数据结构
第一章　绪　论

主讲：陈锦秀

厦门大学信息学院计算机系

- 助教：
  - 叶晗
    - 电话： **15798029572**
    - 邮箱： **yehan@stu.xmu.edu.cn**
  - 宋海昕
    - 电话： **18030212576**
    - 邮箱： **839518063@qq.com**
- **课程中心course.xmu.edu.cn，**
  - 课程号：130130010193
- 实验教室(双周)：4号楼B402

课程考核：

- 期末考50%
- 平时成绩50%（平时作业&实验&实验测试&出勤）

参考书籍

- 计算机程序设计技巧(The Art of Computer Programming) (美)克努特(D.E. Knuth)著
- 数据结构，许卓群等著，北京：高等教育出版社
- 数据结构与算法分析：C++描述 （第三版）(美) (Mark Allen Weiss) 著，张怀勇等译，北京：人民邮电出版社

第一章　绪　论

- 计算机是一门研究用计算机进行信息表示和处理的科学。
- 而信息的表示和处理又直接关系到处理信息的程序的效率。
  - 随着计算机的普及，信息量的增加，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，为了编写出一个“好”的程序，必须分析待处理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！

1.1 什么是数据结构

- 众所周知，计算机的程序是对信息进行加工处理。在大多数情况下，这些信息并不是没有组织，信息（数据）之间往往具有重要的结构关系，这就是数据结构的内容。
- 那么，什么是数据结构呢？先看以下几个例子。

1.1 什么是数据结构

- 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：

    $(a_1, b_1)(a_2, b_2)\ldots(a_n, b_n)$

    其中a_i，b_i(i=1，2…n) 分别表示某人的名字和对应的电话号码。
  - 要求设计一个算法，当给定任何一个人的名字时，该算法能够打印出此人的电话号码，如果该电话簿中根本就没有这个人，则该算法也能够报告没有这个人的标志。
- 算法的设计：依赖于计算机如何存储人的名字和对应的电话号码，或者说依赖于名字和其电话号码的结构。

例1、电话号码查询系统

- 数据的结构，将直接影响算法的选择和效率。
- 上述的问题是一种数据结构问题。可将名字和对应的电话号码设计成向量。
  - 假定名字和其电话号码逻辑上已安排成n元向量的形式，它的每个元素是一个数对(a_i，b_i)， $1 \le i \le n$
  - 在具体实现中，可以使用记录来表示每个向量，那么整个电话薄就对应一个记录数组。
- 数据结构还要提供每种结构类型所定义的各种运算的算法，比如这里要求的查找算法。

