第 8 章 树及其应用

树是无环连通图，是一种重要的非线性数据结构。树结构主要用于描述具有层次关系的数据组织与管理，如文件目录系统、数据库索引、路由算法与决策模型。树结构也是数据压缩、编码优化、搜索排序与语法分析的核心工具，在网络拓扑、最小生成树、机器学习决策树等实际场景中发挥关键作用，能够高效实现数据的快速查找、动态维护与层次化处理。

8.1 无向树

无向树是一个连通、无环、无向的简单图，有向树是指边具有方向的树，是一个有向图。无向树任意两点之间有且仅有唯一一条简单路径，去掉任意一条边就会不再连通，增加任意一条边就会出现回路。无向树常用于描述层次结构、生成树、数据组织、网络拓扑等场景，是优化、搜索、遍历的重要模型。

8.1.1 无向树的定义及性质

1）无向树、平凡树与森林

定义 8.1：无向树、平凡树与森林

若无向图 $G=(V,E)$连通且无简单回路（不包含环），则称 G 为一棵无向树。

仅含单个顶点、无边的平凡图称为平凡树。

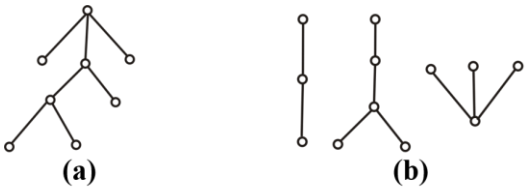
森林是每个连通分支均为无向树的无向图。

非平凡树中度数为 1 的顶点称为树叶。

在非平凡树中，度数 $\geqslant 2$ 的顶点称为分支点（内部顶点）。

在无向树中，不存在任何简单回路（即无圈结构），且任意两个不同顶点之间存在且仅存在一条简单路径。若一个无向图不含简单回路且连通分支数大于 1，则其每个连通分支都是一颗树，这些互不相交的树共同构成森林。

例如，图（a）是树，图（b）是森林。



2）树的等价定理

定理 8.1：树的等价刻画定理（六个等价命题）

设 $G=\langle V,E\rangle$ 是 n 阶无向图，有 m 条边，则以下命题相互等价：

（1）G 连通且不含圈（即 G 是一棵树）。

（2）G 中任意两个顶点之间存在唯一的一条路径。

（3）G 连通，且满足 $m=n-1$。

（4）G 不含圈，且满足 $m=n-1$。

（5）G 不含圈，但在 G 中任意两个不相邻的顶点之间添加一条边，会恰好产生一个简单圈。

（6）G 连通，并且 G 中每条边都是桥。

证明：设 $G=\langle V,E\rangle$ 为 n 阶无向图，m 为边数。

(1) ⇒ (2)

由（1）知 G 连通且无圈，任意两顶点之间至少存在一条路径。若某两顶点之间存在两条不同路径，则这两条路径必构成回路，与无圈矛盾，因此(2)成立。

(2) ⇒ (3)

由（2）知，G 中任意两点间有唯一路径，显然 G 连通。对顶点数 n 归纳证明 $m=n-1$。

$n=1$ 时，$m=0$，成立。

假设 $n\leqslant k$ 时 $m=n-1$。当 $n=k+1$，任取一边 $e=(u,v)$，删除 e 后 G 分成两个连通分支，设顶点数分别为 n_1,n_2，边数分别为 m_1,m_2。由归纳假设 $m_1=n_1-1$，$m_2=n_2-1$，于是 $m=m_1+m_2+1=(n_1-1)+(n_2-1)+1=n-1$ 故连通且 $m=n-1$，因此(3)成立。

(3) ⇒ (4)

由（3）知 G 连通且 $m=n-1$，只需证 G 无圈。假设 G 含圈，则删去圈上一边得连通图 G'，边数 $m'=m-1=n-2$。继续去圈直至得到一棵生成树，其边数必为 $n-1$，与 $n-2$ 矛盾。因此 G 无圈且 $m=n-1$，(4)成立。

(4) ⇒ (5)

由（4）知 G 无圈且 $m=n-1$。任取两个不相邻顶点 u,v，加边 (u,v)。若新图仍无圈，则边数 $m+1=n$，与无圈图边数不超过 $n-1$ 矛盾。故必产生圈。若产生两个不同圈，则去掉新边后 G 中仍有圈，矛盾。因此恰好产生一个简单圈，(5)成立。

(5) ⇒ (6)

由（5）知 G 无圈，且加任意非邻边恰产生一圈。

先证 G 连通：若 G 不连通，则在不同分支间加边不会产生圈，矛盾。

再证每条边都是桥：任取一边 e，若 e 不是桥，则删去 e 仍连通，$G-e$ 无圈。在 $G-e$ 中加回 e 应恰产生一圈，但原图 G 无圈，矛盾。故 G 连通且每条边都是桥，(6)成立。

(6) ⇒ (1)

由（5）知 G 连通且每条边都是桥。若 G 含圈，则圈上任意一边都不是桥，与每条边都是桥矛盾。故 G 连通且无圈，即 G 是树， (1)成立。

至此，六条命题两两等价，证明完毕。

树的 6 条等价命题并非“重复定义”，而是等价刻画。只要满足其中任意一条，就自动满足全部 6 条，并且该图一定是树。这 6 条性质将树朴素直观的概念，转化为一套强大可用的工具。在许多图论证明中，若仅使用原始定义会极为繁琐，而选用等价条件中的某一条，往往一句话即可完成证明。

