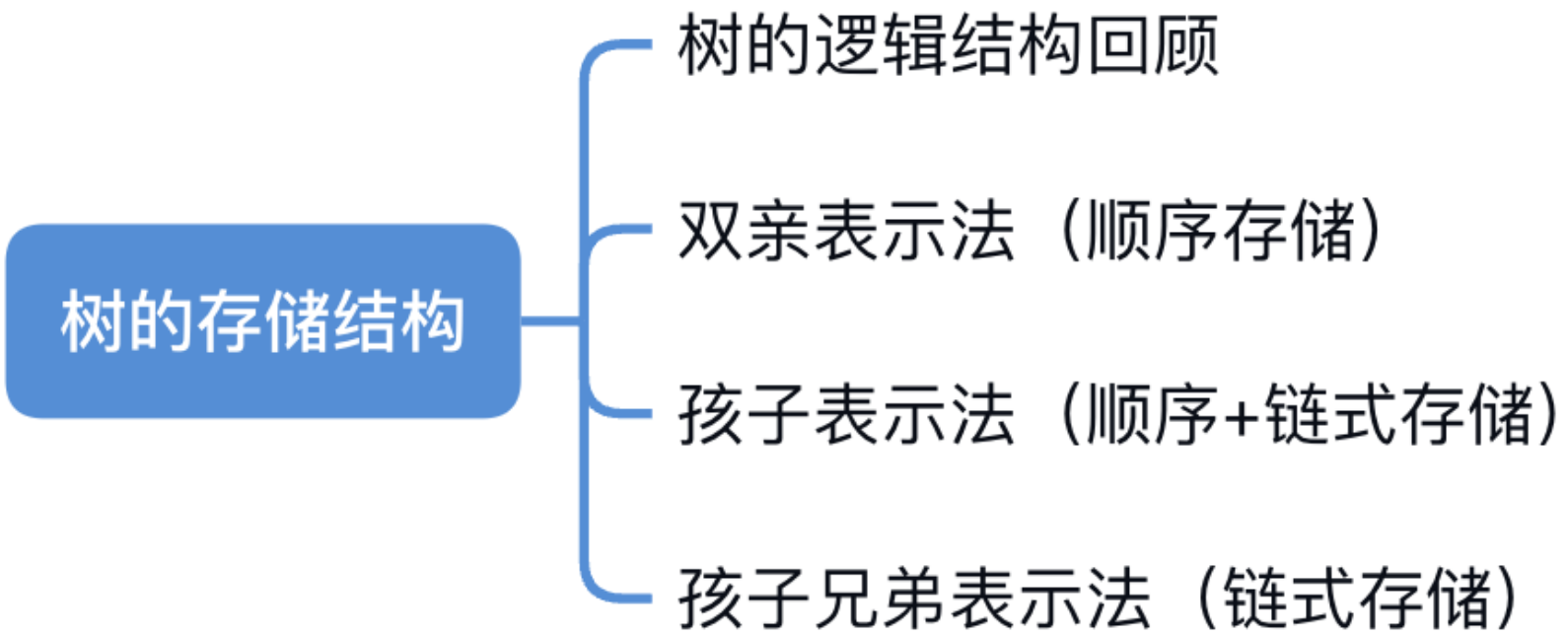
本节内容

树

存储结构

知识总览

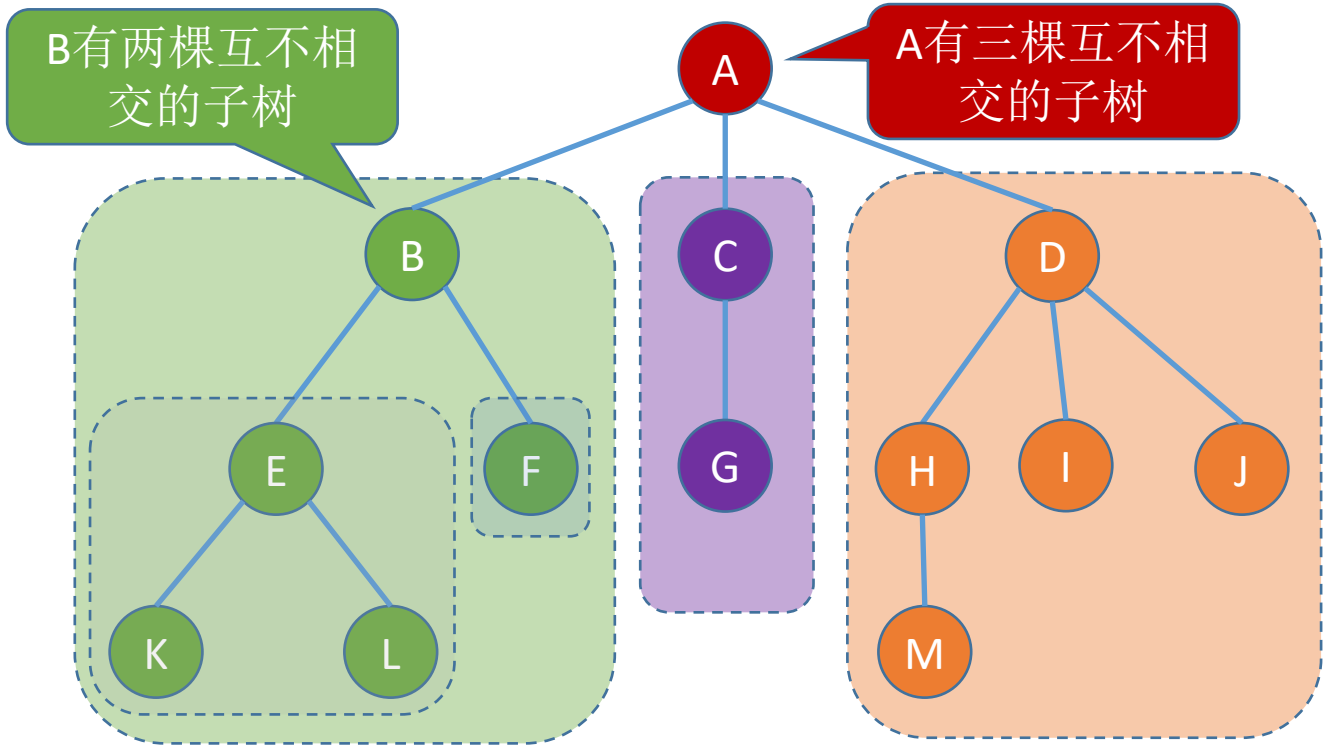


树的逻辑结构

树是n（$n \geq 0$）个结点的有限集合，$n = 0$时，称为空树，这是一种特殊情况。在任意一棵非空树中应满足：

1）有且仅有一个特定的称为根的结点。

2）当$n > 1$时，其余结点可分为m（$m > 0$）个互不相交的有限集合$T_1, T_2, \ldots, T_m$，其中每个集合本身又是一棵树，并且称为根结点的子树。

