6.5二叉树的遍历及线索化（续）

DS/SS

线索二叉树

- 遍历本质是结点之间非线性关系线性化的过程
- 遍历后的元素之间的某种线性关系一般隐藏在遍历规则下
- 需要多次对同一棵树遍历时，如何提高效率？
- 在二叉链表结构中增加线索域，显式描述遍历后的线索关系
- 节省线索域空间，充分利用二叉链表中空的n+1个指针域
- 线索链表：二叉树的存储结构，结点结构定义见前面。
- 线索：指向结点前驱和后继的指针，叫做线索。
- 线索二叉树：加上线索的二叉树。
- 线索化：对二叉树以某种次序遍历使其变为线索二叉树的过程。

6.5二叉树的遍历及线索化（续）

DS/SS

（二叉）线索链表存储

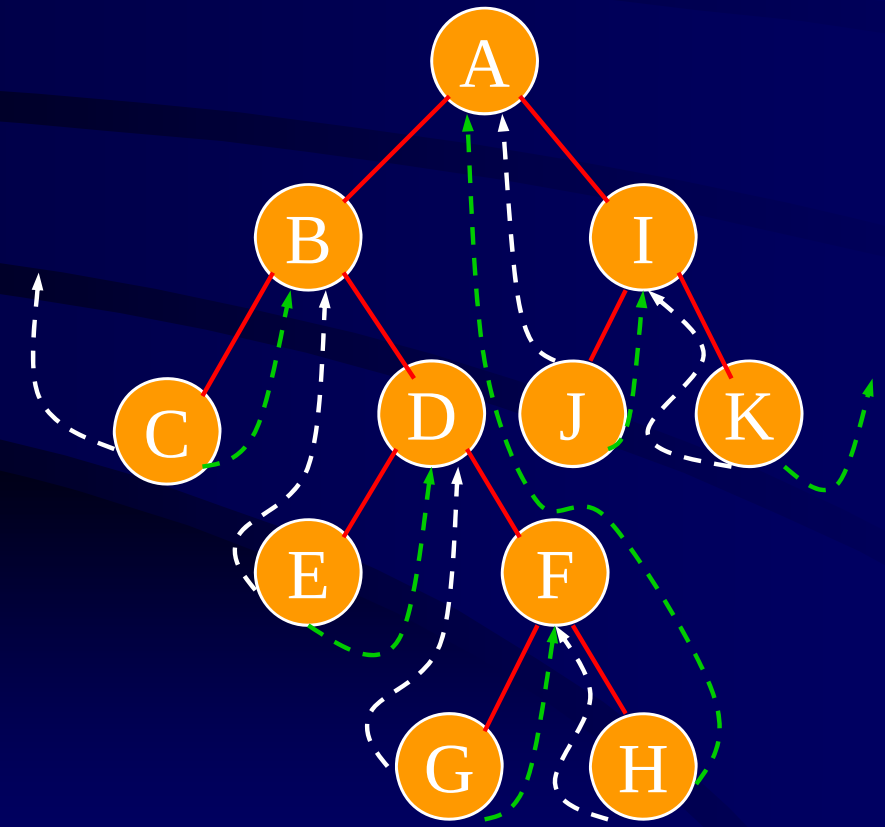
结点	data	lchild	ltag	rchild	rtag

```
//----------二叉树的二叉线索链表存储表示-----------
typedef  enum {Link,Thread}  PointerTag;   //Link = =0; Thread = = 1
typedef  struct  BiThrNode{
            TElemType        data;      //数据域
            struct BiThrNode              *lchild,  *rchild;   //左、右孩子指针
            PointerTag          LTag,   Rtag;                  //左、右标志
} BiThrNode,    *BiThrTree;
```

