离散数学

大连理工大学软件学院

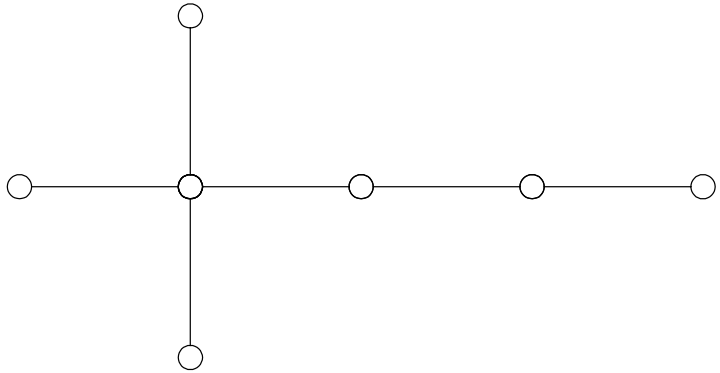
本章内容

- 树与生成树
  - 树及其性质
  - 生成树与最小生成树
- 有向树及其应用
  - 有向树
  - m叉树
  - 有序树
  - 二叉树的遍历
  - 搜索树

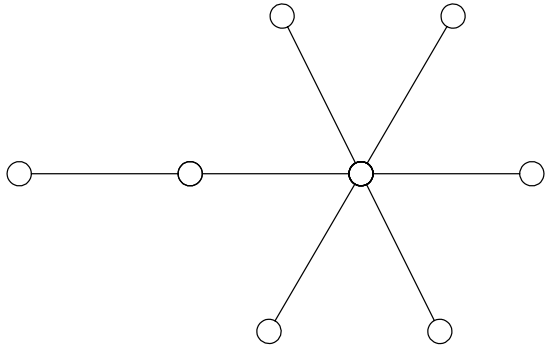
11.1 树与生成树

树是在计算机科学中应用最广泛的一类特殊图。

定义：自由树形或无根树形定义为没有回路的连通无向图。



(a)

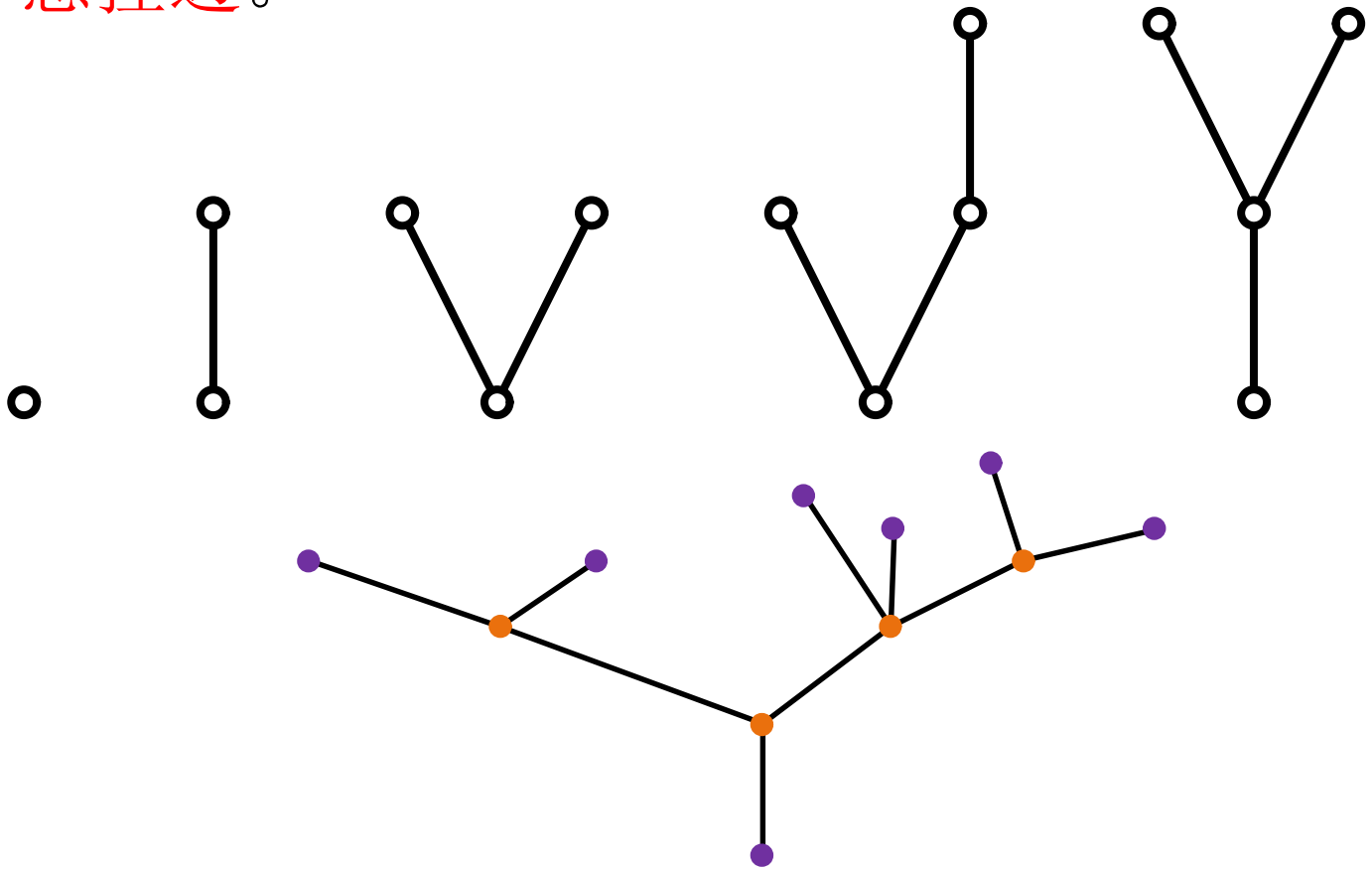


(b)

从定义可以看出：树是一个简单图，没有自环，没有平行边。

树

- 在树中，度为1的结点称为叶结点或悬挂结点；
- 度数大于1的结点称为内结点或分枝结点；
- 而与叶结点或悬挂结点所关联的边，称为叶边或悬挂边。



树

定理：如果G是一个无向图，则下列命题等价：

(1)G是一个自由树形

(2)G是连通的，但若删去任何一条边，则所得图形就不再连通。

(3) 对于G的两个不同结点v和v'恰有一条从v到v'的基本通路。

又当G有限时，记G结点数为$n > 0$，则下列命题也与前诸命题等价：

(4)G不含回路而且有n-1条边。

(5)G连通而且有n-1条边。

树

证：(1) ⇒ (2)

- 倘若删去边$\{v, v'\}$后G仍然连通，则按连通性定义有通路$(vv_1 \cdots v_i \cdots v')$从$v$到$v'$。
- 进而有基本通路$(vv_1 \cdots v_i \cdots v')$，这是因为，取通路之中最短者，则必得一基本通路。
- 所以，若有$v_j = v_k$，对某个$j < k$,则

$$(v \cdots v_j v_{k+1} \cdots v_n)，\ v_n = v'$$

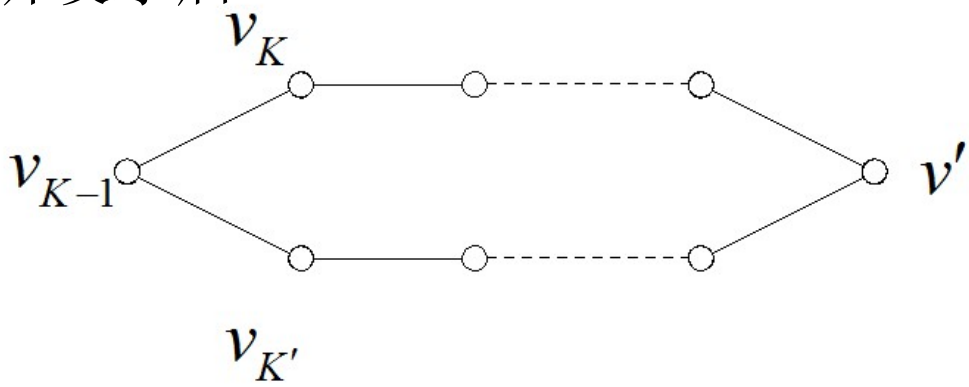
将是一更短的如此之通路，总之，从v到v'有一条基本通路$(vv_1 \cdots v_i \cdots v')$。这条通路的长度≥2。

- 于是，$(vv_1 \cdots v_i \cdots v'v)$将是$G$中的一条回路。这与自由树形的定义矛盾。得证。

树

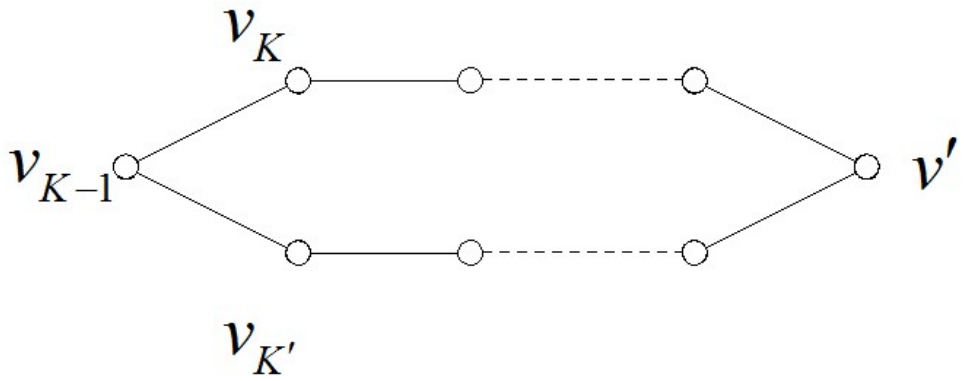
证：(2) ⇒ (3)

- G连通，故对于结点v和v'有一条通路从v到v'。
- 仿前一段证明，如果取此通路中最短者，即得一条从v到v'的基本通路。
- 现在证最多有一条这样的基本通路。
  - 事实上，倘若有两条从v到v'的基本通路$(vv_1 \cdots v_i \cdots v')$和$(vv_1' \cdots v_i' \cdots v')$，则自左而右，可找到最小的$k \geq 1$，使$v_k \neq v_k'$
  - 这样，从G中删去一条边$\{v_{k-1}, v_k\}$后仍将是一个连通图形，因为从v_{k-1}到v_k还有通路$(v_{k-1}v_k' \cdots v' \cdots v_k)$，如下图所示。这与(2)所设矛盾。



树

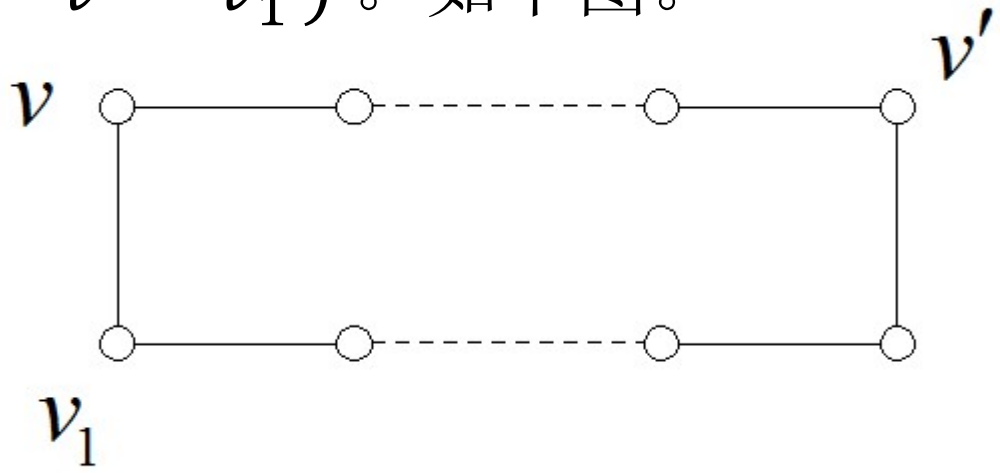
证： (2) ⇒ (3)既设G连通，故对于结点v和v'有一条通路从v到v'。仿前一段证明，如果取此通路中最短者，即得一条从v到v'的基本通路。现在证最多有一条这样的基本通路。事实上，倘若有两条从v到v'的基本通路$(vv_1 \cdots v')$和$(vv_1' \cdots v')$，则自左而右，可找到最小的$k \geq 1$，使 $v_k \neq v'_k$。这样,从G中删去一条边$\{v_{k-1}, v_k\}$后仍将是一个连通图形，因为从v_{k-1}到v_k还有通路$(v_{k-1}v_k' \cdots v' \cdots v_k)$，如下图所示。这与(2)所设矛盾。



树

证：(3) ⇒ (1)

- 既然恰有一条基本通路从v到v'，故G自然连通。
  - 因为连通性只要求有(不必恰有)一条通路(不必为基本)从v到v'。
- 再证G无回路。
  - 事实上，倘若G有回路$(vv_1 \cdots v' \cdots v)$，则从$v$到$v_1$将有两条基本的通路$(vv_1)$和$(v \cdots v' \cdots v_1)$。如下图。



树

设G有限，要证(1)～(5)的等价性，为此，先证明引理F。

引理F：设G为任意有限图形，G无回路但至少有一边，则至少有一个结点恰好相邻于别的一个结点。

- 证：既然G至少有一条边，故可取一结点v_1以及v_1的一个相邻结点v_2。要证其中必有结点恰有一个相邻者。为此，从v_1出发，看v_2。
- 对于$k \geq 2$，或者v_k只与v_{k-1}相邻，从而v_k即为所求者；或者v_k又相邻于别的结点$v_{k+1} \neq v_{k-1}$。如此类推，得一串结点

$$v_1, v_2 \cdots, v_{k-1}, v_k, v_{k+1}, \cdots$$

- 既设G无回路，所以这一串结点不能重复。
- 但G有限，故以上过程必停止于有限步。
  - 即最终可得一结点v_k只与v_{k-1}相邻。证毕。

树

证：(1) ⇒ (4)，对图的结点数用归纳法。

- n=1时，显然定理成立。
- 设n–1时成立，往证n时亦成立。
  - 即设G有n>1个结点：连通，无回路；来证G无回路，有n-1条边。
- 这只要证G有n-1条边即可。
  - 为此，利用引理并命v_n为恰有一相邻结点v_{n-1}者。
- 删去v_n和边$\{v_{n-1}, v_n\}$，得一新图形G'。G'有n-1个结点，而且无回路还连通。
- 这是因为v_n只有一个相邻结点，故v_n就不能出现在G的任何一个结点v到另一个结点v'的基本通路的中间处，而至多只能出现在前头或后头的两个端点处（因为中间处的结点有前后两个相邻结点），从而边$\{v_{n-1},v_n\}$也至多出现在两头（$\{v_{n-1},v_n\}$出现在两头，当且仅当v_n出现在两端）。

树

- 由此可见，原来在G中任何两个结点之间恰有一条基本通路，则经删去v_n和边$\{v_{n-1},v_n\}$之后，现在在G'中任何两个结点之间仍恰有一条基本通路。
- 根据已证的(3)与(1)的等价性即知，G是自由树形，现在G'仍是自由树形。
- 今G'有n-1个结点，按归纳假设，G'有$(n-1)-1=n-2$条边。所以，G有$(n-2)+1=n-1$条边。归纳法完成。

