数据结构

Data Structures

王玮

2023年春季学期•深圳

课程安排

- 课程编号：COMP2022
- 授课学时：40（4-14周，4学时/周）
- 实验学时：16（4/7、4/9、4/11、4/14周）

- 主讲教师：王玮
  - Email：wangwei2019@hit.edu.cn

- 助教：林堉欣、卜贤达

- 考核形式：
  - 期末笔试70% + 实验20% + 平时和作业10%

群聊二维码

群名称:2023春数据结构
群　号:621715655

投票

最多可选2项

设置

比较喜欢哪种形式的课堂互动?

A 主动回答，大胆开麦

B 弹幕互动，一键三连

C 点名提问抽抽乐

D 随堂测验

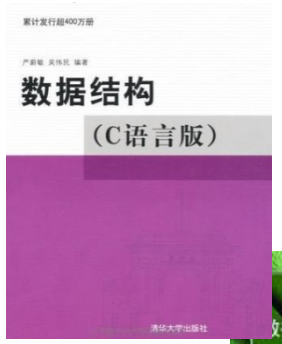
E 我有更好的想法

提交

教材选择

- 课程资源：
  - 课件PPT：集成了各个教材内容，可作为简要学习内容。
  - 主要教材：
    - 《数据结构（C语言版）》
      - 严尉敏、吴伟民 编著，清华大学出版社
  - 参考书目：
    - Data Structures and Program Design in C++

      数据结构与程序设计——C++语言描述（影印版）
      - Robert L. Kurse, Alexandeer J. Ryba
    - 数据结构与算法（第4版）
      - 廖明宏、郭福顺、张岩、李秀坤，高等教育出版社，2007.
  - INTERNET



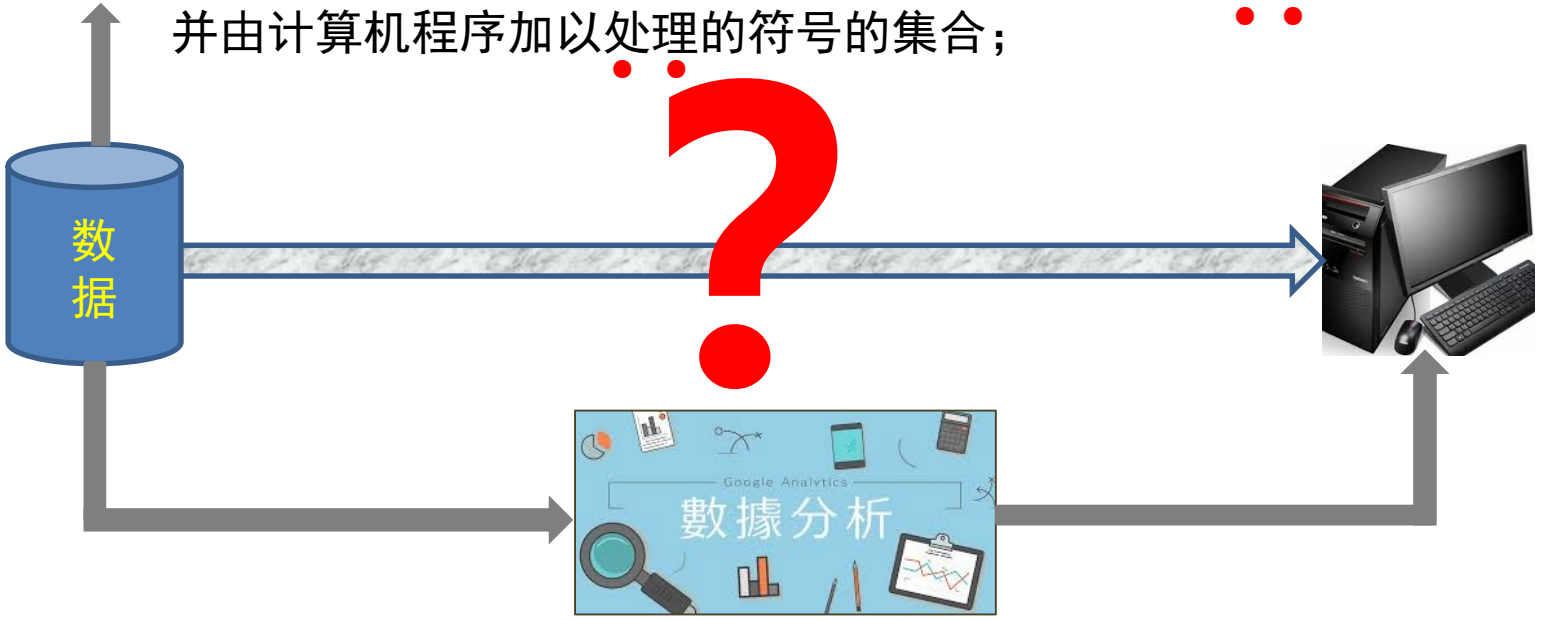
课程地位

- “数据结构”
  - 是计算机专业的核心课程之一，也是其他非计算机专业的主要选修课程之一。
- “数据结构”在计算机科学中是一门综合性的专业基础课
  - 是计算机硬件、操作系统、编译原理、计算机网络、数据库系统及其他系统程序和大型应用程序的重要基础。
- “数据结构”正在不断发展
  - 1968年以前，数据结构的内容大多包含在形如表、树、图论、集合代数论、格、关系等方面。1968年开始，“数据结构”逐渐开始成为独立的一门课程；
  - 例面向各专门领域中特殊问题的数据结构的研究和发展，如：多维图像数据结构等。

课程简介

数据结构在学什么？

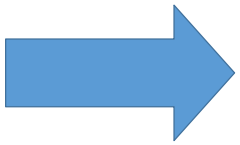
数据：是用以描述客观事物的数值、字符，以及一切可以输入到计算机中并由计算机程序加以处理的符号的集合；



- 如何用程序代码把现实世界的问题信息化
- 如何用计算机高效地处理这些信息从而创造价值


我举个栗子

存钱




17:26
零钱明细
¥
我的零钱
¥128.99
转入零钱通 给自己加加薪 ›
充值
提现
常见问题
本服务由财付通提供

浮点型变量


举两个栗子

排队


14:40
排号成功
您的餐位由门店安排!
益田假日里店
还需等待 3桌
我的号码 17号
当前叫号 14号 小桌
就餐人数 4人
取号时间 2月15日 14:40
门店地址 深圳市南山区桃源街道平山社区留仙大道4168号益田假日里广场5楼
取消排号
预点餐
分享给好友

数组？


举个栗子

小伙伴


我
粉丝
发现用户
新的好友
全部粉丝
本地资
(武汉)
来自荣耀6
+关注
没个性就是有个性
互相关注
让自己快乐快乐这才叫做意义
互相关注
插画
来自微博 weibo.com
互相关注
暂无简介
互相关注

？ ？ ？

数据结构在学什么？

- 如何用程序代码把现实世界的问题信息化
- 如何用计算机高效地处理这些信息从而创造价值

课程一览

- 两个主题：
  - 数据结构：数据及其之间的相互关系，是对数据的抽象。
  - 算法：是求解特定问题的方法，是将输入转为输出的一系列计算步骤。
- 课程的内容组织：
  - 四种基本的数据结构
    - 线性结构：一对一的线性关系；
    - 树型结构：一对多的层次关系；
    - 图型结构：多对多的任意关系；
    - 集合结构：属于同一个集合关系。
  - 查找：线性结构、树型结构和散列结构上查找；
  - 排序：内部排序和外部排序（包括文件）；

课程目标

《数据结构》是计算机大类的专业基础课，既是对前序课程的深入和扩展，也是为将来更加深入地学习其他专业课程打下基础。通过本课程的学习，学生将掌握数据结构和算法基础知识，能够提高程序设计的质量；根据问题的性质，能够选择合理的数据结构，完成算法的设计，并对时间复杂性进行必要的控制，培养解决实际问题的能力。


第1章 绪论
HIT

本章重点与难点

1. 掌握数据结构等基本概念。

2. 数据的逻辑结构、存储结构以及两者之间的关系；

3. 算法及特性；

4. 掌握估算算法时间复杂度的方法。

第一章 绪论

1.1 数据结构及其讨论范畴

1.2 基本概念和术语

1.3 抽象数据类型ADT

1.4 算法和算法分析

第一章 绪论

1.1 数据结构及其讨论范畴

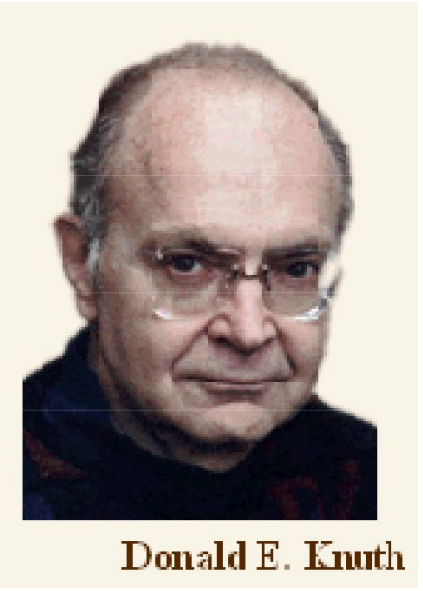
1.2 基本概念和术语

1.3 抽象数据类型ADT

1.4 算法和算法分析

1.1 数据结构及其讨论范畴

- 数据结构的兴起与发展
  - 数据结构的创始人—Donald.E.Knuth



- 1938年出生，25岁毕业于加州理工学院 数学系博士毕业后留校任教，28岁任副教授。29岁，提出“算法”(Algorithm) 和“数据结构”(Data Structure)； 30岁时，加盟斯坦福大学计算机系，任教授。从31岁起，开始出版他 的历史性经典巨著：

  The Art of Computer Programming

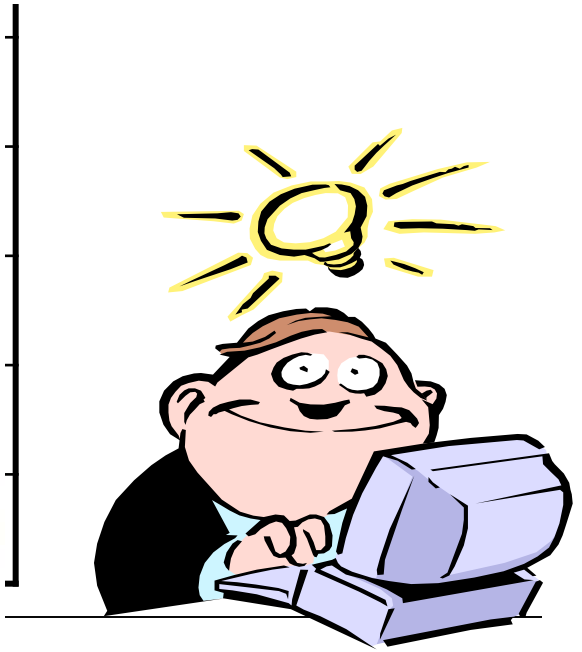
- 他计划共写7卷，然而出版三卷之后， 已震惊世界，使他获得计算机科学界的最高荣誉图灵奖，此时，他年仅36岁。37岁 年当选为美国国家科学院院士。43岁当选为美国工程院院士。