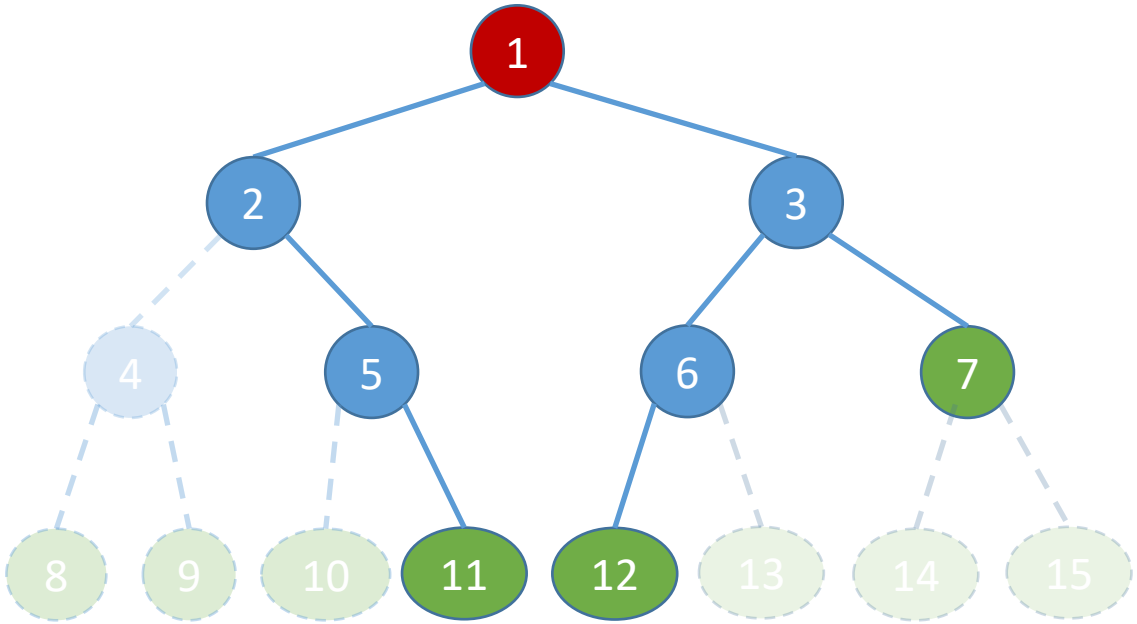


树是一种递归定义的数据结构

二叉树：一个分支结点最多只能有两棵子树

树：一个分支结点可以有多棵子树

回顾：二叉树的顺序存储



二叉树：一个分支结点最多只能有两棵子树

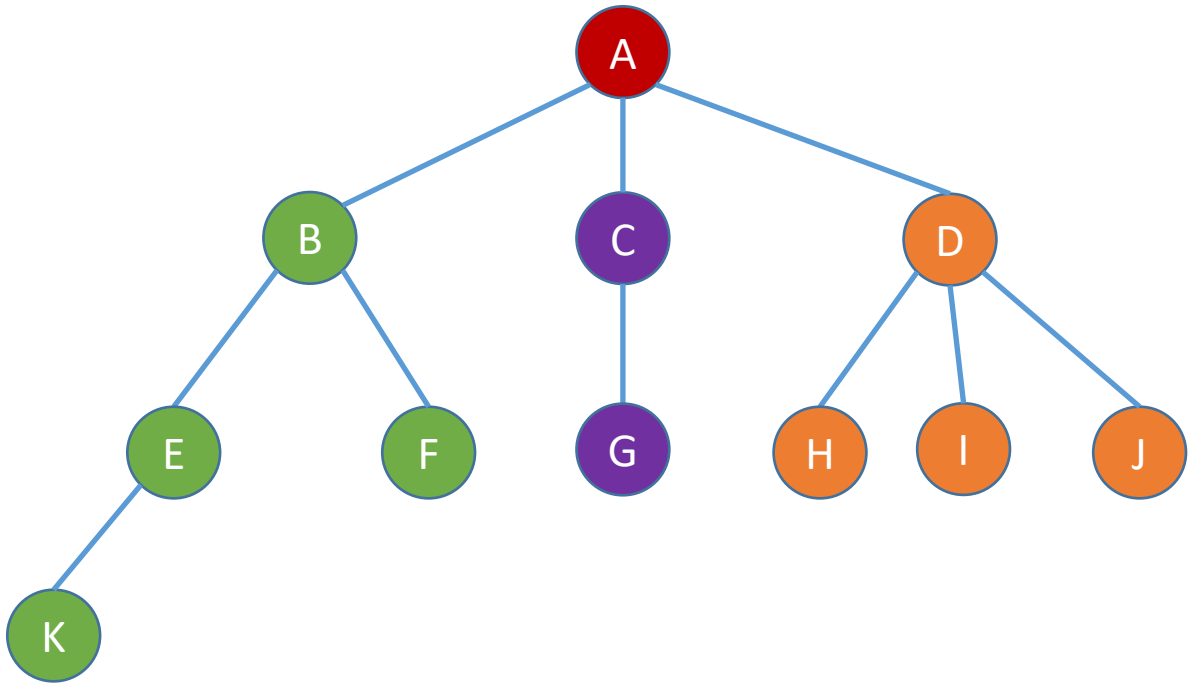
二叉树的顺序存储：
按照完全二叉树中的结点顺序，将各结点存储到数组的对应位置。数组下标反映结点之间的逻辑关系

