数据结构

计算机学院　肖明军

Email: xiaomj@ustc.edu.cn

http://staff.ustc.edu.cn/~xiaomj

助教联系方式

- 李达宇
  - ldy200987@mail.ustc.edu.cn
  - 18326056708
- 侯楠
  - andongniao12@mail.ustc.edu.cn
  - 18756517402
- 李达天
  - li200005@mail.ustc.edu.cn
  - 18078908299

765999077

教材和参考书

- 教材：
  - 《数据结构》(第2版)
  - 编著：严蔚敏，李冬梅，吴伟民
  - 出版社：人民邮电出版社

- 参考书：
  - 《数据结构》，严蔚敏，清华大学出版社
  - 《数据结构——用面向对象方法与C++描述》，殷人昆等，清华大学出版社
  - 《算法艺术与信息学竞赛》，刘汝佳，黄亮清华大学出版社

考核方式

- 平时成绩：30-40%
  - 作业：15-20%
  - 上机：15-20%
- 期末成绩：60-70%（闭卷笔试）

为什么要学习数据结构

- 编程基础
- 计算机及相关专业考研考博课程
- 计算机等级考试课程
- 程序员考试课程

课程学习指导

课程特点：内容抽象、概念性强、内容灵活、不易掌握

- 1.提前预习、认真听课、按时完成书面及上机作业
- 2.注意先修课程的知识准备
  - ✓离散数学、C语言
- 3.注意循序渐进：
  - ✓基本概念、基本思想、基本步骤、算法设计
- 4.注意培养算法设计的能力
  - ✓理解所讲算法、对此多做思考：若问题要求不同，应如何选择数据结构，设计有效的算法

