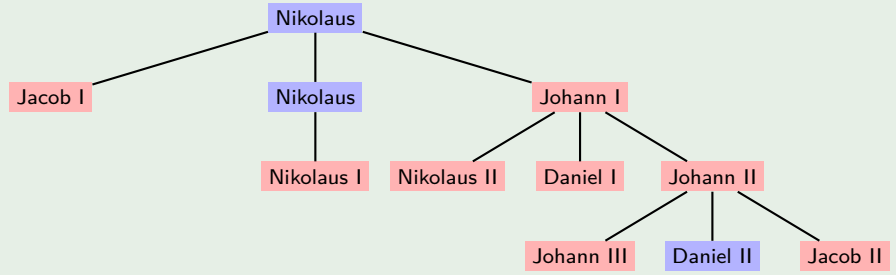
树的模型- 家族关系

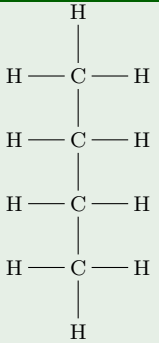
Example



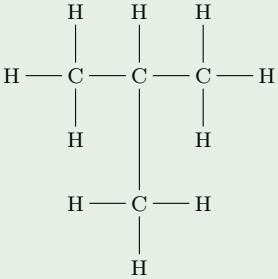
这是瑞士数学家中的著名家族-伯努利家族的族谱图。

树的模型- 分子化合物

Example



丁烷

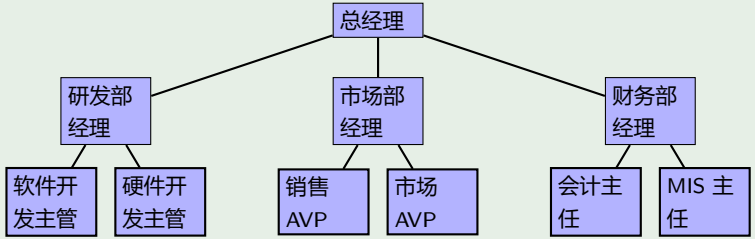


异丁烷

这是英国数学家凯莱用于表示饱和碳氢化合物 (形如 C_nH_{2n+2}) 的方法，从而发现了树。

树的模型- 组织机构

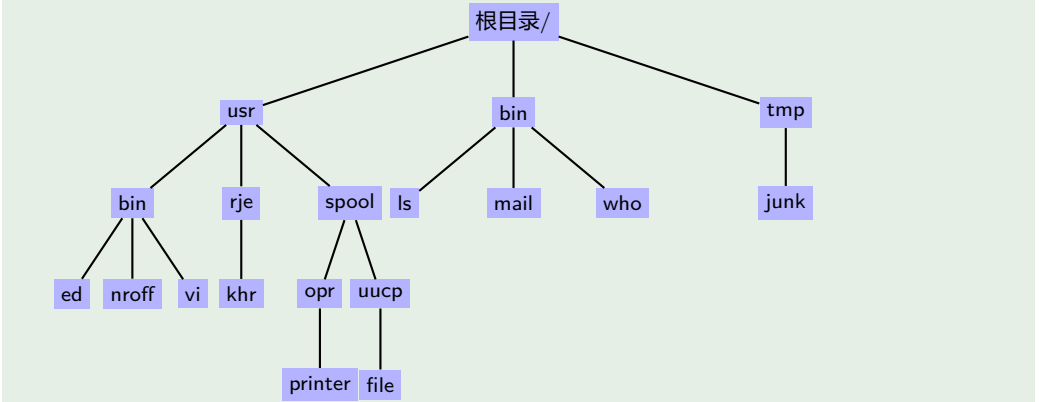
Example



大的组织机构的结构可以用树来建模，每个结点表示一个职务。

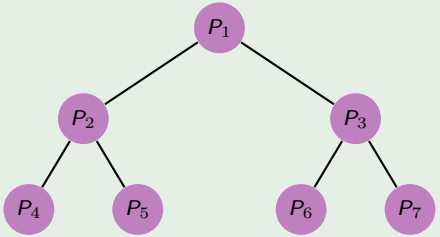
树的模型- 文件系统

Example



树的模型- 并行处理系统

Example

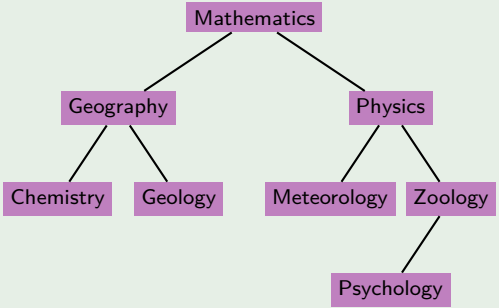


利用完全二元树可以把 $n = 2^k - 1$ 个处理器互联起来 (k 是正整数)，如图所示的带 7 个处理器的树形连接网络可以用三步对 8 个数求和。

树的应用- 二叉搜索树

Example

问题: 如何对列表中的元素进行存储，以便容易的找到元素的位置？

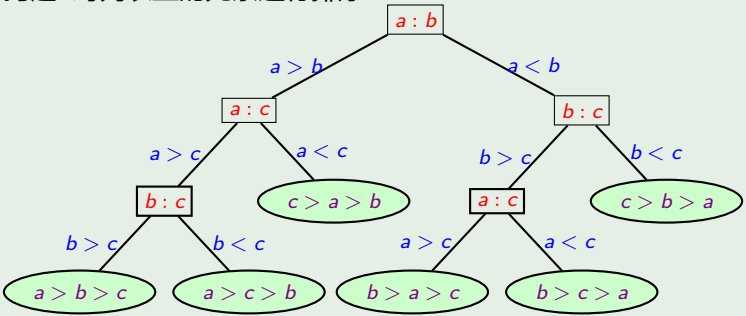


添加一个新的元素所需要的比较次数，最多等于从根到树叶的最长通路的长度。

树的应用- 决策树

Example

问题: 对列表里的元素进行排序



排序 n 个元素所至少需要 $\lceil log n! \rceil$ 比较，如图所示为对三个元素 a, b, c 排序的决策树。

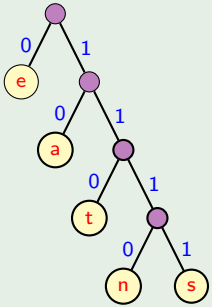
树的应用- 前缀码

Example

问题: 如何构造有效的不定长编码？

不定长编码需解决每个字母的位串在何处开始和结束。例如，若 e 编码为 0，a 编码为 1，t 编码成 01，则位串 0101 可以翻译成 eat，tea，eaea 或 tt，这会导致混淆。解决的方法就是使用前缀码，即一个字母的位串永远不应当出现在另外一个字母的位串的开头部分。

使用二叉树可以表达前缀码，并能用来解码。配合哈夫曼算法，可以完成根据字母出现的频率进行有效的不定长编码。



可从任何二叉树来构造一个前缀码，字符用从根到树叶的最短通路中的边来标记成位串。

无向树

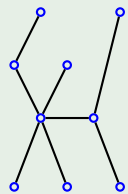
Definition

- 连通而不含回路的无向图称为无向树(undirected tree)，简称树(tree)，常用 T 表示树。
- 树中度数为 1 的结点称为叶(leaf)；度数大于 1 的结点称为分支点(branch point)或内部结点(interior point)。
- 每个连通分支都是树的无向图称为森林(forest)。
- 平凡图称为平凡树(trivial tree)。

