数据结构与算法

Data Structure and Algorithms

西安交通大学自动化系

蔡忠闽 周亚东

本课内容

1. 为什么要学习数据结构与算法

2. 什么是数据结构与算法

3. 如何学习数据结构与算法

4. 数据结构基础知识

1.4 数据结构基础知识

1.4.1 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)：是数据的基本单位，在计算机程序中通常作为一个整体进行考虑和处理。

  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。

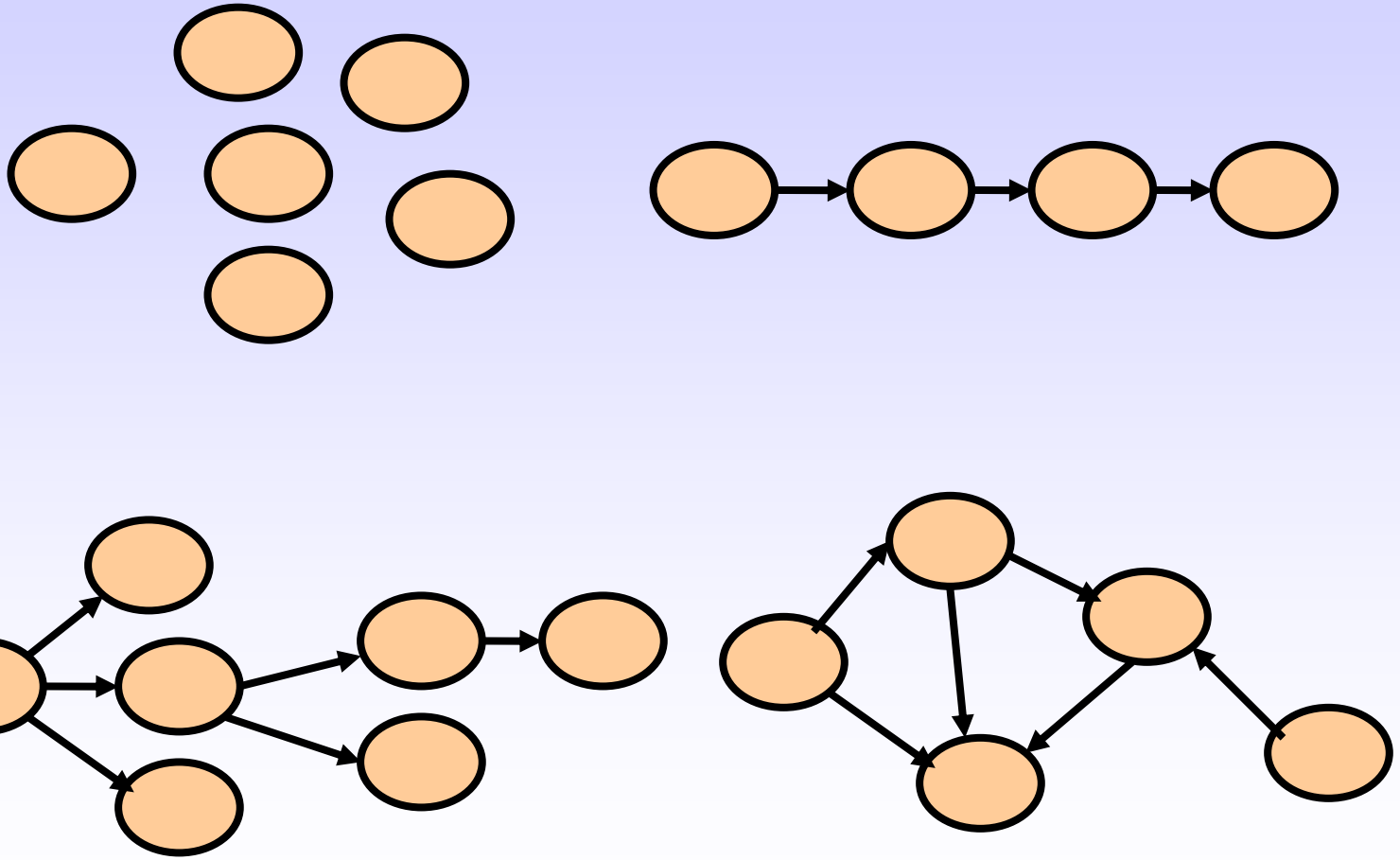
1.4.1 基本概念和术语

结构

- 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。
- 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：

1.4.1 基本概念和术语

逻辑结构示意图



1.4.1 基本概念和术语

结构

- 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。
- 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：数据元素除同属于一种类型外，别无其它关系。

线性结构：数据元素之间存在一对一的关系。

树型结构：数据元素之间存在一对多的关系。

图状结构：数据元素之间存在多对多的关系。

1.4.1 基本概念和术语

【数据结构的形式定义】：

Data_Structure=(D,S)

