微型计算机实用大

第10篇 数据库

10.1 数据结构

10.1.1 线性数据结构

数据 是对客观实体(可见的或抽象的)的一种描述形式。通常使用的有数值数据、符号数据和图形数据等。在计算机科学中,凡是计算机能识别与处理的数字、字符、图形以及它们的汇集通称为数据。

信息 是客观世界向人(或机器)所提供的新的知识。它真正反映客观实体的本质。数据是载荷信息的一种物理符号。即同一种信息可用不同的数据形式表示,因而,数据的形式随着所利用的设备不同而不同,而信息是不受设备变化的影响。信息的量度也与数据不同,信息量的大小是用不肯定性来量度的。信息不同于数据,但二者的关系是不可分割的。所以在许多场合数据与信息的概念是通用的。

数据元素 是系统中数据的基本单位,可以是单个数字或字母,也可以是它们的组合,当一个数据元素是由几个独立的部分组成时,其中每一部分称为数据项。所谓数据项是在数据处理时不可再分割的最小单位。如图10.1-1所示登记卡为数据元素,其中姓名、年龄、性别、籍贯都是数据项。

姓名	年龄	性别	籍贯
赵明	23	男	山西

图10.1-1

数据类型 是把具有某些共同属性的数据个体,用一个共同的标志符来标识它们而组成的一个群体称为数据类型。如PASCAL语言中常用的整型数、实型数、布尔型数等都是实用的数据类型。对于每一种确定的数据类型都含有两个方面的内容:其一,确定了该类型数变量的取值范围。其二,隐含着该类型数据所能进行的操作运算。

数据名字 用来表示数据项的一个标识符称为数据名字。如上例中的姓名,年龄等都是数据名字。

数据映象 是指数据间的对应关系,它反映客观实体之间的联系,有简单的一对一(线性表)、复杂的一对多(如树结构)或多对多(网络)。

数据收集 是指利用某种方式或设备将时间上或空间上分散的数据个体集中起来的过程。

数据处理 是计算机应用的一个重要方面,包括企业管理、库存管理、财务管理、信息情报检索等。数据处理的特点是,存储数据所需要的存储空间远远大于操作数据的程序所占用的空间,因而,数据的结构、数据的存储、检索,分类和维护是数据处理中主要考虑的问题。

数据管理 是操作系统的一种主要功能,它包括对数据的组织,存储,检索,更新等操作。

数据保护 在系统中为避免数据丢失、失真和被破坏而采取的安全措施。

数据检索 是从文件、数据库或存储设备中查找和选取所需要的数据。

数据终端 是由计算机输入、输出设备和数据通信设备所组成的一种数据输入输出设备。用它可向计算机输入数据也可以接收从计算机输出的数据，具有与数据通信线路连接和通讯控制能力。有的还具有数据处理能力。现在有的用小型机、微型机充当终端，其功能就更强。

数据块 是在主存储器与输入输出设备之间作为一个数据单位来进行处理的数据段，它是由一组或几组按顺序连续排列的记录组成。

数据存取 是指数据在主存贮器与输入、输出设备之间的传送过程。

数据转换 为满足数据处理的需要，把数据从一种表示形式变换成另一种表示形式的过程，如将十进制变换成二进制、八进制、十六进制等。

数据误差 是指数据本身固有的偏差值。它不是因外界环境与条件的影响而产生的。

数据延迟 是由于数据自身的内部原因所引起的时间拖延现象。

数据评价 在数据处理时，对数据固有的含义、精确性、完整性、合理性、相关性以及其作用进行分析、研究给予较全面的评议与估价。

线性表 是含有$n(n\geqslant 0)$个元素的有限序列，当$n=0$时为空表，对于一个非空的线性表表示为$(a_1,a_2,\cdots,a_n)$$n$为表的长度，当$2\leqslant i\leqslant n-1$时，$a_{i-1}$为$a_i$的唯一直接前趋，$a_{i+1}$为$a_i$的唯一直接后继，其关系是一对一。其数据结构称为线性结构。○→○→○…→○

顺序存储 是线性结构的一种存储形式，将线性表中的元素按其原来顺序依次存储到内存中从某一已知地址开始的地址相连续的空间存储区中的过程。其主要特点：

①预知最大空间量。②每个元素占用相同的存储单元数。③逻辑相邻的元素物理位置相邻。④可进行随机存取。

存储单元 内存中用来存储一个数据元素，并具有一个地址号的存储空间。有的一个计算机字可作为一个存储单元，此时字地址便是存储单元地址。有时几个计算机字组合作为一个存储单元，此时组成存储单元的第一个字地址作为存储单元地址。

栈 是允许数据元素的输入(进栈)与删除(退栈)从同一端进行的结构。特点：后进先出。其处理的数据是线性结构。栈的构成：有栈名S，容量m，m是在任何时刻栈S中所能存储的最多元素个数；栈顶端：允许元素输入与删除的端称栈顶端，另一固定端为栈底端。栈顶指示器TOP：是指示当前栈顶元素的实际位置。TOP=0为空栈。见图10.1-2。

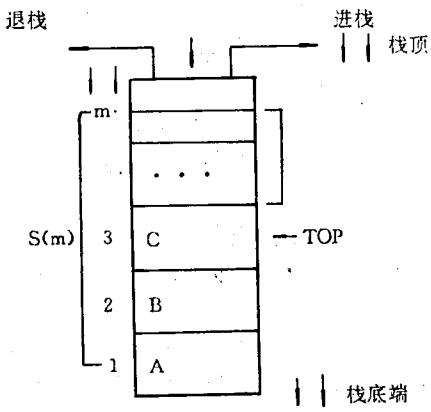


图 10.1-2

多栈结构 是由m个连续的空间单元分成$n(n\geqslant 3)$个容量不知且不等的栈，并能满足在任何时刻只要已存入

总元素个数少于 m 便可任意插入的一种实用结构。构成：$S(m)$，每个栈的栈顶 TOP，栈底 BOT 初值为：TOP(i)=BOT(i)=(i−1) * ⌊m/n⌋；1≤i≤n；⌊m/n⌋为设定的每个栈的容量。当第i个栈中已存元素个数小于⌊m/n⌋时，按单个栈的操作方式；当第i个栈中已存元素个数等于⌊m/n⌋时再动态给第i个栈分配所需空间。一般采用从第i个栈向右，(或向左)寻找空闲单元，若找到则将空闲单元分给第i个栈，若左右都找不到则溢出。图10.1-3是其构造图

