数 据 结 构
Data Structure

代课教师：赵仲孟　博士/教授

联系方式：

Email: xjtuds2013@163.com

数据结构课程目标

- 熟悉基本的数据结构
  - 在什么的情况下使用
  - 设计有效的数据结构和使用它们的算法
- 提高分析算法的能力
  - 算法的效率评价，执行时间，占用的存储空间
- 提高抽象解决问题的能力
- 熟练C语言的使用

- 成绩计算：考试、作业、上机实验

数据结构+算法=程序设计

Data structure + Algorithm = Programming

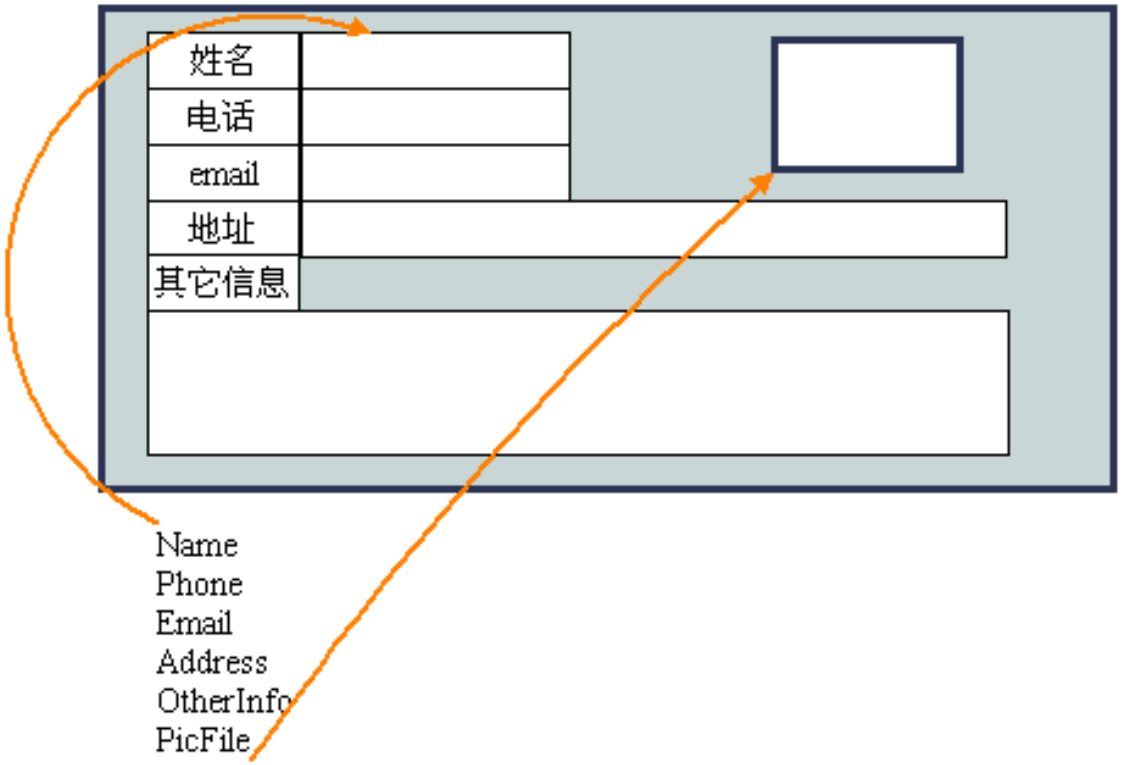
- 设计和实现应用程序需要
  - 数据结构
  - 算法描述
  - 性能分析
  - 性能度量

本章知识点、难点及要求

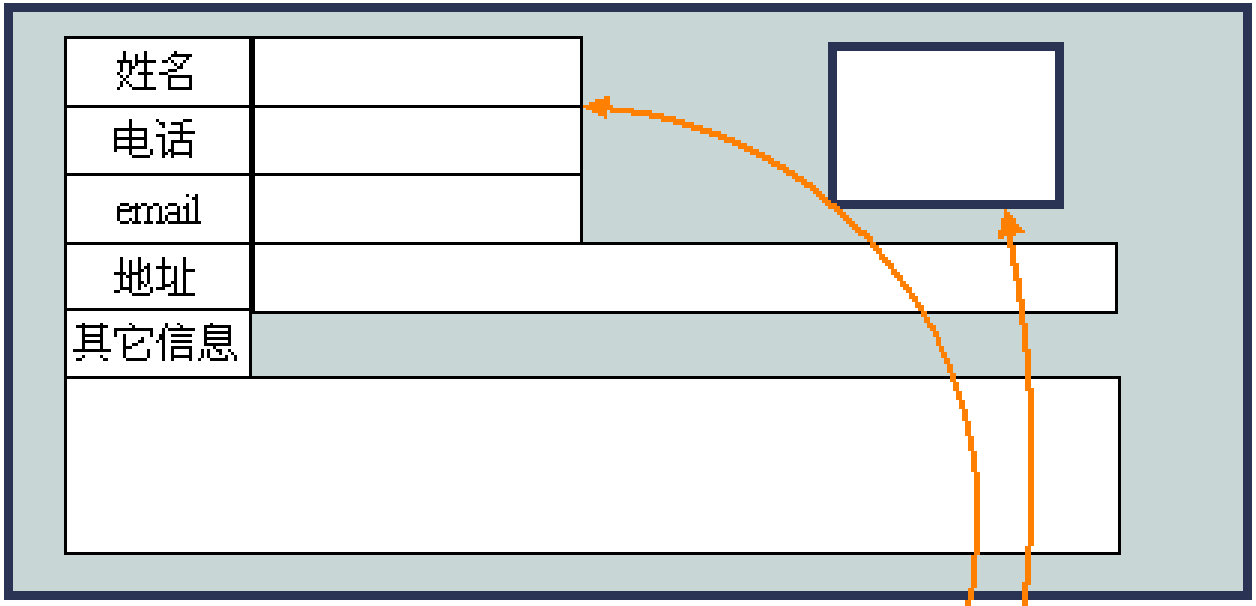
- **知识点：**
  - 数据结构中常用的基本概念和术语
  - 算法描述及分析
- **难点：**
  - 算法复杂度分析
- **要求：**
  - 了解数据结构的逻辑结构和物理结构，算法的基本概念，他们对程序设计的重要性以及关系
  - 掌握算法复杂度的概念和分析方法

