

数据结构与算法

课程主要内容

- 各种数据结构（线性结构、树、图）的概念、特点、存储、基本算法
- 常用的排序、查找等各种算法的设计方法
- 文件的基本概念和各种结构
- 程序性能分析：空间复杂性、时间复杂性
- 算法设计基础与算法分析基础

前导课程

- 离散数学
具备一定的离散数学知识（集合，关系等）和一定的数学知识
- C++及C程序设计
具备C语言、C++语言知识（书中所有程序均须用C++实现）

教材及参考书

- 教材：

《数据结构与算法（C++版）》

唐宁九、游洪跃 主编
清华大学出版社 出版

- 参考书：

《数据结构C语言版》

严蔚敏、吴伟民 主编
清华大学出版社 出版

课程考核

课程成绩 = 平时成绩(50%)+期末考试(50%)

平时成绩 = 课堂参与(40%)+平时作业(40%)+半期随堂练习(20%)
期末成绩 = 期末考试(卷面50分强制达标)

第一章 绪论

- 1. 1 数据结构的概念
- 1. 2 数据结构的基本概念和术语
- 1. 3 抽象数据类型及其实现
- 1. 4 算法和算法分析

数据结构的概念

- 对于数值计算问题的解决方法，主要是用数学方程建立数学模型。
 - 天气预报的数学模型为二阶椭圆偏微分方程
 - 预测人口增长的数学模型为常微分方程
- 对于非数值计算问题，主要采用数据结构的方法建立数学模型。

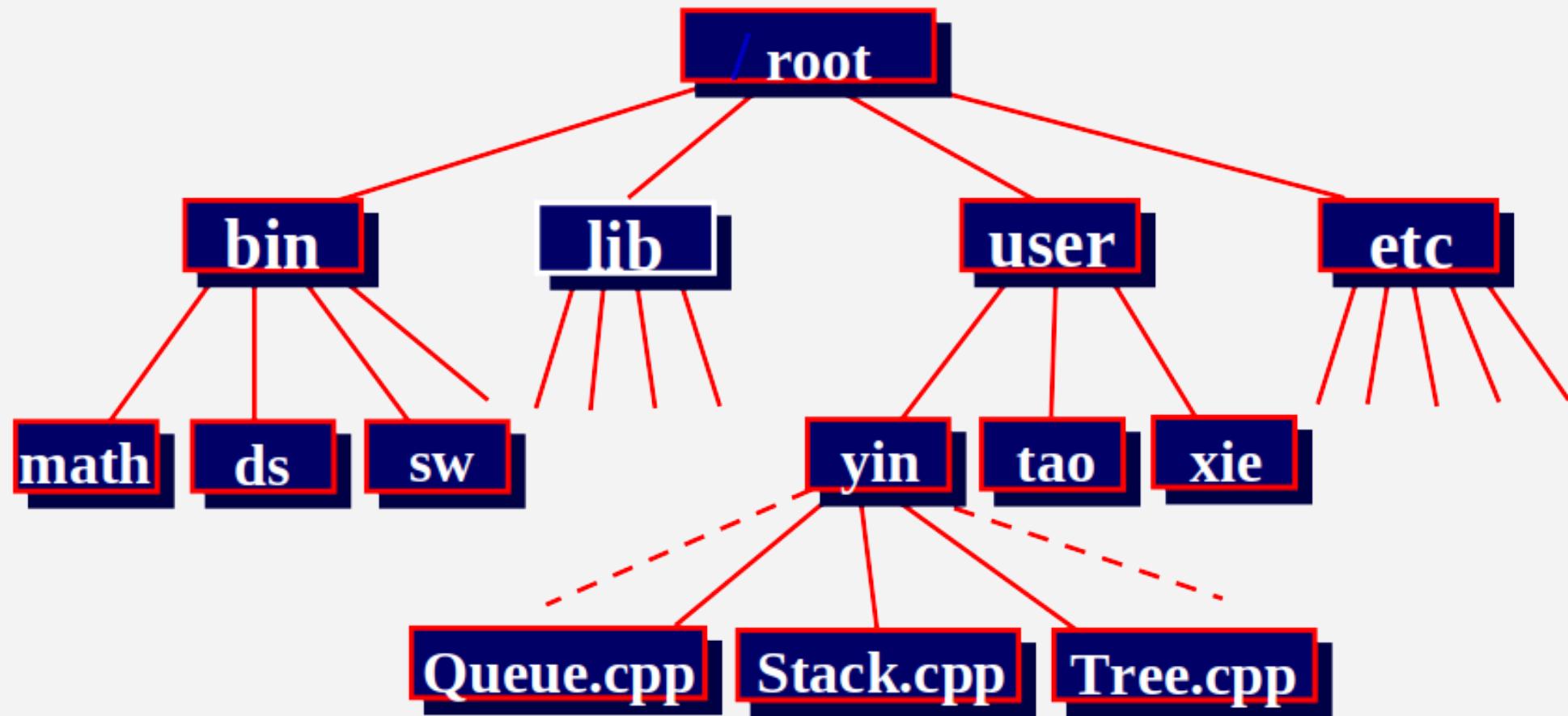
数据结构的概念

学生表格

学号	姓名	性别	籍贯	出生年月
98131	刘激扬	男	北京	1979.12
98164	衣春生	男	青岛	1979.07
98165	卢声凯	男	天津	1981.02
98182	袁秋慧	女	广州	1980.10
98224	洪伟	男	太原	1981.01
98236	熊南燕	女	苏州	1980.03
98297	宫力	男	北京	1981.01
98310	蔡晓莉	女	昆明	1981.02
98318	陈健	男	杭州	1979.12

