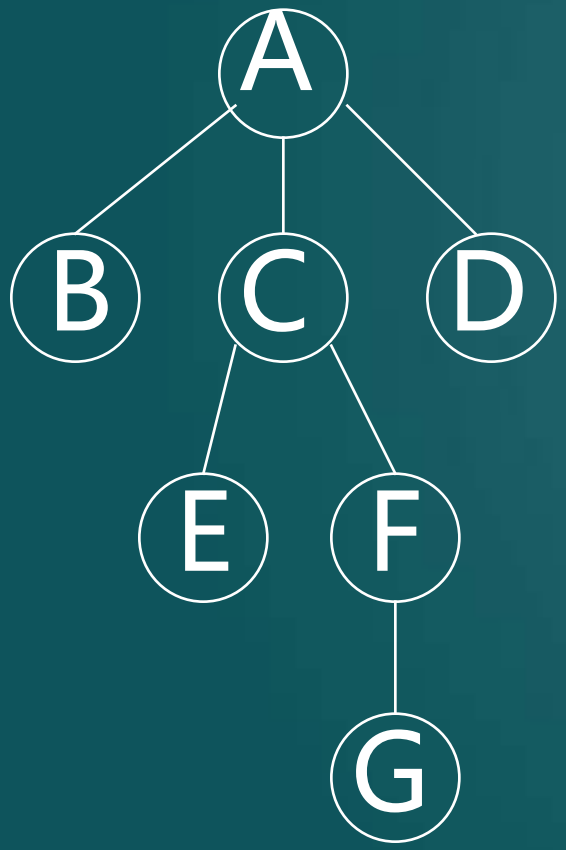
6.4 树和森林

6.4.1 树的存储结构

一、双亲表示法:



	data	parent
0	A	-1
1	B	0
2	C	0
3	D	0
4	E	2
5	F	2
6	G	5

r=0　　n=7

结点结构:

