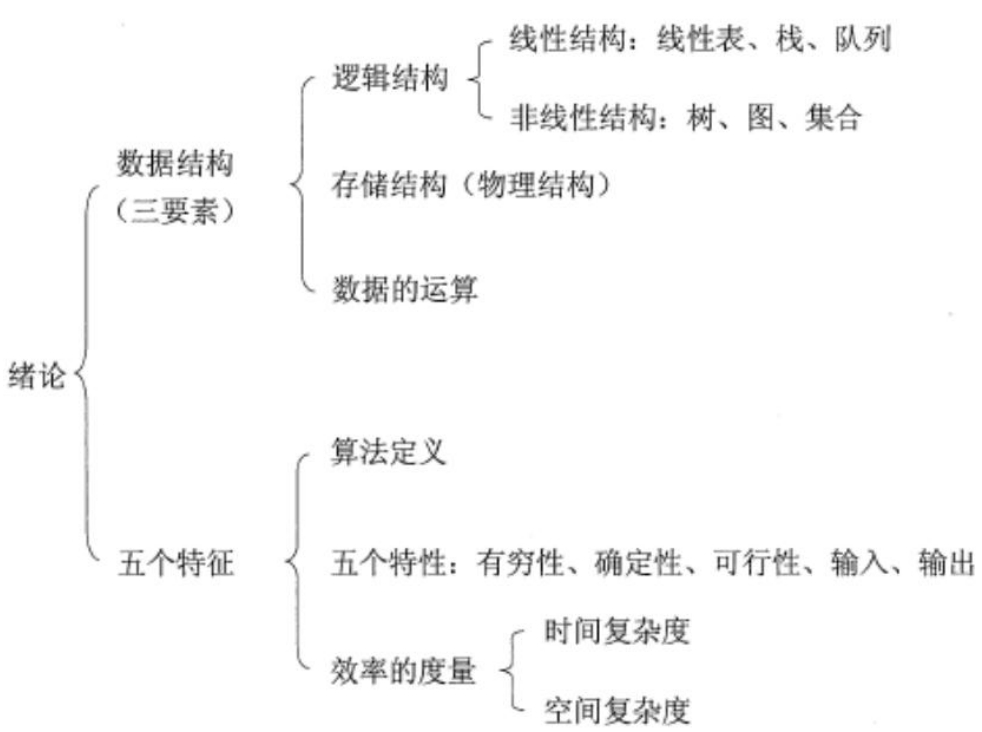
计算机保研知识点整理 数据结构篇

第一章 绪论



1、 时间复杂度

一个语句的频度是指该语句在算法中被重复执行的次数。算法中所有语句的频度之和记为T(n)，它是该算法问题规模 n 的函数，时间复杂度主要分析 T(n) 的数量级。算法中基本运算（最深层循环内的语句）的频度与 T(n) 同数量级，因此通常采用算法中基本运算的频度f(n)来分析算法的时间复杂度。因此，算法的时间复杂度记为 T(n) = O(f(n))

取 f(n) 中随 n 增长最快的项，将其系数置为 1 作为时间复杂度的度量。例如， f(n) = an3 + bn2 + cn 的时向复杂度为 O(n3)

上式中， O 的含义是 T(n) 的数量级，其严格的数学定义是：若 T(n)和 f(n)是定义在正整数集合上的两个函数，则存在正常数 C 和 n0，使得当 n >= n0 时，都满足 0 <=T(n) <=Cf(n) 。

算法的时间复杂度不仅依赖于问题的规模 n，也取决于待输入数据的性质（如输入数据元素的初始状态）

2、 空间复杂度

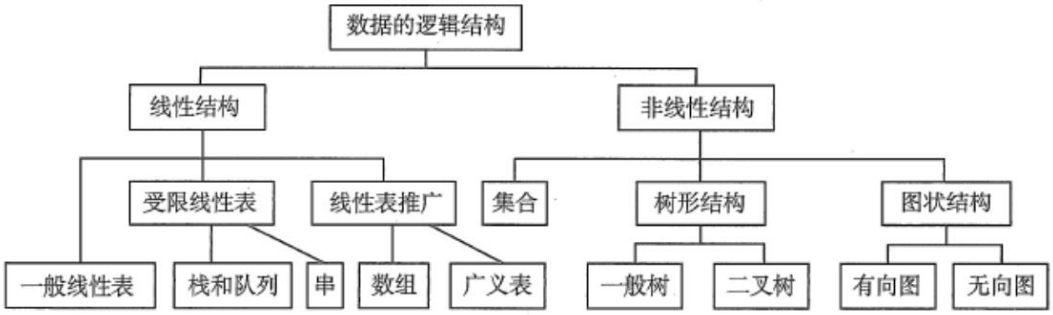
算法的空间复杂度 S(n)定义为该算法所耗费的存储空间，它是问题规模 n 的函数。记为 S(n) = O(g(n))

