数 据 结 构

授课教师：王　瑜

陈一飞

制作：数据结构在线课程课题组

南京审计大学 信息工程学院

2020.09

课前的五个问题

一. 成绩的评定

平时分50%+期末50%

平时分的组成：

1. 单元测验20%（两次：ch1+ch2+ch3；ch6+ch9+ch10）
2. 纸质作业+实验报告+上机作业+考勤10%
3. 上机考试，一共四题，每题25分，每题得分只有0分和25分的区别10%
4. 雨课堂10%

二. 教学进度

参见教学进度表（第七章最后教授）

课前的五个问题

三. 教材

1. 严蔚敏 编著《数据结构（C语言）》 清华大学出版社

2. 自编教材《数据结构实验指导与习题集》

四. 参考书

根据自己的需要选择教材类或者习题类或者实验类参考书

五. C语言准备

回去根据自身情况在自己机器上装VC或者codeblocks

另：尽快进QQ群（727584568）和雨课堂：学号班级姓名

如：194010717数科1班蔡玥

数据结构参考书

- 唐策善等　编著　《数据结构--用C语言描述》
  高等教育出版社
- 严蔚敏 陈文博 编著
  《数据结构及应用算法教程》　清华大学出版社
- 施伯乐等　编著　《数据结构》
  复旦大学出版社
- 李春葆　《数据结构考研指导》
  清华大学出版社

课前准备

- C语言语法
- C语言中函数部分:函数定义、参数传递、返回值等。
- 指针部分：数组和链表
- 结构体和共用体：结构体和共用体的定义、结构体数组、指向结构体类型数据的指针。

内容概要

1、了解数据结构的形成及发展过程

2、理解数据结构中的基本概念和术语

3、掌握抽象数据类型的表示和实现

4、理解算法并掌握算法的分析

1.1 什么是数据结构——数据结构的形成

计算机是处理数据的工具。但是将“一大堆杂乱无章”的数据交给计算机处理是很不明智的，结果是加工处理的效率会非常低，有时甚至根本无法进行。

于是：

人们开始考虑如何更有效地描述、表示、处理数据的问题，除了不断提高计算机技术外，很重要的一个方面是通过研究、分析数据本身的特点，利用这些特点提高数据表示和处理的效率。——数据结构

总结：

信息的表示和组织形式直接影响数据处理的效率！

一个数据结构的例子

例A：假设CPU每秒处理10^6个指令，对于输入规模为=10^8的问题，时间代价$f(n)=2n^2$的算法要运行多长时间？

操作次数：

$f(n)=f(10^8)=2\times(10^8)^2=2\times 10^{16}$

运行时间：

$2\times10^{16}/10^6=2\times 10^{10}$秒

每天有86,400秒，因此需要213,480天

即634年

一个数据结构的例子

例B：假设CPU每秒处理10^6个指令，对于输入规模为$=10^8$的问题，时间代价$f(n)=n\log_2 n$ 的算法要运行多长时间？

操作次数：

$f(n)=f(10^8)=10^8\times\log_2 10^8 = 2.66\times 10^9$

运行时间：

$2.66\times 10^9 / 10^6 = 2.66\times 10^3$秒

即44.33分钟

例1：图书馆的自动化书目检索

书目卡片

登录号：

书名：

作者名：

分类号：

出版单位：

出版时间：

价格：

例1：图书馆的自动化书目检索

线性的数据结构

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，……
……	……..