容易看出，树中没有环和平行边，因此一定是简单图，并且在任何非平凡树中，都无度数为 0 的结点。

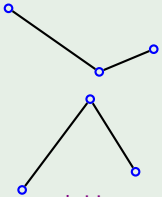
无向树

Example

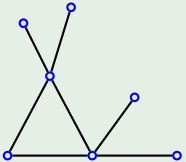


树

树



森林



不是树，也不是森林

考虑：一棵单独的树可以称作森林吗？

树的性质（等价定义）

Theorem

设无向图 $G=<V,E>$，$|V|=n$，$|E|=m$，下列各命题是等价的：

1. G 连通而不含回路(即 G 是树)；
2. G 中无回路，且 $m=n-1$；
3. G 是连通的，且 $m=n-1$；
4. G 中无回路，但在任二结点之间增加一条新边，就得到惟一的一条基本回路；
5. G 是连通的，但删除任一条边后，便不连通；$(n \geqslant 2)$
6. G 中每一对结点之间有惟一一条基本通路。$(n \geqslant 2)$

直接证明这 6 个命题两两等价的工作量太大，一般采用循环论证的方法，即证明

$(1) \Rightarrow (2) \Rightarrow (3) \Rightarrow (4) \Rightarrow (5) \Rightarrow (6) \Rightarrow (1)$

(1)⇒ (2)

(1) G 连通而不含回路
(2) G 中无回路，且 $m = n - 1$

Proof.

对 n 作归纳。 $n = 1$ 时，$m = 0$，显然有 $m = n - 1$。 假设 $n = k$ 时命题成立，现证 $n = k + 1$ 时也成立。
由于 G 连通而无回路，所以 G 中至少有一个度数为 1 的结点 v_0， 在 G 中删去 v_0 及其关联的边，便得到 k 个结点的连通而无回路的图，由归纳假设知它有 $k - 1$ 条边。 再将结点 v_0 及其关联的边加回得到原图 G，所以 G 中含有 $k + 1$ 个结点和 k 条边，符合公式 $m = n - 1$。
所以，G 中无回路，且 $m = n - 1$。 □

(2)⇒ (3)

(2)G 中无回路，且 $m = n - 1$

(3)G 是连通的，且 $m = n - 1$

Proof.

证明只有一个连通分支。

设 G 有 k 个连通分支 $G_1, G_2, \cdots, G_k$，其结点数分别为 $n_1, n_2, \cdots, n_k$，边数分别为 $m_1, m_2, \cdots, m_k$，且 $n = \sum_{i=1}^{k} n_i$ ，$m = \sum_{i=1}^{k} m_i$。 由于 G 中无回路，所以每个 $G_i(i = 1, 2, \cdots, k)$ 均为树，因此 $m_i = n_i - 1(i = 1, 2, \cdots, k)$，于是

$$m = \sum_{i=1}^{k} m_i = \sum_{i=1}^{k} (n_i - 1) = n - k = n - 1$$

故 $k = 1$，所以 G 是连通的，且 $m = n - 1$。 □

(3)⇒(4)

(3)G 是连通的，且 $m = n - 1$

(4)G 中无回路，但在任二结点之间增加一条新边，就得到惟一的一条基本回路

Proof.