3）无向树的树叶存在性定理

树叶存在性定理完善了树的拓扑结构特性， 补充树的等价判定条件，让树的拓扑特征更完整、判定更严谨，为后续图论相关推导（如生成树、边的性质）提供重要依据，避免抽象定义带来的理解偏差。

定理 8.2： 无向树的树叶存在性定理

任意一个 n 阶非平凡的无向树（即顶点数 $n\geqslant 2$ 的无向树），至少含有两片树叶（度数为 1 的顶点）。

证明：

设 T 是 n 阶无向树，$n\geqslant 2$，边数为 m。由树的性质知 $m=n-1$。

设 T 的顶点度数依次为 $d_1,d_2,\ldots,d_n$，由握手定理：$\sum_{i=1}^{n} d_i = 2m = 2(n-1)$

因为 T 是树且连通、非平凡，每个顶点度数满足 $d_i \geq 1$。假设 T 中至多一片树叶，即度数为 1 的顶点个数 ≤ 1，则至少有 $n-1$ 个顶点度数 ≥ 2。于是$\sum_{i=1}^{n} d_i \geq 1 + 2(n-1) = 2n-1$ ，

由握手定理$\sum_{i=1}^{n} d_i = 2n-2$ ，得到 $2n-1 \leq 2n-2$，矛盾。

因此假设不成立，即 T 中至少有两个度数为 1 的顶点，也就是至少有两片树叶。

证毕。

例 8.1: 已知无向树 T 中, 有 1 个 3 度顶点, 2 个 2 度顶点, 其余顶点全是树叶。 试求树叶数, 并画出满足要求的非同构的无向树。

解:设有 x 片树叶，则顶点数 $n=1+2+x=x+3$，边数 $m=n-1=x+2$。

由握手定理：$\sum d(v)=3\times1+2\times2+1\times x=x+7$，又 $\sum d(v)=2m=2(x+2)$，于是 $x+7=2x+4$ 解得 $x=3$。

即树 T 有 3 片树叶，顶点总数 $n=6$，边数 $m=5$，度序列为：3,2,2,1,1,1。

非同构无向树构造：

树的本质是连通、无圈、边数 = 顶点数 - 1，构造非同构树的核心的是围绕 “度数最高的顶点（分支点）” 展开，通过调整分支连接方式，区分不同构的树，避免重复构造。

（1）梳理度序列，确定核心参数与分支点

设核心分支顶点 v_0 度数为 3，2 个 2 度中间分支点为 v_1 和 v_2，3 个 1 度树叶是 v_3、v_4 和 v_5，总顶点数 $n=6$，边数 $m=n-1=5$；

（2）围绕核心分支点，分配度数连接

核心分支点 v_0 需连接 3 个节点（满足度数要求），优先分配连接对象（优先连接分支点，再连接树叶，避免提前出现环）；

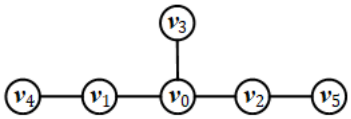
先将 v_0 与 2 个中间分支点（v_1、v_2，度数均为 2）连接，此时 v_0 已用 2 个度数，剩余 1 个度数需连接 1 个树叶（如 v_3）；

中间分支点 v_1 已与 v_0 连接（用 1 个度数），剩余 1 个度数需连接 1 个树叶（如 v_4）；

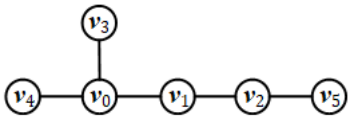
中间分支点 v_2 已与 v_0 连接（用 1 个度数），剩余 1 个度数需连接最后 1 个树叶（如 v_5）。

（3）区分非同构情况，构造不同连接方式

构造 1：v_0 直接连接 v_1、v_2、v_3；v_1 连接 v_4；v_2 连接 v_5（2 个中间分支点均直接与核心点连接，无串联）；



构造 2：v_0 直接连接 v_1、v_3、v_4；v_1 连接 v_2；v_2 连接 v_5（1 个中间分支点与核心点连接，另 1 个中间分支点与前者串联，再连接树叶）。



（4）验证去重，确认符合要求

验证每棵树：均满足 “连通、无圈、边数 = 5、度数序列匹配”，最终得到 2 棵非同构树，均满足度序列（3,2,2,1,1,1）的要求。

4）非同构无向树构建

构建非同构无向树，首先要明确树的顶点数、边数，以及各顶点的度数要求，确保满足树的核心拓扑特性- 连通性、无环性，且边数 = 顶点数 - 1。最大度构建法是最高效、最易操作的方法，该方法通过锁定最大度节点（核心分支点），明确不同连接方式的差异，既能契合树的拓扑特性，又能避免构造重复，确保每棵树的拓扑结构唯一。按“分支点数量”分类构建方法，则以度数≥2 的分支点数量为分类依据，枚举不同数量分支点下的连接方式，保证树满足连通、无环要求，且拓扑结构不重复。按“连接层级”分类构建方法，是以节点连接的层级关系（核心节点直接连接 / 中间节点串联连接）为区分标准，构造不同拓扑层级的树，避免同构重复。