第1章 绪论

第1章　绪论

教学目标

1. 了解数据结构研究的主要内容
2. 掌握数据结构中涉及的基本概念
3. 掌握算法的时间、空间复杂度及其分析的简易方法

教学内容

1.1 数据结构的研究内容

1.2 基本概念和术语

1.3 抽象数据类型的表示与实现

1.4 算法和算法分析

1.1 数据结构的研究内容

📖N.沃思（Niklaus Wirth)教授提出：

程序=算法+数据结构

📖 电子计算机的主要用途：

☞早期：

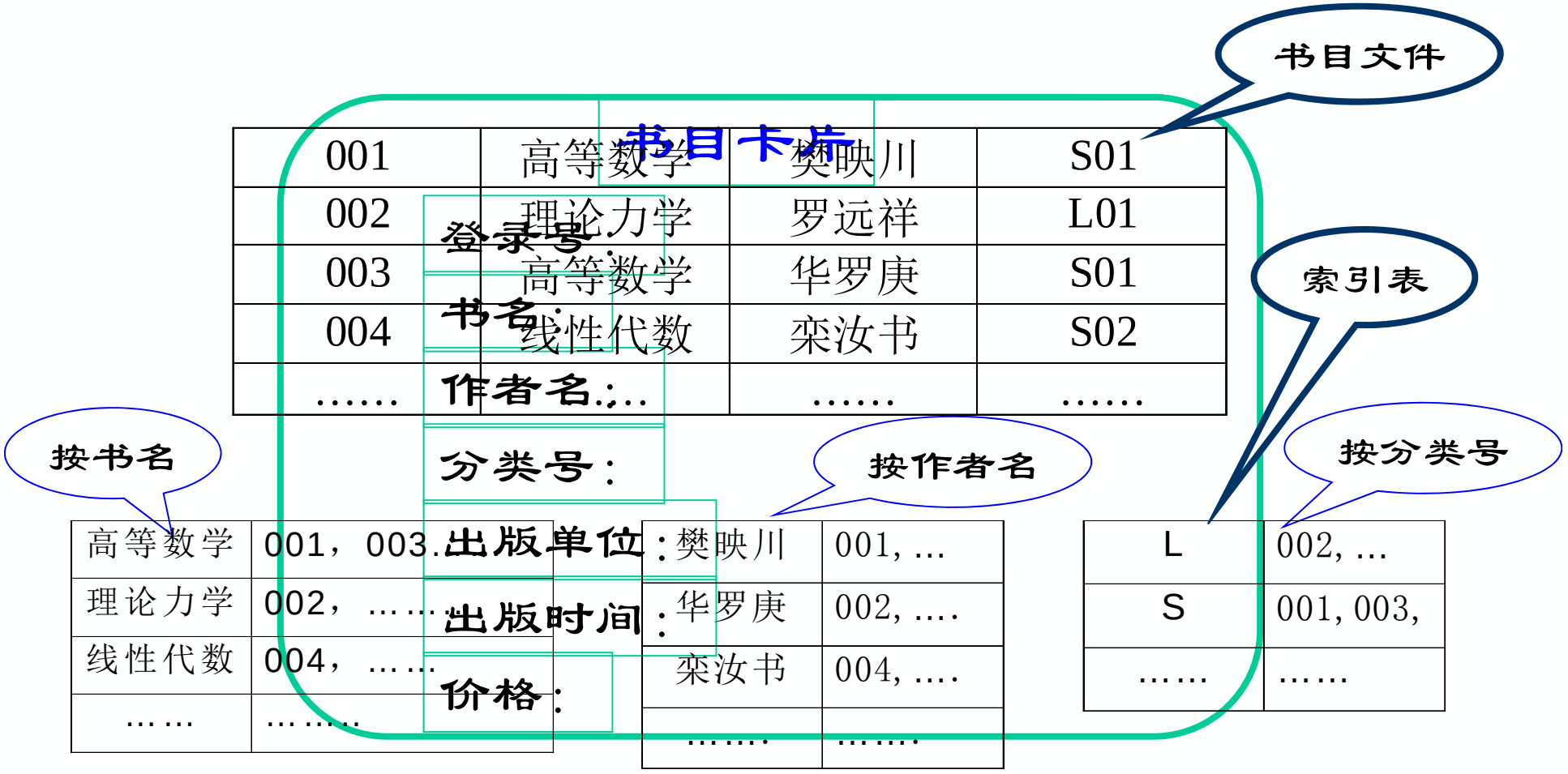
主要用于数值计算

☞后来：