首先证明 G 中无回路。对 n 作归纳。

$n = 1$ 时，$m = n - 1 = 0$，显然无回路。 假设结点数 $n = k - 1$ 时无回路，下面考虑结点数 $n = k$ 的情况。 因 G 连通，故 G 中每一个结点的度数均大于等于 1。 可以证明至少有一个结点 v_0，使得 $deg(v_0) = 1$，因若 k 个结点的度数都大于等于 2，则 $2m = \sum_{v \in V} deg(v) \geqslant 2k$，从而 $m \geqslant k$，即至少有 k 条边，但这与 $m = n - 1$ 矛盾。 在 G 中删去 v_0 及其关联的边，得到新图 G'，根据归纳假设知 G' 无回路，由于 $deg(v_0) = 1$，所以再将结点 v_0 及其关联的边加回得到原图 G，则 G 也无回路。

其次证明在 G 中任二结点 v_i，v_j 之间增加一条边 (v_i, v_j)，得到一条且仅一条基本回路。

由于 G 是连通的，从 v_i 到 v_j 有一条通路 L，再在 L 中增加一条边 (v_i, v_j)，就构成一条回路。若此回路不是惟一和基本的，则删去此新边，G 中必有回路，得出矛盾。 □

树的特点

在结点给定的无向图中，

树是边数最多的无回路图；

树是边数最少的连通图。

由此可知，在无向图 $G = (n, m)$ 中，

若 $m < n-1$，则 G 是不连通的；

若 $m > n-1$，则 G 必含回路。

树的性质

Theorem

任意非平凡树 $T = (n, m)$ 都至少有两片叶。

Proof.

因树 T 是连通的，从而 T 中各结点的度数均大于等于 1。 设 T 中有 k 个度数为 1 的结点 (即 k 片叶)，其余的结点度数均大于等于 2。 由握手定理，可得

$$2m = \sum_{v \in V} deg(v) \geqslant k + 2(n - k) = 2n - k$$

由于树中有 $m = n - 1$，于是 $2(n - 1) \geqslant 2n - k$，因此可得 $k \geqslant 2$，这说明 T 中至少有两片叶。 □

树的性质应用

Example

已知一棵无向树 T 中有 4 度，3 度，2 度的分支点各一个，其余为树叶，问 T 中有几片树叶？

Solution

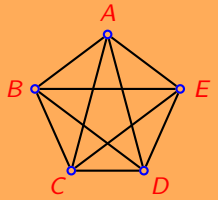
设 T 有 x 片树叶，则 T 共有 $n = 3 + x$ 个结点。 由握手定理以及树的性质，可得

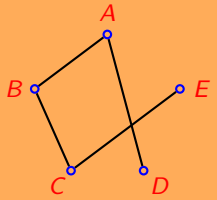
$$4 + 3 + 2 + x = 2(n - 1) = 2(3 + x - 1)$$

解出 $x = 5$，即 T 中有 5 片树叶。

生成树

考虑构建一个包含 5 个信息中心 A,B,C,D,E 的通信系统，可能的光纤连接如下左图所示。由于费用限制，要求铺设尽可能少的光纤线路，但又必须保持网络畅通。这实际上就是要求出一个边最少的连通图，这恰好符合树的特点。因而问题转化成在一个连通图中找到一棵树，一种可能的方式如右图所示。





生成树

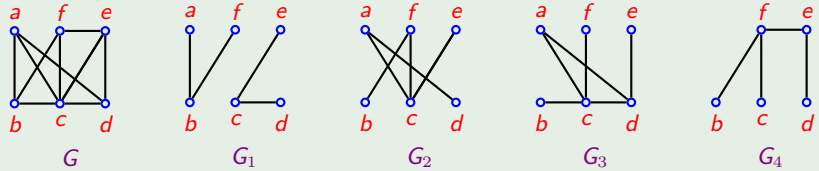
Definition

给定图 G = <V, E>，

- 若 G 的某个生成子图是树，则称之为 G 的生成树 (spanning tree)，记为 T_G。生成树 T_G 中的边称为树枝。
- G 中不在 T_G 中的边称为弦，T_G 的所有弦的集合称为生成树的补。

Example

对下图 G，则 G_1，G_2，G_3，G_4 四个图中，哪一个是它的生成树？(G_2)



生成树存在的条件

Theorem

一个图 $G =< V, E >$ 存在生成树 $T_G =< V_T, E_T >$ 的充分必要条件是 G 是连通的。

Proof.

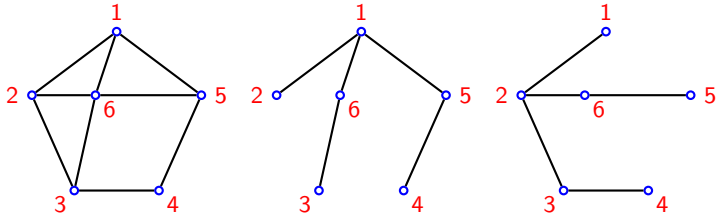
- 必要性：假设 $T_G =< V_T, E_T >$ 是 $G =< V, E >$ 的生成树，由树的定义知，则 T_G 连通的，于是 G 也是连通的。
- 充分性：假设 $G =< V, E >$ 是连通的。如果 G 中无回路，G 本身就是生成树。如果 G 中存在回路 C_1，可删除 C_1 中一条边得到图 G_1，它仍连通且与 G 有相同的结点集。如果 G_1 中无回路，G_1 就是生成树。如果 G_1 仍存在回路 C_2，可删除 C_2 中一条边，如此继续，直到得到一个无回路的连通图 H 为止。可见，H 是 G 的生成树。

□

生成树算法

求连通图 $G = (n, m)$ 的生成树的算法：

- 破圈法: 循环找到图中的回路并删除回路中的一条边，直到删除的边的总数为 $m - n + 1$。
- 避圈法: 循环选取 G 中一条与已选取的边不构成回路的边，直到选取的边的总数为 $n - 1$。



可见，生成树不唯一。(为什么 ?)