例 8.2: 画出所有 6 阶非同构的无向树。

解：按最大度构建 6 阶非同构无向树，核心原因的是最大度直接决定树的拓扑结构，是区分不同树结构的关键依据。

设 6 阶无向树为 T，顶点数 n=6，则边数 m=n−1=5。由握手定理，总度数$\sum_{i=1}^{6} d(v_i) = 2m = 10$ 。

枚举所有合法最大度及对应树结构：

当最大度 Δ=5，对应度序列为(5,1,1,1,1,1) ，1 个核心节点（度 5），连接 5 个叶子节点（1 度），仅 1 种结构（星形树），结果如图（1）。

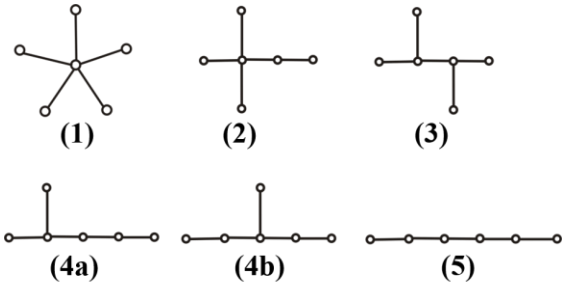
当最大度 Δ=4，对应度序列为(4,2,1,1,1,1)，核心节点（度 4）连接 1 个 2 度中间节点，再连接叶子，仅 1 种结构，结果如图（2）。

当最大度 Δ=3（核心节点），对应 2 种结构：

结构 1：度序列 (3,3,1,1,1,1)，两个 3 度节点串联，各带 1 片叶子，1 种结构，结果如图（3）。

结构 2：度序列 (3,2,2,1,1,1)，有两种结构，两个 2 度节点串联得到结构图（4a）；两个 2 度节点直接连接核心节点，得到树结构图（4b）。

当最大度 Δ=2（无高次分支点），对应度序列为 (2,2,2,2,2,2)，仅 1 种结构（直线链状），得到树结构图（5）。



8.1.2 生成树

生成树满足树的所有核心属性，是树的子类。生成树不能独立存在，必须依附于一个连通图（母图），且包含母图的全部顶点。生成树作为母图的极小连通子图，是树与连通图的桥梁，衔接树的性质与连通图的特点，可广泛用于通信网络、电路连接等设计，能保证网络连通且无冗余（无圈），减少线路和节点冗余。

1）生成树及其余树

定义 8.2：生成树

设 $G=\langle V,E\rangle$ 为无向连通图。

（1）若 $T=\langle V,E_T\rangle$ 是 G 的生成子图，且 T 为树，则称 T 是 G 的一棵生成树。

（2）生成树 T 中的边，称为 T 的树枝。

（3）属于原图 G、但不属于生成树 T 的边，称为 T 的弦。

（4）由生成树 T 的全体弦构成的边集所导出的子图，称为 T 的余树，记作 $\overline{T}$。

例如，图中黑边构成生成树，红边构成余树。

定理 8.3：连通图的生成树存在性定理

任何无向连通图都至少存在一棵生成树。

证明：（破圈法）

设 G 为一无向连通图。若 G 中不含圈，则 G 本身就是一棵生成树。否则，在 G 中任取一个圈，并删除该圈上任意一条边，所得图仍然连通。重复这一过程，直到图中不再含有圈为止，最终得到的连通无圈图即为原图 G 的一棵生成树。

推论 1：边数与度数关联推论

设 G 是 n 阶无向连通图，且有 m 条边，则 $m\geq n-1$。

边数与度数关联推论强调了连通图必须至少有 $n-1$ 条边，少于这个边数就不可能连通。生成树是连通图的“最小连通骨架”，边数恰好为 $n-1$。反过来说，如果一个图连通且边数恰好为 $n-1$，那它本身就是一棵树，就是自己的生成树。该推论从边数下界的角度，再次印证了“连通图一定能找到生成树”的合理性。

推论 2：生成树与原连通图关联推论

设 G 是 n 阶无向连通图，且有 m 条边，T 是 G 的一棵生成树，则 T 的余树中含有 $m-n+1$ 条边。

生成树与原连通图关联推论表明，原图中比生成树多出的边均为冗余边（弦），这些边是构成原图中回路的关键。生成树正是通过删除这类冗余边得到的连通无圈图，余树的边数就是从原图中需要删去的边数。

由此可见，原图中回路的数量与复杂程度，可通过余树的边数直观体现。余树边数越多，说明原图含有的回路越多，整体结构也越复杂。该推论精确刻画了原图、生成树与余树三者之间的边数关系，将生成树与原图的结构特征紧密关联。

2）加权图与最小生成树(MST)

定义 8.3：带权图及其生成子图

设图 $G=\langle V,E\rangle$，其中 V 为顶点集，E 为边集。给图 G 的每一条边 $e\in E$，都附加一个实数 $w(e)$，这个实数 $w(e)$称为边 e 的权。

图 G 连同附加在其边上的权，共同构成带权图，记作 $G=\langle V,E,W\rangle$，其中 W 为边权的集合，即 $W=\{\, w(e) \mid e\in E\}$。

