教材

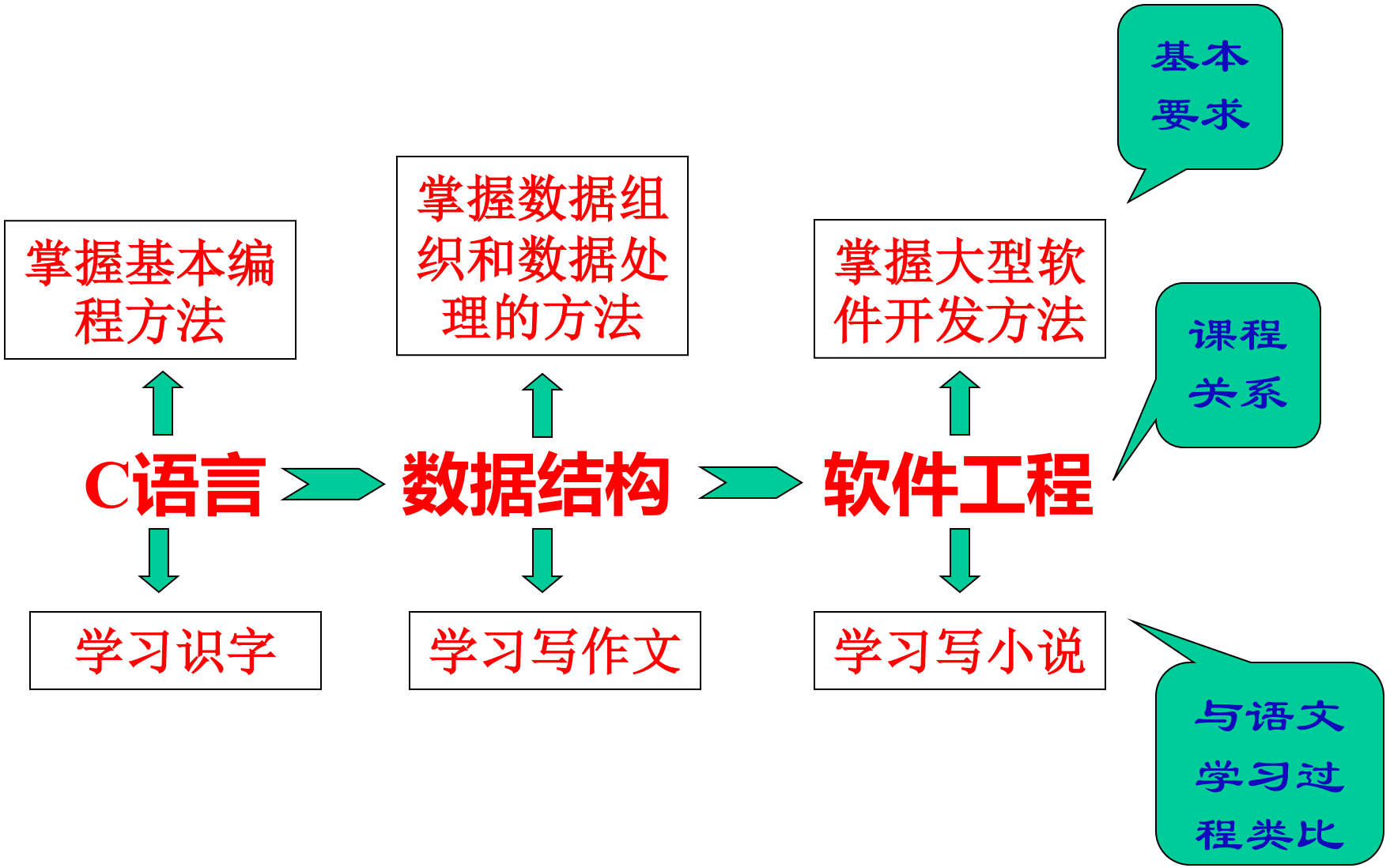
数据结构（C语言版）严蔚敏，吴伟民

数据结构题集（C语言版） 严蔚敏，吴伟民，米宁

作业及考核

- 作业：
  - 时间：周二
  - 数量：每次交三分之一

- 考核：
  - 期末笔试：50%
  - 期末机试：30%
  - 上机实验：10%
  - 平时成绩：10%


基本要求
掌握基本编程方法
掌握数据组织和数据处理的方法
掌握大型软件开发方法
课程关系
C语言
数据结构
软件工程
学习识字
学习写作文
学习写小说
与语文学习过程类比

前期课程

计算机基础

C语言

离散数学

承上启下

数据结构

后期课程

操作系统

编译原理

数据库原理

软件工程

第1章 绪论

第一章 绪论

- 计算机的应用：

  科学计算；

  控制、管理及数据处理等非数值计算的处理工作；

- 计算机加工的对象：

  纯粹的数值；

  文本、表格和图像数据；

- 如何表示、处理这些*新的*、具有一定*结构*的数据？

《数据结构》是一门什么课程

数据结构是一门研究*非数值计算*的程序设计问题时处理的*操作对象*以及它们之间的*关系和操作*等等的学科。

- 解决数值计算问题的**中心**：建立适当的**数学模型**。
- 解决非数值计算问题的**中心**：寻找适当的**数据结构**。

数值问题:

例1，求解梁架结构中的应力。

数学模型：$\mathbf{K\,U = M}$

$$\begin{bmatrix} a_{11} & & \\ & \ddots & \\ & & a_{nn} \end{bmatrix} \times \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$$