处理逐渐扩大到非数值计算领域，能处理多种复杂的具有一定结构关系的数据

书目自动检索系统

线性表



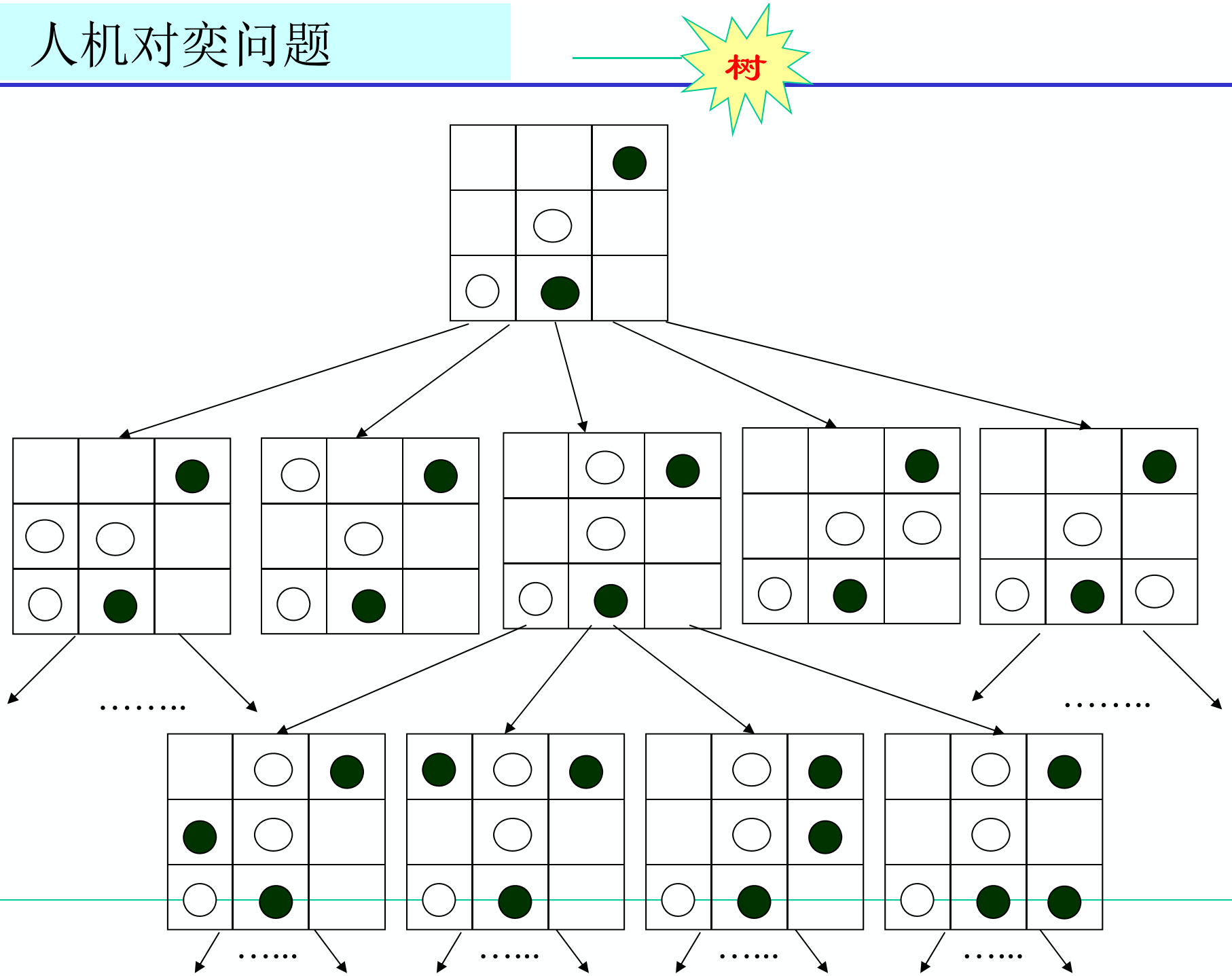
001	高等数学	樊映川	S01
002	理论力学	罗远祥	L01
003	高等数学	华罗庚	S01
004	线性代数	栾汝书	S02
……	……	……	……

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

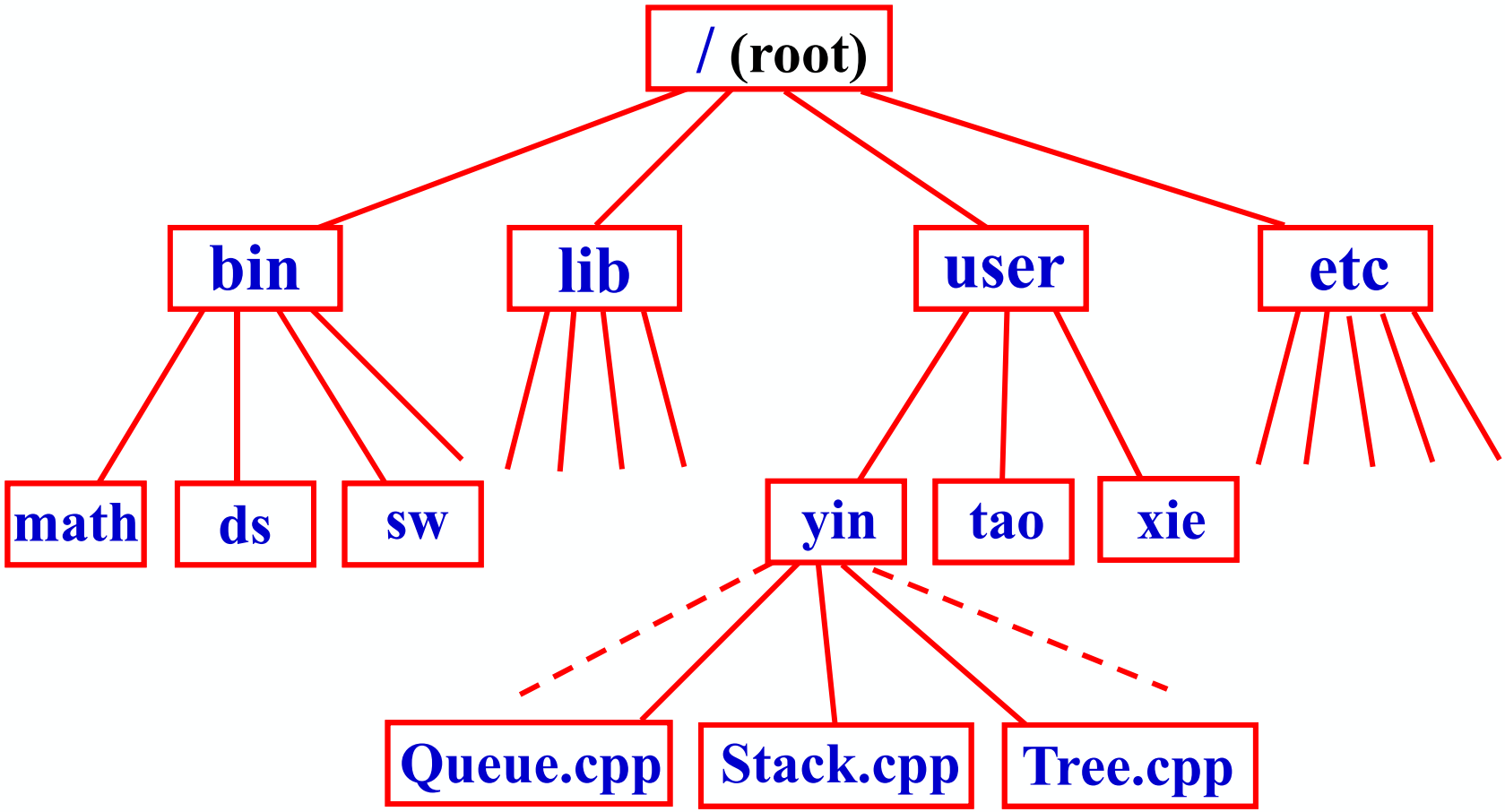
樊映川	001, …
华罗庚	002, ….
栾汝书	004, ….
…….	…….

