数据结构

北京交通大学计算机学院

2014年2月

《数据结构》课程的基本要求

要求掌握各种常用的数据结构，包括线性表、栈、队列、字符串、数组、广义表、树、二叉树以及图等基本类型的数据结构及其应用，掌握排序、查找的各种实现算法，并能够从时间上对各种算法进行定性或定量的分析和比较。

《数据结构》课程的目的

本课程介绍一些最常用的数据结构，阐明数据结构内在的逻辑联系，讨论它们在计算机中的存储表示，并结合各种典型应用说明它们在进行各种运算及操作时的动态性质和实际执行的算法。提高根据问题的性质选择合理的数据结构并控制求解算法的空间和时间复杂性的能力，以达到进一步提高软件设计和编写程序水平的目的。

教材与参考资料

1. 严蔚敏等，《数据结构》（C语言版），清华大学出版社 （教材）
2. 严蔚敏等，《数据结构题集》（C语言版），清华大学出版社 （教材配套习题）
3. 殷人昆等，《数据结构（面向对象方法与C++描述）》，清华大学出版社
4. 张铭等，《数据结构与算法》，高等教育出版社，2008年6月第一版
5. 邓俊辉，数据结构（C++语言版），清华大学出版社，2013年9月

第一章

绪 论

※ 教学内容：

数据结构的基本概念；算法设计；时间和空间复杂度

※ 教学重点：

相关的基本概念；算法五大要素；计算语句频度和估算算法时间复杂度的方法

※ 教学难点：

估算算法时间复杂度的方法

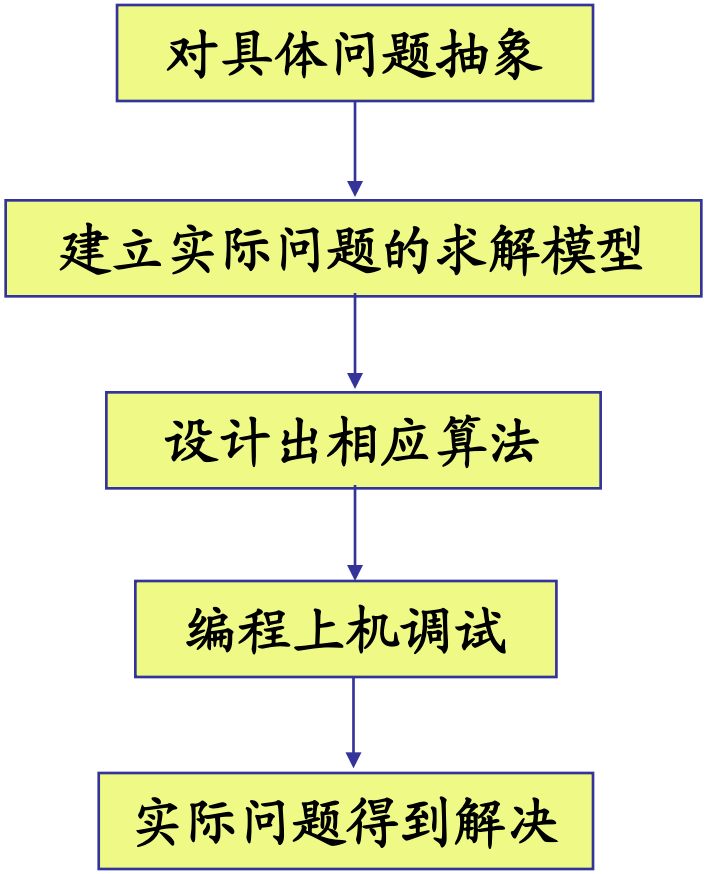
1.1 什么是数据结构

1.2 基本概念和术语

1.3 算法和算法分析

1.1、什么是数据结构

用计算机解决实际问题的流程:



数值计算问题

数学方程

运算对象一般为整型、实型等一些简单的数据类型；

程序设计者的主要精力集中在程序设计的技巧上，而不是数据的组织和存储上。

非数值计算问题

无法用数学方程来进行描述，必须建立相应的数据结构来进行描述，分析问题中所用到的数据是如何组织的，研究数据之间的关系如何，进而设计出合适的算法。

数据结构的概念一般包括：数据之间的逻辑关系、数据在计算机中的存储方式和数据的运算三个方面。

为了叙述上的方便，把上述数据结构的三个方面分别称为：

- **数据的逻辑结构**
- **数据的存储结构**
- **数据的运算**

例1　职工信息管理

一个单位要对所有职工的基本情况进行管理，包括查询职工的姓名、性别及月收入等情况，对整个单位的职工信息进行统计和汇总等。

如何利用计算机来辅助管理这些信息？

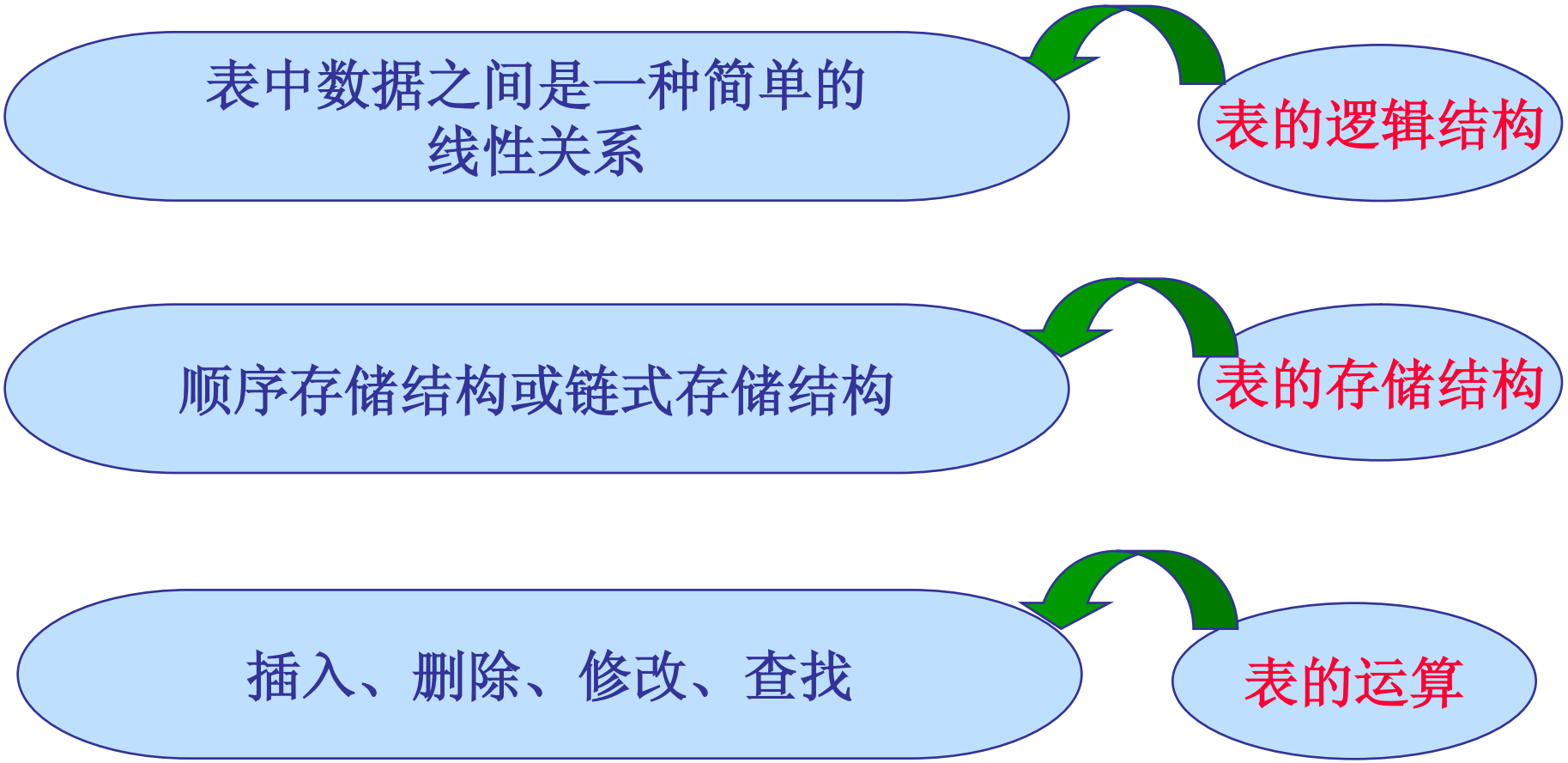
如何有效地组织每个职工的信息？ → 逻辑结构

如何在计算机中储存这些职工信息？ → 存储结构

如何对这些职工信息进行各种处理？ → 数据运算

职工信息表

职工号	姓名	性别	年龄	月收入（元）
1	张敬	男	55	3200
2	李红	女	51	2800
3	孙涛	女	41	2300
4	王宁宁	男	33	1950
……	……	……	……	……

