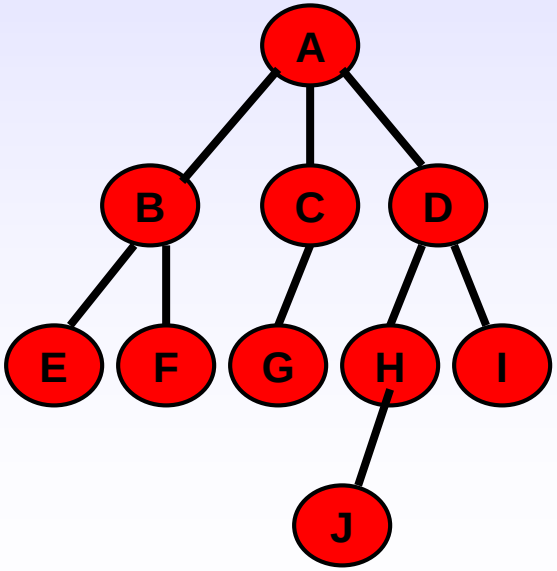
6.4 树和森林

6.4.1 树的存储结构

1 孩子表示法：由数据域、 K 个子结点指针域、父结点指针域组成。

data	child1	child2	.....	childk

例1: 度数 K ＝ 3 的树



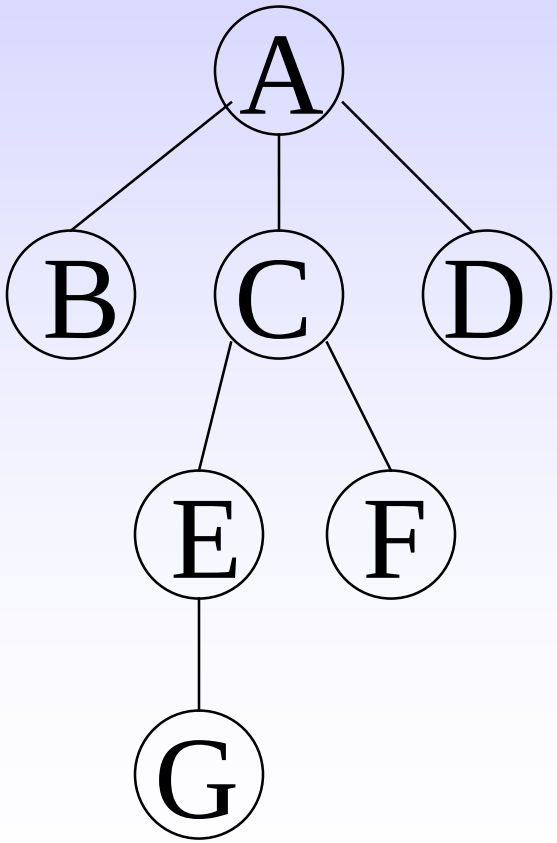
用数组（静态链式存储）表示左图的树，0 表示对应的子树为空。

缺点：空指针域太多，有(K -1)× n + 1 个。

1	A	2	3	4
2	B	5	6	0
3	C	7	0	0
4	D	8	9	0
5	E	0	0	0
6	F	0	0	0
7	G	0	0	0
8	H	10	0	0
9	I	0	0	0
10	J	0	0	0

6.4 树和森林

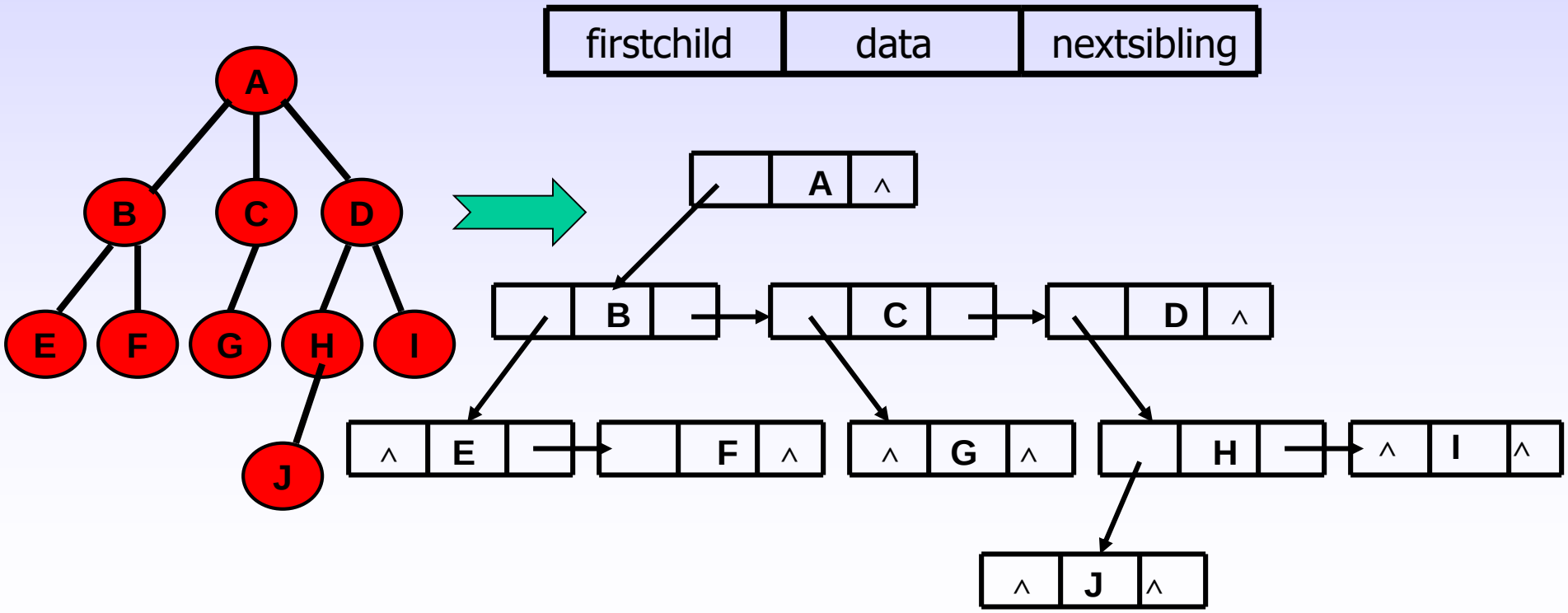
2 单链表表示法：把每个结点的子结点排列起来，看成一个线性表，且以单链表作存储结构。则n个结点有n个子链表。



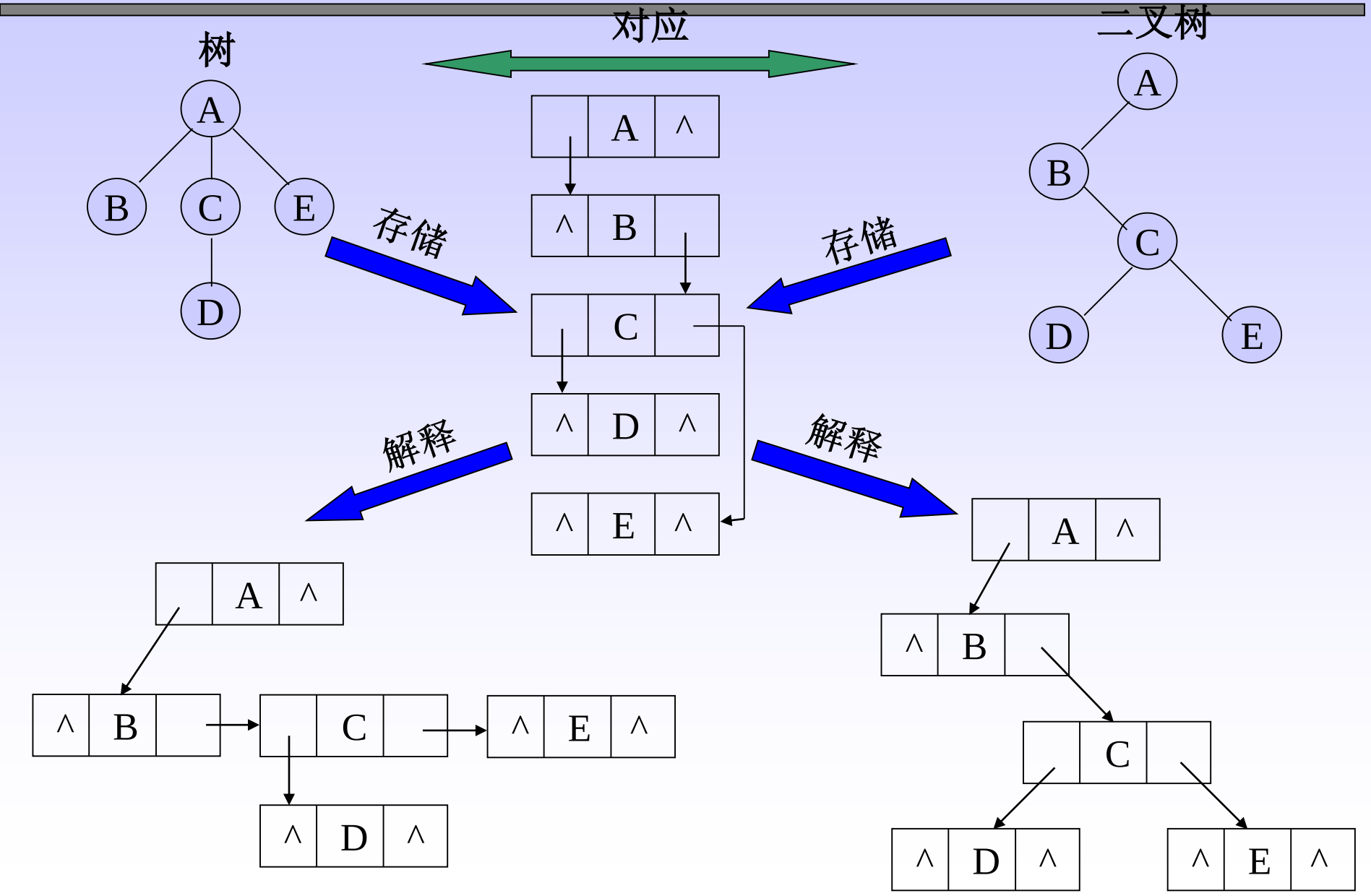
1	A	0	→ 2 → 3 → 4 ∧
2	B	1	∧
3	C	1	→ 5 → 6 ∧
4	D	1	∧
5	E	3	→ 7 ∧
6	F	3	∧
7	G	5	∧