若根结点从数组下标1开始存放，则：

- i 的左孩子　——2i
- i 的右孩子　——2i+1
- i 的父节点　——$\lfloor i/2 \rfloor$

t[0]　t[1]　t[2]　……

如何实现树的顺序存储？

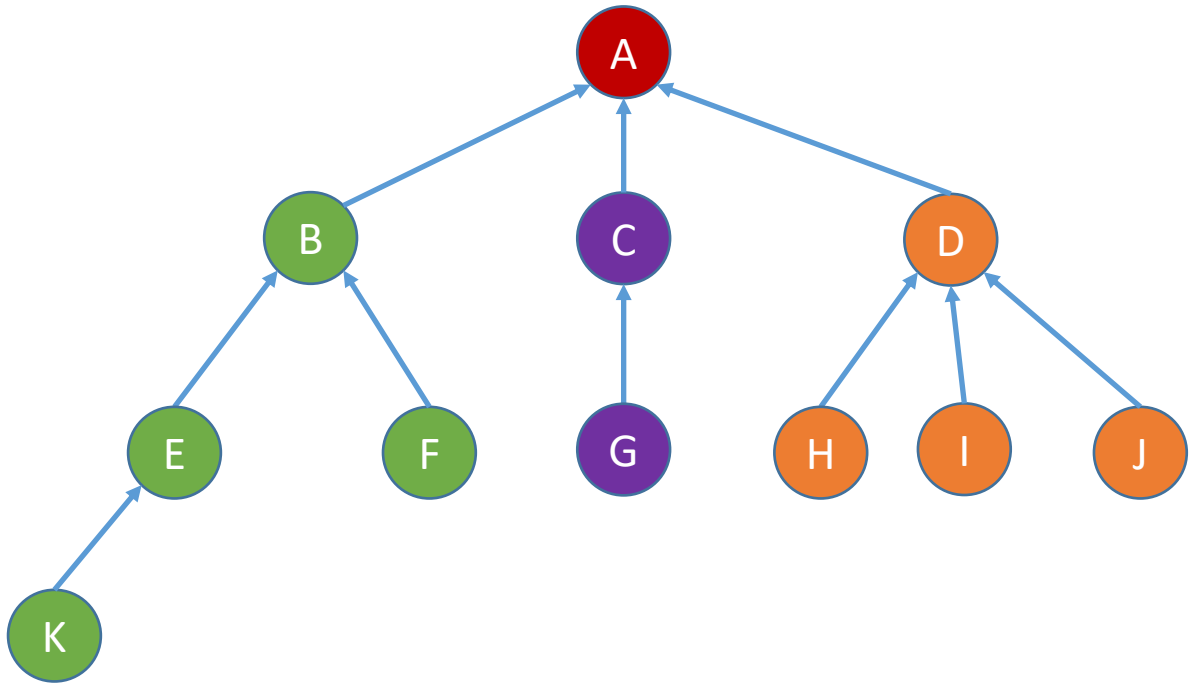


树：一个分支结点可以有多棵子树

只依靠数组下标，无法反映结点之间的逻辑关系

^	A	B	C	D	E	F	G	H	I	J	K	^	^	^	^	^
t[0]	t[1]	t[2]	……													

如何实现树的顺序存储？



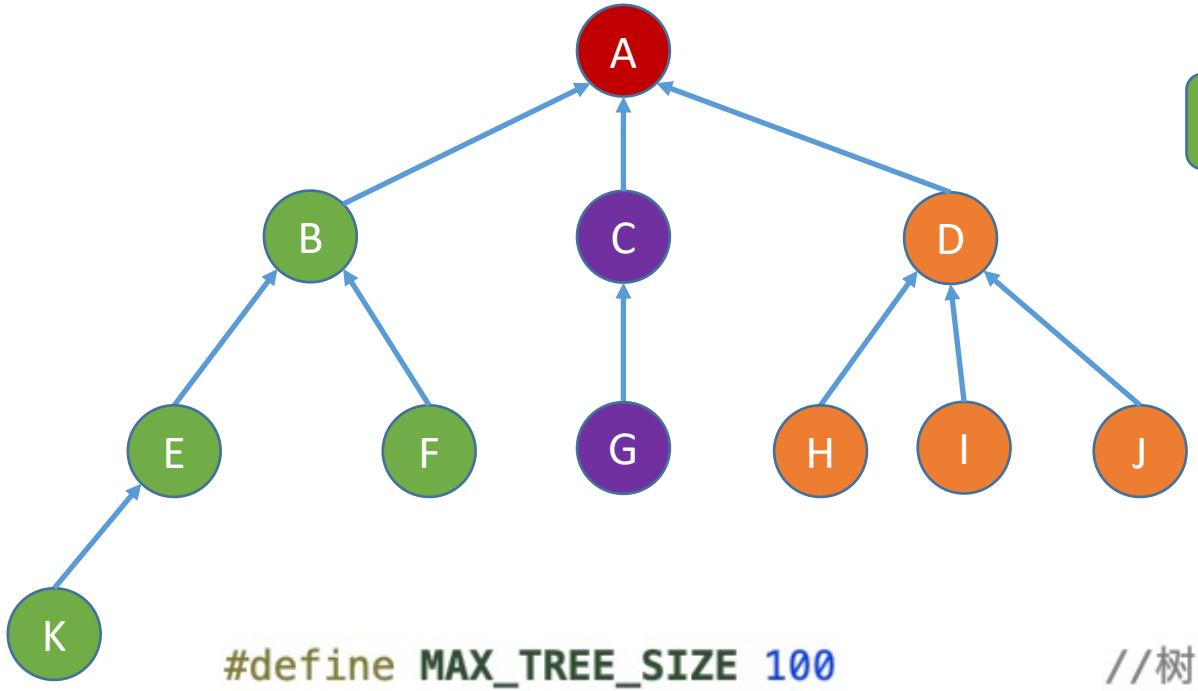
	data	parent
0	A	-1
1	B	0
2	C	0
3	D	0
4	E	1
5	F	1
6	G	2
7	H	3
8	I	3
9	J	3
10	K	4
11		
12		
13		
⋮	⋮	⋮

根节点的双亲指针 = -1

非根节点的双亲指针 = 父节点在数组中的下标

思路：用数组顺序存储各个结点。每个结点中保存数据元素、指向双亲结点（父节点）的“指针”

树的存储1：双亲表示法



顺序存储

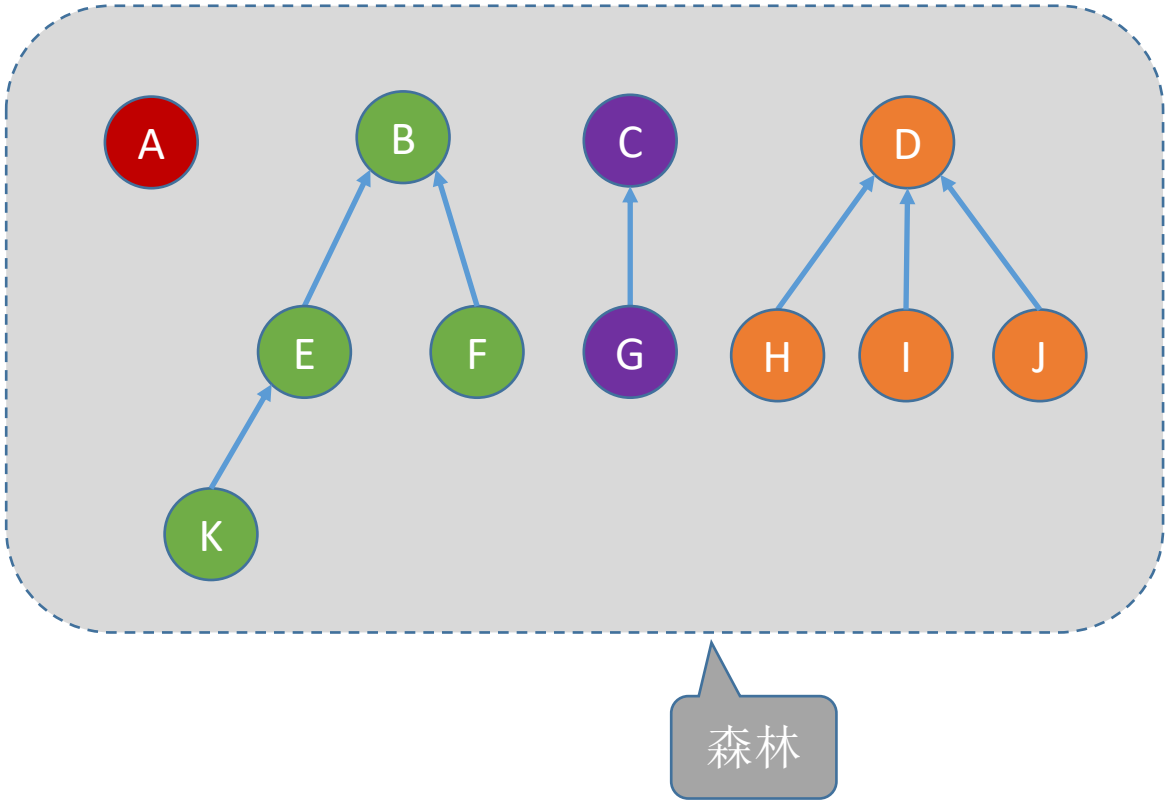
	data	parent
0	A	-1
1	B	0
2	C	0
3	D	0
4	E	1
5	F	1
6	G	2
7	H	3
8	I	3
9	J	3
10	K	4
11		
12		
13		
⋮	⋮	⋮

```
#define MAX_TREE_SIZE 100              //树中最多结点数
typedef struct{                        //树的结点定义
    ElemType data;                     //数据元素
    int parent;                        //双亲位置域
}PTNode;
typedef struct{                        //树的类型定义
    PTNode nodes[MAX_TREE_SIZE];       //双亲表示
    int n;                             //结点数
}PTree;
```

结点总数n=11

拓展：双亲表示法存储“森林”