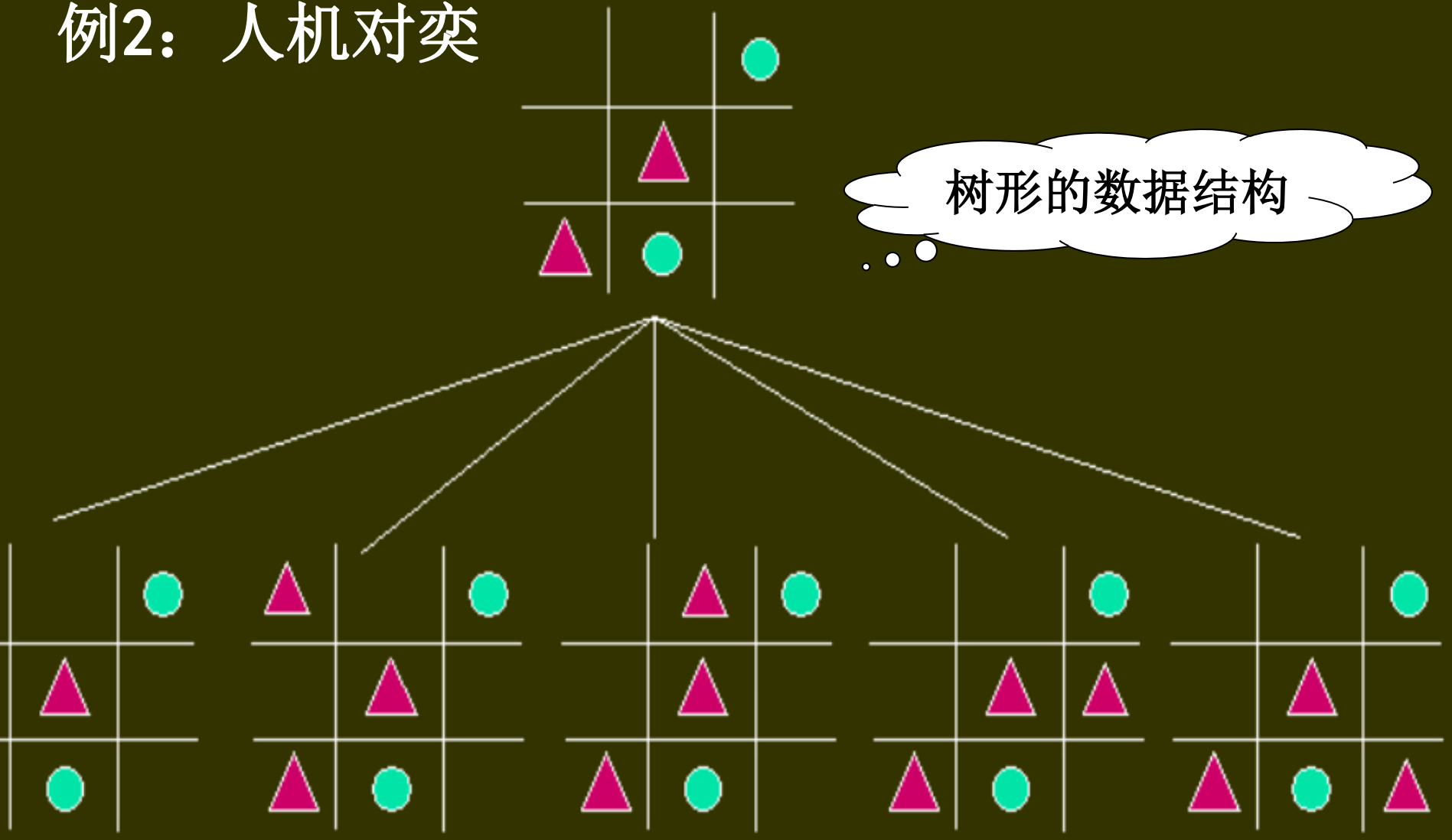
按作者名

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

按分类号

L	002, …
S	001, 003,
……	……

例2：人机对弈



井字棋对弈“树”

