xx（xxx.xx 为给出两点的距离）

输入输出举例：

输入：SAMPLE8.DAT

```
5
6.5 9 10 3
1 5 3 3 6 6
5.25 2 8 2 8 2 3.5
6 10 6 12 9 12
7 6 11 6 11 8
10 7 11 7 11 11
```

输出：屏幕显示

```
route distance:7.28
```

第二节 部分习题测试数据及参考答案

1、电子表格（Table）

测试数据：

TABLE1.DAT

```
5 3
1
2
3
A0+10
A1+10
A2+10
B0+10
B1+10
B2+10
C0+10
C1+10
C2+10
D0+10
D1+10
D2+10
```

TABLE2.DAT

```
5 3
-1
-2
-3
A0-10
A1-10
A2-10
B0-10
B1-10
B2-10
C0-10
C1-10
C2-10
D0-10
D1-10
D2-10
```

TABLE3.DAT

```
3 4
```

B1
B1+B2
B1+B2+B3
0
C1
C1+C2
C1+C2+C3
1
A1
A1+A2
A1+A2+A3
1

TABLE4.DAT
3 4
B1
B1+B2
B1+B2+B3
0
C1
C1+C2
C1+C2+C3
1
A3
A3+C0
A3
1

参考答案：

TABLE1.OUT

```
     1       2       3
    11      12      13
    21      22      23
    31      32      33
    41      42      43
```

TABLE2.OUT

```
    -1      -2      -3
   -11     -12     -13
   -21     -22     -23
   -31     -32     -33
   -41     -42     -43
```

TABLE3.OUT

A0:B1
A1:B1+B2
A2:B1+B2+B3
B0:C1
B1:C1+C2
B2:C1+C2+C3
C0:A1
C1:A1+A2
C2:A1+A2+A3

TABLE4.OUT

```
0    1    2    0
0    0    1    1
0    0    0    1
```

2、DEL 命令 (DEL)

测试数据：

DEL1.DAT
+RARFILES.LST
+RAR.INI
+WINRAR.HLP
-RAR.EXE
-RCVT.EXE
-UNRAR.EXE
-WINRAR.EXE
-WINRAR95.EXE
+RCVT.CFG

DEL2.DAT
+RARFILES.LST
+RAR.INI
-WINRAR.HLP
+RAR.EXE
+RCVT.EXE
+UNRAR.EXE
-WINRAR.EXE
-WINRAR95.EXE
+RCVT.CFG

参考答案：

DEL1.OUT

```
del *.exe
```

DEL2.OUT
DEL WINRAR*.*

3、分割方格 (Divide)
测试数据：
DIVIDE1.DAT
4
....
****
....
....

DIVIDE2.DAT
4
..
****
***.
....

DIVIDE3.DAT
8
....
....
****
.....
.....
.....
.....
.....

DIVIDE4.DAT
6
..**..
.***.*
******
..****
.....
.....

DIVIDE5.DAT
9
...*****.
...*.
....***

..*..
******....
....
....
....
....

DIVIDE6.DAT
7
.*.*...
.****...
.****..
...
.....
******..
******..

DIVIDE7.DAT
10
****
...***
*....*****
**...*.*..
**...*....
******....
....
....
....
....

DIVIDE8.DAT
9
.*.......
.*..*....
.******...
.****.....
******...
.....
***..*...
***.**...
******...

参考答案：
DIVIDE1.OUT

....
AABB
....
....

DIVIDE2.OUT
A..B
AAAB
AAA.
....

DIVIDE3.OUT
AA..BB..
AA..BB..
AAAABBBB
.....
.....
.....
.....
.....

DIVIDE4.OUT
..AA..
.AAA.B
AAAABB
..BBBB
.....
.....

DIVIDE5.OUT
...AABBB.
AA.AA..B.
A...A.BBB
AA..A..BB
AAAAA....
.....
.....
.....
.....

DIVIDE6.OUT
.A.A...
.AAA...

.AAAB..
BA.BB..
B...B..
BBBBB..
BBBBB..

DIVIDE7.OUT
AAAAAABBBB
AA...ABBBB
A....ABBBB
AA...A.B..
AA...A....
AAAAAA....
.....
.....
.....
.....

DIVIDE8.OUT
.A.....
.A..A....
.AAAAA...
.AAA.....
BAAABB...
BA..BB...
BBB..B...
BBB.BB...
BBBBBB...

4、信息编码 (Decode)
测试数据：
DECODE1.DAT
A
00101000

DECODE2.DAT
abb bba
0010010100001101101100000101001101111000

DECODE3.DAT
The robbes rokintteanadsl$5000.
0010101000
0110110110
0000101001

1100000101
1100101111
0000000001
0010111101
1100111010
1011100001
1010001011
1110110011
1101010111
0001101111
0100011100
0111111101
1110010111
1001000100
1001011110
1010111001
0100100101
1111101010
1110011000
1100001110
1111101110
0110111100
1110111110
1000000000
1000100000
1000110010
011111000

DECODE4.DAT
Signng the agremt wsoneth,biint/puting/crryin fcwaoh.
0010101000
0110001101
1000001010
0111001011
1011110000
0000010010
1111011101
1111000011
0100111101
1101111010
1011100010
1011001110
0001000011

0100110101
0111111011
0101111001
1001101111
1010001110
0101011110
1001111001
1100110111
1101000001
1111100001
0111110100
0011111101
1000111100
0001111110
1000100001
1001000010
1001100011
1010001111
1011011111
1010100101
0100101101
1000110111
1111000000
1111101011
1001111100
0010001100
1011111100
0001111110
1100111111
1010001101
1000011111
1001001111
1101100111
1111100001
1111110110
1001010011
1111000011
1111101101
0111111011
0111101111
0110110111
1110000001
0001111101
1001110111

0001111000
0100011001
1111101110
1111100001
0111110111
0101111100
0

参考答案：

DECODE1.OUT
A

DECODE2.OUT
aabb bbaa

DECODE3.OUT
The robbers broke into the bank and stole $50000.

DECODE4.OUT
Signing the agreement was one thing, bringint/putting/carrying into effect was another.

5、海上交通控制 (Lane)

测试数据：

LANE1.DAT
5
ABCDE
0 10 0 10 10
10 0 10 0 10
0 10 0 10 10
10 0 10 0 10
10 10 10 10 0
4
A
0000 1
AE
B
0
0
0
0

1

B

E

C
0000 1
CE
D
0000 1
DE

LANE2.DAT
6
BCDEFG
0 5 6 0 0 0
5 0 0 5 0 0
6 0 0 6 6 0
0 5 6 0 5 5
0 0 6 5 0 0
0 0 0 5 0 0
2
A
0000 1
BF
B
0007 1
BG

LANE3.DAT
6
ABCDEF
0 10 0 0 0 0
10 0 10 0 0 20
0 10 0 10 0 0
0 0 10 0 20 0
0 0 0 20 0 20
0 20 0 0 20 0
2
ShipA
0000 10
AD
ShipB
0130 20
BD

参考答案：

LANE1.OUT
A
0 1000
AE
B
0 1000
BE
C
0 1000
CE
D
0 1000
DE

LANE2.OUT
A
0 1200
BDF
B
7 1507
BCEG

LANE3.OUT
ShipA
0 300
ABCD
ShipB
130 430
BFED

6、投递最佳路线(Best Deliver)
测试数据：
T301.DAT
2
1 A B 0600
2 B C 0900
3
A B 0300
A C 0200
B C 0245

T302.DAT
5
1 B C 0825

```
2 C B 0935
3 C D 1200
4 D A 1300
5 C A 1400
4
A B 0200
A C 0215
A D 0300
B C 0330
B D 0400
C D 0300
```

```
T303.DAT
10
1 C B 1200
2 C D 1300
3 A B 0800
4 A C 0900
5 A D 1015
6 B E 1240
7 D E 1800
8 E C 1600
9 B E 1500
10 A B 1410
5
A B 0100
A C 0315
A D 0300
A E 0315
B C 0320
B D 0400
B E 0325
C D 0135
C E 0130
D E 0100
```

参考答案:

```
T301.OUT
1
2
1 A B
2 B C
0545
0545
```

```
0

T302.OUT
0
5
1 B
2 C
3 C
4 D
5 C

T303.OUT
3
4
3 A B
0 B A
4 A C
1 C B
0735
0835
3
5 A D
0 D C
2 C D
0435
0610
5
10 A B
6 B E
0 E D
7 D E
8 E C
0655
0755
1
9 B
```

7、计算机网络连接(computer network)

测试数据：

```
T401.DAT
6
0 0
0 10
10 0
```

```
10 10
3 5
7 5
```

T402.DAT

```
10
302 150
301 988
141 18
931 96
102 92
427 199
612 112
312 731
724 492
143 683
```

T403.DAT

```
15
940 121
683 376
511 13
266 534
647 18
912 886
80 923
974 34
768 989
833 12
330 291
537 205
72 53
424 93
165 713
```

T404.DAT

```
17
893 916
5 223
877 715
724 62
361 973
245 940
793 637
```

556 856
153 294
797 256
204 123
445 180
133 786
200 857
498 861
607 11
665 689

T405.DAT
29
100 100
98 101
98 105
96 105
96 109
94 109
94 113
92 113
92 117
90 117
90 127
87 122
83 122
80 127
80 117
98 99
98 95
96 95
96 91
94 91
94 87
92 87
92 83
90 83
90 73
87 78
83 78
80 73
80 83

参考答案：

序号	测试文件名	最短长度	连接方案
1	T401.DAT	33.32	1 5 2 4 6 3
2	T402.DAT	3631.34	1 3 5 2 6 7 4 9 8 10
3	T403.DAT	4902.60	9 3 1 11 4 2 12 5 6 14 8 7 15 13 10 10
4	T404.DAT	6161.62	1 3 9 2 4 10 6 5 15 8 7 16 12 11 17 14 13 12
5	T405.DAT	119.12	15 13 14 11 12 10 9 8 7 6 5 4 3 2 1 16 17 18 19 20 21 22 23 24 26 25 28 27 29 15

8、联系圈(Circle)

测试数据：

T501.DAT
5 6
Ben Alexander
Alexander Dolly
Dolly Ben
Dolly Benedict
Benedict Dolly
Alexander Aaron

T502.DAT
5 4
Ben Dolly
Dolly Alexander
Alexander Aaron
Aaron BEN

T503.DAT
11 19
Tom David
David John
John David
Doris John
Doris Tom
John Doris
Mary Doris
Doris Mary
Mary Daniel
Daniel John
David Charles
Charles Rose
Daniel Rose

```
Rose Stephen
Stephen Anne
Anne Rose
Paul Rose
Paul Anne
Paul Stephen

T504.DAT
14 34
John Aaron
Aaron Benedict
Betsy John
Betsy Ringo
Ringo Dolly
Benedict Paul
John Betsy
John Aaron
Benedict George
Dolly Ringo
Paul Martha
George Ben
Alexander George
Betsy Ringo
Alexander Stephen
Martha Stephen
Benedict Alexander
Stephen Paul
Betsy Ringo
Quincy Martha
Ben Patrick
Betsy Ringo
Patrick Stephen
Paul Alexander
Patrick Ben
Stephen Quincy
Ringo Betsy
Betsy Benedict
Betsy Benedict
Betsy Benedict
Betsy Benedict
Betsy Benedict
Betsy Benedict
Quincy Martha
```

参考答案：

T501.OUT

Ben Alexander Dolly Benedict

Aaron

T502.OUT

Ben

Dolly

Alexander

Aaron

BEN

T503.OUT

Tom David John Doris Mary Daniel

Charles

Rose Stephen Anne

Paul

T504.OUT

John Betsy Ringo Dolly

Aaron

Benedict

Paul George Martha Ben Alexander Stephen Quincy Patrick

9、球钟(Ball Clock)

序号	测试数据 x	参考答案
1	27	23
2	45	378
3	87	570
4	88	1026
5	127	2415

10、建筑物(Buildings)

测试数据：

T801.DAT

2

12 1 9 12

2 10 2 2 8 2

16 9 16 1 20 1

T802.DAT

2

12 1 1 12

2 10 2 2 8 2

6 9 6 1 10 1

T803.DAT
5
1 1 100.0000 100.0000
111.6472 55.2142 111.6472 32.6734 75.0000 32.6734
61.3945 28.4794 61.3945 2.9911 34.2711 2.9911
80.0280 105.9668 80.0280 84.9887 68.0690 84.9887
75.0000 60.9854 75.0000 49.6019 45.5433 49.6019
46.3908 51.1279 46.3908 34.0599 14.7642 34.0599

T804.DAT
10
29.5 77 30 60
36.0342 44.7828 36.0342 20.0708 17.7682 20.0708
17.3354 53.4266 17.3354 41.1454 5.5672 41.1454
48.4050 79.8048 48.4050 58.6839 31.2193 58.6839
56.8995 103.3591 56.8995 82.0954 31.2887 82.0954
34.2183 76.5654 34.2183 63.5331 6.6992 63.5331
95.8551 58.9865 95.8551 25.6447 83.8405 25.6447
74.4889 99.9201 74.4889 71.3542 41.0628 71.3542
90.5680 23.8397 90.5680 4.7514 76.2361 4.7514
53.0420 90.1295 53.0420 78.1667 27.4900 78.1667
25.6917 65.6428 25.6917 42.2115 14.7418 42.2115