1.1 什么是数据结构

线性的数据结构

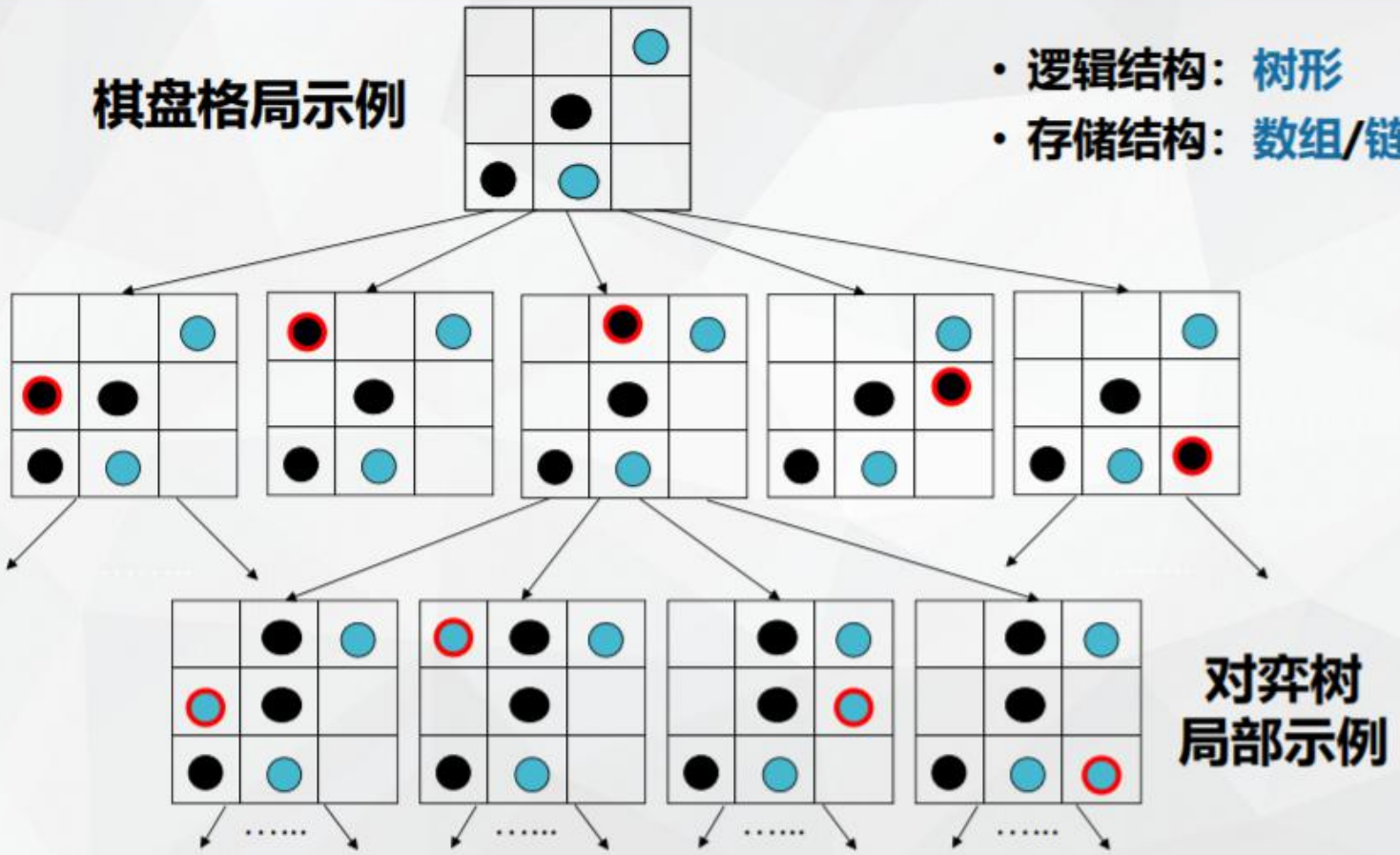
- 例2、图书馆的书目检索系统自动化问题
- 例3、计算机和人对弈问题

树的数据结构

- 例4、货郎担问题(旅行商问题)：给定n个城市，任意两个城市间有路相连，一个货郎从一个城市出发，不重复的遍历所有的城市并回到起点，求一条路程最短的路径。

图的数据结构

实例：井字棋对弈



实例：田径赛的时间安排

【思路】

1、每个项目表示成一个节点

2、同一选手所选中的项目中两两有边相连；

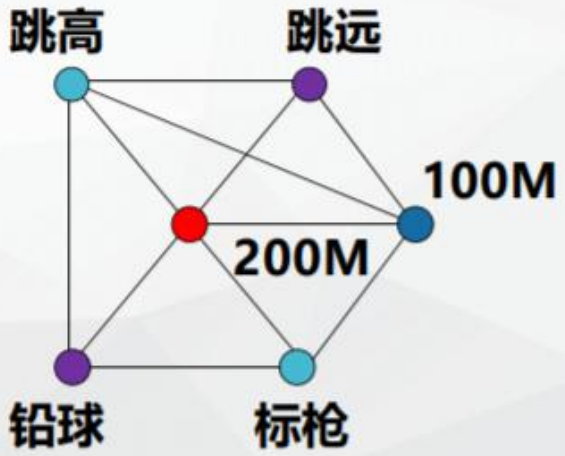
3、任一两个有边相连的顶点颜色（时间）不能相同。

类似问题：

多叉路口交通灯管理

四色地图问题

姓名	项目1	项目2	项目3
丁1	跳高	跳远	100M
马2	标枪	铅球	
张3	标枪	100M	200M
李4	铅球	200M	跳高
王5	跳远	200M	



1.1 什么是数据结构

- 通过以上几例可以直接地认为：数据结构就是研究数据的逻辑结构和物理结构，以及数据之间的相互关系，并在其上设计相关的算法。

1.2 基本概念和术语

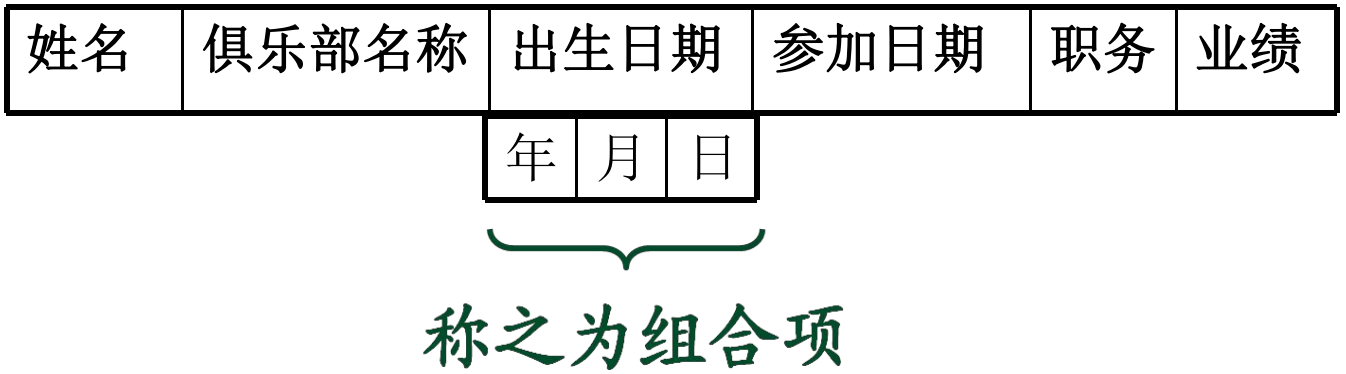
- 数据(Data):是对信息的一种符号表示。在计算机科学中是指所有能输入到计算机中并被计算机程序处理的符号的总称。