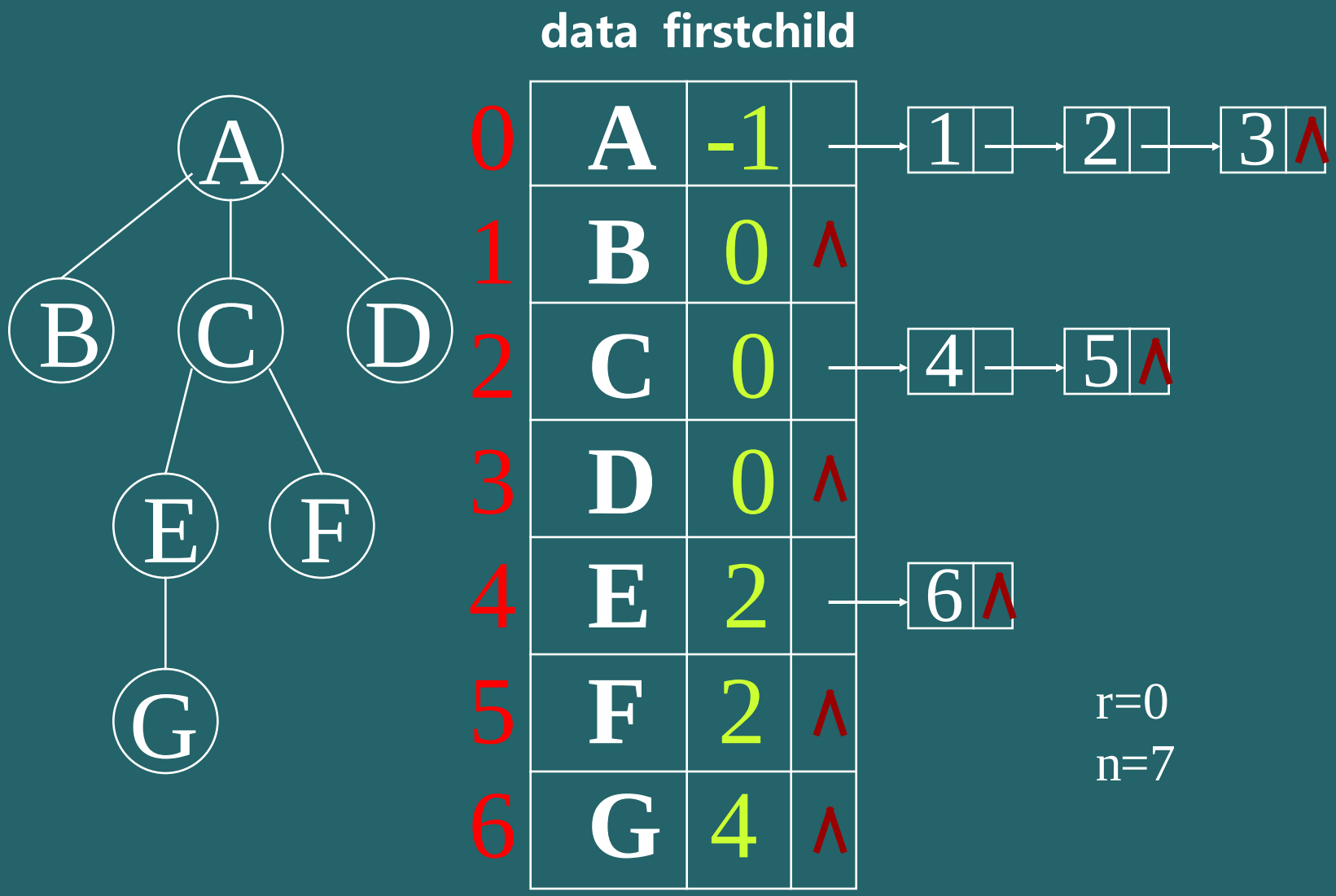
C语言的类型描述:

```
#define MAX_TREE_SIZE  100
  typedef struct PTNode {
      Elem  data;
      int    parent;   // 双亲位置域
   } PTNode;
```

