绪论

抽象数据类型

原子类型（atomic data type）

固定聚合类型（fixed-aggregate data type）

可变聚合类型（variable-aggregate data type）

形式化定义

D是数据对象

S是D上的关系集

P是对D的基本操作

算法特性

（1）输入性

（2）输出性

（3）有穷性

（4）确定性

（5）可执行性

算法和程序的区别

（1）一个程序不一定满足有穷性

（2）程序中的指令必须是机器可以执行的，而算法中的指令没有此要求。

（3）一个算法，若用机器可执行的语言书写，那它就是一个程序了。

（4）一个程序如果对任何输入都不会陷入无限循环，则它就是一个算法。

算法设计要求

正确性

可读性

健壮性

效率和存储量

算法描述

自然语言描述

流程图或N-S图描述

计算机语言描述

伪代码描述

栈和队列

栈顶top始终指向栈顶元素的**下一个**位置。

插入和删除运算的单链表，**其表头指针叫做栈顶指针。**

队列只允许在表的一端**（队尾）进行插入**，在另一端**（队头）进行删除。**

求队列的长度

```
int QueueLength(SqQueue Q){
    L=(Q.rear-Q.front+MAXQSIZE)%MAXQSIZE;
    return (L);
}
```

队列的应用举例

简化的生产者与消费者问题

串

串的逻辑结构与线性表相似，**区别仅在于串的数据对象约束为字符集。**

串的基本操作和线性表的差别：**在串的操作中，多以"串的整体"作为对象。**

KMP

数组和广义表

数组定义方式1

一个n维数组类型可以定义为其数组元素为n-1维数组的一位数组类型。

当n=1时，n维数组就退化为定长的线性表，每个元素不可再分解。

特殊矩阵

矩阵中的元素排列是有规律的

矩阵的非零元素很少

压缩存储

为多个值相同的元素只分配一个存储空间

对零元素不分配空间

对称矩阵

压缩存储：只存主对角线以上或以下的元素，可用一位数组存储。

减少存储空间：$n^2-n(n+1)/2$

$$
\begin{aligned}
&\text{下三角元素}a_{ij},(i\geq j)\text{与一维数组}Sa[k]\text{元素的对应关系}\\
&k=i(i-1)/2+j-1,(0\leq k\leq n(n+1)/2-1)\\
&\text{上三角元素}a_{ij},(i\leq j)\text{与一维数组}Sa[k]\text{元素的对应关系}\\
&k=j(j-1)/2+i-1,(0\leq k\leq n(n+1)/2-1)
\end{aligned}
$$

对角矩阵

性质：一个n阶方阵的所有非零元素都集中在以主对角为中心的带状区域中

压缩存储：将非零元素存储到一维数组中

非零元素数目（最多）：$2+(n-2)*3+2$

稀疏矩阵

性质：矩阵中大多数元素值为0，元素分布没有一定规律。

$$
\begin{aligned}
&\text{设在}m\times n\text{矩阵中，有}t\text{个元素不为零，稀疏因子为：}\sigma=t/(m*n)\\
&\text{当}\sigma\leq 0.05\text{时，称为稀疏矩阵}
\end{aligned}
$$

