

信息学奥林匹克竞赛辅导(Pascal 版)

目录

| | |
|------------------------------------|------------|
| 青少年信息学奥林匹克竞赛情况简介 | 2 |
| 第一部分 计算机基础知识 | 3 |
| 第一节 计算机的基本常识 | 3 |
| 第二节 操作系统 | 7 |
| 第三节 计算机网络常识 | 9 |
| 第四节 计算机信息安全基础知识 | 11 |
| 第二部分 Pascal 语言概述与预备知识 | 12 |
| 第一章 开始编写 Pascal 语言程序 | 13 |
| 第二章 Pascal 语言基础知识 | 14 |
| 第三章 顺序结构程序设计 | 18 |
| 第四章 选择结构程序设计 | 21 |
| 第五章 循环结构程序设计 | 23 |
| 第六章 数组与字符串 | 25 |
| 第七章 函数和过程 | 28 |
| 第八章 子界与枚举类型 | 32 |
| 第九章 集合类型 | 33 |
| 第十章 记录与文件类型 | 34 |
| 第十一章 指针类型 | 37 |
| 第十二章 程序调试 | 40 |
| 第三部分 常用算法与策略 | 44 |
| 第一章 算法 | 44 |
| 第二章 递归 | 46 |
| 第三章 回溯 | 48 |
| 第四章 排序 | 53 |
| 第五章 查找(检索) | 63 |
| 第六章 穷举(枚举)策略 | 67 |
| 第七章 贪心策略 | 71 |
| 第八章 分治策略 | 77 |
| 第四部分 数据结构 | 81 |
| 第一章 什么是数据结构 | 81 |
| 第二章 线性表 | 81 |
| 第三章 栈 | 88 |
| 第四章 队列 | 93 |
| 第五章 树和二叉树 | 97 |
| 第六章 图 | 104 |
| 第五部分 动态规划 | 115 |
| 第一章 什么叫动态规划 | 115 |
| 第二章 用动态规划法解题 | 116 |
| 第三章 典型例题与习题 | 120 |
| 第四章 动态规划的递归函数法 | 125 |
| 第五章 动态规划分类 | 126 |
| 练习: Barn Repair | 132 |
| 第六部分 数学知识及相关算法 | 134 |
| 第一章 有关数论的算法 | 134 |
| 第二章 高精度计算 | 139 |

| | |
|--------------------------|------------|
| 第三章 排列与组合..... | 149 |
| 第四章 计算几何..... | 153 |
| 第五章 其它数学知识及算法..... | 157 |
| 第七部分 图论算法..... | 159 |
| 第一章 最小生成树..... | 159 |
| 第二章 最短路径..... | 161 |
| 第三章 拓扑排序（AOV 网）..... | 167 |
| 第四章 关键路径（AOE 网）..... | 169 |
| 第五章 网络流..... | 172 |
| 第六章 图匹配..... | 179 |
| 第八部分 搜索算法与优化..... | 187 |
| 第一章 广度优先双向搜索..... | 187 |
| 第二章 分支定界法..... | 191 |
| 第三章 A*算法..... | 196 |

青少年信息学奥林匹克竞赛情况简介

信息学奥林匹克竞赛是一项旨在推动计算机普及的学科竞赛活动，重在培养学生能力，使得有潜质有才华的学生在竞赛活动中锻炼和发展。近年来，信息学竞赛活动组织逐步趋于规范和完善，基本上形成了“地级市——省（直辖市）——全国——国际”四级相互接轨的竞赛网络。现把有关赛事情况简介如下：

全国青少年信息学（计算机）奥林匹克分区联赛：

在举办 1995 年 NOI 活动之前，为了扩大普及的面，并考虑到多数省、直辖市、自治区已经开展了多年省级竞赛，举办了首届全国青少年信息学（计算机）奥林匹克分区联赛。考虑到不同年级学生的知识层次，也为了鼓励更多的学生积极参与，竞赛设提高组、普及组，并分初、复赛进行，这样可以形成一个梯队，确保每年的竞赛活动有比较广泛扎实的基础。

从 1995 年起，至 2001 年共举办了七届全国青少年信息学奥林匹克分区联赛，每年举办一次，有选手个人奖项（省、国家级）、选手等级证书、优秀参赛学校奖项。

广东省青少年信息学（计算机）奥林匹克决赛（简称 GDOI）：

省级信息学奥赛是一个水平较高的、有较大影响力的学科竞赛。由各市组织代表队参赛，参赛名额实行动态分配制度，每年举办一次。从 1984 年起广东省奥林匹克竞赛活动得到了蓬勃发展。奖项有个人一、二、三等奖，女选手第一、二、三名，奖励学校团体总分 1-8 名、市团体总分 1-8 名。

全国青少年信息学（计算机）奥林匹克竞赛（简称 NOI）：

由中国计算机学会主办的、并与国际信息学奥林匹克接轨的一项全国性青少年学科竞赛活动。1984 年举办首届全国计算机竞赛。由各省市区组织参赛，每年举办一次。奖项有个人一、二、三等奖，女选手第一、二、三名，各省队团体总分名次排队。

国际青少年信息学（计算机）奥林匹克竞赛（简称 IOI）：

每年举办一次，由各参赛国家组队参赛。

全国青少年信息学（计算机）奥林匹克分区联赛竞赛大纲

一、初赛内容与要求：（#表示普及组不涉及，以下同）

| | | | |
|-----|--------------------------|-----------|------------|
| 计 基 | * 诞生与发展 | *特点 | *在现代社会中的应用 |
| 算 本 | * 计算机系统的基本组成 | | |
| 机 常 | * 计算机的工作原理# | | *计算机中的数的表示 |
| 的 识 | * 计算机信息安全基础知识 | | *计算机网络 |
| 计 基 | * MS DOS 与 Windows 的使用基础 | | |
| 算 本 | * 常用输入/输出设备的种类、功能、使用 | | |
| 机 操 | * 汉字输入/输出方法 | | |
| 的 作 | * 常用计算机屏显信息 | | |
| 程 | 程序的表示 | * 自然语言的描述 | |

| | | |
|---------------------------------|---------|---|
| 序 设 计 基 本 知 识 | | * PASCAL 或 BASIC 语言 |
| | 数据结构的类型 | * 简单数据的类型 * 构造类型：数组、字符串 * 了解基本数据结构（线性表、队列与栈） |
| | 程序设计 | * 结构化程序的基本概念 * 阅读理解程序的基本能力 * 具有完成下列过程的能力： 现实世界（指知识范畴的问题） —>信息世界（表达解法） —>计算机世界（将解法用计算机能实现的数据结构和算法描述出来） |
| | 基本算法处理 | * 简单搜索 * 字符串处理 * 排序 * 查找 * 统计 * 分类 * 合并 * 简单的回溯算法 * 简单的递归算法 |

二、复赛内容与要求：

在初赛的内容上增加以下内容(2002 年修改稿)：

| | | |
|------------------|---|---------------------|
| 计算机 软 件 | *操作系统的使用知识 *编程语言的使用 | |
| 数据 结 构 | *结构类型中的记录类型 *文件（提高组必须会使用文本文件输入） *树 | *指针类型 *链表 *图# |
| 程序 设 计 | *程序设计能力 *设计测试数据的能力 *运行时间和占用空间的估算能力# | |
| 算 法 处 理 | *排列组合的应用 *进一步加深回溯算法、递归算法 *分治法 *搜索算法：宽度、深度优先算法 *表达式处理：计算、展开、化简等# *动态规划# | |

三、初赛试题类型：注：试题语言两者选一

（程序设计语言：基本 BASIC 或 TURBO PASCAL）

*判断 *填空 *完善程序 *读程序写运行结果 *问答

四、推荐读物：

*分区联赛辅导丛书 *学生计算机世界报及少年电世界杂志

第一部分 计算机基础知识

第一节 计算机的基本常识

- 1.1 计算机的产生和发展
- 1.2 计算机的系统及工作原理
- 1.3 计算机中有关数、编码的基本常识

1.4 原码、反码与补码

1.5 逻辑运算

1.1 计算机的产生与发展

计算机的产生是 20 世纪最重要的科学技术大事件之一。世界上的第一台计算机（ENIAC）于 1946 年诞生在美国宾夕法尼亚大学，到目前为止，计算机的发展大致经历了四代：

① 第一代电子管计算机，始于 1946 年，结构上以 CPU 为中心，使用计算机语言，速度慢，存储量小，主要用于数值计算；

② 第二代晶体管计算机，始于 1958 年，结构上以存储器为中心，使用高级语言，应用范围扩大到数据处理和工业控制；

③ 第三代中小规模集成电路计算机，始于 1964 年，结构上仍以存储器为中心，增加了多种外部设备，软件得到了一定的发展，文字图象处理功能加强；

④ 第四代大规模和超大规模集成电路计算机，始于 1971 年，应用更广泛，很多核心部件可集成在一个或多个芯片上，从而出现了微型计算机。

我国从 1956 年开始电子计算机的科研和教学工作，1983 年研制成功 1 亿/秒运算速度的“银河”巨型计算机，1992 年 11 月研制成功 10 亿/秒运算速度的“银河 II”巨型计算机，1997 年研制了每秒 130 亿运算速度的“银河 III”巨型计算机。

目前计算机的发展向微型化和巨型化、多媒体化和网络化方向发展。计算机的通信产业已经成为新型的高科技产业。计算机网络的出现，改变了人们的工作方式、学习方式、思维方式和生活方式。

1.2 计算机系统及工作原理

1. 计算机的系统组成

计算机系统由软件和硬件两部分组成。硬件即构成计算机的电子元器件；软件即程序和有关文档资料。

(1) 计算机的主要硬件

输入设备：键盘、鼠标、扫描仪等。

输出设备：显示器、打印机、绘图仪等。

中央处理器（CPU）：包括控制器和运算器运算器，可以进行算术运算和逻辑运算；控制器是计算机的指挥系统，它的操作过程是取指令——分析指令——执行指令。

存储器：具有记忆功能的物理器件，用于存储信息。存储器分为内存和外存

①内存是半导体存储器(主存)：

它分为只读存储器(ROM)和随机存储器(RAM)和高速缓冲存储器(Cache)；

ROM:只能读，不能用普通方法写入，通常由厂家生产时写入，写入后数据不容易丢失，也可以用特殊方法（如紫外线擦除(EEPROM)或电擦除(EEPROM)存储器)；

RAM:可读可写，断电后内容全部丢失；

Cache:因为 CPU 读写 RAM 的时间需要等待，为了减少等待时间，在 RAM 和 CPU 间需要设置高速缓存 Cache, 断电后其内容丢失。

②外存：磁性存储器——软盘和硬盘；光电存储器——光盘，它们可以作为永久存器；

③存储器的两个重要技术指标：存取速度和存储容量。内存的存取速度最快(与 CPU 速度相匹配)，软盘存取速度最慢。存储容量是指存储的信息量，它用字节(Byte)作为基本单位，

1 字节用 8 位二进制数表示，1KB=1024B，1MB=1024KB，1GB=1024MB

(2) 计算机的软件

计算机的软件主要分为系统软件和应用软件两类：

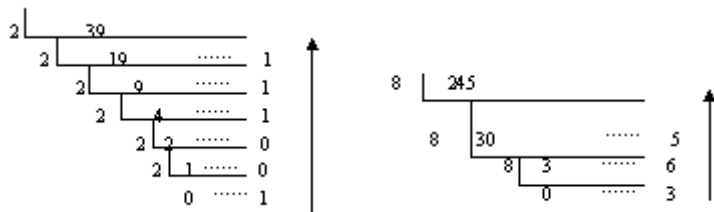
①系统软件：为了使用和管理计算机的软件，主要有操作系统软件如，WINDOWS 95 / 98 / 2000 / NT4. 0、DOS 6. 0、UNIX 等；WINDOWS 95 / 98 / 2000 / NT4. 0 是多任务可视化图形 界面，而 DOS 是字符命令形式的单任务的操作系统。

②应用软件：为了某个应用目的而编写的软件，主要有辅助教学软件(CAI)、辅助设计软件(CAD)、文 字处理软件、工具软件以及其他的应用软件。

2. 计算机的工作原理

到目前为止，电子计算机的工作原理均采用冯·若依曼的存储程序方式，即把程序存储在计算机内，由计算机自动存取指令（计算机可执行的命令=操作码+操作数）并执行它。工作原理图如下：

| 进制 | 基数 | 特点 |
|------|-------------------------|-------|
| 二进制 | 0,1 | 逢二进一 |
| 八进制 | 0,1,2,3,4,5,6,7 | 逢八进一 |
| 十六进制 | 0,1,2,...,9,A,B,C,D,E,F | 逢十六进一 |



$$(7AF)_{16} = 7 \times 16^2 + 10 \times 16^1 + 15 \times 16^0 = 1867$$

4. 定点数与浮点数

定点数是指数据中的小数点位置固定不变。由于它受到字长范围的限制，所能表示的数的范围有限，计算结果容易溢出。浮点数的形式可写成： $N = M \times 2^E$ （其中 M 代表尾数，E 代表阶码）其形式如下：

| | |
|----|-----------|
| 阶码 | 尾数（包括符号位） |
|----|-----------|

5. ASCII 编码

由于计算机是电器设备，计算机内部用二进制数，这样对于从外部输入给计算机的所有信息必须用二进制数表示，并且对于各种命令、字符等都需要转换二进制数，这样就牵涉到信息符号转换成二进制数所采用的编码的问题，国际上统一用美国标准信息编码（ASCII）它可用 7 位二进制数表示，存储时用一个字节，它的最高位为 0。因此基本的 ASCII 字符集有 128 个如：

0-9: 48-57: 00110000-...

A-Z: 65-90 : 01000001-...

a-z: 97-122: 01100000-...

6. 汉字编码与汉字输入法

（1）机内码

ASCII 码不能表示汉字，因此要有汉字信息交换码，我国国家标准是 gb2312，它也被称作国际码。它由两个字节组成，两个字节的最高位都为 1。gb2312 共收纳 6763 个汉字，其中，一级汉字（常用字）3755 个按汉字拼音字母顺序排列，二级汉字 3008 个按部首笔画次序排列。

（2）汉字输入码（外码）

目前，汉字输入法主要有键盘输入、文字识别和语音识别。键盘输入法是当前汉字输入的主要方法。它大体可以分为：

流水码：如区位码、电报码、通信密码，优点重码律少，缺点难于记忆；

音码：以汉语拼音为基准输入汉字，优点是容易掌握，但重码律高；

形码：根据汉字的字型进行编码，优点重码少，但不容易掌握；

音形码：将音码和形码结合起来，能减少重码律同时提高汉字输入速度。

（3）汉字字模

供计算机输出汉字（显示和打印）用的二进制信息叫汉字字形信息也称字模。通用汉字字模点阵规格有 16*16，24*24，32*32，48*48，64*64，每个点在存储器中用一个二进制位(bit)存储，如一个 16*16 点阵汉字需要 32 个字节的存储空间。

1.4 原码、反码与补码

在计算机中，数据是以补码的形式存储的：

在 n 位的机器数中，最高位为符号位，该位为零表示为正，为 1 表示为负；

其余 n-1 位为数值位，各位的值可为 0 或 1。

当真值为正时：原码、反码、补码数值位完全相同；

当真值为负时：

原码的数值位保持原样，

反码的数值位是原码数值位的各位取反，

补码则是反码的最低位加一。

注意符号位不变。

如：若机器数是 16 位：

十进制数 17 的原码、反码与补码均为：0000000000010001

十进制数-17 的原码、反码与补码分别为：1000000000010001、1111111111101110、1111111111101111

1.5 逻辑运算

1. 逻辑运算

逻辑与：同真则真

逻辑或：有真就真

逻辑非：你真我假

逻辑异或：不同则真

2. 按位运算

按位与 \cap ：同 1 则 1 如 $10010101 \cap 10110111 = 10010101$

按位或 \cup ：有 1 则 1 如 $10010101 \cup 10110111 = 10110111$

3. 逻辑化简

化简定律:

- (1) 交换律: $A + B = B + A$, $A \cdot B = B \cdot A$
- (2) 结合律: $(A + B) + C = A + (B + C)$, $(A \cdot B) \cdot C = A \cdot (B \cdot C)$
- (3) 幂等律: $A \cdot A = A$, $A + A = A$
- (4) 吸收律: $A \cdot (A + B) = A$, $A + (A \cdot B) = A$
- (5) 分配律: $A \cdot (B + C) = A \cdot B + A \cdot C$, $A + (B \cdot C) = (A + B) \cdot (A + C)$
- (6) 互补律: $A + \bar{A} = 1$, $A \cdot \bar{A} = 0$
- (7) 非深入: $A + B = A \cdot B$, $A \cdot B = A + B$
- (8) 0-1 律: $A + 0 = A$, $A + 1 = 1$, $A \cdot 1 = A$, $A \cdot 0 = 0$

例:化简函数 $Q = AD + \bar{A}D + AB + ACEF$ 。这个函数有 5 个自变量, 化简过程如下:

$$\begin{aligned} Q &= AD + \bar{A}D + AB + ACEF \\ &= A + AB + ACEF \\ &= A + ACEF \\ &= A \end{aligned}$$

练习:求证: $(A+B)(A+C)=AB+AC$

第二节 操作系统

- 2.1 DOS 的组成
- 2.2 DOS 文件和目录
- 2.3 DOS 命令
- 2.4 Windows 简介

2.1 DOS(Disk Operating System)的组成

MS—DOS 采用模块结构, 它由五部分组成: ROM 中的 BIOS 模块、IO. SYS 模块、MSDOS. SYS 模块、COMMAND. COM 模块和引导程序。

- (1)BIOS 模块: 在 PC 机主板上有一个 ROM 芯片, 该芯片中存有系统自测试程序, CMOS 设置程序和基本输入输出程序 (BIOS)。BIOS 是一组程序和参表, 其中程序部份是可以通过中断方式调用的一组驱动程序, 参数给出外设的地址和参数。BIOS 是计算机硬件和操作系统之间的接口, 通过它操作系统管理计算机硬件资源。
- (2)IO. SYS 模块: IO. SYS 是 MS—DOS 和 ROMBIOS 之间的接口程序。它和 ROM、BIOS 一起完成系统设备的管理。
- (3)MSDOS. SYS 模块: MSDOS. SYS 用于实现文件管理, 包括文件管理、目录管理、内存管理等功能。它以功能调用的形式实现用户和 MS—DOS 之间的程序级接口。
- (4)COMMAND. COM 模块: COMMAND. COM 的主要功能是负责接收、识别、解释和执行用户从键盘输入的 MS—DOS 命令。
- (5)引导程序: 引导程序又叫“引导记录”, 其作用是检查当前盘上是否有两个系统文件, 若有系统文件则把 DOS 系统从磁盘装入内存。

一张系统盘上应该包含有: 引导记录、IO. SYS、MSDOS. SYS 和 COMMAND. COM 等模块。

2.2 DOS 的文件和目录

- 1) 文件概念: 文件是指记录在存储介质 (如磁盘、光盘) 上的一组相关信息的集合。
- 2) 文件标识: 驱动器号+路径+文件名 (1 到 8 各字符)+扩展名 (1 到 3 个字符代表文件的类型)
- 3) 通配符: *代表从该位置起的一个或多个合法字符; ?代表所在位置的任一个合法字符。
- 4) 树形目录: DOS 采用树形目录结构。由一个根目录和若干层子目录组成。这种目录结构一是能够解决文件重名问题, 即不同的目录可以包含相同的文件名或目录名; 二是能够解决文件多而根目录容量有限带来的问题。在查找某个子目录下的一个文件时, 要使用目录路径。指定路径有两种方法: 绝对路径和相对路径。绝对路径是从根目录开始到文件所在目录的路径。例如要查找 UCDDOS 子目录下的二级子目录 DATA 下的 README. TXT 文件, 绝对路径为: \UCDDOS\DATA。路径中第一个“\”符号代表根目录。相对路径是从当前目录开始到文件所在目录的路径。当前目录指在不特意指定路径情况下 DOS 命令所处理的目录。例如系统提示符为: “C: \UCDDOS\DATA>”, 则 DATA 是当前目录。

2.3 DOS 命令

1. 内部命令

1) 内部命令: 当启动 DOS 系统时, 计算机引导程序将系统以及常用的命令处理模块驻留在计算机的内存中, 我们称之为内部命令。

2) 常用的内部命令:

(1) 目录命令:

DIR(显示文件目录)

MD、CD、RD(子目录的建立、进入、删除命令)

(2) 文件操作命令:

COPY(复制命令)、DEL(删除命令)、REN(更改文件名)

TYPE(显示文本文件内容)

(3) 其他内部命令

DATA、TIME、VER、CLS 等

3. 外部命令

1) 外部命令: 存储在外存储器上的 DOS 可执行的文件, 这些文件程序所占的存储容量比较大, 当用户使用外部命令时, 计算机从外存调入内存, 当执行完外部命令, 就自动从内存中退出。

2) 常用的外部命令

(1) 磁盘格式化命令: `FORMAT 盘符 [/S] I/V`

其作用, 能够清除原盘中所有信息, 并将磁盘规范成计算机所能接受的格式, 以便有效存储信息。

(2) 软盘复制命令: `DISKCOPY [盘符 1:] [盘符 2:]`

其作用, 能够进行软盘之间的全盘复制(以磁道方式), 不仅可以复制系统文件而且可以复制隐含文件。

2.4 Windows 简介

Windows 是一个多任务图形用户界面, 该环境可以在基于 MS-DOS 的计算机上运行, 在多任务图形用户环境下, Windows 提供了一个基于下拉菜单、屏幕窗口和鼠标的界面, 在该环境下运行的应用程序必须进行专门的设计才能发挥这些特征的优点。

2. Windows 的特点

Windows 能够充分发挥计算机的作用, 其图形接口能够组织用户程序和文件、同时运行几个用户程序、在文档之间移动和复制信息、在平台上进行应用程序的切换等。为了提高效率, Windows 还提供了一些辅助程序, 如字处理器、画笔及其他标准应用程序等。

Windows 具有以下主要特点。

(1) 图形化的用户界面

Windows 提供了一种不同于 DOS 系统下命令行的工作方式, 它通过对窗口、图标、选单、对话框、命令按钮、滚动框等图形符号与画面的操作来实现对计算机的各种操作。

(2) 标准化的操作界面

在 Windows 中, 所有的操作都是通过窗口中的图形界面进行的。

(3) 多任务机制和执行性能

在 Windows 中, 平稳的多任务机制可以同时运行多道程序以及执行多项任务, 各程序与各任务之间不仅转换容易, 而且还可以方便地交换数据。

(4) 充分利用内存

Windows 利用虚拟内存技术, 允许应用程序超过 640 阳常规内存的运行空间, 从而最大限度地利用了计算机系统的所有内存资源, 从而使内存较小的微机也能运行大型的应用程序。

(5) 强大的联网功能

在 Windows 中, 可以简单直观地实现网络的安装、配置、浏览, 从而可以更加方便地实现网络管理和资源共享。

(6) 丰富的多媒体功能

Windows 提供大量辅助程序, 用以实现文字、图形、图像、声音、视频等多媒体功能, 同时还支持其他厂商基于 Windows 标准开发的各相应软件。

(7) TrueType 技术

TrueType(真实字体)属于内建式比例字体, 可以任意平滑放大与缩小。这种字体能使屏幕上显示的效果与实际打印机输出的信息完全一致, 这就是所谓的“所见即所得”。

[例 4] 在 Windows 95 中, “任务栏”的作用是_____。

A) 显示系统的所有功能 B) 只显示当前活动窗口名

C) 只显示正在后台工作的窗口名 D) 实现窗口之间的切换

解答: 在任务栏中, 显示了所有打开的程序的图标。

本题正确答案为 D。

第三节 计算机网络常识

3.1 网络基础知识

3.2 Internet 简介

3.1 网络基础知识

1. 网络的概念

计算机网络是将地理位置不同的计算机,用通信链路连接起来,共同遵守一定的协议,以实现计算机软硬件资源共享为目标的通信系统。

2. 网络的组成

计算机网络由网络硬件和网络软件组成。

网络软件包括网络操作系统、通信软件、通信协议（计算机之间实现数据通信共同遵守的相关规定）。

网络硬件包括网络的拓扑结构、网络服务器、网络工作站、传输介质和设备。

3. 网络的分类

（1）按通信距离分：

局域网（LAN）：局限于某个范围(10 公里左右)的网络连接情（校园网）。

广域网（WAN）：跨地区的局域网，Internet 是覆盖全球的广域网。

（2）按网络的使用目的分：

共享资源网：使用者可分享网络的各种资源（如 Internet）。

数据处理网：用于数据处理（企业经营管理用的网络）。

数据传输网：用于数据的收集、交换和传输（情报检索网络）。

（3）按网络的拓扑结构分：

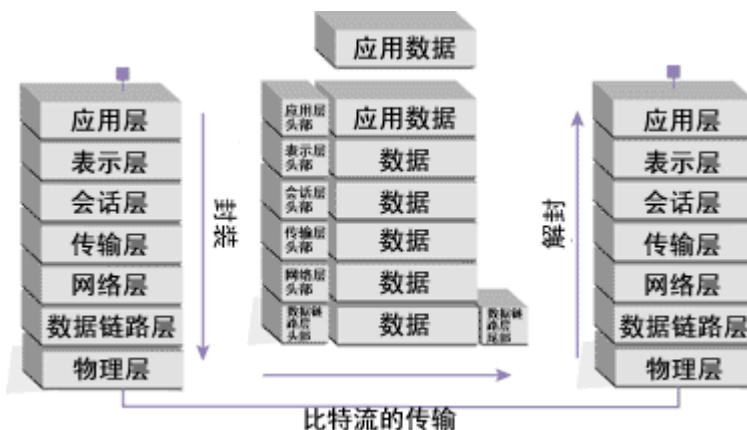
星形网：以一台计算机为中心，以放射状连接若干台计算机。

环形网：传输线路构成一个封闭的环，入网的计算机连到这个环形线路上。

总线网：用一条通信线路作主干，入网的计算机通过相应接口连到线路上。

4. 开放系统互联 模型（OSI 模型）

OSI 模型分 7 层：



各层功能如下：

1. 物理层

物理层与移动二进制数和维护物理连接有关。

2. 数据链路层

数据链路层通过帧在一个给定的物理链路传输分组（报文），保持帧的有序以及发现检测到的各种错误，包括传输错误，但是数据链路层只了解在链路另一端的对等实体。数据链路层的地址是为了将网络中一点的数据帧送到另一点。

3. 网络层

网络层知道每个数据链路的对等进程，并负责在链路间移动分组，把它送到目的地。网络层地址是为了把单一分组从网络的一端送到目的地。

4. 传输层

传输层注意的是整个网络，该层是第一个端到端层。其对等实体位于分组的最终目的地。传输层依靠网络层经过中间节点移动分组。传输层地址是为了把网络一端进程的完整信息送到最终目的地的对等进程。

5-7. 会话层、表示层和应用层提供了如下功能：

处理计算机间数据表示的差别。

确保数据在网络传输中不被窃取和泄露，并且确保网络不允许未经授权就访问数据。

最高效地使用网络资源通过应用程序及活动同步来管理对话和活动。

在网络节点间共享数据。

3.2 Internet 简介

Internet 英文直译为“互联网”，中文名为“因特网”。是世界上众多计算机网络的集合起源于 20 世纪 80 年代。

1. Internet 的 IP 地址、IP 地址类型和主机域名

(1) 在 Internet 网上采用统一的网络协议 TCP/IP, 与 Internet 相连的计算机必须具有唯一的主机地址, 称 IP 地址。IP 地址采用分段地址方式, 使用数字表示; 如: 207.46.130.14, 其中由三个点隔开的四个数是十进制, 其大小是 0-255, 每个数对应一个 8 位二进制数, 所以 IP 地址用 32 位二进制位存放站 4 个字节。

(2) IP 地址类型: 最初设计互联网络时, 为了便于寻址以及层次化构造网络, 每个 IP 地址包括两个标识码 (ID), 即网络 ID 和主机 ID。同一个物理网络上的所有主机都使用同一个网络 ID, 网络上的一个主机 (包括网络上工作站, 服务器和路由器等) 有一个主机 ID 与其对应。IP 地址根据网络 ID 的不同分为 5 种类型, A 类地址、B 类地址、C 类地址、D 类地址和 E 类地址。

A 类 IP 地址

一个 A 类 IP 地址由 1 字节的网络地址和 3 字节主机地址组成, 网络地址的最高位必须是“0”, 地址范围从 1.0.0.0 到 126.0.0.0。可用的 A 类网络有 126 个, 每个网络能容纳 1 亿多个主机。

B 类 IP 地址

一个 B 类 IP 地址由 2 个字节的网络地址和 2 字节的主机地址组成, 网络地址的最高位必须是“10”, 地址范围从 128.0.0.0 到 191.255.255.255。可用的 B 类网络有 16382 个, 每个网络能容纳 6 万多个主机。

C 类 IP 地址

一个 C 类 IP 地址由 3 字节的网络地址和 1 字节的主机地址组成, 网络地址的最高位必须是“110”。范围从 192.0.0.0 到 223.255.255.255。C 类网络可达 209 万余个, 每个网络能容纳 254 个主机。

D 类地址用于多点广播 (Multicast)。

D 类 IP 地址第一个字节以“1110”开始, 它是一个专门保留的地址。它并不指向特定的网络, 目前这一类地址被用在多点广播 (Multicast) 中。多点广播地址用来一次寻址一组计算机, 它标识共享同一协议的一组计算机。

E 类 IP 地址

以“11110”开始, 为将来使用保留。

全零 (“0.0.0.0”) 地址对应于当前主机。全“1”的 IP 地址 (“255.255.255.255”) 是当前子网的广播地址。

在 IP 地址 3 种主要类型里, 各保留了 3 个区域作为私有地址, 其地址范围如下:

A 类地址: 10.0.0.0~10.255.255.255

B 类地址: 172.16.0.0~172.31.255.255

C 类地址: 192.168.0.0~192.168.255.255

(3) 为了使用方便, 在访问 Internet 上的主机时, 通常使用主机域名而不是 IP 地址, 但主机域名和 IP 地址一一对应, 它由圆点分隔的一序列单词组成如 “Public.bta.net.cn”。IP 地址如同电脑的身份证号码, 而域名相当电脑的姓名。

2. Internet 的功能

(1) 信息浏览 (WWW)

WWW (World Wide Web), 中文名为“万维网”, 是基于超文本的、方便用户信息浏览和信息搜索的信息系统。用户在浏览器中输入网址即可得到需要的信息。人们常用的浏览器有网景公司的 Netscape 浏览器和 Microsoft 公司的 Internet Explorer 浏览器。网址的输入是使用协议提供的服务+服务器地址 (IP 地址或主机域名) 如 http://198.105.232.1; ftp://zsyz.com

(2) 文件传输 (FTP)

FTP (File Transfer Protocol) 是 Internet 的一种标准协议, 这一协议使用户能在联网的计算机之间传送文件如上载 (UPLOAD 把本地计算机上地文件复制到远程计算机上) 和下载 (DOWNLOAD 把远程计算机上的文件复制到本地计算机上)。

(3) 传送电子邮件 (E-mail): 电子邮件地址=用户名+@+主机域名; 如: zhangming@yahoo.com

(4) 电子公告牌 (BBS)

(5) 远程登录 (telnet)

(6) 电子商务等

3. TCP/IP 参考模型

TCP/IP 协议的开发研制人员将 Internet 分为五个层次，以便于理解，它也称为互联网分层模型或互联网分层参考模型，如下表：

应用层（第五层）

传输层（第四层）

互联网层（第三层）

网络接口层（第二层）

物理层（第一层）

各层简要说明如下：

物理层：对应于网络的基本硬件，这也是 Internet 物理构成，即我们可以看得见的硬件设备，如 PC 机、互连网服务器、网络设备等，必须对这些硬件设备的电气特性作一个规范，使这些设备都能够互相连接并兼容使用。

网络接口层：它定义了将数据组成正确帧的规程和在网络中传输帧的规程，帧是指一串数据，它是数据在网络中传输的单位。

互联网层：本层定义了互联网中传输的“信息包”格式，以及从一个用户通过一个或多个路由器到最终目标的“信息包”转发机制。

传输层：为两个用户进程之间建立、管理和拆除可靠而又有效的端到端连接。

应用层：它定义了应用程序使用互联网的规程。

第四节 计算机信息安全基础知识

4.1 计算机网络安全

4.2 计算机病毒

4.1 计算机的网络安全

1、不同环境和应用中的网络安全

运行系统安全，即保证信息处理和传输系统的安全。它侧重于保证系统正常运行，避免因系统的崩溃和损坏而对系统存贮、处理和传输的信息造成破坏和损失，避免因电磁泄漏，产生信息泄露，干扰他人，受他人干扰。

网络上系统信息的安全。包括用户口令鉴别，用户存取权限控制，数据存取权限、方式控制，安全审计，安全问题跟踪，计算机病毒防治，数据加密。

网络上信息传播安全，即信息传播后果的安全。包括信息过滤等。它侧重于防止和控制非法、有害的信息进行传播后的后果。避免公用网络上大量自由传输的信息失控。

网络上信息内容的安全。它侧重于保护信息的保密性、真实性和完整性。避免攻击者利用系统的安全漏洞进行窃听、冒充、诈骗等有害于合法用户的行为。本质上是保护用户的利益和隐私。

网络安全的特征

2、网络安全应具有以下四个方面的特征：

保密性：信息不泄露给非授权用户、实体或过程，或供其利用的特性。

完整性：数据未经授权不能进行改变的特性。即信息在存储或传输过程中保持不被修改、不被破坏和丢失的特性。

可用性：可被授权实体访问并按需求使用的特性。即当需要时能否存取所需的信息。例如网络环境下拒绝服务、破坏网络和有关系统的正常运行等都属于对可用性的攻击；

可控性：对信息的传播及内容具有控制能力。

3、主要的网络安全威胁

自然灾害、意外事故；

计算机犯罪；

人为行为，比如使用不当，安全意识差等；

“黑客”行为：由于黑客的入侵或侵扰，比如非法访问、拒绝服务计算机病毒、非法连接等；

内部泄密；

外部泄密；

信息丢失；

电子谍报，比如信息流量分析、信息窃取等；

信息战；

网络协议中的缺陷，例如 TCP/IP 协议的安全问题等等。

4、黑客常用的信息收集工具

信息收集是突破网络系统的第一步。黑客可以使用下面几种工具来收集所需信息：

SNMP 协议，用来查阅非安全路由器的路由表，从而了解目标机构网络拓扑的内部细节。

TraceRoute 程序，得出到达目标主机所经过的网络数和路由器数。

Whois 协议，它是一种信息服务，能够提供有关所有 DNS 域和负责各个域的系统管理员数据。（不过这些数据常常是过时的）。

DNS 服务器，可以访问主机的 IP 地址表和它们对应的主机名。

Finger 协议，能够提供特定主机上用户们的详细信息（注册名、电话号码、最后一次注册的时间等）。

Ping 实用程序，可以用来确定一个指定的主机的位置并确定其是否可达。把这个简单的工具用在扫描程序中，可以 Ping 网络上每个可能的主机地址，从而可以构造出实际驻留在网络上的主机清单。

4.2 计算机病毒

计算机病毒是一种程序，是人为设计的具有破坏性的程序。

计算机病毒具有破坏性、传播性、可激发性、潜伏性、隐蔽性等特点。

病毒的分类：

(1) 按病毒设计者的意图和破坏性大小，可将计算机病毒分为良性病毒和恶性病毒。

① 良性病毒：这种病毒的目的不是为了破坏计算机系统，而只是为了编制者表现自己。此类病毒破坏性较小，只是造成系统运行速度降低，干扰用户正常工作。

② 恶性病毒：这类病毒的目的是人为的破坏计算机系统的数据。具有明显破坏目标，其破坏和危害性都很大，可能删除文件或对硬盘进行非法的格式化。

(2) 计算机病毒按照寄生方式可以分为下列四类：

① 源码病毒：在源程序被编译之前，就插入到用高级语言编写的源程序当中。编写这种病毒程序较困难。但是，一旦插入，其破坏性和危害性都很大。

② 入侵病毒：是把病毒程序的一部分插入到主程序中。这种病毒程序也难编写，一旦入侵，难以清除。

③ 操作系统病毒：是把病毒程序加入或替代部分操作系统进行工作的病毒。这种病毒攻击力强、常见、破坏性和危害性最大。

④ 外壳病毒：是把病毒程序置放在主程序周围，一般不修改源程序的一种病毒。它大多是感染 DOS 下的可执行程序。这种病毒占一半以上，易编制，也易于检测和消除。

在日常维护中应隔离计算机病毒的来源，经常要用杀毒软件检查计算机系统和存储器。

例设一张软盘已染上病毒，能清除病毒的措施是_____。

- A) 删除该软盘上的所有文件
- B) 格式化该软盘
- C) 删除该软盘上的所有可执行文件
- D) 删除该软盘上的所有批处理文件

解答：软盘染毒后，病毒隐藏在磁盘内部，并感染磁盘上的文件，而且可能通过磁盘的使用进而扩散到其他磁盘，造成更大的破坏。为了清除病毒，必须格式化软盘，从而彻底清除染毒文件和病毒本身。

本题正确答案为 B。

第二部分 Pascal 语言概述与预备知识

1、关于 Turbo Pascal

Pascal 是一种计算机通用的高级程序设计语言。它由瑞士 Niklaus Wirth 教授于六十年代末设计并创立。

以法国数学家命名的 Pascal 语言现已成为使用最广泛的基于 DOS 的语言之一，其主要特点有：严格的结构化形式；丰富完备的数据类型；运行效率高；查错能力强。

正因为上述特点，Pascal 语言可以被方便地用于描述各种算法与数据结构。尤其是对于程序设计的初学者，Pascal 语言有益于培养良好的程序设计风格和习惯。IOI(国际奥林匹克信息学竞赛)把 Pascal 语言作为三种程序设计语言之一，NOI(全国奥林匹克信息学竞赛)把 Pascal 语言定为唯一提倡的程序设计语言，在大学中 Pascal 语言也常常被用作学习数据结构与算法的教学语言。

在 Pascal 问世以来的三十余年间，先后产生了适合于不同机型的各种各样版本。其中影响最大的莫过于 Turbo Pascal 系列软件。它是由美国 Borland 公司设计、研制的一种适用于微机的 Pascal 编译系统。该编译系统由 1983 年推出 1.0 版本发展到 1992 年推出的 7.0 版本，其版本不断更新，而功能更趋完善。

下面列出 Turbo Pascal 的编年史：

| 年代 | 版本名称 | 主要特色 |
|------|--------------------|-----------------------------|
| 1983 | Turbo Pascal 1.0 | |
| | Turbo Pascal 2.0 | |
| | Turbo-87 Pascal | 提高实数运算速度并扩大值域 |
| 1985 | Turbo Pascal 3.0 | 增加图形功能 |
| | Turbo BCD Pascal | 特别适合应用于商业 |
| 1987 | Turbo Pascal 4.0 | 提供集成开发环境(IDE)，引入单元概念 |
| 1988 | Turbo Pascal 5.0 | 增加调试功能 |
| 1989 | Turbo Pascal 5.5 | 支持面向对象的程序设计(OPP) |
| 1990 | Turbo Pascal 6.0 | 提供面向对象的应用框架和库(Turbo Vision) |
| 1992 | Turbo Pascal 7.0 | 面向对象的应用系统、更完善的 IDE |
| | | Turbo Vision 2.0 |
| 1993 | Borland Pascal 7.0 | 开发 Object Windows 库 |
| | (For Windows) | 提供对 OLE 多媒体应用开发的支持 |
| 1995 | Delphi | Visual Pascal |

Turbo Pascal 语言是编译型程序语言，它提供了一个集成环境的工作系统，集编辑、编译、运行、调试等多功能于一体。

2. Pascal 的启动

Pascal 的启动

a. DOS 下的启动(适用于 MS-DOS6.22 之前的版本或 Win9X & Win2000 的 Command Mode)

DOS 环境，在装有 Turbo Pascal 的文件目录下，键入 turbo 即可进入 Turbo Pascal 集成环境。

b. Win9X 或 Win2000 模式下的启动(适用于 Turbo Pascal 3.0 以后的版本)

如果在 Win9X 或 Win2000 的“资源管理器”装有 Turbo Pascal 的目录中，双击 turbo.exe 或在“开始—程序”菜单中通过 MS-DOS 方式来运行 turbo.exe，它会提示你“该程序设置为 MS-DOS 方式下运行，并且其它程序运行时，无法运行它。如果选择继续所有其它程序将关闭”，所以在 Win9X 或 Win2000 下无法直接运行它，这时你可以在你希望的地方(比如说桌面上)单击鼠标右键“新建—快捷方式”，单击“浏览”，找到 turbo.exe 选中，然后单击“打开”，再单击“下一步”，再单击完成；这还没完，选中前面新建的快捷方式(应该叫 Turbo Pascal 吧)，单击右键，单击“属性”，选择“程序”，然后再单击“高级”，把“MS-DOS 方式”前面的那个勾去掉，也就是不要选“MS-DOS 方式”，然后单击“确定”，再单击“确定”就大功告成了，以后你运行 Turbo Pascal 的时候，只要双击那个你建立起的快捷方式就可以直接在 Win9X 或 Win2000 下运行 Turbo Pascal。

第一章 开始编写 Pascal 语言程序

1.1 Pascal 编辑环境

1.2 简单的 Pascal 程序的结构

1.3 完整的 Pascal 程序结构

1.1 Pascal 编辑环境

1. 下载 Turbo Pascal6.0

(1) 下载 turbo pascal

(2) 属性设置

2. Turbo Pascal 6.0 环境介绍

(1) 进入与退出(注意:退出时必须使用文件菜单中的退出命令或 Alt+X)

(2) 编辑程序

光标移动键的使用

Backspace 键:删除光标前一个字符

Ctrl+N:在光标前插入一行

Ctrl+Y:删除光标所在行

Home:光标移到行首

End:光标移到行尾

PageUp/PageDown: 上下翻页

Insert: 插入与改写状态切换

(3) 编译程序(Compile): Alt+F9

(4) 运行程序(Run): Ctrl+F9

(5) 保存程序(Save): F2

(6) 打开原有的程序: F3

(7) 查看程序运行结果: Alt+F5

(8) 调试时增加观察项: Ctrl+F7

1.2 简单 Pascal 程序的结构

例 1 下列是一 Pascal 源程序:

```
program lt1; {程序首部}
var          {说明部分}
  a,b:integer;
  sum:integer;
begin        {执行部分}
  a:=3355;
  b:=789;
  sum:=a+b;
  writeln('sum=',sum);
end.
```

1.3 完整的 Pascal 程序结构

一个完全的 Pascal 程序结构

```
program 程序名;
uses      已知单元说明;
label     标号说明;
const     常量说明;
type      类型说明;
var       变量说明;
function  函数说明;
procedure 过程说明;
begin
  语句;
  语句;
  .....
  语句;
end.
```

第二章 Pascal 语言基础知识

2.1 Pascal 的字符与符号

2.2 Pascal 数据类型

2.3 常量与变量

2.4 标准函数

2.5 运算符和表达式

练习与作业

2.1 Pascal 字符与符号

1. 标识符

(1) 标识符的定义: 标识符就是以字母开头的字母数字序列, 有效长度为 63 个字符, 并且大小写等效。可以用来标示常量、变量、程序、函数等。例如例 1.1 中的 Area(程序名), pi(符号常量), s、r(变量名)都是标识符。

(2) 标识符的分类:

a. 保留字(关键字)

所谓保留字是指在 Pascal 语言中具有特定的含义,你必须了解它的含义,以便于正确的使用,否则会造成错误。标准 Pascal 语言中的保留字一共有 35 个, Turbo Pascal 语言一共有 51 个。下面是 Pascal 语言的保留字:

AND, ARRAY, BEGIN, CASE, CONST, DIV, DO, DOWNT0, ELSE, END, FILE, FOR, FUNTION, GOTO, IF, IN, LABEL, MOD, NIL, NOT, OF, OR, PACKED, PROCEDURE, PROGRAM, RECORD, REPEAT, SET, THEN, TO, TYPE, UNTIL, VAR, WHILE, WITH 等

b. 标准标识符: 指 Pascal 语言预先定义的标识符, 具有特殊含义。

以下列举了 Turbo Pascal 语言部分常用的标准标识符:

| | | | | | | | |
|------|---------|--------|---------|---------|-------|---------|------|
| 标准常量 | False | Maxint | True | | | | |
| 标准类型 | Boolean | Char | Real | Integer | | | |
| 标准函数 | Abs | Arctan | Chr | Cos | Eof | Eoln | Exp |
| | Ln | Odd | Ord | Pred | Round | Sin | Sqr |
| | Sqrt | Succ | Trunc | | | | |
| 标准过程 | Dispose | Get | New | Pack | Page | Put | Read |
| | Readln | Reset | Rewrite | Unpack | Write | Writeln | |
| 标准文件 | Input | Output | | | | | |

c. 用户自定义标识符: 由你自己根据需要来定义。

(1) 选用的标识符不能和保留字相同。

(2) 语法上允许预定义的标准标识符作为你自己定义的标识符使用, 但最好还是不要用。

以下列举了你自己定义标识符时可以用的字符:

A——Z; a——z; 0——9; +, -, *, /, =, <>, <=, >=, <, >, (,), [,], {, }, :=, , , ; , ., : , .., ' ,

2.2 Pascal 数据类型

数据是程序设计的一个重要内容, 其重要特征——数据类型, 确定了该数据的形、取值范围以及所能参与的运算。

Turbo Pascal 提供了丰富的数据类型, 这些数据类型可以分为三大类: 简单类型、构造类型和指针类型, 其中简单类型可以分为标准类型(整型、实型、字符型和布尔型)和自定义类型(枚举型和子界型), 构造类型可以分为数组类型、集合类型、记录类型和文件类型。这些数据类型中除了指针类型是动态数据类型外, 其他的都是静态数据类型。在这些数据类型中的简单类型都是有序类型, 除了实型以外的简单类型都是顺序类型, 所谓顺序类型就是他们的值不仅是有序的而且是有顺序号。

在这里主要介绍整型、实型、字符型和布尔型四种常用的数据类型。

1. 整型

一个整型数据用来存放整数。Turbo Pascal 支持五种预定义整型, 它们是 shortint (短整型)、integer (整型)、longint (长整型)、byte (字节型) 和 word (字类型), Turbo Pascal 分别用相同的名字作为他们的标识符。每一种类型规定了相应的整数取值范围以及所占用的内存字节数。

| 类型 | 数值范围 | 占字节数 | 格式 |
|----------|-------------------------|------|----------|
| shortint | -128..128 | 1 | 带符号 8 位 |
| inteter | -32768..32767 | 2 | 带符号 16 位 |
| longint | -2147483648..2147483647 | 4 | 带符号 32 位 |
| byte | 0..255 | 1 | 带符号 8 位 |
| word | 0..65535 | 2 | 带符号 16 位 |

Turbo Pascal 规定了两个预定义整型常量标识符 maxint 和 maxlonint, 他们各表示确定的常数值, maxint 为 32767, longint 为 2147483647, 他们的类型分别是 integer 和 longint

2. 实型

一个实型数据用来存放实数。Turbo Pascal 支持五种预定义实型, 它们是 real (基本实型)、single (单精度实型)、double (双精度实型)、extended (扩展实型)、comp (装配实型), Turbo Pascal 分别用相同的名字作为他们的标识符。每一种类型规定了相应的实数取值范围、所占用的内存字节数以及它们所能达到的精度

| 类型 | 数值范围 | 占字节数 | 有效位数 |
|--------|-------------------|------|--------|
| real | 2.9e-39..1.7e38 | 6 | 11..12 |
| single | 1.5e-45..3.4e38 | 4 | 7..8 |
| double | 5.0e-324..1.7e308 | 8 | 15..16 |

Turbo Pascal 支持两种用于执行实型运算的代码生成模式：软件仿真模式和 80x87 浮点模式。除了 real 可以在软件仿真模式下直接运行以外，其他类型必须在 80x87 浮点模式下运行。

3. 布尔型

一个布尔型数据用来存放逻辑值（布尔值）。布尔型的值只有两个：false 和 true，并且 false 的序号是 0，true 的序号是 1。false 和 true 都是预定义常数标识符，分别表示逻辑假和逻辑真。并且 true<false。boolean 是布尔型的标识符。

4. 字符型

字符型用 char 作为标识符。字符型必须用单引号括起来，字母作为字符型时，大小写是不等价的，并且字符型只允许单引号中有一个字符，否则就是字符串。

2.3 常量与变量

1. 常量

(1) 常量：在某个程序的整个过程中其值不变的量。

(2) 常量定义：常量定义出现在说明部分。它的语法格式是：

```
const
<常量标识符>=<常量>;
...
<常量标识符>=<常量>;
```

常量标识符的类型由定义它的常量的类型决定。例如：const a=12 隐含说明 a 是整型；const r=3.21 隐含说明 r 是实型.....