树

证：(4) ⇒ (5)，还是对结点数n>0使用归纳法。

- n=1显然。
- 假定n-1已证，来证n。即设G有n>1个顶点，n-1>0条边，无回路。来证G有n-1条边，连通。
- 这只要证G连通。
  - 仿前一小段证明，利用引理F命v_n为恰有一相邻结点v_{n-1}者，删去v_n和$\{v_{n-1},v_n\}$而得一新图形G'。G'有n-1个结点，n-2条边，且无回路。
  - 故按归纳假设，G'连通。
  - 如果是这样的话，G亦连通，因为把$\{v_{n-1},v_n\}$和v_n添加到G'之后，G中任何两结点之间仍必有通路（必要时取道(v_{n-1},v_n)，如果其中有一结点是新添加的v_n的话。因为G'中任一结点必可通达v_{n-1}。
- 归纳法完成。

树

- 证：(5) ⇒ (1)即设G有$n>0$个结点，n-1条边、连通，要证G连通、无回路。
- 这只要证G无回路即可。
  - 设G含有一个回路。则删去回路上任何一条边后所得的图形仍连通。
  - 用这种方法，我们可以不断地删去一些边，最后得出一个图形G'仍连通，G'具有n-1-k条边，且无回路，即G'为一自由树形。
  - 由已证的(1)蕴含(4)的事实，G'有n-1条边，可见k=0。
  - 也就是说原来，G=G',即G中原来就无回路。

综上，树的第一个定理全部证完。

森林

树是非循环连通无向图，如果去掉对连通性的要求，就得到森林的概念。

定义：每个分支都是树的无向图称为森林。

定理： 如果森林F有n个结点、m条边和k个分支，则

$$m = n - k$$

11.1.2 生成树

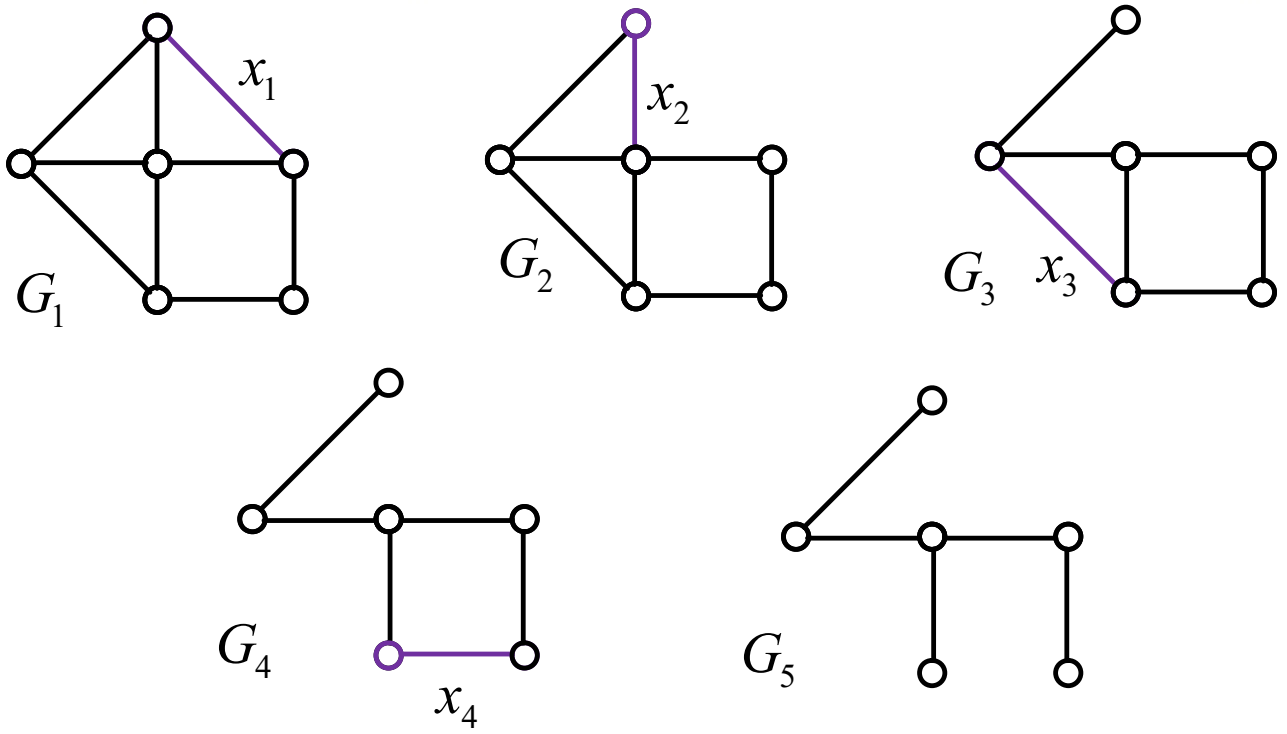
推论1：任何有限连通图G必有一结点数相同的自由子树形，或称G的生成树，记作T_G。

- 事实上，根据（2），尽可能地删去一条条边，即得一自由树形。
- 如果给定的图G是一个(n,m)连通图，根据（5）知，生成树是个$(n,n-1)$图。
- 因而在求得T_G之前，应该删除的边数，应该是$m-(n-1)$。这个数也称为给定图G的基本循环的秩。
- 显然，G的基本循环的秩，是为了打断它的所有基本循环时，必须从G中删除的边数
- 每一条被删除的边，都称作G的弦。T_G中的边称为枝。

生成树不唯一 。

生成树

例：求下列(6,9) 无向图G_1的生成树。

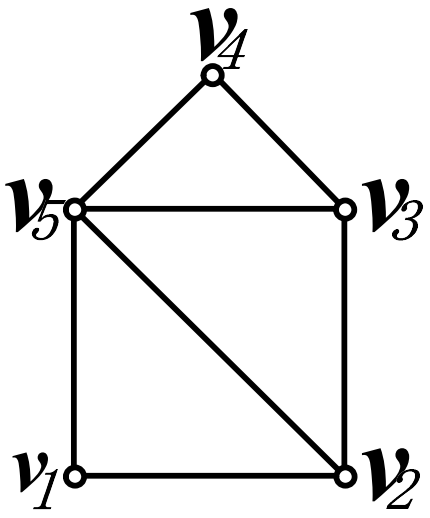


生成树T_G有6-1=5条边；

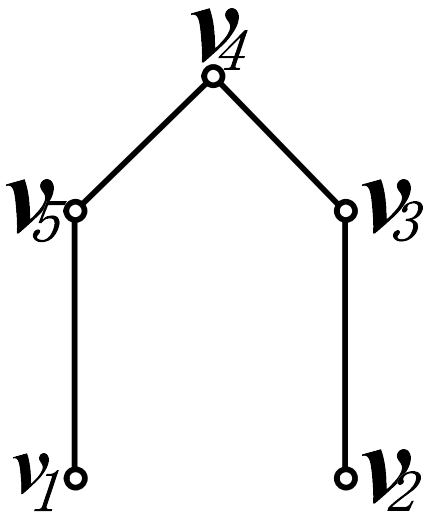
G的基本循环的秩，等于弦的数目，它是9-(6-1)=4

生成树

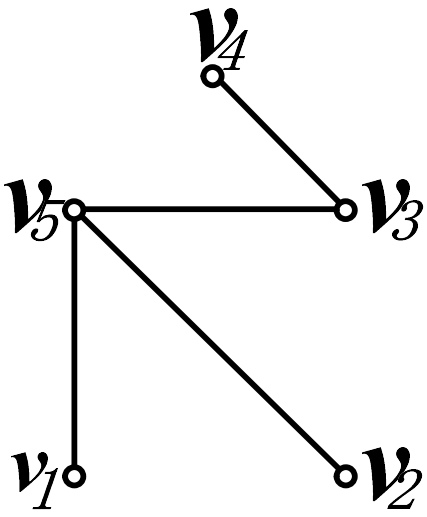
例：求下图的生成树。



G



T_1



T_2

最小生成树

- 定义：
- 设$\langle G, W\rangle$是个加权图，$G' \subseteq G$，G'中所有边的加权长度之和称为G'的加权长度。
- 设G是连通无向图，$\langle G, W\rangle$是加权图，G的所有生成树中加权长度最小者称为$\langle G, W\rangle$的最小生成树。

下面给出一种求最小生成树的方法——Kruskal算法(1956)，它的本质是树生成过程，因此得名避圈法。

最小生成树

设G是有m条边的n阶连通无向图，求加权图$\langle G, W\rangle$的最小生成树的方法如下：

首先，把G的m条边排成加权长度递增的序列$e_1, e_2, \cdots, e_m$，然后执行下面的算法：

(1) $T \leftarrow \varnothing$;

(2) $j \leftarrow 1, i \leftarrow 1$;

(3) 若$j = n$，则算法结束；

(4) 若G的以$T \cup \{e_i\}$为边集合的生成子图没有回路，

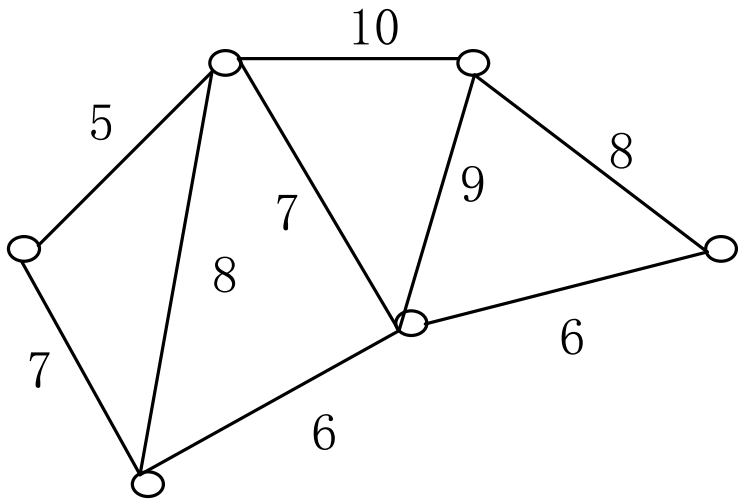
则$T \leftarrow T \cup \{e_i\}$且$j \leftarrow j + 1$;

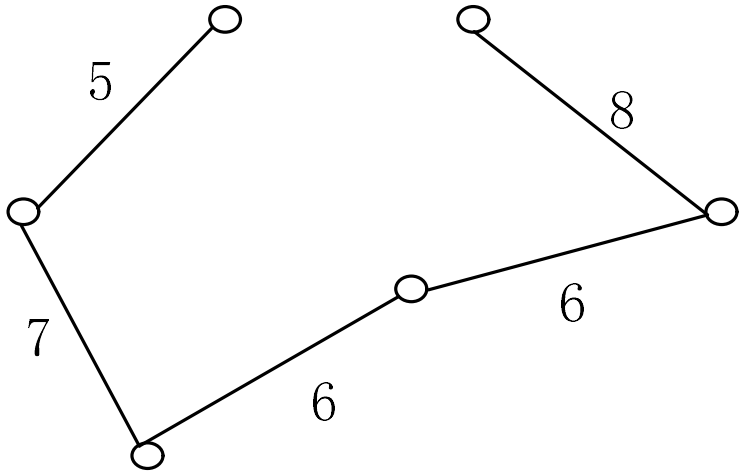
(5) $i \leftarrow i + 1$，转向 (3)。

算法结束时，T即为所求的最小生成树的边的集合。

最小生成树

例：考虑下列道路系统。在冬天里经常下雪，保持道路畅通的唯一方式就是经常扫雪。高速公路部门希望只扫尽可能少的道路上的雪，而确保总是存在连接任何两乡镇的干净道路。

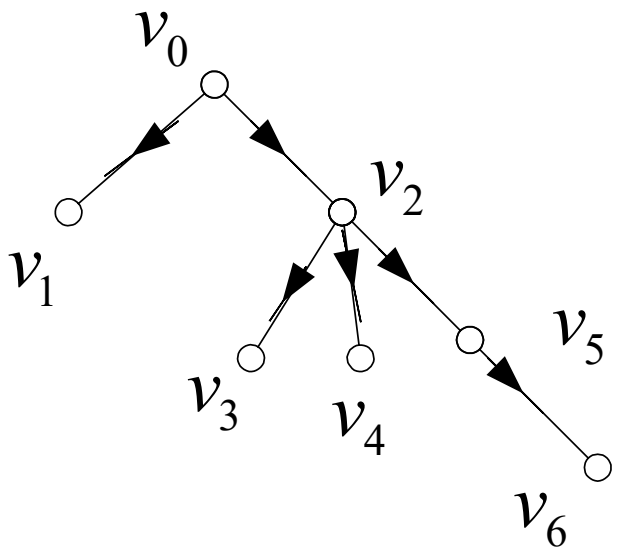




10.2 有向树及其应用

定义：一个结点的入度为0，其余结点的入度均为1的弱连通有向图，称为有向树。

在有向树中，入度为0的结点称为根，出度为0的结点称为叶，出度大于0的结点称为分支结点，从根至任意结点的距离称为该结点的级，所有结点的级的最大值称为有向树的高度。



v_0是根，v_1,v_3,v_4,v_6是叶，
v_0,v_2,v_5是分支结点，

高度为3。

有向树

定理：设v_0是有向图D的结点。D是以v_0为根的有向树，当且仅当从v_0至D的任意结点恰有一条路径。

证：先证必要性。

- 设 $D=\langle V,E,\psi\rangle$是有向树，$v_0$是$D$的根。
- 因为D是弱连通的，取$v'\in V$，从v_0至v'存在半路径，设为$v_0e_1v_1\cdots v_{p-1}e_pv_p$，其中$v_p=v'$。
  - 因为$d_D^-(v_0)=0$，所以e_1是正向边，因为$d_D^-(v_1)=1$，所以e_2是正向边。可归纳证明e_p是正向边。
- 若从v_0至v'有两条路径P_1和P_2，则P_1和P_2的公共点（v_0除外）的入度为2，与D是有向树矛盾。

有向树

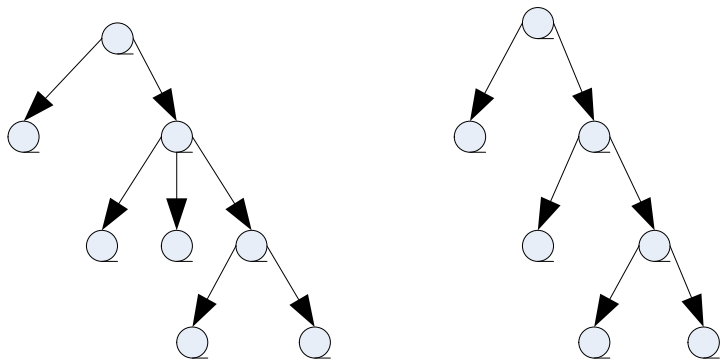
- 再证充分性。
- 显然，D是弱连通的。
- 若$d_D^-(v_0) > 0$，则存在边e以为v_0终点，设v_1是e的起点，P是从v_0至v_1的路径，则在D中存在两条从v_0至v_0的路径Pv_1ev_0和v_0，与已知条件相矛盾，所以$d_D^-(v_0) = 0$。
- 若$d_D^-(v) > 1$，其中v是D的结点，则存在两条边e_1和e_2以v为终点，设e_1和e_2的起点分别是v_1和v_2，从v_0至v_1和从v_0至v_2的路径分别是P_1和P_2，则$P_1v_1e_1v$和$P_2v_2e_2v$是两条不同的从v_0至v的路径，与已知条件矛盾。所以$d_D^-(v) = 1$
- 这就证明了D是有向树且v_0是有向树的根。