例3：多叉路口交通灯管理问题

例3：问题：多叉路口的交通灯管理，即在多叉路口应设置几种颜色的交通灯，以保证交通畅通（图结构）

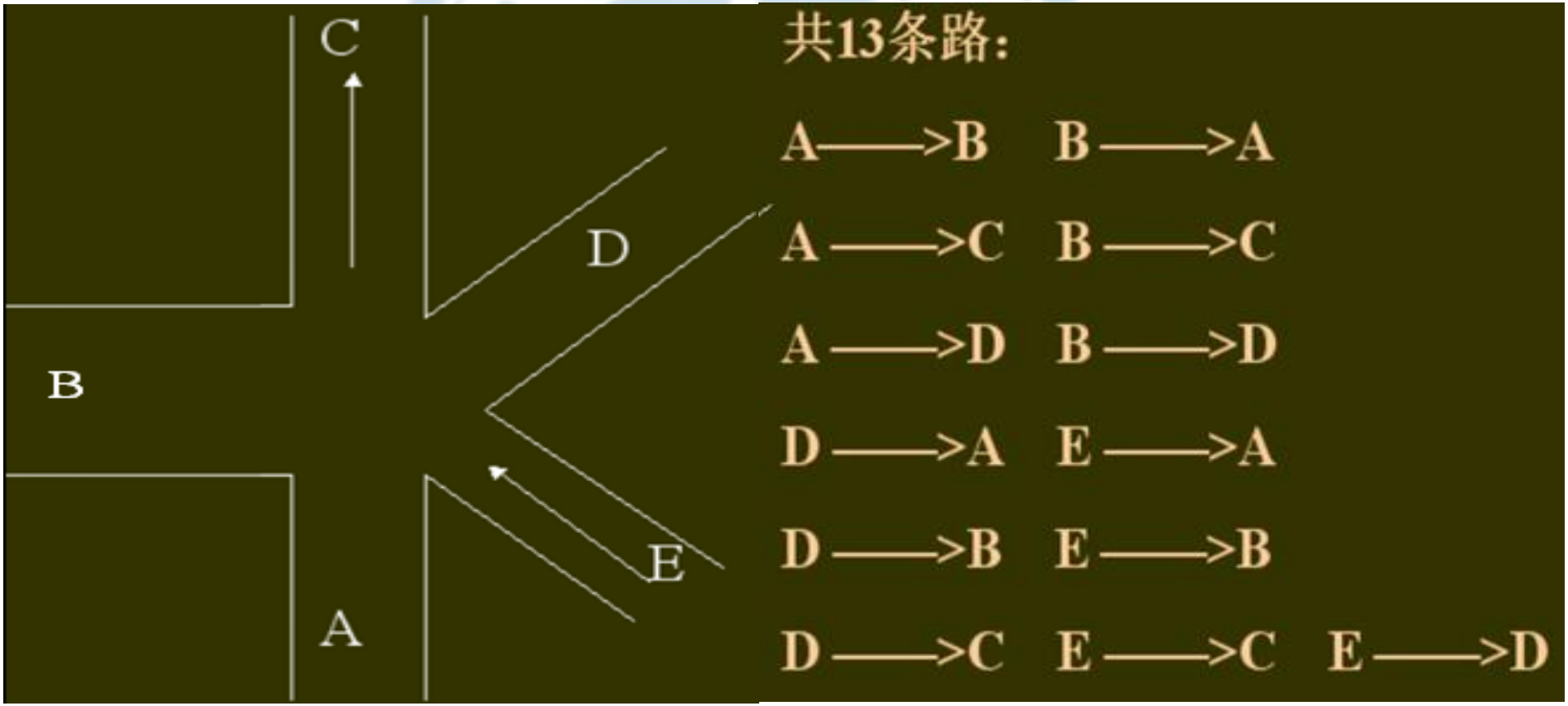
数据：路口各条路的信息。

操作：设置信号灯

（求出各个可同时通行的路的集合）。

多叉路口交通灯管理问题

问题：多叉路口的交通灯管理，即在多叉路口应设置几种颜色的交通灯，才能使车辆相互之间不碰撞，又能达到车辆的最大流量。



多叉路口交通灯管理问题

共13条路：

A——>B　B——>A

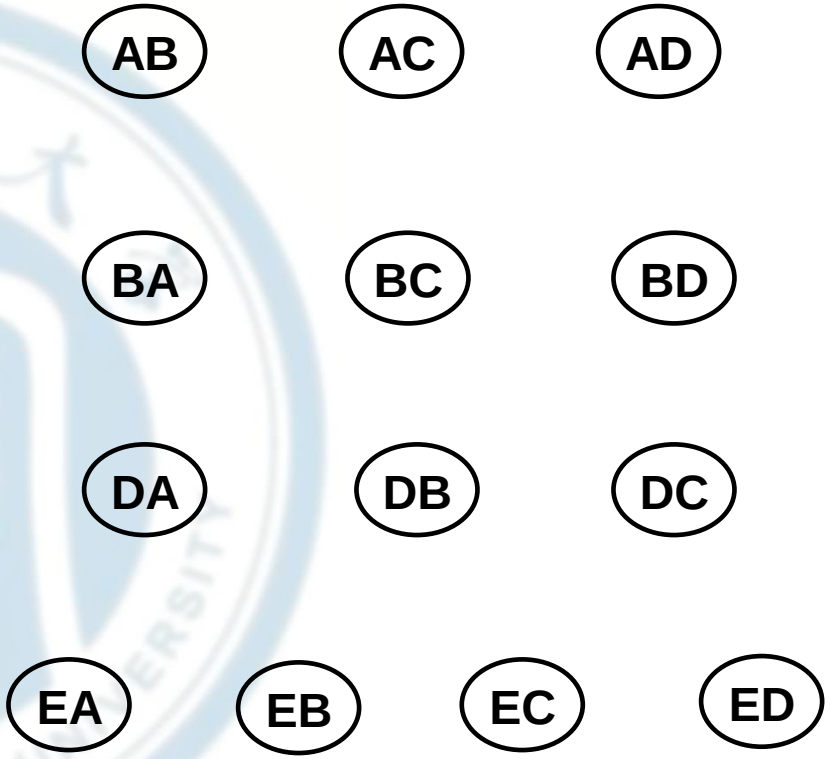
A——>C　B——>C

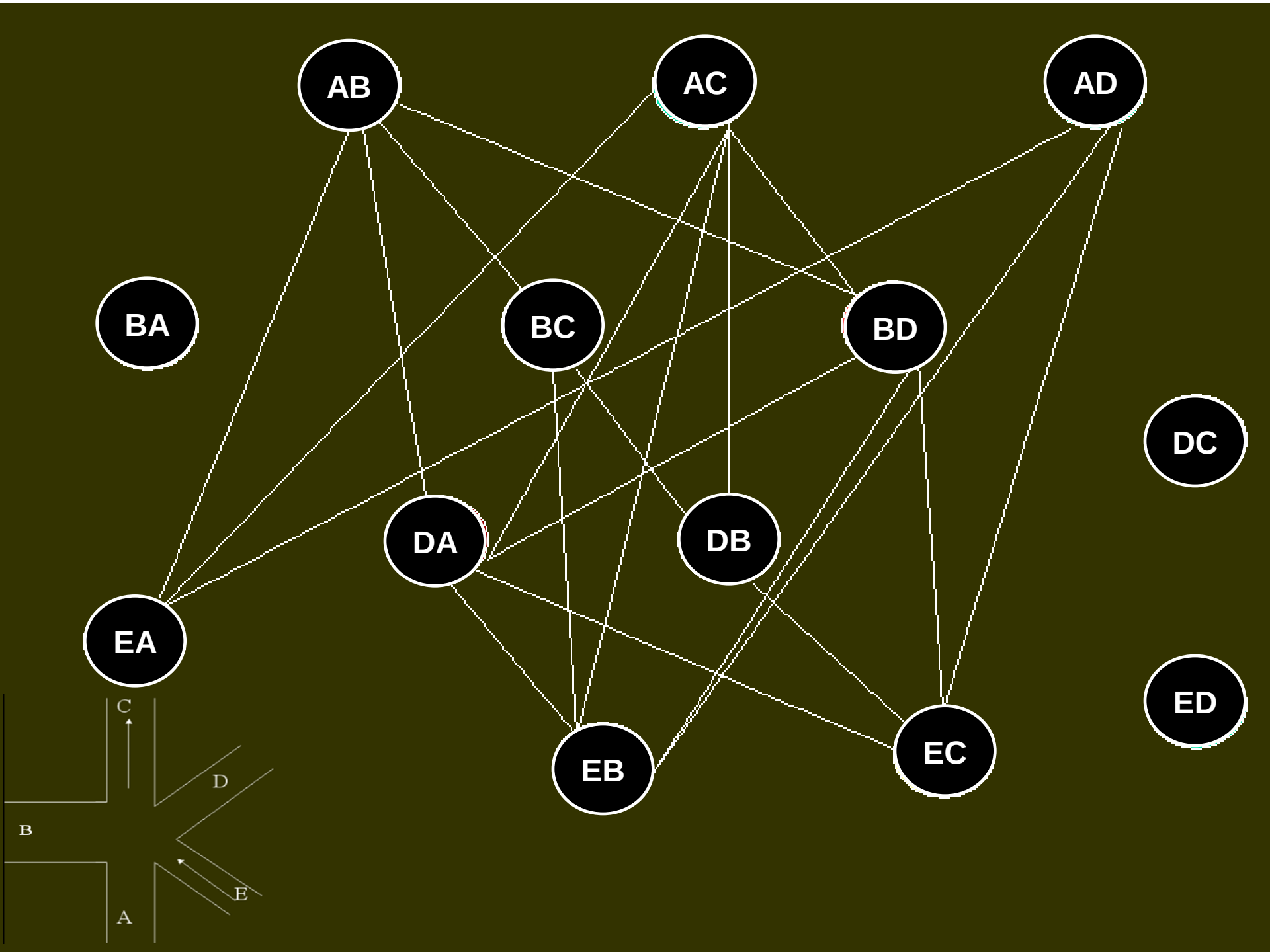
