



# 数据结构与算法

机电工程与自动化学院 L栋301

任卫红 助理教授

[renweihong@hit.edu.cn](mailto:renweihong@hit.edu.cn)

<http://faculty.hitsz.edu.cn/renweihong>

# 课程说明

- 课程编号: COMP2050
- 授课学时: 24学时 (3至10周, 4学时/周) T5404
- 课程分类: 选修
- 考核形式:
  - 作业10% + 实验20% + 期末考试70%
- 作业:
  - 电子版, 按班级提交给助教
  - 文件命名(21级自动化1班\_张三\_2103101XX), 文
- 助教联系方式  
董潜 (自动化1-4班)
  - 电话: 18595373136
  - 1019115845@qq.com罗金国 (自动化5-8班, 通信1班)
  - 电话: 15983350956
  - 594946988@qq.com



.jpg)

扫一扫二维码, 加入群聊



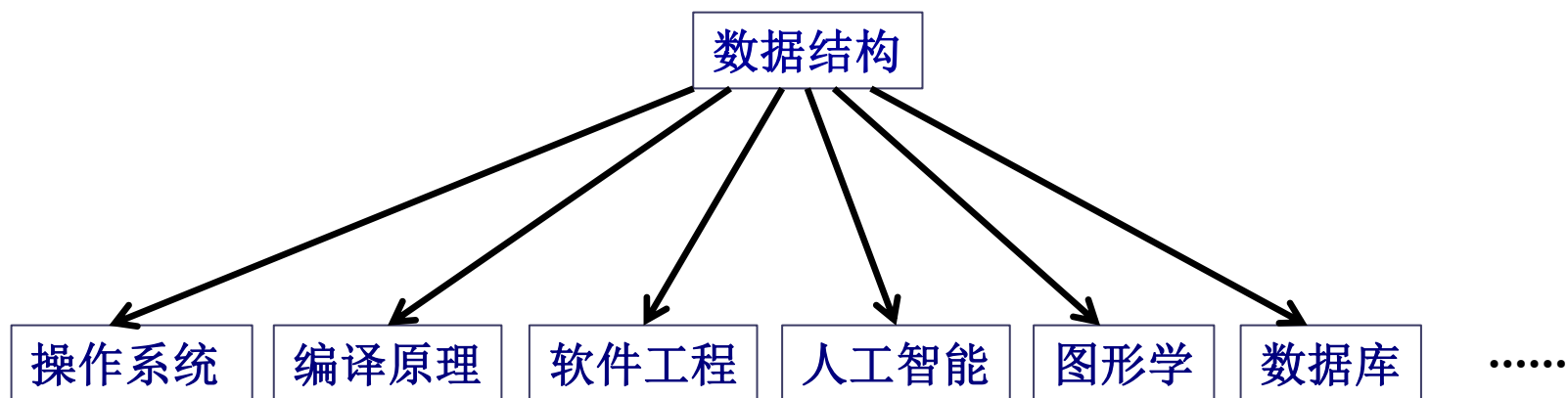
# 教材与参考书

- **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（用面向对象方法与C++语言描述）》。殷人昆编。清华大学出版社。
- 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。



# 数据结构的重要性

- 计算机核心课程
- 许多课程的基础
- 考研、找工作须复习的一门课



# 为什么要学习数据结构

- 数据结构**研究非数值计算**的程序设计问题，是一门探究计算机的操作对象以及他们之间的关系和操作等的科学。
- 数值计算与非数值计算
  - **数值计算**  
方程组求解、求积分、图像处理、音频处理
  - **非数值计算（无法用数学方程式表示）**  
数据表示、数据查询、信息管理系统、遗传算法、模拟退火算法、蚁群算法等

# 学习目标

- ✓ 掌握基本的数据结构（线性表、树、图）
- ✓ 培养算法设计能力、程序设计能力

算法——程序的灵魂

问题求解过程：问题→想法→算法→程序

程序设计研究的层次：算法→方法学→语言→工具

- ✓ 培养算法分析能力

评价算法、改进算法



# 第一章 绪论

机电工程与自动化学院 L栋301

任卫红 助理教授

[renweihong@hit.edu.cn](mailto:renweihong@hit.edu.cn)

<http://faculty.hitsz.edu.cn/renweihong>

# 本章主要内容

- 数据结构的基本概念
- 数据的逻辑结构
- 数据的存储结构
- 抽象数据类型
- 算法定义
- 算法性能分析与度量





Niklaus Wirth（尼克劳斯·维尔特），瑞士计算机科学家，是好几种编程语言的主设计师。其中以Pascal语言最为成功。1984年获得图灵奖。

**Programs = Algorithms + Data Structures**



# 算法 + 数据结构 = 程序

(Algorithms + Data Structures = Programs)

处理问题的  
策略

问题的数  
学模型

为计算机处理  
问题编制的一  
组指令集

# 数据结构所研究的问题

- 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。



# 数据结构例子

## 例1：电话号码查询系统

设有一个电话号码簿，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。数据与数据成简单的一对一的线性关系。