1.1、什么是数据结构

- 例1-1-1通讯录
  - 一个人的信息



1、什么是数据结构




```
Typedef Struct Record
{
        Name
        Phone
        Email
        Address
        OtherInfo
        PicFile
}Record
```

record.Name

record.PicFile

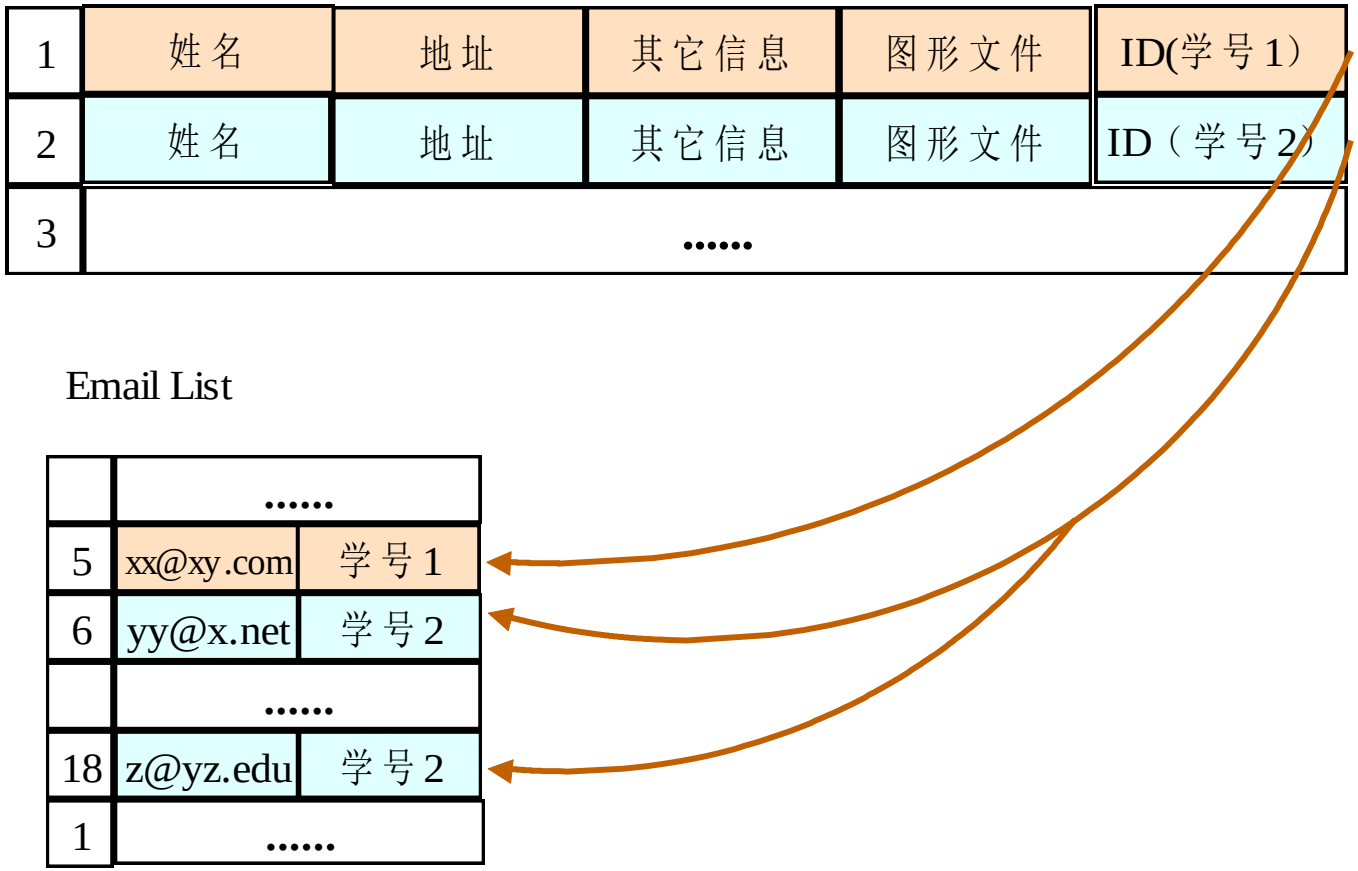
– 十个人的信息

1	姓名	地址	电话	email	其它信息	图形文件
2	姓名	地址	电话	email	其它信息	图形文件
3	.....					

利用数组

–每个人的电话和email不止一个，如何组织这些数据？

```
Typedef Struct Record2
{
    ID → 唯一标示每条信息
    Name
    Address
    OtherInfo
    Picfile
}Record2;
```



- 随着问题的不同，数据的组织和存放也不同，相应的处理有所变化。

1、数据结构概念

A *data structure* is a way to store data and organize data in order to facilitate access and modification.