一个程序在执行时除需要存储空间来存放本身所用的指令、常数、变量和输入数据外，还需要一些对数据进行操作的工作单元和存储一些为实现计算所需信息的辅助空间。若输入数据

所占空间只取决于问题本身，和算法无关，则只需分析除输入和程序之外的额外空间。
算法原地工作是指算法所需的辅助空间为常量，即 O(1）。

3、 数的逻辑结构

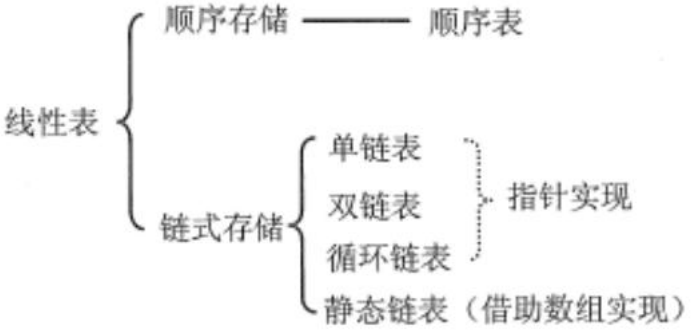
指的是数据元素之间逻辑关系，与数的存储结构无关，是独立于计算机的，以下是分类图。



4、 数的存储结构

存储结构是指数据结构在计算机中的表示，也称物理结构，主要有以下 4 种：
1）顺序存储。把逻辑上相邻的元素存储在物理位置上也相邻的存储单元中，元素之间的关系由存储单元的邻接关系来体现。其优点是可以实现随机存取，每个元素占用最少的存储空间；缺点是只能使用相邻的一整块存储单元，因此可能产生较多的外部碎片。
2）链式存储。不要求逻辑上相邻的元素在物理位置上也相邻，借助指示元素存储地址的指针来表示元素之间的逻辑关系。其优点是不会出现碎片现象，能充分利用所有存储单元；缺点是每个元素因存储指针而占用额外的存储空间，且只能实现顺序存取。
3）索引存储。在存储元素信息的同时，还建立附加的索引表。索引表中的每项称为索引项，索引项的一般形式是（关键字，地址）。其优点是检索速度快；缺点是附加的索引表额外占用存储空间。另外，增加和删除数据时也要修改索引表，因而会花费较多的时间。
4）散列存储。根据元素的关键字直接计算出该元素的存储地址，又称哈希(Hash）存储。其优点是检索、增加和删除结点的操作都很快；缺点是若散列函数不好，则可能出现元素存储单元的冲突，而解决冲突会增加时间和空间开销。

第二章 线性表



1、 顺序表和链表的比较

1.存取（读写）方式

顺序表可以顺序存取，也可以随机存取，链表只能从表头顺序存取元素。例如在第 i 个位置上执行存或取的操作，顺序表仅需一次访问，而链表则需从表头开始依次访问 i 次。

2.逻辑结构与物理结构

采用顺序存储时，逻辑上相邻的元素，对应的物理存储位置也相邻。而采用链式存储时，逻辑上相邻的元素，物理存储位置则不一定相邻，对应的逻辑关系是通过指针链接来表示的。

3.查找、插入和删除操作

对于按值查找，顺序表无序时，两者的时间复杂度均为 O(n)；顺序表有序时，可采用折半查找，此时的时间复杂度为 O(log2n) 。对于按序号查找，顺序表支持随机访问，时间复杂度仅为 O(1)，而链表的平均时间复杂度为 O(n) 。

顺序表的插入、删除操作，平均需要移动半个表长的元素。链表的插入、删除操作，只需修改相关结点的指针域即可。由于链表的每个结点都带有指针域，故而存储密度不够大。

4.空间分配

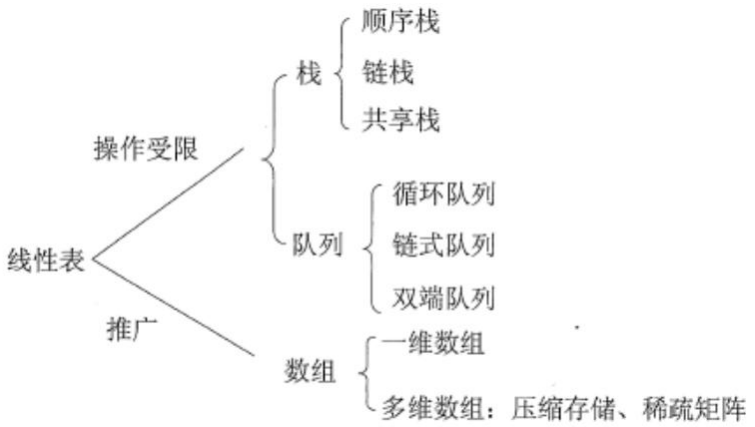
顺序存储在静态存储分配情形下，一旦存储空间装满就不能扩充，若再加入新元素，则会出现内存溢出，因此需要预先分配足够大的存储空间。预先分配过大，可能会导致顺序表后部大量闲置；预先分配过小，又会造成溢出。动态存储分配虽然存储空间可以扩充，但需要移动大量元素，导致操作效率降低，而且若内存中没有更大块的连续存储空间，则会导致分配失败。链式存储的结点空间只在需要时申请分配，只要内存有空间就可以分配，操作灵活、高效。