生成树算法

破圈法和避圈法很多时候不实用，这是由于他们都需要找出回路或验证不存在回路。因而较常用的方法是深度优先搜索算法和广度优先搜索算法来求出生成树。深度优先算法涉及回溯，这里以广度优先搜索算法为例来说明。

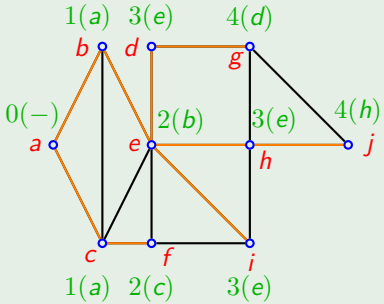
☞ 生成树的广度优先搜索算法

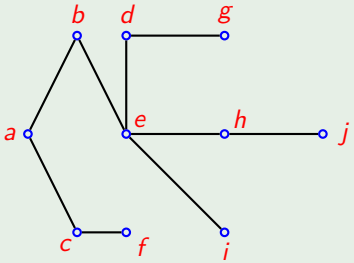
连通图 $G=<V,E>$,

1. 任选 $s \in V$，将 s 标记为 0，令 $L=\{s\}$，$V=V-\{s\}$，$k=0$，$E_G=\varnothing$；
2. 如果 $V=\varnothing$，则结束，E_G 为所求的生成树中包含的所有边。否则令 $k=k+1$；
3. 依次对 L 中所有标记为 $k-1$ 的结点 v，如果它与 V 中的结点 w 相邻接，则将 w 标记为 k，指定 v 为 w 的前驱，令 $L=L\cup\{w\}$，$V=V-\{w\}$，$E_G=E_G\cup\{(v,w)\}$，转 (2)；

广度优先搜索求生成树

Example

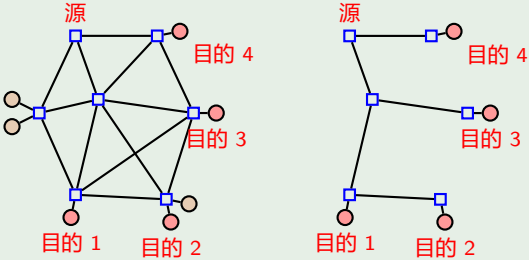




生成树在网络中的应用-IP 组播

Example

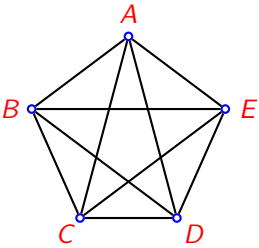
为了让数据能够穿过路由器组成的网络通路尽快到达目的计算机，并且要尽量的节省网络资源，则这些数据走过的通路就不应该存在回路，这也恰好符合树的特点，因而只要找出源计算机，中间通信网络和目的计算机所构成的连通图的生成树即可。下面右图给出了左图所对应的一棵生成树。

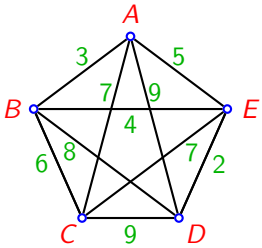


IP 网络示意图　　组播生成树

引子

回顾生成树的学习中提到的构建一个包含 5 个信息中心 A,B,C,D,E 的通信系统的问题，如下左图所示。通常情况下，各中心之间的光纤连接长度并不相同，这会影响总体费用。所以我们建立一个带权图 (以百公里为单位，如右图所示)，希望能从这个图中找出一棵生成树，而且总权值最小。





无向树

Definition

设 $G =< V, E >$ 是连通的赋权图，T 是 G 的一棵生成树，T 的每个树枝所赋权值之和称为 T 的权，记为 $w(T)$。G 中具有最小权的生成树称为 G 的最小生成树(minimal spanning tree)。

一个无向图的生成树不是惟一的，同样地，一个赋权图的最小生成树也不一定是惟一的。求赋权图的最小生成树的方法很多，这里主要介绍 Kruskal 算法和 Prim 算法。

Kruskal 算法

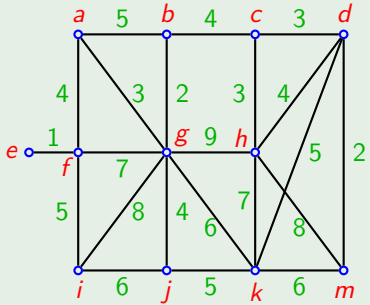
Kruskal 算法是克鲁斯克尔 (Kruskal) 于 1956 年将构造生成树的避圈法推广到求最小生成树，其要点是，在与已选取的边不构成回路的边中选取最小者。