二元组(two-tuple)
数据表示+关系表示

其中，D是数据元素的有限集，S是D上关系的有限集。

该定义仅是对操作对象的一种数学描述，或者说，是从操作对象抽象出来的数学模型。其中的“关系”描述的是数据间的逻辑关系。

1.4.1 基本概念和术语

- 数据结构在计算机中的表示称为数据的存储结构：

数据表示 关系表示

结点（数据元素）和数据域（数据项）；物理结构

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；（简单、不灵活）

链式存储结构：在每一个数据元素中增加若干存放地址的指针，用此指针来表示数据元素之间的逻辑关系。（复杂、灵活）

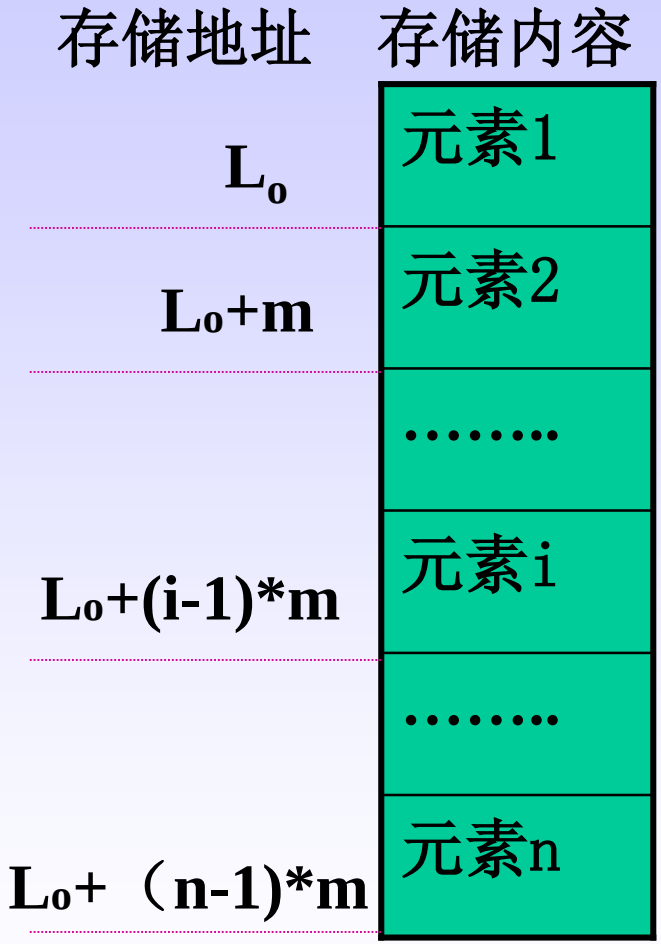
- 数据的逻辑结构与存储结构密切相关：

算法设计 ⟶ 逻辑结构

算法实现 ⟶ 存储结构

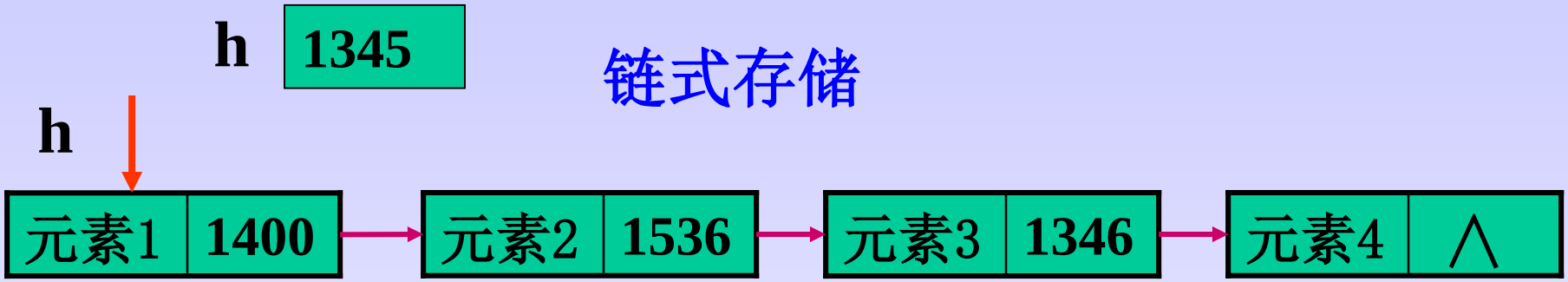
1.4.1 基本概念和术语

顺序存储



Loc(元素i)=Lo+（i-1)*m

1.4.1 基本概念和术语



存储地址	存储内容	指针
1345	元素1	1400
1346	元素4	∧
…….	……..	…….
1400	元素2	1536
…….	……..	…….
1536	元素3	1346