有向森林

定义：每个弱分支都是有向树的有向图，称为有向森林。

定义：设$m \in N$，D为有向树。

1. 如果D的所有结点的出度的最大值为m，则D称为m元有向树。
2. 如果对于m元有向树的每个结点v，$d_D^+(v) = m$或$d_D^+(v) = 0$，则称D为完全m元有向树。
3. 若所有的叶结点的级相同，则称正则m元有向树。



（a）　　（b）

二叉树

完全二元有向树也称二叉树，在计算机科学中最有用。

定义：设V是二叉树D的叶子的集合，R_+是全体正实数的集合，W：$V\to R_+$，则称$\langle D,W\rangle$为叶加权二叉树。对于D的任意叶v，$W(v)$称为v的权，称

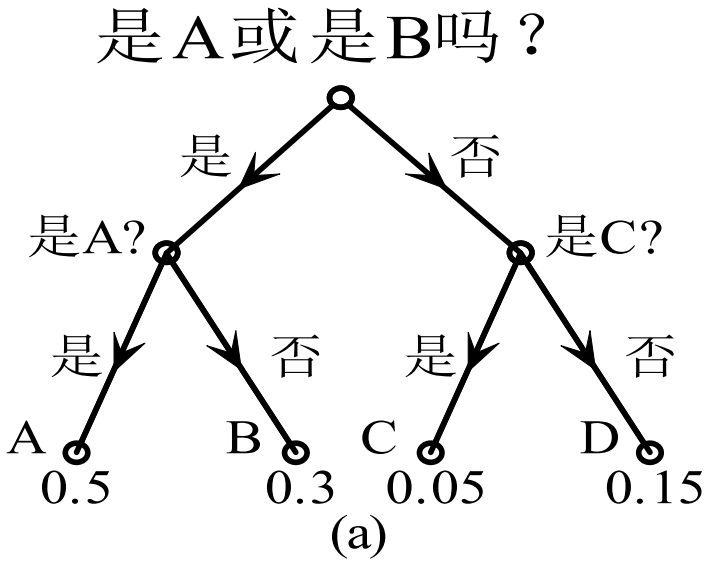
$\sum W(v)L(v)$ （其中$v\in V$,V是叶子的集合）

为$\langle D,W\rangle$的叶加权路径长度，其中$W(v)$是叶子v的权，$L(v)$为v的级。

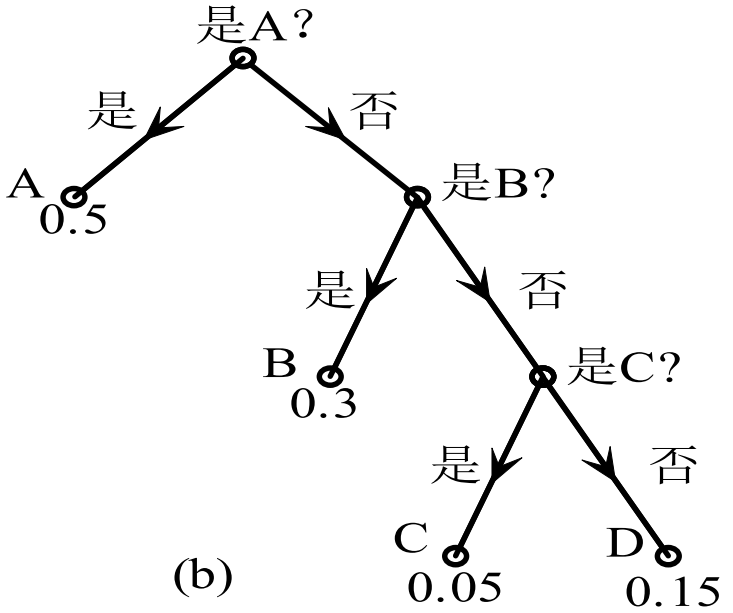
用叶子表示字母或符号，用分支结点表示判断，用权表示字母或符号出现的几率，则叶加权路径长度就表示算法的平均执行时间。

二叉树

例：图(a)和(b)表示了识别的两个算法，出现的概率分别是0.5，0.3，0.05，0.15。判断哪个算法更优？



(a)



(b)

叶加权路径长度
=2*(0.5+0.3+0.05+0.15)=2

叶加权路径长度
=1*0.5+2*0.3+3*(0.05+0.15)=1.7

最优二叉树

定义：设$\langle D,W\rangle$是叶加权二叉树。如果对于一切叶加权二叉树$\langle D',W'\rangle$只要对于任意正实数r，D和D'中权等于r的叶的数目相同，就有$\langle D,W\rangle$的叶加权路径长度不大于$\langle D',W'\rangle$的叶加权路径长度，则称$\langle D,W\rangle$为最优的。

把求某问题的最佳算法就归结为求最优二叉树的问题。

寻找最优二叉树

- 假定我们要找有m片叶，并且它们的权分别为$w_1,w_2,\cdots,w_m$的最优二叉树。不妨设$w_1,w_2,\cdots,w_m$是按递增顺序排列的，即$w_1 \le w_2 \le \cdots \le w_m$。
- 设$\langle D,W\rangle$是满足要求的最优二叉树，D中以$w_1,w_2,\ldots,w_m$为权的叶分别为$v_1,v_2,\cdots,v_m$。
- 显然，在所有的叶中，v_1和v_2的级最大。不妨设v_1和v_2与同一个分支结点v'邻接，令$D' = D - \{v_1, v_2\}$，$W':\{v', v_3, v_4, \cdots, v_m\} \to R_+$，并且$W'(v') = w_1 + w_2$，$W'(v_i) = w_i$。
- 容易证明，$\langle D,W\rangle$是最优的，当且仅当$\langle D',W'\rangle$是最优的。
- 这样把求m片叶的最优二叉树归结为求m-1片叶的最优二叉树。继续这个过程，直到归结为求两片叶的最优二叉树，问题就解决了。

最优二叉树

例：求叶的权分别为2，3，9，18，23，29的最优二叉树。

计算过程如下：

2 3 9 18 23 29

5 9 18 23 29

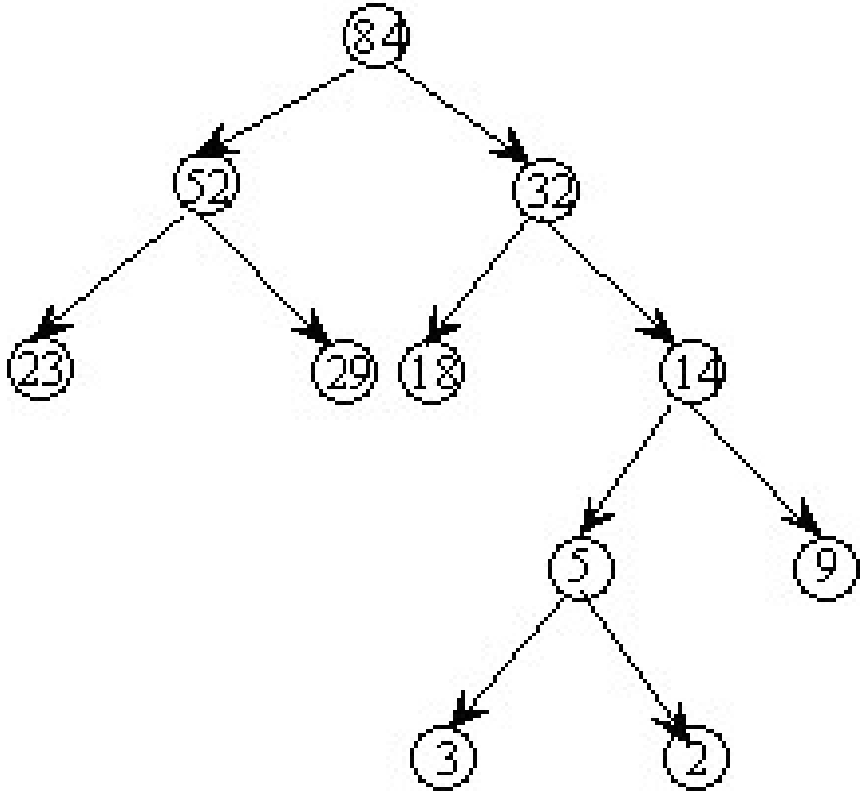
14 18 23 29

32 23 29

32 52

84

结果如下：



最优二叉树

例：求叶的权分别为2，3，9，18，23，29的最优二叉树。

计算过程如下：

2 3 9 18 23 29

5 9 18 23 29

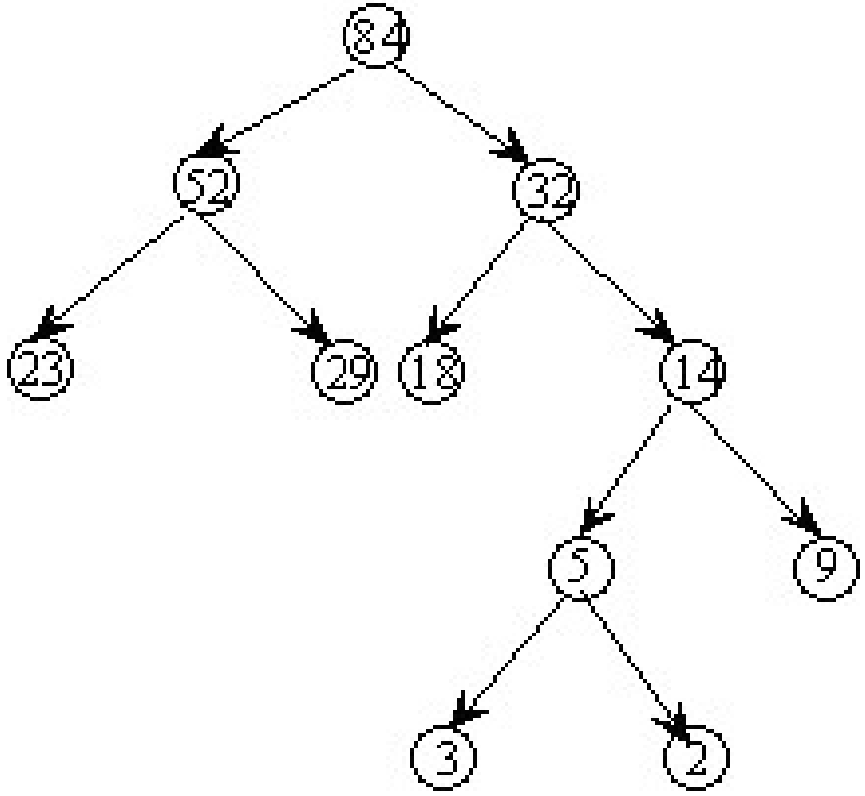
14 18 23 29

32 23 29

32 52

84

结果如下：



有向有序树

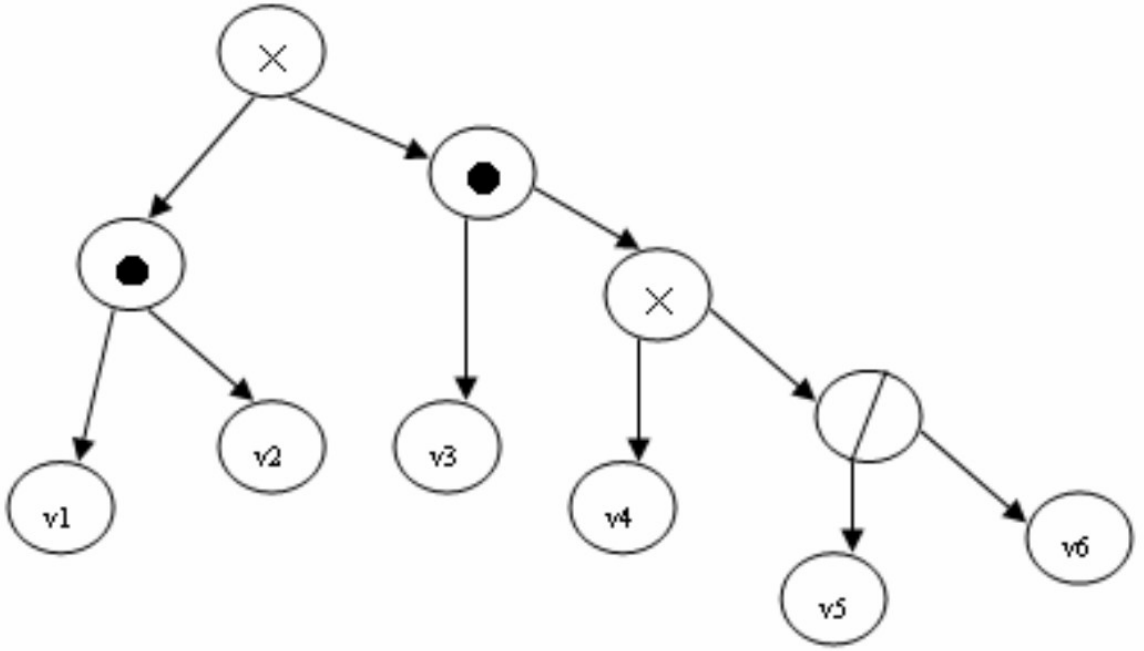
定义：为每一级上的结点规定了次序的有向树，称为有向有序树。

如果有向森林F的每个弱分支都是有序树，并且为F的每个弱分支也规定了次序，则称F为有向有序森林。

并且约定，在画有向有序树时，总是把根画在上部，并规定同一级上结点的次序是从左至右的。在画有序森林时，弱分支的次序也是从左至右的。

有向有序树

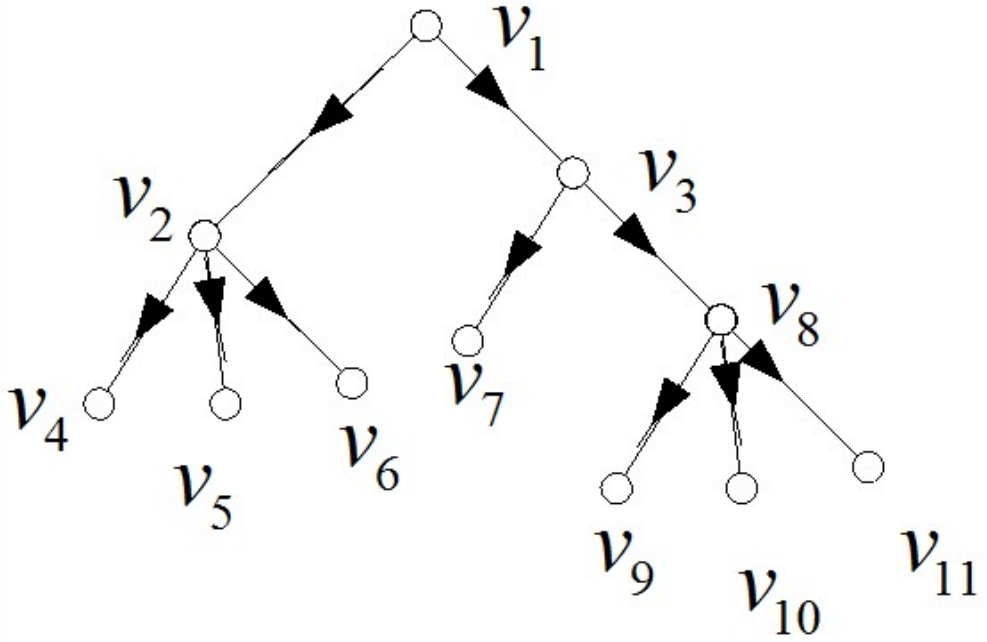
可以用有向有序树表达算术表达式，其中叶表示参加运算的数或变量，分支节点表示运算符。