(3) 常量定义部分必须以保留字 const 开头，可以包含一个或几个常量定义，而且每个常量均以分号结束。

(4) Turbo Pascal 类型常量

类型常量，又称变量常数，它是 Turbo Pascal 的一个扩充特性。类型常量的定义与标准 Pascal 规定的常数定义和变量说明有所区别。类型常量定义的语法格式：

```
const
<简单类型常量标识符>:简单类型=常数;
```

例如：

```
const
counter:integer=0;
flag:boolean=true;
index:0..100=0;
```

2. 变量

(1) 变量：在某个程序中的运行过程中其值可以发生改变的量

(2) 变量说明：变量说明出现在说明部分。它的语法格式是：

```
var
<变量标识符列表>:<类型>;
...
<变量标识符列表>:<类型>;
```

其中，保留字 var 表示开始一个变量说明部分。变量标识符列表是一个用逗号隔开的标识符序列，冒号后面的类型是类型标识符。每个变量说明均以分号结束。

例如：

```
var
a,b,c:integer;
m,n:real;
```

2.4 标准函数

1. 算术函数

| 函数标识符 | 自变量类型 | 意义 | 结果类型 |
|--------|-------|-----|------|
| abs | 整型、实型 | 绝对值 | 同自变量 |
| arctan | 整型、实型 | 反正切 | 实型 |
| cos | 整型、实型 | 余弦 | 实型 |

| | | | |
|--|-------|------|------|
| exp | 整型、实型 | 指数 | 实型 |
| frac | 整型、实型 | 小数部分 | 实型 |
| int | 整型、实型 | 整数部分 | 实型 |
| ln | 整型、实型 | 自然对数 | 实型 |
| pi | 无自变量 | 圆周率 | 实型 |
| sin | 整型、实型 | 正弦 | 实型 |
| sqr | 整型、实型 | 平方 | 同自变量 |
| sqrt | 整型、实型 | 平方根 | 实型 |
| 例：abs(-4)=4 abs(-7.49)=7.49 arctan(0)=0.0 | | | |
| sin(pi)=0.0 cos(pi)=-1.0 frac(-3.71)=-0.71 | | | |
| int(-3.71)=-3.0 sqr(4)=16 sqrt(4)=2 | | | |

2. 标准函数

| 函数标识符 | 自变量类型 | 意义 | 结果类型 |
|---|-------|------|------|
| odd | 整型 | 判断奇数 | 布尔型 |
| pred | 离散类型 | 求前趋 | 同自变量 |
| succ | 离散类型 | 求后继 | 同自变量 |
| 例：odd(1000)=false pred(2000)=1999 succ(2000)=2001 | | | |
| odd(3)=true pred('x')='w' succ('x')='y' | | | |

3. 转换函数

| 函数标识符 | 自变量类型 | 意义 | 结果类型 |
|---|-------|----------|---------|
| chr | byte | 自变量对应的字符 | 字符型 |
| ord | 离散类型 | 自变量对应的序号 | longint |
| round | 实型 | 四舍五入 | longint |
| trunc | 实型 | 截断取整 | longint |
| 例：chr(66)='B' ord('A')=65 round(-4.3)=-5 trunc(2.88)=2 | | | |

4. 杂类函数

| 函数标识符 | 自变量类型 | 意义 | 结果类型 |
|-----------|-------|----------------|---------|
| random | 无自变量 | [0, 1 间的随机实数 | real |
| random | word | [0, 自变量间的随机整数) | word |
| randomize | 无自变量 | 初始化内部随机数产生器 | longint |
| upcase | 字符型 | 使小写英文字母变为大写 | 字符型 |
| downcase | 字符型 | 使小写英文字母变为大写 | 字符型 |

2.5 运算符和表达式

1. 运算符和优先级

(1) 运算符

是实型，如果全部的运算对象都是整型并且运算不是除法，则结果为整型，若运算是除法，则结果是实型 a. 算术运算符

| 运算符 | 运算 | 运算对象 | 结果类型 |
|-----|----|-------|---|
| + | 加 | 整型、实型 | 只要有一个运算对象是实型，结果就是实型，如果全部的运算对象都是整型并且运算不是除法，则结果为整型，若运算是除法，则结果是实型。 |
| - | 减 | 整型、实型 | |
| * | 乘 | 整型、实型 | |
| / | 除 | 整型、实型 | |
| div | 整除 | 整型 | 整型 |
| mod | 取余 | 整型 | 整型 |

b. 逻辑运算符

| 运算符 | 运算 | 运算对象 | 结果类型 |
|-----|------|------|------|
| not | 逻辑非 | 布尔型 | 布尔型 |
| and | 逻辑与 | 布尔型 | 布尔型 |
| or | 逻辑或 | 布尔型 | 布尔型 |
| xor | 逻辑异或 | 布尔型 | 布尔型 |

c. 关系运算符

| 运算符 | 运算 | 运算对象 | 结果类型 |
|-----|------|------|------|
| = | 等于 | 简单类型 | 布尔型 |
| <> | 不等于 | 简单类型 | 布尔型 |
| < | 小于 | 简单类型 | 布尔型 |
| > | 大于 | 简单类型 | 布尔型 |
| <= | 小于等于 | 简单类型 | 布尔型 |
| >= | 大于等于 | 简单类型 | 布尔型 |

(2) 优先级

| 运算符 | 优先级 |
|----------------------|-------|
| not | 1 (高) |
| *, /, div, mod, and | 2 |
| xor, +, -, or | 3 |
| in, =, <>, >=, <=, < | 4 (低) |

2. 表达式

(1) 算术表达式：算术表达式是由算术运算符连接常量、变量、函数的式子。算术表达式中各个运算符的次序为：

() --> 函数 --> *, /, div, mod --> +, -

(2) 布尔表达式：Turbo Pascal 提供给布尔表达式以下基本操作：逻辑运算和关系运算。

(3) 数学上的表达式与 pascal 语言表达式的区别

| 数学表达式 | PASCAL 表达式 | 注意 |
|------------|------------|----------|
| 2a | 2*a | *号不能省略 |
| $a \div b$ | a/b | 除号的写法 |
| $a \neq b$ | a<>b | 不等号的写法 |
| $a \leq b$ | a<=b | 小于等于号的写法 |

思考与练习：

1、熟记 Pascal 的保留字和标准标识符，明确自定义标识符的定义要点。

2、取整函数 int 与截断取整函数 trunc 有什么区别？举例说明。

3、判断以下标识符的合法性：

a3 3a a17 abcd ex9.5 α β λ

5、将下列的数学表达式改写成 PASCAL 表达式：

$b^2 - 4ac$

6、求下列表达式的值：

20 mod 19 15 mod 9 7 div 8 19 div 3
 (4>5) and (7<8)
 (8>9) or (9<10)
 2 and ((3=3) or (3<7))

第三章 顺序结构程序设计

3.1 赋值语句

3.2 读语句

3.3 写语句

练习与作业

3.1 赋值语句

PASCAL 有两个语句可以改变变量的值。赋值语句是其中之一（另一个是读语句）。赋值，顾名思义，就是把一个值赋予某个量。可以这理解：变量相当于装东西的容器，赋值的过程就是把东西放进容器的过程。赋值语句格式如下：

变量：=表达式；

写赋值语句有以下几点要注意：

1、赋值号 “:=”

赋值号由两个字符构成，是一个运算符。如果把这两个字符拆开，那么这两个字符就是别的意思了：“:” 是分隔符而 “=” 是关系运算符，判定两个对象是否相等。刚刚写程序的同学要特别注意这一点。

例：a, b: integer; ——是一个说明语句。“:”是变量名和变量类型的分隔符

a=b——是一个表达式。它的值是一个布尔类型的量：TRUE 或 FALSE

a:=3; ——是一个语句。把整型常量值 3 赋给整型变量 a

2、变量要先说明

在赋值号左边出现的变量，要在程序头的说明部先加以说明，否则编译时出错。

3、表达式必须要有确定的值

赋值号右边出现的表达式，必须是可以求值的。也就是说，经过运算之后，能得出一个具体的、确定的值出来。大家想一想，如果连表达式自己都不知道自己的值是多少，怎么还能把值“赋予”别人呢？

4、赋值号两边的数据类型必须相同或相容

我们知道，PASCAL 中的量不管是变量还是常量都有一个属性称为“数据类型”。数据类型相同的或相容的才可以相互赋值。

怎么来理解这句话呢？打个比方，我们沏功夫茶用的是小茶杯，装饭时用饭碗。如果用饭碗来泡功夫茶，用小茶杯来装饭，那情形一定很滑稽而且是不可行的。回到 PASCAL 中来，赋值号左边变量如果是整型，右边表达式的值的类型也要是整型；赋值号左边变量如果是字符型，右边表达式的值的类型也要是字符型……否则的话，也要出错了。这是数据类型相同的情况。

对于数据类型相容的，我们也可以用例子来帮助理解。我们都喝过功夫茶，也喝过大杯茶。把功夫茶倒在大茶杯里，一般不会出什么问题；但如果把大杯里的茶倒在功夫茶杯里呢？可能小茶杯装不下大茶杯里的茶，茶“溢出”了。在 PASCAL 中也会出现这种情况。当一种数据类型的取值范围包含着另一种数据类型的取值范围时，就可能出现类型相容的情况。如实型与整型，整型、字符型与它们各自的子界类型……如果把整型值赋给实型变量，把整型子界值赋给整型变量，不会出错；但如果反过来，就会出现“溢出”，出错了。

因此，我们在写赋值语句时，要注意两边的类型是否匹配。

例：有程序如下：

```
var a, b: integer; c: real; d: 0..100;
begin
    a:=100;
    b:=a;
    {-----以上是相同数据类型进行赋值}
    d:=100;
    b:=d;
    c:=b;
    {-----以上是相容数据类型进行赋值}
    d:=b;
    a:=c;
    {-----以上两个赋值语句都出现溢出，编译时出错}
end.
```

3.2 读语句

读语句（read 语句）和赋值语句一样，能够改变变量的值。与赋值语句不同，读语句从键盘或文件接收值赋予变量，而赋值语句则直接由程序语句获得。读语句格式如下：

```
read(变量名表);
readln(变量名表);
readln;
```

读语句是编程中用得最多的语句之一。在使用时有几点要注意：

1、变量名表。写在括号中的变量，都要在变量说明中先予以说明；变量与变量之间，以“,”分隔；

例：

```
var a,b:integer;
read(a,b);
```

2、从键盘接收数据时，要注意各种不同数据类型数据的分隔符不同。所谓分隔符就是两个完整的数值之间的标记，也可以这样理解，当计算机从键盘读入数据时，一旦碰到分隔符，就认为当前的数据读入已完成，可以把它赋给相应的变量了。各种数据类型的分隔符如下：

数值型（包括整型、实型以及它们的子界类型）以空格或回车符作为分隔符；

字符型不需分隔符（因为字符型数据的长度固定，只有一个）；

字符串以回车符作为分隔符。

3、注意 read 与 readln 的区别

例：有两段程序有相同的变量说明如下，不同的读语句，我们可以通过比较它们执行结果的异同来理解 read 与 readln 的区别。

| 变量说明 | var a,b,c,d:integer; | 执行结果 | | | |
|------|--------------------------------------|------|---|---|---|
| | | a | b | c | d |
| 程序段一 | read(a); readln(b,c); read(d); | 1 | 2 | 3 | 6 |
| 程序段二 | readln(a); read(b,c); read(d) | 1 | 6 | 7 | 8 |
| 输入数据 | 1 2 3 4 5 6 7 8 | | | | |

在程序段一执行时，“read(a);”语句接收了第一个数据 1 并将它赋给变量 a；接着执行第二个语句“readln(b,c);”，接收了第一行数据中的 2、3 并把它们分别赋给变量 b,c，同时，把本行其它数据全部屏蔽掉，也就是宣布它们全部作废。程序段二的执行情况也是如此。

因此，我们可以得出结论：语句 read 只管接收数据，语句 readln 接收完数据后，还把同行的其它数据全部宣布作废。

4、“readln;”语句从键盘接收一个回车符。这个语句通常用在需要暂停的地方。如输出时用来等待程序员看清结果。

3.3 写语句

写（write）语句是 Pascal 中唯一能将运算结果送出显示在显示器屏幕的语句。格式如下：

write(输出量表); {输出后不换行}

writeln(输出量表); {输出后换行}

writeln; {输出一个回车符}

使用写语句时也有一些小问题需要注意。

1、输出量可以是：

变量。输出变量的值。输出多个变量时，变量间用“,”分隔。

表达式。输出的是表达式的值。

常量。直接输出常量值。

2、场宽的限制在输出不同格式的数值时的作用：

例 1：输出多个空格。

write('':n);句子的意思是以 n 个字符宽度输出冒号前数据项，如果数据项长度不足 n，则前面以空格补齐；如果数据项长度大于 n，则以实际长度输出。如上语句输出 n 个空格。

例 2：数据项间隔。

如输出最多四位的数据：write(x:5)。则数据间至少分隔一个空格。

例 3：实型数据小数位数的确定。

实型数据不带格式限制时，以科学计数法的形式输出，和我们的一般书写习惯不同。如果加上场宽的限制，则可以有不同的效果：

```
var a:real;
begin
a:=15/8;
writeln(a);{输出 1.8750000000E+00}
writeln(a:0:2);{输出 1.88 整数部分按实际位数输出，小数部分保留两位小数，末位四舍五入。}
writeln(a:0:0);{输出 2 只输出整数部分，小数部分四舍五入}
end.
```

3、“writeln;”语句通常用于输出多组数据时在屏幕上输出空行来分隔数据组。

思考与练习：

- 1、用字符输出一个猪锦佳的头像。
 - 2、a, b, c 分别等于 1、12、123，把它们按向左对齐、向右对齐的方式打印出来。
 - 3、输入一个四位整数，把它的各位数字倒序输出。（提示：用 MOD 和 DIV 运算完成）
 - 4、从键盘上读入小写的“pascal”，利用 CHR（）和 ORD（）函数，输出大写的“PASCAL”。
 - 5、从键盘上读入一个实数，利用 ROUND（）和 TRUNC（）函数，输出该实数本身、整数部分、小数部分、四舍五入后的值。
- 要求：分三行输出；输出实数本身时，格式与读入时相同；整数部分、小数部分在同一行输出；其它各占一行。
- 6、从键盘上读入长方形的边长 a, b，计算它的面积和周长，输出。
 - 7、输入一个时、分、秒，把它转换为一个秒数。

第四章 选择结构程序设计

4.1 IF 语句

4.2 CASE 语句

练习与作业

4.1 IF 语句

条件语句用于响应一个条件的两个方面。

例如：今天如果下雨，我们就在家；否则（不下雨）我们就去旅游。

又如：如果已经搜索得到结果，就打印出答案；否则（还没得到结果）就继续搜索。

IF 语句的一般格式是：

IF 条件

THEN 语句 1 {条件为真时的响应、处理}

ELSE 语句 2; {条件为假时的响应、处理}

使用条件语句时要注意：

- 1、条件语句是一个语句。IF、THEN、ELSE 都是语句的一个部分。所以它只能有一个“；”作为分隔符，放在句子的结束，特别要注意不能放在 ELSE 之前。
- 2、如果我们的程序只需对条件为真的情况作出处理，不需要处理条件为假的情况，则 IF 语句省略 ELSE 分句，格式变成：

IF 条件

THEN 语句 1; {条件为真时的响应、处理}

如：如果数 a 大于等于 0 则输出它的平方根。

```
if a>=0 then writeln(sqrt(a));
```

对以上的例子，条件为假时不需处理，于是我们干脆省去 ELSE 分句。

- 3、if 语句可以多层嵌套。嵌套时为了避免误解，可以用 begin, end 括起嵌套部分；else 分句一般和最近的 if 分句配套：

IF 条件

THEN BEGIN

if 条件 1 then else;

END

ELSE BEGIN

if 条件 2 then else;

end;

例 1、输入两个数 a, b，输出较大的数。

```
program tt;
var a,b:integer;
begin
  write('please input a,b:');
  readln(a,b);
  if a>b then writeln(a)
  else writeln(b);
```

```
end.
```

4.2 CASE 语句

分情况语句适用于对一个条件的多种情况的响应。

格式：

```
case 表达式 of
    标号 1: 语句 1;
    标号 2: 语句 2;
    .....
    标号 n: 语句 n;
else 语句 n+1
end;
```

case 语句在使用时有几点要注意：

1. end 与 case 对应；标号与语句之间用“：”分隔；else 与语句之间不用分隔符。
2. 标号必须是一常量，其类型与表达式的类型一致

例 2：某全自动加油站 a, b, c 三种汽油的单价(元/kg)分别是 1.50、1.35 和 1.18, 也提供了“自己加”或“协助加”两个服务等级，这样用户可以得到 5%或 10%的优惠。编一个程序，用户输入加油量、汽油品种和服务类型(f-自动, m-自己, e-协助)，然后计算应付款。

```
program pcase1;
var
    oil,help:char;
    kg,total:real;
begin
    write('Enter the amount in kilograms(kg):'); readln(kg);
    write('Which type of the gasoline(a,b,c):'); readln(oil);
    write('Which type for service(f,m,e):'); readln(help);
    case oil of
        'a': total:=1.50*kg;
        'b': total:=1.35*kg;
        'c': total:=1.18*kg;
        else writeln('Input Error!')
    end;
    {———处理汽油的类型}
    case help of
        'f':;
        'm': total:=total*(1-0.05);
        'e': total:=total*(1-0.10);
        else writeln('Input Error!')
    end;
    {———处理服务类型}
    writeln;
    writeln('Total is ',total:10:2);
end.
```

3. 可以多个标号对应同一条语句
4. 语句可以是多个语句，但必须用语句括号(begin……end)括起
5. case 语句也可以嵌套

例 3：从键盘上读入年和月，输出该月有多少天。

```
program pcase2;
var
    year,month,day:integer;
    runnian:boolean;
begin
```

```

write('Enter year and month:'); readln(year,month);
case month of
  1,3,5,7,8,10,12: day:=31;
  4,6,9,11: day:=30; {——以上处理 31 天和 30 天的情况}
  2:begin
    runnian:=(year mod 400=0) or ((year mod 4=0) and (year mod 100<>0));
    case runnian of
      true: day:=28;
      false: day:=29;
    end;
  end; {——以上处理 2 月的情况：闰年 28 天，平年 29 天}
end;
end.

```

思考与练习：

- 1、从键盘上读入长方形的边长 a, b, 计算它的面积和周长，输出。
- 2、输入一个时、分、秒，把它转换为一个秒数。
- 3、从键盘读入一个数，判断它的正负。是正数，则输出“+”，是负数，则输出“-”。
- 4、输入两个数 a, b, 输出较大数的平方值。
- 5、铁路托运行李规定：行李重不超过 50 公斤的，托运费按每公斤 0.15 元计费；如超 50 公斤，超过部分每公斤加收 0.10 元。编一程序完成自动计费工作。
- 6、某超市为了促销，规定：购物不足 50 元的按原价付款，超过 50 不足 100 的按九折付款，超过 100 元的，超过部分按八折付款。编一程序完成超市的自动计费的工作。
- 7、输入 a, b, c 三个不同的数，将它们按由小到大的顺序输出。13、当前小学生的成绩单由以前的百分制改为优秀、良好、合格、不合格四个等级的等级制。编一程序完成分数的自动转换工作。转换规则如下：60 分以下的为不合格；60 到 69 分为合格；70 到 89 分为良好；90 分以上的为优秀。（提示：可以利用 DIV 运算来使程序更简明）
- 8、打印某年某月有多少天。（提示：A、闰年的计算方法：年数能被 4 整除，并且不能被 100 整除；或者能被 400 整除的整数年份。B、利用 MOD 运算可以判断一个数能否被另一个数整除）
- 9、编程模拟剪刀、石头、布游戏：用 S 表示剪刀，用 R 表示石头，用 P 表示布。规则是：剪刀剪布，石头砸剪刀，布包石头。游戏者分别把自己的选择输入，计算机给出结果。

第五章 循环结构程序设计

5.1 For 语句

5.2 While 语句

5.3 Repeat-Until 语句

练习与作业

5.1 For 语句

1. 递增型 FOR 循环。

FOR 循环控制变量:=循环初值 TO 循环终值 DO 循环的语句（或语段）

例：FOR I:=5 TO 10 DO WRITELN (I);

输出的结果为：5 6 7 8 9 10 即循环一共执行了 6 次

如果要重复多个语句，一定要用 BEGIN-END 形式：

例：

```

FOR I:=1 TO 10 DO
BEGIN
  WRITELN (I);
  WRITELN (10-I);
END;

```

2. 递减型 FOR 循环

FOR 循环控制变量:=循环初值 DOWNTO 循环终值 DO 循环语句

递减型 FOR 循环与递增型 FOR 循环基本相同，只是循环控制变量每次递减。

3. FOR 循环的几点注意内容：

(1) 循环控制变量必须是顺序类型的变量。所谓顺序类型的变量，就是指整型，字符型，枚举型，子界型，不允许是实型。

(2) 不允许在循环体内再对循环控制变量赋值。

例如：

```
A:=10;B:=50;
FOR K:=A TO B DO
BEGIN
K:=K+1;{这一句是错误的!!!!!!}
Writeln (K);
END;
```

(3) 当循环初值或循环终值中包含变量时，允许在循环体内改变这些变量的值，但并不改变原定的循环次数。

例：

```
A:=1;B:=10;
FOR I:=A TO B DO
BEGIN
A:=5;B:=4;
END;
```

在上面例子中，A，B 的值在循环的内部发生了变化，但并不影响循环的次数，依然是 10 次。

4. 多重循环 循环体由 PASCAL 语句构成，当然也可以包含 FOR 语句，这就构成了循环的嵌套，形成多重循环。

例如，以下 FOR 循环输出 5 行，每行输出 10 个星号(*)

```
FOR i:=1 to 5 DO
BEGIN
FOR j:=1 TO 10 DO
Write('*');
END;
```

初学者应当特别注意，内层的循环变量不能和外层的循环变量相同。也就是说，嵌套的各层循环应当使用不同的变量作为循环变量。

5.2 While 语句

1. WHILE 循环的执行形式 WHILE 布尔表达式 DO 语句

例如：

```
k:=10;
WHILE k>0 DO
BEGIN
Writeln (k);
k:=k-1
END;
```

其中

(1) WHILE 和 DO 是 PASCAL 保留关键字，是 WHILE 循环语句的组成部分。

(2) 保留关键字 DO 后面的“语法”只能是一条语句，称为“循环体”；如果循环体中需要包含多个语句则应该如上例所示，采用一条复合语句。

2. WHILE 循环的执行功能 当执行到 WHILE 语句时

(1) 求出布尔表达式的值

(2) 若布尔表达式的值为真，则执行循环体内的语句；若为“假”，执行步骤 4

(3) 重复步骤 1 和 2

(4) 循环结束，执行循环后面的语句。

5.3 Repeat-Until 语句

1. REPEAT-UNTIL 类型的循环的执行形式

```
REPEAT
语句 1
语句 2
```


.....

语句 n

UNTIL 布尔表达式

例如：以下循环求 $n=1+2+3+\cdots+100$

```
n:=0;t:=1;
REPEAT
n:=n+t; t:=t+1;
UNTIL t>100;
```

其中

(1) REPEAT 和 UNTIL 是 PASCAL 保留关键字。

(2) 在 REPEAT 和 UNTIL 之间的语句构成循环。在它们之间可以有任意多个语句，这一点和 FOR，WHILE 循环不同，FOR，WHILE 循环体在语法上只允许一条语句。

2. REPEAT-UNTIL 循环的执行功能

(1) 遇到 REPEAT 语句后，即进入循环体，顺序执行循环体内的语句。

(2) 遇到 UNTIL 语句后，求布尔表达式的值。若值为假，则返回步骤 1；若为“真”，执行步骤 3

(3) 循环结束，执行 UNTIL 后面的下一条语句。

思考与练习：

1、计算下列式子的值：（1） $1+2+\cdots+100$

（2） $1+3+5+\cdots+97+99$

2、输入一个四位数，求它各位上数字的和。

3、求水仙花数。所谓水仙花数，是指一个三位数 abc，如果满足 $a^3+b^3+c^3=abc$ ，则 abc 是水仙花数。

4、宰相的麦子：相传古印度宰相达依尔，是国际象棋的发明者。有一次，国王因为他的贡献要奖励他，问他想要什么。达依尔说：“只要在国际象棋棋盘上（共 64 格）摆上这么些麦子就行了：第一格一粒，第二格两粒，……，后面一格的麦子总是前一格麦子数的两倍，摆满整个棋盘，我就感恩不尽了。”国王一想，这还不容易，刚想答应，如果你这时在国王旁边站着，你会不会劝国王别答应，为什么？

5、打印下列图案：（输入 N 值来控制图案的规模，下列图案均以 N=3 为例）

```
&
& &
& & &
& & &
& & &
& & &
*
* * *
* * * * *
```

```
# @
# # @ @
# # # @ @ @
```

6、输入一整数 A，判断它是否质数。（提示：若从 2 到 A 的平方根的范围内，没有一个数能整除 A，则 A 是质数。）

7、求两个数的最小公倍数和最大公约数。（提示：公约数一定小于等于两数中的小数，且能整除两数中的大数。公倍数一定大于等于两数中的大数，且是大数的倍数，又能给两数中的小数整除。）

8、编写一个译码程序，把一个英语句子译成数字代码。译码规则是以数字 1 代替字母 A，数字 2 代替字母 B，……，26 代替字母 Z，如遇空格则打印一个星号 ‘*’，英文句子以 ‘.’ 结束。

9、“百钱买百鸡”是我国古代的著名数学题。题目这样描述：3 文钱可以买 1 只公鸡，2 文钱可以买一只母鸡，1 文钱可以买 3 只小鸡。用 100 文钱买 100 只鸡，那么各有公鸡、母鸡、小鸡多少只？与之相似，有“鸡兔同笼”问题。

10、输入一个正整数 N，把它分解成质因子相乘的形式。

如： $36=1 \times 2 \times 2 \times 3 \times 3$ ； $19=1 \times 19$

（提示：设因子为 I，从 2 开始到 N，让 N 重复被 I 除，如果能整除，则用商取代 N，I 为一个因子；如果不

能整除，再将 I 增大，继续以上操作，直到 I 等于 N。）

第六章 数组与字符串

- 6.1 一维数组
- 6.2 二维数组
- 6.3 字符串
- 练习与作业

6.1 一维数组

1、定义：

方法一

```
var
    a:array [1..10] of integer;
```

方法二

```
Type munber=array[1..10] of integer;
Var a:munber
```

其中：a 是这一批数据的名称，称为数组名；array、of 是定义数组的保留字；中括号中的数字是数据编号的下限和上限，同时也说明了数据的个数（上限-下限）；最后一个数据的基类型，如 integer，char，real，boolean。

2、数组元素的输入：

数组名代表的并不是一个变量，而是一批变量，因而，不能直接整个数组读入，而是要逐个数组元素读入，通常用循环结构来完成这一功能。下面是几个常用输入数组元素的例子：

```
for i:=1 to 10 do read(a[i]);
    {——从键盘读入数组元素的值；最常用的方法}
for i:=1 to 10 do a[i]:=i;
    {——数组元素 a[1]到 a[10]的值分别为 1 到 10；数据赋初值}
for i:=1 to 10 do a[i]:=0;
    {——数组元素清 0；最常用的数据初始化的方法}
for i:=1 to 10 do a[i]:=random(100);
    {——随机产生 10 个 100 以内的数，赋给各数组元素}

Const c:array[1..10] of integer = (2,4,6,8,10,1,3,5,7)
```

3、数组元素的输出：

和数组元素的输入相同，数组元素的输出也不能由一个 write 语句直接完成。同样要逐个数组元素输出。通常也用循环结构来完成这一功能：

```
for i:=1 to 10 do write(a[i], ' ');{——数组元素之间用空格分隔}
writeln;
```

4、数组的应用：

例 1：从键盘输入 10 个数，将这 10 个数逆序输出，并求这 10 个数的和，输出这个和。

```
program p1;
var
    a:array [1..10] of integer;
    i,s:integer;
begin
    for i:=1 to 10 do read(a[i]);
    for i:=10 downto 1 do write(a[i], ' ');
    writeln;
    s:=0;
    for i:=1 to 10 do s:=s+a[i];
```

```
writeln('s=',s);
end.
```

例 2：用筛法求 100 以内的素数（质数）。

分析：素数是除了 1 和它本身以外没有其它约数的数。用筛法求素数的方法是：用质数筛去合数：从第一个素数 2 开始，把它的倍数去掉；这样 2 以后的第一个非 0 数就一定也是素数，把它的倍数也删了……重复这个删数过程，直到在所找到的素数后再也找不到一个非 0 数。把所有非 0 数输出。

```
program p2;
var
  a:array [1..100] of integer;
  i,j,k:integer;
begin
  for i:=1 to 100 do a[i]:=i;
  a[1]:=0;i:=2;
  while i<=100 do
  begin
    k:=i;
    while k<=100 do
    begin
      k:=k+i;
      a[k]:=0;
    end;
    {——上面将所有 a[i]的倍数清 0}
    i:=i+1;
    while a[i]=0 do i:=i+1;
    {——查找接下来的第一个非 0 数}
  end;
  for i:=1 to 100 do if a[i]<>0 then write(a[i],' ');
end.
```

6.2 二维数组

一维数组在编程中多用于描述线性的关系：如一组数；一组成绩；一组解答等。数组元素只有一个下标，表明该元素在数组中的位置。二维数组在编程中多数用于描述二维的关系：如地图、棋盘、城市街道、迷宫等等。而二维数组元素有两个下标：第一个下标表示该元素在第几行，第二个下标表示在第几列。

1. 二维数组的定义：

```
var
  a: array[1..10,1..5] of integer;
Type
number= array[1..10,1..5] of integer;
Var a:number
```

其中：a 是数组名，由程序员自定；array 和 of 是定义数组的保留字；（这两点和一维数组定义的格式一样）中括号中的两个范围表示二维数组共有多少行、多少列（第一个范围表示行数，第二个范围表示列数）；最后一个表示数组元素的类型，规定和一维数组一样。如上例，定义了一个二维数组 a，共有 10 行 5 列。

2. 使用二维数组：

1、数组元素的指称：数组名[行号，列号]。如第三行第四个元素：a[3,4]。

对某一行进行处理。如累加第 4 行的数据。则固定行号为 4。如：for i:=1 to 5 do s:=s+a[4,i];

对某一列进行处理。如累加第 4 列的数据。则固定列号为 4。如：for i:=1 to 10 do s:=s+a[i,4];

2、二维数组的输入输出要用双重循环来控制：

```
for i:=1 to 10 do{——控制行数}
begin
  for j:=1 to 5 do read(a[i,j]){——第一行读入 5 个元素}
  readln;{——读入一个换行符}
end;
```

```
{——最常用的方法：从键盘读入数据初始化二维数组}
for i:=1 to 10 do
  for j:=1 to 5 do a[i,j]:=0;
{——最常用的方法：将二维数组清 0}
for i:=1 to 10 do
begin
  for j:=1 to 5 do write(a[i,j]:4);
  writeln;
end;
{——最常用的输出方法：按矩阵形式输出二维数组的值}
```

例 1：竞赛小组共有 5 位同学，这学期每位同学共参与了三场比赛，请统计每位同学的平均分。
分析：定义一个 5 行 3 列的二维数组来存放这些成绩。定义一个 5 个元素的一维数组来存放平均分。

```
program p1;
var
  a:array [1..5,1..3] of integer;
  b:array [1..5] of real;
  i,j:integer;
begin
  for i:=1 to 5 do
  begin
    for j:=1 to 3 do read(a[i,j]);
    readln;
  end;
{——从键盘上读入 5 个同学的三次竞赛成绩}
  for i:=1 to 5 do b[i]:=0;
{——先将平均分数组清 0}
  for i:=1 to 5 do
  begin

    for j:=1 to 3 do b[i]:=b[i]+a[i,j];{——计算总分}
    b[i]:=b[i]/3;{——计算平均分}
  end;
  for i:=1 to 5 do write(b[i]:5:1);
{——输出平均分}
writeln;
end.
```

6.3 字符串

1. 字符串用于存放整批的字符数据。通常编程中使用字符串存放字符化了的数字数据。如高精度运算时存放操作数和运算结果。字符串可以看作是特殊的字符串数组来处理。当然，它也有自己的特点。下面是字符串定义的格式：

```
var
s:string;
s1:string[15];
```

字符串定义时，如不指定长度，则按该类型的最大长度（255 个字符）分配空间，使用时最大可用长度为 255 个；如果在 中括号中给出一个具体的值（1—255 之间），则按这个值的大小分配空间。使用时，最大的可用长度即为该值。

2. 字符串的输入、输出：

字符串类型既可按数组方式输入、输出，也可直接输入、输出：readln(s); writeln(s); 多个字符串输入时以回车作为数据间的分隔符；每个 readln 语句只能读入一个字符串。

3. 有关字符串的操作：

| 操作 | 类型 | 作用 | 返回值 | 例子 |
|----|----|----|-----|----|
|----|----|----|-----|----|

| | | | | |
|------------------|-----|---------------------------------|-----|--|
| length(s) | 函数 | 求字符串 s 的长度 | 整型 | s:='123456789'; l:=length(s); {l 的值为 9} |
| copy (s,w,k) | 函数 | 复制 s 中从 w 开始的 k 位 | 字符串 | s:='123456789'; s1:=copy(s,3,5); {s1 的值是 '34567'} |
| val(s,k,code) | 过程 | 将字符串 s 转为数值, 存在 k 中; code 是错误代码 | | var s:string;k,code:integer; begin s:=' 1234' ; val (s, k, code) ; write(k) ; {k=1234} |
| str(i,s) | 过程 | 将数值 i 转为字符串 s | | i:=1234; str(i,s); write(s); {s='1234'} |
| Delete(s,w,k) | 过程 | 在 s 中删除从第 w 位开始的 k 个字符 | | s := 'Honest Abe Lincoln'; Delete(s,8,4); Writeln(s); { 'Honest Lincoln' } |
| Insert(s1, S, w) | 过程 | 将 s1 插到 s 中第 w 位 | | S := 'Honest Lincoln'; Insert('Abe ', S, 8); { 'Honest Abe Lincoln' } |
| Pos(c, S) | 函数 | 求字符 c 在 s 中的位置 | 整型 | S := ' 123.5'; i :=Pos(' ', S); {i 的值为 1} |
| + | 运算符 | 将两个字符串连接起来 | | s1:='1234'; s2:='5678'; s:=s1+s2; {'12345678'} |

思考与练习:

- 1、随机产生 20 个 100 以内的数，输出；按从小到大的顺序排序，输出。
- 2、求一个 5 X 5 数组中的马鞍数，输出它的位置。所谓马鞍数，是指在行上最小而在列上最大的数。如下：

```

5 6 7 8 9
4 5 6 7 8
3 4 5 2 1
2 3 4 9 0
1 2 5 4 8

```

则 1 行 1 列上的数就是马鞍数。

```

var
  a:array [1..5,1..5] of integer;
  b,c:array [1..5] of integer;
  i,j:integer;
begin
  for i:=1 to 5 do for j:=1 to 5 do read(a[i,j]);
  for i:=1 to 5 do begin
    b[i]:=a[i,1]; c[i]:=a[1,i];
    for j:=2 to 5 do begin
      if a[i,j]<b[i] then b[i]:=a[i,j];
      if a[j,i]>c[i] then c[i]:=a[j,i];
    end;
  end;
  for i:=1 to 5 do for j:=1 to 5 do
    if (a[i,j]=b[i]) and (a[i,j]=c[j])
    then writeln('(',i,',',j,')');
  readln;
end.

```

- 3、做一个加法器。完成 30000 以内的加法，两个加数间用“+”连接，可以连加，回车表示式子输入完成；“#”表示

结束运算，退出加法器。

第七章 函数和过程

8.1 过程

8.2 函数

8.3 全局变量和局部变量

8.4 值参数和变量参数

练习与作业

7.1 过程

1. 过程的定义

procedure 过程名(形式参数:参数说明);{也可以不带参数}

var

begin

...

end;

2. 过程的调用

过程名(实在参数表);

例 1: 求 n 个自然数的最大公约数;

```
program gcd1;
const maxn=100;
var n,i,gcd:integer;
    a:array[1..maxn] of integer;
procedure enter;
begin
    write('n=<100) ');readln(n);
    for i:=1 to n do
        repeat
            write('a[' ,i, ']= ');readln(a[i]);
        until a[i]>0;
end;
procedure find_gcd(x,y:integer);{定义过程}
var r:integer;
begin
    r:=x mod y;
    while r<>0 do
        begin x:=y;y:=r;r:=x mod y; end
    gcd:=y;
end;

procedure print;
begin
    writeln('GCD=' ,gcd);
end;
begin
    enter;
    gcd:=a[1];
    for i:=2 to n do
        find_gcd(gcd,a[i]);
    print;
end.
```

7.2 函数

1. 函数的定义

```
function 函数名(形参表):函数类型; { ——函数首部}
var {——局部变量说明部分}
begin {——函数体}
... {——函数语句}
...
函数名:=表达式
end;
```

2. 函数的调用:

函数在语法上相当于一个表达式，所以，调用时，函数不能独立成为一个语句；它可以出现在任何表达式可以出现的地方。

例如赋值语句的右边：

$X := \text{函数名}(\text{实在参数表})$ ；{—— X 的类型与函数类型必须一致}

又，如果函数类型是 boolean，则还可以出现在条件语句中，充当条件表达式：

if 函数名(实在参数表) then

例 3：编一程序，求从 10 名同学中选出 3 名代表，有几种不同的选法。

（公式： $C(m, n) = m! / n! * (m - n)!$ 从 m 中选 n ）

```
program zohel;
var m,n:integer;
    c:longint;
function factor(x:integer):longint;{定义}
var i:integer;
p:longint;
begin
p:=1;
for i:=1 to x do p:=p*i;
factor:=p;{这个语句必须}
end;
begin
write('m,n=');readln(m,n);
c:=factor(m) div (factor(n)*factor(m-n));{调用}
writeln('c(',m,',',n,')=',c);
end.
```

7.3 全局变量和局部变量

在子程序中定义的变量称为局部变量，在程序的一开始定义的变量称为全局变量。全局变量作用域是整个程序；局部变量作用域是定义该变量的子程序。当全局变量与局部变量同名时：在定义局部变量的子程序内，局部变量起作用；在其它地方全局变量起作用。

例 4：全局变量和局部变量。

```
program local_global;
var i,k:integer;
procedure subl;
var i,j:integer;
begin
i:=17;
writeln('i in sub=',i);
writeln('k in sub=',k);
end;
begin
i:=2;k:=9;
writeln('i in main=',i);
```

```
writeln('k in sub=',k);
sub1;
writeln('i in main=',i);
writeln('j in main=',j);
readln;
end.
```

上述程序运行时将出现 Unknown Identifier; 因为变量 j 在主程序中没定义。

当删除本语句时。运行结果如下:

```
i in main=2
k in main=9
i in sub=17
k in sub=9
i in main=2
```

7.4 值参和变量参数

值形参——传值: 调用时可用表达式代替形参, 不该变实在参数的值。

变量形参——传地址: 调用时必须用变量代替行参变量, 改变实在参数的值。

例 4: 请看下列程序的运行结果。

```
program li3_10;
var  a,b,c:integer;
procedure sub(x,y:integer;var z:integer);
begin
  x:=x+1;y:=y+1;z:=x+y;
  writeln('sub:x=',x:2,'y=',y:2,'z=',z:2);
end;
begin
  a:=1;b:=4;c:=9;
  writeln('main:a=',a:2,'b=',b:2,'c=',c);
  sub(a,b,c);
  writeln('main:a=',a:2,'b=',b:2,'c=',c);
  sub(c+5,b*b,a);
  writeln('main:a=',a:2,'b=',b:2,'c=',c);
  readln;
end.
```

```
main: a=1 b=4 c=9
sub: x=2 y=5 z=7
main: a=1 b=4 c=7
sub: x=13 y=17 z=30
main: a=30 b=4 c=7
```

练习与作业

- 1、试编写一个将阿拉伯数字转换为中文大写数字的函数。(如中文状态不方便, 可以替换为 ABCDEFGHIJ)
- 2、输入一个长字符串和一个短字符串以及一个合适的整数, 通过程序从指定位置用指定短字符串替换长字符串中的内容(要求编写过程来实现), 最后输出新字符串。

第八章 子界与枚举类型

8.1 子界类型

8.2 枚举类型

Pascal 有丰富的数据类型, 在程序设计中特殊而方便的应用。

类型定义的语法格式:

```
type
  <标识符 1>=<类型 1>;
  <标识符 2>=<类型 2>;
```


.....

〈标识符 n〉=〈类型 n〉;

8.1 子界与枚举

1. 子界类型:

当某些变量的取值范围很具体时, 可用子界类型, 它更符合实际、便于查错和节省内存。

定义如下:

```
type riqi=1..31;
      zimu='A'..'Z';
var day:riqi;
     chl:zimu;
```

也可以直接定义:

```
day: 1..31;
year:0..200;
chl:'A'..'Z';
```

8.2 枚举类型:

通过预定义列出所有值的标识符来定义一个有序集合, 这些值的次序和枚举类型说明中的标识符的次序是一致的。枚举类型的形式:

(标识符 1,, 标识符 n)

例如:

```
type daystype=(sunday, monday, tuesday, wednesday, thursday, friday, saturday)
```

枚举元素只能是标识符, 而不能是数值常量或字符常量。例如以下的定义是错误的: <

```
type daystype=(' sun', ' mon', ' tue', ' wed', ' thu', ' fri', ' sat')
```

枚举元素是标识符, 不要把作为枚举元素的标识符视作变量名, 它不能被赋值。同一个枚举元素不能出现在两个或两个以上的枚举类型定义中。例如以下的定义是错误的:

```
type daytype1=(monday, tuesday);
daytype2=(monday, wednesday);
```

可以将枚举类型的定义和变量的定义结合在一起。例如: var a: (monday, tuesday, sunday)

枚举类型属于顺序类型。根据定义类型时各枚举元素的排列顺序确定它们的序列, 序列号从 0 开始。

例如: 已经定义 daystype

```
ord(sunday)=0, succ(sunday)=monday, pred(friday)=thursday
```

但是枚举类型中的第一个元素没有前趋, 最后一个元素没有后继。Turbo Pascal 不允许直接读写枚举值, 所以枚举值的输出常用 case 语句间接的输出。枚举值的输入, 则要一一判断读入字符是否是枚举类型的标识符。若是才能赋给枚举变量, 否则就会出错。

例如: 枚举值的输出

```
case day of
sunday:write('sunday');
monday:write('monday');
tuesday:write('tuesday');
wednesday:write('wednesday');
thursday:write('thursday');
friday:write('friday');
saturday:write('saturday');
end;
```

练习:

1. 按月、日顺序输入今年的一个日期, 输出该日是星期几? 已知今年元旦是星期一。

第九章 集合类型

9.1 集合

9.1 集合

1. 集合的定义:

type 类型名=set of 基类型

例如:

```
type
  num=set of char;
var
  n:num;
```

或

```
var
  n: set of char;
```

2. 集合的表示:

用一组方括号括号一组元素来表示, 元素之间用逗号分隔。如:

[A,B,C,D]——有四个枚举量的集合

['A','B','C','D']——有四个字符的集合

[1..20]——包含了 1 到 20 中所有整数的集合

[0]——只有一个元素 0 的单元集

[]——空集

3. 集合的运算:

(1) 并(a+b): 属于 a 或属于 b

[0..7]+[0..4]的值为[0..7]

(2) 交(a*b): 既属于 a 又属于 b

[0..7]*[0..4]的值为[0..4]

(3) 差(a-b): 属于 a 但不属于 b

[0..7]-[0..4]的值为[5..7]

(4) 相等(a=b): a,b 的元素完全一样

[0..7]=[0..4]的值为 false

(5) 不等(<>): 元素不一样

[0..7]<>[0..4]的值为 true

(6) 包含于(<=):

[0..7]<=[0..4]的值为 false

(7) 包含(>=):

[0..7]>=[0..4]的值为 true

(8) 成员(in):

1 in [0..4]的值为 true

4. 注意事项:

(1) 集合运算相当快, 在程序中常用集合表达式来描述复杂的测试。如

A) 条件表达式: (ch='T') or (ch='t') or (ch='Y') or (ch='y') 可用集合表达式表示为:

ch in ['T','t','Y','y']

B) if (ch>=20) and (ch<=50) then ...;

可写成:

if ch in [20..50] then ...;

(2) 集合类型是一种使用简便, 节省内存面又运算速度快的数据类型。

(3) Turbo Pascal 规定集合的元素个数不超过 256 个(当实际问题所需的元素个数大于 256 时, 可采用布尔数组代替集合类型)。所以如下定义是错误的: var i: set of integer;

(4) 集合类型变量不能进行算术运算, 了不允许用读/写语句直接输入/输出集合。所以集合的建立:

A) 要通过赋值语句实现;

B) 或先初始化一个集合, 然后通过并运算向集合中逐步加入各个元素.

(5) 集合元素是无序的, 所以 ord, pred 和 succ 函数不能用于集合类型的变量。

练习:

编程读入两个字符串, 然后输出如下信息:

(1) 出现在某一个字符串中至少一次的字母和数字;

- (2) 同时出现在两个字符串中至少一次的字母和数字;
- (3) 出现在一个字符串中而不出现在另一个字符串中的字母和数字;
- (4) 不出现在任何字符串中的字母和数字。

第十章 记录与文件类型

10.1 记录类型

10.2 文件类型

10.1 记录

1. 记录的定义:

```
type 类型标识符=record
    字段名 1: 类型 1;
    字段名 2: 类型 2;
    ...
    字段名 n: 类型 n;
end;
```

如:

```
type
    studata=record
        num:string[6];
        name:string[8];
        sex:boolean;
        s:array[1..5] of real;
    end;
var
    student:studata;
    students:array[1..10] of studata;
```

2. 记录的运用:

(1) 对记录中和个域的引用, 要写出记录名和域名, 如: student.num

(2) 开域语句: with。

with 记录名 do 语句;

或

with 记录名 1, 记录名 2, ... do 语句;

注意:

1. 在 do 后面语句中使用的记录的域时, 只要简单地写出域名就可以了, 域名前的记录变量和“.”均可省略。
2. 在关键字 with 后面, 语句可以是一个简单语句, 了可以是一个复合语句。
3. 虽然在 with 后可以有多个记录变量名, 但一般在 with 后只使用一个记录变量名。

10.2 文件

文件是一种构造型的数据类型。在程序中都需要产生一些输出, 也需要接受若干个输入。这些输入、输出实际上是用文件的方法来实现的, 在 Pascal 中用标准文件“input”和“output”来实现, 它们分别对应标准输入设备和标准输出设备（可省略不写）这也就是一些程序的程序书写如下的原因了:

```
program ex(input,output);
```

...

但有时大量数据的读入和输出都是来是磁盘文件, 这就要求我们必须熟练掌握对磁盘文件的操作。

对于我们来说, 我们只必须掌握文本文件（或称正文文件, text）的读写即可:

1. 文本文件的定义:

文本文件不是简单地由某类型的元素序列所组成, 它的基本元素是字符, 由它们构成行, 若干行组成一份原文。由于各行的长度可以不同, 所以文本文件只能顺序地处理。文本文件的定义如下:

```
var
    fp:text;
```

2. 文本文件的读操作:

- (1) 调用 assign 过程, 把磁盘文件赋予文本文件变量;
assign(fp, filename);
- (2) 调用 reset 过程, 为读操作做准备;
reset(fp);
- (3) 在需要读数据的位置调用 read 过程或 readln 过程。
readln(fp, var1, var2, ..., varn);

3. 文本文件的写操作:

- (1) 调用 assign 过程, 把磁盘文件赋予文本文件变量;
assign(fp, filename);
- (2) 调用 rewrite 过程, 为读操作做准备;
rewrite(fp);
- (3) 在需要读数据的位置调用 write 过程或 writeln 过程。
writeln(fp, var1, var2, ..., varn);

4. 文本文件的关闭操作:

close(fp);

5. 文本文件的其他操作:

- (1) EOF(fp)—布尔函数, 用于判断文件结束否。
- (2) EOLN(fp)—布尔函数, 用于判断行结束否。

例 1: 下面是一个建立和使用文件的程序:

```
program wenjian;
const n=3; m=2;
type student=record
    name:string;
    score:array[1..m] of 0..100;
end;
var st:array[1..n] of student;
    stfile:file of student ;
    sumst:array[1..n] of integer;
    sumsub:array[1..m] of integer;
    sum:integer;
procedure newfile;
var i,j:integer;
begin
assign(stfile,'score.fil');
rewrite(stfile);
for i:=1 to n do
begin
writeln('Input student ',i,' name and ',m,' score');
readln(st[i].name) ;
for j:=1 to m do
read(st[i].score[j]);
readln;
write(stfile,st[i]);
end;
close(stfile);
writeln ;
writeln;
end;
procedure jisuan;
var i,j:integer;
begin
assign(stfile,'score.fil');
```

```

reset(stfile);
for i:=1 to m do sumsub[i]:=0;
for i:=1 to n do
  begin
    read(stfile,st[i]);
    with st[i] do
      begin
        sumst[i]:=0;
        for j:=1 to m do
          begin
            sumst[i]:=sumst[i]+score[j];
            sumsub[j]:=sumsub[j]+score[j];
          end;
        end;
      end;
close(stfile);
sum:=0;
for i:=1 to n do
  sum:=sum+sumst[i];
for i:=1 to n do
  begin
    with st[i] do
      begin
        write(name);
        for j:=1 to m do write(score[j]:6);
        end;
        writeln(sumst[i]:6);
        end;
write('sum=');
for i:=1 to m do
write(sumsub[i]:6);
writeln(sum:8);
end;
begin
  newfile;
  jisuan;
end.

```

练习：

编写程序从磁盘上读取一个由 100 个实数组成的实型数据文件(indata.dat)，以此文件中所有大于平均值的实数建立一个名为“above.dat”的文件，其余的建立一个名为“rest.dat”的文件。

第十一章 指针类型

11.1 指针

11.2 单链表

11.1 指针

指针是通过地址来访问变量的一种特殊的数据类型，属于动态的数据结构，它可以在需要时产生，用完后则又可以取消或回收，以减少占用的内存空间。指针变量与其他类型的变量不同，它占有的不是数据，而是地址。

由于动态数据结构的变量是在程序执行过程中动态生成的，所以不能预先予以说明，无法预先给这些变量起名字，访问时也无法通过名字直接输出或显示，而只能用指针得到其地址，然后间接访问。

1、定义指针类型

在 Turbo Pascal 中，指针变量用来存放某个存储单元的地址，即指针变量指向某个存储单元。一个指针变量仅能

指向某一种类型的存储单元，这种数据类型是在指针类型的定义中确定的，称为指针类型的基类型。指针类型定义如下：

类型名 = ^基类型名；

例如：type q = ^integer；

var a, b, c: q；

说明：q 是一指向整型存储单元的指针类型，其中“^”为指针符。a, b, c 均定义为指针变量，分别可以指向一个整型存储单元。

上例也可用变量说明为：

var a, b, c: ^integer；

指针也可以指向有结构的存储单元。

例如：type person = record

name: string[10]；

sex: (male, female)；

age: 20..70

end；

var pt: ^person；

pt 为指向记录类型 person 的指针变量。

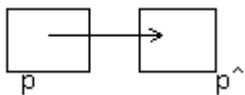
2、动态变量

应用一个指针指向的动态存储单元即动态变量的形式如下：

指针变量名 ^

例如：p^、q^、r^

指针变量 p 和它所指向的动态变量 p^ 之间有如下关系：



以下语句把整数 5 存放到 p 所指向的动态变量 p^ 中去：p^ := 5；

以下语句把 p 所指向的 p^ 中的值赋给整型变量 i：i := p^；

如果指针变量 p 并未指向任何存储单元，则可用下列赋值语句：p := nil；

其中 nil 是保留字，表示“空”，相当于 C 语言里面的 null

3、对动态变量的操作

在 Turbo Pascal 程序中，动态变量不能由 var 直接定义而是通过调用标准过程 new 建立的。过程形式为：

new(指针变量名)；

如果有下列变量定义语句：

var p: ^integer；

仅仅说明了 p 是一个指向整型变量单元的指针变量，但这个整型单元并不存在，在指针变量 p 中还没有具体的地址值。在程序中必须通过过程调用语句：new(p)；才在内存中分配了一个整型变量单元，并把这个单元的地址放在变量 p 中，一个指针变量只能存放一个地址。在同一时间内一个指针只能指向一个变量单元。当程序再次执行 new(p) 时，又在内存中新建立了一个整型变量单元，并把新单元的地址存放在 p 中，从而丢失了旧的变量单元的地址。

为了节省内存空间，对于一些已经不使用的现有动态变量，应该使用标准过程 dispose 予以释放。过程形式为：**dispose(指针变量名)；** 为 new(指针变量名) 的逆过程，其作用是释放由指针变量所指向的动态变量的存储单元。例如在用了 new(p) 后在调用 dispose(p)，则指针 p 所指向的动态变量被撤销，内存空间还给系统，这时 p 的值为 nil。

4、需要注意之处

1、P 与 P^ 的区别

P 是指向该动态变量的指针变量名，P^ 则称为动态变量或标志变量。P 的值是 P^ 的首地址，P^ 的值为与基类型相同的一个值。

2、定义后及时分配存储单元

定义了一个指针变量后，并没有为该指针分配动态存储单元，此时的 P 的值无定义，调用 P^ 则会产生运行错误。若想使该指针可用，可以对指针赋值，也可以通过 NEW（）过程分配存储单元。

3、使用后及时收回存储单元

指针使用后，不会自动归还占用的存储空间，应及时使用 DISPOSE（）过程来释放 P^ 所占用的存储单元，以免浪

费有限的存储空间。

11.2 单链表

单链表的数据类型可定义如下：

```
type dlb=^node;
node=record
    data:datatype;
    next:dlb;
end;
```

例 1 连续输入一序列整数, 组成链表 (并以动态的形式把它们记录下来), 当输入的数为-1 时, 停止输入, 然后把输入的整数按相反的顺序输出。

```
program lianbiao;
type link=^data;
data=record
    num:integer;
    next:link;
end;
var p,q:link;
    i:integer;
begin
    q:=nil;
    readln(i);
    while i<>-1 do
    begin
        new(p);
        with p^ do
        begin
            num:=i;
            next:=q;
        end;
        q:=p;
        readln(i);
    end;
    while p<>nil do
    begin
        write(p^.num:6);
        p:=p^.next;
    end;
    readln;
end.
```

练习: 将例 1 中如果数据不按现反的顺序 (按输入时的顺序) 输出时, 怎样建表. (程序)

上述建表方式其实就是分别从表头和表尾插入元素, 下面是从表中插入元素;

例 2: 输入若干整数 (输入 32767 停止输入) 排序 (小到大) 输出之。

```
program lianbiao;
type link=^data;
data=record
    num:integer;
    next:link;
end;
var head,p,q,r:link;
    i:integer;
begin
    head:=nil;
```

```

readln(i);
while i<>32767 do
begin
  new(p);
  p^.num:=i;
  p^.next:=nil;
  if head=nil then begin head:=p;end
  else
  begin
    q:=head;
    if p^.num<q^.num then begin head:=p;p^.next:=q end else
    begin
      while (p^.num >=q^.num) and (q<>nil) do begin r:=q ;q:=q^.next;end;
      if q=nil then r^.next:=p else begin r^.next:=p;p^.next:=q end
    end;
  end;
  readln(i);
end;
p:=head;
while p<>nil do
begin
  write(p^.num:6);
  p:=p^.next;
end;
readln;
end.

```

练习:建立一个若干整数的链表后(-1 结束) 再从链表中删除从键盘上输入一个整数的所有结点. (程序)

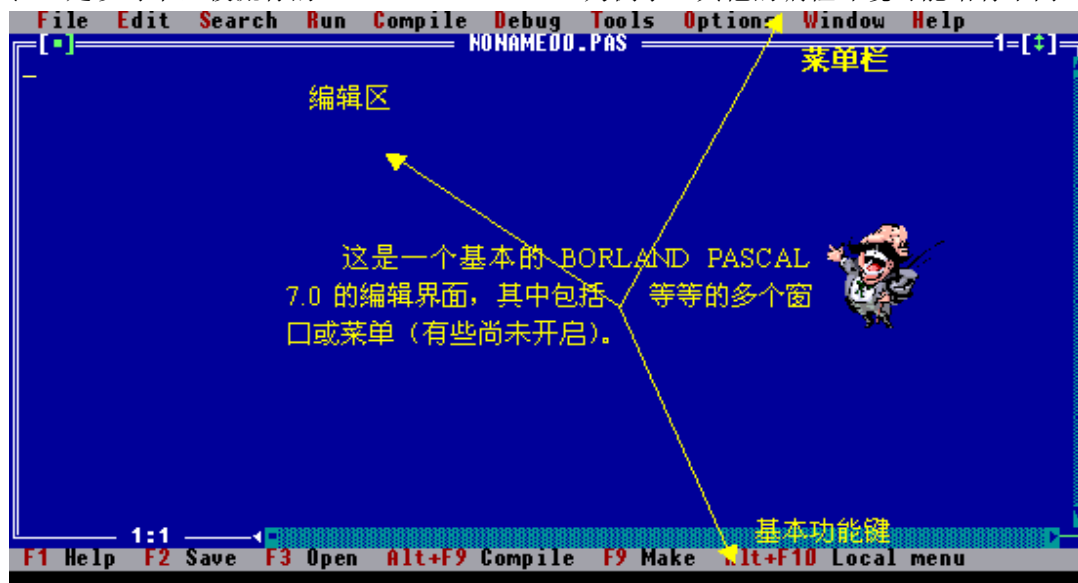
第十二章 程序调试

12.1 单步执行

12.2 断点法

12.1 单步执行

任何一个天才都不敢说，他编的程序是 100%正确的。几乎每一个稍微复杂一点的程序都必须经过反复的调试，修改，最终才完成。所以说，程序的调试是编程中的一项重要技术。我们现在就来掌握一下基本的程序调试。我们以下的示范，是以时下比较流行的 Borland Pascal 7.0 为例子，其他的编程环境可能略有不同，但大致上是一致的。



我们先编一个比较简单的程序，看看程序是如何调试的。

```
program tiaoshi;
```



```

var i:integer;
begin
  for i:=1 to 300 do
  begin
    if i mod 2 = 0 then
      if i mod 3 = 0 then
        if i mod 5 = 0 then
          writeln(i);
        end;
      end;
    end;
  end.

```

该程序是输出 300 以内同时能被 2, 3, 5 整除的整数。现在我们开始调试。调试有多种方法, 先介绍一种, 权且叫步骤法, 步骤法就是模拟计算机的运算, 把程序每一步执行的情况都反映出来。通常, 我们有 F8 即 STEP 这个功能来实现, 如图: 不断地按 F8, 计算机就会一步步地执行程序, 直到执行到最后的 “end.” 为止。



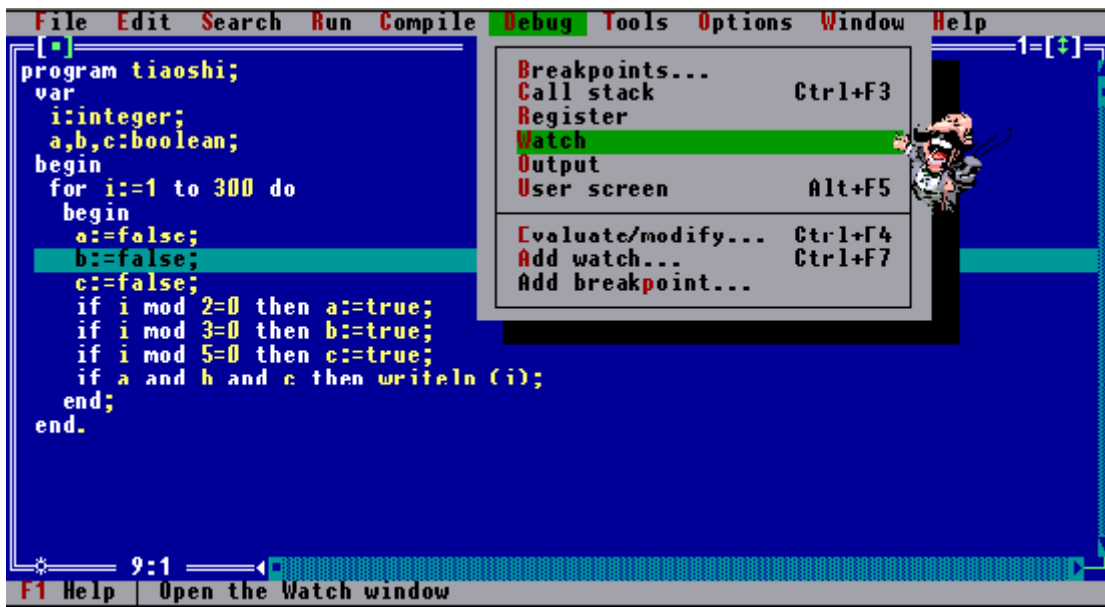
可能你还没有发现 F8 的威力, 我们不妨把上面的程序略微修改一下, 再配合另外的一种调试的利器 watch, 你就会发现步骤法的用处。

```

program tiaoshi;
var i:integer;
    a,b,c:boolean;
begin
  for i:=1 to 300 do
  begin
    a:=false;
    b:=false;
    c:=false;
    if i mod 2 = 0 then a:=true;
    if i mod 3 = 0 then b:=true;
    if i mod 5 = 0 then c:=true;
    if a and b and c then writeln(i);
  end;
end.