——《Introduction to Algorithms》

数据结构是存储和组织数据的一种方法，目的是便于数据的访问和处理。

数据结构——任何数据表示和这种数据相关操作的实现

——研究计算机的操作对象以及他们之间关系和操作的学科

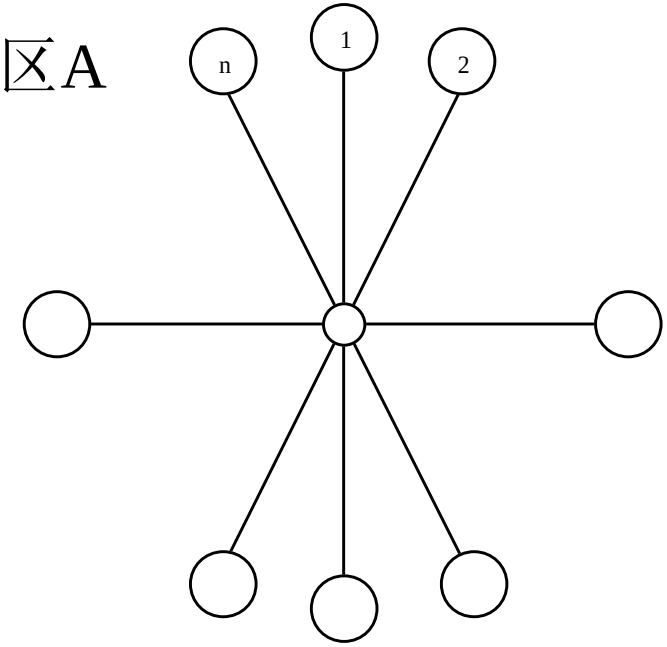
数据、数据的结构/组织、数据的处理/操作

主要学习数据的组织、存储方法和在相应的组织结构上数据处理的实现

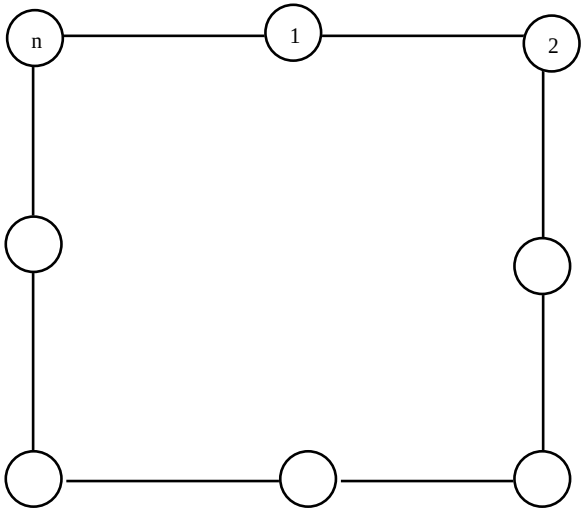
2、结构对操作的影响

例1-1-1:你是户口调查员，要访问社区A和B的所有住家。如果在每个社区，你必须从住户1开始顺次访问所有的住户，并且只访问一次。两个社区的住户数目相同，街道长度相同。

社区A



社区B



户口调查员在社区A经过的总距离>在社区B经过的总距离

结构对操作的影响

- 例1-1-2：在线字典，查询单词
  - 用数组存储，进行顺序查询。利用单词拼写（关键字）匹配。

    算法的执行时间和单词的数量n成正比。如果数据的数目过大，查询速度会很慢。
  - 二叉查找树存储，进行折半查找，每次比较后缩小查找范围。算法的执行时间和码字的数量logn成正比，比较快。
  - Hash表存储，利用关键字可以直接得到存储地址。

1.2、基本概念

- 数据(Data)---能被计算机处理的对象的总称(任何二进制码)
- 数据元素(Data element)---组成数据的基本单位，作为整体处理
- 数据项(Data item)--- 数据的不可分割最小单位，也称字段或域
- 数据对象——性质相同的数据元素的集合
  - 例：图书馆书目查询系统
    - 数据对象：图书馆的所有书目信息
    - 数据元素：一本书的信息