压缩存储方式：

三元组顺序存储

链表存储

树

定义

树的结点：包括一个数据元素及若干个指向其子树的分支。

结点的度：结点具有子树的个数。

树的度：树中所有结点的度的最大值。

分支结点：度大于0的结点。

叶子（终端节点）：度为0的结点。

树的深度：树中结点的最大层次数。

树，森林与二叉树的转换

森林=>二叉树

以第一棵树的根为二叉树的根，先将第一棵树转为二叉树，再一次转换其他子树，最后按**孩子兄弟表示法**将它们连接。

二叉树=>森林

二叉树的根为第一颗树的根，先把左儿子转为第一棵树，再将右子树转换为一颗或若干棵树。

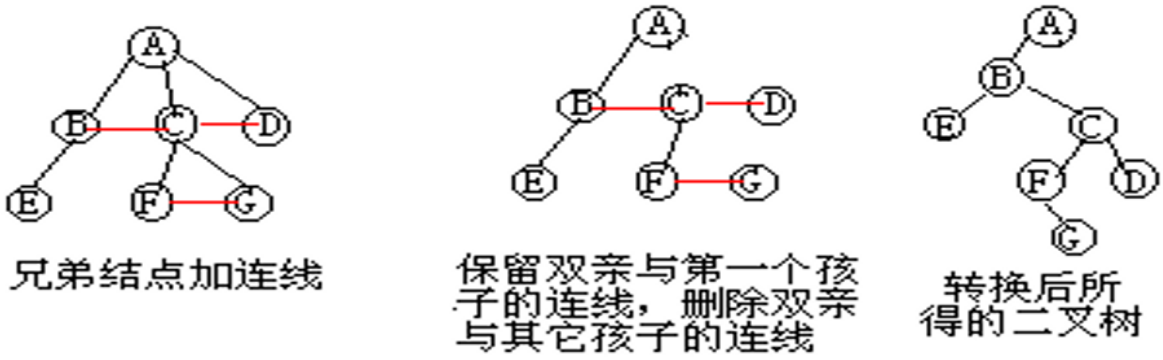
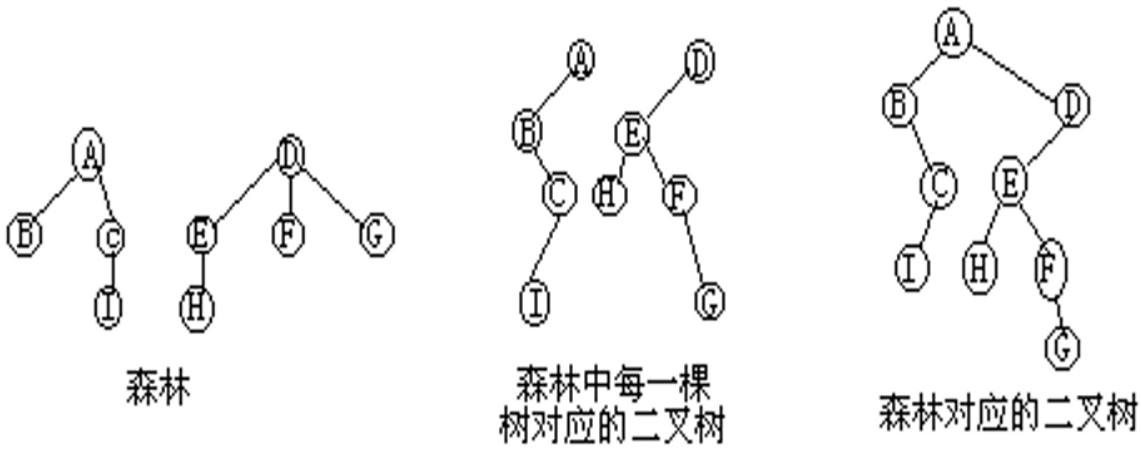
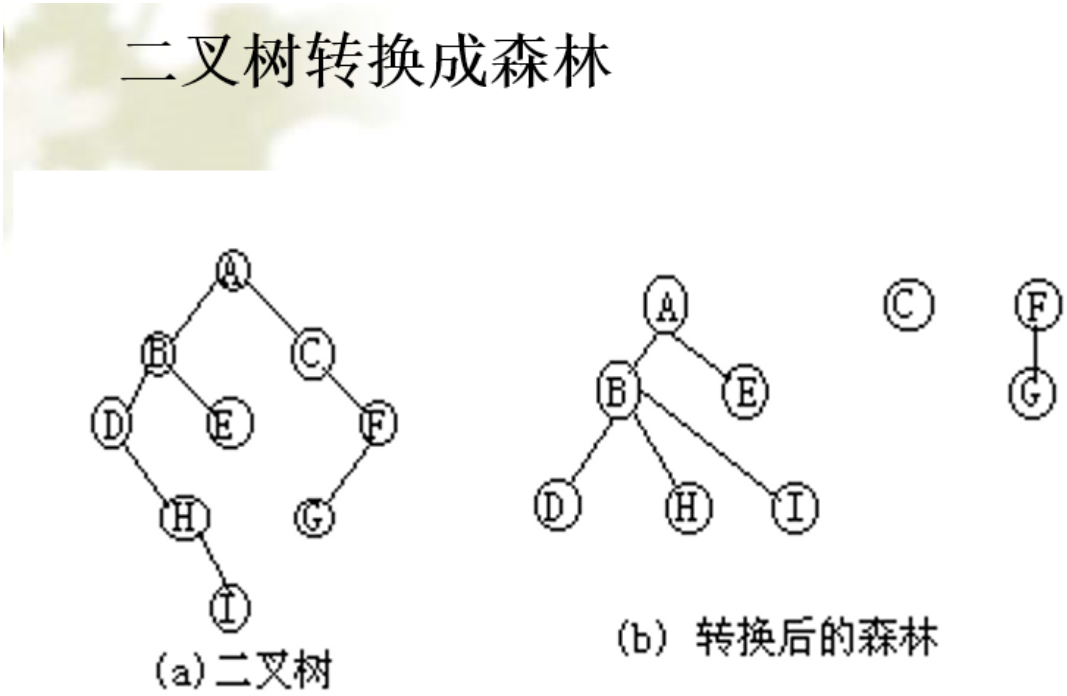