```

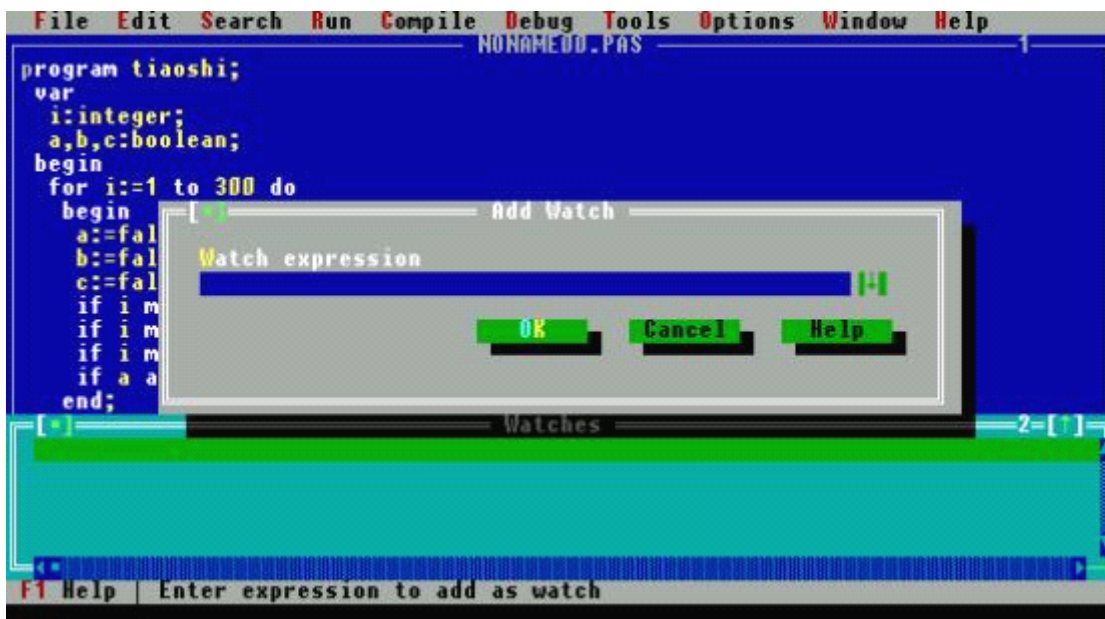
如图, 我们单击菜单栏中 debug 选项, 里面有一项叫 watch 的选项, 我们单击它。



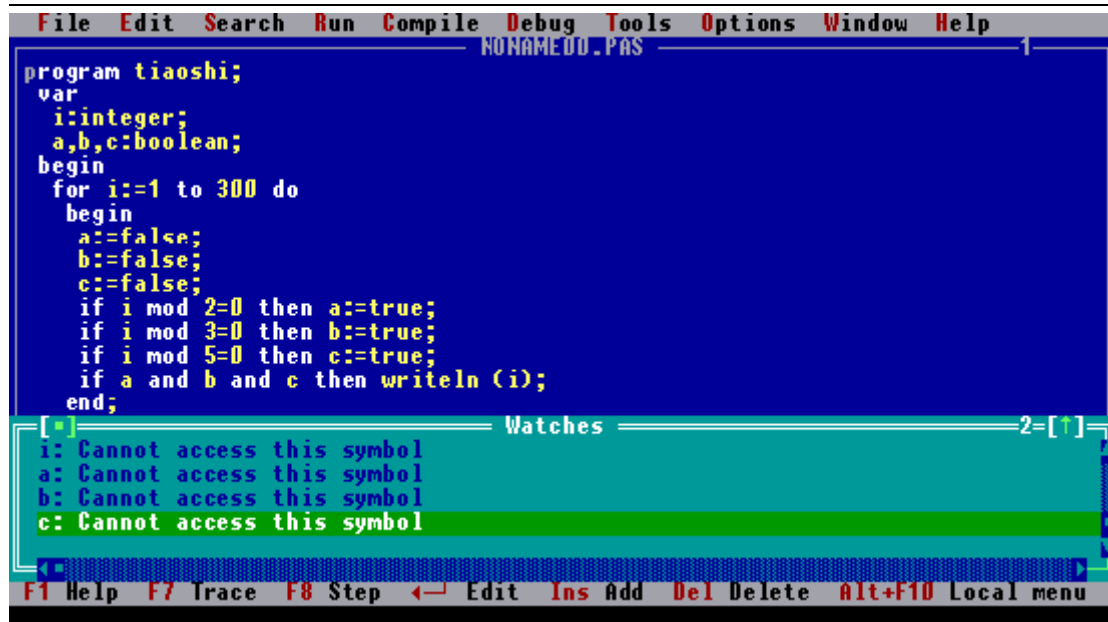
就会出现一个 watch 窗口：



watch 窗口可以让我们观察变量的变化情况，具体操作是在 watches 窗口内按 Insert 键：



这时，屏幕上弹出一个菜单，我们输入所需要观察的变量名，我们分别输入 i, a, b, c 这 4 个变量名，于是 watches 窗口内就有如下的 4 个变量的状态：



这时，我们再次使用步骤法，我们会发现，这 4 个变量的状态随着程序的执行而不断变化，比如：



这样我们就可以方便地知道执行每一步之后，程序的各个变量的变化情况，从中我们可以知道我们的程序是否出错，在哪里出错，方便我们及时地修改。下一次，我们介绍另外一种方法，断点法。

12.2 断点法

在前面我们已经学习了基本的程序调试方法——步骤法。步骤法有一个缺点，就是在遇到循环次数比较多或者语句比较多的时候，用起来比较费时，今天我们来学习一种新的也是常用的调试方法——断点法。

所谓断点法，就是在程序执行到某一行时，计算机自动停止运行，并保留这时各变量的状态，方便我们检查，校对。我们还是以前面求同时能被 2，3，5 整除的 3000 以内的自然数为例，具体操作如下：

我们把光标移动到程序的第 14 行，按下 `ctrl+F8`，这时我们会发现，该行变成红色，这表明该行已经被设置成断点行，当我们每次运行到第 14 行的时候，计算机都会自动停下来供我们调试。



我们必须学以致用，赶快运用刚学的 watch 方法，看看这家伙到底有多厉害。



请记住，计算机是执行到断点行之前的一行，断点行并没有执行，所以这时 `b:=true` 这一句并没有执行。

断点行除了有以上用处之外，还有另外一个重要用处。它方便我们判断某个语句有没有执行或者是不是在正确的时刻执行，因为有时程序由于人为的疏忽，可能在循环或者递归时出现我们无法预料的混乱，这时候通过断点法，我们就能够判断程序是不是依照我们预期的顺序执行。

第三部分 常用算法与策略

第一章 算法

- 1.1 什么是算法
- 1.2 算法的表示方法
- 1.3 算法分析

1.1 什么是算法

算法是程序的精髓，程序设计的实质就是构造解决问题的算法，将其解释为计算机语言。

算法是在有限步骤内求解某一问题所使用的一组定义明确的规则。通俗点说，就是计算机解题的过程。在这个过程中，无论是形成解题思路还是编写程序，都是在实施某种算法。前者是推理实现的算法，后者是操作实现的算法。

一个算法应该具有以下五个重要的特征：

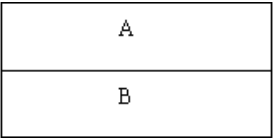
1. **有穷性：** 一个算法必须保证执行有限步之后结束；
2. **确切性：** 算法的每一步骤必须有确切的定义；
3. **输入：** 一个算法有 0 个或多个输入，以刻画运算对象的初始情况；
4. **输出：** 一个算法有一个或多个输出，以反映对输入数据加工后的结果。没有输出的算法是毫无意义的；
5. **可行性：** 算法原则上能够精确地运行，而且人们用笔和纸做有限次运算后即可完成。

1.2 算法的表示方法

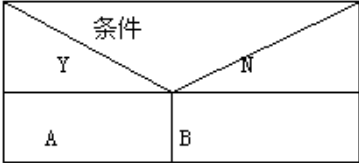
算法通常有三种表示方法：自然语言法、程序流程图法、程序法。

结构化程序设计三种程序结构的流程图（N-S 图）如下：

1. 顺序结构



2. 选择结构



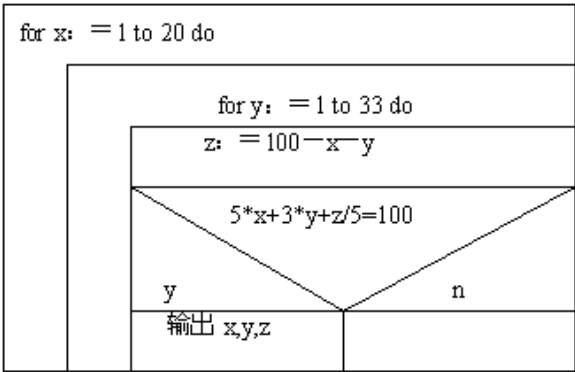
3. 循环结构

当型循环



直到型循环

例题 1: 百钱买百鸡问题:



1.3 算法分析

算法的复杂性

算法的复杂性是算法效率的度量，是评价算法优劣的重要依据。一个算法的复杂性的高低体现在运行该算法所需要的计算机资源的多少上面，所需的资源越多，我们就说该算法的复杂性越高；反之，所需的资源越低，则该算法的复杂性越低。

计算机的资源，最重要的是时间和空间（即存储器）资源。因而，算法的复杂性有*时间复杂性*和*空间复杂性*之分。不言而喻，对于任意给定的问题，设计出复杂性尽可能低的算法是我们在设计算法时追求的一个重要目标；另一方面，当给定的问题已有多种算法时，选择其中复杂性最低者，是我们在选用算法适应遵循的一个重要准则。因此，算法的复杂性分析对算法的设计或选用有着重要的指导意义和实用价值。

简言之，在算法学习过程中，我们必须首先学会对算法的分析，以确定或判断算法的优劣。

1. 时间复杂性:

例 1：设一程序段如下（为讨论方便，每行前加一行号）

- (1) for i:=1 to n do
- (2) for j:=1 to n do
- (3) x:=x+1

.....

试问在程序运行中各步执行的次数各为多少？

解答：

行号 次数(频度)

- (1) n+1
- (2) n*(n+1)
- (3) n*n

可见，这段程序总的执行次数是： $f(n)=2n^2+2n+1$ 。在这里， n 可以表示问题的规模，当 n 趋向无穷大时，如果 $f(n)$ 的值很小，则算法优。作为初学者，我们可以用 $f(n)$ 的数量级 O 来粗略地判断算法的时间复杂性，如上例中的时间复杂性可粗略地表示为 $T(n)=O(n^2)$ 。

2. 空间复杂性：

例 2：将一维数组的数据 (n 个) 逆序存放到原数组中，下面是实现该问题的两种算法：

算法 1: for $i:=1$ to n do

$b[i]:=a[n-i+1];$

for $i:=1$ to n do

$a[i]:=b[i];$

算法 2: for $i:=1$ to $n \div 2$ do

begin

$t:=a[i]; a[i]:=a[n-i+1]; a[n-i+1]:=t$

end;

算法 1 的时间复杂度为 $2n$, 空间复杂度为 $2n$

算法 2 的时间复杂度为 $3*n/2$, 空间复杂度为 $n+1$

显然算法 2 比算法 1 优，这两种算法的空间复杂度可粗略地表示为 $S(n)=O(n)$

信息学比赛中，经常是：只要不超过内存，尽可能用空间换时间。

第二章 递归

2.1 递归的概念

2.2 如何设计递归算法

2.3 典型例题

递归是计算机科学的一个重要概念，递归的方法是程序设计中有效的方法，采用递归编写程序能使程序变得简洁和清晰。

2.1 递归的概念

1. 概念

一个过程(或函数)直接或间接调用自己本身，这种过程(或函数)叫递归过程(或函数)。如：

```
procedure a;
begin
.
a;
.
end;
```

这种方式是直接调用。

又如：

```
procedure b;           procedure c;
begin                 begin
.                     .
c;                   b;
.                     .
end;                 end;
```

这种方式是间接调用。

例 1 计算 $n!$ 可用递归公式如下：

1 当 $n=0$ 时

$fac(n)=\{ \begin{matrix} n*fac(n-1) & \text{当 } n>0 \text{ 时} \end{matrix}$

可编写程序如下：

```
program fac2;
var
n:integer;
function fac(n:integer):real;
```

```

begin
if n=0 then fac:=1 else fac:=n*fac(n-1)
end;
begin
write('n=');readln(n);
writeln('fac(',n,')=',fac(n):6:0);
end.

```

例 2 楼梯有 n 阶台阶,上楼可以一步上 1 阶,也可以一步上 2 阶,编一程序计算共有多少种不同的走法.

设 n 阶台阶的走法数为 $f(n)$

显然有

$$f(n) = \begin{cases} 1 & n=1 \\ 2 & n=2 \\ f(n-1)+f(n-2) & n>2 \end{cases}$$

可编程序如下:

```

program louti;
var n:integer;
function f(x:integer):integer;
begin
if x=1 then f:=1 else
if x=2 then f:=2 else f:=f(x-1)+f(x-2);
end;
begin
write('n=');read(n);
writeln('f(',n,')=',f(n))
end.

```

2.2 如何设计递归算法

1. 确定递归公式
2. 确定边界(终了)条件

练习:

用递归的方法完成下列问题

1. 求数组中的最大数
2. $1+2+3+\dots+n$
3. 求 n 个整数的积
4. 求 n 个整数的平均值
5. 求 n 个自然数的最大公约数与最小公倍数
6. 有一对雌雄兔,每两个月就繁殖雌雄各一对兔子.问 n 个月后共有多少对兔子?
7. 已知:数列 1, 1, 2, 4, 7, 13, 24, 44, ... 求数列的第 n 项.

2.3 典型例题

例 3 梵塔问题

如图:已知有三根针分别用 1, 2, 3 表示,在一号针中从小放 n 个盘子,现要求把所有的盘子从 1 针全部移到 3 针,移动规则是:使用 2 针作为过度针,每次只移动一块盘子,且每根针上不能出现大盘压小盘.找出移动次数最小的方案.

程序如下:

```

program fanta;
var
n:integer;
procedure move(n,a,b,c:integer);
begin
if n=1 then writeln(a,'--->',c)
else begin
move(n-1,a,c,b);

```

```
writeln(a,'--->',c);
move(n-1,b,a,c);
end;
end;
begin
write('Enter n=');
read(n);
move(n,1,2,3);
end.
```

例 4 快速排序

快速排序的思想是：先从数据序列中选一个元素，并将序列中所有比该元素小的元素都放到它的右边或左边，再对左右两边分别用同样的方法处之直到每一个待处理的序列的长度为 1，处理结束。

程序如下：

```
program kspv;
const n=7;
type
arr=array[1..n] of integer;
var
a:arr;
i:integer;
procedure quicksort(var b:arr; s,t:integer);
var i,j,x,t1:integer;
begin
i:=s;j:=t;x:=b[i];
repeat
while (b[j]>=x) and (j>i) do j:=j-1;
if j>i then begin t1:=b[i]; b[i]:=b[j];b[j]:=t1;end;
while (b[i]<=x) and (i<j) do i:=i+1;
if i<j then begin t1:=b[j];b[j]:=b[i];b[i]:=t1; end
until i=j;
b[i]:=x;
i:=i+1;j:=j-1;
if s<j then quicksort(b,s,j);
if i<t then quicksort(b,i,t);
end;
begin
write('input data:');
for i:=1 to n do read(a[i]);
writeln;
quicksort(a,1,n);
write('output data:');
for i:=1 to n do write(a[i]:6);
writeln;
end.
```

练习：

1. 计算 ackerman 函数值：

$$\text{ack}(m, n) = \begin{cases} n+1 & m=0 \\ \text{ack}(m-1, 1) & m < > 0, n=0 \\ \text{ack}(m-1, \text{ack}(m, n-1)) & m < > 0, n < > 0 \end{cases}$$

求 $\text{ack}(5, 4)$

第三章 回溯

3.1 回溯的设计

3.2 回溯算法的递归实现

回溯是按照某种条件往前试探搜索,若前进中遭到失败,则回过头来另择通路继续搜索.

3.1 回溯的设计

1. 用栈保存好前进中的某些状态.

2. 制定好约束条件

例 1 由键盘上输入任意 n 个符号;输出它的全排列.

```
program hh;
const n=4;
var i,k:integer;
    x:array[1..n] of integer;
    st:string[n];
    t:string[n];
procedure input;
var i:integer;
begin
write('Enter string=');readln(st);
t:=st;
end;
function place(k:integer):boolean;
var i:integer;
begin
place:=true;
for i:=1 to k-1 do
if x[i]=x[k] then
begin place:=false; break end ;
end;
end;
procedure print;
var i:integer;
begin
for i:=1 to n do write(t[x[i]]);
writeln;
end;
begin
input;
k:=1;x[k]:=0;
while k>0 do
begin
x[k]:=x[k]+1;
while (x[k]<=n) and (not place(k)) do x[k]:=x[k]+1;
if x[k]>n then k:=k-1
else if k=n then print
else begin k:=k+1;x[k]:=0 end
end ;
end.

```

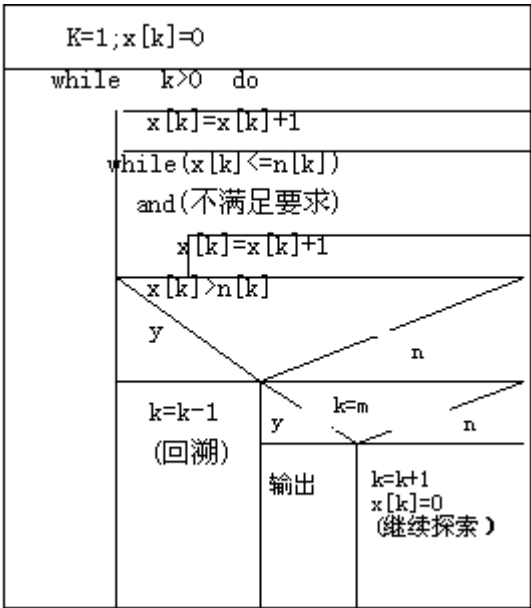
例 2. n 个皇后问题:

```
program hh;
const n=8;
var i,j,k:integer;
    x:array[1..n] of integer;
function place(k:integer):boolean;
var i:integer;

```

```
begin
place:=true;
for i:=1 to k-1 do
  if (x[i]=x[k]) or (abs(x[i]-x[k])=abs(i-k)) then
    place:=false ;
end;
procedure print;
var i:integer;
begin
  for i:=1 to n do write(x[i]:4);
  writeln;
end;
begin
  k:=1;x[k]:=0;
  while k>0 do
    begin
      x[k]:=x[k]+1;
      while (x[k]<=n) and (not place(k)) do x[k]:=x[k]+1;
      if x[k]>n then k:=k-1
        else if k=n then print
          else begin k:=k+1;x[k]:=0 end
      end ;
    end.
end.
```

回溯算法的公式如下：



3.2 回溯算法的递归实现

由于回溯算法用一栈数组实现的, 用到栈一般可用递归实现。

上述例 1 的递归方法实现如下：

```
program hh;
const n=4;
var i,k:integer;
    x:array[1..n] of integer;
    st:string[n];
    t:string[n];
procedure input;
var i:integer;
begin
write('Enter string=');readln(st);
```

```

t:=st;
end;
function place(k:integer):boolean;
  var i:integer;
  begin
    place:=true;
    for i:=1 to k-1 do
      if x[i]=x[k] then
        begin place:=false; break end ;
      end;
    end;
  procedure print;
  var i:integer;
  begin
    for i:=1 to n do write(t[x[i]]);
    writeln;readln;
  end;
  procedure try(k:integer);
  var i :integer;
  begin
    if k=n+1 then begin print;exit end;
    for i:=1 to n do
      begin
        x[k]:=i;
        if place(k) then try(k+1)
      end
    end;
  end;
begin
  input;
  try(1);
end.

```

例 2: n 皇后问题的递归算法如下:

程序 1:

```

program hh;
const n=8;
var i,j,k:integer;
    x:array[1..n] of integer;
function place(k:integer):boolean;
  var i:integer;
  begin
    place:=true;
    for i:=1 to k-1 do
      if (x[i]=x[k]) or (abs(x[i]-x[k])=abs(i-k)) then
        place:=false ;
      end;
    end;
  procedure print;
  var i:integer;
  begin
    for i:=1 to n do write(x[i]:4);
    writeln;
  end;
  procedure try(k:integer);
  var i:integer;

```

```

begin
  if k=n+1 then begin print; exit end;
  for i:= 1 to n do
    begin
      x[k]:=i;
      if place(k) then try(k+1);
    end;
  end ;
begin
  try(1);
end.

```

程序 2:

说明:当 $n=8$ 时有 30 条对角线分别用了 l 和 r 数组控制,

用 c 数组控制列. 当 (i, j) 点放好皇后后相应的对角线和列都为 `false`. 递归程序如下:

```

program nhh;
const n=8;
var s,i:integer;
    a:array[1..n] of byte;
    c:array[1..n] of boolean;
    l:array[1-n..n-1] of boolean;
    r:array[2..2*n] of boolean;
procedure output;
var i:integer;
begin
  for i:=1 to n do write(a[i]:4);
  inc(s);writeln(' total=',s);
end;
procedure try(i:integer);
  var j:integer;
begin
  for j:=1 to n do
    begin
      if c[j] and l[i-j] and r[i+j] then
        begin
          a[i]:=j;c[j]:=false;l[i-j]:=false; r[i+j]:=false;
          if i<n then try(i+1) else output;
          c[j]:=true;l[i-j]:=true;r[i+j]:=true;
        end;
    end;
  end;
begin
  for i:=1 to n do c[i]:=true;
  for i:=1-n to n-1 do l[i]:=true;
  for i:=2 to 2*n do r[i]:=true;
  s:=0;try(1);
  writeln;
end.

```

练习:

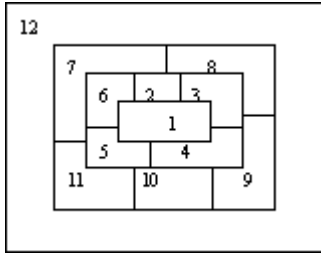
1. 找出所有从 m 个元素中选取 n ($n \leq m$) 元素的组合。
2. 设有 A, B, C, D, E 5 人从事 j_1, j_2, j_3, j_4, j_5 5 项工作每人只能从事一项, 它们的效益表如下:

| | j1 | j2 | j3 | j4 | j5 |
|---|----|----|----|----|----|
| A | 13 | 11 | 10 | 4 | 7 |
| B | 13 | 10 | 10 | 8 | 5 |
| C | 5 | 9 | 7 | 7 | 4 |
| D | 15 | 12 | 10 | 11 | 5 |
| E | 10 | 11 | 8 | 8 | 4 |

求最佳安排,使效益最高.

3. N 个数中找出 M 个数(从键盘上输入正整数 N, M 后再输入 N 个正数),要求从 N 个数中找出若干个数,使它们的和为 M ,把满足条件的数组找出来,并统计组数.

4. 地图着色。如下图 12 个区域用 4 种颜色着色要求相邻的区域着不同的颜色



5. 将任意一正整数 ($1 < n < 100$) 分解成若干正整数的和.

如: $4 = 1 + 1 + 1 + 1$

$= 2 + 1 + 1$

$= 2 + 2$

$= 3 + 1$.

第四章 排序

4.1 简单排序

4.2 快速排序

4.3 希尔排序

4.4 堆排序与二叉树排序

4.5 归并排序

4.6 线形时间排序

4.7 各种算法排序的比较

排序就是将杂乱无章的数据元素,通过一定的方法按关键字顺序排列的过程。

4.1 简单排序

1. 选择排序

选择排序的基本思想是:

对待排序的记录序列进行 $n-1$ 遍的处理,第 1 遍处理是将 $L[1..n]$ 中最小者与 $L[1]$ 交换位置,第 2 遍处理是将 $L[2..n]$ 中最小者与 $L[2]$ 交换位置,……,第 i 遍处理是将 $L[i..n]$ 中最小者与 $L[i]$ 交换位置。这样,经过 i 遍处理之后,前 i 个记录的位置就已经按从小到大的顺序排列好了。

例 1:输入序列数据按非减顺序输出。

程序如下:

```
program xzpx;
const n=7;
var a:array[1..n] of integer;
    i,j,k,t:integer;
begin
  write('Enter date:');
  for i:= 1 to n do read(a[i]);
  writeln;
```

```

for i:=1 to n-1 do
begin
  k:=i;
  for j:=i+1 to n do
    if a[j]<a[k] then k:=j;
  if k<>i then
    begin t:=a[i];a[i]:=a[k];a[k]:=t;end;
end;
write('output data:');
for i:= 1 to n do write(a[i]:6);
writeln;
end.

```

2. 插入排序

插入排序的基本思想:经过 $i-1$ 遍处理后, $L[1..i-1]$ 已排好序。第 i 遍处理仅将 $L[i]$ 插入 $L[1..i-1]$ 的适当位置 p , 原来 p 后的元素一一向右移动一个位置, 使得 $L[1..i]$ 又是排好序的序列。

例 2: 输入序列数据按非减顺序输出。

程序 1:

```

program crpx;
const n=7;
var a:array[1..n] of integer;
    i,j,k,t:integer;
begin
  write('Enter date:');
  for i:= 1 to n do read(a[i]);
  writeln;
  for i:=2 to n do
    begin
      k:=a[i];j:=i-1;
      while (k<a[j]) and (j>0) do
        begin a[j+1]:=a[j];j:=j-1 end;
      a[j+1]:=k;
    end;
  write('output data:');
  for i:= 1 to n do write(a[i]:6);
  writeln;
end.

```

3. 冒泡排序

冒泡排序又称交换排序其基本思想是:对待排序的记录的关键字进行两两比较, 如发现两个记录是反序的, 则进行交换, 直到无反序的记录为止。

例: 输入序列数据按非减顺序输出。

程序 1:

```

program mppx;
const n=7;
var a:array[1..n] of integer;
    i,j,k,t:integer;
begin
  write('Enter date:');
  for i:= 1 to n do read(a[i]);
  for i:=1 to n -1 do
    for j:=n downto i+1 do
      if a[j-1]<a[j] then
        begin t:=a[j-1];a[j-1]:=a[j];a[j]:=t end;

```

```

    write('output data:');
    for i:= 1 to n do write(a[i]:6);
    writeln;
end.

```

程序 2:

```

program mppx;
const n=7;
var a:array[1..n] of integer;
    i,j,k,t:integer;
    bool:boolean;
begin
    write('Enter date:');
    for i:= 1 to n do read(a[i]);
    i:=1;bool:=true;
    while (i<n) and bool do
        begin
            bool:=false;
            for j:=n downto i+1 do
                if a[j-1]<a[j] then
                    begin t:=a[j-1];a[j-1]:=a[j];a[j]:=t;bool:=true end;
            i:=i+1;
        end;
    write('output data:');
    for i:= 1 to n do write(a[i]:6);
    writeln;
end.

```

程序 3:

```

program mppx;
const n=7;
var a:array[1..n] of integer;
    i,j,k,t:integer;
begin
    write('Enter date:');
    for i:= 1 to n do read(a[i]);
    writeln;
    k:=n;
    while k>0 do
        begin
            j:=k-1;k:=0;
            for i:=1 to j do
                if a[i]>a[i+1] then
                    begin t:=a[i];a[i]:=a[i+1];a[i+1]:=t;k:=i;end;
            end;
        write('output data:');
        for i:= 1 to n do write(a[i]:6);
        writeln;
    end.

```

4.2 快速排序

快速排序的思想是:先从数据序列中选一个元素,并将序列中所有比该元素小的元素都放到它的右边或左边,再对左右两边分别用同样的方法处之直到每一个待处理的序列的长度为 1, 处理结束.

例:输入一组数据小到大批序.

程序 1:

```

program kspv;
const n=7;
type
arr=array[1..n] of integer;
var
a:arr;
i:integer;
procedure quicksort(var b:arr; s,t:integer);
var i,j,x,t1:integer;
begin
  i:=s;j:=t;x:=b[i];
  repeat
    while (b[j]>=x) and (j>i) do j:=j-1;
    if j>i then begin t1:=b[i]; b[i]:=b[j];b[j]:=t1;end;
    while (b[i]<=x) and (i<j) do i:=i+1;
    if i<j then begin t1:=b[j];b[j]:=b[i];b[i]:=t1; end
  until i=j;
  b[i]:=x;
  i:=i+1;j:=j-1;
  if s<j then quicksort(b,s,j);
  if i<t then quicksort(b,i,t);
end;
begin
write('input data:');
for i:=1 to n do read(a[i]);
writeln;
quicksort(a,1,n);
write('output data:');
for i:=1 to n do write(a[i]:6);
writeln;
end.

```

程序 2:

```

program kspv;
const n=7;
type
arr=array[1..n] of integer;
var
a:arr;
i:integer;
procedure quicksort(var b:arr; s,t:integer);
var i,j,x:integer;
begin
  i:=s;j:=t;x:=b[i];
  repeat
    while (b[j]>=x) and (j>i) do j:=j-1;
    if j>i then begin b[i]:=b[j];i:=i+1;end;
    while (b[i]<=x) and (i<j) do i:=i+1;
    if i<j then begin b[j]:=b[i];j:=j-1; end
  until i=j;
  b[i]:=x;
  i:=i+1;j:=j-1;
  if s<j then quicksort(b,s,j);

```



```

    if i<t then quicksort(b,i,t);
  end;
begin
write('input data:');
for i:=1 to n do read(a[i]);
writeln;
quicksort(a,1,n);
write('output data:');
for i:=1 to n do write(a[i]:6);
writeln;
end.

```

4.3 希尔排序

基本思想：将整个无序序列分割成若干小的子序列分别进行插入排序或冒泡排序。

序列分割方法：将相隔某个增量 h 的元素构成一个子序列。在排序过程中，逐次减小这个增量，最后当 h 减到 1 时，进行一次插入排序或冒泡排序，排序就完成。增量序列一般采用： $d_i = n \text{ div } 2$, $d_i = d_{i-1} \text{ div } 2$; $i=2, 3, 4, \dots$ 其中 n 为待排序序列的长度。

程序 1: (子序列是插入排序)

```

program xepx;
const n=7;
type
arr=array[1..n] of integer;
var
a:arr;
i,j,t,d:integer;
bool:boolean;
begin
write('input data:');
for i:=1 to n do read(a[i]);
writeln;
d:=n;
while d>1 do
begin
d:=d div 2;
for j:=d+1 to n do
begin
t:=a[j];i:=j-d;
while (i>0) and (a[i]>t) do
begin a[i+d]:=a[i];i:=i-d;end;
a[i+d]:=t;
end;
end;
write('output data:');
for i:=1 to n do write(a[i]:6);
writeln;
end.

```

程序 2: (子序列是冒泡排序)

```

program xepx;
const n=7;
type
arr=array[1..n] of integer;
var
a:arr;

```

```

i,temp,d:integer;
bool:boolean;
begin
write('input data:');
for i:=1 to n do read(a[i]);
writeln;
d:=n
while d>1 do
begin
d:=d div 2;
repeat
bool:=true;
for i:=1 to n-d do
if a[i]>a[i+d] then
begin temp:=a[i];a[i]:=a[i+d];a[i+d]:=temp; bool:=false end;
until bool;
end;
write('output data:');
for i:=1 to n do write(a[i]:6);
writeln;
end.

```

4.4 堆排序与二叉树排序

1. 堆排序

堆：设有数据元素的集合 $(R_1, R_2, R_3, \dots, R_n)$ 它们是一棵顺序二叉树的结点且有

$$R_i \leq R_{2i} \text{ 和 } R_i \leq R_{2i+1} (\text{或} \geq)$$

堆的性质：堆的根结点上的元素是堆中的最小元素，且堆的每一条路径上的元素都是有序的。

堆排序的思想是：

1) 建初始堆（将结点 $[n/2], [n/2]-1, \dots, 3, 2, 1$ 分别调成堆）

2) 当未排序完时

输出堆顶元素，删除堆顶元素，将剩余的元素重新建堆。

程序如下：

```

program duipx;
const n=8;
type arr=array[1..n] of integer;
var a:arr;i:integer;
procedure sift(var a:arr;l,m:integer);
var i,j, t:integer;
begin
i:=l;j:=2*i;t:=a[i];
while j<=m do
begin
if (j<m) and (a[j]>a[j+1]) then j:=j+1;
if t>a[j] then
begin a[i]:=a[j];i:=j;j:=2*i; end
else exit;
a[i]:=t;
end;
end;
begin
for i:=1 to n do read(a[i]);
for i:=(n div 2) downto 1 do
sift(a,i,n);

```

```

for i:=n downto 2 do
begin
write(a[1]:4);
a[1]:=a[i];
sift(a,1,i-1);
end;
writeln(a[1]:4);
end.

```

2. 二叉树排序

排序二叉树：每一个参加排列的数据对应二叉树的一个结点，且任一结点如果有左（右）子树，则左（右）子树各结点的数据必须小（大）于该结点的数据。中序遍历排序二叉树即得排序结果。程序如下：

```

program pxtree;
const
a:array[1..8] of integer=(10,18,3,8,12,2,7,3);
type point=^nod;
nod=record
w:integer;
right,left:point ;
end;
var root,first:point;k:boolean;i:integer;
procedure hyt(d:integer;var p:point);
begin
if p=nil then
begin
new(p);
with p^ do begin w:=d;right:=nil;left:=nil end;
if k then begin root:=p; k:=false end;
end
else with p^ do if d>=w then hyt(d,right) else hyt(d,left);
end;
procedure hyt1(p:point);
begin
with p^ do
begin
if left<>nil then hyt1(left);
write(w:4);
if right<>nil then hyt1(right);
end
end;
begin
first:=nil;k:=true;
for i:=1 to 8 do hyt(a[i],first);
hyt1(root);writeln;
end.

```

4.5 归并排序

归并就是将多个有序的数列合成一个有序的数列。将两个有序序列合并为一个有序序列叫二路归并(merge)。归并排序就是 n 长度为 1 的子序列，两两归并最后变为有序的序列。

1. 二路归并

例 1:将有序表 L1=(1, 5, 7), L2=(2, 3, 4, 6, 8, 9, 10) 合并一个有序表 L.

```

program gb;
const m=3;n=7;
type arr11=array[1..m] of integer;

```

```

arrl2=array[1..n] of integer;
arrl=array[1..m+n] of integer;
var a:arrl1;
b:arrl2;
c:arrl;
i,j,k,r:integer;
begin
write('Enter l1(sorted):');
for i:=1 to m do read(a[i]);
write('Enter l2(sorted):');
for i:=1 to n do read(b[i]);
i:=1;j:=1;k:=0;
while (i<=m) and (j<=n) do
begin
  k:=k+1;
  if a[i]<=b[j] then begin c[k]:=a[i];i:=i+1; end
                    else begin c[k]:=b[j];j:=j+1;end;
end;
if i<=m then for r:=i to m do c[m+r]:=a[r];
if j<=n then  for r:=j to n do c[n+r]:=b[r];
writeln('Output data:');
for i:=1 to m+n do write(c[i]:5);
writeln;
end.

```

2. 归并排序

```

program gbpx;
const maxn=7;
type arr=array[1..maxn] of integer;
var a,b,c:arr;
    i:integer;
procedure merge(r:arr;l,m,n:integer;var r2:arr);
var i,j,k,p:integer;
begin
  i:=1;j:=m+1;k:=l-1;
  while (i<=m)and (j<=n) do
  begin
    k:=k+1;
    if r[i]<=r[j] then begin r2[k]:=r[i];i:=i+1 end
                    else begin r2[k]:=r[j];j:=j+1 end
  end;
  if i<=m then
    for p:=i to m do begin k:=k+1;r2[k]:=r[p] end;
  if j<=n then
    for p:=j to n do begin k:=k+1;r2[k]:=r[p] end;
end;
procedure mergesort( var r,r1:arr;s,t:integer);
var k:integer; c:arr;
begin
  if s=t then r1[s]:=r[s] else
  begin
    k:=(s+t) div 2;
    mergesort(r,c,s,k);

```

```

    mergesort(r,c,k+1,t);
    merge(c,s,k,t,r1)
end;
end;
begin
    write('Enter data:');
    for i:= 1 to maxn do
        read(a[i]);
    mergesort(a,b,1,maxn);
    for i:=1 to maxn do
        write(b[i]:9);
    writeln;
end.

```

4.6 线形排序

以上我们讨论的算法均是基于比较的排序算法，在排序算法中有基于数字本身的算法：计数、桶、基数排序。

1. 计数排序

基本思想是对于序列中的每一元素 x ，确定序列中小于 x 的元素的个数。

例： n 个整数序列且每个值在 $[1, m]$ ，排序之。

```

program jspx;
const m=6;n=8;
var i,j:integer;
    a,b:array[1..n] of integer;
    c:array[1..m] of integer;
begin
    writeln('Enter data:');
    for i:=1 to n do read(a[i]);
    for i:=1 to m do c[i]:=0;
    for i:=1 to n do c[a[i]]:=c[a[i]]+1;
    for i:=2 to m do c[i]:=c[i]+c[i-1];
    for i:=n downto 1 do
        begin
            b[c[a[i]]]:=a[i];
            c[a[i]]:=c[a[i]]-1;
        end;
    writeln('Sorted data:');
    for i:= 1 to n do
        write(b[i]:6);
    end.

```

2. 桶排序

桶排序的思想是若待排序的记录的关键字在一个明显有限范围内(整型)时,可设计有限个有序桶,每个桶装入一个值,顺序输出各桶的值,将得到有序的序列。

例：输入 n 个 0 到 100 之间的整数，由小到大排序输出。

```

program tpx;
const n=7;
var b:array[0..100] of integer;
    k:0..100;
    i:integer;
begin
    write('Enter date:(0-100)');
    for i:=0 to 100 do b[i]:=0;
    for i:= 1 to n do
        begin

```

```

read(k);
b[k]:=b[k]+1;
end;
writeln('Output data:');
for i:=0 to 100 do
  while b[i]>0 do begin write(i:6);b[i]:=b[i]-1 end;
writeln;
end.

```

3. 基数排序

基本思想是对 n 个元素依次按 $k, k-1, \dots, 1$ 位上的数字进行桶排序。

```

program jsp;
const n=8;
type link=^node;
      node=record
        data:integer;
        next:link;
      end;
var i,j,l,m,k:integer;
    a:array[1..n] of integer;
    s:string;
    q,head:array[0..9] of link;
    p,p1:link;
begin
  writeln('Enter data:');
  for i:=1 to n do read(a[i]);
  for i:=5 downto 1 do
    begin
      for j:=0 to 9 do
        begin
          new(head[j]);
          head[j].next:=nil;
          q[j]:=head[j]
        end;
      for j:=1 to n do
        begin
          str(a[j],s);
          for k:=1 to 5-length(s) do
            s:='0'+ s;
          m:=ord(s[i])-48;
          new(p);
          p^.data:=a[j];
          p^.next:=nil;
          q[m].next:=p;
          q[m]:=p;
        end;
      l:=0;
      for j:=0 to 9 do
        begin
          p:=head[j];
          while p^.next<>nil do
            begin
              l:=l+1;p1:=p;p:=p^.next;dispose(p1);a[l]:=p^.data;

```

```

end;
end;
end;
writeln('Sorted data:');
for i:= 1 to n do
  write(a[i]:6);
end.

```

4.7 各种排序算法的比较

1. 稳定性比较

插入排序、冒泡排序、二叉树排序、二路归并排序及其他线形排序是稳定的

选择排序、希尔排序、快速排序、堆排序是不稳定的

2. 时间复杂性比较

插入排序、冒泡排序、选择排序的时间复杂性为 $O(n^2)$

其它非线性排序的时间复杂性为 $O(n \log_2 n)$

线形排序的时间复杂性为 $O(n)$;

3. 辅助空间的比较

线形排序、二路归并排序的辅助空间为 $O(n)$, 其它排序的辅助空间为 $O(1)$;

4. 其它比较

插入、冒泡排序的速度较慢, 但参加排序的序列局部或整体有序时, 这种排序能达到较快的速度。

反而在这种情况下, 快速排序反而慢了。

当 n 较小时, 对稳定性不作要求时宜用选择排序, 对稳定性有要求时宜用插入或冒泡排序。

若待排序的记录的关键字在一个明显有限范围内时, 且空间允许是用桶排序。

当 n 较大时, 关键字元素比较随机, 对稳定性没要求宜用快速排序。

当 n 较大时, 关键字元素可能出现本身是有序的, 对稳定性有要求时, 空间允许的情况下。

宜用归并排序。

当 n 较大时, 关键字元素可能出现本身是有序的, 对稳定性没有要求时宜用堆排序。

第五章 查找(检索)

5.1 顺序查找

5.2 二分查找

5.3 二叉排序树查找

5.4 哈希(Hash)表

5.5 查找第 k 小元素

查找 是在数据结构中查找指定值的结点。

5.1 顺序查找

1. 顺序查找的思想是:

将查找值顺序逐个与结点值进行比较, 相等即为查找成功, 否则查找失败。

程序如下:

```

program sxcz;
const n=7;
type
arr=array[1..n] of integer;
var x1,i:integer;
a:arr;
b:boolean;
place:integer;
procedure search(r:arr;m,x:integer; var found:boolean;var p:integer);
begin
p:=1;found:=false;
while(p<=m) and not found do
if r[p]=x then found:=true else p:=p+1;

```

```

end;
begin
write('Enter array:');
for i:=1 to n do read(a[i]);
writeln;
write('Enter search data:');
read(x1);
search(a,n,x1,b,place);
if b then begin writeln('yes');writeln('Place of',x1:5,'is:',place); end
  else writeln('no');
end.

```

5.2 二分查找

1. 二分查找的基本思想: 首先将结点按关键字排序, 其次将查找值与中间位置的值比较, 相等, 查找成功; 不等, 则中间数据大于或小于查找值, 无论怎样查找将在一半的数据中查找。

2. 例: 输入序列数据查找指定值。

程序:

```

program sxcz;
const n=7;
type
arr=array[1..n] of integer;
var x1,i:integer;
a:arr;
place:integer;
procedure paixv(var r:arr;m:integer);
var k,j,i,t:integer;
begin
k:=m;
while k>0 do
begin
j:=k-1;k:=0;
for i:=1 to j do
if r[i]>r[i+1] then
begin t:=r[i];a[i]:=r[i+1];r[i+1]:=t;k:=i;end;
end;
end;
procedure search(r:arr;m,x:integer; var p:integer);
var low,high,mid:integer;
begin
p:=0;low:=1;high:=m;
while low<=high do
begin
mid:=(low+high) div 2;
if x>r[mid] then low:=mid+1 else
if x<r[mid] then high:=mid-1 else
begin p:=mid;exit;end;
end;
end;
begin
write('Enter array:');
for i:=1 to n do read(a[i]);
writeln;
write('Enter search data:');

```



```
read(x1);
paixv(a,n);
search(a,n,x1,place);
if place<>0 then writeln('yes') else writeln('no');
end.
```

5.3 二叉排序树查找

因为二叉排序树的左子树若不为空则左子树的所有结点的值均小于它的根结点的值, 而右子树若不为空, 则右子树的所有结点的值均不小大于它的根结点的值, 根据这个性质查找算法如下:

```
program pxtree;
const
  a:array[1..8] of integer=(10,18,3,8,12,2,7,3);
type point=^nod;
   nod=record
     w:integer;
     right,left:point ;
   end;
var root,first:point;k:boolean;i,x:integer;
procedure maketr(d:integer;var p:point);
begin
  if p=nil then
    begin
      new(p);
      with p^ do begin w:=d;right:=nil;left:=nil end;
      if k then begin root:=p; k:=false end;
    end
  else with p^ do if d>=w then maketr(d,right) else maketr(d,left);
end;
function searchtr(x:integer;p:point):boolean;
begin
  if p=nil then searchtr:=false
    else if x=p^.w then searchtr:=true
      else if x<p^.w then searchtr:=searchtr(x,p^.left)
        else searchtr:=searchtr(x,p^.right);
end;
begin
  first:=nil;k:=true;
  for i:=1 to 8 do maketr(a[i],first);
  write('want find data x:');read(x);
  if searchtr(x,first) then writeln('yes') else writeln('No');
end.
```

5.4 哈希(Hash)表

以上讲的查找方法基于比较的, 查找效率依赖比较次数, 其实理想的查找希望不经比较, 一次存取便能得到所查记录, 那就必须在记录的存储位置和它的关键字之间建立一个确定的对应关系 f , 这样查找 k 时, 只要根据这个对应关系 f 找到给定值 k 的像 $f(k)$ 。这种对应关系 f 叫哈希 (hash) 函数。按这种思想建立的表叫哈希表 (也叫散列表)。哈希表存取方便但存储时容易冲突 (collision): 即不同的关键字可以对应同一哈希地址。如何确定哈希函数和解决冲突是关键。

1. 哈希函数的构造方法

直接定址法: $H(k)=k$ 或 $H(k)=a*k+b$ (线形函数)

如: 人口数字统计表

| | | | | | |
|----|----|------|-----|-----|-----|
| 地址 | 1 | 2 | 3 | ... | 100 |
| 年龄 | 1 | 2 | 3 | ... | 100 |
| 人数 | 67 | 3533 | 244 | ... | 4 |

数字分析法:取关键字的若干数位组成哈希地址

如:关键字如下:若哈希表长为 100 则可取中间两位 10 进制数作为哈希地址。

81346532 81372242 81387422 81301367 81322817 81338967 81354157 81368537

平方取中法:关键字平方后取中间几位数组成哈希地址

折叠法:将关键字分割成位数相同的几部分(最后一部分的位数可以不同)然后取几部分的叠加和(舍去进位)作为哈希地址。

除留余数法:取关键字被某个不大于表长 m 的数 p 除后所得的余数为哈希地址。

$$H(k)=k \bmod p \quad p \leq m$$

随机数法: $H(k)=\text{rondom}(k)$ 。

2. 处理冲突的方法

假设地址集为 $0..n-1$, 由关键字得到的哈希地址为 j ($0 \leq j \leq n-1$) 的位置已存有记录, 处理冲突就是为该关键字的记录找到另一个“空”的哈希地址。在处理中可能得到一个地址序列 $H_i \quad i=1, 2, \dots, k$

$0 \leq H_i \leq n-1$, 即在处理冲突时若得到的另一个哈希地址 H_1 仍发生冲突, 再求下一地址 H_2 , 若仍冲突, 再求 $H_3 \dots$ 。怎样得到 H_i 呢?

开放定址法: $H_i=(H(k)+d_i) \bmod m$ ($H(k)$ 为哈希函数; m 为哈希表长; d_i 为增量序列)

当 $d_i=1, 2, 3, \dots, m-1$ 时叫线性探测再散列。

当 $d_i=1^2, -1^2, 2^2, -2^2, 3^2, -3^2, \dots, k^2, -k^2$ 时叫二次探测再散列。

当 $d_i=\text{random}(m)$ 时叫伪随机探测序列。

例:长度为 11 的哈希表关键字分别为 17, 60, 29, 哈希函数为 $H(k)=k \bmod 11$, 第四个记录的关键字为 38, 分别按上述方法添入哈希表的地址为 8, 4, 3 (随机数=9)。

再哈希法: $H_i=RH_i(\text{key}) \quad i=1, 2, \dots, k$, 其中 RH_i 均为不同的哈希函数。

链地址法:这种方法很象基数排序, 相同的地址的关键字值均链入对应的链表中。

建立公益区法:另设一个溢出表, 不管得到的哈希地址如何, 一旦发生冲突, 都填入溢出表。

3. 哈希表的查找

例:如下一组关键字按哈希函数 $H(k)=k \bmod 13$ 和线性探测处理冲突所得的哈希表 $a[0..15]$:

| | | | | | | | | | | | | | | | |
|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| | 14 | 01 | 68 | 27 | 55 | 19 | 20 | 84 | 79 | 23 | 11 | 10 | | | |

当给定值 $k=84$, 则首先和 $a[6]$ 比在依次和 $a[7]$, $a[8]$ 比结果 $a[8]=84$ 查找成功。

当给定值 $k=38$, 则首先和 $a[12]$ 比, 再和 $a[13]$ 比, 由于 $a[13]$ 没有, 查找不成功, 表中不存在关键字等于 38 的记录。

5.5 查找第 k 小元素

查找第 k 小元素即在 n 个元素中(未排序)找到第 k 小的元素。方法同快速排序, 采用递归方式。

程序如下:

```

program kspv;
const n=7;
type
arr=array[1..n] of integer;
var
b:arr;
i,k:integer;
function p(s,t:integer):integer;
var i,j,t1,x:integer;
begin
  i:=s;j:=t;x:=b[i];
  repeat
    while (b[j]>=x) and (j>i) do j:=j-1;
    if j>i then begin t1:=b[i]; b[i]:=b[j];b[j]:=t1;end;
    while (b[i]<=x) and (i<j) do i:=i+1;
    if i<j then begin t1:=b[j];b[j]:=b[i];b[i]:=t1; end
  until i=j;

```

```

b[i]:=x;
p:=i;
end;
function find(s,t,k:integer):integer;
var p1,q:integer;
begin
  if s=t then find:=b[s] else
  begin
    p1:=p(s,t);
    q:=p1-s+1;
    if k<=q then find:=find(s,p1,k) else find:=find(p1+1,t,k-q);
  end;
end;
begin
  write('input data:');
  for i:=1 to n do read(b[i]);readln;
  write('input k:');read(k);
  write('output data:');
  writeln('kthsmall: ',find(1,n,k));
end.

```

第六章 穷举（枚举）策略

6.1 穷举策略的概念

6.2 典型例题与习题

6.1 穷举策略的概念

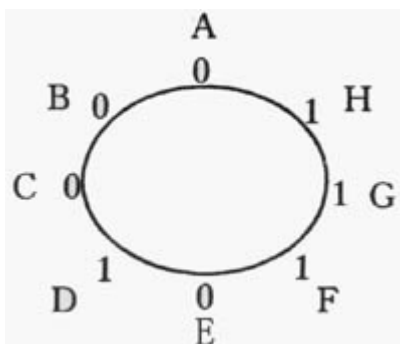
所谓枚举法，指的是从可能的解的集合中一一枚举各元素，用题目给定的检验条件判定哪些是无用的，哪些是有用的。能使命题成立，即为其解。

有些问题可以用循环语句和条件语句直接求解，有些问题用循环求解时循环次数太多，无法编写程序，怎么办？下面是用“千军万马过独木桥，适者存”的方式实现穷举策略的。

6.2 典型例题与习题

例 1. 将 2^n 个 0 和 2^n 个 1，排成一圈。从任一个位置开始，每次按逆时针的方向以长度为 $n+1$ 的单位进行数二进制数。要求给出一种排法，用上面的方法产生出来的 2^{n+1} 个二进制数都不相同。

例如，当 $n=2$ 时，即 2^2 个 0 和 2^2 个 1 排成如下一圈：



比如，从 A 位置开始，逆时针方向取三个数 000，然后再从 B 位置上开始取三个数 001，接着从 C 开始取三个数 010，... 可以得到 000, 001, 010, 101, 011, 111, 110, 100 共 8 个二进制数且都不相同。

程序说明：

以 $n=4$ 为例，即有 16 个 0，16 个 1，数组 a 用以记录 32 个 0，1 的排法，数组 b 统计二进制数出现的可能性。

程序清单

```

PROGRAM NOI00;
VAR

```

```

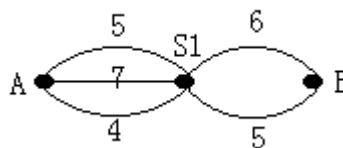
A      :ARRAY[1..36] OF 0..1
B      :ARRAY[0..31] OF INTEGER;
I,J,K,S,P:INTEGER;
BEGIN
  FOR I:=1 TO 36 DO    A[I]:=0;
    FOR I:=28 TO 32 DO    A[I]:=1;
  P:=1;    A[6]:=1;
  WHILE (P=1) DO
    BEGIN
      J:=27
      WHILE A[J]=1 DO    J:=J-1;
      ( A[J]:=1 )
      FOR I:=J+1 TO 27 DO ( A[i]:=0 )
      FOR I:=0 TO 31 DO B[I]:=0;
      FOR I:=1 TO 32 DO
        BEGIN
          ( S:=0 )
          FOR K:=I TO I+4 DO    S:=S*2+A[k];
          ( B[S]:=1 )
        END;
      S:=0;
      FOR I:=0 TO 31 DO    S:=S+B[I];
      IF ( S=32 ) THEN P:=0
    END;
    FOR I:=1 TO 32 DO    FOR J:=I TO I+4 DO    WRITE(A[J]);
  WRITELN
END.

```

例 2: 在 A、B 两个城市之间设有 N 个路站(如下图中的 S1, 且 $N < 100$), 城市与路站之间、路站和路站之间各有若干条路段(各路段数 ≤ 20 , 且每条路段上的距离均为一个整数)。

A、B 的一条通路是指: 从 A 出发, 可经过任一路段到达 S1, 再从 S1 出发经过任一路段, …最后到达 B。通路上路段距离之和称为通路距离(最大距离 ≤ 1000)。当所有的路段距离给出之后, 求出所有不同距离的通路个数(相同距离仅记一次)。

例如: 下图所示是当 $N=1$ 时的情况:



从 A 到 B 的通路条数为 6, 但因其中通路 $5+5=4+6$, 所以满足条件的不同距离的通路条数为 5。

算法说明: 本题采用穷举算法。

数据结构: N: 记录 A、B 间路站的个数

数组 D[I, 0] 记录第 I-1 个到第 I 路站间路段的个数

D[I, 1], D[I, 2], ... 记录每个路段距离

数组 G 记录可取到的距离

程序清单:

```

program CHU7_6;
var i,j,n,s:integer;
    b:array[0..100] of integer;
    d:array[0..100,0..20] of integer;
    g:array[0..1000] of 0..1;
begin

```

```

readln(n);
for i:=1 to n+1 do
begin
  readln(d[i,0]);
  for j:=1 to d[i,0] do read(d[i,j]);
end;
d[0,0]:=1;
for i:=1 to n+1 do b[i]:=1;
b[0]:=0;
for i:=1 to 1000 do g[i]:=0;
while b[0]<>1 do
begin
  s:=0;
  for i:=1 to n+1 do
    s:=s+d[i,b[i]];
    g[s]:=1;j:=n+1;
  while b[j]=d[j,0] do j:=j-1;
  b[j]:=b[j]+1;
  for i:=j+1 to n+1 do b[i]:=1;
end;
s:=0;
for i:=1 to 1000 do
  s:=s+g[i];
writeln(s);readln;
end.

```

练习:

1. 问题描述:将 n 个整数分成 k 组 ($k \leq n$, 要求每组不能为空), 显然这 k 个部分均可得到一个各自的和 s_1, s_2, \dots, s_k , 定义整数 P 为:

$$P = (S_1 - S_2)^2 + (S_1 - S_3)^2 + \dots + (S_1 - S_k)^2 + (s_2 - s_3)^2 + \dots + (S_{k-1} - S_k)^2$$

问题求解:求出一种分法, 使 P 为最小(若有多种方案仅记一种)

程序说明:

数组: $a[1], a[2], \dots, A[N]$ 存放原数

$s[1], s[2], \dots, s[K]$ 存放每个部分的和

$b[1], b[2], \dots, b[N]$ 穷举用临时空间

$d[1], d[2], \dots, d[N]$ 存放最佳方案

程序:

```

program exp4;
Var i,j,n,k : integer;
a :array [1..100] of integer;
b,d:array [0..100] of integer;
s :array[1..30] of integer;
begin
  readln(n,k);
  for I:=1 to n do read(a[I]);
  for I:=0 to n do b[I]:=1;
  cmin:=1000000;
  while (b[0]=1) do
  begin
    for I:=1 to k do ①
    for I:=1 to n do
      ②
    sum:=0;

```

```

for I:=1 to k-1 do
  for j:= ③
    sum:=sum+(s[I]-s[j])*(s[I]-s[j]);
if ④ then
  begin
    cmin:=sum;
    for I:=1 to n do d[I]:=b[I];
  end;
  j:=n;
  while ⑤ do j:=j-1;
  b[j]:=b[j]+1;
  for I:=j+1 to n do ⑥
end;
writeln(cmin);
for I:=1 to n do write(d[I]:40);
writeln;
end.

```

2. 问题描述: 有 n 种基本物质 ($n \leq 10$), 分别记为 P_1, P_2, \dots, P_n , 用 n 种基本物质构造物品, 这些物品使用在 k 个不同地区 ($k \leq 20$), 每个地区对物品提出自己的要求, 这些要求用一个 n

位的数表示: $a_1 a_2 \dots a_n$, 其中:

$a_i = 1$ 表示所需物质中必须有第 i 种基本物质

$a_i = -1$ 表示所需物质中必须不能有第 i 种基本物质 r

$a_i = 0$ 无所谓

问题求解: 当 k 个不同地区要求给出之后, 给出一种方案, 指出哪些物质被使用, 哪些物质不被使用。

程序说明: 数组 $b[1], b[2], \dots, b[n]$ 表示某种物品

$a[1..k, 1..n]$ 记录 k 个地区对物品的要求, 其中:

$a[i, j] = 1$ 表示第 i 个地区对第 j 种物品是需要的

$a[i, j] = 0$ 表示第 i 个地区对第 j 种物品是无所谓的

$a[i, j] = -1$ 表示第 i 个地区对第 j 种物品是不需要的

程序:

```

program gxp2;
Var i, j, k, n :integer;
    p:boolean;
    b :array [0..20] of 0..1;
    a :array[1..20,1..10] of integer;
begin
  readln(n,k);
  for i:=1 to k do
  begin
    for j:=1 to n do read(a[i,j]);
    readln;
  end;
  for i:=0 to n do b[i]:=0;
  p:=true;
  while ① do
  begin
    j:=n;
    while b[j]=1 do j:=j-1;
    ②
    for i:=j+1 to n do b[i]:=0;
  end;

```

```

③
  for i:=1 to k do
    for j :=1to n do
      if( a[i,j]=1 ) and (b[j]=0) or ④
        then p:=true;
    end;
  if ⑤
  then writeln('找不到!')
  else for i:=1 to n do
    if (b[i]=1) then writeln('物质',i,'需要')
    else writeln('物质',i,'不需要');
  end.

```

第七章 贪心策略

7.1 贪心策略的定义

7.2 贪心策略特点

7.3 典型例题与习题

在众多的计算机解题策略中，贪心策略可以算得上是最接近人们日常思维的一种解题策略，正基于此，贪心策略在各级各类信息学竞赛、尤其在对 NPC 类问题的求解中发挥着越来越重要的作用。

7.1 贪心策略的定义

贪心策略是：指从问题的初始状态出发，通过若干次的贪心选择而得出最优值(或较优解)的一种解题方法。

其实，从“贪心策略”一词我们便可以看出，贪心策略总是做出在当前看来是最优的选择，也就是说贪心策略并不是从整体上加以考虑，它所做出的选择只是在某种意义上的局部最优解，而许多问题自身的特性决定了该题运用贪心策略可以得到最优解或较优解。

例 1：在 n 行 m 列的正整数矩阵中，要求从每一行中选一个数，使得选出的 n 个数的和最大。

本题可用贪心策略：选 n 次，每一次选相应行中的最大值即可。

例 2：在一个 $N \times M$ 的方格阵中，每一格子赋予一个数（即为权）。规定每次移动时只能向上或向右。现试找出一条路径，使其从左下角至右上角所经过的权之和最大。

本题用贪心策略不能得到最优解，我们以 2×4 的矩阵为例。

| | | |
|---|---|----|
| 3 | 4 | 6 |
| 1 | 2 | 10 |

若按贪心策略求解，所得路径为：1, 3, 4, 6；

若按动态规划法求解，所得路径为：1, 2, 10, 6。

例 3：设定有 n 台处理机 p_1, p_2, \dots, p_n ，和 m 个作业 j_1, j_2, \dots, j_m ，处理机可并行工作，作业未完成不能中断，作业 j_i 在处理机上的处理时间为 t_i ，求解最佳方案，使得完成 m 项工作的时间最短？

本题不能用贪心算法求解：理由是若 $n=3, m=6$ 各作业的时间分别是 11 7 5 5 4 7

用贪心策略解(每次将作业加到最先空闲的机器上) $\text{time}=15$ ，用搜索策略最优时间应是 14，但是贪心策略给我们提供了一个线索那就是每台处理上的时间不超过 15，给搜索提供了方便。

总之：

1. 不能保证求得的最佳解是最佳的；
2. 只能用来求某些最大或最小解问题；
3. 能确定某些问题的可行解的范围，特别是给搜索算法提供了依据。

7.2 贪心策略的特点

贪心算法有什么样的特点呢？我认为，适用于贪心算法解决的问题应具有以下 2 个特点：

1、贪心选择性质：

所谓贪心选择性质是指应用同一规则 f ，将原问题变为一个相似的、但规模更小的子问题、而后的每一步都是当前看似最佳的选择。这种选择依赖于已做出的选择，但不依赖于未做出的选择。从全局来看，运用贪心策略解决的问题在程序的运行过程中无回溯过程。关于贪心选择性质，读者可在后文给出的贪心策略状态空间图中得到深刻地体会。

2、局部最优解：

我们通过特点 2 向大家介绍了贪心策略的数学描述。由于运用贪心策略解题在每一次都取得了最优解，但能够保证局部最优解得不一定是贪心算法。如大家所熟悉得动态规划算法就可以满足局部最优解，但贪心策略比动态规划时间效率更高站用内存更少，编写程序更简单。

7.3 典型例题与习题

例 4：背包问题：

有一个背包，背包容量是 $M=150$ 。有 7 个物品，物品可以分割成任意大小。

要求尽可能让装入背包中的物品总价值最大，但不能超过总容量。

| 物品 | A | B | C | D | E | F | G |
|----|----|----|----|----|----|----|----|
| 重量 | 35 | 30 | 60 | 50 | 40 | 10 | 25 |
| 价值 | 10 | 40 | 30 | 50 | 35 | 40 | 30 |

分析：

目标函数： $\sum p_i$ 最大

约束条件是装入的物品总重量不超过背包容量： $\sum w_i \leq M$ ($M=150$)

(1) 根据贪心的策略，每次挑选价值最大的物品装入背包，得到的结果是否最优？

(2) 每次挑选所占空间最小的物品装入是否能得到最优解？

(3) 每次选取单位容量价值最大的物品，成为解本题的策略。

程序如下：

```

program beibao;
const
  m=150;
  n=7;
var
  xu:integer;
  i,j:integer;
  goods:array[1..n,0..2] of integer;
  ok:array[1..n,1..2] of real;
procedure init;
var
  i:integer;
begin
  xu:=m;
  for i:=1 to n do
    begin
      write('Enter the price and weight of the ',i,'th goods:');
      goods[i,0]:=i;
      read(goods[i,1],goods[i,2]);
      readln;
      ok[i,1]:=0; ok[i,2]:=0;
    end;
end;
procedure make;
var
  bi:array[1..n] of real;
  i,j:integer;
  temp1,temp2,temp0:integer;
begin
  for i:=1 to n do

```



```

    bi[i]:=goods[i,1]/goods[i,2];
  for i:=1 to n-1 do
    for j:=i+1 to n do
      begin
        if bi[i]<bi[j] then begin
          temp0:=goods[i,0]; temp1:=goods[i,1]; temp2:=goods[i,2];
          goods[i,0]:=goods[j,0]; goods[i,1]:=goods[j,1]; goods[i,2]:=goods[j,2];
          goods[j,0]:=temp0; goods[j,1]:=temp1; goods[j,2]:=temp2;
        end;
      end;
    end;
  end;
begin
  init;
  make;
  for i:=1 to 7 do
    begin
      if goods[i,2]>xu then break;
      ok[i,1]:=goods[i,0]; ok[i,2]:=1;
      xu:=xu-goods[i,2];
    end;
  j:=i;
  if i<=n then
    begin
      ok[i,1]:=goods[i,0];
      ok[i,2]:=xu/goods[i,2];
    end;
  for i:=1 to j do
    writeln(ok[i,1]:1:0,':',ok[i,2]*goods[i,2]:2:1);
end.

