# 数据结构绪论

## 基本概念和术语

### 数据

是描述客观事物的符号，是计算机中可以操作的对象，是能被计算机识别，并输入给计算机处理的符号集合。

### 数据元素

是组成数据的、有一定意义的基本单位，在计算机中通常作为整体处理。也被称为记录。

### 数据项

一个数据元素可以由若干个数据项组成。

数据项是数据不可分割的最小单位。

### 数据对象

是性质相同的数据元素的集合，是数据的子集。

### 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。

## 逻辑结构与物理结构

### 逻辑结构

是指数据对象中数据元素之间的相互关系。

集合结构：集合结构中的数据元素除了同属于一个集合外，它们之间没有其他关系。

线性结构：线性结构中的数据元素之间是一对一的关系。

树形结构：树形结构中的数据元素之间存在一种一对多的层次关系。

图形结构：图形结构的数据元素是多对多的关系。

### 物理结构

是指数据的逻辑结构在计算机中的存储形式。

顺序存储结构：是把数据元素存放在地址连续的存储单元里，其数据间的逻辑关系和物理关系是一致的。

链式存储结构：是把数据元素存放在任意的存储单元里，这组存储单元可以是连续的，也可以是不连续的。

## 抽象数据类型

### 数据类型

是指一组性质相同的值的集合及定义在此集合上的一些操作的总称。

### 抽象数据类型

是指一个数学模型及定义在该模型上的一组操作。

# 算法

## 算法的定义

算法是解决特定问题求解步骤的描述，在计算机中表现为指令的有限序列，并且每条指令表示一个或多个操作。

## 算法的特性

五个基本特性：输入、输出、有穷性、确定性和可行性。

### 输入输出

算法具有零个或多个输入。

算法至少有一个或多个输出。

### 有穷性

算法在执行有限的步骤之后，自动结束而不会出现无限循环，并且每一个步骤在可接受的时间内完成。

### 确定性

算法的每一步骤都具有确定的含义，不会出现二义性。

### 可行性

算法的每一步都必须是可行的，也就是说，每一步都能够通过执行有限次数完成。

## 算法设计的要求

正确性、可读性、健壮性、时间效率高和存储量低。

### 正确性

算法的正确性是指算法至少应该具有输入、输出和加工处理无歧义性、能正确反映问题的需求、能够得到问题的正确答案。

算法的“正确”通常在用法上有很大的差别，大体分为以下四个层次。

算法程序没有语法错误。

算法程序对于合法的输入数据能够产生满足要求的输出结果。

算法程序对于非法的输入数据能够得出满足规格说明的结果。

算法程序对于精心选择的，甚至刁难的测试数据都有满足要求的输出结果。

### 可读性

算法设计的另一目的是为了便于阅读、理解和交流。

### 健壮性

当输入数据不合法时，算法也能做出相关处理，而不是产生异常或莫名其妙的结果。

### 时间效率高和存储量低

时间效率指的是算法的执行时间，对于同一个问题，如果有多个算法能够解决，执行时间短的算法效率高，执行时间长的效率低。

存储量需求指的是算法在执行过程中需要的最大存储空间，主要指算法程序运行时所占用的内存或外部硬盘存储空间。

## 算法效率的度量方法

### 事后统计方法

这种方法主要是通过设计好的测试程序和数据，利用计算机计时器对不同算法编制的程序的运行时间进行比较，从而确定算法效率的高低。

### 事前分析估算方法

在计算机程序编制前，依据统计方法对算法进行估算。

一个用高级程序语言编写的程序在计算机上运行时所消耗的时间取决于下列因素：

算法采用的策略、方法。（算法好坏的根本）

编译产生的代码质量。（需要由软件来支持）

问题的输入规模。

机器执行指令的速度。（硬件性能）

## 函数的渐近增长

函数的渐近增长：给定两个函数f(n)和g(n)，如果存在一个整数N，使得对于所有的n>N，f(n)总是比g(n)大，那么，我们说f(n)的增长渐近快于g(n)。

## 算法时间复杂度

定义：在进行算法分析时，语句总的执行次数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，也就是算法的时间量度，记作：T(n)=O(f(n))。它表示随问题规模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，简称为时间复杂度。其中f(n)是问题规模n的某个函数。

推导大O阶方法：

用常数1取代运行时间中的所有加法常数。

在修改后的运行次数函数中，只保留最高阶项。

如果最高阶项存在且不是1，则去除与这个项相乘的常数。

得到的结果就是大O阶。

### 常数阶

O(1)

### 线性阶

O(n)

### 对数阶

O(logn)

### 平方阶

O(n2)

