离 散 数 学

电子科技大学 计算机科学与工程学院
网络空间安全学 院

2024年5月27日星期一

第10章 树

树是图论中的一个非常重要的概念，而在计算机科学中有着非常广泛的应用，例如现代计算机操作系统均采用树形结构来组织文件和文件夹，本章介绍树的基本知识和应用。

在本章中，所谈到的图都假定是简单图；所谈到的回路均指简单回路或基本回路。并且同一个图形表示的回路(简单的或基本的)，可能有不同的交替序列表示方法，但我们规定它们表示的是同一条回路。

10.0 内容提要

1. 与树相关的概念： 树、森林、根树、根、叶、分支点、生成树、最小生成树、k元树、k元完全树子树、有序树、祖先与后代、父亲与儿子、最优树等；
2. 树的基本性质：$m = n-1$等；
3. 树的算法：求生成树与最小生成树的算法、求最优树的算法、二元树遍历的算法、根树与二元树相互转化的算法等；
4. 树的应用。

10.1 本章学习要求

重点掌握

1

1 与树相关基本概念

2 树的性质

3 树的基本算法

一般掌握

2

树的算法

了解

3

1 树的同构

2 树的应用

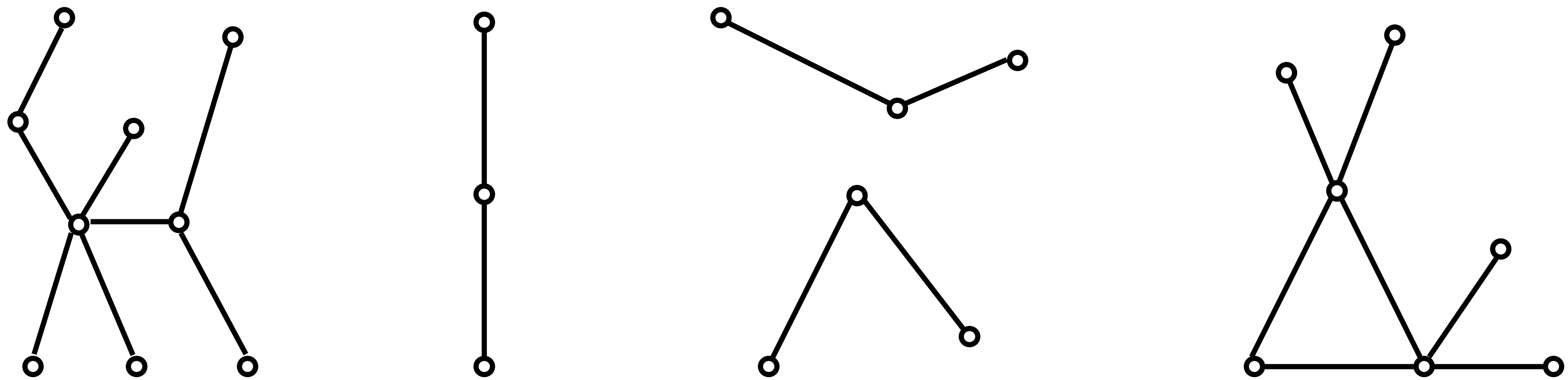
定义10.2.1

- 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)。

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

例10.2.2

判断下图中的图哪些是树？为什么？



(a) (b) (c) (d)

分析 判断无向图是否是树，根据定义10.2.1，首先看它是否连通，然后看它是否有回路。

解 图(a)、(b)都是连通，并且不含回路，因此是树；图(c)不连通，因此不是树，但由于它不含回路，因此是森林；图(d)虽然连通，但存在回路，因此不是树。

树的性质

定理10.2.1 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G中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G中任一条边后，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。$(n\geqslant 2)$

定理10.2.1 分析

直接证明这6个命题两两等价工作量太大，一般采用循环论证的方法，即证明

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

然后利用传递性，得到结论。

树的特点

由定理10.2.1(4)

由定理10.2.1(5)

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

定理10.2.2

任意非平凡树T =（n，m）都至少有两片叶。

分析　利用握手定理和m = n−1即可。

证明　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。于是由握手定理

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

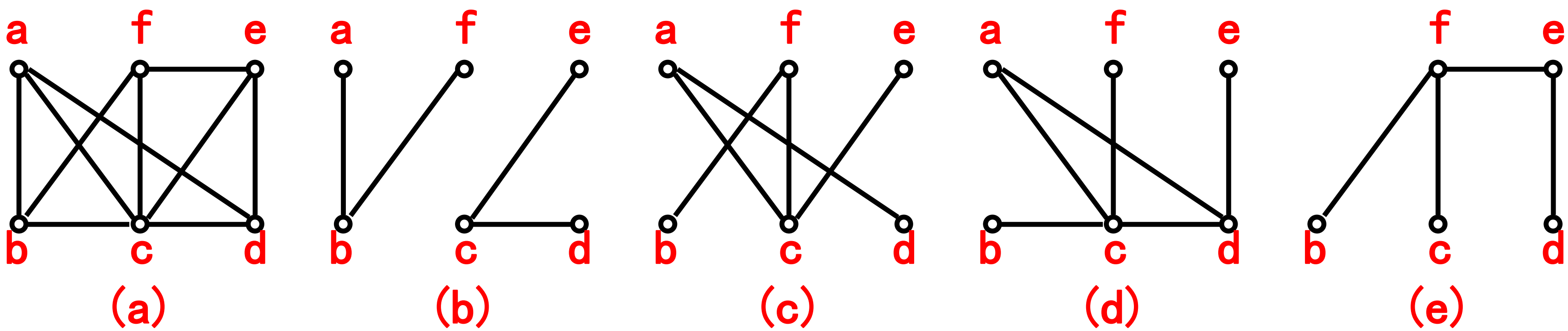
由于树中有m = n−1，于是2(n−1)≥2n−k，因此可得k≥2，这说明T中至少有两片叶。

10.2.2 生成树

定义10.2.2 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(Branch)；G中不在T_G中的边称为弦(Chord)；T_G的所有弦的集合称为生成树的补(Complement)。

例10.2.3

判断下图中的图(b)、(c)、(d)、(e)是否是图(a)的生成树。



分析 判断是否是生成树，根据定义10.2.2，首先看它是否是树，然后再看它是否是生成子图。由于图(b)和(d)不是树，图(e)不是生成子图，因此它们都不是图(a)的生成树，而图(c)既是树，又是生成子图，因此是生成树。

解 图(b)、(d)和(e)不是图(a)的生成树，图(c)是图(a)的生成树，其中边(a, c)、(a, d)、(b, f)、(c, f)、(c, e)是树枝，而(a, b)、(b, c)、(c, d)、(d, e)、(e, f)是弦。

