**洒水a**

**数据结构与算法之——基础概念介绍**

**本教程为本人亲自整理，结合大学及各种习题资料汇总而成**

**版权所有，转载请注明出处，谢谢合作！**

****安伟超

达内时代科技集团有限公司

天津天大中心教学部

[anwc@tedu.cn](mailto:anwc@tedu.cn)

17600945626

最后修改：2019.5.5

目录

[一、学习数据结构的意义 2](#_Toc7961864)

[二、数据结构体系架构 2](#_Toc7961865)

[三、数据结构定义 3](#_Toc7961866)

[数据 3](#_Toc7961867)

[数据结构 3](#_Toc7961868)

[四、数据结构的要素 4](#_Toc7961869)

[数据的逻辑结构 4](#_Toc7961870)

[数据的存储结构 4](#_Toc7961871)

[数据的运算 4](#_Toc7961872)

[逻辑结构表示法 5](#_Toc7961873)

[五、数据的逻辑结构及其类型 5](#_Toc7961874)

[逻辑结构类型 5](#_Toc7961875)

[线性结构 5](#_Toc7961876)

[树形结构 5](#_Toc7961877)

[图形结构 5](#_Toc7961878)

[三种逻辑结构对比图 6](#_Toc7961879)

[逻辑结构层次关系图 7](#_Toc7961880)

[逻辑结构二元组表示法 7](#_Toc7961881)

[六、抽象数据类型 9](#_Toc7961882)

[浅析 9](#_Toc7961883)

[定义 9](#_Toc7961884)

[抽象数据类型的实现思路 10](#_Toc7961885)

[基本思想 10](#_Toc7961886)

[基本操作 10](#_Toc7961887)

[引申：可变与不可变类型 11](#_Toc7961888)

[抽象数据类型在Python中的实际应用：面向对象编程(Object Oriented Programming,简称OOP) 11](#_Toc7961889)

[思考：数据结构是抽象数据类型的物理实现 11](#_Toc7961890)

[举例：抽象数据类型——复数 12](#_Toc7961891)

[七、算法 13](#_Toc7961892)

[什么是算法 13](#_Toc7961893)

[算法的重要特性 13](#_Toc7961894)

[算法的描述 14](#_Toc7961895)

[算法的复杂度 15](#_Toc7961896)

[问题由来——关注效率 15](#_Toc7961897)

[时间复杂度 16](#_Toc7961898)

[空间复杂度 18](#_Toc7961899)

[程序 = 数据结构 + 算法 18](#_Toc7961900)

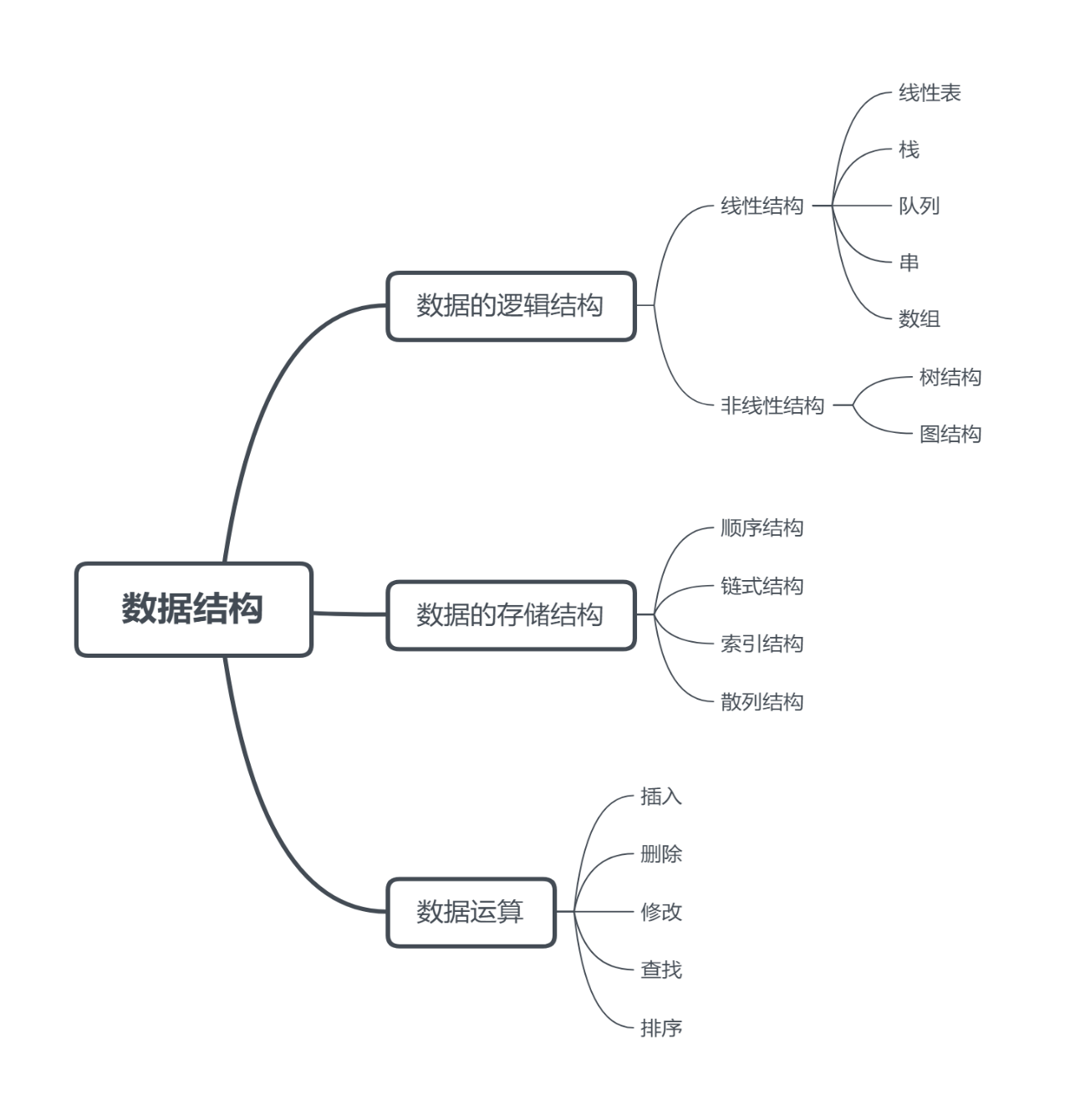
[简介 18](#_Toc7961901)