设 H 是带权图 G 的生成子图，H 所有边的权之和称为 H 的权，记作 $W(H)$，且$w(H) = \sum_{e\in E(H)} w(e)$（$E(H)$为生成子图 H 的边集）。

定义 8.4：最小生成树

设 $G=\langle V,E,W\rangle$为带权连通图，若生成树 T 满足对 G 的任意一棵生成树 T'，都有 $W(T)\leq W(T')$，则称 T 为 G 的最小生成树。简单来说，最小生成树是带权图 G 中，权值最小的生成树，其权值为所有生成树中权值的最小值。

3）寻找最小生成树算法-Kruskal 算法

设 $G=\langle V,E,W\rangle$为 n 阶无向连通带权图。

（1）将图 G 中不含环的所有边，按权值从小到大排序，记为 $w(e_1)\leqslant w(e_2)\leqslant\cdots\leqslant w(e_m)$

（2）初始化生成树边集 $T\leftarrow\emptyset$，已选边数计数器 $k\leftarrow 0$，边的遍历序号 $i\leftarrow 1$。

（3）依次检查当前边 e_i：若将 e_i 加入 T 后不形成回路，则令 $T\leftarrow T\cup\{e_i\}$,$k\leftarrow k+1$

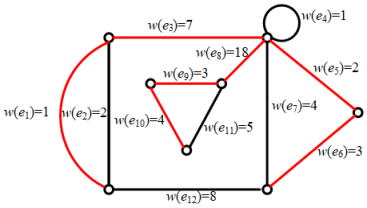
（4）若 $k<n-1$，令 $i\leftarrow i+1$，转回步骤 （3）继续检查下一条边；当 $k=n-1$ 时，算法终止，此时 T 即为图 G 的一棵最小生成树。

除 Kruskal 避圈法外，寻找最小生成树还有多种经典且规范的方法：Prim 算法（普里姆算法）以顶点为核心，从任意顶点出发，每次选择权值最小且能连接当前生成树与外部顶点的边，逐步扩展至包含所有顶点，适合边稠密的图。

Borůvka 算法（博罗夫卡算法）通过合并各连通分量的最小关联边，逐步缩小连通范围，可并行化处理。破圈法与 Kruskal 避圈思路相反，通过删除图中每个圈上的最大权边，直至得到无圈且连通的生成树。

反向删除算法则按边权从大到小删除，仅保留能维持图连通的边，最终得到最小生成树。这些方法与 Kruskal 算法本质一致，均以“获取权值最小的连通无圈子图”为核心，可根据图的顶点密度、边的分布情况选择合适的算法。

例 8.3: 找到图的最小生成树



解：顶点数 8 个，因此最小生成树需要选 8−1=7 条边。

（1）边按权值升序排序（排除自环 e_4）

权值序列：$1(e_1)\rightarrow 2(e_2)\rightarrow 2(e_5)\rightarrow 3(e_6)\rightarrow 3(e_9)\rightarrow 4(e_7)\rightarrow 4(e_{10})\rightarrow 5(e_{11})\rightarrow 7(e_3)\rightarrow 8(e_{12})\rightarrow 18(e_8)$

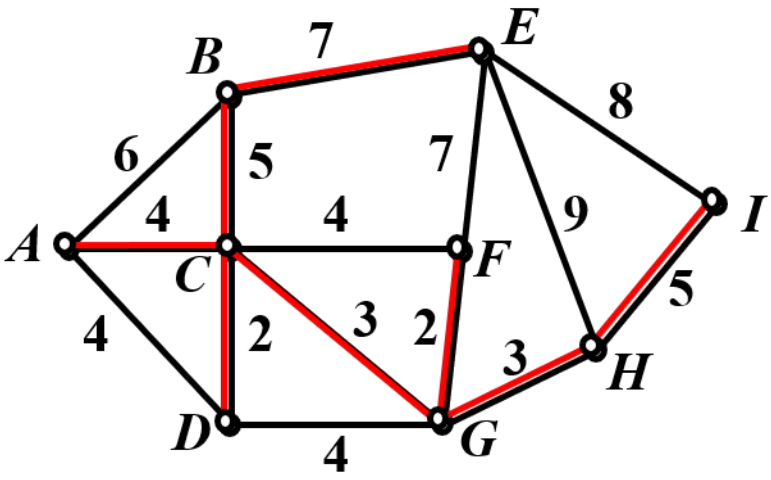
（2）贪心选边（不形成环就加入，目标选 7 条边）

初始化生成树边集 $T\leftarrow\emptyset$，按照权值序列选边，$e_2(2)$、$e_7(4)$、$e_{11}(5)$ 、$e_{12}(8)$会与 T 中已选中的边形成环，不加入 T，最终结果选定的边集为$\{e_1, e_5, e_6, e_9, e_{10}, e_3, e_8\}$。