$$v_1 * v_2 + v_3 * (v_4 + v_5 / v_6)$$

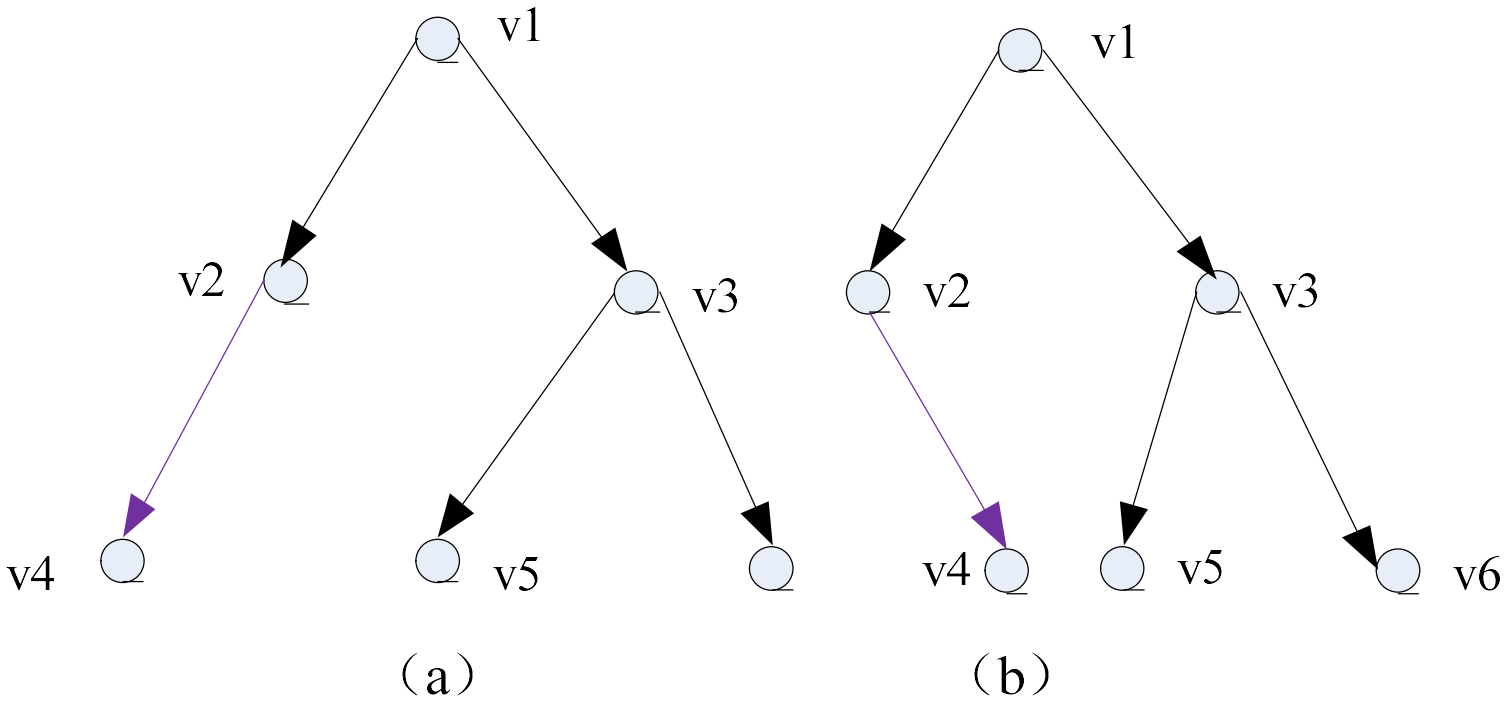
有向有序树

为方便起见，借用家族树的名称来称呼有向有序树的结点。如下图，称v_1是v_2和v_3的父亲，v_2是v_1的长子，v_2是v_3的哥哥，v_6是v_5的弟弟，v_2是v_7的伯父，v_5是v_8的堂兄。



位置有向有序树

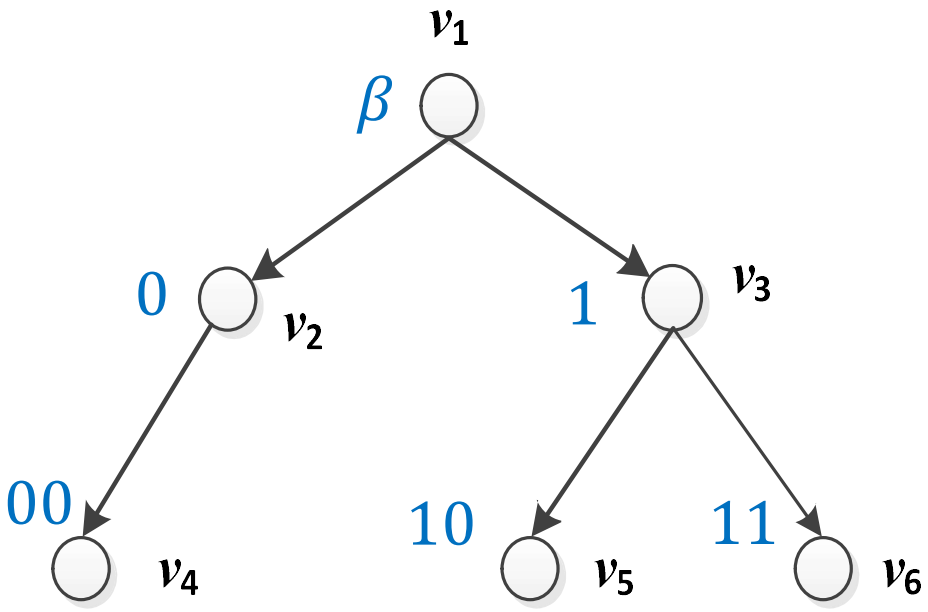
定义：为每个分支结点的儿子规定了位置的有向有序树，称为位置有向有序树。



位置有向有序树

表示方法：

- 在位置二元有向有序树中，可用字母表{0,1}上的字符串唯一地表示每个结点。
- 用空串ε表示根。设用β表示某分支结点，则用β_0表示它的左儿子，用β_1表示它的右儿子。

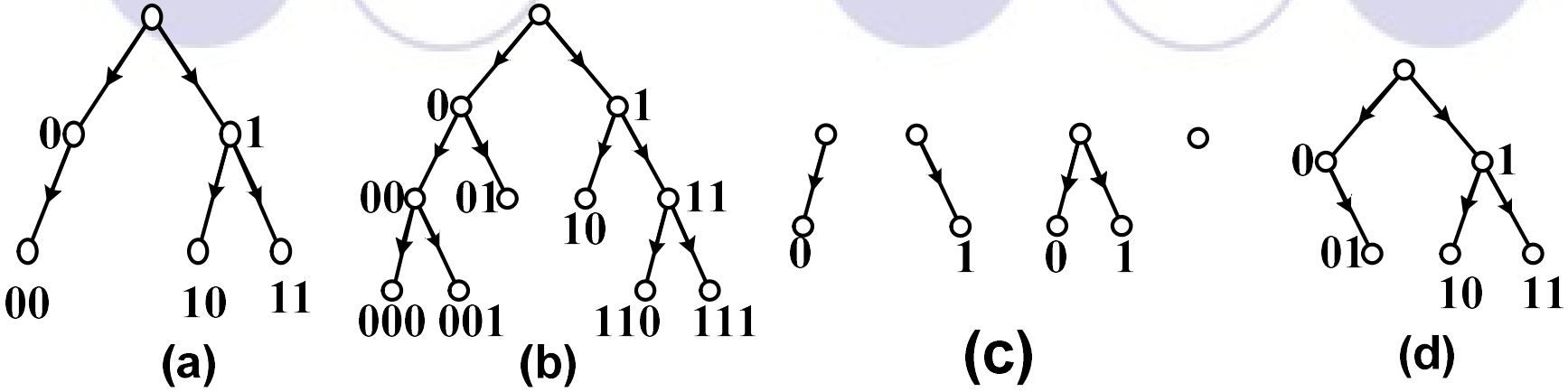


$v_1, v_2, v_3, v_4, v_5, v_6$ 的编码分别为β,0,1,00,10,11

位置二元有向有序树全体叶的编码表示的集合称为它的前缀编码。

左图的前缀编码为：{00,10,11}

位置有向有序树



(*a*)给出一棵二叉树；

(*b*)是一棵完全二叉树；

(*c*)是二叉树中的一个结点的各个儿子的所有可能的四种排列。

(*d*)与(*a*)都是二叉树，但却是不同的位置有向有序树。

位置有向有序森林的转化

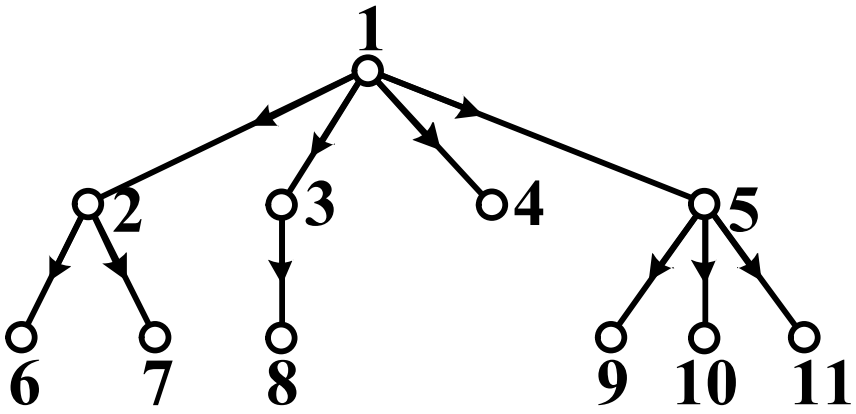
- 可以在有向有序森林和位置二元有向有序树之间建立一一对应关系。
- 在有向有序森林中，我们称位于左边的有向有序树的根为位于右边的有向有序树的根的哥哥。
- 设F为有向有序森林，T为对应的有向有序树。

方法：

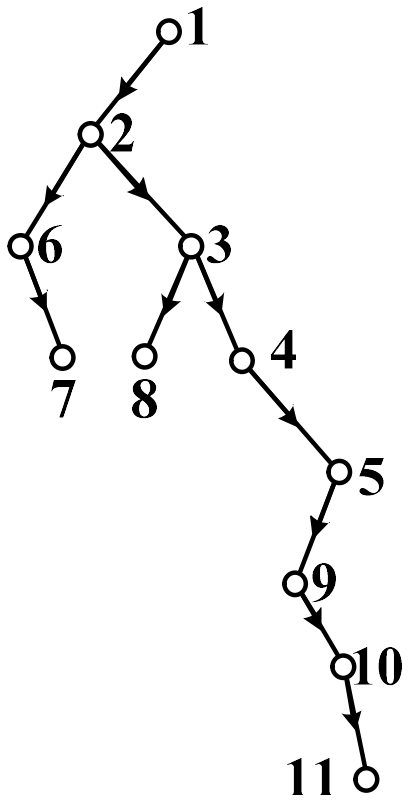
- 在F中，若v_1是v_2的长子，则在T中v_1是v_2的左儿子。
- 在F中，若v_1是v_2的大弟，则在T中，v_1是v_2的右儿子。
- 这种对应关系称为有向有序森林和位置二元有向有序树之间的自然对应关系。

位置有向有序森林

例：求下列图对应的二元有向有序树。



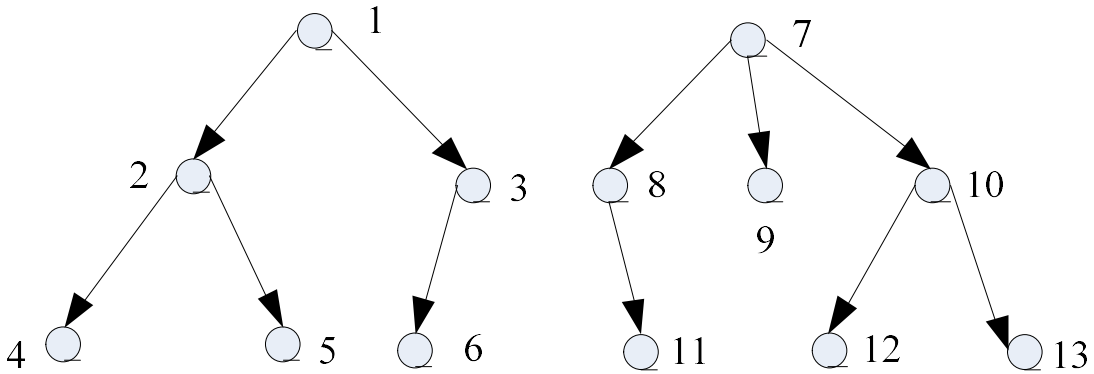
(a)



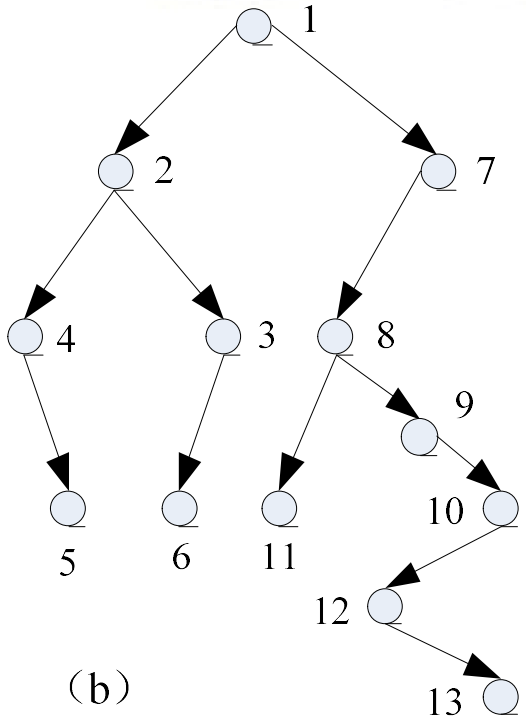
(c)