1.4.1 基本概念和术语

- 抽象数据类型（ADT）：一个数学模型以及定义在该模型上的一组操作。ADT实际上定义了一个数据结构的逻辑结构以及在此结构上的一组算法，包含了该数据结构的全部内容。

```
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
```

⇨ 逻辑结构

- 抽象数据类型与数据类型的联系：
  - 定义了数据关系
  - 范畴更广，可以根据用户需要进行定义。
  - 数据类型与抽象数据类型的存储结构相关

1.4.1 基本概念和术语

小结：数据结构的三个方面：

- 数据的逻辑结构
  - 线性结构
    - 线性表
  - 非线性结构
    - 树形结构
    - 图形结构
- 数据的存储结构
  - 顺序存储
  - 链式存储
- 数据的运算：检索、排序、插入、删除、修改等

1.4 数据结构基础知识

1.4.1 基本概念和术语

1.4.2 抽象数据类型的表示与实现

1.4.3 算法和算法分析

1.4.3.1 算法

1.4.3.2 算法设计的要求

1.4.3.3 算法效率的度量

1.4.3.4 算法的存储空间的需求

1.4.2 抽象数据类型的表示与实现

本课程采用类C语言(介于伪代码和C语言)描述各种抽象数据类型的表示和实现。

- 预定义常量和类型
- typedef
- 算法用函数描述
- 赋值语句
- 选择语句
- 循环语句
- 结束语句
- 输入输出语句
- 注释
- 基本函数
- 逻辑运算

1.4 数据结构基础知识

1.4.1 基本概念和术语

1.4.2 抽象数据类型的表示与实现

1.4.3 算法和算法分析

1.4.3.1 算法

1.4.3.2 算法设计的要求

1.4.3.3 算法效率的度量

1.4.3.4 算法的存储空间的需求

1.4.3 算法和算法分析

1.4.3.1 算法(algorithm)

- 算法：一个有穷的指令集，这些指令为解决某一特定任务规定了一个运算序列。
- 算法的特性：

（1）有穷性　算法应在执行有穷步后结束

（2）确定性　每步定义都是确切的，同输入则同输出

（3）可行性　算法由可实现的基本运算构成

（4）输入　有0个或多个输入

（5）输出　有一个或多个输出(处理结果)

1.4.3 算法和算法分析

例：求数组元素a[0]到a[n]的平均值。

```
float  average(int a[], int n)
{ int k;  float temp=0.0;
  if(n<=0){
          printf("input error!\n");
          return(0);
  }
  else{
          for(k=0;k<n;k++)  temp+=a[k];
          return("average=%f\n",temp/n);
   }
}
```

有穷性、确定性、可行性、输入、输出？

1.4.3 算法和算法分析

1.4.3.2 算法设计的要求

(1) 正确性(Correctness)：算法应满足具体问题的需求。

(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) 健壮性(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) 效率与存储量需求：效率指的是算法执行的时间；存储量需求指算法执行过程中所需要的最大存储空间。这两者与问题的规模有关。

1.4.3 算法和算法分析

1.4.3.3 算法效率的度量

- 事后测试　采用时间函数计算此算法的执行时间。
  - 算法的执行时间通常取决于以下因素：

a. 算法选用的策略 ……算法本身

b. 问题的规模 ……问题相关

c. 使用的程序语言

d. 编译程序所产生的机器代码的质量 ……运行平台

e. 机器执行指令的速度

- 事先分析　只考虑算法本身

1.4.3 算法和算法分析

- 事先分析

假定每条语句的执行时间为单位时间。算法的时间复杂度是该算法中所有语句的执行频度之和。

例1：求两个方阵的乘积 C＝A*B

```
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]+=A[i][k]*B[k][j]
     }
```

1.4.3 算法和算法分析

- 事先分析

假定每条语句的执行时间为单位时间。算法的时间复杂度是该算法中所有语句的执行频度之和。

例1：求两个方阵的乘积 C＝A*B

```
for(i=0;i<n;i++)                    //n+1
  for(j=0;j<n;j++)
    {
     C[i][j]=0;
     for( k=0;k<n;k++)
       C[i][j]+=A[i][k]*B[k][j]
     }
```

1.4.3 算法和算法分析

- 事先分析

假定每条语句的执行时间为单位时间。算法的时间复杂度是该算法中所有语句的执行频度之和。

例1：求两个方阵的乘积 C＝A*B

```
for(i=0;i<n;i++)                    //n+1
  for(j=0;j<n;j++)                  //n(n+1)
    {
     C[i][j]=0;
     for( k=0;k<n;k++)
        C[i][j]+=A[i][k]*B[k][j]
     }
```

