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

绪论

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

1.1 为什么要学习数据结构

- 用计算机来处理事务，完成任务
  - 对象：数据结构
  - 流程：算法
  - 目标：目标数据或算法结束的条件

“C语言程序设计”的联系和区别

- 英语的单词、语法和文法
  - 单词、语法：什么样的英文是正确的
  - 文法：如何用英文表达的我们的思想
- C语言：
  - 学习指令和语法，程序设计语言的基础，
- 数据结构和算法
  - 写出可以解决实际问题的程序

1.2 数据结构与算法的基本问题

- 如何表示和存储数据
- 如何高效的操作和计算数据
  - 删除、插入
  - 排序、搜索

数据结构的三个方面

逻辑结构—数据之间的逻辑关系（抽象出来的数学模型）

物理结构—数据在计算机中如何表示

运算—解决具体问题的基本操作

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。

【答疑安排及联系方式】

答疑时间：随堂答疑，如有需要可单独约时间

地点：科学馆251

E-mail:　zmcai@mail.xjtu.edu.cn

ydzhou@mail.xjtu.edu.cn

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

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

1.4.1 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。

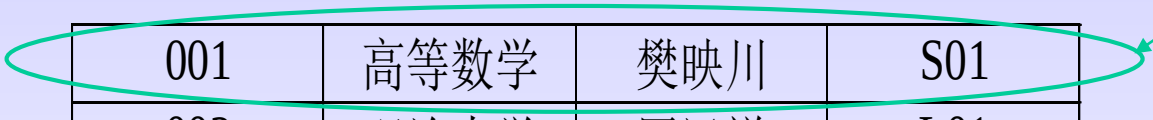
  一个数据元素可由若干个数据项（Data Item）组成。数据项是数据的不可分割的最小单位。

1.1 为什么要学习数据结构

1.4.1 基本概念和术语

图书馆书目数据

数据元素



001	高等数学	樊映川	S01
002	理论力学	罗远祥	L01
003	高等数学	华罗庚	S01
004	线性代数	栾汝书	S02
……	……	……	……

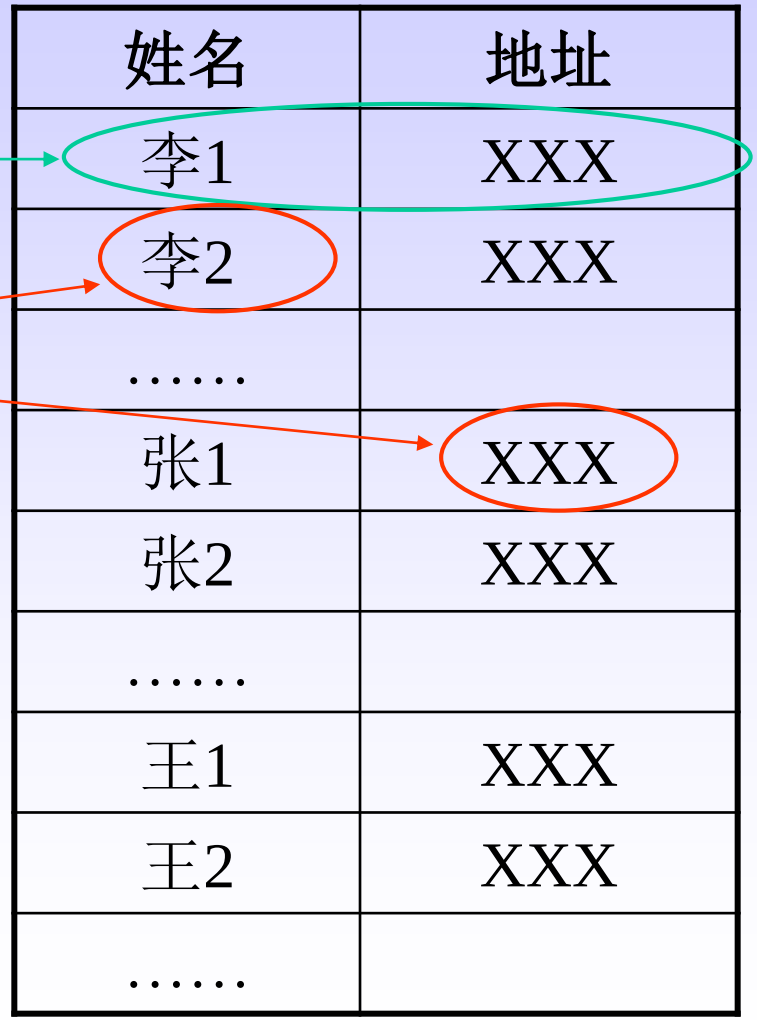
数据项

1.4.1 基本概念和术语

通讯录

数据元素

数据项



姓名	地址
李1	XXX
李2	XXX
……	
张1	XXX
张2	XXX
……	
王1	XXX
王2	XXX
……	

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。

逻辑结构

例1 图书馆的书目检索自动化系统

线性表

书目文件

001	高等数学	樊映川	S01
002	理论力学	罗远祥	L01
003	高等数学	华罗庚	S01
004	线性代数	栾汝书	S02
……	……		….

