数据结构与算法分析

- 教材：《数据结构(C语言版)》。严蔚敏，吴伟民 编著。清华大学出版社。
- 参考文献：
- 1 《数据结构》。张选平，雷咏梅 编， 严蔚敏 审。 机械工业出版社。
- 2 《数据结构与算法分析》。Clifford A. Shaffer著， 张铭，刘晓丹 译。电子工业出版社。
- 3 《数据结构习题与解析(C语实言版)》。李春葆。 清华大学出版社。
- 4 《数据结构与算法》。夏克俭 编著。国防工业出版社。

讲师：杨镭

邮箱：nscczzyanglei@zzu.edu.cn

课程内容

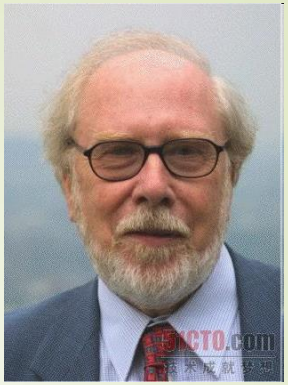
- 凭借一句话获得图灵奖的Pascal语言之父——Nicklaus Wirth，让他获得图灵奖的这句话就是他提出的著名公式：

  “程序=数据结构+算法”

- 这个公式对计算机科学的影响程度足以类似物理学中爱因斯坦的“E=MC^2”

  一个公式展示出了程序的本质

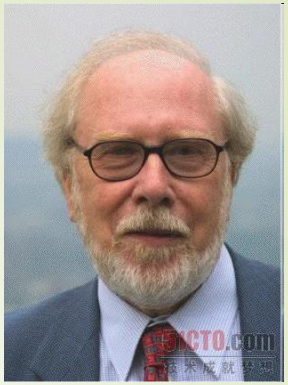
PASCAL语言创始人:Niklaus Wirth



如果说有一个人因为一句话而得到了图灵奖，那么这个人应该就是Niklaus Wirth。让他获奖的这句话就是他提出的著名公式：“算法+数据结构=程序”。

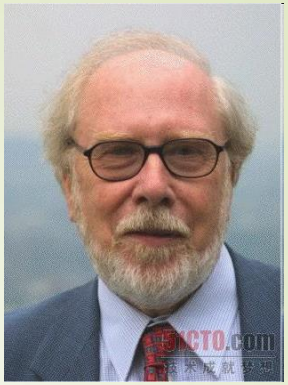
威茨1934年出生于瑞士，从小喜欢动手动脑，最大爱好是组装飞机模型。1958年从苏黎世工学院取得学士学位后，到加拿大的莱维大学深造，后进入美国加州大学伯克利分校获得博士学位，直接被斯坦福大学聘到刚成立的计算机科学系工作。在此成功开发出Algol W以及PL360。PL360是作为辅助工具开发的，却出人意料地在许多地方获得应用。 Algol W及PL360的成功奠定了威茨作为程序设计语言专家的地位。

PASCAL语言创始人:Niklaus Wirth



成名后的他拒绝了斯坦福大学的挽留，于1967年回到祖国，先在苏黎世大学任职，第二年转到母校苏黎世工学院。在这里，他成功开发了PASCAL语言。其初衷只是为了有一个适于教学的语言，没有考虑商业用途。PASCAL一经推出，由于它的简洁明了，更由于它特别适合于由微处理器组成的计算机系统，竟然广泛流传开来。在C语言问世以前，PASCAL成了最受欢迎的语言之一。后来他的学生Philipe Kahn毕业后和Anders Hejlsberg(Delphi之父)创办了Borland公司靠Turbo Pascal起家，很快将Borland发展成为全球最大的开发工作厂商。

PASCAL语言创始人:Niklaus Wirth



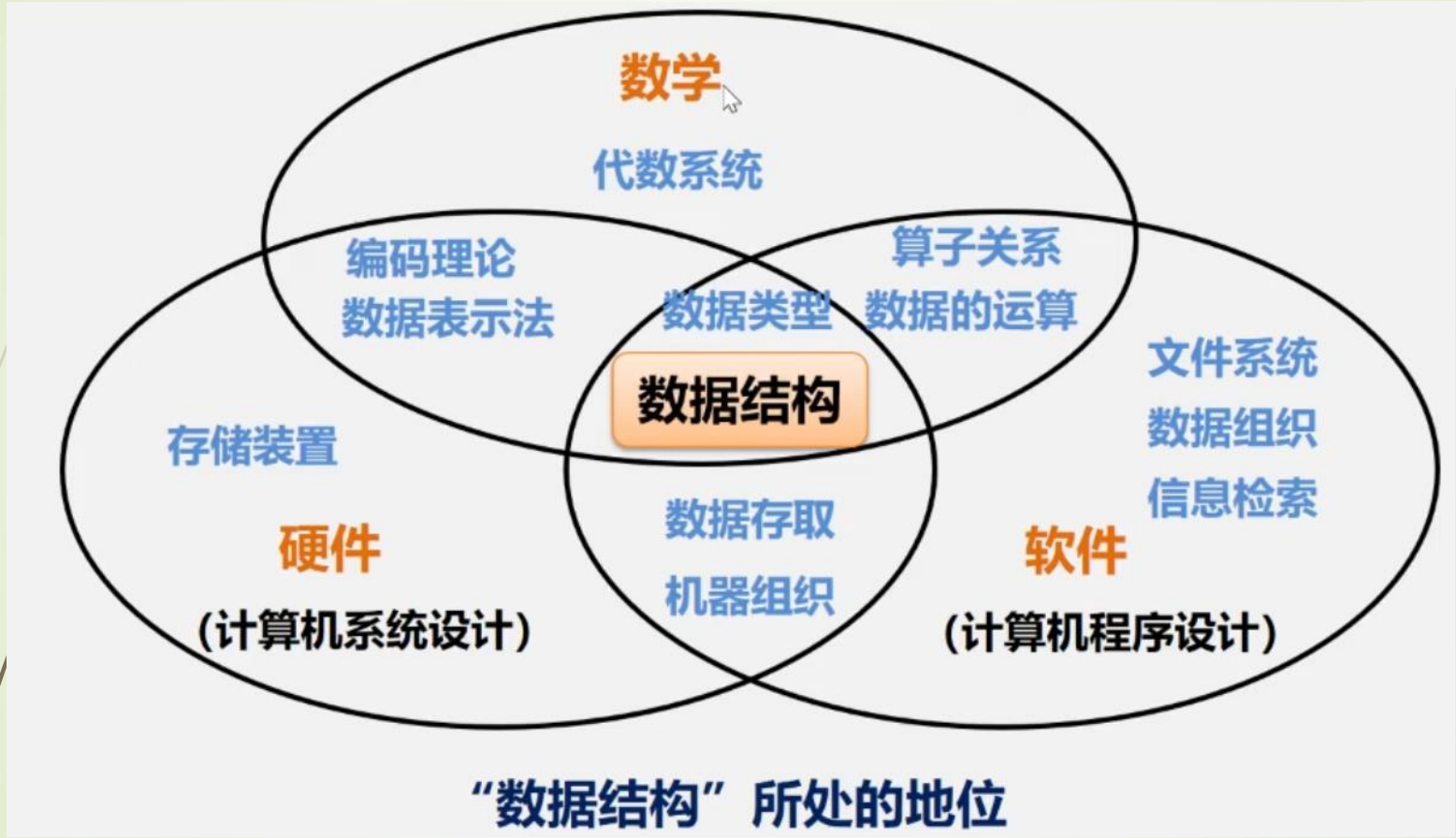
1971年，基于自己的开发程序设计语言和编程的实践经验，威茨首次提出了“结构化程序设计” 概念。这种方法又称为“自顶向下”或“逐步求精”法，在程序设计领域引发了一场革命，成为其中的一种标准，尤其在后来发展起来的软件工程中获得广泛应用。威茨的学术著作很多，著名的包括《系统程序设计导论》、《算法+数据结构=程序》、《算法和数据结构》、《PASCAL用户手册和报告：ISO PASCAL标准》等书籍。除得到了美国计算机制造商协会（ACM）颁发的图灵奖（1984，这是瑞士学者中唯一获此殊荣的人）外，他还得到过ACM授予的另一项大奖：计算机科学教育杰出贡献奖。

图灵奖

美国计算机协会（ACM）于1966年设立，专门奖励那些对计算机事业作出重要贡献的个人。其名称取自计算机科学的先驱、英国科学家阿兰·图灵，这个奖设立目的之一是纪念这位科学家。获奖者的贡献必须是在计算机领域具有持久而重大的技术先进性的。大多数获奖者是计算机科学家。

图灵奖是计算机界最负盛名的奖项，有“计算机界诺贝尔奖”之称。图灵奖对获奖者的要求极高，评奖程序也极严，一般每年只奖励一名计算机科学家，只有极少数年度有两名以上在同一方向上做出贡献的科学家同时获奖。目前图灵奖由Intel公司赞助，奖金为100,000美元。

“数据结构”所处的地位



- 数据结构是一门介于数学、计算机硬件和计算机软件之间的核心课程

课程内容

- 基本结构（三部分）
  - 数据结构的基本概念（第1章 绪论）
  - 基本的数据结构
    - 线性结构
      - 线性表（第2章）
      - 栈和队列（第3章）
      - 串（第4章）
      - 数组和广义表（第5章）
    - 非线性结构
      - 树（第6章）
      - 图（第7章）
  - 基本的数据处理技术
    - 动态存储管理（第8章）
    - 查找技术（第9章）
    - 内部排序（第10章）

这门课好学吗?