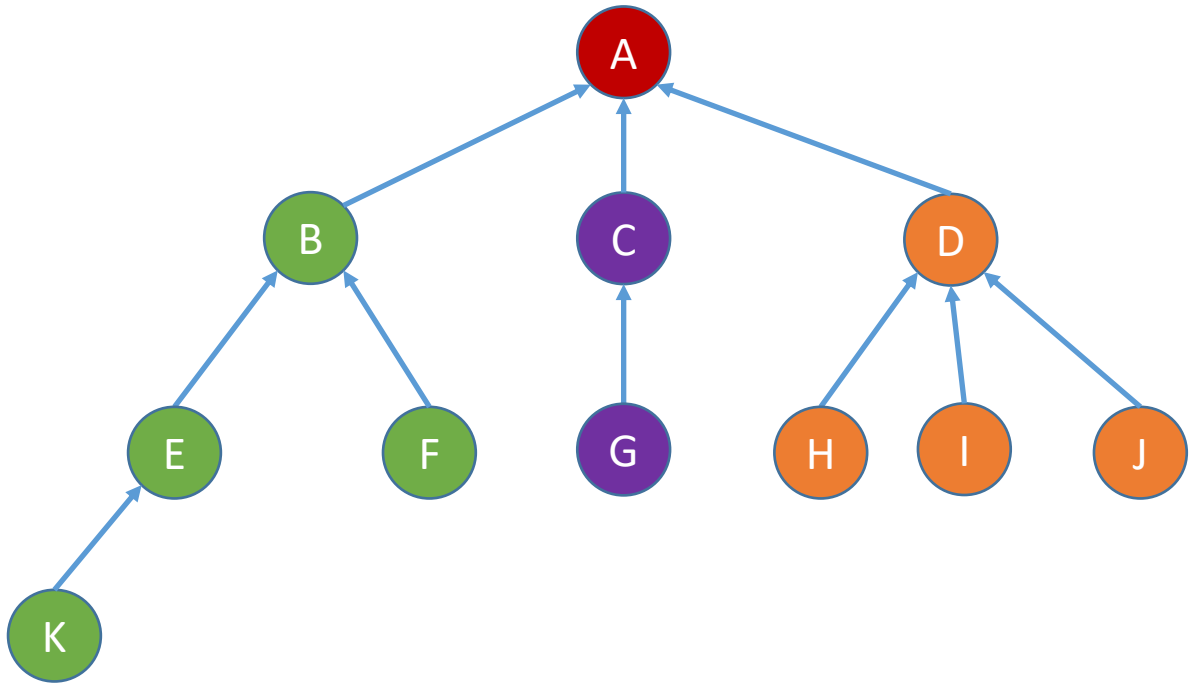
森林。森林是m（$m \geq 0$）棵互不相交的树的集合



	data	parent
0	A	-1
1	B	-1
2	C	-1
3	D	-1
4	E	1
5	F	1
6	G	2
7	H	3
8	I	3
9	J	3
10	K	4
11		
12		
13		
⋮	⋮	

每棵树的根节点双亲指针 = -1

双亲表示法的优缺点



	data	parent
0	A	-1
1	B	0
2	C	0
3	D	0
p → 4	E	1
5	F	1
6	G	2
7	H	3
8	I	3
9	J	3
10	K	4
11		
12		
13		
	⋮	⋮

4号结点的父节点是1号结点

10号结点是4号结点的孩子

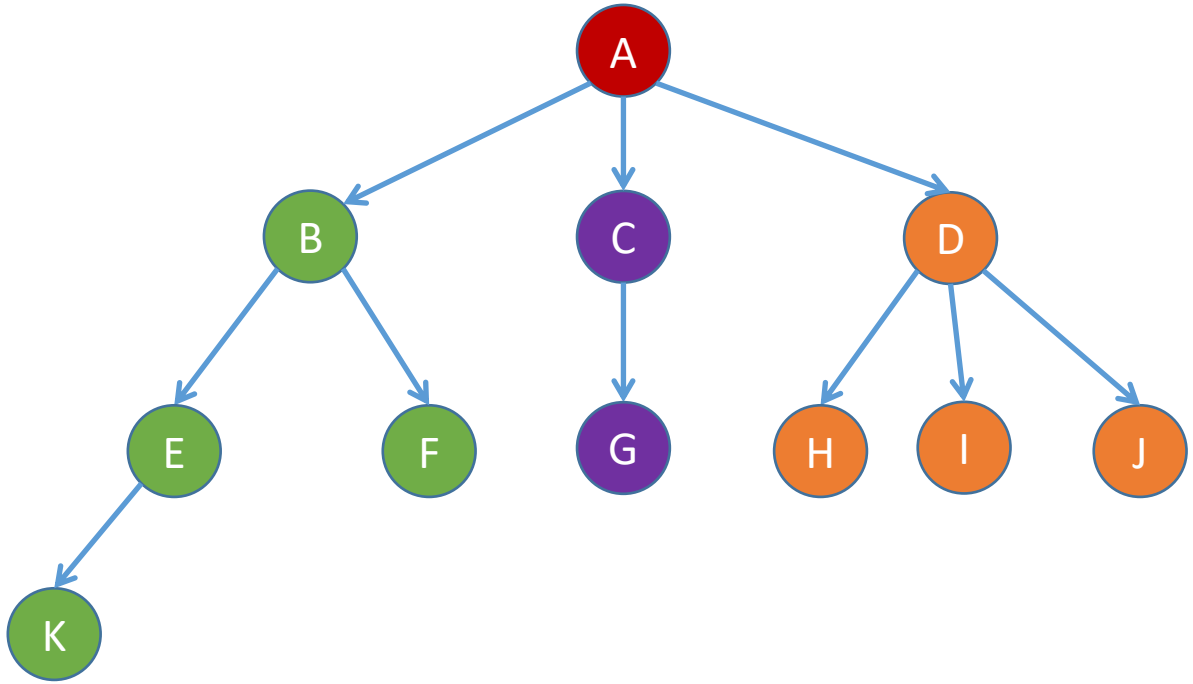
双亲表示法

优点：找双亲（父节点）很方便

缺点：找孩子不方便，只能从头到尾遍历整个数组

适用于“找父亲”多，“找孩子”少 的应用场景。如：并查集

树的存储2：孩子表示法



孩子表示法：用数组顺序存储各个结点。每个结点中保存数据元素、孩子链表头指针

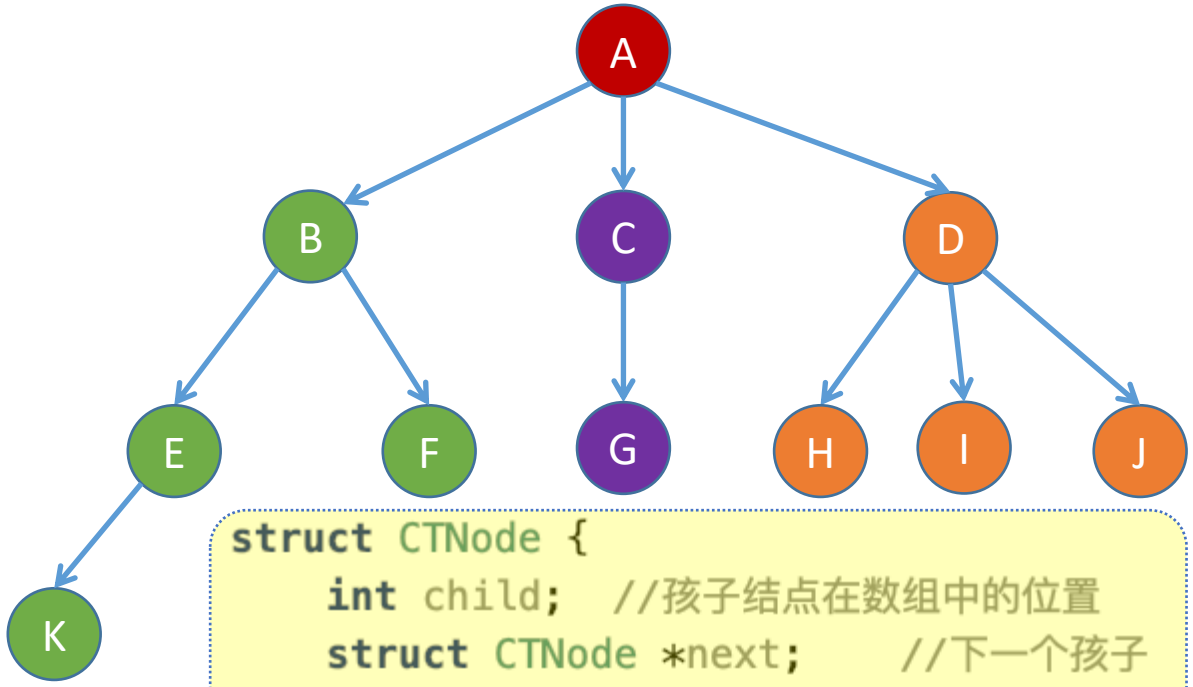
指向第一个孩子编号

用一个链表记录一个结点的孩子编号

	data	*firstChild
0	A	1 → 2 → 3 ^
1	B	4 → 5 ^
2	C	6 ^
3	D	7 → 8 → 9 ^
4	E	10 ^
5	F	^
6	G	^
7	H	^
8	I	^
9	J	^
10	K	^
⋮	⋮	⋮