6.4 树和森林

3 孩子兄弟表示法：每个结点由数据域、指向它的第一棵子树树根的指针域、指向它的兄弟结点的指针域组成。实质上是用二叉树表示一棵树。



6.4 树和森林



6.4 树和森林

6.4.2 树、森林与二叉树的转换

森林转化成二叉树的规则

① 若F为空，即$n = 0$，则对应的二叉树B为空二叉树。

② 若F不空，则对应二叉树B的根$root(B)$是F中第一棵树T_1的根$root(T_1)$；

其左子树为$B(T_{11}, T_{12}, \ldots, T_{1m})$，其中，$T_{11}, T_{12}, \ldots, T_{1m}$是$root(T_1)$的子树；

其右子树为$B(T_2, T_3, \ldots, T_n)$，其中，$T_2, T_3, \ldots, T_n$是除T_1外其它树构成的森林。

6.4 树和森林

二叉树转换为森林的规则

① 如果*B*为空，则对应的森林*F*也为空。

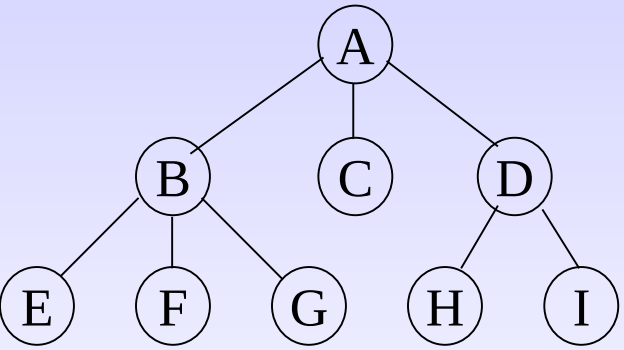
② 如果*B*非空，则*F*中第一棵树T_1的根为*B*的根*root(B)*；根T_1的子树森林{ T_{11}, T_{12}, …, T_{1m} }是由*B* 的左子树 *LB* 转换而来，*F* 中除了 T_1 之外其余的树组成的森林{ T_2, T_3, …, T_n } 是由 *B* 的右子树 *RB* 转换而成的森林。

