第一章 绪论

- 数据结构的概念
- 抽象数据类型概念
- 算法定义
- 算法性能分析与度量

数据结构的概念

- 什么是数据结构
- 基本概念和术语
- 数据结构涵盖的内容

数据结构的概念 --什么是数据结构

- 数据结构是一门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**等等的学科。
  - 地位：
    计算机科学中的一门综合性的专业基础课；
    介于数学、计算机硬件和计算机软件之间的核心课程。
  - 学习方法：
    以学习思想为主，以高级程序设计语言的表达为途径。
    先建立数据结构模型→再上机实践→灵活解决实际问题。

数据结构的概念--基本概念和术语

- **数据(data)**：所有能被计算机识别、存储和处理的符号的集合(包括数字、字符、声音、图像等信息)。
- **数据元素(data element)**：是数据的基本单位，具有完整确定的实际意义(又称元素、结点，顶点、记录等）。
- **数据项(data item)**：构成数据元素的项目。是具有独立含义的最小标识单位（又称字段、域、属性等）。

三者之间的关系：数据>数据元素>数据项

例：班级通讯录 > 个人记录 > 姓名、年龄……

数据结构的概念--基本概念和术语

- **数据结构**(data structure)：由某一数据元素的集合以及该集合中所有数据元素之间的关系组成。记为：

$$Data\_Structure = \{D, R\}$$

其中，D 是某一数据元素的集合，R 是该集合中所有数据元素之间的关系的有限集合。

数据结构的概念--基本概念和术语

- 数据结构是数据的组织形式，包括三个方面：
  - 数据元素间的逻辑关系，即数据的**逻辑结构**；
  - 数据元素及其关系在计算机存储内的表示，即**数据的存储表示**；
  - **数据的运算**，即对数据元素施加的操作。

数据结构的概念--基本概念和术语

- 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；

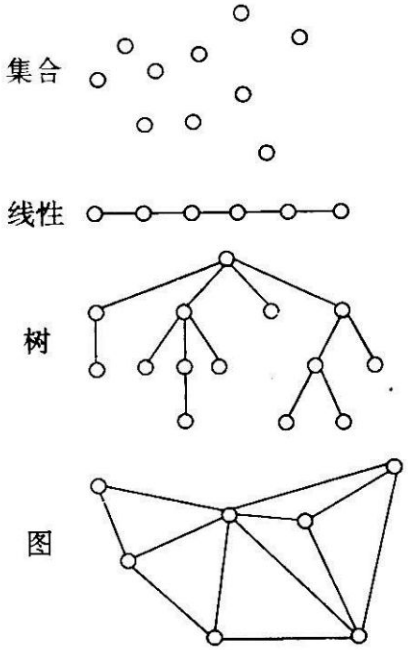
数据的逻辑结构与数据元素本身的形式、内容无关；

数据的逻辑结构与数据元素的相对存储位置无关。

数据结构的概念--基本概念和术语

- 数据的逻辑结构分类：
  - **集合**：结构中的数据元素除了同属于一种类型外，别无其它关系。
  - **线性结构**：结构中的数据元素之间存在一对一的关系。
  - **树型结构**：结构中的数据元素之间存在一对多的关系。
  - **图状结构或网状结构**：结构中的数据元素之间存在多对多的关系。

数据结构的概念--基本概念和术语



数据结构的概念--基本概念和术语

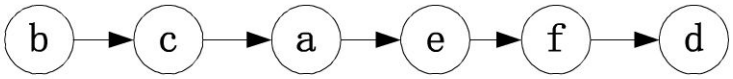
例：用图形表示下列数据结构，并指出它们是属于线性结构还是非线性结构。

1） S=(D, R)

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

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

解：上述表达式可用图形表示为：



此结构为线性的。

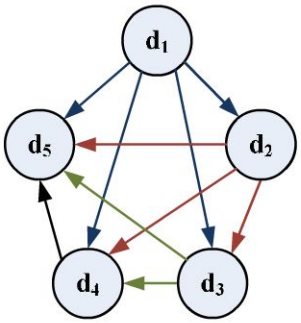
数据结构的概念--基本概念和术语

2） S=(D, R)