```

例 5：旅行家的预算问题：

一个旅行家想驾驶汽车以最少的费用从一个城市到另一个城市，给定两个城市间的距离 d_1 ，汽车油箱的容量是 c ，每升汽油能行驶的距离 d_2 ，出发时每升汽油的价格是 p ，沿途加油站数为 n （可为 0），油站 i 离出发点的距离是 d_i ，每升汽油的价格是 p_i 。

计算结果四舍五入保留小数点后两位，若无法到达目的地输出 “No answer”

若输入：

$d_1=275.6$ $c=11.9$ $d_2=27.4$ $p=8$ $n=2$

$d[1]=102$ $p[1]=2.9$

$d[2]=220$ $p[2]=2.2$

output

26.95

本问题的贪心策略是：找下一个较便宜的油站，根据距离确定加满、不加、加到刚好到该站。

程序如下：

```

program jiayou;
const maxn=10001;
zero=1e-16;
type
  jd=record
    value,way,over:real;
  end;
var oil:array[1..maxn] of ^jd;
    n:integer;
    d1,c,d2,cost,maxway:real;

```

```

function init:boolean;
var i:integer;
begin
  new(oil[1]);
  oil[1]^way:=0;
  read(d1,c,d2,oil[1]^value,n);
  maxway:=d2*c;
  for i:=2 to n+1 do
    begin
      new(oil[i]);
      readln(oil[i]^way,oil[i]^value);
      oil[i]^over:=0;
    end;
  inc(n,2);
  new(oil[n]);
  oil[n]^way:=d1;
  oil[n]^value:=0;
  oil[n]^over:=0;
  for i:=2 to n do
    if oil[i]^way-oil[i-1]^way>maxway then
      begin
        init:=false;
        exit
      end;
  init:=true;
end;
procedure buy(i:integer;miles:real);
begin
  cost:=cost+miles/d2*oil[i]^value;
end;
procedure solve;
var i,j:integer;
    s:real;
begin
  i:=1;j:=i+1;
  repeat
    s:=0.0;
    while( s<=maxway+zero) and (j<=n-1) and (oil[i]^value<=oil[j]^value) do
      begin
        inc(j);
        s:=s+oil[j]^way-oil[j-1]^way
      end;
    if s<=maxway+zero then
      if (oil[i]^over+zero>=oil[j]^way-oil[i]^way) then
        oil[j]^over:=oil[i]^over-(oil[j]^way-oil[i]^way) else
        begin
          buy(i,oil[j]^way-oil[i]^way-oil[i]^over);
          oil[j]^over:=0.0;
        end
      else begin
        buy(i,maxway-oil[i]^over);
        j:=i+1;
        oil[j]^over:=maxway-(oil[j]^way-oil[i]^way);

```

```

    end;
    i:=j;
until i=n;
end;
begin
    cost:=0;
    if init then begin
        solve;
        writeln(cost:0:2);
    end else writeln('No answer');
end.

```

例 6: n 个部件, 每个部件必须经过先 A 后 B 两道工序。

以知部件 i 在 A, B 机器上的时间分别为 a_i , b_i 。如何安排加工顺序, 总加工时间最短?

输入:

5

| 部件 | 1 | 2 | 3 | 4 | 5 |
|-------|---|---|---|---|----|
| a_i | 3 | 5 | 8 | 7 | 10 |
| b_i | 6 | 2 | 1 | 4 | 9 |

输出:

34

1 5 4 2 3

本问题的贪心策略是 A 机器上加工短的应优先, B 机器上加工短的应靠后。

程序如下:

```

program workorder;
const maxn=100;
type jd=record
    a,b,m,o:integer;
end;
var n,min,i:integer;
    c:array[1..maxn] of jd;
    order:array[1..maxn] of integer;
procedure init;
var i:integer;
begin
    readln(n);
    for i:=1 to n do
        read(c[i].a);
        readln;
    for i:=1 to n do
        read(c[i].b);
        readln;
    for i:=1 to n do
        begin
            if c[i].a<c[i].b then c[i].m:=c[i].a else c[i].m:=c[i].b;
            c[i].o:=i;
        end;
    end;
end;
procedure sort;
var i,j,k,t:integer;
    temp:jd;
begin

```

```

for i:=1 to n-1 do
begin
  k:=i;t:=c[i].m;
  for j:=i+1 to n do
    if c[j].m<t then begin t:=c[j].m;k:=j end ;
  if k<>i then begin temp:=c[i];c[i]:=c[k];c[k]:=temp end
end;
end;
procedure playorder;
var i,s,t:integer;
begin
  fillchar(order,sizeof(order),0);
  s:=1;
  t:=n;
  for i:=1 to n do
    if c[i].m=c[i].a then begin order[s]:=i;s:=s+1 end
    else begin order[t]:=i;t:=t-1;end;
end;
procedure calc_t;
var i,t1,t2:integer;
begin
  t1:=0;t2:=0;
  for i:=1 to n do
    begin
      t1:=t1+c[order[i]].a;
      if t2<t1 then t2:=t1;
      t2:=t2+c[order[i]].b;
    end;
  min:=t2;
end;
begin
  init;
  sort;
  playorder;
  calc_t;
  writeln(min);
  for i:=1 to n do
    write(c[order[i]].o,' ');
  writeln;
end.

```

习题：

1. 最佳浏览路线问题：

某旅游区的街道成网格状（见图），其中东西向的街道都是旅游街，南北向的街道都是林荫道。由于游客众多，旅游街被规定为单行道。游客在旅游街上只能从西向东走，在林荫道上既可以由南向北走，也可以从北向南走。阿隆想到这个旅游区游玩。他的好友阿福给了他一些建议，用分值表示所有旅游街相邻两个路口之间的道路值得浏览得程度，分值从-1 0 0到1 0 0的整数，所有林荫道不打分。所有分值不可能全是负值。

例如下图是被打过分的某旅游区的街道图：

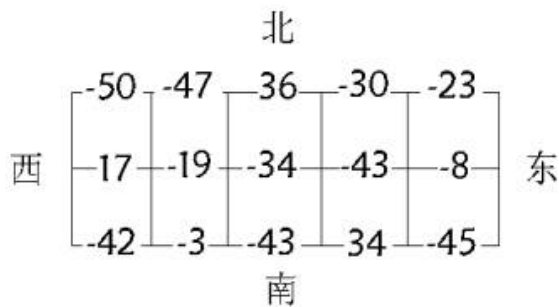


图 6

阿隆可以从任一路口开始浏览，在任一路口结束浏览。请你写一个程序，帮助阿隆寻找一条最佳的浏览路线，使得这条路线的所有分值总和最大。

2. 删数问题

键盘输入一个高精度的正整数 N ，去掉其中任意 S 个数字后剩下的数字按左右次序组成一个新的正整数。对给定的 N 和 S ，寻找一种删数规则使得剩下得数字组成的新数最小

第八章 分治策略

8.1 分治策略的定义

8.2 分治法解题的步骤

8.3 典型例题与习题

任何一个可以用计算机求解的问题所需的计算时间都与其规模有关。问题的规模越小，越容易直接求解，解题所需的计算时间也越少。例如，对于 n 个元素的排序问题，当 $n=1$ 时，不需任何计算。 $n=2$ 时，只要作一次比较即可排好序。 $n=3$ 时只要作 3 次比较即可，…。而当 n 较大时，问题就不那么容易处理了。要想直接解决一个规模较大的问题，有时是相当困难的。

分治法的设计思想是，将一个难以直接解决的大问题，分割成一些规模较小的相同问题，以便各个击破，分而治之。

8.1 分治策略的定义

分治策略是：对于一个规模为 n 的问题，若该问题可以容易地解决（比如说规模 n 较小）则直接解决，否则将其分解为 k 个规模较小的子问题，这些子问题互相独立且与原问题形式相同，递归地解这些子问题，然后将各子问题的解合并得到原问题的解。这种算法设计策略叫做**分治法**。

如果原问题可分割成 k 个子问题， $1 < k \leq n$ ，且这些子问题都可解，并可利用这些子问题的解求出原问题的解，那么这种分治法就是可行的。由分治法产生的子问题往往是原问题的较小模式，这就为使用递归技术提供了方便。在这种情况下，反复应用分治手段，可以使子问题与原问题类型一致而其规模却不断缩小，最终使子问题缩小到很容易直接求出其解。这自然导致递归过程的产生。分治与递归像一对孪生兄弟，经常同时应用在算法设计之中，并由此产生许多高效算法。

分治法所能解决的问题一般具有以下几个特征：

1. 该问题的规模缩小到一定的程度就可以容易地解决；
2. 该问题可以分解为若干个规模较小的相同问题，即该问题具有最优子结构性质。
3. 利用该问题分解出的子问题的解可以合并为该问题的解；
4. 该问题所分解出的各个子问题是相互独立的，即子问题之间不包含公共的子子问题。

上述的第一条特征是绝大多数问题都可以满足的，因为问题的计算复杂性一般是随着问题规模的增加而增加；第二条特征是应用分治法的前提，它也是大多数问题可以满足的，此特征反映了递归思想的应用；第三条特征是关键，能否利用分治法完全取决于问题是否具有第三条特征，如果具备了第一条和第二条特征，而不具备第三条特征，则可以考虑用贪心法或动态规划法。第四条特征涉及到分治法的效率，如果各子问题是不独立的，则分治法要做许多不必要的工作，重复地解公共的子问题，此时虽然可用分治法，但一般用动态规划法较好。

8.2 分治法解题的步骤

分治法的基本步骤

分治法在每一层递归上都有三个步骤：

1. **分解：**将原问题分解为若干个规模较小，相互独立，与原问题形式相同的子问题；
2. **解决：**若子问题规模较小而容易被解决则直接解，否则递归地解各个子问题；
3. **合并：**将各个子问题的解合并为原问题的解。

它的一般的算法设计模式如下：

Divide-and-Conquer(P)

1. if $|P| \leq n_0$
2. then return(ADHOC(P))
3. 将 P 分解为较小的子问题 P_1, P_2, \dots, P_k
4. for $i \leftarrow 1$ to k
5. do $y_i \leftarrow \text{Divide-and-Conquer}(P_i)$ \triangle 递归解决 P_i
6. $T \leftarrow \text{MERGE}(y_1, y_2, \dots, y_k)$ \triangle 合并子问题
7. return(T)

其中 $|P|$ 表示问题 P 的规模； n_0 为一阈值，表示当问题 P 的规模不超过 n_0 时，问题已容易直接解出，不必再继续分解。

ADH

OC(P)是该分治法中的基本子算法，用于直接解小规模的问题 P 。因此，当 P 的规模不超过 n_0 时，直接用算法 **ADHOC(P)** 求解。

算法 **MERGE**(y_1, y_2, \dots, y_k)是该分治法中的合并子算法，用于将 P 的子问题 P_1, P_2, \dots, P_k 的相应的解 y_1, y_2, \dots, y_k 合并为 P 的解。

根据分治法的分割原则，原问题应该分为多少个子问题才较适宜？各个子问题的规模应该怎样才为适当？这些问题很难予以肯定的回

答。但人们从大量实践中发现，在用分治法设计算法时，最好使子问题的规模大致相同。换句话说，将一个问题分成大小相等的 k 个子问题的处理方法是行之有效的。许多问题可以取 $k=2$ 。这种使子问题规模大致相等的做法是出自一种**平衡(balancing)子问题**的思想，它几乎总是比子问题规模不等的做法要好。

8.3 典型例题与习题**例 1：二分查找**

在对线性表的操作中，经常需要查找某一个元素在线性表中的位置。此问题的输入是待查元素 x 和线性表 L ，输出为 x 在 L 中的位置或者 x 不在 L 中的信息。比较自然的想法是一个一个地扫描 L 的所有元素，直到找到 x 为止。这种方法对于有 n 个元素的线性表在最坏情况下需要 n 次比较。一般来说，如果没有其他的附加信息，在有 n 个元素的线性表中查找一个元素在最坏情况下都需要 n 次比较。下面我们考虑一种简单的情况。假设该线性表已经排好序了，不妨设它按照主键的递增顺序排列（即由小到大排列）。在这种情况下，我们是否有改进查找效率的可能呢？

如果线性表里只有一个元素，则只要比较这个元素和 x 就可以确定 x 是否在线性表中。因此这个问题满足分治法的第一个适用条件；同时我们注意到对于排好序的线性表 L 有以下性质：比较 x 和 L 中任意一个元素 $L[i]$ ，若 $x=L[i]$ ，则 x 在 L 中的位置就是 i ；如果 $x < L[i]$ ，由于 L 是递增排序的，因此假如 x 在 L 中的话， x 必然排在 $L[i]$ 的前面，所以我们只要在 $L[i]$ 的前面查找 x 即可；如果 $x > L[i]$ ，同理我们只要在 $L[i]$ 的后面查找 x 即可。无论是在 $L[i]$ 的前面还是后面查找 x ，其方法都和 L 中查找 x 一样，只不过是线性表的规模缩小了。这就说明了此问题满足分治法的第二个和第三个适用条件。很显然此问题分解出的子问题相互独立，即在 $L[i]$ 的前面或后面查找 x 是独立的子问题，因此满足分治法的第四个适用条件。

于是我们得到利用分治法在有序表中查找元素的算法。

```
function Binary_Search(L,a,b,x);
begin
  if a>b then return(-1)
    else begin
      m:=(a+b) div 2;
      if x=L[m] then return(m)
        else if x>L[m]
          then return(Binary_Search(L,m+1,b,x));
          else return(Binary_Search(L,a,m-1,x));
    end;
end;
```

在以上算法中， L 为排好序的线性表， x 为需要查找的元素， b, a 分别为 x 的位置的上下界，即如果 x 在 L 中，则 x 在 $L[a..b]$ 中。每次我们用 L 中间的元素 $L[m]$ 与 x 比较，从而确定 x 的位置范围。然后递归地缩小 x 的范围，直到找到 x 。

例 2：快速排序

快速排序的基本思想是基于分治法的。对于输入的子序列 $L[p..r]$ ，如果规模足够小则直接进行排序，否则分三步处理：

- 分解(Divide)：将输入的序列 $L[p..r]$ 划分成两个非空子序列 $L[p..q]$ 和 $L[q+1..r]$ ，使 $L[p..q]$ 中任一元素的

值不大于 $L[q+1..r]$ 中任一元素的值。

- 递归求解(Conquer): 通过递归调用快速排序算法分别对 $L[p..q]$ 和 $L[q+1..r]$ 进行排序。
- 合并(Merge): 由于对分解出的两个子序列的排序是就地进行的, 所以在 $L[p..q]$ 和 $L[q+1..r]$ 都排好序后不需要执行任何计算 $L[p..r]$ 就已排好序。

这个解决流程是符合分治法的基本步骤的。因此, 快速排序法是分治法的经典应用实例之一。

程序代码如下:

```
program kspv;
const n=7;
type
arr=array[1..n] of integer;
var
a:arr;
i:integer;
procedure quicksort(var b:arr; s,t:integer);
var i,j,x:integer;
begin
  i:=s;j:=t;x:=b[i];
  repeat
    while (b[j]>=x) and (j>i) do j:=j-1;
    if j>i then begin b[i]:=b[j];i:=i+1;end;
    while (b[i]<=x) and (i<j) do i:=i+1;
    if i<j then begin b[j]:=b[i];j:=j-1; end
  until i=j;
  b[i]:=x;
  i:=i+1;j:=j-1;
  if s<j then quicksort(b,s,j);
  if i<t then quicksort(b,i,t);
end;
begin
write('input data:');
for i:=1 to n do read(a[i]);
writeln;
quicksort(a,1,n);
write('output data:');
for i:=1 to n do write(a[i]:6);
writeln;
end.
```

例 3: 归并排序

归并就是将多个有序的数列合成一个有序的数列。将两个有序序列合并为一个有序序列叫二路归并(merge)。归并排序就是 n 长度为 1 的子序列, 两两归并最后变为有序的序列。

程序代码如下:

```
program gbp;
const maxn=7;
type arr=array[1..maxn] of integer;
var a,b,c:arr;
    i:integer;
procedure merge(r:arr;l,m,n:integer;var r2:arr);
var i,j,k,p:integer;
begin
  i:=1;j:=m+1;k:=l-1;
  while (i<=m)and (j<=n) do
    begin
```

```

    k:=k+1;
    if r[i]<=r[j] then begin r2[k]:=r[i];i:=i+1 end
                        else begin r2[k]:=r[j];j:=j+1 end
    end;
    if i<=m then
        for p:=i to m do begin k:=k+1;r2[k]:=r[p] end;
    if j<=n then
        for p:=j to n do begin k:=k+1;r2[k]:=r[p] end;
end;
procedure mergesort( var r,r1:arr;s,t:integer);
var k:integer; c:arr;
begin
    if s=t then r1[s]:=r[s] else
        begin
            k:=(s+t) div 2;
            mergesort(r,c,s,k);
            mergesort(r,c,k+1,t);
            merge(c,s,k,t,r1)
        end;
end;
begin
    write('Enter data:');
    for i:= 1 to maxn do
        read(a[i]);
    mergesort(a,b,1,maxn);
    for i:=1 to maxn do
        write(b[i]:9);
    writeln;
end.

```

练习：

1. 写出二分查找的程序。
2. 赛程问题。有 n ($n=2^m$) 编号为 1 到 n 的运动队, 参加某项运动的单循环比赛, 请安排一个比赛日程?
(共 $n-1$ 天比赛, 每天每队必须比赛一场)。

思考题 1:

最接近点对问题

在应用中, 常用诸如点、圆等简单的几何对象代表现实世界中的实体。在涉及这些几何对象的问题中, 常需要了解其邻域中其他几何对象的信息。例如, 在空中交通控制问题中, 若将飞机作为空间中移动的一个点来看待, 则具有最大碰撞危险的 2 架飞机, 就是这个空间中最接近的一点对。这类问题是计算几何学中研究的基本问题之一。下面我们着重考虑平面上的最接近点对问题。最接近点对问题的提法是: 给定平面上 n 个点, 找其中的一对点, 使得在 n 个点的所有点对中, 该点对的距离最小。严格地说, 最接近点对可能多于 1 对。为了简单起见, 这里只限于找其中的一对。

思考题 2:

导线和开关问题

如图 1 所示, 具有 3 根导线的电缆把 A 区和 B 区连接起来。在 A 区 3 根导线标以 1, 2, 3; 在 B 区导线 1 和 3 被连到开关 3, 导线 2 连到开关 1。

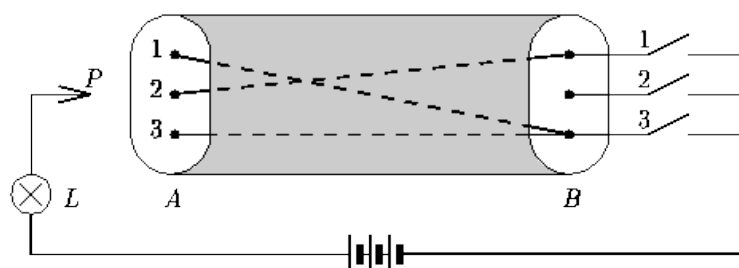


图 1 一个有 3 条导线和 3 个开关的实例

一般说来, 电缆含 m ($1 \leq m \leq 90$) 根导线, 在 A 区标以 $1, 2, \dots, m$ 。在 B 区有 m 个开关, 标为 $1, 2, \dots, m$ 。每一根导线都被严格地连到这些开关中的某一个上; 每一个开关上可以连有 0 根或多根导线。怎样快速测试哪根导线连在哪个开关上?

第四部分 数据结构

第一章 什么是数据结构

1.1 基本概念和术语

1.2 数据的逻辑结构和物理结构

1.1 基本概念和术语

1. 数据 (data):

是对客观事物的符号的表示, 是所有能输入到计算机中并被计算机程序处理的符号的总称。

2. 数据元素 (data element):

是数据的基本单位, 在计算机程序中通常作为一个整体来处理。一个数据元素由多个 **数据项 (data item)** 组成, 数据项是数据不可分割的最小单位。

3. 数据结构 (data structure):

是相互之间存在一种或多种特定关系的数据元素的集合。数据结构是一个二元组, 记为:

$data_structure = (D, S)$. 其中 D 为数据元素的集合, S 是 D 上关系的集合。

数据元素相互之间的关系称为结构 (structure)。根据数据元素之间关系的不同特性, 通常由下列四类基本结构:

- (1) 集合: 数据元素间的关系是同属一个集合。(图 1)
- (2) 线性结构: 数据元素间存在一对一的关系。(图 2)
- (3) 树形结构: 结构中的元素间的关系是一对多的关系。(图 3)
- (4) 图 (网) 状结构: 结构中的元素间的关系是多对多的关系。(图 4)

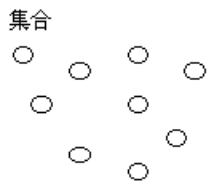


图 1

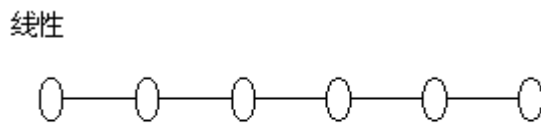


图 2

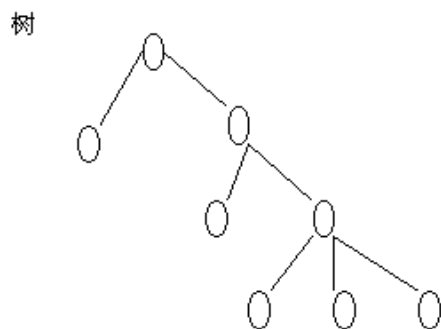


图 3

图

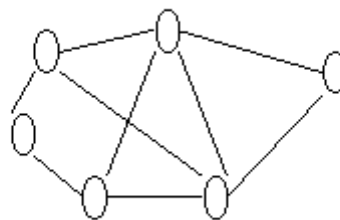


图 4

1.2 数据的逻辑结构和物理结构

逻辑结构: 数据元素之间存在的关系 (逻辑关系) 叫数据的逻辑结构。

物理结构: 数据结构在计算机中的表示 (映象) 叫数据的物理结构。

一种逻辑结构可映象成不同的存储结构: 顺序存储结构和非顺序存储结构 (链式存储结构和散列结构)。

第二章 线性表

2.1 线性表的逻辑结构及基本运算

2.2 线性表的顺序存储结构

2.3 线性表的链式存储结构

2.1 线性表的逻辑结构及基本运算

1. 线性表简单的定义

n 个数据元素的有限序列其特点是除了表头和表尾外, 表中的每一个元素有且仅有唯一的前驱和唯一的后继, 表头有且只有一个后继, 表尾有且只有一个前驱。

2. 线性表的基本运算

| | |
|------------------|--|
| length (L) | 返回表 L 的长度, 即元素个数。 |
| IsEmpty (L) | 如果表 L 为空表 (长度为 0) 则返回 true, 否则返回 false。 |
| next (L, p) | 这是一个函数, 函数值为表 L 中位置 p 的后继位置。如果 p 是 L 中结束元素的位置, 则 $L.Next(p)=L.end$ 。当 L 中没有位置 p 或 $p=L.end$ 时, 该运算无定义。 |
| prev (L, p) | 这是一个函数, 函数值为表 L 中位置 p 的前驱位置。当 L 中没有位置 p 或 p 是 L 中开始元素的位置时, 该运算无定义。 |
| get (L, p) | 这是一个函数, 函数值为 L 中位置 p 处的元素。当 $p=L.end$ 或 L 中没有位置 p 时, 该运算无定义。 |
| insert (L, x, p) | 在表 L 的位置 p 处插入元素 x, 并将原来占据位置 p 的元素及其后面的元素都向后推移一个位置。例如, 设 L 为 a_1, a_2, \dots, a_n , 那么在执行 insert (L, x, p) 后, 表 L 变为 $a_1, a_2, \dots, a_{p-1}, x, a_p, \dots, a_n$ 。若 p 为 L.end, 那么表 L 变为 a_1, a_2, \dots, a_n, x 。若表 L 中没有位置 p, 则该运算无定义。 |
| delete (L, p) | 从表 L 中删除位置 p 处的元素。例如, 当 L 为 a_1, a_2, \dots, a_n 时, 执行 delete (L, p) 后, L 变为 $a_1, a_2, \dots, a_{p-1}, a_{p+1}, \dots, a_n$ 。当 L 中没有位置 p 或 $p=L.end$ 时, 该运算无定义。 |
| locate (L, x) | 这是一个函数, 函数值为元素 x 在 L 中的位置。若 x 在 L 中重复出现多次, 则函数值为 x 第一次出现的位置。当 x 不在 L 中时, 函数值为 0 |
| MakeEmpty (L) | 这是一个将 L 变为空表的方法。 |

例 1 假设两个线性表 LA, LB 分别代表两个集合 A 和 B: 求 $A=A \cup B$

```

proc union(var la:linear_list;lb:linear_list);
begin
  n:=length(la);
  for i:=1 to length(lb) do
    x:=get(lb,i);
    k:=locate(la,x);
    if k=0 then begin insert(la,n+1,x);n:=n+1 end;
  end

```

例 2 已知线性表 la, lb 中的数据元素按值非递减有序排列, 现要求将 la, lb 归并为一个新的线性表 lc 且 lc 按值非递减。

```

proc merge(la,lb:linear_list;var lc:linear_list);
begin
  i:=1;j:=1;k:=0;
  while (i<=length(la)) and (j<=length(lb)) do
    if get(la,i)<=get(lb,j) then begin insert(lc,k+1,get(la,i));k:=k+1;i:=i+1 end
    else begin insert(lc,k+1,get(lb,j));k:=k+1;j:=j+1 end
  while i<=length(la) do
    begin insert(lc,k+1,get(la,i));k:=k+1;i:=i+1; end
  while j<=length(lb) do
    begin insert(lc,k+1,get(lb,j));k:=k+1;j:=j+1 end
end;

```

2.2 线性表的顺序存储结构

线性表的顺序存储即用一组地址连续的存储单元依次存储线性表中的元素。

设线性表中每个元素需占用 r 个存储单元则

$$\text{loc}(a_{i+1}) = \text{loc}(a_i) + r$$

```
loc (ai) = loc (ai) + (i-1) * r
```

这时可以这样定义线性表类型

```
type sqliist=record
    data: array[1..maxlen] of datatype;
    last: 0..maxlen
end
```

在这种存储方式下，容易实现对表的遍历。要在表的尾部插入一个新元素，也很容易。但是要在表的中间位置插入一个新元素，就必须先将其后面的所有元素都后移一个单元，才能腾出新元素所需的位置。执行删除运算的情形类似。如果被删除的元素不是表中最后一个元素，则必须将它后面的所有元素前移一个位置，以填补由于删除所造成的空缺。这两种运算的时间复杂度均为 $O(n)$ n 为表的长度。

2.3 线性表的链式存储结构

1. 单链表

定义：实现表的另一种方法是用指针将存储表元素的那些单元依次串联在一起。这种方法避免了在数组中用连续的单元存储元素的缺点，因而在执行插入或删除运算时，不再需要移动元素来腾出空间或填补空缺。然而我们为此付出的代价是，需要在每个单元中设置指针来表示表中元素之间的逻辑关系，因而增加了额外的存储空间开销。

为了将存储表元素的所有单元用指针串联起来，我们让每个单元包含一个元素域和一个指针域，其中的指针指向表中下一个元素所在的单元。例如，如果表是 a_1, a_2, \dots, a_n ，那么含有元素 a_i 的那个单元中的指针应指向含有元素 a_{i+1} 的单元 ($i=1, 2, \dots, n-1$)。含有 a_n 的那个单元中的指针是空指针 nil 。此外，通常我们还为每一个表设置一个表头单元 $header$ ，其中的指针指向开始元素中所在的单元，但表头单元 $header$ 中不含任何元素。设置表头单元的的目的是为了使表运算中的一些边界条件更容易处理。这一点我们在后面可以看到。如果我们愿意单独地处理诸如在表的第一个位置上进行插入与删除操作等边界情况，也可以简单地用一个指向表的第一个单元的指针来代替表头单元。

上述这种用指针来表示表的结构通常称为**单链表**，或简称为**单链表或链表**。单链表的逻辑结构如图 1 所示。表示空表的单链表只有一个单元，即表头单元 $header$ ，其中的指针是空指针 nil 。

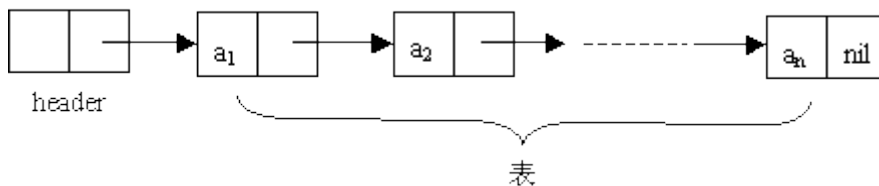


图 1 单链表示意图

为了便于实现表的各种运算，在单链表中位置变量的意义与用数组实现的表不同。在单链表中位置 i 是一个指针，它所指向的单元是元素 a_{i-1} 所在的单元，而不是元素 a_i 所在的单元 ($i=2, 3, \dots, n$)。位置 1 是指向表头单元 $header$ 的指针。位置 $End(L)$ 是指向单链表 L 中最后一个单元的指针。这样做的目的是为了为了避免在修改单链表指针时需要找一个元素的前驱元素的麻烦，因为在单链表中只设置指向后继元素的指针，而没有设置指向前驱元素的指针。

单链表结构的主要类型可形式地定义为：

```
type pointer = ^nodetype
    nodetype = record
        data: datatype;
        next: pointer;
    end;
linklist = pointer;
```

基本运算的实现：

过程 $Insert(x, p)$

功能

在表 L 的位置 p 处插入元素 x ，并将原来占据位置 p 的元素及其后面的元素都向后推移一个位置。例如，设 L 为 a_1, a_2, \dots, a_n ，那么在执行 $Insert(x, p)$ 后，表 L 变为 $a_1, a_2, \dots, a_{p-1}, x, a_p, \dots, a_n$ 。若 p 为 $End(L)$ ，那么表 L 变为 a_1, a_2, \dots, a_n, x 。若表 L 中没有位置 p ，则该运算无定义。

实现

```
Procedure Insert(x:datatype;p:pointer);
var
    temp:integer;
begin
```

```

new(temp);
temp^.data:=x;
temp^.next:= p^.next
p^.next:=temp;
end;

```

上述算法中，链表指针的修改情况见图 2

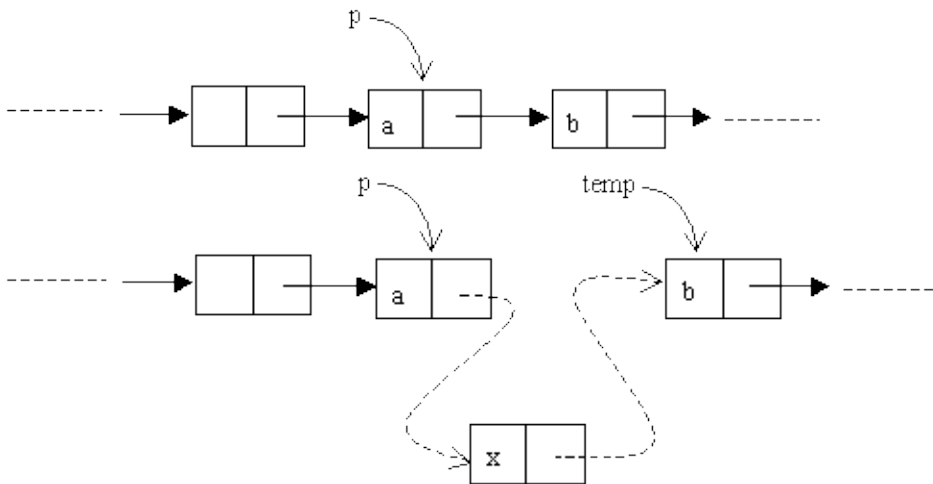


图 2(a) 是执行 Insert 运算之前的情况。我们要在指针 p 所指的单元之后插入一个新元素 x 。图 2(b) 是执行 Insert 运算以后的结果，其中的虚线表示新的指针。

在上述 Insert 算法中，位置变量 p 指向单链表中一个合法位置，要插入的新元素 x 应紧接在 p 所指单元的后面。指针 p 的合法性应在执行 Insert 运算之前判定。往一个单链表中插入新元素通常在表头或表尾进行，因此 p 的合法性容易判定。Insert 运算所需的时间显然为 $O(1)$ 。

过程 Delete(p)

功能

从表 L 中删除位置 p 的下一元素。例如，当 L 为 a_1, a_2, \dots, a_n 时，执行 Delete(p) 后， L 变为 $a_1, a_2, \dots, a_{p-1}, a_{p+1}, \dots, a_n$ 。当 L 中没有位置 p 或 $p = \text{End}(L)$ 时，该运算无定义。

实现

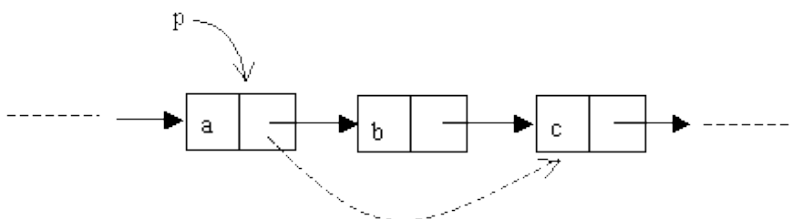
```

Procedure Delete(p:pointer);
var
  q:integer
begin
  q:=p^.next;
  p^.next:=p^.next^.next;
  dispose(q);
end;

```

说明

这个过程很简单，其指针的修改如图 3 所示。



若要从一个表中删除一个元素 x ，但不知道它在表中的位置，则应先用 Locate(x, L) 找出指示要删除的元素的位置，然后再用 Delete 删除该位置指示的元素。Delete 过程所需的时间显然也为 $O(1)$ 。

2. 静态链表

静态链表：用游标指示数组单元地址的下标值，它属整数类型数组元素是记录类型，记录中包含一个表元素和一个作为游标的整数；具体说明如下：

```

type jid=record
  data:datatype;

```

```

    next:integer;
  end;
var alist=array[0..maxsize] of jid

```

游标就是。我们可以用游标来模拟指针，对于一个表 L，我们用一个整型变量 Lhead (如 Lhead=0) 作为 L 的表头游标。这样，我们就可以用游标来模拟指针，实现单链表中的各种运算。当 $i>1$ 时，表示第 i 个位置的位置变量 p_i 的值是数组 alist 中存储表 L 的第 $i-1$ 个元素 next 值，用 $p:=alist(p).next$ 后移。照此，我们虽然是用数组来存储表中的元素，但在作表的插入和删除运算时，不需要移动元素，只要修改游标，从而保持了用指针实现表的优点。因此，有时也将这种用游标实现的表称为**静态链表**。

3. 循环链表

我们在用指针实现表时，表中最后一个元素所在单元的指针域(next)为空指针 nil。如果将这个空指针改为指向表头单元的指针就使整个链表形成一个环。这种首尾相接的链表称为**循环链表**。在循环链表中，从任意一个单元出发可以找到表中其他单元。图 6 所示的是一个单链的循环链表或简称为单循环链表。

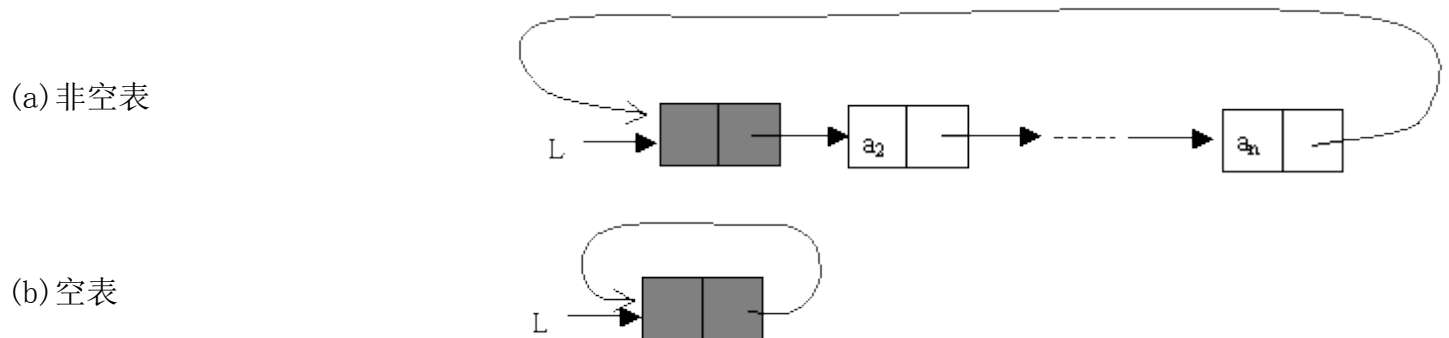


图 6 单循环链表

在单循环链表上实现表的各种运算的算法与单链表的情形是类似的。它们仅在循环终止条件上有所不同：前者是 p 或 $p.next$ 指向表头单元；后者是 p 或 $p.next$ 指向空 (nil)。在单链表中我们用指向表头单元的指针表示一个表 L，这样就可以在 $O(1)$ 时间内找到表 L 中的第一个元素。然而要找到表 L 中最后一个元素就要花 $O(n)$ 时间遍历整个链表。在单循环链表中，我们也可以用指向表头单元的指针表示一个表 L。但是，如果我们用指向表尾的指针表示一个表 L 时，我们就可以在 $O(1)$ 时间内找到表上中最后一个元素。同时通过表尾单元中指向表头单元的指针，我们也可以在 $O(1)$ 时间内找到表 L 中的第一个元素。在许多情况下，用这种表示方法可以简化一些关于表的运算。

4. 双链表

单循环链表中，虽然从任一单元出发，可以找到其前驱单元，但需要 $O(n)$ 时间。如果我们希望能快速确定表中任一元素的前驱和后继元素所在的单元，可以在链表的每个单元中设置两个指针，一个指向后继，另一个指向前驱，形成图 8 所示的**双向链表**或简称为**双链表**。

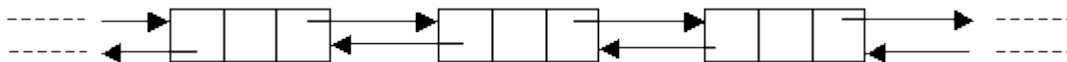


图 8 双链表

由于在双链表中可以直接确定一个单元的前驱单元和后继单元，我们可以用一种更自然的方式表示元素的位置，即用指向双链表中第 i 个单元而不是指向其前一个单元的指针来表示表的第 i 个位置。

双链表的单元类型定义如下。

```

type dupointer=^celltype
  celltype=record
    data:datatype;
    next, previous:dupointer;
  end;
dulist=dupointer;

```

和单链的循环表类似，双链表也可以有相应的循环表。我们用一个表头单元将双链表首尾相接，即将表头单元中的 previous 指针指向表尾，并将表尾单元的 next 指针指向表头单元，构成如图 9 所示的双向循环链表。

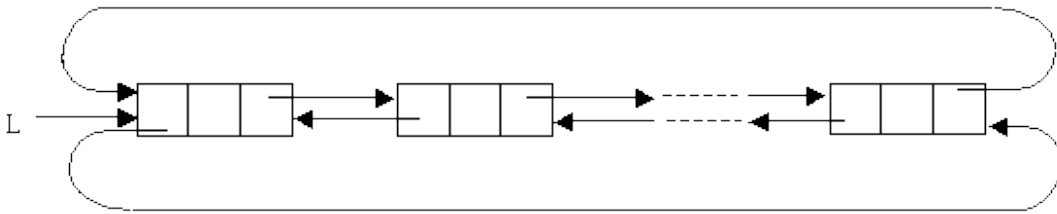


图 9 双向循环链表

下面仅以双向循环链表为例给出三种基本运算的实现。

(1) 在双向循环链表 L 的位置 p 处插入一个新元素 x 的过程 Insert 可实现如下。

```

procedure Insert(x:datatype; p:pointer; var L:duList);
Var
    q:integer
begin
    new(q);
    q^.data:=x;
    q^.previous:=p^.previous;
    q^.next:=p;
    p^.previous^.next:=q;
    p^.previous:=q;
end;
    
```

上述算法对链表指针的修改情况见图 10。

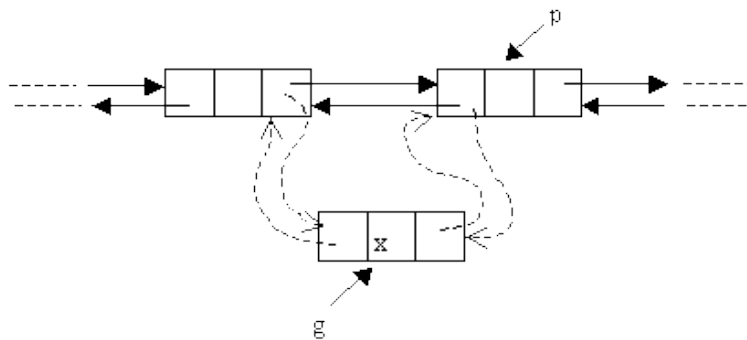


图 10 在双向循环链表中插入一个元素

算法 Insert 中没有检查位置 p 的合法性，因此在调用 Insert 之前应保证位置 p 的合法性。由于插入通常是在表的头尾两端进行的，所以容易检查位置 p 的合法性。

(2) 从双向循环链表 L 中删除位置 p 处的元素可实现如下：

```

procedure Delete(p:integer; var L:duList);
begin
    if (p<>nil) and (p<>L) then
        begin
            p^.previous^.next:=p^.next;
            p^.next^.previous:=p^.previous;
            dispose(p);
        end;
end;
    
```

上述算法对链表指针的修改情况见图 11。

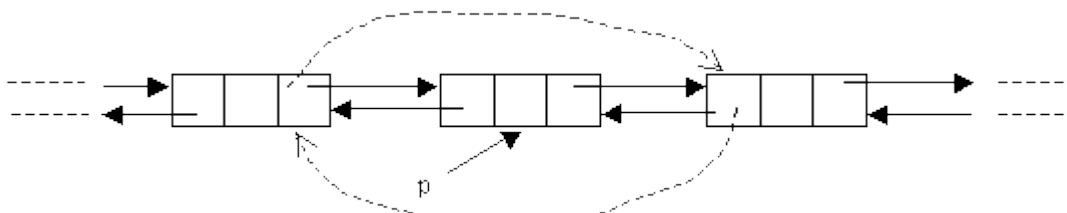


图 11 从双向循环链表中删除一个元素

5. 链表的应用

例 1: 求 $(A-B) \cup (B-A)$ 其中 A, B 代表两个集合 (用静态链表)

```

program jtlb;
const maxsize=20;
type jid=record
    data:integer;
    next:integer;
end;
jtlst=array[0..maxsize] of jid;
var a:jtlst;
last,i,k,pre,x,m,n:integer;
begin
    for i:=0 to maxsize-1 do a[i].next:=i+1;
    a[maxsize].next:=-1;
    readln(m,n);
    for i:=1 to m do
        begin read(x);a[i].data:=x; end;
    readln;
    last:=m;
    for i:=1 to n do
        begin
            read(x);
            k:=1;pre:=0;
            while (a[k].data<>x) and (k<>a[last].next) do
                begin
                    pre:=k;k:=a[k].next;
                end;
            if k=a[last].next then
                begin a[k].data:=x;last:=k end
            else begin a[pre].next:=a[k].next;if k=last then last:=pre end
            end;
        end;
    writeln;
    i:=0;
    repeat
        i:=a[i].next;
        write(a[i].data,' ');
    until i=last;
end.

```

例 2 求 $p_1(x) + p_2(x)$ (两个多项式的和)

```

program dxshi;
type link=^node;
node=record
    coef:real;
    exp:integer;
    next:link
end;
poly=link;
var p, pa,pb:poly;
procedure jl(var a:poly);
var p,q:poly;
    co:real;ex:integer;

```

```

begin
  p:=nil;
  repeat
    read(co,ex);
    new(q);q^.coef:=co;q^.exp:=ex;q^.next:=p;p:=q;
  until (ex=-1) and (co=-1);
  a:=p;readln;
end;
procedure add_poly(var a:poly;b:poly);
var p,q,u,pre:poly;
    x:real;
begin
  p:=a^.next;q:=b^.next;
  pre:=a;
  while (p<>nil) and (q<>nil) do
    if p^.exp>q^.exp then begin pre:=p;p:=p^.next end
    else if p^.exp=q^.exp then begin
      x:=p^.coef+q^.coef;
      if x<>0 then begin p^.coef:=x;pre:=p;end
      else begin pre^.next:=p^.next; dispose(p) end;
      p:=pre^.next;u:=q;q:=q^.next;dispose(u);
    end
    else begin
      u:=q^.next;q^.next:=p;pre^.next:=q;pre:=q;q:=u
    end;
  if q<>nil then pre^.next:=q;
  dispose(b);
end;
begin
  jl(pa);jl(pb);
  add_poly(pa,pb);
  p:=pa;p:=p^.next;
  while p<>nil do
    begin
      writeln(p^.coef:8:2,p^.exp:5);
      p:=p^.next;
    end
  end.
end.

```

练习题：

1. 约瑟夫问题：

有 M 个猴子围成一圈，每个有一个编号，编号从 1 到 M。打算从中选出一个大王。经过协商，决定选大王的规则如下：从第一个开始，每隔 N 个，数到的猴子出圈，最后剩下的就是大王。

要求：从键盘输入 M，N，编程计算哪一个编号的猴子成为大王。（程序）

2. 求多项式的减与乘法。（程序）

第三章 栈

3.1 栈的概念（逻辑结构）及运算

3.2 栈的存储与实现

3.3 栈的应用

3.1 栈的概念及运算

栈的定义：栈是一种特殊的表这种表只在表头进行插入和删除操作。因此，表头对于栈来说具有特殊的意义，称为栈

顶。相应地，表尾称为栈底。不含任何元素的栈称为空栈。

栈的逻辑结构：假设一个栈 S 中的元素为 a_n, a_{n-1}, \dots, a_1 ，则称 a_1 为栈底元素， a_n 为栈顶元素。栈中的元素按 $a_1, a_2, \dots, a_{n-1}, a_n$ 的次序进栈。在任何时候，出栈的元素都是栈顶元素。换句话说，栈的修改是按后进先出的原则进行的，如图 1 所示。因此，栈又称为**后进先出 (Last In First Out) 表**，简称为 **LIFO 表**。所以，只要问题满足 LIFO 原则，就可以使用栈。

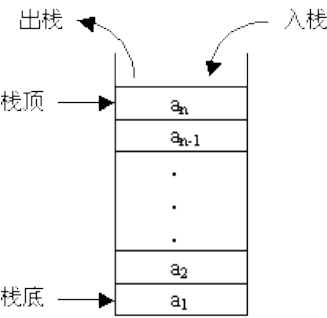


图 1

栈的运算：为一种抽象数据类型，常用的栈运算有：

| 运算 | 含义 |
|-------------|---|
| inistack(S) | 使 S 成为一个空栈。 |
| getTop(S) | 这是一个函数，函数值为 S 中的栈顶元素。 |
| Pop(S) | 从栈 S 中删除栈顶元素，简称为抛栈。 |
| Push(S,x) | 在 S 的栈顶插入元素 x，简称为将元素 x 入栈。 |
| Empty(S) | 这是一个函数。当 S 为空栈时，函数值为 true，否则函数值为 false。 |

3.2 栈的存储与实现

栈的数组实现：由于栈是一个特殊的表，我们可以用数组来实现栈。考虑到栈运算的特殊性，我们用一个数组 `elements[1..maxlength]` 来表示一个栈时，将栈底固定在数组的底部，即 `elements[1]` 为最早入栈的元素，并让栈向数组上方(下标增大的方向)扩展。同时，我们用一个游标 `top` 来指示当前栈顶元素所在的单元。当 `top=0` 时，表示这个栈为一个空栈。在一般情况下，`elements` 中的元素序列 `elements[top], elements[top-1], ..., elements[1]` 就构成了一个栈。这种结构如图 2 所示。

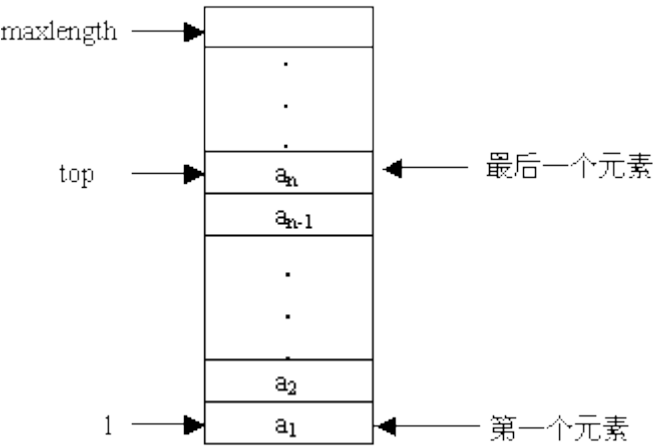


图 2

利用上述结构，我们可以形式地定义栈类型 TStack 如下：

```
Type
  TStack=Record
    top:integer;
    element:array[1..maxlength] of TElement;
  End;
```

在这种表示法下，栈的 5 种基本运算可实现如下。

```
procedure inistack(Var S:TStack);
begin
  S.top:=0;
end;

function Empty(var S:Stack):Boolean;
begin
  return(S.top=0);
end;

function Top(var S:TStack):TElement;
begin
  if Empty(S) then Error('The stack is empty.')
    else return(S.element[S.top]);
end;

procedure Pop(var S:TStack);
begin
  if Empty(S) then Error('The stack is empty.')
    else dec(S.top); {S.top 减 1}
end;

procedure Push(var S:TStack; x:TElement);
begin
  if S.top=maxlength then Error('The stack is full.')
    else begin
      inc(S.top); {S.top 增 1}
      S.elements[S.top]:=x;
    end;
end;
```

以上每种操作的复杂性为 $O(1)$ 。

在一些问题中，可能需要同时使用多个同类型的栈。为了使每个栈在算法运行过程中不会溢出， 要为每个栈顶置一个较大的栈空间。这样做往往造成空间的浪费。实际上，在算法运行的过程中，各个栈一般不会同时满，很可能有的满而有的空。因此，如果我们让多个栈共享同一个数组，动态地互相调剂，将会提高空间的利用率，并减少发生栈上溢的可能性。 假设我们让程序中的两个栈共享一个数组 $S[1..n]$ 。利用栈底位置不变的特性，我们可以将两个栈的栈底分别设在数组 S 的两端，然后各自向中间伸展，如图 3 所示。这两个 S 栈的栈顶初值分别为 0 和 $n+1$ 。只有当两个栈的栈顶相遇时才可能发生上溢。由于两个栈之间可以余缺互补，因此每个栈实际可用的最大空间往往大于 $n/2$ 。

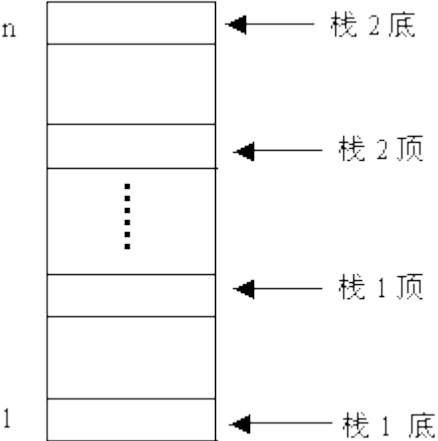


图 3

3.3 栈的应用
1. 表达式的求值

问题:能否设计算法,编制一个程序,让计算机扫描如下表达式,并将其值打印出来。

3 * (4 + 8) / 2 -5

注:给表达式设置#,标志扫描的开始和结束。

提示算法:设两个栈,一个是操作数栈,用来存放操作数,如 3、4、8 等,另一个是运算符栈,用来存放运算符。

首先将标志“#”进运算符栈的栈底。

然后依次扫描,按照栈的后进先出原则进行:

(1) 遇到操作数,进操作数栈;

(2) 遇到运算符时,则需将此运算符的优先级与栈顶运算符的优先级比较,

若高于栈顶元素则进栈,继续扫描下一符号,

否则,将运算符栈的栈顶元素退栈,形成一个操作码 Q,同时操作数栈的栈顶元素两次退栈,形成两个操作数 a、b,让计算机对操作数与操作码完成一次运算操作,即 aQb,并将其运算结果存放在操作数栈中……

模拟计算机处理算术表达式过程。从键盘上输入算术表达式串(只含+、-、×、÷运算符,允许含括号),输出算术表达式的值。设输入的表达式串是合法的。

附源程序:

```
program exsj_1;
const
  max=100;
var
  number:array[0..max] of integer;
  symbol:array[1..max] of char;
  s,t:string;
  i,p,j,code:integer;
procedure push;{算符入栈运算}
begin
  inc(p);symbol[p]:=s[i];
end;
procedure pop;{运算符栈顶元素出栈,并取出操作数栈元素完成相应的运算}
begin
  dec(p);
  case symbol[p+1] of
    '+':inc(number[p],number[p+1]);
    '-':dec(number[p],number[p+1]);
    '*':number[p]:=number[p]*number[p+1];
    '/':number[p]:=number[p] div number[p+1];
  end;
end;
function can:boolean;{判断运算符的优先级别,建立标志函数}
begin
  can:=true;
  if (s[i] in ['+', '-']) and (symbol[p]<>'(') then exit;
  if (s[i] in ['*', '/']) and (symbol[p] in ['*', '/']) then exit;
  can:=false;
end;
begin
  write('String : '); readln(s); s:='('+s+')'; i:=1; p:=0;
  while i<=length(s) do
    begin
      while s[i]='(' do {左括号处理}
        push; inc(i);
```

```

end;
  j:=i;
  repeat {取数入操作数栈}
    inc(i);
  until (s[i]<'0') or (s[i]>'9');
  t:=copy(s,j,i-j); val(t,number[p],code);
  repeat
    if s[i]=')' then {右括号处理}
begin
  while symbol[p]<>'(' do pop;
  dec(p); number[p]:=number[p+1];
end
  else
begin {根据标志函数值作运算符入栈或出栈运算处理}
  while can do pop;
  push;
end;
inc(i);
  until (i>length(s)) or (s[i-1]<>')');
  end;
  write('Result=',number[0]);
  readln;
end.