例2，预报人口增长情况。

数学模型：

$$\begin{cases} \dfrac{dN(t)}{dt} = r\,N(t) \\ N(t)\big|_{t=t_0} = N_0 \end{cases}$$

$$N(t) = N_0 e^{r\,t}$$

非数值问题:

例1，图书馆的书目检索系统自动化问题。

通过提供书名、作者或分类信息，你就可以从图书馆中检索某一本书。

数据结构:线性表。

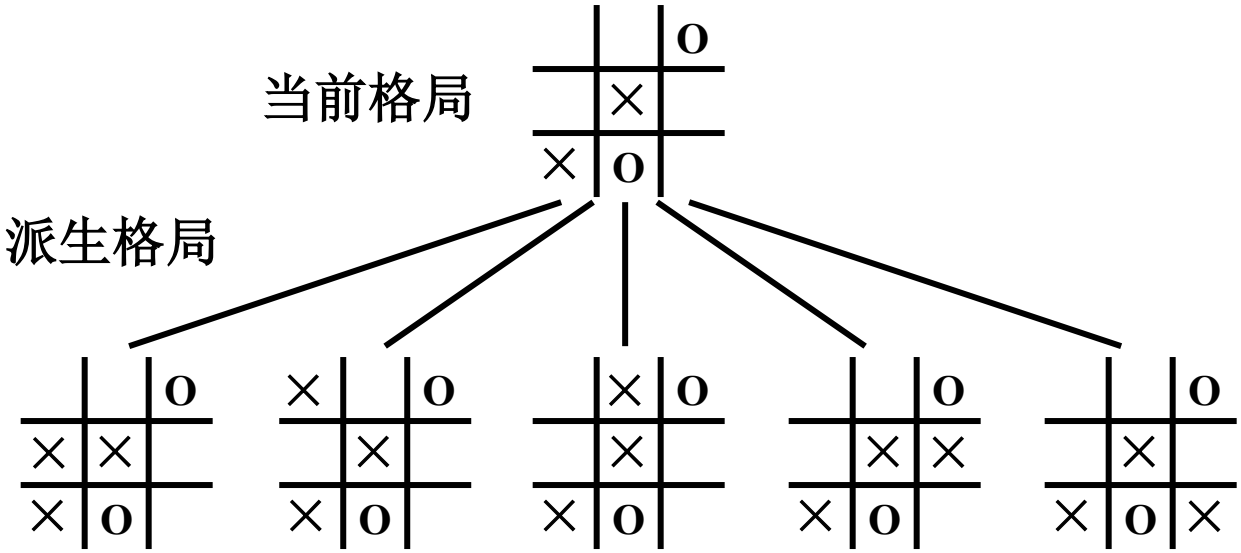
001	数据结构	周劲	S01	…
002	程序设计	潘玉奇	L01	…
003	数据结构	王永燕	S01	…
004	数据库	曲守宁	D01	…
⋮	⋮	⋮	⋮	⋮