6.5二叉树的遍历及线索化（续）

DS/SS

（二叉）线索链表存储（中序遍历）



6.5二叉树的遍历及线索化（续）

DS/SS

线索二叉树

线索二叉树上遍历的过程，就是根据线索和遍历规则不断找当前结点后继结点的过程。

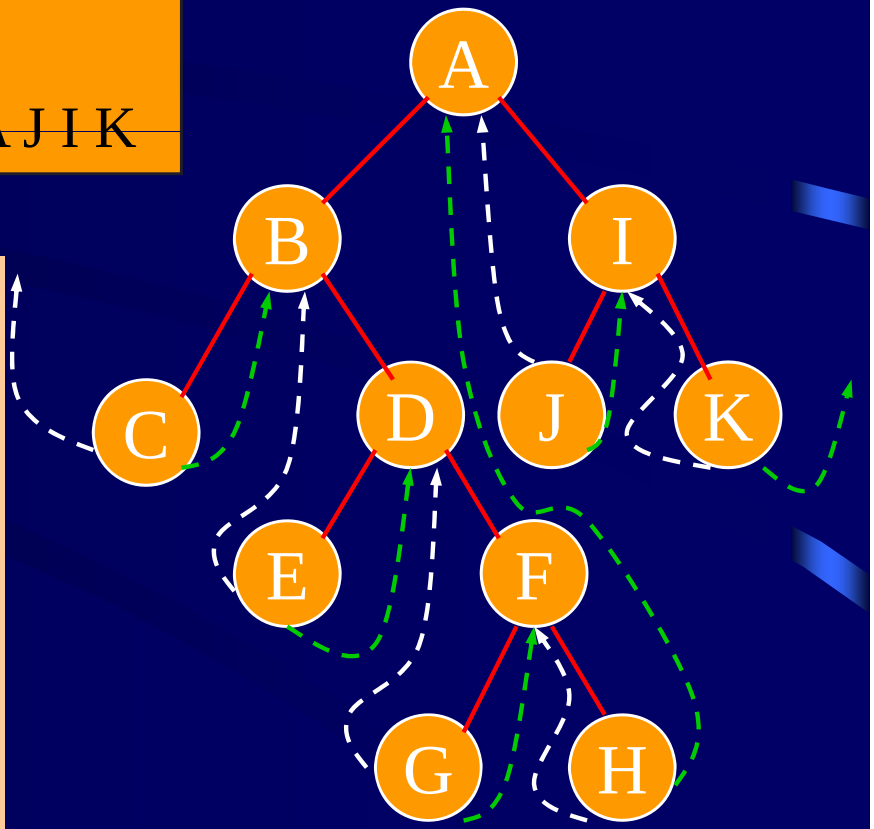
6.5二叉树的遍历及线索化（续）

线索二叉树上的中序遍历

设当前结点指针为p，其前驱结点为q，后继结点指针为s,则有：

InOrder:

C B E D G F H A J I K



求结点p的后继：

1.若 p->rtag = 1

则 s = p->rchild;

2.若p->rtag = 0

s为p的右子树的中序遍历序列的第一个结点，即右子树最左下结点

求结点p的前驱：

1.若 p->ltag = 1

则 q = p->lchild;

2.若p->ltag = 0

q为p的左子树的中序遍历序列的最后一个结点，即左子树最右下结点

6.5二叉树的遍历及线索化（续）

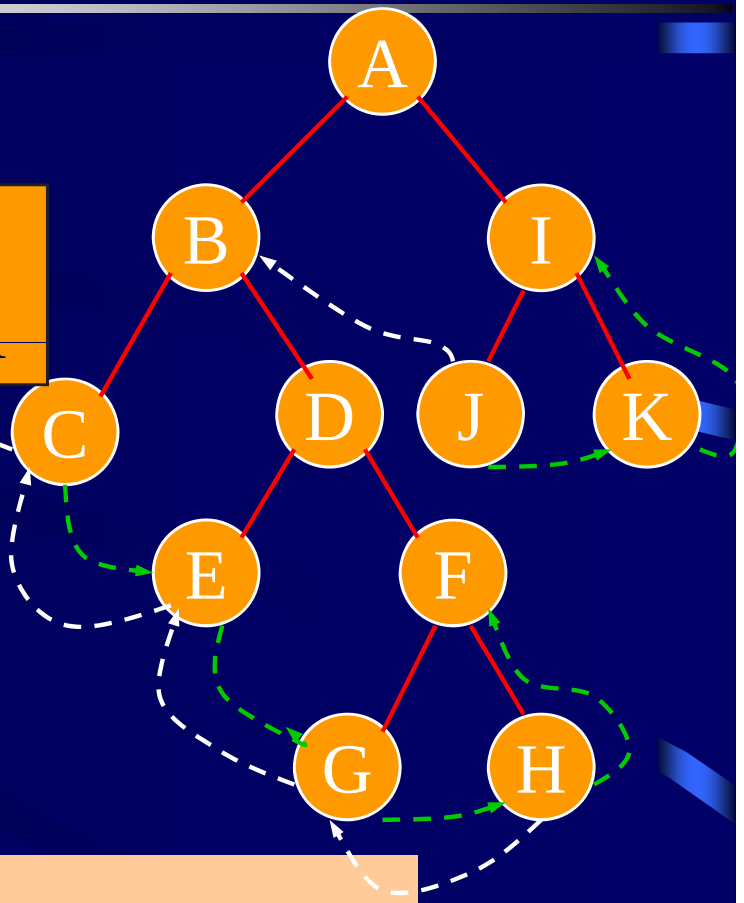
DS/SS

线索二叉树上的后序遍历

设当前结点指针为p，其父结点指针为f,其前驱结点为q，后继结点指针为s,则有：

PostOrder:

C E G H F D B J K I A



求结点p的后继：

1.若 p->rtag = 1, 则 s = p->rchild;

2.若p->rtag = 0

*如果p为其父结点的右儿子，则： s=f;

*如果p为其父结点的左儿子，且其父结点没有右子树，则： s=f

*如果p为其父结点的左儿子，且其父结点有右子树,则s为其父结点的右子树后序序列的第一个结点。

求结点p的前驱：

1.若 p->ltag = 1 则 q = p->lchild;

2.若 p->ltag = 0 则

*若p有右儿子， q = p->rchild

*否则， q = p->lchild

6.5二叉树的遍历及线索化（续）

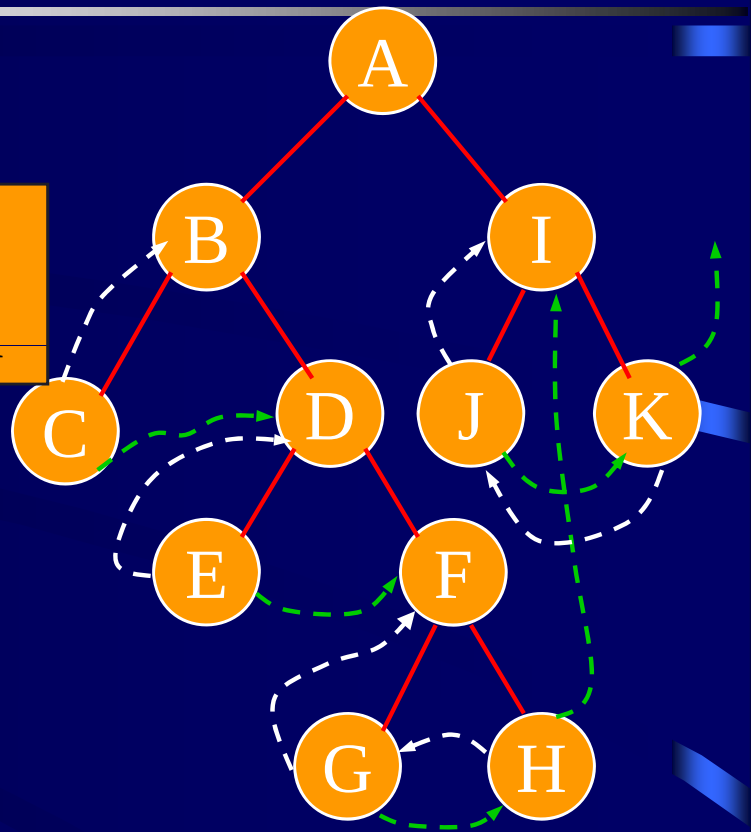
DS/SS

线索二叉树上的前序遍历

设当前结点指针为p，其父结点指针为f,其前驱结点为q，后继结点指针为s,则有：

PreOrder:

A B C D E F G H I J K



求结点p的前驱：

1.若 p->ltag = 1, 则 q = p->lchild;

2.若p->ltag = 0

*如果p为其父结点的左儿子，则：q=f;

*如果p为其父结点的右儿子，且其父结点没有左子树，则：q=f

*如果p为其父结点的右儿子，且其父结点有左子树,则q为其父结点的左子树前序序列的最后一个结点，即其右最下的结点。

求结点p的后继：

1.若 p->rtag = 1 则 s = p->rchild;