$D=\{d_i \mid 1 \leqslant i \leqslant 5\}$

$R=\{(d_i, d_j), i<j\}$

解：上述表达式可用图形表示为：



该结构是非线性的。

数据结构的概念--基本概念和术语

- 数据的存储结构

数据的存储结构（物理结构）是数据的逻辑结构在计算机存储器中的实现。

- 数据的存储结构分类：
  - 主要用于内存的存储表示
    - 顺序存储表示
    - 链接存储表示
  - 主要用于外存的存储表示
    - 索引存储表示
    - 散列存储表示

数据结构的概念--基本概念和术语

顺序存储是把逻辑上相邻的结点存储在物理上相邻的存储单元里，结点之间的关系由存储单元的邻接关系来体现。优点：占用最小的存储空间。缺点：在做插入、删除等操作时需要移动大量的数据。

链式存储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。优点：易于插入、删除等操作。缺点：占用较多的存储空间（指针域）。

数据结构的概念--基本概念和术语

索引存储用结点的索引号来确定结点的存储地址。优点：索引检索速度快。缺点：增加额外的索引表，占用较多的存储空间，在插入、删除时还要修改索引表，花费较多时间。

散列结构是根据结点的关键码通过一个函数计算直接得到该结点的存储地址。

综上，数据的逻辑结构根据问题所要实现的功能建立，数据的物理结构根据问题所要求的响应速度、处理时间、修改时间、存储空间和单位时间的处理量等建立，是逻辑结构的存储映像。

数据结构的概念--基本概念和术语

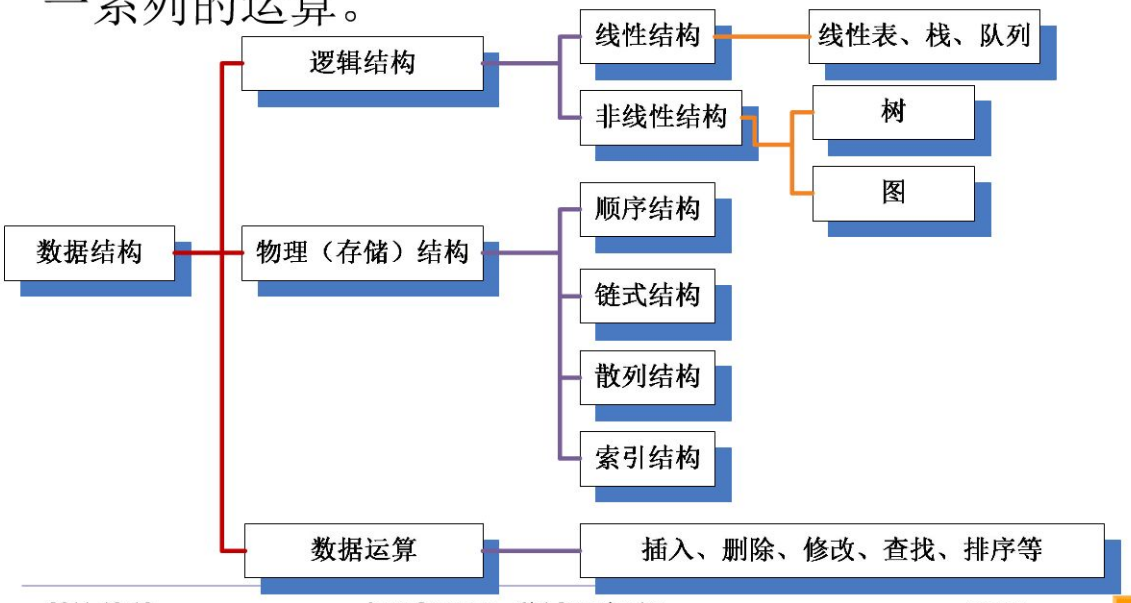
- 数据的运算

在数据的逻辑结构上定义的操作算法，在数据的存储结构上实现。

最常用的数据运算有5种：插入、删除、修改、查找、排序。

数据结构的概念
--数据结构涵盖的主要内容

- 给了一个逻辑关系，选了一个存储结构，进行了一系列的运算。



抽象数据类型概念

- 数据类型与抽象数据类型
- 抽象数据类型的定义
- 抽象数据类型的表示与实现

抽象数据类型概念
--数据类型与抽象数据类型

- **数据类型**：是一个**值的集合**和定义在该值集上的**一组操作**的总称。
- **抽象数据类型**(ADTs: Abstract Data Types)：由用户定义，用以表示应用问题的数据模型。它由基本的数据类型构成，并包括一组相关的服务（或称操作）