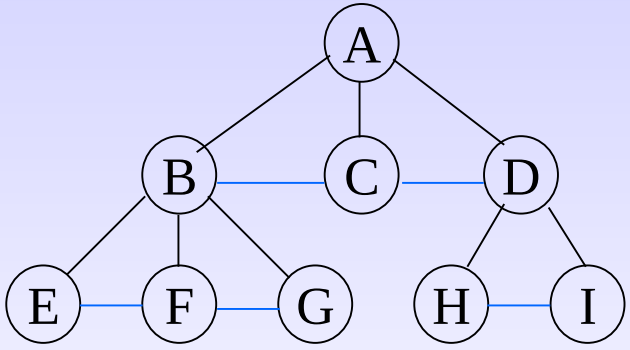
1、将树转换成二叉树

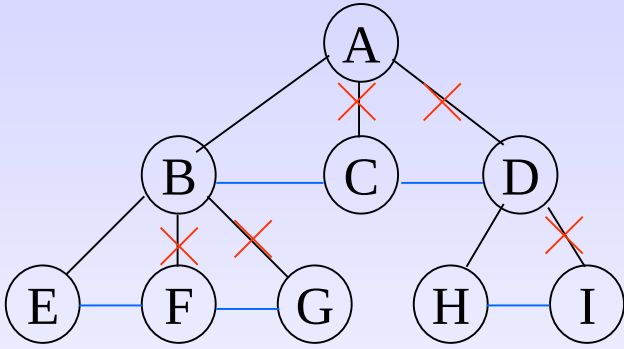
（1）在兄弟结点之间加一连线；

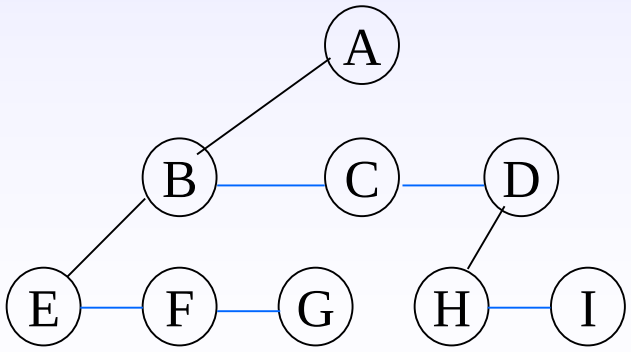
（2）对每个结点，除了其第一个孩子外，去除其与其余孩子之间的关系；

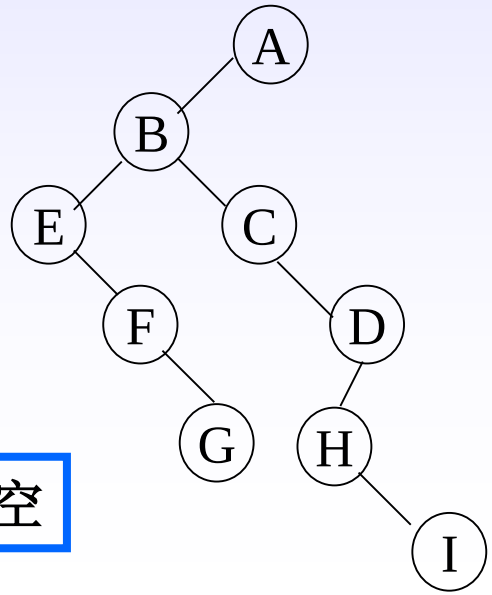
（3）以树的根结点为轴心，将整树顺时针转45°。











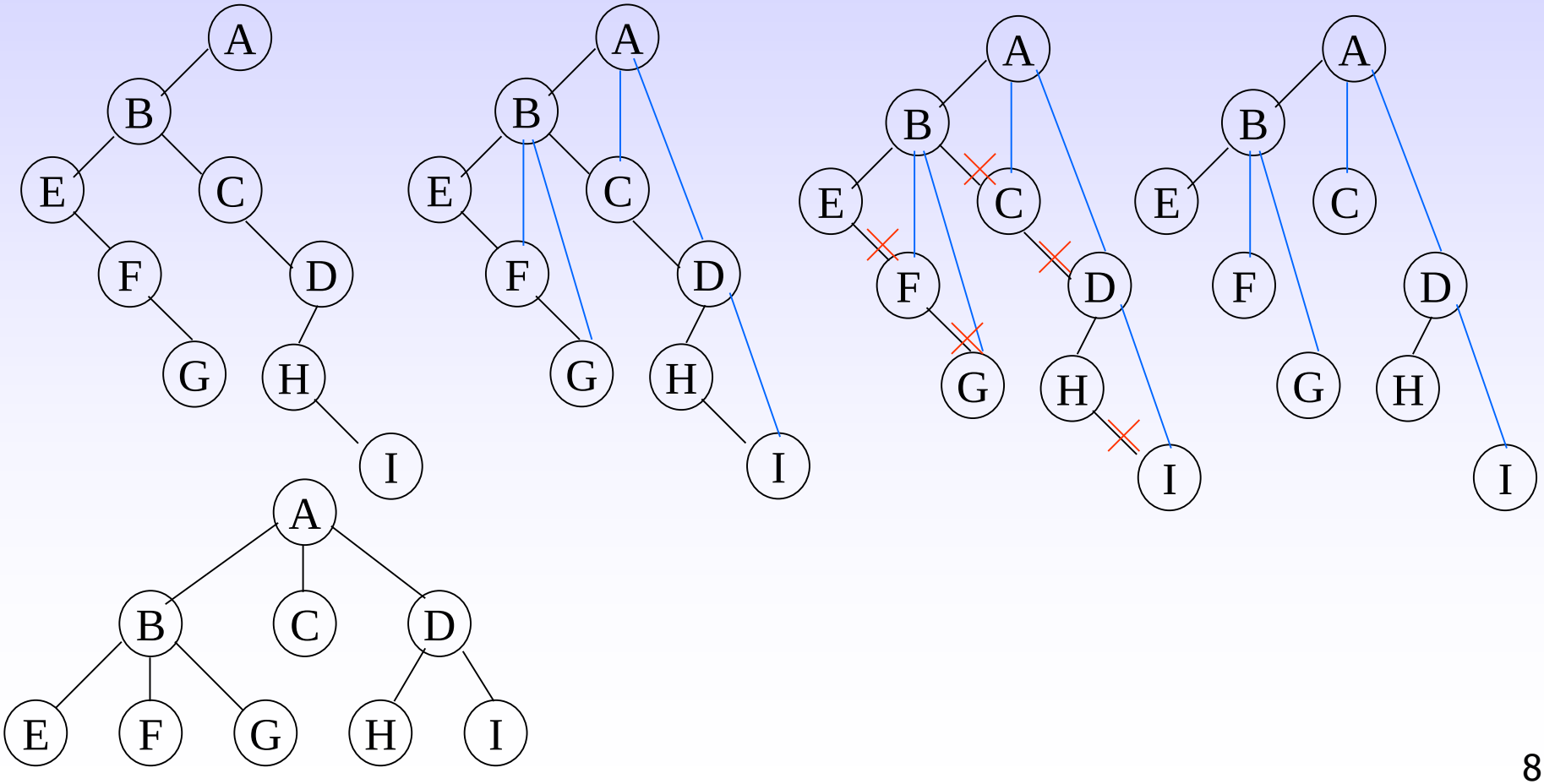