思考 统计总

- 表1是5次C语
  程序计算这5

```c
void main()
{
    int t1,t2,t3,t4,t5;/*各次成绩*/
    int sum;
    t1=98;t2=87;t3=65;t4=54;t5=76;
    sum=t1+t2+t3+t4+t5;
    printf("输出总分: %d\n",sum);
}
```

```c
void main()
{
    int t[5]={98,87,65,54,76};
    int sum=0;
    int i;
    for(i=0;i<5;i++)
    sum+=t[i];
    printf("输出总分: %d\n",sum);
}
```

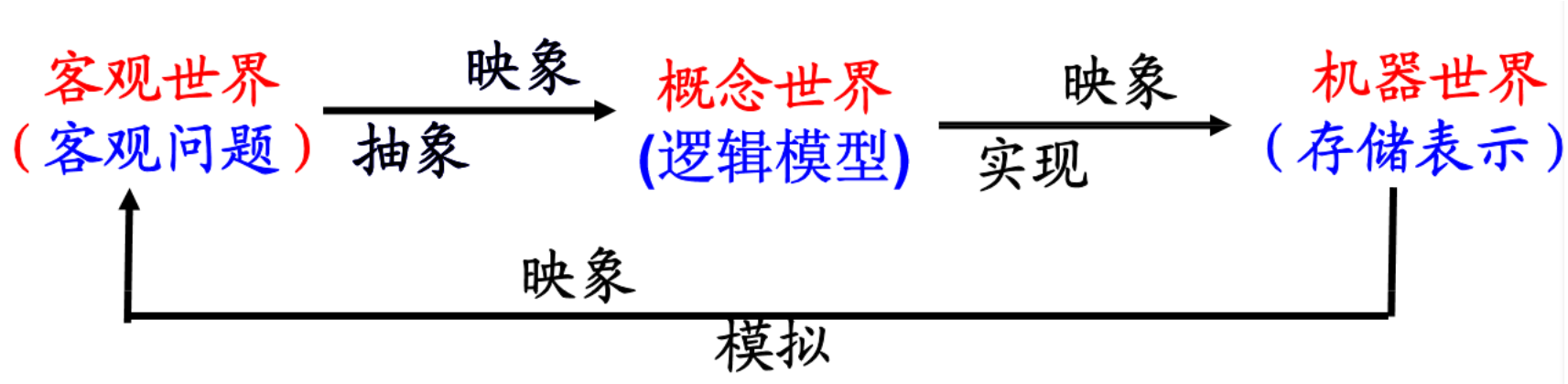


1.1 数据结构及其讨论范畴

- 数据结构问题起源于程序设计
- 程序的实质是什么
  - 程序设计：数据结构（问题的数学模型）和算法（处理问题的策略）。
  - 程序实现：数据表示：将数据存储在计算机中；数据处理：处理数据，求解问题
- 数据结构随着程序设计的发展而发展
  - ① 无结构阶段：在简单数据上作复杂运算
  - ② 结构化阶段：数据结构＋算法＝程序
  - ③ 面向对象阶段：（对象＋行为）＝程序
- 数据结构的发展并未终结。。。

1.1 数据结构及其讨论范畴

- 客观世界与计算机世界的关系
  - 信息是客观世界在人脑中的反映，计算机科学是研究信息表示和信息处理的科学。
  - 数据是信息的载体，信息在计算机内是用数据表示的。
  - 用计算机解决实际问题的实质可以用下图表示：



第一章 绪论

1.1 数据结构及其讨论范畴

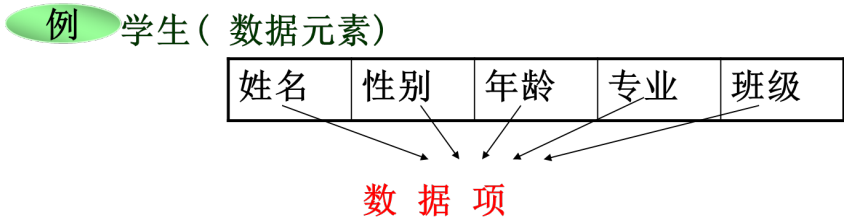
1.2 基本概念和术语

1.3 抽象数据类型ADT

1.4 算法和算法分析

1.2 基本概念和术语

- 数据：一切能输入到计算机中并能被计算机程序识别和处理的符号集合。
  - 数值数据：整数、实数等
  - 非数值数据：图形、图象、声音、文字等



- 数据元素：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。
- 数据项：构成数据元素的不可分割的最小单位。

1.2 基本概念和术语

- 数据对象：具有相同性质的数据元素的集合。
  - 是数据的子集
    - 整数数据对象 $N=\{0, \pm1, \pm2, \ldots\}$
    - 学生数据对象
  - 数据对象中所有成员之间存在某种关系。

    如学生按学号的排序；按性别的分类等。
  - 数据对象中所有成员及其之间关系，是数据结构研究的主要内容。

1.2 基本概念和术语

- **数据结构** 是相互之间存在一种或多种特定关系的数据元素的集合
  - 数据元素之间的相互关系，这种关系是抽象的，即并不涉及数据元素的具体内容。是数据元素及其相互间关系的数学描述 。
  - 数据结构（技术）就是根据各种不同的数据集合和数据元素之间的关系，研究如何表示、存储和操作（查找、插入、删除、修改、排序）这些数据的技术

1.2 基本概念和术语

- 数据结构三要素：
  - ① 数据的逻辑结构
    - 数据元素之间的逻辑关系，是具体关系的抽象。
  - ② 数据的存储结构(物理结构)
    - 数据元素及其关系在计算机内存中的表示；
  - ③ 数据的运算
    - 施加在数据上的运算包括运算的定义和实现。运算的定义是针对逻辑结构的，指出运算的功能；运算的实现是针对存储结构的，指出运算的具体操作步骤

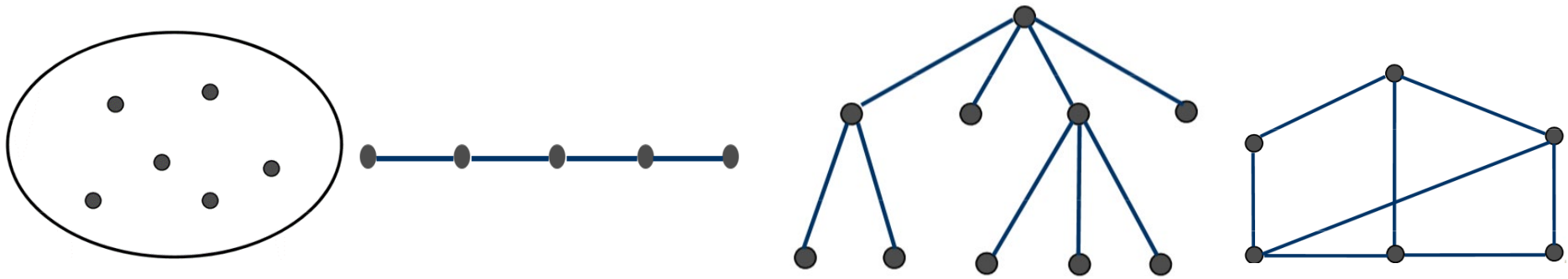
1.2 基本概念和术语

①数据的逻辑结构

- ✓数据的逻辑结构从逻辑关系上描述数据，与数据的存储无关；
- ✓数据的逻辑结构可以看作是从具体问题抽象出来的数据模型；

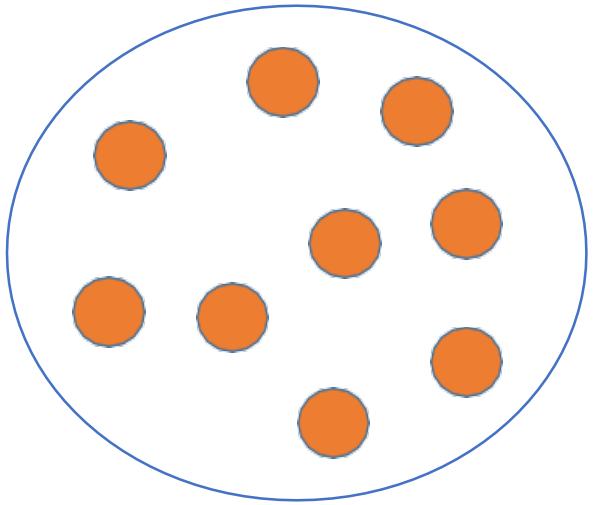
1.2 基本概念和术语

- 数据结构从逻辑上分为四类：
  即四种基本的逻辑结构
  - 集合：数据元素之间就是“属 于同一个集合”；
  - 线性结构：数据元素之间 存在着一对一的线性关系；
  - 树型结构：数据元素之间存在 着一对多的层次关系；
  - 图结构：数据元素之间存在着 多对多的任意关系。



1.2 基本概念和术语

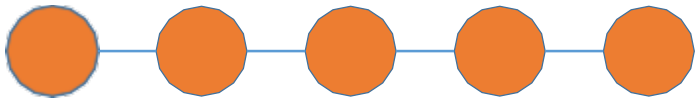
- **数据的逻辑结构：**
  - **集合：**各个元素同属一个集合，别无其他关系



集合

1.2 基本概念和术语

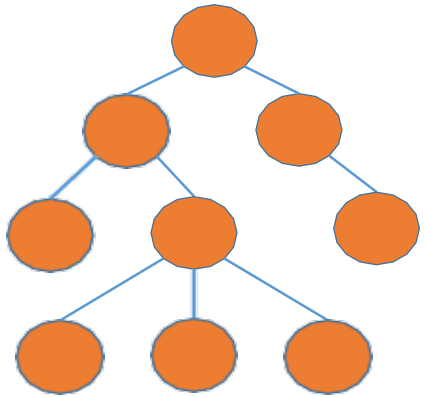
- **数据的逻辑结构：**
  - **线性结构：**数据元素之间是一对一的关系。除了第一个元素，所有元素都有唯一前驱；除了最后一个元素，所有元素都有唯一后继