数据结构	001,003
程序设计	002
数据库	004
曲守宁	004

例2，计算机和人对奕问题。

计算机可以根据当前棋盘格局，来预测棋局发展的趋势，甚至最后结局。

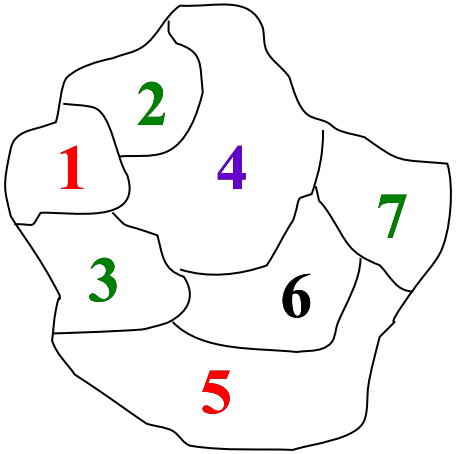
数据结构:对弈树。

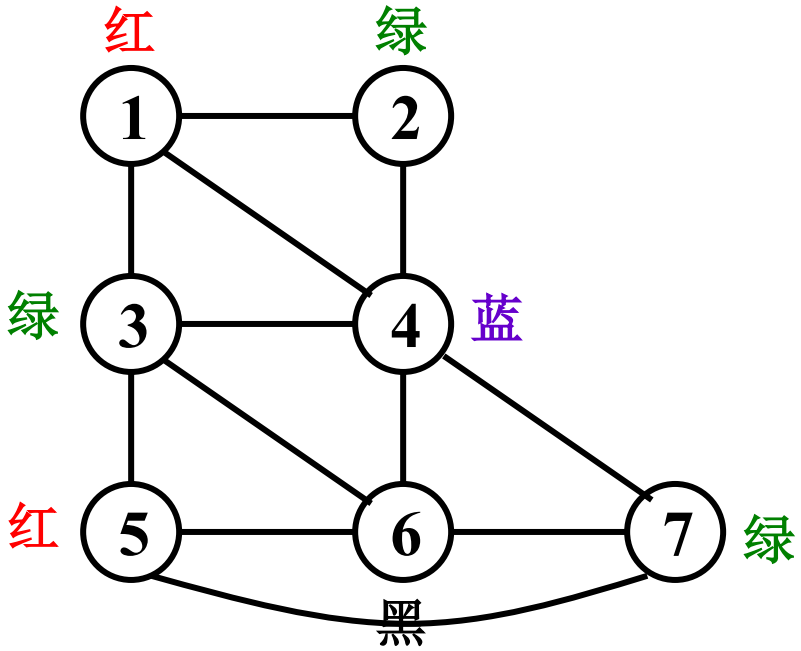


例3，地图的着色问题。

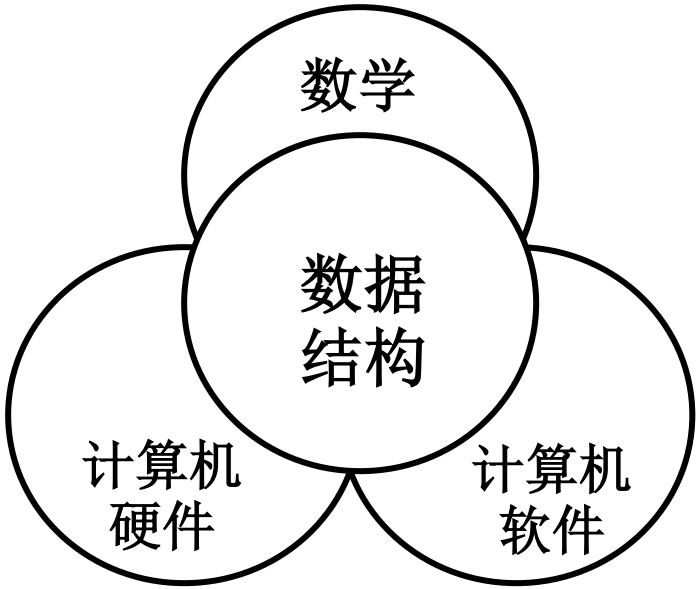
对地图上的每个区域染一种颜色，并且要求相邻的两个区域不能具有相同颜色。

数据结构:图。





用最少的颜色染色


数学
数据
结构
计算机
硬件
计算机
软件

1. 数据（Data）

- 对客观事物的符号描述，能输入到计算机中并被计算机程序处理的符号的总称；
- 能被计算机识别、存储和加工处理的信息的载体。

例，数字：自然数、整数

字母：a～z, 单词

图像：

视频、音频信号等

表格：

2. 数据元素（Data Element）

数据元素是组成数据的基本单位, 是数据集合的个体，在计算机中通常作为一个整体进行考虑和处理。

例，“对弈树”中的一个格局

书目信息中的一条书目

数据项: 一个数据元素可由若干个数据项组成。

例，一条书目信息是由书名、作者名、分类等多个数据项组成的

数据项是数据的不可分割的最小单位。

例如 有一个学生表如下所示。这个表中的数据元素是学生记录,每个数据元素由四个数据项(即学号、姓名、性别和班号)组成。

学号	姓名	性别	班号
1	张斌	男	9901
8	刘丽	女	9902
34	李英	女	9901
20	陈华	男	9902
12	王奇	男	9901
26	董强	男	9902
5	王萍	女	9901

3. 数据结构（Data Structure）

数据结构是指相互之间存在一种或多种特定关系的数据元素集合，

结构（Structure）

数据元素相互之间的关系。

- **在形式上可用二元组表示：**

  Data_Structure = (D,S)

  D: 数据元素的有限集

  S: D上关系的有限集

$D = \{ k_i \mid 1 \le i \le n, n \ge 0 \}$

- k_i表示集合D中的第i个结点或数据元素
- n为D中结点的个数
- 若n=0，则D是一个空集，表示D无结构可言，有时也可以认为它具有任意的结构

$S=\{r_j \mid 1\le j\le m,\ m\ge 0\}$

- r_j 表示集合S中的第j个二元关系(简称关系)
- m为S中关系的个数
- 若m=0，则S是一个空集，表明集合D中的元结点间不存在任何关系，彼此是独立的

D上的一个关系r是序偶的集合，对于r中的任一序偶<x,y>(x,y∈D),我们称序偶的第一结点为第二结点的直接前驱结点(通常简称前驱结点),称第二结点为第一结点的直接后继结点(通常简称后继结点)。如在<x,y>的序偶中,x为y的前驱结点,而y为x的后继结点。

若某个结点没有前驱,则称该结点为开始结点；若某个结点没有后继,则称该结点为终端结点；除此之外的节点称为内部节点。

“尖括号”表示有向关系，“圆括号”表示无向关系。

例如,用二元组表示学生表，学生表中共有7个结点,依次用k1 ~ k7表示,则对应的二元组表示为

Data_Structure=(D,S)

其中:

D={k1, k2, k3, k4, k5, k6, k7}

S= {<k1,k2>, <k2,k3>, <k3,k4>,

<k4,k5>, <k5,k6>, <k6,k7>}

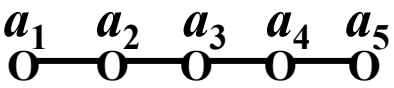
逻辑结构图： 可以将数据结构用图形形象地表示出来,图形中的每个结点对应着一个数据元素,两结点之间的连线对应着关系中的一个序偶。

上述“学生表”数据结构用下图的图形表示。

例1，内部关系，复数Complex = (C，R)

其中: C是含两个实数的集合{$c1$,$c2$}；R = {P}，而P是定义在集合C上的一种关系{<$c1$,$c2$>}，其中有序偶<$c1$,$c2$>表示$c1$是复数的实部，$c2$是复数的虚部。