L	002, …
S	001, 003,
……	……

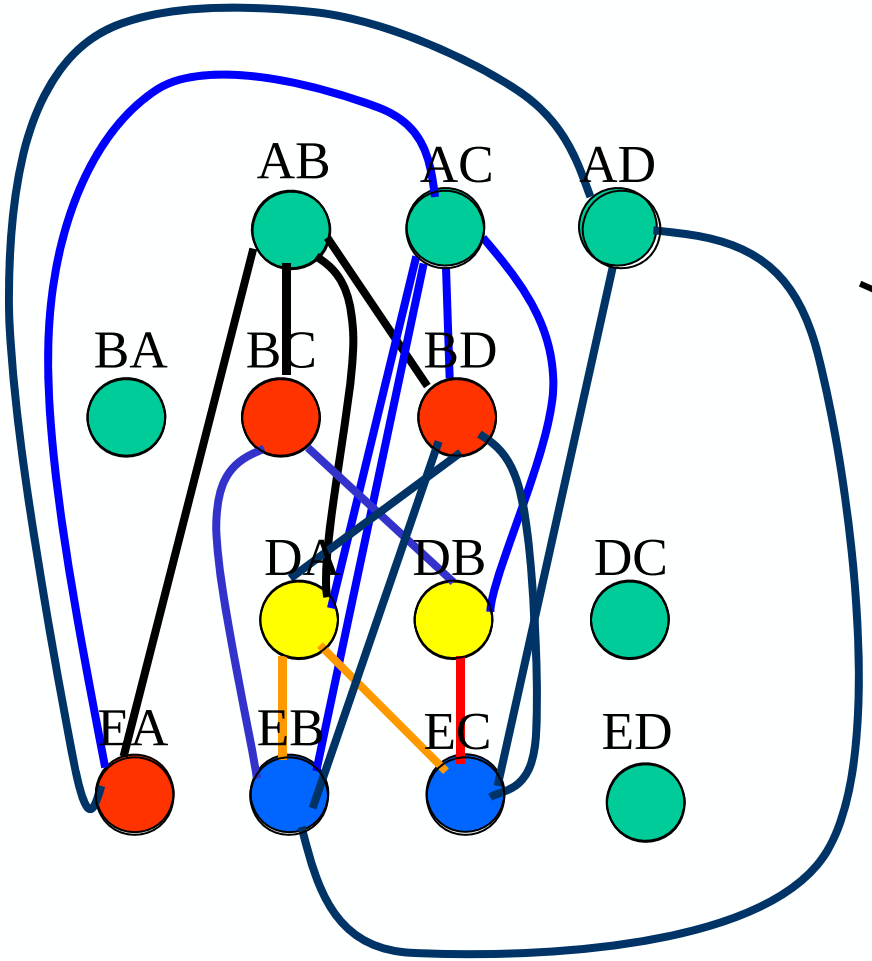
人机对奕问题

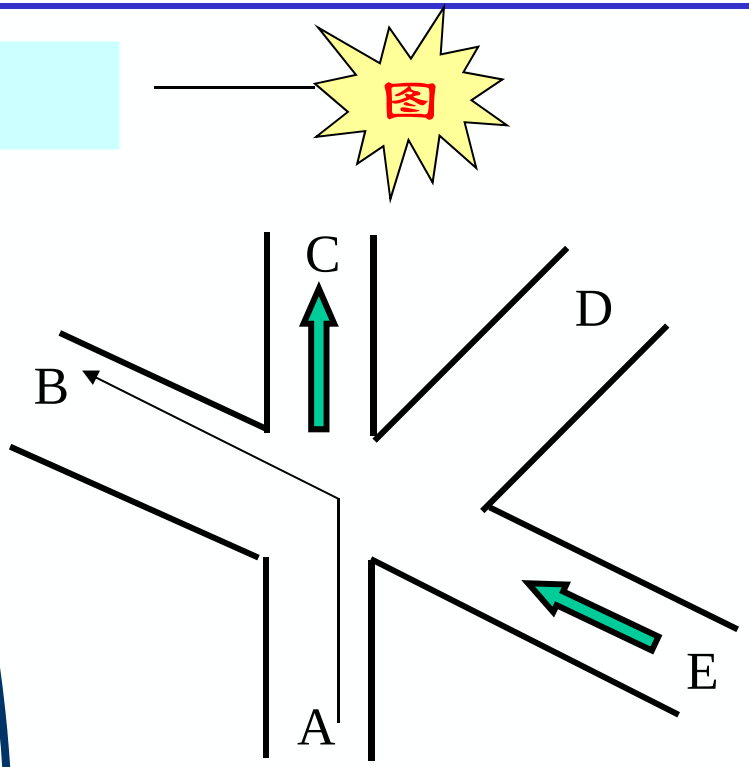


文件系统的系统结构图


树
/ (root)
bin
lib
user
etc
math
ds
sw
yin
tao
xie
Queue.cpp
Stack.cpp
Tree.cpp

多叉路口交通灯管理问题





顶点：一条通路

连线：不能同时通行

染色：有连线的两个顶点不能具有相同颜色

📖 求解非数值计算的问题：

设计出合适的数据结构及相应的算法

即：考虑对相关的各种信息如何表示、组织和处理？

数据结构的研究内容为：

研究非数值计算的程序设计问题中计算机的操作对象以及它们之间的关系和操作。

- 编写解决实际问题的程序的一般过程：
  - 如何用数据形式描述问题?—即由问题抽象出一个适当的数学模型;
  - 问题所涉及的数据量大小及数据之间的关系;
  - 如何在计算机中存储数据及体现数据之间的关系?
  - 处理问题时需要对数据作何种运算?
  - 所编写的程序的性能是否良好?
- 上面所列举问题基本上由数据结构这门课程来回答