数据结构的概念

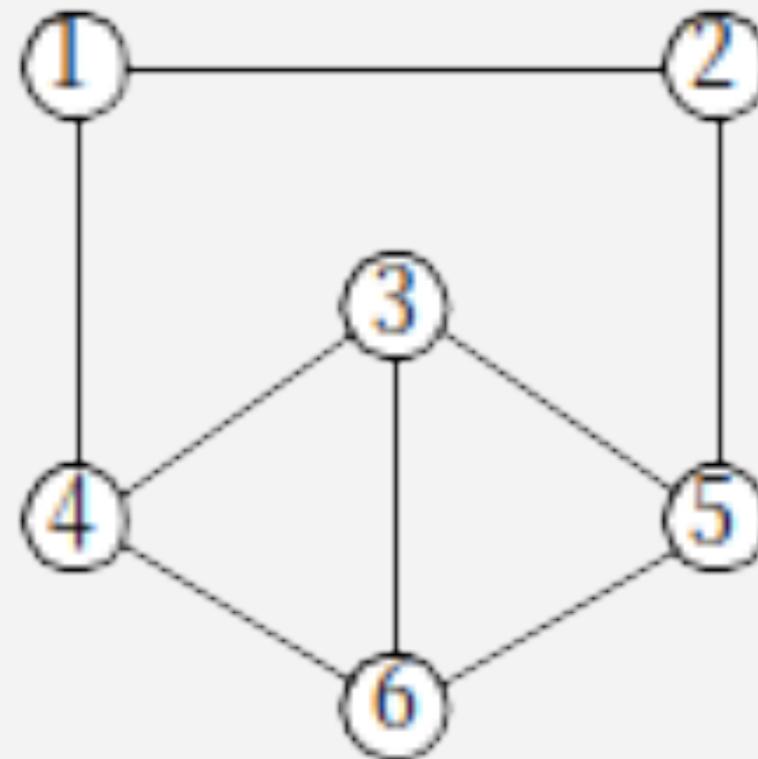
文件系统的系统结构图



数据结构的概念

要在n个网站建立通信网格，要求使得网络中任一网站出现故障时，整个网络仍能正常通信。

如图所示，这些网站之间形成一种图结构。



数据结构的概念

- 综合以上例子可见，描述这类非数值计算问题的数学模型是诸如表、树和图之类的数据结构。
- 因此，数据结构的研究范畴主要是非数值计算问题的操作对象及其它们之间的关系以及在计算机中的表示和实现。
- 算法+数据结构=程序。这里的数据结构指的是数据的逻辑结构和存储结构，而算法就是对数据运算的描述。

什么是数据结构？

- “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。”
(Sartaj Sahni, 《数据结构、算法与应用》)
- “数据结构是抽象数据类型 (Abstract DataType) 的物理实现。”
(Clifford A. Shaffer, 《数据结构与算法分析》)
- “数据结构 (data structure) 是计算机中存储、组织数据的方式。通常情况下，精心选择的数据结构可以带来最优秀率的算法。”
(维基百科)

什么是数据结构？

- 数据对象在计算机中的组织方式
 - ❖ 逻辑结构
 - ❖ 物理存储结构
- 数据对象与施加于其上的操作（算法）相关联
- 解决问题的效率跟数据的组织方式相关
- 解决问题的效率跟空间的利用率有关
- 解决问题的效率跟算法的设计有关

数据结构的基本概念和术语

- 数据 (data)：客观事物的符号表示；是计算机中可以操作的对象；所有能输入到计算机中被计算机程序识别、加工处理的符号的总称。
- 数值型数据：整数、实数和复数等
- 非数值型数据：文字、图形和语音等

数据结构的基本概念和术语

- 数据元素 (data element) : 数据的基本单位，在程序中作为一个整体考虑。称为元素、结点或记录等。一个数据元素可由若干数据项组成。
- 数据项 (data item) : 具有独立意义的不可分割的最小单位。

数据结构的基本概念和术语

学号	姓名	语文	数学	英语
S01	张	85	68	92
S02	李	87	75	79
S03	王	91	68	84

红色的整体为一个数据元素；黄色的为数据项。

数据结构的基本概念和术语

- 数据对象 (data object) : 具有相同性质的数据元素的集合，数据的一个子集。
- 数据类型 (data type) : 具有相同性质的计算机数据的集合及定义在这个数据集合上的一组操作的总称。高级程序语言中的数据类型可以分为两类：非结构的原子类型，如整型、实型等；另一类是结构类型。