T805.DAT
20
120 10 55 55
8 102 12 102 12 8
8 12 102 12 102 8
98 102 102 102 102 8
18 102 102 102 102 98
18 102 22 102 22 18
18 22 92 22 92 18
88 92 92 92 92 18
28 92 92 92 92 88
28 92 32 92 32 28
28 32 82 32 82 28
78 82 82 82 82 28
38 82 82 82 82 78
38 82 42 82 42 38
38 42 72 42 72 38
68 72 72 72 72 38
48 72 72 72 72 68

48 72 52 72 52 48
48 52 62 52 62 48
58 62 62 62 62 48
58 62 62 62 62 58

T806.DAT
20
1 1 1000.0000 1000.0000
1117.8977 905.2469 1117.8977 834.7683 852.6140 834.7683
699.8309 218.3214 699.8309 35.6745 590.3002 35.6745
1067.3609 820.5676 1067.3609 691.1044 795.5880 691.1044
532.9138 783.1889 532.9138 503.6511 485.2861 503.6511
522.5601 876.7385 522.5601 797.8878 288.0793 797.8878
740.7732 425.8293 740.7732 379.8234 674.0581 379.8234
881.9434 281.9053 881.9434 189.1167 659.1033 189.1167
1086.1477 327.9686 1086.1477 147.3035 895.7839 147.3035
1010.1165 503.4139 1010.1165 422.8801 785.3734 422.8801
683.8065 343.3370 683.8065 89.3137 628.7605 89.3137
139.8945 286.7581 139.8945 70.3671 71.3084 70.3671
840.8830 459.4598 840.8830 413.1969 763.0644 413.1969
588.6153 982.0259 588.6153 913.9856 420.4016 913.9856
698.7955 437.4680 698.7955 177.8584 454.7944 177.8584
932.4232 638.4026 932.4232 592.7942 714.6002 592.7942
203.0113 353.1353 203.0113 284.0835 38.9513 284.0835
307.0832 188.1511 307.0832 70.5835 220.0060 70.5835
805.4527 464.5107 805.4527 304.0195 589.8085 304.0195
737.5763 574.5983 737.5763 451.8739 668.5685 451.8739
785.6019 681.0839 785.6019 618.8627 514.7661 618.8627

T807.DAT
20
1 1 1000.0000 1000.0000
1018.0849 205.0535 867.9989 158.2981 914.7543 8.2121
614.3623 679.2637 382.3315 713.5203 348.0749 481.4895
973.4459 536.1402 743.9176 586.4856 693.5722 356.9573
739.0365 312.5855 674.7261 331.6758 655.6358 267.3654
223.3707 688.8061 88.1009 698.6129 78.2941 563.3431
926.0393 1072.4041 812.8962 1001.9959 883.3044 888.8529
875.4617 575.7501 758.3160 537.1502 796.9159 420.0044
690.5170 834.1378 478.2052 875.7887 436.5544 663.4769
496.2192 826.2228 351.9041 809.0691 369.0578 664.7540
655.1866 448.5499 566.2255 441.9342 572.8412 352.9730
516.5254 298.0941 397.6234 259.9654 435.7521 141.0633
237.4385 540.9028 80.4770 537.9380 83.4419 380.9765

743.8112 861.6758 492.8600 839.8589 514.6769 588.9077
270.6601 901.3953 182.1354 863.4667 220.0640 774.9419
989.8902 374.9086 792.9399 367.1133 800.7352 170.1630
946.0774 1082.6547 797.2414 1040.0183 839.8779 891.1823
337.2085 831.5292 259.9863 859.0532 232.4623 781.8310
387.6393 432.3340 222.1013 345.4783 308.9571 179.9403
116.2868 178.3885 30.7750 195.3121 13.8514 109.8003
487.7590 521.0876 299.5330 452.6732 367.9474 264.4472

参考答案：

序号	测试文件名	参考答案 最短距离
1	T801.DAT	11.40
2	T802.DAT	17.76
3	T803.DAT	145.96
4	T804.DAT	141.95
5	T805.DAT	818.06
6	T806.DAT	1505.87
7	T807.DAT	1525.39

第六章 习题解答

第一节 电子表格（Table）

算法分析：

本题要求计算电子表格各单元的值，求单元的值实际上就是对各单元的表达式进行求值。由于题目要求中各单元的表达式只能是加减表达式，因此题目对表达式的运算要求并不高，其重点在于单元格的引用。

由于各单元格可以互相引用，就不可避免地产生单元格的循环引用，也就是说某单元格的表达式直接或间接引用自己。

在例程中，采用递归求值的方法求解：先递归求出单元格表达式中所引用单元的值，再进行加减运算计算本单元格的值。为了简化运算，我们可以把已求出值的单元和已知不能求值的单元记录下来，以后如果需要求该单元的值时，就可以直接得到结果，从而避免了重复的运算。

递归求值的过程描述如下：

```
function value:integer;
 begin
  如已求值，则直接返回该值;
  如已知该单元不可求值，则直接返回不可求值;
  r:=0;
  while 表达式还有单词 do{对单元求值}
  {单词表示表达式中的运算符、数值或单元格名称}
   begin
    取出当前的运算符作为当前项的符号;
    当前单词推进一项;
    if 当前项是数字 then 求出对应的数值->s
    else  递归求当前单元的值->s;{当前项是单元变量}
    if 不能求值 then 返回不可求值;
    根据符号将r加或减s;
   end;
  value:=r;
  返回可以求值;
 end;
```

例如对输入样例Table.Dat中的A0：A1+B1其求值过程为：

```
        A0
      /   \
    A1  +   B1
    |       /  \
    5      B0 - A1
           |    |
           3    5
```

程序分析：

```
program GDOI98_1_Table;{电子表格}
type tsheet=array['A'..'T','0'..'9'] of integer;
     tinit=array['A'..'T','0'..'9'] of string;
     tuse=array['A'..'T','0'..'9'] of boolean;
var sheet:tsheet;{存储电子表格各个单元的值}
    init:tinit;{存储电子表格各个单元的算式}
    r,c,i,j,k,l:char;
    m,n,t:integer;
    f,f2:text;
    s:string;
    use:tuse;{表格各个单元能否被计算}
    calced:tuse;{表格各个单元是否已被计算}
    cancalc:boolean;
{--------------------------------------------------------------------}
function value(a,b:char):integer;
{求表格单元（a，b）的值，如可求值返回该值并置 cancalc 为 true,
 不可求值置 cancalc 为 false}
 var i,j:char;
     l,r,s,c,t:integer;
     e:string;
 begin
  if calced[a,b] then begin cancalc:=true;value:=sheet[a,b];exit;end;
  {如已求值，则直接返回该值}
  if use[a,b] then begin cancalc:=false;exit;end;
  {如已知该单元不可求值，则直接返回不可求值}
  l:=0;
  use[a,b]:=true;
  r:=0;
  while l<=length(init[a,b]) do{对单元求值}
   begin
    if (l=0)then
     begin
      if not (init[a,b,l] in ['+','-']) then c:=1 else
       begin
        inc(l);
```

```
        if init[a,b,l]='+' then c:=1 else c:=-1;
       end;
     end
    else if init[a,b,l]='+' then c:=1 else c:=-1;
    { c 表示当前项的符号}
    if init[a,b,l+1] in ['0'..'9'] then
     begin
      inc(l);e:='';
      while (l<=length(init[a,b]))and(init[a,b,l] in ['0'..'9']) do
       begin
        e:=e+init[a,b,l];inc(l);
       end;
      val(e,s,t);
      cancalc:=true;
     end
     {当前项是数值}
    else
     begin
      s:=value(init[a,b,l+1],init[a,b,l+2]);
      {递归求当前单元的值}
      inc(l,3);
     end;
     {当前项是单元变量}
    if not cancalc then exit;
    r:=r+c*s;
   end;
  value:=r;
  use[a,b]:=false;
  calced[a,b]:=true;
  sheet[a,b]:=r;
 end;
{-------------------------------------------------------}
function calc:boolean;
{电子表格各单元是否都能算出}
 begin
  calc:=true;
  for i:='A' to r do for j:='0' to c do if not calced[i,j] then
   begin
    for k:='A' to r do for l:='0' to c do use[k,l]:=false;
    sheet[i,j]:=value(i,j);
    if not cancalc then calc:=false;
    {任一单元不能算出，则最后返回 false}
   end;
 end;
```

```
begin
 assign(f,'table.dat');
 reset(f);
 readln(f,m,n);
 r:=chr(m+64);{最后一行代号}
 c:=chr(n+47);{最后一列代号}
 for i:='A' to r do
   for j:='0' to c do
    begin
     readln(f,s);
     init[i,j]:='';
     for t:=1 to length(s) do
      if s[t]<>' ' then init[i,j]:=init[i,j]+s[t];
     val(init[i,j],sheet[i,j],t);
     if t<>0 then calced[i,j]:=false else calced[i,j]:=true;
    end;
 close(f);{输入电子表格各单元的值或算式}
 assign(f2,'table.out');
 rewrite(f2);
 if calc then
  begin
   for i:='A' to r do
    begin
     for j:='0' to c do write(f2,sheet[i,j]:6);
     writeln(f2);
    end;
  end
  {全部单元算出，输出各单元的值}
 else
   for i:='A' to r do for j:='0' to c do
     if not calced[i,j] then writeln(f2,i,j,':',init[i,j]);
  {有单元不能算出，输出这些单元的算式}
 writeln(f2);
 close(f2);
end.
```

第二节 DEL 命令 (DEL)

算法分析：

找出适当的DEL命令，就是从给定要删除的文件名中找出相同的部分。

因为文件名可以有11个字符，因此对每一个字符位分别建立一个删除字符集合和保留字符集合。文件名可以分解成以点号为分隔符两个部分：文件名（前8个字符）和扩展名（后3个字符），现在先讨论点号前由8个字符组成的文件名。首先在每个文件名后面加空格补足8

个字符。根据文件保留或删除将各个字符加入相应的保留或删除字符集合。对所有文件名都处理完后，根据各个字符位的删除的字符集合和保留的字符集合构造包含通配符的文件名。

算法简要描述如下：

```
构造各个字符位的删除的字符集合和保留的字符集合：(在集合中用空格代表文件名结束)
for 文件名的每个字符 do
 if 该文件要保留 then 保留的字符集合：=保留的字符集合+[当前字符]
                else 删除的字符集合：=删除的字符集合+[当前字符];
根据各个字符位的删除的字符集合和保留的字符集合构造包含通配符的文件名：
  for 文件名的8个位置 do
     begin
   if (删除的字符集合元素个数=1) then
    begin
     if (该元素<>' ') then
      begin
       该位置：=该元素;
       if 该元素不在保留的字符集合中 then
       {该元素已经可以唯一的代表要删除的文件集合}
        begin
          下一位置为'*';
                    退出构造;
        end;
      end
     else 该位置:=' ';
    end
   else{集合中有多个字符，则只能用？和*}
    if (空格不在删除的字符集合) then 该位置:='?'{? 不能代表没有字符}
    else 该位置:='*';
  end;
```

完成包含通配符的文件名的构造后，用相同的方法构造包含通配符的扩展名。最后，用构造出来的文件名和扩展名对原来输入的文件名进行检验，假如成功则输出DEL命令，否则输出'IMPOSSIBLE'(因为这样构造的文件名和扩展名已经包含最多的信息，如果仍不能匹配，只能认为输入的要求是无法实现的)。