```

2. 背包问题

问题：假设有 n 件质量分配为 w_1, w_2, \dots, w_n 的物品和一个最多能装载总质量为 T 的背包，能否从这 n 件物品中选择若干件物品装入背包，使得被选物品的总质量恰好等于背包所能装载的最大质量，即 $w_{i1}+w_{i2}+\dots+w_{ik}=T$ 。若能，则背包问题有解，否则无解。

算法思想：首先将 n 件物品排成一列，依次选取；若装入某件物品后，背包内物品的总质量不超过背包最大装载质量时，则装入（进栈）；否则放弃这件物品的选择，选择下一件物品试探，直至装入的物品总和正好是背包的最大转载质量为止。这时我们称背包装满。

若装入若干物品的背包没有满，而且又无其他物品可以选入背包，说明已装入背包的物品中有不合格者，需从背包中取出最后装入的物品（退栈），然后在未装入的物品中挑选，重复此过程，直至装满背包（有解），或无物品可选（无解）为止。

具体实现：设用数组 $weight[1..N]$ ， $stack[1,N]$ 分别存放物品重量和已经装入背包（栈）的物品序号， $MaxW$ 表示背包的最大装载量。每进栈一个物品，就从 $MaxW$ 中减去该物品的质量，设 i 为待选物品序号，若 $MaxW-weight[i] \geq 0$ ，则该物品可选；若 $MaxW-weight[i] < 0$ ，则该物品不可选，且若 $i > n$ ，则需退栈，若此时栈空，则说明无解。

用 Pascal 实现的参考函数：

```

Function knapstack(n,MaxW,weight);
begin
  top:=0; i:=1; {i 为待选物品序号}
  while (MaxW>0) and ( i < = n ) do
  begin
    if (MaxW-weight[i]>=0) and ( i < = n ) then
begin top:=top+1; stack[top]:=i;MaxW=MaxW-weight[i] end;
{第 i 件物品装入背包}
    if MaxW=0 then return(true)
else begin
if (i=n) and (top>0) then {背包内有不合适物品}
  begin {取栈顶物品，恢复 MaxW 的值}
    i:=stack[top]; top:=top-1;MaxW=MaxW+weight[i];
    if top>0 then begin

```

```
i:=stack[top]; top:=top-1;MaxW=MaxW+weight[i];
end;
end;
i:=i+1;
end;
end;
return(false) {问题无解}
end;
```

练习:完整完成背包问题

第四章 队列

- 4.1 队列的概念（逻辑结构）及运算
- 4.2 队列的存储与实现
- 4.3 队列的应用

4.1 队列的概念及运算

队列的定义：
队列是一种特殊的线性表，对这种线性表，删除操作只在表头(称为队头)进行，插入操作只在表尾(称为队尾)进行。队列的修改是按先进先出的原则进行的，所以队列又称为先进先出(First In First Out)表，简称 **FIFO** 表。
队列的数学性质：
假设队列为 a_1, a_2, \dots, a_n ，那么 a_1 就是队头元素， a_n 为队尾元素。队列中的元素是按 a_1, a_2, \dots, a_n 的顺序进入的，退出队列也只能按照这个次序依次退出。也就是说，只有在 a_1 离开队列之后， a_2 才能退出队列，只有在 a_1, a_2, \dots, a_{n-1} 都离开队列之后， a_n 才能退出队列。图 1 是队列的示意图。

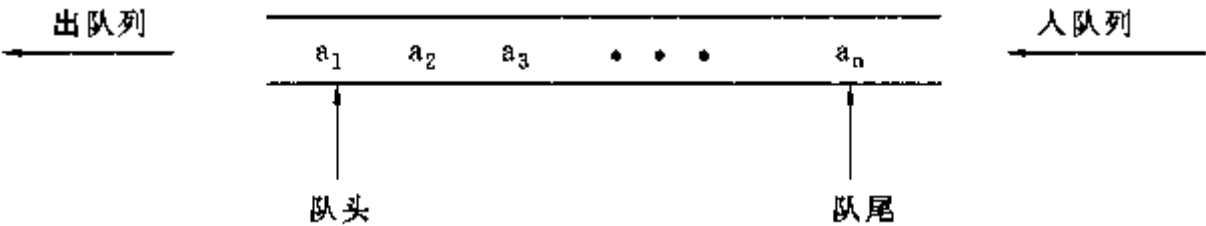


图 1

队列的运算：
为一种抽象数据类型，常用的队列运算有：

| 运算 | 含义 |
|--------------|---|
| Gethead(Q) | 这是一个函数，函数值返回队列 Q 的队头元素。 |
| Enqueue(Q,x) | 将元素 x 插入队列 Q 的队尾。此运算也常简称为将元素 x 入队。 |
| Dlqueue(Q) | 将 Q 的队头元素删除，简称为出队。 |
| Empty(Q) | 这是一个函数，若 Q 是一个空队列，则函数值为 true，否则为 false。 |
| Clear(Q) | 使队列 Q 成为空队列。 |

4.2 队列的存储与实现

队列是一种特殊的表，因此凡是可以用来实现表的数据结构都可以用来实现队列。不过，队列的中的元素的个数通常不固定，因此常用循环数组和链表两种方法实现队列。
链队列：
用指针实现队列得到的实际上是一个单链表。由于入队在队尾进行，所以用一个指针来指示队尾可以使入队操作不必从头到尾检查整个表，从而提高运算的效率。另外，指向队头的指针对于 Gethead 和 Dequeue 运算也是必要的。为了便于表示空队列，我们仍使用一个表头单元，将队头指针指向表头单元。当队头和队尾指针都指向表头单元时，表示队列为一个空队列。

用指针实现队列时，单元类型及队列类型可说明如下：

```

type
queueptr=^queuenode;
queuenode=record
    data:elemtp;
    next:queueptr;
end;
linkedquetp=record
    front,rear:queueptr;
end;

```

其中 front 为队头指针，rear 为队尾指针。图 2 是用指针表示队列的示意图。

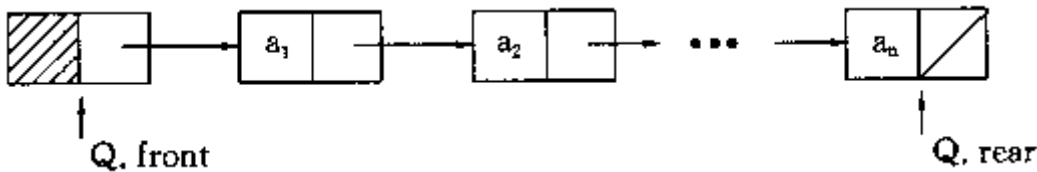


图 2

面我们来讨论队列的 5 种基本运算。

函数 Gethead(Q)

功能：这是一个函数，函数值返回队列 Q 的队头元素。

实现

```

Function Gethead(var Q:linkedquetp):elemtp;
begin
    if Empty(Q) then error('The queue is empty!')
    else return(Q.front^.next^.data);
end;

```

函数 Enqueue(Q, x)

功能：将元素 x 插入队列 Q 的队尾。此运算也常简称为将元素 x 入队。

实现

```

Procedure Enqueue(x:elemtp;var Q:linkedquetp);
begin
    new(Q.rear^.next);
    Q.rear:=Q.rear^.next;
    Q.rear^.data:=x;
    Q.rear^.next:=nil;
end;

```

函数 Empty(Q)

功能：这是一个函数，若 Q 是一个空队列，则函数值为 true，否则为 false。

实现

```

Function Empty(var Q:QueueType):Boolean;
begin
    return(Q.front=Q.rear);
end;

```

函数 Dequeue(Q)

功能：将 Q 的队头元素删除，简称为出队。

实现

```

Procedure Dequeue(var Q:linkedquetp);
var
p:queueptr;
begin

```

```

if Empty(Q) then Error('The queue is empty!')
else begin
    p:=Q.front;
    Q.front:=Q.front^.next;
    dispose(p);
end;
end;

```

函数 Clear(Q)

功能：使队列 Q 成为空队列。

实现

```

Procedure clear(var Q:linkedquetype);
begin
    Q.rear:=Q.front;
    while Q.front<>nil do
        begin
            Q.front:=Q.front^.next;
            dispose(Q.rear);
            Q.rear:=Q.front;
        end;
    new(Q.front);
    Q.front^.next:=nil;
    Q.rear:=Q.front;
end;

```

循环队列：

我们可以将队列当作一般的表用数组实现，但这样做的效果并不好。尽管我们可以用一个游标 last 来指示队尾，使得 Enqueue 运算可在 $O(1)$ 时间内完成，但是在执行 Dequeue 时，为了删除队头元素，必须将数组中其他所有元素都向前移动一个位置。这样，当队列中有 n 个元素时，执行 Dequeue 就需要 $O(n)$ 时间。为了提高运算的效率，我们用另一种方法来表达数组中各单元的位置关系。设想数组中的单元不是排成一行，而是围成一个圆环，我们将队列中从队头到队尾的元素按顺时针方向存放在循环数组中一段连续的单元中。当需要将新元素入队时，可将队尾游标 $Q.rear$ 按顺时针方向移一位，并在新的队尾游标指示的单元中存入新元素。出队操作也很简单，只要将队头游标 $Q.front$ 依顺时针方向移一位即可。容易看出，用循环数组来实现队列可以在 $O(1)$ 时间内完成 Enqueue 和 Dequeue 运算。执行一系列的入队与出队运算，将使整个队列在循环数组中按顺时针方向移动。通常，用队尾游标 $Q.rear$ 指向队尾元素所在的单元，用队头游标 $Q.front$ 指向队头元素所在单元的前一个单元，并且约定只能存放 $maxsize-1$ 个元素如图 3 所示。

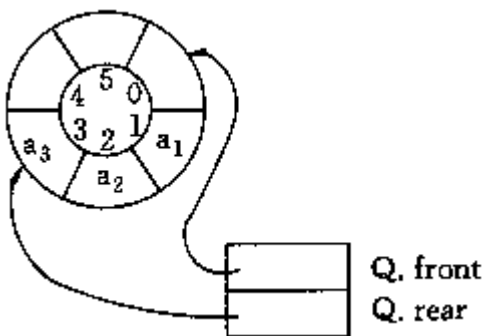


图 3

此时队列的定义如下：

```

const m=maxsize-1
type cyclicquetype=record
    elem:array[0..m] of elemtype;
    rear,front:0..m;
end;
var sq:cyclicquetype;

```

这时 当 $sq.rear=sq.front$ 时队空

当 $(sq.rear+1) \bmod maxsize=sq.front$ 时队满

先 $sq.rear=(sq.rear+1) \bmod maxsize$ 后进队
 先 $sq.front=(sq.front+1) \bmod maxsize$ 后出队
 队列中元素的个数 $(sq.rear-sq.front+maxsize) \bmod maxsize$

4.3 队列的应用

例 1: 一矩形阵列由数字 0 到 9 组成, 数字 1 到 9 代表细胞, 细胞的定义为沿细胞数字上下左右还是细胞数字则为同一细胞, 求给定矩形阵列的细胞个数。

如: 阵列

0234500067

1034560500

2045600671

0000000089

有 4 个细胞。

算法步骤:

1. 从文件中读入 $m \times n$ 矩阵阵列, 将其转换为 boolean 矩阵存入 bz 数组中;
2. 沿 bz 数组矩阵从上到下, 从左到右, 找到遇到的第一个细胞;
3. 将细胞的位置入队 h, 并沿其上、下、左、右四个方向上的细胞位置入队, 入队后的位置 bz 数组置为 FLASE;
4. 将 h 队的队头出队, 沿其上、下、左、右四个方向上的细胞位置入队, 入队后的位置 bz 数组置为 FLASE;
5. 重复 4, 直至 h 队空为止, 则此时找出了一个细胞;
6. 重复 2, 直至矩阵找不到细胞;
7. 输出找到的细胞数。

程序:

```
program xibao;
const dx:array[1..4] of -1..1=(-1,0,1,0);
      dy:array[1..4] of -1..1=(0,1,0,-1);
var int:text;
    name,s:string;
    pic:array[1..50,1..79] of byte;
    bz:array[1..50,1..79] of boolean;
    m,n,i,j,num:integer;
    h:array[1..4000,1..2] of byte;
procedure doing(p,q:integer);
var i,t,w,x,y:integer;
begin
  inc(num);bz[p,q]:=false;
  t:=1;w:=1;h[1,1]:=p;h[1,2]:=q;{遇到的第一个细胞入队}
  repeat
    for i:=1 to 4 do{沿细胞的上下左右四个方向搜索细胞}
      begin
        x:=h[t,1]+dx[i];y:=h[t,2]+dy[i];
        if (x>0) and (x<=m) and (y>0) and (y<=n) and bz[x,y]
          then begin inc(w);h[w,1]:=x;h[w,2]:=y;bz[x,y]:=false;end;{为细胞的入队}
        end;
      inc(t);{队头指针加1}
    until t>w;{直至队空为止}
  end;
begin
  fillchar(bz,sizeof(bz),true);num:=0;
  write('input file:');readln(name);
  assign(int,name);reset(int);
  readln(int,m,n);
```

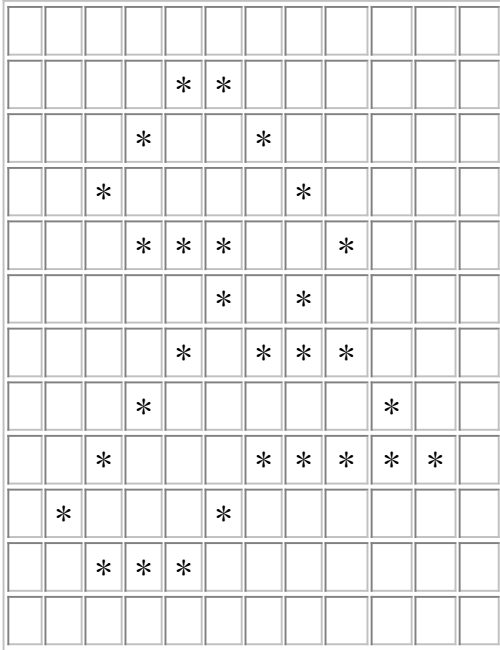


```

for i:=1 to m do
  begin readln(int,s);
  for j:=1 to n do
    begin pic[i,j]:=ord(s[j])-ord('0');
      if pic[i,j]=0 then bz[i,j]:=false;
    end;
  end;
close(int);
for i:=1 to m do
  for j:=1 to n do if bz[i,j] then doing(i,j);{在矩阵中寻找细胞}
writeln('NUMBER of cells=',num);readln;
end.

```

练习：如下图：求图中被*围成的封闭区域的面积（方格的个数不包括*所在的方格，且*不在最外一层）。



第五章 树和二叉树

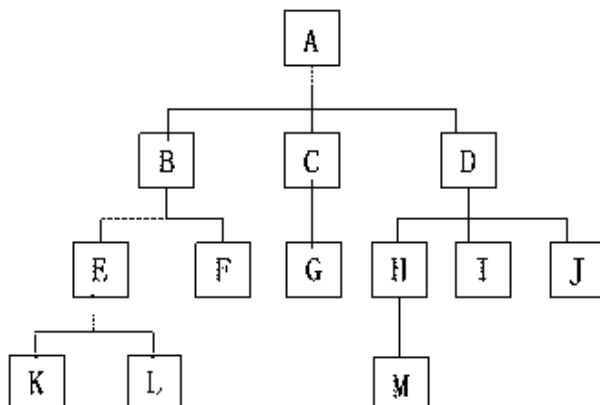
5.1 树的概念

5.2 二叉树

5.3 二叉树的应用

5.1 树的概念

树的递归定义如下：（1）至少有一个结点（称为根）（2）其它是互不相交的子树



1. 树的度——也即是宽度，简单地说，就是结点的分支数。以组成该树各结点中最大的度作为该树的度，如上图的树，其度为3；树中度为零的结点称为叶结点或终端结点。树中度不为零的结点称为分枝结点或非终端结点。除根结点外的分枝结点统称为内部结点。

2. 树的深度——组成该树各结点的最大层次，如上图，其深度为4；

3. 森林——指若干棵互不相交的树的集合，如上图，去掉根结点 A，其原来的二棵子树 T1、T2、T3 的集合 {T1, T2, T3} 就为森林；

4. 有序树——指树中同层结点从左到右有次序排列，它们之间的次序不能互换，这样的树称为有序树，否则称为无序树。

5. 树的表示

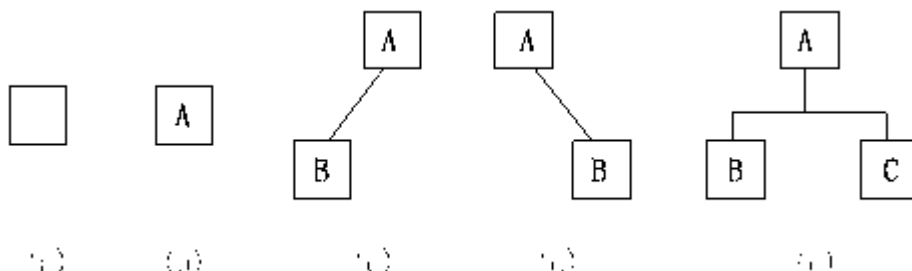
树的表示方法有许多，常用的方法是用括号：先将根结点放入一对圆括号中，然后把它的子树由左至右的顺序放入括号中，而对子树也采用同样的方法处理；同层子树与它的根结点用圆括号括起来，同层子树之间用逗号隔开，最后用闭括号括起来。如上图可写成如下形式：

(A (B (E (K, L), F), C (G), D (H (M), I, J)))

5. 2 二叉树

1. 二叉树的基本形态：

二叉树也是递归定义的，其结点有左右子树之分，逻辑上二叉树有五种基本形态：



(1) 空二叉树——(a)；

(2) 只有一个根结点的二叉树——(b)；

(3) 右子树为空的二叉树——(c)；

(4) 左子树为空的二叉树——(d)；

(5) 完全二叉树——(e)

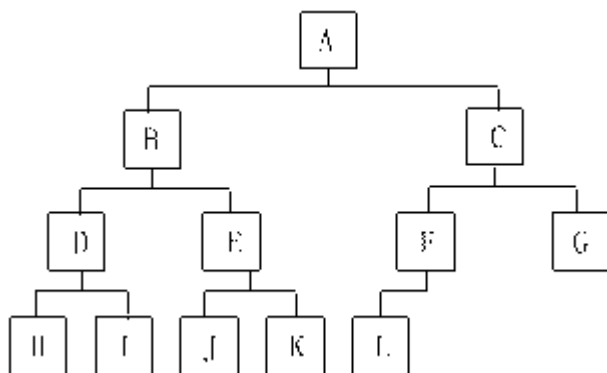
注意：尽管二叉树与树有许多相似之处，但二叉树不是树的特殊情形。

2. 两个重要的概念：

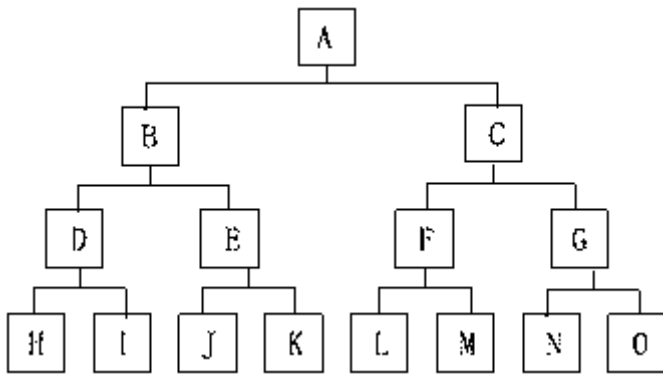
(1) 完全二叉树——只有最下面的两层结点度小于 2，并且最下面一层的结点都集中在该层最左边的若干位置的二叉树；

(2) 满二叉树——除了叶结点外每一个结点都有左右子女且叶结点都处在最底层的二叉树，。

如下图：



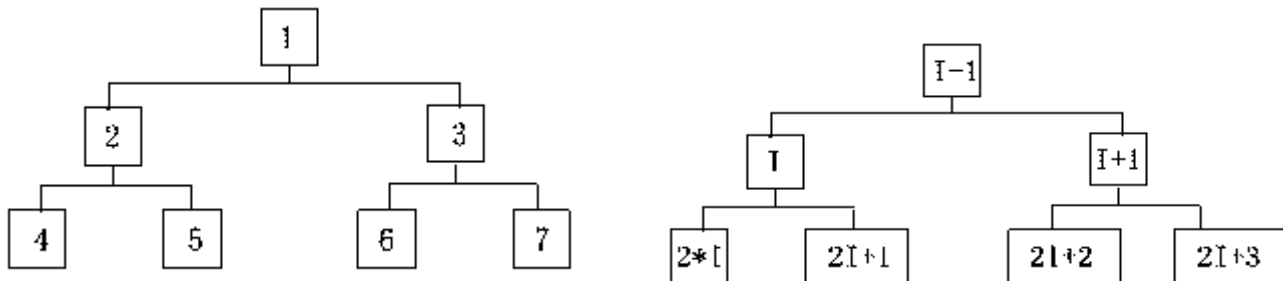
完全二叉树



满二叉树

3. 二叉树的性质

- (1) 在二叉树中，第 i 层的结点总数不超过 2^{i-1} ；
- (2) 深度为 h 的二叉树最多有 2^h-1 个结点 ($h \geq 1$)，最少有 h 个结点；
- (3) 对于任意一棵二叉树，如果其叶结点数为 N_0 ，而度数为 2 的结点总数为 N_2 ，
则 $N_0 = N_2 + 1$ ；
- (4) 具有 n 个结点的完全二叉树的深度为 $\text{int}(\log_2 n) + 1$
- (5) 有 N 个结点的完全二叉树各结点如果用顺序方式存储，则结点之间有如下关系：



若 I 为结点编号则 如果 $I > 1$ ，则其父结点的编号为 $I/2$ ；
 如果 $2*I \leq N$ ，则其左儿子（即左子树的根结点）的编号为 $2*I$ ；若 $2*I > N$ ，则无左儿子；
 如果 $2*I+1 \leq N$ ，则其右儿子的结点编号为 $2*I+1$ ；若 $2*I+1 > N$ ，则无右儿子。

4. 二叉树的存储结构：

(1) 顺序存储方式

```

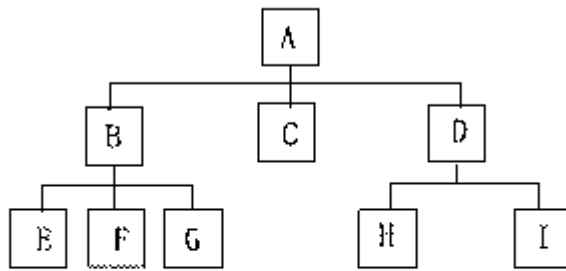
type node=record
    data:datatype
    l,r:integer;
end;
var tr:array[1..n] of node;
  
```

(2) 链表存储方式，如：

```

type btree=^node;
node=record
    data:datatype;
    lchild,rchild:btree;
end;
  
```

5. 普通树转换成二叉树：凡是兄弟就用线连起来，然后去掉父亲到儿子的连线，只留下父母到其第一个子女的连线。



6. 二叉树的遍历运算（递归定义）

(1) 先序遍历

访问根；按先序遍历左子树；按先序遍历右子树

(2) 中序遍历

按中序遍历左子树；访问根；按中序遍历右子树

(3) 后序遍历

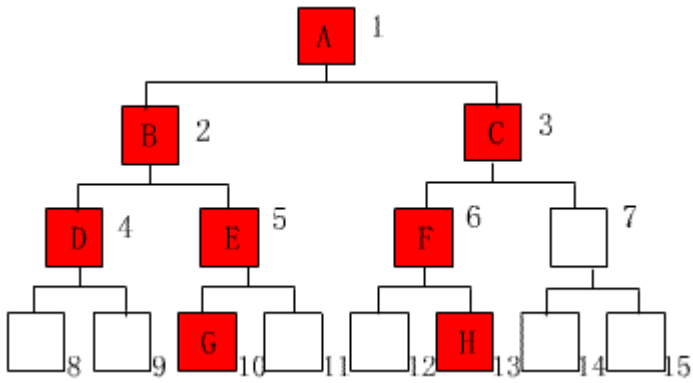
按后序遍历左子树；按后序遍历右子树；访问根

例 1. 用顺序存储方式建立一棵有 31 个结点的满二叉树，并对其进行先序遍历。

```

program erchashu1;
var b:array[1..31] of char;
    e:array[1..63] of byte;
    n,h,i,k:integer;
procedure tree(t:integer);
begin
  if e[t]=0 then exit
  else
    begin
      write(b[t]);e[t]:=0;
      t:=2*t;tree(t);
      t:=t+1;tree(t);
    end;
end;
begin
  repeat
    write('n=');readln(n);
  until (n>0) and (n<6);
  fillchar(e,sizeof(e),0);
  k:=trunc(exp(n*ln(2)))-1;
  for i:=1 to k do e[i]:=1;
  for i:=1 to 26 do b[i]:=chr(64+i);
  for i:=1 to 5 do b[26+i]:=chr(48+i);
  h:=1 ;tree(h);
  writeln;
end.
  
```

例 2. 用顺序存储方式建立一棵如图所示的二叉树，并对其进行先序遍历。



```

program tree1;
const n=15;
type node=record
    data:char;
    l,r:0..n;
end;
var tr:array[1..n] of node;
    e:array[1..n] of 0..1;
    i,j:integer;
procedure jtr;
var i:integer;
begin
    for i:=1 to n do
        with tr[i] do
            readln(data,l,r);
end;
procedure search(m:integer);
begin
    with tr[m] do
        begin
            write(data);
            if l<>0 then search(l);
            if r<>0 then search(r);
            end;
        end;
begin
    jtr;search(1);writeln;
end.

```

例 3 用链表存储方式生成上述二叉树，中序遍历之。

1. 将上述二叉树用广义表表示为 A(B(D, E(G)), C(F(, H)))
2. 根据广义表串（以#结束）生成二叉树。

```

program ltree;
const n=8;
type trlist=^node;
    node=record
        da:char;
        l,r:trlist;
    end;
var s:array[1..n] of trlist;
    p,root:trlist;
    ch:char;
    top,k:integer;

```

```

procedure creat(var head:trlist);
begin
  read(ch);
  top:=0;
  while ch<>'#' do
  begin
    case ch of
      'A'..'Z':begin new(p);p^.da:=ch;p^.l:=nil;p^.r:=nil;
                    if top<>0 then
                      case k of
                        1:s[top]^l:=p;
                        2:s[top]^r:=p;
                      end
                    end;
      '(':begin top:=top+1;s[top]:=p;k:=1;end;
      ')': top:=top-1;
      ',': k:=2;
    end;
    read(ch);
  end;
head:=s[1];
end;
procedure inorder(head:trlist);
begin
  if head^.l<>nil then inorder(head^.l);
  write(head^.da);
  if head^.r<>nil then inorder(head^.r);
end;
begin
  write('Input tree string:');
  creat(root);
  inorder(root);
end.

```

5.3 二叉树的应用

1. 哈夫曼树与哈夫曼码

树的路径长度：一棵树的每一个叶结点到根结点的路径长度的和。

带权二叉树：给树的叶结点赋上某个实数值（称叶结点的权）。

带权路径长度：各叶结点的路径长度与其权值的积的总和。

哈夫曼树（最优二叉树）：带权路径长度最小的二叉树。

如何构建哈夫树：（思想是：权越大离跟越近）

```

program gojiantree;
const n=4;m=7;
type node=record
  w:real;
  parent,lchild,rchild:0..m
end;
htree=array[1..m] of node;
var htreen:htree;
procedure gjtree(var ht:htree);
var i,j:integer;
    small1,small2:real;
    p1,p2:0..m;

```

```

begin
  for i:=1 to m do
    with ht[i] do
      begin
        w:=0;lchild:=0;rchild:=0;parent:=0;
        end;
      for i:=1 to n do read(ht[i].w);
      for i:=n+1 to m do
        begin
          p1:=0;p2:=0;
          small1:=1000;small2:=1000;
          for j:=1 to i-1 do
            if ht[j].parent=0 then
              if ht[j].w<small1 then
                begin small2:=small1;small1:=ht[j].w;p2:=p1;p1:=j end
              else if ht[j].w<small2 then begin small2:=ht[j].w;p2:=j end;
          ht[p1].parent:=i;
          ht[p2].parent:=i;
          ht[i].lchild:=p1;
          ht[i].rchild:=p2;
          ht[i].w:=ht[p1].w+ht[p2].w;
          end;
        end;
      begin
        gjtree(htree1);
      end.

```

哈夫曼码：哈夫曼树的非叶结点到左右孩子的路径分别用 0, 1 表示，从根到叶的路径序列即为哈夫曼码。

哈夫曼码是不会发生译码多义性的不等长编码，广泛应用实际中。

（原因是任何一字符的编码不是更长编码的前缀部分，为什么？）

2. 排序二叉树

排序二叉树：每一个参加排列的数据对应二叉树的一个结点，且任一结点如果有左（右）子树，则左（右）子树各结点的数据必须小（大）于该结点的数据。中序遍历排序二叉树即得排序结果。程序如下：

```

program pxtree;
const
  a:array[1..8] of integer=(10,18,3,8,12,2,7,3);
type point=^nod;
  nod=record
    w:integer;
    right,left:point ;
  end;
var root,first:point;k:boolean;i:integer;
procedure hyt(d:integer;var p:point);
begin
  if p=nil then
    begin
      new(p);
      with p^ do begin w:=d;right:=nil;left:=nil end;
      if k then begin root:=p; k:=false end;
    end
  else with p^ do if d>=w then hyt(d,right) else hyt(d,left);
end;
procedure hyt1(p:point);

```

```

begin
  with p^ do
    begin
      if left<>nil then hyt1(left);
      write(w:4);
      if right<>nil then hyt1(right);
    end
  end;
end;
begin
  first:=nil;k:=true;
  for i:=1 to 8 do hyt(a[i],first);
  hyt1(root);writeln;
end.

```

3. 堆排序

堆：设有数据元素的集合 $(R_1, R_2, R_3, \dots, R_n)$ 它们是一棵顺序二叉树的结点且有

$$R_i \leq R_{2i} \text{ 和 } R_i \leq R_{2i+1} (\text{或} \geq)$$

堆的性质：堆的根结点上的元素是堆中的最小元素，且堆的每一条路径上的元素都是有序的。

堆排序的思想是：

1) 建初始堆（将结点 $[n/2], [n/2]-1, \dots, 3, 2, 1$ 分别调成堆）

2) 当未排序完时

输出堆顶元素，删除堆顶元素，将剩余的元素重新建堆。

程序如下：

```

program duipx;
const n=8;
type arr=array[1..n] of integer;
var a:arr;i:integer;
procedure sift(var a:arr;l,m:integer);
  var i,j, t:integer;
  begin
    i:=l;j:=2*i;t:=a[i];
    while j<=m do
      begin
        if (j<m) and (a[j]>a[j+1]) then j:=j+1;
        if t>a[j] then
          begin a[i]:=a[j];i:=j;j:=2*i; end
        else exit;
      end;
    a[i]:=t;
  end;
begin
  for i:=1 to n do read(a[i]);
  for i:=(n div 2) downto 1 do
    sift(a,i,n);
  for i:=n downto 2 do
    begin
      write(a[1]:4);
      a[1]:=a[i];
      sift(a,1,i-1);
    end;
    writeln(a[1]:4);
  end.

```


第六章 图

- 6.1 图的概念
- 6.2 图的存储
- 6.3 图的遍历
- 6.4 图的应用

6.1 概念

图是由顶点 V 的集合和边 E 的集合组成的二元组记 $G = (V, E)$ 如下图是一无向图(顶点的前后顺序不限)

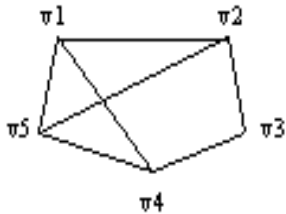


图 1

$V = \{V1, V2, V3, V4, V5\}$

$E = \{(V1, V2), (V2, V3), (V3, V4), (V4, V5), (V5, V1), (V2, V5), (V4, V1)\}$

下图是一有向图（顶点分先后顺序）

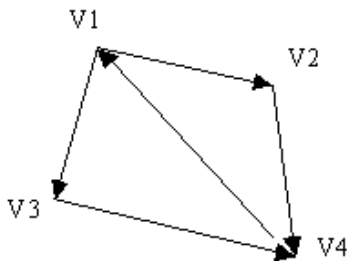


图 2

$V = \{V1, V2, V3, V4\}$

$E = \{ \langle V1, V2 \rangle, \langle V2, V4 \rangle, \langle V1, V3 \rangle, \langle V3, V4 \rangle, \langle V4, V1 \rangle \}$

完全图（每一对不同的顶点都有一条边相连, n 个顶点的完全图共有 $n(n-1)/2$ 条边）

顶点的度：与顶点关联的边的数目，有向图中等于该顶点的入度与出度之和。

入度——以该顶点为终点的边的数目和

出度——以该顶点为起点的边的数目和

度数为奇数的顶点叫做奇点，度数为偶数的点叫做偶点。

[定理 1] 图 G 中所有顶点的度数之和等于边数的 2 倍。因为计算顶点的度数时。每条边均用到 2 次。

[定理 2] 任意一个图一定有偶数个奇点。

6.2 图的存储

1. 邻接矩阵

1(或权值) 表示 顶点 i 和顶点 j 有边(i 和 j 的路程)

$A(i, j) = \{$

0 表示顶点 i 和顶点 j 无边

如图 1 的矩阵表示为

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

2. 邻接表

数组方法：

| 顶点 | 相邻顶点 | 邻接顶点数 |
|------|---------|-------|
| v[i] | d[i, j] | c[i] |

| | | | | | | |
|---|--|---|---|---|--|---|
| 1 | | 2 | 4 | 5 | | 3 |
| 2 | | 1 | 3 | 5 | | 3 |
| 3 | | 2 | 4 | | | 2 |
| 4 | | 1 | 3 | 5 | | 3 |
| 5 | | 1 | 2 | 4 | | 3 |

链表方法:

```

1  --> 2  --> 4  --> 5  ^
2  --> 1  --> 3  --> 5  ^
3  --> 2  --> 4  ^
4  --> 1  --> 3  --> 5  ^
5  --> 1  --> 2  --> 4  ^

```

3. 边目录

如图 2

| | | | | | |
|----|---|---|---|---|---|
| 起点 | 1 | 1 | 2 | 3 | 4 |
| 终点 | 2 | 3 | 4 | 4 | 1 |
| 长度 | | | | | |

6. 3 图的遍历

1. 深度优先遍历

遍历算法:

- 1) 从某一顶点出发开始访问, 被访问的顶点作相应的标记, 输出访问顶点号.
- 2) 从被访问的顶点出发, 搜索与该顶点有边的关联的某个未被访问的邻接点

再从该邻接点出发进一步搜索与该顶点有边的关联的某个未被访问的邻接点, 直到全部接点访问完毕. 如图 1 从 V1 开始的深度优先遍历序列为 V1, V2, V3, V4, V5. 图 2 从 V1 开始的深度优先遍历序列为 V1, V2, V4, V3.

算法过程:

```

procedure shendu(i);
begin
  write(i);
  v[i]:=true;
  for j:=1 to n do
    if (a[i,j]=1) and not(v[j]) then shendu(j);
end;

```

2. 广度优先遍历

遍历算法:

- 1) 从某个顶点出发开始访问, 被访问的顶点作相应的标记, 并输出访问顶点号;
- 2) 从被访问的顶点出发, 依次搜索与该顶点有边的关联的所有未被访问的邻接点, 并作相应的标记。
- 3) 再依次根据 2) 中所有被访问的邻接点, 访问与这些邻接点相关的所有未被访问的邻接点, 直到所有顶点被访问为止。如图 3 的广度优先遍历序列为 C1, C2, C3, C4, C5, C6。

算法过程:

```

procedure guangdu(i);
begin
  write(i);
  v[i]:=true;
  i 进队;
  repeat

```

```

队首元素出队设为 k
for j:=1 to n do
  if (a[k,j]=1) and (not v[j]) then
    begin
      write(j);
      v[j]:=true;
      j 进队;
    end;
until 队列 q 为空;

```

6.3 图的应用

例 1: 有 A, B, C, D, E 5 本书, 要分给张、王、刘、赵、钱 5 位同学, 每人只选一本。每人将喜爱的书填写下表, 输出每人都满意的分书方案。

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 张 | | | 1 | 1 | |
| 王 | 1 | 1 | | | 1 |
| 刘 | | 1 | 1 | | |
| 赵 | | | | 1 | |
| 钱 | | 1 | | | 1 |

用递归方式程序如下:

```

program allotbook;
type five=1..5;
const like:array[five,five] of 0..1=
  ((0,0,1,1,0),(1,1,0,0,1),(0,1,1,0,0),(0,0,0,1,0),(0,1,0,0,1));
name:array[five] of string[5]=('zhang','wang','liu','zhao','qian');
var book:array[five] of five;
flag:set of five;
c:integer;
procedure print;
var i:integer;
begin
  inc(c);
  writeln('answer',c,':');
  for i:=1 to 5 do
    writeln(name[i]:10,': ',chr(64+book[i]));
end;
procedure try(i:integer);
var j:integer;
begin
  for j:=1 to 5 do
    if not(j in flag) and (like[i,j]>0) then
      begin flag:=flag+[j];
        book[i]:=j;
        if i=5 then print else try(i+1);
        flag:=flag-[j];
      end
  end;
end;
begin
  flag:=[];
  c:=0;
  try(1);

```

```

readln;
end.

```

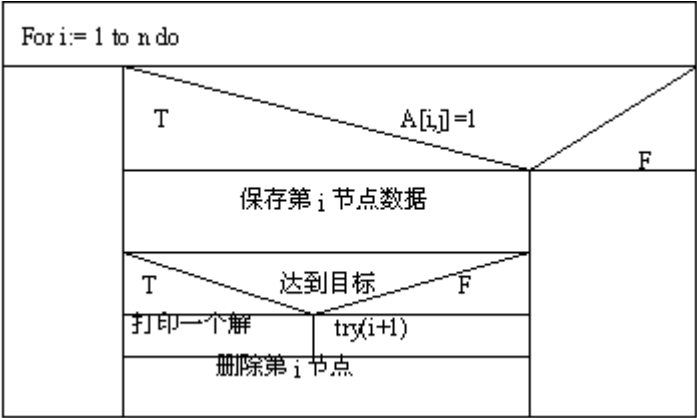
用非递归方法编程如下:

```

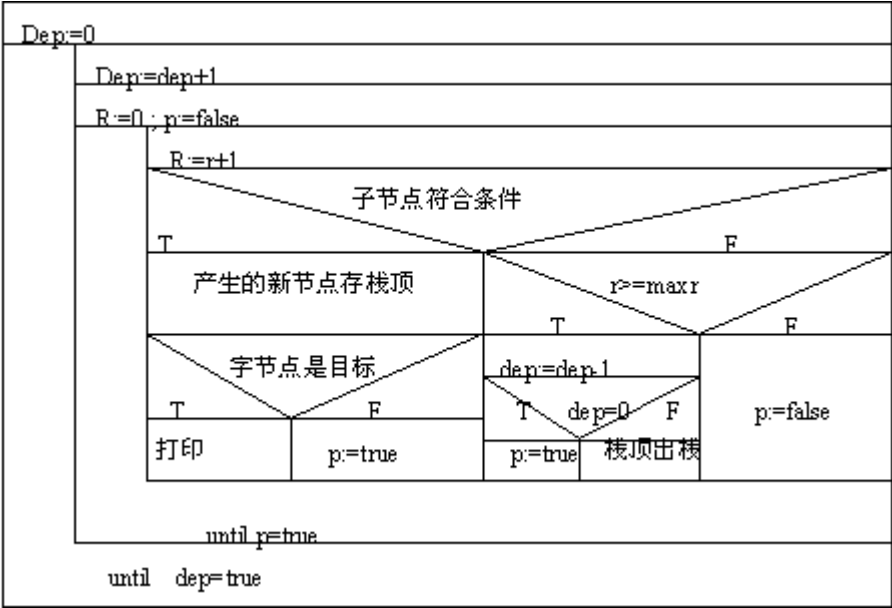
program allotbook;
type five=1..5;
const like:array[five,five] of 0..1=
  ((0,0,1,1,0),(1,1,0,0,1),(0,1,1,0,0),(0,0,0,1,0),(0,1,0,0,1));
name:array[five] of string[5]=('zhang','wang','liu','zhao','qian');
var book:array[five] of five;
    flag:set of five;
    c,dep,r:integer;
    p:boolean;
procedure print;
var i:integer;
begin
  inc(c);
  writeln('answer',c,':');
  for i:=1 to 5 do
    writeln(name[i]:10,':',chr(64+book[i]));
  end;
begin
  flag:=[];
  c:=0;dep:=0;
  repeat
    dep:=dep+1;
    r:=0;
    p:=false;
    repeat
      r:=r+1;
      if not(r in flag) and (like[dep,r]>0) then
        begin
          flag:=flag+[r];
          book[dep]:=r;
          if dep=5 then print else p:=true
        end
      else if r>=5 then
        begin
          dep:=dep-1;
          if dep=0 then p:=true else begin r:=book[dep];flag:=flag-[r] end;
        end else p:=false;
    until p=true;
until dep=0;
  readln;
end.

```

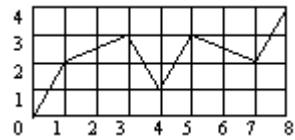
一般的情况的递归方法流程图如下:



非递归方法的流程图如下：



例 2：中国象棋棋盘如下图马自左下角往右上角跳. 规定只许往左跳, 不许往右跳(如图跳法)编程找出所有跳法。



递归算法如下：

```
program tiaoma;
const
di:array[1..4] of integer=(1,2,2,1);
dj:array[1..4] of integer=(2,1,-1,-2);
var x,y:array[0..20] of integer;
c:integer;
procedure print(dep:integer);
var j:integer;
begin
c:=c+1;write(c,':');
for j:=0 to dep-1 do write('(',x[j],',',y[j],')->');
writeln('(',x[dep],',',y[dep],')');
end;
procedure try(dep,i,j:integer);
var r,ni,nj:byte;
begin
for r:=1 to 4 do
```

```

begin
  ni:=i+di[r];
  nj:=j+dj[r];
  if (ni>0)and(ni<=8)and(nj>=0)and(nj<=4) then
    begin
      x[dep]:=ni;y[dep]:=nj ;
      if (ni=8) and (nj=4) then print(dep) else  try(dep+1,ni,nj)
    end;
  end;
end;
begin
  x[0]:=0;y[0]:=0;
  c:=0;
  try(1,0,0);
  readln;
end.

```

非递归算法如下:

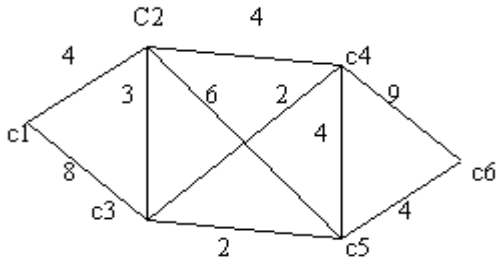
```

program tiaom;
const
  di:array[1..4] of integer=(1,2,2,1);
  dj:array[1..4] of integer=(2,1,-1,-2);
var x,y:array[0..20] of integer;
  b:array[0..20] of 0..4;
  c,r,ni,nj,dep:integer;
  p:boolean;
procedure print(dep:integer);
  var j:integer;
  begin
    c:=c+1;write(c,':');
    for j:=0 to dep-1 do write('(',x[j],',',y[j],')->');
    writeln('(',x[dep],',',y[dep],')');
  end;
begin
  x[0]:=0;y[0]:=0;c:=0;
  dep:=0;
  repeat
    dep:=dep+1;
    r:=0;p:=false;
    repeat
      r:=r+1;
      ni:=x[dep-1]+di[r];
      nj:=y[dep-1]+dj[r];
      if (ni>0)and(ni<=8)and(nj>=0)and(nj<=4) then
        begin
          x[dep]:=ni;y[dep]:=nj ;b[dep]:=r;
          if (ni=8) and (nj=4) then print(dep) else  p:=true;
        end
      else if r>=4 then begin dep:=dep-1;if dep=0 then p:=true  else r:=b[dep]end
        else p:=false;
    until p=true;
  until dep=0 ;
  readln;

```

end.

例 3 :找出图 3: C1 到 C6 的一条经过点最少的路径并求出其路程总长度。



程序如下:

```

program tu3bfs;
type fg=set of 1..6;
const link:array[1..5,1..6] of integer=((0,4,8,0,0,0),
(4,0,3,4,6,0),(8,3,0,2,2,0),(0,4,2,0,4,9),(0,6,2,4,0,4));
var pnt,city:array[1..10] of 0..6;
flag:fg;
r,k,head,tail:integer;
procedure print;
var n, i,cost,y:integer;
s:array[1..7] of 1..6;
begin
y:=tail;n:=0; cost:=0;
while y>0 do begin inc(n);s[n]:=y;y:=pnt[y] end;
writeln('minpath=',n-1);
write('1');
for i:=n-1 downto 1 do
begin
write('->',s[i]);
cost:=cost+link[s[i+1],s[i]];
end;
writeln;
writeln('cost=',cost);
end;
begin
flag:=[1];
pnt[1]:=0; city[1]:=1;
head:=0;tail:=1;
repeat
head:=head+1;
k:=city[head];
for r:=2 to 6 do
if not(r in flag) and (link[k,r]>0) then
begin
inc(tail);city[tail]:=r;
pnt[tail]:=head;
flag:=flag+[r];
if r=6 then begin print;halt end;
end;
until head>=tail;
readln;

```

end.

例 4: 8 数码难题 : 2 8 3 1 2 3
 1 6 4 → 8 4 (用最少的步数)
 7 5 7 6 5

程序如下:

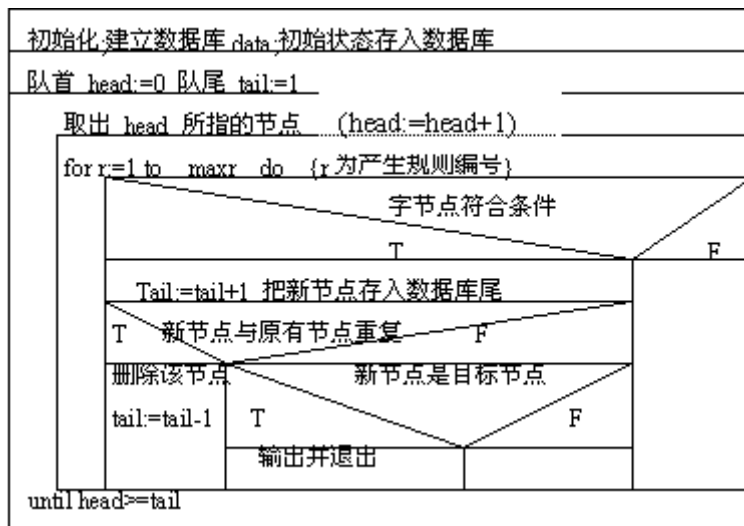
```
program num8;
type a33=array[1..3,1..3] of 0..8;
     a4=array[1..4] of -1..1;
node=record
  ch:a33;
  si,sj:1..3;
  pnt,dep:byte;
end;
const goal:a33=((1,2,3),(8,0,4),(7,6,5));
      start:a33=((2,8,3),(1,6,4),(7,0,5));
      di:a4=(0,-1,0,1);
      dj:a4=(-1,0,1,0);
var data:array[1..100] of node;
    temp:node;
    r,k,ni,nj,head,tail,depth:integer;
function check(k:integer):boolean;
begin
  ni:=temp.si+di[k];nj:=temp.sj+dj[k];
  if (ni in [1..3]) and (nj in [1..3]) then check:=true else check:=false;
end;
function dupe:boolean;
var i,j,k:integer;
    buf:boolean;
begin
  buf:=false;i:=0;
  repeat
    inc(i);buf:=true;
    for j:=1 to 3 do
      for k:=1 to 3 do
        if data[i].ch[j,k]<>data[tail].ch[j,k] then buf:=false;
      until buf or (i>=tail-1);
    dupe:=buf;
  end;
function goals:boolean;
var i,j:byte;
begin
  goals:=true;
  for i:=1 to 3 do
    for j:=1 to 3 do
      if data[tail].ch[i,j]<>goal[i,j] then goals:=false;
    end;
  procedure print;
  var buf:array[1..100] of integer;
      i,j,k,n:integer;
  begin
    n:=1;
    i:=tail;buf[1]:=i;
```



```

repeat
  inc(n);buf[n]:=data[i].pnt;
  i:=data[i].pnt;
until i=0;
writeln('steps=',depth+1);
for i:=1 to 3 do
  begin
    for k:=n-1 downto 1 do
      begin
        for j:=1 to 3 do write(data[buf[k]].ch[i,j]);
        if (i=2) and (k<>1) then write('->') else write(' ');
      end;
    writeln;
  end;
  readln;halt
end;
begin
  head:=0;tail:=1;
  with data[1] do
    begin
      ch:=start;si:=3;sj:=2;pnt:=0;dep:=0;
    end;
  repeat
    inc(head);temp:=data[head];
    depth:=temp.dep;
    for r:=1 to 4 do
      if check(r) then
        begin
          inc(tail);data[tail]:=temp;
          with data[tail] do
            begin
              ch[si,sj]:=ch[ni,nj];
              ch[ni,nj]:=0;si:=ni;sj:=nj;
              pnt:=head;
              dep:=depth+1;
            end;
          if dupe then dec(tail) else if goals then print;
        end;
    until head>=tail;
    writeln('no solution');readln;
  end.

```



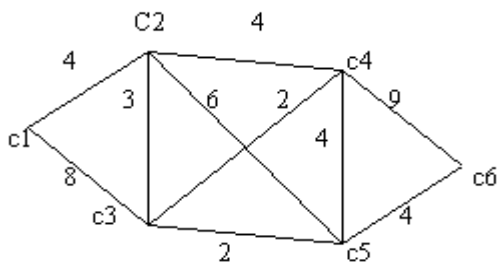
练习：

1. 覆盖问题。设有边长为 N (N 为偶数) 的正方形用 $N \times N/2$ 个长为 2，宽为 1 的长方形覆盖，打印所有覆盖方案。

如 $N=4$

| | | | | | | | | |
|---|---|---|---|--|---|---|---|---|
| 1 | 2 | 2 | 4 | | 1 | 1 | 2 | 2 |
| 1 | 3 | 3 | 4 | | 3 | 3 | 4 | 4 |
| 5 | 6 | 6 | 8 | | 5 | 5 | 6 | 6 |
| 5 | 7 | 7 | 8 | | 7 | 7 | 8 | 8 |

2. 用深度优先法找出 $C1$ 到 $C6$ 的没有重复城市的所有不同路径及路程总长度。



3. 骑士周游世界。在国际象棋的棋盘上，有一位骑士按照国际象棋中马的行走规则从棋盘上的某一方格出发，开始在棋盘上周游。若能不重复地走遍棋盘上的每一个方格，这样的一条周游路线在数学上被称之为国际象棋盘上马的哈密尔顿链。请你设计一个程序，对于从键盘输入的任意一个起始方格的坐标，由计算机自动寻找并按如下格式打印出国际象棋盘上马的哈密尔顿链。例如，当马从坐标点 (5, 8) 出发时，相应的哈密尔顿链如下图所示：

| | | | | | | | |
|----|----|----|----|----|----|----|----|
| 60 | 11 | 26 | 29 | 54 | 13 | 24 | 21 |
| 27 | 30 | 61 | 12 | 25 | 22 | 51 | 14 |
| 10 | 59 | 28 | 55 | 50 | 53 | 20 | 23 |
| 31 | 64 | 57 | 62 | 43 | 48 | 15 | 52 |
| 58 | 9 | 32 | 49 | 56 | 19 | 42 | 1 |
| 33 | 6 | 63 | 44 | 47 | 36 | 39 | 16 |

| | | | | | | | |
|---|----|---|----|----|----|----|----|
| 8 | 45 | 4 | 35 | 18 | 41 | 2 | 37 |
| 5 | 34 | 7 | 46 | 3 | 38 | 17 | 40 |

4. 翻币问题。有 N 个硬币 ($N \geq 6$), 正面朝上排成一排, 每次将 5 个硬币翻过来放在原来的位置, 直到最后全部硬币翻成反面朝上为止。输出最少需要的步数。

5. 分酒问题: 有一酒瓶装有 8 斤酒, 没有量器, 只有分别装 5 斤和 3 斤的空酒瓶。设计一程序将 8 斤酒分成两个 4 斤, 并以最少的步骤给出答案。

6. [问题描述]:

已知有两个字串 $A\$$, $B\$$ 及一组字串变换的规则 (至多 6 个规则):

$A1\$ \rightarrow B1\$$

$A2\$ \rightarrow B2\$$

规则的含义为: 在 $A\$$ 中的子串 $A1\$$ 可以变换为 $B1\$$ 、 $A2\$$ 可以变换为 $B2\$$...。

例如: $A\$ = 'abcd'$ $B\$ = 'xyz'$

变换规则为:

$'abc' \rightarrow 'xu'$ $'ud' \rightarrow 'y'$ $'y' \rightarrow 'yz'$

则此时, $A\$$ 可以经过一系列的变换变为 $B\$$, 其变换的过程为:

$'abcd' \rightarrow 'xud' \rightarrow 'xy' \rightarrow 'xyz'$

共进行了三次变换, 使得 $A\$$ 变换为 $B\$$ 。

[输入]:

键盘输入文件名。文件格式如下:

$A\$ B\$$

$A1\$ B1\$ \backslash$

$A2\$ B2\$ \rightarrow$ 变换规则

... ... /

所有字符串长度的上限为 20。

[输出]:

输出至屏幕。格式如下:

若在 10 步 (包含 10 步) 以内能将 $A\$$ 变换为 $B\$$, 则输出最少的变换步数; 否则输出 "NO ANSWER!"