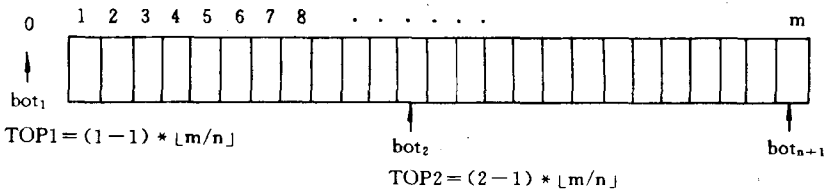


图 10.1-3

队列 是允许数据元素从一端插入而从另一端删除的一种结构。其特点是先进先出。构造：队列名，Q，容量 m，即$Q(m)$。允许元素插入的端称为尾部，允许元素删除的端称首部。尾部指针：r(*rear*)是指示刚刚进入队列元素的所在的位置；首部指针：f(*front*)是指示要从队列中删除的元素的位置。f，r 初值均为0。其结构见图10.1-4。

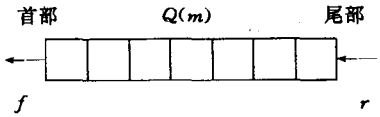


图 10.1-4

循环队列 是利用循环的方式按排使用有限空间单元的队列结构。$Q(m)$注意：这种队列中任何时刻所能存储的最多元素个数为 $m-1$，其中一个单元作为标识单元。

即 $(r+1) mod\ m=f$ 便标识队列满，$r=f$ 时队列为空。

进队列算法为：

```
begin
if(r+1)mod m=f then
write('overflow')
else 〔r=(r+1)mod m,Q(r)=×〕
```

双端队列 是允许数据元素可从任一端插入与删除的队列结构。其结构见图10.1-5。

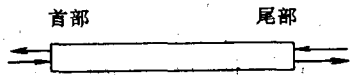


图 10.1-5

具有队列、循环队列、堆栈的功能。

输入受限双队列 允许数据元素从两端删除，而仅从一端插入的队列结构。结构见图10.1-6。

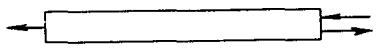


图 10.1-6

具有队列、堆栈的功能。

输出受限双队列 允许数据元素从两端插入，而仅从一端输出的队列结构。结构见图10.1-7。

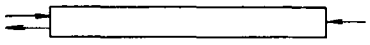
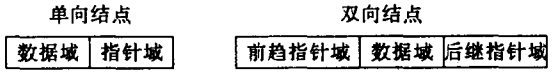


图 10.1-7

具有队列、循环队列、堆栈功能。

结点 是含有数据域、指针域的存储单元。可以是一个计算机字，也可以是由多个计算机字组成。当是一个计算机字时，字地址便是该结点地址；若是多个计算机字组成时，其中第一个计算机字地址为该结点地址。仅含有一个指针域的结点称单向结点，含有两个指针域的结点称双向结点。



链式存储结构 线性表中相邻的元素的逻辑关系是靠结点中的指针域来体现的存储形式：

例 线性表 $L=(a_1,a_2,\cdots,a_n)$

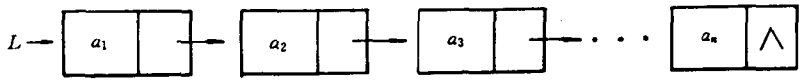


图 10.1-8

可用存储区(可用池) 是由用户所定义的可用结点所组成的可用结点区，所谓可用结点，是指该结点的数据域为空或存有无用的信息。可用存储区的功能向用户提供存储数据元素所需的结点。av 为第一个可用结点的地址指针，当 av=nil 时，无可用结点，不能再存储数据。相当于顺序 存储的上溢。见图 10.1-9。

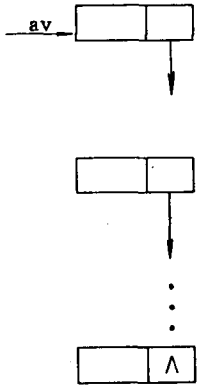


图 10.1-9

单向链表 是由仅有一个指针域的结点所组成的链表称单向链表简称单链表。每个单链表都具有一个头结点，头结点结构与其他数据结点相同，只是其数据域中存的是不同于有用信息的标识符。一个单链表的尾部结点标志是该结点指针域的值为空。

空单链表为：

单链表上的插入运算 是将新的元素结点插到链表中合适的位置上的运算。一般都是插在满足条件的结点的前边。这种运算的关键是查找满足条件结点的前趋结点指针。

例如：将结点 S 插在第 i 个结点的前边，主要是要找到第 $i-1$ 个结点地址。

用类PASCAL 语言的算法描述为，。

```
Procedure insert－list
Begin
    P=L                              (保护头结点)
    J=0                              (计数量置初值)
    NEW(s)                           (得到一个新结点)
    S↑data=x                         (将元素值填入数据域)
    while p≠nil and j<i－1
        do〔p=p↑link j=j+1〕          (寻找第i－1 个结点)
if j<i－1 then
        〔s↑link=p↑link
        p↑lin↑ =s〕                  (新结点插入)
END
```

单链表上的删除 是将 L 为头结点指针的单链表中满足给定删除条件的结点删除掉。

例：删除单链表 L 中数据域为×的结点。

类PASCAL 语言描述：

```
Procedure delete－list
Begin
    P=l↑link                         (保护头结点指针)
    q=l                              (保护前趋结点指针)
    while(p↑data≠×)and(p↑link≠nil)do
    〔q=p,p=p↑link〕
    if p↑data=× then
    〔q↑link=p↑link;dispose(p)〕
END
```