位置有向有序森林

例：求下列图对应的二元有向有序树。



（a）



（b）

二元树

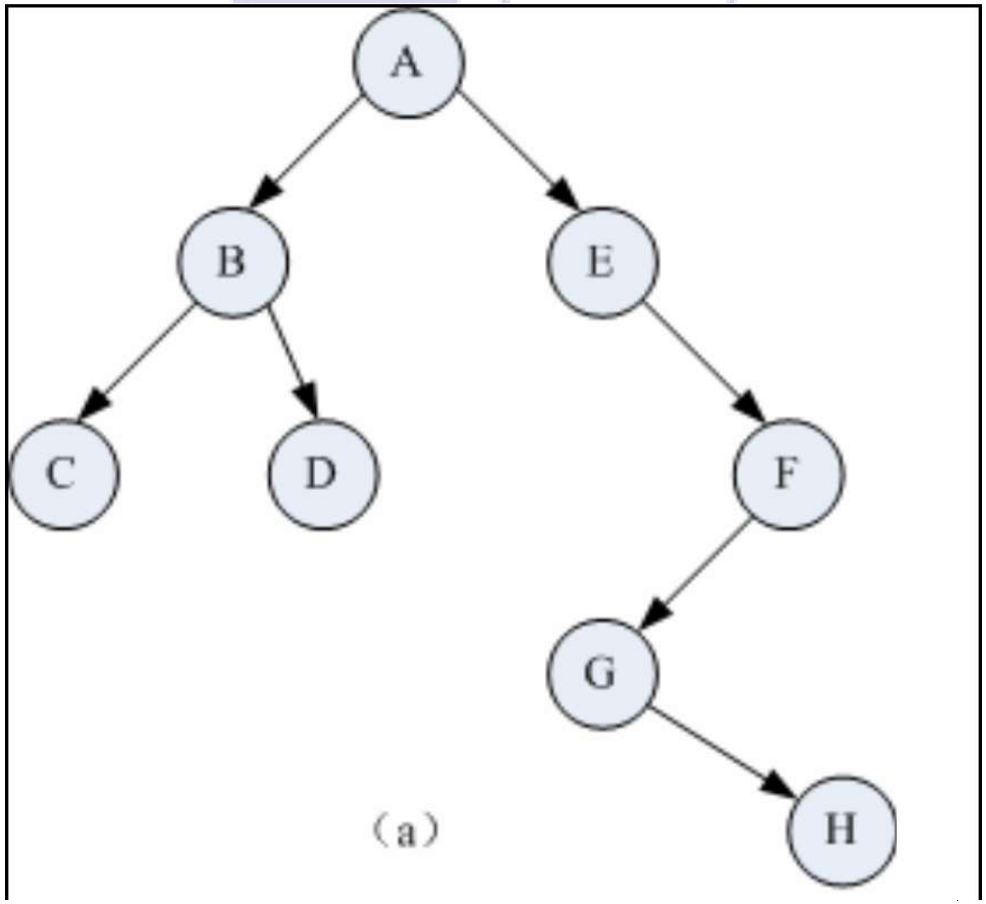
如果一个实际问题可以抽象成一个图，则可以将这个图转化成树（求这个图的生成树），对任何一棵树都可以转化成与其对应的二元树。

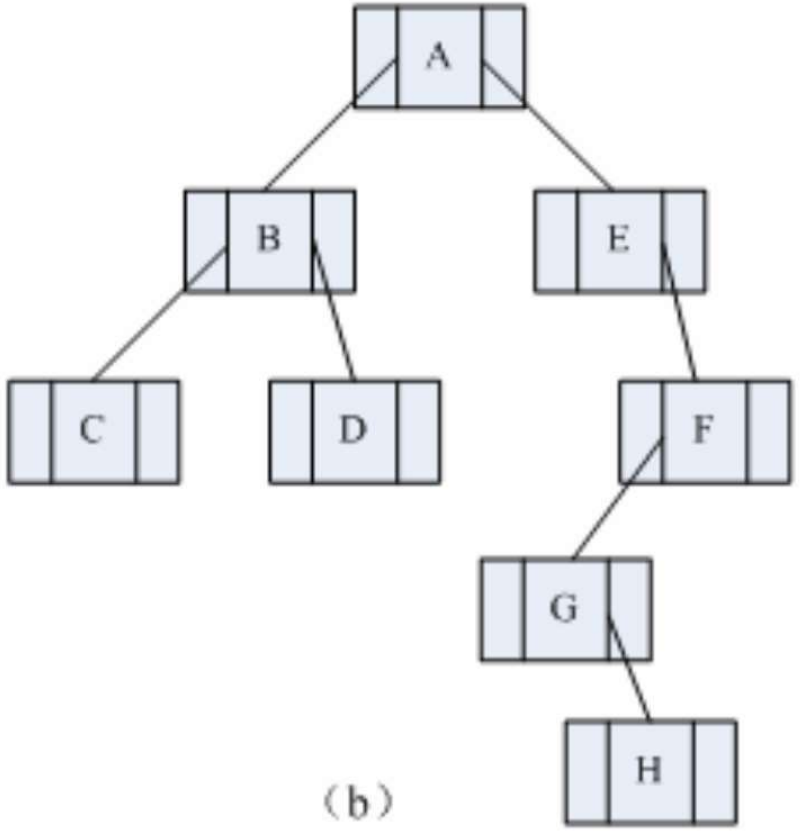
利用连接分配技术可以方便地表示二元树，其中所包含的结点结构为：

LLINK	DATA	RLINK

这里，LLINK或RLINK分别包含一个指向所论结点的左子树或右子树的指针（或称指示数、指示字）、DATA包含与这个结点有关的信息。

二元树

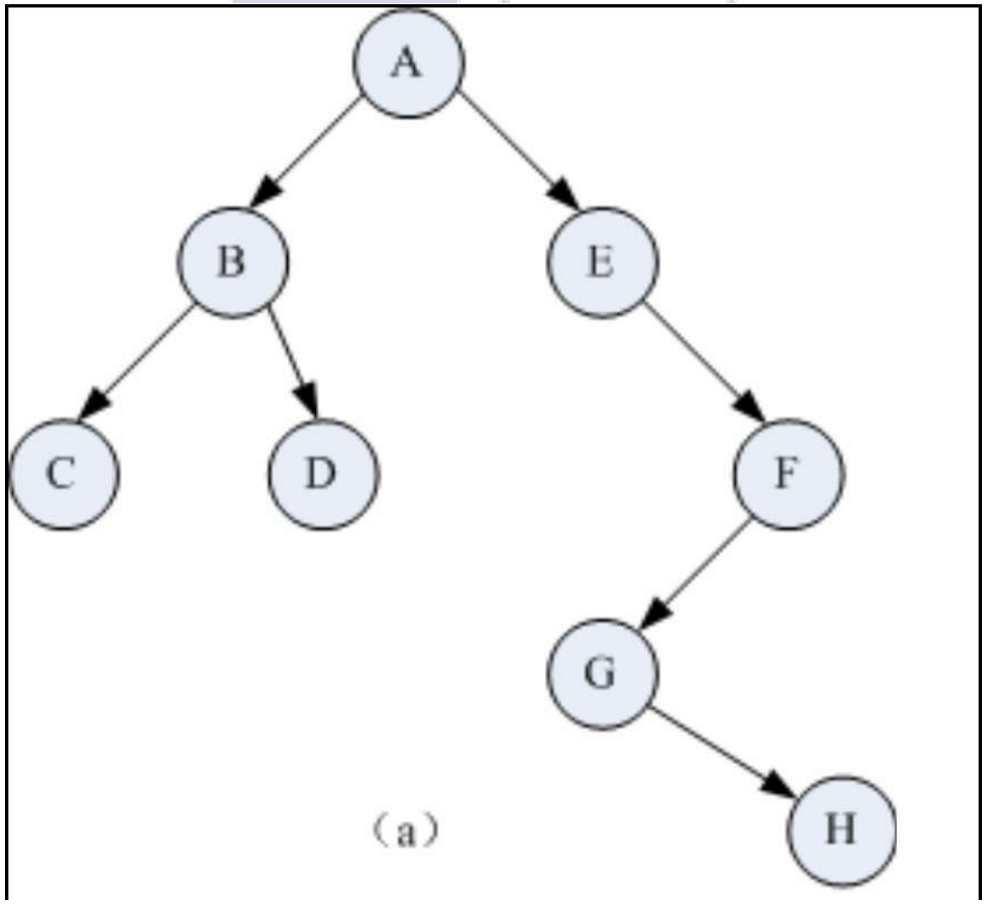




二元树的周游

- 前序周游
  - 处理根结点
  - 按前序周游左子树
  - 按前序周游右子树
- 中序周游
  - 按中序周游左子树
  - 处理根结点
  - 按中序周游右子树
- 后序周游
  - 按后序周游左子树
  - 按后序周游右子树
  - 处理根结点

二元树的周游



A B C D E F G H (前序)

C B D A E G H F (中序)

C D B H G F E A (后序)

前序周游算法

PREORDER算法：给定一棵二元树,它的根结点地址是变量T,它的结点结构和上面描述的相同,本算法按前序周游这棵二元树.利用一个辅助栈S,S的顶点元素的下标是TOP, P是一个临时变量,它表示我们处在这棵树中的位置。

1.[置初态] 如果T=NULL, 则退出(树无根,因此不是真的二元树)；否则$P \leftarrow T; TOP \leftarrow 0$。

2.[访问结点,右分枝地址进栈,并转左] 处理结点P。如果PRLINK(P)≠NULL,则置 $TOP \leftarrow TOP+1$; $S[TOP] \leftarrow RLINK(P); P \leftarrow LLINK(P)$ 。

3.[链结束否?] 如果P≠NULL,则转向第2步。

4.[右分枝地址退栈] 如果TOP=0则退出；否则，令$P \leftarrow S[TOP]; TOP \leftarrow TOP-1$,并转向第2步。

前序周游算法

例：分析对前面二元树的前序周游算法执行过程。

栈的内容	P	访问 P	输出串
	NA	A	A
NE	NB	B	AB
NE ND	NC	C	ABC
NE ND	NULL		
NE	ND	D	ABCD
NE	NULL		
	NE	E	ABCDE
NF	NULL		
	NF	F	ABCDEF
	NG	G	ABCDEFG
NH	NULL		
	NH	H	ABCDEFGH
	NULL		

后序周游算法

POSTORDER算法：假定结点的结构和上面所说的一样, *T*还是一个等于树根地址的变量。*S*也是一个所需要的具有顶元指针的栈,但在本章算法中,每一个结点将进栈两次：一次是当周游它的左子树时；另一次是当周游它的右子树时。这两种周游完成时,就处理这个被考虑的结点。因此,我们必须能区别两种类型的栈元素。第一种类型的元素将采用负指针值。当然,这里假设实际使用的指针数据总是非零并且是正的。

后序周游算法

1.[置初态] 如果T=NULL,则退出算法(树无根,因此不是真的二元数);否则令$P \leftarrow T; TOP \leftarrow 0$ 。

2.[结点进栈并转左] 令$TOP \leftarrow TOP+1; S[TOP] \leftarrow P;$ $P \leftarrow LLINK(P)$ 。

3.[链结束否?] 如果P=NULL,则转2。

4.[结点地址出栈] 如果TOP=0,则退出算法； 否则, 令$P \leftarrow S[TOP]; TOP \leftarrow TOP-1$。

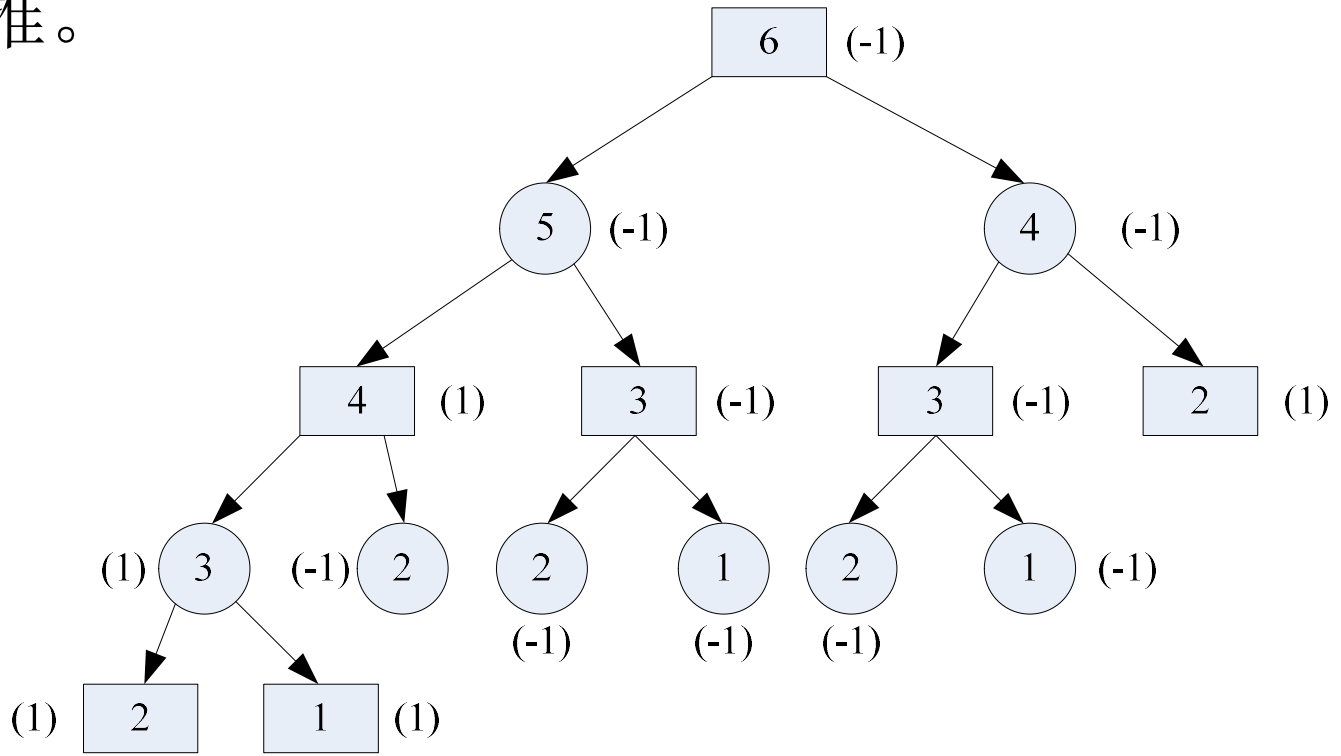
5.[如果右子树还没有周游,则地址重新进栈] 若P<0, 则转6;否则令$TOP \leftarrow TOP+1; S[TOP] \leftarrow P; P \leftarrow RLINK(P)$ 并转3。

6.[访问结点] 令$P \leftarrow -P$,处理结点P,转4。

搜索树

例：设有n根火柴,甲乙两人依次从中取走1或2根,但不能不取,谁取走最后一根谁就是胜利者。

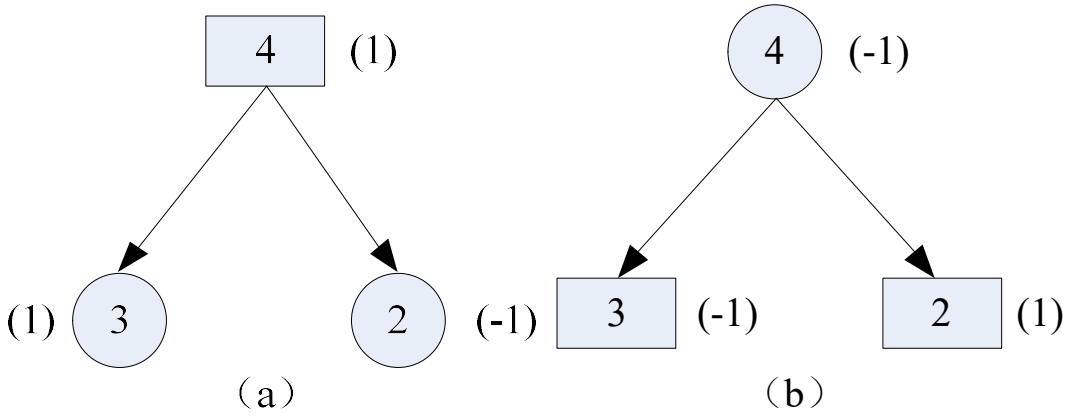
为了说明方法,不妨设n=6。在下图中，[6]表示轮到甲取时有6根火柴，④表示轮到乙取时有4根火柴,余此类推。



搜索树

显然，一当出现1或2状态，甲取胜，不必再搜索下去。同样，①或②是乙取胜的状态。

若甲取胜时，设其得分为1，乙取胜时甲的得分为-1。无疑，轮到甲作出判决时，他一定选（-1，1）中的最大者；而轮到乙作出判决时，他将选取使甲失败，选+1、-1中最小者。