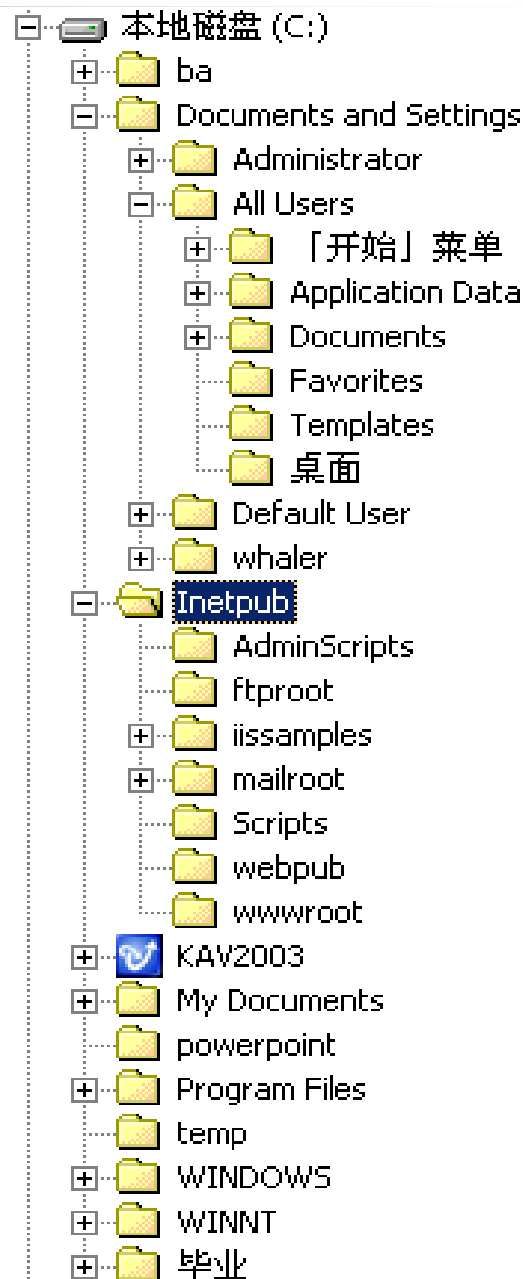
姓名	电话号码
陈海	13612345588
李四锋	13056112345
。 。 。	。 。 。

线性表结构

## 例2：磁盘目录文件系统

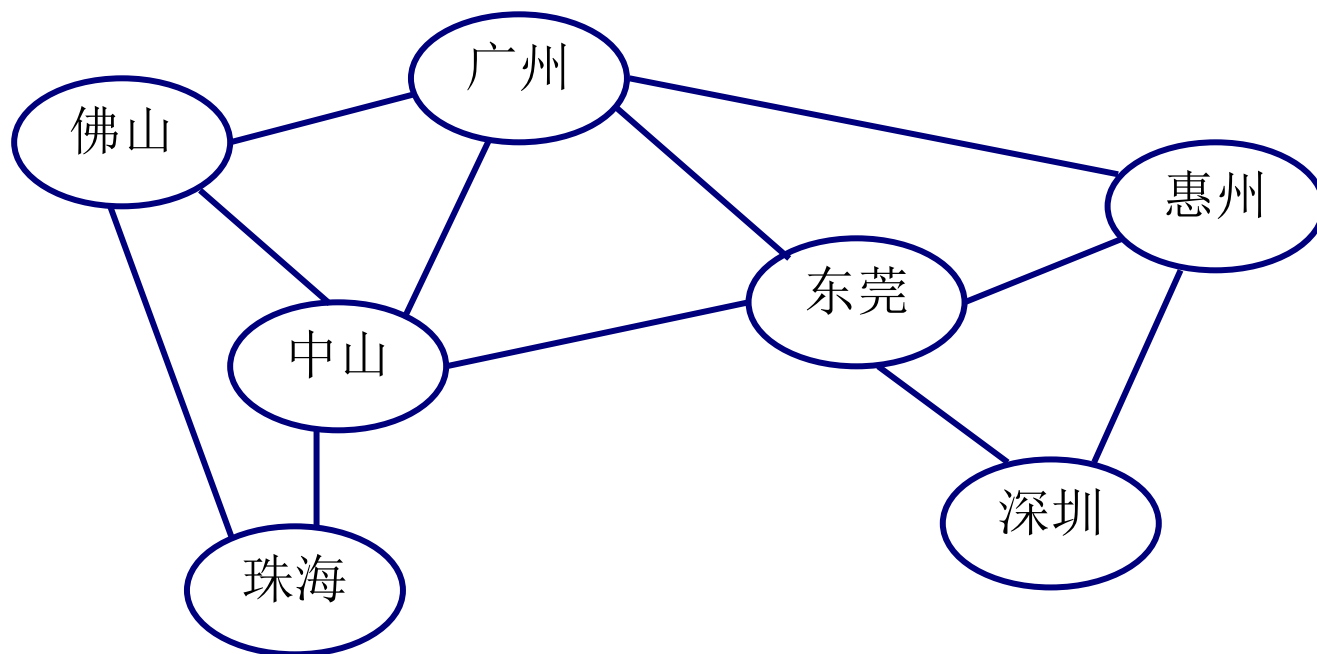
磁盘根目录下有很多子目录及文件，每个子目录里又可以包含多个子目录及文件，但每个子目录只有一个父目录，依此类推：