      （书名，作者，分类号，。。。）

      数据项

1.3、数据的结构

数据元素之间的关系：

1、**逻辑结构**——从用户的角度出发，面向应用，从实际中的情况来描述数据元素之间的关系

- 与具体的实现无关

- **线性结构**——元素之间的存在一个序列关系，这种关系是一对一的
  - 等待队列
- **非线性关系**——元素之间不存在序列关系
  - 树、图

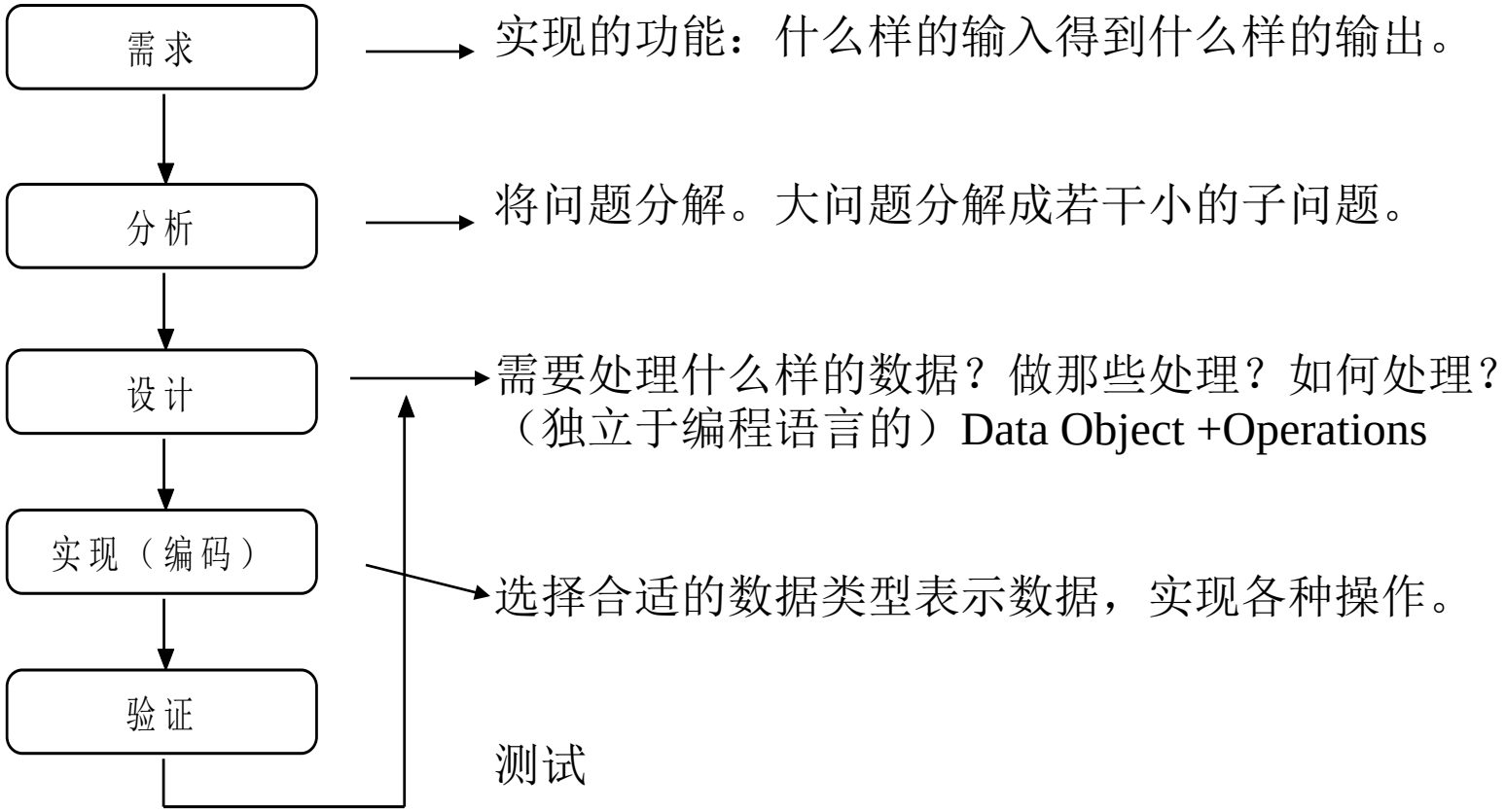
数据元素的逻辑关系反映了数据的特性。

数据的结构

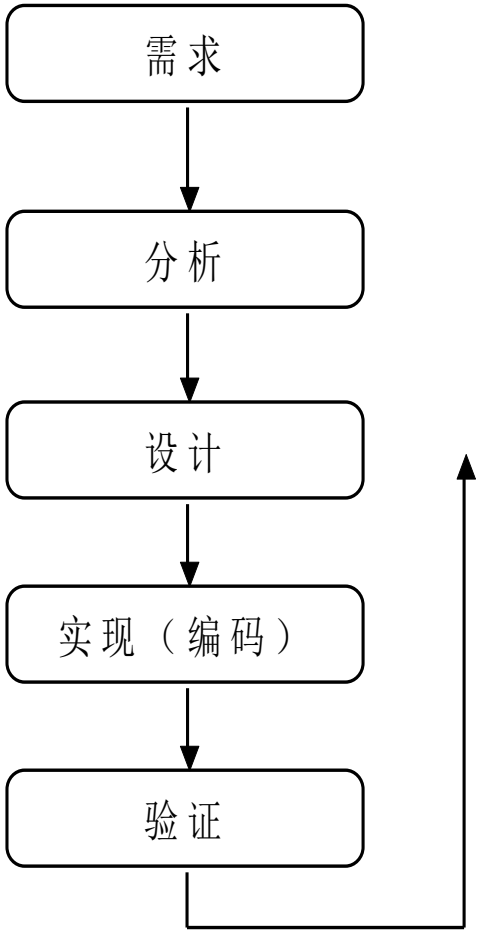
2、**物理结构**——逻辑结构在计算机中的实现。由数据在内存中的存放方式决定

- 顺序存储结构——数据元素按某种顺序存放在地址连续的存储单元中，利用相对位置反映数据间的逻辑关系。
- 链式存储结构——借助指针来表示数据元素间的逻辑关系。

- 学习数据结构是学习语言和编写实际应用之间的自然衔接。
- 分析问题的过程：



程序开发过程



设计：

需要处理什么样的数据？做那些处理？如何处理？（独立于编程语言的）Data Object +Operations。

——〉 ADT

实现：

选择合适的数据结构存储数据，给出实现操作的算法。

- 对于应用而言，如何表示和组织需要处理的数据是很重要的，这直接影响程序的执行效率、简洁性、存储要求等等。

数据和算法是紧密相关的

学习数据结构

- 学习数据结构
  - 按照处理的数据对象的逻辑结构来分类。
    - 表示具有线性结构的数据结构
      - 线性表
      - 队列
      - 栈
    - 表示具有非线性结构的数据结构
      - 树
      - 图
  - 用ADT给出特定数据对象和相关操作的说明，而后选择合适的物理结构进行实现。
    - 线性表+操作===〉顺序表、单链表+函数体或子程序

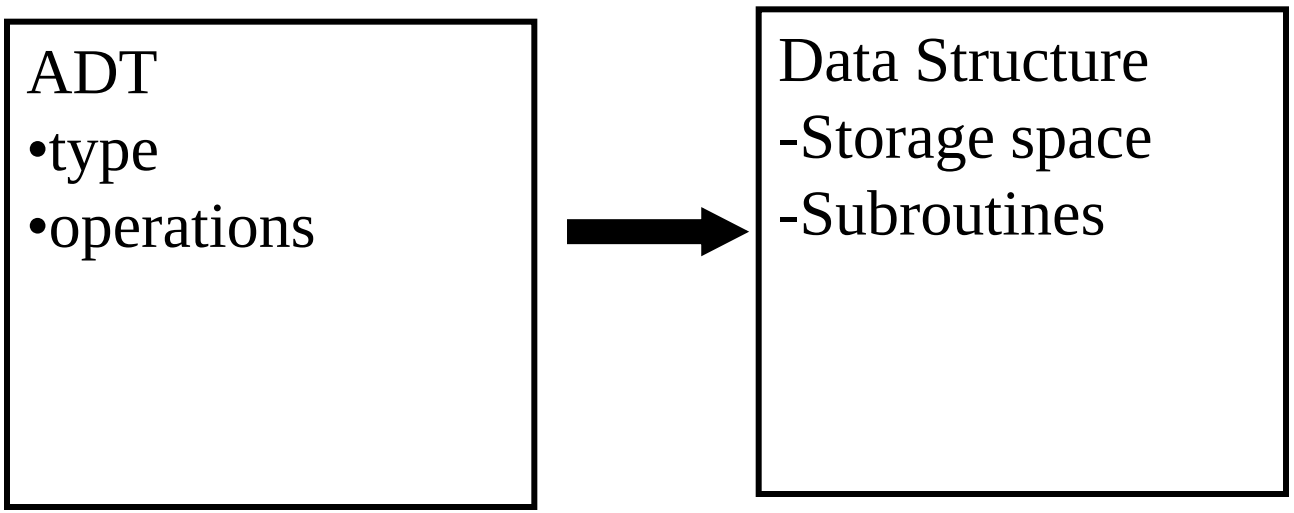
数据的逻辑结构与物理结构

数据对象——逻辑结构——数据+操作　　ADT

|　　|

物理结构 ——存储结构+算法　　DS

Data Type：



1.4、算法

- **算法(Algorithm)：** 完成一个特定功能的一组指令集。
  即，对特定问题求解步骤的一种描述。
- 算法是一个有穷的规则序列，这些规则决定了解决不以特定问题的一系列运算。