定理10.2.3

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。

证明 必要性 假设T_G = <V_T, E_T>是G的生成树，由定义10.2.1，T_G是连通的，于是G也是连通的。

分析 必要性由树的定义即得，充分性利用构造性方法，具体找出一棵生成树即可

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。因此，H是G的生成树。

破圈法与避圈法

- 算法10.2.1　求连通图G = <V, E>的生成树的破圈法：

  每次删除回路中的一条边，其删除的边的总数为m-n+1。

- 算法10.2.2　求连通图G = <V, E>的生成树的避圈法：

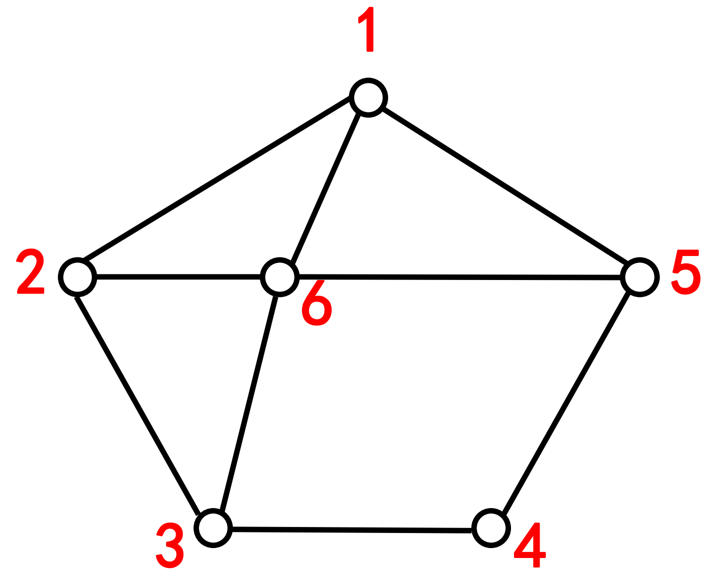
  每次选取G中一条与已选取的边不构成回路的边，选取的边的总数为n-1。

由于删除回路上的边和选择不构成任何回路的边有多种选法，所以产生的生成树不是惟一的。

例10.2.4

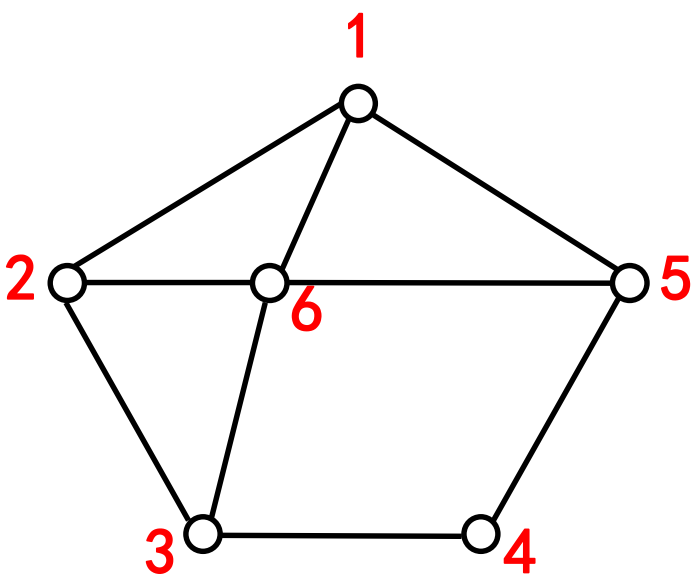
分别用破圈法和避圈法求下图的生成树。

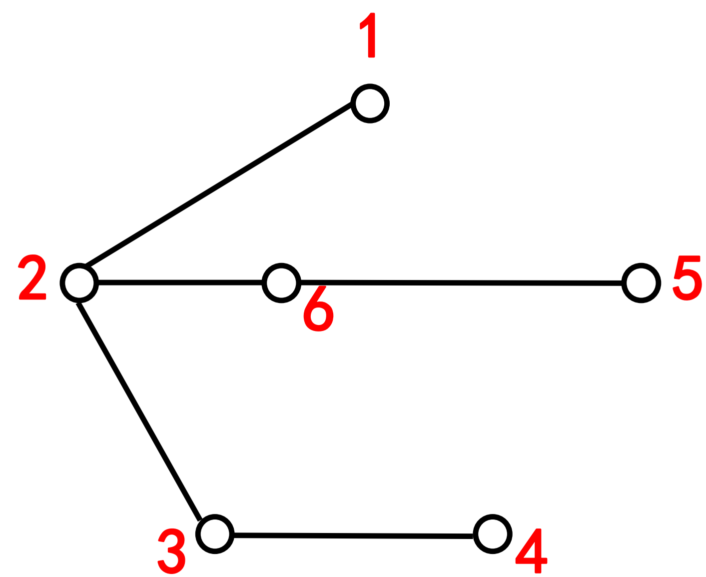
破圈法



分析　分别用破圈法和避圈法依次进行即可。用破圈法时，由于n = 6，m = 9，所以m-n+1 = 4，故要删除的边数为4，因此只需4步即可。用避圈法时，由于n = 6，所以n-1 = 5，故要选取5条边，因此需5步即可。

避圈法





- 由于生成树的形式不惟一，故上述两棵生成树都是所求的。
- 破圈法和避圈法的计算量较大，主要是需要找出回路或验证不存在回路。

算法10.2.3

求连通图G = <V, E>的生成树的广度优先搜索算法：

（1）任选s∈V，将v标记为0，令L = {s}，V = V−{s}，k = 0；

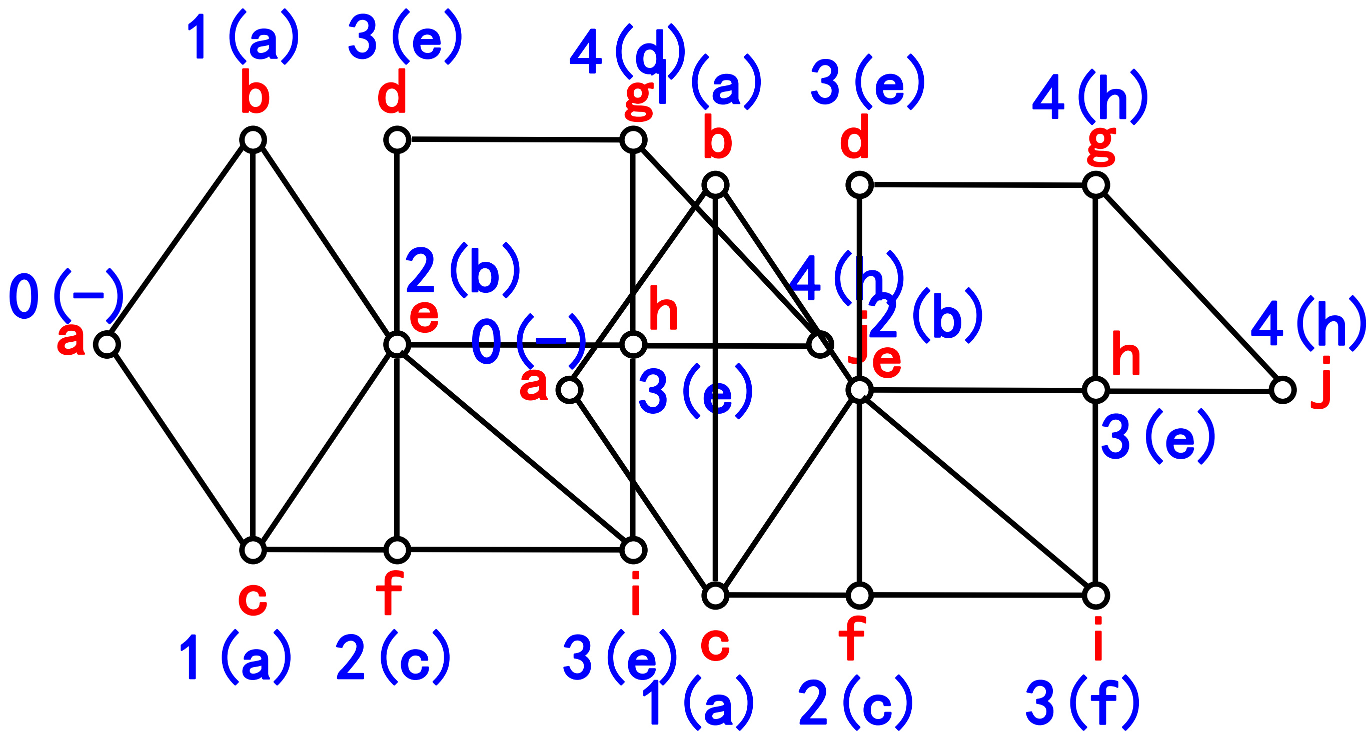
（2）如果V = Φ，则转(4)，否则令k = k+1；

（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∪{w}，V = V−{w}，转(2)；

（4）E_G = {(v, w)|w∈L−{s}，v为w的前驱}，结束。

例10.2.5

利用广度优先搜索算法求下图的生成树。



10.2.3 最小生成树

定义10.2.3 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**(Weight)，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。

算法10.2.3 Kruskal算法

（1）在G中选取最小权边e_1，置$i = 1$。

（2）当$i = n-1$时，结束，否则转(3)。

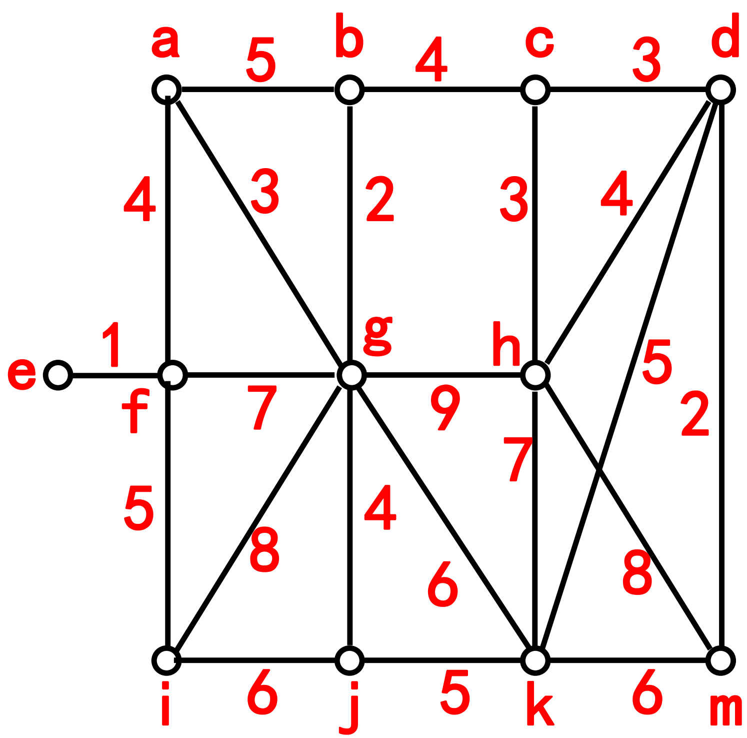
（3）设已选取的边为e_1，e_2，…，e_i，在G中选取不同于e_1，e_2，…，e_i的边e_{i+1}，使$\{e_1, e_2, \dots, e_i, e_{i+1}\}$中无回路且$e_{i+1}$是满足此条件的最小权边。