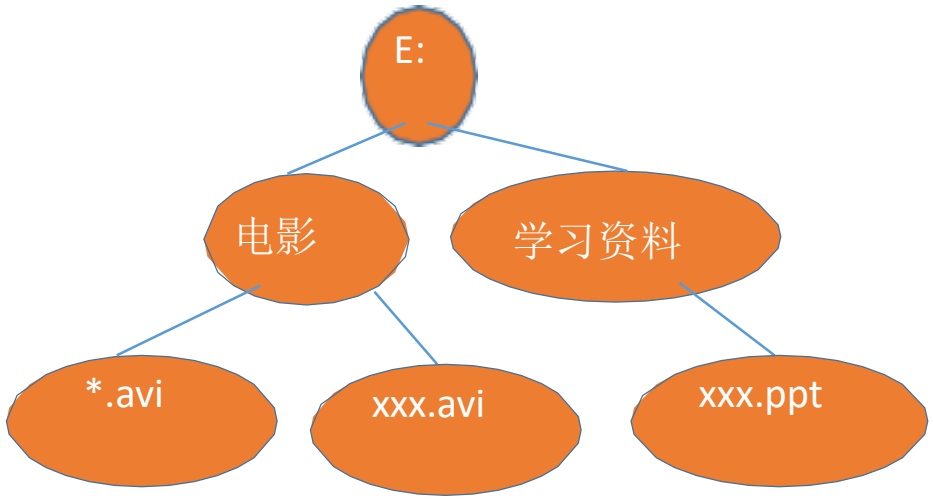
线性结构

1.2 基本概念和术语

- **数据的逻辑结构：**
  - 树形结构：数据元素之间是 一对多的关系

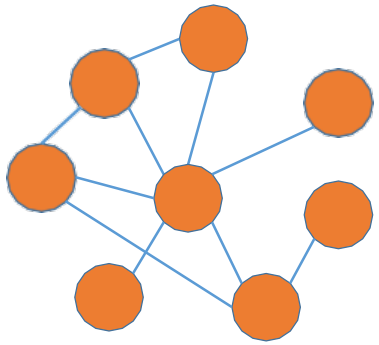


树形结构

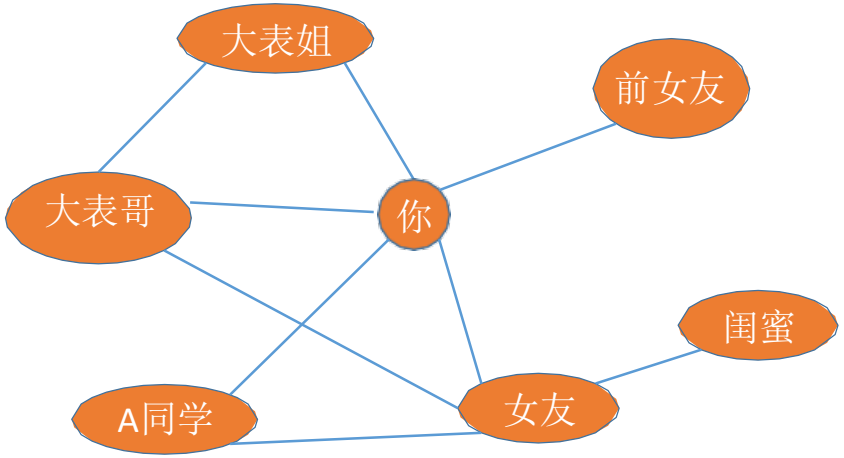


1.2 基本概念和术语

- **数据的逻辑结构：**
  - 图结构：数据元素之间是 多对多的关系



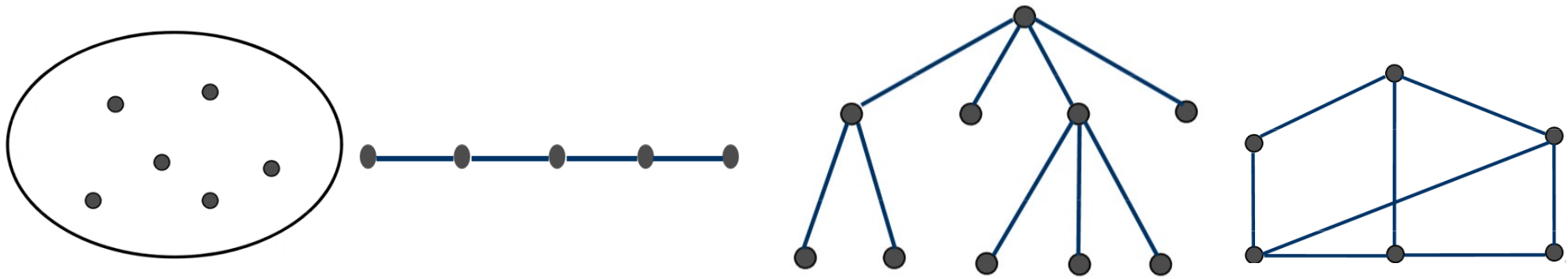
图结构



微信好友

1.2 基本概念和术语

- 数据结构从逻辑上分为四类：
  即四种基本的逻辑结构
  - 集合：数据元素之间就是“属 于同一个集合”；
  - 线性结构：数据元素之间 存在着一对一的线性关系；
  - 树型结构：数据元素之间存在 着一对多的层次关系；
  - 图结构：数据元素之间存在着 多对多的任意关系。



1.2 基本概念和术语

①数据的逻辑结构

形式化表示

数据的逻辑结构通常可以用一个二元组 **DS = （D， R）** 来表示

其中D是数据对象（数据元素的有限集合），R是D中数据元素之间所存在的关系的有限集合。

2、根据二元组关系，画出对应逻辑图形的草图，并指出它们所属的数据结构。

（1）A=（D，R），其中：

D={a，b，c，d，e}，

R={ }

集合

（2）B=（D,R），其中：

D={a,b,c,d,e,f}，R={r}

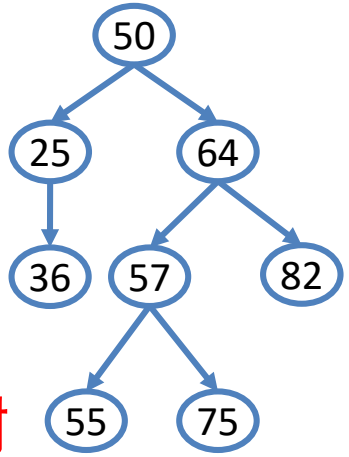
r ={<a,b>,<b,c>,<c,d>,<d,e>,<e,f>}　（尖括号表示结点之间关系是有向的）

线性表

（3）F=（D,R），其中：

D={50,25,64,57,82,36,75,55}，R={r}

r ={<50,25>,<50,64>,<25,36>,<64,57>,<64,82>,<57,55>,<57,75>}

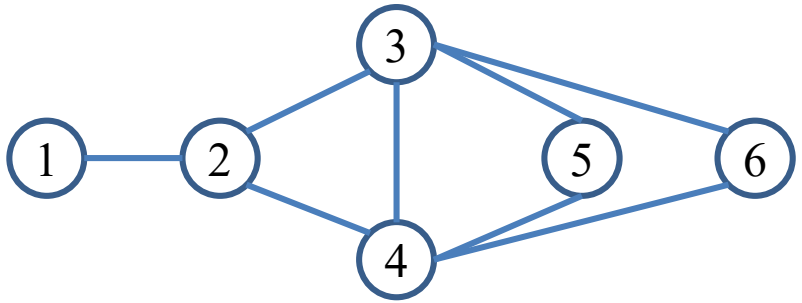


树

（4）C=（D,R），其中：

D={1,2,3,4,5,6}，R={r}

r ={(1,2),(2,3),(2,4),(3,4),(3,5),(3,6),(4,5),(4,6)}（园括号表示结点之间关系是无向的）

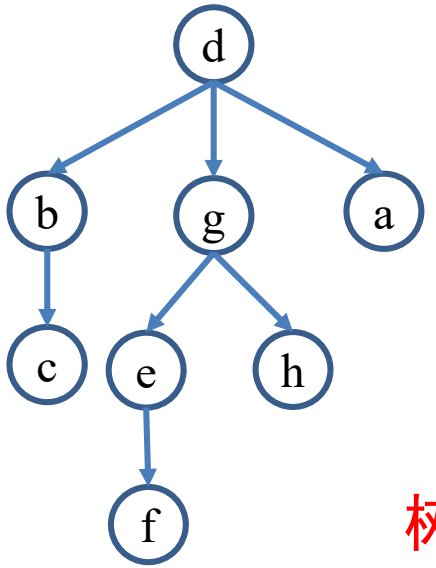


图（无向图）

（5）E=（D,R），其中：

D={a,b,c,d,e,f,g,h}，R={r}

r ={<d,b>,<d,g>,<d,a>,<b,c>,<g,e>,<g,h>,<e,f>}



树

1.2 基本概念和术语

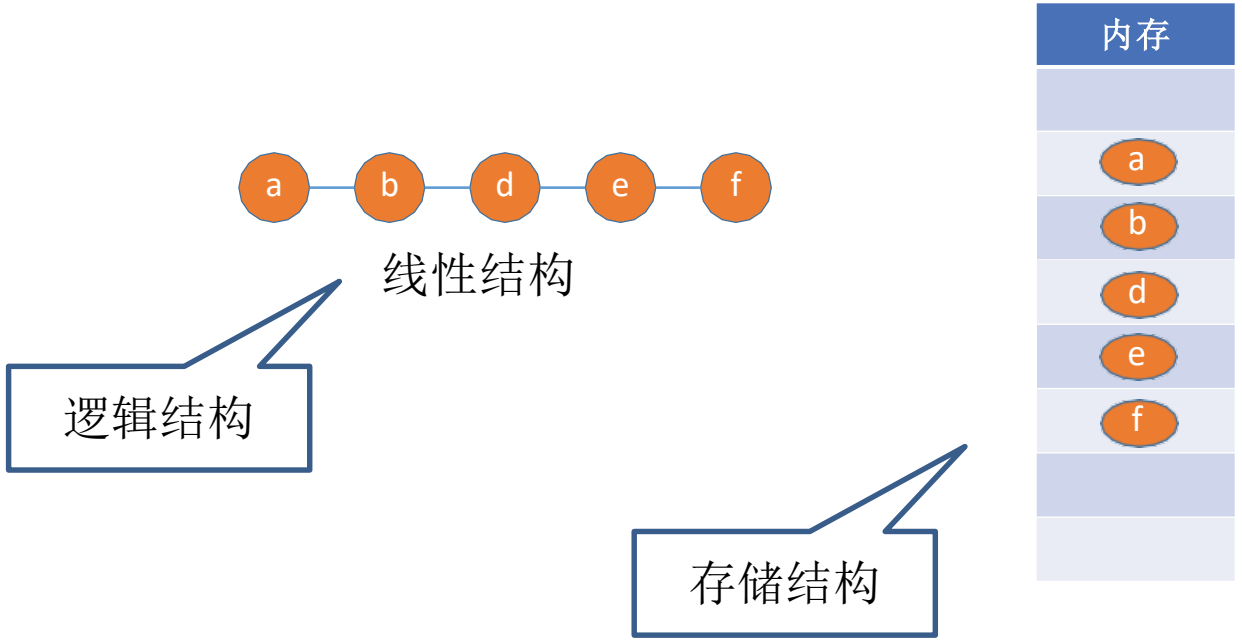
- **②数据的存储结构**
  - 又称物理结构，是数据及其逻辑结构在计算机中的表示。
  - 实质上是内存分配，以确定元素及元素之间关系的表示。
  - 在具体实现时，依赖于计算机语言。