- **算法特性：**
  - 有穷性(finiteness)：执行有限步后完成(总是要结束的)；
  - 确定性（definiteness）：每一步骤定义清楚，解释唯一；
  - 可行性（feasibility）：每一个步骤都是可以完成的；
  - 输入（ input ）：有限数量的输入值；
  - 输出（output)：有一个或多个输出结果值；

算法的描述

- 本课程将采用**类C语言**描述算法。即用类C伪码写算法
- 类C语言是标准C语言的简化，其**主要区别如下：**
  - 所有算法都可以如下所示的函数来表示：

    函数类型 函数名（参数表）{

    语句序列；

    }
  - 类C语言的书写形式比C要简单，比如int abc(int x,y,z)；
  - 临时变量可以免定义；
  - 允许宏命令，比如数组分段赋值B[0,9]=A[20,29]；
  - 重要的是表达算法思想，不强调上机通过性；
- **易于理解(省略了与算法无关的语言细节）**
- **易于转换成具体的(C语言)实现**

算法效率与复杂性

- **有效解**：在问题的资源约束（或限制）条件下可以解决该问题的任何求解方案；
- 资源约束：
  - ✓ **空间（Space）**
  - ✓ **时间（Time）**

- 一个解的代价是指该解所消耗的计算机资源量。
- **算法的复杂性**包括时间复杂性（所需运行的时间）和空间复杂性（所占用的存储空间），重点是时间复杂度。

- 算法的设计
  - 易于理解、实现和调试（软件工程）
  - 有效利用资源（内存，CPU等）（数据结构和算法分析）

问题：

如何度量算法的效率？

同一个问题解决的方法不止一个时如何选择？

例1-4-1：N个数中选择第k大的数

- 方法1：起泡排序,而后选出第K大的数。（k*n步）
- 方法2：先读入k个数进行排序，将剩下的数一个一个读入，如果读入的数比已经有的第k个数小，不处理，否则插入有序序列中。(n*n步)
- 问题：哪个更好？如何对一个算法进行评价？

算法例子

简单选择排序:

S假设n个对象存放在一个线性表list[]中，下面算法是通过选择排序方法以**非递减序**重新排列这些对象。

```
for (i=0; i<n; i++)
    {
        检查 list[i] to list[n-1] ;
        假设最小值在 list[min];
        交换 list[i] and list[min];
    }
```

程序与算法

程序(programs):程序是输入到输出的函数或映射关系。

程序可看做是在某种计算机语言下的算法实例化或相应表达。

算法是解决某一特定问题的方法。

程序与算法的区别：

程序---不要求满足算法性质1 (so,operation system)

算法---满足算法性质，可用自然语言、图形、计算机语言描述

1、算法执行时间的分析

- 算法的执行时间是我们主要想要度量的
  - 影响算法执行时间的因素
    - 基本操作的执行时间——与运行和编译环境有关
      - CPU,Bus,Peripheral hardware 的速度
      - 编程语言、编译器产生的可执行代码的效率
    - 基本操作的执行次数
      - 算法处理一定量的输入数据时，所需要的基本操作次数
- 算法复杂度分析：**算法测量和算法分析**
  - 一个算法所需的运行时间通常与所解决问题的规模大小有关；
  - 用n表示问题规模的量，把算法运行所需的时间T表示为n的函数，记作$T(n)$。

评估的例子

需要多少书柜才能放下总量为**100**万页的图书？

评估(**Estimate**)**:**

- pages/inch
- Feet/shelf
- Shelves/bookcase

严格分析规则

1. 我们可以假设一个时间单位 （time unit）.

2. 以下**每一个操作**的执行需要1个时间单位（time **1**）：

   a) 赋值操作（assignment ）

   b) 单个输入/输出操作（single I/O ）

   c) 单个布尔操作、数值比较（ Boolean, numeric）

   d) 单个算术运算（arithmetic）

   e) 函数返回（function return）

   f) 数组下表操作，指针引用（array index， pointer）

严格分析规则(continuous)

3. **选择语句** (if, switch)运行时间=条件判断+选择子句中最大的运行时间。

4. **循环语句**运行时间是在循环次数下，循环体时间+条件检测时间+更新操作时间，此外还有循环初始化时间。

† **总是假设循环执行可能的最大次数。**

5. **函数调用**时间 = 1初始化+参数计算+函数体执行时间

Analysis Example

规则4 and 2a: **time 1** before loop

规则 4 and 2a: **time 1** on 外循环的每一次迭代

规则 2a and 2f: **time 3** on 内循环的每一趟

```
for (i = 0;  i < n-1;   i++)  {
        for (j = 0;  j < i;  j++ ) {
                array[i][j] = 0;
        }
}
```

规则 4, 2c and 2d: **time 3** on 外循环每次迭代中条件检测及更新操作

规则 4, 2c and 2d: **time 2** (内循环每次迭代中条件检测及更新操作)

*这样, 总的时间 **T(n)** 表达如下:*

$$T(n) = 1 + \sum_{i=0}^{n-2} \left(4 + \sum_{j=0}^{i-1} 5 + 1 \right) + 1 = \frac{5}{2} n^2 - \frac{5}{2} n + 2$$

简化分析

通常，我们**假设一个程序步的执行时间是 time 1**。

【定义】一个程序步是指语法或语义有意义的程序段，它的执行时间不依赖于实例化特性。

我们可以**把一个语言句子看做一个程序步。**

Statement	steps	frequency	total steps

for (i = 0; i < n-1; i++) {
for (j = 0; j < i; j++) {
array[i][j] = 0;
}
}
	1
	1
	1
	0
	0
		n
		1+2...+n-1
		0+1...+n-2
		0
		0
			n
			n(n-1)/2
			(n-1)(n-2)/2
			0
			0

total			n^2-n+2

例1-4-2：计算n个数的和

```
int Sum(Data[] )                    执行时间        执行次数
{
        int s = 0 ;                     c1          1
        for (I = 0; I<n ; I++)          c2          n+1
            s+= Data[I];                c3          n
        return s;                       c4          1
 }
```