- 数据元素(Data Element):是数据（集合）中的一个“个体”，在计算机程序中通常作为一个整体进行考虑和处理。

——是数据结构中讨论的基本单位

1.2 基本概念和术语

- 一个数据元素可由若干个数据项(Data Item)组成。数据项是数据的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



1.2 基本概念和术语

- 数据对象(Data Object)：是性质相同的数据元素的集合。是数据的一个子集。
- 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。

——带结构的数据元素的集合

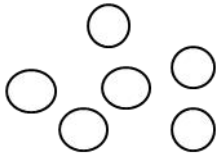
现实中的具体问题		人脑中的数学模型		计算机中的实现
实体	⟺	数据元素	⟺	元素/结点(位串)
属性	⟺	数据项	⟺	数据域
实体集	⟺	数据对象		

1.2 基本概念和术语

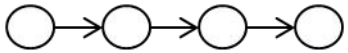
- 数据结构包括逻辑结构和物理结构
- 数据之间的相互关系称为逻辑结构。通常分为四类基本结构：
  - 集合：结构中的数据元素除了同属于一种类型外，别无其它关系。
  - 线性结构：结构中的数据元素之间存在一对一的关系。
  - 树型结构：结构中的数据元素之间存在一对多的关系。
  - 图状结构或网状结构：结构中的数据元素之间存在多对多的关系。