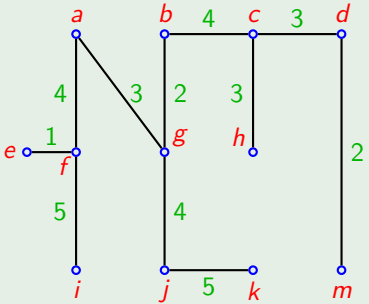
Kruskal 算法

1. 在 G 中选取最小权边 e_1，置 $i=1$，$E_T=\{e_1\}$。
2. 当 $i=n-1$ 时，结束，否则转 (3)。
3. 在 G 中选取不在 E_T 中的边 e_{i+1}，使 $E_T\cup\{e_{i+1}\}$ 中无回路且 e_{i+1} 是满足此条件的最小权边。
4. 置 $i=i+1$，$E_T=E_T\cup\{e_{i+1}\}$，转 (2)。

Kruskal 算法

Example





w(T) = 36

Prim 算法

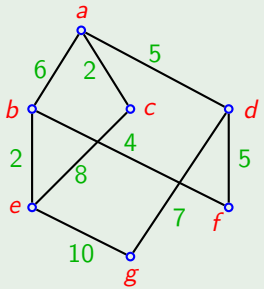
Prim 算法的要点是，从任意结点开始，每次增加一条最小权边构成一棵新树。

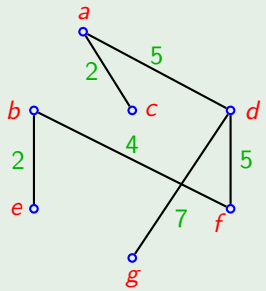
Prim 算法

1. 在 G 中任意选取一个结点 v_1，置 $V_T = \{v_1\}$, $E_T = \varnothing$，$k = 1$；
2. 在 $V - V_T$ 中选取与某个 $v_i \in V_T$ 邻接的结点 v_j，使得边 (v_i, v_j) 的权最小，置 $V_T = V_T \cup \{v_j\}$, $E_T = E_T \cup \{(v_i, v_j)\}$，$k = k + 1$；
3. 重复步骤 2，直到 $k = |V|$。

Prim 算法

Example





w(T) = 25

有向树和根树

定义

一个有向图，若略去所有有向边的方向所得到的无向图是一棵树，则这个有向图称为有向树。

定义

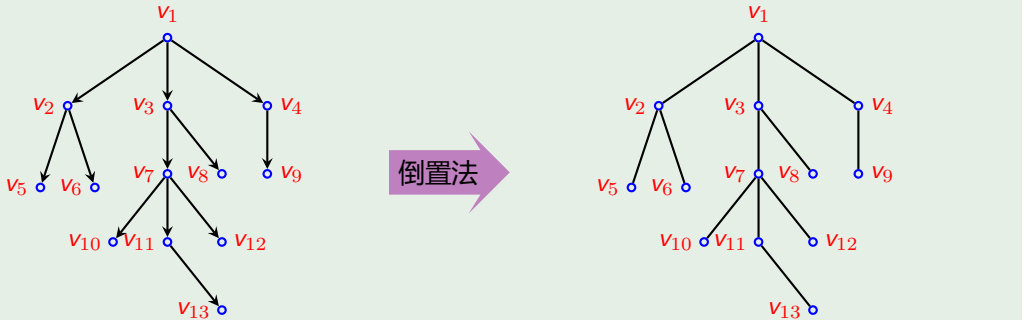
一棵非平凡的有向树，如果恰有一个结点的入度为 0，其余所有结点的入度均为 1， 则称之为根树(root tree) 或外向树(outward tree)。入度为 0 的结点称为根(root)；出度为 0 的结点称为叶(leaf)；入度为 1，出度大于 0 的结点称为内点(interior point)；又将内点和根统称为分支点(branch point)。

定义

在根树中，从根到任一结点 v 的通路长度，称为该结点的层数；称层数相同的结点在同一层上；所有结点的层数中最大的称为根树的高。

根树

例



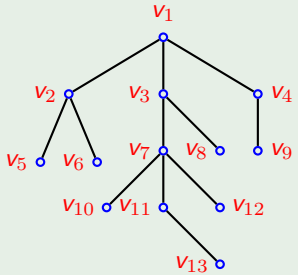
习惯上我们使用倒置法来画根树，即把根画在最上方，叶画在下方，有向边的方向均指向下方，这样就可以省去全部箭头，不会发生误解。

树的家族关系

定义

在根树中，若从结点 v_i 到 v_j 可达，则称 v_i 是 v_j 的祖先，v_j 是 v_i 的后代；又若 $< v_i, v_j >$ 是根树中的有向边，则称 v_i 是 v_j 的父亲，v_j 是 v_i 的儿子；如果两个结点是同一个结点的儿子，则称这两个结点是兄弟。

例



- v_2 是 v_5 和 v_6 的父亲,v_5 和 v_6 是 v_2 的儿子;
- v_2,v_3 和 v_4 是兄弟;v_{10},v_{11},v_{12} 也是兄弟;
- v_8 的祖先有 v_3,v_1;
- v_7 的后代有 v_{10},v_{11},v_{12} 和 v_{13}.