[实际应用思路 19](#_Toc7961902)

# 一、学习数据结构的意义

1. 掌握基本的编程方法
2. 掌握数据组织和数据处理的方法
3. 掌握大型软件开发的方法
4. 数据结构是计算机科学的重要基础。其他的计算机科学的基础理论知识：高等数学，线性代数——工程数学，概率论，数理统计，离散数学，操作系统，编译原理，数据库设计，软件工程，设计模式…

# 二、数据结构体系架构



# 三、数据结构定义

## 数据

1. 定义：数据是所有能被输入到计算机，且能被计算机处理的符号的集合。数据是计算机操作对象的总称，也是计算机处理的信息的某种特定的符号表示形式。
2. 数据元素：包含元素，节点，元素，是数据(集合)中的一个“个体”，是数据的基本单位
3. 数据项：具有独立含义的最小数据单位
4. 数据对象：具有相同性质的若干个元素的集合

## 数据结构

1. 名词定义：数据结构是指相互之间存在着一种或多种关系的数据元素的集合和该集合中数据元素之间的关系组成。记为Data\_Structure=(D,R)，其中D是数据元素的集合，R是该集合中所有元素之间的关系的有限集合。
2. 其他定义：
3. Sartaj Sahni在他的《数据结构、算法与应用》一书中称：“数据结构是数据对象，以及存在于该对象的实例和组成实例的数据元素之间的各种联系。这些联系可以通过定义相关的函数来给出。”他将数据对象（data object）定义为“一个数据对象是实例或值的集合”。
4. Clifford A.Shaffer在《数据结构与算法分析》一书中的定义是：“**数据结构是ADT（抽象数据类型Abstract Data Type） 的物理实现。**”(意思是抽象数据类型的概念都需要借助数据结构来进行实现)
5. Robert L.Kruse在《数据结构与程序设计》一书中，将一个数据结构的设计过程分成抽象层、数据结构层和实现层。其中，抽象层是指抽象数据类型层，它讨论数据的逻辑结构及其运算，数据结构层和实现层讨论一个数据结构的表示和在计算机内的存储细节以及运算的实现。

# 四、数据结构的要素

## 数据的逻辑结构

数据元素之间的逻辑关系，是数据在用户面前呈现的形式

例如现有一个学生信息表：

|  |  |  |  |
| --- | --- | --- | --- |
| 学号 | 姓名 | 性别 | 编号 |
| 1 | 张三 | 男 | 1001 |
| 2 | 李四 | 女 | 1002 |
| 3 | 王二 | 女 | 1003 |

## 数据的存储结构

数据元素及其关系在计算机中存储器中的存储方式，也称为数据的物理结构

例如针对上述逻辑表中张三的数据存储结构：

**class** Student(object):  
 **def** \_\_init\_\_(self):  
 self.name = **'张三'** self.sex = **'男'** self.num = 1001  
   
stu = Student()

## 数据的运算

基于数据之上的各种增删改查操作。

例如针对上述逻辑表：

1. 增加一个学生记录
2. 删除一个学生记录
3. 查找性别为“女”的学生记录
4. 查找班号为‘1001’的学生记录
5. …

## 逻辑结构表示法

1. 针对上述逻辑表，其一的表示方法即使用关系表进行表示
2. 使用线性关系来表示一组数据，在该线性关系中，每一个元素的记录顺序反应了数据元素之间的逻辑关系

# 五、数据的逻辑结构及其类型

## 逻辑结构类型

### 线性结构

定义：线性表（linear list）是数据结构的一种，一个线性表是n个具有相同特性的数据元素的有限序列。数据元素是一个抽象的符号，其具体含义在不同的情况下一般不同。线性表在数学概念中，其本质是一个集合。在稍复杂的线性表中，一个数据元素可由多个数据项（item）组成，此种情况下常把数据元素称为记录（record），含有大量记录的线性表又称文件（file）。线性表中的个数n定义为线性表的长度，n = 0 时称为空表。在非空表中每个数据元素都有一个确定的位置，如用ai表示数据元素，则i称为数据元素ai在线性表中的位序。线性表的相邻元素之间存在着序偶关系。如用（a1，…，ai-1，ai，ai+1，…，an）表示一个顺序表，则表中ai - 1 领先于ai，ai领先于ai+1，称ai - 1 是ai的直接前驱元素，ai + 1 是ai的直接后继元素。当i=1,2，…，n - 1 时，ai有且仅有一个直接后继，当i=2，3，…，n时，ai有且仅有一个直接前驱。