  特点是：

  信息隐蔽和数据封装，使用与实现相分离。

抽象数据类型概念
--抽象数据类型的定义

- 抽象数据类型的定义

抽象数据类型可以用以下的三元组来表示：

ADT = （D，S，P）

其中，D表示数据对象，S是D上的关系集，P是D上的操作集。

- 抽象数据类型的描述

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

抽象数据类型概念
--抽象数据类型的定义

其中数据对象、数据之间的关系用伪码描述；

基本操作定义格式为：

基本操作名（参数表）

前置条件：〈先决条件描述〉

后置条件：〈操作结果描述〉

“前置条件”描述了操作执行之前数据结构和参数应满足的先决条件，若不满足，则操作失败，并返回相应出错信息。

“后置条件”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。若前置条件为空，则省略之。

抽象数据类型概念
--抽象数据类型的定义

例：自然数的抽象数据类型定义

ADT NaturalNumber is

objects: 一个整数的有序子集合,它开始于0,结束于机器能表示的最大整数(MaxInt)。

Function: 对于所有的 x, y ∈ NaturalNumber;

False, True ∈ Boolean, +、-、<、==、= 等都是可用的服务。

Zero( ) : NaturalNumber

//前置条件：无

//后置条件：返回自然数0

抽象数据类型概念
--抽象数据类型的定义

IsZero(x) : Boolean
  //前置条件：x为NaturalNumber
  //后置条件：if (x == 0) then 返回True else 返回False
  Add (x, y) : NaturalNumber
  //前置条件：x, y为NaturalNumber且x+y≤MaxInt
  //后置条件：返回 x+y
  Subtract (x, y) : NaturalNumber
  //前置条件： x, y为NaturalNumber且x≥y
  //后置条件：返回 x- y

抽象数据类型概念
--抽象数据类型的定义

```
  Equal (x, y) : Boolean
    //前置条件:  x, y为NaturalNumber
    //后置条件:  if (x == y) 返回True  else
  返回 False
    Successor (x) : NaturalNumber
    //前置条件:  x为NaturalNumber
    //后置条件:  if (x == MaxInt) 返回 x
  else  返回 x+1
end NaturalNumber
```


抽象数据类型概念
--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。

从C语言的角度理解，它有些类似C语言中的结构（struct)类型，但增加了相关的服务。从C++的角度理解，通过模板类来实现它。

在上机时要用具体语言实现，如C或C++等。

模板（template)：适合多种数据类型的类定义或算法，在特定环境下通过简单地代换，变成针对具体某种数据类型的类定义或算法。

算法定义

- 算法定义
- 算法特性

算法定义--定义与特性

- 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。
- 算法的特性：
  - 有输入 ：有0个或多个输入。
  - 有输出 ：有一个或多个输出(处理结果)。
  - 确定性：算法中每一条指令必须有确切的含义。不存在二义性。
  - 有穷性：一个算法必须总是在执行有穷步之后结束，且每一步都在有穷时间内完成。
  - 可行性：一个算法是可行的。即算法描述的操作都是可以通过已经实现的基本运算执行有限次来实现的。

算法性能分析与度量

- 评价算法的性能标准
- 算法效率的度量

算法性能分析与度量
--评价算法的性能标准

- 评价一个好的算法有以下几个标准:
  - 正确性(Correctness)：算法应满足具体问题的需求。
  - 可读性(Readability)：算法应该好读。以有利于阅读者对程序的理解。
  - 健壮性(Robustness)：算法应具有容错处理。当输入非法数据时，算法应对其做出反应，而不是产生莫名其妙的输出结果。
  - 效率（Efficiency）：效率指的是算法执行的时间和空间利用率。一般，这两者与问题的规模有关。