树转换成的二叉树其右子树一定为空

2、将二叉树转换成树

（1）若p结点是双亲结点的左孩子，则将p的右孩子，右孩子的右孩子，……沿分支找到的所有右孩子，都与p的双亲用线连起来；

（2）抹掉原二叉树中双亲与右孩子之间的连线；

（3）将结点按层次排列，形成树结构。

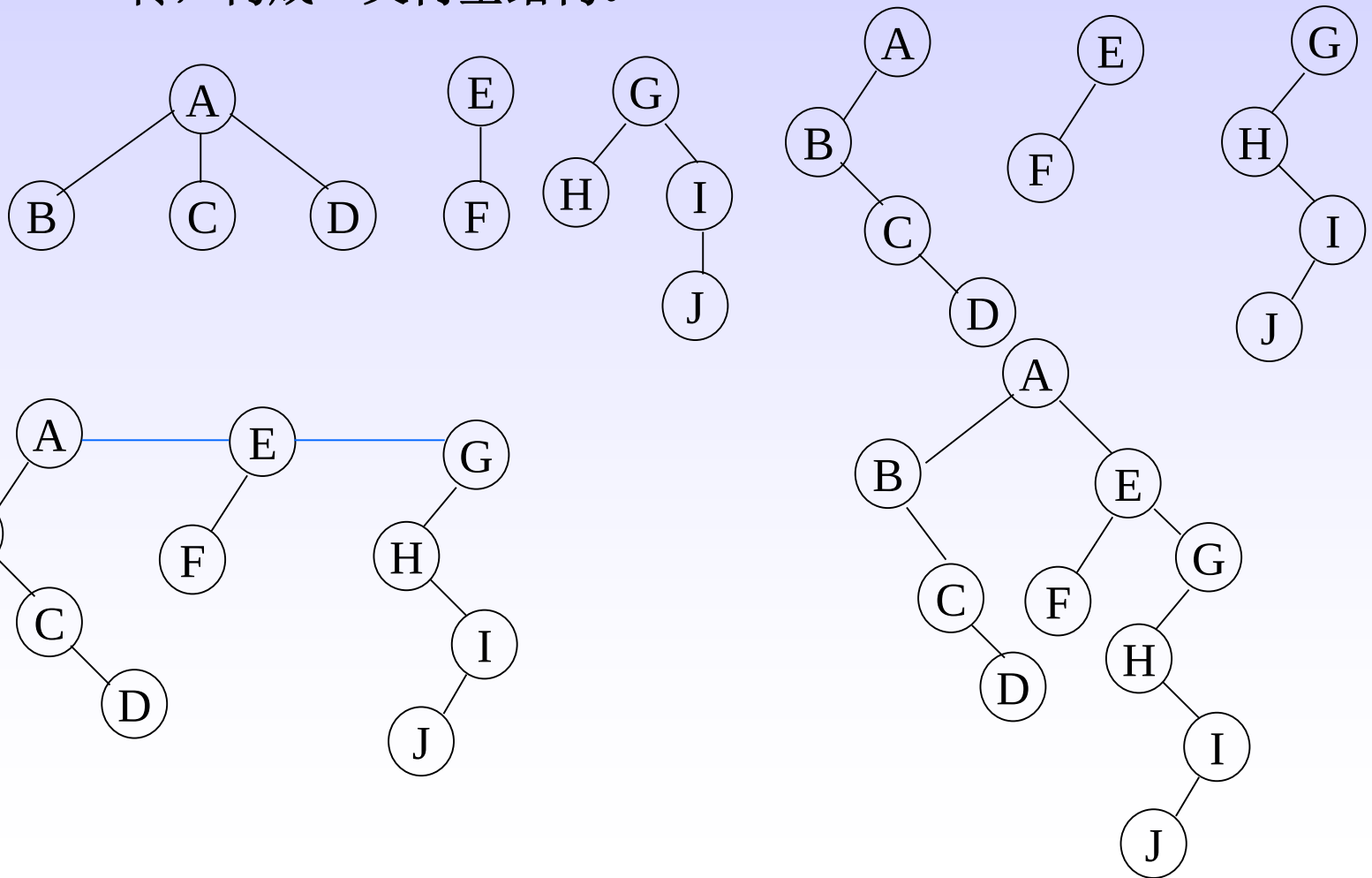


3、森林转换成二叉树

（1）将各棵树分别转换成二叉树；

（2）将每棵树的根结点用线相连；

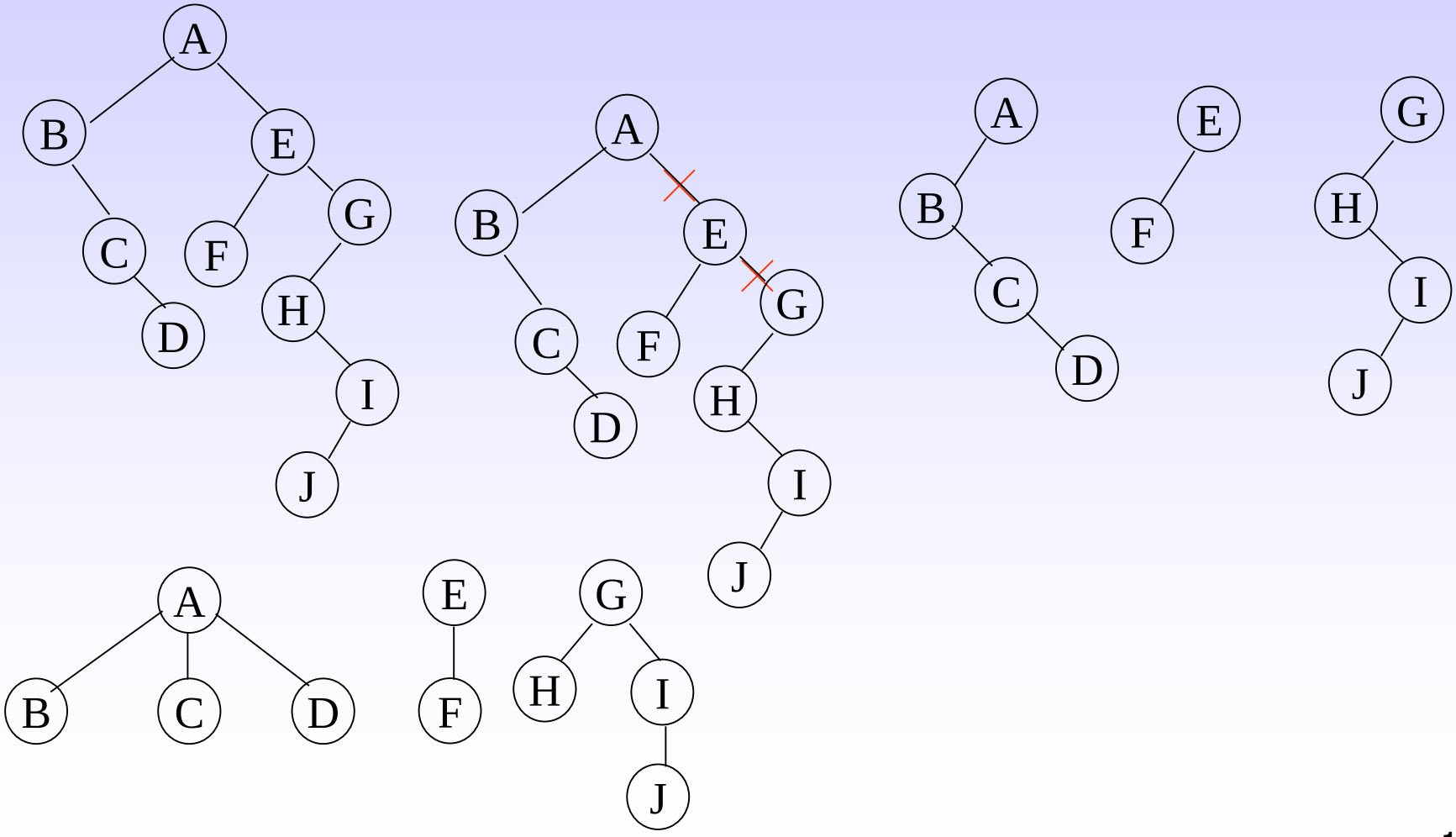
（3）以第一棵树根结点为二叉树的根，再以根结点为轴心，顺时针旋转，构成二叉树型结构。