2.若p->rtag = 0则

*若p有左儿子，则s=p->lchild；

*否则，s = p->rchild

6.6树和森林的遍历

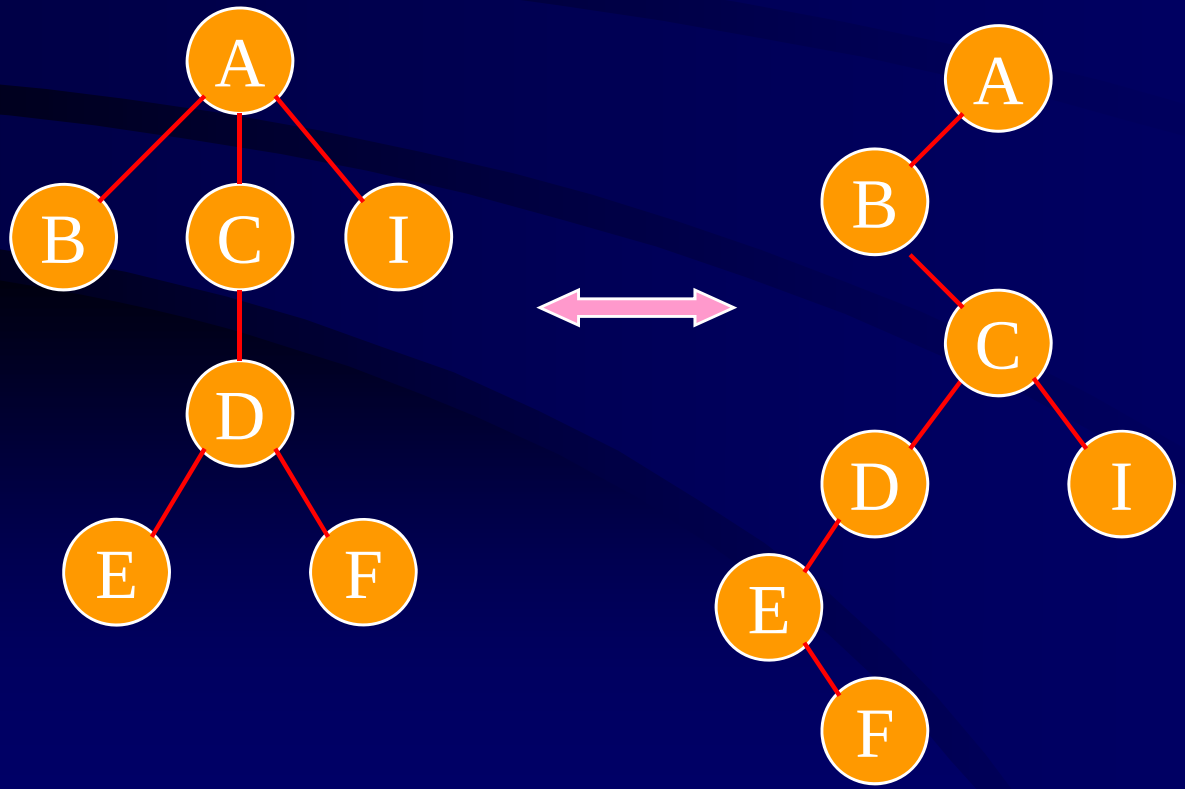
森林与树相互递归定义

- 森林：m棵互不相交的树的集合。
- 树：树是一个二元组Tree = (root，F),root是根，F是m(m>0)棵树的森林，$F = \{T_1, T_2, \ldots, T_m\}$其中$T_i = (r_i, F_i)$称为根的第i棵子树；当m != 0时，在树根和其子树森林之间存在下列关系：$RF = \{< root, r_i > \mid i = 1,2,\ldots,m, m>0\}$

