第一章 绪论

内容提要：

- 数据结构的重要性
- 数据结构的基本概念
- 算法的基础知识

1.1 问题求解

什么是计算机科学？

- 计算机科学是关于计算机（硬件系统）的
- 计算机科学是关于软件（程序）的
- 计算机科学是关于数据（数据表示）的
- 计算机科学是关于算法（数据变换）的
- …

1.1 问题求解

简单说，计算机科学是研究机器如何进行信息表示、信息处理、信息传输的学科。

- 信息表示：
  - 存储器
  - 编码等
  - 效率
- 信息处理：
  - 处理器
  - **算法**
  - 效率
- 信息传输：
  - 信道（介质）
  - 可靠
  - 安全
  - 效率

计算机系统=硬件+软件

1.1 问题求解

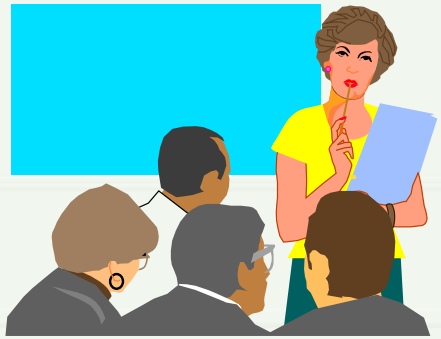
一方面IT技术的发展改变了人类的各种活动，促进了社会发展，另一方面人类的应用需求也对IT技术不断提出新的挑战。那么如何应对？

- 发展硬件技术：
  - 容量更大、更便宜、访问速度更快的存储；
  - 速度更快、处理能力更强的CPU;
  - 安全可靠、快速的传输介质；

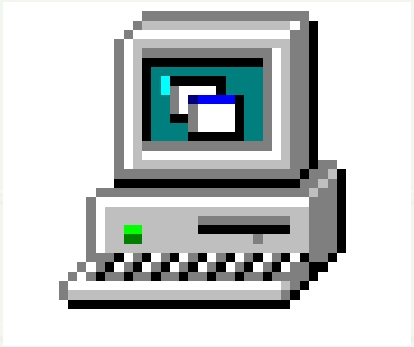
- 发展软件技术：
  - 应用的数据如何存储、如何处理（数据结构+算法）
  - 如何运行应用
  - 如何构建应用（软件工程）

1.1 问题求解

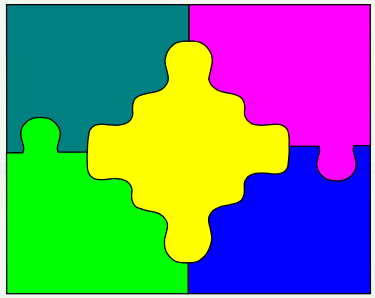
现实世界的问题



计算机系统



问题的解答



问题的描述包括：

要处理的数据

期望的结果

（隐含着对数据实施的操作-运算）

问题处理软件（程序）：

数据在计算机中的存储

某种语言描述的算法（程序）

问题处理结果：

数字、表格

图形等等

1.1 问题求解

- 利用计算机问题求解的基本步骤：

  - 分析：明确和发现问题的内在关系；
  - 设计：如何存储问题的数据、如何处理数据（设计数据结构与算法）
  - 编码：选择适当的程序设计语言编写程序
  - 测试+维护：运行一些实例，以保证能正确解决问题

问题的数据在计算机中的表示及处理是利用计算机求解问题的关键！

1.2 数据结构的产生

随着计算机技术的飞速发展，计算机应用也已经渗透到了社会的各个领域。我们发现有了很大的一些变化：

- ♣ 计算机由最初的单一科学计算到几乎无所不能；
- ♣ 加工处理的对象由数值型变为数值型和非数值型；
- ♣ 处理的数据量由小变为大、巨大（海量存储、计算）；
- ♣ 数据之间的关系由简单变复杂、很复杂；

