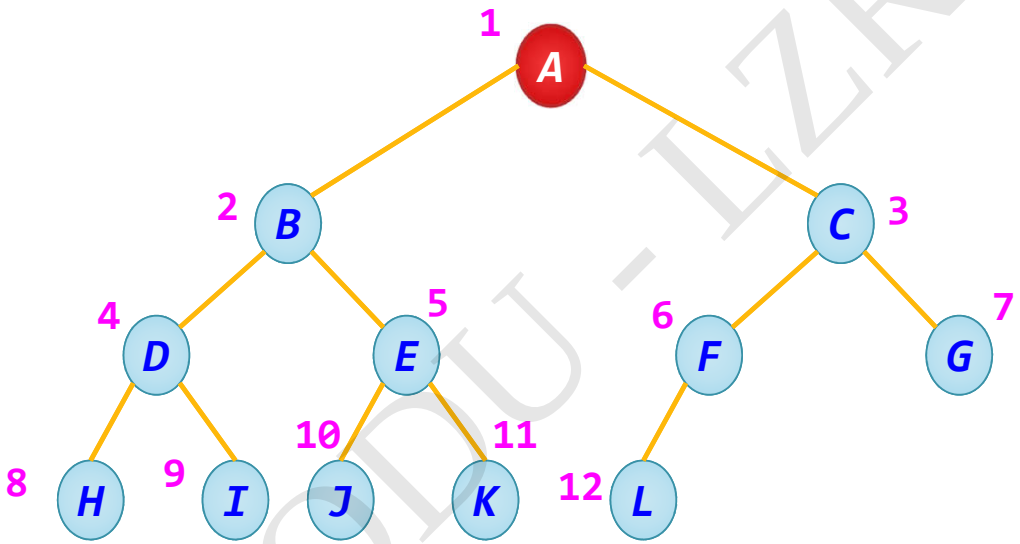
6.2.3 二叉树的存储结构

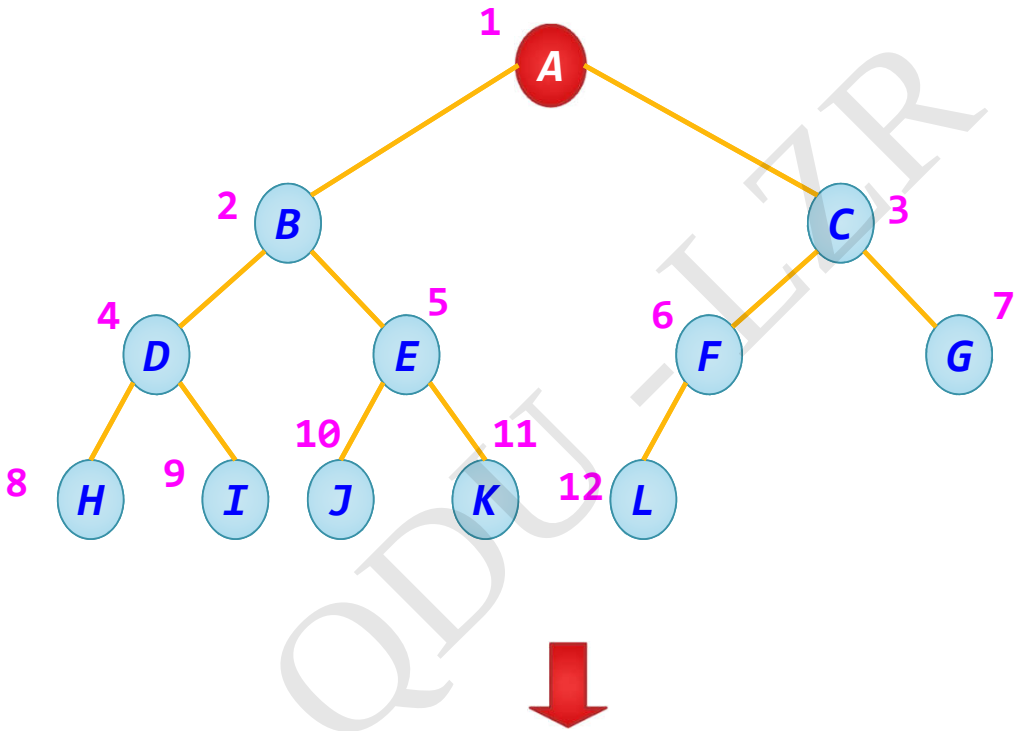
1. 顺序存储结构

- 顺序存储一棵二叉树时，就是用一组连续的存储单元存放二叉树中的结点。
- 由二叉树的性质5可知，对于完全二叉树（或满二叉树），树中结点层序编号可以唯一地反映出结点之间的逻辑关系，所以可以用一维数组按从上到下、从左到右的顺序存储树中所有结点值，通过数组元素的下标关系反映完全二叉树或满二叉树中结点之间的逻辑关系。