数据结构的基本概念和术语

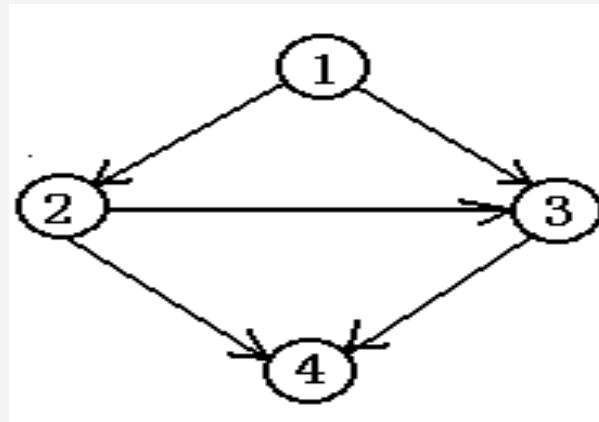
- 数据结构 (data structure)：相互之间存在着一定关系的数据元素的集合；数据元素相互之间的关系称为结构。
- 按某种逻辑关系组织起来的一批数据，按一定的存储表示方式把它们存储在计算机的存储器中，并在这些数据上定义一个运算的集合，这些的总合就叫做一个数据结构。

数据结构的基本概念和术语

数据结构的形式定义是一个二元数组。

$\text{DataStructure} = (D, S)$ ， 其中， D 是某一数据元素的有限集合， S 是该集合中所有数据元素之间的关系组成的有限集合。

数据结构的基本概念和术语



$$DS = \{D, S\}$$

$$D = \{1, 2, 3, 4\}$$

$$S = \{<1, 2>, <1, 3>, <2, 3>, <2, 4>, <3, 4>\}$$

数据结构的基本概念和术语

数据结构包含3个方面的内容：

- 数据之间的逻辑关系，也称为数据的逻辑结构；
- 数据元素及其关系在计算机存储器内的表示，称为数据的存储结构，也就是物理结构；
- 数据的运算，即对数据进行的操作；

数据结构的基本概念和术语

- 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学模型。

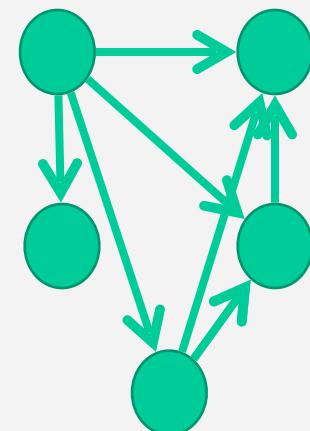
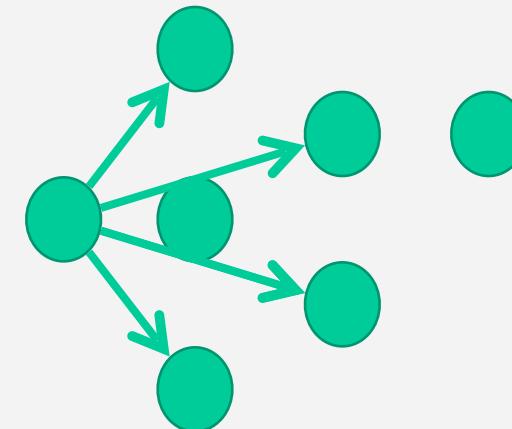
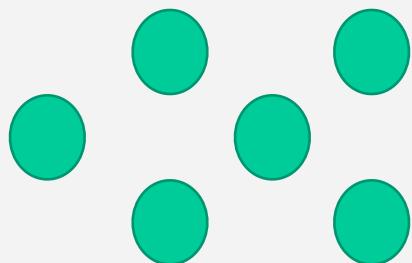
数据结构的基本概念和术语

数据的逻辑结构：一般分为4种

- 集合结构：数据具有符合某一条件的相同的性质，且无其他关系。
- 线性结构：数据之间存在一对一的关系。
- 树形结构：数据元素之间存在一对多的关系。
- 图状结构（网状结构）：数据之间存在多对多的关系。

数据结构的基本概念和术语

数据的逻辑结构：一般分为4种



数据结构的基本概念和术语

- 数据的存储结构：数据的逻辑结构在计算机中的表示或实现，又称数据的物理结构。物理结构是面向计算机的。
- 有2种表示方法，顺序映像和非顺序映像。

数据结构的基本概念和术语

数据的4种存储结构

- 顺序存储结构
- 链式存储结构
- 索引存储结构
- 散列存储结构

数据结构的基本概念和术语

- 顺序存储结构：属于顺序映像。特点是借助元素在存储器中的相对位置来表示数据元素之间的逻辑关系。
- 所有元素存放在一片连续的存储单元中，逻辑上相邻的元素存放到计算机内存仍然相邻。

数据结构的基本概念和术语

顺序存储结构：将逻辑上相邻的结点存储在物理位置上也相邻的存储单元里，结点之间的逻辑关系由存储单元的邻接关系来表示（只存储结点的值，不存储结点之间的关系）。

- 结点中只有自身信息，没有连接信息。存储密度大，空间利用率高。
- 可以通过计算直接确定数据结构中第 i 个结点的存储地址。
- 插入和删除会引起大量结点移动。