本问题是一种典型的树型结构问题，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**。



### 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

# 数据结构的基本概念

- 数据 (Data)
- 数据元素 (Data Element)
- 数据项 (Data Item)
- 数据对象 (Data Object)
- 数据结构 (Data Structure)

# 数据结构的基本概念

## ■ 数据

数据是信息的载体，是描述客观事物的数、字符、以及所有能输入到计算机中，被计算机程序识别和处理的**符号的集合**。

- 数值性数据 如：整数、实数等
- 非数值性数据 如：文件、数值记录等



# 数据结构的基本概念

- 数据
- 数据元素
  - 数据的基本单位。在计算机程序中常作为一个整体进行考虑和处理。
  - 如学生组成班级，学生是数据元素，班级是学生集合。

# 数据结构的基本概念

- 数据
- 数据元素

**例1.1** 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
----	-------	------	------	----	----



称之为组合项

# 数据结构的基本概念

- 数据
- 数据元素
- 数据项
  - 具有独立含义的最小标识单位。

姓名	俱乐部名称	出生日期	参加日期	职务	业绩
----	-------	------	------	----	----

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

← 数据元素

← 数据项

# 数据结构的基本概念

- 数据
- 数据元素
- 数据项
- 数据对象

具有相同性质的数据元素的集合。

- 整数数据对象

$$N = \{0, \pm 1, \pm 2, \dots\}$$

- 字母字符数据对象

$$C = \{ 'A', 'B', 'C', \dots 'F' \}$$

# 数据结构的基本概念

- 数据
- 数据元素
- 数据项
- 数据对象
- 数据结构
  - 某一数据元素集合中数据元素及相互之间的关系。
  - 形式化定义： $\text{Data\_Structure} = \{D, S\}$ 
    - $D$  是数据元素的集合；
    - $S$  是数据元素之间关系的有限集合。

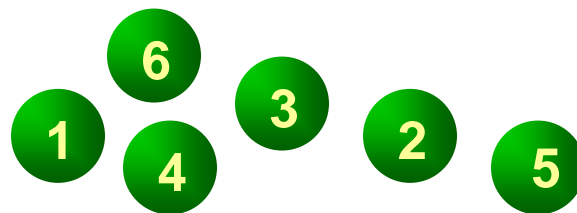
# 数据的逻辑结构

- 数据元素及其之间的抽象关系
  - 集合
  - 线状结构
  - 树状结构
  - 图或网状结构

# 数据的逻辑结构

## ■ 数据元素及其之间的抽象关系

- 集合
- 线状结构
- 树状结构
- 图或网状结构



# 数据的逻辑结构

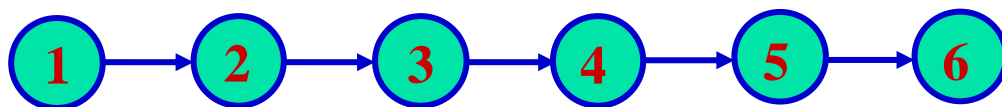
## ■ 数据元素及其之间的抽象关系

- 集合

- 线状结构

- 树状结构

- 图或网状结构



$L = \{K, R\}$

➤  $K = \{1, 2, 3, 4, 5, 6\}$

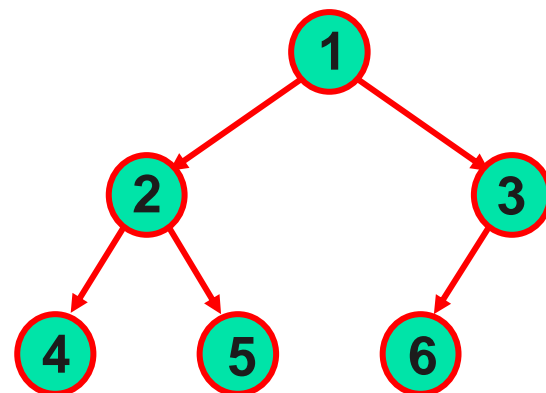
➤  $R = \{\langle 1, 2 \rangle, \langle 2, 3 \rangle, \langle 3, 4 \rangle, \langle 4, 5 \rangle, \langle 5, 6 \rangle\}$



# 数据的逻辑结构

## ■ 数据元素及其之间的抽象关系

- 集合
- 线状结构
- 树状结构
- 图或网状结构



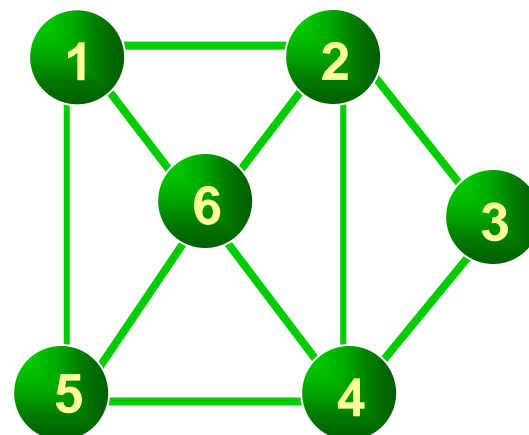
$$T = \{K, R\}$$

- $K = \{1, 2, 3, 4, 5, 6\}$
- $R = \{\langle 1, 2 \rangle, \langle 1, 3 \rangle, \langle 2, 4 \rangle, \langle 2, 5 \rangle, \langle 3, 6 \rangle\}$