有序和 k 元树

定义

如果在根树中规定了每一层上结点的次序，这样的根树称为有序树。

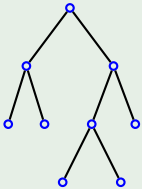
定义

在根树 T 中，

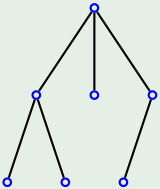
- 若每个分支点至多有 k 个儿子，则称 T 为 k 元树；
- 若每个分支点都恰有 k 个儿子，则称 T 为满 k 元树；
- 若 k 元树 T 是有序的，则称 T 为 k 元有序树；
- 若满 k 元树 T 是有序的，则称 T 为满 k 元有序树。
- 任一结点 v 及其所有后代导出的子图 T' 称为 T 的以 v 为根的子树。

有序和 k 元树

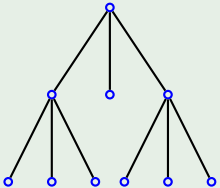
例



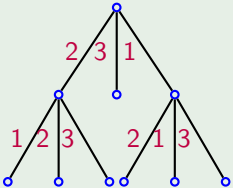
满二元树



3 元树



满 3 元树



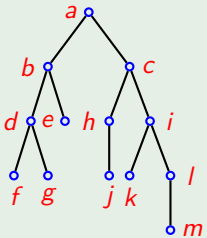
满 3 元有序树

二元有序树

定义

二元有序树的每个结点 v 至多有两个儿子，分别称为 v 的左儿子和右儿子。二元有序树的每个结点 v 至多有两棵子树，分别称为 v 的左子树和右子树。

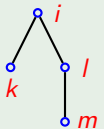
例



T

以 c 为根的子树

c 的左子树



c 的右子树

满 k 元树的性质

定理

在满 k 元树中，若叶数为 t，分支点数为 i，则有

$$(k-1) \times i = t - 1$$。

证明.

由假设知，该树有 $i + t$ 个结点。
由树的定义知，该树的边数为 i+t-1。
由握手定理知，所有结点的出度之和等于边数。
而根据满 k 元树的定义知，所有分支点的出度为 $k \times i$
因此有 $k \times i = i + t - 1$
即 $(k-1) \times i = t - 1$

□

满 k 元树的性质

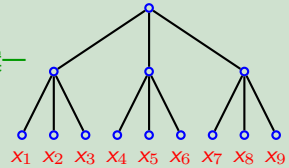
例

假设有一台计算机，它有一条加法指令，可计算 3 个数的和。如果要求 9 个数 $x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9$ 之和，问至少要执行几次加法指令？

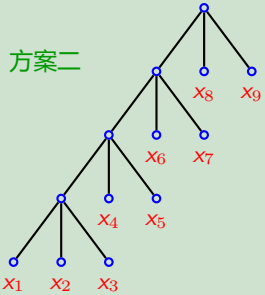
解

本问题可转化为求一个含有 *9* 片树叶的满三元树的分支点个数。由前面的定理知，有 $(3-1) \times i = 9-1$，得 $i = 4$。所以至少要执行 *4* 次加法指令。

方案一



方案二



遍历问题

在使用根树来保存信息时，我们经常需要依次访问树的每个结点，或是查询这些结点中是否有某些特定信息，或是利用这些结点信息来进行计算，诸如此类。基本要求就是能系统地访问树的结点，使得每个结点恰好访问一次，这称作根树的遍历问题。

根树也经常用来表示各种类型的表达式，比如由数字、变量和运算所组成的算术表达式。当需要对表达式求值时，就需要对此根树进行遍历。

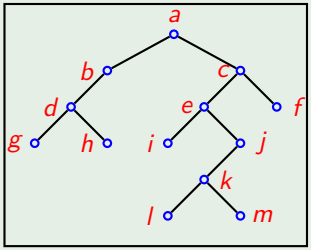
二元树的遍历

k 元树中，应用最广泛的是二元树，这是由于二元树在计算机中最易处理。

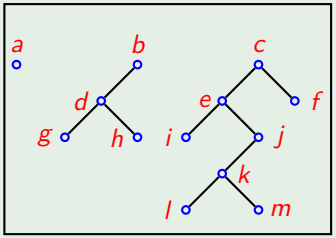
- **二元树的先根次序遍历算法：**
  1. 访问根；
  2. 按先根次序遍历根的左子树；
  3. 按先根次序遍历根的右子树。
- **二元树的中根次序遍历算法：**
  1. 按中根次序遍历根的左子树；
  2. 访问根；
  3. 按中根次序遍历根的右子树。
- **二元树的后根次序遍历算法：**
  1. 按后根次序遍历根的左子树；
  2. 按后根次序遍历根的右子树；
  3. 访问根。

先根遍历: 根、左子树、右子树

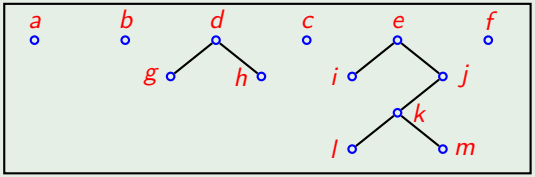
例



分解 a

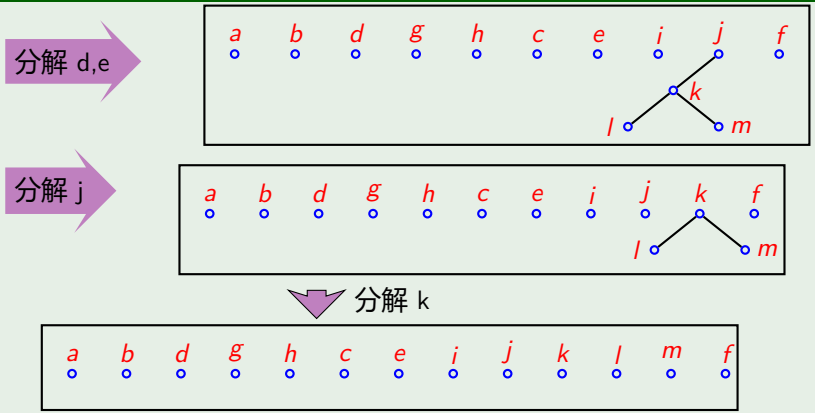


分解 b,c



先根遍历: 根、左子树、右子树

例



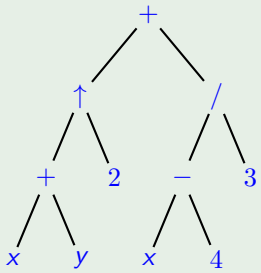
类似的，中根遍历次序为 gdhbaielkmjcf，后根遍历次序为 ghdbilmkjefca。