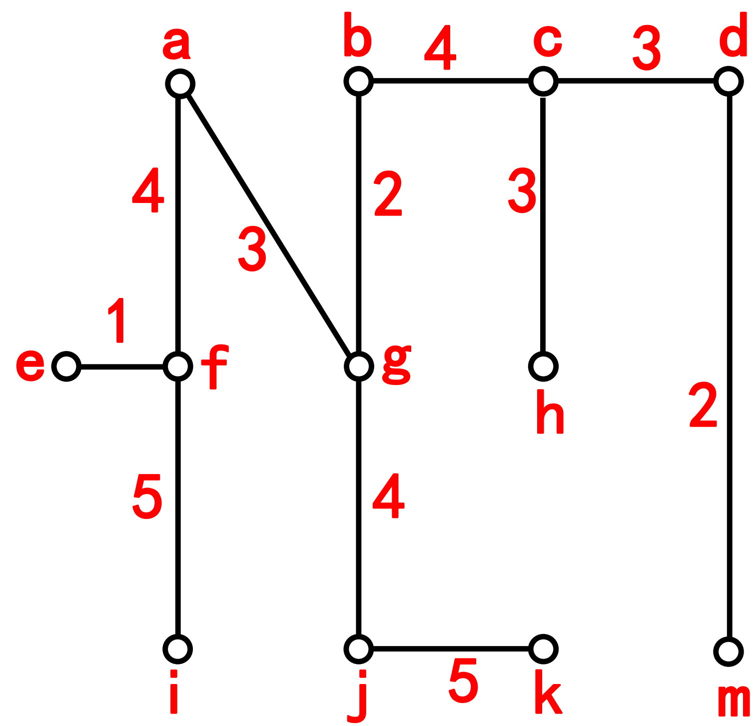
（4）置$i = i+1$，转(2)。

在Kruskal算法的步骤1和3中，若满足条件的最小权边不止一条，则可从中任选一条，这样就会产生不同的最小生成树。

例10.2.6

用Kruskal算法求图中赋权图的最小生成树。





解 n=12，按算法要执行n-1=11次，w(T) = 36。

算法10.2.5 Prim算法

（1）在G中任意选取一个结点v_1，置VT = $\{v_1\}$，E_T = Φ，k = 1；

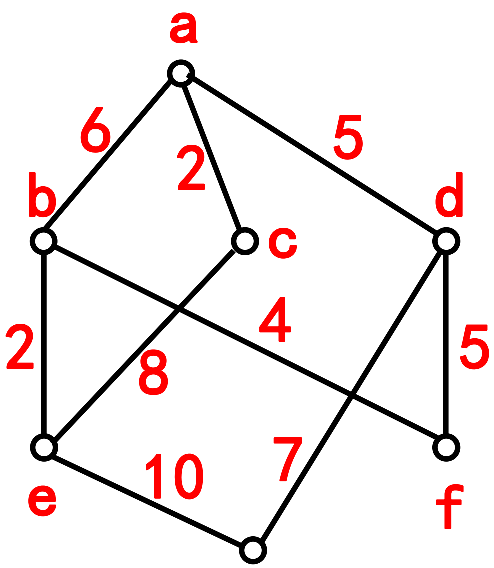
（2）在$V-V_T$中选取与某个$v_i \in V_T$邻接的结点v_j，使得边(v_i, v_j)的权最小，置$V_T = V_T \cup \{v_j\}$，$E_T = E_T \cup \{(v_i, v_j)\}$，k = k+1；

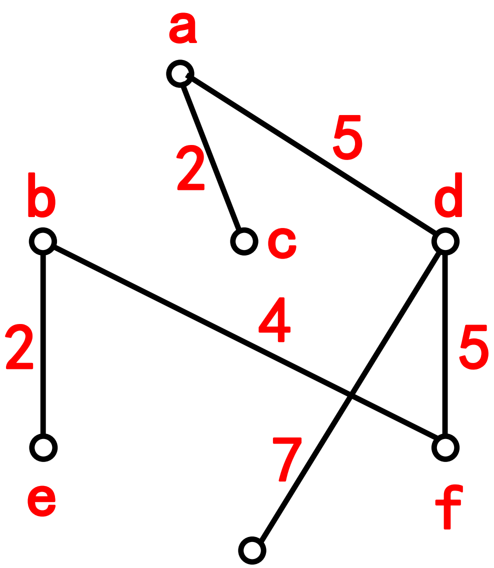
（3）重复步骤2，直到k = |V|。

在Prim算法的步骤2中，若满足条件的最小权边不止一条，则可从中任选一条，这样就会产生不同的最小生成树。

例10.2.7

用Prim算法求图中赋权图的最小生成树。





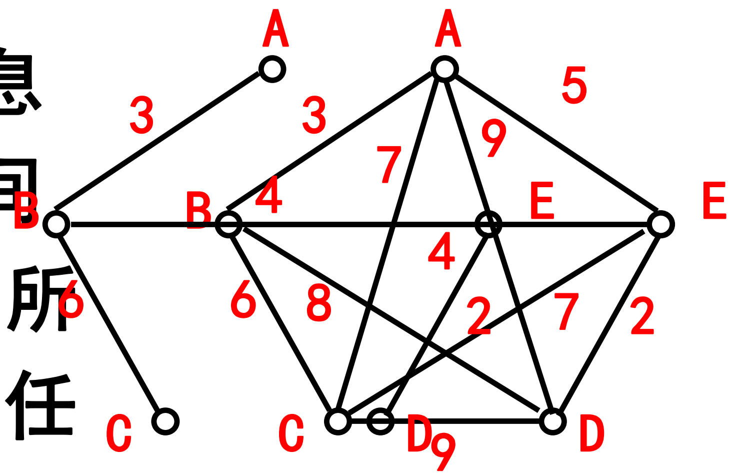
由Prim算法可以看出，每一步得到的图一定是树，故不需要验证是否有回路，因此它的计算工作量较Kruskal算法要小。

10.2.4 无向树的难点

1. 树是不含回路的连通图。注意把握树的性质，特别是树中叶结点的数目及边数与结点数的关系：$m = n-1$；
2. 生成树是无向连通图是树的生成子图。注意把握所有连通图都有生成树，知道生成树的树枝与弦及其数目，会使用避圈法、破圈法和广度优先搜索算法求生成树；
3. 最小生成树是赋权连通图的权值之和最小的生成树。会使用Kruskal算法和Prim算法求最小生成树。

10.2.5 无向树的应用

例10.2.8 假设有5个信息中心A、B、C、D、E，它们之间的距离(以百公里为单位)如图所示。要交换数据，我们可以在任意两个信息中心之间通过光纤连接，但是费用的限制要求铺设尽可能少的光纤线路。重要的是每个信息中心都能和其它中心通信，但并不需要在任意两个中心之间都铺设线路，而可以通过其它中心转发。



解 求得图的最小生成树如图所示，(5) 15百公里。根据图的图铺设可使得铺设的线路最短。

分析 这实际上就是求赋权连通图的最小生成树问题，可用Prim算法或Kruskal算法求解。

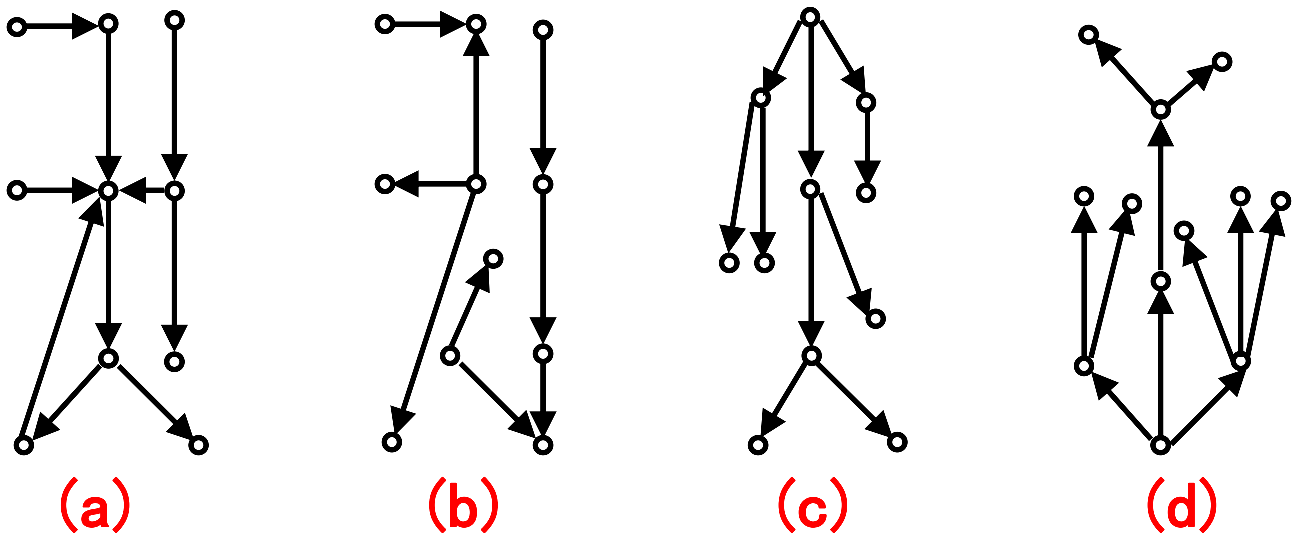
10.3 根树

10.3.1 根树的定义与分类

定义10.3.1 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(Directed Tree)。

例10.3.1

判断下图中的图哪些是树？为什么？

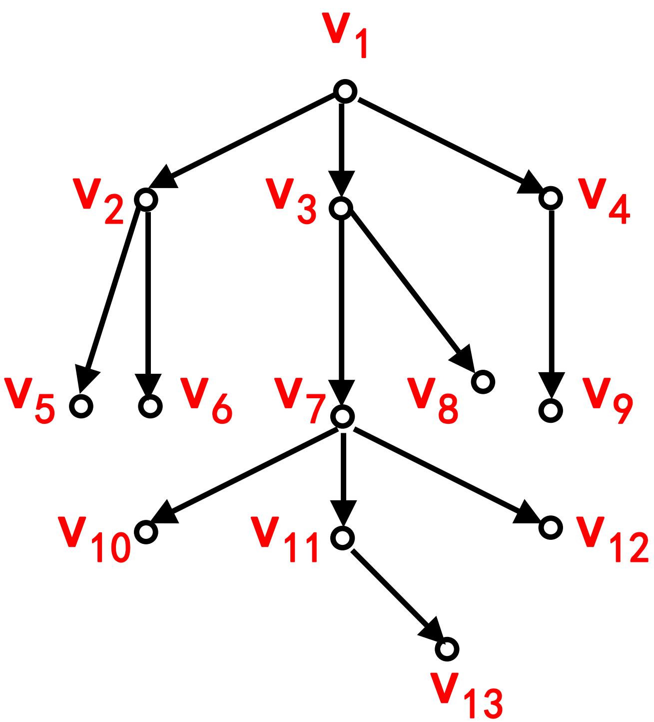


定义10.3.2

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)。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(Layer Number)；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(Height)。