# 数据的逻辑结构

## ■ 数据元素及其之间的抽象关系

- 集合
- 线状结构
- 树状结构
- 图或网状结构



$$G = \{K, R\}$$

$$\text{➤ } K = \{1, 2, 3, 4, 5, 6\}$$

$$\text{➤ } R = \{(1, 2), (1, 5), (1, 6), (2, 3), (2, 4), (2, 6), (3, 4), (4, 5), (4, 6), (5, 6)\}$$

# 数据的存储结构

- 数据及其逻辑结构在计算机中的表示，实质上是存储器的分配
  - 顺序存储结构
  - 链接存储结构
  - 索引存储结构
  - 散列存储结构

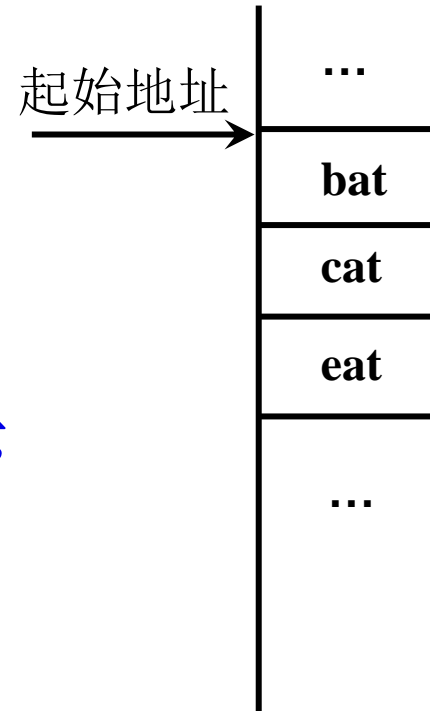
# 数据的存储结构

- 数据及其逻辑结构在计算机中的表示，实质上是存储器的分配

存储 (bat, cat, eat)

- 顺序存储结构
- 链接存储结构
- 索引存储结构
- 散列存储结构

可以用C语言中一维数组表示

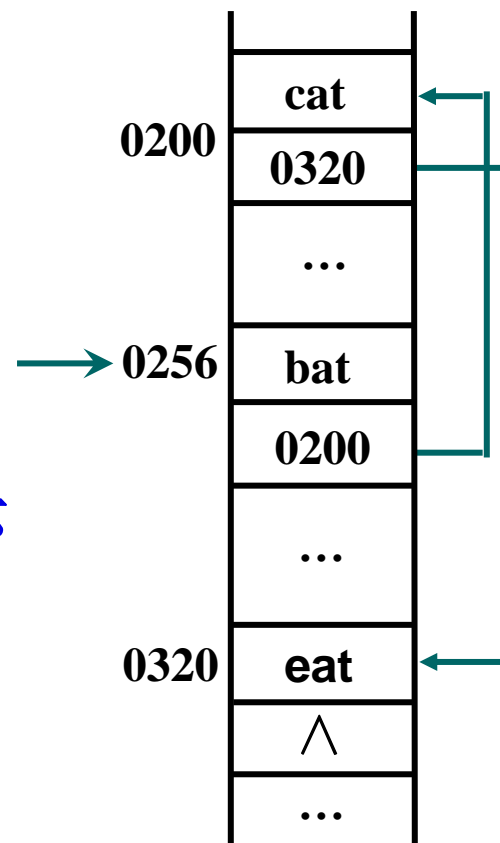


# 数据的存储结构

- 数据及其逻辑结构在计算机中的表示，实质上是存储器的分配  
存储 (bat, cat, eat)