1.4.3 算法和算法分析

- 事先分析

假定每条语句的执行时间为单位时间。算法的时间复杂度是该算法中所有语句的执行频度之和。

例1：求两个方阵的乘积 C＝A*B

```
for(i=0;i<n;i++)                    //n+1
  for(j=0;j<n;j++)                  //n(n+1)
    {
     C[i][j]=0;                     //n*n
     for( k=0;k<n;k++)
       C[i][j]+=A[i][k]*B[k][j]
    }
```

1.4.3 算法和算法分析

- 事先分析

假定每条语句的执行时间为单位时间。算法的时间复杂度是该算法中所有语句的执行频度之和。

例1：求两个方阵的乘积 C＝A*B

```
for(i=0;i<n;i++)                    //n+1
  for(j=0;j<n;j++)                  //n(n+1)
    {
     C[i][j]=0;                     //n*n
     for( k=0;k<n;k++)              //n*n*(n+1)
       C[i][j]+=A[i][k]*B[k][j]
    }
```

1.4.3 算法和算法分析

- 事先分析

假定每条语句的执行时间为单位时间。算法的时间复杂度是该算法中所有语句的执行频度之和。

例1：求两个方阵的乘积 C＝A*B

```
for(i=0;i<n;i++)                    //n+1
  for(j=0;j<n;j++)                  //n(n+1)
    {
     C[i][j]=0;                     //n*n
     for( k=0;k<n;k++)              //n*n*(n+1)
       C[i][j]+=A[i][k]*B[k][j]     //n*n*n
    }
```

1.4.3 算法和算法分析

- 事先分析

假定每条语句的执行时间为单位时间。算法的时间复杂度是该算法中所有语句的执行频度之和。

例1：求两个方阵的乘积 C＝A*B

```
for(i=0;i<n;i++)                    //n+1
  for(j=0;j<n;j++)                  //n(n+1)
   {
    C[i][j]=0;                      //n*n
    for( k=0;k<n;k++)               //n*n*(n+1)
      C[i][j]+=A[i][k]*B[k][j]      //n*n*n
   }
```

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

1.4.3 算法和算法分析

- **频度**：语句可能重复执行的最大次数。
- **问题的规模**：算法求解问题的输入量，用整数n表示。
- **时间复杂度**：一个算法的时间复杂度是该算法的时间耗费，一般地说，时间复杂度是问题规模的函数--T(n)。
- **渐近时间复杂度**：通常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)。若随着n的增大， f(n)是算法执行时间的上界，即

$$\lim_{n\to\infty}\frac{T(n)}{f(n)}=C, C\geq 0$$

则记作T(n)= O(f(n))，称作算法的**渐近时间复杂度**，简称**时间复杂度**。

1.4.3 算法和算法分析

【定理】若$f(n)=a_m n^m+a_{m-1}n^{m-1}+\ldots+a_1 n+a_0$是一个m次多项式，则$f(n)=O(n^m)$。

证明：

$$\lim_{n\to\infty}\frac{f(n)}{n^m}=a_m\neq 0$$

所以有$f(n)=O(n^m)$。

【推论】时间复杂度由频度的最高阶项来决定。

1.4.3 算法和算法分析

2、一般情况下，对循环语句只考虑循环体语句的执行次数，而忽略该语句中步长加一、终值判别、循环转移等成份。因此，当有若干个循环语句时，算法的时间复杂度是由嵌套层数最多的循环语句中最内层语句的频度所决定的。

例2：

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

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

$$[n(n+1)(2n+1)/6+n(n+1)/2]/2$$

$$T(n)=O(n^3/6+\ldots)=O(n^3)$$

1.4.3 算法和算法分析

3、如果算法的执行时间是一个与问题规模n无关的常数，则算法的时间复杂度为常数阶，记作T(n)=0(1)。

例3：

```
temp = i;
i = j;
j = temp;
```

一个算法时间为0(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。

1.4.3 算法和算法分析

4、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4：(冒泡排序算法中数据元素的交换操作)