2、 头指针和头结点的区别

头指针：是指向第一个节点存储位置的指针，具有标识作用，头指针是链表的必要元素，无论链表是否为空，头指针都存在。

头结点：是放在第一个元素节点之前，便于在第一个元素节点之前进行插入和删除的操作，头结点不是链表的必须元素，可有可无，头结点的数据域也可以不存储任何信息。

第三章 栈和队列



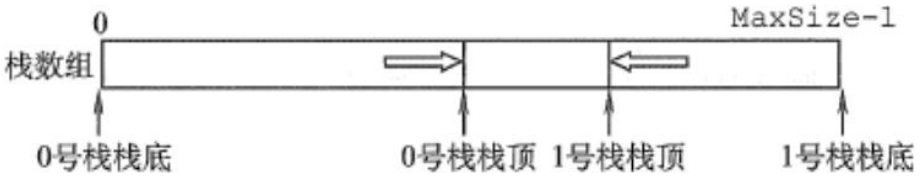
1、 栈和队列的区别

队列是允许在一段进行插入另一端进行删除的线性表。队列顾名思义就像排队一样，对于进入队列的元素按“先进先出”的规则处理，在表头进行删除在表尾进行插入。由于队列要进行频繁的插入和删除，一般为了高效，选择用定长数组来存储队列元素，在对队列进行操作之前要判断队列是否为空或是否已满。如果想要动态长度也可以用链表来存储队列，这时要记住队头和对位指针的地址。

栈是只能在表尾进行插入和删除操作的线性表。对于插入到栈的元素按“后进先出”的规则处理，插入和删除操作都在栈顶进行，与队列类似一般用定长数组存储栈元素。由于进栈和出栈都是在栈顶进行，因此要有一个 size 变量来记录当前栈的大小，当进栈时 size 不能超过数组长度，size+1，出栈时栈不为空，size−1。

2、 共享栈

利用栈底位置相对不变的特性，可以让两个顺序栈共享一个一维数组空间，将两个栈的栈底分别设置在共享空间的两端，两个栈顶向共享空间的中间延伸。这样能够更有效的利用存储空间，两个栈的空间相互调节，只有在整个存储空间被占满时才发生上溢。



3、 如何区分循环队列是队空还是队满？

普通情况下，循环队列队空和队满的判定条件是一样的，都是 Q.front == Q.rear。
ps:队头指针指向第一个数；队尾指针指向最后一个数的下一个位置，即将要入队的位置。
方法一：牺牲一个单元来区分队空和队满，这个时候(Q.rear+1)%MaxSize == Q.front 才是队满标志 。

方法二：类型中增设表示元素个数的数据成员。这样，队空的条件为 Q.size == 0;队满的条件为 Q.size == MaxSize。

4、 栈在括号匹配中的算法思想

（1）出现的凡是“左括号”，则进栈；
（2）出现的是“右括号”，
首先检查栈是否空？
若栈空，则表明该“右括号”多余
否则和栈顶元素比较？
若相匹配，则栈顶“左括号出栈”
否则表明不匹配
（3）表达式检验结束时，
若栈空，则表明表达式中匹配正确
否则表明“左括号”有余；

5、 栈在递归中的应用

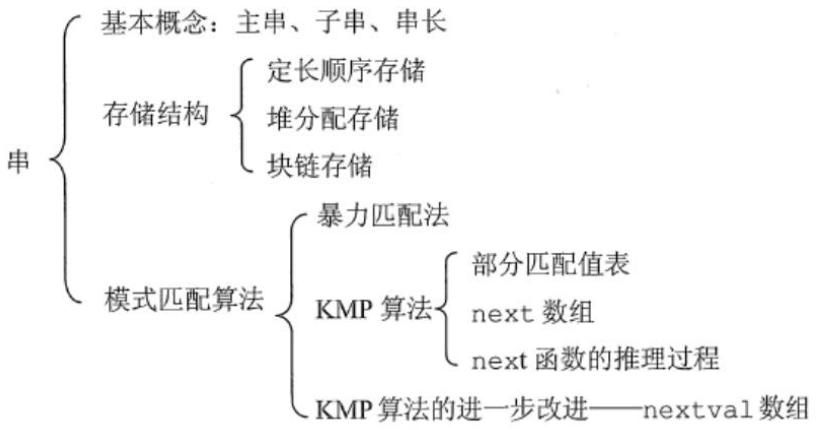
递归是一种重要的程序设计方法。简单地说，若在一个函数、过程或数据结构的定义中又应用了它自身，则这个函数、过程或数据结构称为是递归定义的，简称递归。

它通常把一个大型的复杂问题层层转化为一个与原问题相似的规模较小的问题来求解，递归策略只需少量的代码就可以描述出解题过程所需要的多次重复计算，大大减少了程序的代码量。但在通常情况下，它的效率并不是太高。将递归算法转换为非递归算法，通常需要借助栈来实现这种转换。