例10.3.2

判断下图所示的图是否是根树？若是根树，给出其根、叶和内点，计算所有结点所在的层数和高。



倒置法

解：是一棵根树，其中v_1为根，$v_5, v_6, v_8, v_9, v_{10}, v_{12}, v_{13}$为叶，$v_2, v_3, v_4, v_7, v_{11}$为内点。

v_1处在第零层，层数为0；v_2, v_3, v_4同处在第一层，层数为1；v_5, v_6, v_7, v_8, v_9同处在第二层，层数为2；v_{10}, v_{11}, v_{12}同处在第三层，层数为3；v_{13}处在第四层，层数为4；这棵树的高为4。

家族关系

定义10.3.3　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(Ancestor)，v_j是v_i的后代(Descendant)；又若<v_i, v_j>是根树中的有向边，则称v_i是v_j的父亲(Father)，v_j是v_i的儿子(Son)；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(Brother)。

定义10.3.4　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(Ordered Tree)。

一般地，在有序树中同一层中结点的次序为从左至右。有时也可以用边的次序来代替结点的次序。

定义10.3.5

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(k-ary Tree)；
- 若每个分支点都恰有k个儿子，则称T为k元完全树(k-ary Complete Tree)；
- 若k元树T是有序的，则称T为k元有序树(k-ary Ordered Tree)；
- 若k元完全树T是有序的，则称T为k元有序完全树(k-ary Ordered Complete Tree)。

子树

在根树T中，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(Subtree)。

当然，T’也可以有自己的子树。

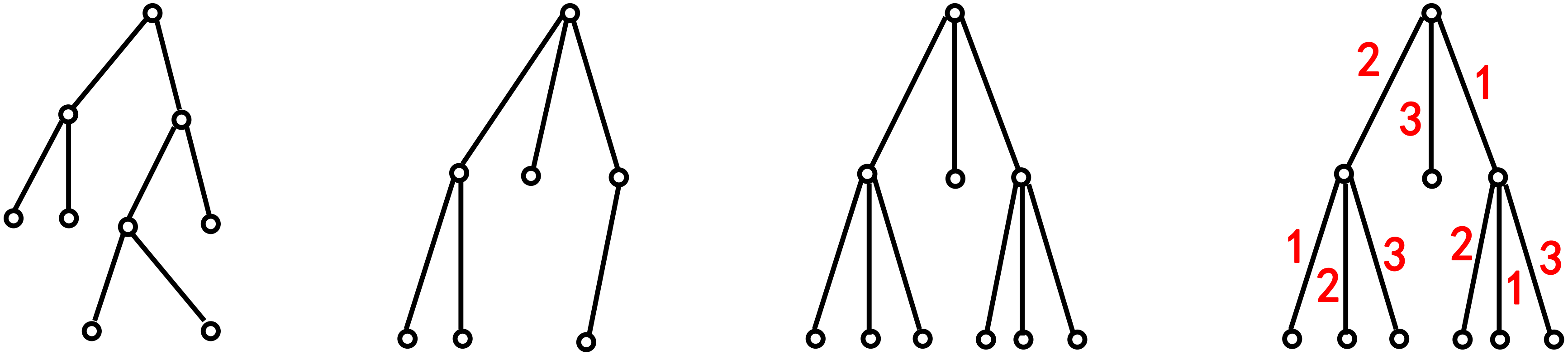
二元有序树的每个结点v至多有两个儿子，分别称为v的左儿子(Left Son)和右儿子(Right Son)。

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(Left Subtree)和右子树(Right Subtree)。

注意区分以v为根的子树和v的左(右)子树，v为根的子树包含v，而v的左(右)子树不包含v。

例10.3.3

判断下图所示的几棵根树是什么树？



(a)　(b)　(c)　(d)

2元
完全树

3元树

3元
完全树

3元有序
完全树

k元完全树中分支点与叶结点数目之间的关系

定理10.3.1 在k元完全树中，若叶数为t，分支点数为i，则下式成立：

$$(k-1)\times i = t-1$$

证明 由假设知，该树有i+t个结点。由定理10.2.1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k元完全树的定义知，所有分支点的出度为k×i。因此有

$$k\times i = i+t-1$$

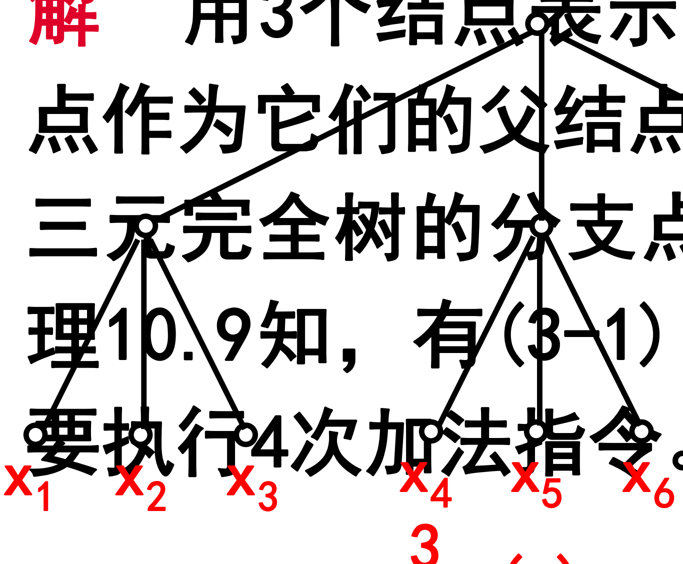
即

$$(k-1)\times i = t-1$$

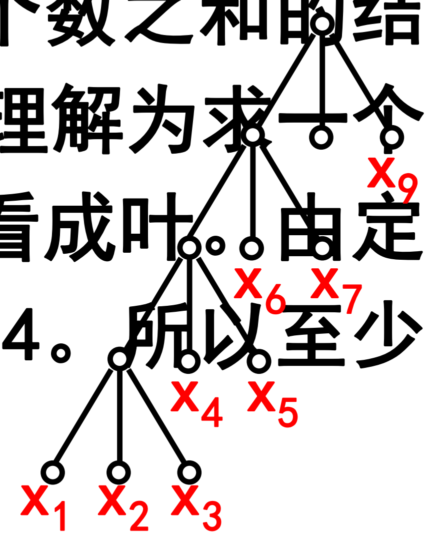
例10.3.4

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　用3个结点表示3个数，将表示3个数之和的结点作为它们的父结点。这样本问题可理解为求一个三元完全树的分支点问题。把9个数看成叶。由定理10.9知，有(3−1)i = 9−1，得i = 4。所以至少要执行4次加法指令。



(a)



(b)

一个多种解法的例子

例10.3.5　设T为任意一棵二元完全树，m为边数，t为叶数，试证明：$m = 2t-2$。这里$t \geq 2$。

证明　方法一：设T中的结点数为n，分支点数为i。根据二元完全树的定义，容易知道下面等式均成立：

$$n = i+t，m = 2i，m = n-1$$

解关于m，n，i的三元一次方程组得

$$m = 2t-2。$$

方法二：

在二元完全树中，除树叶外，每个结点的出度均为2；除根结点外，每个结点的入度均为1。设T中的结点数为n，由握手定理可知

$$2m = \sum_{i=1}^{n}\deg(v_i) = \sum_{i=1}^{n}\deg^{+}(v_i) + \sum_{i=1}^{n}\deg^{-}(v_i)$$

$$= 2(n-t)+n-1 = 3n-2t-1 = 3(m+1)-2t-1$$

故

$$m = 2t-2。$$

方法三：

对树叶数t作归纳法。

当t = 2时，结点数为3，边数m = 2，故m = 2t−2成立。

假设t = k(k≥2)时，结论成立，下面证明t = k+1时结论也成立。

由于T是二元完全树，因此T中一定存在都是树叶的两个兄弟结点v_1，v_2，设v是v_1，v_2的父亲。在T中删除v_1，v_2，得树T’，T’仍为二元完全树，这时结点v成为树叶，树叶数t’ = t−2+1 = k+1−1 = k，边数m’ = m−2，由归纳假设知，m’ = 2t’−2。所以m−2 = 2(t−2+1)−2，故m = 2t−2。

方法四：

对分支点数i作归纳法。

当i=1时，边数m=2，树叶数t=2，故m=2t-2成立。

假设i = k(k≥1)时，结论成立，下面证明i = k+1时结论也成立。

由于T是二元完全树，因此T中一定存在两个儿子都是树叶的分支点，设v就是这样一个分支点，设它的两个儿子为v_1,v_2 。在T中删除v_1,v_2，得树T’，T’仍为二元完全树，这时结点v成为树叶，分支点数i’=i-1=k+1-1= k，树叶数t’ = t-2+1，边数m’ = m-2，由归纳假设知，m’ = 2t’ -2。所以m-2=2(t-2+1)-2，故m=2t-2。