A——>D　B——>D

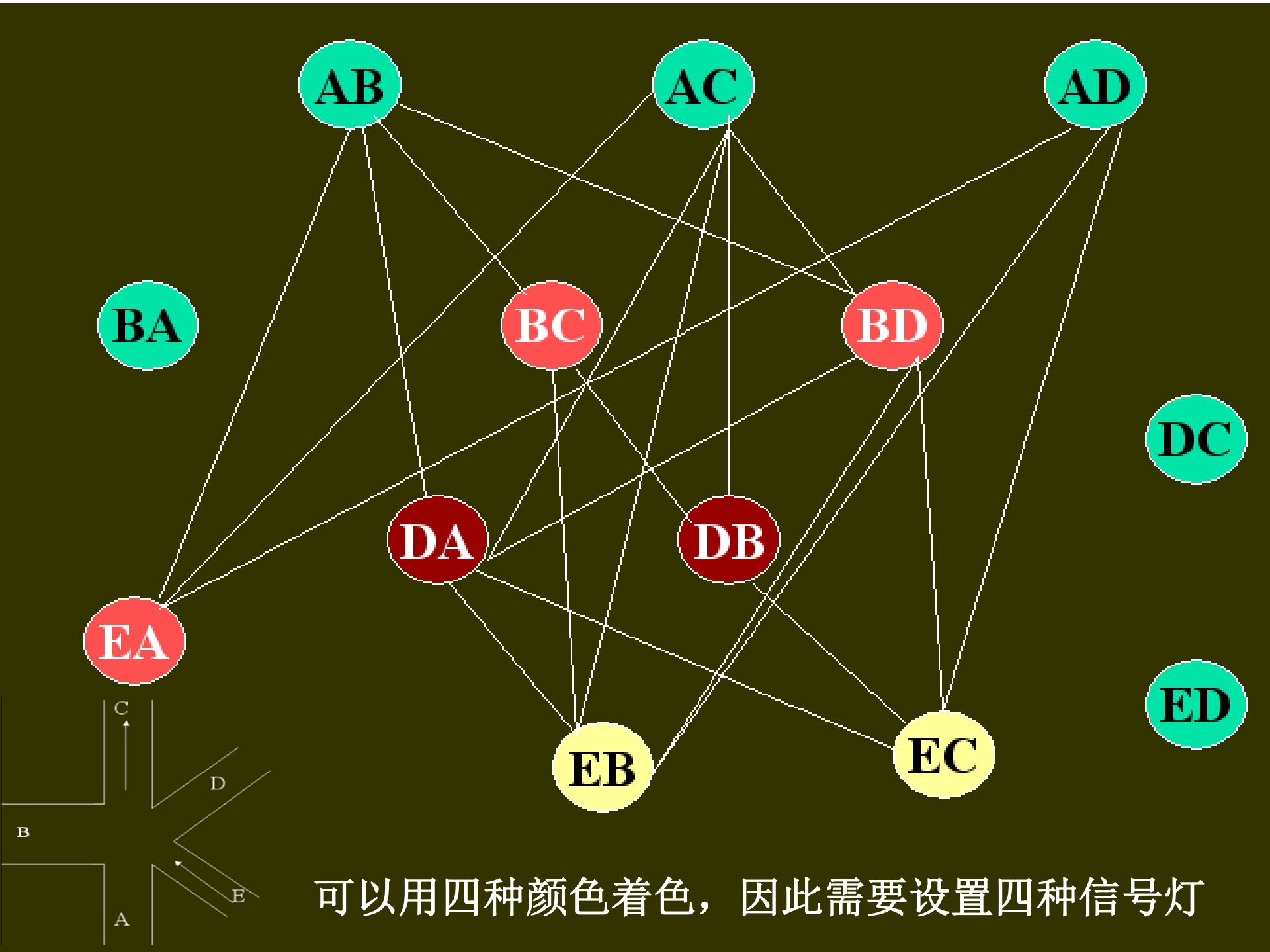
D——>A　E——>A

D——>B　E——>B

D——>C　E——>C　E——>D




AB
AC
AD
BA
BC
BD
DC
DA
DB
EA
ED
EB
EC
C
D
B
E
A



可以用四种颜色着色，因此需要设置四种信号灯

概括地说，

数据结构描述现实世界实体的数学模型（非数值计算）及其上的操作在计算机中的表示和实现。

数据结构发展概况

从时间上看：

50年代末～60年代初　萌芽期：分散于其它课程中

60年代中～70年代初　形成发展期；

68年独立成一门课程

70年代中～80年代初　完善期；

80年代中至今　完善更新期；

数据结构发展概况(2)