## 常见的时间复杂度

|  |  |  |
| --- | --- | --- |
| 执行次数函数 | 阶 | 非正式术语 |
| 12 | O(1) | 常数阶 |
| 2n+3 | O(n) | 线性阶 |
| 3n2+2n+1 | O(n2) | 平方阶 |
| 5log2n+20 | O(logn) | 对数阶 |
| 2n+3nlog2n+19 | O(nlogn) | nlogn阶 |
| 6n3+2n2+3n+4 | O(n3) | 立方阶 |
| 2n | O(2n) | 指数阶 |

O(1)< O(logn)< O(n)< O(nlogn)< O(n2)< O(n3)< O(2n)< O(n!)< O(nn)

## 最坏情况与平均情况

最坏情况运行时间是一种保证，那就是运行时间将不会再坏了。在应用中，这是一种最重要的需求，通常，除非特别指定，我们提到的运行时间都是最坏情况的运行时间。

平均运行时间是所有情况中最有意义的，因为它是期望的运行时间。

## 算法空间复杂度

算法的空间复杂度通过计算算法所需的存储空间实现，算法空间复杂度的计算公式记作：S(n)=O(f(n))，其中，n为问题的规模，f(n)为语句关于n所占存储空间的函数。

# 线性表

定义：零个或多个数据元素的有限序列。

## 线性表的顺序存储结构

定义：用一段地址连续的存储单元依次存储线性表的数据元素。

### 数据长度与线性表长度区别

数组的长度是存放线性表的存储空间的长度，存储分配后这个量一般是不变的。

线性表的长度是线性表中数据元素的个数，随着线性表插入和删除操作的进行，这个量是变化的。

在任意时刻，线性表的长度应该小于等于数组的长度。

### 地址计算方法

存储器中的每个存储单元都有自己的编号，这个编号称为地址。

## 顺序存储结构的插入与删除

### 获得元素操作

将线性表L中的第i个位置元素值返回，只要i的数值在数组下标范围内，就是把数组第i-1下标的值返回即可。

### 插入操作

插入算法思路：

如果插入位置不合理，抛出异常；

如果线性表长度大于等于数组长度，则抛出异常或动态增加容量；

从最后一个元素开始向前遍历到第i个位置，分别将它们都向后移动一个位置；

将要插入元素填入位置i处；

表长加1。

### 删除操作

删除算法思路：

如果删除位置不合理，抛出异常；

取出删除元素；

从删除元素位置开始遍历到最后一个元素位置，分别将它们都向前移动一个位置；

表长减1。

### 线性表顺序存储结构的优缺点

优点：

无须为表示表中元素之间的逻辑关系而增加额外的存储空间。

可以快速地存取表中任一位置的元素。

缺点：

插入和删除操作需要移动大量元素。

当线性表长度变化较大时，难以确定存储空间的容量。

造成存储空间的“碎片”。

## 线性表的链式存储结构

### 线性表链式存储结构定义

为了表示每个数据元素ai与其直接后继数据元素ai+1之间的逻辑关系，对数据元素ai来说，除了存储其本身的信息之外，还需存储一个指示其直接后继的信息（即直接后继的存储位置）。我们把存储数据元素信息的域称为数据域，把存储直接后继位置的域称为指针域。指针域中存储的信息称做指针或链。这两部分信息组成数据元素ai的存储映像，称为结点。