树的存储2：孩子表示法

顺序存储+链式存储结合




```
struct CTNode {
    int child;  //孩子结点在数组中的位置
    struct CTNode *next;    //下一个孩子
};
typedef struct {
    ElemType data;
    struct CTNode *firstChild;  //第一个孩子
} CTBox;
typedef struct {
    CTBox nodes[MAX_TREE_SIZE];
    int n, r;   //结点数和根的位置
} CTree;
```

结点总数n=11，根的位置r=0

	data	*firstChild
0	A	1
1	B	4
2	C	6 ^
3	D	7
4	E	10 ^
5	F	^
6	G	^
7	H	^
8	I	^
9	J	^
10	K	^
	⋮	⋮

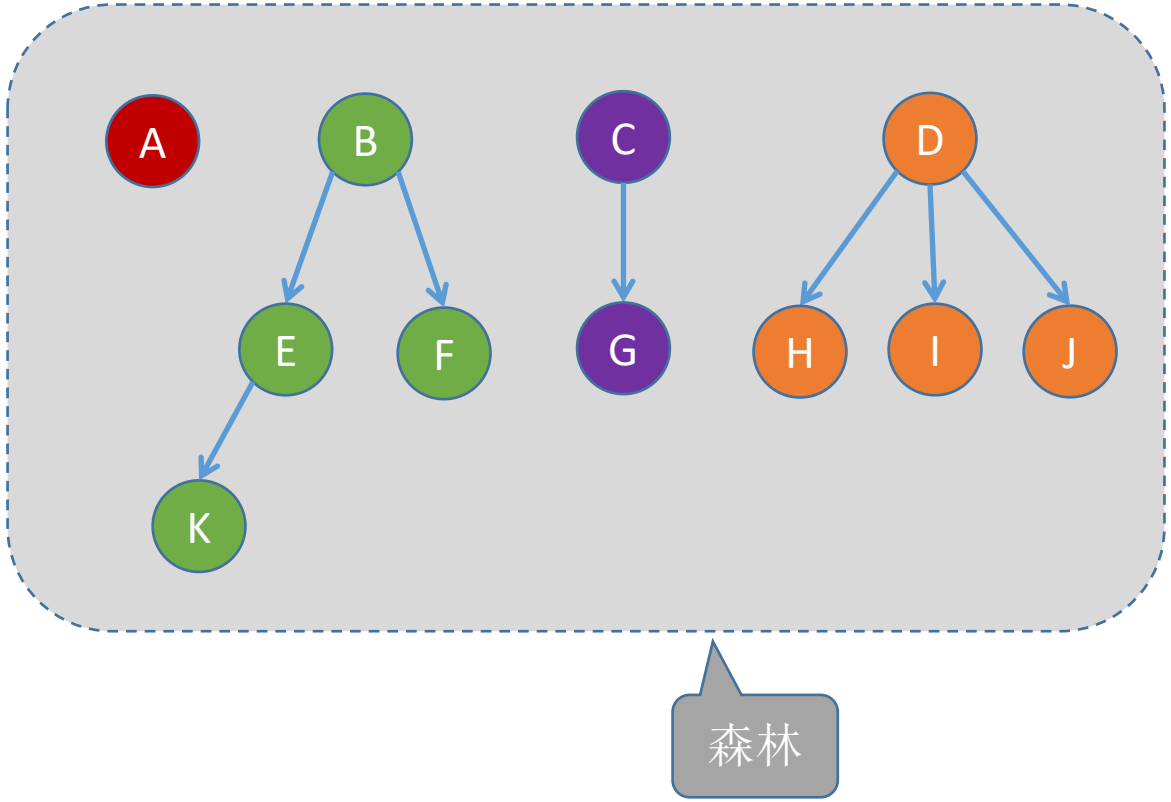
1 → 2 → 3 ^

4 → 5 ^

7 → 8 → 9 ^

拓展：孩子表示法存储“森林”