索引表

按书名

高等数学	001，003……
理论力学	002，……..
线性代数	004，……
……	……..

按作者名

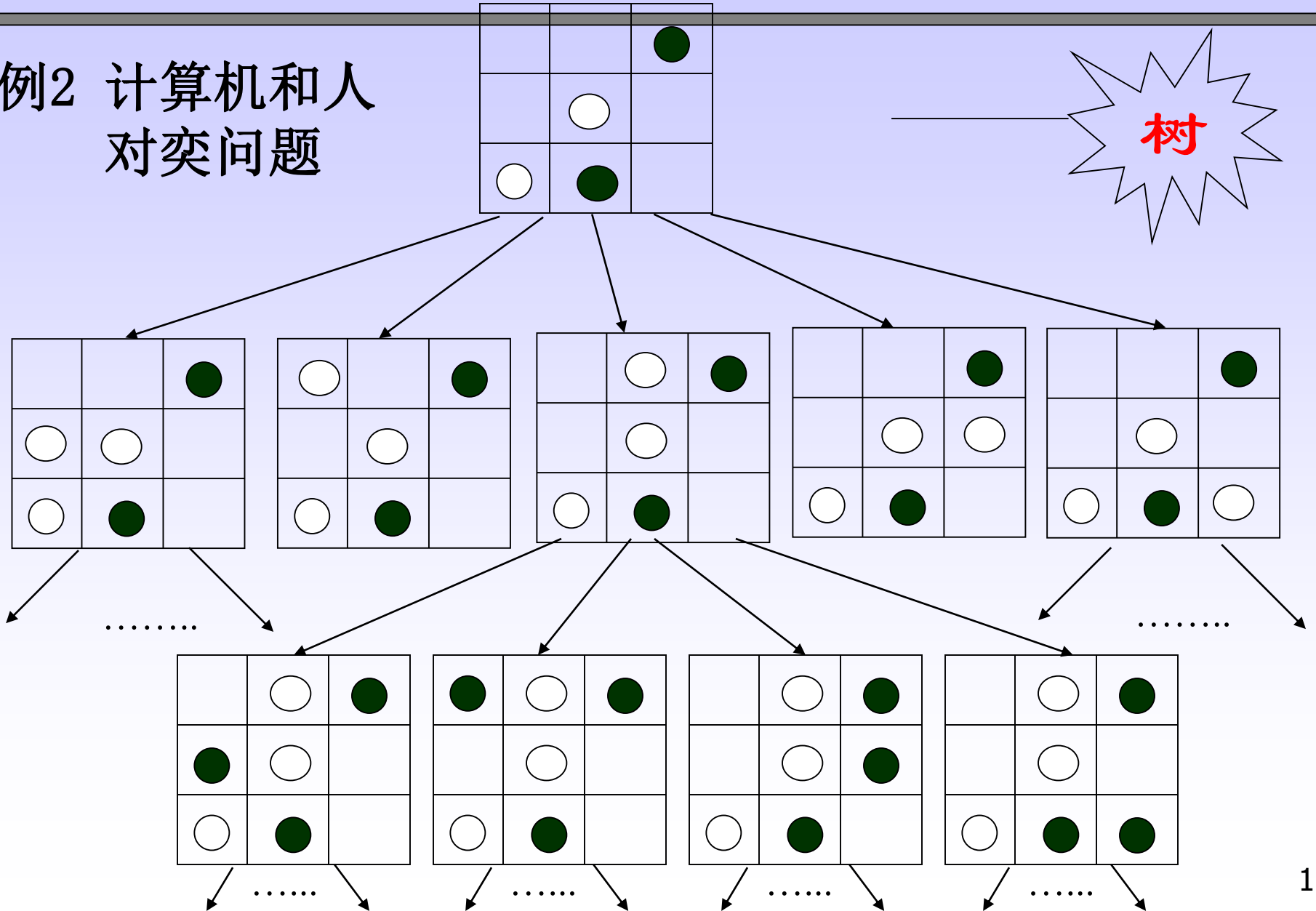
樊映川	001, …
华罗庚	002, ….
栾汝书	004, ….
…….	…….