例2，外部关系，线性表List = (C，R)　　线性



其中: C是数据记录的集合{a_i}；R = {P}，而P是定义在集合C上的一种关系{<a_{i-1}, a_i>}，其中有序偶<a_{i-1}, a_i>表示a_{i-1}是a_i的直接前驱元素，a_i是a_{i-1}的直接后继元素。

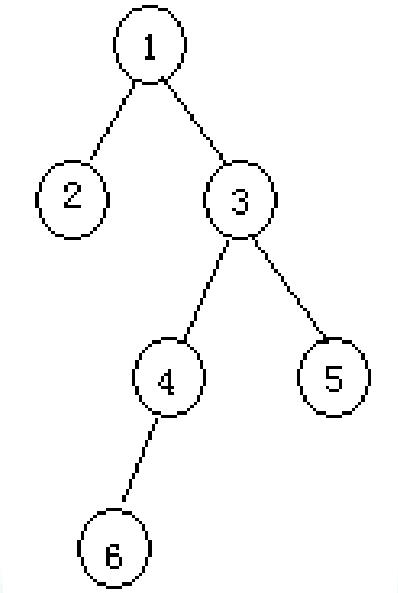
例3、设数据的结构描述如下：

Tree = (D, R)

D = {1,2,3,4,5,6}

R = {<1,2>, <1,3>, <3,4>, <3,5>, <4,6>}

画出其逻辑结构图？



1.2 数据结构的内容

1. 逻辑结构

数据元素之间的关系 。

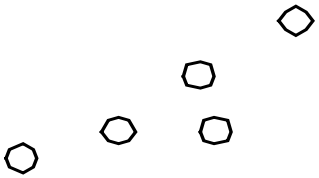
逻辑结构可看作是从具体问题抽象出来的数学模型。

逻辑结构类型的分类

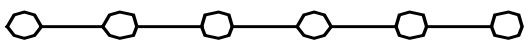
按照逻辑关系的不同特性分类：

- 集合：（同属于一个集合）
- 线性结构：（一对一）
- 非线性结构：
  - 树型结构：（一对多）
  - 图形结构：（多对多）

