**1.3数据结构起源**

数据结构是一门研究**非数值计算**的程序设计问题中的操作对象，以及他们之间关系和操作等相关的问题学科。

**1．4．1数据**

数据是描述事物的符号，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。

**1.4.2数据元素**

人类中，人是数据元素，畜类中，牛、马、羊为数据元素

**1.4.3数据项**

人为上级元素，眼、耳、鼻、嘴、手为数据项，数据项为不可分割的最小单位

**1.4.4数据对象**

是性质相同元素的集合，是数据的子集

**1.4.5数据结构**

简单理解就是关系，比如分子结构，就是说组成分子的原子之间的排列方式。

不同数据元素之间不是独立的，二手存在特定的关系，我们将这些关系称为结构。

**1．5．1逻辑结构**

逻辑结构：指数据对象中数据元素之间的相互关系

1. 集合结构

集合机构中的元素除了同属一个集合外，他们之间没有其他关系，各各平等

2.线性结构

结构中的数据元素之间是一对一的关系

3.树形结构

树形结构中的元素之间存在一种一对多的层次关系

4图形结构

多对多关系

逻辑结构针对具体的问题，为了解决某个问题在对问题理解的基础上，选择一个合适的数据结构表示数据元素之间的逻辑关系。

**1．5.2物理结构**

是指数据的逻辑结构在计算机中存储的形式

1. 顺序存储结构

是吧数据存放在联系的存储单元里，其数据间的逻辑关系和物理关系是一致的。

1. 链式存储结构

是吧数据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

逻辑结构是面向问题的，而物理结构面向计算机的

**1.6.1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。

**2.4算法定义**

算法是解决特定问题求解步骤的描述

**2.5算法的特性**

2.5.1输入或输出

有0个或者多个输入

至少有一个或者多个输出

2.5.2有穷性

执行完成有限步骤后不会出现无限循环

2.5.3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。

2．5.4可行性

算法的每一步一都必须是可行的

**2．6算法设计的要求**

2.6.1正确性

2.6.2可读性

2.6.3健壮性

当输入数据不合法时也能做出相关处理，而不是产生异常或莫名其妙的结果

2.6.4时间效率高和存储量低

**2.7算法效率的度量方法**

2.7．1事后统计法

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

缺陷：1.必须事先编程，花费大量时间精力

2.时间的比较依赖运行时环境移速，有时掩盖算法本事的优劣。

3.算法设计测试数据难

2.7.2事前估算法

在计算机程序编制前，依据统计方法对算法进行估算

**2.8函数的渐进增长**

给定俩个函数f(n)和g(n)，如果存在一个整数N是的所有的n>N，f(n)总是g(n)大，那么，我们说f(n)的增长渐进快于g(n)。

判断一个算法的效率时，函数中的常熟和其他次要项常常可以忽略，而应该关注主项（最高阶项）的阶数

**2.9算法的时间复杂度**

2．9.1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，而进行分析T(n)随n的变化情况确定T(n)的数量级。算法的时间复杂度，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速率和f（n）的增长速率相同，称作算法的渐进时间的复杂度，简称为时间复杂度。其中f(n)是问题规模n的某个函数

这样用O（）体现算法时间复杂度的记法，我们称为大O记法。

显然我们的三个求和算法的时间复杂度分别为O(n),O(1),O(n^2)

2.9.3常数阶

O(1)

2.9．4线性阶

O(n)

分析算法的复杂度，分析循环结构的运行情况。

2.9.5对数阶

O(log n)看具体算法确定

2.9.6平方阶

O(n^2)

**2.10常见的时间复杂度**

从小到大的顺序依次是

O(1) < O(log n) < O(n) < O(n\*log n) < O(n^2) < O(n^3) < O(2^n) < O(n!) < O(n^n)

**2.11最坏情况与平均情况**

**2.12算法空间复杂度**

我们在写代码时，完全可以用空间来换取时间

算法的空间复杂度计算所需存储空间实现，公式S(n) = O(f(n))

**3线性表**

零个或多个数据元素的有限序列

3.5.4线性表顺序存储结构的优缺点

优点：1.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

2．可以快速的存放，读取表中任一位置的元素。

缺点：1.插入和删除操作需要移动大量元素

2.线性表长度变化较大，难以确定存储空间容量

3.造成空间的碎片

3.11单链表结构与顺序存在结构优缺点

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 平均移动一半时间为O(n)

单链表O(1)