甲遇到（a）的状态时，甲应选1，即甲应取1根火柴使状态进入③。同理，乙遇到（b）的状态时，乙应选取-1，使甲进入必然失败的状态为好。

DFS(深度优先Deep-First-Search)算法(图的遍历算法)

（1）当$E(G)$的所有边未经完全搜索时，任取一结点 $v_i \in V(G)$，给v_i以标志且入栈。

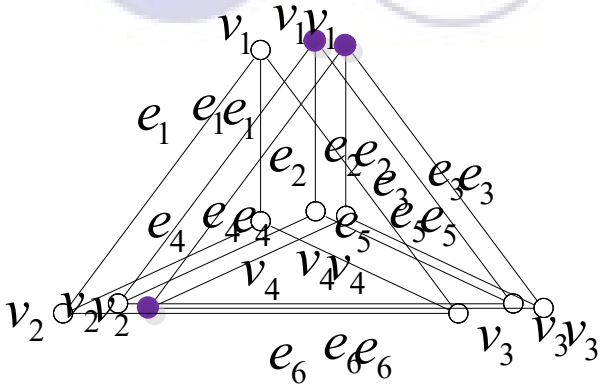
（2）对与v_i点关联的边依次进行搜索，当存在另一端点未给标志的边时，把另一端点作为v_i，给以标志，并且入栈；转（2）。

（3）当与v_i关联的边全部搜索完毕时（即不存在以v_i为端点而未经搜索的边时），则以v_i点从栈顶退出，即让取走v_i后的栈顶元素作为v_i，转（2）。

（4）若栈已空，但还存在未给标志的节点时，取其中任一结点作v_i，转（2）。若所有节点都已给标志时，则算法终止。

DFS算法

例：



图的邻接矩阵为：

$$A=\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array}\begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

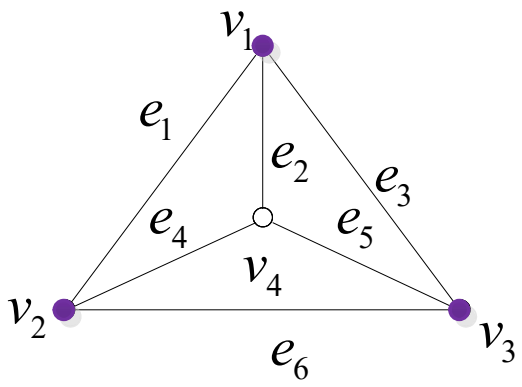
设从v_1开始，给以标志，与相邻的节点依次为$\{v_2,v_3,v_4\}$，即 $A_{di}(v_1)=\{v_2,v_3,v_4\}$

第一个邻接点v_2未给标志，故v_2入栈且给标志。
但$A_{di}(v_2)=\{v_1,v_3,v_4\}$，而第一个邻接结点$v_1$已给标志，故取$\{v_2,v_3\}$边，给$v_3$以标志，且入栈。

DFS算法

又$A_{di}(v_3)=\{v_1,v_2,v_4\}$，由于$v_1$，$v_2$都已给标志，故取边$\{v_3,v_4\}$，给$v_4$以标志并入栈，但与$v_4$相邻的结点全部都给了标志，故退栈。此时栈顶点为$v_3$，但与$v_3$相邻的结点均已给标志，故退栈。$v_2$，$v_1$因类似理由依次退栈。栈空，故结束。

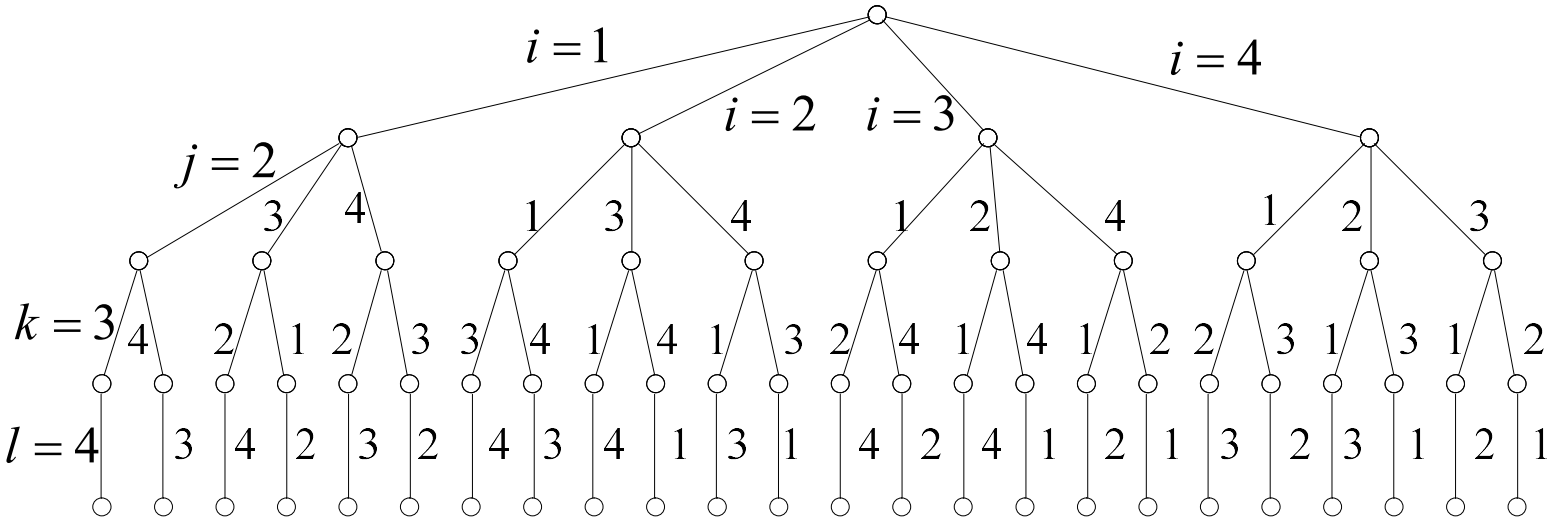
图中结点的访问顺序为$v_1v_2v_3v_4$。



搜索树

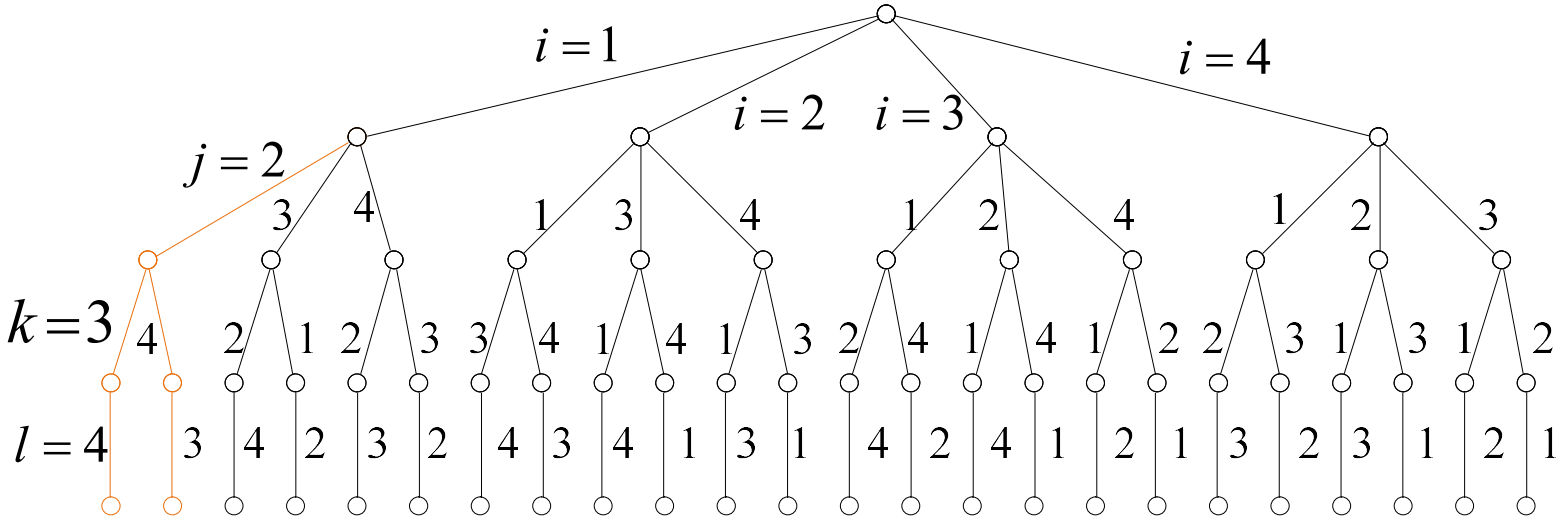
例：设有一个4×4的棋盘，当一个棋子放到其中一个格子里去以后，则这格子所在的行和列以及对角线上所有的格子都不允许放别的棋子。现在有四个棋子，试问它在这个棋盘上有哪几种容许的布局？

解：第一行的格子有四个，故第一行有四种选择，第二行则有三种选择；第三行则有两种选择；最后一行无选择的余地。它的状态可用下面的树表示：

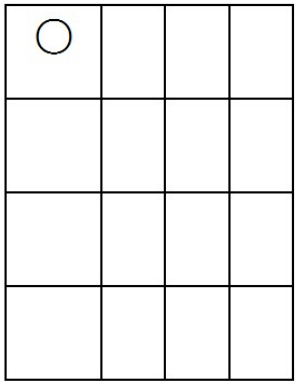


搜索树

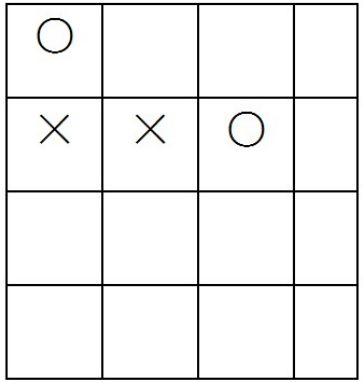
- 要确定哪几种状态是被允许的，就要对这棵树进行搜索。
- 一旦某结点被判定为不被容许，这个结点下的树枝可以全部剪去。
  - 比如i=1时，j=2不被容许，则i=1，j=2，k=3（或4）便无需搜索。



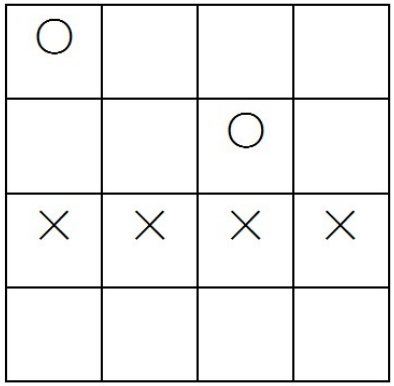
搜索树



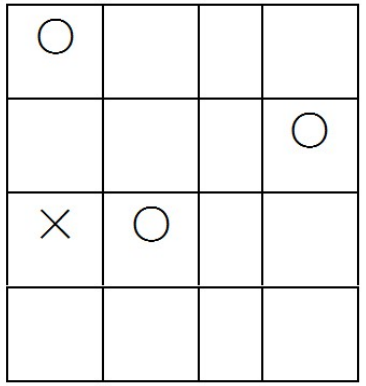
(1)



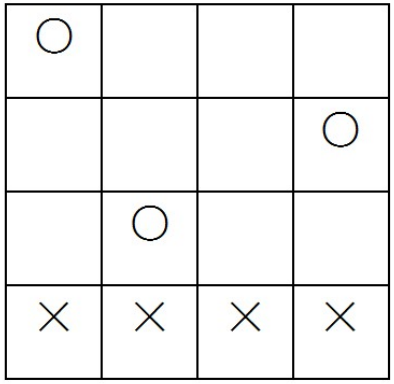
(2)



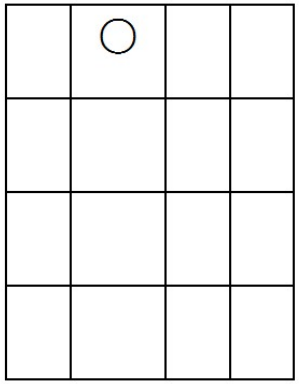
(3)



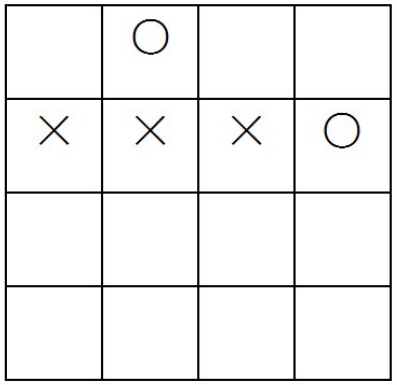
(4)



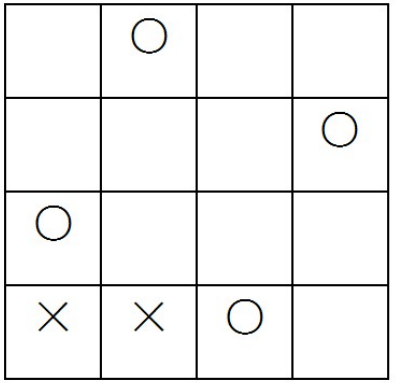
(5)



(6)



(7)



(8)

搜索过程

搜索树

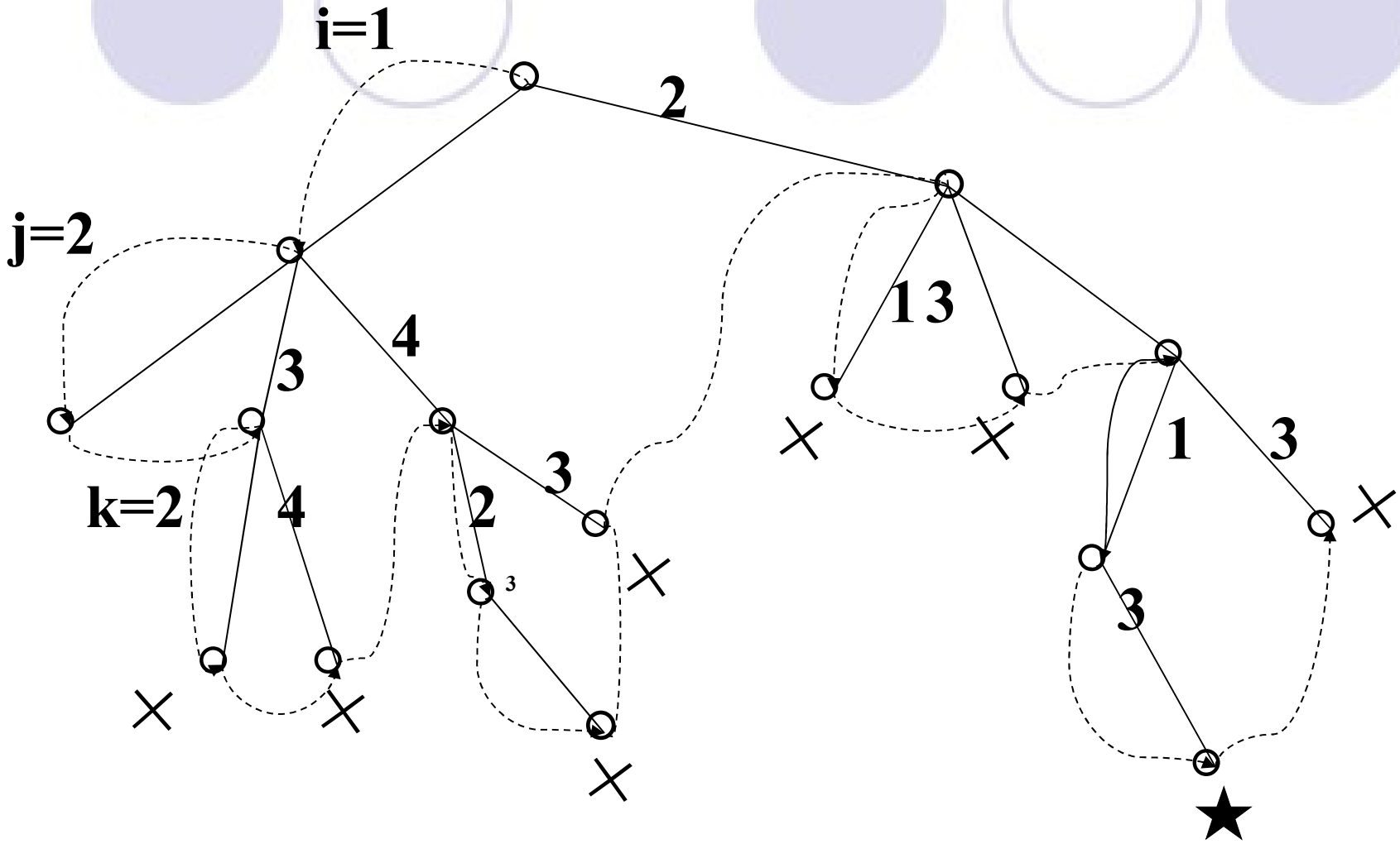


图7.7-20

Dijkstra算法

- Dijkkstra算法是典型最短路算法，用于计算一个节点到其他所有节点的最短路径。
- 主要特点是以起始点为中心向外层层扩展，直到扩展到终点为止。
- Dijkstra算法能得出最短路径的最优解，但由于它遍历计算的节点很多，所以效率低