4、二叉树转换成森林

（1）将二叉树中根结点与其右孩子连线，及沿右分支搜索到的所有右孩子间连线全部抹掉，使之变成孤立的二叉树；

（2）将孤立的二叉树还原成树。

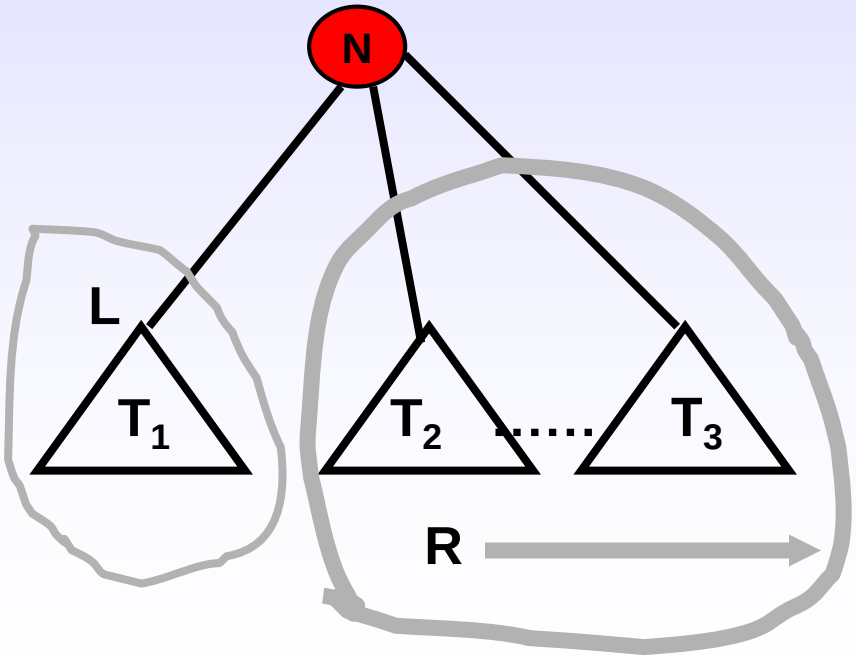


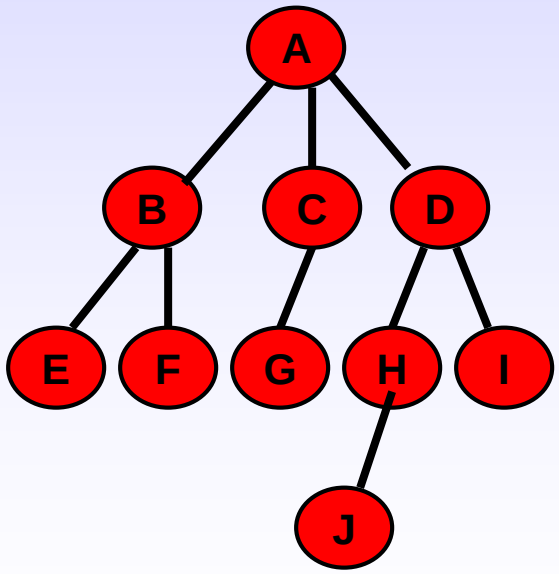
6.4.3 树、森林的遍历

1 树的先根、后根遍历：

1）先根：类似于二叉树的先序遍历（NLR） N:根；L：第一棵子树，R：其余的子树，遍历方向由第二棵子树至最后一棵子树。

2）后根：类似于二叉树的后序遍历（LRN） L：第一棵子树，R：其余的子树，遍历方向由第二棵子树至最后一棵子树， N:根。



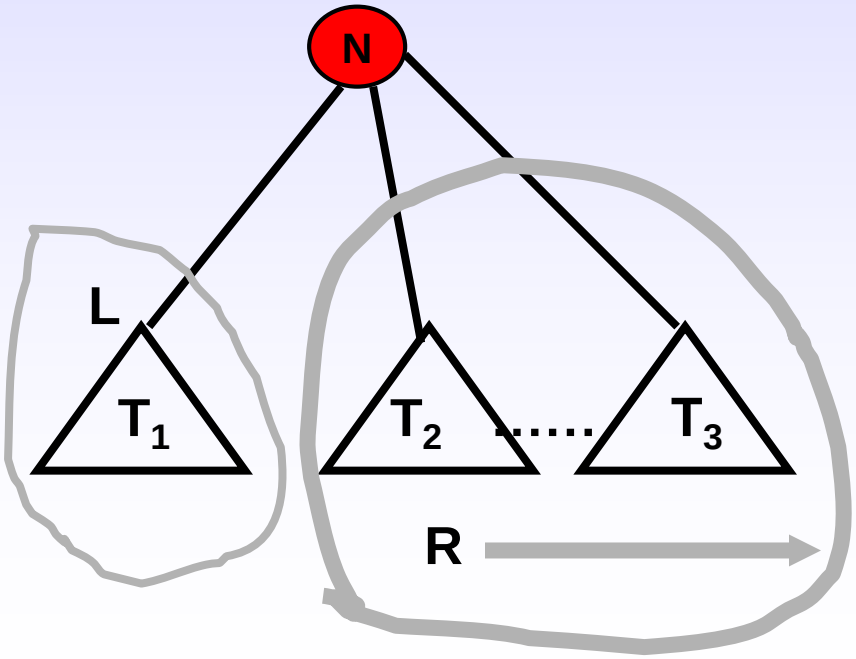


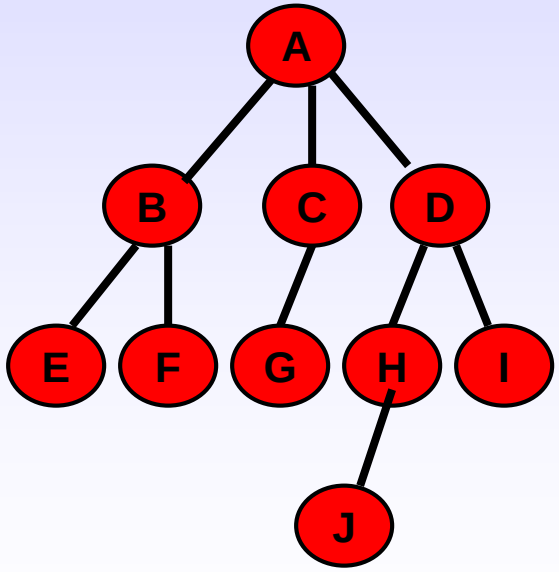
6.4.3 树、森林的遍历

1 树的先根、后根遍历：

1）先根：类似于二叉树的先序遍历（NLR） N:根；L：第一棵子树，R：其余的子树，遍历方向由第二棵子树至最后一棵子树。

2）后根：类似于二叉树的后序遍历（LRN） L：第一棵子树，R：其余的子树，遍历方向由第二棵子树至最后一棵子树， N:根。





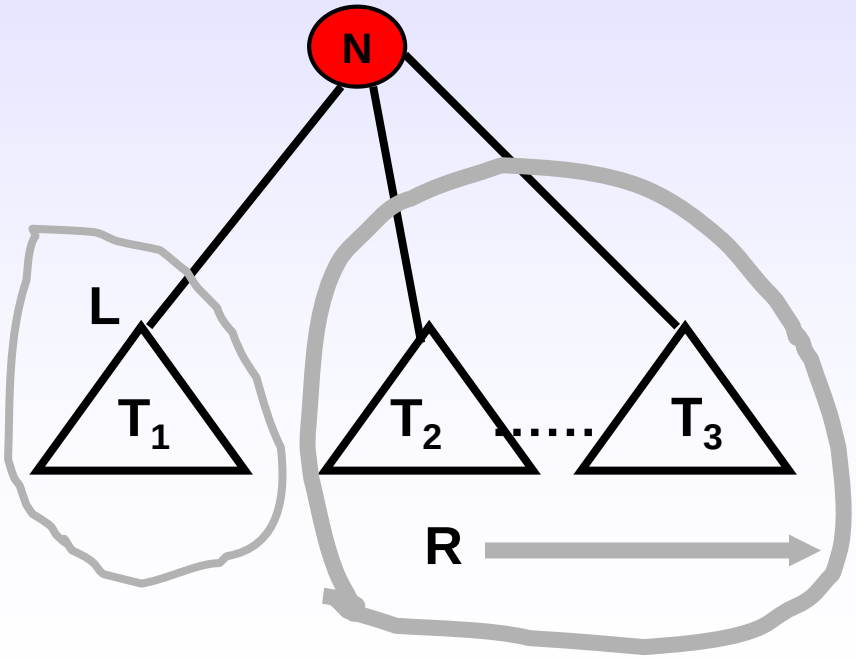
先根：A B E F C G D H J I

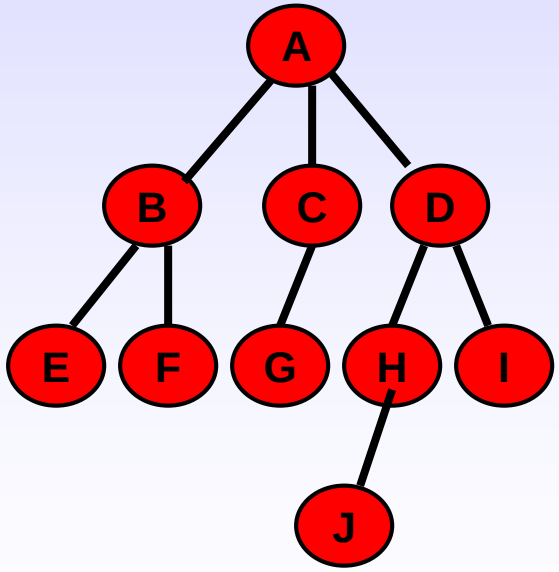