一棵完全二叉树的顺序存储结构

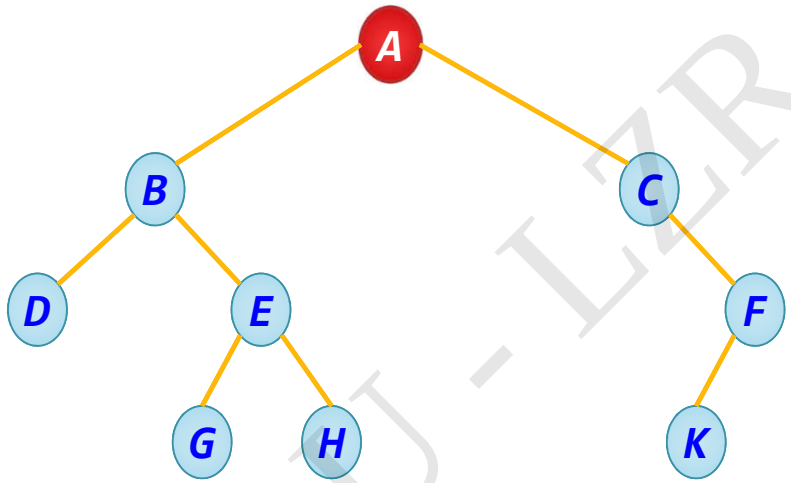


一棵完全二叉树的顺序存储结构



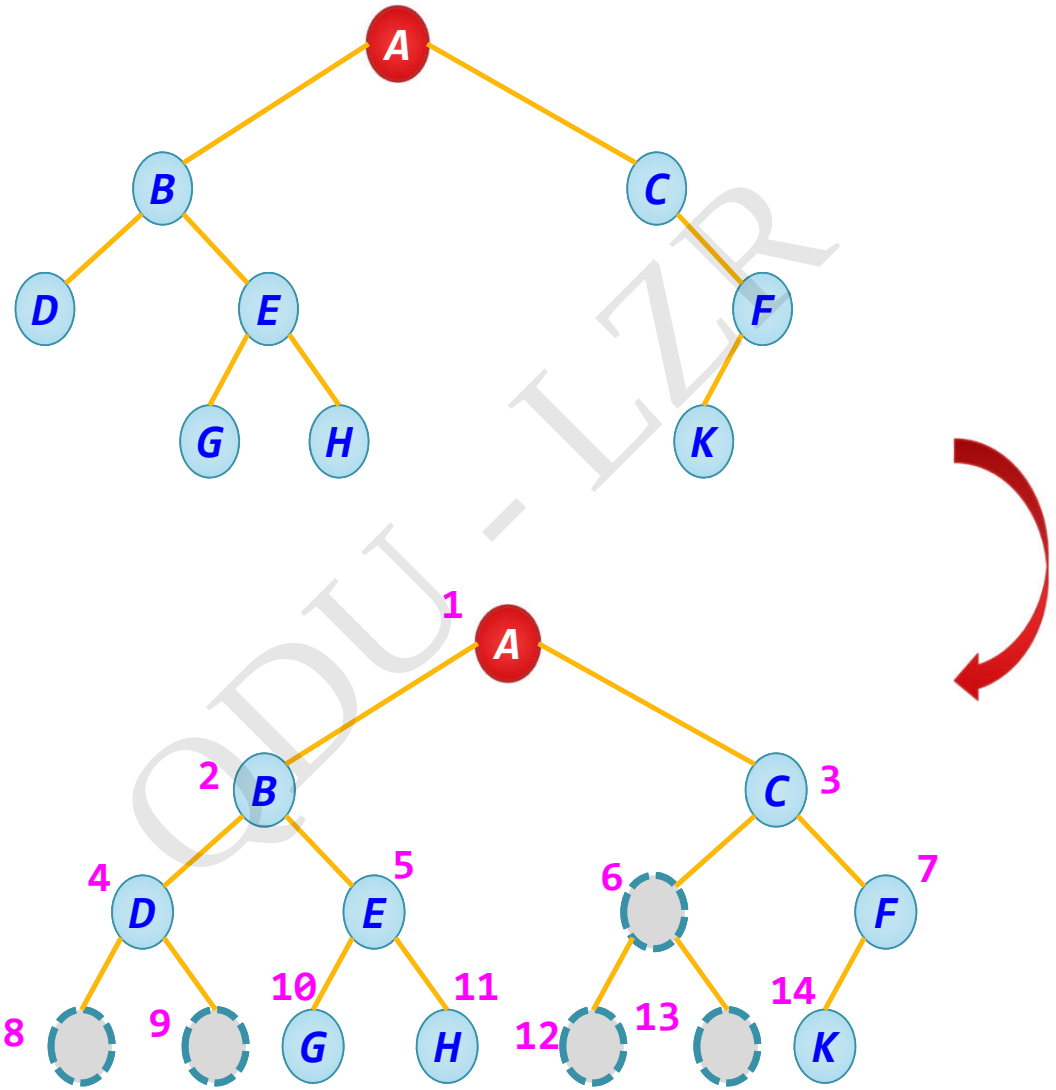
位置	1	2	3	4	5	6	7	8	9	10	11	12	13	14	…
data	A	B	C	D	E	F	G	H	I	J	K	L	#	#	#