循环单链表 是可以从任意一已知结点开始可查到表中所有结点的链表。其构成：是将单链表尾部结点指针域的值用头结点指针值代替。结构见图10.1-10。

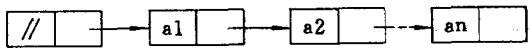


图 10.1-10

双向链表 是链表中的每一个结点都含有两个指针域的链表。结构见图 10.1-11。

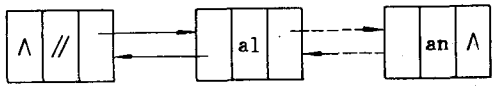


图 10.1-11

其特点：知道任一结点地址 p，则 p↑llink 为结点 p 的直接前趋结点，p↑rlink 为 p 的直接后继结点。

链式队列 用链表结构表示的队列，即利用链表的特点与操作满足队列中元素先进先出的特性。结构见图 10.1-12。

实质上是由前部删除尾部插入的单链表。

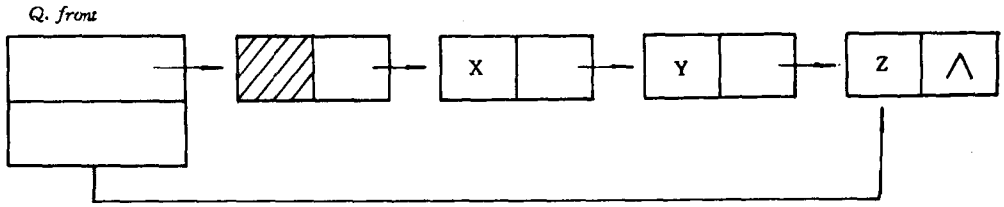


图 10.1-12 链式队列

10.1.2 非线性结构

树 是计算机算法中最重要的非线性结构。其关系是一对多的关系，一般定义为树是含有一个或多个结点的有限集合 T。在任一棵树中，(1)有且仅有一个结点称为根；(2)其余结点可分为 $m \geqslant 0$ 个不相交的子集合 $T1$，$T2$，…，Tm 其中每一个子集合本身又是一棵树，并称为根的子树。显然这是一个递归定义；递归是树结构的一个固有的特征。

树也是图论中一个重要的概念，在图论中不含有回路的连通图称为树。结构见图 10.1-3。

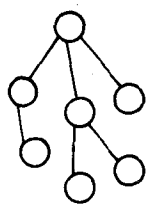


图 10.1-13 树

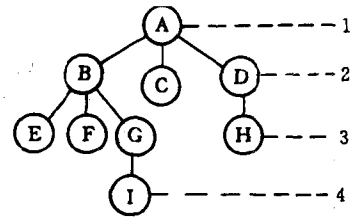


图 10.1-14 树

有序树与无序树 若树中结点的各子树从左至右是有次序的，则称该树为有序树，否则称无序树。在有序树中最左边的子树称为根的第一个孩子。最右边的子树为根的最后一个孩子。

树的度 树内各结点的度的最大值称为树的度。所谓结点的度，是指该结点的分枝数。

树的高度(深度) 是指树中结点的最大层数。根为一层，从根向下逐层增加。图 10.1-14 的树为 3 度树，其高度为 4。

森林 是 $m(m \geqslant 0)$ 棵互不相交的树的集合。一般讲构成森林的各树根结点为兄弟关系。

二叉树 是含有 $n(n \geqslant 0)$ 个结点的有限集合：当 $n=0$ 时为空，在任一棵非空二叉树中，结点的度都不大于 2。并具有下列性质：

(1)在二叉树的第 i 层上至多有 2^{i-1} 个结点。

(2)深度为 K 的二叉树至多有 2^K-1 个结点。

(3)对任何一棵二叉树 T，若其终端结点为 $n0$，度为 2 的结点数为 $n2$，则

$$n0=n2+1$$

(4)具有 N 个结点的完全二叉树的深度为 $\lfloor \log_2 N \rfloor +1$

满二叉树 深度为 k，且具有 2^k-1 个结点的二叉树为满二叉树。对一棵满二叉树的结点可进行连续编号即由根结点起自上而下同层从左至右得到一棵顺序编号二叉树。

完全二叉树 深度为 K，有 n 个结点的二叉树，当且仅当其每一个结点都与深度为 k 的满二叉树中编号从 1 至 n 的结点一一对应时称之为完全二叉树。

对一棵有 n 个结点的完全二叉树(其深度为 $\lceil \log_2 N \rceil +1$)其结点按层序编号(从第一层到第 $\lceil \log_2 N \rceil +1$

层每层从左到右)。则对任一结点$i(1\leqslant i\leqslant n)$,有

(1)若$i=1$,则结点i是二叉树的根,无双亲。若$i>1$,则其双亲结点的序号为$\lfloor i/2\rfloor$。

(2)若$2i\leqslant n$,则结点i的左孩子结点序号为$2i$,否则i无左孩子。

(3)若$2i+1\leqslant n$,则结点i的右孩子结点序号为$2i+1$否则结点i无右孩 子结点。

二叉树的遍历 按照一定的规则对树中每个结点访问且仅访问一次的过程称遍历二叉树。对一棵非空二叉树,一般较常用的有3种:

(1)先根序遍历:①访问根结点;②按先序遍历左子树;③按先序遍历右子树。

(2)中根序遍历:①按中根序遍历左子树;②访问根结点;③按中序遍历右子树。

(3)后根序遍历:①按后序遍历左子树;②按后序遍历右子树;③访问根结点。

中根序遍历算法:

```
procedure inorder(t)
begin
    p=t
    s=empty
    repeat
        while p≠nil do
        〔top=top+1 s(top)=p
        p=p↑Lchild〕
        if top≠0 then
        [p=s(top),top=top-1,
        Write(p4data);
        p=p4Rchild]
    until (top=0)and(p=nil)
end
```

线索二叉树 按某种方式遍历二叉树时,若当前结点p的左孩子域为空(即$p\uparrow lchild=nil$),则用该域存储p的遍历前趋结点地址(即在访问p之前刚刚访问的结点)。若p↑rchild=nil,则用该域存储p的遍历后继结点地址。这样构成的一棵记录遍历次序的树为线索二叉树。图10.1-15为按中序遍历的线索二叉树,虚线为遍历次序。

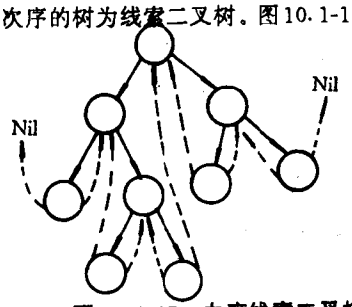


图10.1-15 中序线索二叉树

多叉树转换成二叉树 计算机本身的特点对二叉树的存储及运算非常方便,因而在一些应用中把反映多个因子的多叉树转换成二叉树的形式存于机内。其转换规则:

(1)将当前结点的最左边的孩子结点仍作为该结点的左孩子。

(2)由左边第二个孩子结点开始的所有兄弟结点,都依次变作其 左兄弟结点的右孩子。

(3)对每个结点重复止述两步 ,便得到一棵相应的二叉树。见图10.1-6。

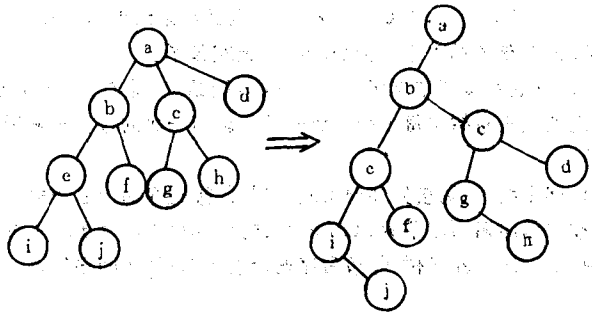


图 10.1-16 多叉树转换成二叉树

树中结点间的路径长度 从树中一个结点到另一个结点之间的分枝数目(即边的条数),称为两结点间路径长度。

树的路径长度 从树根到树中每一结点的路径长度之和称为树的路径长度。

结点的带权路径长度 是指该结点与树根结点之间的路径长度和该结点上权值的乘积。

树的带权路径长度 树中所有带权的叶子结点的路径长度之和称为树的带权路径长度。

记作 $$WPL=\sum_{i=1}^{n}W_iL_i$$

其中,W_i 为第 i 个叶子结点上的权值。

L_i 为第 i 个叶子结点与根结点间的路径长度

最优二叉树(哈夫曼树) 假设 n 个表值$\{w_1,w_2,\cdots\cdots,w_n\}$,试构造一棵有 n 个叶子结点的二叉树,每个叶子结点上的权为 w_i,则其中带权路径长度 WPL 最小的二叉树称为最优二叉树或哈夫曼树。

最优二叉树的算法

(1)将给定的 n 个权值$\{w_1,w_2,\cdots\cdots,w_n\}$构成 n 棵二叉树集合 $B=\{T_1,T_2,\cdots\cdots,T_n\}$,其中任一棵二叉树 $T_i(1\leqslant i\leqslant n)$仅有一个权为 W_i 的根结点,且左右子树为空。

(2)在 B 集合中选取两棵根结点权值最小的树作为左右子树构成一棵新二叉树,其左右子树根结点的权值之和作为新二叉树根结点权值。

(3)在 B 中删除这两棵子树,并将所得新二叉树加到 B 中。

(4)重复(2)、(3),直到集合 B 中仅含一棵树为止,这棵树便为哈夫曼树。

例:权值集合(7,5,2,4)构成具有 4 个叶子结点的哈夫曼树(见图 10.1-17)。

{ ⑦ ⑤ ② ④ }

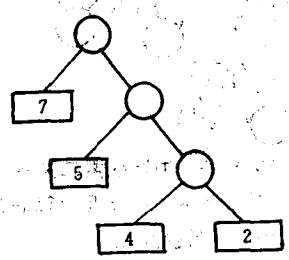


图 10.1-17 哈夫曼树

前缀编码 在快速远距离通信中,为使传送电文总长度最短。根据每个字符在电文中出现的次数不同,设计长度不等的编码。为此满足任一字符的编码都不是另一个字符的编码的前缀,这种编码称为前缀编码。

例 A 的编为 0,B 的为 10,C 的为 110,D 的为 111 便为前缀编码。

哈夫曼编码 实际上是一种二进制前缀编码：具体实现是以几种字符在电文中出现的频率作为权值，构造一棵哈夫曼树。并规定树中任一个非叶子结点的左分支表示“0”，右分支表示“1”，则从根到每一个叶子结点路径上分支组成的字符串便为该叶子结点字符的编码，见图10.1-18。

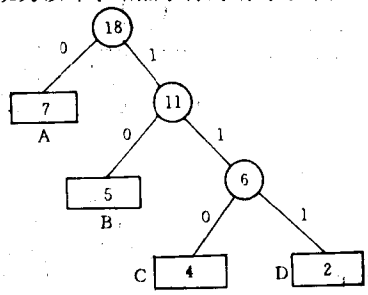


图10.1-18 哈夫曼编码

二叉排序树 二叉排序树可以是一棵空树，对于一棵非空的二叉排序树；若存在左子树，则左子树上所有结点的关键字值均小于它的根结点关键字值；若右子树不空，则右子树上所有结点的值均大于或等于它的根结点的值。

平衡二叉树 它或者是一棵空树，或者是具有下列性质的二叉树；它的左子树和右子树都是平衡二叉树，且左子树(TL)和右子树(TR)的最大深度之差的绝对值不超过1。$|TL-TR|\leqslant 1$。一般把左、右子树深度之差值称为结点的平衡因子。所以平衡二叉树上所有结点的平衡因子只可能是$-1,0,1$三者之一。只要二叉树上有一个结点的平衡因子的绝对值大于1，则说二叉树便不是平衡二叉树。

带重结点 平衡二叉树中平衡因子为$-1,1$的结点称为带重结点。又若$TL-TR=-1$，则称该结点为右重结点，若$TL-TR=1$，则称该结点为左重结点。

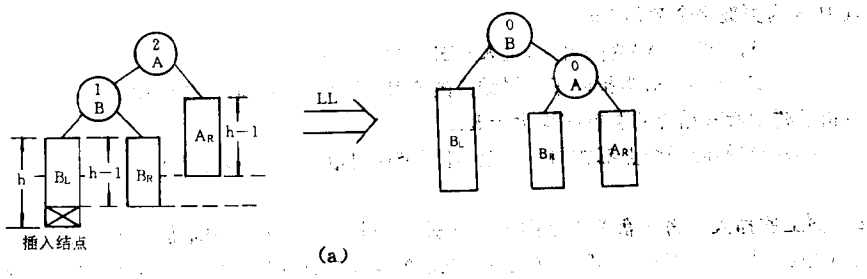
不平衡二叉树的形成及调整 当一棵平衡二叉树上的一个结点的平衡因子绝对值大于1时该树便为不平衡。产生的原因为：