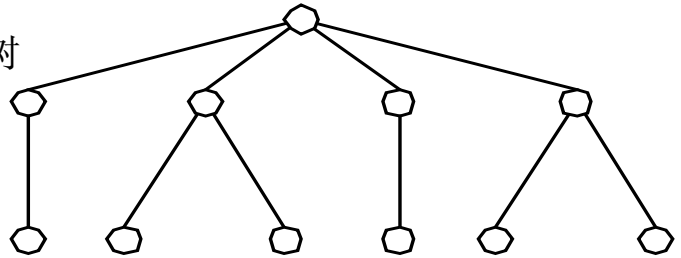
集合



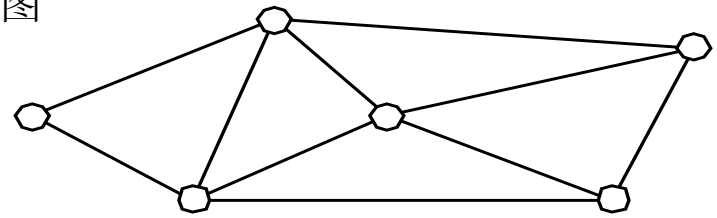
线性表



树



图



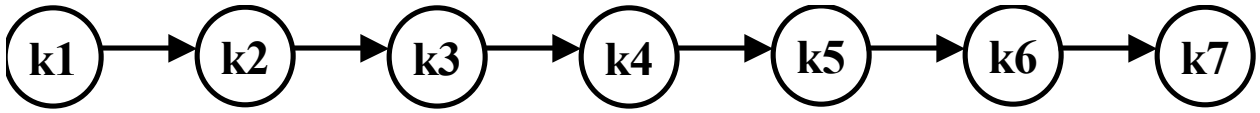
逻辑结构类型

(1)线性结构

所谓线性结构,该结构中的结点之间存在一对一的关系。

其特点是：开始结点和终端结点都是惟一的,除了开始结点和终端结点以外,其余结点都有且仅有一个前驱结点,有且仅有一个后继结点。

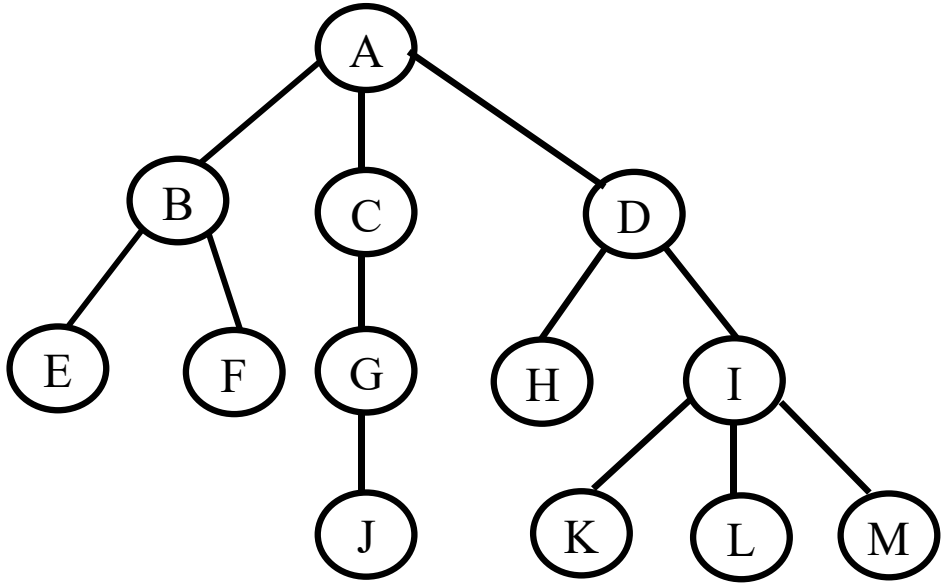
顺序表就是典型的线性结构。



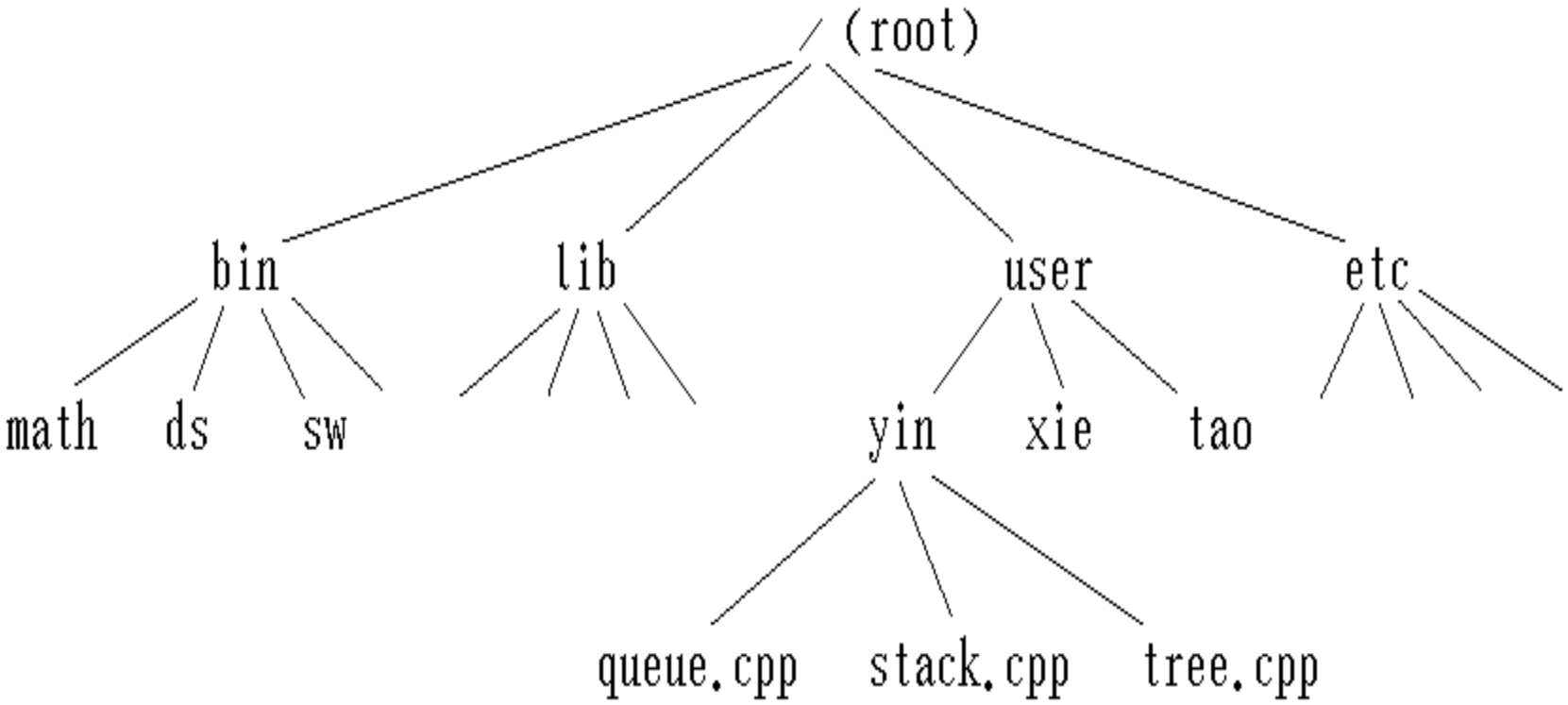
(2)非线性结构

所谓非线性结构,该结构中的结点之间存在一对多或多对多的关系。它又可以细分为树形结构和图形结构两类。

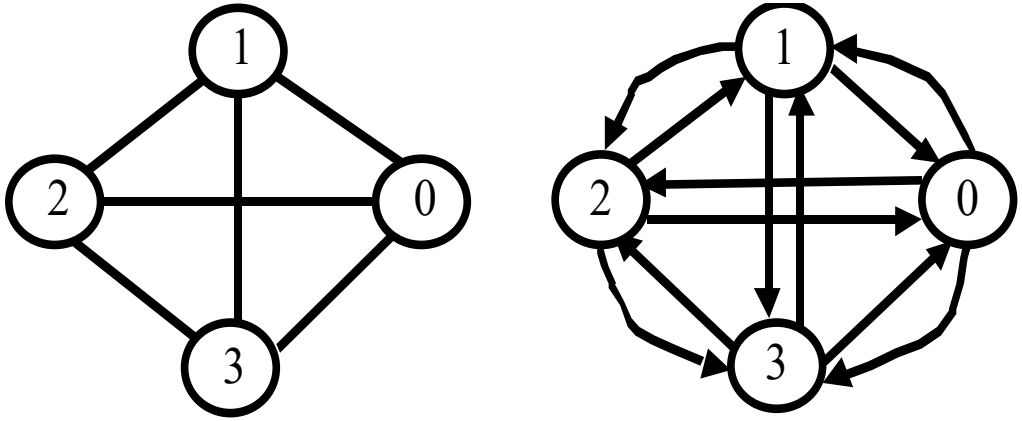
所谓树形结构,该结构中的结点之间存在一对多的关系。其特点是每个结点最多只有一个前驱,但可以有多个后继,可以有多个终端结点。非线性结构树形结构简称为树。