总执行时间：

$$T(n) = c_1 + c_2(n+1) + c_3 n + c_4$$

- **算法的执行时间和处理的数据的多少n有关。**
- **算法的执行时间和语句的执行时间ci有关。**

例1-4-3：在n个数中查找给定的数值。

```
Int Find(data[], v)                          执行时间        执行次数
{
    for(I=0;I<n;I++)                          c1             ？ 1~n+1
    {       if (data[I] == v)                 c2             ? 0~n
            return I;}                        c3              1/0
    return –1;                                c4             1/0
}
```

- 总执行时间: 和比较的参数v有关系。
  - 如果v=data[0] $T(n)=c_1+c_2+c_3$
  - 如果v=data[k](0<=k<n) $T(n)=c_1(k+1)+c_2(k+1)+c_3$
  - 如果 v = data[n-1] $T(n)=c_1n+c_2n+c_3$
  - 如果 v不在data[]中 $T(n)=c_1(n+1)+c_2n+c_4$

- **算法的执行时间和输入的数据有关**
  - **分情况讨论**

算法执行的时间：

1、与处理的数据集的大小（input size)有关；

反映了处理的问题的规模，也称为“问题规模”。

2、 与算法语句的执行次数有关，以及语句的执行时间有关。

“程序步”/基本操作 的次数，特定的基本操作的执行时间是一个常数。

3、什么样的输入

- **最坏情况（Worst case）** 反映了算法执行时间的上限。
- **最好情况（Best case）**
- **平均情况（Average case）**

如何表示算法的运行时间？

语句的执行次数反映了处理的数据集的大小，所以算法 的执行时间是数据集大小的函数。

运行时间评估的例子

假设硬件处理能力为10^6/s. 一个算法复杂度为$T(n)=2n^2$，在问题规模为n = 10^8时需要执行多少时间？

要执行的总操作次数为：

$T(10^8)=2*(10^8)^2=2*10^{16}$

需要执行的总时间秒数为：

$T(10^8)/10^6$

即：　运行时间 = $2*10^{16}/10^6 = 2*10^{10}$

每天的秒数为 是 86,400秒，这样它就是231,480 天 (634 年)

运行时间评估的例子（续）

当我们采用另一个算法，它的复杂度为：**T(N)=N log N**

在问题规模还是 $N = 10^8$时执行时间会是多少呢？

要执行的总操作次数为：

$$T(10^8)= 10^8 \log(10^8) \approx 2.66*10^9$$

需要执行的总时间秒数为：

$$T(N)/10^6= 2.66*10^3$$

总时间大约是 44.33 秒.

运行时间评估的例子（续）

加快硬件处理速度效果如何呢？

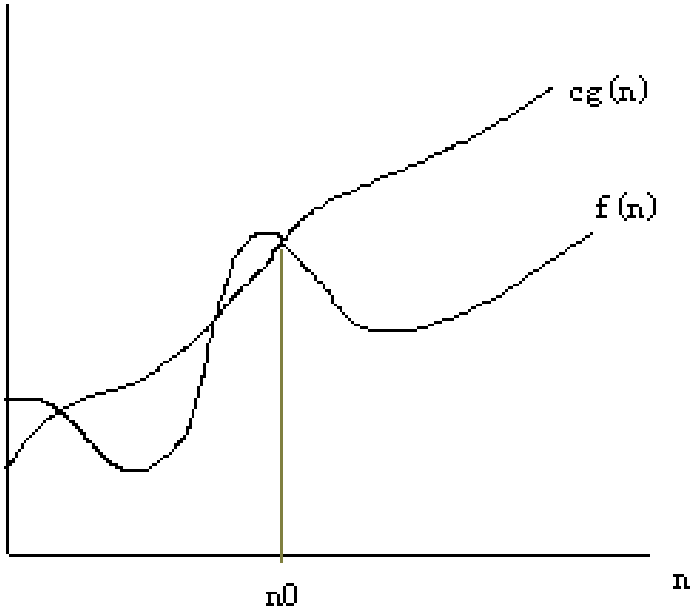
如果我们把应减速度比原来的机器提高**100倍**（前面机器硬件速度是**10^8 /秒**), 那么此时结果会如何？

T(N)	time for $N=10^8$	max N in 1 hour
N(logN)	0.4433 minutes	~10 billion(十亿)
$2N^2$	6.34 years	~4,242,640

函数的增长

- **渐进表示法**
  - O-表示法（big-Oh)
    - f(n) = O(g(n)) :存在正的常数c和n_0，使得当n>=n_0时，有0<=f(n)<=cg(n)

反映了函数f(n)的上限。



例1-4-4：

$$T(n) = c_1 + c_2(n+1) + c_3 n + c_4$$
$$= (c_2 + c_3)n + c_1 + c_2 + c_4$$

如果c=c2+c3+1，
n0=c1+c2+c4，则n>=n0时

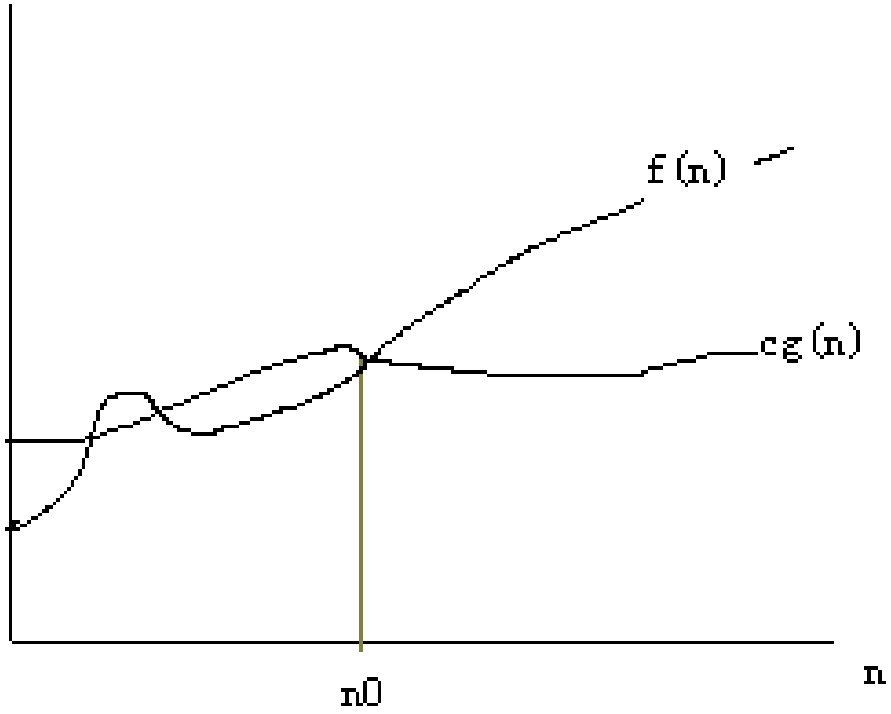
有0<=T(n)<=cn，所以
T(n)=O(n)