6.6树和森林的遍历（续）

树与二叉树的转换

- 目的：利用二叉树运算来简化树和森林上的运算；
- 依据：树与二叉树具有相同的二叉链表存储结构；



6.6树和森林的遍历（续）

森林与二叉树的转换

根据森林与树的相互定义关系以及树与二叉树之间转换规则

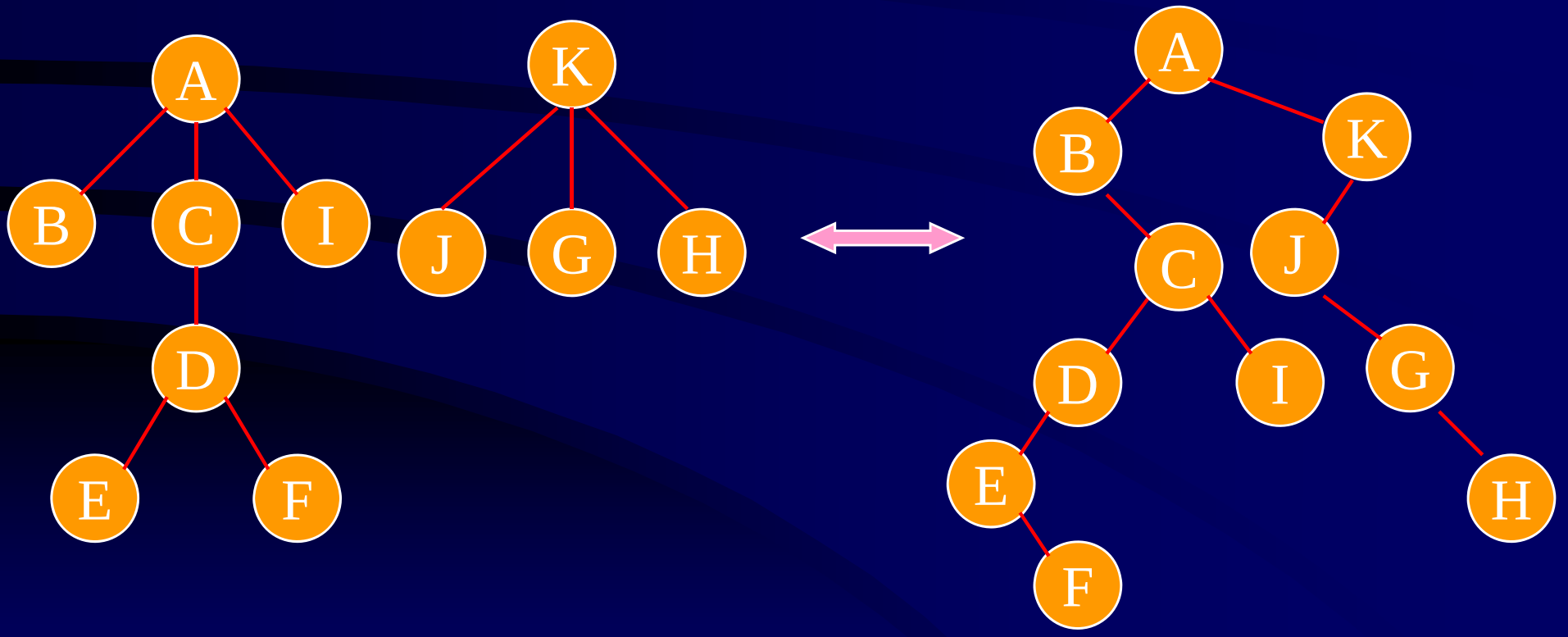
森林转换成二叉树：

如果F ={T_1,T_2,…,T_m}是森林，则可按照如下规则转换成一棵二叉树B = (root，LB，RB)。

（1）若F为空，则B为空树；

（2）若F非空，则B的根root为森林中第一棵树T_1的根;B的左子树LB是从T_1 中根结点的子树森林F_1 ={T_{11},T_{12},…,T_{1m1}}转换而成的二叉树；其右子树RB是从森林F’ = {T_2,…,T_m}转换而成的二叉树。