总权值和 $W(T)=1+2+3+3+4+7+18 = 38$。

例 8.4：某电信公司有如图所示的光纤网络覆盖需求。图中给出了建筑物之间可能铺设的光缆线路及其长度（单位：米）。

问题：在必须保证整个网络连通的前提下，应如何选择光缆线路，才能使光缆总长度最短？



解：

这是一个典型的最小生成树（MST）应用问题。可以使用 Kruskal 算法来求解。

（1）问题分析

目标：铺设光缆总长度最短。

约束：保证所有建筑物（顶点）连通，且线路无环（避免冗余投资）。

算法：Kruskal 算法。

顶点数：9 个（A, B, C, D, E, F, G, H, I）。

需边数：9−1=8 条。

（2）边按权值升序排序：

2 (CD) → 2 (FG) → 3 (CG) → 3 (GH) → 4 (AC) → 4 (AD) → 4 (CF) → 4 (DG) → 5 (BC) → 6 (AB) → 7 (BE) → 7 (EF) → 8 (EI) → 9 (EH)

（3）贪心选边（避圈）

依次选取边，跳过与已选择边构成圈的边，直到选出 8 条边连通所有顶点：

准确避圈后选定的边集为 {CD, FG, CG, GH, AC, AD, BC, EI}（8 条）

总权值：2+2+3+3+4+4+5+8=31

最优光缆线路为上述 8 条，总长度 31 米。

（4）结论

最优光缆线路为上述 8 条，总长度 31 米。

8.2 有根树及其应用

有根树是有唯一源头顶点、边带有统一方向的特殊有向树，属于层次化的有向图结构。它以唯一入度为 0 的顶点作为树根，其余每个顶点入度均为 1，任意顶点到根都存在唯一一条有向通路，整体无环且连通。有根树具有清晰的层次关系，顶点可区分为树根、内点与树叶，从根到叶形成严格的层级通路，删去任意一条边会破坏连通性，新增一条边则必然产生环。有根树是表达层次结构、家族关系、文件系统、语法树、搜索树与决策流程的核心模型，广泛用于遍历、查找、排序与递归结构设计。

8.2.1 有根树及其分类

1）根树的定义

定义 8.5：有向树

一个有向图，如果略去方向后是一棵无向树，就称为一棵有向树。

定义 8.6：根树与树根

有一个顶点入度为 0, 其余的入度均为 1 的非平凡的有向树称为根树，这个入度为 0 的顶称为树根。

定义 8.7：树叶、内点与分支点

有向树中入度为 1, 出度为 0 的顶点称为树叶；

有向树中入度为 1，出度 $\geqslant 1$ 的顶点称为内点；

根与所有内点统称为分支点。

定义 8.8：层数与树高

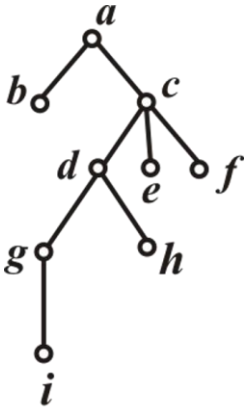
顶点 v 的层数是从树根到 v 的通路长度；

有向树中顶点的最大层数称为树高。

2）根树的画法

根树的画法没有强制的几何坐标标准，根在上、边向下、同层对齐、箭头可省，自上而下的画法是约定俗成的标准形式，自左向右画法（横向树）、带箭头严格画法和嵌套层次画法（缩进表示法）也是可选方法。

例 8.5：分析树的结构



解：在树中， a 是树根， b,e,f,h,i 是树叶， c,d,g 是内点， a,c,d,g 是分支点。

a 为 0 层，1 层有 b,c，2 层有 d,e,f， 3 层有 g,h， 4 层有 i， 树高为 4。

家族树是将根树结构直接应用于层次关系描述的典型模型。它借用亲属关系术语（根为祖先、分支点为父辈、树叶为后代等），把抽象的根树转化为直观、易理解的家谱式结构，用于清晰表达顶点之间的从属、辈分与层级关系。这种应用视角极大简化了对根树中路径、层数、祖先后代、父子兄弟等概念的理解与交流，让根树从纯数学结构落地为可直接使用的实用模型。

3）有序树与无序树

根树按子节点是否有固定左右顺序分为无序树和有序树，无序树只关心父子结构，不关心兄弟节点谁左谁右，调换子节点位置，仍视为同一棵树。有序树每个节点的子节点规定从左到右的次序，调换子节点位置，视为不同的树。二叉树、表达式树、语法树都属于有序树。

4）根树的分类

根据根树中节点和子节点数、子节点的顺序性、树的规则性以及树的完全性对根树进行了细分。

r 元树(*r* 叉树)：若一棵根树中，每个分支点至多有 *r* 个儿子，则称该根树为 *r* 元树。

r 元正则树(满 *r* 叉树)：若一棵根树中，每个分支点恰好有 *r* 个儿子，则称该根树为 *r* 元正则树。

r 元完全正则树(完美 *r* 叉树)：若一棵根树满足：①是 *r* 元正则树；②所有树叶的层数都相同；则称该根树为 *r* 元完全正则树。

r 元有序树：若一棵 *r* 元树 是有序根树，则称其为 r 元有序树。

r 元正则有序树：若一棵 *r* 元正则树是有序根树，则称其为 *r* 元正则有序树。

r 元完全正则有序树：若一棵 *r* 元完全正则树是有序根树，则称其为 *r* 元完全正则有序树。

8.2.2 最优树与霍夫曼算法

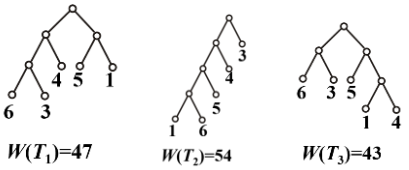
最优二叉树（最优树 / 哈夫曼树）是带权叶子节点构造的二叉树中，带权路径长度 WPL 最小的树。在编码应用中，最优二叉树可使电文总长最短、传输效率最高；在无损压缩算法设计中，能够减少文件存储体积；在决策、判断、称重等问题中，可使平均比较次数与执行代价达到最小。