程序分析：

```
program GDOI98_2_Del;{DEL 命令}
type t1=array[1..3500] of string[11];
     t2=set of char;
     t3=array[1..11] of t2;
var fin,fout:text;
    names:t1;{存储所有的文件名}
    del:array[1..3500] of boolean;
```

```
        {对应的文件要保留(false)还是要删除(true)}
    s,s0,s1,s2,sa,sb:string;
    i,j,k,n:integer;
    set1:t3;{要删除的文件名的字符集合}
    set2:t3;{要保留的文件名的字符集合}
    last:char;
function count(s:t2):integer;
{计算集合 s 中元素的个数，并把最后一个元素放在 last 中}
 var c:char;
     t:integer;
 begin
  t:=0;
  for c:=' ' to 'Z' do
   if c in s then
    begin inc(t);last:=c;end;
  count:=t;
 end;
function match(s1,s2:string):boolean;
{带通配符的文件名 s1 是否与 s2 匹配}
 var i:integer;
 begin
  if (s1='') then
   begin
    if (s2='') then match:=true else match:=false;
    exit;
   end;
  for i:=1 to length(s1) do
   begin
    if s1[i]='*' then
     begin
      match:=true;
      exit;
     end
    else
     begin
      if (length(s2)<i) or((s1[i]<>'?')and(s1[i]<>s2[i])) then
       begin
        match:=false;
        exit;
       end;
     end;
   end;
  match:=true;
 end;
```

```
function ok:boolean;
{判断产生的带通配符的文件名是否符合要求}
 var m1,m2:boolean;
 begin
  ok:=false;
  for j:=1 to n do
   begin
    s1:=copy(names[j],1,8);
    i:=pos(' ',s1);
    if i<>0 then s1:=copy(s1,1,i-1);
    s2:=copy(names[j],9,3);
    i:=pos(' ',s2);
    if i<>0 then s2:=copy(s2,1,i-1);
    m1:=match(sa,s1);{文件名是否匹配}
    m2:=match(sb,s2);{扩展名是否匹配}
    if((not del[j])and(m1 and m2))or((del[j])and((not m1) or (not m2)))
     then exit;
   end;
  ok:=true;
 end;
begin
 for i:=1 to 11 do
  begin
   set1[i]:=[];
   set2[i]:=[];
  end;
 assign(fin,'Del.Dat');
 reset(fin);
 j:=0;
 while not eof(fin) do
  begin
   readln(fin,s0);
   if s0<>'' then
    begin
     inc(j);
     s:='';
     for i:=1 to length(s0) do
      if s0[i]<>' ' then s:=s+s0[i];
     {去掉输入中的无用空格}
     i:=pos('.',s);
     s1:=copy(s,2,i-2);
     s2:=copy(s,i+1,255);
     {将文件名分成文件名和扩展名两部分}
     if s[1]='+' then del[j]:=false else del[j]:=true;
```

```
    s0:=s1;
    for i:=length(s1) to 7 do s0:=s0+' ';
    s0:=s0+s2;
    for i:=length(s0) to 10 do s0:=s0+' ';
    if s[1]='-' then
     for i:=1 to 11 do
      set1[i]:=set1[i]+[s0[i]]
    else
     for i:=1 to 11 do
      set2[i]:=set2[i]+[s0[i]];
    {根据是否保留将字符放入相应的集合}
    names[j]:=s0;
   end;
 end;
close(fin);
n:=j;
sa:='          ';
for i:=1 to 8 do{对文件名的字符集合进行处理}
 begin
  if (count(set1[i])=1) then
   begin
    if (last<>' ') then
     begin
      sa[i]:=last;
      if not (last in set2[i]) then
       begin
        if i<>8 then sa[i+1]:='*';
        break;
       end;
     end
    else sa[i]:=' ';
   end
  else
   if not (' ' in set1[i]) then sa[i]:='?'
   else sa[i]:='*';
 end;
i:=pos('*',sa);{去掉多余的'*'号}
if i<>0 then sa:=copy(sa,1,i);
i:=pos(' ',sa);{去掉多余的空格}
if i<>0 then sa:=copy(sa,1,i-1);
sb:='          ';
for i:=9 to 11 do{对扩展名的字符集合进行处理}
 begin
  if (count(set1[i])=1) then
```

```
    begin
     if (last<>' ') then
      begin
       sb[i-8]:=last;
       if not (last in set2[i]) then
        begin
         if i<>11 then sb[i-7]:='*';
         break;
        end;
      end
     else sb[i-8]:=' ';
    end
   else
    if not (' ' in set1[i]) then sb[i-8]:='?'
    else sb[i-8]:='*';
  end;
 i:=pos('*',sb);{去掉多余的'*'号}
 if i<>0 then sb:=copy(sb,1,i);
 i:=pos(' ',sb);{去掉多余的空格}
 if i<>0 then sb:=copy(sb,1,i-1);
 assign(fout,'Del.Out');
 rewrite(fout);
 if ok then writeln(fout,'DEL ',sa,'.',sb)
       else writeln(fout,'IMPOSSIBLE');
 close(fout);{输出}
end.
```

第三节 分割方格 (Divide)

算法分析：

本题要解决的问题可以形象地看成有一张形状不规则的纸（纸的形状由带“*”的方格确定），我们手头上有一把剪刀，要把这张纸切成两半，这两半纸经过旋转、平移后，可以拼成一个正方形。这样，我们要解决的问题主要有两个：（1）怎样将这两张纸剪成两半。（2）剪成的这两半纸能否拼成一个正方形。

一、怎样将纸剪成两半。

要将一张纸剪成两半，我们可以这样考虑：从每次选取带“*”方格（以下简称方格）的一个组合，被选出的方格为“A”部分，没被选出的方格为“B”部分；然后检查“A”部分的方格是否连通，“B”部分的方格是否也连通；如果这两部分都各自连通，则说明这张纸以成功地剪成两半。例如，纸的形状如下：（不带“*”的部分已省略）

*	*	*	*

那么从中选取情况如下：

B	B	B	B

B	B	B	A

B	B	A	B

B	B	A	A

……

A	A	A	A

如果把 B 看成 0，把 A 看成 1，这些组合方案对应着二进制的 0000、0001、0010、0011、……、1111，这样我们只要产生这一系列的二进制数（对应的十进制数的范围为 0——2^n-1），就可对应出相应的组合。其算法可以简单描述如下：

```
For I:=0 to 2^n-1 do
  Begin
    For j:=0 to n-1 do
    If （I 对应二进制中的第 j 位为 1）then 相应的方格为 A
    Else 相应的方格为 B
  End;
```

判断 A 部分或者 B 部分是否连通，是图论中的判断图是否连通的问题。我们可以求出该图的连通分量（即最大连通子图的个数），若连通分量为 2，则说明该图恰被分成两个连通的部分 A 和 B。例如上面的举例中，连同分量分别为 1、2、3、2、……、1，所以只有 BBBA 和 BBAA 是符合剪纸要求的。其算法可以描述如下：

```
for I=结点 1 to 结点 n do
if not visited[I] then
  begin
    visit(I); {深度搜索，当然用广度搜索也行}
    inc(count)
  end;

procedure visit (m)
  begin
    visited[m]:=true;
    for I=结点 1 to 结点 n do
    if (I 没被访问过且 I 与 m 相邻) then visit(I);
  end;
```

由上面的分析，我们可以知道，这种组合的算法的运算量为 0（2^n），其中有很多的组合是无效的，为此，我们需要对这种取组合的方法进行改进，尽量避免无效的组合。具体做法是：设立一个相连方格链表 e，该链表中的方格都是连通的。每次选取方格时，总是从该链表中选取一个方格，并把与该方格相邻而又不在链表中的方格加入链表。这样每次选取的方格都可以保证其连通性，当然同时也应该用连通分量检查没被选取方格的连通性。也就是说，每一次的选取方格，都是一种符合剪纸要求的方案。其算法简单描述如下：

```
procedure search(m:integer)
begin
  for I=链表 e 中的一个元素 do
    begin
        对当前方案的处理;
        生成与 I 相连的方格，把不在链表 e 中的方格入链表 e;
        search(m+1);
        恢复链表 e;
    end;
end.
```

二、判断剪成的两部分纸是否能拼成一个正方体。

给 A 部分和 B 部分分别矢量化，也就是说取该部分的左上角的方格的坐标为（0，0），其他的单元格的坐标为与左上角方格坐标的相对位移。例如 BBBA，A 部分矢量化后为（0，0），B 部分矢量化后为（0，0），（0，1），（0，2）。这样使得每部分的方格的形状的描述与原位置无关，也就较好的解决了平移问题。

对于旋转的问题，可以利用坐标变换公式（x,y）—>（-y,x），在对各个方格进行旋转变换后，再进行一次矢量化，即可得到旋转 90 度后的方格形状的坐标描述。如果我们连续两次 90 度旋转，就可得到 180 度旋转；连续三次 90 度旋转，就可得到 270 度旋转。

类似地，利用翻转坐标变换公式（x,y）—>（x,-y）可以得到翻转后的方格形状描述。

在解决了平移、旋转、翻转后，剩下的问题是怎样用 A、B 两部分经过平移、旋转和翻转处理后的各种形状拼成正方形，具体做法如下，开辟一个 n*n 的正方形模板，先把 A 部分放入该模板中，并且使（0，0）和模板的左上角重合，再把 B 部分放入该模板，并且使（0，0）和模板中左上方的第一个空位重合。如果这两次铺放，都没有超出模板的边界，而且没有两个方格重叠，则说明成功地拼成正方形。同样地，先放 B，后放 A 也有可能成功。例如：BBBA 有一种拼法如下（左）：并不能拼成正方形，而 BBAA 的一种拼法如下（右），恰好拼成正方形。

B	B	B
A		

A	A
B	B

程序分析：

```
{$R-,S-,V-}
program GDOI98_4_Divide;
type tp=array [1..20,1..20] of byte;棋盘的数据类型
     ts=array [1..2] of shortint;    二维坐标数据类型
     ta=array [1..100] of ts;    坐标组数据类型，描述 A 或 B 部分的形状
var source,extend,ss:tp;
    a,b,c:ta;                  a 描述 A 部分形状，b 描述 B 部分的形状
    e:array [1..100] of ts;                    相连方格链表
    f:array [1..100] of ts;
    es,ep,m,n,i,j,en:integer;cc:char;          ep 链表可选方格的起始位置
    fin,fout:text;
```

```
    p:tp;

const di:array [1..4] of shortint=(1,-1,0,0);    方向的位移
      dj:array [1..4] of shortint=(0,0,1,-1);

function individual(p,n:integer):integer;    判断是否连通，采用广度搜索算法
  var i,j,r,t,t1,t2:integer;
  begin
    ss:=source;
    for i:=1 to p-1 do if source[e[i,1],e[i,2]]=2 then
      begin
        f[1,1]:=e[i,1];f[1,2]:=e[i,2];
        t:=0;r:=1; ss[e[i,1],e[i,2]]:=0;
        repeat
          inc(t);
          for j:=1 to 4 do
            begin
              t1:=f[t,1]+di[j];
              t2:=f[t,2]+dj[j];
              if (t1<1) or (t1>m) or (t2<1) or (t2>m) then continue;
              if ss[t1,t2]<>2 then continue;
              inc(r);
              f[r,1]:=t1;f[r,2]:=t2;
              ss[t1,t2]:=0;
            end;
        until t=r;
        if  r=n then begin individual:=0;exit; end;
        for j:=i+1 to p-1 do if ss[e[j,1],e[j,2]]=2 then
          begin
            individual:=2;exit;
          end;
        individual:=1;exit;
      end;
    individual:=0;
  end;

function order(var a:ta;var la:integer):boolean;     将a部分矢量化
  var i,j:integer;
      temp:ts;
  begin
              先排序，使左上角方格排在第一位
    for i:=1 to la do for j:=i+1 to la do
    if (a[i,1]>a[j,1]) or ((a[i,1]=a[j,1]) and (a[i,2]>a[j,2])) then
      begin
```

```
        temp:=a[i];
        a[i]:=a[j];
        a[j]:=temp;
      end;
    order:=true;
    求各方格与左上角方格的相对位移
    for i:=la downto 1 do
    begin
      dec(a[i,1],a[1,1]);
      if a[i,1]>n-1 then order:=false;
      dec(a[i,2],a[1,2]);
      if (a[i,2]<0) or (a[i,2]>n-1) then order:=false;
    end;
  end;

procedure revote(var a:ta;la:integer); 将a部分旋转90度
  var i,temp:integer;
  begin
    for i:=1 to la do
      begin
        temp:=a[i,1];
        a[i,1]:=a[i,2];
        a[i,2]:=-temp;
      end;
  end;

procedure recover(var a:ta;la:integer); 将a部分翻转
  var i,temp:integer;
  begin
    for i:=1 to la do
      begin
        temp:=a[i,2];
        a[i,1]:=a[i,1];
        a[i,2]:=-temp;
      end;
  end;

function block(var a:ta;var la:integer;var b:ta;var lb:integer):boolean;
                判断a、b两部分是否可以拼成正方形
  var i,j,t1,t2,t3,t4:integer;
  begin
    block:=false;
    fillchar(p,sizeof(p),0);
    for i:=1 to la do p[1+a[i,1],1+a[i,2]]:=1;
```

```
    for t1:=1 to n do for t2:=1 to n do if p[t1,t2]=0 then
      begin
        for i:=1 to lb do
          begin
            t3:=t1+b[i,1];t4:=t2+b[i,2];
            if (t3<1) or (t3>n) or (t4<1) or (t4>n) then exit;
            if p[t3,t4]<>0 then exit;
            p[t3,t4]:=2;
          end;
      end;
    block:=true;
  end;

function achieve(la:integer):boolean;    判断当前的分割方案是否能拼成正方形
  var i,j,lb,ii:integer;ba,bb:boolean;
  begin
    lb:=0;c:=a;
    for i:=1 to m do for j:=1 to m do
    if source[i,j]=2 then
    begin
      inc(lb);
      b[lb,1]:=i;
      b[lb,2]:=j;
    end;
    产生A和B部分的各种变化形式
    for ii:=1 to 2 do
    for i:=1 to 4 do
    begin
      if (i=1) and (ii=2) then recover(c,la);
      ba:=order(c,la);
      for jj:=1 to 2 do
      for j:=1 to 4 do
        begin
          if (jj=2) and (j=1) then recover(b,lb);
          bb:=order(b,lb);
          if ba then if block(c,la,b,lb) then
            begin
              achieve:=true;
              exit;
            end;
          if bb then if block(b,lb,c,la) then
            begin
              achieve:=true;
              exit;
```

```
            end;
          revote(b,lb);
        end;
      revote(c,la);
    end;
    achieve:=false;
  end;

procedure print;          打印结果
  var i,j:integer;
  begin
    rewrite(fout);
    for i:=1 to m do
      begin
        for j:=1 to m do
        case source[i,j] of
        0:write(fout,'.');
        1:write(fout,'A');
        2:write(fout,'B');
        end;
        writeln(fout);
      end;
    close(fout);
    halt;
  end;

procedure t(k,es:shortint);              搜索过程
  var i,j,ked,t1,t2,flag:integer;
  begin
    ked:=ep;
    for i:=es+1 to ked do
      begin
        a[k]:=e[i];
        source[a[k,1],a[k,2]]:=1;

        flag:=individual(i,n*n-k);
        if flag=2 then
        begin
          source[a[k,1],a[k,2]]:=2;
          continue;
        end;

        for j:=1 to 4 do
          begin
```

```
            t1:=e[i,1]+di[j];
            t2:=e[i,2]+dj[j];
            if (t1<1) or (t2<1) or (t1>m) or (t2>m) then continue;
            if (source[t1,t2]=0) or (extend[t1,t2]<>0) then continue;
            inc(ep);extend[t1,t2]:=k;
            e[ep,1]:=t1;e[ep,2]:=t2;
          end;
        if flag=0 then if achieve(k) then print;
        if k<en then t(k+1,i);

        for j:=ked+1 to ep do
          begin
            extend[e[j,1],e[j,2]]:=0;
          end;
        ep:=ked;
        source[a[k,1],a[k,2]]:=2;
      end;
  end;

begin
  assign(fin,'divide.dat');          读文件
  assign(fout,'divide.out');
  reset(fin);
  readln(fin,m);n:=0;
  fillchar(source,sizeof(source),0);
  fillchar(e,sizeof(e),0);
  for i:=1 to m do
    begin
      for j:=1 to m do
        begin
          read(fin,cc);
          if cc='*' then
            begin
              source[i,j]:=2;
              inc(n);
              if e[1,1]=0 then
              begin e[1,1]:=i;e[1,2]:=j; end;
            end;
        end;
      readln(fin);
    end;
  close(fin);
  n:=round(sqrt(n));                          初始化工作
  fillchar(extend,sizeof(extend),0);
```

```
  ep:=1;en:=n*n;
  extend[e[1,1],e[1,2]]:=1;
  t(1,0);                                          搜索
end.
```