(1)新结点插在左重结点a的左子树上，使a的平衡因子变为2，需进行一次顺时针旋转操作，如图10.1-19(a)，称LL型平衡调整。

(2)新结点插在左重结点a左孩子的右子树上，使a的平衡因子变为2失去平衡需进行两次旋操作(先逆时针，后顺时针)。称为LR型平衡调整，见图10.1-19(b)。

(3)新结点插在右重结点a的右子树上，使a的平衡因子变为-2，失去平衡，需进行一次逆时针旋转操作称RR型平衡调整，见图10.1-19(c)。

(4)新结点插在右重结点a的右孩子的左子树上，使a的平衡因子变为-2失去平衡，需进行两次旋转操作(先顺时针，后逆时针)称RL型调整，见图10.1-19(d)。



(a)

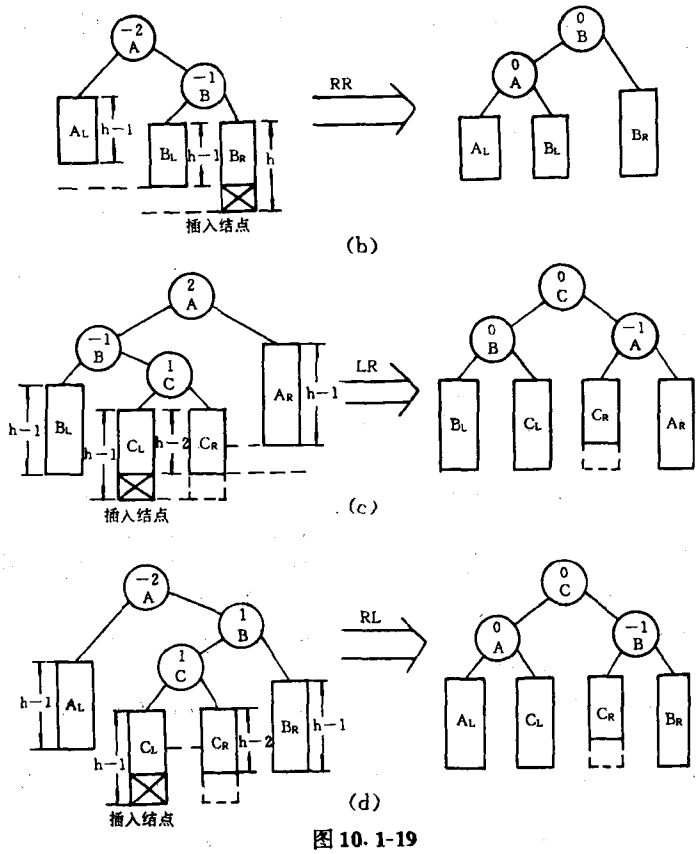


图 10.1-19

B－树 B－树是一棵平衡的多路查找树。一棵m阶的B－树，或是空树，或是满足下列性质的m叉树：

(1)树中每个结点最多有m棵子树；

(2)除非根结点为叶子结点，否则根结点至少有两棵子树；

(3)除根之外的所有非终端结点至少，有$\lceil m/2 \rceil$棵子树；

(4)所有非终端结点中都含有下列信息；

$(n, A_0, K_1, A_1, K_2, A_2, \cdots\cdots, K_n, A_n)$

其中：n为关键字个数$(n \leqslant m-1)$；

$K_i(1 \leqslant i \leqslant N)$为关键字，且$K_i < K_{i+1}(1 \leqslant i \leqslant n)$；

$A_i(0 \leqslant i \leqslant n)$为指向子树根结点的指针，且$A_i(0 \leqslant i \leqslant n)$所指示树中所有结点关键字均小于$K_{i+1}$。$A_n$所指子树中所有结点的关键字均大于$K_n$。

(5)所有的叶子结点都在同一层上，且不带任何信息(称失误结点)。

B－树上的插入 将一新关键字插入到一棵B－树上，不是在树中增加一个叶子结点(因B树结点中的关键字个数必须$\geqslant \lceil m/2 \rceil - 1$)，而是首先在最低层的某个非终端结点中添加这个关键字。若该结点的关键字个数不超过$m-1$，则完成插入，否则要进行结点的'分裂'。分裂过程如下：

设结点p中已有$m-1$个关键字，当插入一新关键字后，结点中含有信息：

$m, A_0, (K_1 A_1)(K_2 A_2) \cdots\cdots (K_m A_m)$

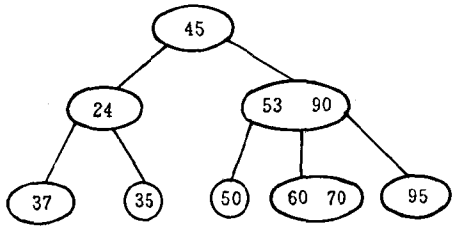


图 10.1-20 一棵3阶B－树

且其中 $K_i<K_{i+1}, 1\leqslant i\leqslant m$

此时可将 p 结点分成 p 与 p' 两个结点，其中 p 结点中含有信息为

$\lceil m/2\rceil-1, A_0, (K_1A_1)(K_2A_2)\cdots\cdots$

$(K\lceil m/2\rceil-1A\lceil m/2\rceil-1)$

p' 结点含有信息为

$m-\lceil m/2\rceil, A\lceil m/2\rceil, (K\lceil m/2\rceil+1, A\lceil m/2\rceil+1), \cdots\cdots, (K_mA_m)$

而将关键字 $K\lceil m/2\rceil$ 和指针 p' 一起插到 p 的双亲结点中。一棵3阶 B－树上的插入实例。见图 10.1-21。

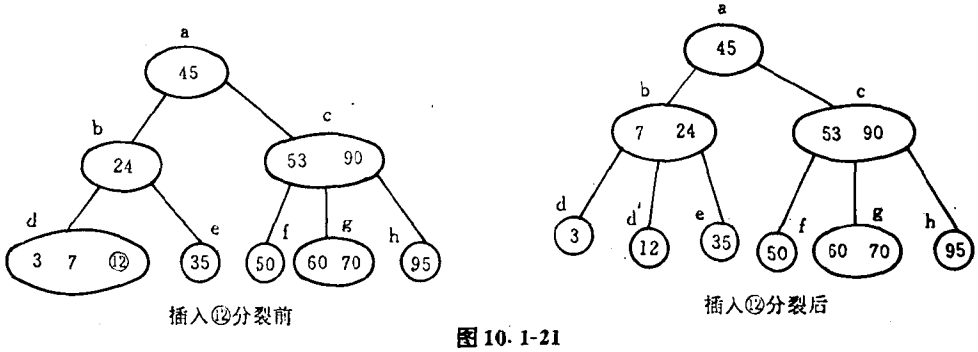


图 10.1-21

B－树上的删除 根据给定值找到该点键字所在结点，并从中删除掉它，若该结点为叶子结点，且其中的关键字数目仍不少于 $\lceil m/2\rceil$，则删除完成，否则要进行“合并”结点的操作，假若所删关键字为非终端结点中的 K_i，则可从指针 A_i 所指子树中的最小关键字 Y 代替 K，然后在相应的叶子结点中删去 Y。例如图 a 中删去45，可以 f 结点中的45替代50，然后删去 f 结点中50。下面讨论删除叶子结点中关键字的情形。有三种可解：