### 树形结构

定义：树形结构指的是[数据元素](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%85%83%E7%B4%A0)之间存在着“一对多”的树形关系的[数据结构](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E7%BB%93%E6%9E%84)，是一类重要的非线性数据结构。在树形结构中，树根结点没有前驱结点，其余每个结点有且只有一个前驱结点。叶子结点没有后续结点，其余每个结点的后续[节点](https://baike.baidu.com/item/%E8%8A%82%E7%82%B9)数可以是一个也可以是多个。另外，数学统计中的树形结构可表示层次关系。树形结构在其他许多方面也有应用。可表示从属关系、并列关系。

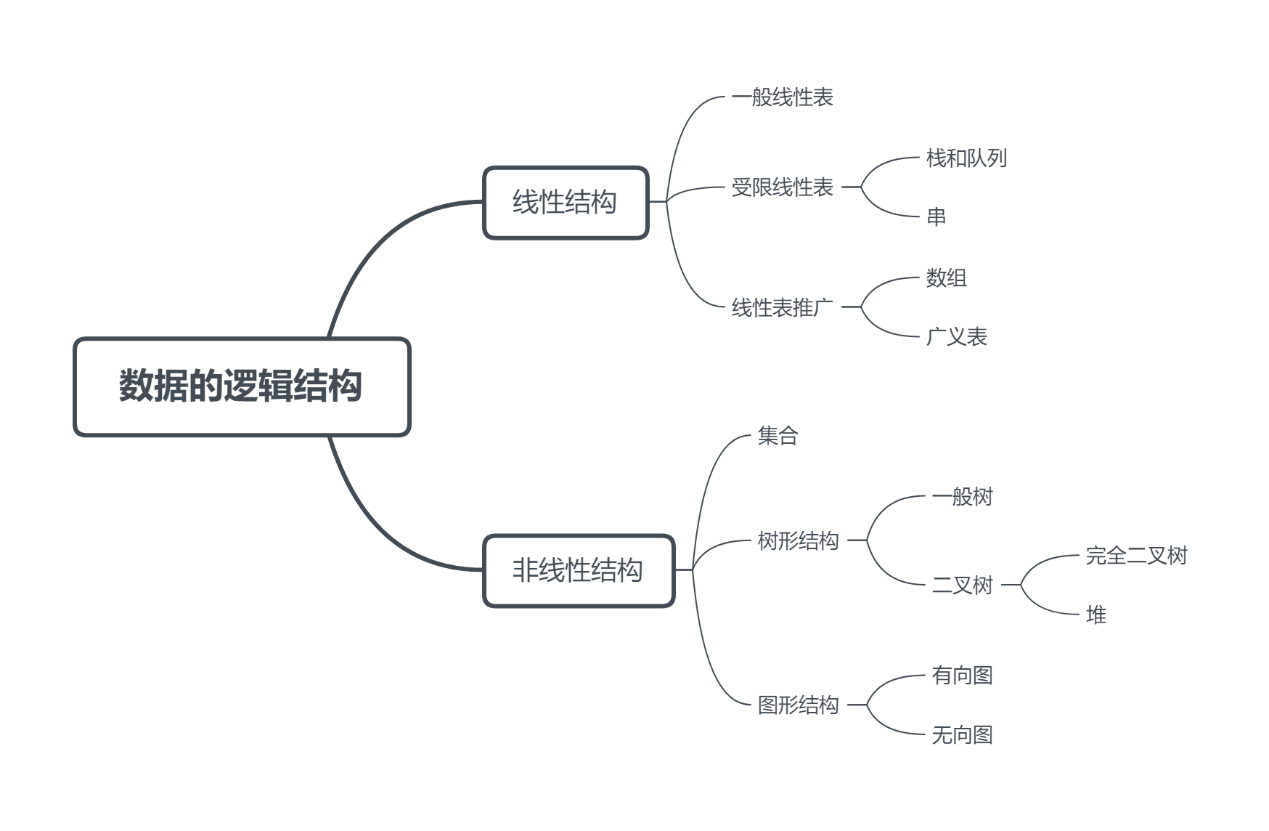
### 图形结构

定义：图型结构也称图案，指个体目标重复排列的空间形式。图案反映了地物的空间分布特征，它可以是自然的，也可以是人为构造的。 [1]  图形结构，简称“图”，是一种复杂的[数据结构](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E7%BB%93%E6%9E%84)。图形结构中，每个结点的前驱结点数和后续结点数可以任意多个。