如何应对？

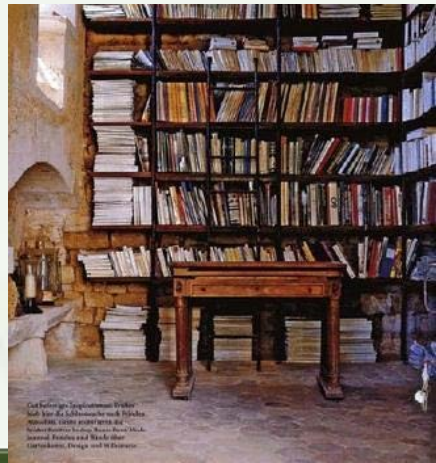
1.2 数据结构的产生

应对策略之一：发展硬件技术（其他课程学习）

应对策略之二：研究问题本身的特点。我们应该想到：

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。

举例：图书管理



1.2 数据结构的产生

信息的表示和组织形式直接影响到数据处理的效率！

于是：

人们开始考虑通过研究、分析问题数据本身的特点，从而利用这些特点提高数据表示和处理的效率。

——*这样就产生了“数据结构”*

提醒：

数据结构是随着计算机应用的深入而产生的（或者说求解简单问题用不到数据结构）。

1.2 数据结构的产生

从现在开始，你想要用计算机求解一个问题，就必须首先想：问题的数据有哪些？它们之间是什么关系？如何操作（处理）？

下面看几个例子：

例1. 问题：图书管理，完成书目的管理

数据：？

关系：？

操作：？

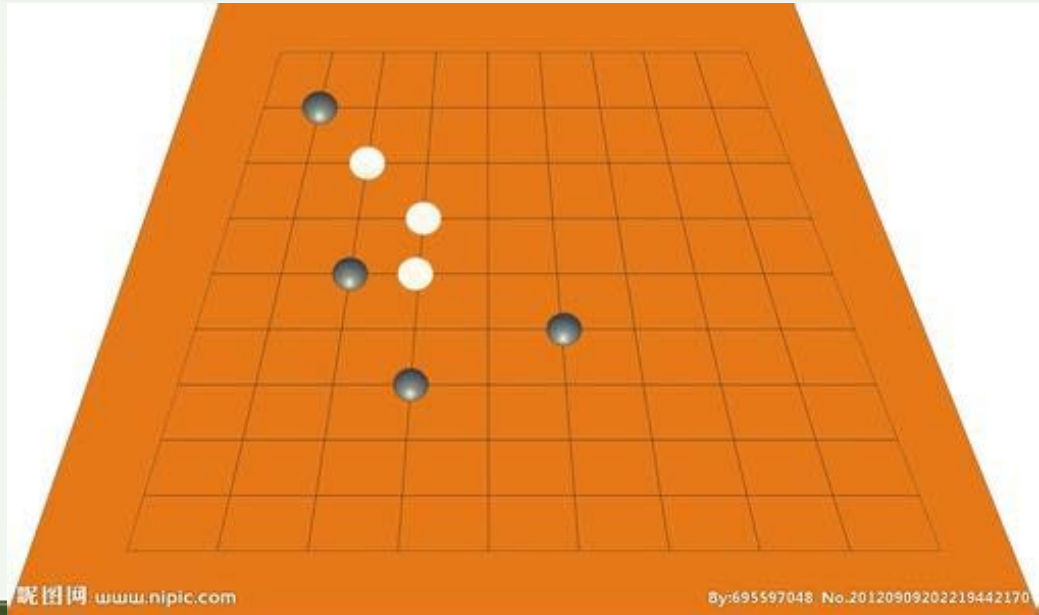
1.2 数据结构的产生

例2. 问题：人机对弈，即人与计算机下棋

数据：？

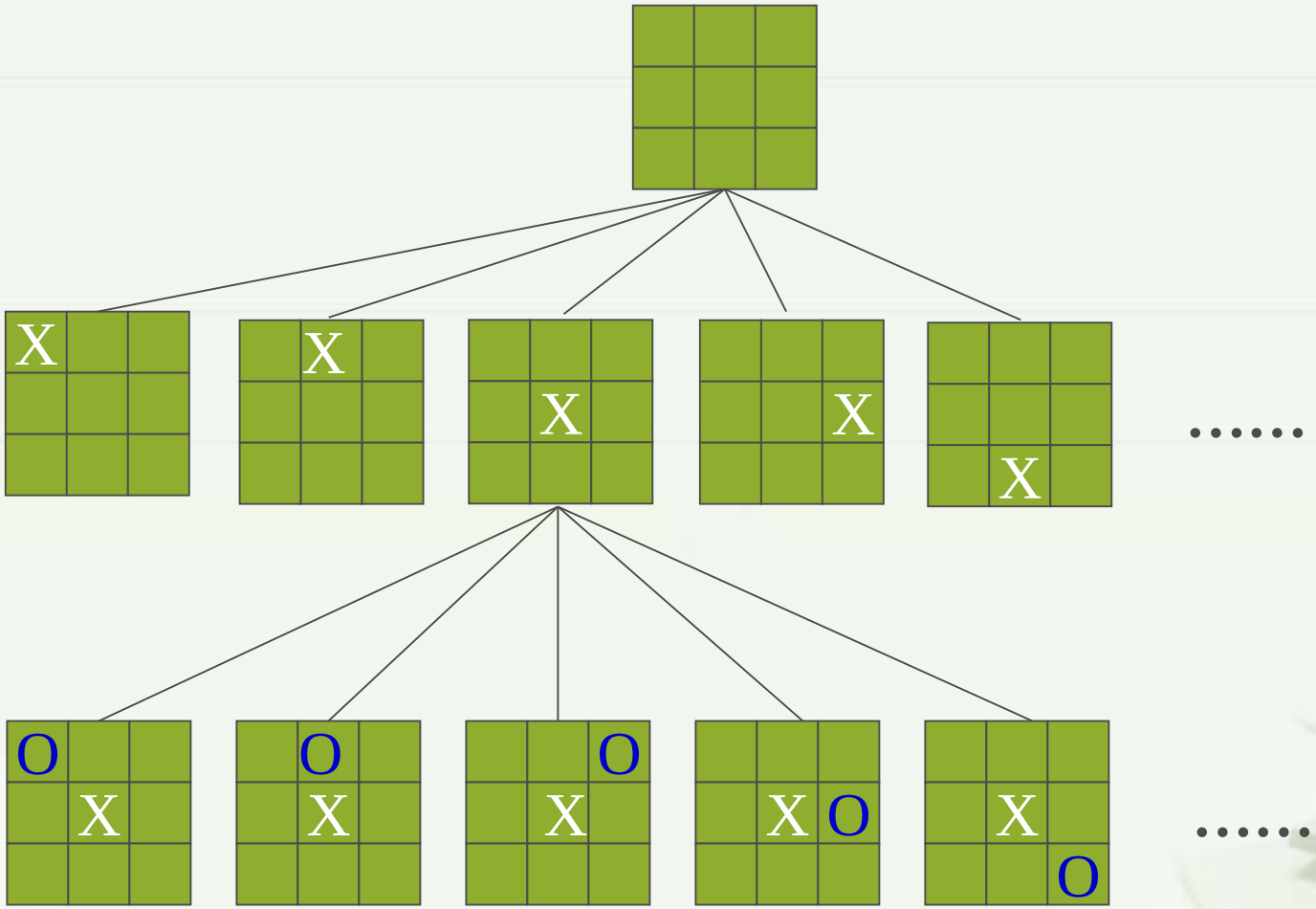
关系：？

操作：？



1.2 数据结构的产生

“井”字棋



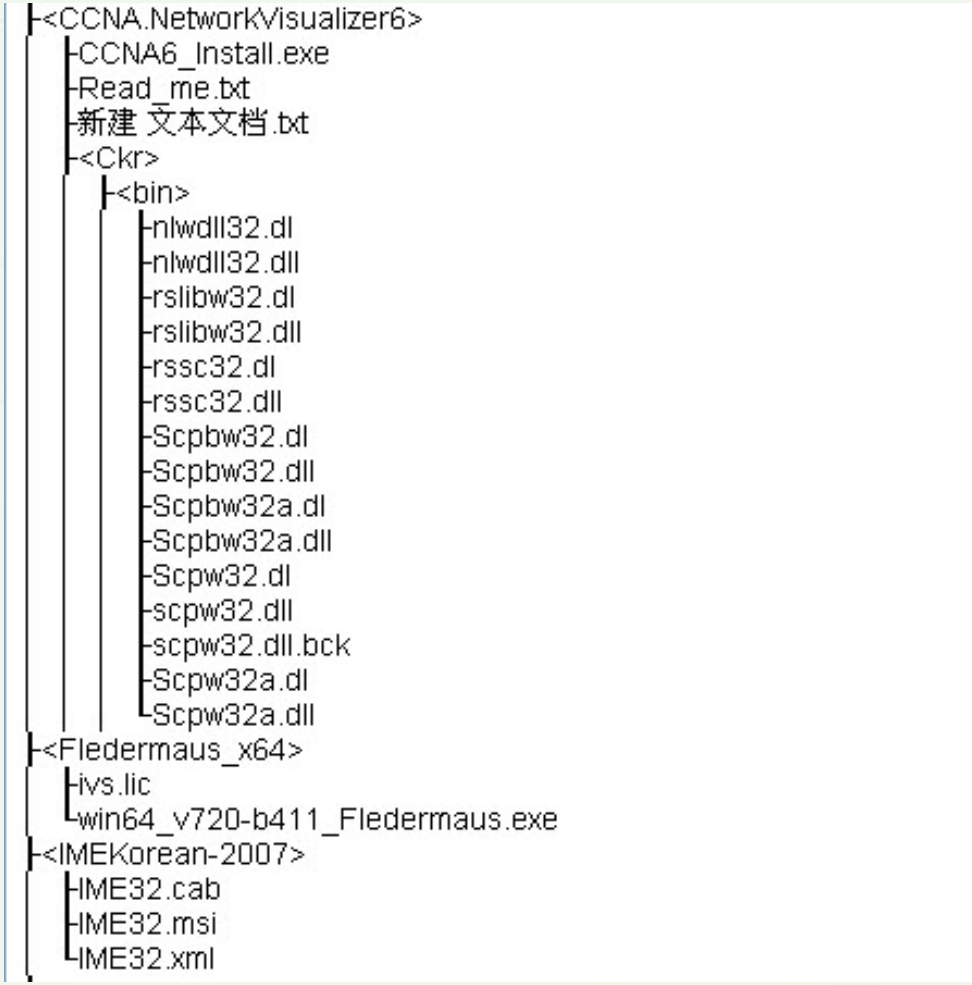
1.2 数据结构的产生

例3. 问题：计算机的文件管理

数据：文件（文件夹）

关系：层次

操作：文件操作



1.2 数据结构的产生

例4. 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。

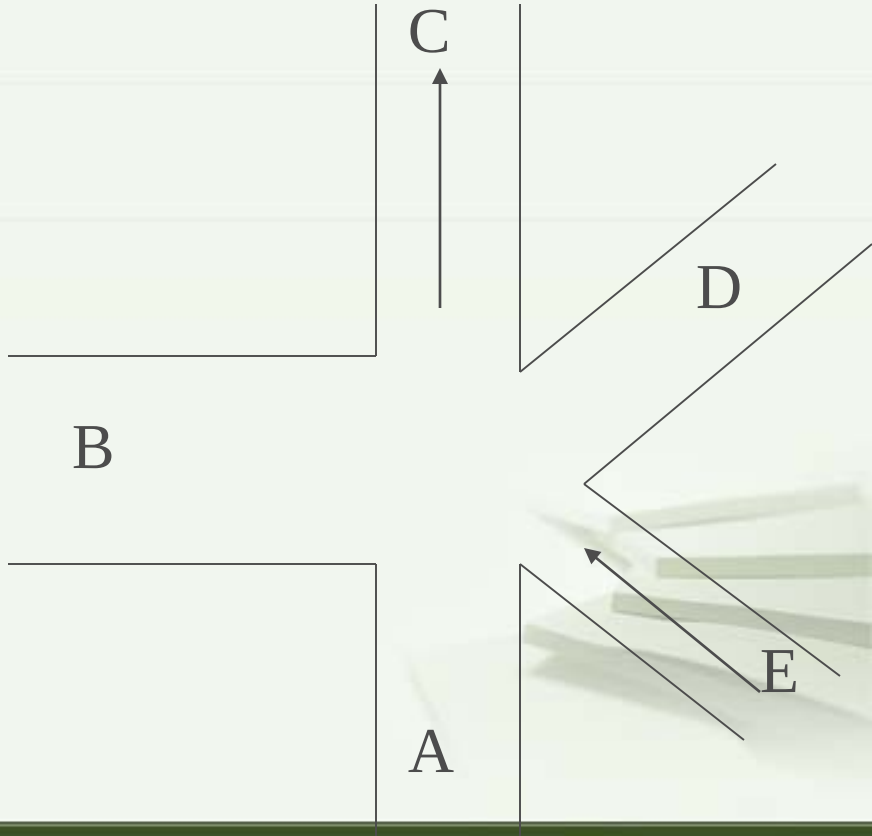
数据：？

关系：？

操作：？

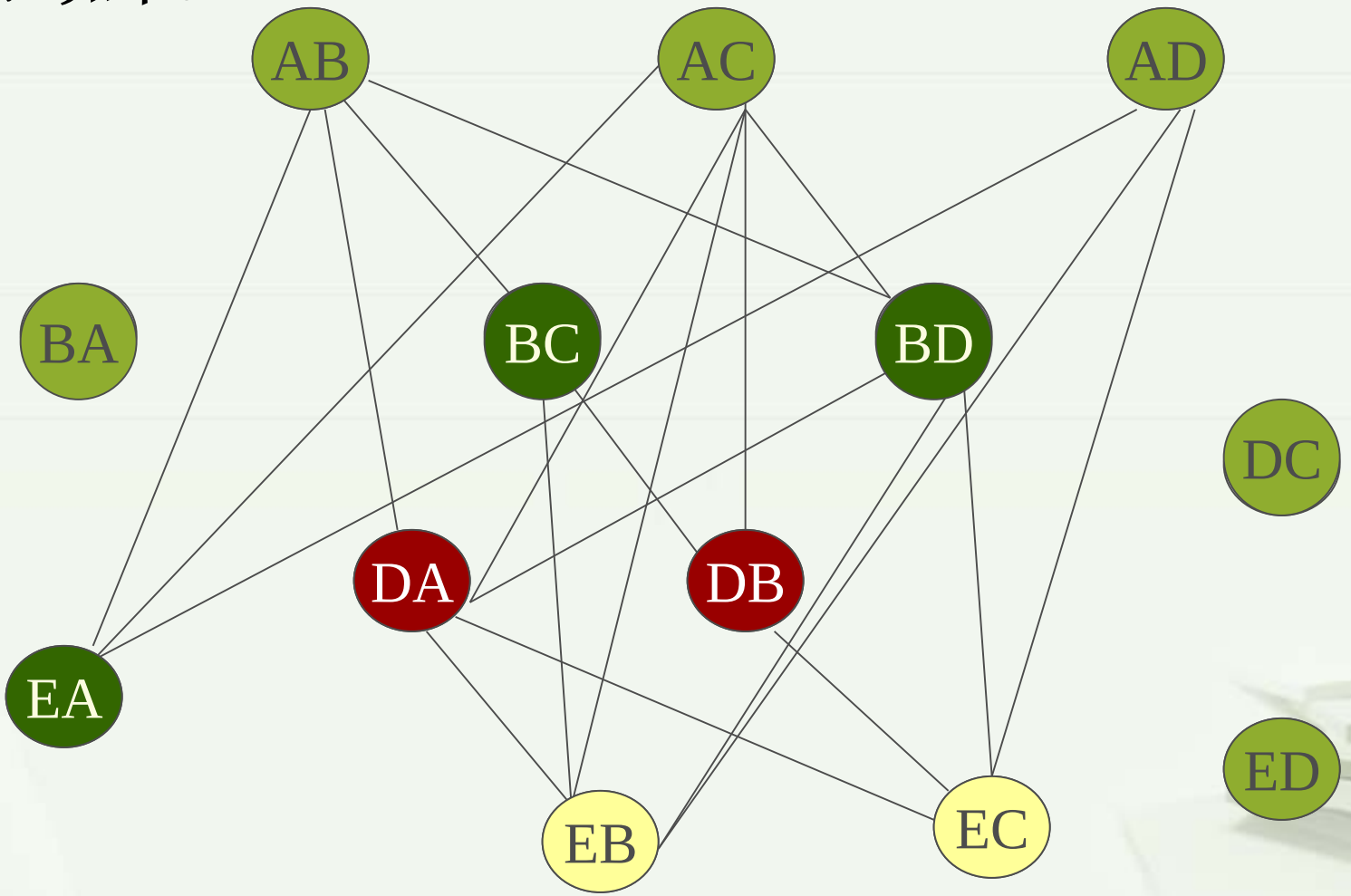
共13条路：

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



1.2 数据结构的产生

以道路为顶点，道路之间相互矛盾的用连线连接起来，形成一个图，如下：



可以用四种颜色着色，因此需设四种信号灯。

1.2 数据结构的产生

- 数据结构的发展概况

从时间上看：

50年代末~60年代初	萌芽期	分散于其他课程中
60年代中~70年代初	形成发展期	68年成为独立课程
70年代中~80年代初	完善期	
80年代中~	完善更新期	

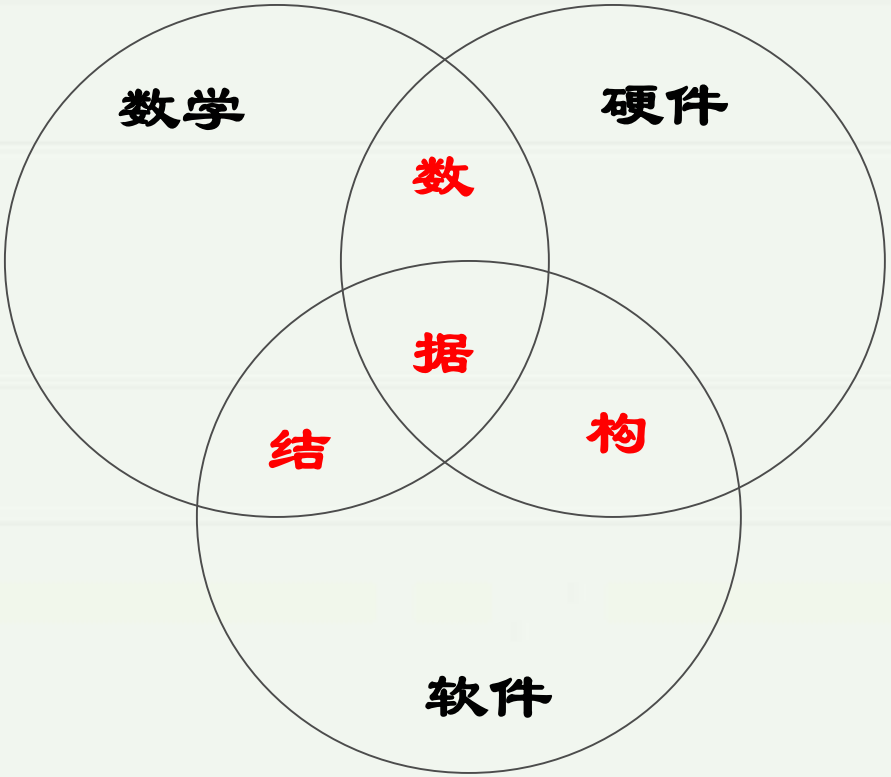