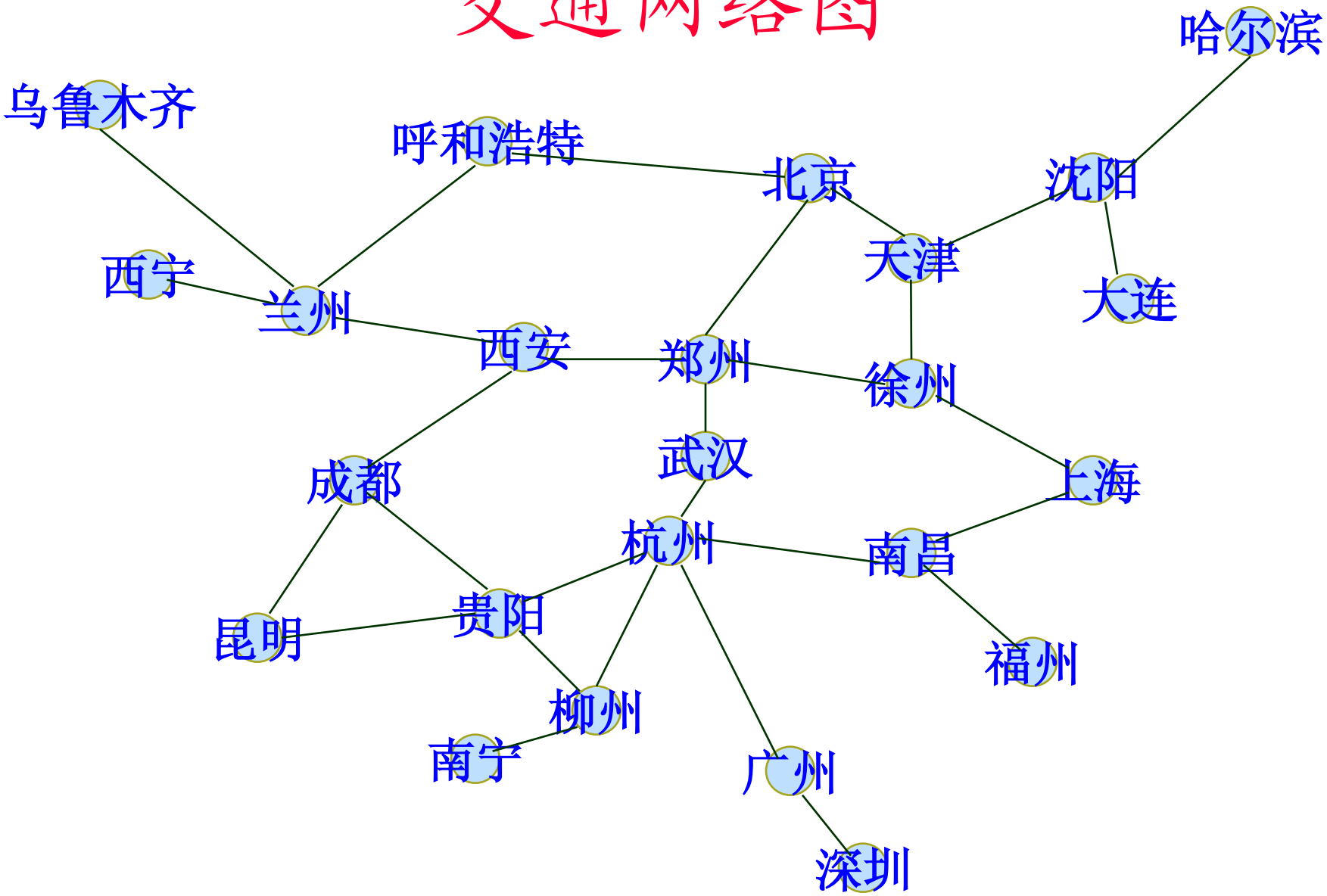

表中数据之间是一种简单的
线性关系
表的逻辑结构
顺序存储结构或链式存储结构
表的存储结构
插入、删除、修改、查找
表的运算

例2　交通信息咨询

建立一个全国范围内的交通信息咨询系统，回答游客提出的各种问题，如合理安排旅游线路等。

此时可采用一种称之为图的结构来表示实际的交通网络，图中顶点表示城市，边表示城市间交通联系，这个交通网络图就表示一个数据结构。

交通网络图


哈尔滨
乌鲁木齐
呼和浩特
北京
沈阳
天津
西宁
兰州
大连
西安
郑州
徐州
武汉
成都
上海
杭州
南昌
贵阳
昆明
福州
柳州
南宁
广州
深圳

结点之间的关系可以是任意的，
图中任意两个结点之间都可以相关

图的逻辑结构

在计算机中既要存储图中每个结点，
又要存储结点之间的关系

图的存储结构

如求最短路径问题，
求关键路径问题等

图的运算

从上面两例可见，要描述非数值计算问题的求解模型，单靠数学方程是无法完成的，需要使用一些诸如表、图、还有树之类的数据结构。

因此，简单说，数据结构是一门研究非数值计算问题的程序设计中计算机的操作对象以及它们之间的关系和运算操作的学科。

《数据结构》课程与高级程序设计语言课程有什么不同？

《数据结构》课程实际上是一门研究非数值计算问题的程序设计中计算机的操作对象以及它们之间的关系和运算操作等的一门学科，所讨论的是在数据组织的基础上的复杂程序的设计问题；而高级程序设计语言所讨论的是简单程序设计问题。综上所述，

按某种逻辑关系组织起来的一批数据，按一定的存储表示方式把它存储在计算机的存储器中，并在这些数据上定义了一个运算的集合，就叫做一个数据结构。

《数据结构》课程的地位

《数据结构》在计算机科学中是一门综合性的专业基础课，其研究不仅涉及到计算机硬件（特别是编码理论、存储装置和存取方法等）的研究范围，而且和计算机软件的研究有着密切的关系，无论是编译程序还是操作系统，都涉及到数据元素在存储器中的分配问题。在研究信息检索时也必须考虑如何组织数据，以便查找和存取数据元素更为方便。因此，可以认为数据结构是介于数学、计算机硬件和计算机软件三者之间的一门核心课程。不仅是一般程序设计（特别是非数值计算的程序设计）的基础，而且是设计和实现编译程序、操作系统、数据库系统及其他系统程序和大型应用程序的重要基础。

著名的瑞士计算机科学家N.Wirth教授提出公式:

Algorithm + Data Structures = Programs

算法:处理问题的策略（对数据运算的描述）

数据结构:问题的数学模型（指数据的逻辑结构和存储结构）

程序设计

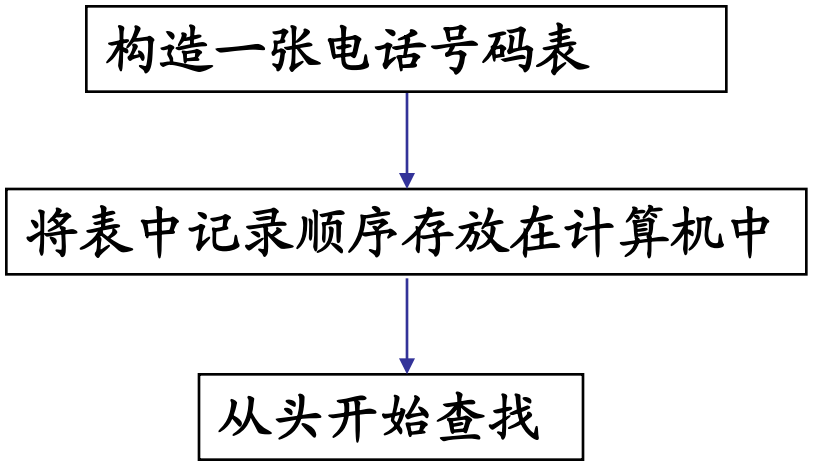
由此可见，程序设计的实质是对实际问题选择一种好的数据结构，再设计一个好的算法，而好的**算法**在很大程度上**取决于**描述实际问题的**数据结构**。