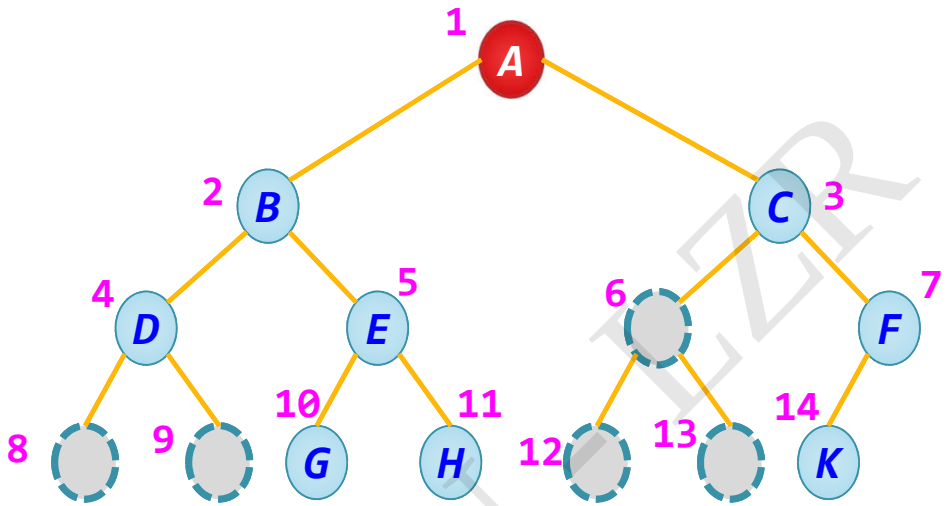
一般的二叉树 的顺序存储结构设计：



1	2	3	4	5	6	7	8	9	10	11	12	13	14	…

- ✓ 增添空结点补齐为一棵完全二叉树
- ✓ 并对所有结点进行编号


A
B
C
D
E
F
G
H
K
1
2
3
4
5
6
7
8
9
10
11
12
13
14



仅保留实际存在的结点值，其他为空

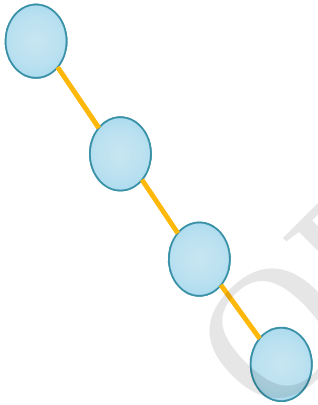
位置	1	2	3	4	5	6	7	8	9	10	11	12	13	14	…
data	A	B	C	D	E	#	F	#	#	G	H	#	#	K	#

```
// ------ 二叉树的顺序存储表示 --------

#define MAX_TREE_SIZE 100 // 二叉树的最大结点数

// 0号单元存储根结点
typedef TElemType SqBiTree[MAX_TREE_SIZE];