从内容上看：

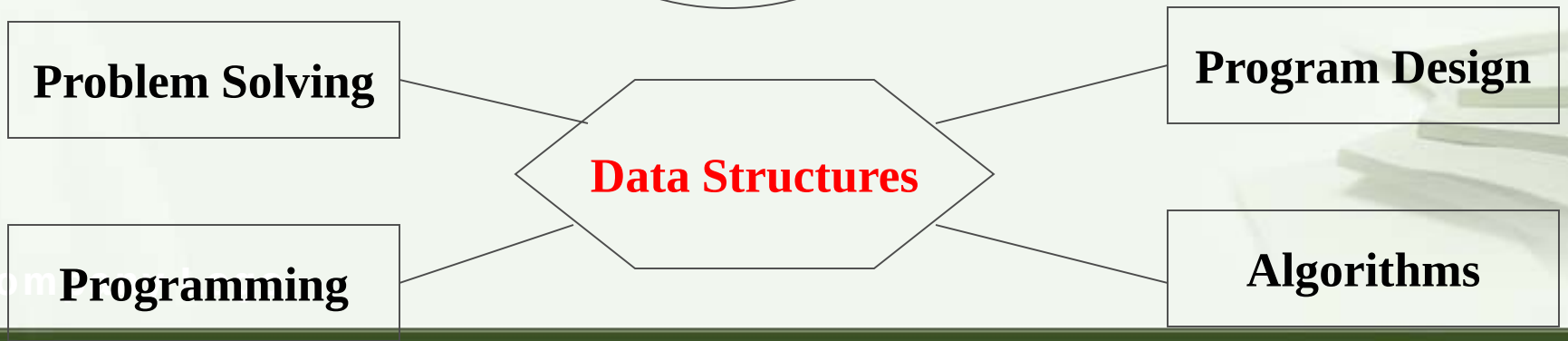
最初：介绍一些表处理系统
稍后：讨论图论，尤其是表和树
再后：讨论离散数学的内容，集合、代数、格
再后：确定了三种基本数据结构及存储结构
再后：加入了分类、检索和文件系统
近几年：涉及到了面向对象等新的内容

在我国，80年代初才列入计算机科学教学计划

1.2 数据结构的产生

- 数据结构的地位





1.3 数据结构的基本概念

[数据 Data] 用于描述客观事物的数值、字符等一切**可以输入到计算机**中，并由**计算机加工处理**的符号集合。
信息（ Information），信息和数据是什么关系？

[数据元素 Data Element] 数据中的一个个个体，是数据的**基本**单位。这是一个相对概念。

[数据项 Data Item] 构成数据元素的成份，是数据不可分割的最小单位。

[数据对象 Data Object] 具有相同特性的数据元素的一个集合，是数据的子集。

1.3 数据结构的基本概念

[结构 Structure] 数据元素之间的关系(Relation)

[数据结构 Data Structure] 相互之间存在一种或多种特定关系的数据元素的集合，即带结构的数据元素的集合。

数据结构=数据+结构　　记作：

Data_Structure=(D,R)

其中：**Data_Structure是数据结构的名称**

D是数据元素的有限集合（一般为一个数据对象）

R是D上关系的有限集

注：这里说的数据元素之间的关系是指元素之间本身固有的逻辑关系，与计算机无关。

因此，又称为：数据的逻辑结构、抽象数据结构

1.3 数据结构的基本概念

数据元素之间的逻辑关系分为：

（1）线性关系

（2）非线性关系：层次关系、网状关系

相应地，数据结构也分为：

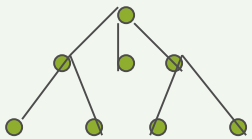
（1）线性结构(linear structure)

线性表

（2）非线性结构(nonlinear structure)

层次结构(hierarchical structure)：树

群结构(group structure)：集合，图



1.3 数据结构的基本概念

研究数据结构的目的是干什么？

[数据的存储结构] 数据结构在计算机中的表示（映象），即数据结构在计算机中的组织形式。又称为数据的物理结构。

数据结构的物理结构是指逻辑结构的存储镜像(image)。数据结构 *DS* 的物理结构 *P* 对应于从 *DS* 的数据元素到存储区*M*（维护着逻辑结构*S*）的一个映射：