树结构:

```
typedef struct {
     PTNode  nodes [MAX_TREE_SIZE];
     int    r, n;    // 根结点的位置和结点个数
   } PTree;
```

二、孩子链表表示法:


data firstchild
A
B
C
D
E
F
G
0 A -1
1 B 0
2 C 0
3 D 0
4 E 2
5 F 2
6 G 4
1
2
3
4
5
6
r=0
n=7

C语言的类型描述：

�孩子结点结构：


```
typedef struct CTNode {
    int         child ;
    struct CTNode *next ;
  } *ChildPtr ;
```

Next: 指向下一孩子结点的指针

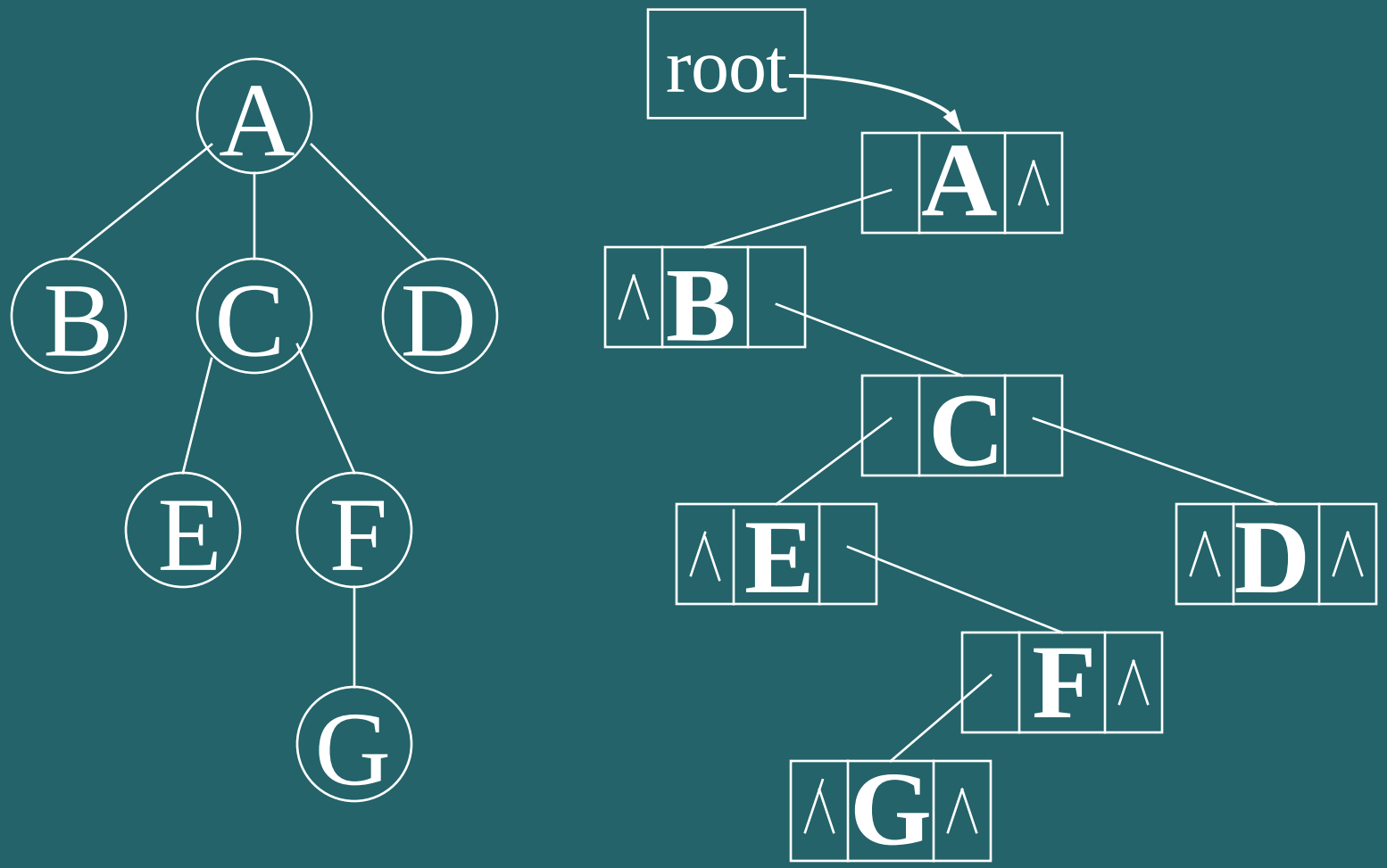
双亲结点结构


```
typedef struct {
  Elem    data ;
  ChildPtr  firstchild; ;    // 孩子链的头指针
} CTBox ;
```

树结构：

```
typedef struct {
    CTBox  nodes[MAX_TREE_SIZE] ;
    int    n , r ;  // 结点数和根结点的位置
  } Ctree ;
```

三、树的二叉链表（孩子-兄弟）存储表示法



C语言的类型描述:

结点结构:


```
typedef struct CSNode{
    Elem        data；
    struct CSNode
        *firstchild,，*nextsibling；
} CSNode，*CSTree；
```

其中，firstchild：指向左边第一个子结点的指针，
nextsibling：指向右边第一个兄弟结点的指针。

6.4.2 森林与二叉树的转换

设 森林 $F = (T_1, T_2, \ldots, T_n)$；

$T_1 = (root, t_{11}, t_{12}, \ldots, t_{1m})$；

二叉树

$B = (LBT, Node(root), RBT)$；

1、由森林转换成二叉树的转换规则为:

若 **F = Φ**，则 **B = Φ**；

否则，

由 **ROOT(T_1)** 对应得到 **Node(root)**；

由 **(t_{11}, t_{12}, …, t_{1m})** 对应得到 **LBT**；

由 **(T_2, T_3,…, T_n)** 对应得到 **RBT**。

2、**由二叉树转换为森林**的转换规则为：

若 **B = Φ**，则 **F = Φ**；

否则，

由 **Node(root)** 对应得到 **ROOT(T_1)**；

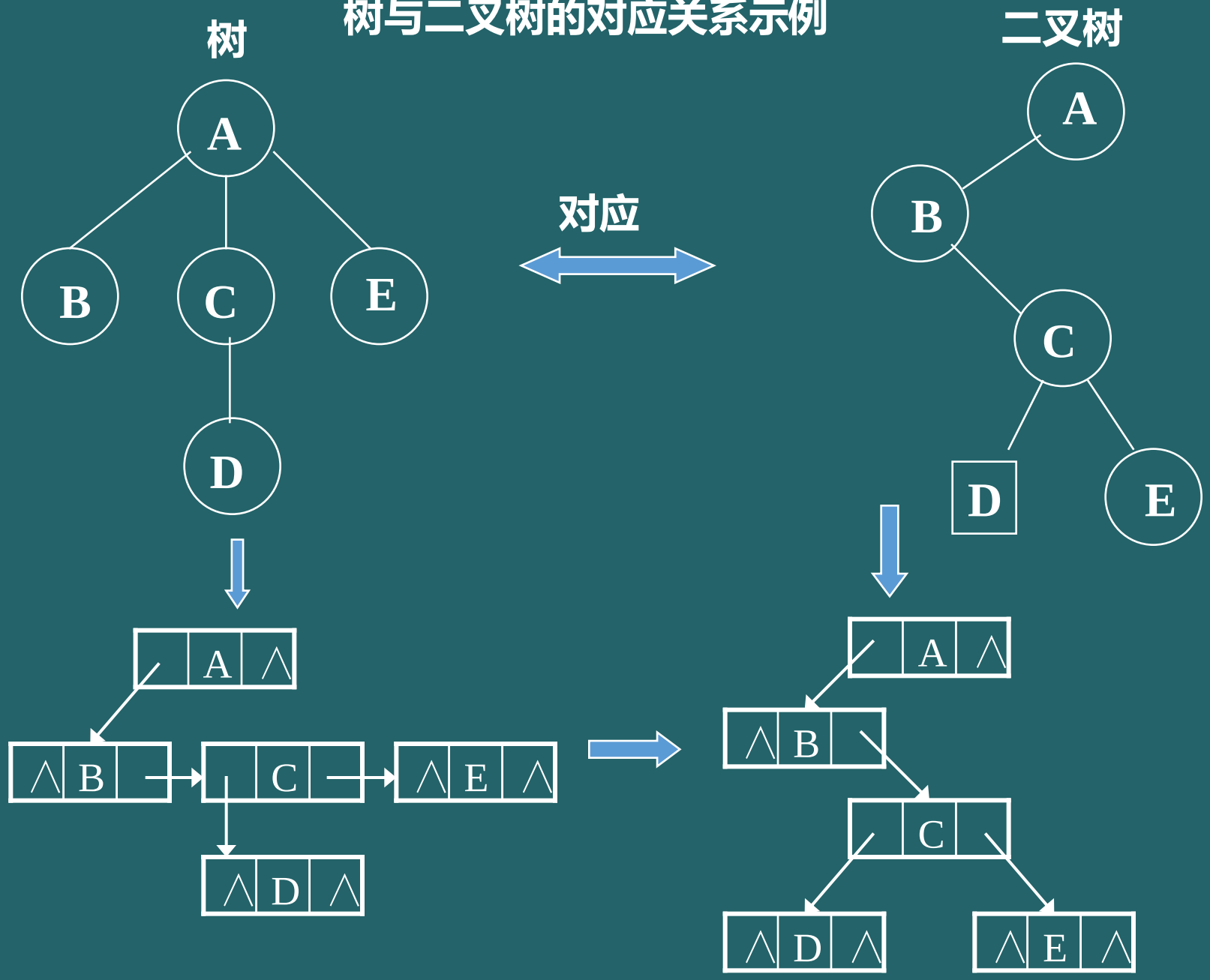
由 **LBT** 对应得到 **(t_{11}, t_{12}, … , t_{1m})**；