数据结构的基本概念和术语

- 链式存储结构：属于非顺序映像。借助指针表示数据元素之间的逻辑关系。所有元素放在可以不连续的存储单元中，但元素之间的关系可以通过地址确定。
- 逻辑上相邻的元素存放到计算机内存后不一定是相邻的。

数据结构的基本概念和术语

- 链式存储结构：不要求逻辑上相邻的结点在物理位置上也相邻，结点间的关系由附加的指针来表示。
- 指针指向结点的邻接结点，将所有结点串联在一起。不仅存储结点的值，而且存储结点之间的连接关系。

数据结构的基本概念和术语

- 链式存储结构中的结点由2部分组成，一个是存储结点本身的值，称为数据域；另一个是存储该结点的后续结点的存储单元地址，称为指针域。

数据结构的基本概念和术语

链式存储结构的主要特点

- 结点中有本身信息，还有表示链接信息的指针域，比顺序存储结构的存储密度小，存储空间利用率低。
- 逻辑上相邻的结点，物理上不必邻接，可用于线性表、树、图等多种逻辑结构的存储表示。
- 删除和插入操作灵活方便，不必移动结点，只要改变结点中的指针值即可。

数据结构的基本概念和术语

- 索引存储结构：在存储结点信息的同时，再建立一个附加的索引表，然后利用索引表中索引项的值来确定结点的实际存储单元地址。
- 索引表中的每一项称为索引项，索引项的一般形式为关键字（地址），关键字能唯一标识一个结点。

数据结构的基本概念和术语

- 散列存储结构：根据结点的关键字直接计算出结点的存储地址。把结点的关键字作为自变量，通过一个称为散列函数的计算规则，确定出该结点的存储单元地址。（Hash表）

数据结构的基本概念和术语

- 数据的运算：数据的运算是定义在数据的逻辑结构之上的，每一种逻辑结构都有一个运算的集合。这些运算实际上是在数据元素上施加的一系列的抽象的操作。
- 主要的运算有：查找、插入、删除、修改、排序。
- 数据的运算是通过算法描述的。一个算法具有几个重要特性：输入、输出、有穷性、确定性、可行性。

抽象数据类型及其实现

- 数据类型：一组性质相同的值的集合，以及定义于这个值集合上的一组操作的总称。
- C语言中有如下数据类型
char int float double
字符型 整型 浮点型 双精度型

抽象数据类型及其实现

- 基本数据类型可以看作是计算机中已实现的数据结构。
- 构造数据类型由不同成员构成；构造数据类型可以由基本数据类型或构造数据类型组成。
- 数据类型是模板，必须定义属于某种数据类型的变量，才能参加运算。

抽象数据类型及其实现

- Abstract data type，简称ADT。就是对象的数学模型。是用户在数据类型基础上新定义的数据类型，包括数据和对数据的处理操作。简单地说，就是指一个数学模型以及定义在此数学模型上的一组操作。
- 由基本的数据类型组成，并包括一组相关的服务（或称操作）。

抽象数据类型及其实现

- 抽象性：用ADT描述程序处理的实体时，强调的是其本质的特征、其所能完成的功能以及它和外部用户的接口（即外界使用它的方法）。
- 封装性：将实体的外部特性和其内部实现细节分离，并且对外部用户隐藏其内部实现细节。
- 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，本书由类模板或类来实现。

算法和算法分析

- 算法：对特定问题求解步骤的一种描述，是指令的有限序列，其中每一条指令表示一个或多个操作。
- 算法是为了解决某类问题而规定的一个有限长的操作序列。

算法和算法分析

- 算法的6个基本特性：正确性、具体性、确定性、有限性、可读性、健壮性。
- 算法还应该具有2个特性：有输入、有输出。

算法和算法分析

- 正确性：正确性指必须完成所期望的功能，对算法是否“正确”的理解可以有如下四个层次
 - 程序中不含任何语法错误
 - 程序对于几组输入数据能够得出满足要求的结果
 - 程序对于精心选择的、典型的、苛刻的并带有刁难性的几组输入数据能够得出满足要求的结果
 - 程序对于一切输入数据都能得出满足要求的结果

算法和算法分析

- 具体性：一个算法必须由一系列具体操作组成，这里的“具体”指的所有操作都必须经过已实现的基本操作有限次来实现，并且所有操作都是可读的、可执行的，每一操作必须在有限时间内完成。

算法和算法分析

- 确定性：算法中的所有操作都必须有确切的含义，不能产生歧义，算法的执行者或阅读者都能明确其含义及如何执行。
- 有限性：对于任意一组合法输入值，在执行有限步骤之后一定能结束，即：算法中的每个步骤都能在有限时间内完成。

算法和算法分析

- 可读性：算法应具备良好的可读性，这样的算法有利于算法的查错及对算法的理解，一般算法的逻辑必须清楚、结构简单，所有标识符必须具有实际含义，算法中能加入适当注释，说明算法的功能、输入输出参数的使用规则等。

算法和算法分析

- 健壮性：健壮性指当输入数据非法时，算法能作适当的处理并作出反应，而不应死机或输出异常结果。
- 有输入：一个算法有零个或多个的输入，这些输入取自于某个特定的对象的集合。
- 有输出：一个算法有一个或多个输出，这些输出是同输入有着某些特定关系的量。