第四节 信息编码 (Decode)

算法分析：

本题是一简单的编码题，关键是能准确地找出其编码方法。每一字符编码由键长及段内偏移组成，其在信息头的位置等于 2 的键长次幂加上段内偏移减去键长。

对每一字符，设其键长为 keylength，其编码（段内地址）为 si，则该字符的编译算法为：

字符位置（ea）:=2 的键长次幂（keylength）+编码（si）-键长（keylength）；

字符:=信息头的第 ea 个字符。

如在题例中，ord('T')=2+0-1=1,ord('M')=2*2+1-2=3,ord('I')=2*2*2+2-3=7,等等。

本题另一注意的地方是读入字符串的长度可能超过 256，因此应“分批”读入字符串，一边编码一边加长字符串。

程序分析：

```
program GDOI98_5_decode;

const
 p2:array[0..10] of integer=(1,2,4,8,16,32,64,128,256,512,1024);

var
 finame,foname:string;
 fi,fo:text;
 infohead:string;          {   信息头          }
 keylength,si:integer;     {   键长,段内地址       }
 st,st0,s:string;          {   读入的字符串    }
 t:integer;

{   追加字符串  }
procedure st_appended;
 begin
  while (length(st)+length(st0)<256) and (st0<>'') do
   begin
    st:=st+st0;
    readln(fi,st0);
   end;
 end;

{   把二进制数转为十进制数  }
```

```
function oct_for_bin(s:string):integer;
 var
  t:integer;
  i:integer;
 begin
  t:=0;
  for i:=length(s) downto 1 do
   if s[i]='1' then t:=t+p2[length(s)-i];
  oct_for_bin:=t;
 end;

{   一段的译码  }
procedure segment_decoded;
 begin
   s:=copy(st,1,keylength);
   si:=oct_for_bin(s);              {   计算编码（段内地址）     }
   st:=copy(st,keylength+1,length(st)-keylength);
   st_appended;
   while si<>p2[keylength]-1 do
    begin
     t:=p2[keylength]+si-keylength;{   计算本字符在信息头的位置     }
     write(fo,infohead[t]);
     s:=copy(st,1,keylength);
     si:=oct_for_bin(s);
     st:=copy(st,keylength+1,length(st)-keylength);
     st_appended;
    end;

   {   计算下一段的键长     }
   s:=copy(st,1,3);
   keylength:=oct_for_bin(s);
   st:=copy(st,4,length(st)-3);
   st_appended;
  end;

begin
 finame:='decode.dat';
 assign(fi,finame);
 foname:='decode.out';
 assign(fo,foname);

 reset(fi);
 rewrite(fo);
```

```
  readln(fi,infohead);
  readln(fi,st); readln(fi,st0);
  st_appended;
  s:=copy(st,1,3);
  keylength:=oct_for_bin(s);          {   计算键长    }
  st:=copy(st,4,length(st)-3);
  st_appended;
  while keylength<>0 do segment_decoded;

  close(fo);
  close(fi);
end.
```

第五节 海上交通控制（Lane）

算法分析：

本题中各个港口可以看作是图的顶点，港口间的航道可以看作顶点间的边，于是问题可以转化为求无向图的最短路径问题。

但是，题目中另外附加了一个条件，就是同一时间不能有多于一艘船占用同一航道。因此，求解本题不能使用Dijkstra和Floyd等求无向图的静态最短路径问题的算法，只能使用动态的方法——搜索。

搜索过程：

```
procedure try(f0:char; d,dt:integer; pass:t3);
{f0 出发港口，d 已经经过的港口数，dt 已经走了的距离，pass 经过的港口的集合}
 begin
  if dt>最短距离 then 回溯;
  if 已经到达目标港口 then
   begin
        将当前路径记录为最短路径;
    回溯;
      end;
  for 每个港口t do
   if (该港口t不在pass中) and (该港口t与f0有直接航道相连) then
        begin
          计算使用航道的时间;
          if 在该时间段中该航道没有被占用 then
        try(t, d+1, dt+航道长度, pass+[t]);
        end;
 end;
```

但是，由于数据量比较大，如果单纯使用回溯法，可能会超时。因此，必须对该算法进行优化。

一种优化方法：

1、用Floyd算法求出港口到港口的最短距离；

2、对每艘船先假设在该船之前出发且在该船出发时还没到达目的港口的船所占用的航道都封闭，用DIjkstra算法求出该船需要航行的最短距离mind(若不通则赋予无限大)，与第一步中求得的最短距离比较，若两者相等，则该船的最短航线上没有阻碍，直接输出该最短航线；
3、否则，以mind为上限进行搜索；
搜索过程修改如下：(下划线为添加部分)

```
procedure try(f0:char; d,dt:integer; pass:t3);
{f0 出发港口，d 已经经过的港口数，dt 已经走了的距离，pass 经过的港口的集合}
 begin
  if dt>最短距离 then 回溯;
  if 已经到达目标港口 then
   begin
        将当前路径记录为最短路径;
    回溯;
      end;
  for 每个港口t do
   if (该港口t不在pass中) and (该港口t与f0有直接航道相连)
      and (dt+航道长度+t与目的港口的最短距离<最短距离)  then
        begin
          计算使用航道的时间;
          if (在该时间段中该航道没有被占用) then
        try(t, d+1, dt+航道长度, pass+[t]);
        end;
 end;
```

另外本题目需要注意的还有精度问题，应该尽量采用精度高的计算方法，以减少误差，例如，计算到保个港口的时间时，应该将距离累加再除以速度来求出时间，和不应该直接将时间进行累加，否则会出现较大的累加误差。

程序分析：

```
program GDOI98_7_Lane;{海上交通控制}
const max=32767;
      ex=0.000001;
type t0=record
         f,t:char;
         ftime,ttime:real;
        end;
     t3=set of 'A'..'Z';
     t4=array[1..20,1..2] of real;
var leng,free,map:array['A'..'Z','A'..'Z'] of longint;
    {存储图的邻接矩阵}
    td,top,buttom,wm,i,j,k,l,n,m,ft,tt,t1,t2,speed,t5:longint;
    tt1,tt2,tt3,tt4,tt5:integer;
    ta1,ta2,ta3,ta4:real;
    a,b,f,t:char;
    name,names:string;
```

```
    boats:array[1..50,1..27] of t0;
    {存储每艘船的信息}
    f1,f2:text;
    dist:array[1..26] of longint;
    path:array[1..26] of string;
    pa,minp:string;
    mindt,mind:longint;
    s:t3;
    go:array['A'..'Z','A'..'Z'] of ^t4;
    gocount:array['A'..'Z','A'..'Z'] of integer;
    {存储港口间航道的占用时间}
function cango(f,t:char;t1,t2:real):boolean;
{判断在时间 t1 和 t2 之间 f 与 t 之间的航道是否能使用}
 var i:integer;
 begin
  cango:=false;
  for i:=1 to gocount[f,t] do
   begin
    if (t1+ex>go[f,t]^[i,1])and(t1+ex<go[f,t]^[i,2]) then exit;
    if (t2>go[f,t]^[i,1]+ex)and(t2+ex<go[f,t]^[i,2]) then exit;
    if (t1+ex<go[f,t]^[i,1])and(t2+ex>go[f,t]^[i,2]) then exit;
   end;
  cango:=true;
 end;
procedure try(f0:char;d,dt:integer;pass:t3);
{递归查找最短路径}
 var i:integer;
 begin
  if (f0=t) then
   begin
    if dt<=mindt then
     begin
      minp:=pa;
      mindt:=dt;
      mind:=d-1;
     end;
    exit;
   end;
  for i:=1 to n do
   if (not(names[i] in pass))and(map[f0,names[i]]<>0) then
    begin
     tt1:=ft div 100;tt2:=ft mod 100;
     ta4:=tt2+(dt+map[f0,names[i]])*60/speed;
     ta3:=tt2+dt*60/speed;
```

```
      ta1:=ta3-trunc(ta3/60)*60+(tt1+trunc(ta3/60))*100;
      ta2:=ta4-trunc(ta4/60)*60+(tt1+trunc(ta4/60))*100;
      if (longint(dt)+map[f0,names[i]]+leng[names[i],t]<=longint(mindt)) then
      {当前的路径长度+当前所处港口到目的港口的最短距离<=已知最短距离}
       if cango(f0,names[i],ta1,ta2) then
       {而且，在 ta1 到 ta2 之间，能从 f0 到第 i 个港口}
       begin
        pa[d]:=names[i];
        try(names[i],d+1,dt+map[f0,names[i]],pass+[names[i]]);
        {尝试到第 i 个港口}
       end;
     end;
 end;
procedure Floyd;
{用 Floyd 算法计算每两个港口之间的最短距离}
 begin
  for i:=1 to n do
   for j:=1 to n do
    begin
     leng[names[i],names[j]]:=map[names[i],names[j]];
     if leng[names[i],names[j]]=0 then leng[names[i],names[j]]:=max;
    end;
  for k:=1 to n do
   for j:=1 to n do
    for i:=1 to n do
     if leng[names[i],names[k]]+leng[names[k],names[j]]<leng[names[i],names[j]]
then

leng[names[i],names[j]]:=leng[names[i],names[k]]+leng[names[k],names[j]];
  for i:=1 to n do leng[names[i],names[i]]:=0;
 end;
procedure Dijkstra;
{用 Dijkstra 算法计算从某港口出发到其它港口的最短距离}
 begin
  for j:=1 to n do
   begin
    dist[j]:=free[f,names[j]];
    if (dist[j]<>0)and(dist[j]<max)
      then path[j]:=f
      else begin path[j]:='';dist[j]:=max; end;
   end;
  s:=[f];
  for l:=1 to n-1 do
   begin
```

```
    wm:=max;
    a:=f;
    for j:=1 to n do
     if not (names[j] in s)and (dist[j]<wm) then
      begin k:=j;wm:=dist[j];end;
    s:=s+[names[k]];
    for j:=1 to n do
     if                                not(names[j]                                in
s)and(free[names[k],names[j]]<>0)and(dist[k]+free[names[k],names[j]]<dist[j])
then
      begin
       dist[j]:=dist[k]+free[names[k],names[j]];
       path[j]:=path[k]+names[k];
      end;
   end;
  for j:=1 to n do
   path[j]:=path[j]+names[j];
 end;
procedure Block;
{将在本船出发之后才到达目的地的船占用的航道对本船封锁}
 begin
  for j:=1 to n do for k:=1 to n do
   begin
    free[names[j],names[k]]:=map[names[j],names[k]];
   end;
  for j:=1 to i-1 do
   begin
    k:=1;
    while boats[j,k].f<>' ' do
     begin
      if {(boats[j,k].ftime<=ft)and}(boats[j,k].ttime>ft)then
       begin
        free[boats[j,k].f,boats[j,k].t]:=max;
        free[boats[j,k].t,boats[j,k].f]:=max;
       end;
      inc(k);
     end;
   end;
 end;
begin
 for a:='A' to 'Z' do for b:='A' to 'Z' do
  begin new(go[a,b]);gocount[a,b]:=0;end;
 assign(f1,'lane.dat');
 reset(f1);
```

```
assign(f2,'lane.out');
rewrite(f2);
readln(f1,n);
readln(f1,names);
for i:=1 to n do
 begin
  for j:=1 to n do
   read(f1,map[names[i],names[j]]);
  readln(f1);
 end;
{输入邻接矩阵}
Floyd;
readln(f1,m);
for i:=1 to m do
 begin
  readln(f1,name);
  readln(f1,ft,speed);
  readln(f1,f,t);
  {输入各船提交的信息}
  if leng[f,t]=max then
  {起点和终点根本无路可通}
   begin
    writeln(f2,name);
    writeln(f2,ft,' -1');
    writeln(f2);
    writeln(name);
    writeln(ft,' -1');
    writeln;
   end
  else
   begin
    Block;
    Dijkstra;
    j:=1;
    if (t in s) then
      while t<>names[j] do inc(j)
    else
     begin
      j:=1;
      dist[j]:=max;
     end;
    writeln(f2,name);
    if (dist[j]=leng[f,t]) then
    {该船的理想航线上没有阻碍}
```

```
 begin
  k:=1;
  td:=0;
  for l:=2 to length(path[j]) do
   {记录该船占用各航道的时间}
   begin
    boats[i,k].f:=path[j,l-1];
    boats[i,k].t:=path[j,l];
    t1:=ft div 100;
    t2:=ft mod 100;
    ta1:=t2+td*60/speed;
    tt:=t1+trunc(ta1/60);
    ta2:=ta1-trunc(ta1/60)*60;
    boats[i,k].ftime:=tt*100+ta2;
    td:=td+map[boats[i,k].f,boats[i,k].t];
    ta1:=t2+td*60 / speed;
    tt:=t1+trunc(ta1/60);
    ta2:=ta1-trunc(ta1/60)*60;
    boats[i,k].ttime:=tt*100+ta2;
    inc(k);
   end;
  boats[i,k].f:=' ';
  writeln(f2,ft,' ',trunc(boats[i,k-1].ttime));
  writeln(f2,path[j]);{输出本船的航线}
 end
else
 begin
  top:=dist[j];
  mindt:=top;{最短路径的上限}
  buttom:=leng[f,t];
  pa:='                                    ';
  try(f,2,0,[f]);{递归查找最短路径}
  if mindt<>max then
   begin
    minp[1]:=f;
    td:=0;
    k:=1;
    t1:=ft div 100;t2:=ft mod 100;
    for l:=2 to mind do
    {记录该船占用各航道的时间}
     begin
      boats[i,k].f:=minp[l-1];
      boats[i,k].t:=minp[l];
      ta1:=t2+td*60/speed;
```

```
            tt:=t1+trunc(ta1/60);
            ta2:=ta1-trunc(ta1/60)*60;
            boats[i,k].ftime:=tt*100+ta2;
            td:=td+map[boats[i,k].f,boats[i,k].t];
            ta1:=t2+td*60 / speed;
            tt:=t1+trunc(ta1/60);
            ta2:=ta1-trunc(ta1/60)*60;
            boats[i,k].ttime:=tt*100+ta2;
            inc(k);
           end;
          boats[i,k].f:=' ';
          writeln(f2,ft,' ',trunc(boats[i,k-1].ttime));
          for j:=1 to mind do write(f2,minp[j]);
          writeln(f2);{输出本船的航线}
         end
        else
         begin
          k:=1;
          boats[i,1].f:=' ';
          writeln(f2,ft,' -1');
          writeln(f2);{输出无路可通}
         end;
       end;
      for j:=1 to k-1 do
      {在各航道上记录占用的时间}
       begin
        inc(gocount[boats[i,j].f,boats[i,j].t]);
        go[boats[i,j].f,boats[i,j].t]^[gocount[boats[i,j].f,boats[i,j].t],1]
            :=boats[i,j].ftime;
        go[boats[i,j].f,boats[i,j].t]^[gocount[boats[i,j].f,boats[i,j].t],2]
            :=boats[i,j].ttime;
        inc(gocount[boats[i,j].t,boats[i,j].f]);
        go[boats[i,j].t,boats[i,j].f]^[gocount[boats[i,j].t,boats[i,j].f],2]
            :=boats[i,j].ttime;
        go[boats[i,j].t,boats[i,j].f]^[gocount[boats[i,j].t,boats[i,j].f],1]
            :=boats[i,j].ftime;
       end;
     end;
   end;
 close(f2);close(f1);
end.
```