n个结点（ai的存储映像）链结成一个链表，即为线性表（a1,a2,...,an）的链式存储结构，因为此链表的每个结点中只包含一个指针域，所以叫做单链表。

链表中第一个结点的存储位置叫做头指针。

线性链表的最后一个结点指针为“空”（通常用NULL或“^”符号表示）。

有时为了更加方便地对链表进行操作，会在单链表的第一个结点前附设一个结点，称为头结点。

头结点的数据域可以不存储任何信息。

### 头指针与头结点的异同

头指针：

头指针是指链表指向第一个结点的指针，若链表有头结点，则是指向头结点的指针。

头指针具有标识作用，所以常用头指针冠以链表的名字。

无论链表是否为空，头指针均不为空。

头指针是链表的必要元素。

头结点：

头结点是为了操作的统一和方便而设立的，放在第一元素的结点之前，其数据域一般无意义（也可存放链表的长度）。

有了头结点，对在第一元素结点前插入结点和删除第一结点，其操作与其它结点的操作就统一了。

头结点不一定是链表必须要素。

### 线性表链式存储结构描述

结点由存放数据元素的数据域存放后继结点地址的指针域组成。

## 单链表的读取

获得链表第i个数据的算法思路：

声明一个结点p指向链表第一个结点，初始化j从1开始。

当j<i时，就遍历链表，让p的指针向后移动，不断指向下一结点，j累加1。

若到链表末尾p为空，则说明第i个元素不存在。

否则查找成功，返回结点p的数据。

## 单链表的插入与删除

### 单链表的插入

单链表第i个数据插入结点的算法思路：

声明一结点p指向链表第一个结点，初始化j从1开始。

当j<i时，就遍历链表，让p的指针向后移动，不断指向下一结点，j累加1。

若到链表末尾p为空，则说明第i个元素不存在。

否则查找成功，在系统中生成一个空结点s。

将数据元素e赋值给s->data。

单链表的插入标准语句s->next=p->next;p->next=s。

返回成功。

### 单链表的删除

单链表第i个数据删除结点的算法思路：

声明一结点p指向链表第一个结点，初始化j从1开始。

当j<i时，就遍历链表，让p的指针向后移动，不断指向下一结点，j累加1。

若到链表末尾p为空，则说明第i个元素不存在。

否则查找成功，将欲删除的结点p->next赋值给q。

单链表的删除标准语句p->next＝q->next。

将q结点中的数据赋值给e，作为返回。

释放q结点。

返回成功。

## 单链表的整表创建

单链表整表创建的算法思路：

声明一结点p和计数器变量i。

初始化一空链表L。

让L的头结点的指针指向NULL，即建立一个带头结点的单链表。

循环：

生成一新结点赋值给p。

随机生成一数据赋值给p的数据域p->data。

将p插入到头结点与前一新结点之间。

## 单链表的整表删除

单链表整表删除算法思路：

声明一结点p和q。

将第一个结点赋值给p。

循环：

将下一结点赋值给q。

释放p。

将q赋值给p。

## 单链表结构与顺序存储结构优缺点

存储分配方式：

顺序存储结构用一段连续的存储单元依次存储线性表的数据元素。

单链表采用链式存储结构，用一组任意的存储单元存放线性表的元素。

时间性能：

查找

顺序存储结构O(1)

单链表O(n)

插入和删除

顺序存储结构需要平均移动表长一半的元素，时间为O(n)

单链表在线出某位置的指针后，插入和删除时间仅为O(1)

空间性能

顺序存储结构需要预分配存储空间，分大了浪费，分小了易发生上溢。

单链表不需要分配存储空间，只要有就可以分配，元素个数也不受限制。

## 静态链表

用数组描述的链表叫做静态链表。

### 静态链表优缺点

优点

在插入和删除操作时，只需要修改游标，不需要移动元素，从而改进了在顺序存储结构中的插入和删除操作需要移动大量元素的缺点。

缺点

没有解决连续存储分配带来的表长难以确定的问题。

失去了顺序存储结构随机存取的特性。

## 循环链表

将单链表中终端结点的指针端由空指针改为指向头结点，就使整个单链表形成一个环，这种头尾相接的单链表称为单循环链表，简称循环链表。

## 双向链表

双向链表是在单链表的每个结点中，再设置一个指向其前驱结点的指针域。

# 栈与队列

## 栈的定义

定义：栈是限定仅在表尾进行插入和删除操作的线性表。

允许插入和删除的一端称为栈顶（top），另一端称为栈底（bottom）。

不含任何数据元素的栈称为空栈。

栈又称为后进先出（Last In First Out）的线性表，简称LIFO结构。

栈的插入操作，叫做进栈，也称压栈、入栈。

栈的删除操作，叫做出栈，也称弹栈。

## 栈的应用－－递归

一个直接调用自己或通过一系列的调用语句间接地调用自己的函数，称做递归函数。

每个递归定义必须至少有一个条件，满足时递归不再进行，即不再引用自身而是返回值退出。

## 队列的定义

定义：队列是只允许在一端进行插入操作，而在另一端进行删除操作的线性表。

队列是一种先进先出（First In First Out）的线性表，简称FIFO。

允许插入的一端称为队尾，允许删除的一端称为队头。

## 循环队列

定义：把队列头尾相接的顺序存储结构称为循环队列。

# 串

定义：是由零个或多个字符串组成的有限序列，又名叫字符串。

## 串的存储结构

### 串的顺序存储结构

### 串的链式存储结构

# 树

定义：树是n（n>=0）个结点的有限集。n=0时称为空树。在任意一棵非空树中：（1）有且仅有一个特定的称为根（Root）的结点；（2）当n>1时，其余结点可分为m（m>0）个互不相交的有限集T1、T2、……、Tm，其中每一个集合本身又是一棵树，并且称为根的子树（SubTree）。