算法和算法分析

算法的4种描述方法：

- 用自然语言描述算法：自然语言（可以是中文形式，也可以是英文形式）可以描述算法
- 用流程图描述算法：一个算法可以用流程图的方式来描述，输入输出、判断、处理分别用不同的框图表示，用箭头表示流程的流向。这是一种描述算法的较好方法，目前在一些高级语言程序设计中仍然采用。也有其他的图形辅助工具
- 用其他方式描述算法：我们还可以用数学语言或约定的符号语言来描述算法
- 在本课中，我们将采用C++来描述算法，所有算法的描述都用C++中的函数模板或函数形式来描述

算法和算法分析

算法设计的要求

- 正确性：能够通过专业测试。
- 可读性：算法易于理解，具有可读性，易于编码，易于调试等。
- 健壮性：当输入非法数据时，算法也能作出适当的反应或进行相应的处理。
- 效率问题：执行算法所耗费的时间。
- 低存储量需求：算法执行过程中所需要的最大存储空间。

算法和算法分析

算法和程序的关系

- 算法着重体现思路和方法，程序着重体现计算机的实现
- 程序不一定满足有穷性（死循环），另外，程序中的指令必须是机器可执行的，算法中的指令无此限制
- 一个算法若用计算机语言来书写，它就是一个程序

算法和算法分析

算法分析的概念

- 对于一个算法的评价，首先考虑正确性，其次是运算量（运行效率），最后是所占存储空间。为定性分析，引入时间复杂度和空间复杂度的概念。
- 算法效率的度量：算法执行的时间需通过依据该算法编写的程序在计算机上运行时所消耗的时间来度量。

算法和算法分析

算法评价标准

- 时间特性：时间复杂度 $T(n)$
- 空间特性：空间复杂度 $S(n)$
- 一个特定算法的时间复杂度 $T(n)$ 和空间复杂度 $S(n)$ 的大小，只依赖于问题的规模（通常用整数量 n 表示），或者说，它是问题规模的函数。
- 时间复杂度和空间复杂度是对立统一的关系。

算法和算法分析

时间复杂度

- 算法由控制结构和基本操作组成，算法的时间性能是这两部分的综合效果。
- 为方便比较不同算法，通常选择一种特定问题的基本操作，以此基本操作的执行次数作为时间度量。