按分类号

L	002, …
S	001, 003,
……	……

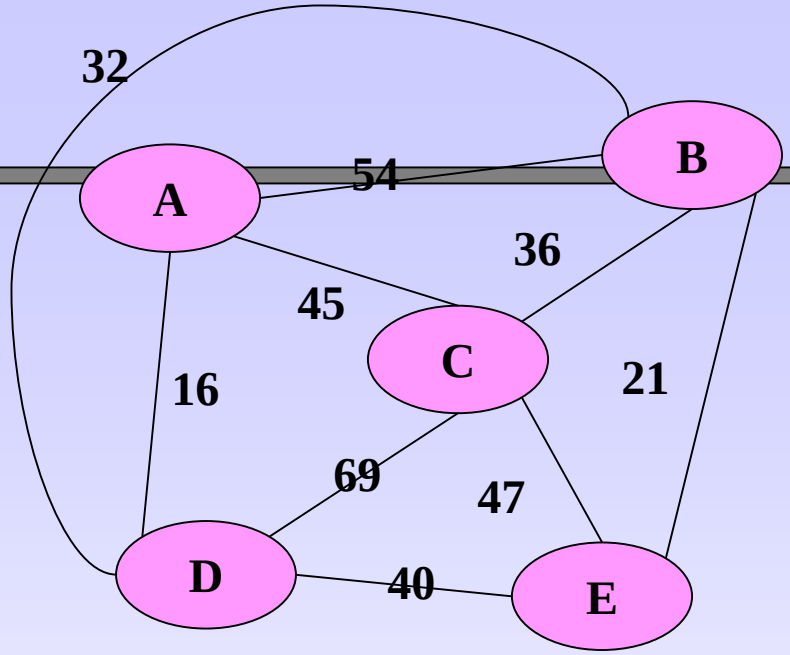
逻辑结构

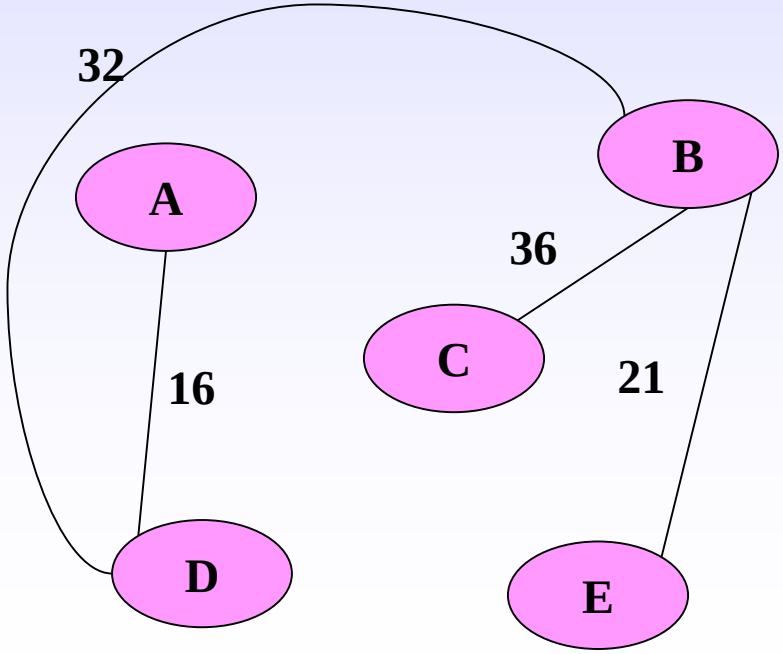
例2 计算机和人对弈问题



例4 铺设煤气管道问题

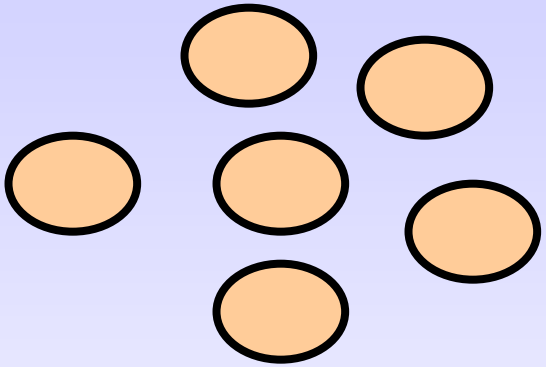
- 假设要在n个居民区之间铺设煤气管道，设任意两个居民区之间都可以架设管道，但每条管道的费用成本不同，求投资最少管道铺设方案。