6.4.3 树、森林的遍历

1 树的先根、后根遍历：

1）先根：类似于二叉树的先序遍历（NLR） N:根；L：第一棵子树，R：其余的子树，遍历方向由第二棵子树至最后一棵子树。

2）后根：类似于二叉树的后序遍历（LRN） L：第一棵子树，R：其余的子树，遍历方向由第二棵子树至最后一棵子树， N:根。

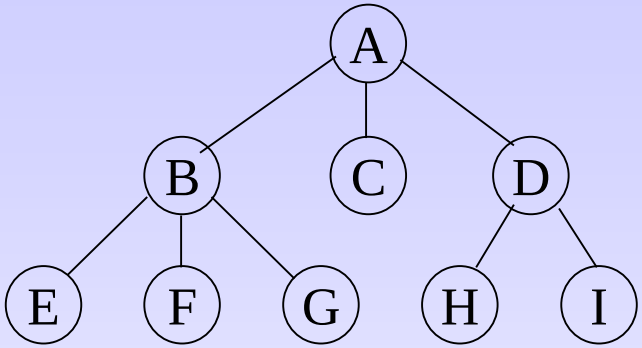




先根：A B E F C G D H J I

后根：E F B G C J H I D A

6.4 树和森林

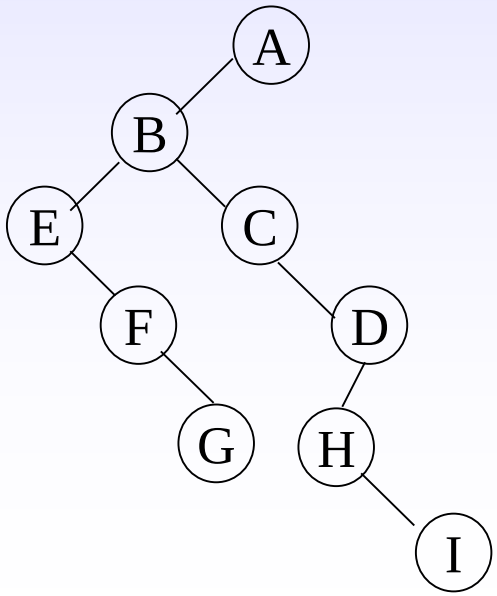


树的遍历：

先根：A B E F G C D H I

后根：E F G B C H I D A

用二叉树的遍历算法
实现树的遍历！



二叉树的遍历：

先序：A B E F G C D H I

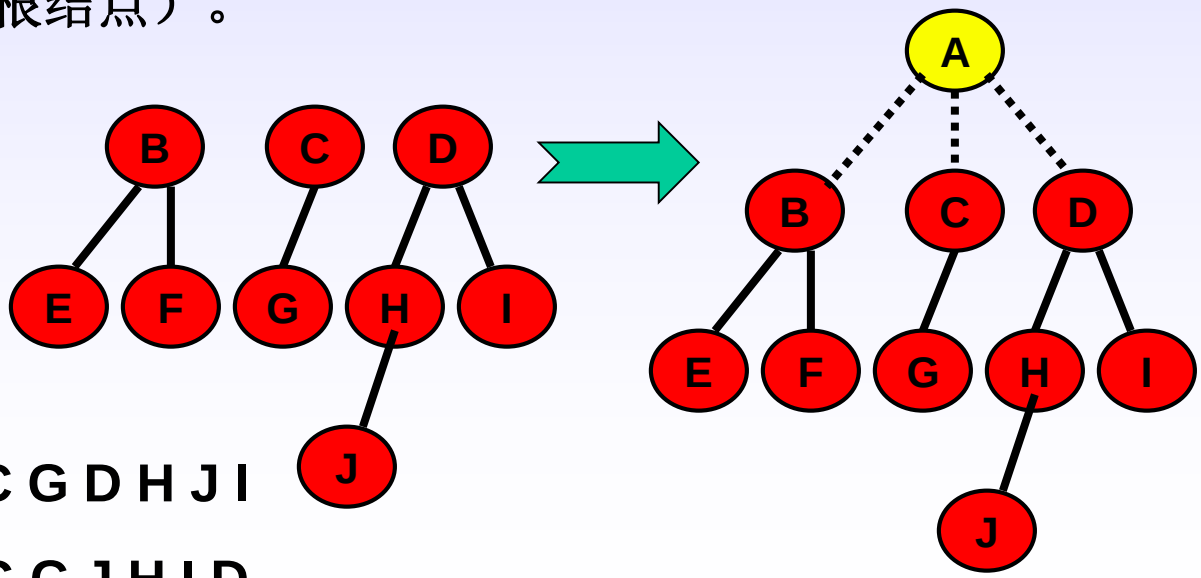
中序：E F G B C H I D A

6.4 树和森林

2 森林的先序、后序遍历：

1）先序遍历类似于树的先序遍历。增加一个虚拟的根结点，它的子结点为各棵树的根。对这棵树进行先根遍历，即得到森林的先序序列（去掉树根结点）。

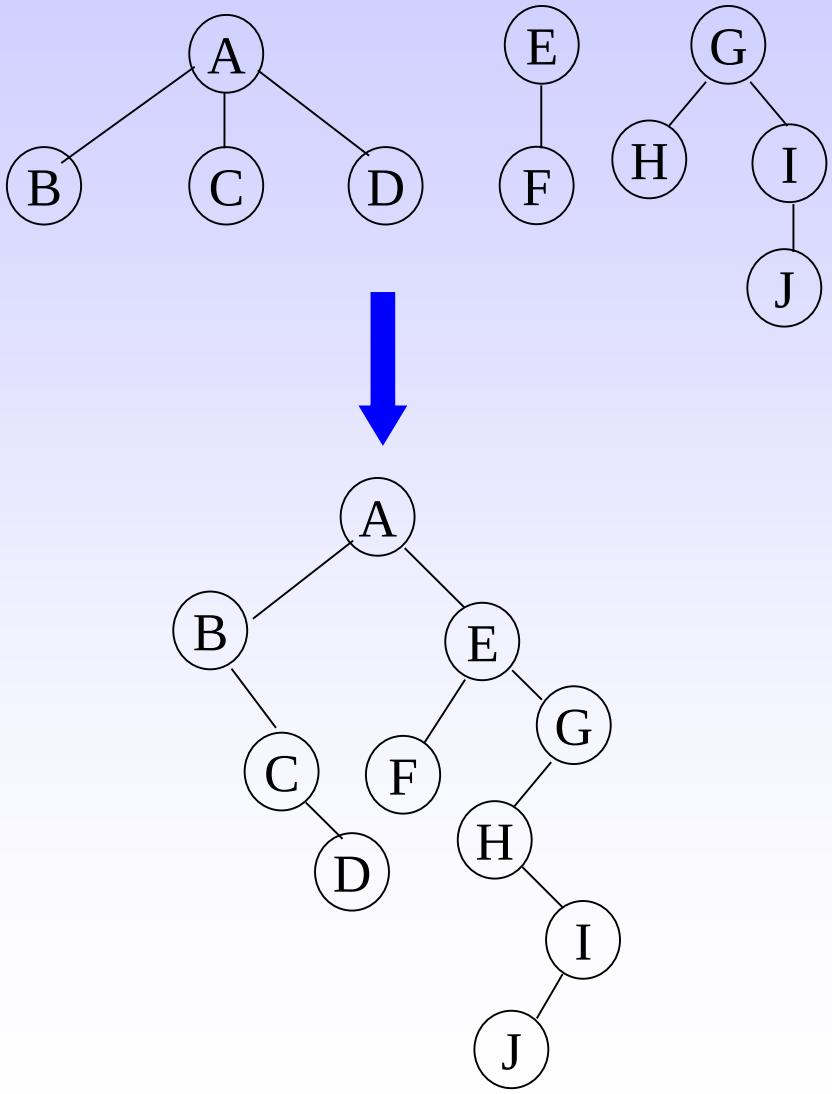
2）后序遍历类似于树的后根遍历。增加一个虚拟的根结点，它的子结点为各棵树的根。对这棵树进行后根遍历，即得到森林的后序序列（去掉树根结点）。