第六节 投递最佳路线(Best Deliver)

算法分析：

根据题目要求，每个司机的路线都要在考虑下一个司机前确定，所以应逐个求出司机的最佳运送路线。而对每一个司机，都用一般的搜索的话，则时间方面会大大超出限制，故如何优化搜索，是一个关键问题。

由题目所述，车站最多只有 26 个，而包裹最多时可有几百个，因此包裹多时每个站会出现多个待送包裹，则不用等待的机会也会增多。所以可在搜索前加上一个贪心搜索，即找完全不用等待的路线，作为一个较优解；然后再用一般的搜索。两个搜索过程的均应以包裹作为对象进行，可忽略站的干扰，省却许多麻烦。而此时的求有关时间的函数应为：

```
procedure gettime(p:char;  t,now:word;  var dt,wt:word);
var
  a,b:char;
  wtime:word;  {等待时间}
begin
  a:=p;  {当前站}
  b:=cg[t].b;  {包裹所在站，可以是当前站}
  if 包裹尚未出现 then wtime:=cg[t].t-(now+ga[a,b]) else wtime:=0;
  dt:=dt+cg[t].pt;
  wt:=now+ga[a,b]+wtime +cg[t].pt;  {工作时间包括空车行走时间、等待时间和运送时间}
end;
```

程序分析：

```
program GDKOI 98_3_ BestDeliver;
1.1  定义和说明；
begin  {主程序}
1.2  init;  {初始化}
  repeat
    found:=true;
1.3  qsearch;  {先用贪心法求一较优解}
    if found then
      begin
        if ldt=600 then found:=true  {达到最理想情况，不用搜索}
1.4       else found:=search; {一般搜索}
        if found then
          begin
1.5           printf(li);  {打印}
1.6           pass;  {过渡处理}
          end;
      end;
  until not found;  {直到所有 A 站包裹运完}
```

```
1.7  done;  {最后处理}
end.
```

求精 1.1－－定义和说明

```
type
  cgt=record  {包裹数据}
    id:word;  {包裹编号}
    b,e:char;  {起止站}
    pt:word;  {从起始站到终点站的时间，即 ga[b,e]}
    t:word;  {出现时间}
  end;
  at1=array[0..1000] of word;
  at2=array[1..1000] of boolean;
var
  fi,fo:text;
  fin,fon:string;
  m,dr:byte;        {          ,          }
  ltt,li,n:word;     {          ,          ，包裹数量}
  ldt,lwt:integer;  {当前最优解}
  found:boolean;  {搜索标志}
  lw:at1;  {运送包裹顺序表}
  ldd:at2;  {已运送包裹表}
  mt:array['A'..'Z'] of word;
     {从某一站出发，最快运完的一个包裹的所需时间。用于判断是否还可运包裹。}
  cg:array[1..1000] of cgt;  {包裹表}
  ga:array['A'..'Z','A'..'Z'] of word;  {站间邻接表}
```

求精 1.2－－init 的过程说明

```
procedure init;  {初始化}
var
  i,j,k,p:word;
  a,b,s1,s2,s3:char;
  t:string[4];
  err:integer;
begin
  clrscr;
  fillchar(ga,sizeof(ga),0);
  fillchar(ldd,sizeof(ldd),0);
  dr:=0;ltt:=0;
  write('Input file name:');readln(fin);
  write('Ouput file name:');readln(fon);
  assign(fi,fin);reset(fi);
  readln(fi,n);
```

```
  for i:=1 to n do
    begin
      readln(fi,j,s1,a,s2,b,s3,t);
      cg[i].id:=j;
      cg[i].b:=a;cg[i].e:=b;
      val(copy(t,1,2),j,err);
      val(copy(t,3,2),k,err);
      cg[i].t:=j*60+k;
    end;
  readln(fi,p);
  m:=p*(p-1) div 2;
  for i:=1 to m do
    begin
      readln(fi,a,s1,b,s2,t);
      val(copy(t,1,2),j,err);
      val(copy(t,3,2),k,err);
      ga[a,b]:=j*60+k;
      ga[b,a]:=j*60+k;
    end;
  close(fi);
  assign(fo,fon);rewrite(fo);
  writeln(fo,'                    ');  {先打一空行，最后再打上总司机数}
  for i:=1 to n do cg[i].pt:=ga[cg[i].b,cg[i].e];
  for a:='A' to 'Z' do
    begin
      mt[a]:=10000;
      for i:=1 to n do
        begin
          j:=ga[a,cg[i].b]+cg[i].pt;
          if j<mt[a] then mt[a]:=j;
        end;
    end;
end;
```

求精 1.3－－qsearch 的过程说明

```
procedure qsearch;  {用贪心法求运送表}
var
  cnt,i,j,tt,p,min:word;
  re:boolean;
  pl:array[0..1000] of char;  {司机所处的当前站}
  w,dt,wt:at1;  {搜索中使用的运送包裹顺序表、运送时间、当前时刻}
  dd:at2;  {搜索中使用的已运送包裹表}
begin
  p:=0;min:=1440;
```

```
for i:=1 to n do if (cg[i].b='A') and (not ldd[i]) and (cg[i].t<min) then
   begin p:=i;min:=cg[p].t end;
if p=0 then begin found:=false;exit end;   {找不到从 A 站发出的包裹}
inc(dr);
dd:=ldd;
tt:=ltt;
ldt:=0;lwt:=32767;
fillchar(w,sizeof(w),0);
fillchar(dt,sizeof(dt),0);
fillchar(wt,sizeof(wt),0);
fillchar(pl,sizeof(pl),'A');
cnt:=0;
w[1]:=p;i:=2;
wt[0]:=cg[p].t;   {wt[0]为开始工作时刻}
dt[1]:=cg[p].pt;
wt[1]:=wt[0]+dt[1];
pl[1]:=cg[p].e;
dd[p]:=true;
inc(tt);
repeat
   repeat
      inc(w[i]);re:=false;
      if w[i]>n then
         begin
            w[i]:=0;re:=true;
            dec(i);if i=1 then exit;
            dd[w[i]]:=false;
            dec(tt);
         end;
      if (not re) and dd[w[i]] then re:=true;
      if (not re) and (cg[w[i]].b<>pl[i-1]) then re:=true;   {不允许空车}
      if (not re) and (cg[w[i]].t>wt[i-1]) then re:=true;   {不允许等待}
   until not re;
   dt[i]:=dt[i-1];
1.3.1   gettime(pl[i-1],w[i],wt[i-1],dt[i],wt[i]);
   dd[w[i]]:=true;
   pl[i]:=cg[w[i]].e;
   inc(tt);
   if (wt[i]-wt[0]>600) or                    {搜索到工作时间大于 10 小时 或 }
      (600-(wt[i]-wt[0])<mt[pl[i]]) or   {不能再运送包裹 或 }
      (tt=n) then                          {全部包裹都已运完为止}
      begin
         if wt[i]-wt[0]>600 then j:=i-1 else j:=i;
         inc(cnt);
```

```
            if cnt=1000 then exit;
                {由于后面还有搜索，不用求最优情况。1000 次不能刷新即退出。}
            if (dt[j]>ldt) or ((dt[j]=ldt) and (wt[j]-wt[0]<lwt)) then
              begin  {记录当前最优解}
                li:=j;
                ldt:=dt[j];
                lwt:=wt[j]-wt[0];
                lw:=w;
                if ldt=600 then exit; {达到最理想情况则马上退出}
                cnt:=0;
              end;
            dd[w[i]]:=false;
            dec(tt);
          end
        else inc(i);
    until false;
end;
```

求精 1.4——search 的过程说明

```
function search:boolean;  {一般搜索}
var
  i,j,tt,p,min:word;
  re:boolean;
  pl:array[0..1000] of char;  {司机所处的当前站}
  w,dt,wt:at1;  {搜索中使用的运送包裹顺序表、运送时间、当前时刻}
  dd:at2;  {搜索中使用的已运送包裹表}
begin
  p:=0;min:=1440;
  for i:=1 to n do if (cg[i].b='A') and (not ldd[i]) and (cg[i].t<min) then
    begin p:=i;min:=cg[p].t end;
  if p=0 then begin search:=false;exit end;  {找不到从 A 站发出的包裹}
  dd:=ldd;
  tt:=ltt;
      {ldt 与 lwt 不初始化，用贪心法求得的解做初始解}
  fillchar(w,sizeof(w),0);
  fillchar(dt,sizeof(dt),0);
  fillchar(wt,sizeof(wt),0);
  fillchar(pl,sizeof(pl),'A');
  w[1]:=p;i:=2;
  wt[0]:=cg[p].t;  {wt[0]为开始工作时刻}
  dt[1]:=cg[p].pt;
  wt[1]:=wt[0]+dt[1];
  pl[1]:=cg[p].e;
  dd[p]:=true;
```

```
  inc(tt);
  repeat
    repeat
      inc(w[i]);re:=false;
      if w[i]>n then
        begin
          w[i]:=0;re:=true;
          dec(i);if i=1 then begin search:=true;exit end;   {搜索结束}
          dd[w[i]]:=false;
          dec(tt);
        end;
      if (not re) and dd[w[i]] then re:=true;
      if not re then
        begin
          dt[i]:=dt[i-1];
          gettime(pl[i-1],w[i],wt[i-1],dt[i],wt[i]);
          if (ldt<>0) and (wt[i]-wt[0]-dt[i]>600-ldt) then re:=true;
              {等待时间不能比当前最优解的未用时间(600-ldt)长}
        end;
    until not re;
    dd[w[i]]:=true;
    pl[i]:=cg[w[i]].e;
    inc(tt);
    if (wt[i]-wt[0]>600) or (600-(wt[i]-wt[0])<mt[pl[i]]) or (tt=n) then
      begin
        if wt[i]-wt[0]>600 then j:=i-1 else j:=i;
        if (dt[j]>ldt) or ((dt[j]=ldt) and (wt[j]-wt[0]<lwt)) then
          begin   {记录当前最优解}
            li:=j;
            ldt:=dt[j];
            lwt:=wt[j]-wt[0];
            lw:=w;
            if ldt=600 then begin search:=true;exit end;   {达到最理想情况则马上退出}
          end;
        dd[w[i]]:=false;
        dec(tt);
      end
    else inc(i);
  until false;
end;
```