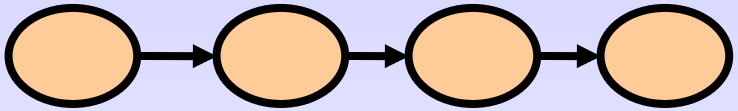


1.4.1 基本概念和术语

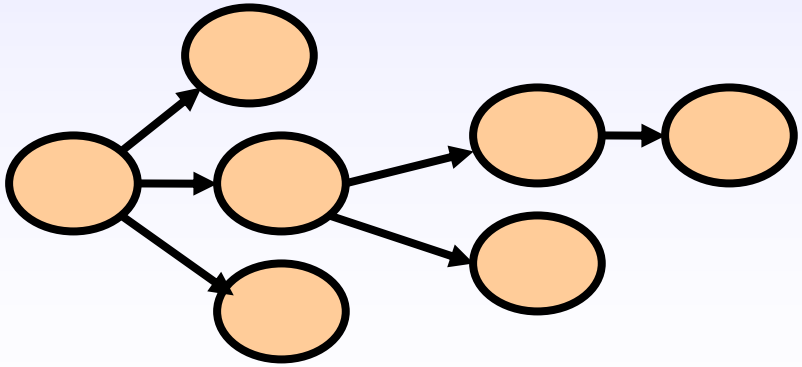
逻辑结构示意图



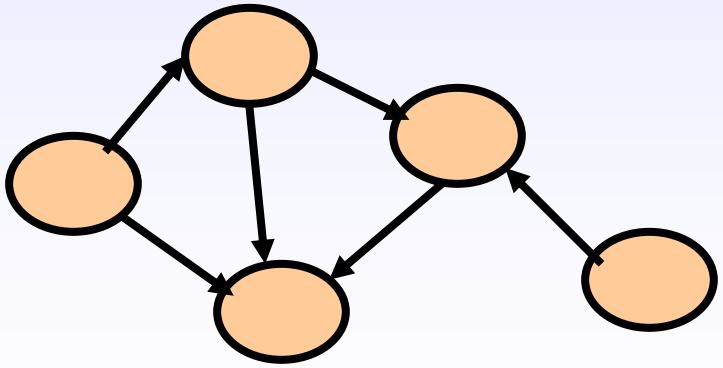
（a）集合



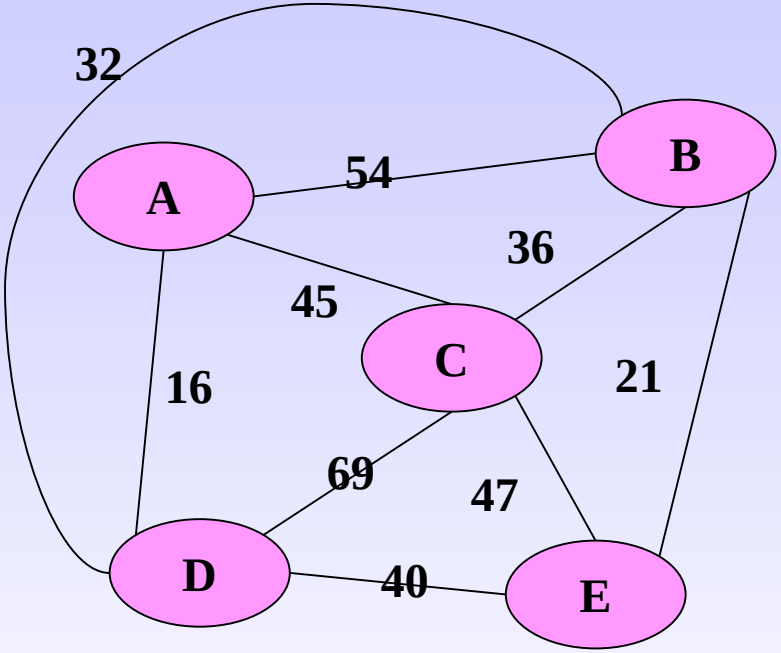
（b）线性结构

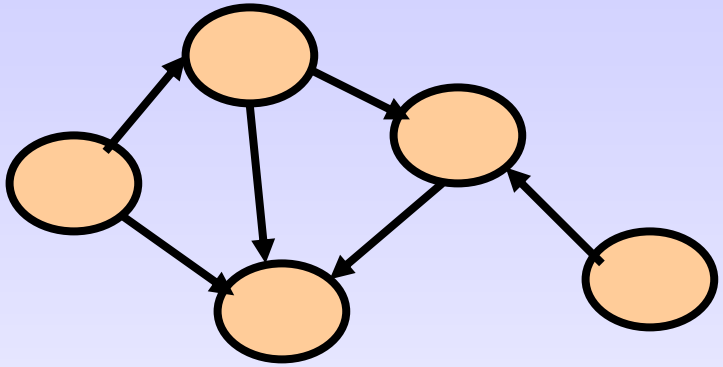


（c）树状结构