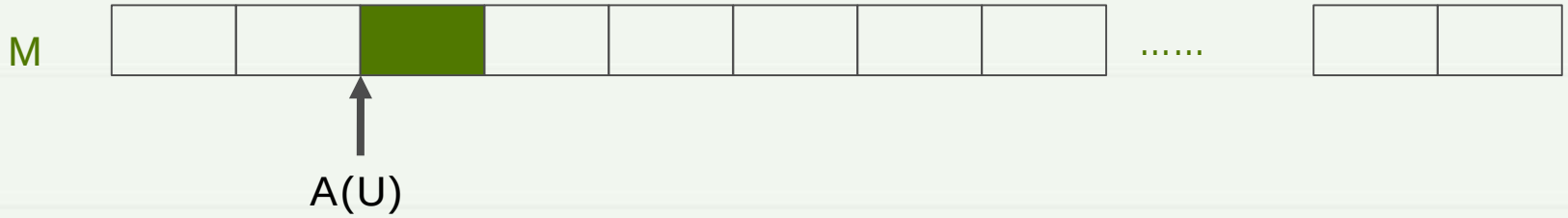
$$P:(D, S) \dashrightarrow M$$

特别要注意：

我们研究数据结构的目的是要利用数据之间的关系（结构），因此，在存储时既要存储元素本身，还要存储（表示）关系！！

1.3 数据结构的基本概念

存储器模型：一个存储器 M 是一系列固定大小的存储单元，每个单元 U 有一个唯一的地址 $A(U)$，该地址被连续地编码。每个单元 U 有一个唯一的后继单元 U'=succ(U)。



P 的四种基本映射模型：

顺序（sequential）
链接（linked）
索引（indexed）
散列（hashing）。

1.3 数据结构的基本概念

顺序映射：映射方式？

链式映射：映射方式？

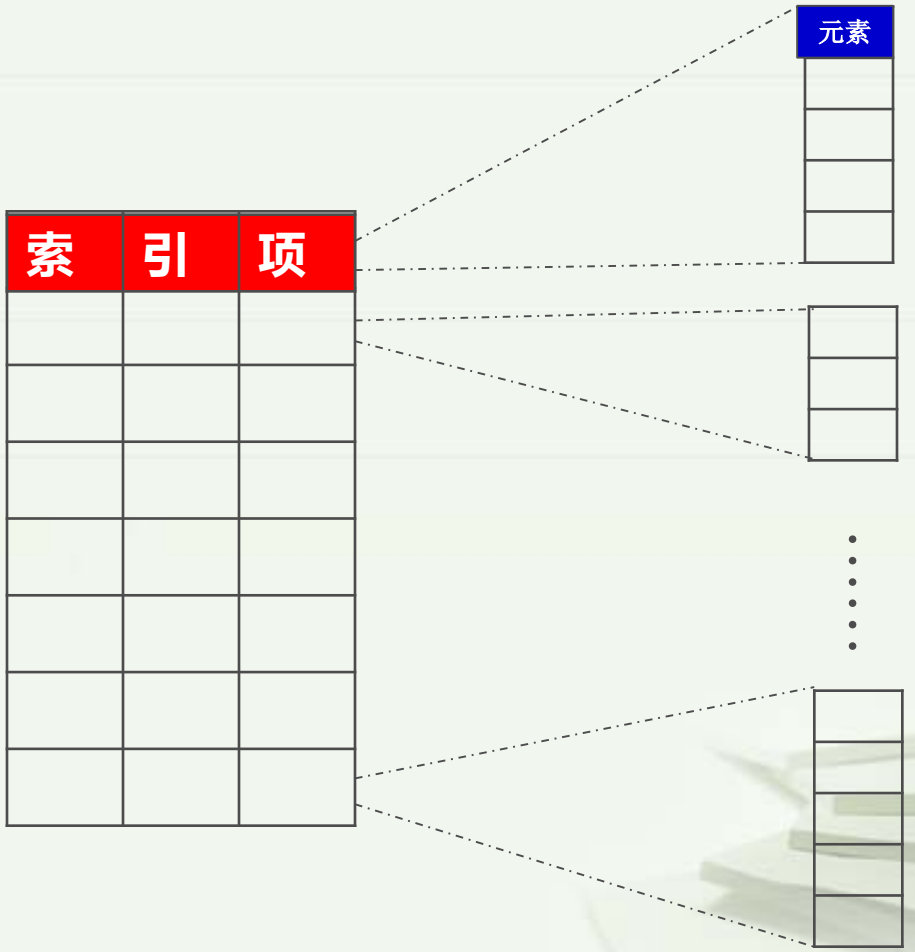
索引（indexed）映射：

如何映射关系？

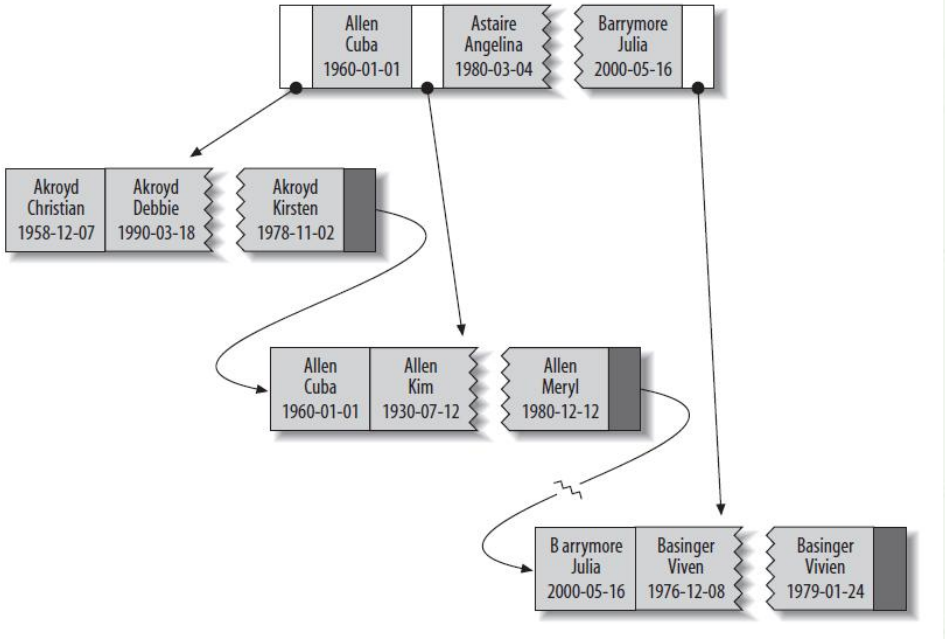
- 稠密索引(Dense Index)
- 稀疏索引(Spare Index)