表明算法的执行时间是随数据集的增长，其增长速度不超过线性级的增长速度。

– Ω-表示法（big-Omega)

- f(n) = Ω(g(n)) :存在正的常数c和n0，使得当n>=n0时，有0<=cg(n) <=f(n)

反映了函数f(n)的下限。



例1-4-5：

$$T(n) = c_1 + c_2(n+1) + c_3 n + c_4$$

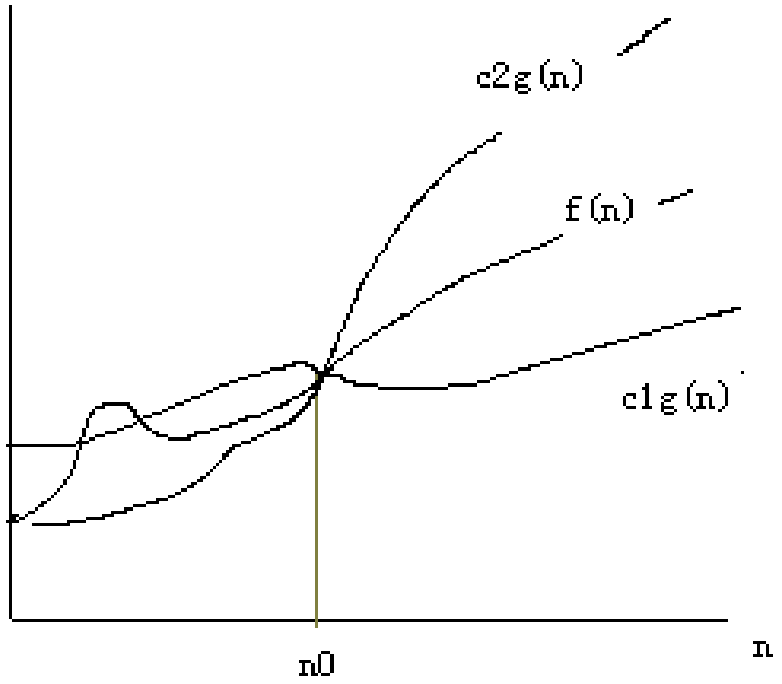
$$= (c_2 + c_3)n + c_1 + c_2 + c_4$$

如果c=c2+c3，n0=0，则n>=n0时

有0<=(c2+c3)n<=T(n)，所以T(n)= Ω(n)

– Θ-表示法（theta)

- f(n) = Θ(g(n)) :存在正的常数c1,c2和n0，使得当n>=n0时，有0<=c1g(n) <=f(n)<=c2g(n).



例1-4-6：

$$T(n) = c_1 + c_2(n+1) + c_3 n + c_4$$
$$= (c_2 + c_3)n + c_1 + c_2 + c_4$$

如果C1=c2+c3,C2=c2+c3+1，n0=c1+c2+c4，则n>=n0时

有C1n<=T(n)<=C2n，所以T(n)= Θ(n)

表明算法的执行时间是随数据集的增长呈线性级增长。

渐进时间复杂度

- 一个算法所需要的执行时间就是该算法所有语句执行时间之和；
- **渐进时间复杂度**：当问题规模n逐渐增大时T(n)的极限情况，也是最常用的复杂度简化表示方法；
- 时间复杂度常用数量级的形式来表示，记作：

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

  其中：大写O为Order(数量级)的头字符，如：$T(n)=O(n^2)$
- 当T(n)为多项式时可取其最高次幂项，系数也可忽略不计；
- 渐进时间复杂度符合**加法规则和乘法规则**；