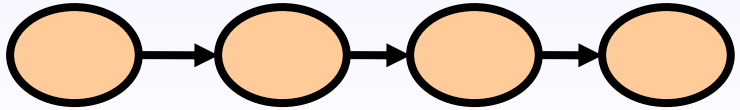


（d）图状或网状结构





001	高等数学	樊映川	S01
002	理论力学	罗远祥	L01
003	高等数学	华罗庚	S01
004	线性代数	栾汝书	S02
……	……	……	……



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）

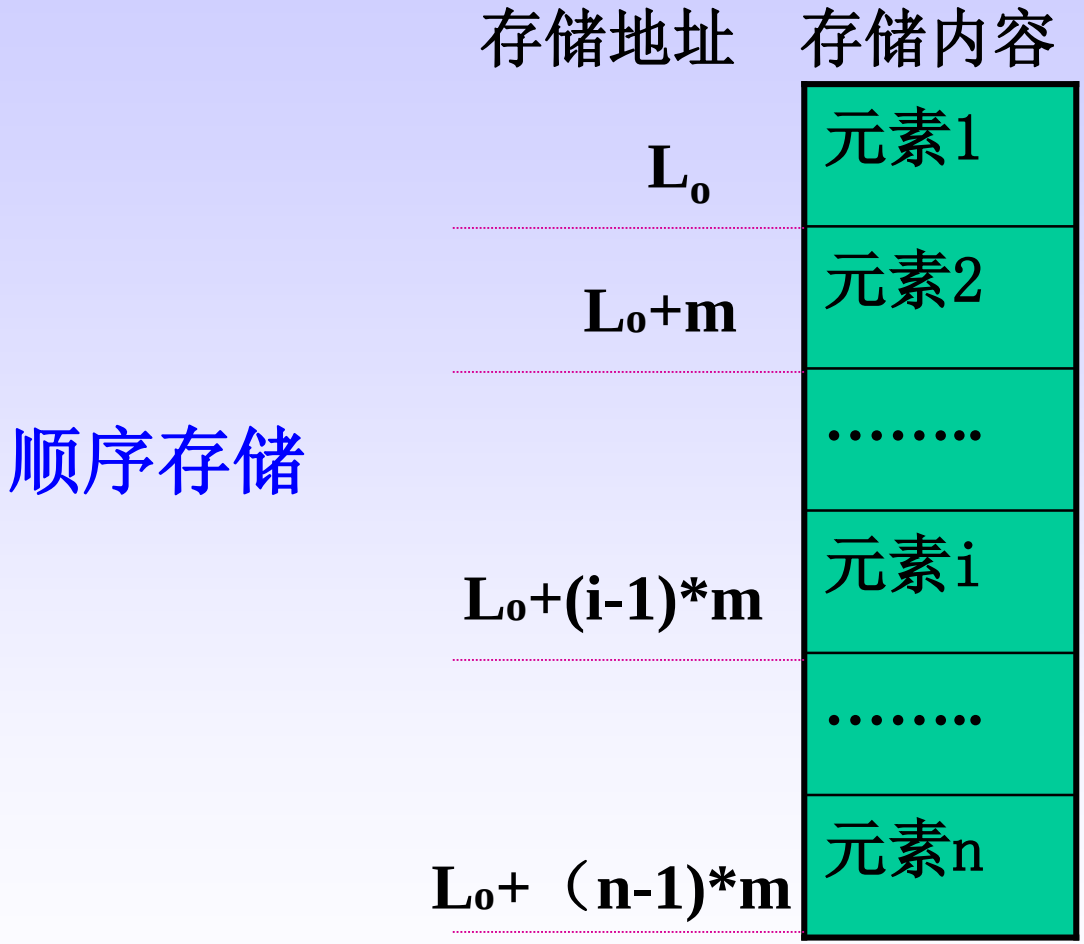
链式存储结构：在每一个数据元素中增加若干存放地址的指针，用此指针来表示数据元素之间的逻辑关系。（复杂、灵活）