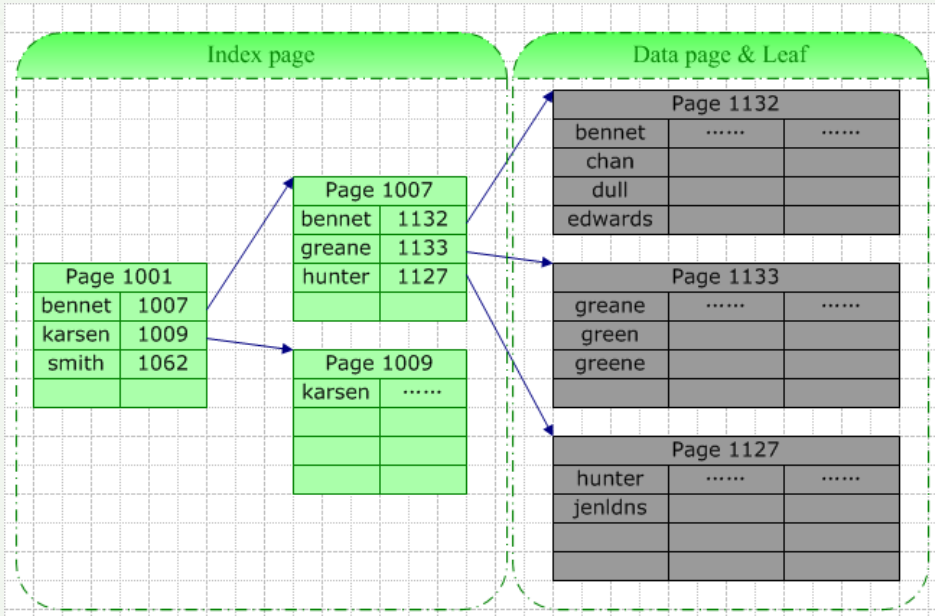
- 多级索引

- 索引的建立与维护



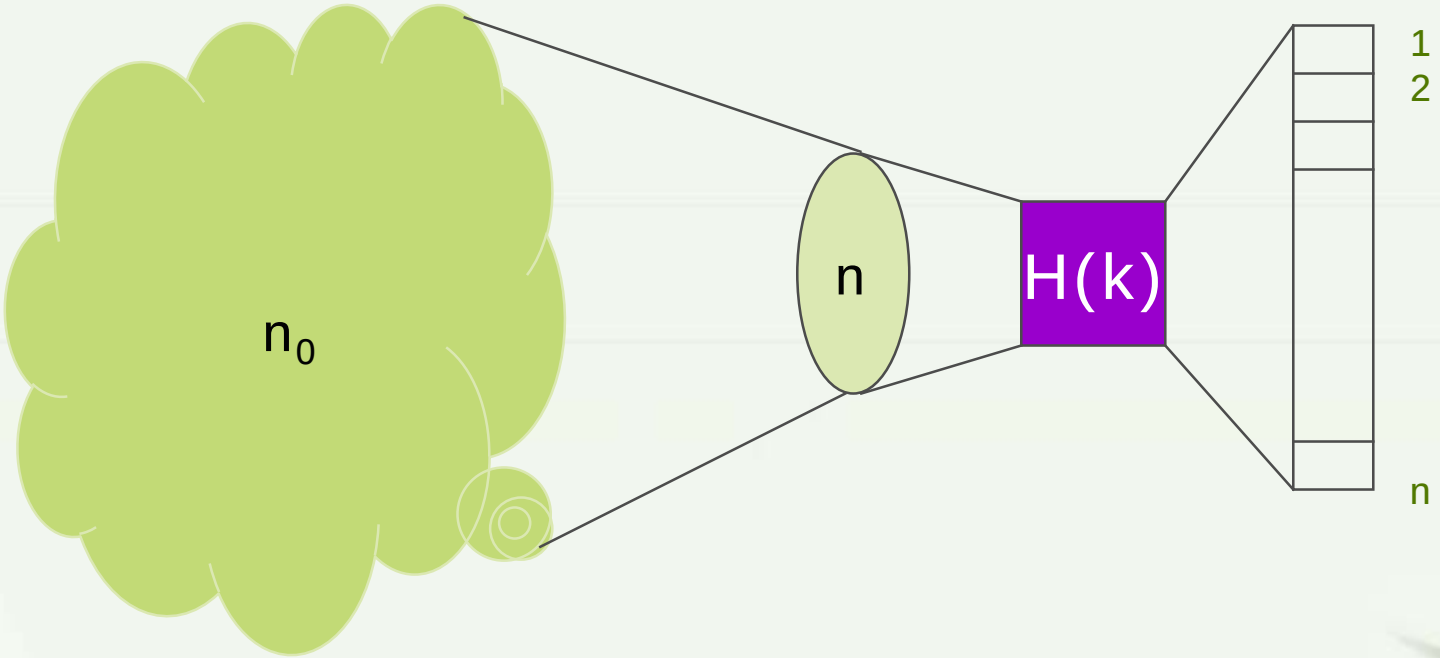
1.3 数据结构的基本概念





1.3 数据结构的基本概念

散列（hashing）映射：



如何映射关系？

关键：

- HASH函数
- 解决冲突

1.3 数据结构的基本概念

因此，我们至少可以得到4×3=12种可能的物理数据结构：

逻辑结构
Lists
Trees
Graphs

存储结构
Sequential
Linked
Indexed
Hashing

同一逻辑结构采用不同的存储方法，可以得到不同的存储结构。选择何种存储结构来存储相应的逻辑结构，具体问题具体分析，主要考虑运算方便及算法的时空要求。

1.3 数据结构的基本概念

- 数据结构和操作

研究数据结构的目的是为了加工和处理，所谓的加工和处理就是对数据结构实施一些操作（或运算），因此数据结构和操作是分不开的。

因此，数据结构主要有三个方面的内容：数据的逻辑结构；数据的物理存储结构；对数据的操作（或算法）。

1.4 抽象数据类型

- 抽象思维方法

舍去复杂系统中非本质的细节，只把其中某些本质的、能反映系统重要宏观特性的东西提炼出来，构成系统的模型，并且深入研究这些特性。

在思考一个复杂问题的解决时，首先应考虑能否将它抽象简化，否则很难理解或者实现它！

1.4 抽象数据类型

- 数据类型

“数据类型”是大家非常熟悉的一个概念，那么，什么是数据类型？

[数据类型 Data Type] 一个数据值的集合和定义在这个值集上的一组操作的总称。

注意： 数据类型是一个非常重要的概念，要正确理解它！！

（1） 高级语言中的数据类型实际上包括：数据的逻辑结构、数据的存储结构及所定义的操作的实现。

（2）高级语言中的数据类型按值的不同特性分为：

原子类型（如整型、实型、字符型、布尔型）

结构类型（如数组）

1.4 抽象数据类型

（3）数据类型并不局限于高级语言，它实际上是一个广义的概念。

例如：“教师”就是一个数据类型，他有值“教龄”，有操做“教书”等；如果具体说小学教师、大学教师，可以看作是一个具体的类型（好象有了存储结构）；

（4）撇开计算机，现实中的任何一个问题都可以定义为一个数据类型

——称为抽象数据类型

[抽象数据类型 Abstract Data Type，ADT] 一个数学模型及定义在这个模型上的一组操作（或运算）的总称。

1.4 抽象数据类型

- 抽象数据类型和数据结构

抽象数据类型 = 数学模型+操作
= 数据结构+操作
= 数据+结构+操作

它是一种描述用户和数据之间接口的抽象模型，亦即它给出了一种用户自定义的数据类型。

一个ADT可以用三元组表示：（D，S，P）

其中：D是数据对象；

S是D上的关系集；

P是对D的基本操作集。

1.4 抽象数据类型

- 抽象数据类型的定义及描述

ADT 抽象数据类型的名称

数据结构（数学模型）：
- 数据元素(D)
- 结构(S)

操作(P)

END ADT

1.4 抽象数据类型

- 用面向对象的规范描述ADT

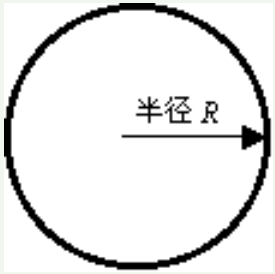
```
ADT   ADT名称  is
      Data
         描述数据的结构
      Operations
         构造函数
         Initial values：用来初始化对象的数据
         Process：初始化对象
         操作1
            Input,Preconditions,process,output,postconditions
         操作2
            .............
         操作n
end ADT ADT名称
```

1.4 抽象数据类型

- **ADT定义举例**

例1. 计算圆的周长和面积

圆是平面上与圆心等距离的所有点的集合。从**图形显示角度看**，圆的抽象数据类型包括圆心和半径；而从**计量角度看**，它所需要的抽象数据类型只需半径即可。如果从计量角度来给出圆的抽象数据类型，那么它的数据取值范围应为半径的取值范围，即为非负实数，而它的操作形式为确定圆的半径（赋值）、求圆的面积、求圆的周长等。



1.4 抽象数据类型

问题描述：给定圆的半径，求其周长和面积

ADT定义：

ADT *circle*
　　data : 0≤r ,实数
　　structure: NULL
　　operations: area {计算面积 s= π r^2*}*
　　　　circumference {计算周长 l=2 π r*}*
END ADT