算法和算法分析

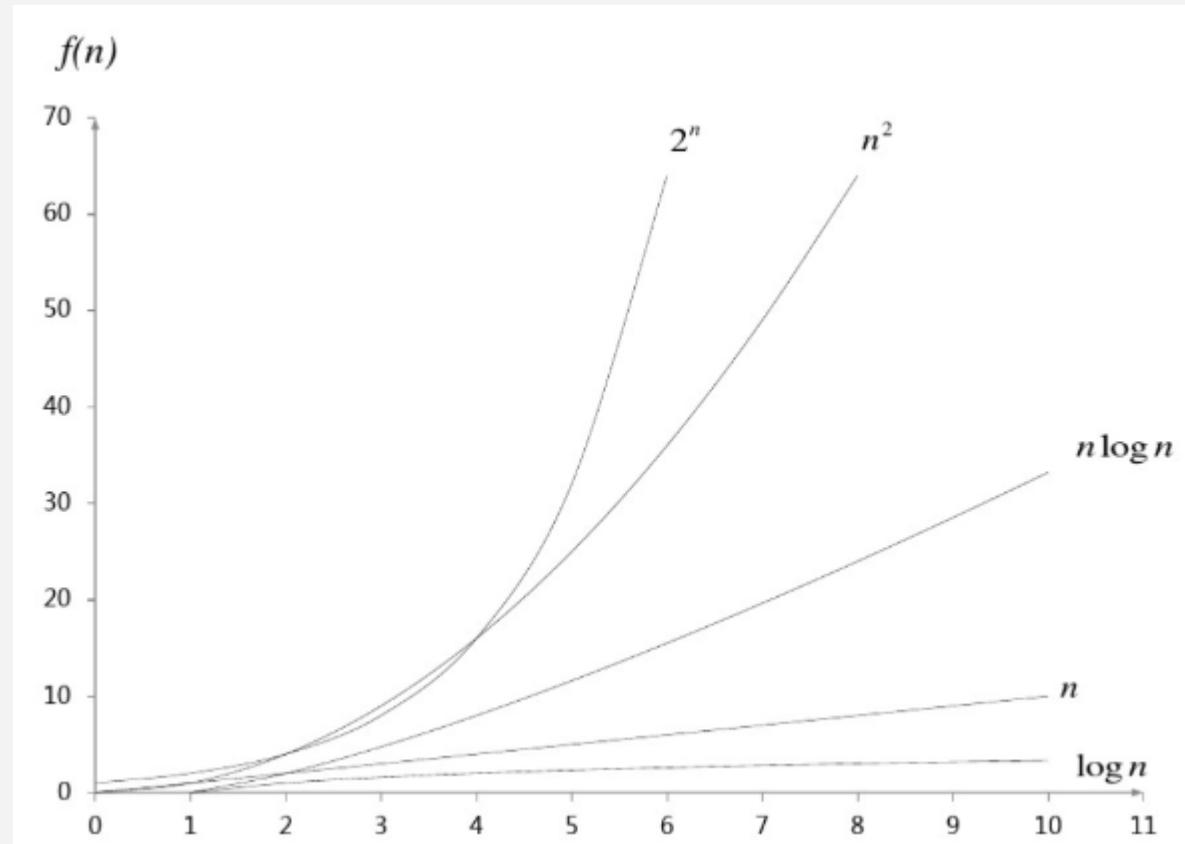
时间复杂度

- 在进行算法分析时，基本操作的执行次数 $T(n)$ 是关于问题规模n的函数，进而分析 $T(n)$ 随n的变化情况并确定 $T(n)$ 的数量级。
- 算法的时间复杂度，也即算法的时间度量，记作 $T(n)=O(f(n))$ ，表示随着问题规模n的增大，算法执行时间的增长率和 $f(n)$ 的增长率相同，称作算法的渐近时间复杂度，简称时间复杂度。其中 $f(n)$ 是问题规模n的函数。
- 一个立方函数就是它的关键项为某一个常数乘以 n^3 的函数；一个平方函数关键项是某个常数乘以 n^2 的函数。对于足够大的n来说，函数值主要由其关键项决定。

算法和算法分析

时间复杂度

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



算法和算法分析

时间复杂度

忽略低阶项和系数

- 用1取代运行时间中的所有加法常数
- 只保留最高阶项
- 最高阶项存在且不为1，则去除其它项

算法和算法分析

时间复杂度

```
int sum=0, n=1;  
sum += n+1;  
sum += n+2;  
sum += n+3;  
sum = sum/3;
```

$$T(n) = O(1)$$

算法和算法分析

时间复杂度

```
int n=1000, sum=0;  
for(int i=0; i<n; i++)  
{  
    sum += i;  
}
```

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

算法和算法分析

时间复杂度

```
int n=1000, sum=0;  
for(int i=0; i<n; i++)  
{  
    for(int j=0; j<n; j++)  
    {  
        sum += i;  
    }  
}
```

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

算法和算法分析

时间复杂度

```
int n=1000, sum=0;  
for(int i=0; i<n; i++)  
{  
    for(int j=i; j<n; j++)  
    {  
        sum += i;  
    }  
}
```

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

$$n + (n-1) + (n-2) + \dots + 1 = n(n+1)/2 = n^2/2 + n/2$$

算法和算法分析

时间复杂度

```
int n=1000, i=1;  
while( i < n )  
{  
    i = i * 2;  
}
```

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

$$2^x = n \quad x = \log_2 n \quad (\log_2 n \text{与} \log n \text{等价})$$

算法和算法分析

空间复杂度

- 算法的空间复杂度定义为： $S(n) = O(g(n))$
- 表示随着问题规模 n 的增大，算法运行所需存储量 $S(n)$ 的增长率与 $g(n)$ 的增长率相同，称 $S(n)$ (渐近) 空间复杂度。
- 算法原地工作指算法所需辅助空间是常量。

练习

- 1、算法的计算量的大小称为计算的（ ）。
A. 效率 B. 复杂性 C. 现实性 D. 难度
- 2、从逻辑上可以把数据结构分为（ ）两大类。
A. 动态结构、静态结构 B. 顺序结构、链式结构
C. 线性结构、非线性结构 D. 初等结构、构造型结构
- 3、以下与数据的存储结构无关的术语是（ ）。
A. 循环队列 B. 链表 C. 哈希表 D. 栈
- 4、连续存储设计时，存储单元的地址（ ）。
A. 一定连续 B. 一定不连续 C. 不一定连续 D. 部分连续，部分不连续
- 5、以下属于逻辑结构的是（ ）。
A. 顺序表 B. 哈希表 C. 有序表 D. 单链表
(B,C,D,A,C)

- 5、健壮的算法不会因非法的输入数据而出现莫名其妙的状态。()
- 6、算法可以用不同的语言描述，如果用C 语言或PASCAL语言等高级语言来描述，则算法实际上就是程序了。() 算法仍然是算法，只是被程序化地描述。
- 7、在顺序存储结构中，有时也存储数据结构中元素之间的关系。()
- 8、数据的逻辑结构说明数据元素之间的顺序关系,它依赖于计算机的储存结构。 ()
- 9、数据的物理结构包括 () 的表示和 () 的表示。
- 10、数据结构中评价算法的两个重要指标是 () 和 () 。
(√, ×, ×, ×, 数据元素、数据元素间关系, 时间复杂度、空间复杂度)

11、下列程序的时间复杂度是()

```
int sum = 0;  
  
for(int i=1; i<n; i*=2)  
  
    for(int j=0; j<i; j++)  
  
        sum++;
```

- A.O(log2n) B. O(n) C. O(nlog2n) D. O(n2)

12、设 n是描述问题规模的非负整数，下列程序段的时间复杂度是（ ）

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

- A. O(log2n) B. O($n^{1/2}$) C.O(n) D.O(n2)

(B, B)