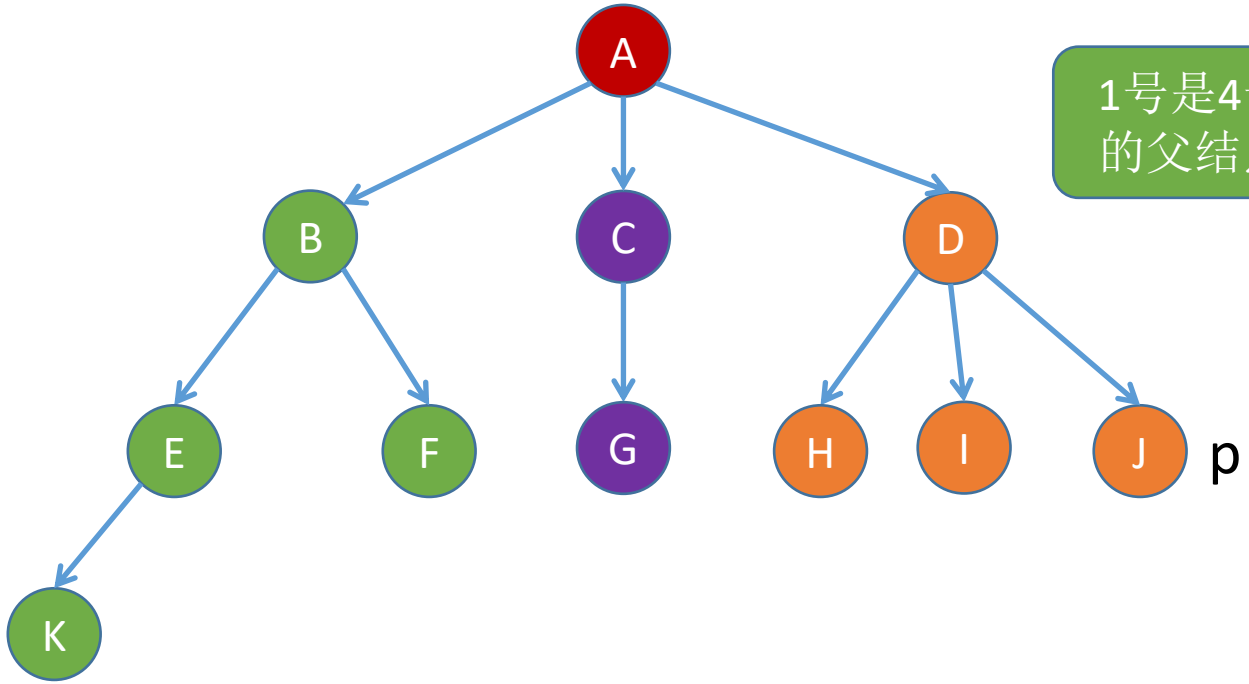
森林。森林是m（$m \geq 0$）棵互不相交的树的集合

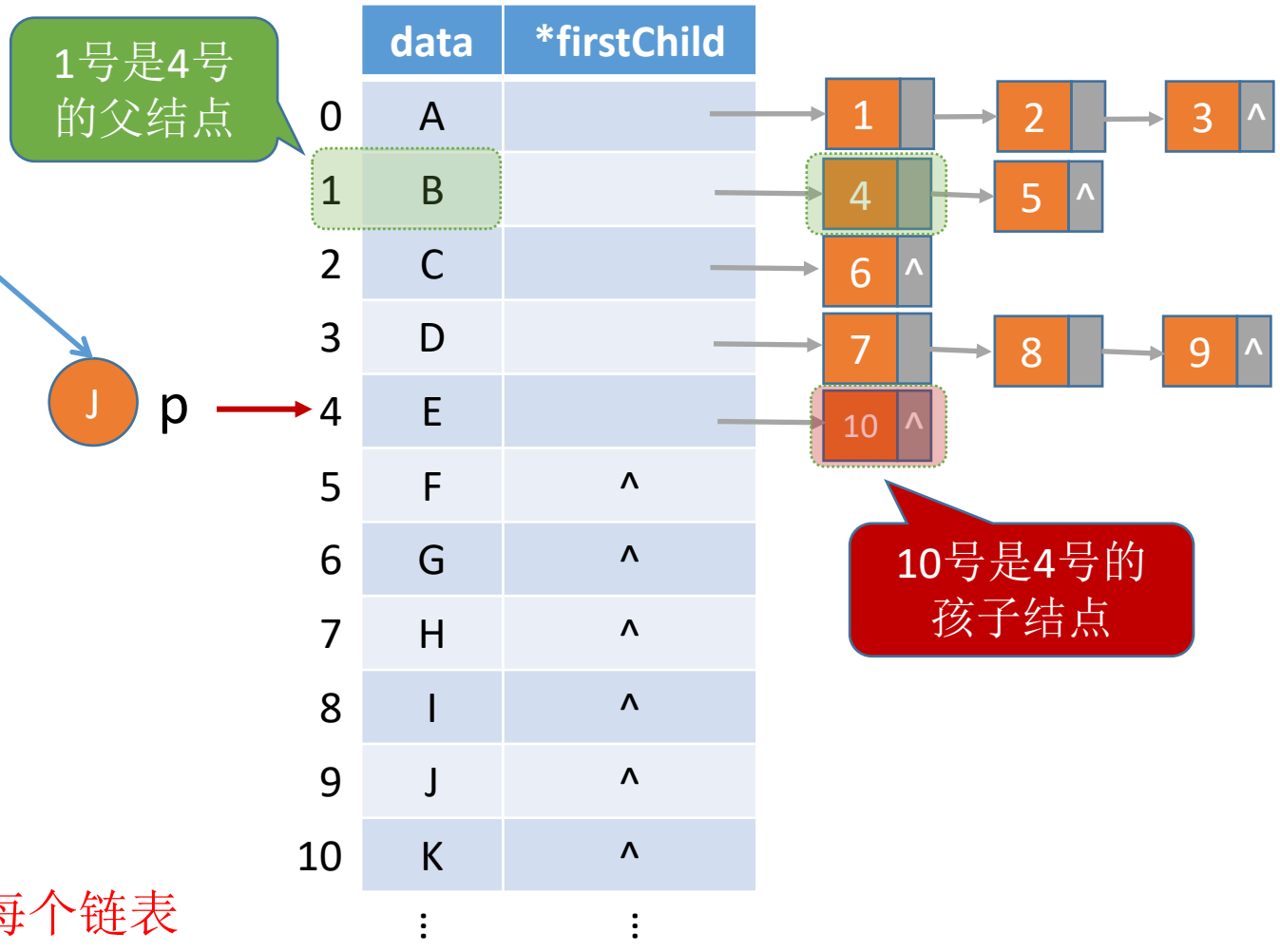


	data	*firstChild
0	A	^
1	B	
2	C	
3	D	
4	E	
5	F	^
6	G	^
7	H	^
8	I	^
9	J	^
10	K	^
⋮	⋮	⋮

注：用孩子表示法存储森林，需要记录多个根的位置

孩子表示法的优缺点





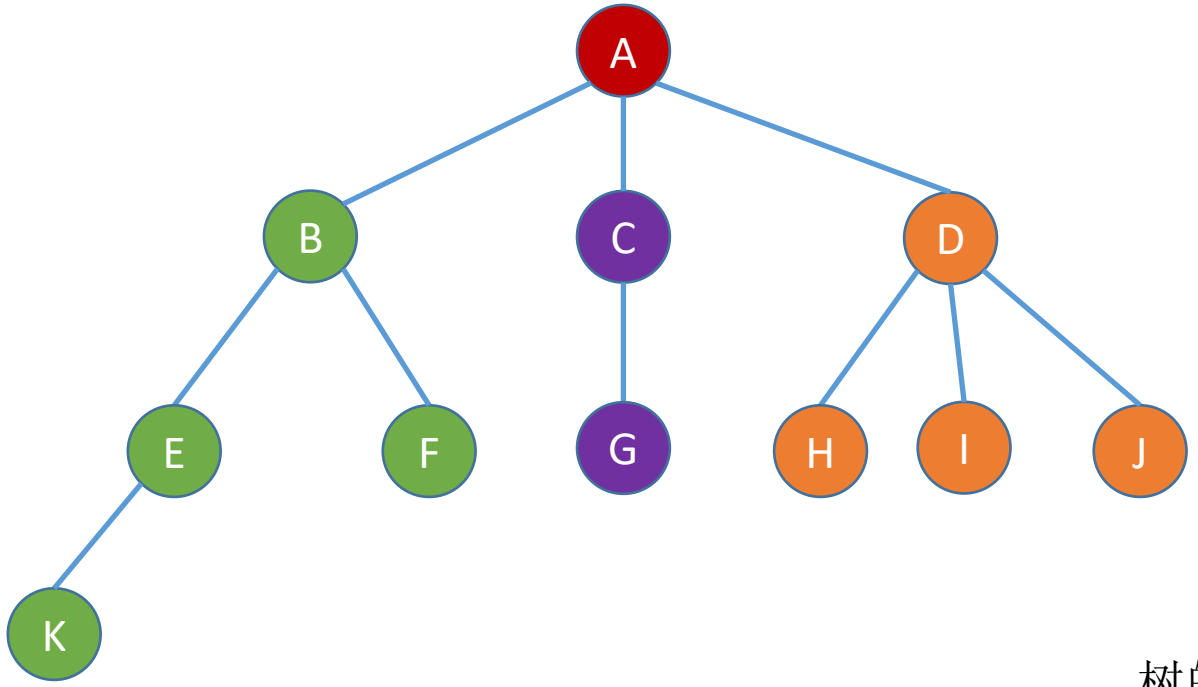
孩子表示法

优点：找孩子很方便

缺点：找双亲（父节点）不方便，只能遍历每个链表

适用于“找孩子”多，“找父亲”少 的应用场景。如：服务流程树

树的存储3：孩子兄弟表示法




```
//二叉树的结点（链式存储）
typedef struct BiTNode{
    ElemType data;
    struct BiTNode *lchild,*rchild;
}BiTNode,*BiTree;
```

