Chapter 5 树

树的基本概念

自由树：二元组$\{V,E\}$，其中V是n个顶点的集合，E是$n-1$条边的集合，E使得V连通

有根树：是n个结点的建有限集合，$n=0$时称为空树，否则T非空，记作$T=\{r,T_1,T_2,..T_n\}$。有一个特殊的结点r是根，其余结点不相交的集合T_i也是一颗树，称为子树。每个子树只有一个直接前驱，有 0 个或多个直接后继

树的表示方法：

- 逻辑表示
- 目录结构表示
- 集合文氏图表示
- 凹入表表示
- 广义表表示

二叉树

二叉树：二叉树是一个结点的有限集合，该集合或者为空，或者是由一个根结点加上两颗分别称为左子树和右子树的互不相交的二叉树构成。子树仍是二叉树，遇到空树时递归定义停止

二叉树每个结点最多有两个子女，且有左右之分，次序不可颠倒

满二叉树：深度为k的满二叉树是有2^k-1个结点的二叉树（根深度按 1）

完全二叉树：具有n个结点的深度为k的二叉树，其中每个结点都与高为k的满二叉树中编号为 1~n 的结点互相对应

二叉树的性质：

- 对一颗非空二叉树，叶结点树为n_0，度为 2 的非叶结点数为n_2，则$n_0=n_2+1$
- 对于完全二叉树，根结点若为 0，则一个结点i的左右子女为$2i+1$与$2i+2$，父结点为$(i-1)/2$

二叉树的存储表示

完全二叉树可使用数组存储，在不越界的情况下，结点 i 的左右子女为 $2i+1$ 与 $2i+2$，父结点为 $(i-1)/2$

一般二叉树若使用数组存储，在二叉树不平衡的情况下，空间利用率非常低，使用链接表示可以解决这个问题

二叉树的链接表示：每个结点包含 3 个域，data 存储数据，left 指向左子树根结点，right 指向右子树根结点，也可增加 parent 域，指向父结点，这样的被称为三叉链表

二叉树的遍历

遍历：遵从某种次序，遍访二叉树的所有结点，使得每个结点都被访问一次且仅被访问一次

- 前序遍历：先访问根结点，再访问左子树，再访问右子树，可记为 VLR
- 中序遍历：先访问左子树，再访问根结点，再访问右子树，可记为 LVR
- 后序遍历：先访问左子树，再访问右子树，再访问根结点，可记为 LRV

递归实现：关键在于访问操作与递归调用的次序关系

后序遍历的应用

求树的结点数：先求左右子树的结点数，求和加一后便是整棵树的结点数

求树的高度：递归求出左右子树的高度，取其大者加一作为树的高度

前序遍历的应用

复制构造：复制根结点，再复制左右子树

判断树是否相等：先判断根结点，再递归判断左右子树

前序序列建立二叉树：要求输入的序列为二叉树前序序列，且空结点用特殊字符（如@）表示

- 若读入的字符为非特殊字符，则递归调用建立左右子树
- 否则返回空，封闭叶结点

二叉树的非递归遍历

非递归前序遍历

- 初始化栈 s，压入 NULL，指向根结点的指针 p
- 访问 p
- 将 p 的右子树压栈

- 若 p 的左子树非空则将 p 指向 p 的左子树
- 否则 p 指向栈顶结点，弹栈
- p 为空时算法结束

或是

- 指向根结点的指针 p，初始化栈 s，将 p 压栈
- 当栈非空时，p 指向栈顶元素，弹栈，访问 p
- p 右子树非空则压栈，左子树非空则压栈（左右颠倒保证弹出时左子树先于右子树）
- s 为空时算法结束

非递归后序遍历

需要为每个结点增加一个 tag，一同压入栈

- 初始化栈 s，指针 p 指向根结点
- 若 p 不为空，将 tag 置 L，将 p 指向的结点压入栈，然后 p 指向其左子树
- 若 p 为空，p 指向栈顶元素，弹栈
  - 若 tag 为 L，将 tag 置 R，将其与被弹出的结点再次入栈，然后 p 指向其右子树
  - 若 tag 为 R，访问结点，将 p 置为空
- s 为空且 p 为空时算法结束

非递归中序遍历

- 初始化栈 s，指针 p 指向根结点
- 若 p 非空或 s 非空
  - 若 p 非空，将其进栈，p 指向其左子树，循环直至 p 为空
  - 若 s 非空，出栈，访问当前根结点，然后将指针指向其右子树
- p 为空且 s 为空时遍历结束

层次序遍历

- 将根结点压入队列
- 当队列非空时，取出队头元素，访问，并将其所有非空子结点加入队列
- 队列空时算法结束

二叉树的计数

给定前序/后序序列与中序序列可以唯一地确定一颗二叉树

给定前序序列，中序序列可能的种数为对应的 Catalan 数

通过前序序列与中序序列确定二叉树：

- 前序的第一个元素为二叉树的根，在中序序列中寻找根，设为 k
- 从 前序+1 的位置开始对中序 0~k-1 的序列递归建左子树
- 从 前序+k+1 的位置开始对中序 k+1~n-1 的序列递归建立右子树

线索二叉树

线索的存储：利用空指针域，为了与左右子女区别，添加两个标志 ltag 与 rtag

寻找后继	rtag == 0	rtag == 1（后继线索）
right == NULL	无此情况	无后继
right != NULL	后继为当前右子树中序下的第一个结点	后继为右子女

寻找前驱	ltag == 0	ltag == 1（前驱线索）
left == NULL	无此情况	无前驱
left != NULL	前驱为当前左子树中序下的最后一个结点	前驱为左子女