空间性能：

顺序结构需要预分配，大了浪费，小了上溢

单链表：不需要分配，有就分配

**5串**

5.5串的存储结构

5.5.1串的顺序存储结构

用一组地址连续的存储单元来存储串中的字符序列

**6树**

6.2.1结点分类

结点下有的子树的数称为节点的度，度为0的节点是叶子节点，度不为0的节点称为非终端节点的分支节点，树的度是最大节点度的值

6.2.2节点的关系

节点的子树为该孩子，该节点称为双亲（父母同体），孩子之间称为兄弟

节点祖先（从根到这个节点经过的所有节点），节点子孙（从该节点到子孙节点经过的）

6.2.3树的其他相关概念

节点的层次，从根为第一层，树中最大层次称为树的深度或者高度

6.5.2特殊二叉树

1.斜树

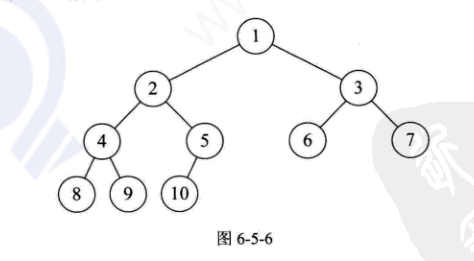
2.满二叉树

左右对称

3.完全二叉树

顺序编号都有节点

满二叉树一定是完全二叉树，完全二叉树不一定是满二叉树



----------

6.8.2二叉树的遍历方法

1.前序遍历

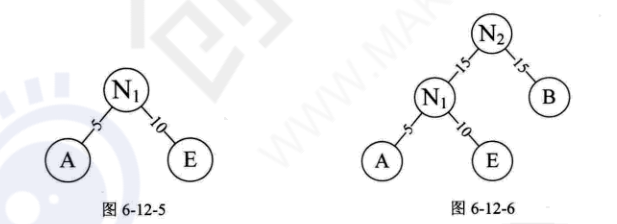
2.中序遍历

3.后序遍历

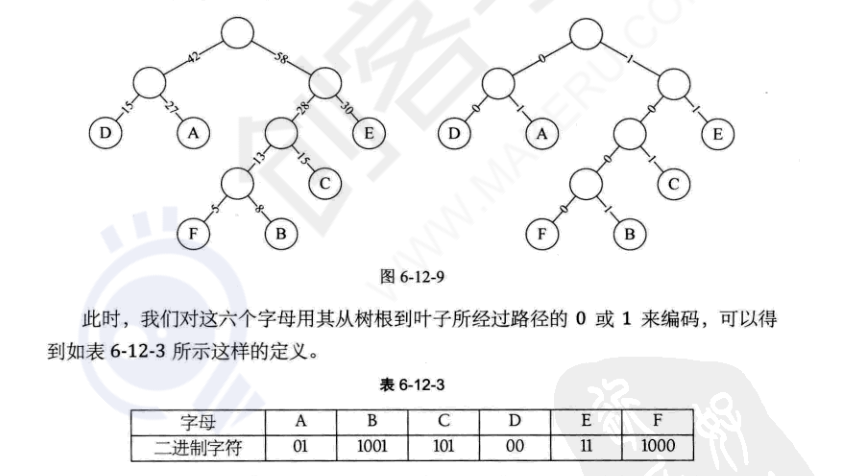
4.层次遍历

6.12赫夫曼树

权值小的相加=N1,N1放入再重复计算，小放左，大放右



赫夫曼编码



**7图**

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

线性表中数据元素称为元素，树中称为节点，图中称为顶点

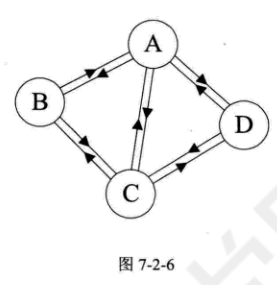
7.2.1各种图的定义

无向边：没有方向的边

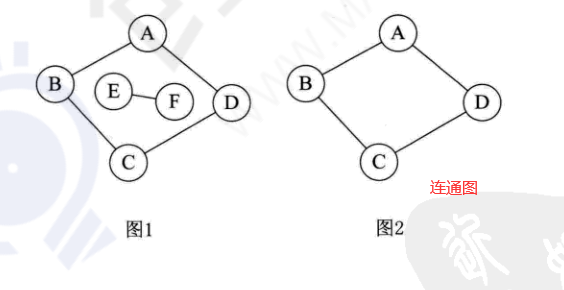
有向边：有方向 ，也称为弧

在无向图中，如果任意俩个顶点之间都存在边，称为无向完全图

有向完全图



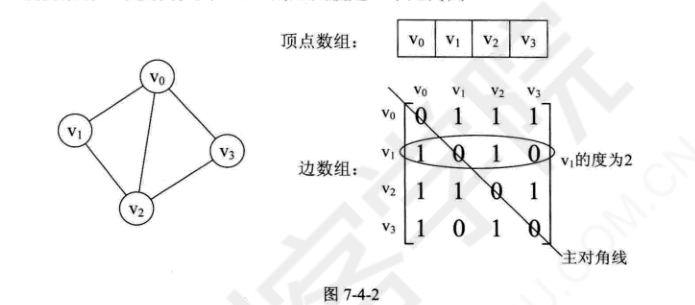
连通图



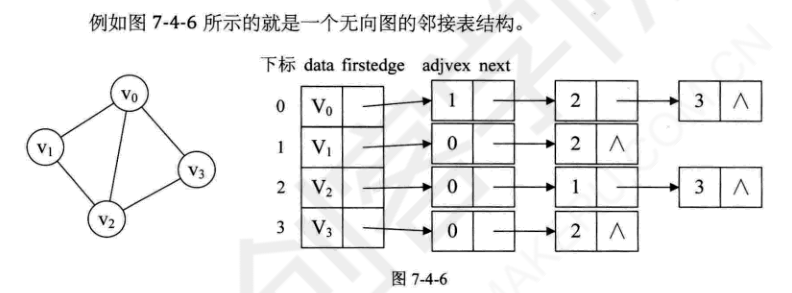
7.4图的存储结构

7.4.1邻接矩阵

一个一维数组，一个二维数组



7.4.2邻接表