左子树　右子树

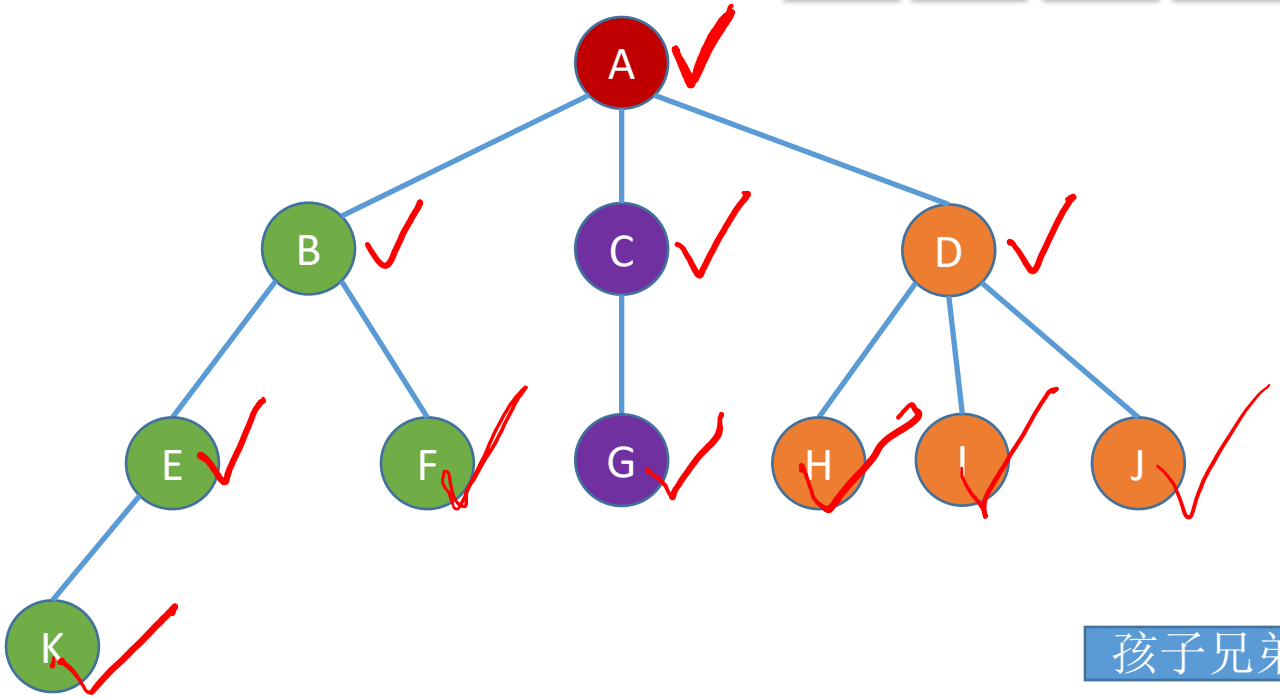
树的孩子兄弟表示法，与二叉树类似，采用二叉链表实现。
每个结点内保存数据元素和两个指针，但两个指针的含义与二叉树结点不同

```
//树的存储——孩子兄弟表示法
typedef struct CSNode{
    ElemType data;
    struct CSNode *firstchild,*nextsibling;
}CSNode,*CSTree;
```

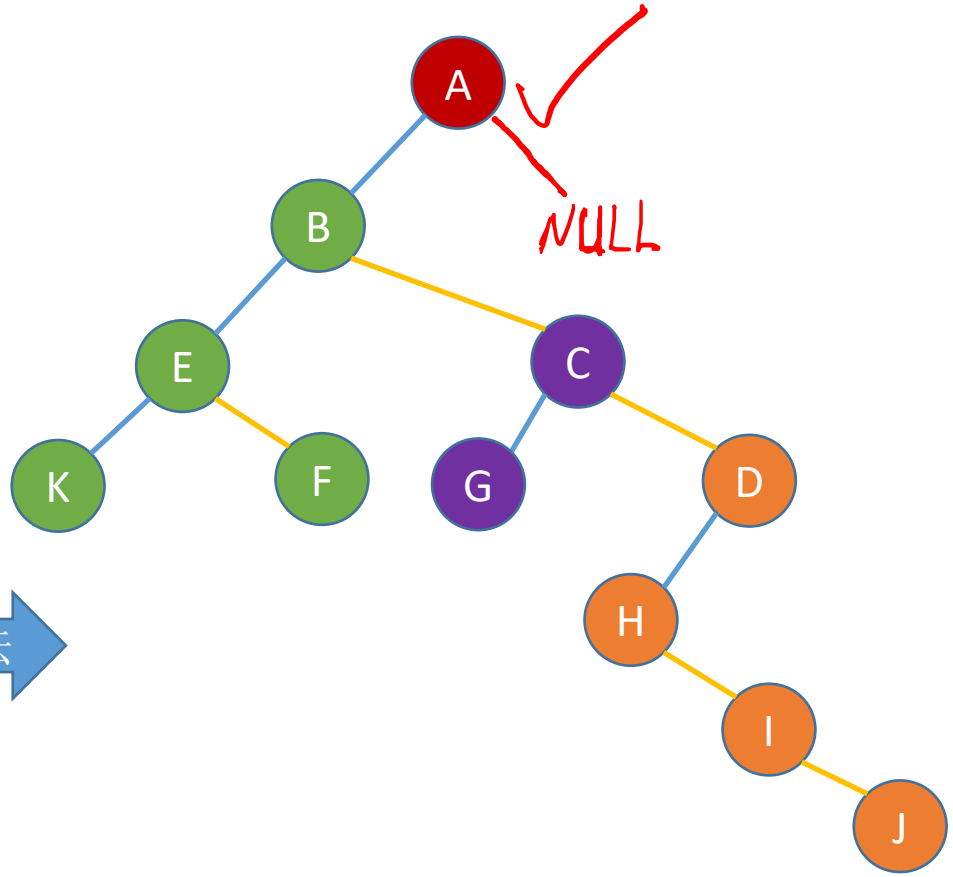
```
//数据域
//第一个孩子和右兄弟指针
```

指向第一个孩子　指向右边一个兄弟

树的存储3：孩子兄弟表示法



孩子兄弟表示法



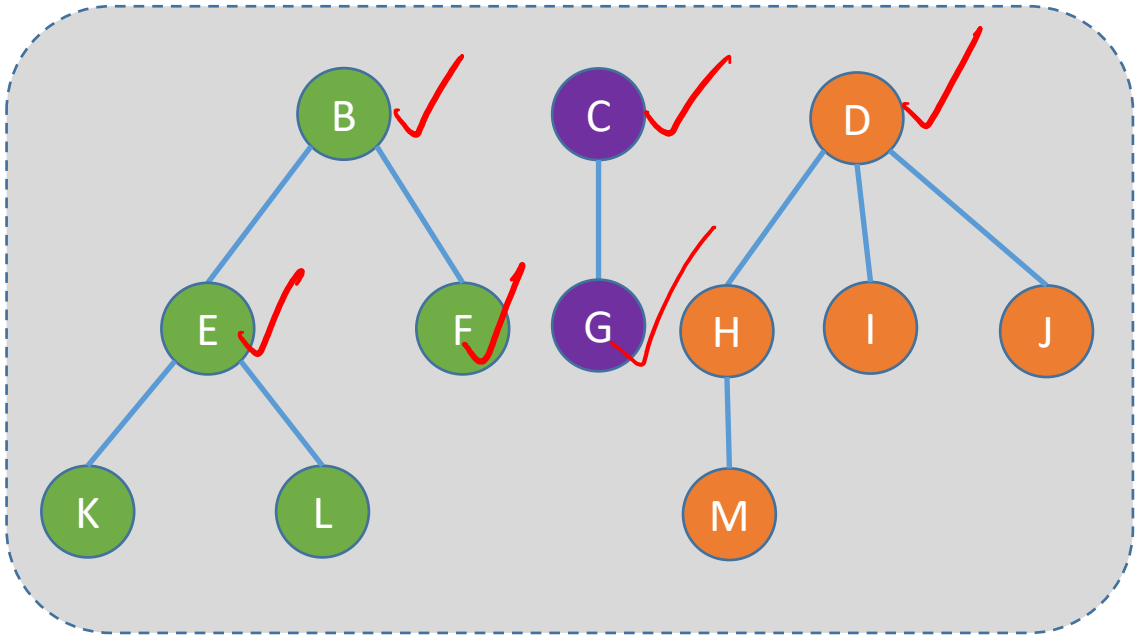

```
//树的存储——孩子兄弟表示法
typedef struct CSNode{
    ElemType data;                          //数据域
    struct CSNode *firstchild,*nextsibling; //第一个孩子和右兄弟指针
}CSNode,*CSTree;
```