几种主要的存储结构：

- 顺序存储
- 链式存储
- 索引存储
- 散列存储

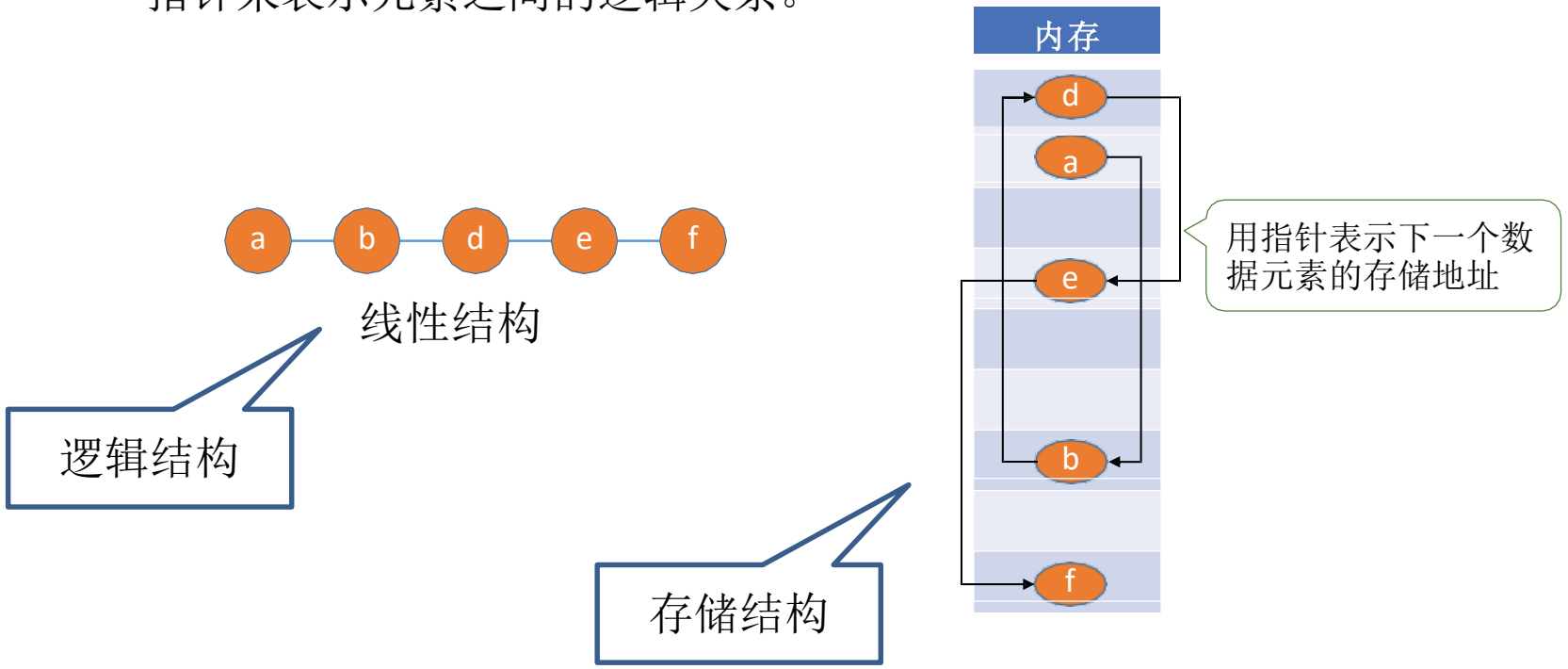
1.2 基本概念和术语

- **顺序存储**
  - 把逻辑上相邻的元素存储在物理位置上也相邻的存储单元中，元素之间的关系由存储单元的邻接关系来体现。



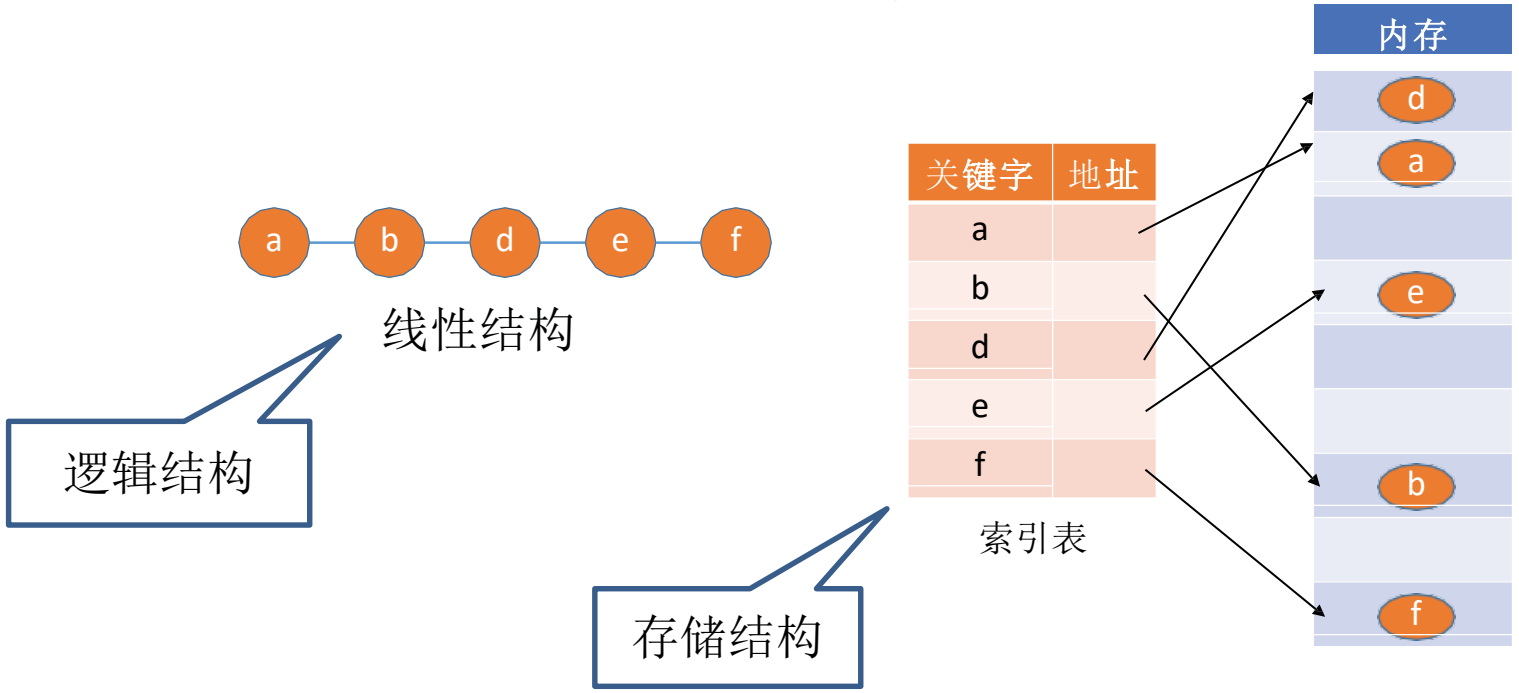
1.2 基本概念和术语

- **链式存储**
  - 逻辑上相邻的元素在物理位置上可以不相邻，借助指示元素存储地址的指针来表示元素之间的逻辑关系。



1.2 基本概念和术语

- **索引存储**
  - 在存储元素信息的同时，还建立附加的索引表。索引表中的每项称为索引项，索引项的一般形式是（关键字，地址）

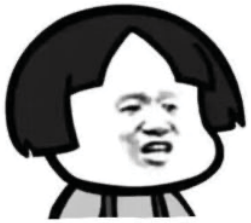


1.2 基本概念和术语

- **散列存储**
  - 根据元素的关键字直接计算出该元素的存储地址，又称哈希（Hash）存储

线性结构

逻辑结构



这真是一门
让人头秃的科目

以后再画吧

存储结构

1.2 基本概念和术语

- **数据的存储结构：**
  - 顺序存储方法
  - 链式存储方法
  - 索引存储方法
  - 散列存储方法

非顺序存储

1.2 基本概念和术语

顺序存储

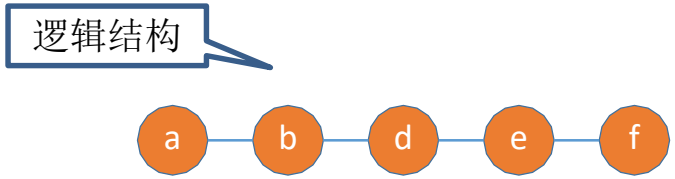
非顺序存储

绪论部分只需要理解两点：

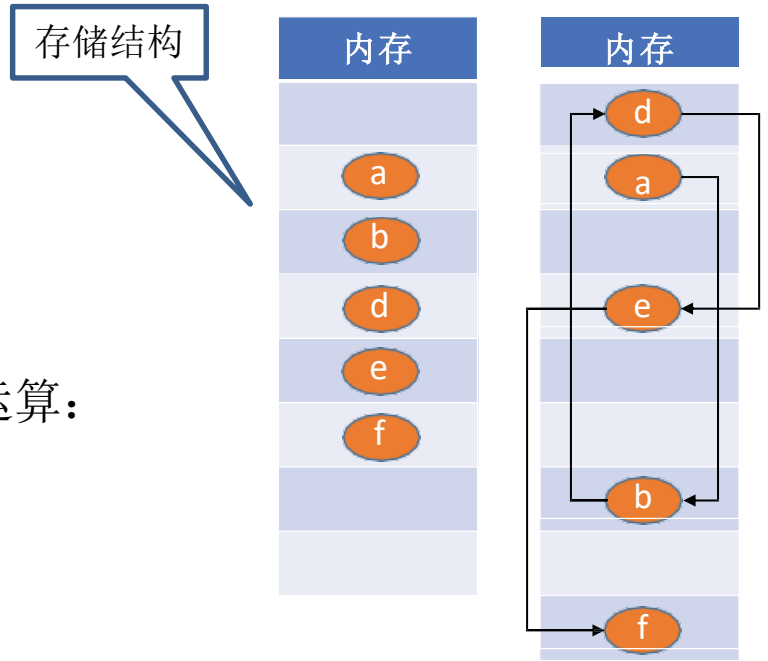
1. 若采用顺序存储，则各个数据元素在物理上必须是连续的；若采用非顺序存储，则各个数据元素在物理上可以是离散的。
2. 数据的存储结构会影响存储空间分配的方便程度
3. 数据的存储结构会影响对数据运算的速度