- 顺序存储结构
- 链接存储结构
- 索引存储结构
- 散列存储结构

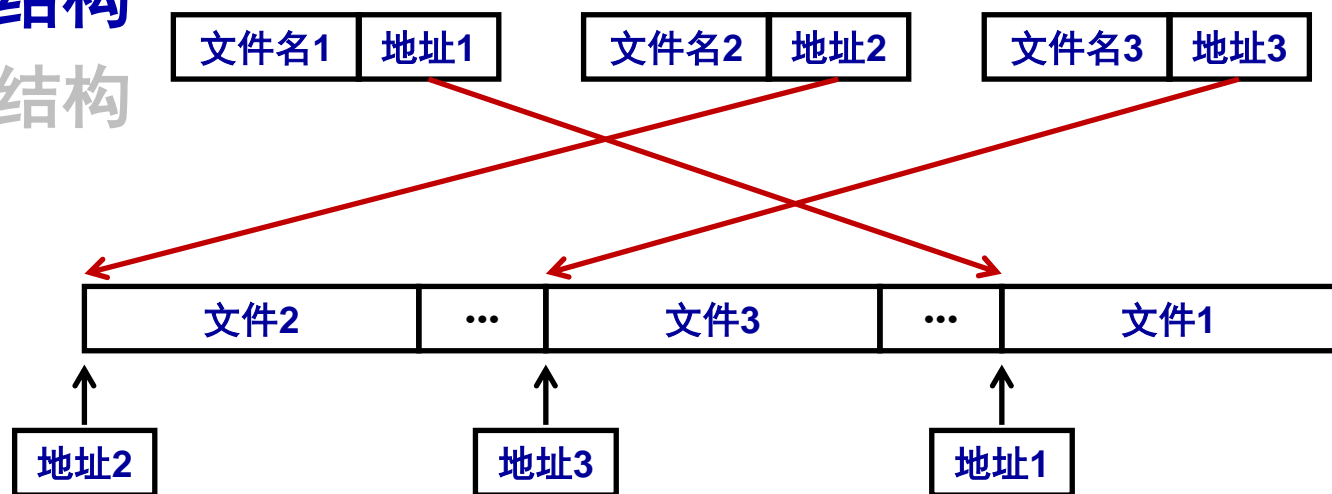
可以用C语言中的指针表示



# 数据的存储结构

## ■ 数据及其逻辑结构在计算机中的表示，实质上是存储器的分配

- 顺序存储结构
- 链接存储结构
- **索引存储结构**
- 散列存储结构



## 数据的存储结构

- 数据及其逻辑结构在计算机中的表示，实质上是存储器的分配

- 顺序存储结构
- 链接存储结构
- 索引存储结构
- 散列存储结构


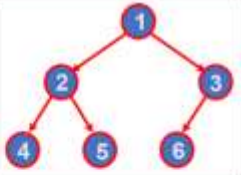

{ 100, 400, 500, 800, 900 }

$\text{hash}(\text{key}) = \text{key} / 100$

100			400	500			800	900
1	2	3	4	5	6	7	8	9

# 数据的存储结构

- 数据的**逻辑结构**和**物理结构**是密切相关的
- 任何一个算法的设计取决于选定的数据结构（逻辑），而算法的实现依赖于采用的存储结构

	逻辑结构	操作	物理结构
线性结构		插入、删除等	顺序、链式
非线性结构		创建、输入、删除、 <b>Huffman</b> 树等	顺序、链式
		创建、最短路径等	链式



# 数据的存储结构

## 补充：STL, Standard Template Library, 标准模板库

- 包含容器、算法、迭代子等
- C++标准库的重要组成部分
- 容器：向量（vector），双端队列（deque），表（list），队列（queue），堆栈（stack），集合（set），多重集合（multiset），映射（map），多重映射（multimap）。

# 抽象数据类型

- 数据类型：一组值的集合以及一组相关的操作
  - 原子类型
    - C语言中int、float、double
    - +、-、\*、/、%、<、>、<=、>=、==、!=、=
  - 结构类型
    - 由若干原子类型或结构类型按某种结构组成的数据类型
    - 结构数据类型可以看作是一种数据结构和定义在其上的一组操作组成的整体
    - 如数组，由若干分量组成，每个分量可以是整数，也可以是数组（如int A[10]）

```
Typedef struct {  
    double data[100];  
    int length;  
} DataList;
```

# 抽象数据类型

- 抽象数据类型：由用户定义，表示问题的数据模型

- 由其他数据类型组成，并包括一组相关操作

```
class Circle { // 对象: 几何圆
    float r; // 圆的半径
public:
    Circle(float r); // 构造函数，创建一个半径为r的对象实例
    float Circumference( ); // 返回该实例的周长
    float Area( ); // 返回该实例的面积
};
```

- 三大特征

- 信息隐藏、数据封装、使用与实现分离

# 抽象数据类型

## ■ 抽象数据类型三大特征

- 信息隐藏：把所有数据和操作分为公有和私有，可减少接口复杂性，从而减少出错机会。
- 数据封装：把数据和操作封装在一起，从语义上更加完整。
- 使用与实现相分离：使用者只能通过接口上的操作来访问数据，一旦将来修改数据结构，可以使得修改局部化，提高系统灵活性。

# 抽象数据类型

- 抽象数据类型（ADT表示）
  - 抽象数据类型可用（D，S，P）三元组表示
  - D是数据对象
  - S是D上的关系集
  - P是对D的基本操作集

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(函数)的定义〉

} ADT 抽象数据类型名

# 抽象数据类型

## ■ 抽象数据类型（ADT表示）

ADT Triplet {

数据对象:  $D = \{e1, e2, e3 \mid e1, e2, e3 \in \text{ElemSet}\}$

数据关系:  $R = \{ \langle e1, e2 \rangle, \langle e2, e3 \rangle \}$

基本操作:  $\text{InitTriplet}(\&T, v1, v2, v3)$

操作结果: 构造三元组T, 元素e1,e2,e3分别赋值  
v1,v2,v3

$\text{Max}(T, \&e)$

初始条件: 三元组T已存在。

操作结果: 用e返回T的3个元素中的最大值。

$\text{Min}(T, \&e)$

初始条件: 三元组T已存在。

操作结果: 用e返回T的3个元素中的最小值。

} #ADT Triplet

大家自行阅读教材1.3，了解抽象数据类型的表示与实现！

# 抽象数据类型

- 抽象数据类型（ADT定义实现）
  - 抽象数据类型可以通过固有数据类型(C语言中已实现的数据类型)来实现
  - 抽象数据类型            类 class
  - 数据对象                数据成员
  - 基本操作                成员函数(方法)