- 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

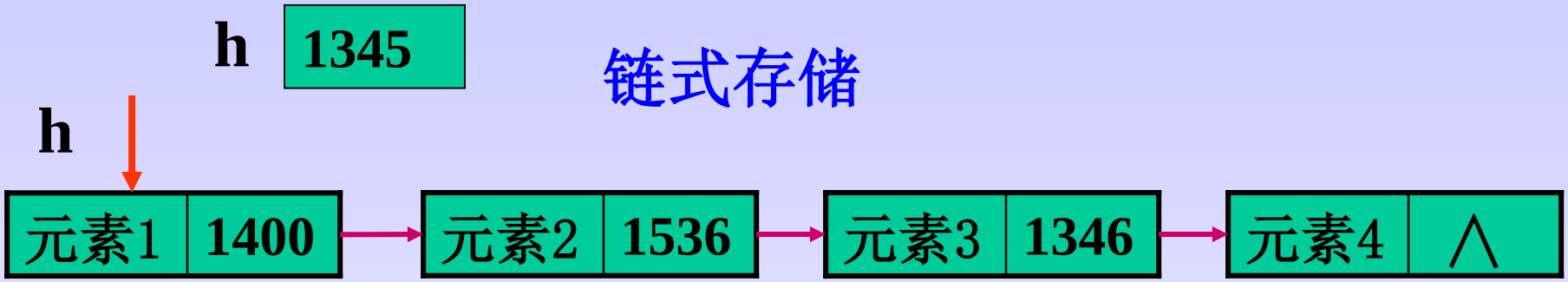
1.4.1 基本概念和术语



顺序存储

Loc(元素i)=Lo+（i-1)*m

1.4.1 基本概念和术语



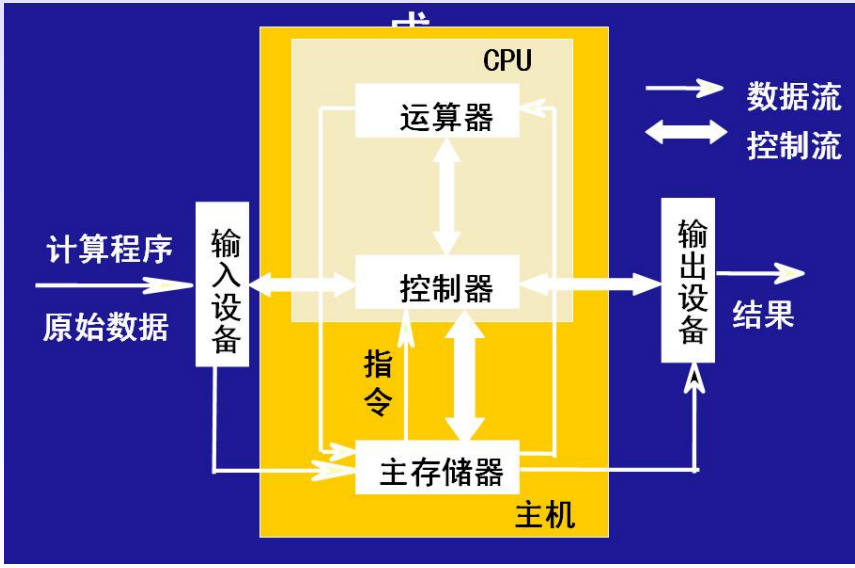
存储地址	存储内容	指针
1345	元素1	1400
1346	元素4	∧
…….	……..	…….
1400	元素2	1536
…….	……..	…….
1536	元素3	1346

冯•诺依曼计算机体系结构

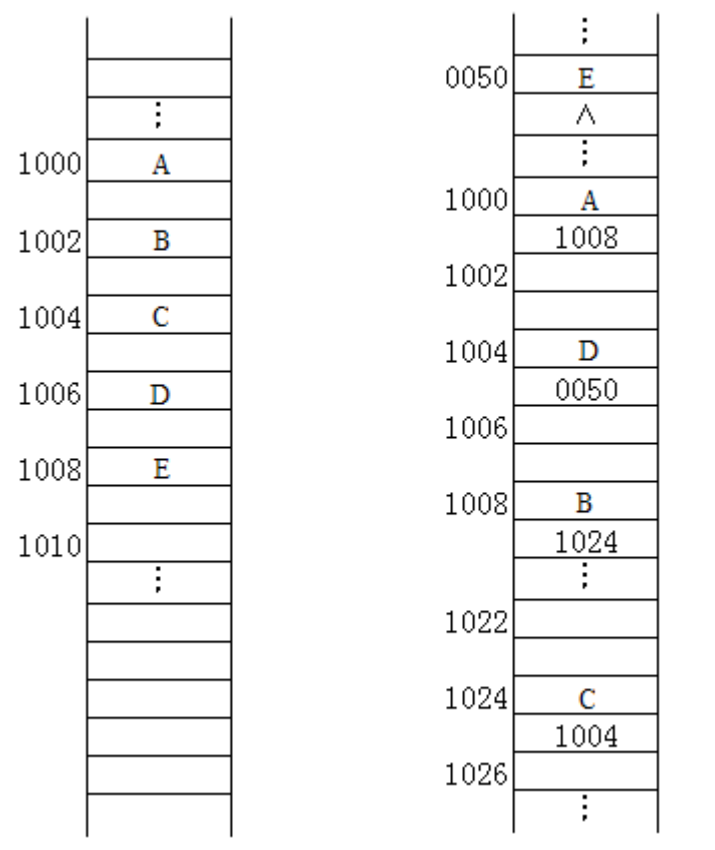
冯•诺依曼型计算机是将程序和数据实现存放在外存储器中，在执行时将程序和数据从外存装入内存中，然后是计算机在工作时自动地从内存中取出指令并加以执行。