1.4 抽象数据类型

例2. 复数的运算

问题描述： 在高级语言中，没有复数类型，但是我们可以借助已有的数据类型解决复数类型的问题，如复数运算。

ADT定义：

ADT *complex*

data : c_1,c_2 {c_1,c_2均为实数}

structure: $z=c_1+c_2i$

operations: *create*(z,x,y) {生成一复数}

add(z_1,z_2,s) {复数加法}

subtract(z_1,z_2,difference) {复数减法}

multiply(z_1,z_2,product) {复数乘法}

get_realpart(z) {求实部}

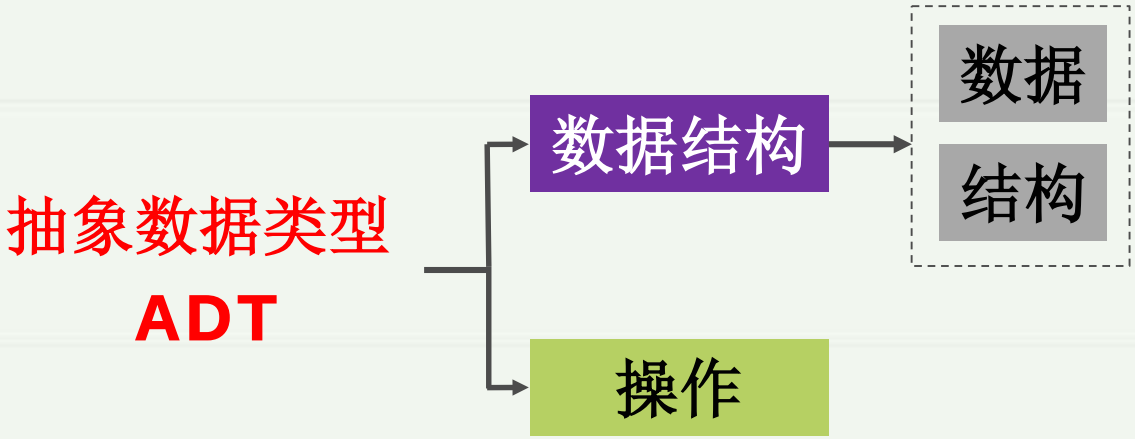
get_imagpart(z) {求虚部}

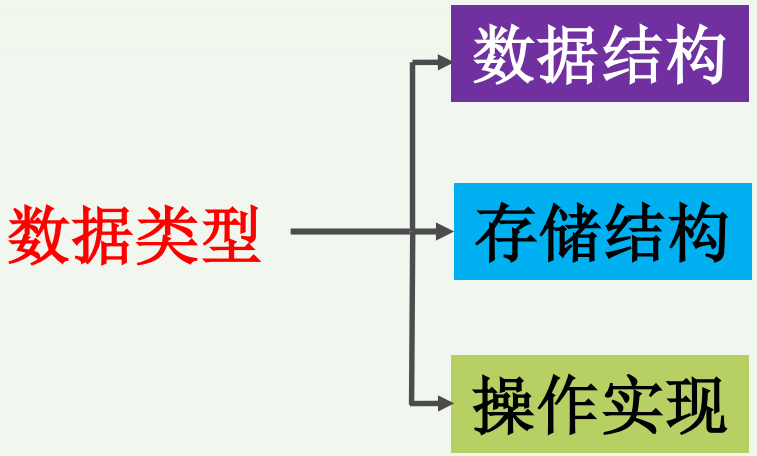
printc(z) {输出一复数}

END ADT

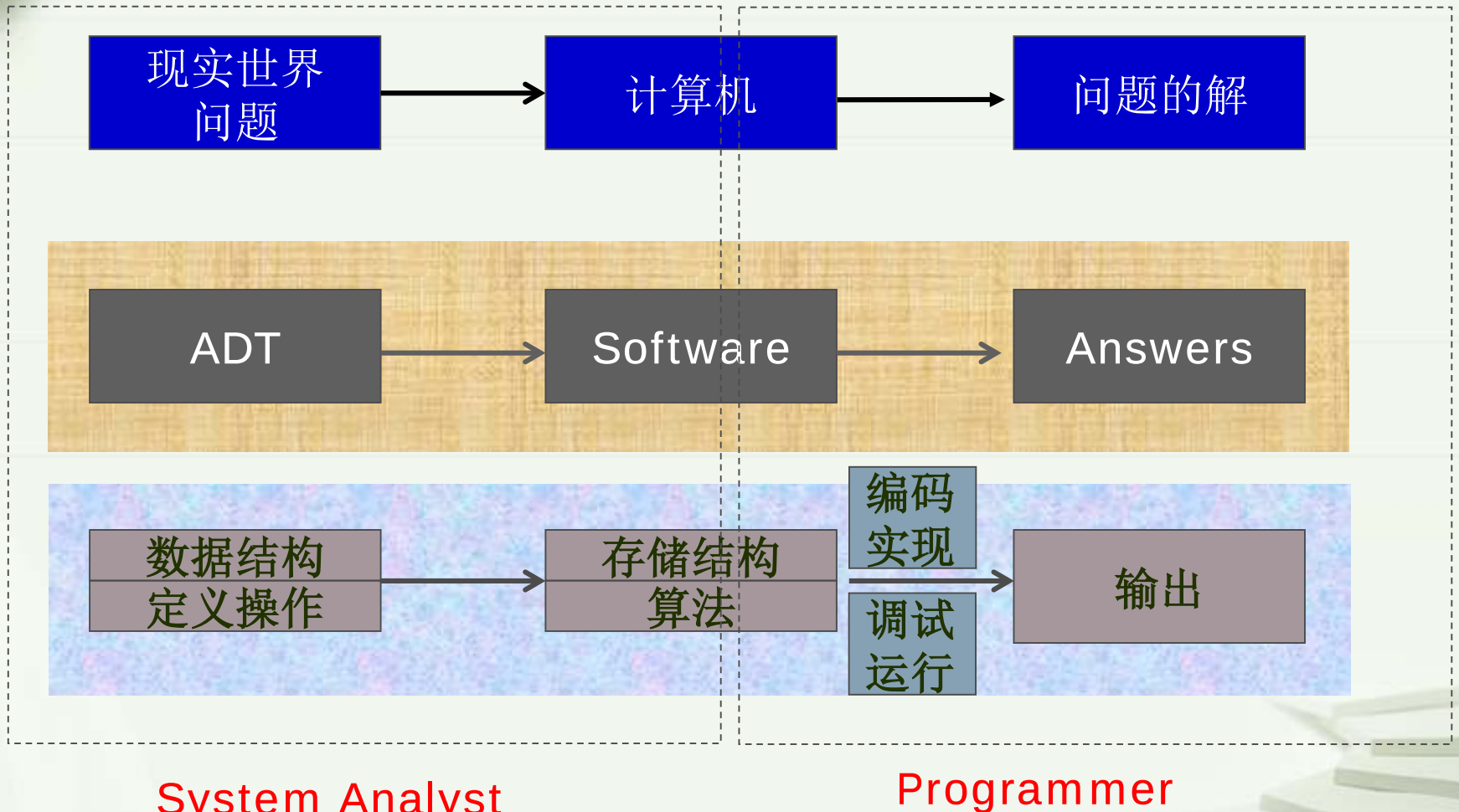
1.4 抽象数据类型

- 抽象数据类型和数据类型





1.4 抽象数据类型


现实世界
问题
计算机
问题的解
ADT
Software
Answers
数据结构
定义操作
存储结构
算法
编码
实现
调试
运行
输出
System Analyst
Programmer

1.5 ADT与C++类

抽象数据类型、面向对象、建模等概念在本质上是一致的！

- 抽象数据类型可以看作是描述问题的模型，它独立于具体实现。其优点是将数据和操作封装在一起，使得用户程序只能通过在ADT里定义的某些操作来访问其中的数据，从而实现了信息隐藏。

- 在C + + 中，可以用类（包括模板类）的说明来表示ADT，用类的实现来实现ADT。因此，C + + 中实现的类相当于是数据结构及其在存储结构上实现的对数据的操作。

- ADT和类的概念实际上反映了程序或软件设计的两层抽象：ADT相当于是在**概念层（或称为抽象层）**上描述问题，而类相当于是在**实现层**上描述问题。而对象或类的实例则可以看作是**应用层**。

关于 ADT与C++类，请同学们自学
（教材 1.3节）

1.6 算法的基础知识

- 算法的定义

算法(Algorithm)：简单说，就是解决问题的一种方法或过程，由一系列计算步骤构成（目的是将问题的输入变换为输出）。即，它是一个定义良好的计算过程，它以一个或一组值作为输入，并产生一个或一组值作为输出。

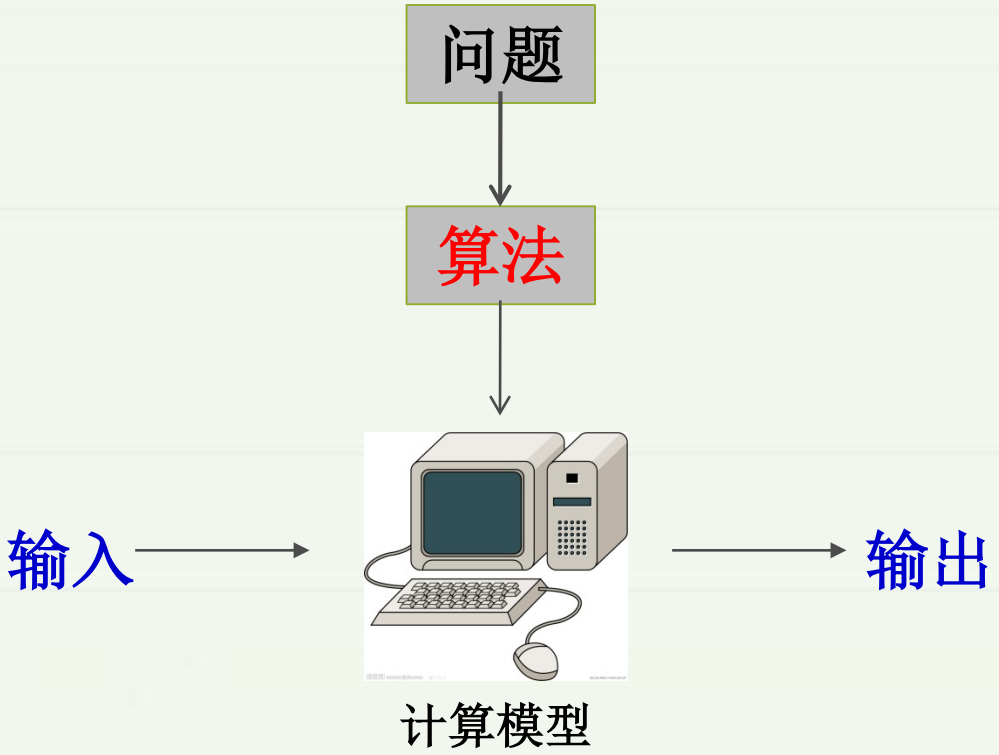
- Algorithm: a method or a process followed to solve a problem.
- A mapping of input to output（Or Transformation from input to output）
- A problem can have many algorithms.