# 算法定义

- 是对特定问题求解步骤的一种描述，是指令的有限序列。
- 算法五大特性
  - 输入：有0个或多个输入
  - 输出：有1个或多个输出
  - 有限性：算法有限步结束，指令有限时间完成
  - 确定性：每条指令有确切含义
  - 可行性：每个运算可由计算机有限条指令完成



# 算法定义

## ■ 算法设计的要求

- 正确性：满足具体问题的需求
- 可读性：便于理解和修改
- 健壮性：当输入数据非法时，也能适当反应
- 效率高：执行时间少
- 空间省：执行中需要的最大存储空间少

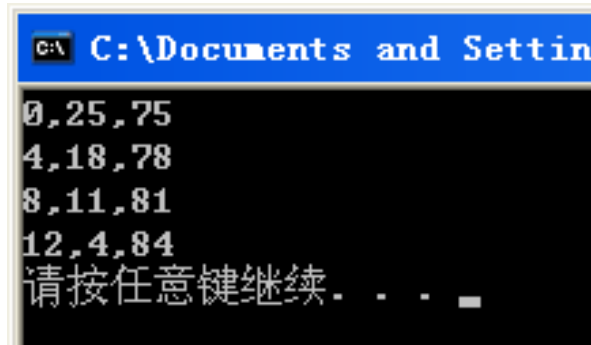
# 算法定义

## ■ 算法举例

- “百钱买百鸡” 问题：公鸡每只5钱，母鸡每只3钱，小鸡3只1钱，100钱买100只鸡，问各种鸡可买多少只？

➤ 令公鸡母鸡小鸡分别为x,y,z只

```
例： for(x=0;x<=100;x++) {  
      for(y=0;y<=100;y++) {  
          for(z=0;z<=100;z++) {  
              if(x+y+z==100 && 5*x+3*y+z/3==100 && z%3==0) {  
                  printf("%d,%d,%d",x,y,z);  
              }  
          }  
      }  
}
```



```
C:\Documents and Settings  
0,25,75  
4,18,78  
8,11,81  
12,4,84  
请按任意键继续...
```

# 算法性能分析与度量

## ■ 算法效率的评价方法：

### □ 事后统计

- 将算法实现，统计其时间和空间开销

### □ 事前分析

- 对算法所消耗时间和空间资源的一种估算方法

## ■ 算法效率的分析

### □ 时间复杂度

### □ 空间复杂度

```
time (&start);  
algorithm();  
time (&stop);  
runTime = stop - start;
```

# 算法性能分析与度量

## ■ 算法的时间复杂度

算法的运行时间 = 每条语句执行时间之和

↓  
每条语句执行次数之和

↓  
基本语句执行次数

↓  
执行次数 × 执行一次的时间

↓  
指令系统、代码质量有关

例: 

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

# 算法性能分析与度量

## ■ 算法的时间复杂度

- 算法的运行时间可表示为基本语句执行次数，它是问题规模的一个函数
- 称这个函数的渐进阶为算法的时间复杂度

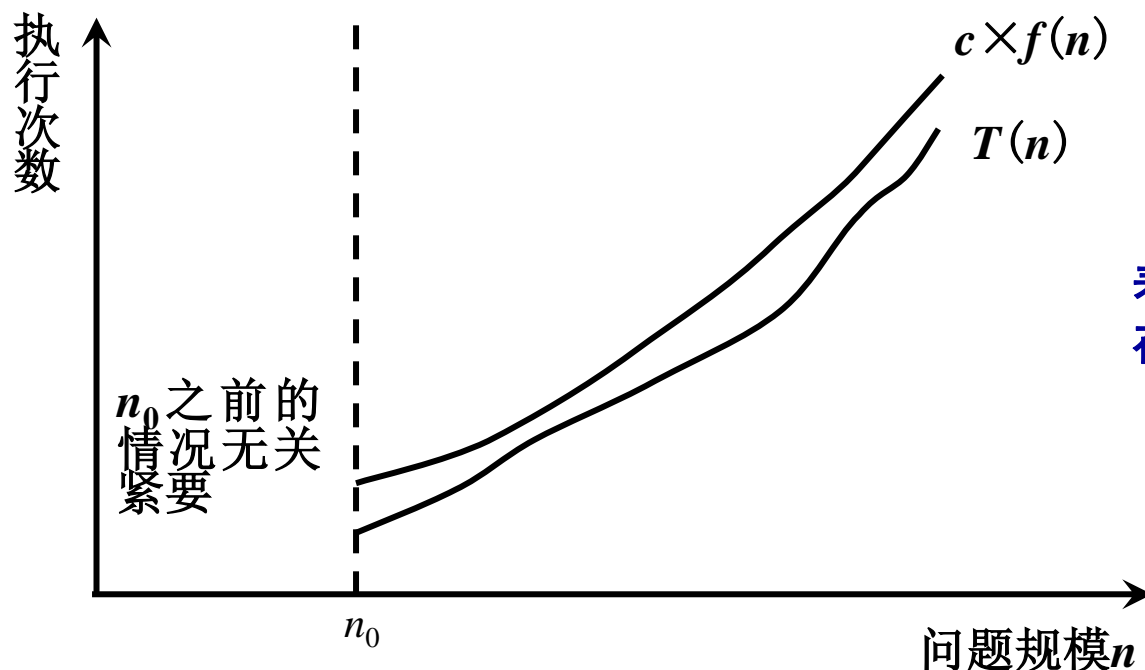
```
例：for(i=0;i<n;i++) {  
    for(j=0;j<n;j++) {  
        c[i][j]=0;  
        for(k=0;k<n;k++) {  
            c[i][j]=c[i][j]+a[i][k]*b[k][j];  
        }  
    }  
}
```