1）最优二叉树

定义 8.9：最优二叉树

设二元树 T 有 t 片树叶 $v_1, v_2, \ldots, v_t$，其权分别为 $w_1, w_2, \ldots, w_t$。称$W(t)=\sum_{i=1}^{t} w_i l(v_i)$为 T 的带权路径长度（权），其中 $l(v_i)$是树叶 v_i 的层数。在所有具有 t 片带权树叶 $w_1, w_2, \ldots, w_t$ 的二叉树中，带权路径长度最小的二叉树称为最优二叉树（也称为哈夫曼树）。

例如，下图中，$W(T)=43$ 为二元树 T 的最优二叉树。



$W(T_1)=47$　$W(T_2)=54$　$W(T_3)=43$

2）最优二元树构造算法与静态哈夫曼前缀码

算法 8.2：最优二元树构造算法（哈夫曼算法）

给定一组互不相同的实数权值 $w_1, w_2, \ldots, w_t$，构造以这些权值为叶子结点、带权路径长度 WPL 最小的最优二元树。

算法步骤：

（1）根据给定权值，生成 t 个孤立叶子结点，构成初始森林。

（2）在当前森林所有结点中，选取权值最小的两个结点。

（3）新建一个父分支结点，将选出的两个结点作为该结点的左右子结点，新结点权值等于两个子结点权值之和。

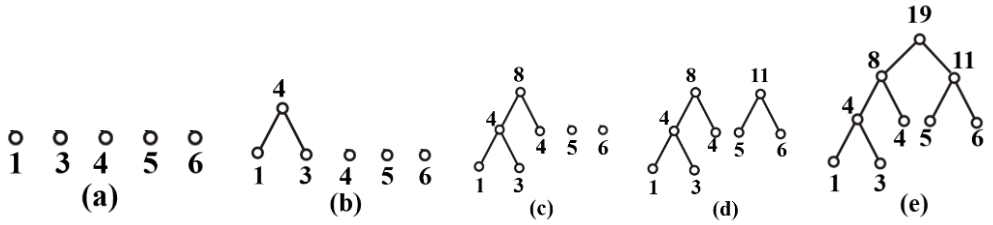
（4）将新建分支结点加入森林，移除原来两个被合并的结点。

（5）重复步骤 （2）~（4），直到森林中仅剩下一棵二叉树、一个根结点为止。

对哈夫曼算法构造出的最优二元树从根到叶，左 0 右 1 依次取值，可生成无前缀性质的静态哈夫曼前缀码。在字符权值（出现频率）预先已知且固定不变的条件下，该算法构造的最优二元树带权路径长度最小，对应编码平均码长达到最优最短。静态哈夫曼前缀码广泛应用于文件无损压缩、图像传输、网络数据通信等领域。

例 8.6：求以 1,3,4,5,6 为权的最优 2 叉树, 并计算它的权。

解： (a)到(e)是哈夫曼算法的计算过程，$W(T)=4+8+11+19=42$。



8.2.3 最优前缀码

在二进制编码传输中，普通不等长编码易出现前缀重叠，导致译码歧义。等长编码又容易造成码元冗余、传输效率低下。为了使编码无歧义且整体平均码长最短，信息编码理论引入前缀码与最优前缀码理论。该理论证明了静态哈夫曼前缀码是全局平均码长最短的编码，在所有合法前缀码中性能达到最优。

1）前缀码与二进制前缀码

定义 8.10：前缀码与二进制前缀码

设$\beta=\alpha_1\alpha_2\ldots\alpha_{n-1}\alpha_n$是长度为 n 的符号串, 称$\alpha_1\alpha_2\ldots\alpha_j$（$1\le j\le n$）为符号串$\beta$的前缀。

若非空符号串集 $B=\{\beta_1, \beta_2, \ldots, \beta_m\}$中任意两个符号串$\beta_i, \beta_j$互$(i\ne j)$互不为前缀, 则称 B 为前缀码。

若前缀码中每一位符号仅取自集合{0,1}，则称该前缀码为二元前缀码。

二元前缀码对应二叉树编码，从根到叶路径唯一，无译码歧义。

例如， {0,10,110, 1111}, {10,01,001,110}是 2 元前缀码； {0,10,010, 1010} 不是前缀码，符号串 0 是 010 的前缀，符号串 10 是 1010 的前缀，集合内存在互为前缀的元素，不满足前缀码定义。