6.6树和森林的遍历（续）



6.6树和森林的遍历（续）

DS/SS

森林与二叉树的转换

根据森林与树的相互定义关系以及树与二叉树之间转换规则

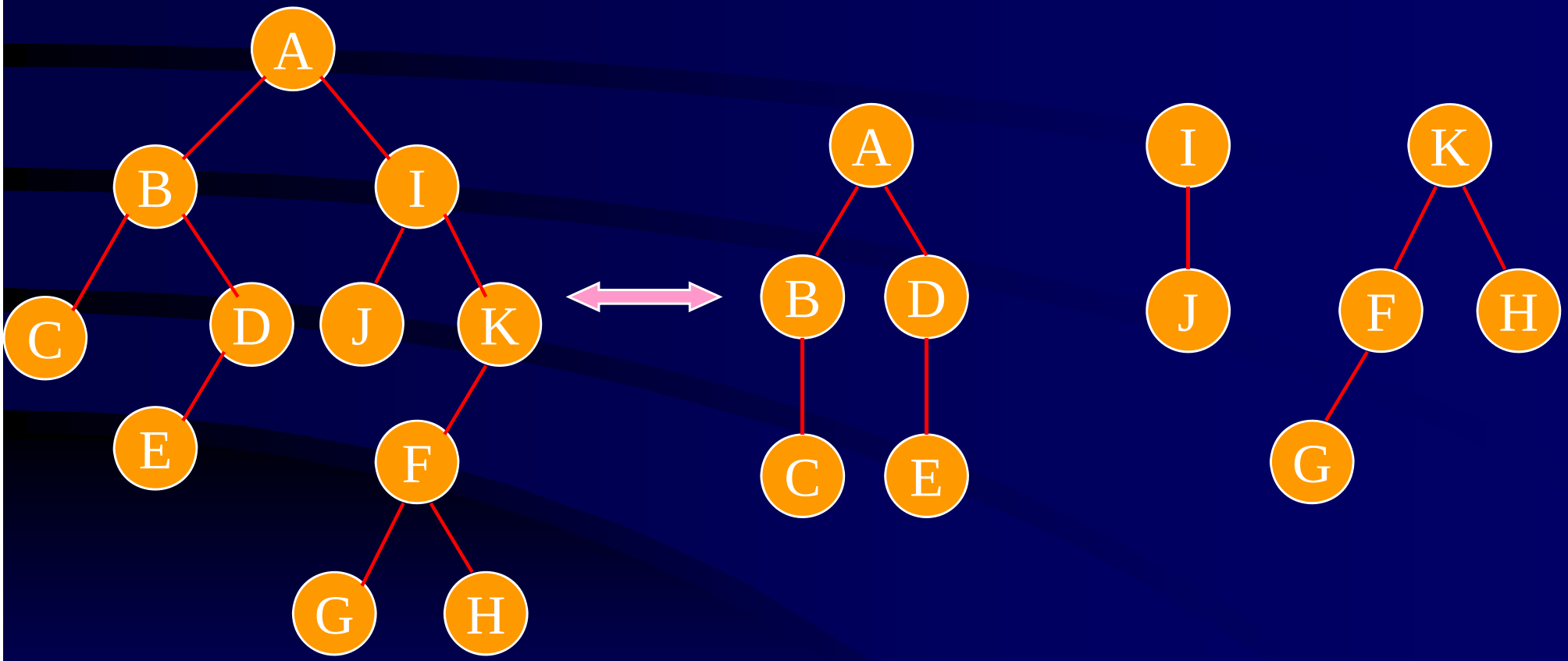
二叉树转换成森林：

二叉树B = (root，LB，RB)可按照如下规则转换成一棵森林$F = \{T_1, T_2, \ldots, T_m\}$。

（1）若B为空，则F为空树；

（2）若B非空，则森林中第一棵树T_1的根为B的根root; T_1中根结点的子树森林F_1是由B的左子树LB转换而成的；F中除T_1之外的森林$F' = \{T_2, \ldots, T_m\}$是由右子树RB转换而成的。

6.6树和森林的遍历（续）



6.6树和森林的遍历（续)

DS/SS

遍历树

策略：

*按照某种规则直接对树进行遍历；

*利用二叉树遍历来实现树上的遍历。

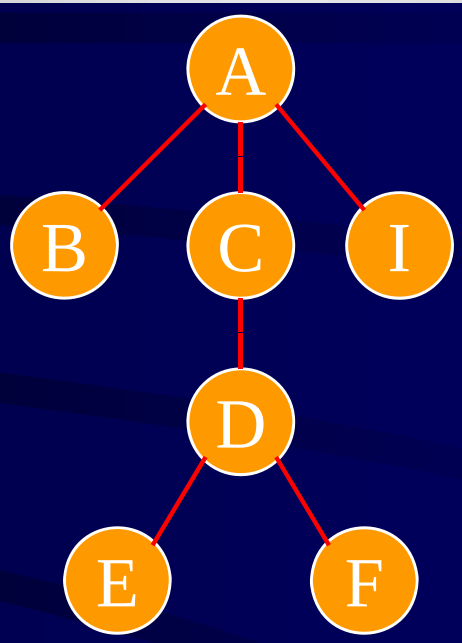
先根遍历：

先访问树的根结点，然后依次先根遍历根的每棵子树；
（对应等价的二叉树的先序遍历）

后根遍历：

先依次后根遍历根的每棵子树，然后访问树的根结点；
（对应等价的二叉树的中序遍历）

6.6树和森林的遍历（续)



- 先根遍历： A,B,C,D,E,F,I
- 后根遍历： B,E,F,D,C,I,A

6.6树和森林的遍历（续)

DS/SS

遍历森林

策略：

*按照某种规则直接对森林进行遍历；

*利用二叉树遍历来实现森林上的遍历。

先序遍历：

(1)访问森林中第一棵树的根结点；

(2)先序遍历第一棵树中根结点的子树森林；

(3)先序遍历除去第一棵树之后剩余的树构成的森林

（对应等价的二叉树的先序遍历）

中序遍历：

(1)中序遍历第一棵树中根结点的子树森林；

(2)访问森林中第一棵树的根结点；

(3)中序遍历除去第一棵树之后剩余的树构成的森林

（对应等价的二叉树的中序遍历）