表达式的二叉树

可以用二叉树表示一些复杂的表达式，如复合命题，集合的组合，以及算术表达式。

例

$$((x+y)\uparrow 2)+((x-4)/3)$$

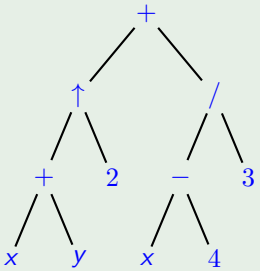


当我们对二叉树进行中根遍历时，就得到了原表达式。考虑到运算顺序问题，我们应当在遍历的时候给表达式加上括号。中缀形式

前缀形式

例

$$+\uparrow+xy2/-x43$$



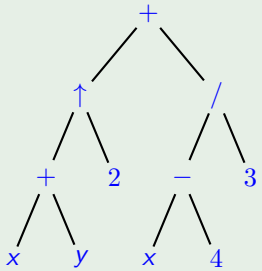
对表达式的二叉树进行先根遍历时，就得到了它的前缀形式。前缀形式的最大优点是无二义性，所以不再需要括号。

写成前缀形式的表达式称为波兰符号法。表达式的求值方式是从右向左。

后缀形式

例

$$xy + 2 \uparrow x4 - 3/+$$



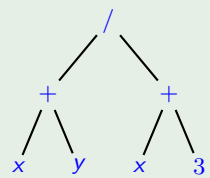
对表达式的二叉树进行后根遍历时，就得到了它的后缀形式。后缀形式同样无二义性，自然也不需要括号。

写成后缀形式的表达式称为逆波兰符号法。表达式的求值方式是从左向右。

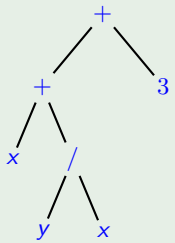
二义性举例

例

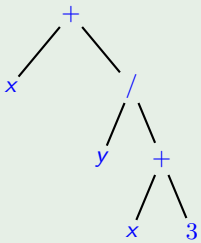
$(x+y)/(x+3)$



$(x+(y/x))+3$



$x+(y/(x+3))$



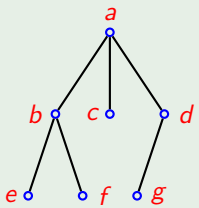
这三个不同表达式的中根遍历结果都是 $x+y/x+3$, 而前缀形式和后缀形式却各不相同。

前缀形式和后缀形式的表达式都是无二义性的，从而只用扫描一次就可以求出它们的值，因此在计算机科学里面大量使用，例如编译器的构造。

根树的遍历

- **根树的先根次序遍历算法：**
  1. 访问根；
  2. 按先根次序从左向右遍历根的各子树；
- **根树的后根次序遍历算法：**
  1. 按后根次序从左向右遍历根的各子树；
  2. 访问根。

例



先根遍历：*abefcdg*

后根遍历：*efbcgda*

引子

在计算机及通讯事业中，常用二进制编码来表示符号。

例如，可用 00、01、10、11 分别表示字母 *A*、*B*、*C*、*D*，这称作等长编码。这在四个字母出现频率基本相等的情况下是非常合理的。

但当四个字母出现的频率很不一样，如 *A* 出现的频率为 50%，*B* 出现的频率为 25%，*C* 出现的频率为 20%，*D* 出现的频率为 5% 时，使用等长编码就不是最优的方式了。

如果此时我们使用不等长编码，如用 000 表示字母 *D*，用 001 表示字母 *C*，01 表示 *B*，1 表示 *A*。在同样传输 100 个字母的情况下，等长编码需 $2\times100=200$ 个二进制位，而不等长编码仅需 $3\times5+3\times20+2\times25+1\times50=175$ 个二进制位。

但不等长编码不能随意定义，否则会引起问题，如当我们用 1 表示 A，用 00 表示 B，用 001 表示 C，用 000 表示 D 时，如果接收到的信息为 001000，则无法辨别它是 CD 还是 BAD。

前缀码

Definition

- 设 $a_1a_2\cdots a_{n-1}a_n$ 为长度为 n 的符号串，称其子串 $a_1, a_1a_2, \cdots, a_1a_2\cdots a_{n-1}$ 分别为 $a_1a_2\cdots a_{n-1}a_n$ 的长度为 $1, 2, \cdots, n-1$ 的前缀。
- 设 $A = \{b_1, b_2, \cdots, b_m\}$ 是一个符号串集合，若对任意 $b_i, b_j \in A$，$b_i \neq b_j$，b_i 不是 b_j 的前缀，b_j 也不是 b_i 的前缀，则称 A 为前缀码。若符号串 $b_i(i = 1, 2, \cdots, m)$ 中，只出现 0 和 1 两个符号，则称 A 为二元前缀码。