- 概念性强，抽象
- 算法灵活，不易掌握
- 逻辑性强，算法设计很烧脑

怎么学好这门课呢?

- 勤于思考;
- 多做练习;
- 多上机;
- 善于寻求帮助;
- 不怕困难，不放弃!

成绩计算方法

- 考核方式：过程性考核（包括：作业考核、上机实验）、期末考试。
- 过程性成绩（包括作业考核+上机实验）（30%）、期末考试（70%）。

第一章 绪论

1. 数据结构研究
2. 基本概念及术语
3. 抽象数据类型的表示与实现
4. 算法和算法分析

第一章 绪论

- 理解数据、数据结构、数据类型相关的定义和术语，了解数据抽象；
- 了解算法的定义、特性、时间代价和空间代价，会计算时间复杂度；
- 掌握用类C语言描述算法的方法，能够使用C语言编写程序。

数据结构研究

目前，计算机已深入到社会生活的各个领域，其应用已不再仅仅局限于科学计算，而更多的是用于控制，管理及数据处理等非数值计算领域。计算机是一门研究用计算机进行信息表示和处理的科学。这里面涉及到两个问题：信息的表示，信息的处理。

信息的表示和组织又直接关系到处理信息的程序的效率。随着应用问题的不断复杂，导致信息量剧增与信息范围的拓宽，使许多系统程序和应用程序的规模很大，结构又相当复杂。因此，必须分析待处理问题中的对象的特征及各对象之间存在的关系，这就是数据结构这门课所要研究的问题。

计算机求解问题的一般步骤

编写解决实际问题的程序的一般过程：

- 如何用数据形式描述问题?——即由问题抽象出一个适当的数学模型;
- 提取操作对象（数据），及操作对象之间的关系（数据结构）；
- 如何在计算机中存储数据及体现数据之间的关系?
- 处理问题时需要对数据作何种运算?
- 所编写的程序的性能是否良好?

上面所列举的问题基本上由数据结构这门课程来回答。

数据结构研究

早期计算机主要用于数值计算

例1，求解梁架结构中的应力。

数学模型：K U = M 线性方程组

$$\begin{bmatrix} a_{11} & & \\ & \ddots & \\ & & a_{nn} \end{bmatrix} \times \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$$

例2，预报人口增长情况。

数学模型：微分方程

$$\begin{cases} \dfrac{dN(t)}{dt} = r\,N(t) \\ N(t)\big|_{t=t_0} = N_0 \\ N(t) = N_0\,e^{rt} \end{cases}$$

首先，分析问题、提取操作对象；
然后，找出操作对象之间的关系，用数学语言加以描述，建立相应数学方程；
最后，求解数学方程
——计算机数学范畴

特点：数据元素之间的关系简单，计算复杂

数据结构研究

随着计算机应用领域不断扩展，计算机被越来越多地应用于非数值计算

例如学生学籍管理系统：

学生基本信息表

学号	姓名	性别	籍贯	专业
2024001	XX	男	安徽	计算机科学与技术
2024002	XXX	男	河南	计算机科学与技术
2024003	XX	女	吉林	计算机科学与技术
2024004	XXX	女	山东	计算机科学与技术

操作对象：每位学生的信息（学号、姓名、性别、籍贯、专业...）

操作算法：新增，修改，删除，查询等

操作对象之间的关系：线性关系　　数据结构：线性数据结构，线性表

数据结构研究

例1：电话号码查询系统

设有一个电话号码薄，它记录了N个人的名字和其相应的电话号码，假定按如下形式安排：(a_1, b_1)，(a_2, b_2)，…(a_n, b_n)，其中$a_i, b_i(i=1，2…n)$ 分别表示某人的名字和电话号码。 本问题是一种典型的表格问题。如表1-1，数据与数据成简单的一对一的线性关系。