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

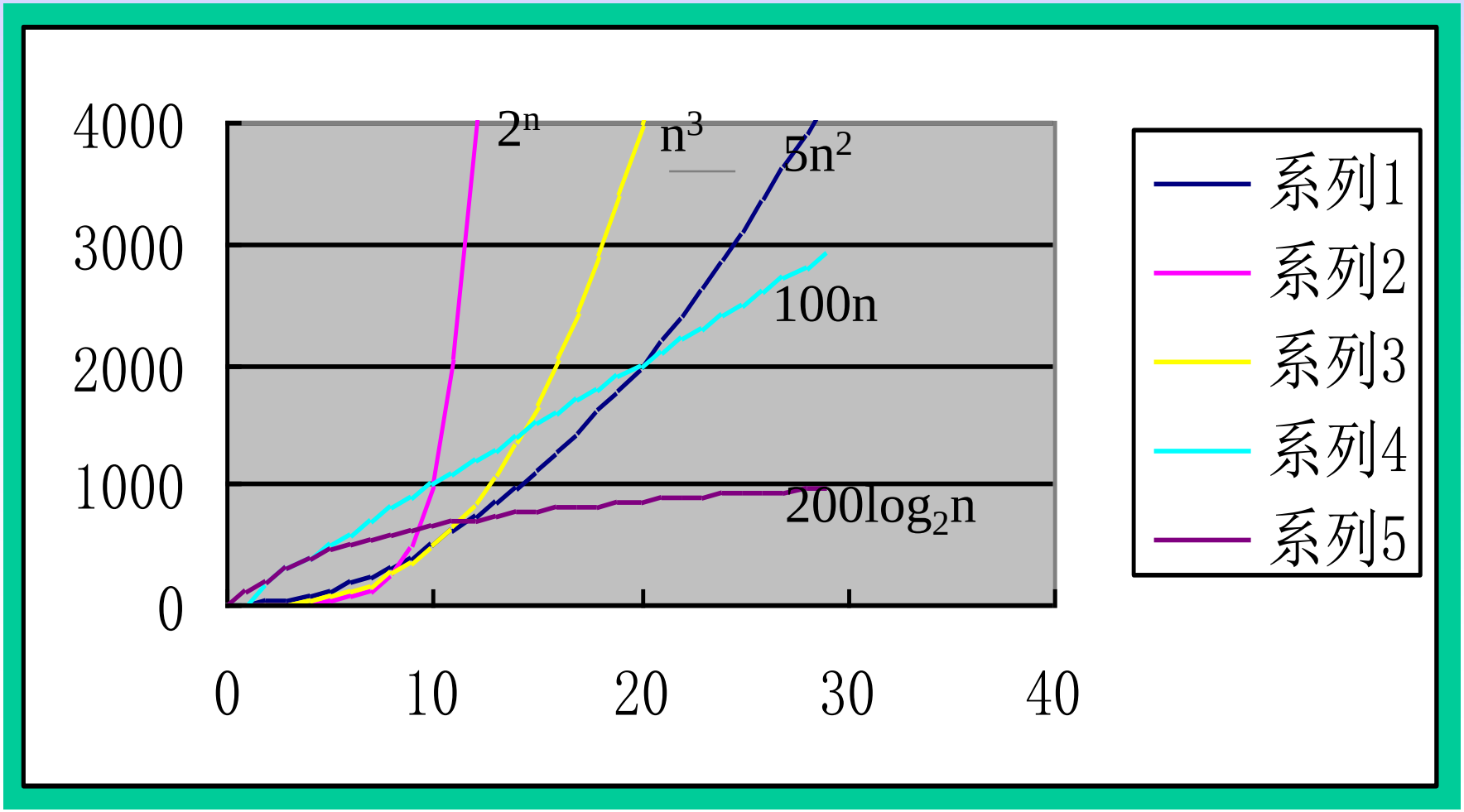
最好情况：0次；最坏情况：1+2+3+...+n-1=n(n-1)/2。

平均时间复杂度为:$O(n^2)$

1.4.3 算法和算法分析

- 以下是最常用的计算算法时间的多项式，其关系为：

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



1.4.3 算法和算法分析

下表给出了复杂性为$f(n)$的程序在每秒执行10亿条指令的计算机上执行时所需要的时间。

n \ f(n)	n	$n\log_2 n$	n^2	n^3	2^n
10	0.01us	0.03us	0.1us	1us	1us
20	0.02us	0.09us	0.4us	8us	1ms
30	0.03us	0.15us	0.9us	27us	1s
40	0.04us	0.21us	1.6us	64us	18m
50	0.05us	0.28us	2.5us	125us	13d
100	0.10us	0.66us	10us	1ms	$4*10^{13}$y
1000	1us	9.96us	1ms	1s	$32*10^{283}$y
10000	10us	130us	100ms	16.67m	
100000	100us	1.66ms	10s	11.57d	
1000000	1ms	19.92ms	16.67m	31.71y	

1.4.3 算法和算法分析

1.4.3.4 算法的存储空间需求

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)。

例：数组逆序问题。

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

该算法需额外使用n个存储单元，S(n)= O(n)

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

该算法需额外使用1个存储单元，S(n)= O(1)