1.6 算法的基础知识

- 算法的定义

$$y = f(x)$$

x 是输入
y 是输出



算法与问题：

如果问题的一个算法能应用于该问题的任何一个实例，得到该问题的解，称这个算法解决了这个问题。

1.6 算法的基础知识

- 算法的特性

（1）有穷性/终止性：有限步内必须停止；

（2）确定性：每一步都是严格定义和确定的动作；

（3）能行性：每一个动作都能够被精确地机械执行；

（4）输入：满足给定约束条件的输入（可以没有）；

（5）输出：满足给定约束条件的结果（必须有）；

算法与程序：

程序是使用某种语言编写出来的、计算机执行的代码集合。一个算法用某种语言写出来就是程序，但程序描述的不一定都是算法。

1.6 算法的基础知识

- 算法的设计

算法设计是求解问题的关键，也是软件的核心。这是人的脑力创新的一种表现形式。算法设计是一种艺术。但人们总结出了一些算法设计策略：

最朴素的策略：蛮力

高级策略：

（1）分治策略

（2）贪心策略

（3）动态规划策略

（4）回溯策略

……

1.6 算法的基础知识

- 算法的描述

（1）自然语言描述
特点：容易，但有时罗嗦、有二义性；

（2）图示（流程图、N-S图、PAD图等）
特点：直观清晰，但不宜实现；

（3）算法语言（伪代码）
特点：严谨、简洁，易程序实现；

（4）程序设计语言
特点：可以直接运行，但太严格；

1.6 算法的基础知识

- 算法的评价标准

✓ 正确性(Correctness)

✓ 易读性(Readability)

✓ 健壮性(Robustness)

✓ 有效性(Efficiency)：时间、空间

1.6 算法的基础知识

- 算法的性能分析

计算机可以做的很快，比如现在的千万亿次/秒，但是还不能无限快；存储器可以做到很便宜、很大，但是不会免费和无限大。因此计算时间和存储空间是一种有限资源，既然是有限资源就必须要有效使用，因为人的追求应用更广、完成更大、速度更快、效果更好……的欲望是无止境的!

解决同一问题的各种不同方案（核心就是算法）的资源效率有时差别很大，这种差距的影响往往比硬件和软件方面的差距还要大。

[算法分析 Algorithm Analysis] 估量一个算法效率的方法，包括运行时间效率和空间效率——算法的复杂性

1.6 算法的基础知识

看一个例子：

计算机A，每秒能执行10亿条指令；机器语言编程；用插入排序来对n个数进行排序需要$2n^2$条指令。

计算机B，每秒执行1000万条指令；高级语言编程；用归并排序来对n个数进行排序需要50nlgn条指令。

现在有100万个数要进行排序，两台计算机所用的时间分别为：

计算机A花时间：$\dfrac{2\bullet(10^6)^2\text{条指令}}{10^9\text{条指令/秒}}=2000\text{秒}$

计算机B花时间：$\dfrac{50\bullet10^6\lg10^6\text{条指令}}{10^7\text{条指令/秒}}\approx100\text{秒}$

可以看出，尽管计算机B速度慢，而且是效率低的编译器，但是其排序速度比计算机A快了近20倍！而且随着数据规模的变大，这种差距更大！（例如当数据达到1000万，插入排序要2.3天，归并排序只要20分钟）。

1.6 算法的基础知识

- 算法性能的分析方法

经验测试（后期测试）：选择样本数据、运行环境运行算法计算出空间、时间。

先验估计（事前估计）：根据算法的逻辑特征（基本数据量、操作）来估算。

优点：精确
缺点：可比性差，效率低

优点：可比性强
缺点：不精确，仅仅是估计

那么，如何撇开计算机本身来估算一个算法的复杂性呢？

1.6 算法的基础知识

- 算法的空间性能分析——空间复杂度

（1） 算法的空间性能的影响因素

- 指令空间（由机器决定）；
- 数据空间（常量、变量占用空间）；
- 环境栈空间；

数据空间—用来存储所有常量和变量的值。 分成两部分：

存储简单量

存储复合变量

1.6 算法的基础知识

- 算法的空间性能分析——空间复杂度

（2）度量方法：

单个常量、变量：由机器和编译器规定的类型存储决定

数组变量：所占空间等于数组大小乘以单个数组元素所占的空间。

结构变量：所占空间等于各个成员所占空间的累加；

例如：

double a[100]; 所需空间为100×8＝800

int matrix[r][c]; 所需空间为 2×r×c

1.6 算法的基础知识

- **算法的时间性能分析**——时间复杂度

（1）算法的时间性能的影响因素：

- 机器的运行速度（执行代码的速度）；
- 书写程序的语言；
- 编译产生代码的质量；
- 算法的策略；
- 问题的规模；

1.6 算法的基础知识

- 算法的时间性能分析——时间复杂度

（2）算法的时间性能度量：

令T(P)表示算法P需要的时间，则有：

$$T(P) = c + t_P(\text{实例特征})$$

$$t_P(n)=c_aADD(n)+c_sSUB(n)+c_mMUL(n)+c_dDIV(n)+\cdots$$

其中，c_a，c_s，c_m，c_d表示一个加、减、乘、除操作所需的时间，函数ADD，SUB，MUL，DIV分别表示P中所使用的加、减、乘、除等基本操作的次数。

即T(P)是算法中一些基本操作的执行次数的累加和。

它一般是关于规模(n)的函数

1.6 算法的基础知识

- 算法的时间性能分析——时间复杂度

（3）渐近时间复杂性：

统计出算法的基本操作次数和T(P) 后，怎么下结论呢？

例如：T(P1)=1000n+28

T(P2)=n^2

问：哪个算法更好？

既然我们的目的是为了比较，就应该将注意力集中在被比较的对象之间最主要的差别。

1.6 算法的基础知识

- 算法的时间性能分析——时间复杂度

这样，就人们找到了一个关注点， 算法性能的增长率！

算法的增长率(growth rate)：是指当问题规模增长时，算法代价的增长速率。具体就体现在两个函数的变化趋势上。

T(n)=f(n)——时间代价

S(n)=g(n)——空间代价

简单说，就是看看这个算法的资源消耗的增长有多快，增长快的（阶高），其资源消耗就大，复杂性就高！

——这就是复杂性渐近态的思想

1.6 算法的基础知识

- 算法的时间性能分析——时间复杂度

例如： $T_1(n)=3n^2+4n\log n$

$T_2(n)=2n^2$

在n很大后，两个函数变化趋势接近一致。可以说$n\to\infty$时，$T_1(n)$渐近于$T_2(n)$，可以用简单的$T_2(n)$来表示算法的复杂性。当然，更简单的函数是n^2。 显然，应该用最简单的函数来代表增长率一样的这一类。

因此，渐近复杂性分析只关心度量函数的阶就够了，因为按照渐近的思想，度量函数中的常数因子对算法复杂性的影响不敏感。

可以看出，渐近复杂性分析主要专注于问题的“规模”、“基本的操作”和增长率。

1.6 算法的基础知识

- **算法的时间性能分析**——时间复杂度

复杂性的渐近态

对算法，精确计算其增长率有时很难，如果能粗略比较出哪个更快或更慢就可以了。于是在关注增长率的基础上提出渐近估计的概念。即规模变大时，趋势渐近于什么。即**“和谁差不多一样快”**

渐近复杂性分析（Asymptotic Algorithm Analysis）

确切说：渐近分析是指当输入规模很大，或者说大到一定程度时对算法的研究和分析。（从微积分意义上说是达到极限时）。 这样得到的就是算法的渐近复杂性。

1.6 算法的基础知识

渐近时间复杂度的本质：

从增长率可以看出：算法不同，算法的时间复杂性不同（增长率不同），当计算机速度提高时，你得到的解决问题规模的**增益**是不同的！

例如，当计算机的运行速度是提高为原来的**10**倍时，不同的算法在解决问题的规模上得到的收益是有很大不同的。

f(n)	n	n’	变化	n’/n
10n	1000	10000	n’=10n	10
20n	500	5000	n’=10n	10
5nlogn	250	1842	3.16n<n’<10n	7.37
$2n^2$	70	223	n’= $\sqrt{10}$ n	3.16
2^n	13	20	n’=n+7	-