问题规模：  $n$   
基本语句：  $c[i][j]=0$   
 $c[i][j]=c[i][j]+a[i][k]*b[k][j]$   
时间复杂度：  $O(n^3)$

# 算法性能分析与度量

## ■ 算法的时间复杂度

- 大O表示法：若存在两个正的常数 $c$ 和 $n_0$ ，对于任意 $n \geq n_0$ ，都有 $T(n) \leq c \times f(n)$ ，则称 $T(n) = O(f(n))$



表示当问题规模充分大时在渐进意义下的阶

同时有渐进下界和渐进紧界！

# 算法性能分析与度量

## ■ 算法的时间复杂度

- 大O表示法：若存在两个正的常数 $c$ 和 $n_0$ ，对于任意 $n \geq n_0$ ，都有 $T(n) \leq c \times f(n)$ ，则称 $T(n) = O(f(n))$
- 例1：  $T(n) = 3n + 2$
- 当 $n \geq 2$ 时，  $3n + 2 \leq 3n + n = 4n$
- 因此 $T(n) = O(n)$

表示当问题规模充分大时，在渐进意义下的阶！

# 算法性能分析与度量

## ■ 算法的时间复杂度

- 大O表示法：若存在两个正的常数 $c$ 和 $n_0$ ，对于任意 $n \geq n_0$ ，都有 $T(n) \leq c \times f(n)$ ，则称 $T(n) = O(f(n))$
- 例2：  $T(n) = 10n^2 + 4n + 2$
- 当 $n \geq 2$ 时，  $10n^2 + 4n + 2 \leq 10n^2 + 5n$
- 又有当 $n \geq 5$ 时，  $10n^2 + 5n \leq 10n^2 + n^2 = 11n^2$
- 因此 $T(n) = O(n^2)$



# 算法性能分析与度量

## ■ 算法的时间复杂度

- 大O表示法：若存在两个正的常数 $c$ 和 $n_0$ ，对于任意 $n \geq n_0$ ，都有 $T(n) \leq c \times f(n)$ ，则称 $T(n) = O(f(n))$
- 求解 $T(n) = a_m n^m + a_{m-1} n^{m-1} + \cdots + a_2 n^2 + a_1 n + a_0$ ，并给出证明过程

# 算法性能分析与度量

## ■ 算法的时间复杂度

### □ $T(n)=O(f(n))$

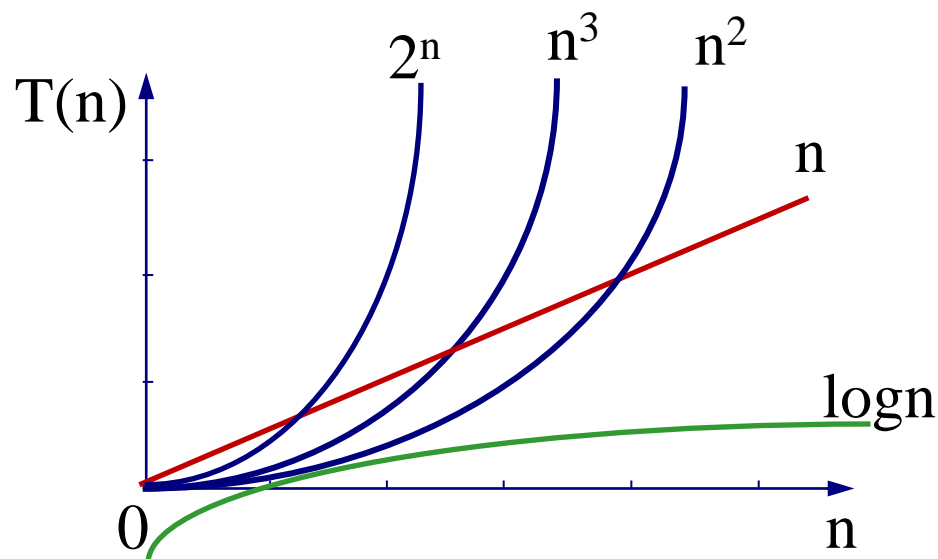
- 若存在两个正的常数 $c$ 和 $n_0$ ，对于任意 $n \geq n_0$ ，都有 $T(n) \leq c \times f(n)$ ，则称 $T(n)=O(f(n))$
- 给出算法复杂度的上界，不可能比 $c \cdot f(n)$ 更大
- 例：  $T(n) = 6 \cdot 2^n + n^2$
- 当 $n \geq 4$ 时，  $6 \cdot 2^n + n^2 \leq 6 \cdot 2^n + 2^n = 7 \cdot 2^n$
- 因此 $T(n)=O(2^n)$

# 算法性能分析与度量

## ■ 算法的时间复杂度

- 常见的时间复杂度

- $O(1) \leq O(\log_2 n) \leq O(n) \leq O(n \log_2 n) \leq O(n^2) \leq O(n^3) \leq \dots \leq O(2^n) \leq O(n!)$