10.3.2 根树的遍历

对于根树，一个十分重要的问题是要找到一些方法，能系统地访问树的结点，使得每个结点恰好访问一次，这就是根树的遍历(Ergode)问题。

k元树中，应用最广泛的是二元树。由于二元树在计算机中最易处理，下面先介绍二元树的3种常用的遍历方法，然后再介绍将任意根树转化为二元树。

算法10.3.1

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。

算法10.3.2

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。

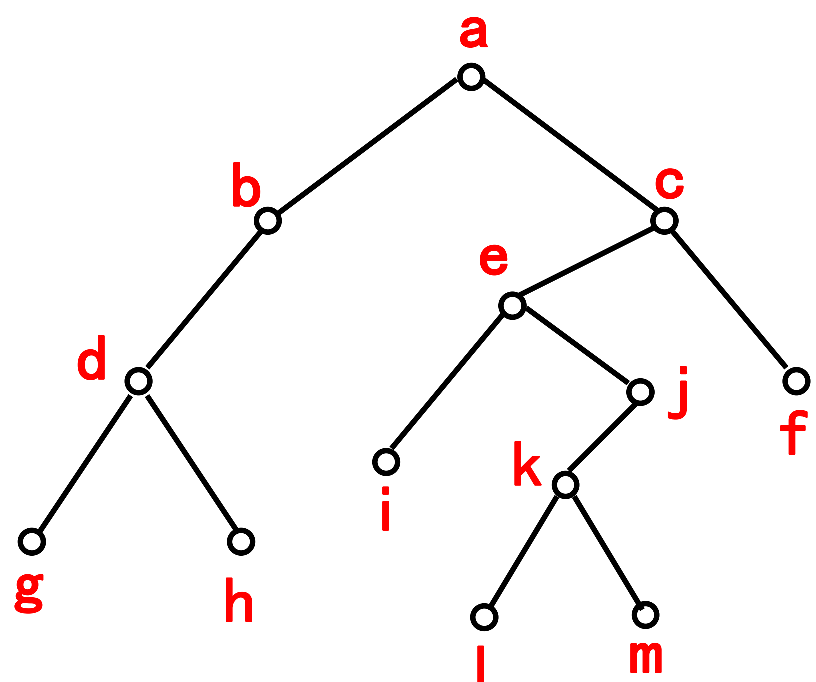
算法10.3.3

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。

例10.3.6

写出对图中二元树的3种遍历方法得到的结果。



解析 根据遍历方法容易写出：只要先将该树分解为根、左子树、右子树三部分，然后再对子树作分解，直到叶为止。

先根遍历次序为abdghceijklmf；

中根遍历次序为gdhbaielkmjcf；

后根遍历次序为ghdbilmkjefca。

算法10.3.4

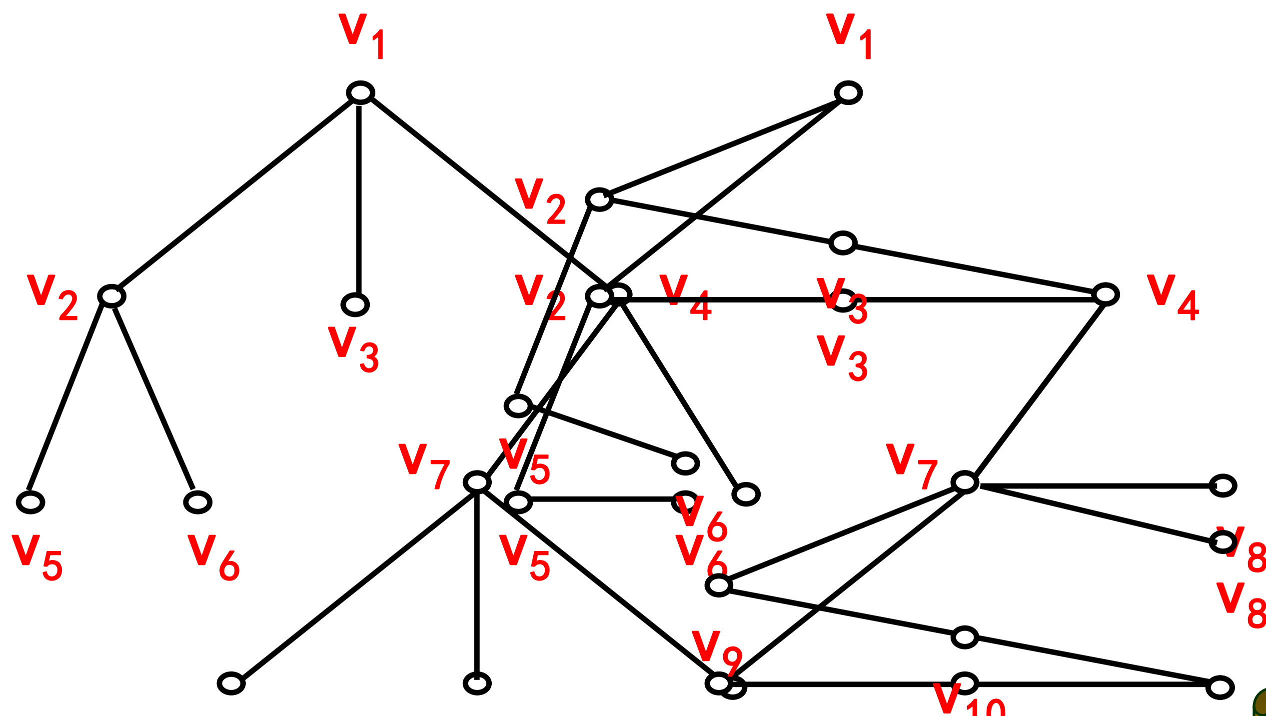
根树转化为二元树算法：

1. 从根开始，保留每个父亲同其最左边儿子的连线，撤销与别的儿子的连线。
2. 兄弟间用从左向右的有向边连接。
3. 按如下方法确定二元树中结点的左儿子和右儿子：直接位于给定结点下面的结点，作为左儿子，对于同一水平线上与给定结点右邻的结点，

转化的要点：弟弟变右儿子

例10.3.7

将下图转化为一棵二元树。



反过来，也可以还原。要点：右儿子变弟弟

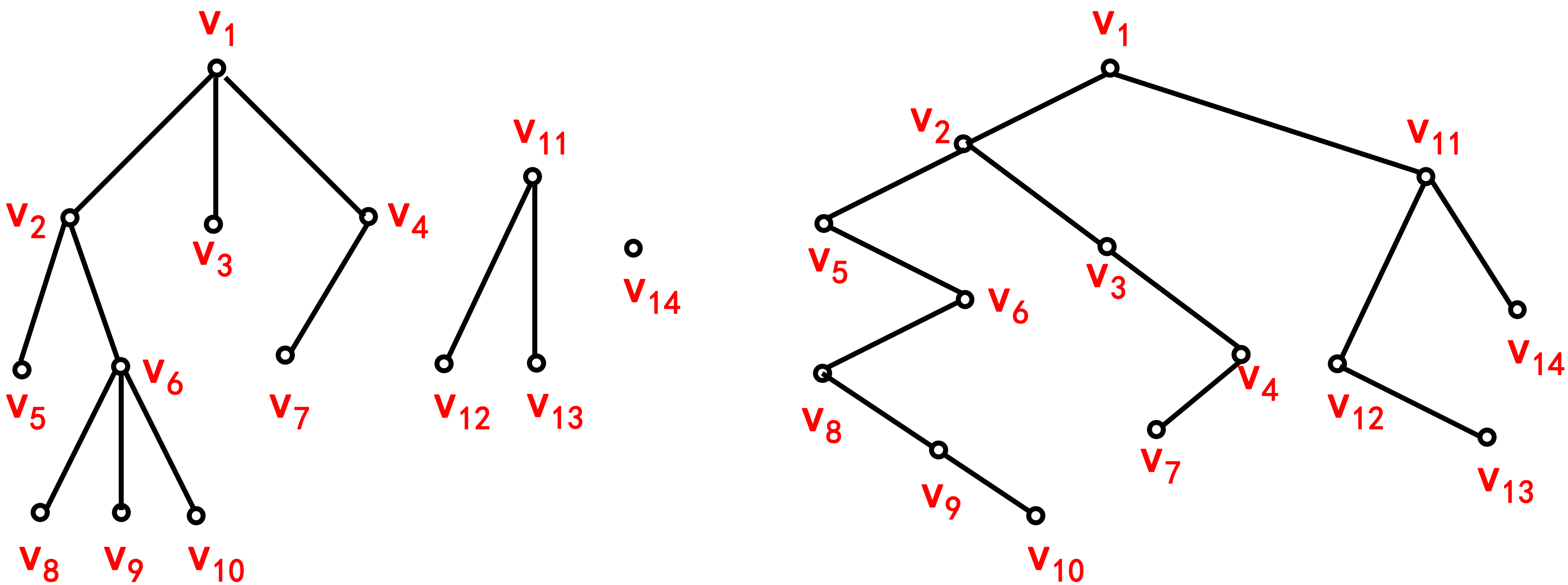
有序森林业转换成二元树

算法10.3.5　森林转化为二元树算法：

1. 先把森林中的每一棵树都表示成二元树；
2. 除第一棵二元树外，依次将剩下的每棵二元树作为左边二元树的根的右子树，直到所有的二元树都连成一棵二元树为止。

例10.3.8

将图所示的森林转化成一棵二元树。



将二元树转化为森林，要点是"先将根的右子树变为新二元树，再将这些二元树还原为根树"。

10.3.3 最优树与哈夫曼算法

在计算机及通讯事业中，常用二进制编码来表示符号，称之为码字(codeword)。例如，可用00、01、10、11分别表示字母A、B、C、D。如果字母A、B、C、D出现的频率是一样的，传输100个字母用200个二进制位。但实际上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%。能否用不等长的二进制序列表示字母A、B、C、D，使传输的信息的二进制位尽可能少呢？事实上，可用000表示字母D，用001表示字母C，01表示B，1表示A。这样表示，传输100个字母所用的二进制位为

$$3\times5+3\times20+2\times25+1\times50 = 175。$$

这种表示比用等长的二进制序列表示法好，节省了二进制位。但当我们用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。因而，不能用这种二进制序列表示A、B、C、D，要寻找另外的表示法。