图6.4.5 树转换成二叉树的过程示意图

森林转换成二叉树的过程





赫夫曼树及其应用

带权路径长度：从结点到根之间的路径长度与结点上权值的乘积

树的带权路径长度：树中所有叶子结点的带权路径长度之和。

图

有向图的逆邻接表

十字链表

邻接多重表

非连通图生成森林

拓扑排序

最小生成树

Kruskal算法：将边归并，适于求稀疏网的最小生成树。

Prime算法：将顶点归并，与边数无关，适于稠密网。

AOE-网与关键路径

AOE-网：用边表示活动的网（无环），顶点表示事件，弧表示活动，权表示持续的时间。**可用于估算工程的完成时间。**

AOE网具有以下性质：

（1）只有在某个顶点代表的事件发生后，从该顶点出发的各条弧所代表的活动才能开始。

（2）只有在进入某一顶点的各条弧代表的活动结束后，该顶点所代表的事件才能发生。

关键路径(Critical Path)：从源点到汇点之间所有路径中最长的路径。

关键活动：满足$el(i) == ee(i)$的a_i。关键路径是关键活动组成的路径。

若AOE — 网中只存在一条关键路径，则提高任一关键活动的速度可缩短工期。

若AOE — 网中存在几条关键路径，提高它们的共同关键活动，才可缩短工期。

查找

两种查找表

静态查找表：对查找表的查找仅是以查询为目的，不改动查找表中的数据。

动态查找表：在查找过程中同时伴随插入不存在的元素或删除某个已存在的元素。

折半查找法

只适用于有序表，且限于顺序存储结构。

分块查找

把线性表分成若干块：前一块的最大关键字小于后一块的最小关键字值。

建立一个索引表。

分块查找的步骤：先确定待查记录所在的块；任何在块中顺序查找。

二叉排序树

属于动态查找表。

在最好情况下，平均查找长度为$log_2 n$

在最坏情况下，平均查找长度为$(n+1)/2$

平衡二叉树

又称AVL树，具有性质：它的左右子树都是平衡二叉树，且左右子树的深度之差绝对值不超过1。

BF(平衡因子)：为某结点的左子树的深度减去右子树深度，只可能为1，-1，0

四种调整情况

（1）LL型调整（单向右旋平衡处理）

（2）RR型调整（单向左旋平衡处理）

（3）LR型调整（双向旋转，先左后右）

（4）RL型调整（双向调整，先右后左）

B-树

满足以下特性：

①树中每个结点至多有m棵子树；

②若根结点不是叶子节点，则至少有两颗子树；

③除根以外的所有非终结点至少有[m/2]棵子树。

④所有的非终端结点中包含下列信息数据，$K_i < K_{i+1}$，指针A_{i-1}所指子树中所有结点的关键字均小于K_i,A_n所指子树中所有结点的关键字均大于K_n

⑤所有的叶子结点都出现在同一层次上，且不带信息。

B-树的每个结点中还包括n个指向每个关键字的记录的指针。

用途：主要用作文件的索引。

B+树

与B-树的差异在于：

（1）有n棵子树的结点中含有n个关键字。

（2）所有叶子结点中包含了全部关键字的信息，以及指向这些关键字记录的指针，且叶子结点本身依关键字从小到大链接。

（3）所有的非终端结点可以看成索引部分，仅含有其子树中的最大或最小关键字。

注意：查找时，若非终端结点上的关键字等于给定值，并不终止，而是继续向下直到叶子结点。

哈希函数构造方法

直接定址法

数字分析法

平方取中法

折叠法

除留余数法

随机数法

处理冲突方法

开放定址法

$$H_i = (H(key) + d_i) \mod m, i = 1, 2, \ldots, m-1$$

其中：$H(key)$为哈希函数；m为哈希表表长；d_i为增量序列

线性探测再散列：$d_i = 1, 2, 3, \ldots, m-1$

二次探测再散列：$d_i = 1^2, -1^2, 2^2, -2^2, \ldots$

随机探测再散列：$d_i =$ 伪随机数序列

链地址法：把具有相同哈希地址的关键字值放在同一链表中，称为同义词链表。

散列表的查找效率主要取决于散列表造表时选取的散列函数和处理冲突的方法。

集合可以使用位向量、有序链表_和并查集来实现。

内部排序

排序的类型

插入排序：直接插入，折半插入，2-路插入，表插入，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，树形选择排序，堆排序