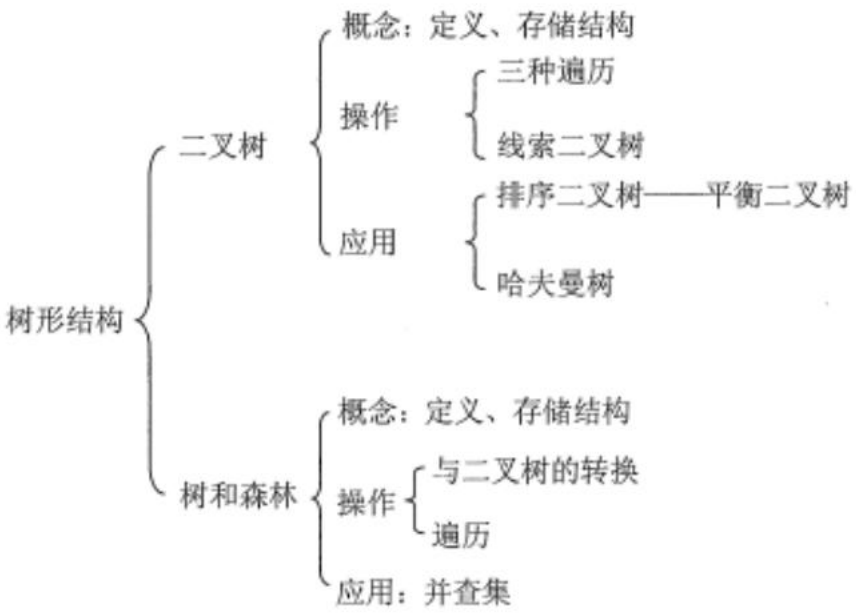
6、 队列在层次遍历中的作用

在信息处理中有一大类问题需要逐层或逐行处理。这类问题的解决方法往往是在处理当前层或当前行时就对下一层或下一行做预处理，把处理顺序安排好，待当前层或当前行处理完毕，就可以处理下一层或下一行。使用队列是为了保存下一步的处理顺序。

第四章 串


串
基本概念：主串、子串、串长
存储结构
定长顺序存储
堆分配存储
块链存储
模式匹配算法
暴力匹配法
KMP 算法
部分匹配值表
next 数组
next 函数的推理过程
KMP 算法的进一步改进——nextval 数组

第五章 树与二叉树



1、 概念

树是非线性结构，其元素之间有明显的层次关系。在树的结构中，每个节点都只有一个前件称为父节点，没有前件的节点为树的根节点，简称为树的根；每个节点可以有多个后件成为节点的子节点，没有后件的节点称为叶子节点。

在树的结构中，一个节点所拥有的子节点个数称为该节点的度，树中最大的节点的度为树的度，树的最大的层次称为树的深度

二叉树：二叉树是另一种树形结构，其特点是每个结点至多只有两棵子树，并且二叉树的子树有左右之分，其次序不能任意颠倒。与树相似，二叉树也以递归的形式定义。二叉树是 n (n >=0) 个结点的有限集合：
1)或者为空二叉树，即 n=0 。
2)或者由一个根结点和两个互不相交的被称为根的左子树和右子树组成。左子树和右子树又分别是一棵二叉树。
二叉树是有序树，若将其左、右子树颠倒，则成为另一棵不同的二叉树。即使树中结点只有一棵子树，也要区分它是左子树还是右子树

满二叉树：满二叉树是指除了最后一层外其他节点均有两颗子树。

完全二叉树：完全二叉树是指除了最后一层外，其他任何一层的节点数均达到最大值，且最后一层也只是在最右侧缺少节点

二叉树的存储：二叉树可以用链式存储结构来存储，满二叉树和完全二叉树可以用顺序存储

结构来存储

二叉树的遍历：二叉树有先序遍历（根左右），中序遍历（左根右）和后续遍历（左右根）；还有层次遍历，需要借助一个队列。三种遍历算法中，递归遍历左、右子树的顺序都是固定的，只是访问根结点的顺序不同。不管采用哪种遍历算法，每个结点都访问一次且仅访问一次，故时间复杂度都是 O(n) 。在递归遍历中，递归工作栈的栈深恰好为树的深度，所以在最坏情况下，二叉树是有 n 个结点且深度为 n 的单支树，遍历算法的空间复杂度为 O(n) 。

2、 如何遍历序列构造一颗二叉树？

1)由二叉树的先序序列和中序序列可以唯一地确定一棵二叉树。

在先序遍历序列中，第一个结点一定是二叉树的根结点；而在中序遍历中，根结点必然将中序序列分割成两个子序列，前一个子序列是根结点的左子树的中序序列，后一个子序列是根结点的右子树的中序序列。根据这两个子序列，在先序序列中找到对应的左子序列和右子序列。在先序序列中，左子序列的第一个结点是左子树的根结点，右子序列的第一个结点是右子树的根结点。如此递归地进行下去，便能唯一地确定这棵二叉树。

2)由二叉树的后序序列和中序序列也可以唯一地确定一棵二叉树。因为后序序列的最后一个结点就如同先序序列的第一个结点，可以将中序序列分割成两个子序列，然后采用类似的方法递归地进行划分，进而得到一棵二叉树。