指向第一个孩子

指向右边一个兄弟

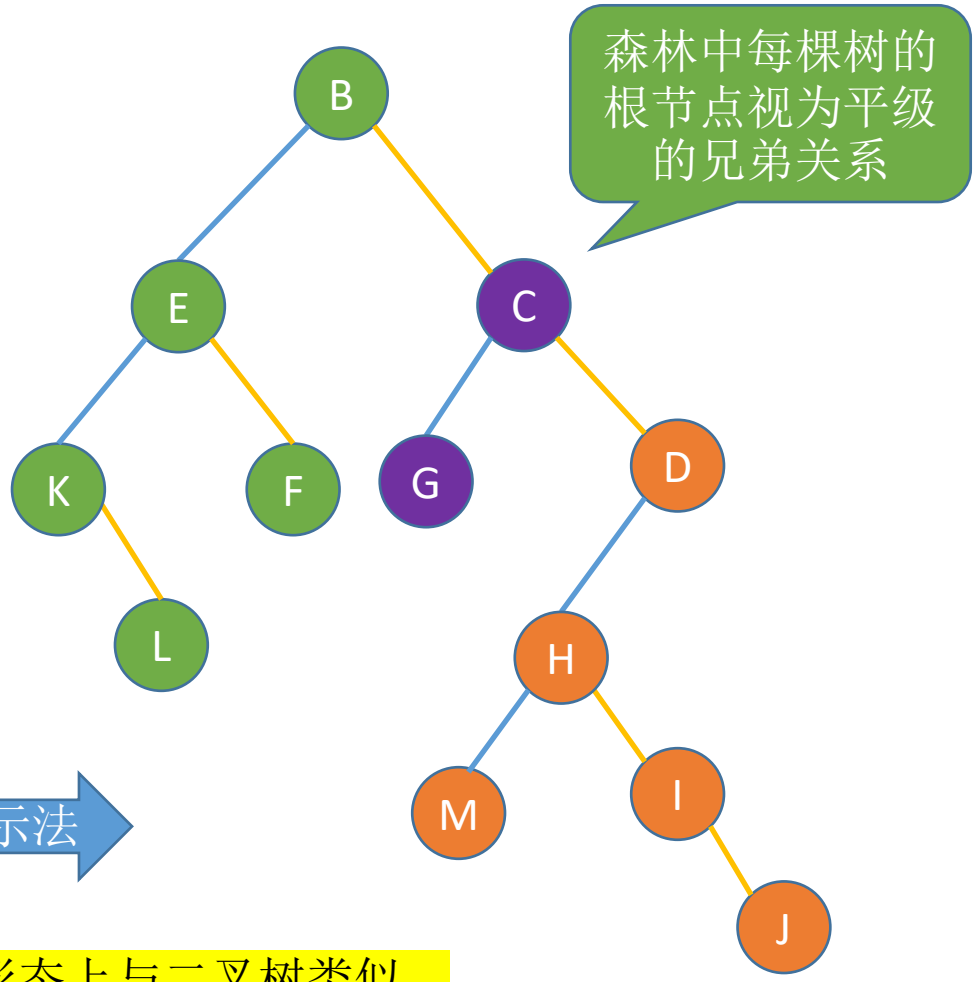
拓展：孩子兄弟表示法存储“森林”

森林。森林是m（$m\geq 0$）棵互不相交的树的集合



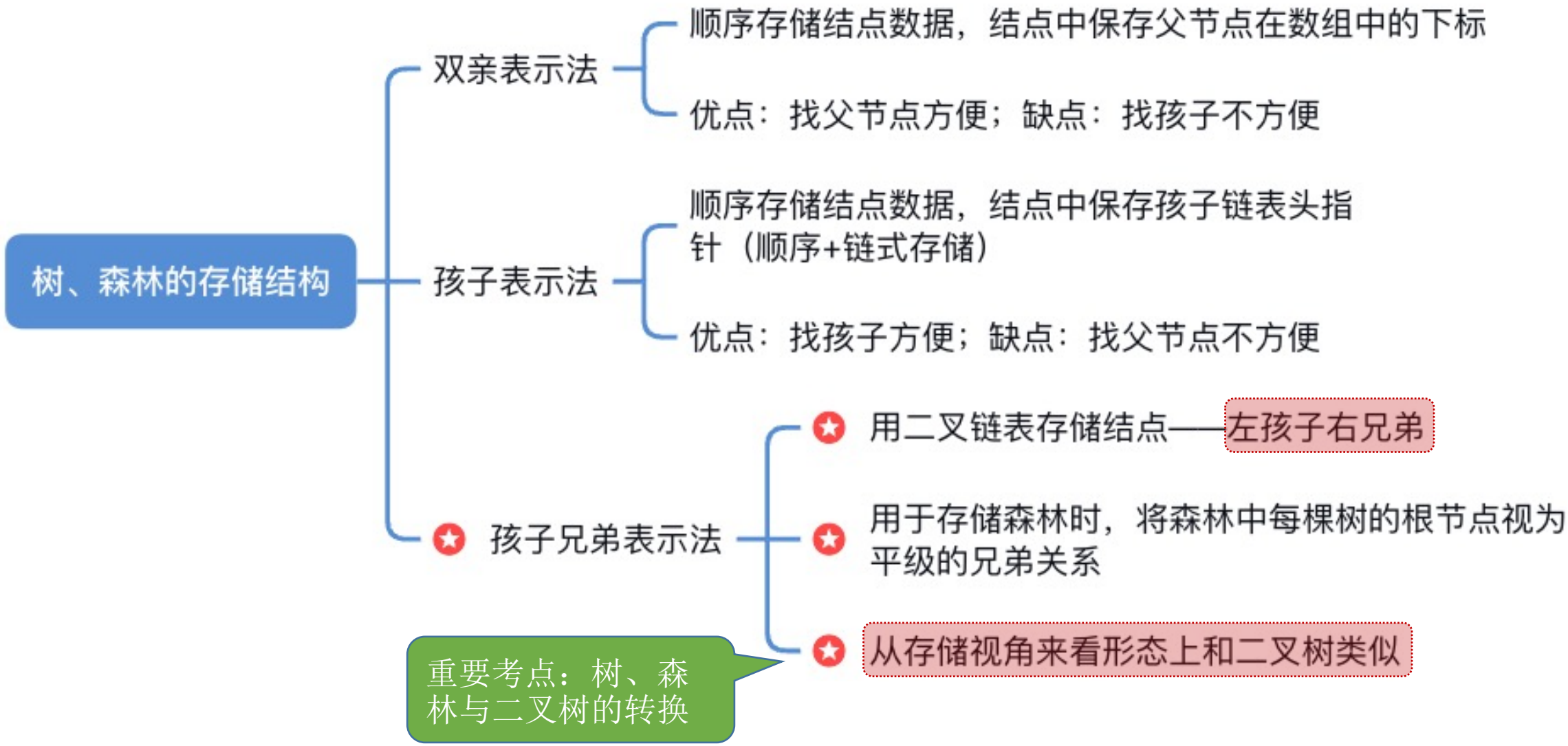
森林

孩子兄弟表示法



当使用“孩子兄弟表示法”存储树或森林时，从存储视角来看形态上与二叉树类似。

知识回顾与重要考点



欢迎大家对本节视频进行评价~

学员评分：4.1_2 串的存储结构

扫一扫二维码打开或分享给好友

– 腾讯文档 –

可多人实时在线编辑，权限安全可控

公众号：王道在线

b站：王道计算机教育

抖音：王道计算机考研