UNIX文件系统的系统结构图



所谓图形结构,该结构中的结点之间存在多对多的关系。其特点是每个结点的前驱和后继的个数都可以是任意的。因此,可能没有开始结点和终端结点,也可能有多个开始结点、多个终端结点。图形结构简称为图。



2. 存储结构（物理结构）

逻辑结构在计算机中的存储映象，是逻辑结构在计算机中的实现，它包括数据元素的表示和关系的表示。

- 顺序存储结构
- 非顺序存储结构（链式存储结构）
- 索引存储结构
- 散列存储结构

例如　用顺序存储法和链式存储法表示下面的学生表。

学号	姓名	性别	班号
1	张斌	男	9901
8	刘丽	女	9902
34	李英	女	9901
20	陈华	男	9902
12	王奇	男	9901
26	董强	男	9902
5	王萍	女	9901

用顺序存储法存放学生表的结构体定义为：

```
struct  Stud {
        int no;                 /*学号*/
        char name[8];    /*姓名*/
        char sex[2];       /*性别*/
        char class[4];    /*班号*/
}  Studs[7]={
        {1, “张斌” , “男” , “9901” },
        …,
        {5,"王萍","女","9901"}
};
```

结构体数组Studs各元素在内存中按顺序存放,即第i(1≤i≤6)个学生对应的元素Studs[i]存放在第i+1个学生对应的元素Studs[i+1]之前,Studs[i+1]正好在Studs[i]之后。

1	张斌	男	9901
8	刘丽	女	9902
34	李英	女	9901
20	陈华	男	9902
12	王奇	男	9901
26	董强	男	9902
5	王萍	女	9901

用链式存储法存放学生表的结构体定义为：

```
typedef struct node
{
    int no;                 /*学号*/
    char name[8];           /*姓名*/
    char sex[2];            /*性别*/
    char class[4];          /*班号*/
    struct node *next;      /*指向下个学生的指针*/
}  StudType;
```

学生表构成的链表如右图所示。其中的head为第一个数据元素的指针。

学生表构成的链表

链式存储法的缺点：

- 存储空间占用大
- 无法随机访问

链式存储法的优点：

- 便于修改（插入、删除、移动）

逻辑结构与存储结构的关系

- 存储结构是逻辑结构用计算机语言的实现；
- 如何用计算机语言表示数据元素之间的各种关系。

存储结构是逻辑关系的映象与元素本身的映象。逻辑结构是数据结构的抽象，存储结构是数据结构的实现，两者综合起来建立了数据元素之间的结构关系。

3. 数据的运算

– 就是施加于数据的操作，如查找、添加、修改、删除等。在数据结构中运算不仅仅实加减乘除这些算术运算，它的范围更为广泛，常常涉及算法问题。

举例：线性表的初始化、查找、插入、删除操作等

- **算法的设计取决于选定的数据（逻辑）结构，而算法的实现依赖于采用的存储结构。**
- **抽象运算定义在逻辑结构上，而实现在存储结构上。**

- 数据结构的内容可归纳为三个部分：逻辑结构、存储结构和运算集合。按某种逻辑关系组织起来的一批数据，按一定的映象方式把它存放在计算机的存储器中，并在这些数据上定义了一个运算的集合， 就叫做数据结构。

数据类型

在用高级程序语言编写的程序中,必须对程序中出现的每个变量、常量或表达式,明确说明它们所属的**数据类型**。

不同类型的变量,其所能取的值的范围不同,所能进行的操作不同。**数据类型**是一个值的集合和定义在此集合上的一组操作的总称。

如C/C++中的int就是整型数据类型。它是所有整数的集合(在16位计算机中为 - 32768到32767的全体整数)和相关的整数运算(如 + 、 - 、 * 、 / 等)。

(2)抽象数据类型

抽象数据类型(Abstract Data Type简写为ADT)指的是用户进行软件系统设计时从问题的数学模型中抽象出来的逻辑数据结构和逻辑数据结构上的运算，而不考虑计算机的具体存储结构和运算的具体实现算法。

一个抽象数据类型的模块通常应包含定义、表示和实现三部分。

抽象数据类型的形式定义：抽象数据类型是一个三元组

(D, S, P)

其中：D是数据对象

S是D上数据关系的有限集

P是对D的基本操作的有限集

ADT 抽象数据类型名
{
数据对象的定义
数据关系的定义
基本操作的定义
} *ADT 抽象数据类型名*

其中,数据对象、数据关系用伪码描述;