1.3 如何学习数据结构与算法

【课程内容】

☞数据的各种逻辑结构和物理结构（存储结构），以及它们之间的相应关系

☞并对每种结构定义相适应的各种运算

☞设计出相应的算法：增加、删除、遍历、查找、排序

☞分析算法的效率

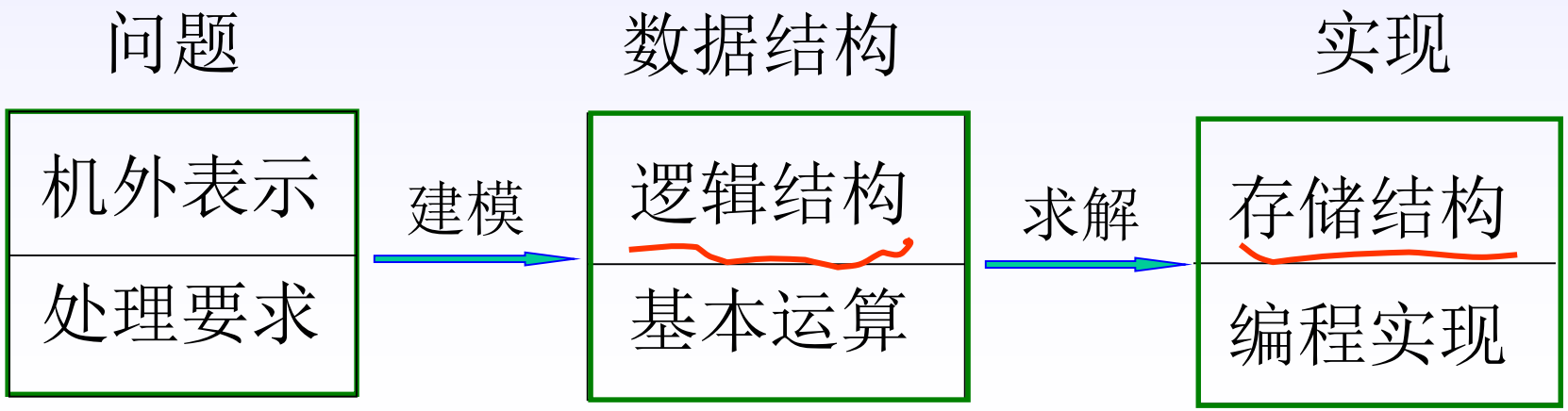
常用数据结构类型： 线性表、栈和队列、串、数组和特殊矩阵、树和二叉树、图。

第二章　线 性 表

什么是数据结构

计算机求解问题的步骤：

- 分析问题；
- 建立求解问题的数据结构并设计算法—通过算法来表示对象数据及其相互关系；
- 实现：编制程序模拟对象领域中的求解过程。



第二章　线 性 表

- 2.1 线性表的类型定义
- 2.2 线性表的顺序表示和实现
- 2.3 线性表的链式表示和实现
  - 2.3.1　线性链表
  - 2.3.2　循环链表
  - 2.3.3　双向链表
- 2.4 一元多项式的表示及相加

2.1 线性结构

在数据元素的非空有限集中：

- 存在唯一一个被称做“第一个”的数据元素；
- 存在唯一一个被称做“最后一个”的数据元素；
- 除第一个元素之外，每个元素都只有一个前驱；
- 除最后一个元素之外，每个元素都只有一个后继。

线性结构

数据元素之间存在着一个对一个的关系

2.1 线性表的类型定义

1. 定义

一个线性表是n个数据元素的有限序列。

例1 英文字母表（A,B,C,…..Z)是一个线性表

例2 学生名单是一个线性表

数据元素(记录)

学号	姓名	性别	年龄
001	王小林	男	18
002	陈　红	女	19
003	刘建平	男	17
…	…	…	…

2.1 线性表的类型定义

2. 线性表的特点

- 元素个数n：即表的长度，n=0时为空表
- 1<i<n时：a_i的前驱是a_{i-1}，

  a_i的后继是a_{i+1}，i为的a_i位序
- a_1无前驱

  a_n无后继
- 元素同构，

$$(a_1, a_2, \ldots\ldots, a_i, \ldots\ldots a_n)$$

线性表的操作

例2 学生名单是一个线性表

学号	姓名	性别	年龄
001	王小林	男	18
002	陈　红	女	19
003	刘建平	男	17
…	…	…	…

1 创建　　销毁
2 添加　　删除 清空
3 访问　　修改

2.1 线性表的类型定义

3. 线性表的抽象数据类型

ADT List {

数据对象：$D=\{ a_i \mid a_i \in ElemSet, i=1,2,...,n, n\geq 0 \}$

数据关系：

基本操作：

2.1 线性表的类型定义

3. 线性表的抽象数据类型

ADT List {

数据对象：$D=\{ a_i \mid a_i \in ElemSet, i=1,2,...,n, n\geq 0 \}$

数据关系：$R1=\{ <a_{i-1}, a_i> \mid a_{i-1}, a_i \in D, i=2,...,n \}$

基本操作：

（1）结构初始化　**InitList(&L)**

操作结果：构造一个空的线性表L。

（2）销毁结构　**DestroyList(&L)**

初始条件：线性表L已存在。

操作结果：销毁线性表L。

2.1 线性表的类型定义

（3）引用型操作

ListEmpty(L)