1.2 基本概念和术语

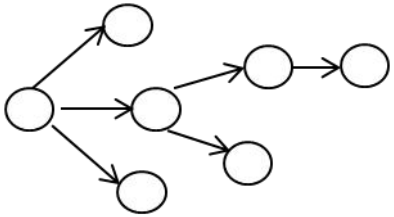
- 集合：



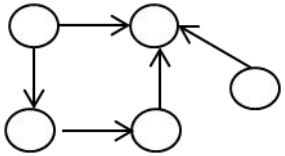
- 线性结构：



- 树型结构：



- 图状结构或网状结构：



1.2 基本概念和术语

- 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：复数的数据结构定义如下：

  Complex=(C，R)

  其中：**C**是含两个实数的集合｛**C1**，**C2**｝，分别表示复数的实部和虚部。**R={P}**，**P**是定义在集合上的一种关系〈**C1**，**C2**〉。

1.2 基本概念和术语

逻辑结构在存储器中的映像：数据元素映像和关系的映像

- 数据结构在计算机中的表示称为数据的物理结构，又称为存储结构。
- 数据元素的映象方法：用二进制位(bit)的位串表示数据元素

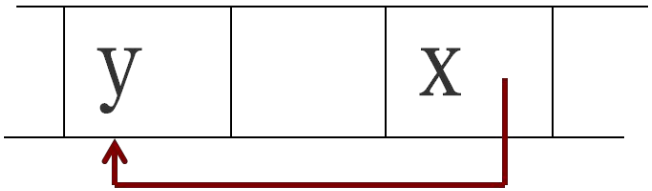
  (321)10 = (501)8 = (101000001)2

  A = (101)8 = (001000001)2
- 数据元素之间的关系在计算机中有两种不同的表示方法：顺序表示和非顺序表示

  ——由此得出两种不同的存储结构：顺序存储结构和链式存储结构

1.2 基本概念和术语

- 顺序存储结构：用数据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。
- 链式存储结构：在每一个数据元素中增加一个存放地址的指针(pointer)，用此指针来表示数据元素之间的逻辑关系。



1.2 基本概念和术语

- 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。

- 例如：以三个带有次序关系的整数表示一个长整数时，可利用 C 语言中提供的整数数组类型。

  定义长整数为:

  typedef int Long_int [3]

1.2 基本概念和术语

- 数据类型：在一种程序设计语言中，变量所具有的数据种类。
- 例1： 在FORTRAN语言中，变量的数据类型有整型、实型、和复数型
- 例2：在C语言中
  - 数据类型：基本类型和构造类型
  - 基本类型：整型、浮点型、字符型、枚举型、指针、空类型
  - 构造类型：数组、结构、联合、自定义