数据结构课程的形成和发展：

形成阶段：

20世纪60年代初期，“数据结构”有关的内容散见于操作系统、编译原理和表处理语言等课程。1968年，“数据结构”被列入美国一些大学计算机科学系的教学计划。

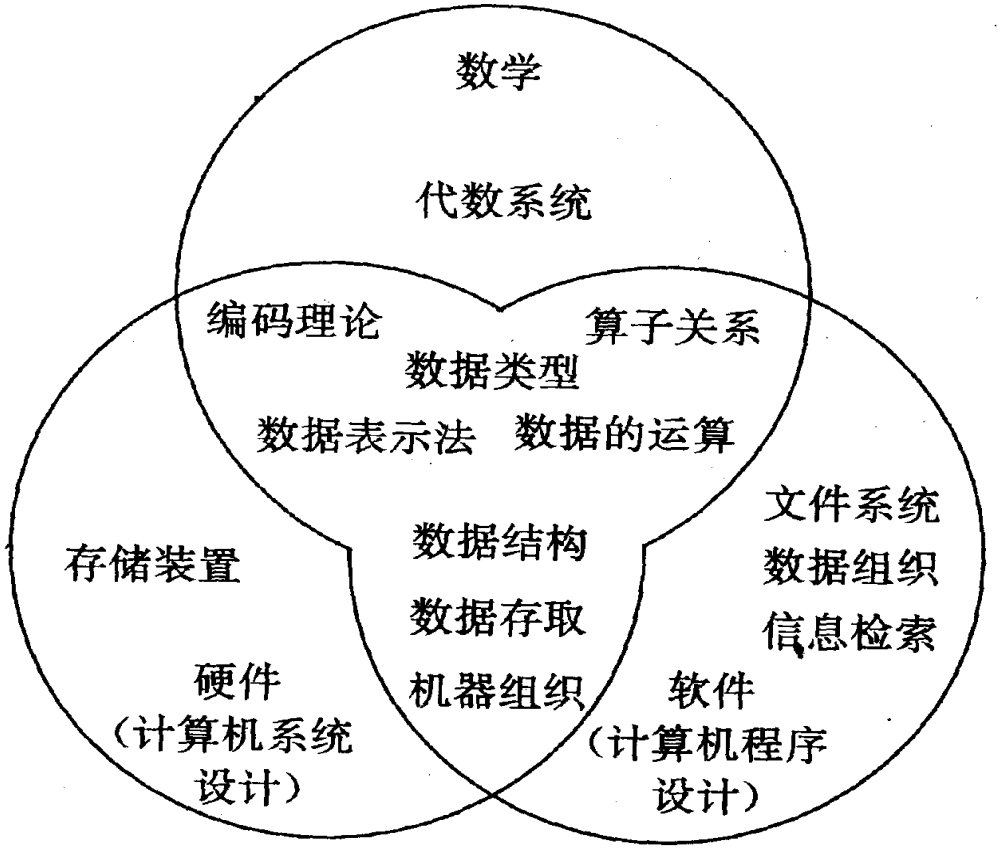
发展阶段：

数据结构的概念不断扩充，包括了网络、集合代数论、关系等“离散数学结构”的内容。

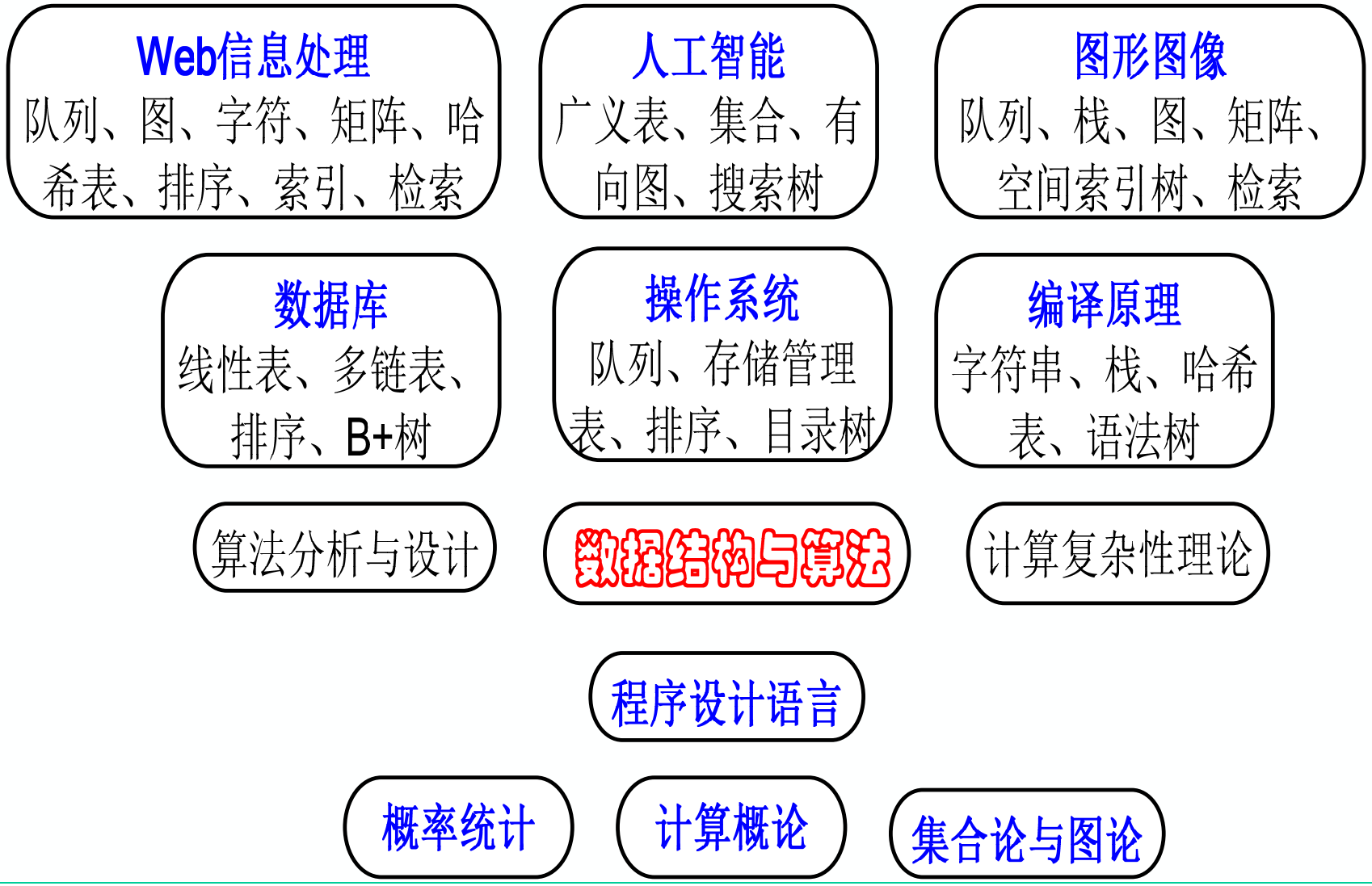
20世纪70年代后期，我国高校陆续开设该课程。

《数据结构》所处的地位：

介于数学、计算机硬件和计算机软件三者之间的一门核心课程



数据结构在计算机学科中的地位



课程目的

- 能够分析研究计算机加工的对象的特性，获得其逻辑结构，根据需求，选择合适存贮结构及其相应的算法；
- 学习一些常用的算法；
- 复杂程序设计的训练过程，要求编写的程序结构清楚和正确易读；
- 初步掌握算法的时间分析和空间分析技术