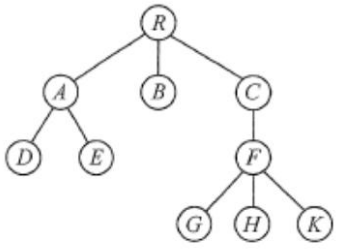
3）由二叉树的层序序列和中序序列也可以唯一地确定一棵二叉树。需要注意的是，若只知道二叉树的先序序列和后序序列，则无法唯一确定一棵二叉树。

3、 树的存储结构

1、 双亲表示法

这种存储方式采用一组连续空间来存储每个结点，同时在每个结点中增设一个伪指针，指示其双亲结点在数组中的位置。

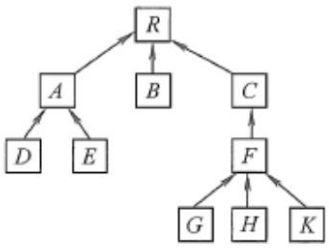
该存储结构利用了每个结点（根结点除外）只有唯一双亲的性质，可以很快得到每个结点的双亲结点，但求结点的孩子时需要遍历整个结构。



(a) 一棵树

	data	parent
0	R	−1
1	A	0
2	B	0
3	C	0
4	D	1
5	E	1
6	F	3
7	G	6
8	H	6
9	K	6

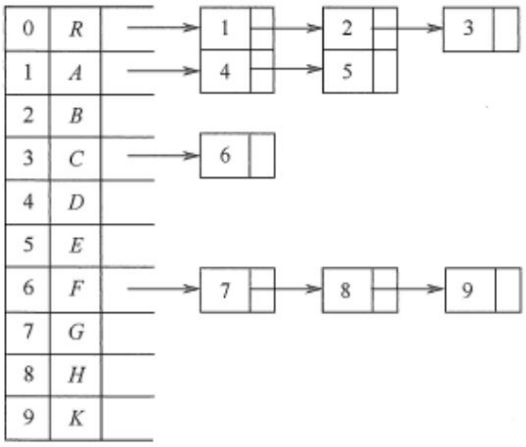
(b) 双亲表示



(c) 双亲指针图示

2、 孩子表示法

孩子表示法是将每个结点的孩子结点都用单链表链接起来形成一个线性结构，此时 n 个结点就有 n 个孩子链表（叶子结点的孩子链表为空表），这种存储方式寻找子女的操作非常直接，而寻找双亲的操作需要遍历 n 个结点中孩子链表指针域所指向的 n 个孩子链表。

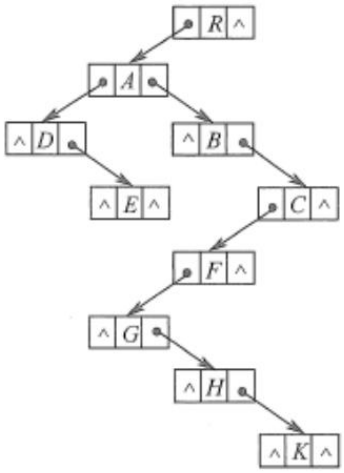


(a) 孩子表示法

3、 孩子兄弟表示法

孩子兄弟表示法又称二叉树表示法，即以二叉链表作为树的存储结构。孩子兄弟表示法使每个结点包括三部分内容：结点值、指向结点第一个孩子结点的指针，及指向结点下一个兄弟结点的指针（沿此域可以找到结点的所有兄弟结点）

这种存储表示法比较灵活，其最大的优点是可以方便地实现树转换为二叉树的操作，易于查找结点的孩子等，但缺点是从当前结点查找其双亲结点比较麻烦。若为每个结点增设一个 parent 域指向其父结点，则查找结点的父结点也很方便。



(b) 孩子兄弟表示法

4、 二叉排序树

1.二叉排序树的定义：

二叉排序树（也称二叉查找树）或者是一棵空树，或者是具有下列特性的二叉树：

1）若左子树非空，则左子树上所有结点的值均小于根结点的值。

2）若右子树非空，则右子树上所有结点的值均大于根结点的值。

3）左、右子树也分别是一棵二叉排序树。

根据二叉排序树的定义，左子树结点值＜根结点值＜右子树结点值，所以对二叉排序树进行中序遍历，可以得到一个递增的有序序列。

2、二叉排序树的查找

二叉排序树的查找是从根节点开始的，延某个分支逐层向下比较的过程。若二叉树非空，先将给定值与根结点的关键字比较，若相等，则查找成功；若不等，如果小于根结点的关键字，则在根结点的左子树上查找，否则在根的右子树上查找。这显然是一个递归的过程。

3. 平衡二叉树

为避免树的高度增长过快，降低二叉排序树的性能，规定在插入和删除二叉树结点时，要保证任意结点的左、右子树高度差的绝对值不超过 1，将这样的二叉树称为平衡二叉树(Balanced BinaryTree)，简称平衡树。定义结点左子树与右子树的高度差为该结点的平衡因子，则平衡二叉树结点的平衡因子的值只可能是−1 、0 或 1 。因此，平衡二叉树可定义为或者是一棵空树，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，且左子树和右子树的高度差的绝对值不超过 1 。