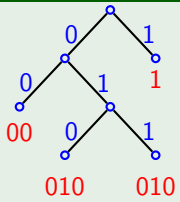
Example

- $\{1, 01, 001, 000\}$ 是前缀码；
- $\{1, 11, 001, 0011\}$ 不是前缀码。

用二元树产生二元前缀码

给定一棵二元树 T，假设它有 t 片树叶。设 v 是 T 任意一个分支点，则 v 至少有一个儿子，至多有两个儿子。若 v 有两个儿子，则在由 v 引出的两条边上，左边的标上 0，右边的标上 1；若 v 只有一个儿子，在 v 引出的边上可标 0 也可标 1。设 w 为 T 的任意一片树叶，从树根到 w 的通路上各边的标号组成的符号串放在 w 处，t 片树叶处的 t 个符号串组成的集合为一个二元前缀码。

Example



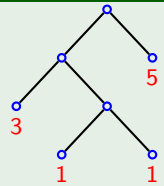
此二元树产生的前缀码为 $\{1, 00, 010, 011\}$

最优树

Definition

设有一棵二元树 T，若对其所有的 t 片叶赋以权值 $w_1, w_2, \cdots, w_t$，则称之为赋权二元树；若权为 w_i 的叶的层数为 $L(w_i)$，则称 $W(T) = \sum_{i=1}^{t} w_i \times L(w_i)$ 为该赋权二元树的权；而在所有赋权 $w_1, w_2, \cdots, w_t$ 的二元树中，$W(T)$ 最小的二元树称为最优树。

Example



则此赋权二元树的权为：

$$5 \times 1 + 3 \times 2 + 1 \times 3 + 1 \times 3 = 17$$

哈夫曼算法

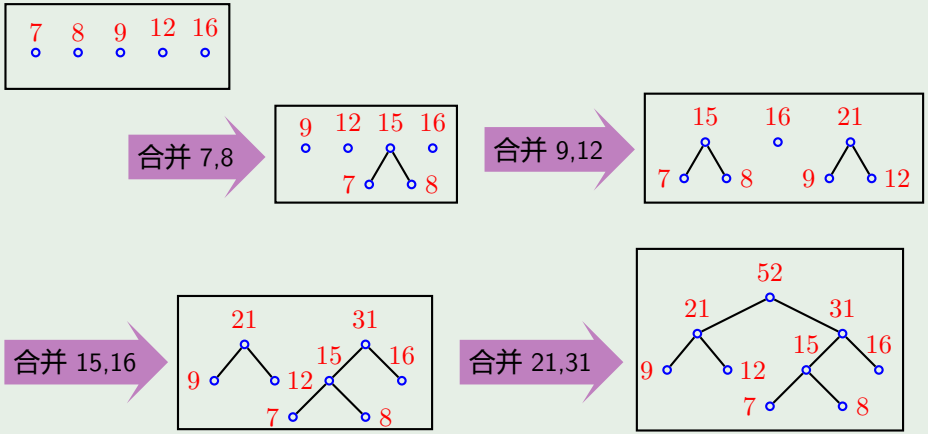
1952 年哈夫曼 (Huffman) 给出了求最优树的方法。

哈夫曼算法：

1. 初始：令 $S = \{w_1, w_2, \cdots, w_t\}$；
2. 从 S 中取出两个最小的权 w_i 和 w_j，画结点 v_i 和 v_j，分别带权 w_i 和 w_j。画 v_i 和 v_j 的父亲 v，令 v 带权 $w_i + w_j$；
3. 令 $S = (S - \{w_i, w_j\}) \cup \{w_i + w_j\}$；
4. 判断 S 是否只含一个元素？若是，则停止，否则转 2。

哈夫曼算法

Example



前缀码构造

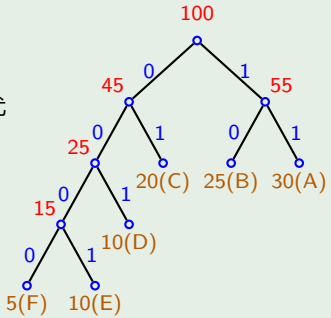
Example

已知字母 A、B、C、D、E、F 出现的频率如下：

A—30%，B—25%，C—20% D—10%，E—10%，F—5%

构造一个表示 A、B、C、D、E、F 前缀码，使得传输的二进制位最少。

1. 构造带权 30,25,20,10,10,5 的最优二元树 T;
2. 在 T 上构造前缀码;
3. 将前缀码对应于字母;

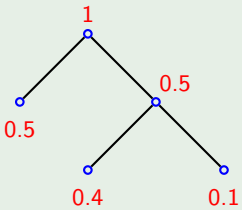


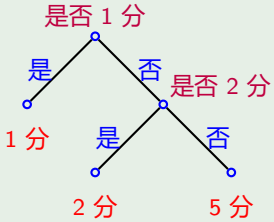
字母	编码
A	11
B	10
C	01
D	001
E	0001
F	0000

决策问题

Example

用机器分辨一些币值为 1 分、2 分、5 分的硬币，假设各种硬币出现的概率分别为 0.5、0.4、0.1。问如何设计一个分辨硬币的算法，使所需的时间最少？(假设每作一次判别所用的时间相同，以此为一个时间单位)





所需时间：$2 \times 0.1 + 2 \times 0.4 + 1 \times 0.5 = 1.5$(时间单位)。