定义10.3.6

{1，01，001，000}是前缀码

{1，11，001，0011}不是前缀码。

设A = {b_1，b_2，…，b_m}是一个符号串集合，若对任意b_i，b_j∈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**(Prefixed Code)。若符号串b_i(i = 1，2…，m)中，只出现0和1两个符号，则称A为**二元前缀码**(Binary Prefixed Code)。

用二元树产生二元前缀码

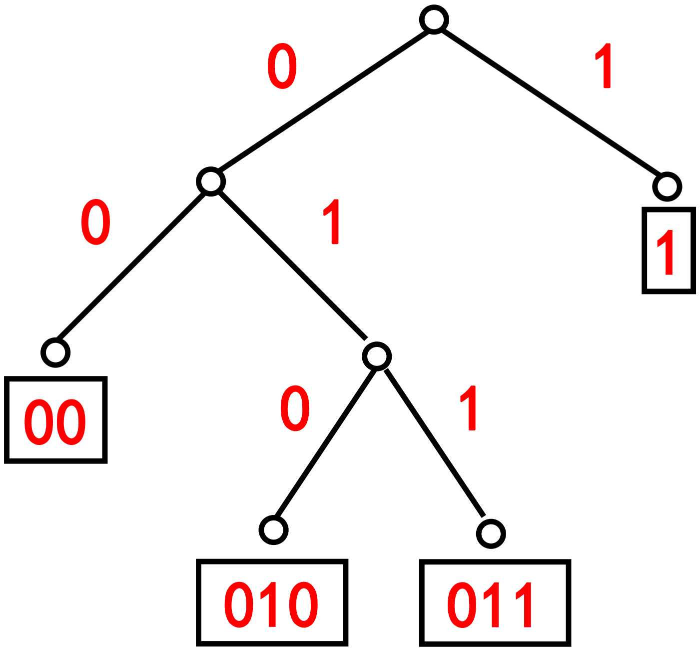
给定一棵二元树T，假设它有t片树叶。

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v_i为T

v_i中的符号串的前缀均在v_i所在的通路上

若T存在带一个儿子的分支点，则由T产生的前缀码不唯一，但T若为完全二元树，则T产生的前缀码就是唯一的了。

例



图中所示的二元树产生的前缀码为：{1，00，010，011}。

因此用1表示A，用00表示B，用010表示 C，用011表示 即可满足要求。

当知道了传输的符号出现的频率时，如何选择前缀码，使传输的二进制位尽可能地少呢？
先产生最优二元树T，然后用T产生二元前缀码。

最优树

定义10.3.7 设有一棵二元树，若对其所有的t片叶赋以权值w_1、w_2、…、w_t，则称之为**赋权二元树**(Power Binary Tree)；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、…、w_t的二元树中，W(T)最小的二元树称为**最优树**(Optimal Tree)。

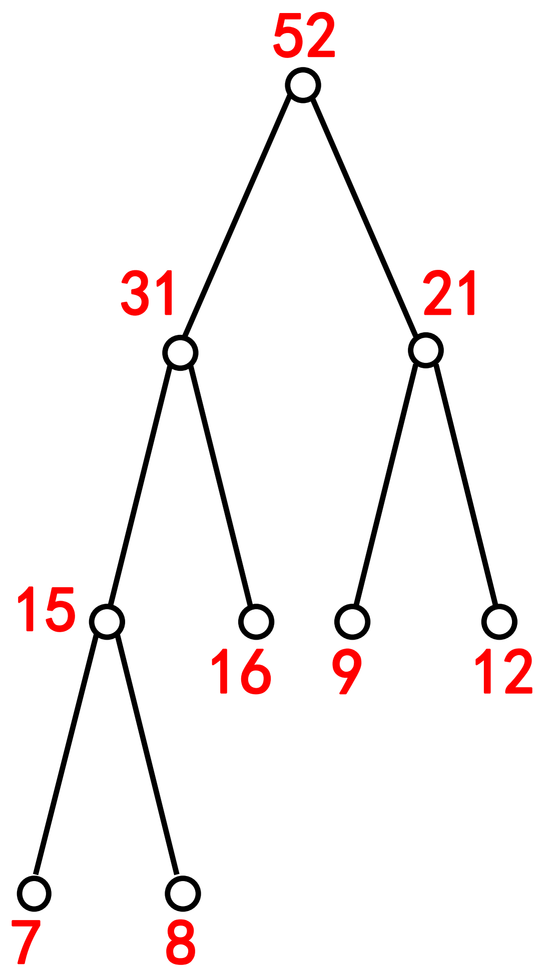
1952年哈夫曼(Huffman)给出了求最优树的方法。该方法的关键是：从带权为W_1+W_2、W_3、…、W_t(这里假设$W_1 \leqslant W_2 \leqslant \cdots \leqslant W_t$)的最优树T'中得到带权为$W_1$、$W_2$、…、$W_t$的最优树。

算法10.3.6 哈夫曼算法：

1. 初始：令S＝｛W_1，W_2，…，W_t｝；
2. 从S中取出两个最小的权W_i和W_j，画结点v_i，带权W_i，画结点v_j，带权W_j。画v_i和v_j的父亲v，连接v_i和v，v_j和v，令v带权W_i+W_j；
3. 令S＝（S－｛W_i，W_j｝）∪｛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。

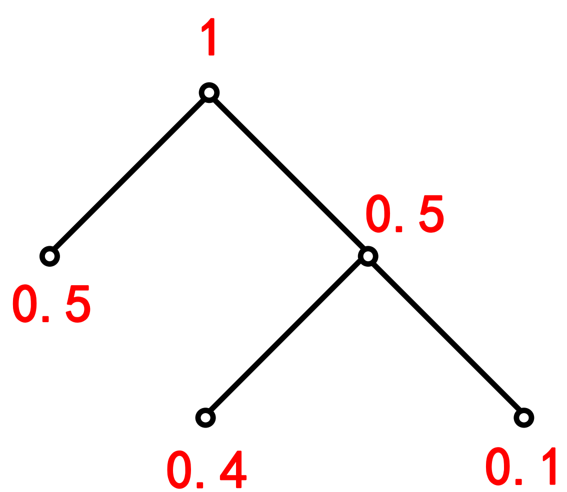
例10.3.9

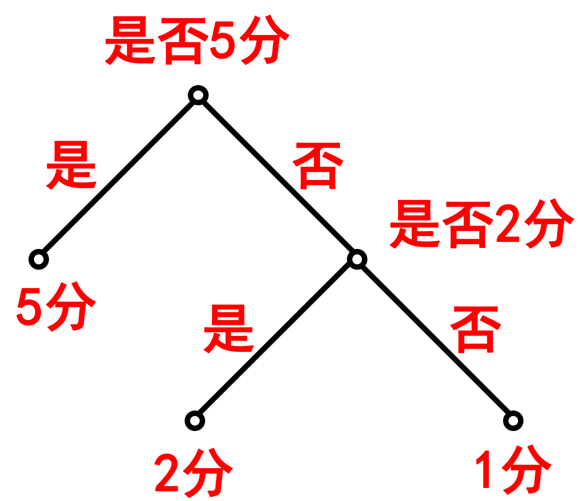
求带权7、8、9、12、16的最优树。



例10.3.10

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：2×0.1+2×0.4+1×0.5 ＝ 1.5(时间单位)。