初始条件：线性表L已存在。

操作结果：若L为空表，则返回TRUE，否则FALSE。

ListLength(L)

初始条件：线性表L已存在。

操作结果：返回L中元素个数。

GetElem(L, i, &e)

初始条件：线性表L已存在， 1≤i≤LengthList(L)

操作结果：用e返回L中第i个元素的值。

2.1 线性表的类型定义

LocateElem(L, e, compare( ))

初始条件：线性表L已存在，compare( )是元素判定函数。

操作结果：返回L中第1个与e满足关系compare( )的元素的位序。若这样的元素不存在，则返回值为0。

ListTraverse(L,visit( )) 线性表遍历

初始条件：线性表L已存在。

操作结果：依次对L的每个元素调用函数visit( )。一旦visit( )失败，则操作失败。

2.1 线性表的类型定义

（4） 修改型操作

ClearList(&L)

初始条件：线性表L已存在。

操作结果：将L重置为空表。

PutElem(&L, i, e)

初始条件：线性表L已存在，

1≤i≤LengthList(L)

操作结果：将e的值赋值给L中第i个元素。

2.1 线性表的类型定义

ListInsert(&L, i, e)

初始条件：L已存在，1≤i≤LengthList(L)+1

操作结果：在L的第i个元素之前插入新的元素e，L的长度增1。

ListDelete(&L, i, &e)

初始条件：L已存在，1≤i≤LengthList(L)

操作结果：删除L的第i个元素，并用e返回其值，L的长度减1。

} ADT List

2.1 线性表的类型定义

4. 算法举例

假设有两个集合A和B分别用两个线性表LA和LB表示（即：线性表中的数据元素即为集合中的成员），现要求一个新的集合A＝A∪B。

要求对线性表作如下操作：扩大线性表LA，将存在于线性表LB中而不存在于线性表LA中的数据元素插入到线性表LA中去。

(注：∪ —并集，属于A或者属于B)

2.1 线性表的类型定义

解题思路：

1 从线性表LB中依次取得每个数据元素：

GetElem(LB, i, e)

2 依e的值在线性表LA中进行查访：

LocateElem(LA, e, equal( ))

3 若不存在，则插入之。

ListInsert(LA, n+1, e)

2.1 线性表的类型定义

```
void union(List &La, List Lb)
{       La_len = ListLength(La);
        Lb_len =ListLength(Lb);
        for (i = 1; i <= La_len; i++) {
                GetElem(Lb, i, e);
                if(!LocateElem(La, e, equal( ))
                        ListInsert(La, ++La_len, e);
        }
}
```

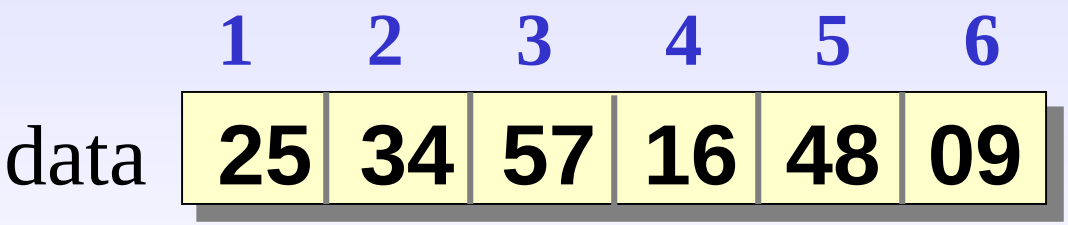
第二章　线 性 表

- 2.1 线性表的类型定义
- 2.2 线性表的顺序表示和实现
- 2.3 线性表的链式表示和实现
  - 2.3.1　线性链表
  - 2.3.2　循环链表
  - 2.3.3　双向链表
- 2.4 一元多项式的表示及相加