$O(\log_2 n) \neq O(\log_3 n)$   
 $O(2^n) \neq O(3^n)$

# 算法分析应用举例

一般地，常用**最深层循环内**的语句中的原操作的执行频度(重复执行的次数)来表示。

$O(1)$ ：常量阶                       $O(n)$ ：线性阶

$O(\log n)$ ：对数阶               $O(n \log n)$ ：线性对数阶

$O(n^k)$ ：  $k \geq 2$ ， $k$ 次方阶

**例1** `{++x; s=0 ;}`

将x自增看成是基本操作，则语句频度为 1，即时间复杂度为  $O(1)$ 。如果将s=0也看成是基本操作，则语句频度为 2，其时间复杂度仍为  $O(1)$ ，即常量阶。

**例2** `for(i=1; i<=n; ++i)`

`{ ++x; s+=x ; }`

语句频度为：2n，其时间复杂度为： $O(n)$ ，即为线性阶。

**例3** `for(i=1; i<=n; ++i)`

`for(j=1; j<=n; ++j)`

`{ ++x; s+=x ; }`

语句频度为： $2n^2$ ，其时间复杂度为： $O(n^2)$ ，即为平方阶。

**例4** `for(i=2;i<=n;++i)`

`for(j=2;j<=i-1;++j)`

`{++x; a[i][j]=x; }`

语句频度为:  $1+2+3+\dots+n-2=(1+n-2) \times (n-2)/2$

$= (n-1)(n-2) = n^2 - 3n + 2$ 。时间复杂度为  $O(n^2)$ ，即此算法的时间复杂度为平方阶。

以下六种计算算法时间的多项式是最常用的。其关系为：

$$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3)$$

■ 指数时间的关系为：

$$O(2^n) < O(n!) < O(n^n)$$

当 $n$ 取得很大时，指数时间算法和多项式时间算法在所需时间上非常悬殊。因此，只要有人能将现有指数时间算法中的任何一个算法化简为多项式时间算法，那就取得了一个伟大的成就。

■ 有的情况下，算法中基本操作重复执行的次数还随问题的输入数据集不同而不同。

## 例1:

```
void func( int n)
/* n是一个正整数 */
{
    int i = 1;
    while (i<=n) i*=2;
}
```

嵌套的最深层语句是  $i*=2$ ；其频度由条件( $i \leq n$ ) 决定，显然，时间复杂度  $O(\log_2 n)$ 。



## 例2：素数的判断算法。

```
void prime( int n)
/* n是一个正整数 */
{ int i=2 ;
  while ( (n% i)!=0 && i < sqrt(n) ) i++ ;
  if (i>sqrt(n) )
    printf(“%d 是一个素数\n” , n) ;
  else
    printf(“%d 不是一个素数\n” , n) ;
}
```

嵌套的最深层语句是 $i++$ ；其频度由条件 $(n\% i) \neq 0 \ \&\& \ i < \sqrt{n}$ 决定，显然 $i < \sqrt{n}$ ，时间复杂度 $O(n^{1/2})$ 。

# 算法性能分析与度量

## ■ 算法的空间复杂度

- 指算法在执行过程中所需最大存储空间
- 空间复杂性的渐进分析

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

一维数组 $a[n]$ ：空间复杂度 $O(n)$

二维数组 $a[n][m]$ ：空间复杂度 $O(n*m)$

# 算法性能分析与度量

算法的存储量包括:

1. 程序本身所占空间
2. 输入数据所占空间
3. 辅助变量所占空间

与算法无关

若所需额外空间相对于输入数据量来说是常数, 则称此算法为**原地工作**。否则, 按最坏情况分析

# 作业

- 思考下列程序的时间复杂度。

```
1、 for (i=1;i<=n;i++)  
    for (j=1;j<=m;j++)  
        A[i,j]=0;
```

```
2、 i=0;s=0;  
while(s<n)  
    { i++;s=s+i;}
```

# 作业

```
3、 m=0;  
   while(n>=m*m)  
       m++;
```

```
4、 int rec(int n)  
   {  
       if (n==1) return 1;  
       else return (n*rec(n-1));  
   }
```

# 作业

```
5、 i=1;j=0;  
   while(i+j<=n)  
       if (i>j) j++;  
       else i++;
```

```
6.1、 x=91;y=100;  
      while(y>0)  
          if (x>10) y--;
```

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

分析6.1,6.2两题中执行  
频度最大的语句及其  
频度

# 作业

- 7、求下列程序的时间复杂度及语句K++的执行频度。

```
k=0;  
for(i=1;i<=n;i++)  
    for (j=i;j<=n;j++)  
        k++;
```

# 作业

- 8、求下列程序的时间复杂度。

```
for(i=1;i<=n;i=2*i)
{
    cout<<"Hello!"<<endl;
}
```



# 练习

- 理解数据、数据元素、数据对象的概念。
- 理解数据的四种逻辑结构和两种存储结构。
- 掌握算法的概念。
- 掌握算法的五个特性和设计算法的准则。
- 掌握算法时间复杂度和空间复杂度的概念和分析方法。



谢谢