注：
n是原来规模；
n’是现在规模；

1.6 算法的基础知识

最好、最坏、平均时间复杂性：

算法性能还与输入样本的分布有关，这样，同样的规模，如果输入样本分布不同，其算法性能可能有不同的表现，于是就有了：

最好时间复杂性：

最坏时间复杂性：

平均时间复杂性：

问题的时间复杂性怎么定义？

解决一个问题的所有算法中，时间复杂性最好的那个算法的时间复杂性定义为问题的时间复杂性。（即代表着问题目前求解的最好策略！）

1.6 算法的基础知识

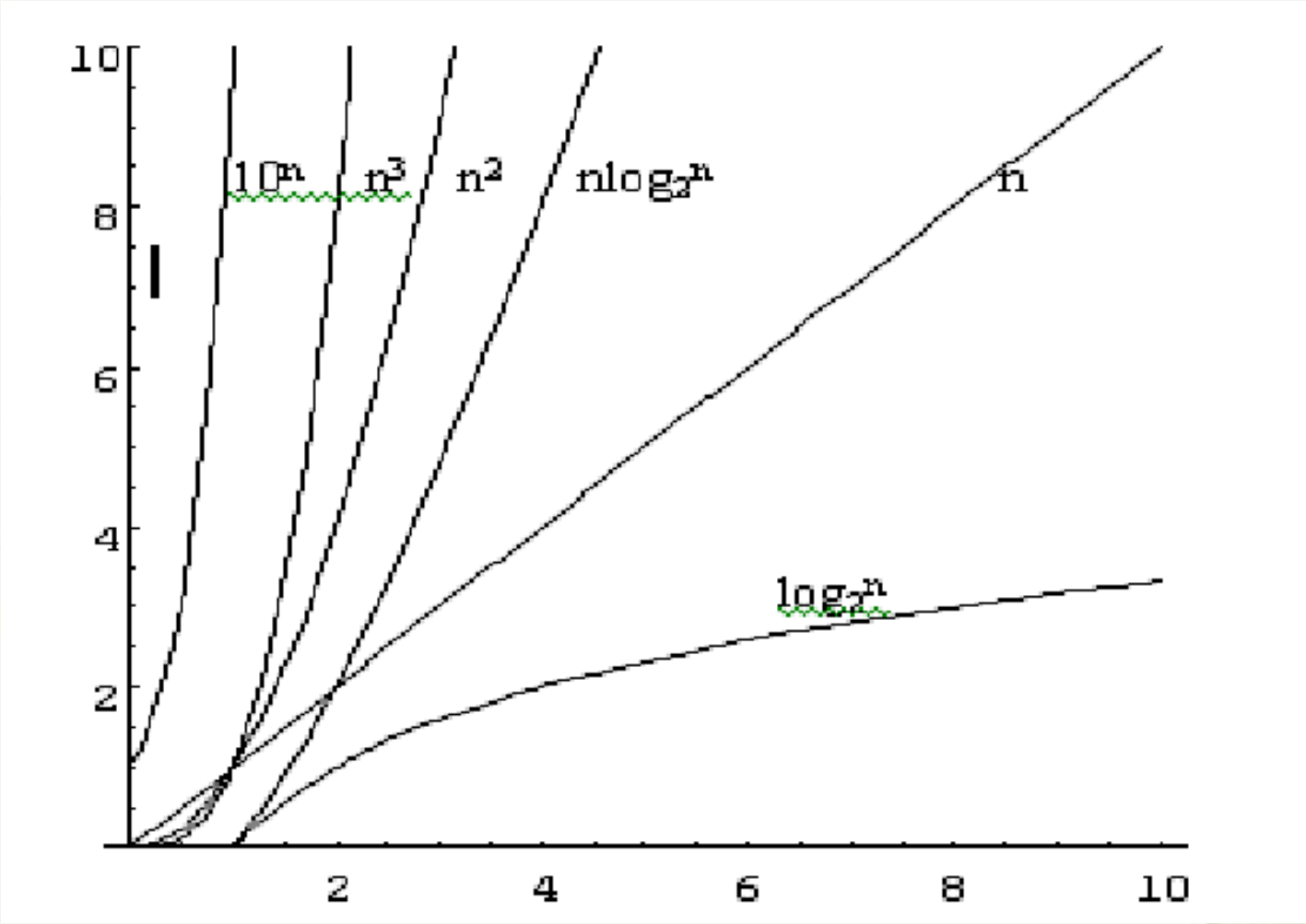
渐近时间复杂性的数学符号：

渐近上界：O

如：f(n)=(n+1)

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

1.6 算法的基础知识



f(n)函数曲线变化速度的比较

内容小结

1. 如何理解抽象数据类型和数据类型的概念？
2. 算法的评价标准有哪几个？在数据结构中重点关注什么？
3. 算法性能的评价分为哪两种方法？分别有什么优缺点？
4. 渐近时间复杂性是怎么来的？

1.6 算法的基础知识

算法分析的一般步骤：

（1）确定问题规模参数；

（2） 确定关键（基本）操作、存储；

（3）建立各个量之间的“和”关系模型；

（4）利用数学方法求出累计和函数T(n),S(n)；

（5）给出渐近表示。

1.6 算法的基础知识

- 一般算法的时间复杂性分析：

1. 决定哪个（哪些）参数表示输入规模；
2. 确定基本操作。赋值、读、写等一般看作基本操作；
3. 划分三种控制结构；
4. 对各种控制结构，有
   （1）顺序操作序列，可以累加或取最大值；
   （2）分支的条件判断一般看作基本操作，分支结构的时间定义为条件成立和不成立时的和；
   （3）循环结构的时间为循环体的时间乘循环次数；
5. 函数调用时要考虑函数的执行时间。
6. 建立其基本操作执行次数的“求和”表达式。
7. 利用数学知识求和得到其T(n)
8. 确定阶，得到渐近表示：一般给出O表示。

1.6 算法的基础知识

- 一般算法的时间复杂性分析：

例1. 分析下面程序段表示的算法的时间复杂性：

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

T(n)=1+1+n+2(n-1)=3n

T(n)=O(n)

1.6 算法的基础知识

- 一般算法的时间复杂性分析：

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

$T(n)=n+1+n(n+1)+2(n*n)=3n^2+2n+1$

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

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

$$T(n)=n+1+\sum_{i=1}^{n}(i+1)+\sum_{i=1}^{n}i$$

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

1.6 算法的基础知识

- 一般算法的时间复杂性分析：

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

$T(n)=1+\log_2 n+1+\log_2 n=2\log_2 n+2$

$T(n)=O(\log_2 n)$

1.6 算法的基础知识

- 一般算法的时间复杂性分析：

例2. 有算法如下：

```
MaxElementA[0..n-1])
  {
     //功能：？
     //输入：数组A[0..n-1]
     //输出：？
     maxval=a[0];
     for(i=1;i<=n-1;i++)
        if(a[i]>maxval)
           maxval=a[i];
     return maxval;
  }
```

$T(n)=O(n)$

1.6 算法的基础知识

- 一般算法的时间复杂性分析：

例4. 有算法如下(p34 程序1.26)

```
void example(float x[][n],int m)
{  float sum[m];int i,j;
   for(i=0;i<m;i++)
   { sum[i]=0;
     for(j=0;j<n;j++)
         sum[i]=sum[i]+x[i][j];
   }
  for(i=0;i<m;i++)
    cout<<“Line”<<i<<“:”<<sum[i]<<endl;
}
```

T(m,n)=O(m×n)

1.6 算法的基础知识

- 一般算法的时间复杂性分析：

例5. 分析顺序查找算法的复杂性(p36 程序1.27)

```
int SequenceSearch(int a[],int n,int key)
{ //若找到，则返回元素的下标，否则返回-1；
   for(int i=0;i<n;i++)
      if(a[i]==key) return i;
   return -1;
}
```

$T_{min}(n)=O(1)$

$T_{max}(n)=O(n)$

$T_{ave}(n)=O(n)$

```
int SequenceSearch (int a[],int n,int key)
{ a[0]=x;
  i=n;
  while(a[i]!=x)
    i=i-1;
  return i
}
```

本章小结

重点和难点：

（1）数据结构的重要性：研究问题本身的数据特性，并在计算机中存储，以实现问题的高效求解。

（2）基本概念：数据结构、存储结构、操作、抽象数据类型。

数据结构的种类；

存储结构的种类，分别是如何映射数据结构的？

如何定义一个抽象数据类型；

抽象数据类型的实现；

本章小结

重点和难点：

（3）算法的定义：输入转换为输出

（4）算法特性:5个

（5）算法的评价（有几个指标？）

- 时间、空间效率
- 事前还是事后？
- 空间性能如何评价？
- 时间性能如何评价？
- 渐近复杂性的由来

本章小结

重点和难点：

- 渐近时间复杂性的本质

  ■同一问题，规模相同，用不同的算法解决，所花费时间是不同的；
  ■同一问题，用不同的算法解决，在相同的时间内所能解决的问题规模大小不同；

- 能基本掌握算法的分析方法

  ■确定基准——基本操作
  ■求累加和
  ■渐近表示

本章小结

思考：

1. “复杂性渐进阶比较低的算法总是比复杂性渐进阶比较高的算法有效”，这种说法正确吗？

2. 有5个算法如下，你可以得出什么分析结论？

算法	T(n)	时间复杂度
A_1	1000n	O(n)
A_2	100nlogn	O(nlogn)
A_3	$10n^2$	$O(n^2)$
A_4	n^3	$O(n^3)$
A_5	2^n	$O(2^n)$

结论：

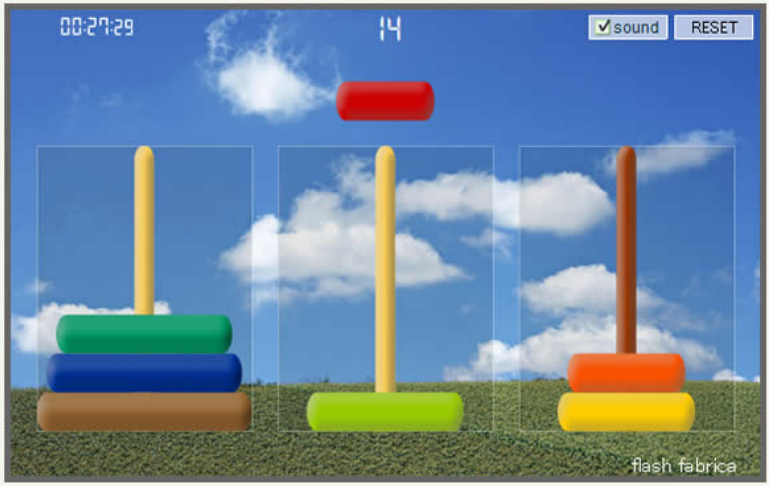
$2 \le n \le 9$	A_5最好
$10 \le n \le 58$	A_3最好
$59 \le n \le 1024$	A_2最好
$n > 1024$	A_1最好

本章小结

课外兴趣编程（1）：

自己设计开发一款Hanoi游戏。功能包括：

1. 可以自己输入盘子数、自己定义源和目标柱；
2. 可以让计算机自动演示移动过程，能自己调节移动速度。移动过程中同时文字显示，例如a->b。实时显示移动步数。
3. 用户手工移动（通过键盘或鼠标），记录并显示移动轨迹和次数。如果移动失败可以复位重新开始。
4. 可以多次玩，不想玩了就退出。




END