1.2 基本概念和术语

- **③数据的运算**
  - 施加在数据上的运算包括运算的定义和实现。
  - 运算的定义是针对逻辑结构的，指出运算的功能；
  - 运算的实现是针对存储结构的，指出运算的具体操作步骤



线性结构



运算

结合现实需求定义这种逻辑结构的运算：

1、插入元素；
2、删除元素；
3、输出表长；

1.2 基本概念和术语

- 数据结构三要素：
  - ① 数据的逻辑结构
    - 数据元素之间的逻辑关系，是具体关系的抽象。
  - ② 数据的存储结构(物理结构)
    - 数据元素及其关系在计算机内存中的表示；
  - ③ 数据的运算
    - 施加在数据上的运算包括运算的定义和实现。运算的定义是针对逻辑结构的，指出运算的功能；运算的实现是针对存储结构的，指出运算的具体操作步骤

1.2 基本概念和术语

- **逻辑结构与存储结构的关系：**
  - 数据的逻辑结构属于用户视图，是面向问题的，反映了数据内部的构成方式；数据的存储结构属于具体实现的视图，是面向计算机的。
  - 一种数据的逻辑结构可以用多种存储结构来存储，而采用不同的存储结构，其数据处理的效率往往是不同的。
- **数据结构的学习内容：**
  - 数据对象的结构形式，各种数据结构的性质(逻辑结构)；
  - 数据对象和“关系”在计算机中的表示(物理结构/存储结构)；
  - 数据结构上定义的基本操作(算法)及其实现；
  - 算法的效率（时间和空间）；
  - 数据结构的应用，如数据分类、检索等。

第一章 绪论

1.1 数据结构及其讨论范畴

1.2 基本概念和术语

1.3 抽象数据类型ADT

1.4 算法和算法分析

1.3 抽象数据类型ADT

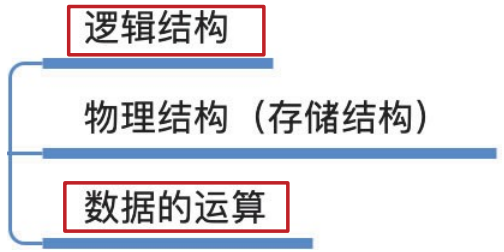
- **数据类型**
- 数据类型是一个值的集合和定义在此集合上的一组操作的总称。
  - 原子类型。其值不可再分的数据类型。
    - **Bool类型：值的范围：true、false；可进行操作：与、或、非…**
    - **Int类型：值的范围：-2147483648 ~ 2147483647；可进行操作：加、减、乘、除、模运算…**
  - 结构类型。其值可以再分解为若干成分（分量）的数据类型。

```
struct Customer{
    int num;          //号数
    int people;       //人数
    ……                //其他必要的信息
};
```

值的范围：num（1~9999）、people（1~12） 可进行操作：如“拼桌”运算，把人数相加合并

1.3 抽象数据类型ADT

- 抽象数据类型（Abstract Data Type）
  - 定义：一个数学模型和在该模型上定义的操作集合的总称。



ADT = (D, R, P),其中：

D 是数据对象; R是 D 上的关系集;P是 D 的基本操作集。

数学模型： 线性表 LIST = (D , R)

$D = \{ a_i \mid a_i \in ElementType , i = 1, 2, \ldots, n, n \geq 0 \}$

$R = \{ <a_{i-1}, a_i> \mid a_{i-1}, a_i \in D , i = 2 , \ldots , n \}$

操作集P：设L的型为LIST线性表实例，e 的型为ElementType的元素实例，i为位置变量。所有操作描述为：

① ListInsert(&L,i, e);
② ListDelete(&L,i,&e);
③ LocateElem(L,e,compare());
④ GetElem(L,i,&e);

1.3 抽象数据类型ADT

- **抽象数据类型（Abstract Data Type）**

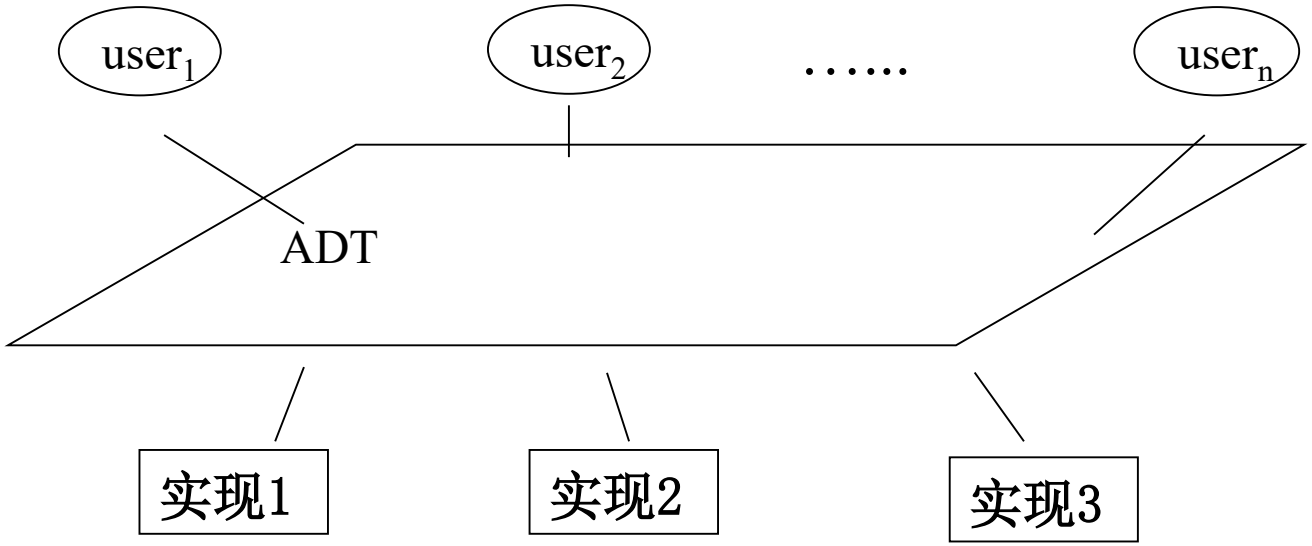
ADT 抽象数据类型名 {
　数据对象：（数据对象的定义或描述）
　数据关系：（数据关系的定义或描述）
　基本操作 *i*：（基本操作的定义或描述）
} ADT 抽象数据类型名；

几点说明：

（1）数据对象+数据关系：刻画数据结构的数学模型；
（2）基本操作：结构上的常用的基本操作(多个)，脱离存储结构；
（3）基本操作往往都是通过函数来实现的，在存储结构确定后，要一一实现所有的基本操作；
（4）基本操作是非标准的（库函数），由用户自己定义。

1.3 抽象数据类型ADT

- 抽象数据类型（Abstract Data Type）
  - 目的
    - 隐藏运算实现的细节和内部数据结构
    - 提高复用的力度和粒度
    - 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ADT

- 抽象数据类型（Abstract Data Type）

ADT特点：

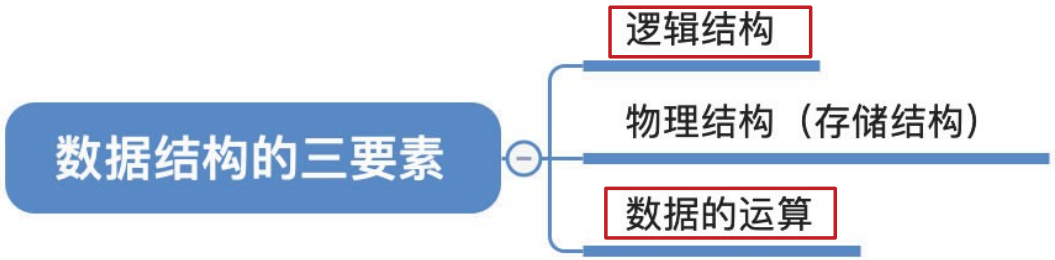
①降低了软件设计的复杂性；

②提高了程序的可读性和可维护性；

③程序的正确性容易保证。

1.3 抽象数据类型ADT

- 抽象数据类型和数据结构的关系



  - 定义一个ADT，就是定义了数据的逻辑结构、数据的运算。 也就是定义了一个数据结构。
  - 确定一种存储结构，就意味着在计算机中表示出数据的逻辑结构。存储结构不同，也会导致运算的具体实现不同。 确定了存储结构，才能实现数据结构。

单选题 1分

设置

以下说法中,正确的是( )。

A 数据元素是数据的最小单位

B 数据项是数据的基本单位

C 数据结构是带有结构的各数据项的集合

D 一些表面上很不相同的数据可以有相同的逻辑结构

提交

单选题 1分

设置

在数据结构中,从逻辑上可以把数据结构分成( )。

A 动态结构和静态结构

B 紧凑结构和非紧凑结构

C 线性结构和非线性结构

D 内部结构和外部结构

提交

填空题 12分

设置

1）若数据结构的形式被定义为(D,R)，其中，D是 [填空1] 的有限集合，R是D上的 [填空2] 的有限集合。
2）在线性结构中，第一个结点 [填空3] 前驱结点，其余每个结点有且只有 [填空4] 个前驱结点；最后一个结点 [填空5] 后续结点，其余每个结点有且只有 [填空6] 个后续结点
3）数据结构包括数据的 [填空7] 、数据的 [填空8] 和数据的 [填空9] 三个方面的内容。
4）线性结构中，元素之间存在 [填空10] 的关系；树形结构中元素之间存在 [填空11] 的关系；图形结构中，元素之间存在 [填空12] 的关系。

作答

第一章 绪论

1.1 数据结构及其讨论范畴

1.2 基本概念和术语

1.3 抽象数据类型ADT

1.4 算法和算法分析

1.4 算法和算法分析

• 算法（algorithm）的概念

算法（Algorithm）：是对特定问题求解步骤的一种描述，它是指令（规则）的有限序列，其中每一条指令表示一个或多个操作。

- 算法是在有限步骤内求解某一问题所使用的一组定义明确的规则；
- 通俗点说，就是计算机解题的过程；
- 程序（program）是算法的一种实现，计算机按照程序逐步执行算法，实现对问题的求解。