从内容上看：

最初：　介绍一些表处理系统；

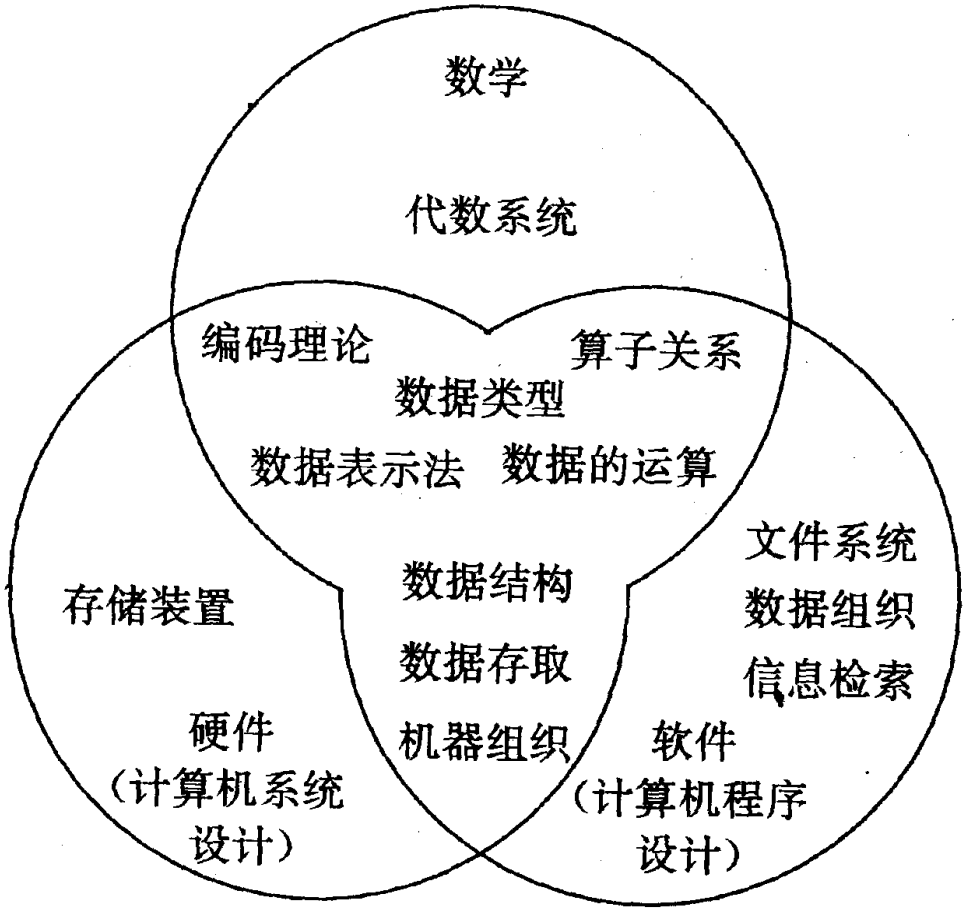
稍后：　介绍图论，尤其是表和树；

再后：　讨论离散数学的内容，集合、代数、格；

再后：　确定了三种基本数据结构及存储结构；

再后：　加入了分类、检索和文件系统；

近几年：涉及到了面向对象的内容。



在我国，80年代初才列入计算机科学教学计划，是编译原理、操作系统、数据库系统等课程的基础。

1.2 基本概念和术语

- 数据
  - 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。
  - 是计算机加工的“原料”。
  - 包括：图像、声音、数值、字符串等。

- 数据元素
  - 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。
  - 例如：5，N，记录，格局，顶点。

- 数据项:
  - 一个数据元素可由多个数据项组成。
  - 数据项是数据的不可分割的最小单位。

- 数据对象
  - 是性质相同的数据元素的集合。
  - 是数据的一个子集。
  - 例如：整数数据对象是集合N={0，±1，±2，……}
  - 字母字符数据对象是集合C={“A”，“B”，……“Z”}

- 数据结构
  - 相互之间存在一种或多种特定关系的数据元素的集合。
  - 形式化定义

    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。

数据结构可分为以下四类：

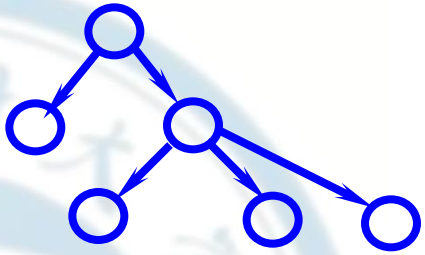
集合 

- 集合：数据元素间除“同属于一个集合”外，无其它关系。

线性结构 

- 线性结构：数据元素间存在一个对一个的关系。

树形结构



- 树形结构：数据元素间存在一个对多个的关系。

图状结构
（网状结构）

- 图形结构：数据元素间存在多个对多个的关系。

例：用图形表示下列数据结构，并指出它们是属于线性结构还是非线性结构。

(1) S=(D, S)

D={ a, b, c, d, e, f }

S={<a,e>, <b,c>, <c,a>, <e,f>, <f,d>}

解：　上述表达式可用图形表示为：

$$b \longrightarrow c \longrightarrow a \longrightarrow e \longrightarrow f \longrightarrow d$$

此结构为线性的。

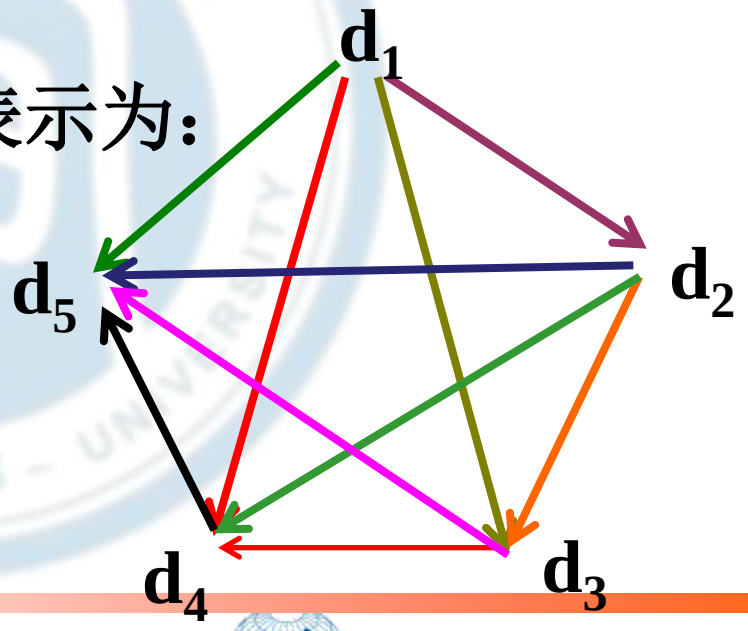
（2） S=(D, S)

D={di | 1≤i≤5}

S={<di , dj >, i<j}

解：上述表达式可用图形表示为：

结构是非线性的。



此结构为非线性的。

数据结构包括“逻辑结构” 和“物理结构”两个方面。

逻辑结构：是对数据元素之间的逻辑关系的描述，它可以用一个数据元素的集合和定义在此集合上的若干关系来表示;