先序：B E F C G D H J I

后序：E F B G C J H I D

6.4 树和森林



森林的遍历：

先根：A B C D E F G H I J

后根：B C D A F E H J I G

用二叉树的遍历算法
实现森林的遍历！

二叉树的遍历：

先序：A B C D E F G H I J

中序：B C D A F E H J I G

6.6 赫夫曼树及其应用

6.6.1 最优二叉树（赫夫曼树）

●路径长度：结点之间的树枝的总数

●树的路径长度：从根到每一结点的路径长度之和

●树的带权路径长度(WPL)：

设树有n个叶子结点,每个叶子结点都带有一个权值，则树的带权路径长度之和为：

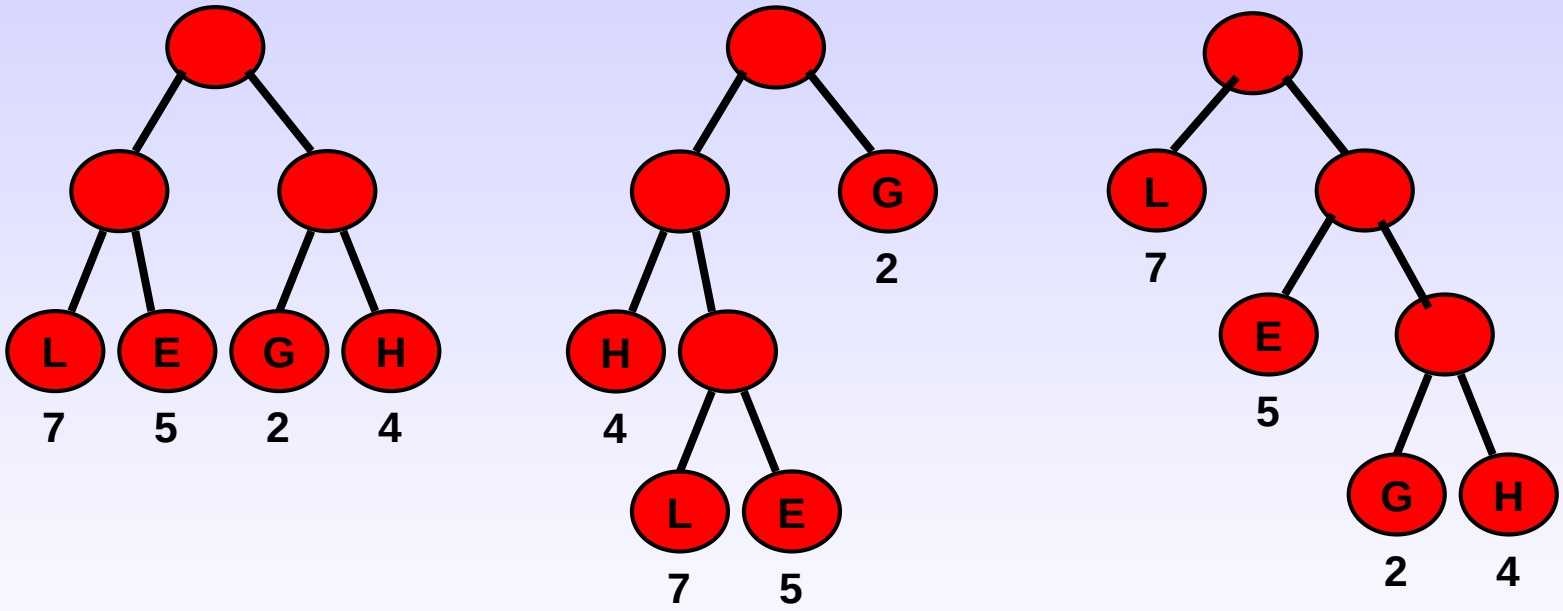
$$WPL = \sum_{i=1}^{n} w_i * l_i$$

其中：l_i为第i个叶子结点的路径长度；

w_i为第i个叶子结点的权值。

6.6 赫夫曼树及其应用

- 最优二叉树(赫夫曼树)：给定n个权值{w1,w2,…,wn},构造一个有n个叶子结点的树，使得树的带权路径长度 WPL 最小的二叉树。

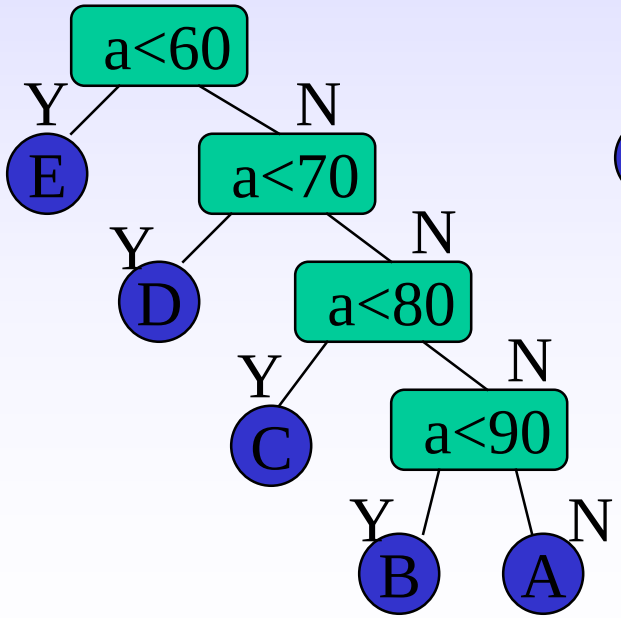


- 叶子结点权值相同时，最优二叉树的WPL唯一确定，但树的形态不唯一。

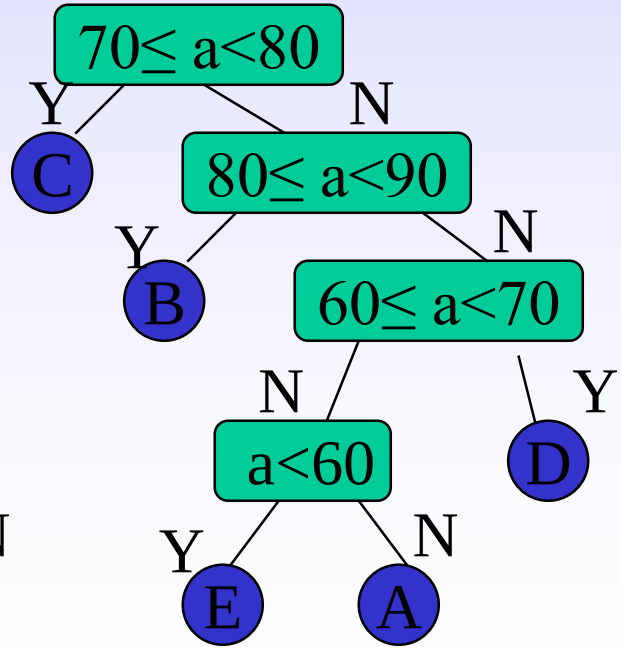
6.6 赫夫曼树及其应用

例：判定树：

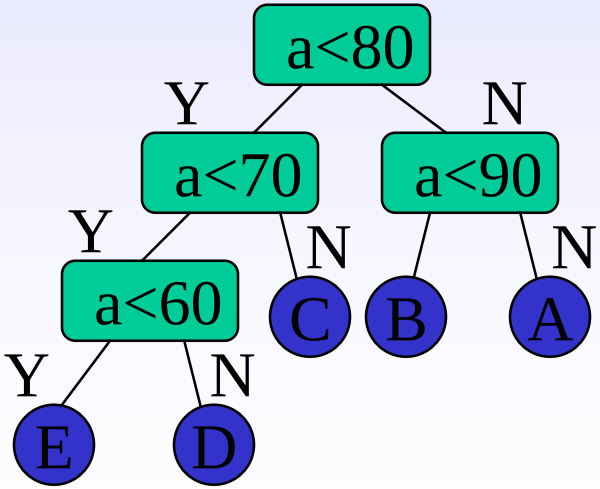
分数	0~59	60~69	70~79	80~89	90~100
等级	E	D	C	B	A
比例	0.05	0.15	0.40	0.30	0.10



WPL=3.15



WPL=2.05



WPL=2.2

6.6 赫夫曼树及其应用

赫夫曼算法：由给定的权值构造赫夫曼树

(1) 由给定的n个权值$\{w_0, w_1, w_2, \ldots, w_{n-1}\}$，构造具有$n$棵二叉树的集合$F = \{T_0, T_1, T_2, \ldots, T_{n-1}\}$，其中每一棵二叉树$T_i$只有一个带有权值$w_i$的根结点，其左、右子树均为空。

(2) 重复以下步骤，直到F中仅剩下一棵树为止：

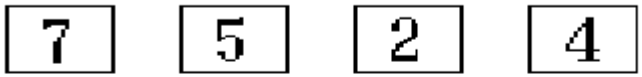
① 在F中选取两棵根结点的权值最小的二叉树，做为左、右子树构造一棵新的二叉树。置新的二叉树的根结点的权值为其左、右子树上根结点的权值之和。

② 在F中删去这两棵二叉树。

③ 把新的二叉树加入F。

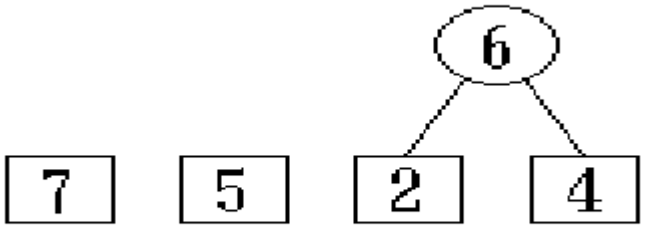
6.6 赫夫曼树及其应用

F：{7}{5}{2}{4}



（a） 初始

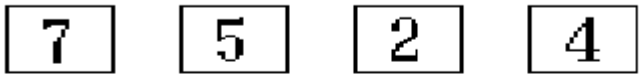
F：{7}{5}{6}



（b） 合并{2}{4}

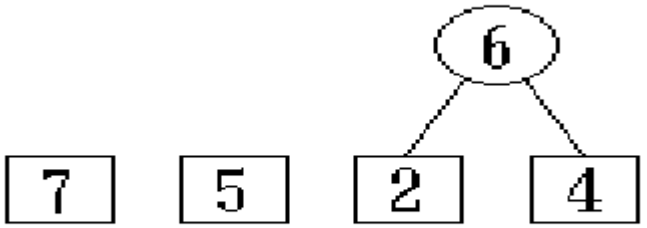
6.6 赫夫曼树及其应用

F：{7}{5}{2}{4}



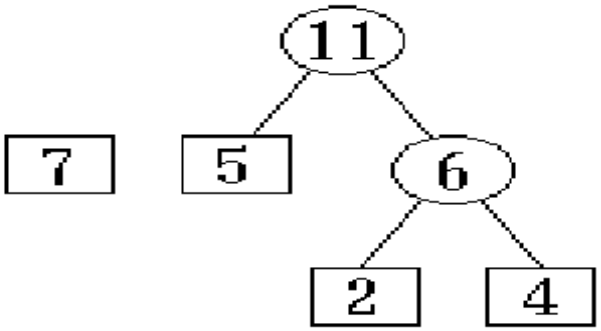
（a）初始

F：{7}{5}{6}



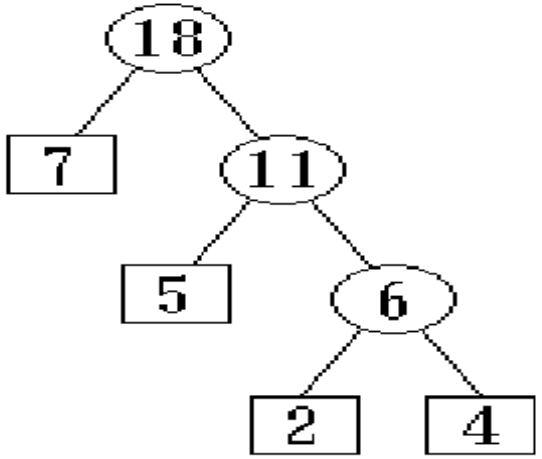
（b）合并{2}{4}

F：{7}{11}



（c）合并{5}{6}

F：{18}



（d）合并{7}{11}

6.6 赫夫曼树及其应用

6.3.2 赫夫曼编码

赫夫曼编码主要用途是实现文本数据的压缩。

设一段电文：CAST CAST SAT AT A TASA，字符集合是{C,A,S,T}，各个字符出现的频度(次数)是W={2,7,4,5}。若给每个字符以***等长编码***A:00；T:10；C:01；S:11，则总编码长度为(2+7+4+5)*2=36。若按各个字符出现的频度不同而给予***不等长编码***，即频度大的字符采用短的编码，频度小的字符采用长的编码，则可望减少总编码长度。

6.6 赫夫曼树及其应用

6.3.2 赫夫曼编码

不等长编码要注意任何一个字符的编码都不是另外一个字符编码的前缀。

否则译码时将产生二义性。

例如，编码不等长为 A:0；T:1；　C:01；S:11，则“0110”可译为

“ATTA”、“CTA”、“ASA”等。

6.6 赫夫曼树及其应用

[赫夫曼编码]：

设：$d=\{d_1, d_2, \cdots, d_n\}$，$w=\{w_1, w_2, \cdots, w_n\}$，其中d为要编码的字符集合，w为d中各种字符出现的频率。

要求：（1）编码总长最短；

（2）若$d_i \neq d_j$，d_i编码不是d_j编码的前缀。

[前缀编码]：

一个编码集合中，任何一个字符的编码都不是其它字符编码的前缀，这种编码称为前缀编码。

可以利用二叉树来设计前缀编码，约定叶结点表示字符。从根结点到叶的路径中，左分支表示‘0’，右分支表示‘1’，从根结点到叶结点上的路径分支所组成的01串可作为该叶结点字符的前缀编码。

6.6 赫夫曼树及其应用

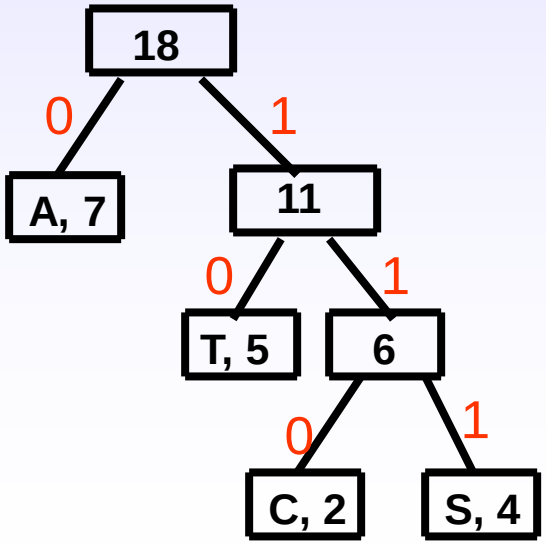
[用赫夫曼树完成赫夫曼编码]：

把$d_1, d_2, \cdots, d_n$ 作为叶子结点，把$w_1, w_2, \cdots, w_n$作为叶子结点的权，构造赫夫曼树。在赫夫曼树中结点的左分支赋0，右分支赋1，从根结点到叶子结点的路径上的数字拼接起来就是这个叶子结点字符的编码。