基本操作定义格式为

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〉

- **基本操作有两种参数：赋值参数只为操作提供输入值；引用参数以&打头， 除可提供输入值外，还将返回操作结果。**
- **“初始条件”描述了操作执行之前数据结构和参数应满足的条件，若不满足，则操作失败，并返回相应出错信息。**
- **“操作结果”说明了操作正常完成之后，数据结构的变化状况和应返回的结果。若初始条件为空，则省略之。**

例如，定义抽象数据类型“复数”

ADT Complex {

数据对象：

$D = \{e1,e2 \mid e1,e2 \in RealSet \}$

数据关系：

$R1 = \{<e1,e2> \mid$ e1是复数的实数部分，

$\mid$ e2 是复数的虚数部分 $\}$

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

$$z = \frac{(8+6i)(4+3i)}{(8+6i)+(4+3i)}$$

```
# include <iostream.h>
# include "complex.h"
  void main()
{
      … …
}
```

```
complex z1,z2,z3,z4,z;
float RealPart,ImagPart;
AssignComplex(z1,8.0,6.0);
AssignComplex(z2,4.0,3.0);
Add(z1,z2,z3);
Multiply(z1,z2,z4);
if (Division (z4,z3,z))   {
        GetReal (z, RealPart);
        GetImag (z, ImagPart);
}//if
```

ADT有两个重要特征:

数据抽象：用ADT描述程序处理的实体时，强调的是其**本质的特征、其所能完成的功能**以及它和**外部用户的接口**（即**外界使用它的方法**）

数据封装：将实体的**外部特性和其内部实现细节分离**，并且**对外部用户隐藏其内部实现细节**

抽象数据类型的表示和实现

抽象数据类型需要通过**固有数据类型**(高级编程语言中已实现的数据类型)来实现。

例如，对以上定义的复数

```
// -----存储结构的定义
typedef  struct {
    float realpart;
    float imagpart;
}complex;

// -----基本操作的函数原型说明

void AssignComplex( 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 }

1.3 算 法

1. 算法（Algorithm）的定义

Algorithm is a finite set of rules which gives a sequence of operation for solving a specific type of problem. (算法是规则的有限集合， 是为解决特定问题而规定的一系列操作。)

2. 算法的特性

(1)有穷性：有限步骤之内正常结束，不能形成无穷循环。

(2)确定性： 算法中的每一个步骤必须有确定含义，无二义性。

(3) 可行性： 原则上能精确进行，操作可通过已实现的基本运算执行有限次而完成。

(4) 输入： 有多个或0个输入。

(5) 输出： 至少有一个或多个输出。

在算法的五大特性中， 最基本的是有限性、 确定性和可行性。

```
void exam1()
{
    n=2;
    while(n%2==0)
        n=n+2;
     printf("%d\n",n);
}
```

违反了有穷性。

```
void exam2()
{
    y=0;
    x=3/y;
    printf("%d,%d\n"
            ,x,y);
}
```

违反了可行性。

算法和数据结构是两个不可分割的统一体

程序 = 数据结构 + 算法

数据结构通过算法实现操作

算法根据数据结构设计程序

算法设计的要求：

- 正确性　　正确反映需求(通过测试)
- 可读性　　有助于理解、调试和维护
- 健壮性　　完备的异常和出错处理
- ✓ 高效率与低存储的需求　　时间、空间的要求

1.4 算法描述的工具

- 描述算法的方法
  - 自然语言：优点——简单。缺点——有歧异，表达复杂思想不明晰，不能和实现方式很好结合
  - 高级程序设计语言，如Pascal, C/C++, Java等。优点——克服了自然语言的缺点，可直接执行。缺点——对部分问题的描述比较烦杂，啰嗦
  - *类语言。和高级程序设计语言类似，但是对其中一些比较烦杂的部分进行简化（原因：算法主要目的是为了清晰的表述思想）

    *举例：两个数据a, b交换空间

    自然语言：交换a, b的存储空间；

    高级语言：{ x = a; a = b; b = x; }

    类语言：a←→b; //交换空间

1.5 对算法作性能评价

- 衡量算法效率的方法主要有两大类：
  - 事后统计：利用计算机的时钟；
  - 事前分析估算：用高级语言编写的程序运行的时间主要取决于如下因素：
    - 算法；
    - 问题规模；
    - 使用语言：级别越高，效率越低；
    - 编译程序；
    - 机器；

通常，从算法中选取一种对于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执行的时间度量。

例，
```
X = X + 1 ;
```
(a)

```
for ( i = 1 ; i<=n ; i++ )
    X = X + 1 ;
```
(b)

```
for ( i = 1 ; i<=n ; i++ )
    for ( j = 1 ; j<=n ; j++ )
        X = X + 1 ;
```
(c)

基本操作重复执行的次数分别为 1，n，n^2

设算法的问题规模为n；

频度：语句重复执行的次数称为该语句的频度，记$f(n)$。

对算法各基本操作的频度求和，便可得算法的时间复杂度。

但实际中我们所关心的主要是一个算法所花时间的数量级，即取算法各基本操作的最大频度数量级。

时间复杂度：算法执行时间度量，记$T(n)=O(\ \text{maxlevel}(f(n))\ )$。

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

$$T(n) = O(\ n^3\ )$$

O的数学定义:

若T(n)和f(n)是定义在正整数集合上的两个函数，则如果存在正常数C和n_0,使得当$n \geq n_0$时，总满足$0 \leq T(n) \leq Cf(n)$,则记做T(n)=O(f(n))

- 也就是只求出T(n)的最高阶（数量级）,忽略其低阶项和常系数,这样既可简化T(n)的计算,又能比较客观地反映出当n很大时,算法的时间性能。

2个N*N矩阵相乘