## 三种逻辑结构对比图

|  |  |  |  |
| --- | --- | --- | --- |
| 逻辑结构名称 | 线性结构 | 树形结构 | 图形结构 |
| 图示 |  |  |  |
| 分析 | 1. 开始节点和终端节点都是唯一的 2. 除了开始节点和终端节点以外，其余节点都有且仅有一个后继节点 | 1. 开始节点唯一，终端节点不唯一 2. 除终端节点外，每个节点都有一个或多个后继节点 3. 除开始节点外，每个节点有且仅有一个前驱节点 | 没有开始节点和终端节点，所有节点都可能有多个前驱节点和多个后继节点 |

## 逻辑结构层次关系图



## 逻辑结构二元组表示法

对于数据的逻辑结构还有一种二元组表示方法。

逻辑结构的二元组表示法：。其中，B是一种数据结构，由数据元素的集合D与D上的二元关系的集合R所组成。

数据元素的集合D：

,表示集合D中的第i个节点或数据元素，n为D中节点的个数。特别地，若n=0,则D是空集。

二元关系的集合R：

表示集合R中的第j个关系，每个关系用序偶表示

何为序偶：

序偶<x,y>(x,yD)

x为第一元素

y为第二元素

x为y的前驱元素

y为x的后继元素

开始元素：没有前驱元素的节点

终端元素：没有后继元素的节点

<x,y>表示有向关系

(x,y)表示无向关系

m为R中关系的个数，特别地，若m=0，则R是空集，表明集合D中的元素间不存在任何关系，彼此独立

例如：城市表的逻辑结构表示

|  |  |  |
| --- | --- | --- |
| 区号 | 城市名 | 说明 |
| 010 | Beijing | 首都 |
| 021 | Shanghai | 直辖市 |
| 027 | Wuhan | 湖北省省会 |
| 029 | Xian | 陕西省省会 |
| 025 | Nanjing | 江苏省省会 |

现将其采用二元组进行表示：

City=(D,R)

D={010,021,027,029,025}

R={r}

r={<010,021>,<021,027>,<027,029>,<029,025>}

再例：有如下矩阵：

现将其采用二元组进行表示：

B=(D,R)

D={2,6,3,1,8,12,7,4,5,10,9,11}

R={}(表示行，表示列)

={<2,6>,<6,3>,<3,1>,<8,12>,<12,7>,<7,4>,<5,10>,<10,9>,<9,11>}

={<2,8>,<8,5>,<6,12>,<12,10>,<3,7>,<7,9>,<1,4>,<4,11>}

# 六、抽象数据类型

## 浅析

在实际开发中，无论是设计所谓的xx管理系统，亦或是开发一个xx控制系统，其实无非就是在跟数据打交道。而为了实现这些项目的开发，我们需要用到各种各样的编程语言，诸如C++，Java，Python，PHP……可是，在这些高级编程语言中，只有一些最最基本的数据类型，比如Python，有数字类型，字符型，序列，容器等，可是，仅凭这些基本数据类型，是无法满足我们的开发需求的，道理也很简单，难道只用这几种数据类型的数据，我就能开发出来一个网站？一个APP？当然不可能。这就是有关数据，产生的第一个问题：数据是否够用？

再比如，我独立开发出来了一个软件，哇很牛逼，非常受欢迎，那我肯定得指着他挣钱，但是为了让他挣钱就必须让他商业化，商业化意味着该软件必须投入使用，那么无论这个软件是C/S还是B/S架构，任何人都可以看得到你的源代码，甚至可以随意修改。显然，这是不安全的。这就是有关数据，产生的第二个问题：数据是否安全？何为安全，即：数据是否可以随意访问，数据内部是否可以随意修改？

再举例，比如我想开发一个电子控制系统，需要硬件，软件，硬件需要CPU，单片机，电源等等，软件需要采用c进行嵌入式开发；或者我想开发一个开关控制系统，也是需要硬件，软件，硬件同样也需要单片机，软件同样要用到c，那么问题来了：难道我每想做一个具体的产品，我都得一点一点的从头开始设计吗？这些软件之间是否有共性的问题呢？换句话表达：数据之间是否有共性呢？如何以数据为中心，如何围绕数据进行思考和设计？这就是有关数据，产生的第三个问题。

基于以上三个问题，我们现在引入抽象数据类型的概念。

## 定义

**全称：Abstract Data Type**

**中文名称：抽象数据类型**

**英文简称：ADT**

**定义：**

**1.百度百科定义：**抽象数据类型(ADT)是指一个**数学模型以及定义在此数学模型上的一组操作。**抽象数据类型需要通过固有数据类型（高级编程语言中已实现的数据类型）来实现。抽象数据类型是与表示无关的数据类型，是一个数据模型及定义在该模型上的一组运算。对一个抽象数据类型进行定义时，必须给出它的名字及各运算的运算符名，即函数名，并且规定这些函数的参数性质。一旦定义了一个抽象数据类型及其具体实现，程序设计中就可以像使用基本数据类型那样，十分方便地使用抽象数据类型

**2.通俗解释**：最开始的计算机语言，关注的都是如何更加有效率地计算，可以说其目的是计算层面的抽象。然而随着这个行业的不断发展，计算机不仅仅用于计算，开发也不仅只关注计算过程了，**数据层面**的抽象也变得同样重要。虽然计算机语言一开始就有对数据的抽象，但是那些都只是对一些最基本的数据类型而不包括我们想要用的，多种多样的数据。