[输入输出样例]

b. in:

abcd xyz

abc xu

ud y

y yz

第五部分 动态规划

第一章 什么叫动态规划

1.1 多阶段决策过程的最优化问题

1.2 动态规划的概念

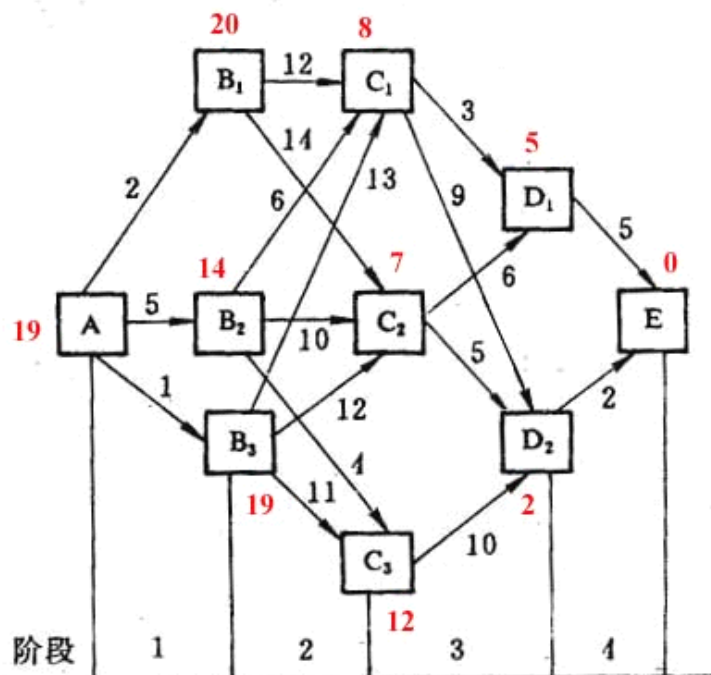
1.3 动态规划适合解决什么样的问题

1.1 多阶段决策过程的最优化问题

1、问题的提出

首先, 例举一个典型的且很直观的多阶段决策问题:

[例] 下图表示城市之间的交通路网, 线段上的数字表示费用, 单向通行由 $A \rightarrow E$ 。求 $A \rightarrow E$ 的最省费用。



如图从 A 到 E 共分为 4 个阶段，即第一阶段从 A 到 B，第二阶段从 B 到 C，第三阶段从 C 到 D，第四阶段从 D 到 E。除起点 A 和终点 E 外，其它各点既是上一阶段的终点又是下一阶段的起点。例如从 A 到 B 的第一阶段中，A 为起点，终点有 B₁, B₂, B₃ 三个，因而这时走的路线有三个选择，一是走到 B₁，一是走到 B₂，一是走到 B₃。若选择 B₂ 的决策，B₂ 就是第一阶段在我们决策之下的结果，它既是第一阶段路线的终点，又是第二阶段路线的始点。在第二阶段，再从 B₂ 点出发，对于 B₂ 点就有一个可供选择的终点集合 (C₁, C₂, C₃)；若选择由 B₂ 走至 C₂ 为第二阶段的决策，则 C₂ 就是第二阶段的终点，同时又是第三阶段的始点。同理递推下去，可看到各个阶段的决策不同，线路就不同。很明显，当某阶段的起点给定时，它直接影响着后面各阶段的行进路线和整个路线的长短，而后面各阶段的路线的发展不受这点以前各阶段的影响。故此问题的要求是：在各个阶段选取一个恰当的决策，使由这些决策组成的一个决策序列所决定的一条路线，其总路程最短。具体情况如下：

- (1) 由目标状态 E 向前推，可以分成四个阶段，即四个子问题。如上图所示。
- (2) 策略：每个阶段到 E 的最省费用为本阶段的决策路径。
- (3) D₁, D₂ 是第一次输入的结点。他们到 E 都只有一种费用，在 D₁ 框上面标 5，D₂ 框上面标 2。目前无法定下，那一个点将在全程最优策略的路径上。第二阶段计算中，5, 2 都应分别参加计算。
- (4) C₁, C₂, C₃ 是第二次输入结点，他们到 D₁, D₂ 各有两种费用。此时应计算 C₁, C₂, C₃ 分别到 E 的最少费用。C₁ 的决策路径是 $\min\{(C_1D_1), (C_1D_2)\}$ 。计算结果是 C_1+D_1+E ，在 C₁ 框上面标为 8。同理 C₂ 的决策路径计算结果是 C_2+D_2+E ，在 C₂ 框上面标为 7。同理 C₃ 的决策路径计算结果是 C_3+D_2+E ，在 C₃ 框上面标为 12。此时也无法定下第一，二阶段的城市那二个将在整体的最优决策路径上。
- (5) 第三次输入结点为 B₁, B₂, B₃，而决策输出结点可能为 C₁, C₂, C₃。仿前计算可得 B₁, B₂, B₃ 的决策路径为如下情况。
 B₁: B₁C₁ 费用 $12+8=20$ ，路径: B₁+C₁+D₁+E
 B₂: B₂C₁ 费用 $6+8=14$ ，路径: B₂+C₁+D₁+E
 B₃: B₃C₂ 费用 $12+7=19$ ，路径: B₃+C₂+D₂+E
 此时也无法定下第一，二，三阶段的城市那三个将在整体的最优决策路径上。
- (6) 第四次输入结点为 A，决策输出结点可能为 B₁, B₂, B₃。同理可得决策路径为
 A: A B₂，费用 $5+14=19$ ，路径 A+B₂+C₁+D₁+E。

此时才正式确定每个子问题的结点中，那一个结点将在最优费用的路径上。19 将结果显然这种计算方法，符合最优原理。子问题的决策中，只对同一城市（结点）比较优劣。而同一阶段的城市（结点）的优劣要由下一个阶段去决定。

1.2 动态规划的概念

在上例的多阶段决策问题中，各个阶段采取的决策，一般来说是与时间有关的，决策依赖于当前状态，又随即引起状态的转移，一个决策序列就是在变化的状态中产生出来的，故有“动态”的含义，称这种解决多阶段决策最优化问题的方法为动态规划方法。

与穷举法相比，动态规划的方法有两个明显的优点：

(1) 大大减少了计算量

(2) 丰富了计算结果

从上例的求解结果中，我们不仅得到由 A 点出发到终点 E 的最短路线及最短距离，而且还得到了从所有各中间点到终点的最短路线及最短距离，这对许多实际问题来讲是很有用的。

动态规划的最优化概念是在一定条件下，找到一种途径，在对各阶段的效益经过按问题具体性质所确定的运算以后，使得全过程的总效益达到最优。

应用动态规划要注意阶段的划分是关键，必须依据题意分析，寻求合理的划分阶段(子问题)方法。而每个子问题是一个比原问题简单得多的优化问题。而且每个子问题的求解中，均利用它的一个后部子问题的最优化结果，直到最后一个子问题所得最优解，它就是原问题的最优解。

1.3 动态规划适合解决什么样的问题

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。

动态规划的最优化原理是无论过去的状态和决策如何，对前面的决策所形成的当前状态而言，余下的诸决策必须构成最优策略。可以通俗地理解为子问题的局部最优将导致整个问题的全局最优在上例中例题 1 最短路径问题中，A 到 E 的最优路径上的任一点到终点 E 的路径也必然是该点到终点 E 的一条最优路径，满足最优化原理。

动态规划的无后效性原则某阶段的状态一旦确定，则此后过程的演变不再受此前各状态及决策的影响。也就是说，“未来与过去无关”，当前的状态是此前历史的一个完整总结，此前的历史只能通过当前的状态去影响过程未来的演变。具体地说，如果一个问题被划分各个阶段之后，阶段 I 中的状态只能由阶段 I+1 中的状态通过状态转移方程得来，与其他状态没有关系，特别是与未发生的状态没有关系，这就是无后效性。

第二章 用动态规划法解题

2.1 为什么要用动态规划法解题

2.2 怎样用动态规划法解题

2.3 用动态规划法解题的一般模式

2.1 为什么要用动态规划法解题

首先，看下面一个问题：

【例题 1】数字三角形问题。

```

      7
     3 8
    8 1 0
   2 7 7 4
  5 5 2 6 5
  
```

示出了一个数字三角形宝塔。数字三角形中的数字为不超过 100 的正整数。现规定从最顶层走到最底层，每一步可沿左斜线向下或右斜线向下走。假设三角形行数 ≤ 100 ，编程求解从最顶层走到最底层的一条路径，使得沿着该路径所经过的数字的总和最大，输出最大值。输入数据：由文件输入数据，文件第一行是三角形的行数 N。以后的 N 行分别是从最顶层到最底层的每一层中的数字。

如输入： 5

```

7
3 8
8 1 0
2 7 7 4
4 5 2 6 5
  
```

输出：30

【分析】对于这一问题，很容易想到用枚举的方法（深度搜索法）去解决，即列举出所有路径并记录每一条路径所经过的数字总和。然后寻找最大的数字总和，这一想法很直观，很容易编程实现其程序如下：

```

program sjx;
const maxn=10;
var
a:array[1..maxn,1..maxn] of integer;
max:longint;
n,i,j:integer;
fname:string;
inputf:text;
procedure try(x,y,dep:integer;sum:longint);
begin
  if (dep=n) then
    begin
      if sum>max then max:=sum;
      exit
    end;
  try(x+1,y,dep+1,sum+a[x+1,y]);
  try(x+1,y+1,dep+1,sum+a[x+1,y+1]);
end;
begin
  readln(fname);
  assign(inputf,fname);
  reset(inputf);
  readln(inputf,n);
  for i:=1 to n do
    for j:= 1 to i do
      read(inputf,a[i,j]);
  max:=0;
  try(1,1,1,a[1,1]);
  writeln(max);
end.

```

但是当行数很大时，当三角形的行数等于 100 时，其枚举量之大是可想而知的，用枚举法肯定超时，甚至根本不能得到计算结果，必须用动态规划法来解。

2.2 怎样用动态规划法解题

1. 逆推法:

按三角形的行划分阶段，若行数为 n ，则可把问题看做一个 $n-1$ 个阶段的决策问题。先求出第 $n-1$ 阶段(第 $n-1$ 行上各点)到第 n 行的最大和，再依次求出第 $n-2$ 阶段、第 $n-3$ 阶段……第 1 阶段(起始点)各决策点至第 n 行的最佳路径。

设: $f[i, j]$ 为从第 i 阶段中的点 j 至第 n 行的最大的数字和;

则: $f[n, j]=a[n, j] \quad 1 \leq j \leq n$

$$f[i, j]=\max\{a[i, j]+f[i+1, j], a[i, j]+f[i+1, j+1]\} \quad 1 \leq j \leq i.$$

$f[1, 1]$ 即为所求。

程序如下:

```

program datasjx;
const maxn=100;
var
  fname:string;
  inputf:text;
  n,i,j:integer;
  a:array[1..maxn,1..maxn] of integer;
  f:array[1..maxn,1..maxn] of integer;
begin
  readln(fname);

```

```

assign(inputf, fname);
reset(inputf);
readln(inputf, n);
for i:=1 to n do
  for j:=1 to i do
    read(inputf, a[i, j]);
for i:=1 to n do
  f[n, i]:=a[n, i];
for i:=n-1 downto 1 do
  for j:=1 to i do
    if f[i+1, j]>f[i+1, j+1] then f[i, j]:=a[i, j]+f[i+1, j]
    else f[i, j]:=a[i, j]+f[i+1, j+1];
writeln(f[1, 1]);
end.

```

2. 顺推法

按三角形的行划分阶段，若行数为 n ，则可把问题看做一个 $n-1$ 个阶段的决策问题。先求第 2 行各元素到起点的最大和

, 再依次求出第 3, 4, 5, ..., $n-1$, n 到起点的最大和, 最后找第 n 行的最大值

设 $f[i, j]$ 为 第 i 行第 j 列上点到起点的最大和

则 $f[1, 1]=a[1, 1]$;

$$f[i, 1]=a[i, 1]+f[i-1, 1];$$

$$f[i, j]=\max\{a[i, j]+f[i-1, j-1], a[i, j]+f[i-1, j]\} \quad 2 \leq j \leq i$$

$\max\{f[n, 1], f[n, 2], \dots, f[n, n]\}$ 即为所求。

程序如下:

```

program datasjx;
const maxn=100;
var
  fname:string;
  inputf:text;
  n,i,j:integer;
  a:array[1..maxn,1..maxn] of integer;
  f:array[1..maxn,1..maxn] of integer;
  maxsum:integer;
begin
  readln(fname);
  assign(inputf, fname);
  reset(inputf);
  readln(inputf, n);
  for i:=1 to n do
    for j:=1 to i do
      read(inputf, a[i, j]);
  fillchar(f, sizeof(f), 0);
  f[1, 1]:=a[1, 1];
  for i:=2 to n do
    begin
      f[i, 1]:=a[i, 1]+f[i-1, 1];
      for j:=2 to i do
        if f[i-1, j-1]>f[i-1, j] then f[i, j]:=a[i, j]+f[i-1, j-1]
        else f[i, j]:=a[i, j]+f[i-1, j];
      end;
  maxsum:=0;
  for i:=1 to n do

```

```

    if f[n,i]>maxsum then maxsum:=f[n,i];
    writeln(maxsum);
end.

```

说明一下:

1. 用动态规划解题主要思想是用**空间换时间**.
2. 本题如果 n 较大, 用 2 维数组空间可能不够, 可以使用 1 维数组.

程序如下:

```

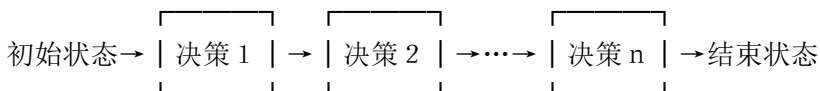
program datasjx;
const maxn=100;
var
    fname:string;
    inputf:text;
    n,i,j:integer;
    a:array[1..maxn,1..maxn] of integer;
    f:array[1..maxn] of integer;
    maxsum:integer;
begin
    readln(fname);
    assign(inputf,fname);
    reset(inputf);
    readln(inputf,n);
    for i:=1 to n do
        for j:=1 to i do
            read(inputf,a[i,j]);
    fillchar(f,sizeof(f),0);
    f[1]:=a[1,1];
    for i:=2 to n do
        begin
            for j:=i downto 2 do
                if f[j-1]>f[j] then f[j]:=a[i,j]+f[j-1]
                else f[j]:=a[i,j]+f[j];
            f[1]:=a[i,1]+f[1];
        end;
    maxsum:=0;
    for i:=1 to n do
        if f[i]>maxsum then maxsum:=f[i];
    writeln(maxsum);
end.

```

练习:用一维数组和逆推法解本题.

2.3 用动态规划法解题的一般模式

动态规划所处理的问题是一个多阶段决策问题, 一般由初始状态开始, 通过对中间阶段决策的选择, 达到结束状态。这些决策形成了一个决策序列, 同时确定了完成整个过程的一条活动路线(通常是求最优的活动路线)。如图所示。动态规划的设计都有着一定的模式, 一般要经历以下几个步骤。



(1) 划分阶段: 按照问题的时间或空间特征, 把问题分为若干个阶段。在划分阶段时, 注意划分后的阶段一定要是有序的或者是可排序的, 否则问题就无法求解。

(2) 确定状态和状态变量: 将问题发展到各个阶段时所处于的各种客观情况用不同的状态表示出来。当然, 状态的选择要满足无后效性。

(3) 确定决策并写出状态转移方程: 因为决策和状态转移有着天然的联系, 状态转移就是根据上一阶段的状态和决策来

导出本阶段的状态。所以如果确定了决策, 状态转移方程也就可写出。但事实上常常是反过来做, 根据相邻两段各状态之间的关系来确定决策。

(4) 寻找边界条件: 给出的状态转移方程是一个递推式, 需要一个递推的终止条件或边界条件。

(5) 程序设计实现: 动态规划的主要难点在于理论上的设计, 一旦设计完成, 实现部分就会非常简单。

根据上述动态规划设计的步骤, 可得到大体解题框架如下:

1. 初始化(边界条件)
2. for $i:=2$ to n (顺推法) 或 for $i:=n-1$ to 1 (逆推法)
 - 对 i 阶段的每一个决策点求局部最优
3. 确定和输出结束状态的值。

第三章 典型例题与习题

3.1 最长不降子序列

3.2 背包问题

3.3 最短路径

4.3 习题

3.1 最长不降子序列

(1) 问题描述

设有由 n 个不相同的整数组成的数列, 记为:

$a(1)$ 、 $a(2)$ 、……、 $a(n)$ 且 $a(i) < a(j)$ ($i < j$)

例如 3, 18, 7, 14, 10, 12, 23, 41, 16, 24。

若存在 $i_1 < i_2 < i_3 < \dots < i_e$ 且有 $a(i_1) < a(i_2) < \dots < a(i_e)$ 则称为长度为 e 的不下降序列。如上例中 3, 18, 23, 24 就是一个长度为 4 的不下降序列, 同时也有 3, 7, 10, 12, 16, 24 长度为 6 的不下降序列。程序要求, 当原数列给出之后, 求出最长的不下降序列。

(2) 算法分析

根据动态规划的原理, 由后往前进行搜索。

1• 对 $a(n)$ 来说, 由于它是最后一个数, 所以当从 $a(n)$ 开始查找时, 只存在长度为 1 的不下降序列;

2• 若从 $a(n-1)$ 开始查找, 则存在下面的两种可能性:

①若 $a(n-1) < a(n)$ 则存在长度为 2 的不下降序列 $a(n-1)$, $a(n)$ 。

②若 $a(n-1) > a(n)$ 则存在长度为 1 的不下降序列 $a(n-1)$ 或 $a(n)$ 。

3• 一般若从 $a(i)$ 开始, 此时最长不下降序列应该按下列方法求出:

在 $a(i+1)$, $a(i+2)$, ..., $a(n)$ 中, 找出一个比 $a(i)$ 大的且最长的不下降序列, 作为它的后继。

4. 用数组 $b(i)$, $c(i)$ 分别记录点 i 到 n 的最长的不降子序列的长度和点 i 后继接点的编号

(3) 程序如下: (逆推法)

```
program l11;
const maxn=100;
var a,b,c:array[1..maxn] of integer;
    fname:string;
    f:text;
    n,i,j,max,p:integer;
begin
  readln(fname);
  assign(f,fname);
  reset(f);
  readln(f,n);+
  for i:=1 to n do
  begin
    read(f,a[i]);
    b[n]:=1;
    c[n]:=0;
  end;
```

```

for i:= n-1 downto 1 do
begin
  max:=0;p:=0;
  for j:=i+1 to n do
    if (a[i]<a[j]) and (b[j]>max) then begin max:=b[j];p:=j end;
  if p<>0 then begin b[i]:=b[p]+1;c[i]:=p end
end;
max:=0;p:=0;
for i:=1 to n do
  if b[i]>max then begin max:=b[i];p:=i end;
writeln('maxlong=',max);
write('result is:');
while p<>0 do
  begin write(a[p]:5);p:=c[p] end;
end.

```

3.2 背包问题

背包问题有三种

1. 部分背包问题

一个旅行者有一个最多能用 m 公斤的背包, 现在有 n 种物品, 它们的总重量分别是 W_1, W_2, \dots, W_n , 它们的总价值分别为 C_1, C_2, \dots, C_n . 求旅行者能获得最大总价值。

解决问题的方法是贪心算法: 将 $C_1/W_1, C_2/W_2, \dots, C_n/W_n$, 从大到小排序, 不停地选择价值与重量比最大的放入背包直到放满为止。

2. 0/1 背包

一个旅行者有一个最多能用 m 公斤的背包, 现在有 n 件物品, 它们的重量分别是 W_1, W_2, \dots, W_n , 它们的价值分别为 C_1, C_2, \dots, C_n . 若每种物品只有一件求旅行者能获得最大总价值。

<1>分析说明:

显然这个题可用深度优先方法对每件物品进行枚举(选或不选用 0, 1 控制)。

程序简单, 但是当 n 的值很大的时候不能满足时间要求, 时间复杂度为 $O(2^n)$ 。按递归的思想我们可以把问题分解为子问题, 使用递归函数

设 $f(i, x)$ 表示前 i 件物品, 总重量不超过 x 的最优价值

则 $f(i, x) = \max(f(i-1, x-W[i]) + C[i], f(i-1, x))$

$f(n, m)$ 即为最优解, 边界条件为 $f(0, x) = 0$, $f(i, 0) = 0$;

动态规划方法(顺推法)程序如下:

程序如下:

```

program knapsack02;
const maxm=200;maxn=30;
type ar=array[1..maxn] of integer;
var m,n,j,i:integer;
c,w:ar;
f:array[0..maxn,0..maxm] of integer;
function max(x,y:integer):integer;
begin
  if x>y then max:=x else max:=y;
end;
begin
  readln(m,n);
  for i:= 1 to n do
    readln(w[i],c[i]);
  for i:=1 to m do f(0,i):=0;
  for i:=1 to n do f(i,0):=0;
  for i:=1 to n do
    for j:=1 to m do

```

```

begin
  if j>=w[i] then f[i,j]:=max(f[i-1,j-w[i]]+c[i],f[i-1,j])
  else f[i,j]:=f[i-1,j];
end;
writeln(f[n,m]);
end.

```

使用二维数组存储各子问题时方便，但当 maxm 较大时如 maxn=2000 时不能定义二维数组 f，怎么办，其实可以用一维数组，但是上述中 j:=1 to m 要改为 j:=m downto 1，为什么？请大家自己解决。

3. 完全背包问题

一个旅行者有一个最多能用 m 公斤的背包，现在有 n 种物品，每件重量分别是 W_1, W_2, \dots, W_n ，每件的价值分别为 C_1, C_2, \dots, C_n 。若的每种物品的件数足够多。

求旅行者能获得的最大总价值。

本问题的数学模型如下：

设 $f(x)$ 表示重量不超过 x 公斤的最大价值，

则 $f(x) = \max\{f(x-w[i])+c[i]\}$ 当 $x \geq w[i]$ $1 \leq i \leq n$

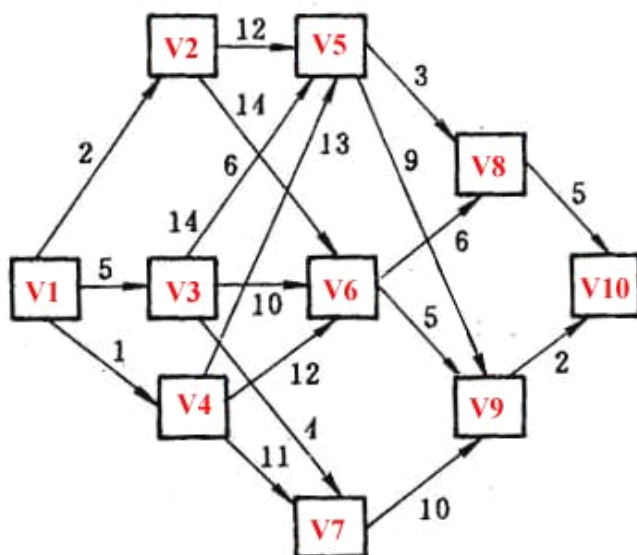
程序如下：（顺推法）

```

program knapsack04;
const maxm=2000;maxn=30;
type ar=array[0..maxn] of integer;
var m,n,j,i,t:integer;
c,w:ar;
f:array[0..maxm] of integer;
begin
  readln(m,n);
  for i:= 1 to n do
    readln(w[i],c[i]);
    f(0):=0;
  for i:=1 to m do
    for j:=1 to n do
      begin
        if i>=w[j] then t:=f[i-w[j]]+c[j];
        if t>f[i] then f[i]:=t
      end;
    writeln(f[m]);
  end.

```

3.3 最短路径



问题描述：

如图：求 v1 到 v10 的最短路径长度及最短路径。

图的邻接矩阵如下：

```

0 2 5 1 -1 -1 -1 -1 -1 -1
-1 0 -1 -1 12 14 -1 -1 -1 -1
-1 -1 0 -1 6 10 4 -1 -1 -1
-1 -1 -1 0 13 12 11 -1 -1 -1
-1 -1 -1 -1 0 -1 -1 3 9 -1
-1 -1 -1 -1 -1 0 -1 6 5 -1
-1 -1 -1 -1 -1 -1 0 -1 10 -1
-1 -1 -1 -1 -1 -1 -1 0 -1 5
-1 -1 -1 -1 -1 -1 -1 -1 0 2
-1 -1 -1 -1 -1 -1 -1 -1 -1 0

```

采用逆推法

设 $f(x)$ 表示点 x 到 v_{10} 的最短路径长度

则 $f(10)=0$

$$f(x)=\min\{f(i)+a[x,i] \mid a[x,i]>0, x<i\leq n\}$$

程序如下：（逆推法）

```

program lt3;
const n=10;
var
  a:array[1..n,1..n] of integer;
  b,c:array[1..n] of integer;
  fname:string;
  f:text;
  i,j,x:integer;
begin
  readln(fname);
  assign(f,fname);
  reset(f);
  for i:=1 to n do
    for j:=1 to n do
      read(f,a[i,j]);
  close(f);
  for i:=1 to n do
    b[i]:=maxint;
  b[n]:=0;
  for i:= n-1 downto 1 do
    for j:=n downto i+1 do
      if (a[i,j]>0) and (b[j]<>maxint) and (b[j]+a[i,j]<b[i])
        then begin b[i]:=b[j]+a[i,j];c[i]:=j end;
  writeln('minlong=',b[1]);
  x:=1;
  while x<>0 do
    begin
      write(x:5);
      x:=c[x];
    end;
end.

```

3.4 习题

1. 若城市路径示意图如下图所示：

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 2 | 3 | 1 | 2 | B | | | |
| 2 | 2 | 1 | 3 | 2 | 4 | 2 | 5 | 3 |
| 3 | 2 | 4 | 2 | 1 | 1 | 2 | 4 | 3 |
| 3 | 1 | 2 | 3 | 1 | 2 | 2 | 3 | 4 |
| A | | | | | | | | |

图中，每条边上的数字是这段道路的长度。条件：从 A 地出发，只允许向右或向上走。试寻找一条从 A 地到 B 地的最短路径和长度。

2. 求一个数列中的连续若干个数和的最大值。

3. 资源分配问题: n 个资源分配到 m 个项目上， i 项目分配 j 个资源可获益 $a[i, j]$, 求最大总效益。

4. 装箱问题

问题描述: 有一个箱子容量为 V (正整数, $0 \leq V \leq 20000$), 同时有 n 个物品 ($0 < n \leq 30$, 每个物品有一个体积 (正整数))。要求 n 个物品中, 任取若干个装入箱内, 使箱子的剩余空间为最小。

样例

输入:

24 一个整数, 表示箱子容量
6 一个整数, 表示有 n 个物品
8 接下来 n 行, 分别表示这 n 个物品的各自体积
3
12
7
9
7

输出:

0 一个整数, 表示箱子剩余空间。

第四章 动态规划的递归函数法

4.1 原始递归法

4.2 改进递归法

4.3 习题

4.1 原始递归法

先看完全背包问题

一个旅行者有一个最多能用 m 公斤的背包, 现在有 n 种物品, 每件的重量分别是 W_1, W_2, \dots, W_n , 每件的价值分别为 C_1, C_2, \dots, C_n . 若的每种物品的件数足够多.

求旅行者能获得的最大总价值。

本问题的数学模型如下:

设 $f(x)$ 表示重量不超过 x 公斤的最大价值,

则 $f(x) = \max\{f(x-i) + c[i]\}$ 当 $x \geq w[i]$ $1 \leq i \leq n$

可使用递归法解决问题程序如下:

```
program knapsack04;
  const maxm=200;maxn=30;
  type ar=array[0..maxn] of integer;
  var m,n,j,i,t:integer;
      c,w:ar;
function f(x:integer):integer;
var i,t,m:integer;
begin
  if x=0 then f:=0 else
```

```

begin
  t:=-1;
  for i:=1 to n do
    begin
      if x>=w[i] then m:=f(x-i)+c[i];
      if m>t then t:=m;
    end;
  f:=t;
end;
begin
  readln(m,n);
  for i:= 1 to n do
    readln(w[i],c[i]);
    writeln(f(m));
  end.

```

说明:当 m 不大时,编程很简单,但当 m 较大时,容易超时.

4.2 改进的递归法

改进的的递归法的思想还是以空间换时间,这只要将递归函数计算过程中的各个子函数的值保存起来,开辟一个一维数组即可

程序如下:

```

program knapsack04;
const maxm=2000;maxn=30;
type ar=array[0..maxn] of integer;
var m,n,j,i,t:integer;
    c,w:ar;
p:array[0..maxm] of integer;
function f(x:integer):integer;
var i,t,m:integer;
begin
  if p[x]<>-1 then f:=p[x]
  else
    begin
      if x=0 then p[x]:=0 else
        begin
          t:=-1;
          for i:=1 to n do
            begin
              if x>=w[i] then m:=f(i-w[i])+c[i];
              if m>t then t:=m;
            end;
          p[x]:=t;
        end;
      f:=p[x];
    end;
end;
begin
  readln(m,n);
  for i:= 1 to n do
    readln(w[i],c[i]);
  fillchar(p,sizeof(p),-1);
  writeln(f(m));

```

end.

4.3 习题

用改进的递归法解资源分配问题:

n 个资源分配到 m 个项目上, i 项目分配 j 个资源可获益 $a[i, j]$, 求最大总效益。

第五章 动态规划分类

5.1 例 1

5.2 例 2

5.3 例 3

5.1 例 1

乘积最大问题

问题描述:

今年是国际数学联盟确定的“2000—世界数学年”, 又恰逢我国著名数学家华罗庚先生诞辰 90 周年。在华罗庚先生的家乡江苏金坛, 组织了一场别开生面的数学智力竞赛的活动, 你的一个好朋友 XZ 也有幸得以参加。活动中, 主持人给所有参加活动的选手除了这样一道题目:

设有一个长度为 N 的数字串, 要求选手使用 K 个乘号将它分成 $K+1$ 个部分, 找出一种分法, 使得这 $K+1$ 部分的乘积能够为最大。

同时, 为了帮助选手能够正确理解题意, 主持人还举了如下的一个例子:

有一个数字串: 312, 当 $N=3, K=1$ 时会有一下两种分法:

1) $3*12=36$

2) $31*2=62$

这时, 符合题目要求的结果是: $31*2=62$

现在, 请你帮助你的好朋友 XZ 设计一个程序, 求得正确的答案。

输入:

程序的输入共有两行:

第一行共有 2 个自然数 N, K ($6 \leq N \leq 50, 1 \leq K \leq 20$)

第二行是一个长度为 N 的数字串。

输出:

结果显示在屏幕上, 相对于输入, 应输出所求得的最大乘积 (一个自然数)。

样例:

输入

4 2

1231

输出

62

程序如下:

```
program tg20002;
const maxn=50;maxk=20 ;
type num=record
    len:byte;
    s:array[1..maxn] of byte;
end;
type str=string[maxn];
var n,k,i,j,l:integer;
    st,temps:str;
    m,tempm:num;
    f:array[1..maxn] of num;
    sl:array[1..maxn] of str;
procedure strtonum(str1:str;var num1:num);
var i:integer;
```

```

begin
  num1.len:=length(str1);
  for i:=1 to num1.len do
    num1.s[i]:=ord(str1[num1.len-i+1])-ord('0');
end;
procedure mult(x,y:num;var z: num );
var i,j,len:integer;
begin
  fillchar(z,sizeof(z),0);
  for i:= 1 to x.len do
    for j:=1 to y.len do
      begin
        inc(z.s[i+j-1], x.s[i]*y.s[j]);
        inc(z.s[i+j],z.s[i+j-1] div 10);
        z.s[i+j-1]:=z.s[i+j-1] mod 10;
      end;
    len:=x.len+y.len+1;
    while (len>1) and (z.s[len]=0) do len:=len-1;
    z.len:=len;
  end;
procedure bigger(x,y:num;var z:num);
var i:integer;
begin
  if x.len>y.len then begin z.len:=x.len;z.s:=x.s end else
  if x.len<y.len then begin z.len:=y.len;z.s:=y.s end else
  begin
    z.len:=x.len;
    i:=z.len;
    while (i>1) and (x.s[i]=y.s[i]) do i:=i-1;
    if x.s[i]>=y.s[i] then z.s:=x.s else z.s:=y.s;
  end;
end;
begin
  readln(n,k);
  readln(st);
  fillchar(f,sizeof(f),0);
  for i:=1 to n do
    begin
      s1[i]:=copy(st,1,i);
      strtonum(s1[i],f[i]);
    end;
  for i:=2 to k+1 do
    for j:=n downto i do
      begin
        fillchar(m,sizeof(m),0);
        for l:=i-1 to j-1 do
          begin
            temps:=copy(s1[j],l+1,j-1);
            strtonum(temps,tempm);
            mult(f[l],tempm, tempm);
            bigger(m,tempm,m);
          end;
        f[j]:=m;
      end;
    end;
  end;
end;

```



```

end;
l:=f[n].len;
while (l>1) and (f[n].s[l]=0) do l:=l-1;
  for i:=l downto 1 do write(f[n].s[i]);
writeln;
end.

```

5.2 例 2

统计单词个数

[问题描述]

给出一个长度不超过 200 的由小写英文字母组成的字符串（约定：该字符串以每行 20 个字母的方式输入，且保证每行一定为 20 个）。要求将此字符串分成 k 份 ($1 < k \leq 40$)，且每份中包含的单词个数加起来总数最大（每份中包含的单词可以部分重叠。当选用一个单词之后，其第一个字母不能再选。例如字符串 this 中可以包含 this 和 is，选用 this 之后就不能包含 th）。

单词在给出的一个不超过 6 个单词的字典中。

要求输出最大的个数。

[输入格式]

全部输入数据存放在文本文件 input3.dat 中，其格式如下：

第一行为一个正整数 ($0 < n \leq 5$) 表示有 n 组测试数据

每组的第二行有二个正整数：(p, k)

p 表示字符串的行数；

k 表示分为 k 个部分。

接下来的 p 行，每行均有 20 个字符。

再接下来有一个正整数 s ，表示字典中单词个数。 ($1 \leq s \leq 6$)

接下来的 s 行，每行均有一个单词。

[输出格式]

结果输出至屏幕，每行一个整数，分别对应每组测试数据的相应结果。

[样例]

输入：1

```

1 3
thisisabookyouareao
4
is
a
ok
sab

```

```

输出：          //说明：（不必输出）
7             // this/isabookyoua/reaoh
              sab

```

```

program tg20013;
const maxl=200;
type chararr=string[maxl];
var n,p,k,s,l,i,max: integer;
word:array[1..6] of chararr;
s1:chararr;
str:array[1..10] of string[20];
fi:text;
f:array[1..maxl] of integer;
len:array[1..maxl] of integer;
g:array[1..maxl,1..maxl] of byte;
function plus(str:chararr):integer;
  var l1,r,i,j,x:integer;

```

```

    b:array[1..200] of boolean;
begin
    x:=0;
    l1:=length(str);
    fillchar(b,sizeof(b),true);
    for j:= 1 to s do
        begin
            r:=length(word[j]);
            for i:= 1 to l1 do
                if (word[j]=copy(str,i,r)) and b[i] then begin b[i]:=false;x:=x+1 end
            end;
        plus:=x;
        end;
    procedure calcclen;
    var i,j:integer;
    begin
        for i:=1 to l do len[i]:=200;
        for i:=1 to l do
            for j:=1 to s do
                if copy(s1,i,length(word[j]))=word[j] then if
                    len[i]>length(word[j]) then len[i]:=length(word[j]);
            end;
        end;
    procedure calc_g;
    var i,j:integer;
    begin
        fillchar(g,sizeof(g),0);
        for i:=1 downto 1 do
            begin
                if len[i]=1 then g[i,i]:=1 ;
                for j:=1-1 downto 1 do
                    if len[j]<=i-j+1 then g[j,i]:=g[j+1,i]+1 else g[j,i]:=g[j+1,i];
                end;
            end;
        end;
    procedure calc;
    var i,j,m,maxt,t:integer;stri:string;
    begin
        for i:=1 to l do
            f[i]:=plus(copy(s1,1,i));
        for i:=2 to k do
            begin
                for j:=1 downto 1 do
                    begin
                        maxt:=0;
                        for m:=i-1 to j-1 do
                            begin
                                t:=f[m]+g[m+1,j];
                                if t>maxt then maxt:=t;
                            end;
                        f[j]:=maxt;
                    end;
                end;
            end;
        end;
    end;
begin

```

```

assign(fi,'input3.dat');
reset(fi);
readln(fi,n);
repeat
  dec(n);
  readln(fi,p,k);
  s1:='';
  for i:=1 to p do
    begin
      readln(fi,str[i]);
      s1:=s1+str[i];
    end;
  l:=length(s1);
  readln(fi,s);
  for i:=1 to s do
    readln(fi,word[i]);
  calcclen;
  calc_g;
  calc;
  writeln(f[l]);
until n=0;
end.

```

5.3 例 3

2002 年第四题:

求: 平面上的 n 个点用 k 个矩形覆盖的最小面积?

程序如下:

```

program tg20024;
const maxn=50;
type pointset=array[1..maxn,1..2] of longint;
var
  n,k,i,j,l:integer;
  a:pointset;
  f:array[1..maxn] of longint;
maxs,min,t:longint;
procedure init;
  var i:integer;
  filename:string[20];
  f:text;
begin
  readln(filename);
  assign(f,filename);
  reset(f);
  readln(n,k);
  for i:= 1 to n do
    readln(a[i,1],a[i,2]);
  close(f);
end;
procedure sort(var x:pointset);
var i,j,m:integer;
    t1,t2,t3:integer;
begin
  for i:=1 to n-1 do

```

```

begin
  t1:=x[i,1];t2:=x[i,2];t3:=t2;m:=i;
  for j:= i+1 to n do
    if t3>x[j,2] then begin m:=j; t3:=x[j,2];end;
  if m<>i then
    begin x[i,1]:=x[m,1];x[i,2]:=x[m,2];x[m,1]:=t1;x[m,2]:=t2 end;
  end;
end;
function maxy(x0,xn:integer):integer;
var i,max:integer;
begin
  max:=0;
  for i:=x0 to xn do
    if max<a[i,1] then max:=a[i,1];
maxy:=max;
end;
function miny(x0,xn:integer):integer;
var i,min:integer;
begin
  min:=32767;
  for i:=x0 to xn do
    if min>a[i,1] then min:=a[i,1];
miny:=min;
end;
begin
  init;
  sort(a);
  fillchar(f,sizeof(f),0);
  maxs:=(a[n,2]-a[1,2])*(maxy(1,n)-miny(1,n));
  for i:= 1 to n do
    f[i]:=(a[i,2]-a[1,2])*(maxy(1,i)-miny(1,i));
  for i:=2 to k do
    for j:= n downto 1 do
      begin
        min:=maxs;
        for l:=i-1 to j-1 do
          begin
            t:=f[l]+(a[j,2]-a[l+1,2])*(maxy(l+1,j)-miny(l+1,j));
            if (t<min) and (a[l,2]<>a[l+1,2]) then min:=t;
          end;
        f[j]:=min;
      end;
    writeln(f[n]);
  end.

```

练习：Barn Repair

It was a dark and stormy night that ripped the roof and gates off the stalls that hold Farmer John's cows. Happily, many of the cows were on vacation, so the barn was not completely full. The cows spend the night in stalls that are arranged adjacent to each other in a long line. Some stalls have cows in them; some do not. All stalls are the same width. Farmer John must quickly erect new boards in front of the stalls, since the doors were lost. His new lumber supplier will supply him boards of any length he wishes, but the supplier can only deliver a small number

of total boards. Farmer John wishes to minimize the total length of the boards he must purchase. Given M ($1 \leq M \leq 50$), the maximum number of boards that can be purchased; S ($1 \leq S \leq 200$), the total number of stalls; C ($1 \leq C \leq S$) the number of cows in the stalls, and the C occupied stall numbers ($1 \leq \text{stall_number} \leq S$), calculate the minimum number of stalls that must be blocked in order to block all the stalls that have cows in them.

Print your answer as the total number of stalls blocked.

PROGRAM NAME: barn1

INPUT FORMAT

| | |
|--------------|--|
| Line 1: | M, S, and C (space separated) |
| Lines 2-C+1: | Each line contains one integer, the number of an occupied stall. |

SAMPLE INPUT (file barn1.in)

```
4 50 18
3
4
6
8
14
15
16
17
21
25
26
27
30
31
40
41
42
43
```

OUTPUT FORMAT

A single line with one integer that represents the total number of stalls blocked.

SAMPLE OUTPUT (file barn1.out)

```
25
[One minimum arrangement is one board covering stalls 3-8, one covering 14-21, one covering 25-31, and one covering 40-43.]
```

程序如下:

```
program barn1;
const maxm=50;maxs=200;
var m,s,c,i,n:integer;
    a:array[1..maxs,1..maxs] of byte;
    b:array[1..maxs] of boolean;
    f:array[1..maxs] of integer;
f1,f2:text;
function minl(x,y:integer):integer;
var i,j:integer;
begin
i:=x;j:=y;
while not(b[i]) and (i<=y) do inc(i);
while not(b[j]) and (j>=x) do dec(j);
```

```

if i<=j then minl:=j-i+1 else minl:=0;
end;
procedure calac_a;
var i,j:integer;
begin
for i:=1 to s do
  for j:=1 to s do
    a[i,j]:=minl(i,j);
end;
procedure main;
var i,j,k,min:integer;
begin
  fillchar(f,sizeof(f),0);
  calac_a;
  for i:=1 to s do
    f[i]:=a[1,i];
  for i:= 2 to m do
    for j:=s downto i do
      begin
        min:=200;
        for k:=i-1 to j-1 do
          if f[k]+a[k+1,j]<min then min:=f[k]+a[k+1,j];
        f[j]:=min;
      end;
    end;
  end;
begin
  assign(f1,'barn1.in');
  reset(f1);
  readln(f1,m,s,c);
  fillchar(b,sizeof(b),false);
  for i:=1 to c do
    begin
      readln(f1,n);
      b[n]:=true;
    end;
  close(f1);
  assign(f2,'barn1.out');
  main;
  rewrite(f2);
  writeln(f2,f[s]);
  close(f2);
end.

```

第六部分 数学知识及相关算法

第一章 有关数论的算法

- 1.1 最大公约数与最小公倍数
- 1.2 有关素数的算法
- 1.3 方程 $ax+by=c$ 的整数解及应用
- 1.4 求 $a^b \bmod n$

1.1 最大公约数与最小公倍数

1. 算法 1: 欧几里德算法求 a, b 的最大公约数

```
function gcd(a,b:longint):longint;
begin
  if b=0 then gcd:=a
  else gcd:=gcd(b,a mod b);
end;
```

2. 算法 2: 最小公倍数 $acm=a*b \div gcd(a, b)$;3. 算法 3: 扩展的欧几里德算法, 求出 $gcd(a, b)$ 和满足 $gcd(a, b)=ax+by$ 的整数 x 和 y

```
function exgcd(a,b:longint;var x,y:longint):longint;
var
  t:longint;
begin
  if b=0 then
  begin
    result:=a;
    x:=1;
    y:=0;
  end
  else
  begin
    result:=exgcd(b,a mod b,x,y);
    t:=x;
    x:=y;
    y:=t-(a div b)*y;
  end;
end;
```

(理论依据: $gcd(a, b)=ax+by=bx_1+(a \bmod b)y_1=bx_1+(a-(a \div b)*b)y_1=ay_1+b(x_1-(a \div b)*y_1)$)

1. 2 有关素数的算法

1. 算法 4: 求前 n 个素数:

```
program BasicMath_Prime;
const
  maxn=1000;
var
  pnum,n:longint;
  p:array[1..maxn] of longint;
function IsPrime(x:longint):boolean;
var i:integer;
begin
  for i:=1 to pnum do
    if sqrt(p[i])<=x then
      begin
        if x mod p[i]=0 then
          begin
            IsPrime:=false;
            exit;
          end;
        end;
      end
  else
    begin
      IsPrime:=true;
      exit;
    end;
  end;
```

```

IsPrime:=true;
end;
procedure main;
var x:longint;
begin
pnum:=0;
x:=1;
while(pnum<n) do
begin
inc(x);
if IsPrime(x) then
begin
inc(pnum);
p[pnum]:=x;
end;
end;
end;
end;
procedure out;
var i,t:integer;
begin
for i:=1 to n do
begin
write(p[i]:5);t:=t+1;
if t mod 10=0 then writeln;
end;
end;
begin
readln(n);
main;
out;
end.

```

2. 算法 5: 求不大于 n 的所有素数

```

program sushu3;
const maxn=10000;
var
i,k,n:integer;
a:array[1..maxn] of integer;
begin
readln(n);
for i:=1 to n do a[i]:=i;
a[1]:=0;
i:=2;
while i<n do
begin
k:=2*i;
while k<=n do
begin
a[k]:=0;
k:=k+i;
end;
i:=i+1;
while (a[i]=0) and (i<n) do i:=i+1;

```



```

end;
k:=0;
for i:=1 to n do
  if a[i]<>0 then
    begin
      write(a[i]:5); k:=k+1;
      if k mod 10 =0 then writeln;
    end
  end;
end.

```

3. 算法 6: 将整数分解质因数的积

```

program BasicMath_PolynomialFactors;
const
maxp=1000;
var
pnum,n:longint;
num,p:array[1..maxp] of longint;
procedure main;
var x:longint;
begin
  fillchar(num,sizeof(num),0);
  fillchar(p,sizeof(p),0);
  pnum:=0;
  x:=1;
  while(n>1) do
    begin
      inc(x);
      if n mod x=0 then
        begin
          inc(pnum);
          p[pnum]:=x;
          while(n mod x=0) do
            begin
              n:=n div x;
              inc(num[pnum]);
            end;
          end;
        end;
    end;
end;
procedure out;
var j,i:integer;
begin
  for i:=1 to pnum do
    for j:=1 to num[i] do
      write(p[i]:5);
    writeln;
  end;
begin
  main;
  out;
end.

```

1.3 方程 $ax+by=c$ 的整数解及应用

1. 算法 7: 求方程 $ax+by=c$ 的整数解

```

procedure equation(a,b,c:longint;var x0,y0:longint);
var d,x,y:longint;
begin
  d:=exgcd(a,b,x,y);
  if c mod d>0 then
  begin
    writeln('no answer');
    halt;
  end else
  begin
    x0:=x*(c div d);
    y0:=y*(c div d);
  end;
end;

```

2. 方程 $ax+by=c$ 整数解的应用

例 1: 有三个分别装有 a 升水、 b 升水和 c 升水的量筒 ($\gcd(a, b)=1, c>b>a>0$), 现 c 筒装满水, 问能否在 c 筒个量出 d 升水 ($c>d>0$)。若能, 请列出一种方案。

算法分析: 量水过程实际上就是倒来倒去, 每次倒的时候总有如下几个特点:

1. 总有一个筒中的水没有变动;
2. 不是一个筒被倒满就是另一个筒被倒光;
3. c 筒仅起中转作用, 而本身容积除了必须足够装下 a 筒和 b 筒的全部水外, 别无其它限制。

程序如下:

```

program mw;
type
  node=array[0..1] of longint;
var
  a,b,c:node;
  d,step,x,y:longint;
function exgcd(a,b:longint;var x,y:longint):longint;
var t:longint;
begin
  if b=0 then
  begin
    exgcd:=a;;x:=1;y:=0;
  end
  else
  begin
    exgcd:=exgcd(b,a mod b,x,y);
    t:=x;x:=y;y:=t-(a div b)*y
  end;
end;
end;
procedure equation(a,b,c:longint;var x0,y0:longint);
var d,x,y:longint;
begin
  d:=exgcd(a,b,x,y);
  if c mod d>0 then
  begin
    writeln('no answer');
    halt;
  end else
  begin
    x0:=x*(c div d);

```

```

    y0:=y*(c div d);
end;
end;
procedure fill(var a,b:node);
var t:longint;
begin
    if a[1]<b[0]-b[1] then t:=a[1]
                        else t:=b[0]-b[1];
    a[1]:=a[1]-t;
    b[1]:=b[1]+t;
end;
begin
    write('a,b,c,d=');
    readln(a[0],b[0],c[0],d);
    equation(a[0],b[0],d,x,y);
    step:=0;
    a[1]:=0;b[1]:=0;c[1]:=c[0];
    writeln(step:5,':',a[1]:5,b[1]:5,c[1]:5);
    if x>0 then
        repeat
            if a[1]=0 then fill(c,a) else
                if b[1]=b[0] then fill(b,c) else fill(a,b);
            inc(step);
            writeln(step:5,':',a[1]:5,b[1]:5,c[1]:5);
        until c[1]=d
    else
        repeat
            if b[1]=0 then fill(c,b) else
                if a[1]=a[0] then fill(a,c) else fill(b,a);
            inc(step);
            writeln(step:5,':',a[1]:5,b[1]:5,c[1]:5);
        until c[1]=d;
    end.

```

1.4 求 $a^b \bmod n$

1. 算法 8: 直接叠代法求 $a^b \bmod n$

$$d = a^b \bmod n = (\cdots (((a \bmod n) * a) \bmod n) * a) \bmod n \cdots * a) \bmod n$$



```

function f(a,b,n:longint): longint;
var d,i:longint;
begin
    d:=a;
    for i:=2 to b do d:=d mod n*a;
    d:=d mod n;
    f:=d;
end;

```

2. 算法 9: 加速叠代法

```

function f(a,b,n:longint):longint;
var d,t:longint;
begin
    d:=1;t:=a;

```

```

while b>0 do
  begin
if t=1 then begin
f:=d;exit end  ;
if b mod 2 =1 then d:=d*t mod n;
  b:=b div 2;
  t:=t*t mod n;
end;
f:=d
end;

```

练习：熟记并默写以上算法。

第二章 高精度计算

2.1 高精度加法

2.2 高精度减法

2.3 高精度乘法

2.4 高精度除法

2.1 高精度加法

高精度加法程序如下：

```

program HighPrecision1_Plus;
const
  fn_inp='hp1.inp';
  fn_out='hp1.out';
  maxlen=100; { max length of the number }
type
  hp=record
    len:integer; { length of the number }
    s:array[1..maxlen] of integer
    { s[1] is the lowest position
      s[len] is the highest position }
  end;
var
  x:array[1..2] of hp;
  y:hp; { x:input ; y:output }
  procedure PrintHP(const p:hp);
  var i:integer;
  begin
    for i:=p.len downto 1 do write(p.s[i]);
  end;
  procedure init;
  var
    st:string;
    j,i:integer;
  begin
    assign(input,fn_inp);
    reset(input);
    for j:=1 to 2 do
      begin
        readln(st);
        x[j].len:=length(st);
        for i:=1 to x[j].len do { change string to HP }
          x[j].s[i]:=ord(st[x[j].len+1-i])-ord('0');

```

```

end;
close(input);
end;
procedure Plus(a,b:hp;var c:hp); { c:=a+b }
var i,len:integer;
begin
  fillchar(c,sizeof(c),0);
  if a.len>b.len then len:=a.len { get the bigger length of a,b }
    else len:=b.len;
  for i:=1 to len do { plus from low to high }
  begin
    inc(c.s[i],a.s[i]+b.s[i]);
    if c.s[i]>=10 then
    begin
      dec(c.s[i],10);
      inc(c.s[i+1]); { add 1 to a higher position }
    end;
  end;
  if c.s[len+1]>0 then inc(len);
  c.len:=len;
end;
procedure main;
begin
  Plus(x[1],x[2],y);
end;
procedure out;
begin
  assign(output,fn_out);
  rewrite(output);
  PrintHP(y);
  writeln;
  close(output);
end;
begin
  init;
  main;
  out;
end.

```

2. 2 高精度减法

高精度减法程序如下:

```

program HighPrecision2_Subtract;
const
  fn_inp='hp2.inp';
  fn_out='hp2.out';
  maxlen=100; { max length of the number }
type
  hp=record
    len:integer; { length of the number }
    s:array[1..maxlen] of integer
    { s[1] is the lowest position
      s[len] is the highest position }
  end;

```

```

var
  x:array[1..2] of hp;
  y:hp; { x:input ; y:output }
  positive:boolean;
  procedure PrintHP(const p:hp);
  var i:integer;
  begin
    for i:=p.len downto 1 do write(p.s[i]);
  end;
  procedure init;
  var
    st:string;
    j,i:integer;
  begin
    assign(input,fn_inp);
    reset(input);
    for j:=1 to 2 do
      begin
        readln(st);
        x[j].len:=length(st);
        for i:=1 to x[j].len do { change string to HP }
          x[j].s[i]:=ord(st[x[j].len+1-i])-ord('0');
        end;
        close(input);
      end;
  end;
  procedure Subtract(a,b:hp;var c:hp); { c:=a-b, suppose a>=b }
  var i,len:integer;
  begin
    fillchar(c,sizeof(c),0);
    if a.len>b.len then len:=a.len { get the bigger length of a,b }
      else len:=b.len;
    for i:=1 to len do { subtract from low to high }
      begin
        inc(c.s[i],a.s[i]-b.s[i]);
        if c.s[i]<0 then
          begin
            inc(c.s[i],10);
            dec(c.s[i+1]); { add 1 to a higher position }
          end;
      end;
    while(len>1) and (c.s[len]=0) do dec(len);
    c.len:=len;
  end;
  function Compare(const a,b:hp):integer;
  {
    1 if a>b
    0 if a=b
    -1 if a<b
  }
  var len:integer;
  begin
    if a.len>b.len then len:=a.len { get the bigger length of a,b }
      else len:=b.len;

```

```

while(len>0) and (a.s[len]=b.s[len]) do dec(len);
{ find a position which have a different digit }
if len=0 then compare:=0 { no difference }
    else compare:=a.s[len]-b.s[len];
end;
procedure main;
begin
    if Compare(x[1],x[2])<0 then positive:=false
        else positive:=true;
    if positive then Subtract(x[1],x[2],y)
        else Subtract(x[2],x[1],y);
end;
procedure out;
begin
    assign(output,fn_out);
    rewrite(output);
    if not positive then write('-');
    PrintHP(y);
    writeln;
    close(output);
end;
begin
    init;
    main;
    out;
end.

```

2.3 高精度乘法

1. 高精度乘单精度(1 位数)

程序如下:

```

program HighPrecision3_Multiply1;
const
    fn_inp='hp3.inp';
    fn_out='hp3.out';
    maxlen=100; { max length of the number }
type
    hp=record
        len:integer; { length of the number }
        s:array[1..maxlen] of integer
        { s[1] is the lowest position
          s[len] is the highest position }
    end;
var
    x,y:hp; { x:input hp ; y:output }
    z:integer; { z:input lp }
    procedure PrintHP(const p:hp);
    var i:integer;
    begin
        for i:=p.len downto 1 do write(p.s[i]);
    end;
    procedure init;
    var
        st:string;

```

```

i:integer;
begin
  assign(input,fn_inp);
  reset(input);
  readln(st);
  x.len:=length(st);
  for i:=1 to x.len do { change string to HP }
    x.s[i]:=ord(st[x.len+1-i])-ord('0');
  readln(z);
  close(input);
end;
procedure Multiply(a:hp;b:integer;var c:hp); { c:=a*b }
var i,len:integer;
begin
  fillchar(c,sizeof(c),0);
  len:=a.len;
  for i:=1 to len do
  begin
    inc(c.s[i],a.s[i]*b);
    inc(c.s[i+1],c.s[i] div 10);
    c.s[i]:=c.s[i] mod 10;
  end;
  inc(len);
  while(c.s[len]>=10) do
  begin
    inc(c.s[len+1],c.s[len] div 10);
    c.s[len]:=c.s[len] mod 10;
    inc(len);
  end;
  while(len>1) and (c.s[len]=0) do dec(len);
  c.len:=len;
end;
procedure main;
begin
  Multiply(x,z,y);
end;
procedure out;
begin
  assign(output,fn_out);
  rewrite(output);
  PrintHP(y);
  writeln;
  close(output);
end;
begin
  init;
  main;
  out;
end.

```

2. 高精度乘一个整型数据(integer)

只需要将上述程序的 hp 类型定义如下即可:

```
type
```



```

hp=record
    len:integer { length of the number }
    s:array[1..maxlen] of longint
    { s[1]    is the lowest position
      s[len] is the highest position }
end;

```

3. 高精度乘高精度

程序如下:

```

program HighPrecision4_Multiply2;
const
    fn_inp='hp4.inp';
    fn_out='hp4.out';
    maxlen=100; { max length of the number }
type
    hp=record
        len:integer; { length of the number }
        s:array[1..maxlen] of integer
        { s[1]    is the lowest position
          s[len] is the highest position }
    end;
var
    x:array[1..2] of hp;
    y:hp; { x:input ; y:output }
    procedure PrintHP(const p:hp);
    var i:integer;
    begin
        for i:=p.len downto 1 do write(p.s[i]);
    end;
    procedure init;
    var
        st:string;
        j,i:integer;
    begin
        assign(input,fn_inp);
        reset(input);
        for j:=1 to 2 do
            begin
                readln(st);
                x[j].len:=length(st);
                for i:=1 to x[j].len do { change string to HP }
                    x[j].s[i]:=ord(st[x[j].len+1-i])-ord('0');
                end;
            end;
        close(input);
    end;
    procedure Multiply(a,b:hp;var c:hp); { c:=a+b }
    var i,j,len:integer;
    begin
        fillchar(c,sizeof(c),0);
        for i:=1 to a.len do
            for j:=1 to b.len do
                begin
                    inc(c.s[i+j-1],a.s[i]*b.s[j]);

```

```

        inc(c.s[i+j],c.s[i+j-1] div 10);
        c.s[i+j-1]:=c.s[i+j-1] mod 10;
    end;
len:=a.len+b.len+1;
{
    the product of a number with i digits and a number with j digits
    can only have at most i+j+1 digits
}
while(len>1)and(c.s[len]=0) do dec(len);
c.len:=len;
end;
procedure main;
begin
    Multiply(x[1],x[2],y);
end;
procedure out;
begin
    assign(output,fn_out);
    rewrite(output);
    PrintHP(y);
    writeln;
    close(output);
end;
begin
    init;
    main;
    out;
end.

```

2.4 高精度除法

1. 高精度除以整型数据(integer);

程序如下:

```

program HighPrecision3_Multiply1;
const
    fn_inp='hp5.inp';
    fn_out='hp5.out';
    maxlen=100; { max length of the number }
type
    hp=record
        len:integer; { length of the number }
        s:array[1..maxlen] of integer
        { s[1] is the lowest position
          s[len] is the highest position }
    end;
var
    x,y:hp;
    z,w:integer;
    procedure PrintHP(const p:hp);
    var i:integer;
    begin
        for i:=p.len downto 1 do write(p.s[i]);
    end;
    procedure init;

```

```

var
  st:string;
  i:integer;
begin
  assign(input,fn_inp);
  reset(input);
  readln(st);
  x.len:=length(st);
  for i:=1 to x.len do { change string to HP }
    x.s[i]:=ord(st[x.len+1-i])-ord('0');
  readln(z);
  close(input);
end;
procedure Divide(a:hp;b:integer;var c:hp;var d:integer);
{ c:=a div b ; d:=a mod b }
var i,len:integer;
begin
  fillchar(c,sizeof(c),0);
  len:=a.len;
  d:=0;
  for i:=len downto 1 do { from high to low }
  begin
    d:=d*10+a.s[i];
    c.s[i]:=d div b;
    d:=d mod b;
  end;
  while(len>1) and (c.s[len]=0) do dec(len);
  c.len:=len;
end;
procedure main;
begin
  Divide(x,z,y,w);
end;
procedure out;
begin
  assign(output,fn_out);
  rewrite(output);
  PrintHP(y);
  writeln;
  writeln(w);
  close(output);
end;
begin
  init;
  main;
  out;
end.