姓名	电话号码
陈海	13612345588
李四锋	13056112345
。 。 。	。 。 。

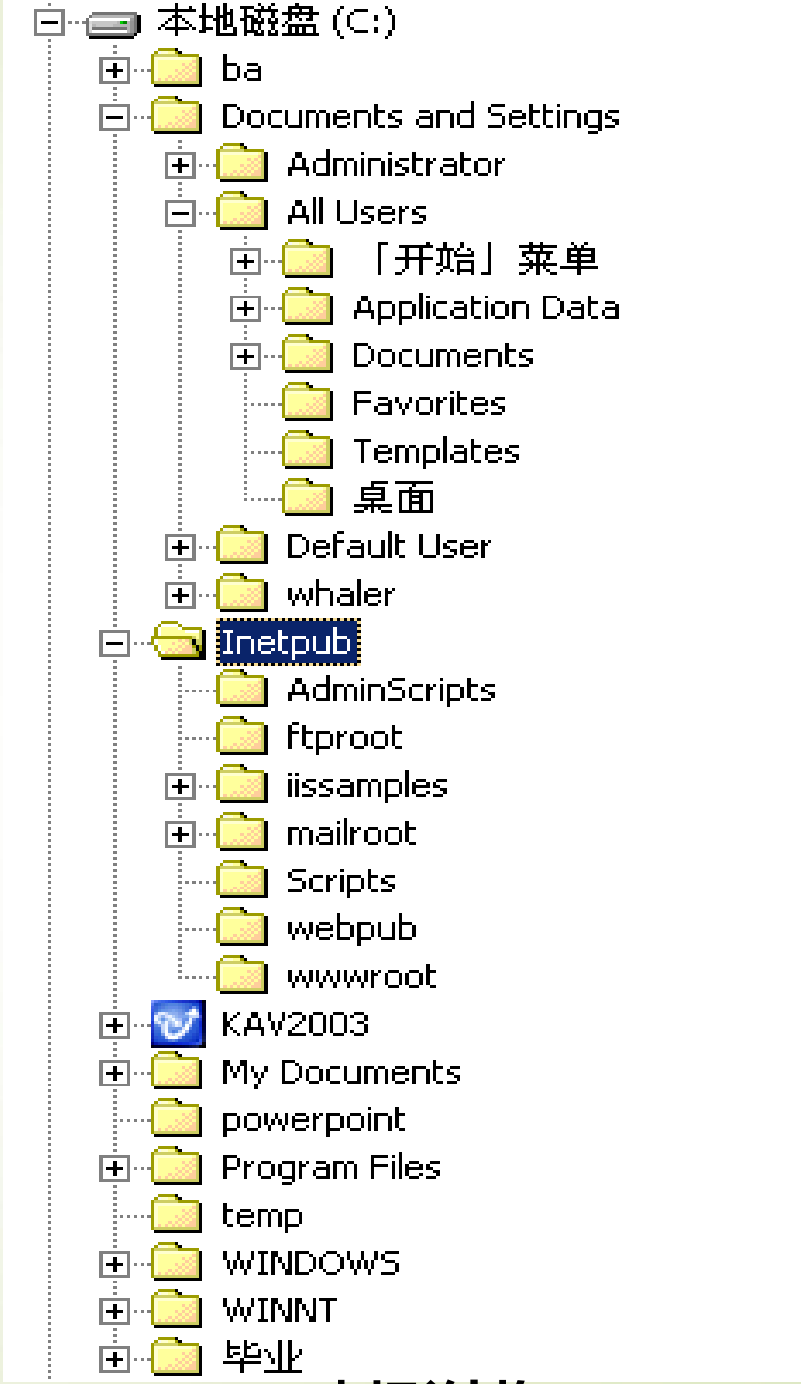
线性表结构

数据结构研究

例2：磁盘目录文件系统

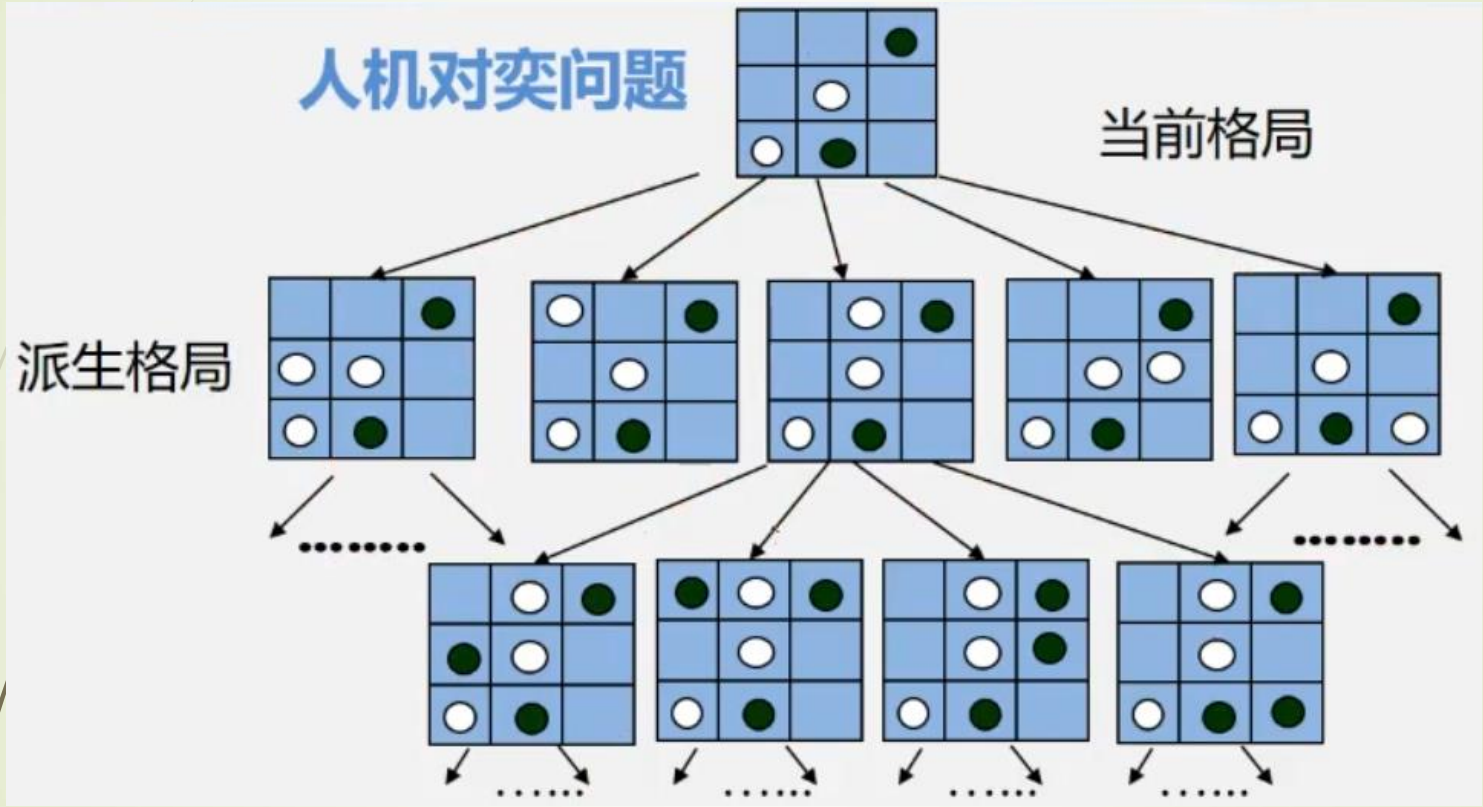
磁盘根目录下有很多子目录及文件，每个子目录里又可以包含多个子目录及文件，但每个子目录只有一个父目录，依此类推：

本问题是一种典型的树型结构问题，如图，数据与数据成一对多的关系，是一种典型的非线性关系结构—树形结构。



树形结构

数据结构研究

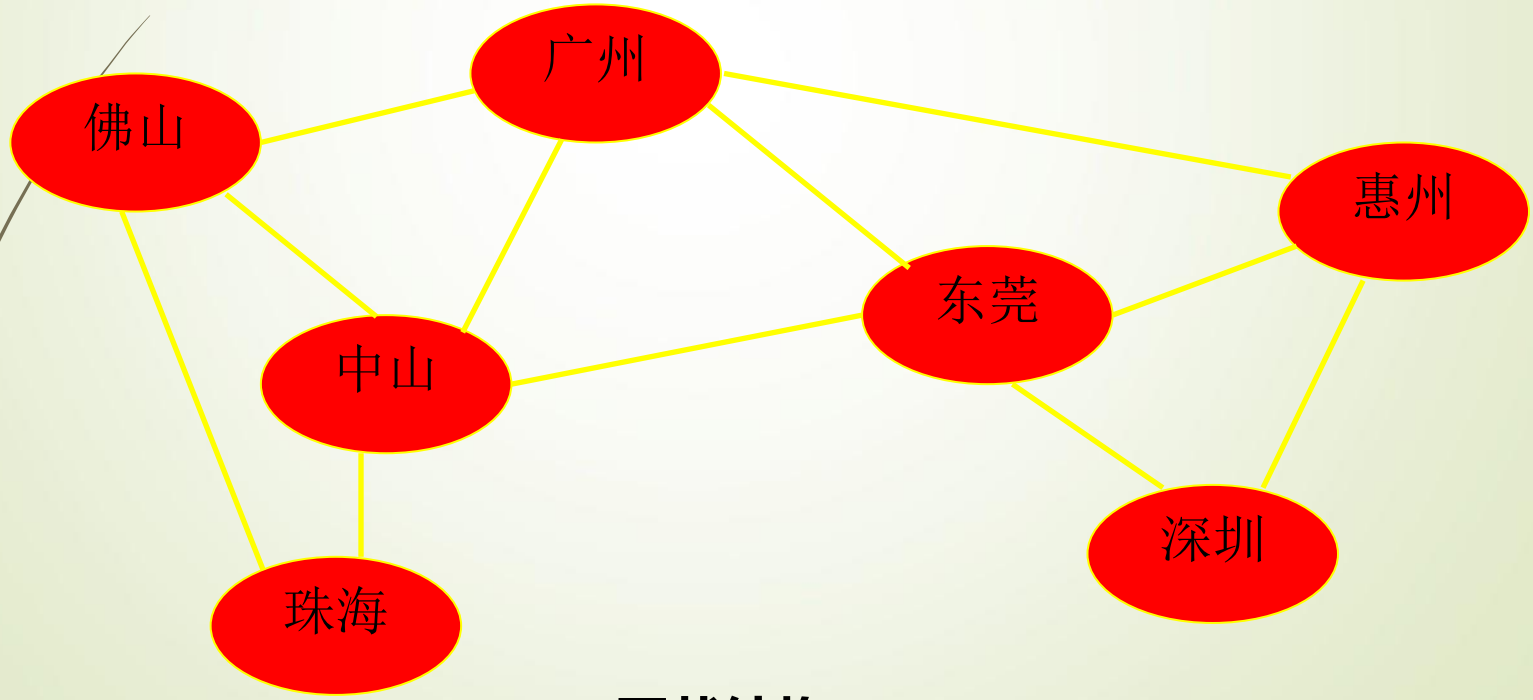


计算机的操作对象：各种棋局状态，即描述棋盘的格局信息
计算机的算法：走棋，即选择一种策略使棋局状态发生变化（由一个格局派生出另一个格局）

数据结构的例子

例3：交通网络图

从一个地方到另外一个地方可以有多条路径。本问题是一种典型的网状结构问题，数据与数据成多对多的关系，是一种非线性关系结构。



网状结构

数据结构研究内容总结

- 这些问题的共性是都无法用数学的公式或方程来描述，是一些“非数值计算”的程序设计问题
- 描述非数值计算问题的数学模型不是数学方程，而是诸如表、树和图之类具有逻辑关系的数据
- 数据结构是一门研究非数值计算的程序设计中计算机的操作对象以及它们之间的关系和操作的学科
- 要想有效地使用计算机，就必须学习数据结构

基本概念和术语：关于数据

数据：是对客观事物的符号表示，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。

数据元素：是数据的基本单位，在计算机系统中通常作为一个整体进行考虑和处理，也称结点或记录。

- 有时一个数据元素可由若干个数据项组成

图书编号	书名	作者	出版单位	出版时间
N.3.73.762	数据结构	严蔚敏 吴伟民	清华大学出版社	2003.12

- 数据项是有独立含义的，是数据不可分割的最小单位。

基本概念和术语：关于数据

数据对象：性质相同的数据元素的集合，是数据的一个子集。

- 可以是有限集。例如：字母字符数据对象C={‘A’，‘B’，…’Z’}
- 可以是无限集。例如：整数数据对象 $N = \{ 0, \pm1, \pm2, \dots \}$
- 可以是象上面例子中的单一数据项的元素的集合
- 也可以是多数据项组成的元素的集合

图书编号	书名	作者	出版单位	出版时间
N.3.73.762	数据结构	严蔚敏 吴伟民	清华大学出版社	2003.12
H. 41.683	实用英语语法	张道真	外语教学与研究出版社	1995.4
N.3.73.768	数据结构	吴小平 马桂媛	机械工业出版社	2009.8
…	…	…	…	…