算法性能分析与度量
--算法效率的度量

- 算法分析的目的在于分析算法的效率以求改进，算法分析的两个主要方面是时间复杂度和空间复杂度。
- 对一个算法要作出全面的分析可分成两个阶段进行，即事前分析和事后测试。
  - 事前分析要求事前求出该算法的一个时间界限函数。
  - 事后测试则要求在算法执行后通过算法执行的时间和实际占用空间的统计资料来分析。

算法性能分析与度量
--算法效率的度量

- 算法的事后统计

  事后分析要求在算法中的某些部位插装时间函数time ( )，测定算法完成某一功能所花费时间。

插装 time( ) 的计时程序

```
double start, stop;
 time(&start);
 int k = seqsearch (a, n, x);
 time(&stop);
 double runTime = stop - start;
 cout << " " << n << " " << runTime <<
endl;
```


算法性能分析与度量
--算法效率的度量

事实上，算法运行时间要受输入规模、利用编译程序生成的目标代码的质量、计算机程序指令系统的品质和速度等制约。

算法性能分析与度量
--算法效率的度量

- 算法的事前估计

  算法的事前估计主要包括时间复杂性和空间复杂性的分析：

  - 问题的规模：如：矩阵的阶数、图的结点个数、被分类序列的正整数个数等。
  - 时间复杂性：算法所需时间和问题规模的函数，记为 T(n)。当 $n\to\infty$时的时间复杂性，称为渐进时间复杂性。
  - 空间复杂性：算法所需空间和问题规模的函数。记为 S(n)。当 $n\to\infty$时的空间复杂性，称为渐进空间复杂性。

算法性能分析与度量
--算法效率的度量

- 空间复杂度度量
  - 存储空间的固定部分

    程序指令代码的空间，常数、简单变量、定长成分(如数组元素、结构成分、对象的数据成员等)变量所占空间
  - 可变部分

    尺寸与实例特性有关的成分变量所占空间、引用变量所占空间、递归栈所用空间、通过new和delete命令动态使用空间

算法性能分析与度量
--算法效率的度量

- 时间复杂度度量
  - 程序步的概念

    程序步：语法上或语义上有意义的一段指令序列，而且这段指令序列的执行时间与问题规模无关。

    例如：声明语句：程序步数为0;

    表达式：程序步数为1。
  - 程序步确定方法
    - a）插入计数全局变量count。
    - b）建表，列出各语句的程序步。

算法性能分析与度量
--算法效率的度量

- 通过插入计数全局变量count，计算程序步

例：以迭代方式求累加和的函数

```
float sum (float a[ ], int n) {
    float s = 0.0;
    for (int i = 0; i < n; i++)
        s = s + a[i];
    return s;
}
```


算法性能分析与度量
--算法效率的度量

在求累加和程序中加入 count 语句

```
float sum (float a[ ], int n) {
      float s = 0.0;
      count++;      //count 统计执行语句条数
      for (int i = 0; i < n; i++) {
            count += 2;   //针对 for 语句
         s += a[i];
         count++; }//针对赋值语句
      count += 2;      //针对 for 的最后一次
      count++;     //针对 return 语句
      return s; }
```

执行结束得程序步数 count = 3*n+4

算法性能分析与度量
--算法效率的度量

为统计程序步，程序的简化形式：

```
void sum (float a[ ], int n) {
      for (int i = 0; i < n; i++)
            count += 3;
      count += 4;
}
```


算法性能分析与度量
--算法效率的度量

注意：

一个语句本身的程序步数可能不等于该语句一次执行所具有的程序步数。

例如：

- 赋值语句x = sum (R, n) 本身程序步数为 1；
- 一次执行对函数 sum (R, n) 的调用需要的程序步数为 3*n+4；
- 一次执行的程序步数为