Dijkstra's Algorithm

- **Dijkstra's algorithm** is an iterative procedure that finds the shortest path between to vertices a and z in a weighted graph.
- It proceeds by finding the length of the shortest path from a to successive vertices and adding these vertices to a distinguished set of vertices S.
- The **algorithm terminates** once it reaches the vertex z.

Dijkstra's Algorithm

- **procedure** Dijkstra(G: weighted connected simple graph with vertices $a = v_0, v_1, \ldots, v_n = z$ and positive weights $w(v_i, v_j)$, where $w(v_i, v_j) = \infty$ if $\{v_i, v_j\}$ is not an edge in G)

  for i := 1 **to** n

    $L(v_i) := \infty$

    L(a) := 0

    S := $\varnothing$

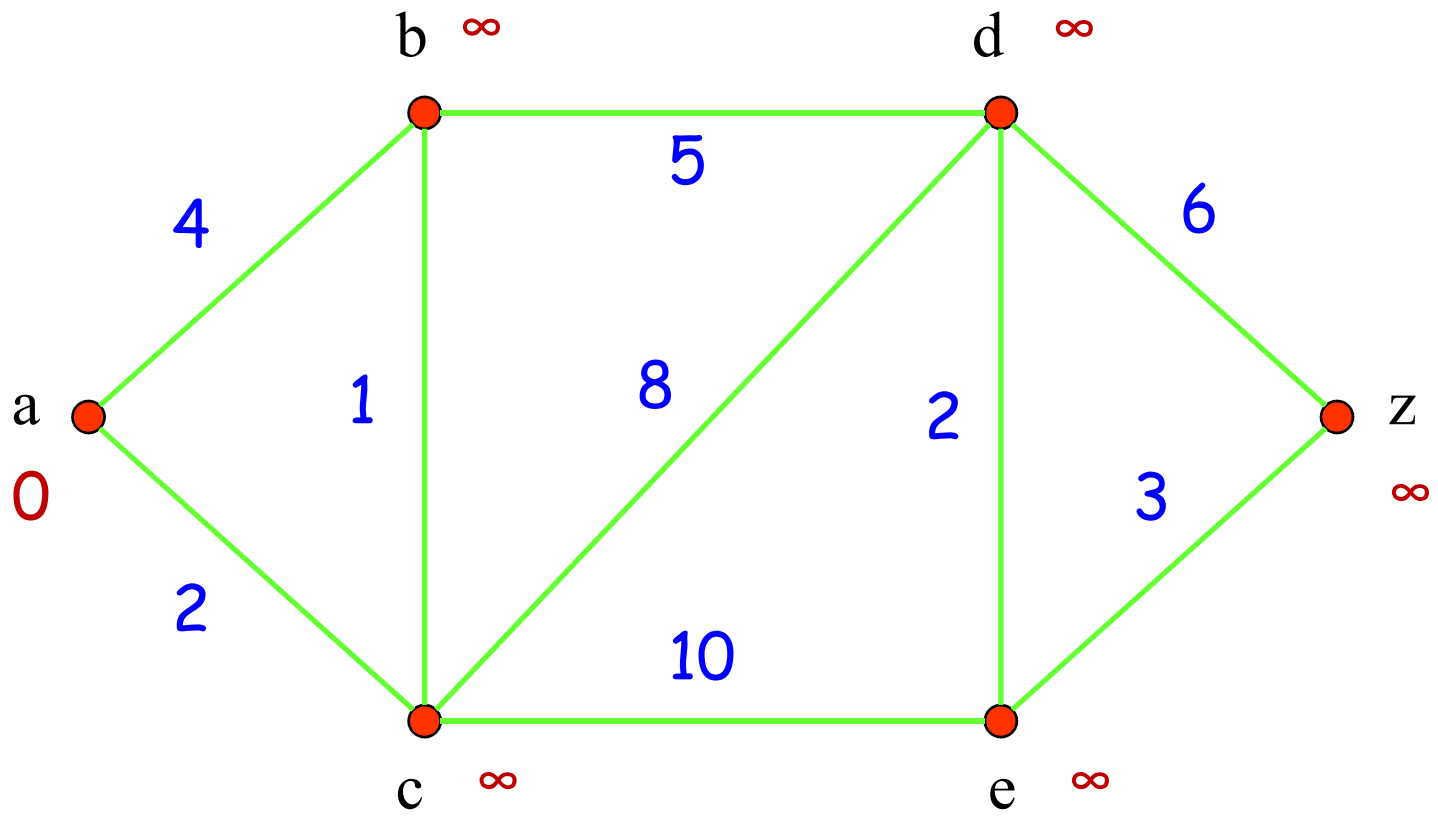
{the labels are now initialized so that the label of a is zero and all other labels are ∞, and the distinguished set of vertices S is empty}

Dijkstra's Algorithm

```
while z∉S
begin
   u := the vertex not in S with minimal L(u)
   S := S∪{u}
   for all vertices v not in S
        if L(u) + w(u, v) < L(v) then L(v) := L(u) +
   w(u, v)
   {this adds a vertex to S with minimal label
   and updates the labels of vertices not in S}
end {L(z) = length of shortest path from a to
   z}
```