数据元素与数据对象

- 数据元素：组成数据的基本单位
  - 与数据的关系：是集合的个体
- 数据对象：性质相同的数据元素的集合
  - 与数据的关系：集合的子集

基本概念和术语：数据结构

数据结构：是相互之间存在一种或多种特定关系的数据元素的集合。狭义地讲，它就是数据的逻辑结构；广义地讲，它是数据的逻辑与物理结构的总和。

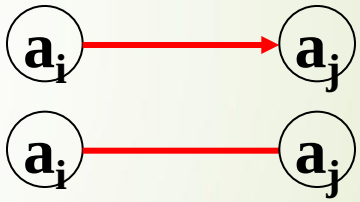
- 任意一个算法的设计取决于选定的逻辑结构
- 算法的实现依赖于采用的物理结构

基本概念和术语：数据的逻辑结构

数据的*逻辑结构*：描述数据元素之间的逻辑关系。存在集合、线性、树、图四种基本结构。

Data_Structure=(D,S)

- D：数据元素的有限集合，$D=\{a_1,a_2,\ldots,a_n\}$
- S：定义在D上的关系的有限集合。
  - 若a_iRa_j，则$<a_i,a_j>\in S$
  - 若a_iRa_j且a_jRa_i，则$(a_i,a_j)\in S$



- 例：复数可定义为一种数据结构 **COMPLEX=(C, R)**

  $C=\{C_1, C_2\}$，是含有两个实数的集合；

  $R=\{P\}$是定义在C上的一种关系$\{<C_1,C_2>\}$

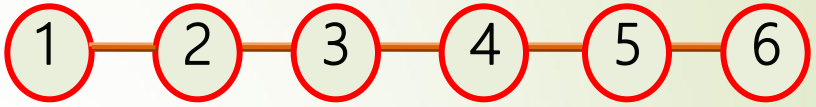
  有序！

- 课堂练习：**T=(K, R)**，画出它所对应的逻辑结构。其中：
  K={1,2,3,4,5,6}； R ={r}， r={<1,2>,<1,3>,<2,4>,<2,5,>,<3,6>}

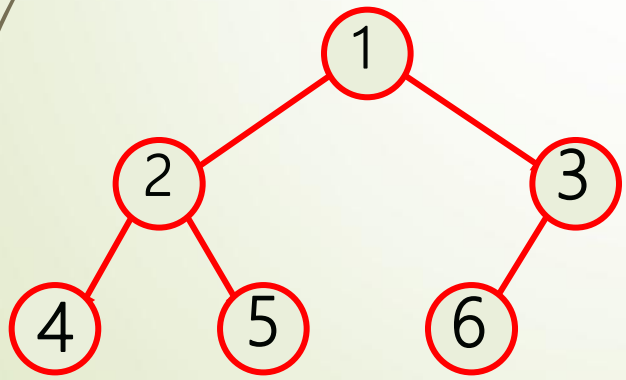
基本概念和术语：四类基本逻辑结构图

集合结构

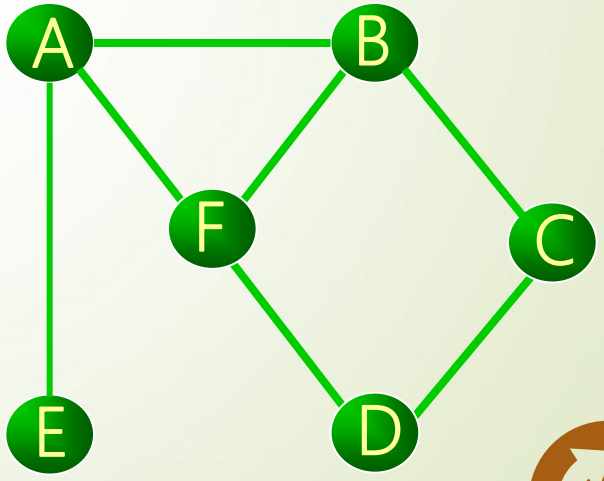
线性结构



树形结构



图形结构



基本概念和术语：数据的物理结构

数据的*物理结构*：数据元素和元素间关系在计算机中的表示，即数据的存储结构。

- 数据元素的表示
- 数据元素之间关系的表示

基本概念和术语：数据的物理结构

四种基本的的存储结构：

- 顺序存储结构
- 链式存储结构
- 索引存储结构
- 散列存储结构

基本概念和术语：数据的物理结构

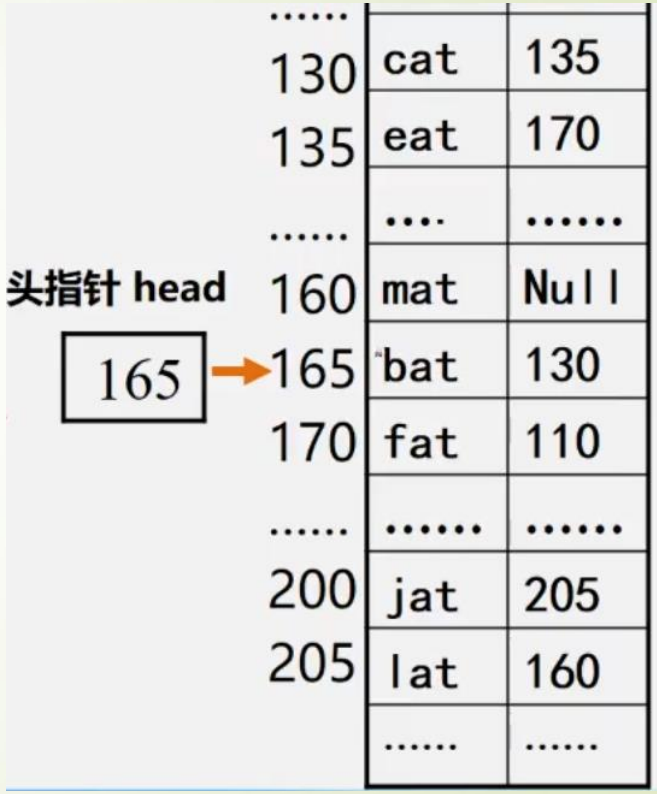
顺序存储结构：

- 用一组连续的存储单元依次存储数据元素，数据元素之间的逻辑关系由元素的存储位置来表示
- C语言中用数组来实现顺序存储结构
- 例如：顺序存储字符串（bat，cat，eat…）

基本概念和术语：数据的物理结构

链式存储结构：

- 用一组任意的存储单元存储数据元素，数据元素之间的逻辑关系由指针来表示
- C语言中用指针来实现链式存储结构
- 例如：（bat, cat, eat, …, mat)



基本概念和术语：数据的物理结构

索引存储结构：

- 在存储节点信息的同时，建立附加的索引表
- 索引表中的每一项称为一个索引
- 索引项的一般形式是（关键字，地址）

基本概念和术语：数据的物理结构

散列存储结构：

- 根据节点的关键字直接计算出该节点的存储地址

key	32	75	70	63	48	94	25	36	18
key%11	10	9	4	8	4	6	3	3	7

基本概念和术语：逻辑结构和物理结构

两者的关系：

- 物理结构是逻辑关系的映像与元素本身的映像
- 逻辑结构是数据结构的抽象，物理结构是数据结构的实现
- 两者综合起来建立了数据元素之间的结构关系

基本概念和术语：逻辑结构和物理结构

- 分析解决问题时可能遇到的元素及元素间典型的逻辑关系

- 元素及其逻辑关系在计算机内的存储

- 数据结构的相关操作及其实现(算法)

基本概念和术语：数据类型

数据类型：是和数据结构密切相关的一个概念，用以刻画（程序）操作对象的特性，是一个值的集合和定义在这个值集上的一组操作的总称。

数据类型=广义的数据结构 + 相关操作

- 例：C语言中的整型
  - 值：定义在某区间上的整数
  - 操作：加、减、乘、除、取模
- 按值的不同特性分：
  - 原子类型：值不可再分
    - C的五种基本数据类型: char, int, float, double, void
  - 结构类型：值由若干个成分按某种结构组成

基本概念和术语：抽象数据类型

抽象数据类型：**Abstract Data Type（ADT）**是指一个数学模型以及定义在该模型上的一组操作，即数据的逻辑结构+相关操作。

- 抽象数据类型可分为原子、固定聚合和可变聚合3种类型
  - 原子类型：值不可分解。如数位为100的整数
  - 固定聚合类型：值由确定数目的成分按某种结构组成。如复数
  - 可变聚合类型：值的成分数目不确定。如有序整数序列
- *多型数据类型*：是指其值的成分不确定的抽象数据类型
  - 线性表
  - 栈、队列
  - 树、二叉树
  - 图