2）二叉树与二元前缀码

任意一棵二叉树均可构造一组二元前缀码：

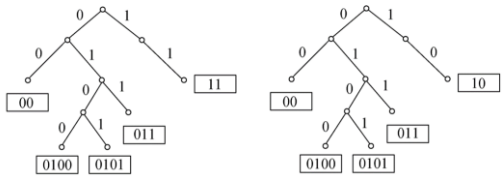
（1）对二叉树分支结点，左分支标记 0，右分支标记 1；

（2）仅有一条出边的单分支结点，该边可任意标记 0 或 1；

（3）从根结点到每一片叶子结点的路径标号依次拼接，所得二进制符号串集合，即为二元前缀码。

由二叉树生成的编码，叶子结点路径互不包含，因此集合内任意符号串互不互为前缀，天然满足前缀码定义。

例如：由下面二叉树可得前缀码{00,11,011,0100,0101}、{00,10,011,0100,0101}，均为合法二元前缀码。



同理，任意一组二元前缀码，也都可以对应一棵二叉树。

3）最优前缀码

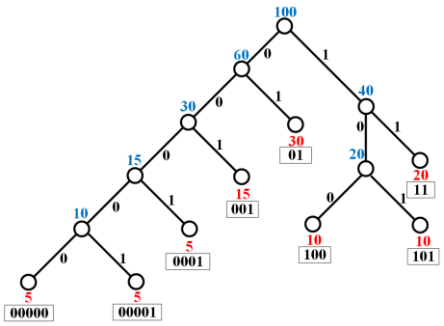
定义 8.11：最优二元前缀码

设字符集各字符对应权值（出现频率）为 $w_1, w_2, \ldots, w_m$，对应二元前缀码码长为 $l_1, l_2, \ldots, l_m$。在所有合法二元前缀码中，使得平均码长$L = \sum_{i=1}^{m} w_i l_i$ 取得最小值的二元前缀码，称为最优二元前缀码。

例 8.7：在通信中,设八进制数字出现的频率(%)如下: 0: 30, 1: 20, 2: 15, 3: 10, 4: 10, 5: 5, 6: 5, 7: 5。采用 2 元前缀码, 求传输数字最少的 2 元前缀码 (称作最佳前缀码), 并求传输 10000 个按上述比例出现的八进制数字需要多少个二进制数字？若用等长的 (长为 3) 的码字传输需要多少个二进制数字?

解：

（1）用 Huffman 算法求以频率(乘以 100)为权的最优 2 元树。设 w_1=5,w_2=5,w_3=5,w_4=10,w_5=10,w_6=15,w_7=20,w_8=30。



（2）最优 2 元树与八进制编码对应

哈夫曼树左支标 0，右支标 1，从树根走到叶子结点，一串 01 就是该八进制数的最优前缀码。使用频率越高的八进制数离根越近，其码长越短。

八进制数	0	1	2	3	4	5	6	7
使用频率	30%	20%	15%	10%	10%	5%	5%	5%
哈夫曼编码	01	11	001	100	101	0001	00001	00000
编码长度	2	2	3	3	3	4	5	5

定长编码	000	001	010	011	100	101	110	111

（3）根据给定的频率分布，计算传输 10000 位八进制数字所需的二进制位数：

计算加权平均码长 L=0.3×2+0.2×2+0.15×3+0.1×3+0.1×3+0.05×4+0.05×5+0.05×5=2.75；

传输 10000 个数字

采用哈夫曼码：10000×2.65=26500 位二进制

采用等长 3 位码：10000×3=30000 位二进制

采用最优前缀码节省的位数：30000-27500=2500，约占 8.33%。

8.2.4 有根树的遍历及其应用

有根树用来描述具有层次从属、分支递归关系的非线性离散结构模型，常用于因果推理、决策分类、嵌套算术表达式、文件目录结构、程序递归调用、层级组织结构等场景。有根树遍历的意义是将非线性树形结构转化为一维线性有序序列，按照规定次序不重复、不遗漏地访问所有节点，从而实现树的查询、数值计算、结构还原与拓扑分析。

1）二叉树的遍历方法

二叉树是特殊的二元有序有根树，树中每个结点至多拥有左、右两个子结点，且严格区分左子树与右子树，左右次序不可随意交换。通过二叉树的前序、中序、后序遍历，可实现表达式求值、有序数据检索、树形结构存储与层次化数据处理，是离散数学与数据结构中最基础、应用最广泛的树形模型。