程序处理的数据，通常是不同的类型的。只有事先约定好不同类型的数据的存储方式，计算机才能正确理解逻辑上不同的数据类型。所有编程语言都会有一组内置的基本数据类型。另外在实际工作过程中，或早或晚总会碰到一些没法用现有数据类型解决的问题，这时就需要自定义一些数据类型来解决。像Python这样比较高级的语言，在基本类型的基础上还添加了一些额外的数据结构如tuple,list,dict（这些如果从广义上来说也算是Python的数据类型）。

以上基本数据类型都是比较simple的结构，但是他们具有一个共通的问题，就是都会把数据暴露在外。这点类似于一个人如果有了对某个变量的权限的话他就可以看到这个变量代表的数据结构中的所有数据。而如果我们不希望被看到，那么这个问题可能就比较严重了。为了解决这个问题，就必须要有一种数据类型，它可以让使用者只需要考虑如何去使用，而不需要去关注类型内部的具体实现方式以及数据的表示等等。这样的类型从概念上来说就是抽象数据类型了。

**抽象数据类型 = 逻辑结构+抽象运算。**逻辑结构，即我们所说的数据之间的逻辑关系，抽象运算，即我们赋予抽象数据类型能够提供的自定义“运算”（操作||功能）

抽象数据类型，实质上，就是在描述问题本身，而跟计算机，是没有任何的关系的，也就是我们前面所说到的，脱离计算机，以数据为中心，围绕数据展开思考与设计。

抽象数据类型存在的目标是什么？很简单，即：在不涉及具体的，和计算机系统相关的细节情况下，优先考虑，理解，思考问题本身，在此基础之上，实现用计算机求解问题的过程。

## 抽象数据类型的实现思路

### 基本思想

实现一个抽象数据类型，其基本思想均是把数据定义为抽象的数据对象集合，只为他们定义可用的合法操作，而不暴露具体的内部细节。注意：不论是操作细节还是数据存储细节均不暴露

### 基本操作

基于以上的基本实现思想，一般而言，抽象数据类型应该至少具备以下三种操作：

1. 构造操作：即如何通过抽象数据类型，创建抽象数据对象
2. 解析操作(也可以理解为获取操作)：一定会有一系列getxxx()方法去获取该抽象数据类型内部的任意数据及操作
3. 修改操作：一定会有一系列的setxxx()方法去给当前抽象数据类型修改任意数据及操作

### 引申：可变与不可变类型

基于以上说到的三种基本操作，可以判断一个数据类型的可变性

1. 如果一个抽象数据类型同时具备以上三种操作，那么这个抽象数据类型就是可变的
2. 如果一个抽象数据类型只具备一二两种操作，不具备第三种操作，那么这个抽象数据类型就是不可变的

### 抽象数据类型在Python中的实际应用：面向对象编程(Object Oriented Programming,简称OOP)

### 思考：数据结构是抽象数据类型的物理实现

基于第六节：抽象数据类型的所有内容，我们对这句话应该就不难理解了。抽象数据类型，是我们自定义的，区别于各大编程语言中的那些基本的数据类型，它高度自定义，可以存储任意数据，解决了开发过程中数据不够使用的问题，同时，抽象数据类型只提供功能，不提供类型内部的数据及操作的访问权限，解决了数据使用过程中的安全性问题，再者，抽象数据类型将开发过程中对计算机运算的关注转移到对数据的抽象，实现了以数据为中心的思考与设计。但，其终归只是一种类型。而在数据结构的概念中，一切的概念都是**围绕一个个具体的数据对象展开的**，这些数据对象，就是**通过一个个具体的抽象数据类型来创建的，由类型得到对象，由抽象的概念实例化出一个具体存在的数据，这个过程，就叫做物理实现。**所以，Clifford A.Shaffer在《数据结构与算法分析》一书中，对数据结构的定义是：“**数据结构是ADT（抽象数据类型Abstract Data Type） 的物理实现。**”

## 举例：抽象数据类型——复数

存储结构：

**class** Complex(object):  
 *'''  
 创建一个复数抽象数据类型  
 定义成员：  
 数据：实部，虚部  
 操作：  
 1.构造操作  
 2.删除操作  
 3.返回复数z实部值操作  
 4.返回复数z虚部值操作  
 ...  
 '''* **def** \_\_init\_\_(self):  
 self.real;#定义实部  
 self.imag;#定义虚部  
   
 **def** AssignComplex(self,z,v1,v2):  
 *'''构造一个复数z'''* **pass** ...  
   
 **def** DestroyComplex(self,z):  
 *'''删除(销毁)一个复数z'''* **pass** ...  
   
 **def** getReal(self,z):  
 *'''返回复数z的实部值'''* **pass** ...  
   
 **def** getImag(self,z):  
 *'''返回复数z的虚部值'''* **pass** ...  
 ...

逻辑结构（采用二元组表示法）：

ADT Complex{

数据对象：D={real,imag|real,imag均为实数}

数据关系：R={<real，imag>|real是实部，imag是虚部}

基本操作:

AssignComplex(z,v1,v2):构造复数z

DestroyComplex(z):删除(销毁)一个复数

GetReal(z):返回复数z的实部

GetImag(z):返回复数z的虚部

…

}

# 七、算法

## 什么是算法

1. 通俗定义：算法是指解题方案的准确而完整的描述，是一系列解决问题的清晰指令

算法代表着用系统的方法描述解决问题的策略机制。也就是说，能够对一定规范的输入，在有限时间内获得所要求的输出。

1. 其他定义1：算法是在具体存储结构上，实现某个抽象运算
2. 其他定义2：算法是对特定求解方法(步骤)的一种描述，是指令的有限序列，

其中每一条指令表示一个或多个操作。

## 算法的重要特性

1. 有穷性：即算法需要再有限个步骤之后结束
2. 确定性：即无二义性，无歧义产生
3. 可行性：可通过基本运算有限次执行来实现
4. 有一个或多个确定的输入
5. 有一个或多个确定的输出

一个优秀的算法，还应该具备以下特点：

1. 正确性：算法应该满足具体问题的需求
2. 可读性：算法应容易供人阅读和交流。可读性好的算法有助于对算法的理解和修改
3. 健壮性：算法应具有容错处理。当输入非法或者错误的数据时，算法应能适当的做出反应或进行处理，而不会莫名其妙的输出错误结果。
4. 通用性：算法应该具有一般性，即算法的处理结果对于一般的数据集合都是成立的
5. 效率与存储量需求：效率指的是算法执行的时间，存储量需求值算法执行过程中所需要的最大存储空间。一般的，这两者与问题的规模有关。

## 算法的描述

1. 自然语言
2. 伪代码

伪代码（Pseudocode）是一种非正式的，类似于英语结构的，用于描述模块结构图的语言。人们在用不同的编程语言实现同一个算法时意识到，他们的实现（注意：这里是实现，不是功能）很不同。尤其是对于那些熟练于不同编程语言的程序员要理解一个（用其他编程语言编写的程序的）功能时可能很难，因为程序语言的形式限制了程序员对程序关键部分的理解。这样伪代码就应运而生了。伪代码提供了更多的设计信息，每一个模块的描述都必须与[设计结构图](https://baike.baidu.com/item/%E8%AE%BE%E8%AE%A1%E7%BB%93%E6%9E%84%E5%9B%BE" \t "_blank)一起出现。

使用伪代码的目的是使被描述的算法可以容易地以任何一种编程语言（Pascal，C，Java等）实现。因此，伪代码必须结构清晰、代码简单、可读性好，并且类似自然语言。 介于自然语言与编程语言之间。以编程语言的书写形式指明算法职能。使用伪代码， 不用拘泥于具体实现。相比程序语言（例如Java, C++,C, Dephi 等等）它更类似自然语言。它是[半角](https://baike.baidu.com/item/%E5%8D%8A%E8%A7%92" \t "_blank)式化、不标准的语言。可以将整个算法运行过程的结构用接近自然语言的形式（可以使用任何一种你熟悉的文字，关键是把程序的意思表达出来）描述出来。

说白了，平时我们写程序时编写的每一行代码的注释，实质上就是伪代码。

l = [3,1,5,7,2,4,9,6]  
**for** i **in** range(0, len(l)):  
 *# 默认设置最小值的下标为当前值* min = i  
 *# 在剩余n-1个元素的序列中找到最小元素的下标* **for** j **in** range(i + 1, len(l)):  
 *# if l[i] > l[i+1]  
 # 如果找到，就把最小元素的下标赋值给min* **if** l[min] > l[j]:  
 min = j  
 *# 将找到的最小值的元素和当前元素做位置交换* temp = l[min]  
 l[min] = l[i]  
 l[i] = temp  
print(l)

以上是某排序算法的代码举例，如果将其中的代码删除并将注释加以完善，得到如下伪代码：

*# 获取一个随机序列  
# 遍历当前序列  
 # 默认设置最小值的下标为当前值  
 # 在剩余n-1个元素的序列中找到最小元素的下标  
 # 如果找到，就把最小元素的下标赋值给min  
 # 将找到的最小值的元素和当前元素做位置交换  
# 输出结果*

1. 传统流程图

使用图形表示算法的思想是一种极好的方法，因为千言万语敌不过一张图来表示。以特定的图形符号加上说明，表示算法的图，称为流程图或[框图](https://baike.baidu.com/item/%E6%A1%86%E5%9B%BE)。

流程图是流经一个系统的信息流、观点流或部件流的图形代表。在企业中，流程图主要用来说明某一过程。这种过程既可以是生产线上的[工艺流程](https://baike.baidu.com/item/%E5%B7%A5%E8%89%BA%E6%B5%81%E7%A8%8B" \t "_blank)，也可以是完成一项任务必需的管理过程。

1. N-S结构化流程图

在传统流程图的使用过程中，人们发现流程线不一定是必需的，随着[结构化程序设计方法](https://baike.baidu.com/item/%E7%BB%93%E6%9E%84%E5%8C%96%E7%A8%8B%E5%BA%8F%E8%AE%BE%E8%AE%A1%E6%96%B9%E6%B3%95)(structured programming, SP）的出现，1973年美国学者Ike Nassi和Ben Shneiderman提出了一种新的流程图形式，这种流程图完全去掉了流程线，算法的每一步都用一个矩形框来描述，把一个个矩形框按执行的次序连接起来就是一个完整的算法描述。这种流程图同两位学者名字的第一个字母来命名，称为N-S流程图。