算法描述： ①自然语言；②程序设计语言；③类语言*；

1.4 算法和算法分析

例 欧几里德算法—辗转相除法求两个自然数m和n的最大公约数。

– 自然语言描述

- ①输入m和n；
- ②求m除以n的余数r；
- ③若 r等于0，则n为最大公约数，算法结束；否则执行第④步；
- ④将n的值放在m中，将r的值放在n中；
- ⑤重新执行第②步 。

1.4 算法和算法分析

例 欧几里德算法—辗转相除法求两个自然数m和n的最大公约数。

– 伪代码描述

1. r = m % n;
2. 循环直到r等于0
   - 2.1 m = n;
   - 2.2 n = r ;
   - 2.3 r = m % n;
3. 输出n;

– 表达能力强，抽象性强，容易理解

1.4 算法和算法分析

- 算法的五个基本特征：
  - 有穷性：算法中的操作步骤为有限个，且每个步骤都能在有限时间内完成。
  - 确定性：算法中每条指令必须有确切的含义，对于相同的输入只能得出相同的输出。
  - 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0个或多个输入；
  - 输出：有一个或多个输出（处理结果）

1.4 算法和算法分析

例 考虑下列两段描述：

```
(1)
void exam1()
{
      n=2;
     while (n%2==0)
           n=n+2;
     printf("%d\n",n);
}
```

```
(2)
void exam2()
{
     y=0;
     x=5/y;
     printf( "%d,%d\n" ,x,y);
}
```

这两段描述均不能满足算法的特征，试问它们违反了哪些特征？

解：

(1)算法是一个死循环，违反了算法的有穷性特征。

(2)算法包含除零错误，违反了算法的可行性特征。

1.4 算法和算法分析

- “好”算法的特质：
  - 算法必须是“正确的”:算法应能够正确地解决求解问题。 对于一切合法的输入均得到满足要求的结果。
  - 必须具有“健壮性”： 指算法应对非法输入的数据作出恰当反映或进行相应处理，一般情况下，应向调用它的函数返回一个表示错误或错误性质的值。
  - 应有很好的“可读性”:首先方便阅读与交流，其次才是机器执行。
  - 算法的效率：应考虑所设计的算法具有“高效率与低存储量”。

1.4 算法和算法分析

- 算法的执行效率及其度量
  - 解决同一个问题存在多种算法，如何评估各算法的好坏？或据此设计出更好的算法？
    - 运行该算法所需要的计算机资源的多寡，所需越大复杂性越高
    - 最重要的资源：时间（处理器）和空间（存储器）
  - 评价一个算法优劣的重要依据是看执行该算法的程序需要占用多少机器资源：
    - 程序所用算法运行时所要花费的时间代价
    - 程序中使用的数据结构占有的空间代价

  需要明确：
  - 如何表达一个算法的复杂性
  - 怎样计算一个算法的复杂性

1.4 算法和算法分析

- 不能用诸如微秒、纳秒这样的真实时间单位
  - 机器性能会影响算法的执行时间
    - 一个运行在超级计算机上的算法若放在PC机会慢很多
    - 一个运行在超级计算机上的效率极差的算法也许比一个运行在PC上的效率很高的算法花费更少的时间
  - 和编程语言有关
    - 越高级的语言执行效率也就越低
  - 只能事后统计
    - 本末倒置，浪费时间和精力
    - 对于有些算法并不适用
- 评价算法感兴趣的不是具体的资源占用量，而是与具体的平台无关、具体的输入实例无关，且值是可预测的

1.4 算法和算法分析

和算法执行时间相关的因素：

① 算法选用的策略

② 问题的规模

③ 编写程序的语言

④ 编译程序产生的机器代码的质量

⑤ 计算机执行指令的速度

与计算机软件和硬件相关，不考虑

1.4 算法和算法分析

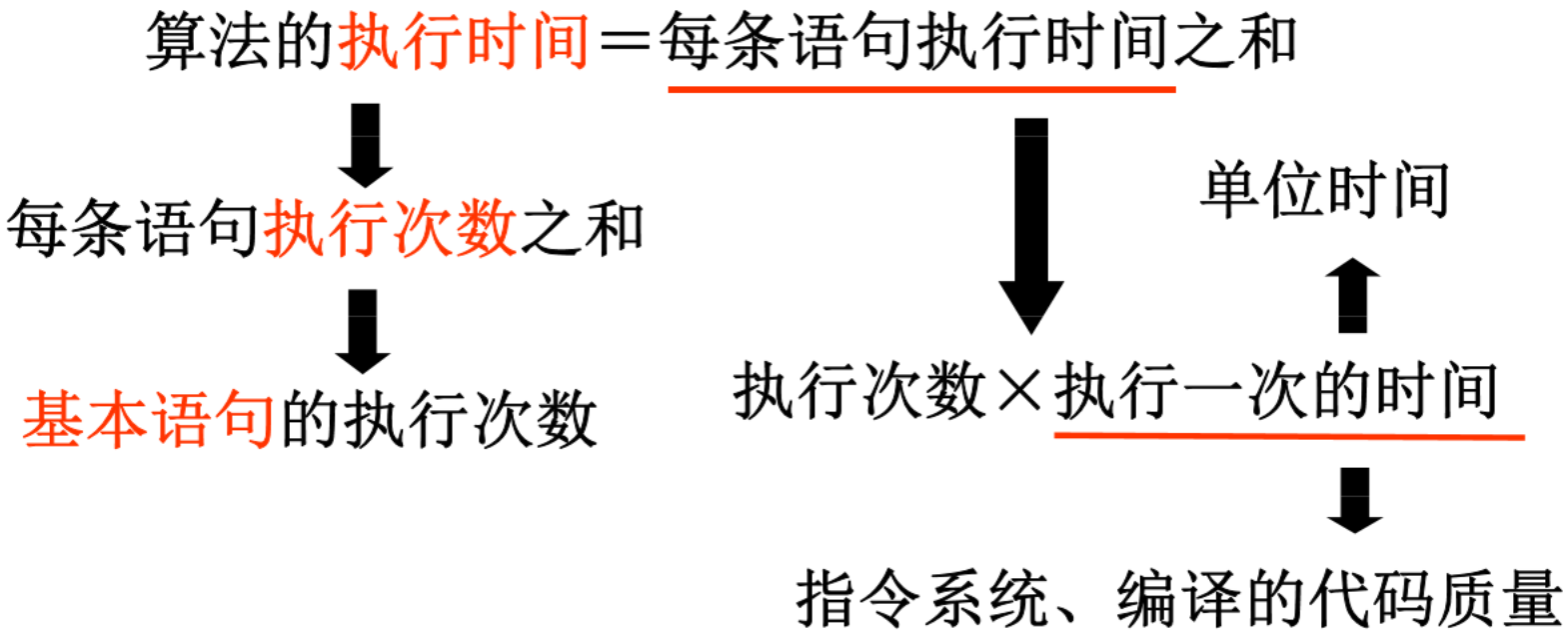
- 算法渐进分析asymptotic analysis
  - 简称算法分析
    - 由于算法的复杂性与其所求解的问题规模直接有关，因此通常将问题规模 n 作为一个参照量，求算法的时间（空间）开销 t 和 n 的关系

1.4 算法和算法分析

- 算法分析（算法渐进分析）：
  - 一般这种函数关系都相当复杂，计算时只考虑可以显著影响函数量级的部分，即结果为原函数的一个近似值
  - 是对算法所需要的计算机资源的一种不精确估计，提供对于算法资源开销进行评估的简单化模型
  - 时间和空间进行估算：
    - 时间复杂度（Time Complexity）*
    - 空间复杂度（Space Complexity）

1.4 算法和算法分析

- 算法分析-时间复杂度（性）：



■ 执行一次的时间对于不同的算法来说都是一样的，所以一般忽略不计

1.4 算法和算法分析

时间复杂度：算法时间开销T(n)与问题规模n 的关系

- 例：对数组a[n]中的各个元素求和的代码：

```
sum = 0;
i = 0;
While (i < n) {
  i++;
  sum += a[i];
}
```

- 在循环开始之前有两次赋值，分别对 i 和对sum进行；
- 条件判断一共执行了n+1次操作；
- 循环进行了n 次，每次循环中执行两次赋值，分别对sum和对 i 进行更新操作；
- 总共执行了3n+3次操作，T(n) = 3n + 3

1.可否忽略式子中的某些项？
2. 当程序复杂时是否需要一行一行的数代码？

1.4 算法和算法分析

时间复杂度：算法时间开销T(n)与问题规模n 的关系

$T(n) = 2n + 2$

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

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

$T(n) = O(n)$

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

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

算法的时间复杂度通常用大O符号表述，定义为T[n] = O(f(n))。称函数T(n)以f(n)为界或者称T(n)受限于f(n)。

大O记号的性质:

(1)对于任一常数$c > 0$，有$O(f(n)) = O(c \cdot f(n))$

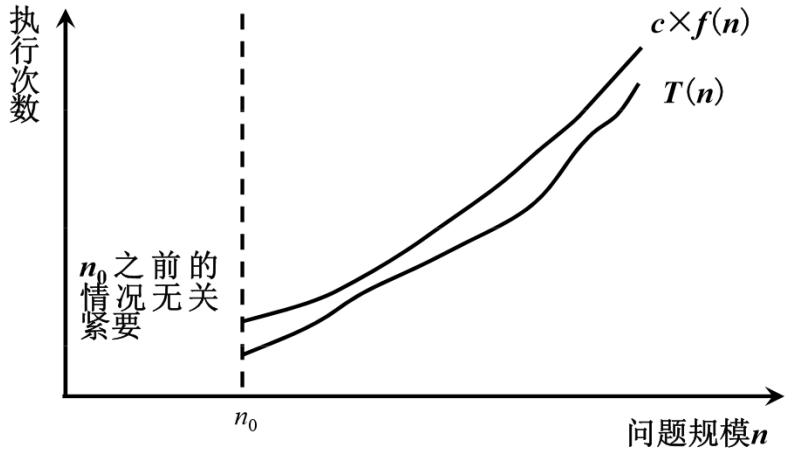
在大O记号的意义下，函数各项正的常系数可以忽略并等同于1。

(2) 对于任意常数$a > b > 0$，有$O(n^a + n^b) = O(n^a)$

多项式中的低次项均可忽略，只需保留最高次项。

1.4 算法和算法分析

大O符号定义 若存在两个正的常数 c 和 n_0，对于任意 $n \geq n_0$，都有 $T(n) \leq c \times f(n)$，则称 $T(n) = O(f(n))$。