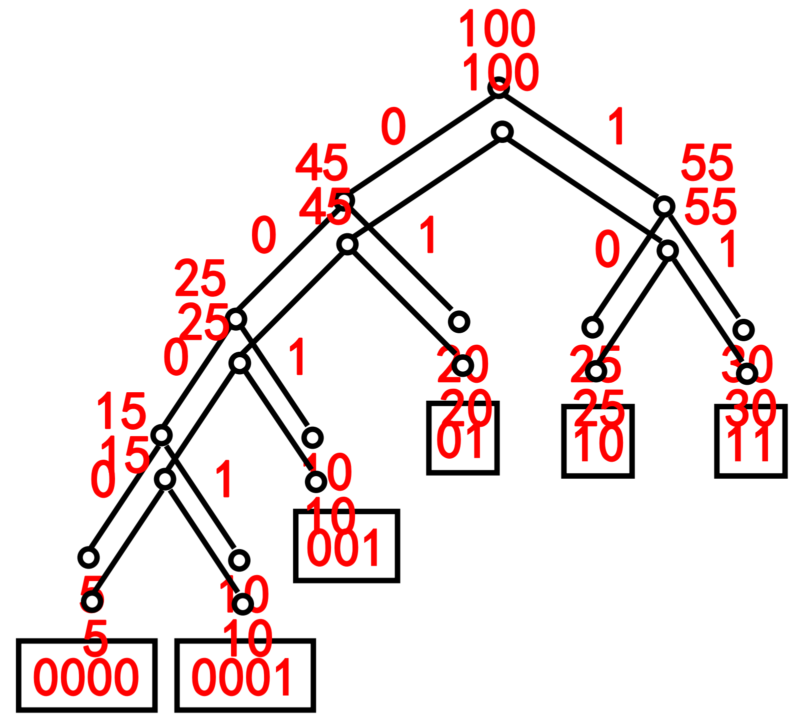
例10.3.11

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。



解

（1）求带权30, 25, 20, 10, 10, 5的最优二元树T。

（2）在T上求一个前缀码。

（3）设树叶v_i带权为w%×100 = w，则v_i处的符号串表示出现频率为w%的字母。

{01，10，11，001，0001，0000}

为一前缀码，其中

0000表示F，　　0001表示E，

001表示D，　　01表示C，

10表示B，　　11表示A。

传输100个这样的字母所用的二进制位为：

$$4\times(5+10)+3\times10+2\times(20+25+30) = 240。$$

10.3.4 根树的难点

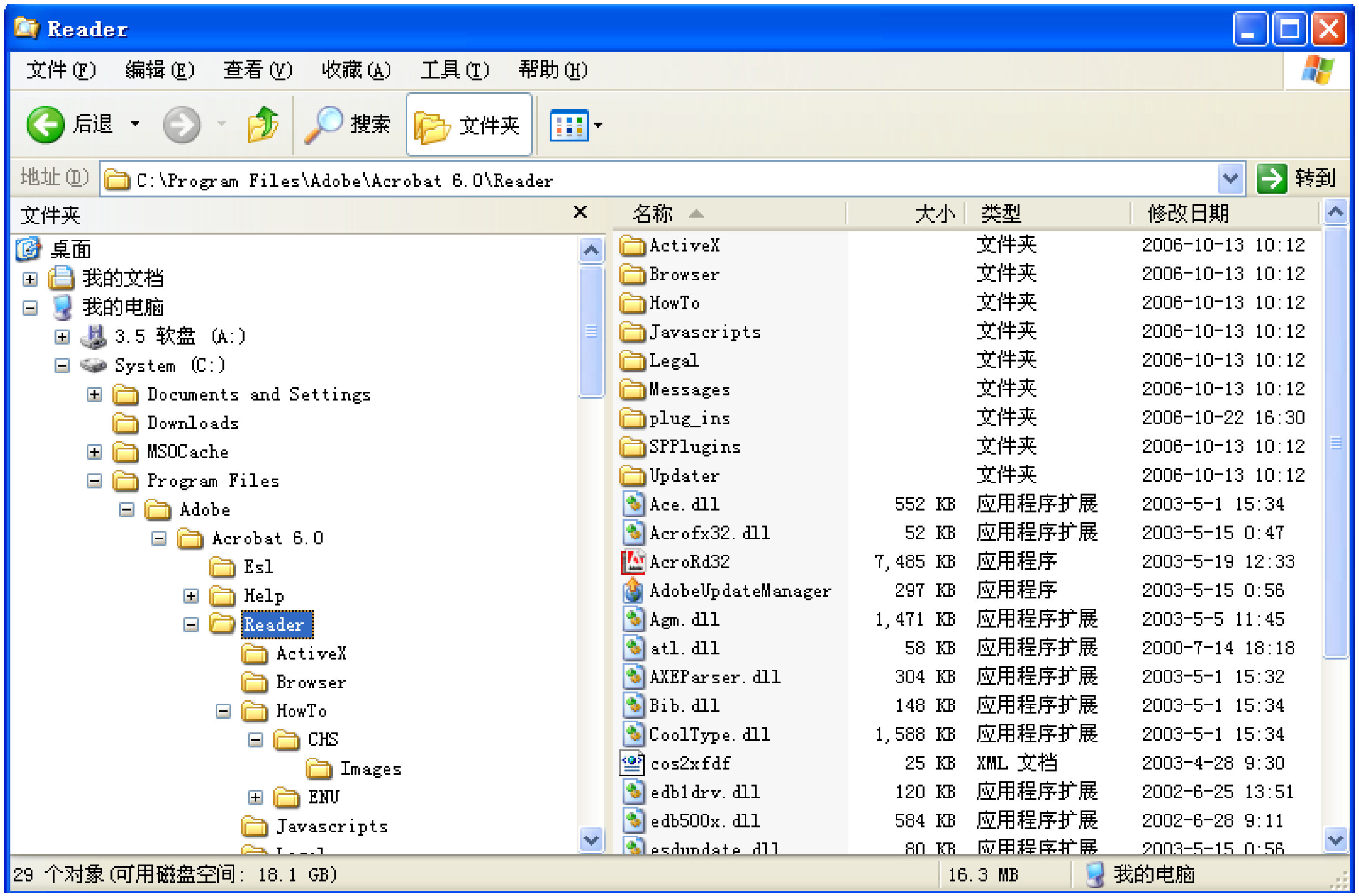
1. 有向树是略去所有边的方向所得无向图为无向树的有向图；根树是只有一个结点的入度为0，其余结点的入度均1的有向树；注意根树的叶和分支点与无向数的叶和分支点的细微差别；
2. 有序树是对结点标定了顺序的根树；
3. k元树的每个分支点至多有k个儿子，因此其出度均小于等于k，一般取满足条件的最小值k；k元完全树的每个分支点都恰有k个儿子，因此其出度均恰好等于k；

根树的难点

4. 注意区分以v为根的子树和v的左子树、右子树；
5. 二元树遍历的3种方法：先根次序遍历法、中根次序遍历法、后根次序遍历法；
6. 有序树、森林与二元树之间的相互转换；
7. 一棵根树也可看成一个家族，若边<a, b>在树中，则称a是b的父亲，b是a儿子。若<a, b>, <a, c>都在树中，且b≠c, 则b, c称为兄弟。若a可达b, 则称a是b的祖先，b是a的后代；
8. 最优树与哈夫曼算法。