1.2 基本概念和术语

- 1、数据（data)—所有能输入到计算机中的描述客观事物的符号的总称
  - 数值性数据
  - 非数值性数据（多媒体信息处理）
- 2、数据元素（data element）—数据的基本单位，用于完整地描述一个对象，也称结点（node）或记录（record）
- 3、数据项（data item）—有独立含义的数据最小单位，也称域(field)

三者之间的关系：数据 > 数据元素 > 数据项

例：学生表 > 个人记录 > 学号、姓名……

4、数据对象(Data Object)：相同特性数据元素的集合，是数据的一个子集

- 整数数据对象

  $N = \{ 0, \pm1, \pm2, \cdots \}$

- 学生数据对象
  - 学生记录的集合

5、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。

数据结构是带“结构”的数据元素的集合，“结构”就是指数据元素之间存在的关系。

数据结构的两个层次：

逻辑结构---

数据元素间之间的逻辑关系，即抽象化的相互关系，与数据的存储无关，独立于计算机，它是从具体问题抽象出来的数学模型。

存储结构（物理结构）----

数据元素及其关系在计算机存储器中的存储方式。

数据的运算----

对数据施加的操作。

逻辑结构

划分方法一

（1）线性结构----

有且仅有一个开始和一个终端结点，并且所有结点都最多只有一个直接前趋和一个后继。

例如：线性表、栈、队列、串

（2）非线性结构----

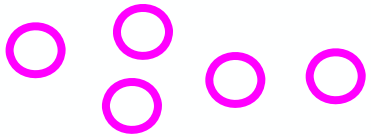
一个结点可能有多个直接前趋和直接后继。

例如：树、图

逻辑结构

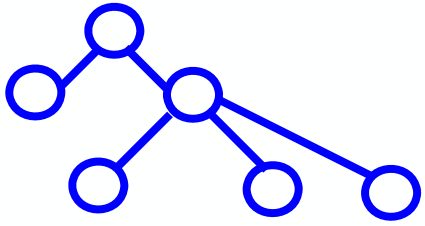
划分方法二

集合——数据元素间除“同属于一个集合”外，无其它关系

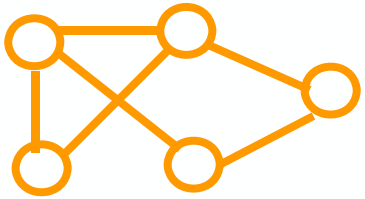


线性结构——一个对一个，如线性表、栈、队列

树形结构——一个对多个，如树



图形结构——多个对多个，如图



存储结构

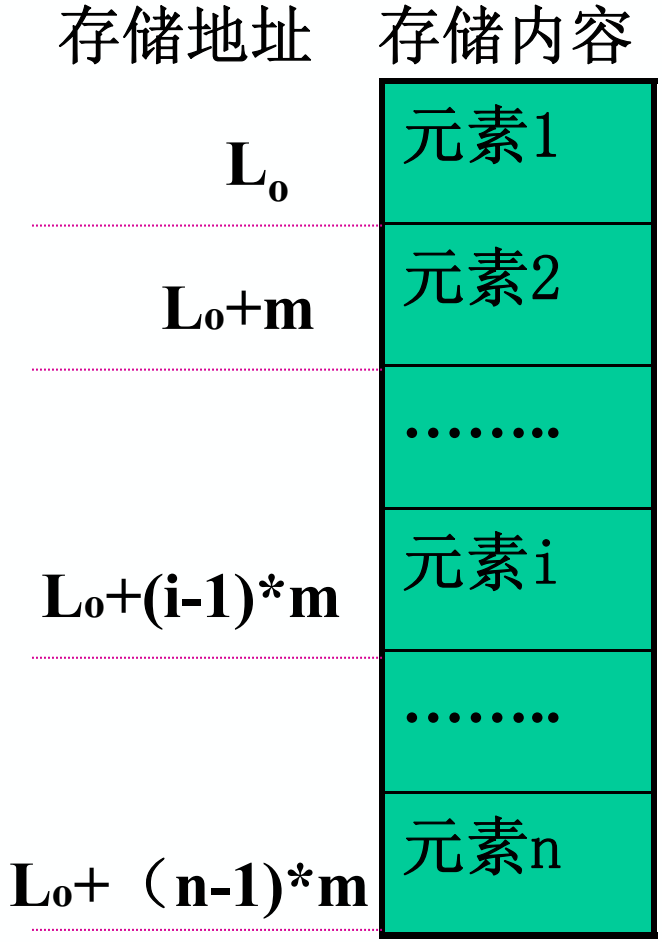
存储结构分为：

顺序存储结构——借助元素在存储器中的相对位置来表示数据元素间的逻辑关系

借助于数组描述，用于线性数据结构，非线性数据结构的线性化

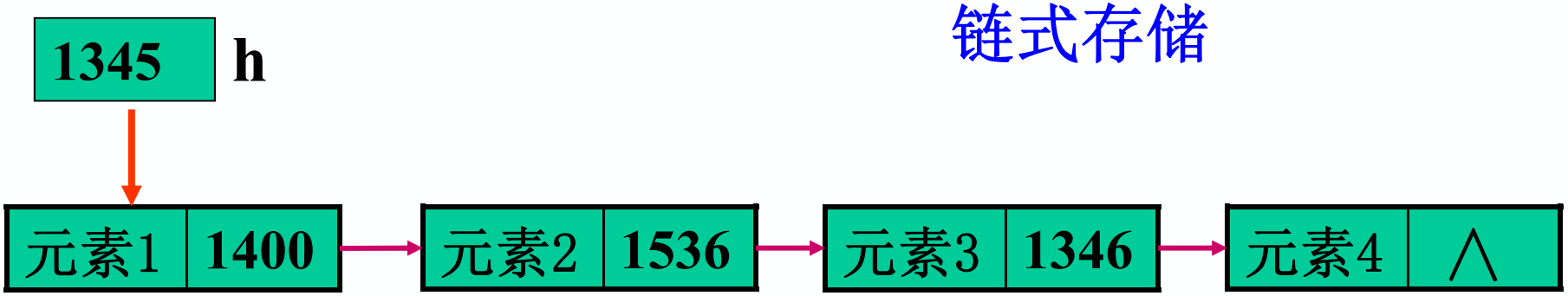
链式存储结构——借助指示元素存储地址的指针表示数据元素间的逻辑关系

顺序存储



Loc(元素i)=Lo+（i-1)*m

链式存储



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

数据的运算

数据结构的主要运算：

(1) 建立(Create)一个数据结构；

(2) 消除(Destroy)一个数据结构；

(3) 从一个数据结构中删除(Delete)一个数据元素；

(4) 把一个数据元素插入(Insert)到一个数据结构中；

(5) 对一个数据结构进行访问(Access)；

(6) 对一个数据结构(中的数据元素)进行修改(Modify)；

(7) 对一个数据结构进行排序(Sort)；

(8) 对一个数据结构进行查找(Search)。

数据结构3方面之联系

- 同一逻辑结构的不同存储结构，则用不同名称称谓

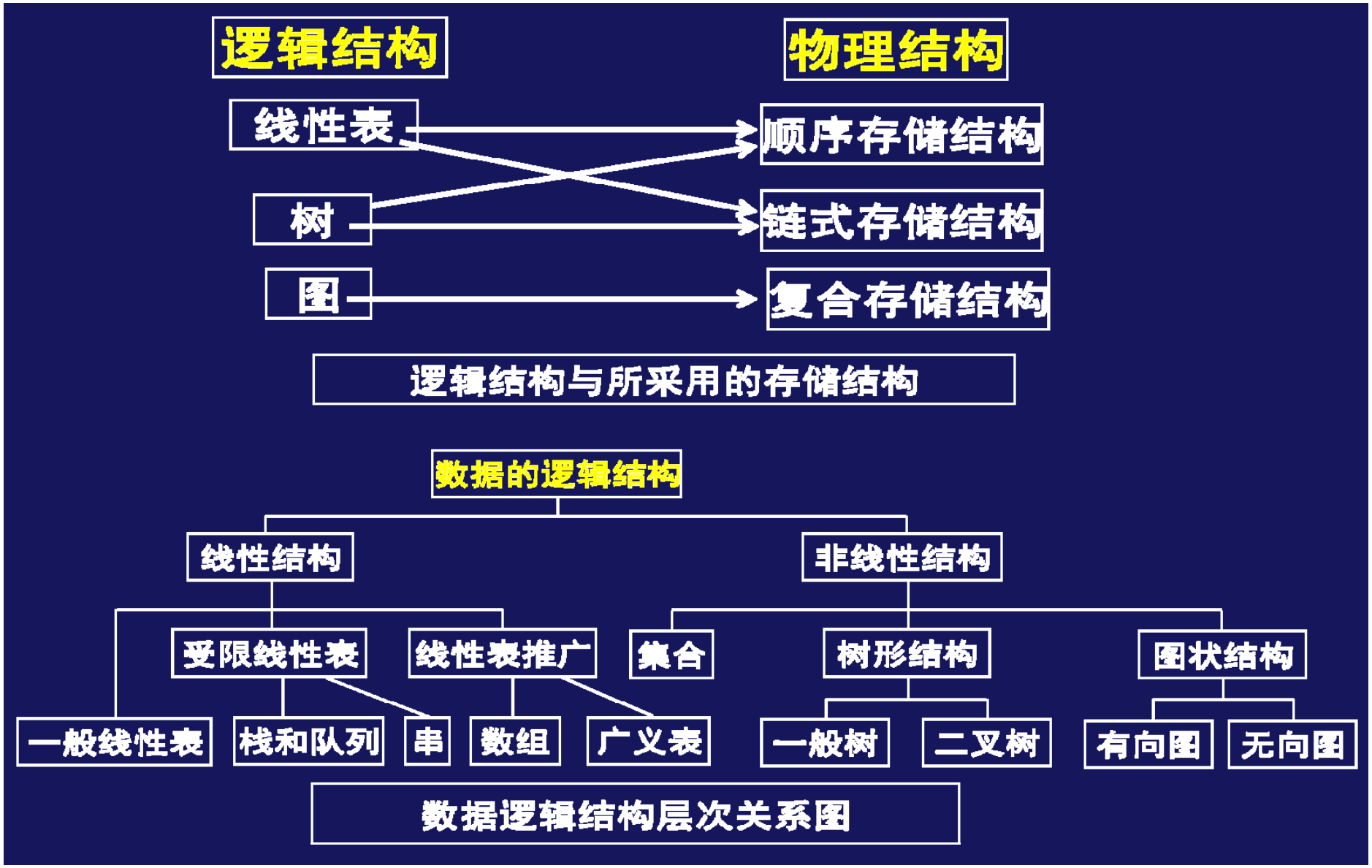
  例如，线性表⇨顺序表，链表，散列表

- 运算不同，称谓也不同

  线性表⇨栈、队列⇨ 顺序栈、顺序队列

- 数据的逻辑结构和物理结构是密不可分的两个方面，一个**算法的设计**取决于所选定的逻辑结构，而**算法的实现**依赖于所采用的存储结构。
- 在C语言中，用**一维数组**表示顺序存储结构；用**结构体类型**表示链式存储结构。

数据结构3方面之联系


逻辑结构
物理结构
线性表
树
图
顺序存储结构
链式存储结构
复合存储结构
逻辑结构与所采用的存储结构
数据的逻辑结构
线性结构
非线性结构
受限线性表
线性表推广
集合
树形结构
图状结构
一般线性表
栈和队列
串
数组
广义表
一般树
二叉树
有向图
无向图
数据逻辑结构层次关系图

数据类型

- 数据类型是高级程序设计语言提供的与数据结构相对应的一个基本概念，变量所具有的数据种类

FORTRAN语言：整型、实型、和复数型

C语言：

基本数据类型： char int float double void

构造数据类型：数组、结构体、共用体、文件

✓ 定义：数据类型是一组性质相同的值的集合，以及定义于这个集合上的一组运算的总称

抽象数据类型
(ADTs: Abstract Data Types)

- 抽象数据的组织和与之相关的操作，可看作是数据的逻辑结构以及在逻辑结构上定义的抽象操作
- 一般由用户定义，用来表示应用问题的数据模型，由基本的数据类型组成，并包括一组相关的操作