2.2 线性表的顺序表示和实现

1、顺序表的定义

把线性表的结点按逻辑顺序依次存放在一组地址连续的存储单元里。



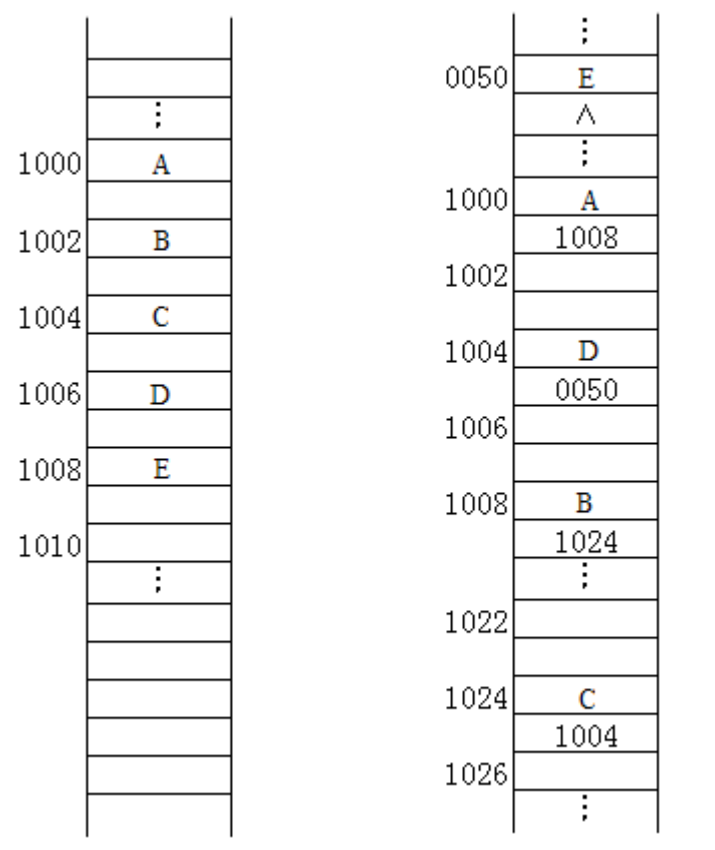
用物理位置来表示逻辑结构。

$LOC(a_{i+1})=LOC(a_i)+l$;

$LOC(a_i)=LOC(a_1)+(i-1)*l$

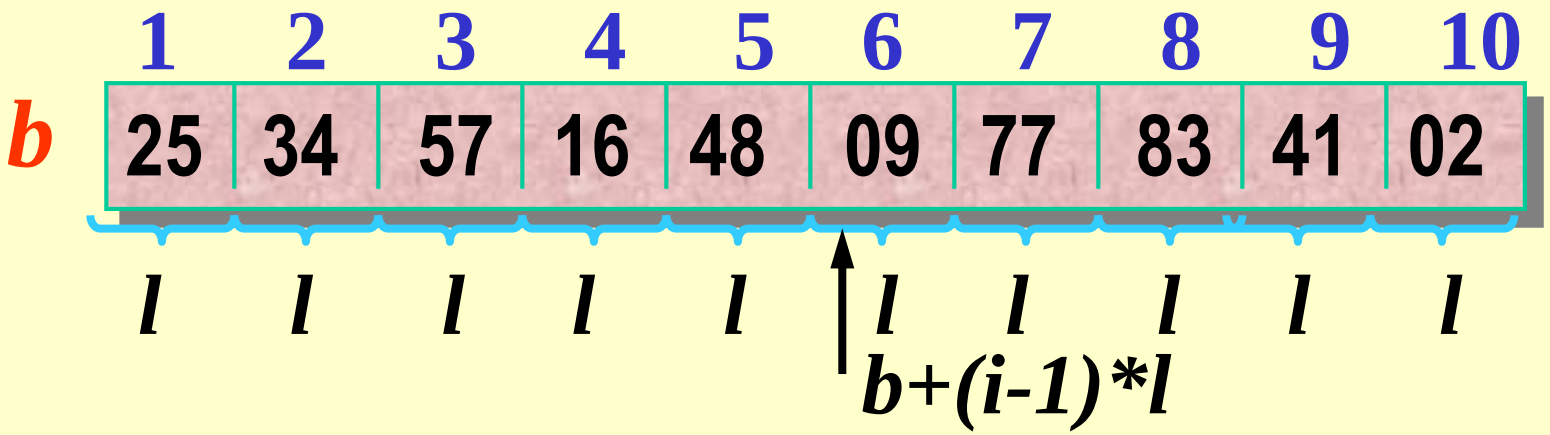
特点：
随机存取的存储结构。只要确定了存储线性表的起始位置，线性表中的任一数据元素可随机存取。

内存结构


1000
A
1002
B
1004
C
1006
D
1008
E
1010
0050
E
^
1000
A
1008
1002
1004
D
0050
1006
1008
B
1024
1022
1024
C
1004
1026

2.2 线性表的顺序表示和实现

$$LOC(a_i) = LOC(a_{i-1}) + l = b + (i-1)*l$$



第一章　作业

- 《数据结构题集》P.7 - P.10
  - 1.4
  - 1.8 (1) (2) (3) (4)
  - 1.12
  - 1.17
- 下周二（9月20日）上课时交