(1)被删关键字所在叶子结点中的关键字个数不小于 $\lceil m/2\rceil$，则只需从该结点中删除关钾键字 K_i 和相应的指针 A_i，树的其它部分不变。

(2)被删关键字所在叶子结点中的关键字个数等于 $\lceil m/2\rceil-1$，而与该结点相邻的右兄弟结点或左兄弟结点中的关键字数目大于 $\lceil m/2\rceil-1$，则需将其兄弟结点中的最小或最大的关键字上移至双亲结点中，而将双亲结点中小于或大于该上移关键字的关键字下移至被删关键字所在结点中。例如删去图 10.1-22(a)中50变为图 10.1-22(b)。

(3)被删关键字所在叶子结点和其相邻的兄弟结点中的关键字数目均等于 $\lceil m/2\rceil-1$。假设该结点有右兄弟，且其右兄弟结点地址由双亲结点中指针 A_i 所指，则在删除关键字之后，它所在结点中剩余的关键字和指针加上双亲结点中的关键字 K_i 一起合并到 A_i 所指兄弟结点中(若没有右兄弟，则合并到左兄弟结点中)如图(b)中删去53，则应删去 f 结点，并将双亲结点 e 中的61合并到右兄弟结点 g 中，如图(c)所示，如果因此使双亲结点中的关键字个数目小于 $\lceil m/2\rceil-1$ 则依次类推。如在(c)图中删去37后，双亲结点 b 中剩余信息(“指针”:)应和其双亲结点 a 中关键字45一起合并到右兄弟结点 c 中。如图(a)所示。

B+树 这是一棵应文件系统所需而产生的B－树的变型树，一棵 m 阶的B+树和 m 阶B－树区别在于：

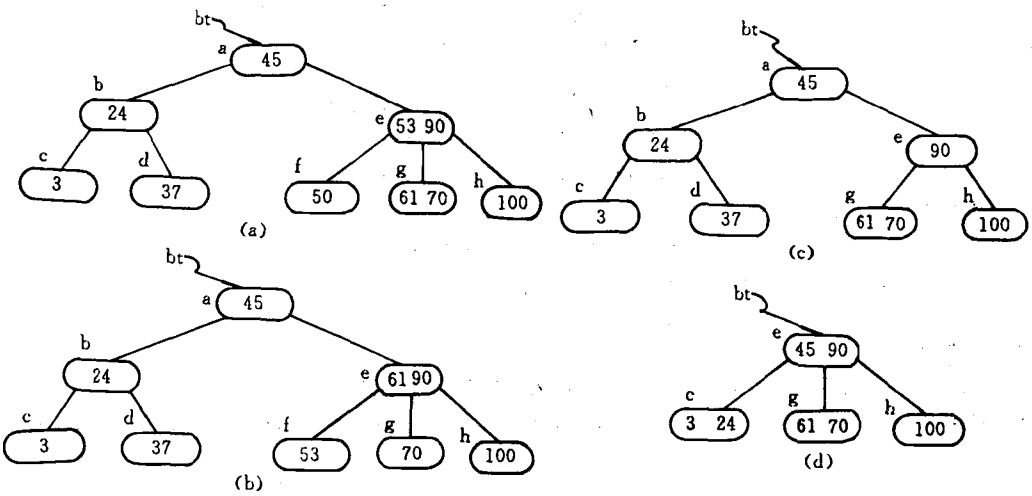


图 10.1-22

(1)有 n 棵子树的结点中含有 n 个关键字。

(2)所有的叶子结点中包含了全部关键字的信息及指向相应记录的指针,且叶子结 点本身依关键字的大小自小而大顺序链接。

(3)所有非终端结点可以看成是索引部分,结点中仅含有其子树(根结点)中的最 大(或最小)关键字。

在B+树上通常有两个头指针,一个指向根结点,另一个指向关键字最小的叶子结点。因此对B+树进行两种查找运算,一种是从最小关键字起顺序查找,另一种是从根结点开始,进行随机查找。图 10.1-23 为一3阶B+树。

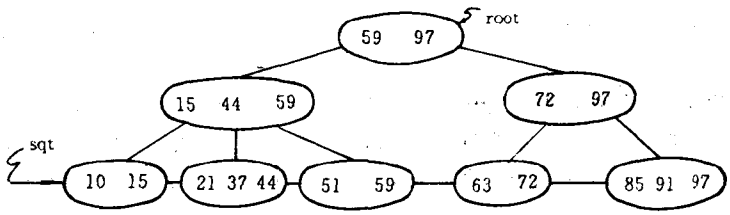


图 10.1-23

B+树上的插入 B+树上的插入与B-树类似,也只能在叶子结点上进行,当结点中的关键字个数大于 m 时,要分裂成两个结点,它们所含关键字的个数分别为$\lceil(m+1)/2\rceil$和$\lceil(m+1)/2\rceil$,并且,它们的双亲结点中应同时包含这两个结点的最大关键字。

B+树上的删除 B+树上的删除仅在叶子结点进行,当叶子结点中的最大关键字被删除时,其在非终端结

点中的值可以作为一个“分界关键字”存在，若因删除而依结点中关键字的个数少于$m/2$时，其和兄弟结点的合并过程与B树类似。

键树 它是一棵度≥2的树，树中每个结点不是包含一个或几个关键字，而是仅含有组成关键字的符号。例如，若关键字是由数字组成的，则结点中只包含其中一位数字，(故又称键树为数字查找树)，若关键字是单词，则结点中仅含一个字母字符。

例如，有关键字集合{235,230,318,496,2539,3852,4126}，其键树中每个结点的最大度d与关键字的“基”有关，若关键字是单词，则$d=27$，若关键字是数值，则$d=11$。键树的深度则取决于关键字中字符或数位的个数。见图10.1-24。