抽象数据类型可以用以下的三元组来表示：

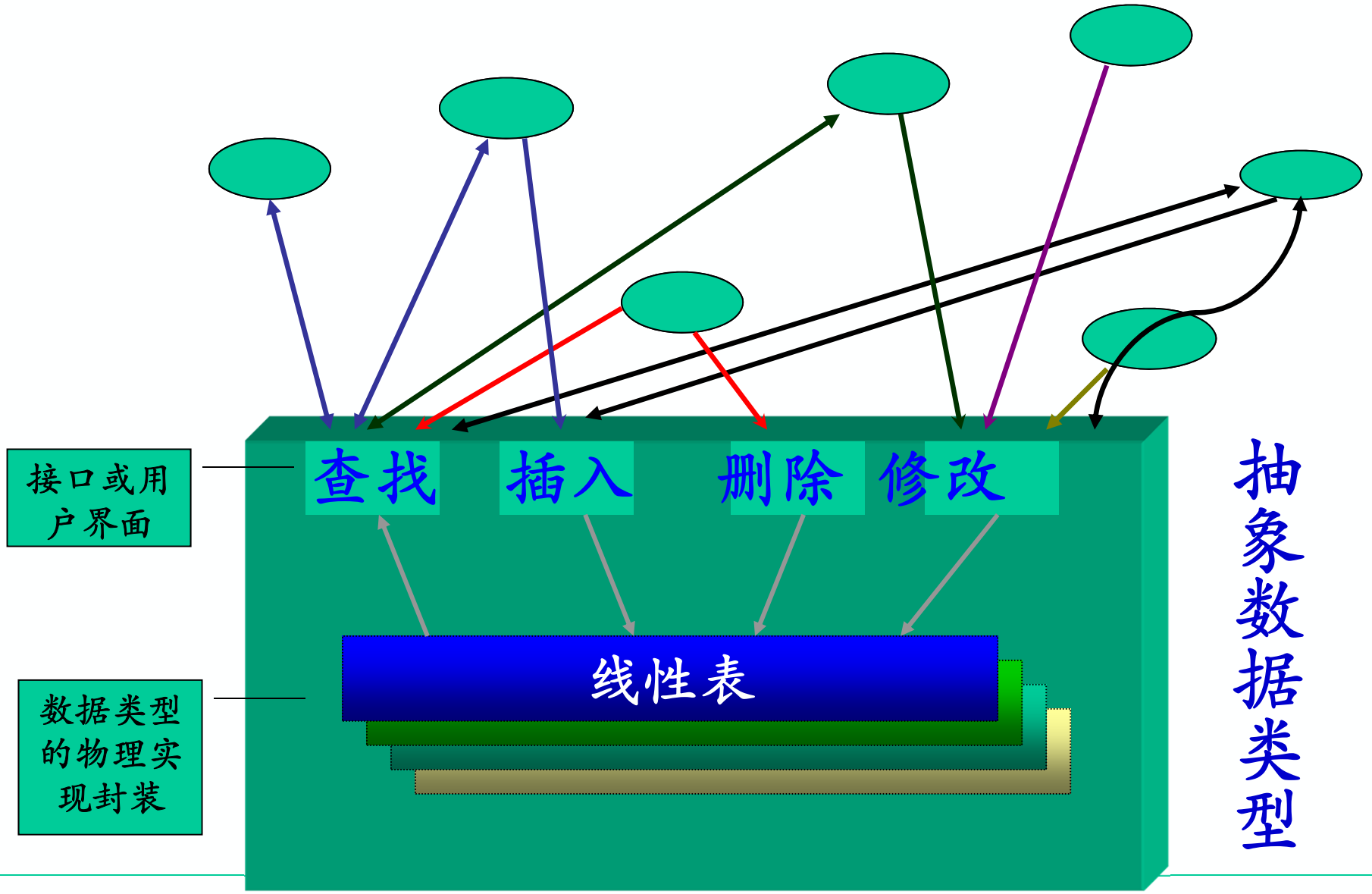
ADT = （D，S，P）

数据对象　D上的关系集　D上的操作集

ADT常用定义格式

```
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
```

信息隐蔽和数据封装，使用与实现相分离



1.3 抽象数据类型的表示与实现

抽象数据类型可以通过固有的数据类型（如整型、实型、字符型等）来表示和实现。

它有些类似C语言中的结构（struct)类型，但增加了相关的操作

教材中用的是类C语言（介于伪码和C语言之间）作为描述工具

但上机时要用具体语言实现，如C或C++等

- (1) 预定义常量及类型
- //函数结果状态代码

```
#define OK 1
#define ERROR 0
#define OVERFLOW -2
```

- // Status是函数返回值类型，其值是函数结果状态代码。

```
typedef  int  Status;
```

- (2)数据元素被约定为ElemType 类型，用户需要根据具体情况，自行定义该数据类型。

(3)算法描述为以下的函数形式：

```
函数类型 函数名（函数参数表）
{
        语句序列;
}
```

（4）内存的动态分配与释放

使用new和delete动态分配和释放内存空间

分配空间　指针变量=new数据类型;

释放空间　delete指针变量;

（5）赋值语句

（6）选择语句

（7）循环语句

（8）使用的结束语句形式有：

函数结束语句 return

循环结束语句 break;

异常结束语句 exit（异常代码）；

（9）输入输出语句形式有：

输入语句　cin（scanf( )）

输出语句　cout (printf( ))

（10）扩展函数有：

求最大值 max

求最小值 min

例：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：S＝{<e1,e2> | e1是复数的实数部分,e2 是复数的虚数部分}

基本操作：

Create(&C, x, y)

操作结果：构造复数C,其实部和虚部分别被赋以参数x和y的值。

GetReal(C)

初始条件：复数C已存在。

操作结果：返回复数C的实部值。

GetImag(C)

初始条件：复数C已存在。

操作结果：返回复数C的虚部值。

Add(C1, C2)

初始条件：C1, C2是复数。

操作结果：返回两个复数C1、C2的和值。

Sub(C1, C2)

初始条件：C1, C2是复数。

操作结果：返回两个复数C1、C2的差值。

} ADT Complex

抽象数据类型复数的表示与实现

```
typedef struct {                                        //复数类型
    float   realpart;
    float   imagepart;
}Complex;

void Create(&Complex  C, float  x, float  y){           //构造一个复数
    C.realpart=x;
    C.imagepart=y;
}

float GetRealComplex  C){                               //取复数的实部
    return  C.realpart;
}

Complex Add(Complex  C1, Complex  C2){                  //求两复数的和
    Complex sum;
    sum.realpart=C1.realpart+C2.realpart;
    sum.imagepart=C1.imagepart+C2.imagepart;
    return sum;
}
```

1.4 算法和算法分析

- 算法定义：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

- 算法的描述：
  - 自然语言
  - 流程图
  - 程序设计语言
  - 伪码

- 算法的特性：
  - 输入 有0个或多个输入
  - 输出 有一个或多个输出(处理结果)
  - 确定性 每步定义都是确切、无歧义的(只有唯一执行路径)
  - 有穷性 算法应在执行有穷步后结束(有穷步骤，每步时间有限)
  - 有效性 每一条运算应足够基本(所有操作可通过已经实现的基本运算有限次实现之)
- 算法与程序的联系与区别

算法的评价

- 正确性
- 可读性
- 健壮性
- 高效性（时间代价和空间代价）

算法的效率的度量

- 算法效率：用依据该算法编制的程序在计算机上执行所消耗的时间来度量

  事后统计

  事前分析估计

1.事后统计：利用计算机内的计时功能，不同算法的程序可以用一组或多组相同的统计数据区分

```
double start, stop;
time (&start);
main process(n, …);
time (&stop);
double runTime = stop -start;
printf ("%d%d\n" , n, runTime );
```

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

2.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程序语言

④编译程序产生机器代码质量

⑤机器执行指令速度

同一个算法用不同的语言、不同的编译程序、在不同的计算机上运行，效率均不同，————使用绝对时间单位衡量算法效率不合适

- **算法分析**

  算法的时间是每语句执行时间的总和

  每语句的执行时间=该语句执行次数（频度）

  X该语句执行1次的时间

  假定：每语句执行1次的时间为1个时间单位

  则：算法的执行时间=∑各语句频度

- **问题的规模（Size）n**

  输入量的大小，如…

- **时间复杂度**：算法的运行时间，是问题规模的函数

■ 时间复杂度

例：矩阵乘法

```
for ( i=0; i＜n; i++)                                    //n+1
    for ( j=0; j＜n; j++) {                              //n(n+1)
        C[i] [j]=0;                                      //n²
        for ( k=0;k＜n; k++)                             //n²(n+1)
            C[i][j]=C[i][j]+A[i][k]*B[k][j];             //n³
    }
```