基本概念和术语：抽象数据类型

- 抽象数据类型可通过固有的数据类型（整数、实型、字符型等）来表示和实现
- 即利用处理器中已存在的数据类型来说明新的结构，用以经实现的操作来组合新的操作

基本概念和术语：抽象数据类型

- 抽象数据类型可用三元组（D，S，P）表示。
- ADT的定义格式（仅适用于本书）：

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

伪码描述

伪码: 以夹杂程序语法和自然语言的形式来描述解决问题的方法

基本概念和术语：抽象数据类型

基本操作的定义格式为：

基本操作名（参数表）
初始条件：…
操作结果：…

- 参数表有两种参数：
  - 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。
- 初始条件描述操作执行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。若初始条件为空，则省略之。
- 操作结果说明操作正常完成之后数据结构的变化状况和应返回的结果

基本概念和术语：抽象数据类型描述示例

[例] 抽象数据类型“复数”的形式化描述：

ADT Complex{

数据对象：D={c1, c2 | c1, c2∈R(实数集)}

数据关系：R={< c1, c2> | c1是复数的实部，c2是复数的虚部}

基本操作：

Create(x,y,&z)

初始条件：x,y是已存在的实数

操作结果：构造复数 **z = x+yi**

Add(z1,z2,&sum)

初始条件：z1、z2是已存在的复数

操作结果：用**sum**返回z1+z2的结果

Subtract (z1,z2,&difference)

Multiply(z1,z2,&product)

Get_realpart(z)

Get_imagpart(z)

} ADT Complex

基本概念和术语：抽象数据类型的实现示例

[例] 数据类型"复数"的实现：

```
typedef struct {
    float realpart;    //实部
    float imagpart;   //虚部
} Complex; //复数类型
void Create(float x, float y, Complex *z)
{ //生成一个以x为实部，y为虚部的实数，由指针z指示它
    z->realpart=x;      z->imagpart=y;
}
void Add(Complex z1, Complex z2, Complex &sum)
{ //求复数z1和复数z2的和，结果由指针sum指示
    sum.realpart=z1.realpart+z2.realpart;
    sum.imagpart=z1.imagpart+z2.imagpart;
}    ……
```

抽象数据类型的表示与实现：算法

- 算法：是对特定问题求解步骤的描述，是指令的有限序列，其中每条指令表示一个或多个操作。
- 算法的特性：
  - 有穷性：算法应在执行有穷步后结束
  - 确定性：每步定义都是确切、无歧义的
  - 可行性：算法中描述的操作都可通过已经实现的基本运算执行有限次实现
  - 输　入：有0个或多个输入
  - 输　出：有一个或多个输出(处理结果)

抽象数据类型的表示与实现：算法的描述方法

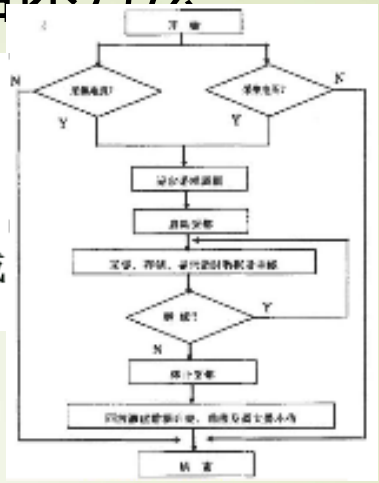
- 描述数据结构
  - 自然语言
  - 流程图
  - 表格
  - 高级程序设计语言
  - ……

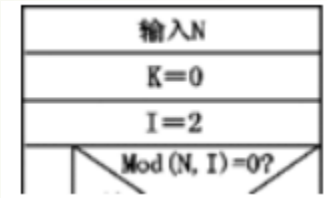
•拓扑排序步骤：

Step1:在有向图中选一个无前驱的顶点输出之；

Step2:从有向图中删去此顶点及所有以它为尾的弧；

Step3:重复Step1,2到图中顶点全部输出，此时图中无环；或图不空但找不到无前驱的顶点(有环)为止，此时图中有环。






```
template<class Type>
int BinarySearch(Type a[],const Type& x,int n)
{
    int left = 0;
    int right = n-1;
    while(left<=right)
    {
        int mid = (left + right)/2;
        if(x == a[mid])
        {
            return mid;
        }
        if(x>a[mid])
        {
            left = mid + 1;
        }
        else
        {
            right = mid - 1;
        }
    }
    return -1;
}
```

- 本书采用介于伪码和C语言之间的类C语言作为描述工具

抽象数据类型的表示与实现：类C语言

类C语言精选了C语言的一个核心子集，同时又做了若干扩充修改，增强了语言的描述功能，使得数据结构和算法的描述和讨论简明清晰，既不拘泥于C语言的细节，又能容易转换为C或C++程序。

抽象数据类型的表示与实现： 类C算法示例

例

```
typedef struct student{
    char id[5];
    char name[11];
    int age;
    int math;
    int eng;
    int ds;
    int os;
}student;
```

```
void example_3(student
    &s)
{
    s=(“01001”,
    “Wang”,0,0,0,0,0);
     ….
}
```

抽象数据类型的表示与实现

- 预定义常量和类型

```
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
#define OVERFLOW -2

typedef  int  Status;
```

抽象数据类型的表示与实现

- 数据结构的类型定义
  - 数据结构的表示用 C 中的类型定义（typedef)或C++中的类(class)来描述。
  - 新的数据类型可通过固有数据类型来表示与实现。
  - 数据元素类型约定为ElemType，使用时由用户自行定义。如:

```
typedef int ElemType;
```

抽象数据类型的表示与实现

- 基本操作算法的描述——函数

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

- 当函数返回值是函数结果的状态代码时，函数定义为Status类型
- 函数内的辅助变量可以不作声明；注意变量取名习惯
- 引用参数的定义方式：&

抽象数据类型的表示与实现

- 类C语言语句系统
  - 赋值语句
  - 选择语句
  - 循环语句
  - 结束语句
  - I/O语句
  - 注释

简单赋值：
变量名=表达式；
变量名++；　++变量名；
变量名--；　--变量名；
串联赋值：
变量名1= … =变量名n=表达式；
条件赋值：
变量名=条件表达式？表达式T：表达式F；
交换赋值：变量名1 <->变量名2;
成组赋值：数组名[]=表达式
数组名1[s..e]=数组名2[s..e]
结构名=（值1，值2，…，值k)

抽象数据类型的表示与实现

- 基本函数
  - 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)　　fabs(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/行结束：eof(文件变量)或eof

抽象数据类型的表示与实现

- 逻辑运算约定
  - &&
  - ||
  - !

类C语言描述的算法和C程序之间的区别

一个算法实例：

```
Status example(ElemType &x, ElemType &y, ElemType a[], int n)
{
    if(x>y)    x<->y;
    b[0..9]=a[0..9];  ….
    return OK;
}
```

由上述算法改编而成的C

```
typedef int ElemType;
Status example(ElemType
{
    ElemType b[10], temp;
    if  ( *x > *y )    {temp=*
    for( int i=0; i<10; i++) b[
    ……
    return OK;
}
```

C++程序实例：

```
typedef int ElemType;
Status example(ElemType &x,
ElemType &y, ElemType a[], int n){
    ElemType b[10], temp;
    if (x>y)    {temp=y; y=x; x=temp;}
    for( int i=0; i<n; i++) b[i]=a[i];
    ……
    return OK;
}
```

随着计算机产业的发展应用领域

- 数值计算领域：加工处理的对象—纯粹的数值
- 非数值计算领域：工业检测、过程管控、系统管理、文字处理、……

加工处理的对象：字符、表格、图像、声音、……

新的、杂乱无章、具有一定结构的数据，如何处理？

研究对象的特性及其相互之间的关系	有效地组织计算机存储	有效地实现对象之间的“运算”关系
逻辑结构	物理结构	算法

《数据结构与算法》的研究内容

基本概念和术语：逻辑结构和物理结构

- 分析解决问题时可能遇到的元素及元素间典型的逻辑关系

- 元素及其逻辑关系在计算机内的存储

- 数据结构的相关操作及其实现(算法)

抽象数据类型的表示与实现：算法

- 算法：是对特定问题求解步骤的描述，是指令的有限序列，其中每条指令表示一个或多个操作。
- 算法的特性：
  - 有穷性：算法应在执行有穷步后结束
  - 确定性：每步定义都是确切、无歧义的
  - 可行性：算法中描述的操作都可通过已经实现的基本运算执行有限次实现
  - 输　入：有0个或多个输入
  - 输　出：有一个或多个输出(处理结果)

算法的描述方法

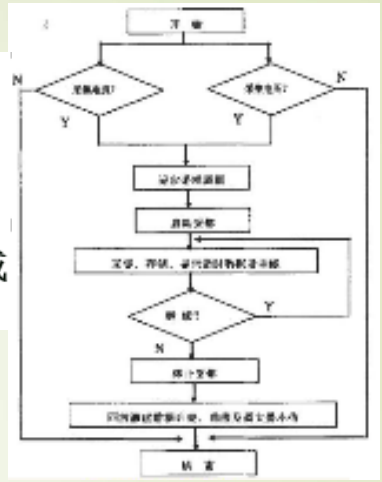
- 描述数据结构
  - 自然语言
  - 流程图
  - 表格
  - 高级程序设计语言
  - ……