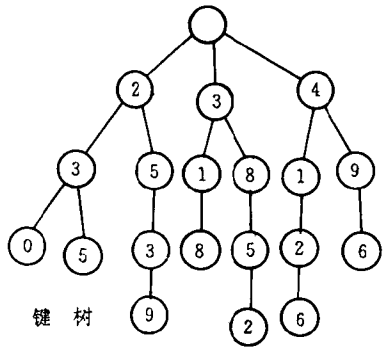


图10.1-24

图 在图论中，图是由若干个顶点(客观实体)和一些连接各顶点的边(关系)所组成的。表示为$G=(V,E)$。式中，V——顶点的有限非空集合。

E——边的有限集合。

无向图 若图G中的每一条边都没有方向，则称其为无向图，表示为$G=(V,E)$。E为无向边的集合：其中任一元素$e_i=(V_i,V_j)=(V_j,V_i)$为无向边。$(V_i,V_j\in V,e_i\in E)$，该无向边e_i代表的是对称关系。

有向图 若图G中每条边都有一定的方向则称其为有向图，表示为$G=(V,E)$，E为有向边的集合。其中e_i为任一元素，$e_i=<V_i,V_j>$为有向边，$(V_i,V_j\in V,e_i\in E)$，该有向边e_i代表非对称关系。

子图 设存在图G与G'，若$G'=(V',E')V'\subseteq V,E'\subseteq E$则称$G'$为$G$的子图。即由图$G$中的部分点与部分边组成的图$G'$为$G$的子图。

连接图 人工智能反向规则演译系统中的一种生成子结点的连接图法。在这种规则中，某结点生成子结点的过程是，将某结点与事先给定的规则相比较，如能匹配，则必规则生成子结点。依此类推，直到各个端结点都不能产生子结点或者达到目标为止。

完全图(稠密图)

(1)在具有n个顶点的无向图中，若每一对顶点间都有一条边相连，则称其为完全无向图，即有$\frac{n(n-1)}{2}$条边。

(2)在具有n个顶点的有向图中，若每一对顶点向彼此都有一条有向边相连，则称为完全有向图。即共有$n(n-1)$条有向边。

例 含有4个顶点的完全无向图为图10.1-25(a)，而完全有向图为图10.1-25(b)

连通图 对于一个无向图G，如果图中任意两个顶点$V_i,V_j\in V$，V_i和V_j都是连通的，则称G为连通图。

连通分量 无向图G的一个极大的连通子图称为G的连通分量。

强连通图 对于一个有向图G，若每一对顶点间都彼此存在路径则称G为强连通图。

强连通分量 有向图G的一个极大的强连通子图称其为G的强连通分量。由此可推出有向图中的每一个顶点都必须处在一个强连通分量之中，也只能存在于一个强连通之中。

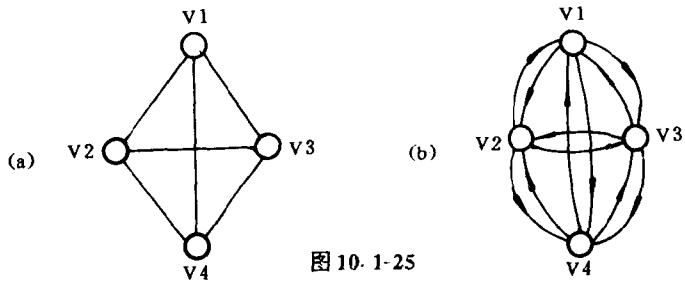


图10.1-25

图的深度优先搜索 设定图中每个顶点都没有被处理，首先选任一顶点作为V_0，处理V_0，然后依次从V_0的未被处理的邻接点出发深度优先遍历图，直至图中所有和V_0有路径相通的顶点都被处理过。若此时图中尚有顶点未被处理，则在下一条顶点中另选一个作为始点重复上述过程，直至图中全部顶点都被处理为止。如图10.1-26所示。

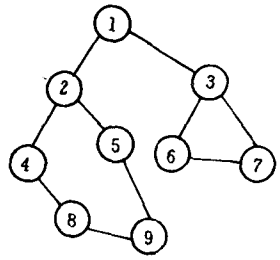


图10.1-26

例：

深度优先搜索顺序为：$V_1V_2V_4V_8V_9V_5V_3V_6V_7$。

广度优先搜索 设从未被处理的顶0点V_0出发，处理完V_0后依次处理V_0的各个未曾处理过的邻接顶点，然后分别从这些邻接点出发作广度优先遍历图，直到图中所有被处理过的邻接点都被处理过。若此时图中尚有顶点未被处理，则另选图中一个未曾处理的顶点作始点，重复上述过程，直至图中全部顶点都被处理过为止。图从广度优先搜索顺序为：

$V_1V_2V_3V_4V_5V_6V_7V_8V_9$

二部图 对于无向图$G=(V,E)$，如果顶点集V可分割为两个互不相交的子集，并且图中每条边相关联的两个顶点都分属两个不同的子集，则称图G为二部图。例如，商店与商品、学生与课程等都可用二部图描述。二部图是数据库系统中的重要数据结构。

图匹配 对于图$G=(V,E)$，若边集E的一个子集M中任意两条边都与图中同一个顶点相关联，则称M是图的一个匹配，选择这样的边的最大子集称为图的最大匹配问题。

完全匹配 如果在一个匹配中，每个顶点都和图中某条边相关联，则称此匹配为完全匹配，任何完全匹配都是一个最大匹配。求最大匹配的方法是，先求出全部匹配，然后从中选出含边数最多的一个即为最大匹配。

10.1.3 查找

查找 是根据给定的值按照某种方式寻找与给定值相匹配的元素的过程。

顺序查找 是最简单的一种查找方式，过程是从表的任一端开始，将表中元素逐个与给定值相比较，若相等查找成功，若超界，则查找失败。这种方式对被顺序存储与链式存储 的元素都适用。平均查找长度为$\frac{n+1}{2}$。减少时间的方式有两种：一是按元素的查找概率排序；二是增加一个辅助空间减少超界的判定次数。