树与森林

树的存储表示

树的每个结点分支数不等，常用的表示方法有四种

- 广义表表示法：叶结点对应原子，根结点对应表头，非叶结点对应子表
- 父指针表示法：用顺序表存储结点，每个结点有两个域，data 为数据，parent 为父指针
- 子女链表表示法：为每个子女设置一个指针，若子女数（结点的度）相差较大，为了兼容只能选择最大的度数 d，共占用空间 $n \times d$，而树中只有 $n-1$ 条边，造成了大量的空间浪费
- **子女-兄弟链表表示法**：每个结点由三个域组成，data 存储数据，child 指向第一个子女结点， sibling 指向下一个兄弟结点

森林和二叉树的转换

森林可转换为二叉树，因为每颗树的根结点都没有右兄弟，将多个根结点的右兄弟域依次链接起来，便得到了二叉树

树也可转为二叉树，左子树为树的子树森林，兄弟间连线，同时抹去除最左结点以外结点与上层结点的关系

树和森林的遍历

树的深度优先遍历

设树 $T=\{r,T_1,T_2...T_m\}$

先根（与树对应二叉树的**前序遍历**结果相同）：

- 若树为空，返回
- 否则访问根结点 r
- 依次遍历子树 $T_1,T_2...T_m$

后根（与树对应二叉树的**中序遍历**结果相同）：

- 若树为空，返回
- 否则依次遍历子树 $T_1,T_2...T_m$
- 访问根结点 r

树的广度优先遍历

- 将根结点进队列
- 队列非空时，取出队头元素，访问，指针指向其子女结点，将其所有兄弟结点加入队列
- 队空时算法结束

森林的深度优先遍历

设森林 $F=\{T_1,T_2,T_3...T_m\}$，每个元素都是一颗树

先根（与森林对应二叉树的**前序遍历**结果相同）：

- 若森林为空，返回
- 否则访问森林的根结点（第一颗树的根结点）
- 先根遍历第一根树的子树森林
- 先根遍历除第一颗树以外的森林 $T_2,T_3...T_m$

后根（与森林对应二叉树的**中序遍历**结果相同）：

- 若森林为空，返回
- 否则后根遍历第一根树的子树森林
- 访问森林的根结点（第一颗树的根结点）
- 后根遍历除第一颗树以外的森林 $T_2, T_3 \dots T_m$

森林的广度优先遍历

- 将所有树的根结点进队列
- 队列非空时，取出队头元素，访问，指针指向其子女结点，将其所有兄弟结点加入队列（即将其所有子女加入队列）
- 队空时算法结束

堆

堆：将一个关键码的集合 K 按完全二叉树的顺序存储在一个一维数组，并能保证 $K_i \leqslant K_{2i+1} \& K_i \leqslant K_{2i+2}$，则称其为最小堆（最大堆定义同理），堆的二叉树形式表现为任意结点的关键码小于其左右子结点的关键码，位于堆顶（二叉树根）的结点关键码是最小的

堆的下滑调整：

- 结点 i，取其左右子女关键码较小的一方（记为 j），将其与 i 的关键码比较（此时假定 i 的左右子树均已为最小堆）
  - 若 i < j 或 j = m（已经到达底部），不用调整，返回
  - 否则 交换 i 和 j 的位置，并将 j 的值赋给 i，再次比较其关键码与左右子女的关键码

堆的上浮调整：

- 将结点 i 的关键码与父结点 j 的关键码比较
  - 若 i > j 或 j = 0（到达顶部），停止比较，退出
  - 否则交换 i 与 j，并将 j 的值赋给 i，再次比较其关键码与父结点关键码的大小

建立堆（从数组，设数组有 n 个元素）：

- 找到最后一个非叶结点的位置 $\frac{n-2}{2}$
- 从这里开始连续使用下滑调整方法直至调整至堆顶（每次调整完后将调整的位置减一）

从堆中删除元素时将最后一个元素移到堆顶的位置并使用下滑调整，向堆中添加元素的时候将元素插入末尾并使用上浮调整。

堆的操作时间复杂度均为对数级别

Huffman 树与其应用

路径

路径：从树种一个结点到另一个结点之间的分支 **路径长度**：路径上的分支条数。树的路径长度是从树的根结点到每个结点的路径长度之和

树的根结点到每个结点的路径是唯一的

n 个结点的树的路径长度下界为：0,1,1,2,2,2,2,3,3,3,...该数列的和，即 $\sum_{i=0}^{n-1}\lfloor log(i+1)\rfloor$

扩充二叉树：假设有 n 个权值的集合 W，有 n 个叶结点的二叉树 T，将每个权值分别赋给 T 的叶结点，则将 T 称为权值为 W 的扩充二叉树，有权值的叶结点称为外结点，没有权值的分支结点称为内结点

带权路径长度(WPL)：路径长度与扩充二叉树外结点权值的乘积的和

WPL 最小的树称为最优二叉树

WPL 最小的二叉树应该是权值大的外结点离根结点近的树，即**Huffman 树**

Huffman 算法：

- 根据 n 个权值构造有 n 颗树的森林，每棵树只有一个根结点，其值为权值
- 在其中选择根结点权值最小的两棵树，将其作为左右子树构造一颗树（其中权值较小的作为左子树），根结点权值为两个子树根结点权值的和（有多个选择时选择结点数少的树），将这两颗树删除，将生成的新树加入集合
- 重复直至只有一棵树，此时生成的即为最优二叉树（Huffman 树）

Huffman 树的应用：编码

- 将字符出现的频率作为权，构建 Huffman 树
- 由根结点下行，左分支标记为 0，右分支标记为 1，为每个叶结点产生前缀编码（每个码字不可能是其他码字的前缀），此变长编码平均编码长度最小