求精 1.5－－printf 的过程说明

```
procedure printf(l:word);   {打印一个司机的运送表}
var
```

```
  i,tot:word;
  pl:char;
begin
  pl:='A';tot:=0;
  for i:=1 to l do
    begin
      if pl<>cg[lw[i]].b then inc(tot);
      inc(tot);
      pl:=cg[lw[i]].e;
    end;
  writeln(fo,tot);
  pl:='A';
  for i:=1 to l do
    begin
      if pl<>cg[lw[i]].b then writeln(fo,0,' ',pl,cg[lw[i]].b:2);
      writeln(fo,cg[lw[i]].id,' ',cg[lw[i]].b,cg[lw[i]].e:2);
      pl:=cg[lw[i]].e;
    end;
1.5.1  writeln(fo,m2hm(ldt));
  writeln(fo,m2hm(lwt));
end;
```

求精 1.6——pass 的过程说明

```
procedure pass;  {过渡处理}
var i:word;
begin
  for i:=1 to li do ldd[lw[i]]:=true;  {已运包裹登记}
  ltt:=0;
  for i:=1 to n do if ldd[i] then inc(ltt);  {求出已运包裹数}
end;
```

求精 1.7——done 的过程说明

```
procedure done;  {最后处理}
var
  ff:file of char;
  s:string;
  i:word;
begin
  writeln(fo,n-ltt);
  for i:=1 to n do
    if not ldd[i] then writeln(fo,cg[i].id,' ',cg[i].b);  {打印未运送包裹}
  close(fo);
  assign(ff,fon);reset(ff);  {把文本文件当字符型文件打开}
  str(dr,s);  {把司机数转为字符串}
```

```
  for i:=1 to length(s) do write(ff,s[i]);  {逐个字符写入文件}
  close(ff);
end;
```

求精 1.3.1－－gettime 的过程说明

```
procedure gettime(p:char; t,now:word; var dt,wt:word);  {求出运送时间和工作时间}
var
  a,b:char;
  wtime:word;
begin
  a:=p;
  b:=cg[t].b;
  if cg[t].t>now+ga[a,b] then wtime:=cg[t].t-(now+ga[a,b]) else wtime:=0;
  dt:=dt+cg[t].pt;
  wt:=now+ga[a,b]+wtime+cg[t].pt;
end;
```

求精 1.5.1－－m2hm 的函数说明

```
function m2hm(t:word):string;  {转换时间为字符串}
var s,r:string;
begin
  s:='';
  if t div 60<10 then s:=s+'0';
  str(t div 60,r);s:=s+r;
  if t mod 60<10 then s:=s+'0';
  str(t mod 60,r);s:=s+r;
  m2hm:=s;
end;
```

第七节 计算机网络连接(computer network)

算法分析：

如果把各台计算机看成一个点，计算机之间的连线看成边，连线的长度看成边的权，这样原来的计算机网络可以抽象成一个赋权图。那么原来求连线最短的问题可以转化为图论中一种特殊的最短路问题：求可以覆盖图中各点的简单路的最短路。

求这条最短路的基本思想是求出可以覆盖图中各点的所有简单路，再计算各条路的长度，取其长度最短的一条。

用递归过程 search 描述如下：

```
procedure search(m:byte);
  begin
    for i:=1 to n do if mark[i]=0 then
      begin
        取点 i 作为当前路径上的新结点：route[m]:=i;
```

```
        为点 i 作以经过的标记： mark[i]:=1;
        计算路径长度：distance:=distance+r[route[m-1],route[m]];
        if 当前路长度比以知最短路短（distance<min） then
        if 没有覆盖所有点 then search(m+1)
        else
          begin
            保存最短路：backup:=route;
            min:=distance;
          end;
        恢复路径长度：distance:=distance-r[route[m-1],route[m]];
        清标记：mark[i]:=0;
      end;
  end;
```

为了加快程序运行速度，减少搜索量，可以考虑加入一些剪枝条件。例如：

1. 防止方向性的重复。

例如，路径 A_1—A_2—…—A_N 与路径 A_N—A_{N-1}—…—A_1，这两条路径长度相同，只是方向不同，对本题而言是没有本质区别的。所以可以只取其中的一条路径，即可以规定起点 A_S 的下标小于终点 A_E 的下标（S<E）。

2. 在最短路中有这样一个特性：最短路中的任一段任是最短路；换一种说法，含有一段不是最短路的路径必不是最短路。

例如，若 A—B—C—D 的长度小于 A—C—B—D 的长度，那么，含有 A—C—B—D 的路一定不是最短路。如 E—A—C—B—D—F，因为存在路 E—A—B—C—D—F 比它短。

利用这个特性，我们可以得到相应的剪枝条件：含有非最短路的路可以剪去。

* 对于四个点的情况，可以开一个四维数组直接标记。如 A—B—C—D 是最短路，则[A，B，C，D]=1 否则为 0。

* 对于五个点的情况，如路 A—B—C—D—E，起点为 A 终点为 E，途中覆盖 B、C、D 的还有 A—B—D—C—E，A—C—B—D—E，A—C—D—B—E，A—D—B—C—E，A—D—C—B—E，若 A—B—C—D—E 并非是这条长度最小的，则剪枝。

* 对更多点的情况，考虑起来会更为复杂，这里只考虑这样的一种剪枝情况：若 A_1—A_2—…—A_{N-1}—A_N 和 A_1—A_{N-1}—…—A_2—A_N 前者的长度比后者长，那么剪去包含前面的路径的枝。

程序分析：

```
{$N+,R-,S-,V-}
program gdkoi98_4_ computer_network;
uses dos;
const max=20;
      negative_error=-(1e-6);positive_error=-negative_error;
      {negative_error:负误差；positive_error:正误差}
type tarray=array [1..max,1..max,1..max] of byte;
var n,i,j,k,l,more:integer;
    coordinate:array [1..max,1..2] of real;      {坐标}
```

```
    r:array [1..max,1..max] of real;              {任两点间的距离}
    flag:array [1..max] of ^tarray;               {标记数组，记录任四点之间的最短路情况}
    file_name:string;f:text;
    backup,route,mark:array [0..max] of byte;{route：路径}
                                                  {backup：保存的最短路}
                                                  {mark：以经过的点标记为 1，否则为 0}
    temp,min,distance:real;

procedure search(m:byte);
{搜索过程}
  var i,j,k:integer;
  begin
    if more=0 then exit;
    for i:=1 to n do if mark[i]=0 then
    {判断第 m 步到达点 i 是否合理}
      begin
        route[m]:=i;
        {判断四点的情况}
        if m>=3 then
          if flag[route[m-3]]^[route[m-2],route[m-1],route[m]]=0 then continue;
        {判断五点的情况}
        if m>=4 then
          begin
            temp:=r[route[m-4],route[m-3]]+r[route[m-3],route[m-2]]
            +r[route[m-2],route[m-1]]+r[route[m-1],route[m]];
            if temp>r[route[m-4],route[m-2]]+r[route[m-2],route[m-1]]
            +r[route[m-1],route[m-3]]+r[route[m-3],route[m]]+positive_error then continue;
            if temp>r[route[m-4],route[m-1]]+r[route[m-1],route[m-3]]
            +r[route[m-3],route[m-2]]+r[route[m-2],route[m]]+positive_error then continue;
            if temp>r[route[m-4],route[m-1]]+r[route[m-1],route[m-2]]
            +r[route[m-2],route[m-3]]+r[route[m-3],route[m]]+positive_error then continue;
          end;
        {判断多点的情况}
        if m>4 then
          begin
            for j:=0 to m-5 do
            begin
              if r[route[j],route[j+1]]+r[route[m-1],route[m]]>r[route[j],route[m-
                1]]+r[route[j+1],route[m]]+positive_error
              then continue;
            end;
          end;

        {下一层的搜索}
```

```
            mark[i]:=1;
            if i>route[0] then dec(more);
            distance:=distance+r[route[m-1],route[m]];{计算距离}
            if distance+positive_error<min then{判断是否比以知的最短路短}
            if m<n-1 then search(m+1)
            else
               begin
                  backup:=route;
                  min:=distance;
               end;
            distance:=distance-r[route[m-1],route[m]];
            if i>route[0] then inc(more);
            mark[i]:=0;
         end;
   end;

begin
   {读入各个计算机的坐标}
   write('Input File Name:');readln(file_name);
   assign(f,file_name);
   reset(f);
   readln(f,n);
   for i:=1 to n do
   readln(f,coordinate[i,1],coordinate[i,2]);
   close(f);
   write('Output File Name:');readln(file_name);
   assign(f,file_name);


   {计算任意两点间的距离}
   fillchar(r,sizeof(r),0);
   for i:=1 to n do for j:=i+1 to n do
      begin
         r[i,j]:=sqrt(sqr(coordinate[i,1]-coordinate[j,1])
            +sqr(coordinate[i,2]-coordinate[j,2]));
         r[j,i]:=r[i,j];
      end;

   {构造一个四维数组，记录覆盖四个点的最短路情况 }
   for i:=1 to n do
      begin
         new(flag[i]);
         fillchar(flag[i]^,sizeof(flag[i]^),0);
      end;
```

```
  for i:=1 to n do for j:=i+1 to n do
  for k:=1 to n do if (k<>i) and (k<>j) then
  for l:=1+k to n do if (l<>i) and (l<>j) then
    begin
      temp:=(r[i,k]+r[l,j])-(r[i,l]+r[k,j]);
      if temp>negative_error then flag[i]^[l,k,j]:=1;
      if temp<positive_error then flag[i]^[k,l,j]:=1;
    end;

  {搜索起点为 p[0]的最短路}
  min:=1e10;
  for i:=1 to n do
    begin
      route[0]:=i;
      fillchar(mark,sizeof(mark),0);
      mark[i]:=1;
      more:=n-i;
      distance:=0;
      search(1);
    end;

  {输出结果}
  rewrite(f);
  writeln(f,min:0:2);
  for i:=0 to n-1 do
    begin
      write(f,backup[i]);
      if i<n-1 then write(f,' ')
      else writeln(f);
    end;
  close(f);
end.
```

第八节 联系圈(Circle)

算法分析：

本题实际上是要求找出有向图中互相都能到达的结点的集合。一种标准的算法是利用深度优先搜索，其步骤如下：(1)在有向图 G 上，从某个顶点出发沿以该顶点为尾的弧进行深度优先搜索遍历，并按其所有邻接点的搜索都完成的顺序将顶点排列起来。(2)在有向图上，从最后完成搜索的顶点出发，沿着以该顶点为头的弧作逆向的深度优先搜索遍历，若此次遍历不能访问到有向图中所有顶点，则从余下的顶点中最后完成搜索的那个顶点出发，继续做逆向的深度优先搜索遍历，依次类推，直至有向图中所有顶点都被访问到为止。由此，

每一次作逆向深度优先遍历所访问到的顶点集便是有向图中的一个所求的顶点集合。这种方法复杂度为 O(n+e)(其中 n 为顶点数目，e 为边数)，但编程较复杂。

另一种方法是利用 Warshall 算法求出每两点间的连通性，然后把互相都能到达的结点找出。这种方法时间复杂度为 O(n^3)，但编程十分简单，因此对 n 比较小的情况十分适用。例程就是使用这种方法。

Warshall 算法基本结构：

建立顶点的邻接矩阵；
for i:=1 to n do
  for j:=1 to n do
    for k:=1 to n do
      if (j 能到达 i) and (i 能到达 k) then 标志 j 能到达 k；