主要特点：

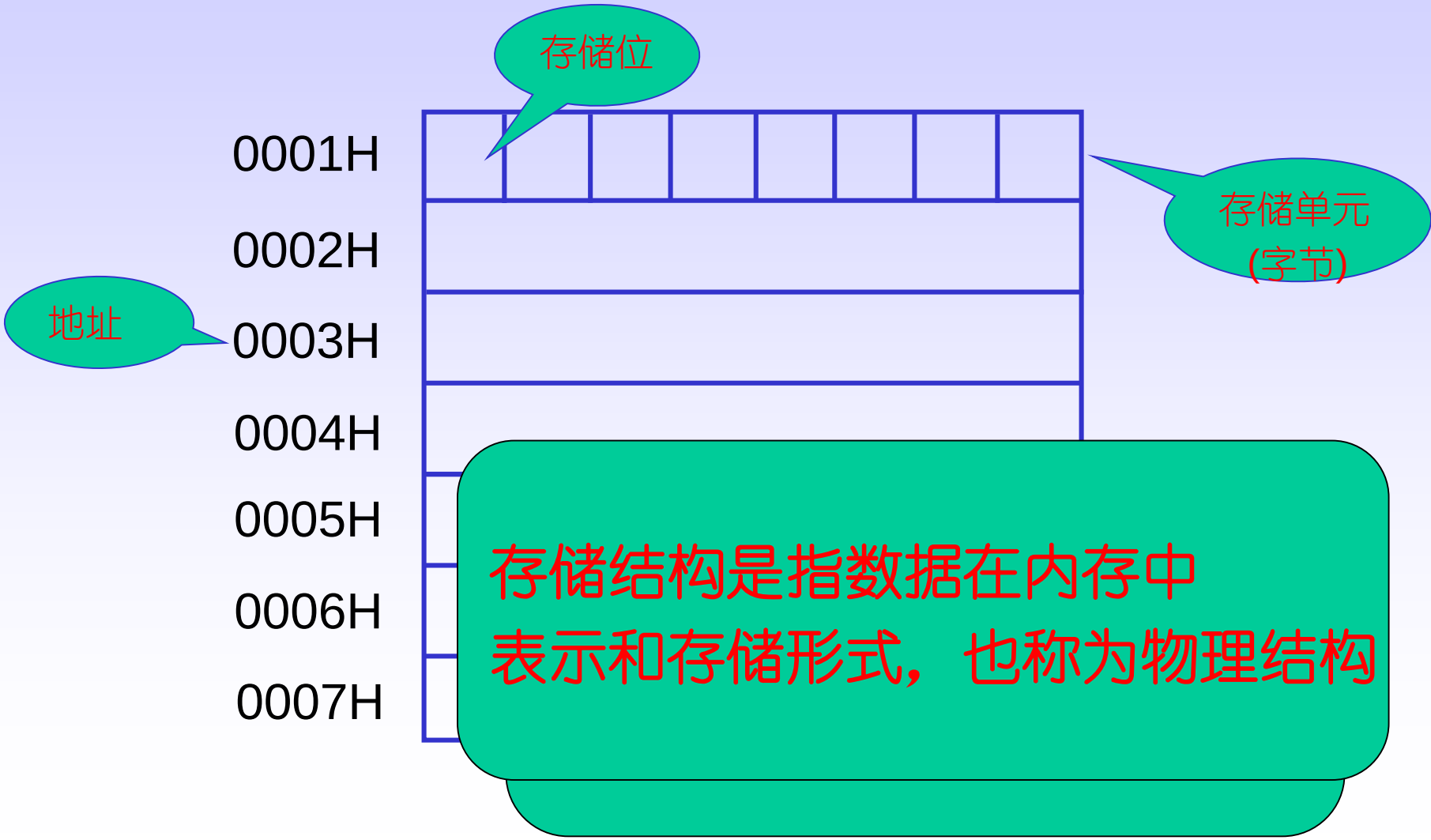
1. 采用二进制形式表示程序和数据
2. 硬件由运算器、控制器、存储器、输入设备和输出设备五部分构成
3. 程序和数据以二进制存放在存储器中
4. 控制器可以根据存放在存储器中的指令工作



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

存储结构


存储位
存储单元
(字节)
地址
0001H
0002H
0003H
0004H
0005H
0006H
0007H
存储结构是指数据在内存中
表示和存储形式，也称为物理结构

1.4.1 基本概念和术语

- 数据类型(DT)：一个值的集合及定义在该集合上的一组操作，用以刻画操作对象的特性。

  在高级语言中，包含：

  - 原子类型 - 值不可分解
    - int, char, unsigned char, char * 等
  - 结构类型— 可以分解
    - 数组等

  在底层硬件系统中，包含：

  - 位、字节、字等原子类型（与、或、移位等）

  “数据类型”的作用：

  - 对计算机来说，解释计算机内存信息的手段
  - 对用户来说，实现底层信息封装，将用户不必了解的细节封装在类型中。

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

ADT bookitem {

数据对象：D＝{ a_i | a_i ∈ElemSet, i=1,2,...,n, n≥0 }

数据关系：R1＝{ <a_{i-1} ,a_i >| a_{i-1}, a_i ∈D, i=2,...,n }

基本操作：InitList(&L) //结构初始化

DestroyList(&L) //销毁结构

ListLength(L) //判断长度

GetElem(L, i, &e) //返回第i个元素的值

ListInsert(&L, i, e) //插入e至第i个

…… }

选择逻辑结构

选择物理结构

实现逻辑结构

编写算法

解决问题

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

1.4.2 抽象数据类型的表示与实现

- 预定义常量和类型：

```
//函数结果状态代码
  #define TRUE          1
  #define FALSE         0
  #define OK            1
  #define ERROR         0
  #define INFEASIBLE    -1
  #define OVERFLOW      -2
```

- typedef：类型定义描述。

```
typedef int Status // Status是函数的类型,其值是函数结果状态代码
typedef xxx ElemType
```

1.4.2 抽象数据类型的表示与实现

- 算法用函数描述： 注意C++中的引用调用。

例1:c中的指针调用.

```
void main()
{  int a, b;
   a=5; b=3;
   swap(&a, &b);
   printf(a, b);
}
void swap(int *x, int *y)
{ int temp;
  temp=*x; *x=*y; *y=temp;
}
```

function( )

```
swap(a, b);
void swap(int x, int y)
```

```
swap(a, b);
void swap(int &x, int &y)
```