13、可以用（ ）定义一个完整的数据结构

- A. 数据元素
- B. 数据对象
- C. 数据关系
- D. 抽象数据类型

14、在链接存储结构中，要求（ ）

- A. 每个结点占用一片连续的存储区域
- B. 所有结点占用一片连续的存储区域
- C. 结点的最后一个域是指针类型
- D. 每个结点有多少个后继就设多少个指针

15、计算机的算法指的是（ ）

- A. 计算方法
- B. 排序方法
- C. 解决某一问题的有限运算序列
- D. 调度方法

16、线性表是具有n个（ ）的有限序列（ $n > 0$ ）

- A. 表元素
- B. 字符
- C. 数据元素
- D. 数据项

(D, A, C, C)

求时间复杂度

```
void func(int n) {  
    int count = 0;  
    for (int i = 0; i < n; i++) {  
        for (int j = i; j < n; j++) {  
            for (int k = 0; k < j - i + 1; k++) {  
                count++;  
            }  
        }  
    }  
}
```

外层i从0到n-1，中层j从i到n-1，内层k执行次数为 $j - i + 1$ 。总次数可转化为三重求和，展开后最高次项为 $n^3 / 6$ ，忽略系数。

时间复杂度： $O(n^3)$

求时间复杂度

```
void func(int n) {  
    int count = 0;  
    for (int i = 1; i <= n; i++) {  
        if (i % 2 == 0) {  
            for (int j = 0; j < i; j++) {  
                count++;  
            }  
        } else {  
            for (int j = 0; j < n; j++) {  
                count++;  
            }  
        }  
    }  
}
```

偶数i（共n/2个）：内层执行i次，总和约 $(2+4+\dots+n) = n(n+2)/4 \approx n^2/4$ ；

奇数i（共(n+1)/2个）：内层执行n次，总和约 $n \times n/2 = n^2/2$ ；

总次数最高次项为 $n^2/2$ ，忽略系数。

时间复杂度：O(n^2)

求时间复杂度

```
int func(int n) {  
    if (n <= 1) return 1;  
    int sum = 0;  
    for (int i = 0; i < n; i++) {  
        sum += i; // 额外 O(n) 操作  
    }  
    return func(n/2) + func(n/2) + sum;  
}
```

递归树深度 $\log_2 n$, 每一层总操作数:

第 0 层 (根): n 次循环;

第 1 层: 2 个 $n/2$ 循环, 总 n 次;

... 每一层总操作数均为 n , 共 $\log_2 n$ 层;

总操作数 $n \times \log_2 n$ 。

时间复杂度: $O(n \log n)$

求时间复杂度

```
void func(int n) {  
    int dp[n][n];  
    // 初始化对角线  
    for (int i = 0; i < n; i++) {  
        dp[i][i] = 1;  
    }  
    // 填充上三角 (长度从 2 到 n)  
    for (int len = 2; len <= n; len++) {  
        for (int i = 0; i <= n - len; i++) {  
            int j = i + len - 1;  
            dp[i][j] = dp[i+1][j] + dp[i][j-1];  
        }  
    }  
}
```

初始化对角线: n 次操作;

填充上三角: len 从 2 到 n (共 $n-1$ 个值), 每个 len 对应 $n - len + 1$ 个 i , 总次数为 $1+2+\dots+(n-1) = n(n-1)/2$;

最高次项为 $n^2/2$ 。时间复杂度: $O(n^2)$

求时间复杂度

```
void func(int n) {  
    int count = 0;  
    for (int i = n; i > 0; i = i / 2) {  
        for (int j = 0; j < i; j++) {  
            if (j > 100) break; // 超过 100 次终止  
            count++;  
        }  
    }  
}
```

外层循环次数 $\log_2 n$ ，但内层j最多执行101次（0到100），与i无关（当*i*>100时，*j*=101 *break*；当*i*≤100时，*j*执行*i*次，最多100次）。总次数约 $101 \times \log_2 n$ ，忽略常数。

时间复杂度：O(log n)

求时间复杂度

```
void func(int n) {  
    if (n == 0) return;  
    // 嵌套循环  
    for (int i = 0; i < n; i++) {  
        for (int j = 0; j < n; j++) {  
            printf("*");  
        }  
    }  
    func(n - 1);  
}
```

递归调用n次（n到0），每次调用执行 n^2 次循环。总次数为 $n^2 + (n-1)^2 + \dots + 1 = n(n+1)(2n+1)/6$ ，最高次项 $2n^3/6 = n^3/3$ 。

时间复杂度：O(n^3)

求时间复杂度

```
void func(int n) {  
    int count = 0;  
    for (int i = 0; i < n; ) {  
        if (i % 3 == 0) {  
            i += 2;  
        } else if (i % 3 == 1) {  
            i += 3;  
        } else {  
            i += 1;  
        }  
        count++;  
    }  
}
```

i每次最少增加1，最多增加3，循环次数在n/3到n之间，均与n成正比，忽略系数。
时间复杂度：O(n)

求时间复杂度

```
int func(int n) {  
    if (n <= 3) return 1;  
    return func(n/3) + func(n/3) + func(n/3);  
}
```

递归树深度 $\log_3 n$, 每一层节点数:

第 0 层: 1 个;

第 1 层: 3 个;

第 k 层: 3^k 个;