例1-4-4：

$$T(n)=n^2+100n \Rightarrow T(n)=O(n^2)$$

N_0=100,c=2满足O表示法的定义。

例1-4-8：

$$T(n)=\frac{1}{2}n^2-3n$$

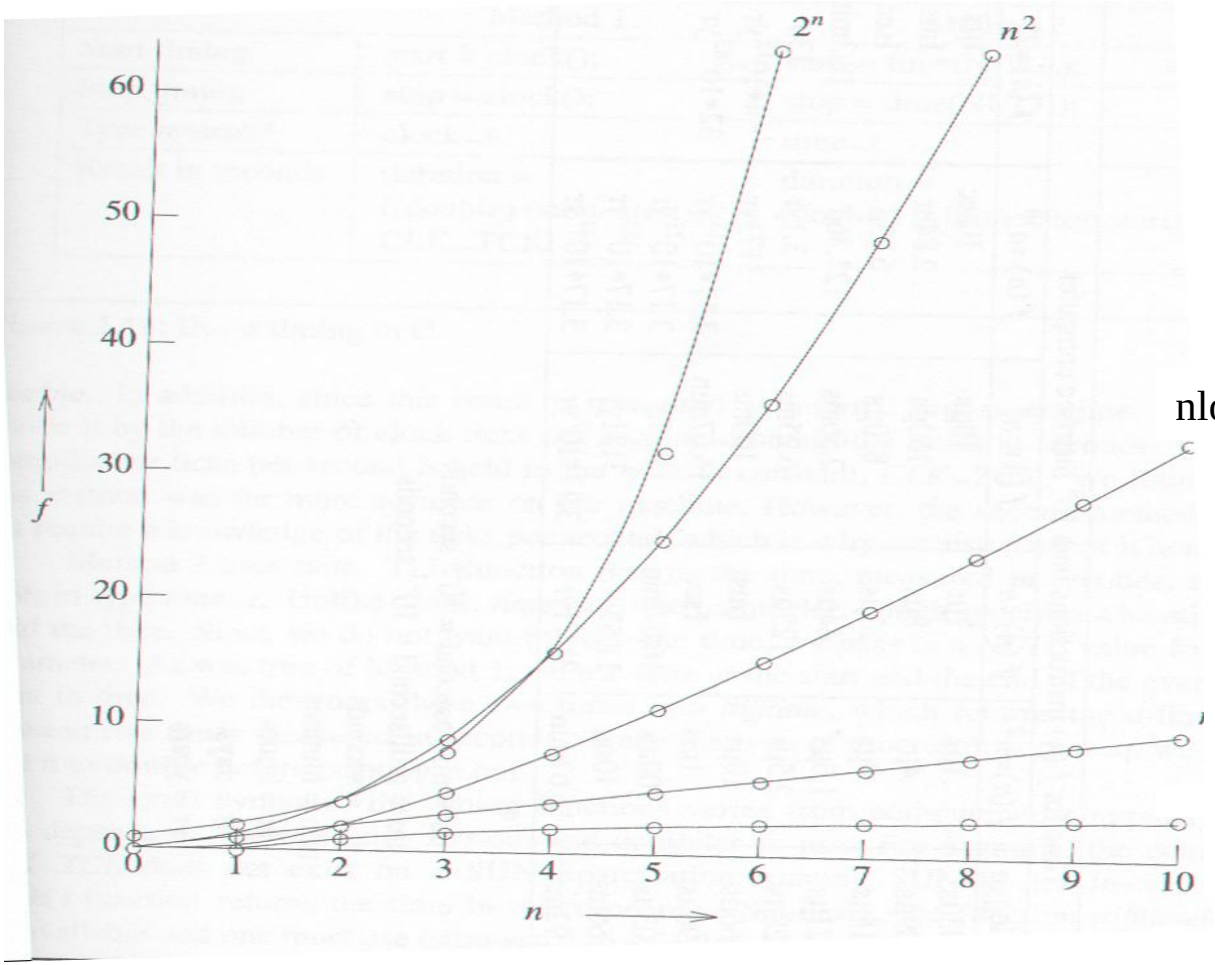
$$C_1n^2 \le \frac{1}{2}n^2-3n \le C_2n^2 \Rightarrow C_1 \le \frac{1}{2}-\frac{3}{n} \le C_2$$

显然：$n \ge 1, \frac{1}{2}-\frac{3}{n} \le \frac{1}{2}, \therefore C_2 \ge \frac{1}{2}$

$$n \ge 7, \frac{1}{2}-\frac{3}{n} \ge \frac{1}{14}, \therefore C_1 \le \frac{1}{14}$$

取$C_1=\frac{1}{14}, C_2=\frac{1}{2}, T(n)=\Theta(n^2)$

常用函数的增长趋势



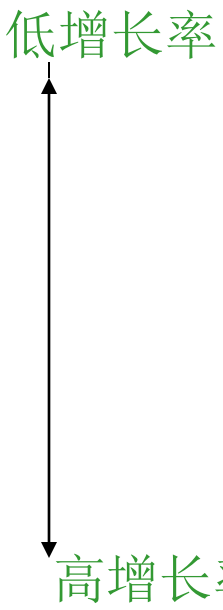
nlogn

n

logn

- 求算法执行时间的渐进表示法的目的：
  - 我们熟悉简单的函数n,logn,n^2, n^3等等这些函数的增长规律，可以利用这些函数来评估算法的增长率。
  - 称为算法的渐进分析（Asymptotic Analysis）
  - 算法的执行时间渐进分析结果称为**渐进时间复杂度**，存储需求的渐进分析结果称为**渐进空间复杂度**。

常见的如下：



constant: O(1)

logarithmic: O(log n)

linear: O(n)

log-linear: O(n log n)

quadratic: $O(n^2)$

exponential: $O(c^n)$ (c is a constant > 1)

算法的执行时间渐进分析举例

例1-4-5：修改矩阵A(nxn)，给每行的数据加上一个常数，但是各行的常数不同。

	执行时间	执行次数
void ChangeMatrix(&A[])		
{		
for(i =0; i <n; i ++)	c1	n+1
read(&constant[i]);	c2	n
for(i =0; i <n; i ++)	c1	n+1
{ for(j=0;j<n;j++)	c1	n*(n+1)
A[i,j]+=constant[i];	c4	n*n
}		
}		

– 总执行时间:

$$T(n) = c_1(n+1) + c_2 n + c_1(n+1) + c_1 n(n+1) + c_4 n^2$$

$$= (c_4 + c_1)n^2 + (3c_1 + c_2)n + 2c_1$$

例1-4-6

```
int fun_a(int N)
{
        if (N<=1)                          c1
            return 100;                    c2
        else
            return 2*fun_a(N-1);           c3+T(n-1)
}
```

算法的执行时间：

$$T(n)=\begin{cases} c1+c2 & n\le 1 \\ c1+c3+T(n-1) & n>1 \end{cases}$$

所以n>1:

$$\begin{aligned} T(n) &= c1+c3+T(n-1) \\ &= c1+c3+(c1+c3+T(n-2)) \\ &= \cdots\cdots \\ &= n(c1+c3)+T(1)=n(c1+c3)+c1+c2 \\ &= O(n) \end{aligned}$$

n<=1: T(n)=O(0)

2、算法的占用空间的分析

算法的存储空间：

- 固定部分——代码，常数，与输入输出无关的变量定长结构等等
- 可变部分——处理数据的存储，与问题规模有关。不仅包括真正数据的存储，还包括附加信息的存储，即一些结构开销。
  - 例：链式存储结构中，指针占用的空间就属于结构性开销。

关心：**可变部分**

- 例1-4-7 求两个整数的和
- Int sum(int a, int b)
- {

```
    Return a+b;
}
```

算法占用的空间（可变部分）多少

S = 0 ，算法的工作量是固定的，不存在执行时随处理问题的不同而变动的问题。

例1-4-8 计算n个数的和

```
int Sum(Data[] )
{       int s = 0 ;
        for (I = 0; I<n ; I++)
          s+= Data[I];
        return s;    }
```

算法占用的空间（可变部分）多少

S(n) = n*sizeof(int) = cn = O(n)

用渐进表示法表示算法的空间消耗和问题规模之间的关系。

- 答疑信箱
  - xjtuds2013@163.com
  - 格式
    - 标题：第？章问题
    - 请将问题论述清楚

谢谢！