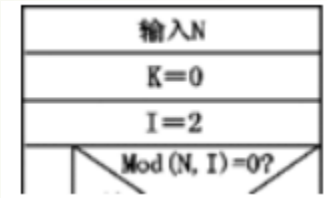
•拓扑排序步骤：

Step1:在有向图中选一个无前驱的顶点输出之；

Step2:从有向图中删去此顶点及所有以它为尾的弧；

Step3:重复Step1,2到图中顶点全部输出，此时图中无环；或图不空但找不到无前驱的顶点(有环)为止，此时图中有环。






```
template<class Type>
int BinarySearch(Type a[],const Type& x,int n)
{
    int left = 0;
    int right = n-1;
    while(left<=right)
    {
        int mid = (left + right)/2;
        if(x == a[mid])
        {
            return mid;
        }
        if(x>a[mid])
        {
            left = mid + 1;
        }
        else
        {
            right = mid - 1;
        }
    }
    return -1;
}
```

- 本书采用介于伪码和C语言之间的类C语言作为描述工具

算法和算法分析：算法设计的要求

正确性

- 不含语法错误；
- 对于几组输入数据能够得出满足要求的结果；
- 对精心选择带有刁难性的几组数据能得出满足要求的结果；
- 对于一切合法的输入数据都能产生满足要求的结果

可读性

- 首先应该易于阅读和交流
- 其次是机器运行

健壮性

- 输入非法时能适当作出反应或进行处理
- 出错时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

效率

- 时间上高效率
- 空间上低存储量

算法和算法分析：算法时间效率的度量方法

- 一个好的算法首先要具备正确性，然后是健壮性，可读性，在几个方面都满足的情况下，主要考虑算法的效率，通过算法的效率高低来评判不同算法的优劣程度

- 算法效率通过以下两个方面来考虑：
  - 时间效率：指的是算法所耗费的时间；
  - 空间效率：指的是算法执行过程中所耗费的存储空间。
- 时间效率和空间效率有时候是矛盾的

算法和算法分析：算法时间效率的度量方法

- 算法时间效率可以用依据该算法编制的程序在计算机上执行所消耗的时间来度量

➡两种度量方法：

➡事后统计：将算法实现，测算其时间和空间开销；

➡事前分析：对算法所消耗的资源的一种估算方法。

算法和算法分析：算法时间效率的度量方法

时间效率的度量方法：

- 事后统计：用计时功能统计算法转换成的程序的运行时间。
  - 缺陷：要运行算法对应的程序，花费时间和精力
    时间统计结果依赖于计算机软、硬件环境

```
#include <time.h>
using namespace std;
clock_t start, stop;
start=clock();
……
stop=clock();
clock_t runTime = (stop - start) /CLOCKS_PER_SEC;
cout<<runTime;
```

算法和算法分析：

时间效率的度量方法：

- 事后统计：用计时功能统
  - 缺陷：要运行算法对应的
    时间统计结果依赖于计
    影响程序消耗时间的因素
- 事前分析估计：
  - 算法采用的策略
  - 问题的规模
  - 书写程序的语言
  - 编译器产生的机器代码的质量
  - 计算机执行指令的速度

n越大算法的执行时间越长

- 排序：n为记录数
- 矩阵：n为矩阵的阶数
- 多项式：n为多项式的项数
- 集合：n为元素个数
- 树：n为树的结点个数
- 图：n为图的顶点数或边数

算法和算法分析：事前分析方法

- 一个算法的运行时间是指算法在计算机上运行所耗费的时间，大致可以等于计算机执行一种简单的操作（如赋值、比较、移动等）所需的时间与算法中进行的简单操作次数的乘积

  算法运行时间 = 一个简单操作所需时间 X 简单操作次数

- 也即算法中每条语句的执行时间之和

  算法运行时间 = ∑每条语句执行次数 X 语句执行一次所需时间

又称为语句频度

算法和算法分析：事前分析方法

算法运行时间 = ∑每条语句频度 X 语句执行一次所需时间

- 每条语句执行一次所需的时间一般是随机器而异的。取决于及其的指令性能、速度以及编译的代码质量。由及其本身软硬件环境决定的，它与算法无关。
- 所以我们可以假设每条语句执行所需的时间均为单位时间。此时对算法的运行时间的讨论就可以转化为讨论该算法中所有语句的执行次数，即频度之和了

算法和算法分析：事前分析方法

- 如果去除与软硬件有关的因素, 则一个算法“运行工作量”的大小应只依赖于问题的规模，或者说是问题规模 n 的某个函数。
- 从算法中选取一种对于所研究问题来说是基本操作的原操作，以该基本操作重复执行次数作为算法的时间量度。算法中基本操作重复执行次数是问题规模n的某个函数f(n)
- 语句频度：语句重复执行的次数。

算法和算法分析：算法的时间复杂度

算法时间效率的度量——渐近时间复杂度

时间复杂度：一般情况下，算法中基本操作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

$$T(n)=\mathbf{O}(f(n))$$

表示随问题规模n的增大，算法执行时间的增长率和f(n)的增长率相同，称作算法的*渐近时间复杂度*，简称*时间复杂度*

例 1

两个n*n矩阵相乘的算法

```
for(i=1;i<=n;++i)                          // n+1次
 for(j=1; j<=n; ++j) {                     // n(n+1)次
   c[i][j] = 0;                            // n*n次
   for(k=1;k<=n;++k)                       // n*n*(n+1)次
       c[i][j]+=a[i][k]*b[k][j];           // n*n*n次
 }
```

上述算法消耗时间T(n)为：

$$T(n)=2n^3+3n^2+2n+1$$

算法和算法分析：算法的时间复杂度

为了便于比较不同算法的时间复杂度，我们仅比较他们的数量级

例如：两个不同的算法，时间消耗分别是：

$T_1(n)=10n^2$ 与 $T_2(n)=5n^3$

若有某个辅助函数f(n)，使得当n趋近于无穷大时，T(n)/f(n)的极限值为不等于0的常数，则称f(n)是T(n)的同数量级函数。记作T(n)=O(f(n))，称O(f(n))为算法的*渐近时间复杂度*（O是数量级的符号），简称*时间复杂度*

算法和算法分析：算法的时间复杂度

- 定理1.1:

  $f(n)=a_m n^m+a_{m-1}n^{m-1}+\ldots+a_1 n+a_0$是m次多项式，则

$T(n)=O(n^m)$

- 忽略低次幂项和最高次幂项的系数，体现出增长率的含义

例 1

两个n*n矩阵相乘的算法

```
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];
 }
```

时间复杂度分析

- 问题规模

  n*n

- 基本操作

  c[i][j] += a[i][k] * b[k][j];

- 基本操作重复执行的次数

  $f(n) = n^3$

- 时间复杂度

  $T(n)=2n^3+3n^2+2n+1$

  $n\to\infty$时，$T(n)/ n^3\to 2$

  $T(n)=O(f(n))=O(n^3)$

例 1

两个n*n矩阵相乘的算法

```
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];
 }
```

一般情况下，只需考虑算法中基本操作执行的次数，它是问题规模n的某个函数，用T(n)表示

时间复杂度分析

- 问题规模

  n*n

- 基本操作

  c[i][j] += a[i][k] * b[k][j];

- 基本操作重复执行的次数

  $f(n) = n^3$

- 时间复杂度

  $T(n)=2n^3+3n^2+2n+1$

  $n\to\infty$时，$T(n)/ n^3\to 2$

  $T(n)=O(f(n))=O(n^3)$

例2

程序段	语句频度	时间复杂度
{++x;s=0}@	1	O(1)
for(i=1;i<=n;++i) {++x;s+=x;}@	n	O(n)

for(i=1;i<=n;++i)
for(j=1;j<=n;++j)
{++x;s+=x;}@
	1+2+…+n
	n^2
		O(n^2)

i=1;
while(i<=n)
i=i*2; @
	$\log_2 n$	O($\log_2 n$)

例2

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

- 语句频度：

$$\sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=1}^{j}1 = \sum_{i=1}^{n}\sum_{j=1}^{i}j = \sum_{i=1}^{n}\frac{i(i+1)}{2} = \frac{n(n+1)(n+2)}{6}$$

- 时间复杂度：

$$T(n)=O(n^3)$$

例2

```
i=1;
while(i<=n)
    i=i*2@;
```

关键是要找出执行次数x与n的关系，并表示成n的函数

若循环执行1次：i=1*2=2,

若循环执行1次：$i=2*2=2^2$,

若循环执行3次：$i=2*2*2=2^3$,

若循环执行X次：$i=2^X$,

设语句@执行次数为X次，由循环条件

$i<=n$，$\therefore 2^X<=n$　$\therefore X<=\log_2 n$

$2f(n)\leq \log_2 n$

即$f(n)\leq \log_2 n$，

取最大值$f(n)= \log_2 n$

算法和算法分析：时间复杂度计算的基本方法

- 时间复杂度计算的基本方法
  - 找出语句频度最大的语句作为基本语句。
  - 计算基本语句的频度得到问题规模n的某个函数f(n)。
  - 取其数量级用符号O表示