最后，如果 A 能到达 B，而 B 也能到达 A，则 A 和 B 在同一联系圈中。

程序分析：

```
Program GDKOI  98_5_Circle;{联系圈}

type tlink=array[1..25,1..25] of boolean;
     tnames=array[1..25] of string;

var name:tnames;{人名表}
    link:tlink;{邻接矩阵}
    infile,outfile:text;
    nsum,n,m,i,j,k:integer;{nsum 为总人数}
    name1,name2:string;
    s:set of 1..25;
    inname,outname:string;
{-------------------------------------------------------}
procedure get2name(var name1,name2:string);
{从文件中读入一行中两个以空格分开的人名}
 var c:char;
 begin
  read(infile,c);
  while c=' ' do read(infile,c);
  name1:='';
  while c<>' ' do
   begin
    name1:=name1+c;
    read(infile,c);
   end;
 readln(infile,name2);
end;
{-------------------------------------------------------}
function find(s:string):integer;
{在人名表中找人名 s，并返回其序号，如果没有找到，就加入表中}
```

```
  var i:integer;
  begin
    for i:=1 to nsum do
      if name[i]=s then
        begin
          find:=i;exit;
        end;
    inc(nsum);
    name[nsum]:=s;
    find:=nsum;
  end;
{-------------------------------------------------------------}
begin
  assign(infile,inname);
  reset(infile);
  assign(outfile,outname);
  rewrite(outfile);
  readln(infile,n,m);
  for i:=1 to n do
    for j:=1 to n do
      link[i,j]:=false;
  {将邻接矩阵清空}
  nsum:=0;
  for i:=1 to m do
    begin
      get2name(name1,name2);
      {从文件中读入一次联系}
      j:=find(name1);
      k:=find(name2);
      {将两个人名加入人名表中，并换为表中的序号}
      link[j,k]:=true;
      {表示第 j 个人曾打电话给第 k 个人}
    end;
  for i:=1 to n do
    for j:=1 to n do
      for k:=1 to n do
        if (i<>k)and(j<>i)and(j<>k)and(link[j,i])and(link[i,k]) then
          link[j,k]:=true;
  {Warshall 算法的核心部分，计算两顶点间是否有通路}
  s:=[1..n];
  for i:=1 to n do
    if i in s then
      begin
        write(outfile,name[i]);
```

```
      s:=s-[i];
      for j:=i+1 to n do
       if (j in s)and link[i,j] and link[j,i] then
         begin
          s:=s-[j];write(outfile,' ',name[j]);
         end;
      writeln(outfile);
      {将同一个联系圈上的人名输出}
    end;
  writeln(outfile);
  close(infile);
  close(outfile);
end.
```

第九节 球钟(Ball Clock)

算法分析：

本题是一个典型的模拟题，只需按题目的描述对球钟进行操作即可得到正确的答案。需要注意的是对球钟这种数据结构的描述方式，可以看到该球钟是由空闲球轨道，分钟指示器，5 分钟指示器和小时指示器 4 个部分组成。其中空闲球轨道是一个队列，其它三个指示器分别是三个栈。它们均可由数组来描述。其中需要注意的是，队列的先进先出操作如果用移动数组元素的方法实现的话，效率会比较低，可能会出现超时的情况，因此应该使用循环队列的方法把数组的头尾连接起来以减少数据的移动，或者使用链表来描述。另外，还须要注意的是当小时指示器满的时候，前十一个球先逆向返回空闲球队列，第十二个球再返回空闲球队列。

其核心部分是对半天的模拟：

```
  for HourTrackCount:=1 to 12 do
   begin
    for MiniteTrackCount:=1 to 12 do
     begin
      从空闲球轨道取出 5 个球；
      将前 4 个球逆向送到空闲球轨道；
      最晚来的球送往分钟指示器；
     end;
    {5 分钟指示器满}
    归还 5 分钟指示器中的球，最晚来的球送往小时指示器；
   end;
  {小时指示器满}
  归还小时指示器中的球，最晚来的球送回空闲球轨道；
  半天数加一；
  判是否回到原状态，若是，则结束；
```

程序分析：

```
{$A+,B-,D+,E-,F-,G+,I-,L+,N+,O-,P-,Q-,R-,S-,T-,V+,X+}
{$M 16384,0,655360}

Program GDKOI 98_6_Ball Clock;{球钟}

type tclock=array[1..200] of integer;
var CurrentFreeBalls:tclock;
      {空闲球的轨道，是一个循环队列}
      first,last,i,j,k:integer;
       {first 是循环队列的队头指针，last 是循环队列的队尾指针}
      trackofminite,trackofhour:array[1..12] of integer;
      TotalBalls,MiniteTrackCount,HourTrackCount:integer;
      {总球数，  5 分钟指示器球数，小时指示器球数}
      t:longint;{总半天数}
      tmp:array[1..10] of integer;
{-------------------------------------------------------}
function DelQueue:integer;{出队}
 begin
   DelQueue:=CurrentFreeBalls[first];
   first:=first mod 200+1;
 end;
{-------------------------------------------------------}
procedure AddQueue(d:integer);{入队}
 begin
   last:=last mod 200+1;
   CurrentFreeBalls[last]:=d;
 end;
{-------------------------------------------------------}
function match:boolean;
 {判断队列中的球是否按原顺序排列}
 var i,l:integer;
 begin
   match:=false;
   if last<first
     then l:=last+200
     else l:=last;
   for i:=first to l do
     if CurrentFreeBalls[i mod 200]<>i-first+1
        then exit;
   match:=true;
 end;
{-------------------------------------------------------}
```

```
function calc:boolean;
{对 5 分钟的一次模拟}
 var i,j:integer;
 begin
  for HourTrackCount:=1 to 12 do
   begin
    for MiniteTrackCount:=1 to 12 do
     begin
      for i:=1 to 5 do
       tmp[6-i]:=DelQueue;
      for i:=2 to 5 do
       AddQueue(tmp[i]);{将分钟指示器中的球逆向送到空闲球轨道}
      trackofminite[MiniteTrackCount]:=tmp[1];
     end;
    {5 分钟指示器满}
    for i:=1 to 11 do
     AddQueue(trackofminite[12-i]);
    {MiniteTrackCount:=0;
    {归还 5 分钟指示器中的球}
    trackofhour[HourTrackCount]:=tmp[1];
   end;
  {小时指示器满}
  for i:=1 to 11 do
   AddQueue(trackofhour[12-i]);
  AddQueue(tmp[1]);
  {归还小时指示器中的球}
  t:=t+1;
  if match {判是否回到原状态，若是返回 false}
     then calc:=false
     else calc:=true;
 end;
{------------------------------------------------------------}
begin
 readln(TotalBalls);
 if not(TotalBalls in [27..127])
  then writeln('Input data ',TotalBalls,' out of range.')
  else
   begin
    for i:=1 to TotalBalls do
     CurrentFreeBalls[i]:=i;
    first:=1;
    last:=TotalBalls;
    HourTrackCount:=0;
    MiniteTrackCount:=0;
```

```
    {设置初始状态}
    t:=0;
    while calc do;
    writeln(TotalBalls,' balls cycle after ',t div 2,' days.');
  end;
end.
```

第十节 建筑物(Buildings)

算法分析：

此题的思路其实不复杂，只要把所有可行路径求出，再用标号法求最短路径即可。所谓可行路径，即任意两已知顶点（含起止点）连线中，不与已知矩形的边相交的线段；再加上原来矩形的不相交边，就是全部的可行路径。所以最关键的问题就是如何准确地判断两线段是否相交。

判断一线段是否与其它线段相交的算法如下：

1.对该线段与其余全部线段进行相交判断；

对每两条线段，设线段的顶点为(x1,y1), (x2,y2)，求出两线段的点斜式方程中的系数，分别为 k1,b1,k2,b2。

对斜率 k1 和 k2 的值分为四种情况：

(1)k1 和 k2 都存在，但 k1 不等于 k2；

(2)k1 存在而 k2 不存在；

(3)k2 存在而 k1 不存在；

对以上三种情况：求出线段所在直线的交点，判断是否在两线段上；

(4)k1=k2 或 k1 和 k2 都不存在，此时两线段重合和平行，不相交。

```
function cross(p1n,p2n,p3n,p4n:byte):boolean;
var
  k1,k2,b1,b2,u,v:real;
  p1,p2,p3,p4:point;  {p1,p2 为一线段的起止点，p3,p4 为另一线段的起止点}
begin
  if p1n<p2n
    then begin k1:=kb[p1n,p2n];b1:=kb[p2n,p1n] end
    else begin k1:=kb[p2n,p1n];b1:=kb[p1n,p2n] end;
  if p3n<p4n
    then begin k2:=kb[p3n,p4n];b2:=kb[p4n,p3n] end
    else begin k2:=kb[p4n,p3n];b2:=kb[p3n,p4n] end;
  p1:=pp[p1n];p2:=pp[p2n];p3:=pp[p3n];p4:=pp[p4n];
  if  k1 不等于 k2  then
    if  k1 存在  then
      begin
```

```
        if  k2 存在 then
          begin  {k1 与 k2 均存在的情况}
            u:=(b1-b2)/(k2-k1);v:=k1*u+b1;  {交点坐标}
            cross:=  u 在区间(p1.x,p2.x) 且 u 在区间(p3.x,p4.x);
          end
        else
          begin  {k1 存在而 k2 不存在的情况，此时 b2 为线段 2 的横坐标}
            if  b2 在区间(p1.x,p2.x)  then
              begin
                u:=b2;v:=k1*u+b1;  {交点坐标}
                cross:= v 在区间(p3.y,p4.y);
              end
            else cross:=false;
          end
      end
    else
      begin
        if  k2 存在 then
          begin  {k1 不存在而 k2 存在的情况，此时 b1 为线段 1 的横坐标}
            if  b1 在区间(p3.x,p4.x)  then
              begin
                u:=b1;v:=k2*u+b2;  {交点坐标}
                cross:=v 在区间(p1.y,p2.y);
              end
            else cross:=false;
          end
        else cross:=false;  {k1 与 k2 均不存在则一定不相交}
      end
  else cross:=false;  {重合或平行都不算相交}
end;
```

2.因以上相交判断时不包括线段的顶点，故对某些情况如：矩形内的对角线，则不能判出。所以再对该线段与全部矩形进行相交判断；

对矩形的两对平行线求 k 与 b，判断线段的中点是否都夹于两对平行线中。

```
function inbox(p:point; q:byte):boolean;
var
  i:byte;
  k,b:array[1..4] of real;  {矩形的四条边的直线参数}
  inb:boolean;
begin
  for i:=1 to 4 do  {求四条边的直线参数}
    if box[q,i-1]<box[q,i]then
      begin
```

```
            k[i]:=kb[box[q,i-1],box[q,i]];
            b[i]:=kb[box[q,i],box[q,i-1]];
          end
        else
          begin
            k[i]:=kb[box[q,i],box[q,i-1]];
            b[i]:=kb[box[q,i-1],box[q,i]];
          end;
    if   k[1]存在
      then inb:= p.y-p.x*k[1]  在区间  (b[1],b[3])
              {p.y-p.x*k[1]表示平行于第 1,3 边，且过 p 点的直线的截距}
      else inb:= p.x  在区间  (b[1],b[3]);
    if not inb then begin inbox:=false;exit end;
    if   k[2]存在
      then inb:= p.y-p.x*k[2]  在区间  (b[2],b[4])
              {p.y-p.x*k[2]表示平行于第 2,4 边，且过 p 点的直线的截距}
      else inb:=p.x  在区间  (b[2],b[4]);
    inbox:=inb;
end;
```

经过以上两步的判断，就可准确地去除不可行路径。

程序分析：

```
program GDKOI 98_8_ Buildings;
{$M 65000,0,655360}
1.1  定义和说明；
begin   {主程序}
1.2    Init;   {初始化}
1.3    GetCross;   {求出已知矩形的所有相交线段}
1.4    GetAllPath;   {求出所有可行路径}
1.5    ShortestPath;   {求出最短路径}
        writeln('route distance:',dis[2]:2:2);
end.
```

```
求精 1.1－－定义和说明
const
   eps=1e-8;   {实数误差}
   inf=1e36;   {无限大}
   meps=1e28;   {相对于无限大 inf 的实数误差}
type
   point=record x,y:real; cr:byte end;
     {x,y 为坐标，cr 为以该点为起点的矩形边是否相交}
   gat=array[1..82,1..82] of real;
var
```

```
  n,vn:byte;  {矩形数，顶点总数}
  bx,by,ex,ey:real;  {起止点坐标}
  box:array[1..20,0..4] of byte;
    {矩形数据，第二分量的相邻两点表示一边，即 box[i,j]和 box[i,j+1]为表示一边}
  ga:^gat;  {所有顶点的可行路径的邻接表}
  kb:gat;
    {所有线段的斜率(K)和截距(B)，
     kb[i,j]=      i<j 时     表示点 i 和点 j 连线的斜率(K)
                   i>j 时     表示点 i 和点 j 连线的截距(B)
    }
  pp:array[1..82] of point;  {顶点数据}
  dis:array[1..82] of real;  {从起点出发到各点的最短路径长度}

求精 1.2－－Init 的过程说明
procedure Init;
var
  fin:string;
  fi:text;
  x,y:array[1..4] of real;
  i,j,k,t,o1,o2:byte;
  max:real;
begin
  fillchar(box,sizeof(box),0);
  new(ga);
  clrscr;
  write('Input file name:');readln(fin);
  assign(fi,fin);reset(fi);
  readln(fi,n);
  readln(fi,bx,by,ex,ey);
  vn:=2;
  pp[1].x:=bx;pp[1].y:=by;  {起点为 1 号点}
  pp[2].x:=ex;pp[2].y:=ey;  {终点为 2 号点}
  for i:=1 to n do
    begin
      readln(fi,x[1],y[1],x[2],y[2],x[3],y[3]);
      max:=0;
      for j:=1 to 2 do for k:=j+1 to 3 do
        if dist(x[j],y[j],x[k],y[k])>max then
          begin  {选出已知三点中距离最大的两点，其连线即为一对角线}
1.2.1       max:=dist(x[j],y[j],x[k],y[k]);
            t:=6-j-k;
            o1:=j;o2:=k;
          end;
      x[4]:=x[o1]+(x[o2]-x[t]);y[4]:=y[o1]+(y[o2]-y[t]);  {求第四点坐标}
```

```
        pp[vn+1].x:=x[4]; pp[vn+1].y:=y[4];
        pp[vn+2].x:=x[o1];pp[vn+2].y:=y[o1];
        pp[vn+3].x:=x[t]; pp[vn+3].y:=y[t];
        pp[vn+4].x:=x[o2];pp[vn+4].y:=y[o2];
        for j:=0 to 3 do box[i,j]:=vn+j+1;
        box[i,4]:=box[i,0];
        inc(vn,4);
      end;
  close(fi);
1.2.2  for i:=1 to vn-1 do for j:=i+1 to vn do getkb(i,j,kb[i,j],kb[j,i]);
    {一开始先计算全部线段的方程}
end;
```