物理结构：是逻辑结构在计算机中的表示和实现，故又称“存储结构” 。

物理结构的划分

- 顺序存储结构

把逻辑上相邻的结点存储在物理位置上相邻的存储单元，结点间的逻辑关系由存储单元的邻接关系来体现。

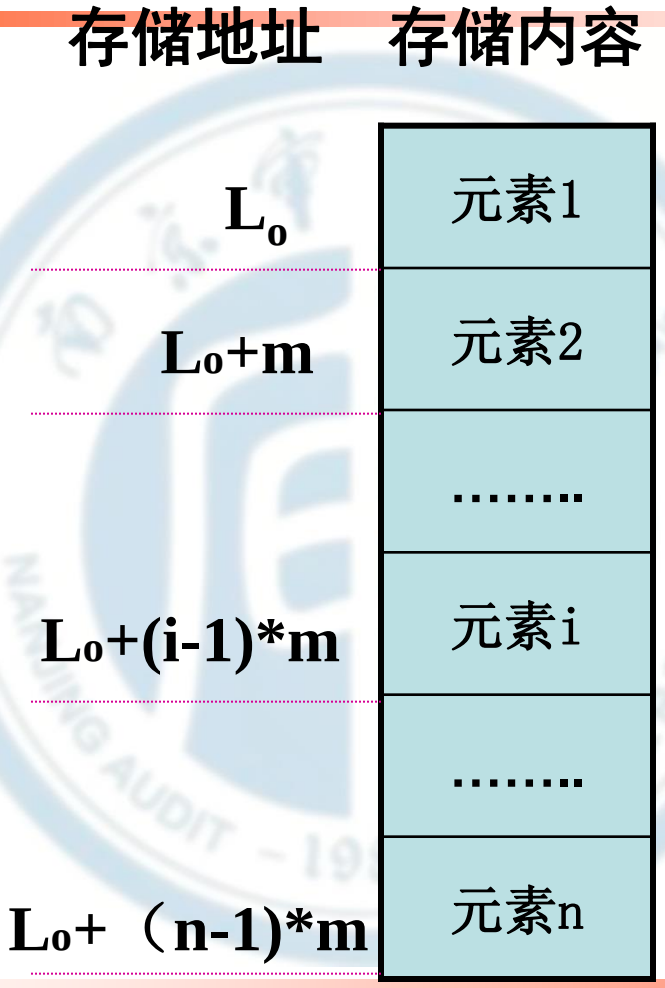
通常顺序存储结构是借助于语言的数组来描述的。

- 链式存储结构

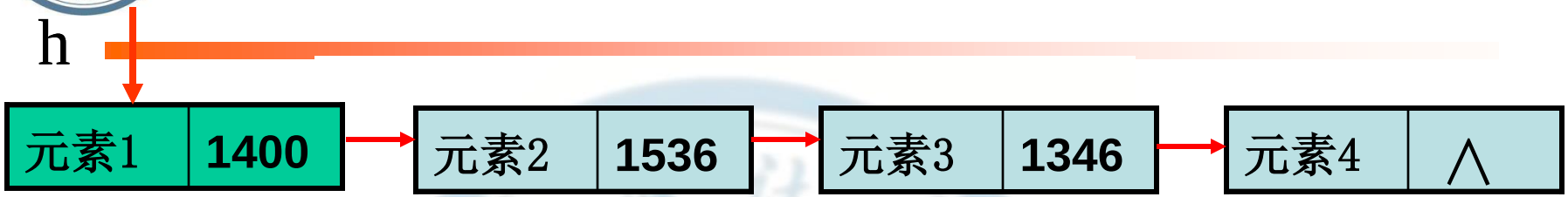
不要求逻辑上相邻的结点物理上也相邻，结点间的逻辑关系是由附加的指针字段表示的。

通常要借助于语言的指针类型来描述。

顺序存储结构



链式存储结构



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

数据的逻辑结构与存储结构密切相关

- 算法设计取决于选定的逻辑结构。
- 算法实现依赖于采用的存储结构。

数据结构的三个方面

- 数据的逻辑结构
  - 线性结构
    - 线性表
    - 栈
    - 队列
  - 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

1.3 抽象数据类型的表示与实现

- 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，明确说明他们所属的数据类型。

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

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。

- 抽象数据类型
  - 是指一个数学模型以及定义在该模型上的一组操作。
  - 抽象数据类型的形式定义：用一个三元组来表示一个抽象数据类型。

ADT = (D, S, P)

其中：D 是数据对象，

S 是 D 上的关系集，

P 是 D 的基本操作集。

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。

- **数据封装：**将实体的外部特征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。

描述抽象数据类型的形式

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

数据对象和数据关系的定义用伪码描述。

数据基本操作的定义格式：

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〉

抽象数据类型"复数"的定义

D

S

P

ADT Complex {

数据对象：D = {e1,e2 | e1,e2∈RealSet }

数据关系：R1 = {<e1,e2> | e1是复数的实部，e2是复数的虚部 }

基本操作：

InitComplex(&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)

初始条件：复数已存在。

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

伪码描述

- 预定义常量和类型

```
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
#define OVERFLOW –2
typedef int Status;
```

- 数据结构

  - 数据结构的表示（存储结构）用类型（typedef）定义来描述。

  - 数据元素类型约定为ElemType，由用户在使用时自行定义。

图书数据的表示

```
typedef struct {
   int id;
   char name[20];
   char author[20];
}ElemType;
```

- 基本操作的算法用函数描述

```
函数类型 函数名（函数参数表）{
    //算法说明
    语句序列
}//函数名
```

增加了C++语言中的引用传递方式，&表示为引用参数

利用C语言实现的"复数"类型描述

```
typedef struct {
    float realpart;
    float imagpart;
    } Complex; // 存储结构的定义
// 基本操作的函数原型说明
    void InitComplex( 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
    void DestroyComplex( Complex &Z) ;// 销毁复数 Z
    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;
    }
```

1.4 算法和算法分析

- 算法（Algorithm）

  - 是对特定问题求解步骤的一种描述，它是指令的有限序列。

算法是对特定问题求解步骤的一种描述，它是指令的有限序列。具有有穷性，确定性，可行性，输入和输出

- 算法的特性

  - 有穷性：一个算法必需总是在执行有穷步之后结束，且每一步都可在有穷时间内完成。

- 确定性：算法中每一条执行都必须有确切的含义，使算法的执行者或阅读者不会产生二义性。并且在任何条件下，算法都只有一条执行路径。
- 可行性：算法中的所有操作都必须足够基本，都可以通过已经实现的基本操作运算有限次实现之。
- 输入：有零个或多个的输入。
- 输出：有一个或多个的输出。

- 算法的描述—采用类C语言
- 算法的评价—衡量算法优劣的标准
  - 正确性(correctness)
  - 可读性(readability)
  - 健壮性(robustness)
  - 效率与低存储量

例子：两种求和的算法

```
int i, sum=0,n=100;
for(i=1;i<=n;i++)
{
    sum=sum+i;
}
printf("%d",sum);
```

```
int i, sum=0,n=100;
sum=(1+n)*n/2;
printf("%d",sum);
```

算法效率的衡量方法

- 事后统计
  - 利用计算机内记时功能。
  - 缺点：①必须先运行依据算法编制的程序；
    ②所得时间统计量依赖于硬件、软件等环境因素，掩盖算法本身的优劣；
    ③算法的测试数据设计困难。