5、 哈夫曼树和哈弗曼编码

1、 哈弗曼树的构造

给定 n 个权值分别为 W1,W2…， Wn 的结点，构造哈夫曼树的算法描述如下：

1）将这 n 个结点分别作为 n 棵仅含一个结点的二叉树，构成森林 F 。

2）构造一个新结点，从 F 中选取两棵根结点权值最小的树作为新结点的左、右子树，并且将新结点的权值置为左、右子树上根结点的权值之和。

3）从 F 中删除刚才选出的两棵树，同时将新得到的树加入 F 中。

4）重复步骤 2）和 3)'直至 F 中只剩下一棵树为止。

从上述构造过程中可以看出哈夫曼树具有如下特点：

1）每个初始结点最终都成为叶结点，且权值越小的结点到根结点的路径长度越大。

2）构造过程中共新建了 n — 1 个结点（双分支结点），因此哈夫曼树的结点总数为 2n —1 。

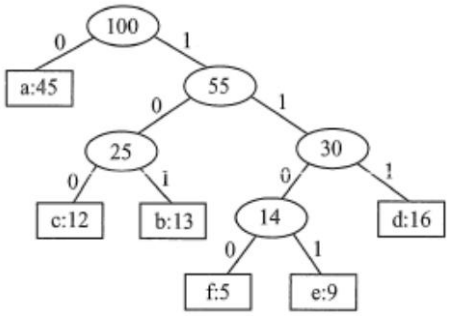
3）每次构造都选择 2 棵树作为新结点的孩子，因此哈夫曼树中不存在度为 1 的结点。

2、哈夫曼编码

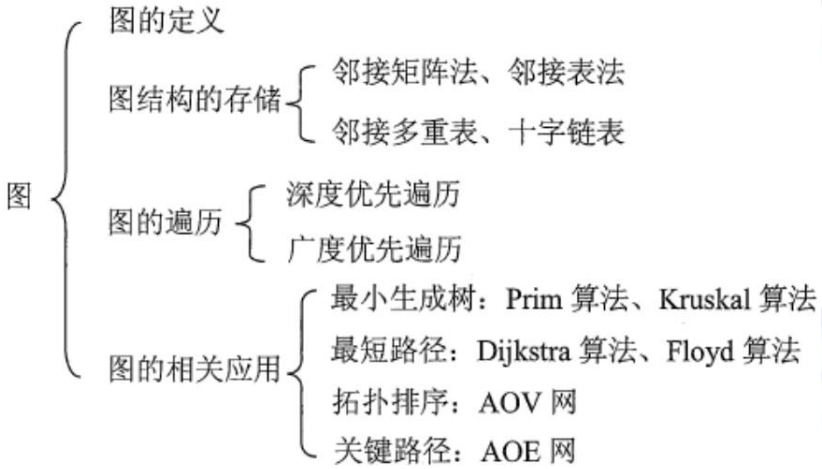
在数据通信中，若对每个字符用相等长度的二进制位表示，称这种编码方式为固定长度编码。若允许对不同字符用不等长的二进制位表示，则这种编码方式称为可变长度编码。可变长度编码比固定长度编码要好得多，其特点是对频率高的字符赋以短编码，而对频率较低的字符则赋以较长一些的编码，从而可以使字符的平均编码长度减短，起到压缩数据的效果。哈夫曼编码是一种被广泛应用而且非常有效的数据压缩编码。若没有一个编码是另一个编码的前缀，则称这样的编码为前缀编码。

由哈夫曼树得到哈夫曼编码是很自然的过程。首先，将每个出现的字符当作一个独立的结点，其权值为它出现的频度（或次数），构造出对应的哈夫曼树。显然，所有字符结点都出现在叶结点中。我们可将字符的编码解释为从根至该字符的路径上边标记的序列，其中边标记为 0 表示“转向左孩子”，标记为 1 表示“转向右孩子“。

各字符编码为
a:0
b:101
c:100
d:111
e:1101
f:1100



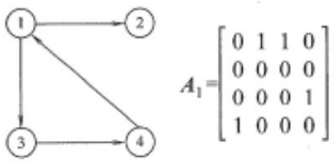
第六章 图



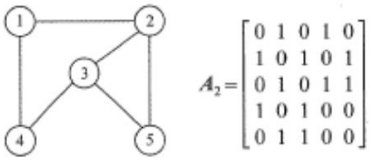
1、 图的存储结构

1、 邻接矩阵法

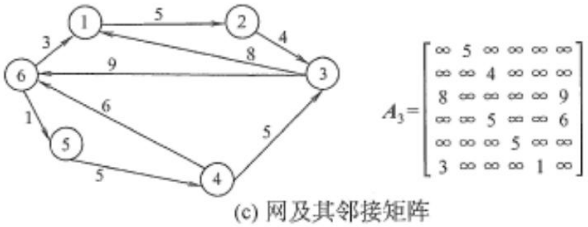
所谓邻接矩阵存储，是指用一个一维数组存储图中顶点的信息，用一个二维数组存储图中边的信息（即各顶点之间的邻接关系），存储顶点之间邻接关系的二维数组称为邻接矩阵。有向图、无向图和网对应的邻接矩阵实例图如下：