  1+3*n+4 = 3*n+5

算法性能分析与度量
--算法效率的度量

- 通过建表，列出各语句的程序步

程序语句	一次执行所需程序步数	执行频度	程序步数
{	0	1	0
float s = 0.0;	1	1	1
for (**int** i = 0; i < n; i++)	2	n+1	2(n+1)
s = s + a[i];	1	n	n
return s;	1	1	1
}	0	1	0
	总程序步数		3n+4

算法性能分析与度量
--算法效率的度量

- 算法的渐进分析

  算法的渐进分析直接与它所求解的问题的规模n相关，通常将问题规模作为分析的参数，求算法的时间和空间开销与问题规模n的关系。

  - 时间复杂度的渐进表示法

  算法中所有语句的频度之和是矩阵阶数n的函数

  $$T(n) = 3n^3 + 5n^2 + 4n + 2$$

  一般地，称 n 是问题的规模。则时间复杂度 T(n) 是问题规模 n 的函数。

算法性能分析与度量
--算法效率的度量

当n趋于无穷大时，把时间复杂度的数量级（阶）称为算法的渐进时间复杂度

$T(n) = O(n^3)$ — 大O表示法

- **加法规则** 针对并列程序段

$$T(n, m) = T_1(n) + T_2(m)$$
$$= O(\max(f(n), g(m)))$$

各种函数的增长趋势

$$c < \log_2 n < n < n\log_2 n < n^2 < n^3 < 2^n < 3^n < n!$$

算法性能分析与度量
--算法效率的度量

例：有如下程序段

```
x = 0;  y = 0;                          ----T1 (n) = O(1);
for ( int k = 0; k < n; k ++ )          ----T2(n) = O(n);
    x ++;
for ( int i = 0; i < n; i++ )           ----T3(n) = O(n2)
   for ( int j = 0; j < n; j++ )
      y ++;
```

$T(n) = T_1(n)+T_2(n)+T_3(n) = O(max(1, n, n^2))$

$= O(n^2)$

算法性能分析与度量
--算法效率的度量

- **乘法规则**　针对嵌套程序段

$$T(n, m) = T_1(n) * T_2(m)$$
$$= O(f(n)*g(m))$$

任何非0正常数都属于同一数量级，记O(1)。

算法性能分析与度量
--算法效率的度量

例：两个并列循环的例子

```
void exam (float x[ ][ ], int m, int n) {
    float sum [ ];
    for (int i = 0; i < m; i++) {      //x中各行
       sum[i] = 0.0;                   //数据累加
       for (int j = 0; j < n; j++)
       sum[i] += x[i][j]; }
    for (i = 0; i < m; i++)        //打印各行数据和
      cout << i << " : " <<sum [i] << endl;
 }
```

渐进时间复杂度为 O(*max* (*m***n*, *m*))

算法性能分析与度量
--算法效率的度量

例：起泡排序

```
void  bubbleSort (int a[ ],  int n ) {
 //对表 a[ ] 逐趟比较, n 是表当前长度
      for (int i = 1; i <= n-1; i++) {     //n-1趟
        for (int j = n-1; j >= i; j--)   //n-i次比较
            if (a[j-1] > a[j]) {
               int tmp = a[j-1];  a[j-1] = a[j];
               a[j] = tmp; } //一趟比较
    }
}
```


算法性能分析与度量
--算法效率的度量

渐进时间复杂度

$$O(f(n)*g(n)) = O(n^2)$$

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

bubbleSort

外层循环 n-1 趟

内层循环 n-i 次比较

算法性能分析与度量
--算法效率的度量

- 空间复杂度的渐进表示法

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

计算解决问题所需的辅助存储空间。通常，只有完成同一功能的几个算法之间才具有可比性。

算法性能分析与度量
--算法效率的度量

例：设有两个算法在同一机器上运行，其执行时间分别为 $100n^2$ 和 2^n，问：要使前者快于后者，n 至少要取多大？

解答：

问题是找出满足 $100n^2 < 2^n$ 的最小的n。用试探法：

$n = 13$时，$100n^2 = 16900 > 2^n = 8192$

$n = 14$时，$100n^2 = 19600 > 2^n = 16384$

$n = 15$时，$100n^2 = 22500 < 2^n = 32764$

取 $n = 15$ 满足要求。