```

2. 高精度除以高精度

程序如下:

```

program HighPrecision4_Multiply2;
const
  fn_inp='hp6.inp';

```

```

fn_out='hp6.out';
maxlen=100; { max length of the number }
type
  hp=record
    len:integer; { length of the number }
    s:array[1..maxlen] of integer
    { s[1] is the lowest position
      s[len] is the highest position }
  end;
var
  x:array[1..2] of hp;
  y,w:hp; { x:input ; y:output }
  procedure PrintHP(const p:hp);
  var i:integer;
  begin
    for i:=p.len downto 1 do write(p.s[i]);
  end;
  procedure init;
  var
    st:string;
    j,i:integer;
  begin
    assign(input,fn_inp);
    reset(input);
    for j:=1 to 2 do
    begin
      readln(st);
      x[j].len:=length(st);
      for i:=1 to x[j].len do { change string to HP }
        x[j].s[i]:=ord(st[x[j].len+1-i])-ord('0');
      end;
      close(input);
    end;
  procedure Subtract(a,b:hp;var c:hp); { c:=a-b, suppose a>=b }
  var i,len:integer;
  begin
    fillchar(c,sizeof(c),0);
    if a.len>b.len then len:=a.len { get the bigger length of a,b }
      else len:=b.len;
    for i:=1 to len do { subtract from low to high }
    begin
      inc(c.s[i],a.s[i]-b.s[i]);
      if c.s[i]<0 then
      begin
        inc(c.s[i],10);
        dec(c.s[i+1]); { add 1 to a higher position }
      end;
    end;
    while(len>1) and (c.s[len]=0) do dec(len);
    c.len:=len;
  end;
  function Compare(const a,b:hp):integer;
  {

```

```

1 if a>b
0 if a=b
-1 if a<b
}
var len:integer;
begin
  if a.len>b.len then len:=a.len { get the bigger length of a,b }
    else len:=b.len;
  while(len>0) and (a.s[len]=b.s[len]) do dec(len);
  { find a position which have a different digit }
  if len=0 then compare:=0 { no difference }
    else compare:=a.s[len]-b.s[len];
end;
procedure Multiply10(var a:hp); { a:=a*10 }
var i:Integer;
begin
  for i:=a.len downto 1 do
    a.s[i+1]:=a.s[i];
  a.s[1]:=0;
  inc(a.len);
  while(a.len>1) and (a.s[a.len]=0) do dec(a.len);
end;
procedure Divide(a,b:hp;var c,d:hp); { c:=a div b ; d:=a mod b }
var i,j,len:integer;
begin
  fillchar(c,sizeof(c),0);
  len:=a.len;
  fillchar(d,sizeof(d),0);
  d.len:=1;
  for i:=len downto 1 do
  begin
    Multiply10(d);
    d.s[1]:=a.s[i]; { d:=d*10+a.s[i] }
    { c.s[i]:=d div b ; d:=d mod b; }
    { while(d>=b) do begin d:=d-b;inc(c.s[i]) end }
    while(compare(d,b)>=0) do
    begin
      Subtract(d,b,d);
      inc(c.s[i]);
    end;
  end;
  while(len>1)and(c.s[len]=0) do dec(len);
  c.len:=len;
end;
procedure main;
begin
  Divide(x[1],x[2],y,w);
end;
procedure out;
begin
  assign(output,fn_out);
  rewrite(output);
  PrintHP(y);

```

```

writeln;
PrintHP(w);
writeln;
close(output);
end;
begin
  init;
  main;
  out;
end.

```

练习：求 $n!$ ($n < 10000$) 的最后一位不为 0 的数字。

第三章 排列与组合

3.1 加法原理与乘法原理

3.2 排列与组合概念与计算公式

3.3 排列与组合的产生算法

3.1 加法原理与乘法原理

1. 加法原理：

做一件事情，完成它可以有 n 类办法，在第一类办法中有 m_1 种不同的方法，在第二类办法中有 m_2 种不同的方法，……，在第 n 类办法中有 m_n 种不同的方法。那么完成这件事共有 $N = m_1 + m_2 + \dots + m_n$ 种不同的方法。

2. 乘法原理：

做一件事情，完成它需要分成 n 个步骤，做第一步有 m_1 种不同的方法，做第二步有 m_2 种不同的方法，……，做第 n 步有 m_n 种不同的方法，那么完成这件事有 $N = m_1 * m_2 * \dots * m_n$ 种不同的方法。

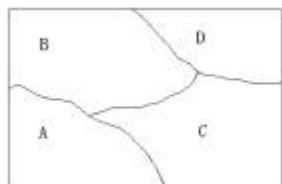
3. 两个原理的区别：一个与分类有关，一个与分步有关；加法原理是“分类完成”，乘法原理是“分步完成”。

练习：

1. 由数字 1, 2, 3, 4, 5 可以组成多少个三位数（各位上的数字允许重复）？

2. 由数字 0, 1, 2, 3, 4, 5 可以组成多少个三位数（各位上的数字允许重复）？

3. 由数字 0, 1, 2, 3, 4, 5 可以组成多少个十位数字大于个位数字的两位数？



例 4. 一个三位密码锁，各位上数字由 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 十个数字组成，可以设置多少种三位数的密码（各位上的数字允许重复）？首位数字不为 0 的密码数是多少种？首位数字是 0 的密码数又是多少种？

5. 如图，要给地图 A、B、C、D 四个区域分别涂上 3 种不同颜色中的某一种，允许同一种颜色使用多次，但相邻区域必须涂不同的颜色，不同的涂色方案有多少种？

6. 某班有 22 名女生，23 名男生。

① 选一位学生代表班级去领奖，有几种不同选法？

② 选出男学生与女学生各一名去参加智力竞赛，有几种不同的选法？

7. 105 有多少个约数？并将这些约数写出来。

8. 从 5 幅不同的国画、2 幅不同的油画、7 幅不同的水彩画中选不同画种的两幅画布置房间，有几种选法？

9. 若 x 、 y 可以取 1, 2, 3, 4, 5 中的任一个，则点 (x, y) 的不同个数有多少？

10. 一个口袋内装有 5 个小球另一个口袋内装有 4 个小球，所有这些小球的颜色各不相同① 从两个口袋内任取一个小球，有_____种不同的取法；

11. 从两个口袋内各取一个小球，有_____种不同的取法。

12. 乘积 $(a_1 + a_2 + a_3)(b_1 + b_2 + b_3 + b_4)(c_1 + c_2 + c_3 + c_4 + c_5)$ 展开共有_____个项。

13. 有四位考生安排在 5 个考场参加考试有_____种不同的安排方法。

(答案: 125; 180; 15; 1000; 900; 100; 6; 45; 506; 8; 59; 25; 9; 20; 60; 625)

3.2 排列与组合的概念与计算公式

1. 排列及计算公式

从 n 个不同元素中，任取 m ($m \leq n$) 个元素按照一定的顺序排成一列，叫做从 n 个不同元素中取出 m 个元素的一个排

列；从 n 个不同元素中取出 $m(m \leq n)$ 个元素的所有排列的个数，叫做从 n 个不同元素中取出 m 个元素的排列数，用符号 $p(n, m)$ 表示。

$p(n, m) = n(n-1)(n-2) \cdots (n-m+1) = n! / (n-m)!$ (规定 $0! = 1$)。

2. 组合及计算公式

从 n 个不同元素中，任取 $m(m \leq n)$ 个元素并成一组，叫做从 n 个不同元素中取出 m 个元素的一个组合；从 n 个不同元素中取出 $m(m \leq n)$ 个元素的所有组合的个数，叫做从 n 个不同元素中取出 m 个元素的组合数。用符号

$c(n, m)$ 表示。

$c(n, m) = p(n, m) / m! = n! / ((n-m)! * m!)$ ； $c(n, m) = c(n, n-m)$ ；

3. 其他排列与组合公式

从 n 个元素中取出 r 个元素的循环排列数 $= p(n, r) / r = n! / r(n-r)!$ 。

n 个元素被分成 k 类，每类的个数分别是 n_1, n_2, \dots, n_k 这 n 个元素的全排列数为

$n! / (n_1! * n_2! * \dots * n_k!)$ 。

k 类元素，每类的个数无限，从中取出 m 个元素的组合数为 $c(m+k-1, m)$ 。

练习：

1. (1) 用 0, 1, 2, 3, 4 组合多少无重复数字的四位数？(96)

(2) 这四位数中能被 4 整除的数有多少个？(30)

(3) 这四位数中能被 3 整除的数有多少个？(36)

2. 用 0, 1, 2, 3, 4 五个数字组成无重复数字的五位数从小到大依次排列。

(1) 第 49 个数是多少？(30124)

(2) 23140 是第几个数？(40)

3. 求下列不同的排法种数：

(1) 6 男 2 女排成一行，2 女相邻；($p(7,7) * p(2,2)$)

(2) 6 男 2 女排成一行，2 女不能相邻；($p(6,6) * p(7,2)$)

(3) 5 男 3 女排成一行，3 女都不能相邻；($p(5,5) * p(6,3)$)

(4) 4 男 4 女排成一行，同性者相邻；($p(4,4) * p(4,4) * p(2,2)$)

(5) 4 男 4 女排成一行，同性者不能相邻。($p(4,4) * p(4,4) * p(2,2)$)

4. 有四位医生、六位护士、五所学校。

(1) 若要选派三位医生到五所学校之中的三所学校举办健康教育讲座，每所学校去一位医生有多少种不同的选派方法？

($c(5,3) * p(4,3)$)

(2) 在医生或护士中任选五人，派到五所学校进行健康情况调查，每校去且仅去一人，有多少种不同的选派方法？

($p(10,5)$)

(3) 组成三个体检小组，每组一名医生、两名护士，到五所学校中的三所学校为老师体检，有多少种不同的选派方法？

($c(5,3) * p(4,3) * c(6,2) * c(4,2) * c(2,2)$)

5. 平面上有三条平行直线，每条直线上分别有 7, 5, 6 个点，且不同直线上三个点都不在同一条直线上。问用这些点为顶点，能组成多少个不同四边形？(2250)

6. 平面上有三条平行直线，每条直线上分别有 7, 5, 6 个点，且不同直线上三个点都不在同一条直线上。问用这些点为顶点，能组成多少个不同三角形？(751)

7. 将 N 个红球和 M 个黄球排成一行。例如： $N=2, M=2$ 可得到以下 6 种排法：

红红黄黄 红黄红黄 红黄黄红 黄红红黄 黄红黄红 黄黄红红

问题：当 $N=4, M=3$ 时有多少种不同排法？(不用列出每种排法) (35)

8. 用 20 个不同颜色的念珠穿成一条项链，能做多少个不同的项链。(20!/20)

9. 在单词 MISSISSIPPI 中字母的排列数是 ($11! / (1! * 4! * 4! * 2!)$)

10. 求取自 $1, 2, \dots, k$ 的长为 r 的非减序列的个数为 ($c(r+k-1, r)$)

3.3 排列与组合的产生算法

1. 排列的产生

方法 1：（递归，深度优先产生）

程序如下：

```
program pailei;
const m=4;
var a:array[1..m] of integer ;
```

```

    b:array[1..m] of boolean;
procedure print;
var i:integer;
begin
    for i:=1 to m do
        write(a[i]);
        writeln;
    end;
procedure try(dep:integer);
var i:integer;
begin
    for i:=1 to m do
        if b[i] then
            begin
                a[dep]:=i; b[i]:=false;
                if dep=m then print else try(dep+1);
                b[i]:=true;
            end;
    end;
begin
    fillchar(b,sizeof(b),true);
    try(1);
end.

```

方法 2. 根据上一个排列产生下一个排列.

程序如下:

```

program pailei;
const m=5;
var a:array[1..m] of integer ;
    i,j,temp,k,l:integer;
procedure print;
var i:integer;
begin
    for i:=1 to m do
        write(a[i]);
        writeln;
    end;
begin
    for i:=1 to m do a[i]:=i;
    repeat
        print;
        i:=m-1;
        while (i>0) and (a[i]>a[i+1]) do i:=i-1;
        if i>0 then
            begin
                j:=m;
                while a[j]<a[i] do j:=j-1;
                temp:=a[i];a[i]:=a[j];a[j]:=temp;
                k:=i+1;l:=m;
                while k<l do
                    begin
                        temp:=a[k];a[k]:=a[l];a[l]:=temp;
                        k:=k+1;l:=l-1
                    end;
            end;
    until i=0;
end.

```



```

    end;
  end;
until i=0;
end.

```

2. 组合的产生算法

算法 1: (递归, 深度优先产生)

程序如下:

```

program zuhe;
const n=6;m=4;
var a:array[0..m] of integer;
    i,j:integer;
procedure print;
var i:integer;
begin
  for i:=1 to m do write(a[i]);
  writeln;
end;
procedure try(dep:integer);
var i:integer;
begin
  for i:=a[dep-1]+1 to n-(m-dep) do
    begin
      a[dep]:=i;
      if dep=m then print else try(dep+1);
    end
  end;
end;
begin
  a[0]:=0;
  try(1);
end.

```

算法 2: 根据前一个组合产生下一个组合

程序如下:

```

program zuhe;
const n=6;m=4;
var a:array[1..m] of integer;
    i,j:integer;
procedure print;
var i:integer;
begin
  for i:=1 to m do write(a[i]);
  writeln;
end;
begin
  for i:=1 to m do
    a[i]:=i;
  repeat
    print;
    i:=m;
  while (i>0) and (a[i]=n-(m-i)) do dec(i);
  if i>0 then
    begin
      a[i]:=a[i]+1;

```

```

for j:=i+1 to m do a[j]:=a[j-1]+1;
end;
until i=0;
end.

```

练习：

1. 已知 n ($1 \leq n \leq 20$) 个整数 x_1, x_2, \dots, x_n ($1 \leq x_i \leq 5000000$)，以及一个整数 k ($k \leq n$)。从 n 个整数中任选 k 个整数相加，可分别得到一系列的和。现在，要求你计算出和为素数共有多少种。

2. n 个部件, 每个部件必须经过先 A 后 B 两道工序。

以知部件 i 在 A, B 机器上的时间分别为 a_i, b_i 。如何安排加工顺序，总加工时间最短？

输入： 5

| | | | | | |
|-------|---|---|---|---|----|
| 部件 | 1 | 2 | 3 | 4 | 5 |
| a_i | 3 | 5 | 8 | 7 | 10 |
| b_i | 6 | 2 | 1 | 4 | 9 |

输出：

34

1 5 4 2 3

第四章 计算几何

4.1 基础知识

4.2 线段相交判断

4.3 寻找凸包算法

4.1 基础知识

1. 两点间的距离公式：

已知：平面上的两点的直角坐标分别为 $P_1(x_1, y_1)$, $P_2(x_2, y_2)$ ，则 P_1 和 P_2 两点间的距离为

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

2. 线段的中点坐标公式：

已知：平面上的两点的直角坐标分别为 $P_1(x_1, y_1)$, $P_2(x_2, y_2)$ ，则线段 P_1P_2 的中点坐标为

$$x = (x_1 + x_2) / 2 \quad y = (y_1 + y_2) / 2$$

3. 直线的斜率公式：

已知：平面上的两点的直角坐标分别为 $P_1(x_1, y_1)$, $P_2(x_2, y_2)$ ，则线段 P_1P_2 所在的直线的斜率为

$$k = (y_2 - y_1) / (x_2 - x_1)$$

4. 直线的点斜式方程：

已知：直线过点 $P_0(x_0, y_0)$ ，斜率为 k ，则该直线所在的方程为

$$y - y_0 = k(x - x_0) \quad \text{或} \quad y = kx + b \quad (\text{与 } y \text{ 轴交点的纵坐标: 纵截距})$$

练习

1. 已知: 矩形上三点的坐标 $p_1(x_1, y_1)$, $p_2(x_2, y_2)$, $p_3(x_3, y_3)$

(1) 求矩形另外一点的坐标 $p_4(x_4, y_4)$ 。

(2) 判断点 $p(x, y)$ 是在矩形内、矩形外还是在矩形的边上。

4.2 线段的相交判断

1. 叉积

已知：平面上的两点的直角坐标分别为 $p_1(x_1, y_1)$, $p_2(x_2, y_2)$ 则

(1) 该两点相对坐标原点 $(0, 0)$ 的叉积为 $m = x_1 * y_2 - x_2 * y_1$

若 $m > 0$ 则相对坐标原点, 点 p_1 在点 p_2 的顺时针方向

若 $m < 0$ 则相对坐标原点, 点 p_1 在点 p_2 的逆时针方向

若 $m = 0$ 则原点和 p_1 、 p_2 在一条直线上

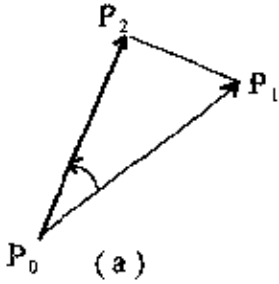
(2) 该两点相对点 $p_0(x_0, y_0)$ 的叉积为 $m = (x_1 - x_0) * (y_2 - y_0) - (x_2 - x_0) * (y_1 - y_0)$

若 $m > 0$ 则相对 p_0 点, 点 p_1 在点 p_2 的顺时针方向

若 $m < 0$ 则相对 p_0 点, 点 p_1 在点 p_2 的逆时针方向

若 $m = 0$ 则 p_0 和 p_1 、 p_2 在一条直线上

2. 确定两条连续的有向线段 p_0p_1 和 p_1p_2 在 p_1 点是向左转还是向右转



(1) 计算叉积 $m = (x_1 - x_0) * (y_2 - y_0) - (x_2 - x_0) * (y_1 - y_0)$

(2) 判断 m

若 $m > 0$ 则 p_1 点向左拐

若 $m < 0$ 则 p_1 点向右拐

若 $m = 0$ 则点 p_0 、 p_1 、 p_2 在一条直线上

3. 确定两条线段 p_1p_2 、 p_3p_4 是否相交

程序如下：

```
program xdxj;
type p=record
    x, y:real
end;
var p1,p2,p3,p4:p;
function m(p1,p2,p0:p):real;
begin
    m:=(p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
end;
function max(a,b:real):real;
begin
    if a>b then max:=a else max:=b;
end;
function min(a,b:real):real;
begin
    if a<b then min:=a else min:=b;
end;
function across(p1,p2,p3,p4:p):boolean;
begin
    if (max(p1.x,p2.x)>=min(p3.x,p4.x)) and
       (max(p3.x,p4.x)>=min(p1.x,p2.x)) and
       (max(p1.y,p2.y)>=min(p3.y,p4.y)) and
       (max(p3.y,p4.y)>=min(p1.y,p2.y)) and
       (m(p2,p3,p1)*m(p2,p4,p1)<0) and (m(p4,p1,p3)*m(p4,p2,p3)<0)
    then across:=true else across:=false;
end;
begin
    readln(p1.x,p1.y,p2.x,p2.y);
    readln(p3.x,p3.y,p4.x,p4.y);
    if across(p1,p2,p3,p4) then writeln('yes') else writeln('no');
end.
```

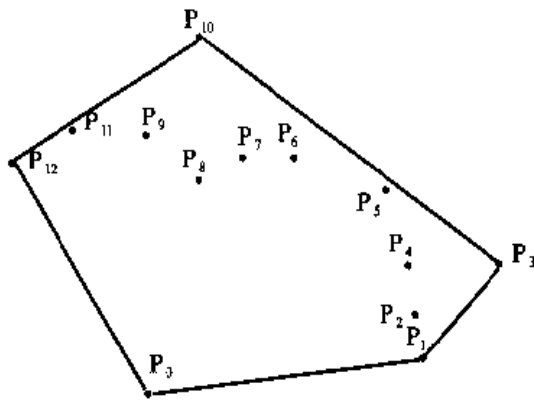
4.3 寻找凸包算法

1. 凸包的概念

一个点集 $Q = (p_0, p_1, p_2, \dots, p_{n-1})$ ，它的凸包是一个最小的凸多边形 P ，且满足 Q 中的每个点或者在 P 的边界上，或者在 P 的内部。在直观上，我们可以把 Q 中的每个点看作露在板外的铁钉。那么凸包就是包含所有铁钉的一个拉紧的橡皮绳

所构成的形状.

如图:



2. 寻找凸包算法

算法如下(Graham 算法):

- 1) 求 q 中 y 坐标最小的点 p_0 , 若具有最小坐标的点有多个, 则取最左边的点作为 p_0 .
- 2) 对 q 中剩余的点按逆时针相对 p_0 的极角排序, 若有数个保留其中距 p_0 最远的点得到序列 $(p_1, p_2, \dots, p_{n-1})$;
- 3) p_0, p_1, p_2 相继入栈
- 4) for $i=3$ to $n-1$ do
 - 1) while 由次栈顶元素、栈顶元素和 P_i 所形成角不是向左转 do 栈顶元素出栈 s ;
 - 2) p_i 入栈
- 5) 打印按逆时针排列的栈中的各顶点

程序如下:

```

program tubao;
const maxn=500;
type p=record
    x,y:real;
end;
var n,top:integer;
list:array[0..maxn]of p;
s:array[1..maxn] of integer;
f:text;
procedure swap(var a,b:p);
var t:p;
begin t:=a;a:=b;b:=t end;
function m(p1,p2,p0:p):real;
begin
    m:=(p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
end;
function comp(p1,p2:p):boolean;
var t:real;
begin
    t:=m(p1,p2,list[0]);
    if (t>0) or (t=0) and (sqr(p1.x-list[0].x)+sqr(p1.y-list[0].y)<
        sqr(p2.x-list[0].x)+sqr(p2.y-list[0].y))
    then comp:=true else comp:=false;
end;
procedure sort(l,r:integer);
var i,j:integer;
x:p;

```

```

begin
  if r-l+1<=5 then begin
    for j:=l+1 to r do
      begin
        i:=j;
        while (i>l) and comp(list[i],list[i-1]) do
          begin swap(list[i],list[i-1]); dec(i) end
        end;
      end else
      begin
        x:=list[l+random(r-l+1)];
        i:=l;j:=r;
        repeat
          while comp(list[i],x) do inc(j);
          while comp(x,list[j]) do dec(j);
          if i<j then swap(list[i],list[j])
        until i>=j;
        sort(l,j);
        sort(j+1,r);
      end
    end;
  end;
  procedure init;
  var i:integer;
  begin
    assign(f,'input.txt');
    reset(f);
    readln(f,n);
    for i:=0 to n-1 do
      begin
        readln(f,list[i].x,list[i].y);
        if (list[i].y<list[0].y) or
          (list[i].y=list[0].y) and (list[i].x<list[0].x)
          then swap(list[0],list[i]);
        end ;
      end;
    sort(1,n-1)
  end;
  procedure graham;
  var i:integer;
  begin
    for i:=1 to 3 do s[i]:=i-1;
    top:=3;
    for i:=3 to n-1 do
      begin
        while m(list[i],list[s[top]],list[s[top-1]])>=0 do dec(top);
        inc(top);
        s[top]:=i;
      end;
    for i:=1 to top do
      write('(',list[s[i]].x:7:2,',',list[s[i]].y:7:2,')');
    writeln
  end;
begin
  init;

```

```

graham;
readln
end.

```

练习:已知:平面上有 n 个点 ($n \leq 10000$), 用 Graham 算法找出彼此间最远的两个点.

第五章 其它数学知识及算法

5.1 鸽巢原理

5.2 容斥原理

5.3 常见递推关系及应用

5.1 鸽巢原理

1. 简单形式

如果 $n+1$ 个物体被放进 n 个盒子, 那么至少有一个盒子包含两个或更多的物体。

例 1: 在 13 个人中存在两个人, 他们的生日在同一个月里。

例 2: 设有 n 对已婚夫妇。为保证有一对夫妇被选出, 至少要从这 $2n$ 个人中选出多少人? ($n+1$)

2. 加强形式

令 q_1, q_2, \dots, q_n 为正整数。如果将

$q_1 + q_2 + \dots + q_n - n + 1$ 个物体放入 n 个盒子内, 那么或者第一个盒子至少含有 q_1 个物体, 或者第二个盒子至少含有 q_2 个物体, \dots , 或者第 n 个盒子含有 q_n 个物体。

例 3: 一篮子水果装有苹果、香蕉、和橘子。为了保证篮子内或者至少 8 个苹果或者至少 6 个香蕉或者至少 9 个橘子, 则放入篮子中的水果的最小件数是多少? (21 件)

5.2 容斥原理及应用

原理: 集 S 的不具有性质 P_1, P_2, \dots, P_m 的物体的个数由下式给出:

$$|A_1 \cap A_2 \cap \dots \cap A_m| = |S| - \sum |A_i| + \sum |A_i \cap A_j| - \sum |A_i \cap A_j \cap A_k| + \dots + (-1)^m |A_1 \cap A_2 \cap \dots \cap A_m|$$

如: $m=3$, 时上式为:

$$|A_1 \cap A_2 \cap A_3| = |S| - (|A_1| + |A_2| + |A_3|) + (|A_1 \cap A_2| + |A_1 \cap A_3| + |A_2 \cap A_3|) - |A_1 \cap A_2 \cap A_3|$$

推论: 至少具有性质 P_1, P_2, \dots, P_m 之一的集合 S 的物体的个数有:

$$|A_1 \cup A_2 \cup \dots \cup A_m| = |S| - |A_1 \cap A_2 \cap \dots \cap A_m| + \sum |A_i| - \sum |A_i \cap A_j| + \sum |A_i \cap A_j \cap A_k| + \dots + (-1)^{m+1} |A_1 \cap A_2 \cap \dots \cap A_m|$$

例 4: 求从 1 到 1000 不能被 5, 6, 和 8 整除的整数的个数?

$$(1000 - (200 + 166 + 125) + (33 + 25 + 41) - 8) = 600$$

5.3 常见递推关系及应用

1. 算术序列

每一项比前一项大一个常数 d ;

若初始项为 h_0 : 则递推关系为 $h_n = h_{n-1} + d = h_0 + nd$;

对应的各项为: $h_0, h_0 + d, h_0 + 2d, \dots, h_0 + nd$;

前 n 项的和为 $(n+1)h_0 + dn(n+1)/2$

例 5: 1, 2, 3, \dots

例 6: 1, 3, 5, 7, \dots 等都是算术序列。

2. 几何序列

每一项是前面一项的常数 q 倍

若初始项为 h_0 : 则递推关系为 $h_n = h_0 q^{n-1} q = h_0 q^n$;

对应的各项为: $h_0, h_0 q^1, h_0 q^2, \dots, h_0 q^n$

例 7: 1, 2, 4, 8, 16, \dots

例 8: 5, 15, 45, 135, \dots 等都是几何序列;

前 n 项和为 $((q^{n+1} - 1) / (q - 1)) h_0$

3. Fibonacci 序列

除第一、第二项外每一项是它前两项的和;

若首项为 f_0 为 0, 则序列为 0, 1, 1, 2, 3, 5, 8, \dots 递推关系为 ($n \geq 2$) $f_n = f_{n-1} + f_{n-2}$

前 n 项的和 $S_n = f_0 + f_1 + f_2 + \dots + f_n = f_{n+2} - 1$

例 9: 以下是 Fibonacci 的示例:

1. 楼梯有 n 阶台阶, 上楼可以一步上 1 阶, 也可以一步上 2 阶, 编一程序计算共有多少种不同的走法?
2. 有一对雌雄兔, 每两个月就繁殖雌雄各一对兔子. 问 n 个月后共有多少对兔子?
3. 有 $n \times 2$ 的一个长方形方格, 用一个 1×2 的骨牌铺满方格. 求铺法总数?

4. 错位排列

首先看例题:

例 10: 在书架上放有编号为 $1, 2, \dots, n$ 的 n 本书. 现将 n 本书全部取下然后再放回去, 当放回去时要求每本书都不能放在原来的位置上。

例如: $n=3$ 时:

原来位置为: 123

放回去时只能为: 312 或 231 这两种

问题: 求当 $n=5$ 时满足以上条件的放法共有多少种? (不用列出每种放法) (44)

$\{1, 2, 3, \dots, n\}$ 错位排列是 $\{1, 2, 3, \dots, n\}$ 的一个排列 $i_1 i_2 \dots i_n$, 使得 $i_1 \neq 1, i_2 \neq 2, i_3 \neq 3, \dots, i_n \neq n$

错位排列数列为

0, 1, 2, 9, 44, 265, ...

错位排列的递推公式是: $d_n = (n-1)(d_{n-2} + d_{n-1}) \quad (n \geq 3)$

$$= n d_{n-1} + (-1)^{n-2}$$

5. 分平面的最大区域数

1. 直线分平面的最大区域数的序列为:

2, 4, 7, 11, ...

递推公式是: $f_n = f_{n-1} + n = n(n+1)/2 + 1$

2. 折线分平面的最大区域数的序列为:

2, 7, 16, 29, ...

递推公式是: $f_n = (n-1)(2n-1) + 2n$;

3. 封闭曲线 (如一般位置上的圆) 分平面的最大区域数的序列为:

2, 4, 8, 14, ...

递推公式是: $f_n = f_{n-1} + 2(n-1) = n^2 - n + 2$

6. Catalan 数列

先看下面两个例题:

例 11: 将一个凸多边形区域分成三角形区域的方法数?

令 h_n 表示具有 $n+1$ 条边的凸多边形区域分成三角形的方法数. 同时令 $h_1=1$, 则 h_n 满足递推关系

$h_n = h_1 h_{n-1} + h_2 h_{n-2} + \dots + h_{n-1} h_1 \quad (n \geq 2)$ (想一想, 为什么?)

该递推关系的解为 $h_n = c(2n-2, n-1)/n \quad (n=1, 2, 3, \dots)$

其对应的序列为 1, 1, 2, 5, 14, 42, 132, ... 从第二项开始分别是三角形, 四边形, ... 的分法数

即 k 边形分成三角形的方法数为 $h_k = c(2k-4, k-2)/(k-1) \quad (k \geq 3)$

例 12: n 个 +1 和 n 个 -1 构成 $2n$ 项 a_1, a_2, \dots, a_{2n}

其部分和满足 $a_1 + a_2 + \dots + a_k \geq 0 \quad (k=1, 2, 3, \dots, 2n)$ 对与 n 该数列为

$C_n = c(2k, k)/(k+1) \quad (k \geq 0)$ 对应的序列为 1, 1, 2, 5, 14, 42, 132, ...

序列 1, 1, 2, 5, 14, 42, 132, ... 叫 Catalan 数列。

例 13: 下列问题都是 Catalan 数列。

1. 有 $2n$ 个人排成一行进入剧场. 入场费 5 元. 其中只有 n 个人有一张 5 元钞票, 另外 n 人只有 10 元钞票, 剧院无其它钞票, 问有多少中方法使得只要有 10 元的人买票, 售票处就有 5 元的钞票找零?
2. 一位大城市的律师在她住所以北 n 个街区和以东 n 个街区处工作. 每天她走 $2n$ 个街区去上班. 如果他从不穿越 (但可以碰到) 从家到办公室的对角线, 那么有多少条可能的道路?
3. 在圆上选择 $2n$ 个点, 将这些点成对连接起来使得所得到的 n 条线段不相交的方法数?
4. n 个结点可构造多少个不同的二叉树?
5. 一个栈 (无穷大) 的进栈序列为 $1, 2, 3, \dots, n$, 有多少个不同的出栈序列?

第七部分 图论算法

第一章 最小生成树

1.1 实际背景与算法

1.2 例题与习题

1.1 实际背景与算法

实际背景：

要在 n 个居民点之间架设煤气管道, 很显然最多可架设 $n(n-1)/2$ 条管道, 然而要

连通 n 个居民点架设 $n-1$ 条管道(无环出现)就可以了, 为了使造价最小, 选择哪 $n-1$ 条边? 这就是最小生成树问题。

算法 1 (prim 算法)：

1) 图采用邻接矩阵存储。

2) 找到目前情况下能连上的权值最小的边的另一端点, 加入之, 直到所有的顶点加入完毕。

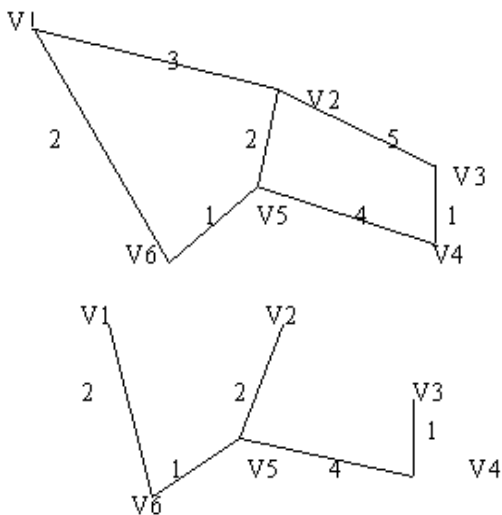
算法 2 (kruskal 算法)：

1) 图采用边目录方式存储。

2) 选目前情况下能连上的权值最小的边, 若与以生成的树不够成环, 加入之, 直到 $n-1$ 条边加入完毕。

1.2 例题与习题

1: 求下图的最小生成树



程序 1 (使用算法 1) 如下:

```
program mintree;
const maxn=100;
var cost:array[1..maxn,1..maxn] of integer;
    lowcost,closet:array[1..maxn] of integer;
    n,mincost:integer;
procedure init;
var i,j:integer;
begin
    readln(n);
    for i:=1 to n do
        for j:=1 to n do
            read(cost[i,j]);
    for i:=1 to n do
        begin
            lowcost[i]:=cost[1,i];
            closet[i]:=1
        end ;
    mincost:=0;
end;
procedure prim;
var i,j,k,min:integer;
```



```

begin
  for i:=1 to n-1 do
    begin
      min:=32767;
      for j:=1 to n do
        if (lowcost[j]<>0) and (lowcost[j]<min) then
          begin
            min:=lowcost[j];k:=j;
          end;
      mincost:=mincost+cost[closet[k],k];
      writeln('(',closet[k],',',k,')');
      lowcost[k]:=0;
      for j:=1 to n do
        if cost[k,j]<lowcost[j] then
          begin lowcost[j]:=cost[k,j];closet[j]:=k end;
      end;
    end;
begin
  init;
  prim;
  writeln('treeminlength=',mincost);
  readln;
end.

```

程序 2 (使用算法 2) 如下:

```

program shchtree;
var n,m,i,j:integer;
    selected:array[1..100] of integer;
    e:array[1..100,1..2] of integer;
    value:array[1..100] of integer;
    t:array[1..100] of integer;
    min,mine,valuets:integer;
begin
  write('Input n and m:');read(n,m);
  writeln('input data:');
  for i:=1 to m do readln(e[i,1],e[i,2],value[i]);
  fillchar(selected,sizeof(selected),0);
  fillchar(t,sizeof(t),0);
  valuets:=0;
  for i:=1 to n-1 do
    begin
      min:=maxint;
      mine:=0;
      for j:=1 to m do
        if selected[j]=0 then
          if ((t[e[j,1]]=0) xor (t[e[j,2]]=0)) or (i=1) then
            if value[j]<min then
              begin min:=value[j];mine:=j; end;
      selected[mine]:=1;
      t[e[mine,1]]:=1;
      t[e[mine,2]]:=1;
      valuets:=valuets+min;
    end;
end;

```

```

for i:=1 to m do
  if selected[i]=1 then
    begin writeln('(',e[i,1],',',e[i,2],',')'); end;
    writeln('tree:  ', 'length=',valuet);
  readln;
end.

```

2:最小生成树练习

现在政府计划在某个区域内的的城市间架设高速公路，以使任意两个城市间能够直接或间接到达，怎样修路，费用最小。

输入文件：第一行一个整数 n ($n \leq 100$) 表示城市数目。

第二行至第 $n+1$ 行每行两个数 x_i, y_i ($0 \leq x_i, y_i \leq 100$) 表示第 i 个城市的坐标；

输出最小费用。

第二章 最短路径

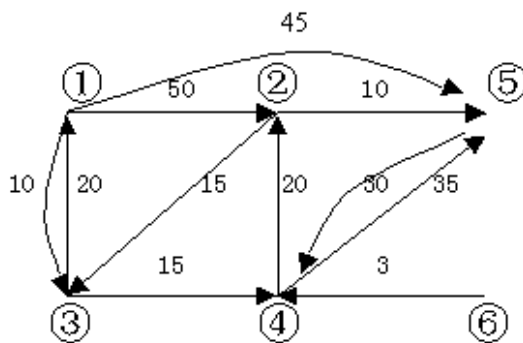
2.1 单对顶点间的最短路径

2.2 一点到其它所有点的最短路径

2.3 所有点间的最短路径

2.1 单对顶点间的最短路径

例 1：输入起点，终点，求其最短路径？



1. 下面是 Dijkstra 算法：

基本思想是：设置一个顶点的集合 s ，并不断地扩充这个集合，一个顶点属于集合 s 当且仅当从源点到该点的路径已求出。开始时 s 中仅有源点，并且调整非 s 中点的最短路径长度，找当前最短路径点，将其加入到集合 s ，直到终点在 s 中。

```

program zudloutj;
const n=6;max=10000 ;
      cost:array[1..6,1..6] of real=((0,50,10,max,45,max),(max,0
      ,15,max,10,max),(20,max,0,15,max,max),(max,20,max,0,35,max),(
      max,max,max,30,0,max),(max,max,max,3,max,0));
var dist:array[1..n] of real;
    path,p:array[1..n] of 1..n;
    first,tail,u:1..n;
    s:set of 1..n;
    i,j,y,m:integer;
    min:real;
begin
  read(first,tail);
  for i:=1 to n do dist[i]:=max;
  dist[first]:=0;
  s:=[first]; u:=first;
  while u<>tail do
    begin
      for j:= 1 to n do

```

```

    if not(j in s) and (dist[u]+cost[u,j]<dist[j]) then
        begin dist[j]:=dist[u]+cost[u,j];path[j]:=u end;
min:=max;
for j:=1 to n do
    if not(j in s) and (dist[j]<min) then begin u:=j;min:=dist[j];end;
if min=max then begin writeln('No answer');halt end;
s:=s+[u];
end;
writeln('mindist(',first,',',tail,')=',dist[tail]:8:2);
y:=tail;m:=0;
while (y<>first) do
    begin inc(m);p[m]:=y;y:=path[y]; end;
write('path:',first);
for j:=m downto 1 do
    write('->',p[j]);
writeln;
end.

```

2. 迭代算法(ford 算法)

Dijkstra 算法只能适合权为正值的情况,但当权为负值时,是 Dijkstra 算法爱莫能助,这时只要图中不存在负权回路可用迭代算法。

迭代算法的思想:不停地调整图中的顶点值(源点到该点的最短路径值),直到没有点的值调整了为止。

程序如下:

```

program zudloutj;
const n=6;max=10000 ;
    cost:array[1..6,1..6] of real=((0,50,10,max,45,max),(max,0
    ,15,max,10,max),(20,max,0,15,max,max),(max,20,max,0,35,max),(
    max,max,max,30,0,max),(max,max,max,3,max,0));
var dist:array[1..n] of real;
    path,p:array[1..n] of 1..n;
    first,tail,u:1..n;
    s:set of 1..n;
    i,j,y,m:integer;
    min:real;
    quit:boolean;
begin
    read(first,tail);
    for i:=1 to n do dist[i]:=max;
    dist[first]:=0;
    repeat
        quit:=true;
        for i:=1 to n do
            if dist[i]<max then
                for j:=1 to n do
                    if dist[i]+cost[i,j]<dist[j] then
                        begin
                            dist[j]:=dist[i]+cost[i,j];
                            path[j]:=i;
                            quit:=false;
                        end;
                end;
        until quit;
    writeln('mindist(',first,',',tail,')=',dist[tail]:8:2);
    y:=tail;m:=0;

```

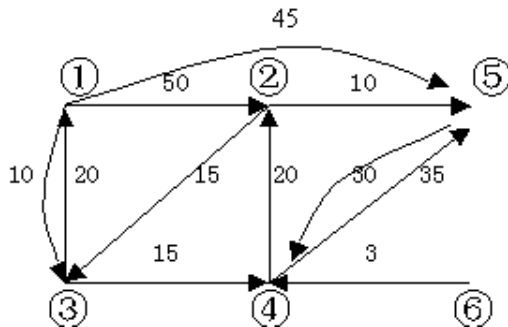
```

while (y<>first) do
  begin inc(m);p[m]:=y;y:=path[y]; end;
write('path:',first);
for j:=m downto 1 do
  write('->',p[j]);
writeln;
end.

```

2. 2 一点到其它所有点的最短路径

例 2：如下图：求起点到所有点的最短路径？



1. 下面是 Dijkstra 算法：

基本思想是：设置一个顶点的集合 s ，并不断地扩充这个集合，一个顶点属于集合 s 当且仅当从源点到该点的路径已求出。开始时 s 中仅有源点，并且调整非 s 中点的最短路径长度，找当前最短路径点，将其加入到集合 s 中，直到所有的点都在 s 中。

```

program zudlounj;
const n=6;max=10000 ;
      cost:array[1..6,1..6] of real=((0,50,10,max,45,max),(max,0
      ,15,max,10,max),(20,max,0,15,max,max),(max,20,max,0,35,max),(
      max,max,max,30,0,max),(max,max,max,3,max,0));
var dist:array[1..n] of real;
    path,p:array[1..n] of 1..n;
    first,u:1..n;
    s:set of 1..n;
    i,j,y,m:integer;
    min:real;
begin
  read(first);
  for i:=1 to n do dist[i]:=max;
  dist[first]:=0;
  s:=[first]; u:=first;
  for i:=1 to n-1 do
    begin
      for j:= 1 to n do
        if not(j in s) and (dist[u]+cost[u,j]<dist[j]) then
          begin dist[j]:=dist[u]+cost[u,j];path[j]:=u end;
      min:=max;
      for j:=1 to n do
        if not(j in s) and (dist[j]<min) then begin u:=j;min:=dist[j];end;
      s:=s+[u];
    end;
  for i:=1 to n do
    if i<>first then
      if dist[i]<>max then
        begin
          writeln('mindist(','first',' ',i,')=',dist[i]:8:2);

```

```

y:=i;m:=0;
while (y<>first) do
  begin inc(m);p[m]:=y;y:=path[y] end;
write('path:',first);
for j:=m downto 1 do
write('->',p[j]);
writeln;
end
else
  writeln('mindist(',first,',',i,'):', 'No answer');
end.

```

2. ford 算法

```

program zudlouj;
const n=6;max=10000 ;
      cost:array[1..6,1..6] of real=((0,50,10,max,45,max), (max,0
      ,15,max,10,max), (20,max,0,15,max,max), (max,20,max,0,35,max), (
      max,max,max,30,0,max), (max,max,max,3,max,0));
var dist:array[1..n] of real;
    path,p:array[1..n] of 1..n;
    first,tail,u:1..n;
    s:set of 1..n;
    i,j,y,m:integer;
    min:real;
    quit:boolean;
begin
  read(first);
  for i:=1 to n do dist[i]:=max;
  dist[first]:=0;
  repeat
    quit:=true;
    for i:=1 to n do
      if dist[i]<max then
        for j:=1 to n do
          if dist[i]+cost[i,j]<dist[j] then
            begin
              dist[j]:=dist[i]+cost[i,j];
              path[j]:=i;
              quit:=false;
            end;
        until quit;
    for i:=1 to n do
      if i<>first then
        if dist[i]<>max then
          begin
            writeln('mindist(',first,',',i,')=',dist[i]:8:2);
            y:=i;m:=0;
            while (y<>first) do
              begin inc(m);p[m]:=y;y:=path[y] end;
            write('path:',first);
            for j:=m downto 1 do
              write('->',p[j]);
            writeln;
          end;
        end;
    end.

```

```

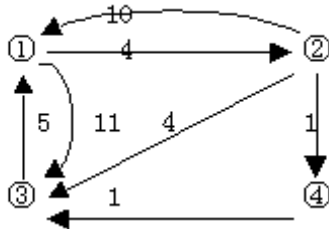
end
else
  writeln('mindist(',first,',',i,'):','No answer');
end.

```

2. 3 所有点间的最短路径

弗洛耶德算法（Floyed 算法）：

基本思想：求解所有点间的路径需要进行 n 次试探。对于顶点 i 到顶点 j 的路径长度，首先考虑让路径经过顶点 1，比较路径 (i, j) 和 $(i, 1, j)$ 的长度取其短者为当前求得的最短路径长度。对每一对顶点的路径都做这样的试探，则可求得一个矩阵设为 $A(1)$ ，求 n 次即得每对顶点间的最短路径 $A(n)$ 。 $A(0)=A$ ：邻接矩阵。如下图：



$A(0) \rightarrow A(4)$

| | | | |
|----|---|----|---|
| 0 | 4 | 11 | ~ |
| 10 | 0 | 4 | 1 |
| 5 | ~ | 0 | ~ |
| ~ | ~ | 1 | 0 |

| | | | |
|----|---|----|---|
| 0 | 4 | 11 | ~ |
| 10 | 0 | 4 | 1 |
| 5 | 9 | 0 | ~ |
| ~ | ~ | 1 | 0 |

| | | | |
|----|---|---|----|
| 0 | 4 | 8 | 5 |
| 10 | 0 | 4 | 1 |
| 5 | 9 | 0 | 10 |
| ~ | ~ | 1 | 0 |

| | | | |
|---|----|---|----|
| 0 | 4 | 8 | 5 |
| 9 | 0 | 4 | 1 |
| 5 | 9 | 0 | 10 |
| 6 | 10 | 1 | 0 |

| | | | |
|---|----|---|----|
| 0 | 4 | 6 | 5 |
| 7 | 0 | 2 | 1 |
| 5 | 9 | 0 | 10 |
| 6 | 10 | 1 | 0 |

$P(0) \rightarrow P(4)$

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 |

| | | | |
|---|---|---|---|
| 0 | 0 | 2 | 2 |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 2 |
| 0 | 0 | 0 | 0 |

| | | | |
|---|---|---|---|
| 0 | 0 | 2 | 2 |
| 3 | 0 | 0 | 0 |
| 0 | 1 | 0 | 2 |
| 3 | 3 | 0 | 0 |

| | | | |
|---|---|---|---|
| 0 | 0 | 4 | 2 |
| 3 | 0 | 0 | 0 |
| 0 | 1 | 0 | 2 |
| 3 | 3 | 0 | 0 |

程序如下：

```

program floyed;
const n=4;
var
  cost,a:array[1..n,1..n]of integer;
  p:array[1..n,1..n] of 0..n;
  i,j,k:integer;
procedure init;
var i,j:integer;
begin
  for i:=1 to n do
    for j:=1 to n do
      begin
        read(cost[i,j]);
        a[i,j]:=cost[i,j];
        p[i,j]:=0;
      end;
    end;
end;
procedure path(i,j:integer);
var k:integer;
begin
  k:=p[i,j];
  if k<>0 then begin path(i,k);write('-',k);path(k,j);end
end;

```

```

begin
  init;
  for k:=1 to n do
    for i:=1 to n do
      for j:=1 to n do
        if a[i,k]+a[k,j]<a[i,j] then
          begin
            a[i,j]:=a[i,k]+a[k,j];
            p[i,j]:=k;
          end;
      end;
    end;
  for i:=1 to n do
    for j:=1 to n do
      if i<>j then
        begin
          writeln('a[' ,i ,',',j ,']= ',a[i,j]);
          write(i);
          path(i,j);
          writeln('->',j)
        end;
    end;
  end.

```

注意：弗洛耶德算法（Floyed 算法）思想可用与判断有向图中任意两点是否连通？算法如下：

```

for k:=1 to n do
  for i:=1 to n do
    for j:=1 to n do
      if (a[i,k]=1) and (a[k,j]=1) then a[i,j]=1 (a[i,j]=1 表示 i 可达 j, a[i,j]=0 表示 i 不可达 j)。

```

练习：

1. 调用 Dijkstra 算法 n 次，求各顶点间的最短距离
2. N ($N \leq 100$) 个接点带权图，可将一条边的权值减半，减哪条边可使点 1 到 N 的最短路径下降最多。

输入文件：第一行是一个整数 N ，图中的接点数。

第二行到第 $N+1$ 行是图的邻接矩阵。

输出文件：第一行输出减半的边的两个顶点。

第二行输出减半后的 1 到 N 的最短路程。

第三章 拓扑排序（AOV 网）

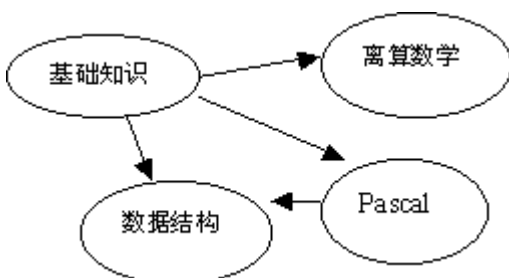
3.1 AOV 网(Activity On Vertex network)

3.2 拓扑排序

3.3 应用举例与练习

3.1 AOV 网

在现代化管理中，人们常用有向图来描述和分析一项工程的计划和实施过程，一个工程常被分为多个小的子工程，这些子工程被称为活动（Activity），在有向图中若以顶点表示活动，有向边表示活动之间的先后关系，这样的图简称为 AOV 网。如下图是计算机专业课程之间的先后关系：



基础知识课程应先于其它所有课程，pascal 语言课程应先于数据结构。

3.2 拓扑排序

在 AOV 网中为了更好地完成工程，必须满足活动之间先后关系，需要将各活动排一个先后次序即为**拓扑排序**。如上图的拓扑排序

基础知识：Pascal；数据结构；离散数学。或

基础知识：离散数学 Pascal；数据结构。

拓扑排序的方法和步骤：

(1) 在图中选一个没有前趋的顶点并输出之

(2) 删除该顶点及由它发出的各边，直到图中不存在没有前趋的顶点为止。

若图中存在回路，拓扑排序无法进行。

以下是将一 AOV 网进行拓扑排序的算法：

网采用邻接矩阵 A 表示，若 $a[i, j]=1$ ，表示活动 i 先于 j， $a[i, j]=0$ ，表示活动 i 与 j 不存在先后关系。

(1) 计算各顶点的入度

(2) 找入度为零的点输出之，删除该点，且与该点关联各点的入度减 1

(3) 若所有顶点都输出完毕。

程序如下：

```
program tppv;
const maxn=100;
var
  map:array[1..maxn,1..maxn] of byte;
  into:array[1..maxn] of byte;
  n,i,j,k:byte;
procedure init;
var
  i,j:integer;
begin
  read(n);
  for i:=1 to n do
    for j:=1 to n do
      begin
        read(map[i,j]);
        inc(into[j]);
      end;
end;
begin
  init;
  for i:=1 to n do
    begin
      j:=1;
      while (j<=n)and(into[j]<>0) do inc(j);
      write(j,' ');
      into[j]:=255;
      for k:=1 to n do
        if map[j,k]=1 then dec(into[k]);
      end;
    end;
end.
```

3.3 应用举例与练习

例：士兵排队问题：

有 N 个士兵 ($1 \leq N \leq 100$)，编号依次为 1, 2, ..., N. 队列训练时，指挥官要把士兵从高到矮排成一行，但指挥官只知道“1 比 2 高，7 比 5 高”这样的比较结果。

输入文件：第一行为数 N ($N \leq 100$)；表示士兵的个数。以下若干行每行两个数 A, B 表示 A 高于 B。

输出文件：给出一个合法的排队序列。

程序如下：


```

program tppv;
const maxn=100;
var
  map:array[1..maxn,1..maxn] of byte;
  into:array[1..maxn] of byte;
  n,i,j,k:byte;
  fp:text;
procedure init;
var
  i,j:integer;
begin
  assign(fp,'tp.txt');
  reset(fp);
  readln(fp,n);
  fillchar(map,sizeof(map),0);
  fillchar(into,sizeof(into),0);
  while not(seekeof(fp)) do
    begin
      readln(fp,i,j);
      map[i,j]=1 ;
      inc(into[j]);
    end;
  close(fp);
end;
begin
  init;
  for i:=1 to n do
    begin
      j:=1;
      while (j<=n)and(into[j]<>0) do inc(j);
      write(j,' ');
      into[j]:=255;
      for k:=1 to n do
        if map[j,k]=1 then dec(into[k]);
      end;
    end;
end.

```

练习：

Z 语言问题：Z 语言的基本字母也是 26 个，不妨用 a 到 z 表示，但先后顺序与英语不同。

现在按 Z 语言的顺序给出了 N 个单词，请依照这些单词给出一个可能的 Z 语言字母顺序。

输入文件：第一行一个数 N ($N \leq 100$) 表示单词的个数。

第二行到第 N+1 行，每行一个单词。

输出文件：仅一行，可能的字母顺序。

第四章 关键路径（AOE 网）

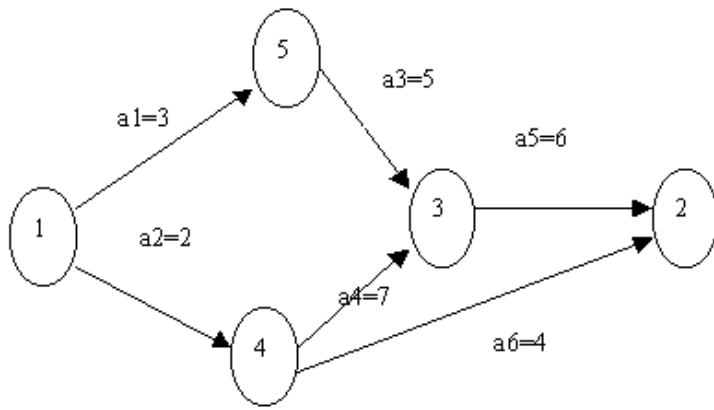
4.1 AOE 网(Activity On Edge network)

4.2 关键路径算法

4.3 练习

4.1 AOE 网

在现代化管理中，人们常用有向图来描述和分析一项工程的计划和实施过程，一个工程常被分为多个小的子工程，这些子工程被称为活动（Activity），在带权有向图中若以顶点表示事件，有向边表示活动，边上的权值表示该活动持续的时间，这样的图简称为 AOE 网，如下图。



AOE 网具有以下性质：

- (1) 只有在某顶点所代表的事件发生后，从该顶点出发的各有向边所代表的活动才能开始。
- (2) 只有在进入某点的各有向边所代表的活动都已结束，该顶点所代表的事件才能发生。

可以将上图假想一个工程有 6 项活动，网中 5 个顶点，分别表示 5 个事件，边上的权值分别表示各项活动所需要的时间，事件 v_1 表示工程开始，事件 v_3 表示活动 3 和 4 完成后，活动 5 可以开始，事件 v_4 表示活动 2 完成活动 4 和活动 6 开始， v_5 表示活动 1 完成活动 3 开始，事件 v_2 表示工程结束。

4. 2 关键路径及其算法

1. 关键路径：

在 AOE 网中所关心的是完成整个工程至少需要多少时间和哪些活动是影响整个工程进度的关键。

由于 AOE 网中的某些活动能够平行地进行，故完成整个工程所需的时间是从开始顶点到结束顶点的最长路径长度（路径上的权值和）最长路径叫**关键路径**。如上述工程的关键路径是 $1 \rightarrow 4 \rightarrow 3 \rightarrow 2$ ，关键路径长度为 $2+7+6=15$ ，关键活动是 a_2 ， a_4 ， a_5 。

2. 关键路径算法 1：

- 1) 将网拓扑排序
- 2) 以拓扑序列划分阶段
- 3) 用动态规划求解关键路径

程序如下：

```

program gjlu;
const maxn=10;
var
  map:array[1..maxn,1..maxn] of integer;
  a:array[1..maxn] of 0..maxn;
  b:array[1..maxn] of integer;
  c:array[1..maxn] of 0..maxn;
  n:integer;
procedure init;
var i,j:integer;
begin
  readln(n);
  for i:=1 to n do
    for j:=1 to n do
      read(map[i,j]);
  fillchar(a,sizeof(a),0);
  fillchar(b,sizeof(b),0);
  fillchar(c,sizeof(c),0);
end;
function tporder:boolean;
var i,j,k:integer;
into:array[1..maxn] of byte;
begin

```

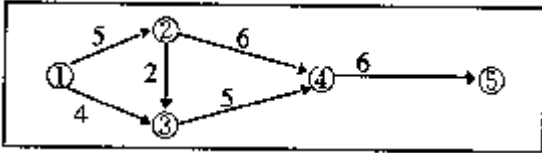
```

tporder:=false;
fillchar(into,sizeof(into),0);
for i:=1 to n do
  for j:=1 to n do
    if map[i,j]>0 then inc(into[j]);
for i:=1 to n do
begin
  j:=1;
  while (j<=n)and(into[j]<>0) do inc(j);
  if j>n then exit;
  a[i]:=j;
  into[j]:=$ff;
  for k:=1 to n do
    if map[j,k]>0 then dec(into[k]);
end;
tporder:=true;
end;
procedure calc;
var i,j:integer;
begin
  b[a[1]]:=0;c[a[1]]:=0;
  for i:=2 to n do
    for j:= 1 to i-1 do
      if b[a[j]]+map[a[j],a[i]]>b[a[i]] then
        begin b[a[i]]:=b[a[j]]+map[a[j],a[i]];c[a[i]]:=a[j]; end;
end;
procedure print;
var i,j,k:integer;
d:array[1..maxn] of 0..maxn;
begin
  writeln('long=',b[a[n]]);
  k:=c[a[n]];i:=0;
  while k<>0 do
    begin i:=i+1;d[i]:=k;k:=c[k] end;
for j:=i downto 1 do
  write(d[j],' ');
writeln(a[n])
end;
begin
  init;
  if tporder then
    begin
      calc;
      print;
    end;
end.

```

3. 关键路径算法 2:

若结点的排列已经过拓扑排序,即序号前面的结点会影响序号后面结点的活动,如下图。



可使用如下算法：

(1) 求活动最早可以开始的时间

```
eet[1]:=0; eet[k]:=max(eet[j]+r[j,k])
```

(2) 求活动最迟应该开始的时间

```
et[n]:=eet[n]; et[k]:=min(et[j]-r[k,j]);
```

(3) 关键路径通过点 J，具有如下的性质：eet[j]=et[j]

程序如下：

```

program gao7_6;
var i,j,n,max,min,w,x,y:integer;
    r:array[1..20,1..20] of integer;
    eet,et:array[1..20] of integer;
begin
  readln(n);
  for i:=1 to n do
    for j:=1 to n do
      r[i,j]:=-1;
  readln(x,y,w);
  while x<>0 do
    begin
      r[x,y]:=w;
      readln(x,y,w);
    end;
  eet[1]:=0;
  for i:=2 to n do
    begin
      max:=0;
      for j:=1 to n do
        if r[j,i]<>-1 then
          if r[j,i]+eet[j]>max then max:=r[j,i]+eet[j];
      eet[i]:=max;
    end;
  et[n]:=eet[n];
  for i:=n-1 downto 1 do
    begin
      min:=10000;
      for j:=1 to n do
        if r[i,j]<>-1 then
          if et[j]-r[i,j]<min then min:=et[j]-r[i,j];
      et[i]:=min;
    end;
  for i:=1 to n-1 do
    if et[i]=eet[i] then write(i,'->');
    write(n);readln;
end.