结点拥有的子树数称为结点的度。

度为0的结点称为叶结点或终端结点。

度不为0的结点称为非终端结点或分支结点。

树的度是树内各结点的度的最大值。

结点的子树的根称为该结点的孩子，该结点称为孩子的双亲。

同一个双亲的孩子之间互称兄弟。

结点的层次：从根开始定义起，根为第一层，根的孩子为第二层。若某结点在第l层，则其子树的根就在l+1层。其双亲在同一层的结点互为堂兄弟。

树中结点的最大层次称为树的深度或高度。

如果将树中结点的各子树看成从左至右是有次序的，不能互换的，则称该树为有序树，否则称为无序树。

## 二叉树

定义：二叉树（Binary Tree）是n（n>=0）个结点的有限集合，该集合或者为空集（称为空二叉树），或者由一个根结点和两棵互不相交的、分别称为根结点的左子树和右子树的二叉树组成。

### 二叉树的基本形态

空二叉树。

只有一个根结点。

根结点只有左子树。

根结点只有右子树。

根结点既有左子树又有右子树。

### 特殊二叉树

斜树

所有的结点都只有左子树的二叉树叫左斜树。

所有结点都是只有右子树的二叉树叫右斜树。

这两者统称为斜树。

满二叉树

在一棵二叉树中，如果所有分支结点都存在左子树和右子树，并且所有叶子都在同一层上，这样的二叉树称为满二叉树。

完全二叉树

对一棵具有n个结点的二叉树按层序编号，如果编号为i（1=<i<=n）的结点与同样深度的满二叉树中编号为i的结点在二叉树中位置完全相同，则这棵二叉树称为完全二叉树。

### 二叉树的性质

性质一：在二叉树的第i层上至多有2i-1个结点（i>=1）

性质二：深度为k的二叉树至多有2k-1个结点（i>=1）

性质三：对任何一棵二叉树T，如果其终端结点数为n0，度为2的结点数为n2，则n0=n2+1。

性质四：具有n个结点的完全二叉树的深度为[log2n]+1（[x]表示不大于x的最大整数）。

性质五：如果对一棵树有n个结点的完全二叉树（其深度为[log2n]+1）的结点按层序编号（从第1层到第[log2n]+1层，每层从左到右），对任一结点i（1=<i<=n）有：

如果i=1，则结点i是二叉树的根，无双亲；如果i>1，则其双亲是结点[i/2]。

如果2i>n，则结点i无左孩子（结点i为叶子结点）；否则其左孩子是结点2i。

如果2i+1>n，则结点i无右孩子；否则其右孩子是结点2i+1。

### 二叉树的遍历

定义：二叉树的遍历是指从根结点出发，按照某种次序依次访问二叉树中所有结点，使得每个结点被访问一次且仅被访问一次。

遍历方法一：前序遍历

若二叉树为空，则空操作返回，否则先访问根结点，然后前序遍历左子树，再前序遍历右子树。

遍历方法二：中序遍历

若二叉树为空，则空操作返回，否则从根结点开始（注意并不是先访问根结点），中序遍历根结点的左子树，然后是访问根结点，最后中序遍历右子树。

遍历方法三：后序遍历

若二叉树为空，则空操作返回，否则从左到右先叶子后结点的方式遍历访问左右子树，最后访问根结点。

遍历方法四：层序遍历

若二叉树为空，则空操作返回，否则从树的第一层，也就是根结点开始访问，从上而下逐层遍历，在同一层中，按从左到右的顺序对结点逐个访问。

# 图

定义：图是由顶点的有穷非空集合和顶点之间边的集合组成，通常表示为：G(V,E)，其中，G表示一个图，V是图G中顶点的集合，E是图G中边的集合。

无向表：若顶点vi到vj之间的边没有方向，则称这条边为无向边，用无序偶对(vi,vj)表示。

无向图：如果图中任意两个顶点之间的边都是无向边，则称该图为无向图。

有向边：若从顶点vi到vj的这有方向，则称这条边为有向边，也称为弧（Arc）。用有序偶<vi,vj>来表示，vi称为弧尾，vj称为弧头。

有向图：如果图中任意两个顶点之间的边都是有向边，则称该图为有向图。

连接顶点A到D的有向边就是弧，A是弧尾，D是弧头，<A,D>表示弧，不能写成<D,A>。

简单图：在图中，若不存在顶点到其自身的边，且同一条边不重复出现。

无向完全图：在无向图中，如果任意两个顶点之间都存在边。

有向完全图：在有向图中，如果任意两个顶点之间都存在方向互为相反的两条弧。

稀疏图、稠密图：有很少条边或弧的图称为稀疏图，反之称为稠密图。

有些图的边或弧具有它相关的数字，这种与图的边或弧相关的数叫做权。

这种带权的图通常称为网。

# 查找

定义：查找就是根据给定的某个值，在查找表中确定一个其关键字等于给定值的数据元素（或记录）。