7.4.3十字链表

7.4.4邻接多重表

7.4.5边集数组

**7.5图的遍历**

7.5.1深度优先遍历 DFS

7.5.2广度优先遍历BFS

**7.6最小生成树**

吧构造连通图的最小代价生成树称为最小生成树

7.6.1普里姆Prim算法

某顶点为起点逐步找哥哥顶点上最小权值的边

7.6.2克鲁斯卡尔(Kruskal)算法

针对边开展边数少时效率高

**7.7最短路径**

7.7.1迪卡斯特拉（Dijkstra）算法

7．7.2费洛伊德(Floyd)算法

**7.8拓扑排序**

用弧表示优先级关系，这样的有向图为顶点表示活动的网称为AOV网

7.8.2拓扑排序算法

选一个入度为0的顶点输出，然后删除此顶点，删除此顶点为尾的弧，重复此步骤，知道输出全部顶点，或者AOV网中不存在入度为0的顶点为止

**7.9关键路径**

拓扑排序主要是解决一个工程是否顺序进行的问题，但是有时还需要解决最短时间的问题

在一个表示工程的带权有向图中，用顶点表示时间，用有向边表示活动，用边上的权值表示活动的持续时间，这种有向图的边表示活动的网，我们称为AOE网

吧路径上活动持续时间之和称为路径长度，从源点到汇点有最大长度的路径叫做关键路径，在关键路径上的活动叫关键活动

**8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

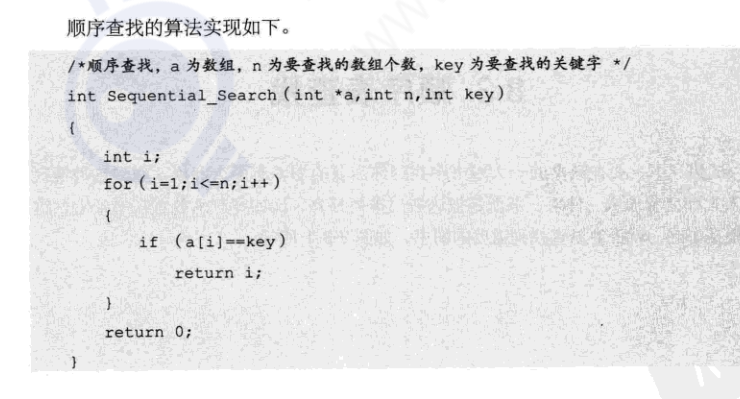
静态查找表：只作查找操作的查找表

动态查找表：在查找过程中同时插入查找表中不存在的数据元素，或者从查找表中删除已经存在的某个数据元素

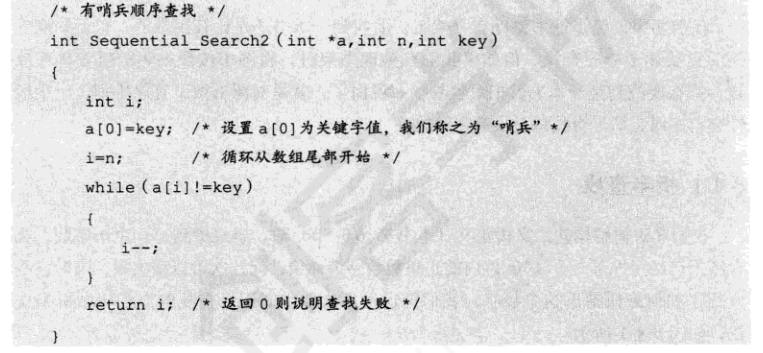
8．3顺序表查找

O(n)

顺序查找表优化



优化后，不让n与i做比较

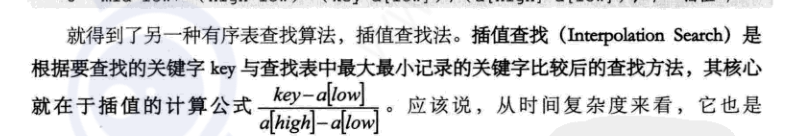


8.4有序表查找

8.4.1折半查找（二分查找）

关键码有序，线性表采用顺序存储

8.4．2插值查找



从时间复杂度来说也是O(log n)，但是对于表长较大，二关键字分布均匀的查找来说性能比二分查找要好的多

8.4.3斐波那契查找

时间复杂度O(log n)

8.5线性索引查找

8.5.1稠密索引

在线性索引中，讲数据集中的每个记录对应一个索引项