定义 8.12：二叉树遍历

系统地访问二叉树的每一个顶点，且每个顶点恰好只被访问一次的有序访问过程，称为二叉树遍历。

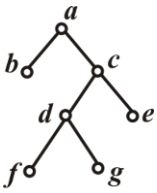
定义 8.13：二叉树的三种递归遍历次序

（1）前序遍历：先访问根结点，再递归遍历左子树，最后递归遍历右子树。

（2）中序遍历：先递归遍历左子树，再访问根结点，最后递归遍历右子树。

（3）后序遍历：先递归遍历左子树，再递归遍历右子树，最后访问根结点。

例如，二叉树



的遍历结果如下：

（1）前序遍历：a (b) (c (d (f) (g)) (e))，得到序列：a,b,c,d,f,g,e

（2）中序遍历：(b) a ((f) d (g)) c (e) ，得到序列：b,a,f,d,g,c,e

（3）后序遍历：(b) ((f)(g)d) (e)c a，得到序列：b,f,g,d,e,c,a

圆括号嵌套清晰划分每一棵子树，严格对应递归遍历规则。

2）算术表达式二叉树构造规则

表达式树是将算术运算嵌套结构，转化为递归有序二叉树结构，通过二叉树遍历完成表达式与前缀 / 中缀 / 后缀式的相互转换与求值运算。

定义 8.14：算术表达式二叉树递归构造规则

算术表达式二叉树是满足以下递归条件的二叉树：

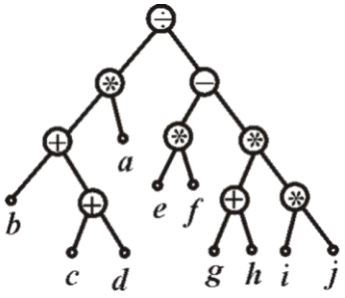
（1）所有参与运算的常数、变量，均放置在树的叶子结点上。

（2）表达式所有运算符均放置在树的内部分支结点。

（3）二元运算符对应的结点，恰有左右 2 个子结点，左子树对应左运算分量，右子树对应右运算分量。 减法、除法运算中，被减数、被除数固定位于左子树。

（4）一元运算符对应的结点，仅有唯一子结点，该子结点为根的子树即为该运算符对应的运算分量。

例如，下图是表示算式 $((b+(c+d))*a)\div((e*f)-(g+h)*(i*j))$的 2 叉树，该 2 叉有序树的中序行遍结果是原算式。



3）波兰符号法和逆波兰符号法

为解决二叉树遍历生成的中缀表达式存在歧义、前序遍历序列运算低效的问题，使表达式符号序列在计算机编译系统自动处理中消除歧义、提高实现效率、降低存储资源需求，由波兰数学家扬·卢卡西维茨提出的波兰（前缀）符号法，及澳大利亚计算机科学家查尔斯·哈姆林改进的逆波兰（后缀）符号法，共同成为适合计算机处理的表达式规范。

波兰/逆波兰符号从计算机易于处理的视角表示算术表达式符号序列，二叉树遍历是生成波兰 / 逆波兰符号的标准路径。

定义 8.15：波兰符号法（前缀符号法）

波兰符号法是一种无括号的算术与逻辑表达式表示体系，其核心规则基于“运算符前置”的固定顺序：

（1）单个运算对象（常数、变量或原子命题）本身即构成一个合法的波兰符号表达式。

（2）若 $\star$ 是二元运算符（如 +、- 、×、÷、∧、∨），E_1 和 E_2 分别是波兰符号表达式，则“$\star\, E_1\, E_2$” 也是合法的波兰符号表达式，其中 E_1 对应左运算分量，E_2 对应右运算分量。

（3）若 ¬ 是一元运算符（如否定、平方根），E 是波兰符号表达式，则 “$\neg\, E$” 是合法的波兰符号表达式。

（4）该符号法的表达式序列与算术 / 逻辑表达式二叉树的前序遍历结果完全一致，通过固定的 “运算符 - 运算对象” 顺序，无需额外括号即可唯一确定运算优先级与结合顺序。

定义 8.16：逆波兰符号法（后缀符号法）

逆波兰符号法是波兰符号法的改进形式，核心特征为“运算符后置”，是计算机系统中高效处理表达式的标准格式：

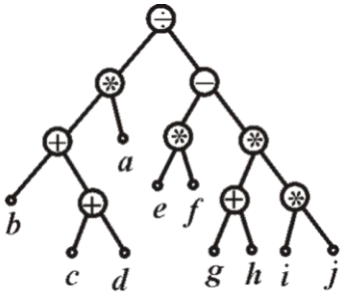
（1）单个运算对象（常数、变量或原子命题）本身即构成一个合法的逆波兰符号表达式。

（2）若 $\star$ 是二元运算符（如 +、- 、×、÷、∧、∨），E_1 和 E_2 分别是逆波兰符号表达式，则 “$E_1\ E_2\star$” 也是合法的逆波兰符号表达式，其中 E_1（左运算分量）、E_2（右运算分量）的顺序固定，适配减法、除法等非交换运算。

（3）若 ¬ 是一元运算符（如否定、平方根），E 是逆波兰符号表达式，则 “$E\neg$” 是合法的逆波兰符号表达式。

（4）该符号法的表达式序列与算术 / 逻辑表达式二叉树的后序遍历结果完全一致，可通过 “顺序扫描 + 栈结构” 实现无歧义快速求值，是编译器、计算器底层运算的核心表达式规范。

例如：图



所示表达式的波兰式与逆波兰式如下：

波兰表达式（前缀表达式）：$\div*+b+cda-*ef*+gh*ij$

逆波兰表达式（后缀表达式）：$bcd++a*ef*gh+ij**-\div$

波兰/逆波兰表达式的计算机处理方法：

对于波兰式表示的表达式，运算从根节点开始——先遇到运算符，再遇到对应的操作数。

对于逆波兰式表达式，运算从叶子节点开始，逐层向上计算，直至根节点。

知识扩展提示词

1. 树的等价刻画定理（六个等价命题）每一个都在完整的定义树吗？
2. 破圈法与 Kruskal 避圈都是寻找最小生成树算法，二者的主要区别在哪里？

第 8 章主要数学符号列表

序号	符号	含义	示例
1	$\bar{T}$	生成树 T 的余树	$\bar{T}$由生成树 T 的全体弦构成的边集所导出的子图
2	$G=\langle V,E,W\rangle$	带权图 G	W 为边权的集合 $W=\{\ w(e)\mid e\in E\}$