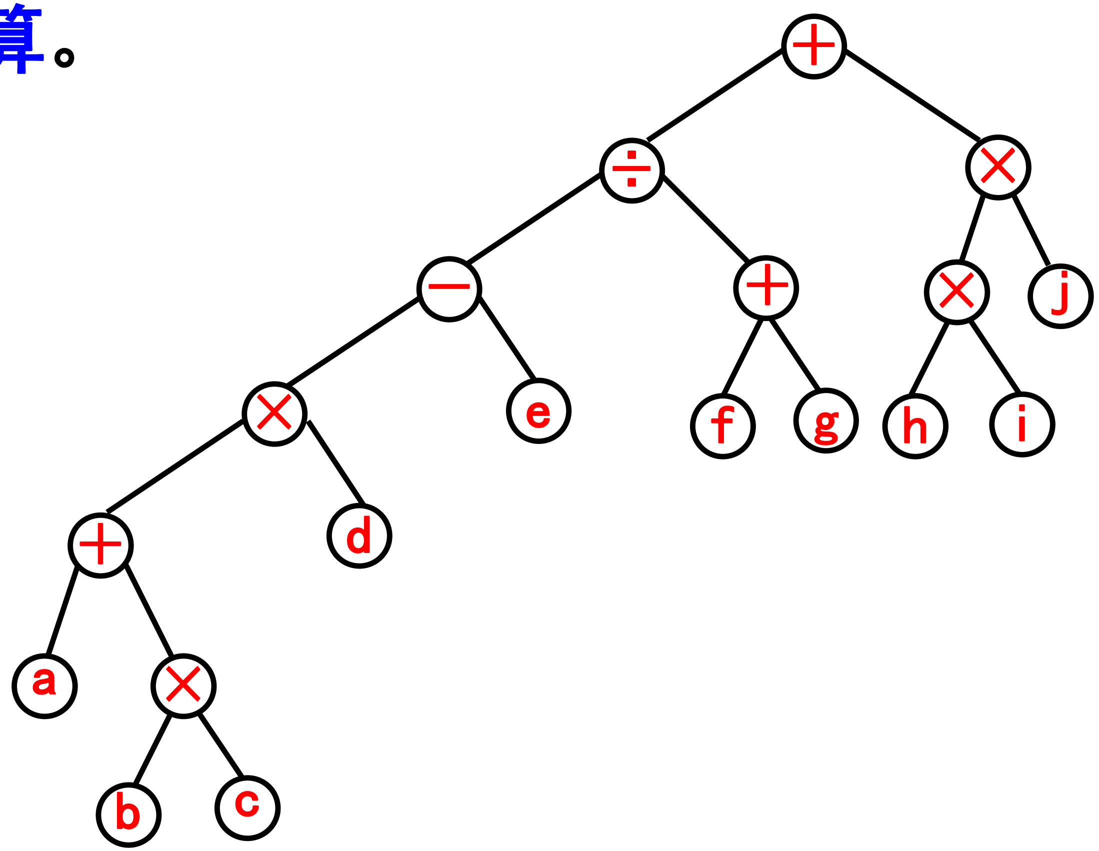
10.3.5 根树的应用

1、计算机的文件结构



2、波兰符号法与逆波兰符号法

规定用树叶表示参加运算的元素，分支结点表示相应的运算。



((a+(b×c))×d−e)÷(f+g)+(h×i)×j

中缀符号法

按中根次序遍历算法访问T，其结果为

$$((((a+(b\times c))\times d)-e)\div(f+g))+((h\times i)\times j),$$

根据运算符的优先次序可以省去部分括号，得

$$((a+b\times c)\times d-e)\div(f+g)+h\times i\times j。$$

因为运算符夹在两数之间，故称此种表示法为中缀符号法。

波兰符号法

按先根次序遍历算法访问T，其结果为

$+(\div(-(\times(+a(\times bc))d)e)(+fg))(\times(\times hi)j)$，

省去全部括号后，规定每个运算符对它后面紧邻的两个数进行运算，仍是正确的。因而可省去全部括号，其结果为

$$+\div-\times+a\times bcde+fg\times\times hij。$$

因为运算符在参加运算的两数之前，故称此种表示法为前缀符号法，或称为波兰符号法。

逆波兰符号法

按后根次序遍历算法访问T，其结果为

((((a(bc×)+)d×)e−)(fg+)÷)((hi×)j×)+,

省去全部括号后，规定每个运算符对它前面紧邻的两个数进行运算，仍是正确的。因而可省去全部括号，其结果为

abc×+d×e−fg+÷hi×j×+,

因为运算符在参加运算的两数之后，故称此种表示法为后缀符号法，或称为逆波兰符号法。

3、决策树

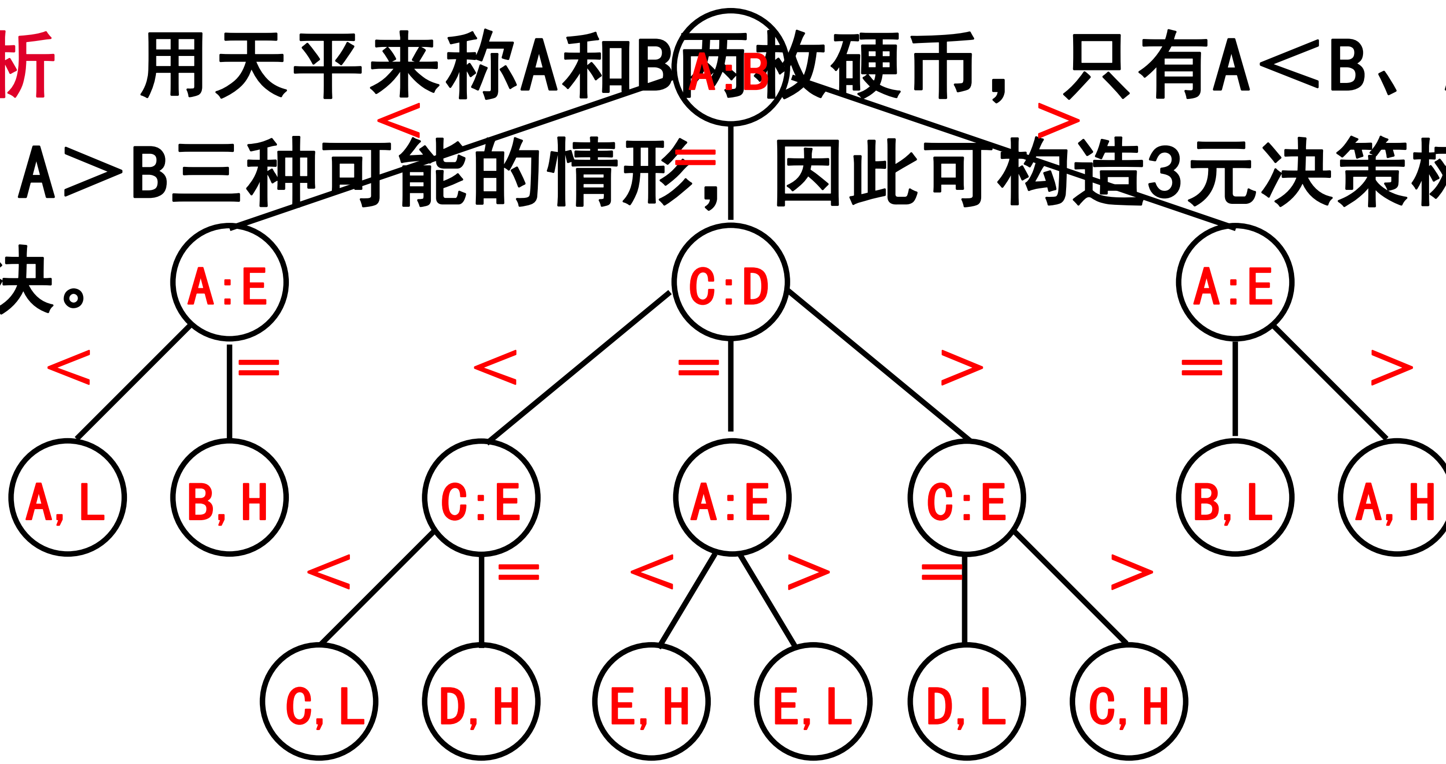
定义10.3.8 设有一棵根树，如果其每个分支点都会提出一个问题，从根开始，每回答一个问题，走相应的边，最后到达一个叶结点，即获得一个决策，则称之为决策树(Decision Tree)。

下面我们用决策树表示算法，并使得在最坏情形下花费时间最少。

5硬币问题

从根到叶就是一种求解过程，由于该树有10片叶子，因此最多有10种可能的解。又由于该树高为3，因此最坏情形下需要3次判别就能得到结论。

分析 用天平来称A和B两枚硬币，只有A＜B、A ＝ B、A＞B三种可能的情形，因此可构造3元决策树来解决。



4、博弈树

实际生活中有许多有益的博弈，与围棋、象棋、五子棋等，每名选手交替动作，直至结束。

下面介绍如何将根树应用到博弈比赛策略的研究中。

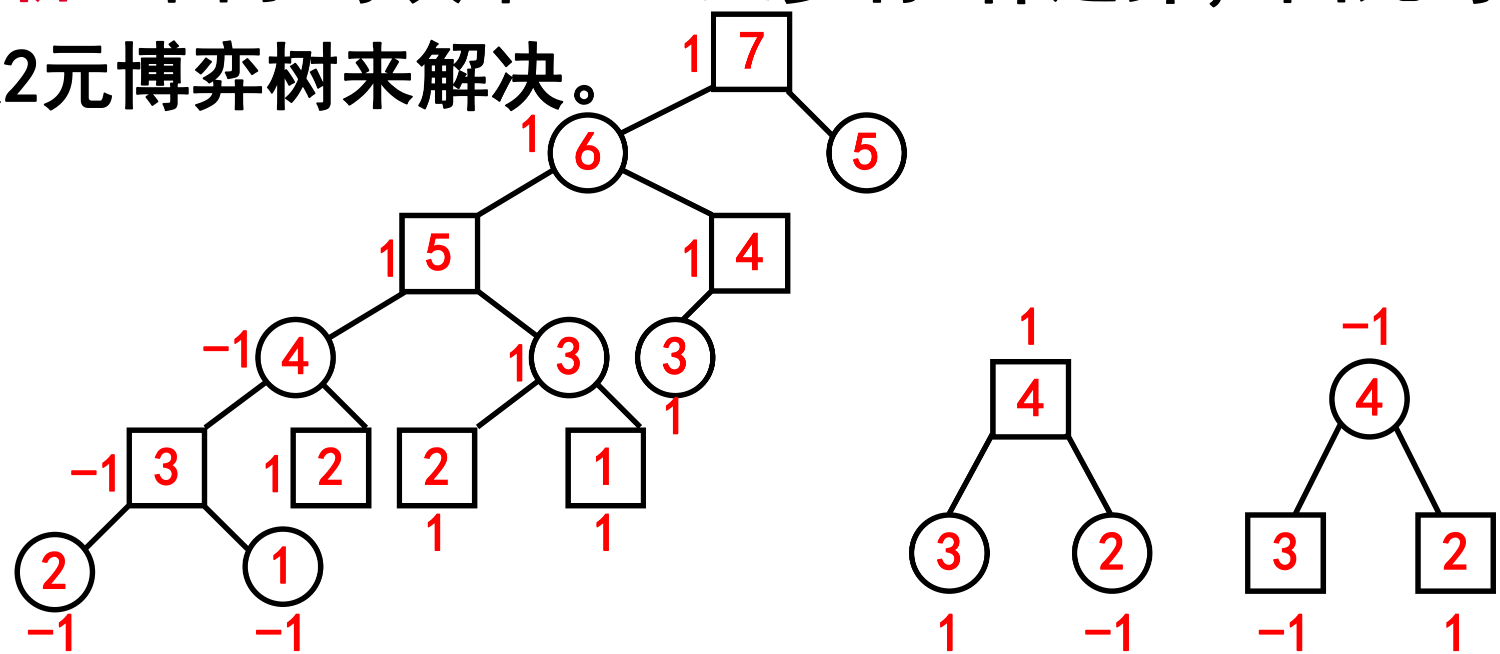
这种方法已应用到很多的计算机程序研究中，使得人类可以同计算机比赛，或者甚至计算机同计算机比赛。

作为一般的一个例子，考虑一个取火柴的博弈。

例10.3.14

现有7根火柴，甲、乙两人依次从中取走1根或2根，但不能不取。取走最后一根的就是胜利者。

分析 由于每次甲、乙至多有2种选择，因此可构造2元博弈树来解决。



(a) (b) (c)

10.4 本章总结

1 主要知识点汇集

2 习题类型

3 解题分析和方法

1、主要知识点汇集

① 树的概念： 树、森林、根树、根、叶、分支点、生成树、最小生成树等。

② 树的基本性质：$m = n-1$等。

③ 与根树相关的概念：有向树、根树、根、叶、内点、分支点、层数、高、有序树、祖先与后代、父亲与儿子、k元树、k元完全树、k元有序树、k元有序完全树、子树、根树的遍历、最优树。

④ 树的算法： 破圈法、避圈法、广度优先搜索算法、Kruskal算法、Prim算法、哈夫曼算法、二元树的先(中、后)根次序遍历算法、根树转化为二元树算法、森林转化为二元树算法。

⑤ 树的应用：最小费用问题、计算机的文件结构、波兰符号法与逆波兰符号法、决策树、博弈树。

2、习题类型

① 基本概念题：主要观测点在于树的基本概念、分类；

② 判断题：主要观测点在于判定是否是树、生成树等；

③ 计算题：主要观测点在于叶和分支点的数目、利用现有算法计算相应的问题等；

④ 证明题：主要观测点在于树性质的证明

3、解题分析和方法

① 利用树的性质：m=n-1，将其与握手定理配合使用，在有关无向树、有向树的很多证明和计算中都很重要；

② 构造性证明方法的使用，只有找到了就一定存在；

③ 利用各种问题的现有算法，可机械地计算求解；

④ 对于某些问题，可以尝试用多种方法求解；

⑤ 反证法非常有用，特别是在证明惟一性和不存在的时候。

第七次作业（第十章课后习题）

1.
3.
7.
10.
11.
14.
19.

Thank You！

http://202.115.21.136:8080/lssx