总节点数为等比数列求和 $(3^{(\log_3 n + 1)} - 1)/2 = (3n - 1)/2 \approx 3n/2$ 。

时间复杂度: $O(n)$

求时间复杂度

```
void func(int n) {  
    int count = 0;  
    for (int i = 1; i <= n; i *= 2) {  
        for (int j = i; j <= n; j += i) {  
            count++;  
        }  
    }  
}
```

外层*i*为 $1, 2, 4, \dots, 2^k$ ($k=\log_2 n$)，内层*j*每次增加*i*，执行次数为*n/i*。总次数为 $n/1 + n/2 + n/4 + \dots + n/n = n(1 + 1/2 + 1/4 + \dots + 1) \approx 2n$ 。

时间复杂度: $O(n)$

求空间复杂度

```
void func(int n) {  
    int arr[100]; // 固定大小数组  
    for (int i = 0; i < 100; i++) {  
        arr[i] = i;  
    }  
}
```

局部变量arr大小固定（100 个 int），i为单个 int，均与n无关，占用空间为常数。

空间复杂度：O(1)

求空间复杂度

```
void func(int n) {  
    int arr[n]; // 大小随n变化的变长数组 (VLA)  
    for (int i = 0; i < n; i++) {  
        arr[i] = i;  
    }  
}
```

变长数组arr的大小为n个 int，占用空间与n成正比；i为常数空间。总空间由arr主导。

空间复杂度：O(n)

求空间复杂度

```
int func(int n) {  
    if (n == 0) {  
        return 1;  
    }  
    return func(n - 1) * 2; // 每次递归 n 减 1  
}
```

递归调用时，每次调用会创建栈帧（存储参数、返回地址等）。递归深度为n（从n到0），栈帧数量与n成正比，每个栈帧空间为常数。

空间复杂度：O(n)

求空间复杂度

```
int func(int n) {  
    int arr[5]; // 固定大小局部数组  
    if (n <= 1) {  
        return 1;  
    }  
    return func(n - 1) + func(n - 2); // 斐波那契递归  
}
```

递归深度最大为n（如计算func(n)需先算func(n-1)，直到func(1)），每个栈帧包含固定大小的arr和参数，总栈空间与递归深度n成正比。

空间复杂度：O(n)

求空间复杂度

```
void func(int n) {  
    int* p = (int*)malloc(n * sizeof(int)); // 动态分配n个int  
    if (p != NULL) {  
        free(p);  
    }  
}
```

动态分配的内存大小为n个 int，与n成正比；指针p为常数空间。总空间由动态内存主导。

空间复杂度：O(n)

求空间复杂度

```
void func(int n) {  
    int arr[10][20]; // 10 行 20 列的固定二维数组  
    for (int i = 0; i < 10; i++) {  
        for (int j = 0; j < 20; j++) {  
            arr[i][j] = i + j;  
        }  
    }  
}
```

二维数组arr总元素数为 $10 \times 20 = 200$ （固定），i、j为常数空间，整体占用空间与n无关。

空间复杂度：O(1)

求空间复杂度

```
void func(int n) {  
    // 动态分配n行指针数组  
    int** mat = (int**)malloc(n * sizeof(int*));  
    if (mat == NULL) return;  
    // 每行分配n个int  
    for (int i = 0; i < n; i++) {  
        mat[i] = (int*)malloc(n * sizeof(int));  
    }  
    // 释放内存 (省略)  
}
```

指针数组mat占用 $n \times \text{sizeof}(\text{int}^*)$ 空间；

每行动态数组占用 $n \times \text{sizeof}(\text{int})$ 空间，共n行，总 $n^2 \times \text{sizeof}(\text{int})$ 空间；

总空间最高次项为 n^2 ，与 n^2 成正比。

空间复杂度： $O(n^2)$

求空间复杂度

```
int func(int n) {  
    if (n <= 1) {  
        return 1;  
    }  
    // 二分递归：两个 n/2 子问题  
    return func(n / 2) + func(n / 2);  
}
```

递归树深度为 $\log_2 n$ （从n到1，每次除以 2），栈空间由递归深度决定（同一时间栈中最多存在 $\log_2 n$ 个栈帧），每个栈帧为常数空间。

空间复杂度： $O(\log n)$

求空间复杂度

```
// 全局固定大小数组  
int global_arr[1000];  
  
void func(int n) {  
    for (int i = 0; i < 1000; i++) {  
        global_arr[i] = n;  
    }  
}
```

全局数组global_arr大小固定（1000 个 int），与n无关；局部变量i为常数空间。全局空间不随输入变化，整体为常数空间。

空间复杂度：O(1)

求空间复杂度

```
void func(int n) {  
    // 循环分配 n 块，每块固定 10 个 int  
    int* p[n];  
    for (int i = 0; i < n; i++) {  
        p[i] = (int*)malloc(10 * sizeof(int));  
    }  
    // 释放内存 (省略)  
}
```

指针数组p占用 $n \times \text{sizeof}(\text{int}^*)$ 空间；

每块动态内存大小固定（10 个 int），n块总占用 $10n \times \text{sizeof}(\text{int})$ 空间；
总空间与n成正比（最高次项为n）。

空间复杂度： $O(n)$