1.4.2 抽象数据类型的表示与实现

- **赋值语句：**

- 简单赋值： 变量名=表达式；
- 串联赋值： 变量名1=变量名2=…=变量名k=表达式；
- 成组赋值：

  （变量名1，…，变量名k）=（表达式1，…，表达式k）；

  结构变量名=结构变量名；

  结构变量名= （表达式1，…，表达式k）；

  变量名[ ]=表达式；

  变量名[起始下标..终止下标]=变量名[起始下标..终止下标]；
- 交换赋值： 变量名<—>变量名；
- 条件赋值： 变量名=条件表达式？表达式T：表达式F；

1.4.2 抽象数据类型的表示与实现

- 赋值语句：

•成组赋值：

结构变量名= （表达式1，…，表达式k）；

```
typedef  struct {
   float realpart;
   float imagpart;
}complex;
```

•条件赋值：　变量名=条件表达式？表达式T：表达式F；

1.4.2 抽象数据类型的表示与实现

- 选择语句：

- 条件语句1：　　if(表达式)　　语句；
- 条件语句2：　　if(表达式)　　语句；
　　else 语句；
- 开关语句：

```
switch(表达式){
        case 值1：    语句序列1;
                      break;
                …;
        case 值n ：   语句序列n;
                      break；
        default ：    语句序列n+1;
}
```

1.4.2 抽象数据类型的表示与实现

- 循环语句：
  - for循环：　　for(初值表达式;条件; 修改表达式)
    循环语句;
  - while 循环：　　while(条件表达式)
    循环语句;
  - do-while循环：　　do{　循环语句;
    }while(条件表达式);
- 结束语句：
  - 函数结束：　return(表达式); 或　return;
  - case结束：　break;
  - 异常结束：　exit(异常代码);

1.4.2 抽象数据类型的表示与实现

- 输入输出语句：
  - 输入语句： scanf([格式串]，变量名1，…，变量名n)；
  - 输出语句： printf([格式串]，表达式1，…，表达式n)；
- 注释：
  - 单行注释： // 文字序列
  - 多行注释： /* 文字序列 */
- 基本函数：

  max(表达式1,2,…,n), min(表达式1,2,…,n), abs(表达式)

  floor(表达式), ceil(表达式), eof(文件变量)
- 逻辑运算约定：

  &&, || (同C语言)

1.4.2 抽象数据类型的表示与实现

例 1-6 抽象数据类型三元组的定义：

```
ADT Triplet {
    数据对象：D = {e1,e2,e3 | e1,e2,e3∈ElemSet（定义了关系运算的某个集
    数据关系：R1 = {<e1,e2> , <e2,e3>}
    基本操作：
      InitTriplet( &T, v1, v2, v3 )
        操作结果：构造了三元组 T，元素 e1,e2 和 e3 分别被赋以参数 v1,v2
      DestroyTriplet( &T )
        操作结果：三元组 T 被销毁。
      Get( T, i, &e )
        初始条件：三元组 T 已存在，1≤i≤3。
        操作结果：用 e 返回 T 的第 i 元的值。
      Put( &T, i ,e )
        初始条件：三元组 T 已存在，1≤i≤3。
        操作结果：改变 T 的第 i 元的值为 e。
      IsAscending( T )
        初始条件：三元组 T 已存在。
        操作结果：如果 T 的 3 个元素按升序排列，则返回 1，否则返回 0。
      IsDescending( T )
        初始条件：三元组 T 已存在。
        操作结果：如果 T 的 3 个元素按降序排列，则返回 1，否则返回 0。
      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。
      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
```

1.4.2 抽象数据类型的表示与实现

例：抽象数据类型Triplet表示和实现

```
#define OK 1
#define OVERFLOW -2
```

– 顺序存储结构

```
typedef  ElemType *Triplet;  /* 由InitTriplet分配三个元素存储空间 */
```

– 基本操作的函数说明

```
Status InitTriplet(Triplet &T, ElemType v1, ElemType v2, ElemType v3)
 //构造三元组T,依次置T的三个元素的初值为v1,v2和v3
Status Get(Triplet T, int i, ElemType &e)
// 1≤i≤3, 用e返回T的第i元的值
……
```

– 基本操作的实现

```
Status InitTriplet(Triplet &T, ElemType v1, ElemType v2, ElemType v3)
 {T=(ElemType *)malloc(3*sizeof(ElemType));
   if(!T)      exit(OVERFLOW);     //分配存储失败
   T[0]=v1, T[1]=v2, T[2]=v3;
   return OK;
 }
……
```

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

1.4.3.2 算法设计的要求

(1)正确性(Correctness)：算法应满足具体问题的需求。

1）程序不含语法错误（编译通过）；

2）对于几组输入数据能够得到要求的结果；

3）对于典型的、苛刻的几组数据得到要求的结果；

4）对一切合法的输入数据都能得到要求的结果。

1.4.3 算法和算法分析

1.4.3.2 算法设计的要求

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

宁可写清晰但长一点的代码；

函数体长度合适；

变量定义规范；

注释；

代码的层级关系（缩进），画流程图，等等。

以前的程序设计不重视可读性，而重视效率（cpu慢，内存小）

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]
     }
```