1.2 基本概念和术语

- 抽象数据类型(Abstract Data Type 简称ADT)：一个数学模型以及定义在该模型上的一组操作。

  ——抽象数据类型的定义仅取决于它的一组逻辑特性，而与其在计算机内部如何表示和实现无关。

- 抽象数据类型实际上就是对数据结构的表示以及在此结构上的一组算法。用三元组描述如下：

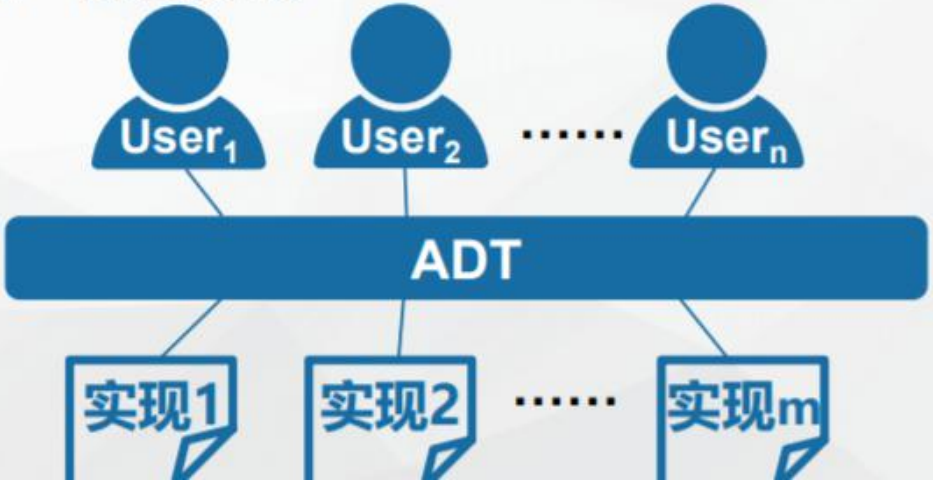
  （D，S，P）

  D是数据对象，S是D上的关系集，P是对D的基本操作集，见例P9

“抽象”的意义在于数据类型的数学抽象特性。例如：各个计算机都拥有的“整数”类型是一个抽象数据类型，尽管在不同处理器上实现方法不同，但数学特性相同。

抽象数据类型（Abstract Data Type，简称ADT）：指一个数学模型以及定义在该模型上的一组操作。

- “抽象”—— 数学抽象特性
- 使用与实现分离，实现“封装”和“信息隐蔽”
  - 隐藏运算实现的细节和内部数据结构，同时向用户提供该数据类型的完整信息
  - 提高软件的复用率，有助于程序的维护、修改和移植



1.2 基本概念和术语

- 抽象数据类型定义：

  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〉

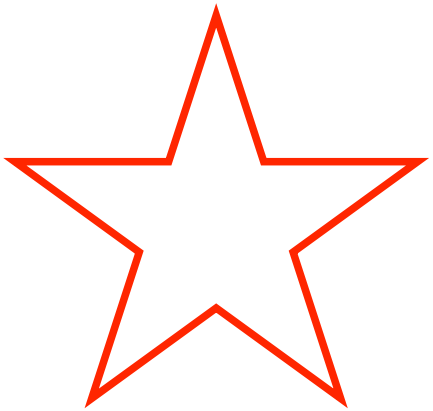
  } ADT 抽象数据类型名

  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〉



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和e2分别是复数的实数部分和虚数部分 }

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企求的结果：

z3 = z1 + z2

1.2 基本概念和术语

- 抽象数据类型可通过固有数据类型来表示和实现，即利用处理器中已存在的数据类型来说明新的结构，用已经实现的操作来组合新的操作。

1.3 抽象数据类型的表示与实现

- 本书采用的类C语言　P10
- 抽象数据类型Triplet的表示和实现　P12
- 例如，对以上定义的复数的表示和实现如下：

```
// ---存储结构的定义，用类型定义typedef描述
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( complex Z );
  // 返回复数 Z 的实部值

float Getimag( complex 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 }
```

1.4 算法和算法分析

- 研究和设计算法是为了解决问题。
- 解决问题的大致流程：分析问题 → 确定算法 → 选择语言并编码 → 调试运行→ 解决问题
- 所谓算法是对计算过程步骤（或状态）的一种刻划，是计算方法的一种能行实现方式。

1.4.1 Knuth对算法的定义

- 算法是对特定问题求解步骤的一种描述。此外，算法的规则序列须满足如下五个条件：

（1）有穷性　一个算法必须总是在执行有穷步之后结束，且每一步都在有穷时间内完成。

（2）确定性　算法中每一条指令必须有确切的含义，不存在二义性。且算法只有一个入口和一个出口，即只有一条执行路径。

（3）可行性　一个算法是可行的。即算法描述的操作都是可以通过已经实现的基本运算执行有限次来实现的。

（4）输入　一个算法有零个或多个输入，这些输入取自于某个特定的对象集合。

（5）输出　一个算法有一个或多个输出，这些输出是同输入有着某些特定关系的量。

1.4.2 算法设计的要求

- 评价一个好的算法有以下几个标准:

(1) 正确性(Correctness) 算法应满足具体问题的需求，及4个层次的“正确”的理解（见P14)。

(2) 可读性(Readability) 算法应该好读。以有利于阅读者对程序的理解。

(3) 健状性(Robustness) 算法应具有容错处理。当输入非法数据时，算法应对其作出反应，而不是产生莫名其妙的输出结果。

(4) 高效率与低存储量需求 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。

1.4.3 算法效率的度量

- 对一个算法要作出全面的分析可分成两个阶段进行，即事先分析和事后测试
- 事后测试　收集此算法的执行时间和实际占用空间的统计资料。
- 事先分析　求出该算法的一个时间界限函数。
  ——该时间界限函数应该如何设计？

例 1：

```
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];
  }
```

- 这个时间界限函数与问题规模有关。
- 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量度。

例１：

```
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];
   }
```

- 语句频度：语句的执行次数。
  c[i][j]=0的语句频度是n^2；
  乘法运算的语句频度是n^3。
- 时间复杂性：以乘法运算作为基本操作，整个算法的执行时间与该基本操作重复执行的次数n^3成正比，记作$T(n)=O(n^3)$。

- 大O的形式定义：
  对于函数$g(n)$和$f(n)$，若存在正常数c和n_0，对于所有的$n \geqq n_0$，$|g(n)| \leqq c|f(n)|$，则记作$g(n)=O(f(n))$
- 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，算法的时间量度记作$T(n)=O(f(n))$，称作算法的渐近时间复杂度，简称时间复杂度。
- 显然，被称做基本操作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，它的执行次数和包含它的语句的频度相同。
- 语句的频度：是指该语句重复执行的次数。

- 以下例子里，将x自增看成是基本操作。
- 例 2 ： **{++x;s=0; }**

  x自增的语句频度为 1 ， 其时间复杂度仍为 O **(1)**，即常量阶。
- 例 3 ： **for(i=1;i<=n;++i)**

  {++x; s+=x;}

  语句频度为**n**，其时间复杂度为：**O(n)**，即时间复杂度为线性阶。

- 例 4 ： for(i=1;i<=n;++i)

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

  {++x;s+=x;}

语句频度为n^2，其时间复杂度为：$O(n^2)$，即时间复杂度为平方阶。

- 例 5 ： for(i=2;i<=n;++i)

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

  {++x;a[i，j]=x;}

语句++x的执行次数关于n的增长率为n^2,它是语句频度表达式中增长最快的项。

总的语句频度为：

$1+2+3+\ldots+n-2=(1+n-2)\times(n-2)/2=(n-1)(n-2)/2$

$=(n^2-3n+2)/2$

此时，如何表示它的时间复杂度？

- 定理：若$A(n)=a_m n^m+a_{m-1} n^{m-1}+\ldots+a_1 n+a_0$是一m次多项式，则$A(n)=O(n^m)$

  证略。

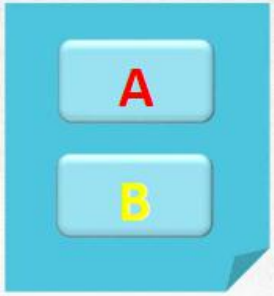
  所以，例5的时间复杂度为$O(n^2)$，即此算法的时间复杂度为平方阶.