- 事前分析估计
  - 一个高级语言程序在计算机上运行所消耗的时间取决于：

    ①依据的算法选用何种策略

    ②问题的规模

    ③编译产生代码的质量

    ④机器执行指令速度

例子：两种求和的算法

```
int i, sum=0,n=100;
for(i=1;i<=n;i++)
{
    sum=sum+i;
}
printf("%d",sum);
```

执行了2n+3次

```
int i, sum=0,n=100;
sum=(1+n)*n/2;
printf("%d",sum);
```

执行了3次

函数的渐进增长

次数	算法 A（2n + 3）	算法 A'（2n）
n = 1	5	2
n = 2	7	4
n = 3	9	6
n = 10	23	20
n = 100	203	200

可以忽略这些加法常数

函数的渐进增长（2）

次数	算法 C（4n+8）	算法 C'（n）	算法 D（$2n^2+1$）	算法 D'（n^2）
n = 1	12	1	3	1
n = 2	16	2	9	4
n = 3	20	3	19	9
n = 10	48	10	201	100
n = 100	408	100	20 001	10 000
n = 1000	4 008	1 000	2 000 001	1 000 000

与最高次项相乘的系数可以忽略

函数的渐进增长（3）

次数	算法 E（$2n^2+3n+1$）	算法 E'（n^2）	算法 F（$2n^3+3n+1$）	算法 F'（n^3）
n = 1	6	1	6	1
n = 2	15	4	23	8
n = 3	28	9	64	27
n = 10	231	100	2 031	1 000
n = 100	20 301	10 000	2 000 301	1 000 000

最高次项的指数大的，函数随着n的增长也快，次要项可以忽略。

算法语句频度和时间复杂度

在已证明算法正确性的前提下，评价算法的好坏主要是关注算法在时间和空间上性能的优劣。

算法时间性能的分析是通过计算算法时间复杂度实现的，其关键就是计算算法的执行时间。一个算法的执行时间，就是算法中每条语句的执行时间的总和。

但是在算法实际运行过程中，每次执行所耗费的时间会受到诸如问题规模、输入特性和具体硬件环境等各种外界因素的影响，想得到一个绝对准确的执行时间，几乎不可能。为此，在计算算法执行时间时，一般都忽略硬件及环境因素，并且假设每次执行时，硬件条件和环境条件都是完全一致的，每条语句执行一次所需的时间均是单位时间。

算法语句频度和时间复杂度

算法中一条语句的执行时间取决于该语句的执行次数和执行一次所需的时间。语句执行次数被称为语句频度，执行一次的时间被假设为单位时间，因此算法的执行时间就可以看作是该算法中所有语句的语句频度之和。

当问题规模很大时，精确的计算T(n)是很难实现而且也是没有必要的，对于算法时间性能的分析无需非要得到时间复杂度T(n) 的精确值，它的变化趋势和规律也能清楚地反映算法的时间耗费. 基于此，引入了渐进时间复杂度作为时间性能分析的依据，它的含义就是: 在问题规模n趋于无穷大时算法时间复杂度 T(n)的渐进上界，即函数T(n)的数量级阶。算法时间复杂度和渐进算法时间复杂度在实际的算法分析过程中是不予区分的，渐进时间复杂度可以简称为时间复杂度，记为T(n)=O(f(n))。其中，O表示数量级(阶)。函数f(n)一般是算法中最大的语句频度。

例1　计算下面程序段的时间复杂度

```
for（a=0;a<m;a++）
    x=x+1;
```

step 1　找出基本操作，确定规模 m

1、基本操作就是组成算法的操作，这些操作执行完的时候就是算法结束的时候，一般情况下，我们选取最深层循环内的语句所描述的操作作为基本操作，本题中x=x+1为基本操作；

2、确定规模，由循环条件 a<m 可以知道，循环循环执行的次数，即基本操作执行的次数和参数 m 有关，因此参数 m 就是我们所说的规模 m ；

```
for（a=0;a<m;a++）
    x=x+1;
```

step 2　计算出 m的函数 f（m）

a的初值为0，每次自增1， x=x+1语句执行一次，则 a自增 m 次后循环结束

即f(m)=m

step 3　计算算法的时间复杂度T(m)

T(m)＝O(f(m))=O(m)

例2 计算下面程序段的时间复杂度

```
void fun(int n)

{  int i=1,j=100;

   while (i<n)

   {  j++;

      i+=2;

    }

}
```

step 1 找出基本操作，确定规模 n

step 2 计算出 n的函数 f（n）

2、基本操作就是循环算法的操作，以i的初值为1，每次自增2，假设i自增m次后循环结束，则i最后的值为1+2×m，因此有1+2×m+k=n（其中k为常数），解得m=(n-1-k)/2

本操作，本题中j++；和i+=2；都可以作为基本操作；

即f(n)=(n-1-k)/2,可以发现其中增长最快的项是n/2，因此时间复杂度

T(n)=O(n)

例3　计算下面程序段的时间复杂度

```
void  fun(int  n)
{   int  i=0;s=0;
    while  (s<n)
    {   i++;
        s=s+i;
     }
}
```

step 1　找出基本操作，确定规模 n

step 2　计算出 n的函数 f（n）

- **常量阶**

  将x自增看成是基本操作，则语句频度为1，即时间复杂度为O(1),如果将s=0也看成是基本操作，则语句频度为2，其时间复杂度仍为O(1)，即常量阶。

- **线性阶**

```
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。

- 平方阶

```
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。

- 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。因此，只要有人能将现有指数时间算法中的任何一个算法化简为多项式时间算法，那就取得了一个伟大的成就。

例4　计算下面程序段的时间复杂度

```
for（a=0;a<m;a++）
 {
  b=a;
   while（b>=2）
  b=b/2;
 }
```

step 1　找出基本操作，确定规模 m

1、基本操作就是组成算法的操作，这些操作执行完的时候就是算法结束的时候，一般情况下，我们选取最深层循环内的语句所描述的操作作为基本操作，本题中b=b/2为基本操作；