(a) 有向图G_1及其邻接矩阵



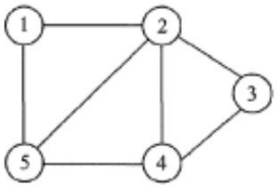
(b) 无向图G_2及其邻接矩阵



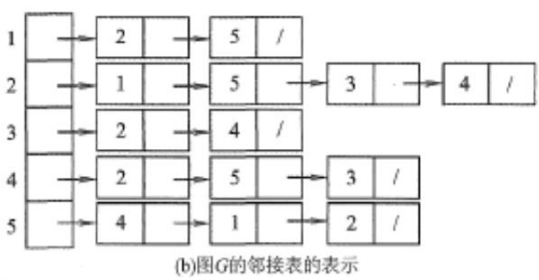
(c) 网及其邻接矩阵

2、 邻接表法

当一个图为稀疏图时，使用邻接矩阵法显然要浪费大量的存储空间，而图的邻接表法结合了顺序存储和链式存储方法，大大减少了这种不必要的浪费。所谓邻接表，是指对图 G 中的每个顶点 V 建立一个单链表，第 i 个单链表中的结点表示依附于顶点 v，的边（对于有向图则是以顶点 v，为尾的弧），这个单链表就称为顶点 vi 的边表（对于有向图则称为出边表）。边表的头指针和顶点的数据信息采用顺序存储（称为顶点表），所以在邻接表中存在两种结点：顶点表结点和边表结点。



(a)无向图G



(b)图G的邻接表的表示

3、 十字链表法

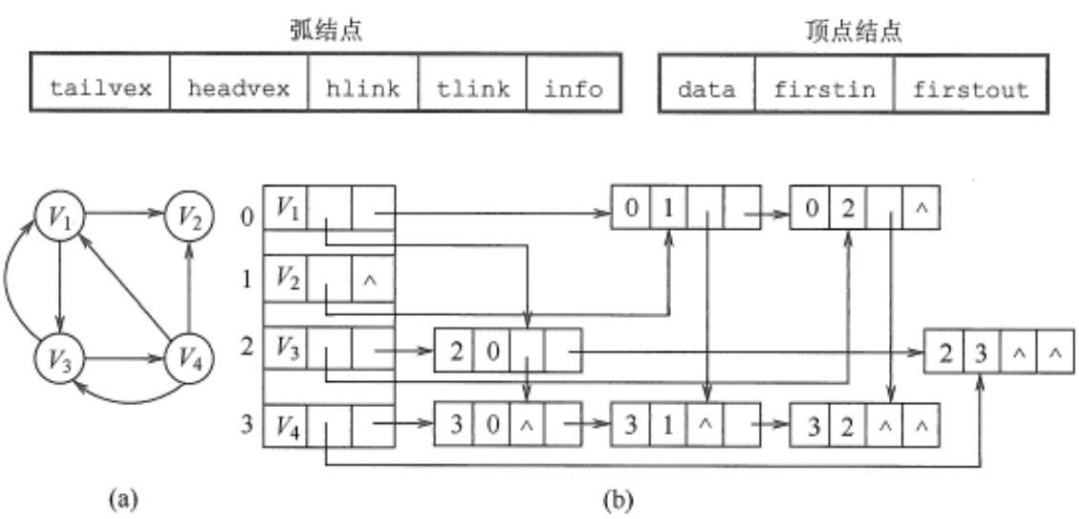


图 6.9　有向图的十字链表表示

2、 图的遍历

1、 广度优先搜索

类似于二叉树的层序遍历算法。基本思想是：首先访问起始顶点 V，接着由 V 出发，依次访问 V 的各个未访问过的邻接顶点 W1，W2，. . . . Wn，然后依次访问 W1，W2，. . . . ，Wn 的所有未被访问过的邻接顶点；再从这些访问过的顶点出发，访问它们所有未被访问过的邻接顶点，直至图中所有顶点都被访问过为止。若此时图中尚有顶点未被访问，则另选图中一个未曾被访问的顶点作为初始点，重复上述过程。Dijkstra 源最短路径算法和 Prim 最小生成树算法也应用了类似的思想。

2、 深度优先搜索

它的基本思想如下：首先访问图中某一起始顶点 V，然后由 v 出发，访问与 v 邻接且未被访问的任一顶点 W1，再访问与 W1 邻接且未被访问的任一顶点 W2. . . .…重复上述过程。当不能再继续向下访问时，依次退回到最近被访问的顶点，若它还有邻接顶点未被访问过，则从该点开始继续上述搜索过程，直至图中所有顶点均被访问过为止。