算法和算法分析： O( )函数

- O( )函数的运算规则
  - O(c*f(n))=O(f(n)),其中c为常数
  - O(f(n)+g(n))=O(max(f(n),g(n))) (针对并列程序段)
  - O(f(m)+g(n))=O(f(m))+O(g(n))
  - O(f(n)*g(m))=O(f(n))*O(g(m)) (针对嵌套程序段)
- 算法中常遇到的两大类O( )函数
  - 多项式阶的时间复杂度：
    $O(1)<O(\log n)<O(n)<O(n\log n)<O(n^2)$
  - 指数阶的时间复杂度： $O(2^n)<O(n!)<O(n^n)$

例3

```
void add ( float x[ ][ ], int m, int n ) {
   for ( int i = 0; i < m; i++ )
    {   sum[i] = 0.0;
        for ( int j = 0; j < n; j++ )    sum[i] += x[i][j];
    }
    for ( i = 0; i < m; i++ )   printf("%f",sum[i]) ;
}
```

问题规模: m*n

原操作: sum[i] += x[i][j]; printf();

基本操作重复执行的次数: f(m) = m*n, g(m)=m

时间复杂度: T(m) = O(f(m)+g(m))=O(m*n)

O(max(m*n, m))

例4

```
for(i=2;i<=n;++i)
    for(j=2;j<=i-1;++j){
        ++x;
        a[i][j]=x;
    }
```

问题规模: n

原操作: a[i][j]=x;

基本操作重复执行的次数: 难以精确计算(1+2+…+n-2)

时间复杂度: $T(n) = O(n^2)$

算法和算法分析：时间复杂度计算的规则

- 时间复杂度计算的规则
  - 计算时间复杂度所选取的原操作应是其重复执行次数和算法的执行时间成正比的原操作，多数情况下它是最深层循环内的语句中的原操作。
  - 有时，算法中基本操作重复执行的次数随问题输入数据集的不同而不同，此时最常用的办法就是讨论算法在最坏情况下的时间复杂度。 ?
  - 算法的时间复杂度只考虑问题规模n的增长率，在难以精确计算基本操作执行次数（或语句频度）时，只需求得它关于n的增长率或阶即可

算法和算法分析：时间复杂度计算的规则

- 时间复杂度计算的规则
  - 有时，算法中基本操作重复执行的次数随问题输入数据集的不同而不同，此时最常用的办法就是讨论算法在最坏情况下的时间复杂度。

【例】顺序查找，在数组a[i]中查找值等于e的元素，返回其所在位置。

```
for (i=0;i< n;i++)
  if (a[i]==e) return i+1;   //找到，则返回是第几个元素
return 0;
```

最好情况：1次

最坏情况：n次

平均时间复杂度：O(n)

算法和算法分析：时间复杂度计算的规则

- 最坏时间复杂度：指在最坏情况下，算法的时间复杂度
- 平均时间复杂度：指在所有可能输入实例在等概率出现的情况下，算法的期望运行时间
- 最好时间复杂度：指在最好的情况下，算法的时间复杂度
- 一般我们总是考虑在最坏情况下的时间复杂度，以保证算法的运行时间不会比它更长

例5

```
void bubble_sort( int a[], int n ){
    for(i=n-1,change=TRUE; i>=1&&change; --i){
        change=FALSE; //判断是否有相邻元素交换
        for(j=0;j<i;++j)
            if(a[j]>a[j+1]){ a[j]<->a[j+1];change=TRUE;}
    }
}
```

问题规模: n

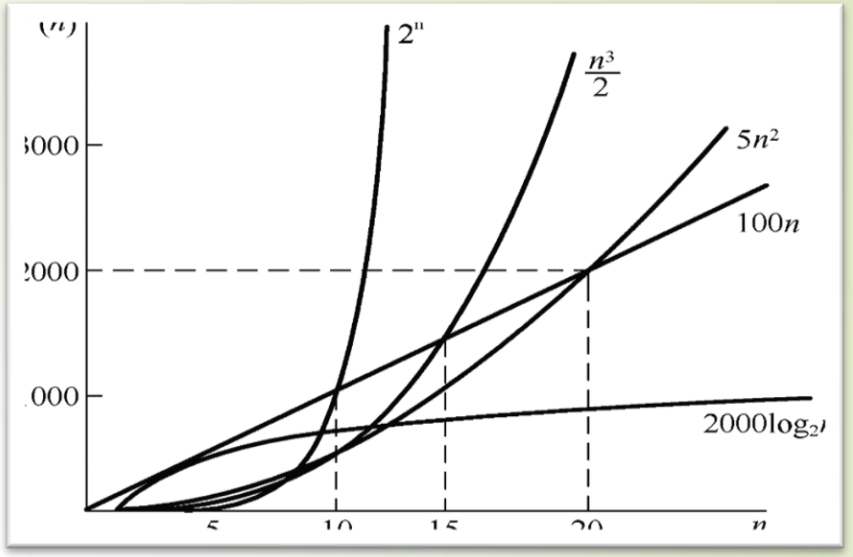
原操作: a[j]<->a[j+1];

基本操作重复执行的次数: 不定

时间复杂度: $T(n) = O(n^2)$

88 时间复杂度的比较

当n很大时，指数时间算法和多项式时间算法在所需时间上非常悬殊



时间复杂度T(n)按数量级递增顺序为：

复杂度低 → 复杂度高

常数阶	对数阶	线性阶	线性对数阶	平方阶	立方阶	……	K次方阶
$O(1)$	$O(\log_2 n)$	**O(n)**	$O(n\log_2 n)$	$O(n^2)$	$O(n^3)$		$O(n^k)$

算法和算法分析：时间复杂度计算的规则

- 时间复杂度计算的规则
  - 一般情况下，我们应尽可能选用多项式阶的算法，而不是指数阶的算法，但有时，在n的较小范围内，多项式阶的算法的性能还不如指数阶的算法。
    - 思考：已知有实现同一功能的2个算法，其时间复杂度分别为$O(2^n)$和$O(n^{10})$，设计算机可连续运算的时间为10^7秒（115天17时46分40秒），每秒可执行基本操作10^5次，则这2个算法在这台计算机上可解问题的规模（即n的范围）各为多少？哪个算法更适宜？

$2^{n1}=10^7*10^5$ → $n_1 \approx 40$

$n_2^{10}=10^7*10^5$ → $n_2 \approx 16$

$n_1 > n_2$，说明在特定条件下，第1个算法可解的问题规模更大，因此更适宜。

算法和算法分析：算法空间效率的度量方法

- 渐进的空间复杂度

算法所需存储空间的量度。记作：S(n)=O(g(n))，其中n为问题的规模，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g(n)的增长率相同。

- 算法的存储量包括:
  - 输入数据所占空间
  - 程序本身所占空间，输入/输出，指令，常数，变量等
  - 额外辅助空间

算法和算法分析：算法空间效率的度量方法

- 渐进的空间复杂度

例：将一维数组a中的*n*个数逆序存放到原数组中。

S(n) = O(1)
原地工作

S(n) = O(n)