1. 程序设计语言

## 算法的复杂度

### 问题由来——关注效率

所谓效率，指的是单位时间完成的工作量，指最有效的使用社会资源以满足人类的愿望和需要。而算法的效率，指的是有效的使用计算机资源满足需求。考察算法的效率有两个维度：

时间：即算法的执行时间（实质是计算机CPU的计算资源占用及执行效率）

空间：及算法执行时所需的存储资源

但，我们不能使用程序执行时间去比较效率，原因：

1. 执行环境的不同。这个问题一般是跟PC的操作系统有关系。Windows一般比不过linux，而linux相较于unix也不及unix的效率。
2. 实现的语言不同。C是效率最高的高级编程语言，但其开发效率很低。Python是效率较低的高级编程语言，但其开发效率很高。（注意：目前PC的发展非常迅速，硬件的性能越来越高，比如CPU核数越来越多，线程数越来越高，内存，硬盘的转速越来越高，显卡的性能越来越强悍，所以，即使是Python这样的执行效率相对较低的高级编程语言，实质上其速度已经很快了，在这里我们只是和执行效率更高的高级编程语言有一个相对的比较而已）
3. 其他因素。

由此，引出算法复杂度的概念。

### 时间复杂度

#### 定义

时间复杂度是同一问题可用不同[算法](https://baike.baidu.com/item/%E7%AE%97%E6%B3%95/209025)解决，而一个算法的质量优劣将影响到算法乃至[程序](https://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F/13831935)的效率。算法分析的目的在于选择合适算法和改进算法。

计算机科学中，算法的时间复杂度是一个函数，它定性描述了该算法的运行时间。这是一个关于代表算法输入值的[字符串](https://baike.baidu.com/item/%E5%AD%97%E7%AC%A6%E4%B8%B2/1017763)的长度的函数。