2、确定规模，由循环条件 a<m 可以知道，循环循环执行的次数，即基本操作执行的次数和参数 m 有关，因此参数 m就是我们所说的规模 m ；

```
for（a=0;a<m;a++）
 {
  b=a;
    while（b>=2）
  b=b/2;
 }
```

step 2　计算出 m的函数 f（m）

设语句频度是f(m),

则：2^f(m)<=m-2;f(m)<=mlog_2(m-2)取最大值

f(m)=m log_2(m-2)

step 3　计算算法的时间复杂度T(m)

T(m)＝O(f(m))=O(m log_2m)

例5　设n为正整数。试确定下面程序段中前置以记号@的语句的频度：

```
i=1; k=0;
while(i<=n-1){
  @  k += 10*i;
    i++;
}
```

step 1　找出基本操作，确定规模 n

1、本题中需要分析的基本操作为语句k += 10*i;

2、确定规模，由循环条件 i<=n-1 可以知道，循环循环执行的次数，即基本操作执行的次数和参数 n 有关，因此参数 n 就是我们所说的规模 n ；

```
i=1; k=0;
while(i<=n-1){
  @  k += 10*i;
     i++;
}
```

step 2　计算出 语句频度f（n）

i的初值为1，每次自增1， k += 10*i语句执行一次，当i>n-1循环结束，则i自增n-1次循环结束，k += 10*i语句执行n-1次。

即f(n)=n-1

例6 设n为正整数。试确定下面程序段中前置以记号@的语句的频度：

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

step 1 找出基本操作，确定规模 n

1、本题中需要分析的基本操作为语句 x += delta ;

2、确定规模，由循环条件 i<=n 可以知道，循环循环执行的次数，即基本操作执行的次数和参数 n 有关，因此参数 n 就是我们所说的规模 n ;

step 2　计算出 语句频度f（n）

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

在三重循环内，该语句的频度可通过求和公式求得：

所以**f(n)**= $\sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=1}^{j}1=\sum_{i=1}^{n}\sum_{j=1}^{i}j=\sum_{i=1}^{n}\frac{i(i+1)}{2}=\frac{1^2+2^2+\cdots+n^2}{2}+\frac{n(n+1)}{4}$

$=\frac{n(n+1)(2n+1)}{6}\cdot\frac{1}{2}+\frac{n(n+1)}{4}=\frac{\boldsymbol{n(n+1)(n+2)}}{\boldsymbol{6}}$

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

i=1	j=1	k=1
i=2	j=1	k=1
	j=2	k=1
		k=2
		⋮
i=n	j=1	k=1
	⋮	⋮
	j=n	k=1
		k=2
		⋮
		k=n

$$f(n)=\underline{1}+\underline{1+2}+\underline{1+2+3}+\cdots\cdots$$

$$+\underline{1+2+3+\cdots\cdots+n}$$

$$=1+3+6+\cdots\cdots+\frac{n(n+1)}{2}$$

$$=\frac{1^2+2^2+\cdots+n^2}{2}+\frac{1+2+\cdots+n}{2}$$

$$=\frac{n(n+1)(2n+1)}{6}\cdot\frac{1}{2}+\frac{n(n+1)}{4}$$

$$=\frac{n(n+1)(n+2)}{6}$$

算法分析举例

分析下面程序段的时间复杂性

P1：x=x+1;　　O(1)

P2：for(i=1;i<=n;i++)

　　x=x+1;　　O(n)

算法分析举例(2)

P3：for(i=1;i<=n;i++)

for(j=1;j<=n;j++) $O(n^2)$

x=x+1;

P4：for(i=1;i<=n;i++)

for(j=1;j<=i;j++) $O(n^2)$

x=x+1;

算法分析举例(3)

P5:

```
int count =1;
while (count<=n)
{
    count=count*2;
 }
```

$O(\log_2 n)$

算法分析举例(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。例如：

```
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:$O(n^2)$
- 最坏时间复杂度$O(n^2)$
- 除特别指明外，本书以后章节中讨论的时间复杂度均指最坏情况下的时间复杂度。

算法分析举例(5)

```
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;
        for( k=i+1; k<n; ++k )
            if(a[k]<a[j]) j=k;
        if( j!=i ) { w=a[j]; a[j]=a[i]; a[i]=w;}
    } // select_sort
```

基本操作

$$\sum_{i=0}^{n-2}(n-i-1)=\frac{n(n-1)}{2}=\frac{n^2}{2}-\frac{n}{2}$$

算法的时间复杂度为$O(n^2)$

算法分析举例(6)

```
void Mult_matrix(int c[][],int a[][],int b[][]，int n)
   {// a、b和c均为n阶方阵，且c是a和b的乘积
       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];
          }
    }// Mult_matrix
```

基本操作

算法的时间复杂度为$O(n^3)$

空间复杂度S(n)=O(g(n))

- 算法的存储量指的是算法执行过程中所需的最大存储空间。记作:

$$S(n)=O(g(n))$$

其中n为问题的规模(或大小)

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g(n)的增长率相同。

算法的存储量包括以下三部分：

(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。

若所需额外空间相对于输入数据量来说是常数，则称此算法为原地工作。

本章小结(一)

- 数据是计算机操作对象的总称，它是计算机处理的符号的集合，集合中的个体为一个数据元素。
- 数据元素可以是不可分割的原子，也可以由若干数据项合成，因此在数据结构中讨论的基本单位是数据元素，而最小单位是数据项。
- 数据结构是由若干特性相同的数据元素构成的集合，且在集合上存在一种或多种关系。由关系不同可将数据结构分为四类：线性结构、树形结构、图状结构和集合结构。

本章小结(二)

数据的存储结构是数据逻辑结构在计算机中的映像，由关系的两种映像方法可得到两类存储结构：一类是顺序存储结构；另一类是链式存储结构

- 数据的存储结构是数据逻辑结构在计算机中的映象，由关系的两种映象方法可得到两类存储结构：一类是顺序存储结构；另一类是链式存储结构。
- 抽象数据类型是一个数学模型以及定义在该模型上的一组操作。抽象数据类型的三大要素为数据对象、数据关系和基本操作，同时数据抽象和数据封装是抽象数据类型的两个重要特性。