- 大O表示法提供了一种表达函数增长率上限的方法，函数T(n) 的增长最终至多趋同于f(n)的增长
- 一个函数增长率的上限可能不止一个。大O表示法给出了所有上限中最“紧”（也即最小）的那个上限
- 大O表示“同阶”，同等数量级。即：当n→∞时，二者之比为常数

1.4 算法和算法分析

大O表示法的运算规则

- 单位时间即O(1)
  - 简单布尔或算术运算
  - 简单I/O
  - 函数返回
- 加法规则：$T(n) = T1(n) + T2(n) = O(f(n)) + O(g(n)) = O(\max(f(n), g(n)))$
  - 顺序结构，if 结构，switch结构
- 乘法规则：$T(n) = T1(n) \times T2(n) = O(f(n)) \times O(g(n)) = O(f(n) \times g(n))$
  - for, while, do-while结构

1.4 算法和算法分析

时间复杂度：算法时间开销T(n)与问题规模n 的关系

- 例：对数组a[n]中的各个元素求和的代码：

```
sum = 0;
i = 0;
While (i < n) {
  i++;
  sum += a[i];
}
```

- 在循环开始之前有两次赋值，分别对 i 和对sum进行；
- 条件判断一共执行了n+1次操作；
- 循环进行了n 次，每次循环中执行两次赋值，分别对sum和对 i 进行更新操作；
- 总共执行了3n+3次赋值操作，T(n) = 3n + 3

1.可否忽略式子中的某些项？
时间复杂度为O(n)
2. 当程序复杂时是否需要一行一行的数代码？

1.4 算法和算法分析

时间复杂度：算法时间开销T(n)与问题规模n 的关系

- 例：对数组a[n]中的各个元素求和的代码：

```
sum = 0;
i = 0; j =0;
While (i < n) {
  i++;
  j++;
  sum += a[i];
}
printf (“%d”, sum);
```

结论：

1、顺序执行的代码只会影响常数项，可以忽略

2、只需挑循环中的一个基本操作分析它的执行次数与n 的关系即可

1.4 算法和算法分析

时间复杂度：算法时间开销T(n)与问题规模n 的关系

- 例：对数组a[n]中的各个元素求和的代码：

```
sum = 0;
i = 0;
While (i < n) {
  i++;
  sum += a[i];
  for (j =0;j<n; j++) {
    printf (“%d”, sum);
  }
}
printf (“%d”, sum);
```

```
j=1;
While(j < n)
{
   printf (“%d”, sum);
   j++ ;
}
```

结论：

1、顺序执行的代码只会影响常数项，可以忽略

2、只需挑循环中的一个基本操作分析它的执行次数与n 的关系即可

3、如果有多层嵌套循环，只需关注最深层循环循环了几次

【例】分析下面各程序段的时间复杂度

```
（1）for (i=0;i<n;i++)
        for (j=0;j<m;j++)
            A[i][j]=0;
```

T(n) = O(n*m)

```
（2） T=A;
      A=B;
      B=T;
```

T(n) = O(1)

```
（3）s=0;
     for (i=0;i<n;i++)
          for (j=0;j<n;j++)
               s+=B[i][j];
     sum=s;
```

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

```
（4）s1(int n)
    { int p=1,s=0;
      for (i=1;i<=n;i++)
          { p*=i; s+=p; }
      return(s);
    }
```

T(n) = O(n)

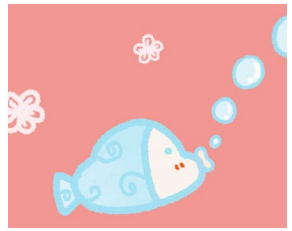
```
（5）for ( i=1; i<=n ; ++i )
        for ( j=1 ; j <=n ; ++j )
        { c[i][j] = 0;
          for ( k=1 ; k <= n;  ++k )
                c[i][j] += a[i][k] * b[k][j] ;
        }
```

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

```
（6）s2(int n)
    { x=0;                          O(1)
      y=0;                          O(1)
      for(k=1;k<=n;k++)             O(n+1)
        x++;                        O(n)
      for(i=1;i<=n;i++)             O(n+1)
        for(j=1;j<=n;j++)           O(n*(n+1))
          y++;                      O(n²)
    }
```

$\sum(\ldots)=O(n^2)$

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



【例1-4】求冒泡排序算法的时间复杂度

```
Void  BUBBLE(A)
int A[n];
{   int I,j,temp;
    for(i=0;i<n-1;i++)
        for(j=n-1;j>=i+1;j--)
            if(A[j-1]>A[j]) {
                temp=A[j-1];        O(1)
                A[j-1]=A[j];        O(1)
                A[j]=temp;          O(1)
            }
}
```

O(1)

O(1)

O((n-i-1) ×1)

=(n-i-1)

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

=O(n(n-1)/2)

$=O(n^2)$

【例1-5】

(1) for(p=1,i=2; i<=n; p=p*i++) ;

展开：

```
p = 1;
i  = 2;
While (i <= n)
{   p=p*i ;
    i=i+1;
}
```

T(*n*)=O(*n*).

【例1-5】

(2) Long fact (int n)

```
{   if ( n==0 ) || ( n ==1 )
        return( 1 );
    else
        return( n * fact( n – 1 ) );
}
```

$$f(n)=\begin{cases} C & \text{当 } n=0,\ n=1 \\ G+f(n-1) & \text{当 } n>1 \end{cases}$$

$f(n) = G_1 + f(n-1)$

$f(n-1) = G_2 + f(n-2)$

$f(n-2) = G_3 + f(n-3)$

…… ……

$f(2) = G_{n-1} + f(1)$

\+ $f(1) = C$

$f(n) = G_1 + G_2 + G_3 + \cdots\cdots + G_{n-1} + C$

$f(n) = n\,G'$

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

【例1-6】分析下面各程序段的时间复杂度

```
for ( i=1; i<=n; i=2*i )
      pringtf( "%d", i );
```

or

```
i=1;
While(i<=n)
{    i=2*i ;
     pringtf( "%d", i );
}
```

T(n) = ?

i = 1, 2, 4, 8, 16 ...

假设循环次数为x, 那么$i = 2^{x-1}$

由条件： i <= n,　　可以得到：$2^{x-1} <= n$

所以有： $x <= \log_2 n+1$

也就是循环次数x 从1 到$\log_2 n+1$，一共执行循环体$\log_2 n+1$次。

故有：$T(n) = o(\log_2 n)$

```
for ( i=n; i>=1; i=i/2 )
      pringtf( "%d", i );
```

i=i/3

or

```
i=n;
While(i>=1)
{    i=i/2 ;
     pringtf( "%d", i );
}
```

$T(n) = O(\log_2 n)$ or $O(\log_3 n)$

◆若一个算法中的语句频度之和为 $f(n) = 6n+3n\log_2 n$，则算法的时间复杂度T(n)=?

◆若一个算法中的语句频度之和为 $f(n) = 3n+n\log_2 n+n^2$，则算法的时间复杂度T(n)=?

1.4 算法和算法分析

- 常见阶的比较

$O(1) < O(\log_2 n) < O(n) < O(n\log_2 n) < O(n^2) < O(n^3) < O(2^n) < O(n!) < O(n^n)$

问题：两个算法的时间复杂度分别如下，哪个的阶数更高（时间复杂度更高）？

$T_1(n) = O(n)$

$T_2(n) = O(\log_2 n)$

$$\lim_{n\to\infty} \frac{\log_2 n}{n} \xrightarrow{\text{洛必达}} \lim_{n\to\infty} \frac{1}{n\ln 2} = 0$$

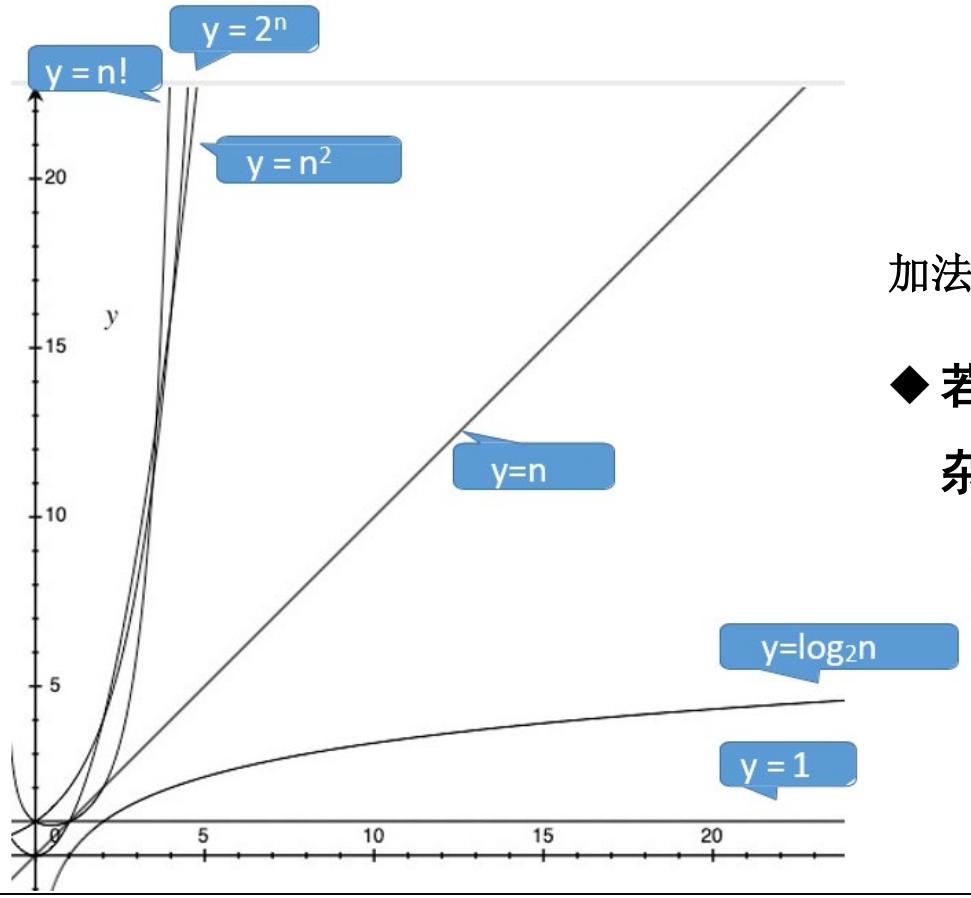
当n→∞时，n比$\log_2 n$变大的速度快很多

$$\lim_{n\to\infty} \frac{n^2}{2^n} \xrightarrow{\text{洛必达}} \lim_{n\to\infty} \frac{2n}{2^n \ln 2} \xrightarrow{\text{洛必达}} \lim_{n\to\infty} \frac{2}{2^n \ln^2 2} = 0$$

当n→∞时，2^n比n^2变大的速度快很多

1.4 算法和算法分析

- 常见阶的比较



加法规则： $f_1(n)+f_2(n)=O(\max(f_1(n), f_2(n)))$

◆ 若$f(n) = 3n+n\log_2 n+n^2$，则算法的时间复杂度$T(n)$=?