- 详细分析： $T(n)=2n^3+3n^2+2n+1$
- 简单分析：频度最大的语句

$$T(n)=\sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\sum_{k=0}^{n-1}1 \quad =n^3$$

时间复杂度的渐进表示法

•算法中基本语句重复执行的次数是问题规模的某个函数f(n),算法的时间复度记作：

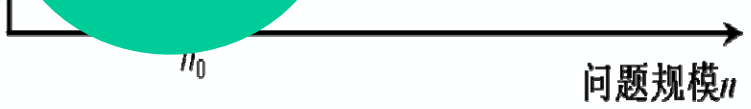
$$T(n)=O(f(n))$$

，算法执行的时间的增
，称渐
为：
在正
合
数，则
$T(n) = O(f(n))$表示存在正
C和n_0，使得当$n \geqslant n_0$时都
$0 \leqslant T(n) \leqslant Cf(n)$。

- *算法中重复执行次数和算法的执行时间成正比的语句*
- *对算法运行时间的贡献最大*

n越大算法的执行时间越长

- *排序：n为记录数*
- *矩阵：n为矩阵的阶数*
- *多项式：n为多项式的项数*
- *集合：n为元素个数*
- *树：n为树的结点个数*
- *图：n为图的顶点数或边数*



时间复杂度的渐进表示法

- **渐近时间复杂度（简称时间复杂度）**

从宏观上评价时间性能，只关心当n趋向无穷大时，T(n)的数量级(阶)

$$\lim_{n \to \infty} T(n) / n^3 =$$

$$\lim_{n \to \infty} (2n^3 + 3n^2 + 2n + 1) / n^3$$

$$= 2$$

即：T(n) 和n^3同阶，数量极相同

记作：$T(n)=O(n^3)$

分析算法时间复杂度的基本方法

- 找出语句频度最大的那条语句作为基本语句
- 计算基本语句的频度得到问题规模n的某个函数$f(n)$
- 取其数量级用符号“O”表示

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

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

$f(n)=n^2$

时间复杂度是由嵌套最深层语句的频度决定的

```
void exam ( float x[ ][ ], int m, int n ) {
    float sum [ ];
    for ( int i = 0; i < m; i++ ) {
        sum[i] = 0.0;
        for ( int j = 0; j < n; j++ )
            sum[i] += x[i][j];
    }
    for ( i = 0; i < m; i++ )
        cout << i << " : " <<sum [i] << endl;
}
```

T(n) = O(m*n)

f(n)=m*n

例1：N×N矩阵相乘

```
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]=c[i][j]+a[i][k]*b[k][j];
     }
```

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

算法中的基本操作语句为

```
c[i][j]=c[i][j]+a[i][k]*b[k][j];
```

$$T(n)=\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}1=\sum_{i=1}^{n}\sum_{j=1}^{n}n=\sum_{i=1}^{n}n^2=n^3=o(n^3)$$

例2：

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

定理1.1

若$f(n)=a_m n^m+a_{m-1}n^{m-1}+\ldots+a_1 n+a_0$是$m$次多项式，则$T(n)=O(n^m)$。

忽略所有低次幂项和最高次幂系数，体现出增长率的含义

$$\text{语句频度}=\sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=1}^{j}1=\sum_{i=1}^{n}\sum_{j=1}^{i}j=\sum_{i=1}^{n}\frac{i(i+1)}{2}$$

$$=\frac{1}{2}\left(\sum_{i=1}^{n}i^2+\sum_{i=1}^{n}i\right)$$

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

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

例3：分析以下程序段的时间复杂度

```
i=1;                    ①
while(i<=n)
    i=i*2;              ②
```

$2^{f(n)} \le n$　即$f(n) \le \log_2 n$，取最大值$f(n)=\log_2 n$

所以该程序段的时间复杂度$T(n)=O(\log_2 n)$

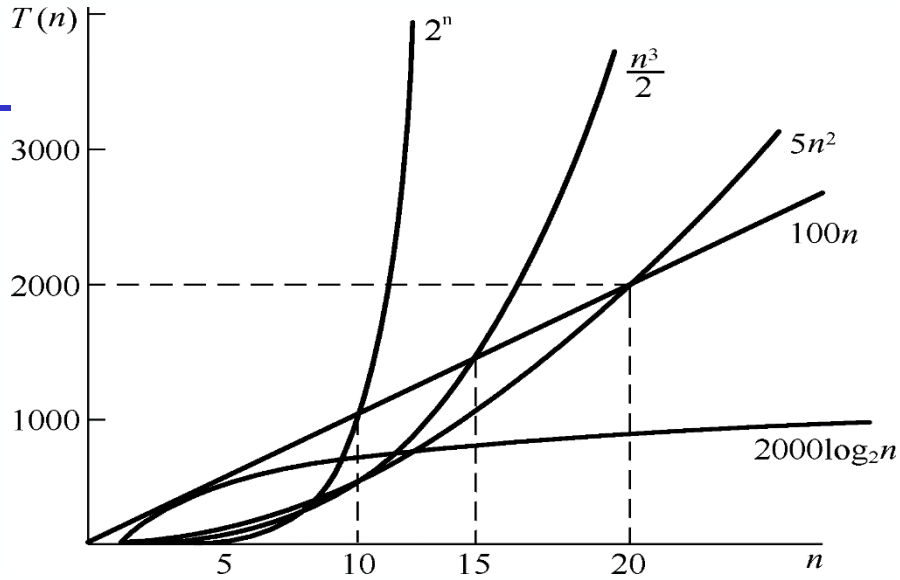
有的情况下，算法中基本操作重复执行的次数还随问题的输入数据集不同而不同

例4：顺序查找，在数组a[i]中查找值等于e的元素，返回其所在位置。

```
for (i=0;i< n;i++)
    if (a[i]==e) return i+1;
        return 0;
```

- 最好情况：1次
- 最坏情况：n
- 平均时间复杂度为:O(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



时间复杂度T(n)按数量级递增顺序为：

复杂度低 → 复杂度高

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

渐进空间复杂度

- 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)

- 算法要占据的空间
  - 算法本身要占据的空间，输入/输出，指令，常数，变量等
  - 算法要使用的辅助空间

S(n) = O(1)
原地工作

…组a中的n个数逆序存放到…

S(n) = O(n)

【算法1】

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

【算法2】

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

小结

1、数据、数据元素、数据项、数据结构等基本概念

2、对数据结构的两个层次的理解

- 逻辑结构
- 存储结构

3、抽象数据类型的表示方法

4、算法、算法的时间复杂度及其分析的简易方法