再看两个例子：

例3：电话号码查询

编写一个程序，查询某单位的职工电话号码。要求对任意给出的一个姓名，若该人有电话号码，则要快速找到其电话号码；否则指出该人没有电话号码。

最简单的方法：

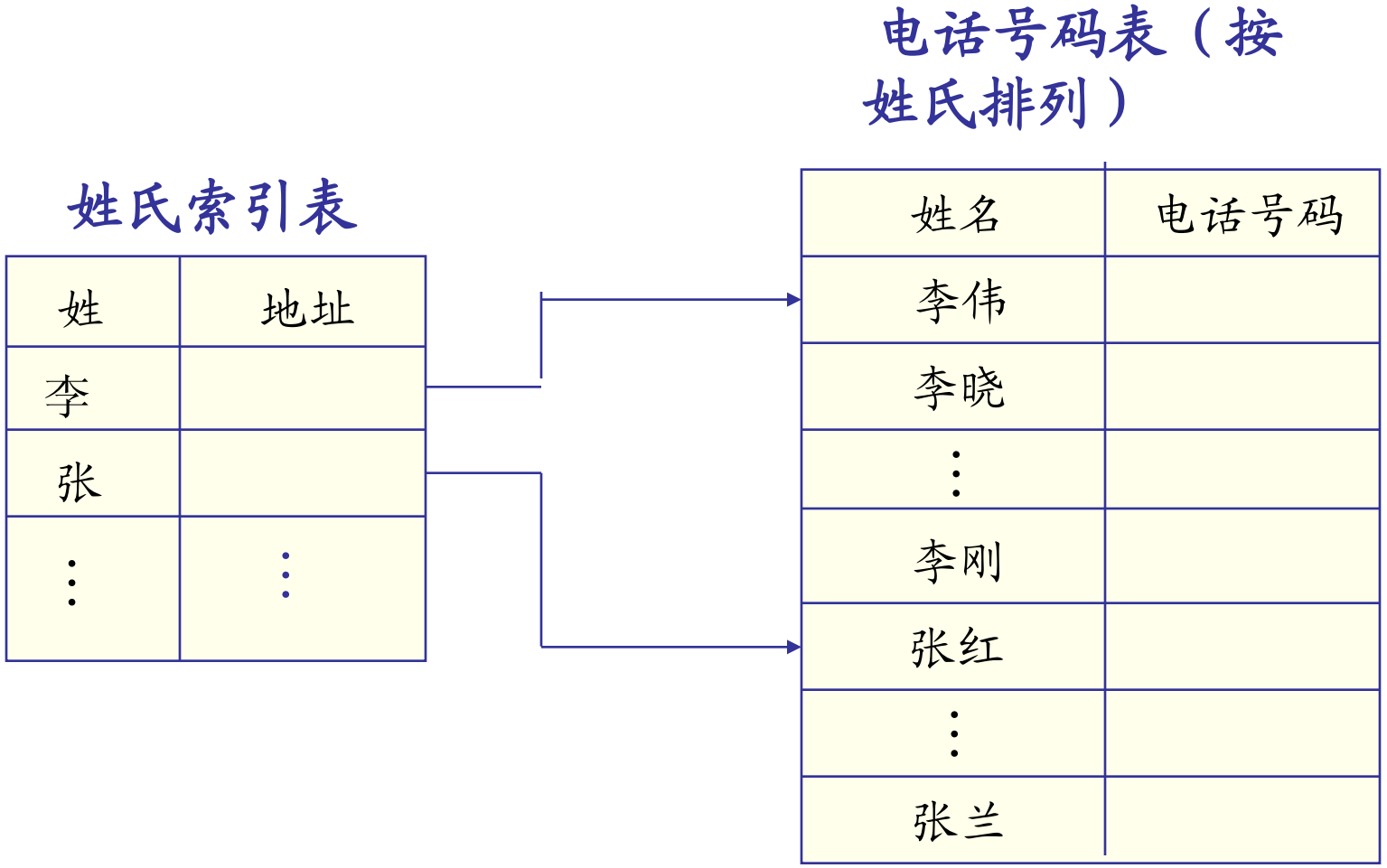


电话号码表

姓名	电话号码
张红	135***
李伟	136***
王刚	137***
⋮	
⋮	

改进的方法：

将电话号码表按姓氏排列，然后建立姓氏索引表。



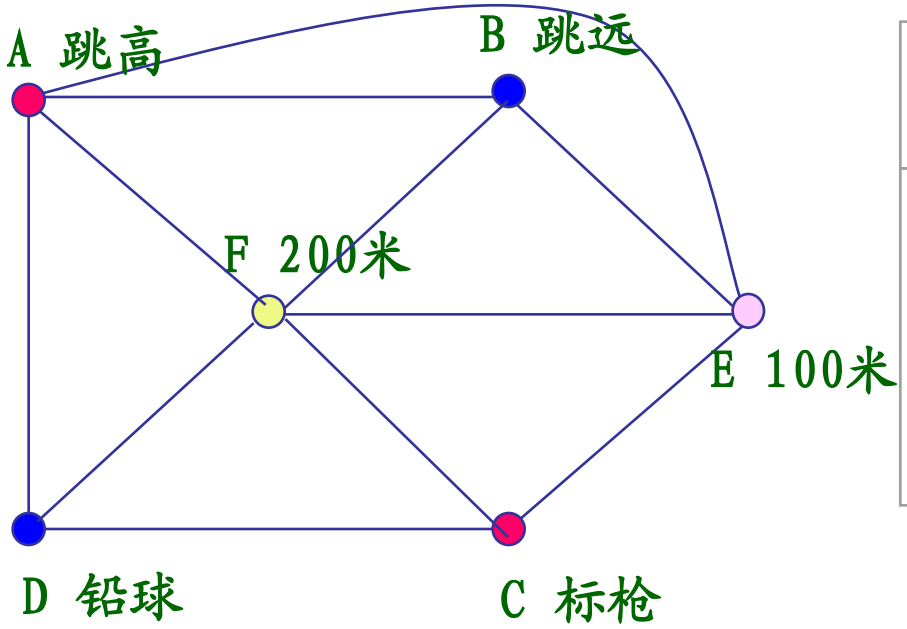
例4：运动会田径比赛的时间安排

某高校运动会共设六个田径比赛项目，即跳高、跳远、铅球、标枪、100米和200米短跑，规定每个选手至多参加三个项目的比赛。现有五名选手报名参赛，选手所选择的项目如下表所示：

姓名	项目1	项目2	项目3
王一	跳高	跳远	100米
王二	标枪	铅球	
王三	标枪	100米	200米
王四	铅球	200米	跳高
王五	跳远	200米	

要求设计一个竞赛日程安排表，使得在尽可能短的时间内安排完比赛。

首先应选择一个合适的数据结构来表示它，为此，设计一个图，图中顶点代表竞赛项目，在所有的两个不能同时进行比赛的项目之间连上一条边。显然，同一个选手选择的几个项目是不能在同一时间内比赛的，因此该选手所选择的项目中应该两两有边相连，由此得到如下数据结构模型（无向图），图中，A，B，…，F分别对应于六个竞赛项目。