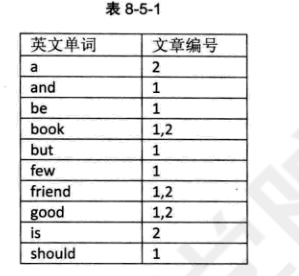
一定按照关键码有序的排列

8.5.2分块索引

吧数据集的记录分成了若干块

块内无序，块间有序

8.5.3倒排索引



**8.6二叉排序树**

**8.7平衡二叉树 AVL树**

是一种二叉排序树，每一个节点的左子树和右子树的高度差至多1

**8.8多路查找树（B树）**

其每一个节点的孩子可以多余2个，每一个节点处可以存储多个元素

8.8.1 2-3树

每一个节点都有俩个或者三个孩子

8.8.2 2-3-4树

8.8.3 B+树，特别适合带有范围的查找

**8.9散列表查找（哈希表）概述**

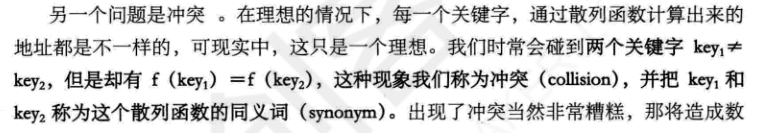
8.9.1散列表查找定义

散列技术是在记录的存储位置和它的关键字之间建立一个确定的对应关系f，使得每个关键字key对应一个存储位置f。

吧这种对应关系称为散列函数，又称哈希函数，采用散列技术将记录存储在一块连续的存储空间中，这块空间称为哈希表或者散列表

8.9.2散列表查找步骤

散列技术既是一种存储方法，也是一种查找方法



**8.10散列函数的构造方法**

1.计算简单

2．散列地址分布均匀

8.10.1直接定址法

8.10.2数字分析法

8.10.3平方取中法

8.10.4折叠法

8.10.5除留余数法

8.10.6随机数法

**8.11处理散列冲突的方法**

8.11.1开放定址法

8．11.2再散列函数法

8.11.3练地址法

8．11.4公共溢出区法

**9排序**

9．2.1排序的稳定性



**9.3冒泡排序**

时间复杂度O(n^2)

**9.4简单选择排序**

选最小的放最前面

时间复杂度O(n^2)

**9.5直接插入排序**

拿出来一个个插

时间复杂度O(n^2)

**9.6希尔排序**

除2间隔数，如果有交换回溯

时间复杂度O(n^(3/2))

**9.7堆排序**

时间复杂度O(n logn)

**9.8归并排序**

分组，比较合并

时间复杂度O(n logn)

**9.9快速排序**

选一个数，从俩遍吧比他小的放左边，比他大的放右边

分成几组后重复操作

最好时间复杂度O(n logn)

最坏时间复杂度O(n^2)