由 **RBT** 对应得到 **(T_2, T_3, …, T_n)**。

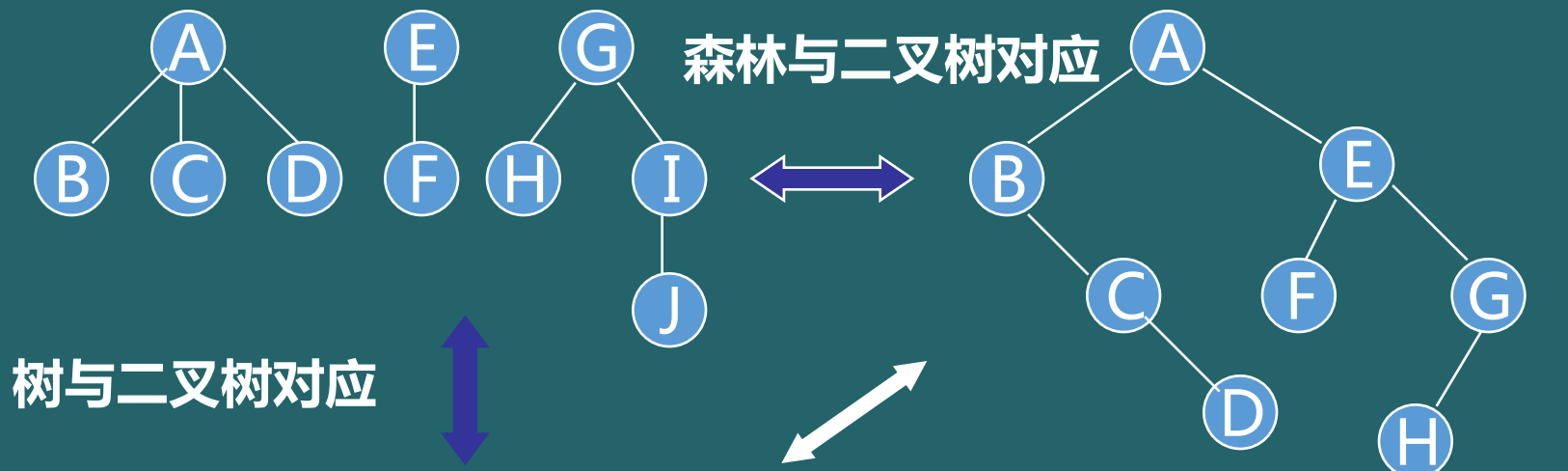
由此，树的各种操作均可对应二叉树的操作来完成。

应当注意的是，和树对应的二叉树，其左、右子树的概念已改变为：左是孩子，右是兄弟。由于树的根结点无兄弟，因此对应二叉树的根结点无右子树。

树与二叉树的对应关系示例



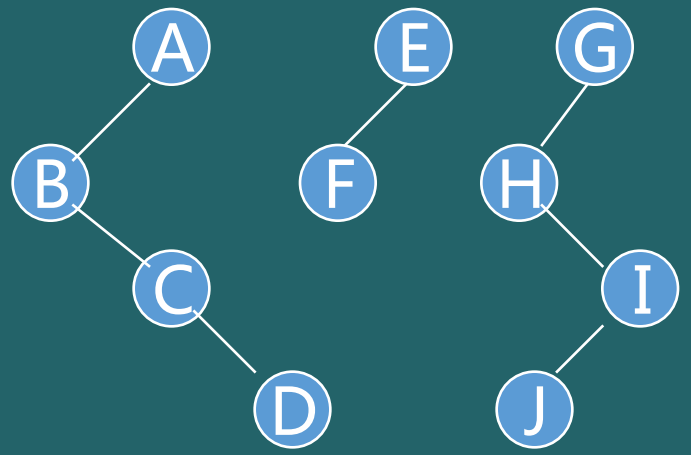
森林与二叉树的对应关系示例



森林与二叉树对应

树与二叉树对应

树根相连



6.4.3 树和森林的遍历

一、树的遍历

二、森林的遍历

三、树的遍历的应用

树的遍历可有三条搜索路径:

先根(次序)遍历:

若树不空，则先访问根结点，然后依次先根遍历各棵子树。

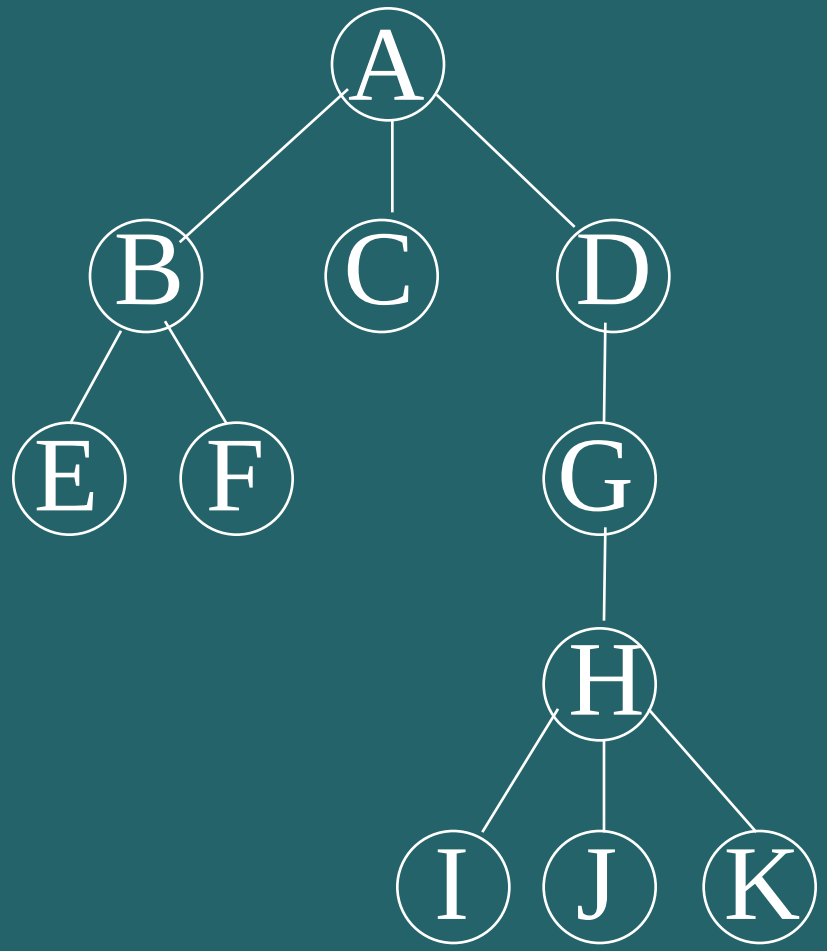
后根(次序)遍历:

若树不空，则先依次后根遍历各棵子树，然后访问根结点。

按层次遍历:

若树不空，则自上而下自左至右访问树中每个结点。

例如，对下面树



先根遍历时顶点的访问次序：

A B E F C D G H I J K

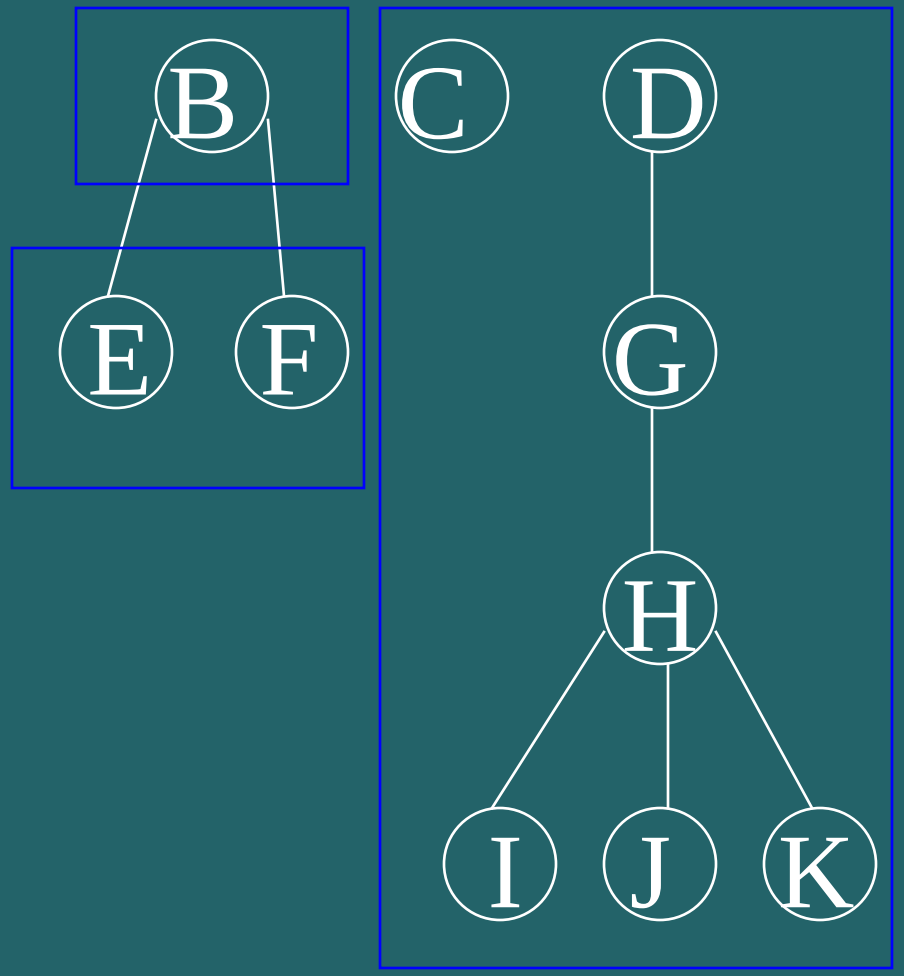
后根遍历时顶点的访问次序：

E F B C I J K H G D A

层次遍历时顶点的访问次序：

A B C D E F G H I J K

森林由三部分构成：



1．森林中第一棵树的根结点；

2．森林中第一棵树的子树森林；

3．森林中其它树构成的森林。

森林的遍历

1. 先序遍历森林

若森林不空，则可按下述规则遍历之：

（1）**访问**森林中第一棵树的根结点；

（2）**先序遍历**森林中第一棵树的子树森林；

（3）**先序遍历**森林中(除第一棵树之外)其余树构成的森林。

即：依次从左至右对森林中的每一棵树进行先根遍历。

2．中序遍历森林

若森林不空，则可按下述规则遍历之：

（1）**中序遍历**森林中第一棵树的子树森林；

（2）**访问**森林中第一棵树的根结点；

（3）**中序遍历**森林中(除第一棵树之外)其余树构成的森林。

即：依次从左至右对森林中的每一棵树进行后根遍历。

树的遍历和二叉树遍历的对应关系 ？

树	森林	二叉树
先根遍历	先序遍历	先序遍历
后根遍历	中序遍历	中序遍历

设树的存储结构为孩子兄弟链表

```
typedef struct CSNode{
    Elem        data;
    struct CSNode *firstchild, *nextsibling;
} CSNode, *CSTree;
```

求树的深度？

求树的深度的算法：

```
int TreeDepth(CSTree T) {
  if(!T)  return 0;
  else {
     h1 = TreeDepth( T->firstchild );
     h2 = TreeDepth( T->nextsibling);

  }  return(max(h1+1, h2));
} // TreeDepth
```