```

4.3 练习

1. 某工厂发现厂里的机器在生产产品时需要消耗大量的原材料, 现在厂里想找出消耗原材料最大的一条生产线路加以改进以降低成本。已知厂里的生产线路是一个有向的无环网络, 有 n 个机器分别代表网络中的 n 个结点, 弧 $\langle i, j \rangle$

$\langle i, j \rangle$ 表示原材料从机器 i 传到机器 j 的损耗数量。

输入文件：第一行两个整数 n, m ($n \leq 100, m \leq 100$) 分别表示网络中的结点的个数与弧数，第二行至 $m+1$ 行，每行三个整数 a, b, c ，表示弧 $\langle a, b \rangle$ 的损耗为 c 。

输出文件：只有一个整数，损耗最大线路的损耗量。

第五章 网络流

5.1 基本概念

5.2 最大流

5.3 最小费用最大流

5.1 基本概念

在实际生活中有许多流量问题，例如在交通运输网络中的人流、车流、货物流，供水网络中的水流，金融系统中的现金流，通讯系统中的信息流，等等。50 年代以福特(Ford)、富克逊(Fulkerson)为代表建立的“网络流理论”，是网络应用的重要组成部分。在最近的奥林匹克信息学竞赛中，利用网络流算法高效地解决问题已不是什么稀罕的事了。本节着重介绍最大流(包括最小费用)算法，并通过实际例子，讨论如何在问题的原型上建立一个网络流模型，然后用最大流算法高效地解决问题。

1. 问题描述 如图 5-1 所示是联结某产品地 v_1 和销售地 v_4 的交通网，每一弧 (v_i, v_j) 代表从 v_i 到 v_j 的运输线，产品经这条弧由 v_i 输送到 v_j ，弧旁的数表示这条运输线的最大通过能力。产品经过交通网从 v_1 到 v_4 。现在要求制定一个运输方案使从 v_1 到 v_4 的产品数量最多。

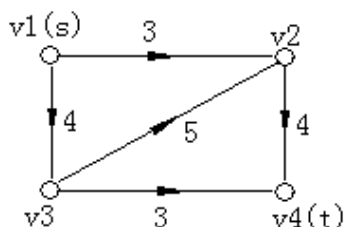


图 5-1

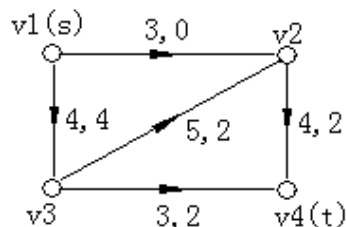


图 5-2

2. 网络与网络流

给一个有向图 $N=(V, E)$ ，在 V 中指定一点，称为源点(记为 v_s ，和另一点，称为汇点(记为 v_t)，其余的点叫中间点，对于 E 中每条弧 (v_i, v_j) 都对应一个正整数 $c(v_i, v_j) \geq 0$ (或简写成 c_{ij})，称为 f 的容量，则赋权有向图 $N=(V, E, c, v_s, v_t)$ 称为一个网络。如图 5-1 所给出的一个赋权有向图 N 就是一个网络，指定 v_1 是源点， v_4 为汇点，弧旁的数字为 c_{ij} 。

所谓网络上的流，是指定义在弧集合 E 上一个函数 $f=\{f(v_i, v_j)\}$ ，并称 $f(v_i, v_j)$ 为弧 (v_i, v_j) 上的流量(下面简记为 f_{ij})。如图 5-2 所示的网络 N ，弧上两个数，第一个数表示容量 c_{ij} ，第二个数表示流量 f_{ij} 。

3. 可行流与最大流

在运输网络的实际问题中，我们可以看出，对于流有两个显然的要求：一是每个弧上的流量不能超过该弧的最大通过能力(即弧的容量)；二是中间点的流量为 0，源点的净流出量和汇点的净流入量必相等且为这个方案的总输送量。因此有：

(1) 容量约束： $0 \leq f_{ij} \leq c_{ij}, (v_i, v_j) \in E$,

(2) 守恒条件

对于中间点：流入量=流出量；对于源点与汇点：源点的净流出量 $v_s(f)$ =汇点的净流入量 $(-v_t(f))$ 的流 f ，称为网络 N 上的可行流，并将源点 s 的净流量称为流 f 的流值 $v(f)$ 。

网络 N 中流值最大的流 f^* 称为 N 的最大流。

4. 可增广路径

所谓可增广路径，是指这条路径上的流可以修改，通过修改，使得整个网络的流值增大。

设 f 是一个可行流， P 是从源点 s 到汇点 t 的一条路，若 p 满足下列条件：

(1) 在 p 上的所有前向弧 $(v_i \rightarrow v_j)$ 都是非饱和弧，即 $0 \leq f_{ij} < c_{ij}$

(2) 在 p 上的所有后向弧 $(v_i \leftarrow v_j)$ 都是非零弧，即 $0 < f_{ij} \leq c_{ij}$

则称 p 为(关于可行流 f 的)一条可增广路径。

5. 最大流定理：当且仅当不存在关于 f^* 的增广路径，可行流 f^* 为最大流。

5.2 最大流算法

算法思想:最大流问题实际上是求一可行流 $\{f_{ij}\}$,使得 $v(f)$ 达到最大。若给了一个可行流 f ,只要判断 N 中有无关于 f 的增广路径,如果有增广路径,改进 f ,得到一个流量增大的新的可行流;如果没有增广路径,则得到最大流。

1. 寻求最大流的标号法(Ford, Fulkerson)

从一个可行流(一般取零流)开始,不断进行以下的标号过程与调整过程,直到找不到关于 f 的可增广路径为止。

(1) 标号过程

在这个过程中,网络中的点分为已标号点和未标号点,已标号点又分为已检查和未检查两种。每个标号点的标号信息表示两个部分:第一标号表明它的标号从哪一点得到的,以便从 v_t 开始反向追踪找出也增广路径;第二标号是为了表示该顶点是否已检查过。

标号开始时,给 v_s 标上 $(s, 0)$,这时 v_s 是标号但未检查的点,其余都是未标号的点,记为 $(0, 0)$ 。

取一个标号而未检查的点 v_i ,对于一切未标号的点 v_j :

A. 对于弧 (v_i, v_j) ,若 $f_{ij} < c_{ij}$,则给 v_j 标号 $(v_i, 0)$,这时, v_j 点成为标号而未检查的点。

B. 对于弧 (v_i, v_j) ,若 $f_{ji} > 0$,则给 v_j 标号 $(-v_i, 0)$,这时, v_j 点成为标号而未检查的点。

于是 v_i 成为标号且已检查的点,将它的第二个标号记为1。重复上述步骤,一旦 v_t 被标上号,表明得到一条从 v_i 到 v_t 的增广路径 p ,转入调整过程。

若所有标号都已检查过去,而标号过程进行不下去时,则算法结束,这时的可行流就是最大流。

(2) 调整过程

从 v_t 点开始,通过每个点的第一个标号,反向追踪,可找出增广路径 P 。例如设 v_t 的第一标号为 v_k (或 $-v_k$),则弧 (v_k, v_t) (或相应地 (v_t, v_k))是 p 上弧。接下来检查 v_k 的第一标号,若为 v_i (或 $-v_i$),则找到 (v_i, v_k) (或相应地 (v_k, v_i))。再检查 v_i 的第一标号,依此类推,直到 v_s 为止。这时整个增广路径就找到了。在上述找增广路径的同时计算 Q :

$$Q = \min\{\min(c_{ij} - f_{ij}), \min f_{ij}\}$$

对流 f 进行如下的修改:

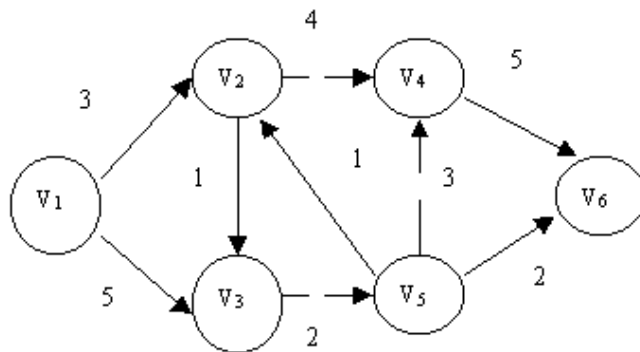
$$f'_{ij} = f_{ij} + Q \quad (v_i, v_j) \in P \text{ 的前向弧的集合}$$

$$f'_{ij} = f_{ij} - Q \quad (v_i, v_j) \in P \text{ 的后向弧的集合}$$

$$f'_{ij} = f_{ij} \quad (v_i, v_j) \text{ 不属于 } P \text{ 的集合}$$

接着,清除所有标号,对新的可行流 f' ,重新进入标号过程。

例1:下图表示一个公路网, V_1 是某原材料产地, V_6 表示港口码头,每段路的通过能力(容量)如图上的各边上的数据,找一运输方案,使运输到码头的原材料最多?



程序如下:

```

Program Max_Stream;
Const Maxn=20;
type
  nettype=record
    C,F:integer;
  end;
  nodetype=record
    L,P:integer;
  end;
var
  Lt:array[0..maxn] of nodetype;

```

```

G:Array[0..Maxn,0..Maxn] of Nettype;
N,S,T:integer;
F:Text;
Procedure Init;{初始化过程,读人有向图,并设置流为0}
Var Fn :String;
    I,J :Integer;
Begin
    Write( 'Graph File = ' ); Readln(Fn);
    Assign(F,Fn);
    Reset(F);
    Readln(F,N);
    Fillchar(G,Sizeof(G) ,0);
    Fillchar(Lt,Sizeof(Lt),0);
    For I:=1 To N Do
        For J:=1 To N Do Read(F,G[I,J].C);
    Close(F);
End;
Function Find: Integer; {寻找已经标号未检查的顶点}
Var I: Integer;
Begin
    I:=1;
    While (I<=N) And Not((Lt[I].L<>0)And(Lt[I].P=0)) Do Inc(I);
    If I>N Then Find:= 0 Else Find:= I;
End;
Function Ford(Var A: Integer):Boolean;
Var {用标号法找增广路径,并求修改量A}
    I,J,M,X:Integer;
Begin
    Ford:=True;
    Fillchar(Lt,Sizeof(Lt),0);
    Lt[S].L:=S;
    Repeat
        I:= Find;
        If i=0 Then Exit;
        For J:=1 To N Do
            If (Lt[J].L= 0)And((G[I,J].C<>0)or(G[J,I].C<>0)) Then
                Begin
if (G[I,J].F<G[I,J].C) Then Lt[J].L:= I;
If (G[J,I].F>0) Then Lt[J].L:=-I;
                    End;
                    Lt[I].P:=1;
                Until (Lt[T].L<>0);
                M:=T;A:=Maxint;
                Repeat
                    J:=M;M:=Abs(Lt[J].L);
                    If Lt[J].L<0 Then X:= G[J,M].F;
                    If Lt[J].L>0 Then X:= G[M,J].C- G[M,J].F;
                    If X<A Then A:= X;
                Until M= S;
                Ford:=False;
            End;
End;
Procedure Change(A: Integer);{调整过程}

```

```

Var M, J: Integer;
Begin
  M:= T;
  Repeat
    J:=M;M:=Abs(Lt[J].L);
    If Lt[J].L<0 Then G[J,M].F:=G[J,M].F-A;
    If Lt[J].L>0 Then G[M,J].F:=G[M,j].F+A;
  Until M=S;
End;

Procedure Print; {打印最大流及其方案}
VAR
  I ,J: Integer;
  Max: integer;
Begin
  Max:=0;
  For I:=1 To N DO
    Begin
      If G[I,T].F<>0 Then Max:= Max + G[I,T].F;
      For J:= 1 To N Do
        If G[I,J].F<>0 Then Writeln( I, '-> ' ,J, ' ' ,G[I,J].F);
    End;
    Writeln('The Max Stream=',Max);
  End;

Procedure Process;{求最大流的主过程}
Var Del:Integer;
    Success:Boolean;
Begin
  S:=1;T:=N;
  Repeat
    Success:=Ford(Del);
    If Success Then Print
    Else Change(Del);
  Until Success;
End;

Begin {Main Program}
  Init;
  Process;
End.

```

测试数据文件(zd1.txt)如下:

```

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

```

运行结果如下:

Graph File=zd1.txt

```

1->2  3
1->3  2
2->4  3

```


3→5 2

4→6 3

5→6 2

The Max Stream=5

5.3 最小费用最大流及算法

上面的网络，可看作为辅送一般货物的运输网络，此时，最大流问题仅表明运输网络运输货物的能力，但没有考虑运送货物的费用。在实际问题中，运送同样数量货物的运输方案可能有多个，因此从中找一个输出费用最小的方案是一个很重要的问题，这就是最小代价流所要讨论的内容。

1. 最小费用最大流问题的模型

给定网络 $N=(V, E, c, w, s, t)$ ，每一弧 (v_i, v_j) 属于 E 上，除了已给容量 c_{ij} 外，还给了一个单位流量的费用 $w(v_i, v_j) \geq 0$ (简记为 w_{ij})。所谓最小费用最大流问题就是要求一个最大流 f ，使得流的总输送费用取最小值。

$$W(f) = \sum w_{ij} f_{ij}$$

2. 用对偶法解最小费用最大流问题

定义：已知网络 $N=(V, E, c, w, s, t)$ ， f 是 N 上的一个可行流， p 为 v_s 到 v_t (关于流 f) 的可增广路径，称 $W(p) = \sum w_{ij} f_{ij} - \sum w_{ij} f_{ji}$ 为路径 p 的费用。

若 p^* 是从 v_s 到 v_t 所有可增广路径中费用最小的路径，则称 p^* 为最小费用可增广路径。

定理：若 f 是流量为 $v(f)$ 的最小费用流， p 是关于 f 的从 v_s 到 v_t 的一条最小费用可增广路径，则 f 经过 p 调整流量 Q 得到新的可行流 f' (记为 $f' = f + Q$)，一定是流量为 $v(f) + Q$ 的可行流中的最小费用流。

3. 对偶法的基本思路

①取 $f = \{0\}$

②寻找从 v_s 到 v_t 的一条最小费用可增广路径 p 。

若不存在 p ，则 f 为 N 中的最小费用最大流，算法结束。

若存在 p ，则用求最大流的方法将 f 调整成 f^* ，使 $v(f^*) = v(f) + Q$ ，并将 f^* 赋值给 f ，转②。

4. 迭代法求最小费用可增广路径

在前一节中，我们已经知道了最大流的求法。在最小费用最大流的求解中，每次要找一条最小费用的增广路径，这也是与最大流求法唯一不同之处。于是，对于求最小费用最大流问题余下的问题是怎样寻求关于 f 的最小费用增广路径 p 。为此，我们构造一个赋权有向图 $b(f)$ ，它的顶点是原网络 N 中的顶点，而把 N 中每一条弧 (v_i, v_j) 变成两个相反方向的弧 (v_i, v_j) 和 (v_j, v_i) 。定义 $w(f)$ 中的弧的权如下：

如果 $(f_{ij} < c_{ij})$ ，则 $b_{ij} = w_{ij}$ ；如果 $(f_{ij} = c_{ij})$ ，则 $b_{ij} = +\infty$

如果 $(f_{ij} > 0)$ ，则 $b_{ji} = -w_{ij}$ ；如果 $(f_{ij} = c_{ij})$ ，则 $b_{ji} = +\infty$

于是在网络 N 中找关于 f 的最小费用增广路径就等价于在赋权有向图 $b(f)$ 中，寻求从 v_s 到 v_t 的最短路径。求最短路有三种经典算法，它们分别是 Dijkstra 算法、Floyd 算法和迭代算法 (ford 算法)。由于在本问题中，赋权有向图 $b(f)$ 中存在负权，故我们只能用后两种方法求最短路，其中对于本问题最高效的算法是迭代算法。为了程序的实现方便，我们只要对原网络做适当的调整。将原网络中的每条弧 (v_i, v_j) 变成两条相反的弧：

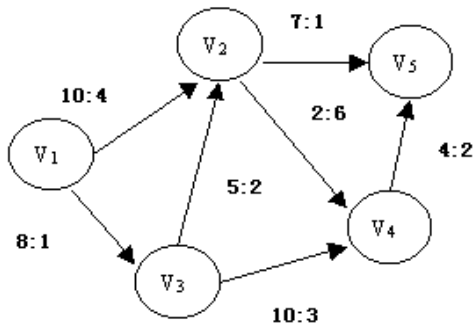
前向弧 (v_i, v_j) ，其容量 c_{ij} 和费用 w_{ij} 不变，流量为 f_{ij} ；

后向弧 (v_j, v_i) ，其容量 0 和费用 $-w_{ij}$ ，流量为 $-f_{ij}$ 。

事实上，对于原网络的数据结构中，这些单元已存在，在用标号法求最大流时是闲置的。这样我们就不必产生关于流 f 的有向图 $b(f)$ 。同时对判断 (v_i, v_j) 的流量可否改时，可以统一为判断是否 “ $f_{ij} < c_{ij}$ ”。因为对于后向弧 (v_j, v_i) ，若 $f_{ij} > 0$ ，则 $f_{ji} = -f_{ij} < 0 = c_{ji}$ 。

例 2：求输送量最大且费用最小的运输方案？

如下图是一公路网， v_1 是仓库所在地（物资的起点）， v_5 是某一工地（物资的终点）每条弧旁的两个数字分别表示某一时间内通过该段路的最多吨数和每吨物资通过该段路的费用。



程序如下:

```

{最小费用最大流参考程序}
program Maxflow_With_MinCost;
const
    name1='flow.in';
    name2='flow.out';
    maxN=100; {最多顶点数}
type
    Tbest=record {数组的结构}
        value,fa:integer;
    end; {到源点的最短距离, 父辈}
    Nettype=record
        {网络邻接矩阵类型}
        c,w,f:integer;
        {弧的容量, 单位通过量费用, 流量}
    end;
var
    Net:array[1..maxN,1..maxN] Of Nettype;
        {网络 N 的邻接矩阵}
    n,s,t:integer; {顶点数, 源点、汇点的编号}
    Minw:integer; {最小总费用}
    best:array[1..maxN] of Tbest; {求最短路时用的数组}
procedure Init; {数据读入}
var inf:text;
    a,b,c,d:integer;
begin
    fillchar(Net,sizeof(Net),0);
    Minw:=0;
    assign(inf,name1);
    reset(inf);
    readln(inf,n);
    s:=1;t:=n; {源点为 1, 汇点 n}
    repeat
        readln(inf,a,b,c,d);
        if a+b+c+d>0 then
            begin
                Net[a,b].c:=c; {给弧 (a,b) 赋容量 c}
                Net[b,a].c:=0; {给相反弧 (b,a) 赋容量 0}
                Net[a,b].w:=d; {给弧 (a,b) 赋费用 d}
                Net[b,a].w:=-d; {给相反弧 (b,a) 赋费用 -d}
            end;
    end;

```

```

until a+b+c+d=0;
close(inf);
end;
function Find_Path:boolean;
{求最小费用增广路函数,若 best[t].value<>MaxInt,
则找到增广路,函数返回 true,否则返回 false}
var Quit:Boolean;
    i,j:Integer;
begin
    for i:=1 to n do best[i].value:=Maxint;best[s].value:=0;
    {寻求最小费用增广路径前,给 best 数组赋初值}
    repeat
        Quit:=True;
        for i:=1 to n do
            if best[i].Value < Maxint then
                for j:=1 to n do
                    if (Net[i,j].f < Net[i,j].c) and
                        (best[i].value + Net[i,j].w <
                        best[j].value) then
                        begin
                            best[j].value:=best[i].value + Net[i,j].w;
                            best[j].fa := i;
                            Quit:= False;
                        end;
                until Quit;
            if best[t].value<Maxint then find_path:=true else find_path:=false;
end;
procedure Add_Path;
var i,j: integer;
begin
    i:= t;
    while i <> s do
        begin
            j := best[i].fa;
            inc(Net[j,i].f); {增广路是弧的数量修改,增量 1}
            Net[i,j].f:=-Net[j,i].f; {给对应相反弧的流量赋值-f}
            i:=j;
        end;
        inc(Minw,best[t].value); {修改最小总费用的值}
end;
procedure Out; {输出最小费用最大流的费用及方案}
var ouf: text;
    i,j: integer;
begin
    assign(ouf,name2);
    rewrite(ouf);
    writeln(ouf,'Min w = ',Minw);
    for i := 1 to n do
        for j:= 1 to n do
            if Net[i,j].f > 0 then
                writeln(ouf,i, ' -> ',j,': ',

```

```

        Net[i,j].w, '*', Net[i,j].f);
    close(ouf);
end;
Begin {主程序}
    Init;
    while Find_Path do Add_Path;
    {当找到最小费用增广路, 修改流}
    Out;
end.

```

flow.in 如下:

```

5
1 2 10 4
1 3 8 1
2 4 2 6
2 5 7 1
3 2 5 2
3 4 10 3
4 5 4 2
0 0 0 0

```

运行结果 flow.out 如下:

```

Min w = 55
1 -> 2: 4*3
1 -> 3: 1*8
2 -> 5: 1*7
3 -> 2: 2*4
3 -> 4: 3*4
4 -> 5: 2*4

```

练习:

1. 熟记上述两种算法。
2. 将例 2 程序中的寻找费用最小增广路径的算法改成 Floyd 算法。

第六章 图匹配

- 6.1 二分图的概念
- 6.2 最大匹配
- 6.3 匹配的最大（小）效应值

6.1 二分图的概念

设 $G=(V, E)$ 是一个无向图, 如果顶点 V 可分割两个互不相交的子集 (A, B) , 并且图中的每条边 (i, j) 所关联的两个顶点 i 和 j 分别属于这两个不同的顶点集 ($i \in A, j \in B$), 则称图 G 为一个二分图。如下图是一个二分图

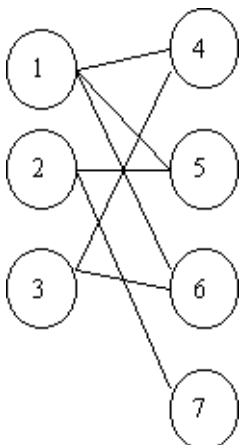


图 1

上图可以存储为邻接矩阵:

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |

也可以存储为 (1 1 1 0
0 1 0 1
1 0 1 0)

5. 2 最大匹配

1. 最大匹配的概念

设 $G=(V, E)$ 是一个图, 如果 M 是边 E 的子集, 且 M 中的任意两条边都不与同一个顶点相关联, 则称 M 是 G 的一个匹配。
 G 的边数最多的匹配称为 G 的最大匹配。

2. 最大匹配算法

最大流方法

1) 将图变成单向有向图 $A \rightarrow B$

2) 添加源点 s 和汇点 t 将图变为 $s \rightarrow A \rightarrow B \rightarrow t$, 如上图一可变为

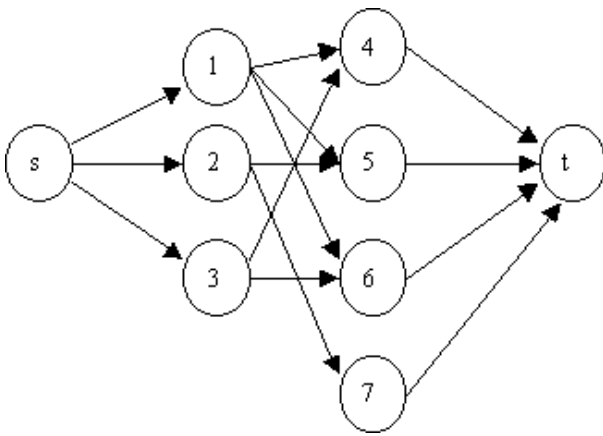


图 2

3) 设每条有向边的容量 c 为 1

4) 源点到汇点的最大流即为最大匹配

例一：混双配对问题：

乒乓球队中有 N 名男运动员和 M 名女运动员。由于某种原因，在混双比赛中，某些男运动员和某些女运动员不能配对比赛，这使得教练很苦恼，他希望你帮他找出一种最优得配对方案，组成今尽可能最多得混双配对。

输入文件：第一行两个正整数 N, M ($N+M \leq 100$), 表示男、女运动的个数。以下是一个 $N \times M$ 的矩阵。若 $A_{ij}=1$ 表示第 i 名男运动员可与第 j 名女运动员配对；若 $A_{ij}=0$ 则表示不能配对。

输出文件：只有一个整数，为最多的混双配对。

程序如下：

```
program tppl;
const inf='input.txt';
      maxn=100;
type
  arc=record
    c, f:integer;
  end;
  node=record
    last:integer;
    checked:boolean;
  end;
var
  map:array[1..maxn+2,1..maxn+2] of arc;
  imp:array[1..maxn+2] of node;
  n,m,s,t:integer;
```

```

    fp:text;
procedure init;
var i,j:integer;
begin
    assign(fp,inf);
    reset(fp);
    readln(fp,n,m);
    fillchar(map,sizeof(map),0);
    for i:=1 to n do
        for j:=1 to m do
            read(fp,map[i,n+j].c);
    s:=n+m+1;
    t:=n+m+2;
    for i:=1 to n do map[s,i].c:=1;
    for i:=1 to m do map[i+n,t].c:=1;
    close(fp);
end;
procedure main;
var flow,i:integer;
    function find:boolean;
        var i,j:integer;
        begin
            find:=false;
            fillchar(imp,sizeof(imp),0);
            imp[s].last:=s;
            repeat
                i:=1;
                while (i<=n+m+2) and ((imp[i].last=0) or imp[i].checked) do inc(i);
                if i>n+m+2 then exit;
                for j:=1 to n+m+2 do
                    if imp[j].last=0 then begin
                        if map[i,j].f<map[i,j].c then imp[j].last:=i else
                            if map[j,i].f>0 then imp[j].last:=-i; end;
                        imp[i].checked:=true;
                    until imp[t].last<>0;
                    find:=true;
                end;
            procedure updata;
            var i,j:integer;
            begin
                i:=t;j:=abs(imp[i].last);
                repeat
                    if imp[i].last>0 then inc(map[j,i].f) else dec(map[i,j].f);
                    i:=j;j:=abs(imp[i].last);
                until i=s;
            end;
            begin
                while find do updata;
                flow:=0;
                for i:=1 to n do inc(flow,map[s,i].f);
                writeln('maxpp=',flow);
            end;
begin

```

```
init;  
main;  
end.
```

5. 3 匹配的最大（小）效应值

先看一个问题

最优工作问题(work)

CCS 集团有 n 台机器。为了生产需要，新招募了 n 名技术人员。他们有丰富的专业技术，每个人都熟悉各种机器的操作。现在每个人只能操作一台机器，而每个人对不同机器的控制能力又有差别，技术员 i 对机器 j 的控制能力称为 $ability(i, j)$ 。要求给每人分配一台机器，使得所有人控制能力之和最大。

输入格式：

第一行仅有一个整数 $n(n \leq 100)$ ，第二行开始是一个 $n \times n$ 的矩阵，第 i 行第 j 列表示 $ability(i, j)$ ($ability(i, j) \leq 300$)。

输出格式：

仅一行：包含一个整数 S ，表示最大控制能力和。

样例：

INPUT. TXT

3
3 5 2
3 2 8
7 5 8

OUTPUT. TXT

20

对于本问题，可以建立如图 3 所示的图， x_1, x_2, x_3 分别代表 3 个技术人员， y_1, y_2, y_3 分别表示 3 台机器，有向边 (x_i, y_j) 表示 x_i 控制机器 y_j ，边上的权表示技术员对该机器的控制能力。显然图 3 是一个二分图。现在要求每个技术员唯一控制一台机器，且要控制能力之和最大。这样问题就转化为一个关于二分图求最佳匹配的问题

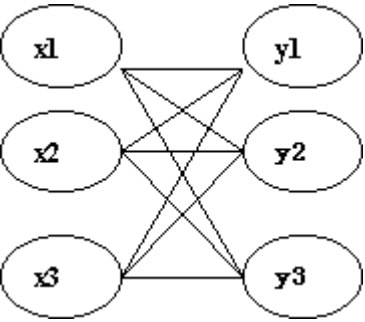
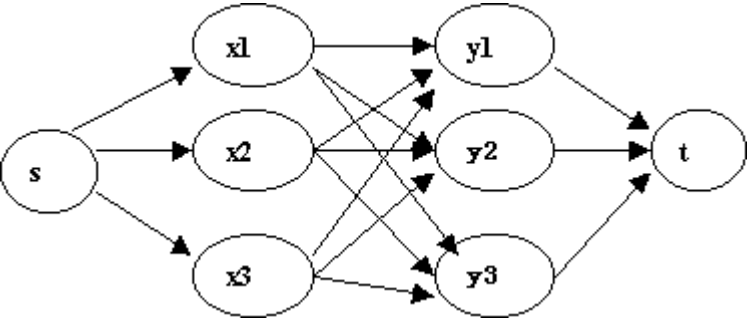


图 3

算法分析 下面用流的算法，求最佳匹配和最大控制能力和。

- (1) 在二分图基础上，构造一个网络 N (如图 4)。
 - ①增加一个源点 s 与一个汇点 t ;
 - ②从 s 向原图中顶点 x_i 引一条弧；从顶点 y_i 向 t 引一条弧；
 - ③令所有的弧的容量都为 1;
 - ④令弧 (s, x_i) 和 (y_i, t) 上的费用为 0。



(2) 对网络 N 求最大费用最大流(只要将所有权改为负，就是最小费用最大流)。这样构造网络的正确性道理很显然。首先因为网络中每个弧流量都为 1，所以就能保证每个技术员控制一台机

器，一台机器也唯一由一个技术员控制，最大流量必为 n ，其次要求最大费用最大流，即可保证所有技术员的控制机器的能力之和最大。

在编程时，可以边读入数据边产生网络 N ，接着运用一次求最大费用最大流算法，最后，在最大流的方案中通过找所有流为 1 的边 (v_i, v_j) ，就可得到各个技术员控制机器的情况及最大控制能力之和。

程序如下：

最大费用最大流方法：

```

program work;
const maxn=100;
type node1=record
    w,f,c:integer
end;
type node2=record
    value:integer;
    fa:integer;
end;
var a:array[1..maxn+2,1..maxn+2] of node1;
    best:array[1..maxn+2] of node2;
    n,maxwork,s,t:integer;
procedure init;
var i,j:integer;
begin
    readln(n);
    fillchar(a,sizeof(a),0);
    for i:=1 to n do
        for j:=n+1 to 2*n do
            begin
                read(a[i,j].w);
                a[i,j].c:=1;
                a[j,i].w:=-a[i,j].w;
            end;
        s:=2*n+1;
        t:=2*n+2;
        for i:=1 to n do
            begin
                a[s,i].c:=1;
                a[n+i,t].c:=1;
            end;
        maxwork:=0;
    end;
function find:boolean;
var i,j:integer;
quit:boolean;
begin
    fillchar(best,sizeof(best),0);
    best[s].value:=1;
    repeat
        quit:=true;
        for i:=1 to 2*n+2 do
            if best[i].value>0 then
                for j:=1 to 2*n+2 do
                    if (a[i,j].f<a[i,j].c) then
                        if best[i].value+a[i,j].w>best[j].value then

```



```

        begin
            best[j].value:=best[i].value+a[i,j].w;
            best[j].fa:=i;
            quit:=false;
        end;
until quit;
if best[t].value>1 then find:=true else find:=false;
end;
procedure add;
var i,j:integer;
begin
    i:=t;
    while i<>s do
        begin
            j:=best[i].fa;
            inc(a[j,i].f);
            a[i,j].f:=-a[j,i].f;
            i:=j
        end;
    inc(maxwork,best[t].value-1);
end;
begin
    init;
    while find do add;
    writeln(maxwork);
end.

```

最小费用最大流方法:

```

program work;
const maxn=100;
type node1=record
    w,f,c:integer
end;
type node2=record
    value:integer;
    fa:integer;
end;
var a:array[1..maxn+2,1..maxn+2] of node1;
    best:array[1..maxn+2] of node2;
    n,maxwork,s,t:integer;
procedure init;
var i,j,x:integer;
begin
    readln(n);
    fillchar(a,sizeof(a),0);
    for i:=1 to n do
        for j:=n+1 to 2*n do
            begin
                read(x);
                a[i,j].c:=1;
                a[i,j].w:=-x;
                a[j,i].w:=x;
            end;
        end;

```

```

s:=2*n+1;
t:=2*n+2;
for i:=1 to n do
begin
  a[s,i].c:=1;
  a[n+i,t].c:=1;
end;
maxwork:=0;
end;
function find:boolean;
var i,j:integer;
quit:boolean;
begin
  for i:=1 to 2*n+2 do best[i].value:=maxint;
  best[s].value:=0;
  repeat
  quit:=true;
  for i:=1 to 2*n+2 do
    if best[i].value<>maxint then
      for j:=1 to 2*n+2 do
        if (a[i,j].f<a[i,j].c) then
          if best[i].value+a[i,j].w<best[j].value then
            begin
              best[j].value:=best[i].value+a[i,j].w;
              best[j].fa:=i;
              quit:=false;
            end;
          end;
  until quit;
  if best[t].value<maxint then find:=true else find:=false;
end;
procedure add;
var i,j:integer;
begin
  i:=t;
  while i<>s do
  begin
    j:=best[i].fa;
    inc(a[j,i].f);
    a[i,j].f:=-a[j,i].f;
    i:=j
  end;
  inc(maxwork,best[t].value);
end;
begin
  init;
  while find do add;
  writeln(-maxwork);
end.

```

练习:

1. 丘比特之剑:

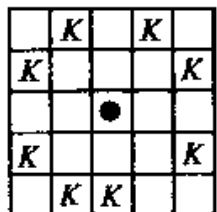
在东方男女相爱是千里姻缘一线牵, 两人无论身处何处都能相爱, 但在西方丘比特之剑射程有限, 只有射程内的男女两人才能相爱, 求最大缘分和。

输入文件: 第一行是一个正整数 k , 表示丘比特的射程。第二行为正整数 n ($n < 30$), 第三到 $n+2$ 行是男人的姓名和位置坐标, 第 $n+3$ 到第 $2n+2$ 行是女人的姓名和位置坐标: 格式是 `name x y`。剩下的部分是男人和女人间的缘分值 p 格式为 `name1 name2 p`, 以 `End` 作为文件的结束标志, 其中姓名字符串的长度不超过 20, $p \leq 255$, 每两人之间的缘分值只描述一次, 没描述的缘分值为 1。

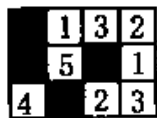
输出文件: 只有一个数, 最大的缘分和的值。

2. 最少皇后控制

在国际象棋中, 皇后能向八个方向攻击 (如图 4-5 所示, 图中黑点格子为皇后的位置, 标有 K 的格子为皇后可攻击到的格子)。现在给定一个 $M \times N$ (N, M 均不大于 10) 的棋盘, 棋盘上某些格子有障碍。每个皇后被放置在没有障碍的格子中, 且它就控制了这个格子, 除此, 它可以从它能攻击到的最多 8 个格子中选一个格子来控制, 如图 4-5(b) 所示, 标号为 1 的格子被一个皇后所控制。



(a)



(b)

图4-5

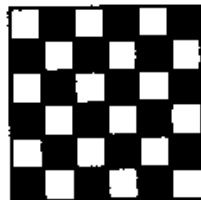


图4-6

请你编一程序, 计算出至少有多少个皇后才能完全控制整个棋盘。

输入格式:

输入文件的第一行有两个整数 M 和 N , 表示棋盘的行数与列数。接下来 M 行 N 列为一个字符矩阵, 用 ‘.’ 号表示空白的格子, ‘x’ 表示有障碍的格子。

输出格式:

输出文件的第一行仅有一个数 S , 表示需要皇后的数目, 接下来输出一个皇后的放置方案,

输入范例:

3 4

x. . .

x. x.

. x. .

输出范例

5

x132

x5x1

4x23

(相同类型的题目如下)

1. 已知一 $n \times m$ 方格棋盘上的某些方格禁止使用, 求能被放到棋盘上非攻击型车的最多个数?
2. 已知一 $n \times m$ 方格棋盘上的某些方格禁止使用, 能够放到棋盘上的 1×2 多米诺牌的最多个数是多少? (多米诺牌必须放在允许的方格内, 且两个多米诺牌不能重叠)

第八部分 搜索算法与优化

第一章 广度优先双向搜索

- 1.1 广度双向搜索的概念
- 1.2 广度双向搜索算法
- 1.3 例题与习题

1.1 广度双向搜索的概念

所谓双向搜索指的是搜索沿两个力向同时进行:

正向搜索: 从初始结点向目标结点方向搜索;

逆向搜索: 从目标结点向初始结点方向搜索;

当两个方向的搜索生成同一子结点时终止此搜索过程。

1.2 广度双向搜索算法

广度双向搜索通常有两中方法：

1. 两个方向交替扩展
2. 选择结点个数较少的那个方向先扩展。

方法 2 克服了两方向结点的生成速度不平衡的状态，明显提高了效率。

算法说明：

设置两个队列 $c:\text{array}[0..1, 1..maxn]$ of jid ，分别表示正向和逆向的扩展队列。

设置两个头指针 $head:\text{array}[0..1]$ of $integer$ 分别表示正向和逆向当前将扩展结点的头指针。

设置两个尾指针 $tail:\text{array}[0..1]$ of $integer$ 分别表示正向和逆向的尾指针。

$maxn$ 表示队列最大长度。

算法描述如下：

1. 主程序代码

```
repeat
  {选择节点数较少且队列未空、未满的方向先扩展}
  if (tail[0]<=tail[1]) and not((head[0]>=tail[0])or(tail[0]>=maxn)) then expand(0);
  if (tail[1]<=tail[0]) and not((head[1]>=tail[1])or(tail[1]>=maxn)) then expand(1);
  {如果一方搜索终止，继续另一方的搜索，直到两个方向都终止}
  if not((head[0]>=tail[0])or(tail[0]>=maxn)) then expand(0);
  if not((head[1]>=tail[1])or(tail[1]>=maxn)) then expand(1);
Until ((head[0]>=tail[0])or(tail[0]>=maxn)) And
((head[1]>=tail[1])or(tail[1]>=maxn))
```

2. expand(st:0..1)程序代码如下：

```
inc(head[st]);
取出队列当前待扩展的结点 c[st,head[st]]
for i:=1 to maxk do
  begin
    if tail[st]>=maxn then exit;
    inc(tail[st]);
    产生新结点;
    check(st);{检查新节点是否重复}
  end;
```

3. check(st:0..1)程序代码：

```
for i:=1 to tail[st]-1 do
  if c[st,tail[st]]^.*=c[st,i]^.* then begin dec(tail[st]);exit end;
bool(st);{如果节点不重复，再判断是否到达目标状态}
```

4. bool(st:0..1)程序代码：

```
for i:=1 to tail[1-st] do
  if c[st,tail[st]]^.*=c[1-st,i]^.* then print(st,tail[st],i);
  {如果双向搜索相遇（即得到同一节点），则输出结果}
```

5. print(st,tail,k)程序代码：

```
if st=0 then begin print0(tail);print1(k) end;
else begin print0(k);print1(tail) end;
```

6. print0(m)程序代码：

```
if m<>0 then begin print(c[0,m]^.*f);输出 c[0,m]^.* end;
{输出正方向上产生的结点}
```

7. print1(m)程序代码：

```
n:=c[1,m]^.*f
while m<>0
begin
  输出 c[1,n]^.*;
```

```

    n:=c[1,n]^f;
end
{输出反方向上产生的结点}

```

1.3 例题与习题

例 1: 8 数码难题 :

| | | |
|-------|-----|------------|
| 2 8 3 | | 1 2 3 |
| 1 6 4 | → 8 | 4 (用最少的步数) |
| 7 | 5 | 7 6 5 |

程序如下:

```

program num8;
const maxn=4000;
type jid=record
    str:string[9];
    f:0..maxn;
    dep:byte;
end;
bin=0..1;
var c:array[0..1,1..maxn] of ^jid;
    head,tail:array[0..1] of integer;
    start,goal:string[9];
procedure init;
var i,j:integer;
begin
    start:='283164705';
    goal:='123804765';
    for i:=0 to 1 do
        for j:=1 to maxn do
            new(c[i,j]);
        c[0,1]^str:=start; c[0,1]^f:=0; c[0,1]^dep:=0;
        c[1,1]^str:=goal; c[1,1]^f:=0; c[1,1]^dep:=0;
        for i:=0 to 1 do
            begin head[i]:=0;tail[i]:=1;end;
        end;
    procedure print(st:bin;tail,k:integer);
    procedure print0(m:integer);
    begin
        if m<>0 then
            begin print0(c[0,m]^f);writeln(c[0,m]^str) end;
        end;
    procedure print1(m:integer);
    var n:integer;
    begin
        n:=c[1,m]^f;
        while n<>0 do
            begin writeln(c[1,n]^str); n:=c[1,n]^f end;
        end;
    begin
        if st=0 then
            begin writeln('step=',c[0,tail]^dep+c[1,k]^dep);
                print0(tail); print1(k);end
            else begin writeln('step=',c[0,k]^dep+c[1,tail]^dep);
                print0(k); print1(tail); end ;
        end;
    end;

```

```

halt;
end;
procedure check(st:bin);
procedure bool(st:bin);
var i:integer;
begin
  for i:=1 to tail[1-st] do
    if c[st,tail[st]]^.str=c[1-st,i]^.str then print(st,tail[st],i);
end;
var i:integer;
begin
  for i:=1 to tail[st]-1 do
    if c[st,tail[st]]^.str=c[st,i]^.str then
      begin dec(tail[st]);exit end;
  bool(st);
end;
procedure expand(st:bin);
var i,p0,p1,d:integer;
str0,str1:string[9];
begin
  inc(head[st]);
  str0:=c[st,head[st]]^.str;
  d:=c[st,head[st]]^.dep;
  p0:=pos('0',str0);
  for i:=1 to 4 do
    begin
      if tail[st]>=maxn then exit;
      p1:=p0+2*i-5;
      if (p1 in [1..9]) and not ((p0=3) and (p1=4)) and not ((p0=4) and (p1=3))
        and not ((p0=6) and (p1=7)) and not ((p0=7) and (p1=6))
      then
        begin
          inc(tail[st]);
          str1:=str0;
          str1[p0]:=str1[p1];
          str1[p1]:='0';
          c[st,tail[st]]^.str:=str1;
          c[st,tail[st]]^.f:=head[st];
          c[st,tail[st]]^.dep:=d+1;
          check(st);
        end;
    end;
  end;
begin
  init;
  check(0);
  repeat
    if (tail[0]<=tail[1]) and not((head[0]>=tail[0]) or (tail[0]>=maxn))
      then expand(0);
    if (tail[1]<=tail[0]) and not((head[1]>=tail[1]) or (tail[1]>=maxn))
      then expand(1);
    if not((head[0]>=tail[0]) or (tail[0]>=maxn)) then expand(0);
    if not((head[1]>=tail[1]) or (tail[1]>=maxn)) then expand(1);
  until (head[0]>=tail[0]) or (tail[0]>=maxn) or (head[1]>=tail[1]) or (tail[1]>=maxn);
end;

```

```
Until ((head[0]>=tail[0])or(tail[0]>=maxn))And((head[1]>=tail[1])or(tail[1]>=maxn));
writeln('No answer');
end.
```

练习:

1. [问题描述]:

已知有两个字串 A\$, B\$ 及一组字串变换的规则 (至多 6 个规则):

A1\$ → B1\$

A2\$ → B2\$

规则的含义为: 在 A\$ 中的子串 A1\$ 可以变换为 B1\$、A2\$ 可以变换为 B2\$ …。

例如: A\$='abcd' B\$='xyz'

变换规则为:

'abc' → 'xu' 'ud' → 'y' 'y' → 'yz'

则此时, A\$ 可以经过一系列的变换变为 B\$, 其变换的过程为:

'abcd' → 'xud' → 'xy' → 'xyz'

共进行了三次变换, 使得 A\$ 变换为 B\$。

[输入]:

键盘输入文件名。文件格式如下:

A\$ B\$

A1\$ B1\$ \

A2\$ B2\$ |→ 变换规则

... ... /

所有字符串长度的上限为 20。

[输出]:

输出至屏幕。格式如下:

若在 10 步 (包含 10 步) 以内能将 A\$ 变换为 B\$, 则输出最少的变换步数; 否则输出 "NO ANSWER!"

[输入输出样例]

b. in:

abcd xyz

abc xu

ud y

y yz

屏幕显示: 3

2. 如下魔方版:

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 8 | 7 | 6 | 5 |

图 1

可经过下列三个变换

1) 上下行互换

图 1 可变换如下

| | | | |
|---|---|---|---|
| 8 | 7 | 6 | 5 |
| 1 | 2 | 3 | 4 |

2) 第一行同时循环右移一个格

图 1 可变换如下:

| | | | |
|---|---|---|---|
| 4 | 1 | 2 | 3 |
| 8 | 7 | 6 | 5 |

3) 中间四个格顺时针旋转一格

图 1 可变换为

| | | | |
|---|---|---|---|
| 1 | 7 | 2 | 4 |
| 8 | 6 | 3 | 5 |

输入初态和目态；输出变换步数和变换过程。

第二章 分支定界法

2.1 广度搜索定界

2.2 深度搜索定界

分支定界法的思想是：首先确定目标值的上下界，边搜索边减掉搜索树的某些支，提高搜索效率。

2.1 广度搜索定界

例 1：设有 A, B, C, D, E 5 人从事 j1, j2, j3, j4, j5 5 项工作每人只能从事一项，它们的效益表如下：

| | j1 | j2 | j3 | j4 | j5 |
|---|----|----|----|----|----|
| A | 13 | 11 | 10 | 4 | 7 |
| B | 13 | 10 | 10 | 8 | 5 |
| C | 5 | 9 | 7 | 7 | 4 |
| D | 15 | 12 | 10 | 11 | 5 |
| E | 10 | 11 | 8 | 8 | 4 |

求最佳安排，使效益最高？

本题可用回溯或其它方法解，但当人数稍多是，运行超时？用分支定界法，搜索中减掉不必要的分支？

程序如下：

```

program plan_job;
const maxn=20;
type arr=array[1..maxn] of integer;
  pnt=^node;
  node=record
    job,flag:arr;
    up,dep:integer;
    nxt:pnt;
  end;
var tp,p:pnt;
    n,min,row,depth:integer;
    goal:arr;
    a:array[1..maxn,1..maxn] of integer;
    f:text;
function cost(p:pnt):integer;
var i,j,max,y:integer;
begin
  y:=0;
  with p^ do
    begin
      for j:=1 to n do
        if j<dep+1 then y:=y+a[job[j],j]
        else
          begin
            max:=0;
            for i:=1 to n do
              if (max<a[i,j]) and (flag[i]=0) then max:=a[i,j];
            y:=y+max;
          end;
        end;
    end;
end;

```



```

        end;
    end;
    cost:=y;
end;
procedure init;
var i,j:integer;
begin
    assign(f,'inpb.txt');
    reset(f);
    readln(f,n);
    for i:=1 to n do
        for j:=1 to n do
            read(f,a[i,j]);
        end;
    end;
    new(tp);
    with tp^ do
        begin
            for i:=1 to n do
                begin flag[i]:=0; job[i]:=0 end;
            end;
            dep:=0;
            up:=cost(tp);
            nxt:=nil;
        end;
    end;
    min:=0;
end;
procedure process(p:pnt);
var i,j:integer;
begin
    with p^ do
        begin
            dep:=depth;
            flag[dep]:=dep;
            job[dep]:=row;
            up:=cost(p);
        end;
    end;
end;
procedure sort(p:pnt);
var x,y:pnt;
begin
    y:=tp;
    repeat
        x:=y;
        y:=x^.nxt;
    until (y=nil) or (p^.up>=y^.up);
    x^.nxt:=p;
    p^.nxt:=y;
end;
procedure goals(p:pnt);
begin
    if p^.up>min then
        begin
            goal:=p^.job;min:=p^.up;
        end;
end;
end;
```

```

procedure print;
var i,k:word;
begin
  for i:=1 to n do write('j',i,':',chr(goal[i]+64),' ');
  writeln;
  writeln('maxcost=',min);
  readln;
end;
begin
  init;
  repeat
    if tp^.up>min then
      begin
        depth:=tp^.dep+1;
        for row:=1 to 5 do
          if tp^.flag[row]=0 then
            begin
              new(p);
              p^:=tp^;
              process(p);
              if p^.up<min then dispose(p) else sort(p);
              if depth=n then goals(p);
            end;
          end;
        tp:=tp^.nxt;
      until tp=nil;
      print;
    end.

```

2. 2 深度搜索定界

例 2: 设定有 n 台处理机 p_1, p_2, \dots, p_n , 和 m 个作业 j_1, j_2, \dots, j_m , 处理机可并行工作, 作业未完成不能中断, 作业 j_i 在处理机上的处理时间为 t_i , 求解最佳方案, 使得完成 m 项工作的时间最短?

说明: 本题有两重搜索法, 搜索处理机和搜索作业, 当 m, n 较大时, 搜索处理机不可能? 搜索作业容易确定上下界, 容易剪支。

程序如下:

```

program jobs;
const maxn=100;maxm=100;
var
  t:array[1..maxm] of integer;
  l,bestl:array[1..maxn] of integer;
  a,besta:array[1..maxn,1..maxm] of integer;
  time:array[0..maxn] of integer;
  done:array[1..maxm] of boolean;
  least,i,j,k,n,m,rest,min:integer;
procedure print ;
var i,j:integer;
begin
  for i:=1 to n do
    begin
      for j:=1 to bestl[i] do
        write(besta[i,j],' ');
      writeln;
    end ;

```

```

writeln('Time=',time[0]+1);
end;
procedure init;
var
f:text;
fname:string;
i,j,k:integer;
begin
write('filename=');readln(fname);
assign(f,fname);
reset(f);
readln(f,n,m);
for i:=1 to m do
begin
read(f,t[i]);inc(rest,t[i]);
end;
close(f);
least:=trunc(rest div n)+1;{确定下界}
for i:=1 to m-1 do
for j:=i+1 to m do
if t[i]<t[j] then
begin
k:=t[i];t[i]:=t[j];t[j]:=k
end;
end;
procedure try(p,q:integer);{从 p..m 中选取作业放在处理机 q 上}
var i:integer;
begin
for i:=p to m do
if done[i] and (time[q]+t[i]<=time[q-1]) then
begin
done[i]:=false;
inc(l[q]);
a[q,l[q]]:=t[i];
inc(time[q],t[i]);
dec(rest,t[i]);
try(i+1,q);
done[i]:=true;
dec(l[q]);
dec(time[q],t[i]);
inc(rest,t[i]);
end;
if rest<=(n-q)*time[q] then
if rest=0 then
begin
bestl:=l;besta:=a;
time[0]:=time[1]-1;
if time[1]=least then
begin
print;halt;
end;
end else

```

```

    if q<n then try(1,q+1);
end;
begin
    init;
    fillchar(done,sizeof(done),true);
    fillchar(time,sizeof(time),0);
    fillchar(a,sizeof(a),0);
    fillchar(besta,sizeof(besta),0);
    fillchar(l,sizeof(l),0);
    fillchar(bestl,sizeof(bestl),0);
    for i:=1 to m do{确定上界}
        begin
            k:=1;
            for j:=2 to n do
                if time[j]<time[k] then k:=j;
            time[k]:=time[k]+t[i];
            bestl[k]:=bestl[k]+1;
            besta[k,bestl[k]]:=t[i];
        end;
    min:=time[1];
    for i:=2 to n do
        if time[i]>min then min:=time[i];
    time[0]:=min-1;
    if min=least then begin print; halt end;
    fillchar(time,sizeof(time),0);
    time[0]:=min-1;
    try(1,1);
    print;
end.

```

若输入文件是：

```

3 6
11 7 5 5 4 7
输出结果如下：
7 7
5 5 4
11
time=14

```

练习：上述例 2 中若只求最短时间，不求方案，怎样修改程序？

第三章 A*算法

3.1 A*算法思想

3.2 A*算法例题与习题

3.1A*算法思想（启发函数法）

A*算法属于一种启发式搜索。它扩展结点的次序类似于广度优先搜索，但不同的是每生成一个子结点需要计算估价函数 F，以估算起始结点到该结点的代价及它到达目标结点的代价的和；每当扩展结点时，总是在所有待扩展结点中选择具有最小 F 值的结点作为扩展对象，以便使搜索尽量沿最有希望的方向进行。

因此，A*算法只要求产生问题的全部状态空间的部分结点，就可以求解问题了，搜索效率较高。

确定估价函数方法通常是：搜索到该结点的深度+ 距离目标最近的程度。

3. 2 A*算法例题与习题

例 1：8 数码难题：2 8 3 1 2 3
 1 6 4 → 8 4（用最少的步数）

用 A*算法程序如下:

```

program num8;
type a33=array[1..3,1..3] of 0..8;
     a4=array[1..4] of -1..1;
node=record
  ch:a33;
  si,sj:1..3;
  f:byte;
  pnt,dep,next:byte;
end;
const goal:a33=((1,2,3),(8,0,4),(7,6,5));
      start:a33=((2,8,3),(1,6,4),(7,0,5));
      di:a4=(0,-1,0,1);
      dj:a4=(-1,0,1,0);
var data:array[0..100] of node;
    temp:node;
    r,k,ni,nj,head,tail,depth:integer;
function check(k:integer):boolean;
begin
  ni:=temp.si+di[k];nj:=temp.sj+dj[k];
  if (ni in [1..3]) and (nj in [1..3]) then check:=true else check:=false;
end;
function dupe:boolean;
var i,j,k:integer;
    buf:boolean;
begin
  buf:=false;i:=0;
  repeat
    inc(i);buf:=true;
    for j:=1 to 3 do
      for k:=1 to 3 do
        if data[i].ch[j,k]<>data[tail].ch[j,k] then buf:=false;
      until buf or (i>=tail-1);
    dupe:=buf;
  end;
function goals:boolean;
var i,j:byte;
begin
  goals:=true;
  for i:=1 to 3 do
    for j:=1 to 3 do
      if data[tail].ch[i,j]<>goal[i,j] then goals:=false;
    end;
  end;
procedure print;
var buf:array[1..100] of integer;
    i,j,k,n:integer;
begin
  n:=1;
  i:=tail;buf[1]:=i;
  repeat
    inc(n);buf[n]:=data[i].pnt;
  
```

```

    i:=data[i].pnt;
until i=0;
writeln('steps=',depth+1);
for i:=1 to 3 do
begin
    for k:=n-1 downto 1 do
    begin
        for j:=1 to 3 do write(data[buf[k]].ch[i,j]);
        if (i=2) and (k<>1) then write('->') else write(' ');
    end;
    writeln;
end;
readln;halt
end;
function calc_f(a:a33):byte;
var i,j,temp:byte;
begin
    temp:=0;
    for i:=1 to 3 do
        for j:=1 to 3 do
            if (a[i,j]<>goal[i,j]) and (a[i,j]>0) then inc(temp);
calc_f:=temp+depth+1;
end;
procedure sort(num:integer);
var x,y:word;
begin
    y:=head;
    repeat
        x:=y;y:=data[x].next;
    until (y=0) or (data[y].f>data[num].f);
    data[x].next:=num;data[num].next:=y;
end;
begin
    head:=0;tail:=1; data[0].next:=1;
    with data[1] do
    begin
        ch:=start;si:=3;sj:=2;
        pnt:=0;dep:=0;next:=0;
        f:=calc_f(ch);
    end;
    repeat
        head:=data[head].next;temp:=data[head];
        depth:=temp.dep;
        for r:=1 to 4 do
            if check(r) then
                begin
                    inc(tail);data[tail]:=temp;
                    with data[tail] do
                    begin
                        ch[si,sj]:=ch[ni,nj];
                        ch[ni,nj]:=0;si:=ni;sj:=nj;
                        pnt:=head;
                        dep:=depth+1;

```

```
    f:=calc_f(ch);
  end;
  if dupe then dec(tail) else if goals then print else sort(tail);
  end;
until data[head].next=0;
writeln('no solution');readln;
end.
```

练习：

1. 移动棋子游戏：在下列所示的 10 个格子里，前面两格是空格，后面相间的放着 4 个 A 和 4 个 B

| | | | | | | | | | |
|--|--|---|---|---|---|---|---|---|---|
| | | A | B | A | B | A | B | A | B |
|--|--|---|---|---|---|---|---|---|---|

若每次可移动任意两个相邻的棋子进入空格，移动时两棋子不得更动其原来次序目标是将 4 个 A 连在一起，空格位置不限。试编程，求出一种方案并输出每移动一次后得棋子状态。