【例1-6】分析下面各程序段的时间复杂度

（1）
```
x=n;                          //n>1
y=0;
while(x >= (y+1)* (y+1))
    y++;
```

单选

A. $T(n)= O(n)$

B. $T(n)= O(\sqrt{n})$

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

D. $T(n)= O(\log n)$

while循环的退出条件为：$x<(y+1)^2$，或 $(y+1)^2>n$；

y 的初值为0，第 k 次循环后，$y=k$；

则循环的退出条件变为：$(k+1)^2>n$，即：$k>\sqrt{n}-1$

∵ k 为正整数，∴ $k=\lfloor n^{0.5} \rfloor$

所以有：$T(n)=O(\sqrt{n})$，或 $O(n^{0.5})$

【例1-6】分析下面各程序段的时间复杂度

（2）
```
x=90;
y=100;
while(y>0)
if(x>100)
  {  x=x-10;
     y--;
  }
 else
     x++;
```

单选

A. T(*n*)= O(n)

B. T(*n*)= O(1)

C. T(*n*)= O(n^2)

D. T(*n*)= O(lg*n*)

【例1-7】二分查找, x[n] 为递增数组, 求数组中值等于 p 的下标。

```
left = 0;
right = n - 1;
while (left <= right)
{
   mid = (left + right)/2;
   case:
    x[mid] < p : left = mid + 1;
    x[mid] = p : p = mid;  break;
    x[mid] > p : right = mid - 1;
}
```

共同n个元素，二分执行，剩余元素依次为$n/2^1$，$n/2^2$，$n/2^3$，…，$n/2^k$
其中 k 为程序执行次数。
令： $n/2^k = 1$， 则 $n = 2^k$， $k = \log_2 n$, 复杂度T(n)= **O($\log_2 n$)**

1.4 算法和算法分析

- 算法分析---好情况、坏情况、平均情况：
  - 例：在一维整型数组A[n]中顺序查找与给定值k相等的元素（假设该数组中有且仅有一个元素值为k）。

```
int Find(int A[ ], int n)
{
    for (i=0; i<n; i++)
        if (A[i]= =k) break;
    return i;
}
```

  - 基本语句的执行次数是否只和问题规模有关？
  - 结论：如果问题规模相同，时间代价与输入数据（的分布）有关，则需要分析最好情况、最坏情况、平均情况。

1.4 算法和算法分析

- 算法分析---好情况、坏情况、平均情况：
- 对于时间开销，一般不注意算法的“最好估计”。特别是处理应急事件，计算机系统必须在规定的响应时间内做完紧急事件处理。这时，最坏估计是唯一的选择
- 对于多数算法而言，最坏情况和平均情况估计两者，它们的时间开销的公式虽然不同，但是往往只是常数因子大小的区别，或者常数项的大小区别。因此不会影响渐进分析的增长率函数估计

1.4 算法和算法分析

算法分析---时间复杂度分析的基本方法：

- 算法的时间复杂度通常用大O符号表述，$T(n) = O(f(n))$
  - 大0表示法提供了一种表达函数增长率上限的方法，函数T(n) 的增长最终至多趋同于f(n)的增长
- 时间复杂性的运算法则
  - 加法规则：$T(n) = T1(n) + T2(n) = O(f(n)) + O(g(n)) = O(\max(f(n), g(n)))$
  - 乘法规则：$T(n) = T1(n) \times T2(n) = O(f(n)) \times O(g(n)) = O(f(n) \times g(n))$
- 时间复杂性的分析方法
  - 首先求出程序中各语句、各模块的运行时间，
  - 再求整个程序的运行时间。

1.4 算法和算法分析

- 算法分析---各种语句和模块分析应遵循的规则：

① 基本操作（赋值语句或读/写语句等）：

- 运行时间通常取O(1)。有函数调用的除外，此时要考虑函数的执行时间。

② 语句序列:

- 运行时间由加法规则确定，即该序列中耗时最多的语句（模块）的运行时间。

③ 分支语句:

- 运行时间由条件测试（通常为O(1)）加上分支中运行时间最长的语句的运行时间

1.4 算法和算法分析

- 算法分析---各种语句和模块分析应遵循的规则：

④ 循环语句：

– 运行时间是对输入数据重复执行n次循环体所耗时间的总和 。

– 每次重复所耗时间包括两部分：一是循环体本身的运行时间；二是计算循环参数、测试循环终止条件和跳回循环头所耗时间。

– 通常，将常数因子忽略不计，可以认为上述时间是循环重复次数n和m的乘积，其中m是n次执行循环体当中 时间消耗多的那一次的运行时间(乘法规则) 。

– 当遇到多重循环时，要由内层循环向外层逐层分析。 因此，当分析外层循环的运行时间时，内层循环的运行时间应该是已知的。此时可以把内层循环看成是外层循环的循环体的一部分。

1.4 算法和算法分析

- 算法分析---各种语句和模块分析应遵循的规则：

⑤ 函数调用语句:

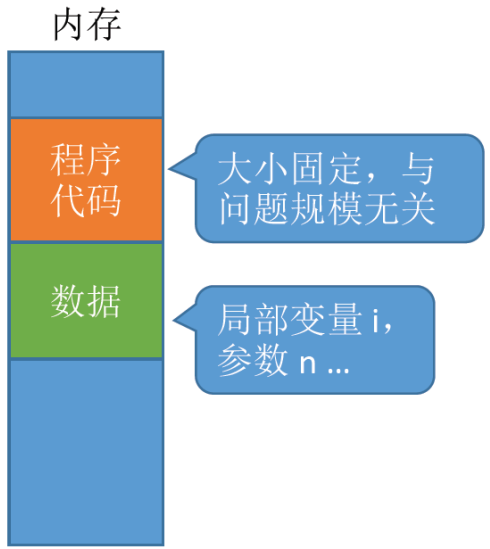
a) 若程序中只有非递归调用，则从没有函数调用的被调函数开始，计算所有这种函数的运行时间。然后考虑有函数调用的任意一个函数P，在P调用的全部函数的运行时间都计算完之后，即可开始计算P的运行时间。

b) 若程序中有递归调用，则令每个递归函数对应于一个未知的时间开销函数T(n)，其中n是该函数参数的大小，之后列出关于T的递归方程并求解之。

1.4 算法和算法分析

- 算法分析---空间复杂性
  - 算法的空间复杂性是指算法在执行过程中的存储量需求

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



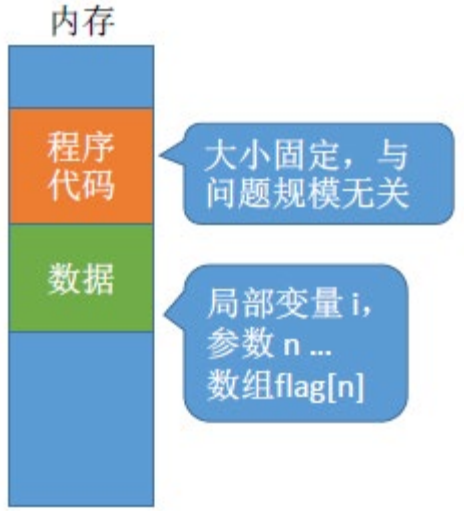
无论问题规模怎么变，算法运行所需的内存空间都是固定的常量，算法空间复杂度为

$S(n) = O(1)$

1.4 算法和算法分析

- 算法分析---空间复杂性
  - 算法的空间复杂性是指算法在执行过程中的存储量需求

```
Void sum (int n)
 {
   int flag[n];
   int i;
   …..
}
```



  - S(n) = 4+4n+4=4n+8
  - 关注存储空间大小和问题规模相关的变量

S(n) = O(n)

1.4 算法和算法分析

- 算法分析---空间复杂性
  - 算法的空间复杂性是指算法在执行过程中的存储量需求

```
Void sum (int n)
 {
    int flag[n][n];
    int other[n];
    int i;
    …..
}
```

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

1.4 算法和算法分析

- 算法分析---空间复杂性
  - 一个算法的存储量需求除了存放算法本身所有的指令、 常数变量和输入数据外，还包括对数据进行操作的工作单元和存储实现计算所需信息的辅助空间
  - 算法的存储量需求与输入的规模、表示方式、算法采用的数据结构、算法的设计以及输入数据的性质有关

1.4 算法和算法分析

- 算法分析---空间复杂性
  - 很多算法使用的数据结构是静态的存储结构，即存储空间在算法执行过程中并不发生变化
  - 使用动态数据结构算法的存储空间是变化的，在算法运行过程中有时会有数量级的增大或缩小。

1.4 算法和算法分析

- 时空资源的折中原理
  - 同一个问题求解，一般会存在多种算法，这些算法在时空开销上的优劣往往表现出“时空折中”（trade-off）的性质
  - 即，为了改善一个算法的时间开销，往往以增大空间开销为代价，而设计出一个新算法来
  - 有时，为了缩小算法的空间开销，也可以牺牲计算机的运行时间，通过增大时间开销来换取存储空间的节省

1.4 算法和算法分析

- **数据结构的选择和评价**
- 仔细分析所要解决的问题，特别是求解问题所涉及的数据类型和数据间逻辑关系
- 数据结构的初步设计往往在算法设计之先
  - 给定3个整型数，设计算法将3个整型数按升序排列
  - 给定100个整型数，设计算法将这100个整型数按升序排列
    - 1）int A[100]：确定数据结构，组织数据,……
    - 2）排序算法：气泡排序、插入排序、快排、SHELL排序、堆排序、……
- 注意数据结构的可扩展性。包括考虑当输入数据的规模发生改变时，数据结构是否能够适应 。同时，数据结构应该适应求解问题的演变和扩展
- 数据结构的设计和选择也要比较算法的时空开销的优劣

1.4 算法和算法分析

《数据结构》或《数据结构与算法》

- 设计适合算法实现的数据结构；
- 是对数据结构适用性的验证；
- 侧重算法的实现；

《算法设计与分析》

- 用数学方法研究算法；
- 侧重算法的正确性证明和算法分析；

提高算法设计能力

本章小结

- 数据结构的定义
- 数据结构的主要研究内容
- 抽象数据类型的概念
- 算法的特性
- 算法的复杂度分析