```
for (i= 1; i<=n; ++i)                          n+1
    for (j= 1; j<=n; ++j){                     n(n+1)
        c[i][j] = 0;                           n^2
        for (k= 1; k<=n; ++k)                  n^2(n+1)
            c[i][j] += a[i][k] * b[k][j];      n^3
    }
```

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

$$T(n) = O(n^3)$$

（1） x=x+1 ； 其时间复杂度为O(1)， 我们称之为常量阶；

（2） for (i=1; i<= n; i++) x=x+1; 其时间复杂度为O(n)， 我们称之为线性阶；

（3） for (i=1; i<= n; i++)

for (j=1; j<= n; j++) x=x+1; 其时间复杂度为$O(n^2)$, 我们称之为平方阶。

此外算法还能呈现的时间复杂度有对数阶$O(\log_2 n)$, 指数阶$O(2^n)$等。

常见的时间复杂度，按数量级递增排序：

常数阶	$O(1)$
对数阶	$O(\log_2 n)$
线性阶	$O(n)$
线性对数阶	$O(n\log_2 n)$
平方阶	$O(n^2)$
立方阶	$O(n^3)$
……	
K次方阶	$O(n^k)$
指数阶	$O(2^n)$

例如： 下列程序段：

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

- 语句x++的执行频度为

  $n+(n-1)+(n-2)+...+3+2+1 = n(n+1)/2$

- 而该语句执行次数关于n的增长率为n^2,

  即时间复杂度为$O(n^2)$。

5） 最坏时间复杂度

有时，算法中基本操作重复执行的次数随问题的输入不同而不同，通常分析最坏情况下的时间复杂度。

例，顺序查找算法

```
Status search ( int a[ ] , int n , int e )
 {
   for ( i = 0 ; i <=n - 1 ; ++i )
       if ( e == a[i] ) return TRUE ;

   return FALSE ;
 }
```

最好 1 次比较，最坏 n 次比较，平均 (n+1)/2 次比较。

6） 算法的空间复杂度

关于算法的存储空间需求，类似于算法的时间复杂度，我们采用空间复杂度作为算法所需存储空间的量度，记作：

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

课程的基本结构分为三个部分：

第一部分： 数据结构的基本概念（第1章）。

第二部分： 基本的数据结构，包括：线性结构—线性表、栈和队列、 串、 数组（第2~5章）；非线性结构—树、 图（第6、 7章）。

第三部分： 基本技术，包括：查找技术与排序技术（第9、10章）。

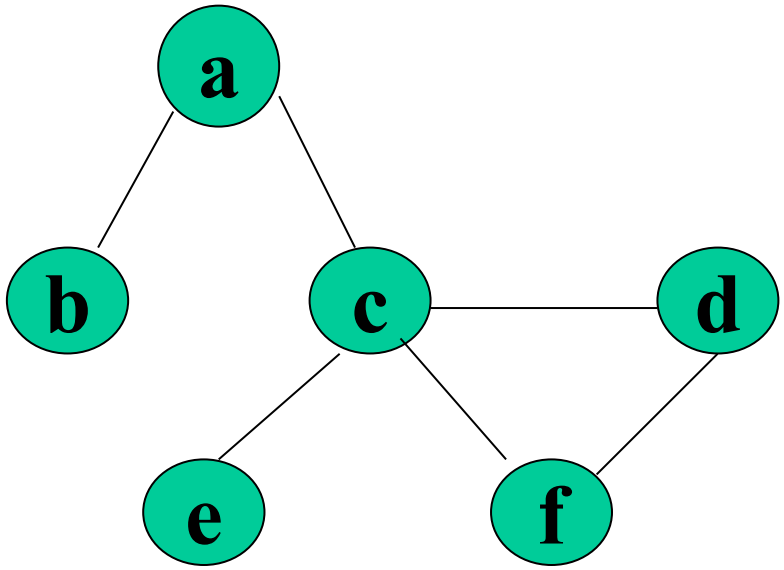
- 数据结构
  - 逻辑结构
    - 线性结构 （线性表、栈和队列、串）
    - 非线性结构 （广义表、树、图、文件）
  - 存储结构
    - 顺序存储结构
    - 链式存储结构
    - 索引存储结构
    - 散列存储结构
  - 数据运算 （算法）

- 作业《数据结构题集》

  P_7：（3、9、17）

- 练习

例1：设逻辑结构图如下，试给出其数据结构表示。



数据结构定义为：

Data_Structure = (D,S)

其中 D={a,b,c,d,e,f}

S={R}

R={(a,b),(a,c),(c,d),(c,e),(c,f),(d,f)}

例2：设n为正整数，利用大“O”记号，将下列程序段中x++的执行时间表示为n的函数，写出其时间复杂度。

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

$$f(n)=\sum_{i=1}^{n-1}\sum_{j=1}^{n-i}1=\sum_{i=1}^{n-1}(n-i)=(n-1)+(n-2)+\ldots+1=\frac{n(n-1)}{2}$$

$$T(n)=O(n^2)$$

例3：试说明下列计算过程是否是一个算法？

（1）开始；

（2）n= 0;

（3）n++;

（4）重复（3）；

（5）结束；

不具备算法的有穷性，不是一个算法。