3、 最小生成树和最短路径

迪杰斯特拉（dijkstra）算法：迪杰斯特拉算法是经典的单源最短路径算法，用于求某一顶点到其他顶点的最短路径，它的特点是以起始点为中心层层向外扩展，直到扩展的终点为止，迪杰斯特拉算法要求边的权值不能为负权。

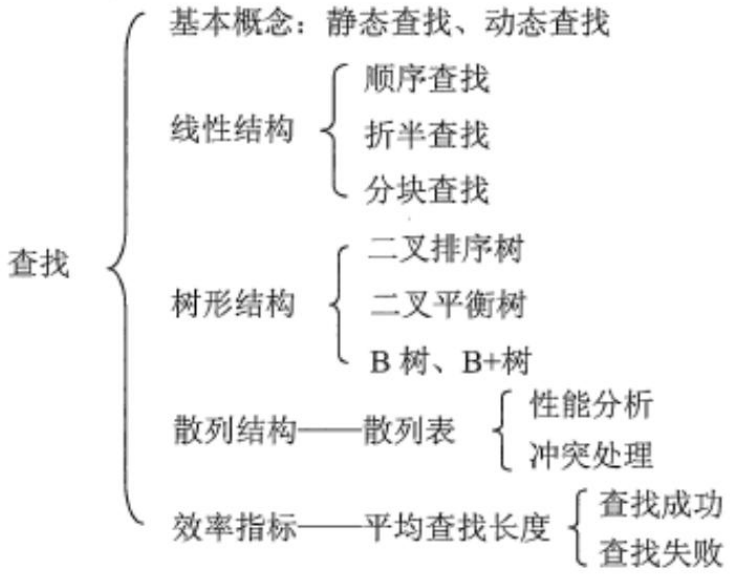
弗洛伊德（Floyd）算法：弗洛伊德算法是经典的求任意顶点之间的最短路径，其边的权值可为负权，该算法的时间复杂度为 O(N3),空间复杂度为 O（N2)。

普里姆（prim）算法：用来求最小生成树，其基本思想为：从联通网络 N={V,E}中某一顶点 u0 出发，选择与他关联的最小权值的边，将其顶点加入到顶点集 S 中，此后就从一个顶点在 S 集中，另一个顶点不在 S 集中的所有顶点中选择出权值最小的边，把对应顶点加入到 S 集中，直到所有的顶点都加入到 S 集中为止。

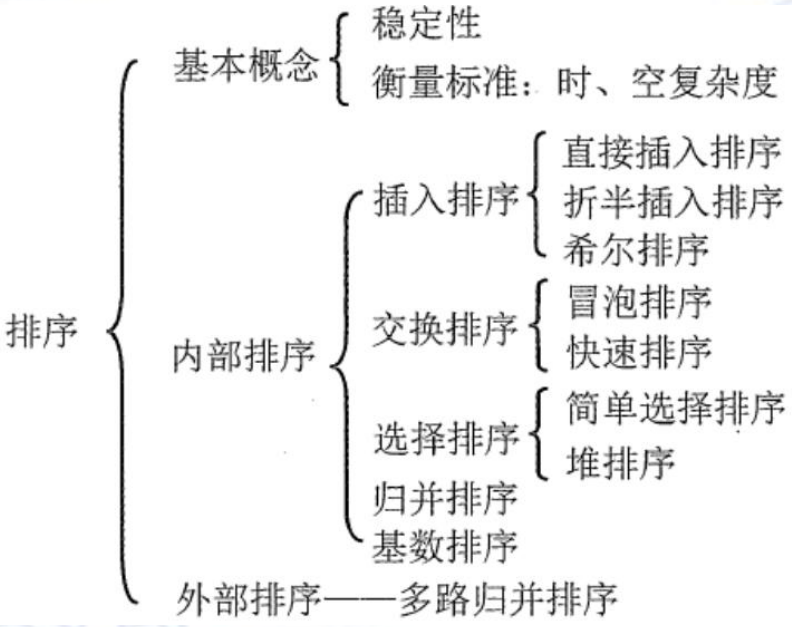
克鲁斯卡尔（kruskal）算法：用来求最小生成树，其基本思想为：设有一个有 N 个顶点的联

通网络 N={V,E},初试时建立一个只有 N 个顶点，没有边的非连通图 T，T 中每个顶点都看作是一个联通分支,从边集 E 中选择出权值最小的边且该边的两个端点不在一个联通分支中,则把该边加入到 T 中，否则就再从新选择一条权值最小的边，直到所有的顶点都在一个联通分支中为止。

第七章 查找


查找
基本概念：静态查找、动态查找
线性结构
顺序查找
折半查找
分块查找
树形结构
二叉排序树
二叉平衡树
B 树、B+树
散列结构——散列表
性能分析
冲突处理
效率指标——平均查找长度
查找成功
查找失败

第八章 排序


排序
基本概念
稳定性
衡量标准：时、空复杂度
内部排序
插入排序
直接插入排序
折半插入排序
希尔排序
交换排序
冒泡排序
快速排序
选择排序
简单选择排序
堆排序
归并排序
基数排序
外部排序——多路归并排序