【算法1】

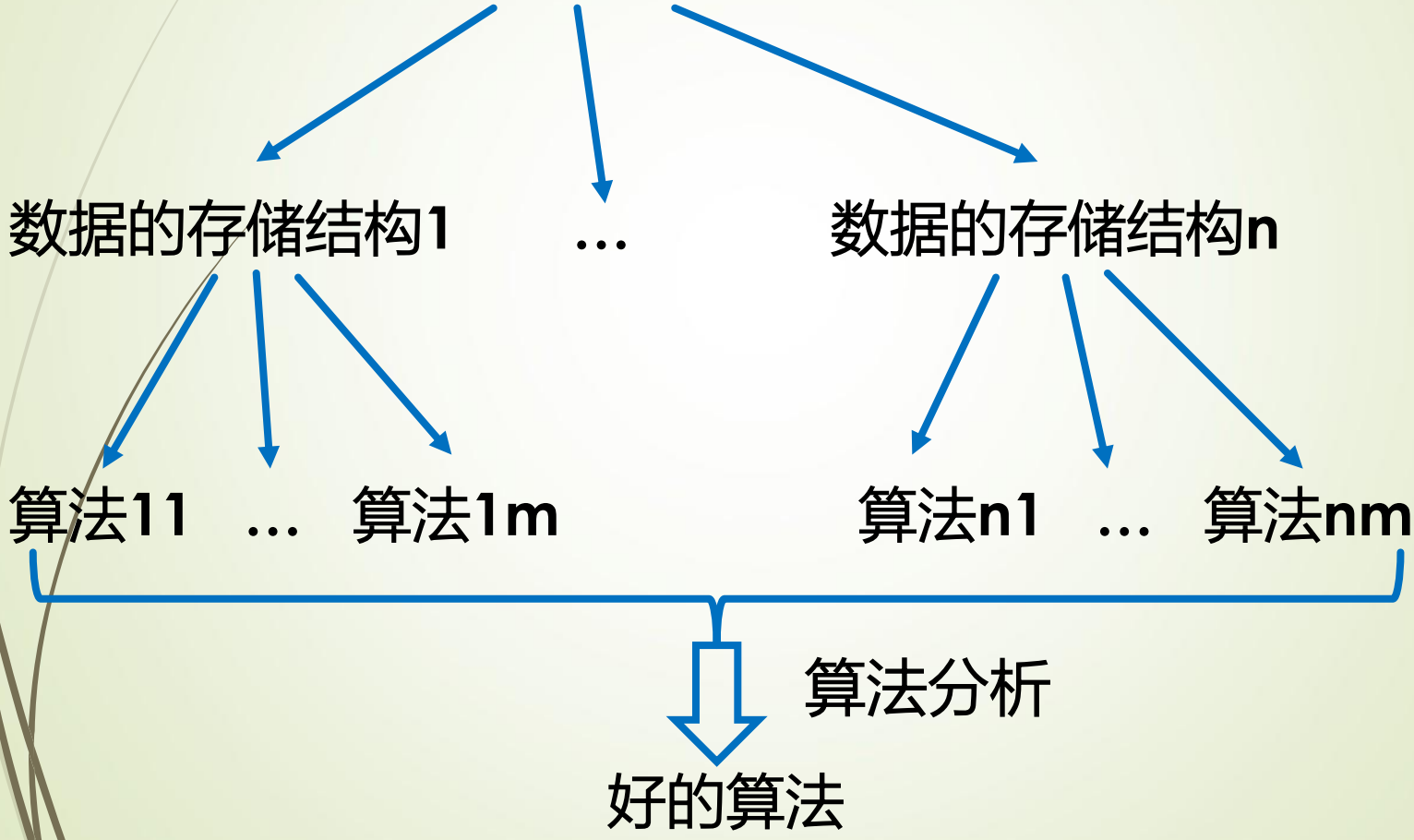
```
for(i=0;i<n/2;i++){
        t=a[i];
        a[i]=a[n-i-1];
        a[n-i-1]=t;
    }
```

【算法2】

```
for(i=0;i<n;i++)
    b[i]=a[n-i-1];
for(i=0;i<n;i++)
    a[i]=b[i];
```

算法和算法分析：设计好算法的过程

抽象数据类型=数据的逻辑结构+抽象运算（运算的功能描述）



第一章总结

本章介绍了与数据、数据结构、数据类型相关的基本概念和术语，并描述了类C语言系统，最后介绍了算法的时间性能分析方法。

重点和难点

重点

掌握基本概念和术语，掌握类C语言系统及求时间复杂度的方法原则

难点

时间复杂度的求法；类C算法转换为C/C++程序的方法

第一章作业

1.8 设n为正整数。确定下列程序段中有@标记的语句的执行频度：

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;
while(x>=(y+1)*(y+1)){
   @  y++;
}
```

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

第一章作业

1.3 设有数据结构（D，R），其中D={d1,d2,d3,d4}, R={r}, r={(d1,d2),(d2,d3),(d3,d4)}。试按图论中图的画法惯例画出其逻辑结构图。

1.9 假设n为2的乘幂，并且n>2，试求下列算法的时间复杂度及变量count的值（以n的函数形式表示）。

```
int  Time(int n){
    count=0;  x=2;
    while(x<n/2){
       x*=2;  count++;
    }
    return count;
```

第一章作业

算法设计题；

1.16 试写一算法，自大至小依次输出顺序读入的3个整数X，Y和Z的值。

1.17 已知k阶斐波那契序列的定义为

$f_0=0,\ f_1=0,\ \ldots,\ f_{k-2}=0,\ f_{k-1}=1;$

$f_n=f_{n-1}+f_{n-2}+\ldots+f_{n-k},\ n=k, k+1,\ldots$

试编写求k阶斐波那契序列的第m项值的函数算法， k和m均以值调用的形式在函数参数表中出现。

类C语言描述的算法和C程序之间的区别

- 变量的声明
- 变参的定义
- 语句的具体写法

算法转变为程序的一个实例

一个算法实例：

```
Status example(ElemType &x, ElemType &y, ElemType a[], int n)
{
    if(x>y)    x<->y;
    b[0..9]=a[0..9];  ….
    return OK;
}
```

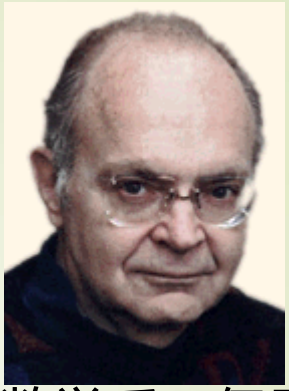
由上述算法改编而成的C

```
typedef int ElemType;
Status example(ElemType
{
    ElemType b[10], temp;
    if  ( *x > *y )    {temp=*
    for( int i=0; i<10; i++) b[
    ……
    return OK;
}
```

C++程序实例：

```
typedef int ElemType;
Status example(ElemType &x,
ElemType &y, ElemType a[], int n){
    ElemType b[10], temp;
    if (x>y)    {temp=y; y=x; x=temp;}
    for( int i=0; i<n; i++) b[i]=a[i];
    ……
    return OK;
}
```

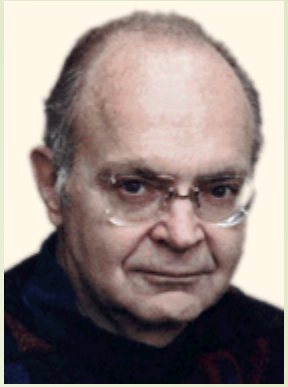
算法大师Donald E. Knuth



1938年出生于Wisconsin。1960年毕业于Case Institute of technology数学系，随即进入大名鼎鼎的加州理工学院数学系，仅用三年时间便取得博士学位，年仅25岁。毕业后留校任教，30岁时加盟斯坦福大学计算机系，任正教授。

31岁开始出版历史性经典巨著：The Art of Computer Programming。他计划共写7卷，然而仅仅出版三卷之后已经震惊世界，使他获得计算机科学界的最高荣誉Turing Award（1974年），时年36岁，是历史上最年轻的图灵奖获得者。后来，此书与牛顿的"自然哲学的数学原理"等一起，被评为"世界历史上最伟大的十种科学著作"之一。

算法大师Donald E. Knuth



作为世界顶级计算机科学家之一，Knuth完成了编译程序、属性文法和运算法则的前沿研究，并编著完成了已在程序设计领域中具有权威标准和参考价值的书目的前三卷。他还用十年时间发明了两个鼎鼎大名的数字排版系统（Tex以及Metafont），它们被广泛地运用于全世界的数学刊物的排版中。他还发明了文件程序设计的两种语言，以及"文章性程式语言"相关的方法论。他很早就提前退休，为的是集中精力把巨著The Art of Computer Programming写完。

在不多的业余时间里，Knuth不仅写小说，还是一个音乐家、作曲家和管风琴设计师。独特的审美感决定了Knuth兴趣广泛、富有多方面造诣的特点，他传奇般的生产力也源于这一点。对于Knuth来说，衡量一个计算机程序是否完整的标准不仅仅在于它是否能够运行，他认为一个计算机程序应该是雅致的、甚至可以说是美的。计算机程序设计应该是一门艺术，一个算法应该像一段音乐，而一个好的程序应该如一部文学作品一般。

算法大师Donald E. Knuth

The Art of Computer Programming

Vol 1: Fundamental Algorithms

全书首先介绍编程的基本概念和技术，然后详细讲解信息结构方面的内容。1968年出版

Vol 2: Seminumerical Algorithms

对半数值算法领域做了全面介绍，分“随机数”和“算术”两章。本卷总结了主要算法范例及这些算法的基本理论。 1969年出版。

Vol 3: Sorting and Searching

不仅对经典计算机排序和查找技术的最全面的介绍，而且还对第一卷中的数据结构处理技术作了进一步的扩充，通盘考虑了大小型数据库和内外存储器。 1973年出版。

算法大师Donald E. Knuth

The Art of Computer Programming

Vol 4: Combinatorial Algorithms

2011年出版。讨论如何生成所有可能性。特别是讨论所有n元组的生成，然后把这些思想扩充到所有排列上。这样一些算法提供了一个自然的导引，借助于此，关于组合数学的许多关键思想都可加以介绍和剖析。在第4卷的这一册和其他册中，通过讨论有关的游戏和数学难题，Knuth阐明一个重要的理论：严肃的程序设计也可以是一种乐趣。

计划中的卷五、卷六和卷七为：

Vol 5: Syntactic Algorithms

Vol 6: The Theory of Context-Free Languages

Vol 7: Compiler Techniques