求精 1.3－－GetCross 的过程说明

```
procedure GetCross;
var
  cr,i,j,e,f:byte;
  cen:point;
begin
  for e:=1 to n-1 do for f:=e+1 to n do
    begin
      for i:=0 to 3 do for j:=0 to 3 do
1.3.1   if cross(box[e,i],box[e,i+1],box[f,j],box[f,j+1]) then
          begin  {线段相交则令域 cr=1}
            pp[box[e,i]].cr:=1;pp[box[f,j]].cr:=1
          end;
      for i:=0 to 3 do
        begin
          cen.x:=(pp[box[e,i]].x+pp[box[e,i+1]].x)/2;
          cen.y:=(pp[box[e,i]].y+pp[box[e,i+1]].y)/2;
1.3.2      if inbox(cen,f) then
              pp[box[e,i]].cr:=1;  {若线段的中点在矩形内，也算相交}
        end;
      for i:=0 to 3 do
        begin
          cen.x:=(pp[box[f,i]].x+pp[box[f,i+1]].x)/2;
          cen.y:=(pp[box[f,i]].y+pp[box[f,i+1]].y)/2;
          if inbox(cen,e) then pp[box[f,i]].cr:=1;
        end;
    end;
end;
```

求精 1.4－－GetAllPath 的过程说明

```
procedure GetAllPath;
var
```

```
  cen:point;
  vn1,vn2,ln,ln1,i,j,k,r,tt:word;
  l:array[1..7000,0..2] of byte;  {可行路径表，第二分量的 0 表示是否相交线段，1 和 2 表示顶点号，即在顶点表 pp 中的编号}
  be:array[0..82] of word;  {每个矩形的边在顶点表 pp 中的起止编号}
  cro:boolean;
begin
  ln:=0;  {ln 为可行路径总数}
  tt:=0;be[0]:=2;
  vn2:=2;vn1:=2;
  for i:=1 to n do  {将所有矩形的边加到可行路径表中}
    begin
      inc(tt);
      vn1:=vn2;
      for j:=0 to 3 do
        begin
          inc(vn2);inc(ln);
          l[ln,1]:=vn2;
          if j=3 then l[ln,2]:=vn1+1 else l[ln,2]:=vn2+1;
          l[ln,0]:=pp[box[i,j]].cr;  {相交标志}
        end;
      be[tt]:=vn2;
    end;
  ln1:=ln;  {ln1 为矩形边的总数}
  for i:=1 to tt do for j:=be[i-1]+1 to be[i] do
    for k:=1 to vn do  {对任意两点所成线段进行相交判断}
      if not (k in [be[i-1]+1..be[i]]) then
        begin
          inc(ln);l[ln,1]:=j;l[ln,2]:=k;l[ln,0]:=0;
          cro:=false;
          for r:=1 to ln1 do
            if cross(l[ln,1],l[ln,2],l[r,1],l[r,2]) then
              begin cro:=true;r:=ln1 end;
          if not cro then
            begin
              cen.x:=(pp[j].x+pp[k].x)/2;
              cen.y:=(pp[j].y+pp[k].y)/2;
              for r:=1 to tt do if inbox(cen,r) then
                begin cro:=true;r:=tt end;
            end;
          if cro then dec(ln);
        end;
  inc(ln);l[ln,1]:=1;l[ln,2]:=2;l[ln,0]:=0;  {对起止点直接连线进行判断}
  cro:=false;
```

```
  for r:=1 to ln1 do
    if cross(l[ln,1],l[ln,2],l[r,1],l[r,2]) then
      begin cro:=true;r:=ln1 end;
  if not cro then
    begin
      cen.x:=(pp[1].x+pp[2].x)/2;
      cen.y:=(pp[1].y+pp[2].y)/2;
      for r:=1 to tt do if inbox(cen,r) then
        begin cro:=true;r:=tt end;
    end;
  if cro then dec(ln);
  for i:=1 to 82 do for j:=1 to 82 do
    if i=j then ga^[i,j]:=0 else ga^[i,j]:=1e36;
  for i:=1 to ln do if l[i,0]=0 then   {求出可行路径的长度}
    begin
      ga^[l[i,1],l[i,2]]:=dist(pp[l[i,1]].x,pp[l[i,1]].y,pp[l[i,2]].x,pp[l[i,2]].y);
      ga^[l[i,2],l[i,1]]:=ga^[l[i,1],l[i,2]];
    end;
end;
```

求精 1.5——ShortestPath 的过程说明

```
procedure ShortestPath;
var
  s:set of 1..82;
  i,j,k:byte;
  min:real;
begin
  for i:=1 to vn do dis[i]:=ga^[1,i];
  s:=[1];
  for k:=1 to vn-1 do
    begin
      min:=1e35;j:=1;
      for i:=1 to vn do
        if not (i in s) and (dis[i]<min) then
          begin j:=i;min:=dis[i] end;
      s:=s+[j];
      for i:=1 to vn do
        if not (i in s) and (dis[j]+ga^[j,i]<dis[i]) then
          dis[i]:=dis[j]+ga^[j,i];
    end;
end;
```

求精 1.2.1——dist 的函数说明

```
function dist(x1,y1,x2,y2:real):real;  {求两点间距离}
```

```
begin
  dist:=sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
end;
```

求精 1.2.2——getkb 的过程说明

```
procedure getkb(p1n,p2n:byte; var k,b:real);
  {求过 p1,p2 的直线方程 y=k*x+b 中的 k,b}
var p1,p2:point;
begin
  p1:=pp[p1n];p2:=pp[p2n];
  if abs(p1.x-p2.x)>eps then
    begin
      k:=(p1.y-p2.y)/(p1.x-p2.x);
      b:=p1.y-k*p1.x;
    end
  else
    begin
      k:=inf;b:=p1.x;  {当 k 不存在时，b 为直线的 x 坐标}
    end;
end;
```

求精 1.3.1——cross 的函数说明

```
function cross(p1n,p2n,p3n,p4n:byte):boolean;
  {判断 p1,p2 构成的线段是否与 p3,p4 构成的线段相交（不含重合）}
var
  k1,k2,b1,b2,u,v:real;
  p1,p2,p3,p4:point;
begin
  if p1n<p2n
    then begin k1:=kb[p1n,p2n];b1:=kb[p2n,p1n] end
    else begin k1:=kb[p2n,p1n];b1:=kb[p1n,p2n] end;
  if p3n<p4n
    then begin k2:=kb[p3n,p4n];b2:=kb[p4n,p3n] end
    else begin k2:=kb[p4n,p3n];b2:=kb[p3n,p4n] end;
  p1:=pp[p1n];p2:=pp[p2n];p3:=pp[p3n];p4:=pp[p4n];
  if abs(k1-k2)>eps then
    if abs(k1-inf)>meps then
      begin
        if abs(k2-inf)>meps then
          begin  {k1 与 k2 均存在的情况}
            u:=(b1-b2)/(k2-k1);v:=k1*u+b1;
1.3.1.1       cross:=include(u,p1.x,p2.x) and include(u,p3.x,p4.x);
          end
        else
```

```
            begin   {k1 存在而 k2 不存在的情况}
              if include(b2,p1.x,p2.x) then
                begin
                  u:=b2;v:=k1*u+b1;
                  cross:=include(v,p3.y,p4.y);
                end
              else cross:=false;
            end
        end
      else
        begin
          if abs(k2-inf)>meps then
            begin   {k1 不存在而 k2 存在的情况}
              if include(b1,p3.x,p4.x) then
                begin
                  u:=b1;v:=k2*u+b2;
                  cross:=include(v,p1.y,p2.y);
                end
              else cross:=false;
            end
          else cross:=false;   {k1 与 k2 均不存在则一定不相交}
        end
    else cross:=false;   {重合或平行都不算相交}
end;
```

求精 1.3.2－－inbox 的函数说明

```
function inbox(p:point; q:byte):boolean;
  {判断点 p 是否在矩形 q 内，方法为判断点 p 是否夹于矩形的两对平行线中}
var
  i:byte;
  k,b:array[1..4] of real;
  inb:boolean;
begin
  for i:=1 to 4 do
    if box[q,i-1]<box[q,i]then
      begin
        k[i]:=kb[box[q,i-1],box[q,i]];
        b[i]:=kb[box[q,i],box[q,i-1]];
      end
    else
      begin
        k[i]:=kb[box[q,i],box[q,i-1]];
        b[i]:=kb[box[q,i-1],box[q,i]];
      end;
```

```
  if abs(k[1]-inf)>meps
    then inb:=include(p.y-p.x*k[1],b[1],b[3])
    else inb:=include(p.x,b[1],b[3]);
  if not inb then begin inbox:=false;exit end;
  if abs(k[2]-inf)>meps
    then inb:=include(p.y-p.x*k[2],b[2],b[4])
    else inb:=include(p.x,b[2],b[4]);
  inbox:=inb;
end;
```

求精 1.3.1.1－－include 的函数说明

```
function include(x,a,b:real):boolean;   {判断 x 是否在开区间(a,b)中}
begin
  if a<b
    then include:=(x>a+eps) and (x<b-eps)
    else include:=(x>b+eps) and (x<a-eps)
end;
```

附录： 1997-2000 年（第 22~25 届）ACM 国际大学生程序设计竞赛（ACM/ICPC）亚洲区预赛成绩

ACM 国际大学生程序设计竞赛亚洲预赛上海赛区 1997 年竞赛成绩
(1997.11.7)

名次	参赛队名	解题数	得分
1	上海交通大学二队	7	813
2	清华大学	5	412
3	上海交通大学一队	5	510
4	上海大学一队	5	718
5	复旦大学	5	863
6	中山大学	5	1049

ACM 国际大学生程序设计竞赛亚洲预赛上海赛区 1998 年竞赛成绩
(1998.11.28)

名次	参赛队名	解题数	得分
1	韩国高等理工学院	6	1026
2	清华大学	5	511
3	中山大学	5	516
4	上海青少年科技指导站	5	889
5	上海交通大学二队	4	432
6	上海大学一队	3	485

ACM 国际大学生程序设计竞赛亚洲预赛上海赛区 1999 年竞赛成绩
(1999.11.28)

名次	参赛队名	解题数	得分
1	清华大学一队	5	537
2	清华大学二队	5	684
2	上海交通大学二队	4	554
3	中山大学一队	4	871
4	中山大学二队	3	451
4	南京大学	3	643
5	新加坡国立大学	3	701
6	上海交通大学一队	2	83
6	复旦大学	2	238

ACM ICPC Asia Regional Taipei Site Final Standing
(1999.12.13 台北)

Rank	Team Name	School Name	Problems	Solved Penalty
1	ZhongShan University Team	Zhongshan University	5	710
2	Ants	National Tsing-Hua University	4	547
3	Apple-Car Movie	National Taiwan University	4	661
4	Zero	National Kaohsiung Institute of Technology	4	707
5	Vman	National Taiwan University	4	744
5	TREND Setter	Nanjing University	4	744
6	The Seven Wonders	National Taiwan University	3	378
6	ATM Team	National Taiwan University	3	459
6	Guardian	National Chiao Tung University	3	465
6	Dove	National Tsing-Hua University	3	850

ACM 国际大学生程序设计竞赛亚洲预赛上海赛区 2000 年竞赛成绩
（2000.10.22）

名次	参赛队名	解题数

1	清华大学 Violet 队	7
2	上海交通大学二队	7
3	中山大学一队	5
4	北京大学一队	5
4	上海交通大学二队	5
4	清华大学 Hurricane 队	5
4	中山大学二队	5
5	复旦大学 ChosenI 队	5
5	中山大学三队	-
6	北京航空航天大学 BUAA I	-
6	南京大学 Settler 队	-
6	合肥工业大学队	-
6	东华大学 A 队	-
6	华南理工大学 Tiger 队	-
6	同济大学队	-

The 2000 ACM Asia Programming Contest, Hong Kong
(2000.10.28 香港)

Rank	Team	Citation
1	Tsinghua University=>Tsinghua Hurricane	First Place
2	ZhongShan University=>Zhongshan University Team A	Second Place
3	The University of Hong Kong=>HKU TICK	Third Place
4	Fundan University=>Fundan Chosen-I	Fourth Place
5	Shanghai JiaoTong University=>SJTU	Fifth Place
6	National Taiwan University=>Taiwan University	Sixth Place

ACM/ICPC Asian Regional Contest, Tsukuba Final Standings

(2000.11.13 日本筑波)

Rank	Team	#Solved	Penalty
1	Yonsei University=>Yonsei Eagle	5	799
2	Zhongshan University=>Zhongshan University Team	4	464
3	National University of Singapore=>NUS Team A	4	816
4	Kyoto University=>NewWillowField	4	863
5	University of Tokyo=>edajima_heihachi	3	309
6	The University of Tokyo=>unti-tle	3	390
6	Kyoto university=>BO_TSU Black	3	512
6	Tokyo Institute of Technology=>fishdock	2	136