姓名	项目1	项目2	项目3
王一	跳高	跳远	100米
王二	标枪	铅球	
王三	标枪	100米	200米
王四	铅球	200米	跳高
王五	跳远	200米	

竞赛项目的时间安排问题可以抽象为对无向图进行“着色”操作，即：用尽可能少的颜色给图中每个顶点着色，使得任意两个有边连接的相邻顶点着上不同的颜色，每一种颜色表示一个比赛时间，着上同一颜色的顶点是可以安排在同一时间内竞赛的项目。

由此看出，解决问题的一个关键步骤是：选取合适的数据结构表示该问题，然后才能写出有效的算法。

1.2 基本概念和术语

一、数据与数据结构

二、数据类型

三、抽象数据类型

一、数据与数据结构

- **数据**

  是指所有能被输入到计算机中并被计算机程序处理的符号的总称。例如：整数、实数、字符串、图象、声音等。（凡能被计算机存储、加工的对象通称为数据。）

  是计算机操作的对象的总称。

  是计算机处理的信息的某种特定的符号表示形式。

- **数据元素**

  是数据（集合）中的一个“个体”。

  是数据的基本单位，是数据结构中讨论的基本单位。

- **数据项**

一个数据元素由若干个**数据项**组成，**数据项**是数据不可分割的最小单位。

例如：描述一个运动员的数据元素可以是

姓名	俱乐部名称	出生日期	参加日期	职务	业绩

数据、数据元素和数据项反映了数据组织的三个层次

- **数据对象**

是性质相同的数据元素的集合，是数据的一个子集。

例如：整数数据对象是集合

$$N=\{0,\pm1,\pm2,\ldots\}$$

字母字符数据对象是集合

$$C=\{\text{‘A’},\text{’B’},\ldots,\text{’Z’}\}$$

▪ 结构

在任何问题中，数据元素都不是孤立存在的，而是在他们之间存在着某种关系，这种数据元素相互之间的关系称为结构。

假设用三个 4 位的十进制数表示一个含 12 位数的十进制数。

例如：3214,6587,9345 — **a1**(3214),**a2**(6587),**a3**(9345)

则在数据元素 a1、a2 和 a3 之间存在着“次序”关系 <a1,a2>、<a2,a3>

3214，6587，9345 ≠ **6587，3214，9345**

a1　a2　a3　　**a2　a1　a3**

又例，在2行3列的二维数组{a1, a2, a3, a4, a5, a6}中六个元素之间存在两个关系:

a1	a2	a3
a4	a5	a6

行的次序关系:

row = {<a1,a2>,<a2,a3>,<a4,a5>,<a5,a6>}

列的次序关系:

col = {<a1,a4>,<a2,a5>,<a3,a6>}

a1 a3 a5
a2 a4 a6

≠

a1 a2 a3
a4 a5 a6

再例，在一维数组 {a1, a2, a3, a4, a5, a6}
的数据元素之间存在如下的**次序关系**:

$$\{<a_i, a_{i+1}> \mid i=1, 2, 3, 4, 5\}$$

可见，不同的“关系”构成不同的“结构”

- **数据结构：**

带结构的数据元素的集合。

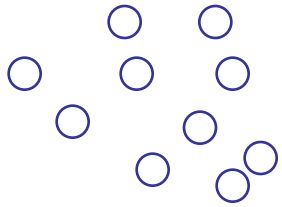
或者说，数据结构是相互之间存在着某种逻辑关系的数据元素的集合。

- **逻辑结构：**

结构定义中的“关系”描述的是数据元素之间的逻辑关系，因此又称为数据的逻辑结构。

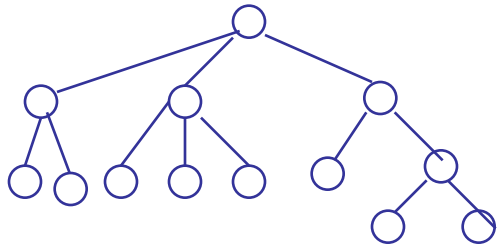
数据的逻辑结构可归结为以下四类：

- 集合：结构中的数据元素之间除了“同属于一个集合”的关系之外，别无其它关系。

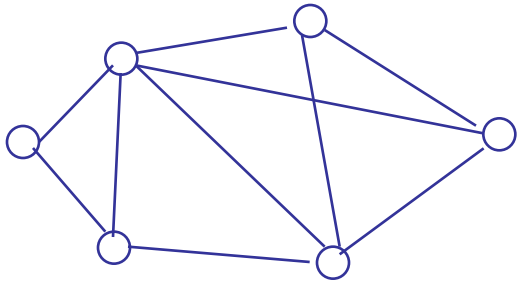


- 线性结构：结构中的数据元素之间存在一个对一个的关系。

- 树形结构：结构中的数据元素之间存在一个对多个的关系。



- 图状结构或网状结构：结构中的数据元素之间存在多个对多个的关系。



数据结构的形式定义为:

数据结构是一个二元组

Data_Structure = (D , S)

其中: D是数据元素的有限集

S是D上关系的有限集

例如: 复数可取如下定义: 复数是一种数据结构

Complex = (C , R)

其中: C是含两个实数的集合{c1,c2}; R = {P},

而P是定义在集合C上的一种关系{〈c1,c2〉},

其中有序偶〈c1,c2〉表示c1是复数的实部, c2是复数的虚部。

数据的存储结构：

数据结构在计算机中的表示(又称映象)称为数据的**物理结构**，又称为存储结构。存储结构是逻辑结构在存储器中的**映象**。

包括：
- “**数据元素**”的映象
- “**关系**”的映象

数据元素的映象方法:

计算机中表示信息的最小单位是二进制的一位(bit)。用二进制位(bit)的位串表示数据元素。

$$(321)_{10} = (501)_8 = (101000001)_2$$

$$A = (101)_8 = (001000001)_2$$

关系的映象方法：（表示<x，y>的方法）

数据元素之间的关系在计算机中有两种表示方法:

顺序映象，　非顺序映象

对应两种存储结构:

顺序存储结构，链式存储结构

顺序映象:

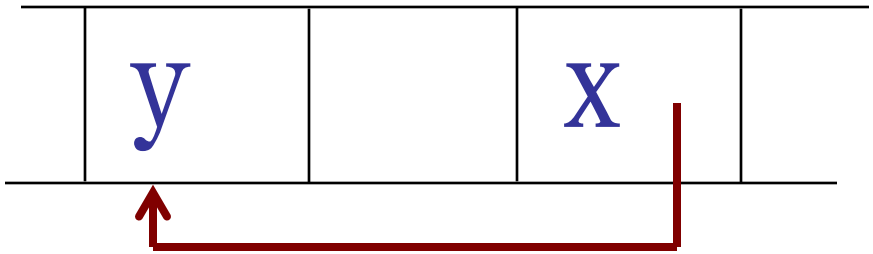
以元素在存储器中的相对位置来表示数据元素之间的逻辑关系

例如: 令 y 的存储位置和 x 的存储位置之间差一个常量 C，而 C 是一个隐含值，整个存储结构中只含数据元素本身的信息

链式映象:

用指示元素存储地址的指针来表示数据元素之间的逻辑关系。

以附加信息(指针)表示后继关系，需要用一个和 x 在一起的附加信息指示 y 的存储位置。



在不同的编程环境中，存储结构可有不同的描述方法。

当用高级程序设计语言进行编程时，通常可用高级编程语言中提供的数据类型描述之，不能直接用内存地址来描述存储结构。

因此，在高级程序语言中可以用：

“一维数组”类型描述顺序存储结构

“指针”类型描述链式存储结构

二、数据类型

在用高级程序语言编写的程序中，必须对程序中出现的每个变量、常量或表达式，明确说明它们所属的数据类型。

例如，C 语言中提供的**基本数据类型**有:

整型 **int**

浮点型 **float**

双精度型 **double**

} 实型（**C++**语言）

字符型 **char**

逻辑型 **bool** （**C++**语言）

不同类型的变量，其所能取的**值的范围**不同，所能**进行的操作**不同。

数据类型 是一个 **值的集合**和定义在此集合上的 **一组操作**的总称。

三、抽象数据类型
（Abstract Data Type 简称ADT）

1、抽象数据类型定义

是指一个数学模型以及定义在该模型上的一组操作。

例如，抽象数据类型**复数**的定义:

ADT Complex {

数据对象:

D = {e1,e2 ｜ e1,e2∈RealSet }

数据关系:

R1 = {<e1,e2> | e1是复数的实数部分

| e2 是复数的虚数部分 }

基本操作:

AssignComplex(&Z, v1, v2)

操作结果：构造复数 Z,其实部和虚部分别被赋以参数 v1 和 v2 的值。

DestroyComplex(&Z)

操作结果：复数Z被销毁。

GetReal(Z, &realPart)

初始条件：复数已存在。

操作结果：用realPart返回复数Z的实部值。

GetImag(Z, &ImagPart)

初始条件：复数已存在。

操作结果：用ImagPart返回复数Z的虚部值。

Add(z1,z2, &sum)

初始条件：z1，z2是复数。

操作结果：用sum返回两个复数z1，z2 的和值。

} ADT Complex

假设z1和z2是上述定义的复数，

则 **Add(z1, z2, z3)** 操作的结果即为用户要求的结果:

$$z3 = z1 + z2$$

2、抽象数据类型的描述方法

抽象数据类型可用（D，S，P）三元组表示。

其中：D是数据对象；

S 是D上的关系集；

P 是对D 的基本操作集。

抽象数据类型的定义格式:

ADT 抽象数据类型名 **{**

数据对象: 〈数据对象的定义〉

数据关系: 〈数据关系的定义〉

基本操作: 〈基本操作的定义〉

}ADT 抽象数据类型名

其中，数据对象和数据关系的定义用伪码描述。

基本操作的定义格式为：

基本操作名（参数表）

初始条件：〈初始条件描述〉

操作结果：〈操作结果描述〉

基本操作有两种参数:

赋值参数: 只为操作提供输入值

引用参数: 以&打头，除可提供输入值外，还将返回操作结果。

初始条件: 描述了操作执行之前数据结构和参数应满足的条件，若不满足，则操作失败，并返回相应出错信息。

操作结果: 说明了操作正常完成之后，数据结构的变化状况和应返回的结果。若初始条件为空，则省略之。

3、抽象数据类型的表示和实现

抽象数据类型需要通过固有数据类型(高级编程语言中已实现的数据类型)来实现。

例如，对以上定义的复数

```
// -----存储结构的定义
typedef  struct {
    float realpart;
    float imagpart;
}complex;
// -----基本操作的函数原型说明
void Assign( complex &Z,
                float realval, float imagval );
  // 构造复数 Z, 其实部和虚部分别被赋以参数
  //  realval 和 imagval 的值
```

```
float GetReal( cpmplex Z );
                    // 返回复数 Z 的实部值

float Getimag( cpmplex Z );
                    // 返回复数 Z 的虚部值

void add( complex z1, complex z2,
                complex  &sum );
            // 以sum返回两个复数 z1, z2 的和
```

```
// -----基本操作的实现

void add( complex z1, complex z2,
          complex  &sum )
{
    // 以 sum 返回两个复数 z1, z2 的和
  sum.realpart = z1.realpart + z2.realpart;
 sum.imagpart = z1.imagpart +z2.imagpart;
}
  { 其它省略 }
```

1.3 算法和算法分析

一、算法
二、算法设计的原则
三、算法效率的衡量方法和准则
四、算法的存储空间需求
五、算法的描述
六、算法的评价

一、算法

算法（Algorithm)是对特定问题求解步骤的一种描述，它是指令的有限序列，其中，每一条指令表示一个或多个操作。

例：计算 1-1/2+1/3-1/4+1/5-.....+1/99-1/100

算法1：自左至右逐项相加或相减；

算法2：将多项式写成两个多项式之差：

（1+1/3+1/5+...+1/99）-（1/2+1/4+1/6+...+1/100）

算法3：先通分，再运算；

一个算法必须满足以下五个重要特性：

- **有穷性：**

  对于任意一组合法输入值，在执行有穷步骤之后一定能结束，即算法中的每个步骤都能在有限时间内完成。

- **确定性：**

  在算法中每一条指令都有确切的含义，不会产生二义性，使算法的执行者或阅读者都能明确其含义及如何执行。并且在任何条件下，算法都只有一条执行路径。相同的输入只能得到相同的结果。