Dijkstra's Algorithm

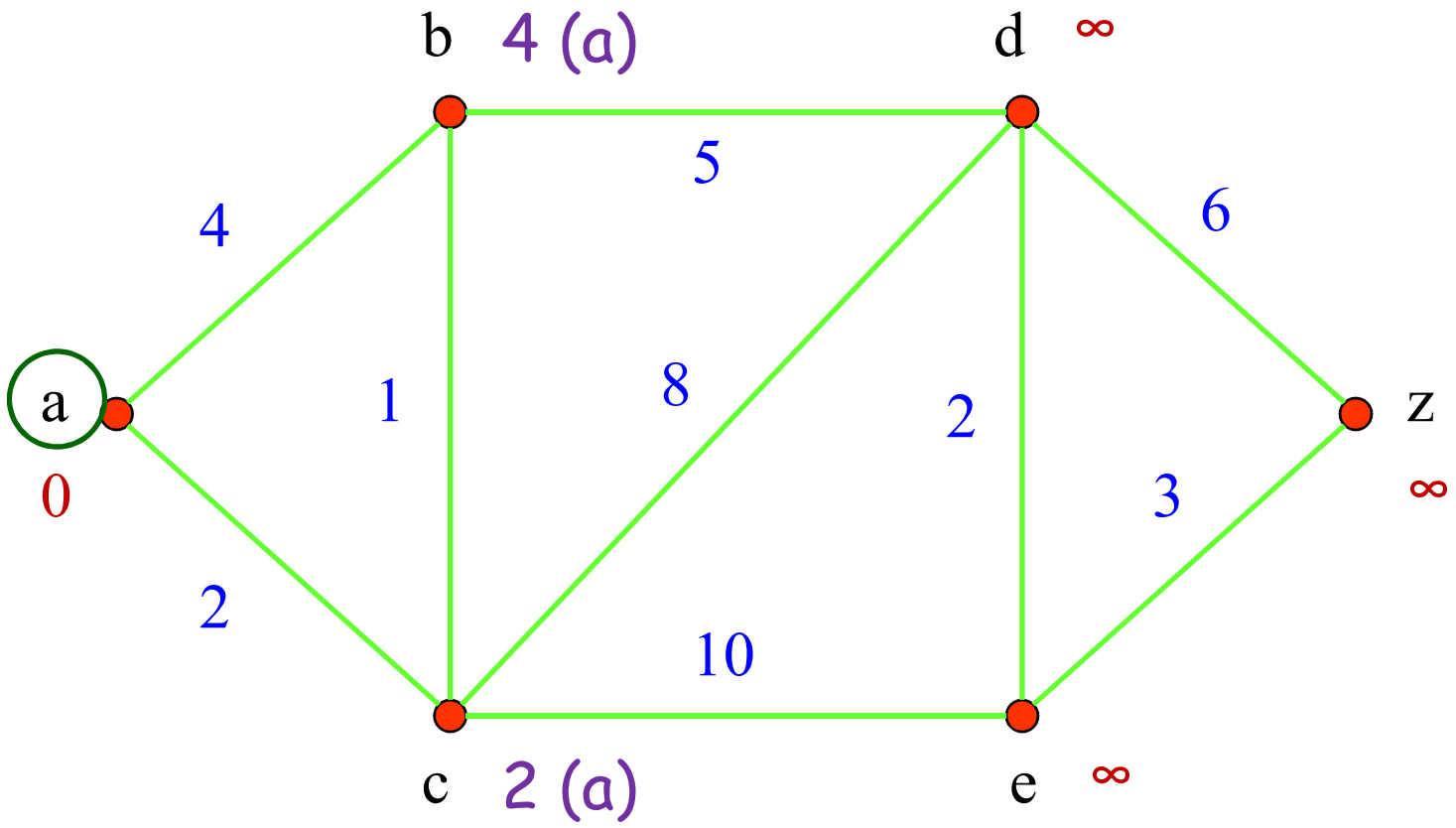
- **Example:**



Step 0

Dijkstra's Algorithm

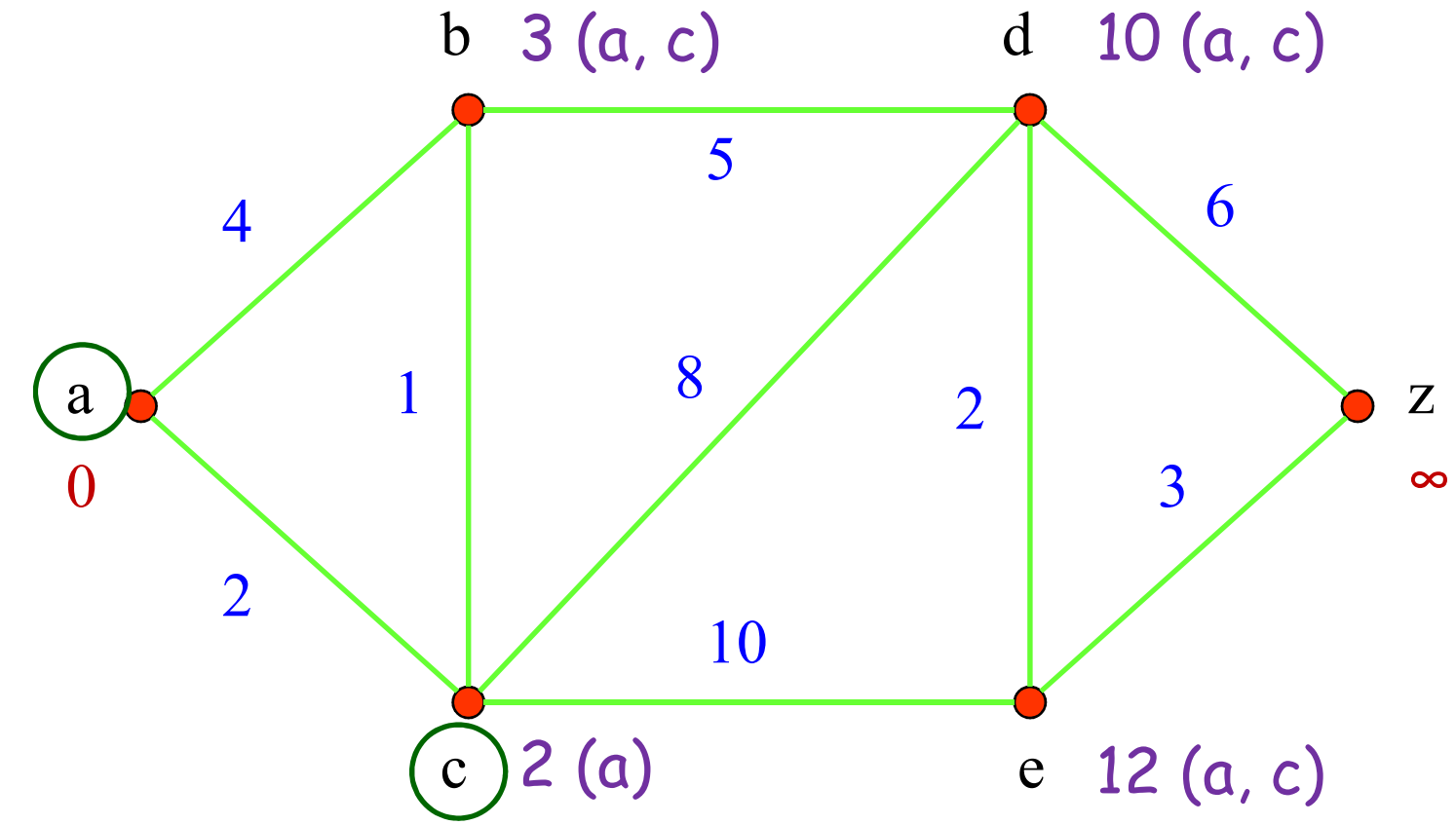
- **Example:**



Step 1

Dijkstra's Algorithm

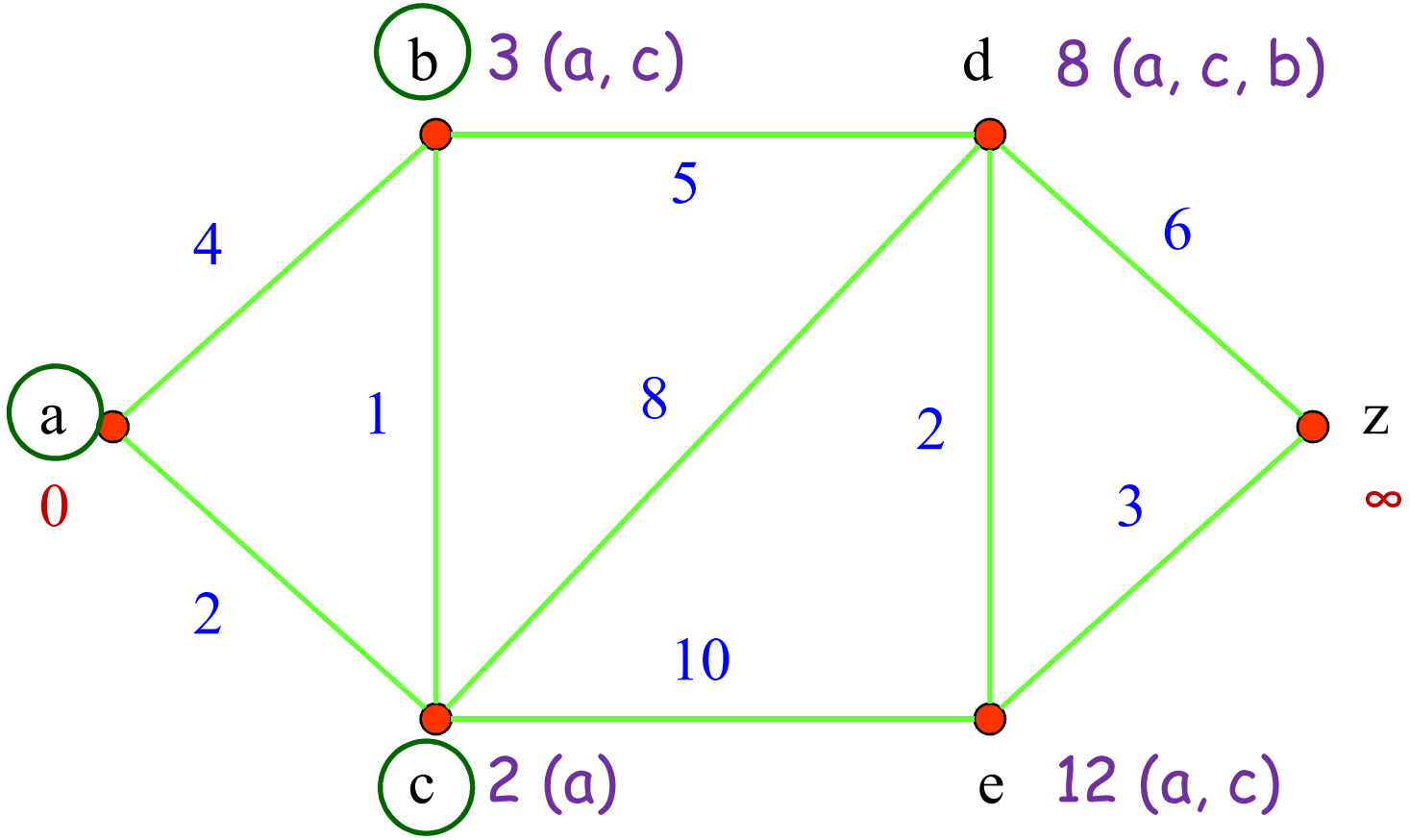
- **Example:**



Step 2

Dijkstra's Algorithm

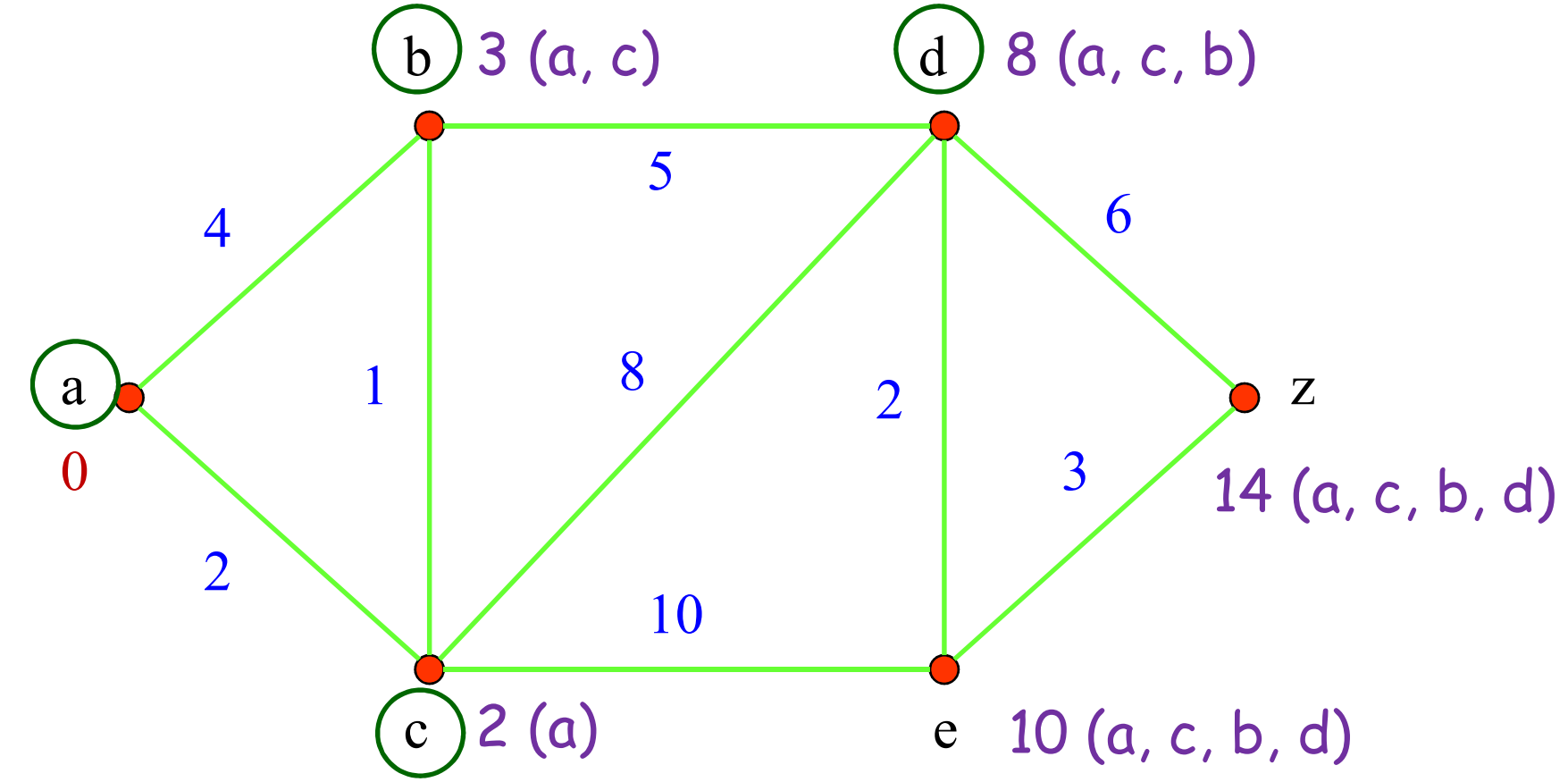
- **Example:**



Step 3

Dijkstra's Algorithm

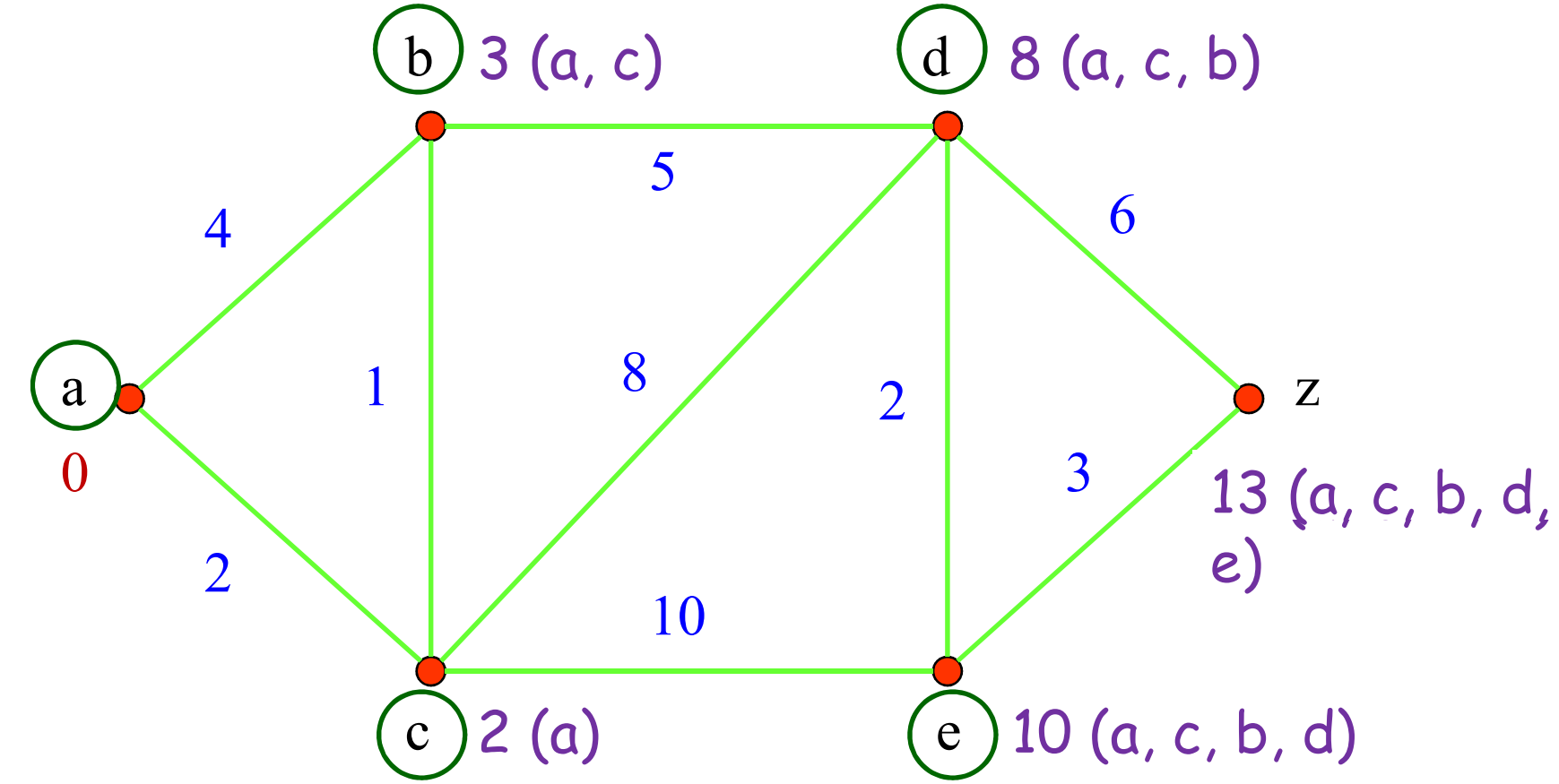
- **Example:**



Step 4

Dijkstra's Algorithm

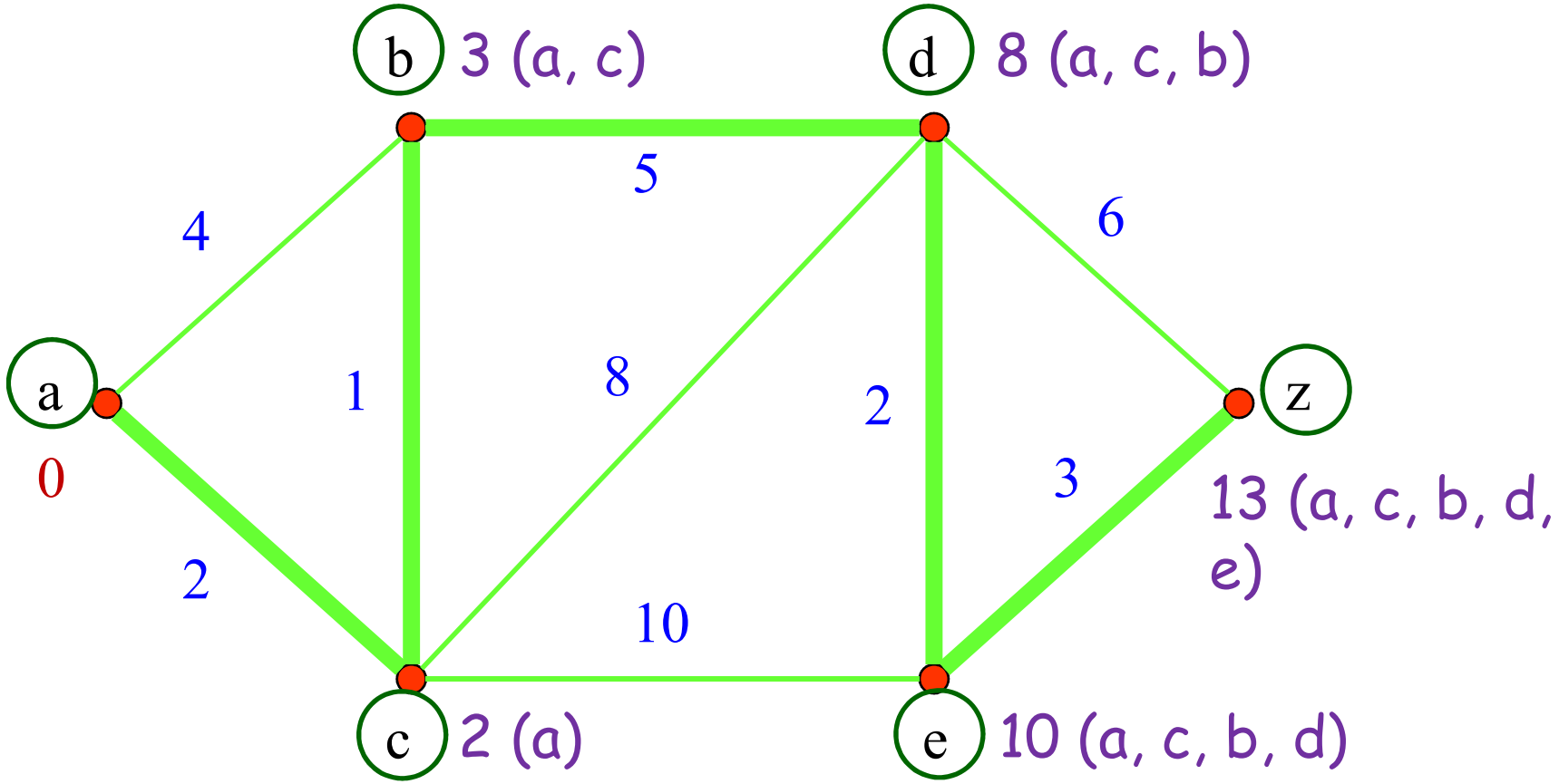
- **Example:**



Step 5

Dijkstra's Algorithm

- **Example:**



Step 6

Dijkstra's Algorithm

- **Theorem:** Dijkstra's algorithm finds the length of a shortest path between two vertices in a connected simple undirected weighted graph.
- **Theorem:** Dijkstra's algorithm uses $O(n^2)$ operations (additions and comparisons) to find the length of the shortest path between two vertices in a connected simple undirected weighted graph.
- Please take a look at Data Structure textbook for a comprehensive description and analysis of Dijkstra's algorithm.

作业

- 2,6,10