- 一个算法时间为O(1)的算法，它的基本运算执行的次数是固定的。因此，总的时间由一个常数（即零次多项式）来限界。而一个时间为$O(n^2)$的算法则由一个二次多项式来限界。

常用的计算规则：

1. 加法规则

$$T(n) = T_1(n) + T_2(n)$$
$$= O(f_1(n)) + O(f_2(n))$$
$$= O(\max(f_1(n), f_2(n)))$$



2. 乘法规则

$$T(n) = T_1(n) \times T_2(n)$$
$$= O(f_1(n)) \times O(f_2(n))$$
$$= O(f_1(n) \times f_2(n))$$

- 例6：选择排序

```
void select_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 = 0;  i< n-1;  ++i ) {
    j = i;   // 选择第 i 个最小元素
    for ( k = i+1;  k < n;  ++k )
       if (a[k] < a[j] )  j = k;
    if ( j != i )  a[j] ←→ a[i]
    }
} // select_sort
```

每一趟从n-i+1个记录中选出关键字最小的记录，并和第i个记录交换之。

- 基本操作：比较(数据元素)操作，
- 时间复杂度: $O(n^2)$

算法示例：起泡排序算法

【自然语言描述】

1. 将整个待排序序列划分成有序区和无序区，初始无序区包括所有n个待排序记录，有序区为空，且始终位于无序区之后。
2. 一趟起泡排序：从前向后依次比较无序区中相邻记录的关键字，若逆序则交换相邻记录，从而使得无序区中关键字最大的记录到了最后，这个记录划入有序区，无序区相应减少1个记录。
3. 如果一趟起泡排序中没有进行过交换记录的操作，则算法结束，否则重复执行步骤2至多n-1趟，直到无序区仅剩1个记录为止。

算法示例：起泡排序算法


a[0] 3
a[1] 8
a[2] 2
a[3] 5
a[4] 4
a[5] 7
a[0] 3
a[1] 8
a[2] 2
a[3] 5
a[4] 4
a[5] 7
a[0] 3
a[1] 2
a[2] 8
a[3] 5
a[4] 4
a[5] 7
a[0] 3
a[1] 2
a[2] 5
a[3] 8
a[4] 4
a[5] 7
a[0] 3
a[1] 2
a[2] 5
a[3] 4
a[4] 8
a[5] 7
无序区
有序区
第1次比较
第2次比较
第3次比较
第4次比较
第5次比较
第1趟起泡排序

算法示例：起泡排序算法


a[0] 3
a[1] 2
a[2] 5
a[3] 4
a[4] 7
a[5] 8
a[0] 2
a[1] 3
a[2] 5
a[3] 4
a[4] 7
a[5] 8
a[0] 2
a[1] 3
a[2] 5
a[3] 4
a[4] 7
a[5] 8
a[0] 2
a[1] 3
a[2] 4
a[3] 5
a[4] 7
a[5] 8
无序区
有序区
第1次比较
第2次比较
第3次比较
第4次比较
第2趟起泡排序

算法示例：起泡排序算法


a[0] 2
a[1] 3
a[2] 4
a[3] 5
a[4] 7
a[5] 8
a[0] 2
a[1] 3
a[2] 4
a[3] 5
a[4] 7
a[5] 8
a[0] 2
a[1] 3
a[2] 4
a[3] 5
a[4] 7
a[5] 8
无序区
有序区
第1次比较
第2次比较
第3次比较
第3趟起泡排序

算法示例：起泡排序算法

【自然语言描述】

1. 将整个待排序序列划分成有序区和无序区，初始无序区包括所有n个待排序记录，有序区为空，位于无序区之后。
2. 一趟起泡排序：从前向后依次比较无序区中相邻两记录的关键字，若逆序则交换相邻记录，从而使得无序区中关键字最大的记录到了最后，这个记录划入有序区，无序区相应减少1个记录。
3. 如果一趟起泡排序中没有进行过交换记录的操作，则算法结束，否则重复执行步骤2至多n-1趟，直到无序区仅剩1个记录为止。