折半查找 是查找速度最快的一种方式。要求被查表是按关键字递增(或递减)排序的且顺序存储的。查找过

程根据表的上下限计算出表中间位置，mid 将给定值 K 与 中间 K mid 相比较，$\left[\frac{Low+hig}{2}\right]=mid$

若 $K=K1mid$，则查找成功。

若 $K>K1mid$，则下次在后半表折半查找，$Low=mid+1$。

若 $K<K1mid$，则下次在前半表折半查找，$hig=mid-1$。

n

时间量级为 $O(\log n)$。

分块查找(索引顺序查找) 把要查文件分成若干块，每块建立一个索引项，一个索引项含有两个域，其一是关键字域：存放该块中最大关键字；其二是指针域：存放该块第一个记录地址，所有块的索引项构成索引表。索引表中关键字有序。每块中记录可以是有序的，也可以是无序的，但块与块之间一定有序，即后一块中的最小关键字一定要大于前一块中的最大关键字，查找过程：根据给定值 K 首先在索引表中利用折半查找方式确定 K 所在的块，然后在块中采取顺序方式查找。其时间量级界于顺序查找与折半查找之间。见图10.1-27。

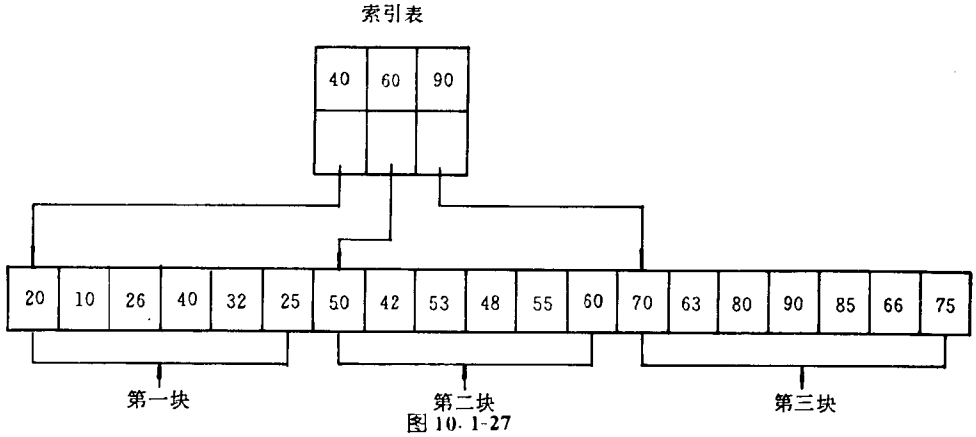


图 10.1-27

散列函数 散列函数定义为一个映射 $H:K\rightarrow A$，其中 K 是关键字集合，而 A 为地址连续的空间区 $A:(C+1,C+2,\cdots,C+m)$，H：作用把集合 K 中的每个关键字转换成 A 区间的一个地址号，而 K 代表的记录便存在所转换成的地址单元形成散列表。

散列查找 是根据给定的关键字利用构造散列表时采用的同一个散列函数，便可得到其关键字记录的地址。因而使查找时间与被查记录所在表记录个数 n 无关。

冲突 是不同的记录关键字通过相同的散列函数映射到内存同一个地址号的现象。解决冲突的办法有：

1. 开放定址法

若关键字 K 映射到存储地址 $D=H(Key)$，而 地址空间已被占用，则需扫描其他地址。寻找一个没有被占用的地址。

查找公式：$H(i)=(H(Key)+d_i)MOD, m_i=1,2,\cdots,n$。

其中：$H(key)$ 为散列函数；m 为散列表长度；d_i 为增量序列其值有三种取法：

(1) $d_i=1,2,\cdots,n$ 称线性探测再散列。

(2) $d_i=\pm1^2,\pm2^2,\pm3^2,\cdots,\pm n^2$ 称二次性探测再散列。

(3) $d_i=$ 伪随机序列，称伪随机探测再散列。

2. 再散列法

$H_i=RH_i(Key)$ $\quad i=1,2,\cdots,n$

RH_i 均为不同的散列函数。

3. 链地址法

将所有散列到同一地址的记录构成一个以散列得到的地址为头结点的单链表，即凡冲突的记录构成一个单链表。

4. 建立公共溢出区

为解决冲突除散列表的地址空间区外，专门另设置一部分冲突时公用的存储空间区。凡是发生冲空的记录按一定方式存放到公共溢出区构成溢出表。查找时，若在散列表中查找失败，再到溢出表中继续查找。

10.1.4 排序

排序 是根据记录的关键字，按照一定的规则，将一些无序的记录表整理成有 序记录表的过程。

内排序 是指要排序的记录表较小，全部记录能一次调入内存，且排序的整个 过程中(除全部排完序最后输出外)不与外设进行信息交换，这样的排 序称内排序。

外排序 是指要排序的文件大，记录多，需将记录分批调入内存进行部分排序，并将排好序的部分作为中间结果输出到外设，这样重复执行，需经多次 与外设交换信息最后才能完成全部记录的排序。称为外排序。

稳定与不稳定排序 在文件中若两个记录的关键字相同，即 $K_i=K_j$，且 $i\neq j$，若排序前 K_i 领先于 K_j，而全部记录排好序后，K_i 仍领先于 K_j 则称这种排序为稳定排序；否则称不稳定排序。

直接插入排序 (1)指将当前记录插到已排好序的子文件中合适的位置上的方式称直接插入。(2)对含有 n 个记录的未排序文件，当做 n 个仅含有一个记录的有序子文件看 待，将当前记录插到排序表合适位置上的方式称直接插入。时间复杂度为 $O(n^2)$。

例：设有 n 个记录 $R_1,R_2,\cdots,R_n$ 递增

算法：

```
ROGCEDURE INSERST SORT
BEGIN
    FOR i:=2 TO N DO                (从第2个记录起插入)
      R[0]:=R[i]                    (R[i]为当前要插入记录)
        J:=i-1                      (已排序子文件上限)
        WHILE R[0].KEY<R[J].KEY DO
            [R[J+1]:=R[J];J:=J-1]
    R[J+1]:=R[0]]                   (将第I个记录插入)
END
```

折半插入排序 在已排序的子文件中，用折半查找的方式寻找要插入的第 i 个记录应插入位置的方式称折半插入排序。对含有 $R_1,R_2\cdots,R_n$，n 个记录的表折半查找。

算法为：

```
RPOCEDURE BINRY SORT
  BEGIN
    FOR I:=2 TO N DO                (从第2个记录起进行插入)
    [X:=R[I];L:=1;H:=I-1            (L,H分别为折半查找上,下限)
    WHILE L≤H DO
      [M:=(L+H) DIV 2
      IF X.KEY<R[M].KEY THEN H:=M-1
                ELSE L:=M+1]
```