- **可行性**

  算法中的所有操作都必须足够基本，都可以通过已经实现的基本操作运算有限次实现之。

- **有输入**

  一个算法有零个或多个输入，作为算法加工对象的量值，通常体现为算法中的一组变量。有些输入量需要在算法执行过程中输入，而有的算法表面上可以没有输入，实际上已被嵌入算法之中。

- **有输出**

一个算法有零个或多个输出，它是一组与“输入”有确定关系的量值，是算法进行信息加工后得到的结果，这种确定关系即为算法的功能。

二、算法设计的原则

通常设计一个“好”的算法应考虑达到以下目标:

- 正确性
- 可读性
- 健壮性
- 高效率与低存储量需求

1. 正确性

首先，算法应当**满足**以特定的“**规格说明**”方式给出的**需求**。

其次，对算法是否“**正确**”的理解可以有以下**四个层次**：

a. 程序中不含语法错误；

b. 程序对于几组输入数据能够得出满足要求的结果

c. **程序对于精心选择的、典型、苛刻且带有刁难性的几组输入数据能够得出满足要求的结果；**

d. 程序对于一切合法的输入数据都能得出满足要求的结果；

通常以**第 c 层**意义的正确性作为衡量一个算法是否合格的标准

2. 可读性

算法主要是为了人的**阅读与交流**，其次才是为计算机执行，因此算法应该**易于**人的**理解**；另一方面，晦涩难读的程序易于隐藏较多错误而难以调试。

3. 健壮性

当**输入的数据非法**时，算法应当恰当地作出反映或**进行相应处理**，而不是产生莫名奇妙的输出结果。并且，**处理出错的方法**不应是中断程序的执行，而应是**返回一个表示错误或错误性质的值**，以便在更高的抽象层次上进行处理。

4. 高效率与低存储量需求

通常，效率指的是算法执行时间；存储量指的是算法执行过程中所需的最大存储空间，两者都与问题的规模有关。

三、算法效率的衡量方法和准则

通常有**两种**衡量算法效率的方法:

1、事后统计法

缺点:

1. 必须先运行依据算法编制的程序

2. 所得时间的统计量依赖于计算机的硬件、软件等环境因素，有时容易掩盖算法本身的优劣。

2、事前分析估算法

和算法执行时间相关的因素：

1. 算法选用的策略
2. 问题的规模
3. 编写程序的语言
4. 编译程序产生的机器代码的质量
5. 计算机执行指令的速度

显然，同一个算法用不同的语言实现，或者用不同的编译程序进行编译，或者在不同的计算机上运行时，效率均不相同。这表明用绝对的时间单位衡量算法的效率是不合适的。撇开这些与计算机硬件、软件有关的因素，可以认为：

一个特定算法的“运行工作量”的大小，只依赖于问题的规模（通常用整数量n表示），或者说，它是问题规模的函数。

一般情况下，算法中基本操作重复执行的次数是问题规模n的某个函数f (n)，算法的时间量度记作

$$T(n) = O(f(n))$$

假如，随着问题规模 n 的增长，算法执行时间的增长率和 f(n) 的增长率相同，则

称T (n) 为算法的(渐近)时间复杂度。

一个算法有控制结构和原操作构成，则算法的时间取决于两者的综合效果。

算法 = 控制结构 + 原操作

（顺序、分支、循环） （固有数据类型的操作）

为了便于比较同一问题的不同算法，通常的做法是，从算法中选取一种对于所研究的问题（或算法类型）来说是基本操作的原操作，以该基本操作重复执行的次数作为算法的时间度量。

算法的执行时间 =

$\sum$原操作(i)的执行次数 × 原操作(i)的执行时间

算法的执行时间

与

原操作执行次数之和 成正比

从算法中选取一种对于所研究的问题来说是 **基本操作** 的原操作，以该基本操作 **在算法中重复执行的次数** 作为算法运行时间的衡量准则。

语句**重复执行的次数**称为语句的**频度**。

由于算法的**时间复杂度**考虑的只**是**对于问题**规模**n的**增长率**，则在难以精确计算基本操作执行次数（或语句频度）的情况下，只需求出它关于n的增长率**或阶**即可。

例:

① ++x; O(1) 常量阶

② for (i=1;i<=n;i++)

{++x;} O(n) 线性阶

③ for (i=1;i<=n;i++)

for (j=1;j<=n;j++) O(n^2) 平方阶

{++x;}

还可能呈现的时间复杂度有：对数阶O(log n)、指数阶O(2^n)等。

例一 两个矩阵相乘

```
void mult(int a[], int b[], int c[] ) {
   //以二维数组存储矩阵元素，c 为 a 和 b 的乘积*/
for (i=1; i<n; i ++)
     for (j=1; j<n; j ++) {
        c[i,j] = 0;
        for (k=1; k<n; k ++)
           c[i,j] += a[i,k]*b[k,j];
     }
}//mult
```

基本操作：**乘法操作**

时间复杂度：$O(n^3)$

例二 起泡排序

```
void bubble_sort(int& a[], int n) {
  // 将 a 中整数序列重新排列成自小至大有序的整数序列。
for (i=n-1, change=TRUE;  i>1 && change;  --i)
 {  change = FALSE;  // change 为元素进行交换标志
      for (j=0;  j<i;  ++j)
        if (a[j] > a[j+1])
          { a[j] ←→ a[j+1];   change = TRUE ;}
 } // 一趟起泡
} // bubble_sort
```

基本操作：赋值操作

时间复杂度：$O(n^2)$

四、算法的存储空间需求

算法的空间复杂度定义为:

$$S(n) = O(f(n))$$

表示随着问题规模 n 的增大，算法运行所需存储量的增长率与 f(n) 的增长率相同。

算法的存储量包括:

1. 输入数据所占空间

2. 程序本身所占空间

3. 辅助变量所占空间

若**输入数据**所占空间只取决于问题本身，**和算法无关**，则只需要分析除输入和程序之外的**辅助变量**所占**额外空间**。

若所需存储量依赖于特定的输入，则通常按**最坏情况**考虑。

五、算法的描述

描述一个算法可以采用不同的方法，常用的有：

- 自然语言
- 流程图及改进的流程图
- 伪代码
- N-S结构流程图
- PAD图

本课程采用介于伪码和C语言之间的类C语言，有时也用伪码描述一些只含抽象操作的抽象算法；这使得数据结构和算法的描述和讨论简明清晰，不拘泥于C语言的细节，又能容易转换成C或C++程序。

类C语言语法概要

1、预定义常量和类型：

```
//函数结果状态代码
# define TRUE      1
# define FALSE     0
# define OK        1
# define ERROR     0
# define INFEASIBLE     -1
# define OVERFLOW      -2
typedef  int  Status;
        // Status 是函数的类型，其值是函数结果状态代码
```

2、数据结构的表示（存储结构）用类型定义（typedef）描述。数据元素类型约定为ElemType，由用户在使用该数据类型时自行定义。

3、基本操作的算法都用以下形式的函数描述：

```
函数类型  函数名（函数参数表）
{  // 算法说明
   语句序列
} //函数名
```

除了函数的参数需要说明类型外，算法中使用的辅助变量可以不作变量说明，必要时对其作用给予注释。一般而言，

a、b、c、d、e 等用作数据元素名；

i、j、k、l、m、n 等用作整形变量名；

p、q、r 等用作指针变量名。

当函数返回值为函数结果状态代码时，函数定义为**Status**类型。为了便于算法描述，除了值调用方式外，增添了**C++**语言的引用调用的参数传递方式。在形参表中，以**&**打头的参数即为引用参数。

4、赋值语句有：

简单赋值　变量名 = 表达式；

串联赋值　变量名1 = 变量名2 = … = 变量名k = 表达式；

成组赋值　（变量名1，…，变量名k）=（表达式1，…，表达式k）；

结构名 = 结构名；

结构名 =（值1，…，值k)；

变量名[ ] = 表达式；

变量名[起始下标…终止下标]=变量名[起始下标…终止下标]；

交换赋值　变量名←→变量名；

条件赋值　变量名 = 条件表达式？表达式T：表达式F；

5、选择语句有：

条件语句1　if （表达式）语句；

条件语句2　if （表达式）语句；
　　else 语句；

开关语句1　switch（表达式）
　　{ case 值1：语句序列1；break；
　　. . .
　　case 值n：语句序列n；break；
　　default：语句序列n+1；
　　}

开关语句2　switch
　　{ case 条件1：语句序列1；break；
　　. . .
　　case 条件n：语句序列n；break；
　　default：语句序列n+1；
　　}

6、循环语句有：

for 语句　　for（赋初值表达式序列；条件；修改表达式序列）
　　语句；

while 语句　　while (条件) 语句；

do-while 语句　　do {
　　语句序列；
　　} while (条件）；

7、结束语句有：

函数结束语句　　return 表达式；

　　return;

case结束语句　　break;

异常结束语句　　exit (异常代码）；

8、输入和输出语句有：

输入语句　　**scanf ([格式串]，变量1，···，变量n）；**

输出语句　　**printf ([格式串]，表达式1，···，表达式n）；**

通常省略格式串。

9、注释

单行注释　　// 文字序列

10、基本函数有

求最大值　　max（表达式1，···，表达式n）

求最小值　　min（表达式1，···，表达式n）

求绝对值　　abs (表达式）

求不足整数值　floor (表达式）

求进位整数值　ceil (表达式）

判定文件结束　eof (文件变量）或 eof

判定行结束　　eoln (文件变量）或 eoln

11、逻辑运算约定

与运算&&：对于A&&B，当A的值为0时，不再对B求值。

或运算||：　对于A||B，当A的值为非0时，不再对B求值。

六、算法的评价

1、执行算法所耗费的时间，即时间复杂度。

$$T(n)=O(f(n))$$

2、执行算法所耗费的存储空间，其中主要考虑辅助存储空间的大小，即空间复杂度。记作

$$S(n)=O(f(n))$$

其中，n为问题的规模（或大小）。

3、算法是否易读、是否容易转换成任何其它可运行的语言编制的程序以及是否易被测试等等。

本章学习要点

1. 熟悉各名词、术语的含义，掌握基本概念。

2. 理解算法五个要素的确切含义。

3. 掌握计算语句频度和估算算法时间复杂度的方法。

习题： 1.8　1.12

1.8　设n为正整数，试确定下列程序段中前置以记号@的语句的频度

（1）i=1; k=0;
```
while(i<=n-1){
@ k+=10*i;
i++;}
```

（2）i=1; k=0;
```
do {
@ k+=10*i;
i++;
}while(i<=n-1);
```

（3）i=1; k=0;
```
while(i<=n-1){
i++;
@ k+=10*i;}
```

(4) k=0;
```
for(i=1;i<=n;i++){
for(j=i;j<=n;j++)
@ k++;}
```

(5)
```
for(i=1;i<=n;i++){
    for(j=1;j<=i;j++){
        for(k=1;k<=j;k++)
        @ x+=delta;
}}
```

(6)
```
i=1; j=0;
while(i+j<=n){
    @ if(i>j) j++;
        else i++;
}
```

(7)
```
x=n;  y=0;  //n不小于1
while(x>=(y+1)*(y+1)){
    @ y++;
}
```

(8)
```
x=91;  y=100;
while(y>0){
    @ if (x>100)
        {x-=10;  y--;}
    else x++;
}
```

1.12　设有以下三个函数：　$f(n)=21n^4+n^2+1000$,

$g(n)=15n^4+500n^3$,　　$h(n)=5000n^{3.5}+n\log n$

请判断以下断言正确与否:

（1）f(n)是O(g(n))
（2）h(n)是O(f(n))
（3）g(n)是O(h(n))
（4）h(n)是$O(n^{3.5})$)
（5）h(n)是O(nlogn)