第3趟起泡排序

- 例7：冒泡排序

```
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: $O(n^2)$

- 以下六种计算算法时间的多项式是最常用的。其关系为：

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

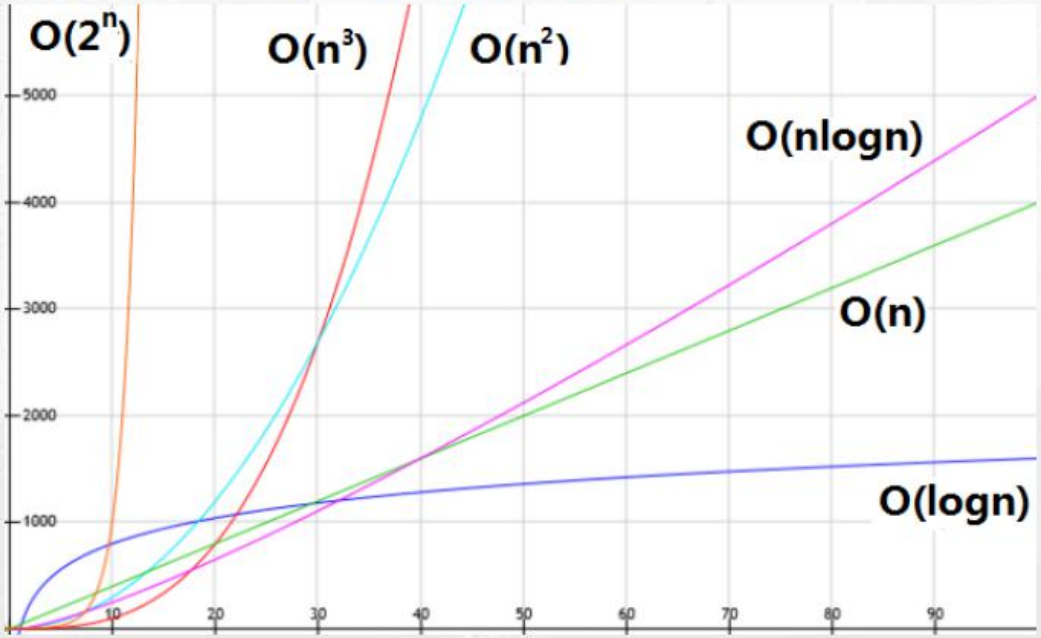
- 指数时间的关系为：

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

- 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。

- **按数量级递增排列，常见的时间复杂度有：**

  常数阶O(1)、对数阶O(logn)、线性阶O(n)、线性对数阶O(nlogn)、平方阶O(n^2)、立方阶O(n^3)、指数阶O(2^n)



- 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。例如：

```
void bubble_sort(int a[]，int n){
for(i=n-1, change=TRUE; i>1 && change; --i){
        change=FALSE;
         for(j=0 ; j<i ; ++j)
            if (a[j]>a[j+1]) {
                a[j] ←→a[j+1];
                 change=TRUE;
              }
   }
}
```

- 最好情况：0次（输入数据为递增有序）
- 最坏情况：1+2+3+…+n-1=n(n-1)/2 （输入数据为递减）

- 平均时间复杂度：分析各种可能输入的概率，以及每种可能情况下的时间复杂度，以求得平均时间复杂度。在很多情况下，各种输入数据集的概率难以确定，算法的平均时间复杂度也就难以确定。
- 在本书中，除非特别指明，时间复杂度一般指最坏情况下的时间复杂度，本例为$O(n^2)$

1.4.4 算法的存储空间需求

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。

- 空间复杂度:算法所需存储空间的度量，记作:

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

  其中n为问题的规模(或大小)
- 以空间为代价，往往可以减少时间，比如查表。
- 以时间为代价，往往可以减少空间：比如压缩。

1.4.4 算法的存储空间需求

- 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

算法与程序的关系：

- 程序是算法在计算机系统中的具体实现
  - 与所用的软硬件平台以及程序设计语言有关
- 算法是对程序中计算过程本质的抽象与描述，是计算方法在抽象计算模型上的一种实现
  - 只注重对抽象数据对象的操作序列，不关心数据对象的存储表示和语言表达

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（5）（7）（8）

1.10

1.20