例右图中各叶子结点的赫夫曼编码为：

A:0；T:10；C:110；S:111。

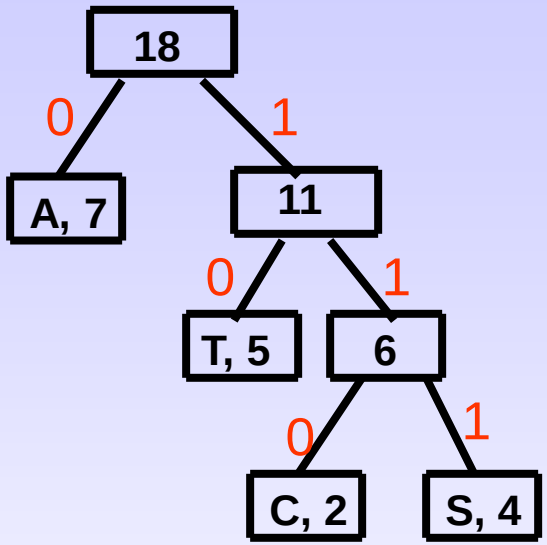
总编码长度为：7*1+5*2+(2+4)*3=35。



采用赫夫曼编码，电文的总编码长度正好等于赫夫曼树的带权路径长度WPL，即电文的总编码长度最短。

6.6 赫夫曼树及其应用

```
typedef struct
{     unsigned int weight ;
      unsigned int  parent, lchild, rchild ;
 } HTNode, * HuffmanTree;
 typedef char ** HufmanCode ;
```



赫夫曼编码的实现：

1、建立具有2n-1个单元的数组，其中n个单元用于保存初始结点，n-1个结点用于表示内部结点。

2、构造一棵赫夫曼树，即执行n-1次循环，每次产生一个内部结点。权值最小的两个结点为其左右子结点。

3、根据赫夫曼树计算每个字符的赫夫曼编码。

	wt	pa	lc	rc
1	7	0	0	0
2	5	0	0	0
3	2	0	0	0
4	4	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0

(1)

	wt	pa	lc	rc
1	7	0	0	0
2	5	0	0	0
3	2	5	0	0
4	4	5	0	0
5	6	0	3	4
6	0	0	0	0
7	0	0	0	0

(2)

	wt	pa	lc	rc
1	7	0	0	0
2	5	6	0	0
3	2	5	0	0
4	4	5	0	0
5	6	6	3	4
6	11	0	2	5
7	0	0	0	0

(3)

`	wt	pa	lc	rc
1	7	7	0	0
2	5	6	0	0
3	2	5	0	0
4	4	5	0	0
5	6	6	3	4
6	11	7	2	5
7	18	0	1	6

(4)

6.6 赫夫曼树及其应用

```
void  hufmanCode(HufumanTree &HT, HumanCode &HC, int *w, int n)
{  if  ( n<=1 ) return;
    m = 2*n-1;
    HT=(HuffmanTree)malloc((m+1)*sizeof(HTNode)); // 不用0号单元
    for ( p=HT+1, i=1; i<=n; ++i, ++p, ++w )   *p = { *w, 0, 0, 0 } ;
    for ( ; i <= m; ++i, ++p )  *p = { 0, 0, 0, 0 } ;
    for ( i = n+1; i <= m; ++i )  // 构造赫夫曼树
    {  Select ( HT, i-1, s1, s2 ) ;
        HT[s1].parent = i; HT[s2].parent = i;
        HT[i].lchild = s1; HT[i].rchild = s2  ;
        HT[i].weight =  HT[s1].weight + HT[s2].weight;
    }
```

6.6 赫夫曼树及其应用

```
void  hufmanCode(HufumanTree &HT, HumanCode &HC, int *w, int n)
{  if  ( n<=1 ) return;
    m = 2*n-1;
    HT=(HuffmanTree)malloc((m+1)*sizeof(HTNode)); // 不用0号单元
    for ( p=HT+1, i=1; i<=n; ++i, ++p, ++w )   *p = { *w, 0, 0, 0 } ;
    for ( ; i <= m; ++i, ++p )  *p = { 0, 0, 0, 0 } ;
    for ( i = n+1; i <= m; ++i )  // 构造赫夫曼树
    {  Select ( HT, i-1, s1, s2 ) ; //小于等于i-1、根结点权值最小、无双亲
        HT[s1].parent = i; HT[s2].parent = i;
        HT[i].lchild = s1; HT[i].rchild = s2  ;
        HT[i].weight =  HT[s1].weight + HT[s2].weight;
    }
```

6.6 赫夫曼树及其应用

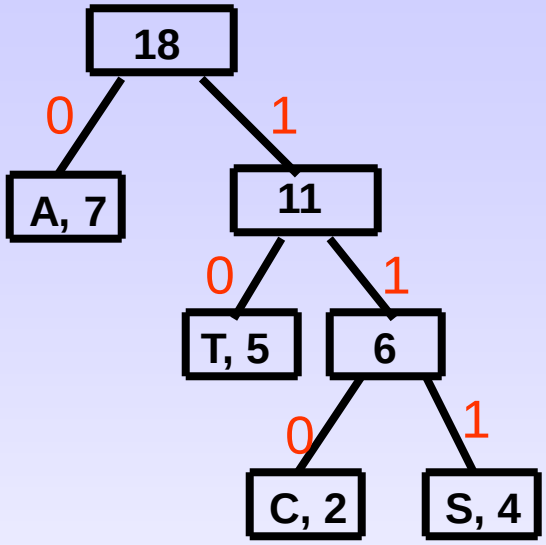
HC

字符	编码			
A，7	0	‘/0’		
T，5	1	0	‘/0’	
C，2	1	1	0	‘/0’
S，4	1	1	1	‘/0’

cd（在求T的编码时的状态）

	1	0	‘/0’

start=1



	wt	pa	lc	rc
1	7	7	0	0
2	5	6	0	0
3	2	5	0	0
4	4	5	0	0
5	6	6	3	4
6	11	7	2	5
7	18	0	1	6

6.6 赫夫曼树及其应用

```
    HC = ( HuffmanCode ) malloc (( n + 1) * sizeof ( char * ));
    cd = ( char * ) malloc ( n * sizeof ( char ) );
    cd[n-1] ='/0' ;
    for ( i = 1; i <= n; ++i )    // 求每个字符的赫夫曼编码
    {  start = n-1;
       for ( c=i, f=HT[i].parent;  f != 0;  c=f, f=HT[f].parent )
          if (HT[f].lchild == c )   cd[--start ] = '0' ;
          else  cd [--start ] = '1' ;
       HC[i] = ( char * ) malloc (( n - start ) * sizeof ( char ));
       strcpy(HC[i], &cd[start]);
    }
    free(cd);
}
```

6.6 赫夫曼树及其应用

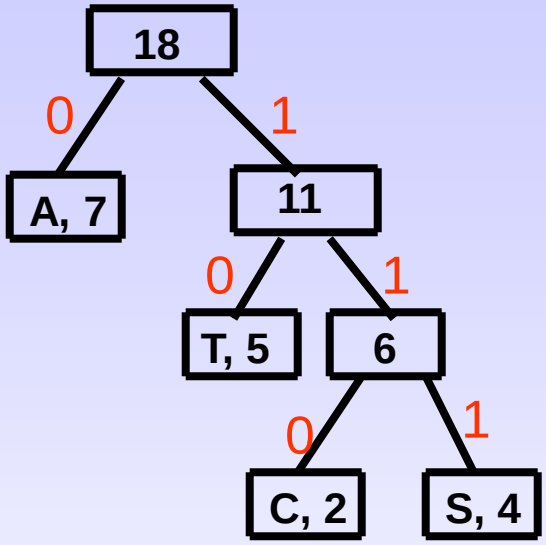
HC

字符	编码			
A，7	0	‘/0’		
T，5	1	0	‘/0’	
C，2	1	1	0	‘/0’
S，4	1	1	1	‘/0’

cd（在求T的编码时的状态）

	1	0	‘/0’

start=1



	wt	pa	lc	rc
1	7	7	0	0
2	5	6	0	0
3	2	5	0	0
4	4	5	0	0
5	6	6	3	4
6	11	7	2	5
7	18	0	1	6