#### 基本运算

一个算法是由控制结构（顺序，分支，循环）和源操作（固有数据类型的操作，比如比较，加减乘除取余等）构成的。

算法的时间复杂度取决于以上两者的综合。**一般地，我们使用算法的基本运算的次数度量一个算法的时间复杂度**

算法执行时间大致为**基本运算所需的时间与其运算次数的乘机。**

何为基本运算：**基本运算一般是最深层循环内的语句。**

#### 度量

算法中**基本运算次数T(n)是问题规模n的某个函数f(n),记作：T(n)=O(f(n))**

#### 算法的时间复杂度表示法

算法中**基本运算次数T(n)是问题规模n的某个函数f(n),记作：T(n)=O(f(n))**

“大O”：

1. 表示时间复杂度的“量级”
2. 随着问题规模n的增大，算法执行时间的增长率和f(n)的增长率相同

#### “O”的形式意义

若f(n)是一个正整数n的一个函数，存在一个正常数C和,使得当时，有

则，记：

其实质是：

#### 常见的复杂度量级

O(1)：常数阶

复杂度与问题规模n无关

O():对数阶

O(n):线性阶

O(n):线性对数阶

O():平方阶

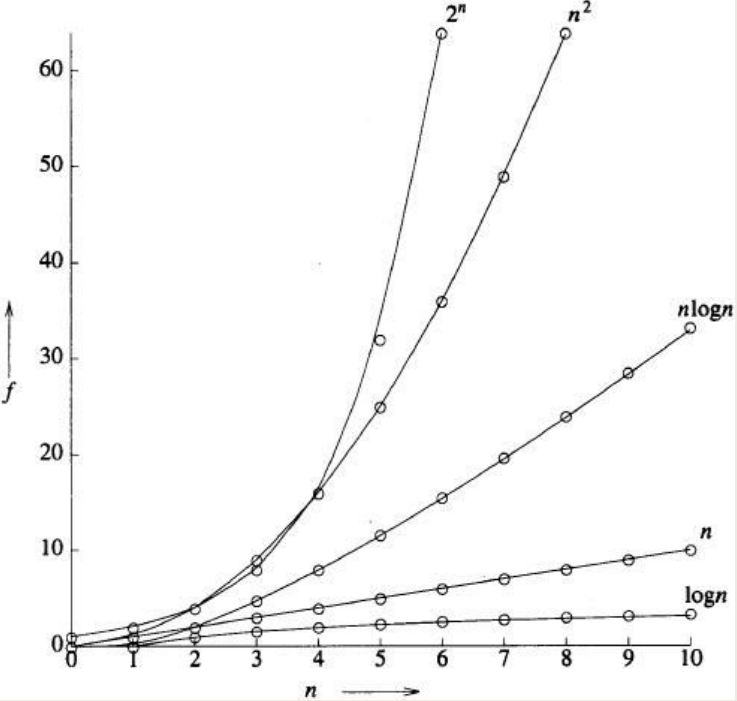
O():平方阶

O():指数阶

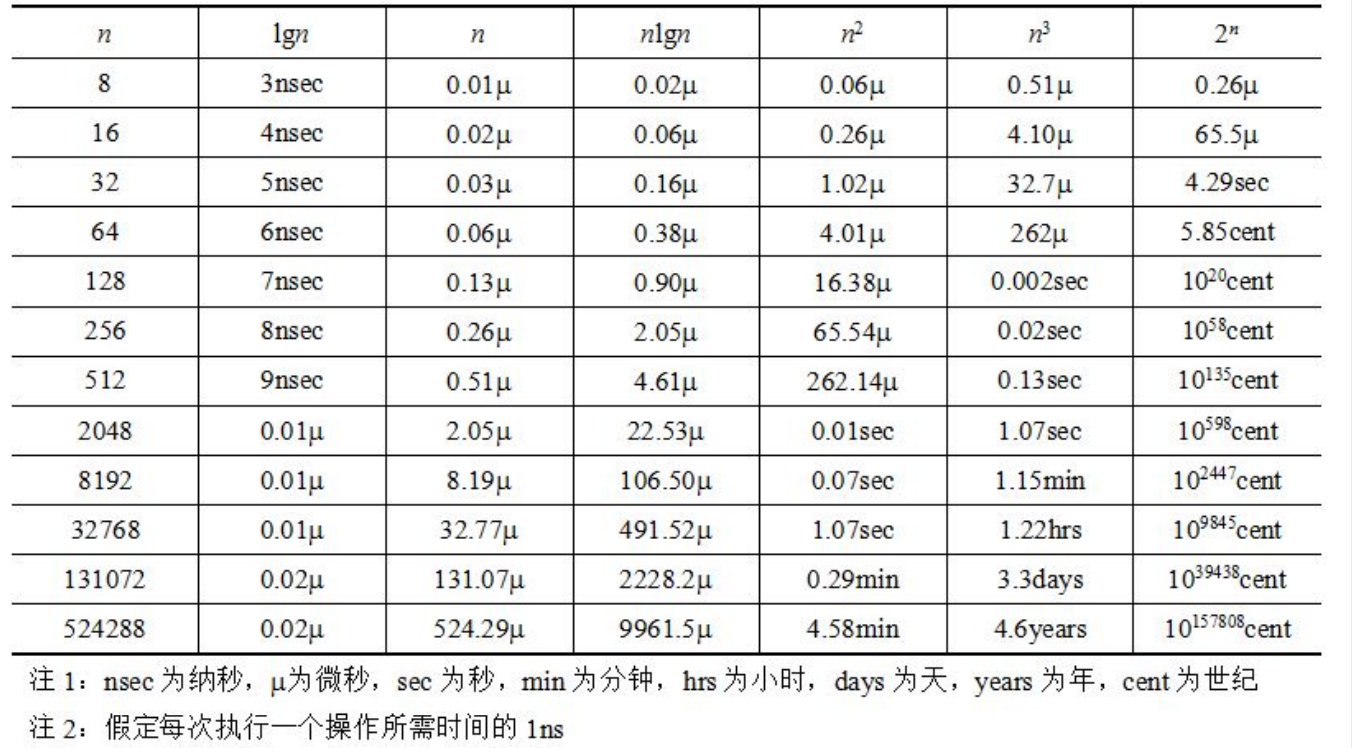
各个复杂度量级关系：

O(1) < O() < O(n) < O(n) < O() < O() < O()

各个复杂度量级图示如下图所示：



运行时间随问题规模n的变化规律，如下表所示：



### 空间复杂度

空间复杂度是对一个算法在运行过程中临时占用的存储空间大小的量度。空间复杂度一般也作为问题规模n的函数，以数量级的形式给出，记作：S(n)=O(g(n))

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——O(1)

若所需存储量依赖于特定的输入，则通常按照最坏情况去考虑。

## 程序 = 数据结构 + 算法

### 简介

Pascal之父——Nicklaus Wirth(尼古拉斯·沃斯)的至理名言：程序 = 数据结构 + 算法

这个公式对计算机科学的影响程度足以类似物理学中爱因斯坦的相对论e = mc^2，即，一个公式说明了程序的本质！

该公式可以继续细化为如下：

问题 --> 数据结构 + 算法 == 程序 -->解决问题

具体含义：程序，就是在数据的某些特定的表示方法和结构的基础上，对抽象算法的具体描述。

将松散，无组织的数据，按照某种要求组成一种数据结构，对于设计一个简明，高效，可靠的程序，是大有益处的

程序设计语言提供了实现数据结构和算法的机制

程序设计需要在数据结构和算法设计工作的基础之上才能完成

由程序设计语言描述的算法就是计算机程序

求解问题中，算法就是解决问题的方法，没有算法，程序就成了无本之末，无源之水

算法在程序设计，软件开发，甚至可以在整个计算机科学中的地位都是非常重要的

### 实际应用思路

一般地，拿到一个问题，我们都可以通过抽象数据类型进行描述，它由数据的逻辑结构和抽象运算两部分组成

一种数据的逻辑结构可以映射成多种存储结构，抽象运算算在不同的存储结构上实现可以对应多种算法，在同一存储结构上实现也可能有多种算法

通过算法的时间复杂度和空间复杂度等分析，就可以得到好的算法

好算法的标准：

正确、可读和可维护等

时间复杂度，空间复杂度较小

**抽象数据类型 = 数据的逻辑结构 + 抽象运算(运算的功能描述)**