归并排序

基数排序

内部排序：指的是数据结构全部放在内存中，不涉及外存的排序方法。

外部排序：指的是待排序的数据量很大，以至内存一次不能容纳全部数据。在排序过程中尚需要对外存进行访问的排序过程。

如果一个排序对任意一个排序过的关键字K_i, K_j,它们的相对位置不变，那么我们说该排序方法是**稳定的。**

插入排序

折半插入：针对已排序列"查找操作"，用"折半查找"来实现。减少关键字间的比较次数，而记录的移动次数不变。

2-路插入：以折半插入排序为基础，将第一个数看成处于中间位置数，小于它的放在它的左边，大于它的放在它的右边。可以减少移动记录次数，但需要$O(n)$的辅助空间。

表插入：通过修改记录指针的方式插入到静态链表中。减少在排序过程中移动记录的次数。

希尔排序：若待排记录序列按关键字基本有序时，只要作记录的少量比较和移动即可完成排序。特点：子序列的构成是将相隔某个增量的记录组成一个子序列。

交换排序

冒泡排序：稳定。

快速排序：不稳定。去第一个记录作为枢轴记录，将其他元素按大小分隔两部分，这一过程称作一趟快速排序。

选择排序

简单选择排序

树形选择排序:

堆排序：不稳定。辅助空间$O(1)$,时间效率$O(nlog_2n)$

归并排序

基数排序

多关键字排序

最高位优先（MSD）

最低位优先（LSD）

数据的逻辑结构

集合

线性结构

树型结构

图状结构

数据的存储结构

顺序存储结构

链式存储结构

在算法设计中一些常用的存储结构，例如：

-线性表：SqList,LinkList

-栈：SqStack,LinkList

-队列：Queue,LinkQueue,SqQueue

-串：SStirng,HString,LString

-矩阵：TSMatrix,RLSMatrix,CrossList

-二叉树：SqBiTree(仅适合完全二叉树)，Bitree,BiThrTree

-树：PTree,CSTree,B-树，B+树

-图：MGraph,ALGraph

基本操作

线性表

创建，读取，定位，插入，删除，合并等

栈

入栈Push,出栈Pop

队列

入队EnQueue,出队DeQueue

串

复制，比较，联接，求子串，插入，删除等

矩阵

初始化，赋值，特殊矩阵创建，稀疏矩阵加、减、相乘

广义表

求表头，表尾，深度等

二叉树

遍历（先中后），创建，由两个序列构成一棵树

树与森林

树的先根、后根遍历，森林的先序、中序遍历，树、二叉树、森林之间相互转换

图

创建，深度、广度遍历

应用问题

数制转换 √

括号匹配 √

行编辑

#：表示退格符

@：表示退行符

表达式求值 √

```
OperandTypeEvaluateExpression(){
    InitStack(OPTR);
    push(OPTR,'#');
    InitStack(OPND);
    c=getchar();
    while(c!='#'||GetTop(OPTR)!='#'){
        if(!In(c,op)){ //c不是运算符
            push(OPND,c);
            c=getchar();
        }else{
            switch(precede(GetTop(OPTR),c))//Precede函数比较运算符栈顶元素和c的优先级
            case '<':push(OPTR,c);c=getchar();break;
            case '=':pop(OPTR,c);c=getchar();break;
            case
'>':pop(OPTR,theta);pop(OPND,b);pop(OPND,a);push(OPND,Operate(a,theta,b));
        }
    }
    return GetTop(OPND);
}
```

哈夫曼树的构造，编码 √

求最小生成树 √

拓扑排序 √

求关键路径 √

求最短路径 √

顺序查找 √

折半查找 √

二叉排序树的建立、插入、删除节点 √

平衡二叉树的调整 √

B-树，B+树插入、删除调整 √

哈希查找表的构建与冲突解决 √

各种排序方法的实现过程 √

题型

单选、填空、判断、简答/应用题、算法设计题