SqBiTree bt;
```

- 其中，TElemType为二叉树中结点的数据值类型；
- MAX_TREE_SIZE为顺序表的最大长度。
- 将下标为0的单元存放根结点的下标，值为‘#’的结点为空结点。

- 完全二叉树或满二叉树采用顺序存储结构比较合适，既能够最大可能地节省存储空间，又可以利用数组元素的下标确定结点在二叉树中的位置以及结点之间的关系。
- 对于一般二叉树，如果它接近于完全二叉树形态，需要增加的空结点个数不多，也可适合采用顺序存储结构。

- 如果需要增加很多空结点才能将一棵二叉树改造成为一棵完全二叉树，采用顺序存储结构会造成空间的大量浪费，这时不宜用顺序存储结构。

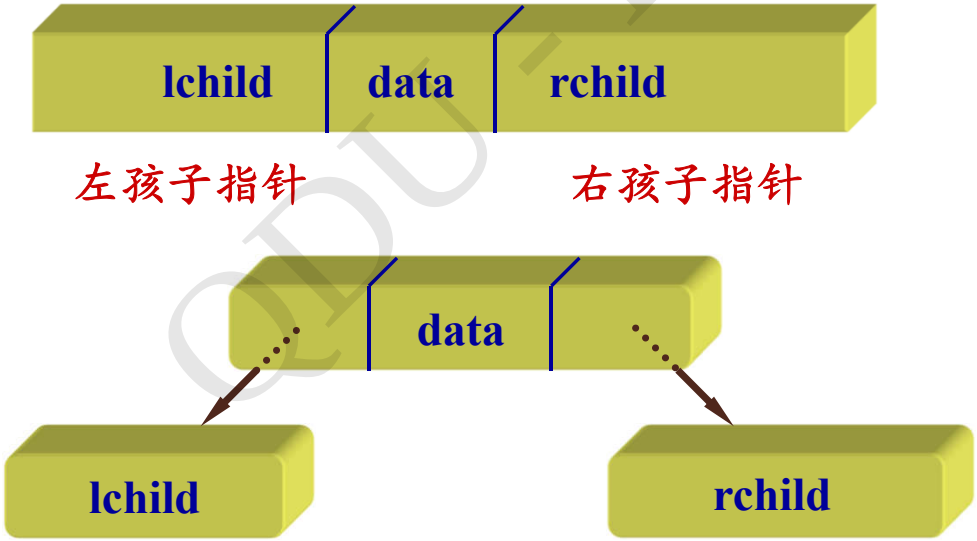


- $h=4$
- $MaxSize=2^4-1=15$
- 空间利用率=4/15=27%

2．链式存储结构

- 对于一般的二叉树，有两种方法：二叉链表表示、三叉链表表示。

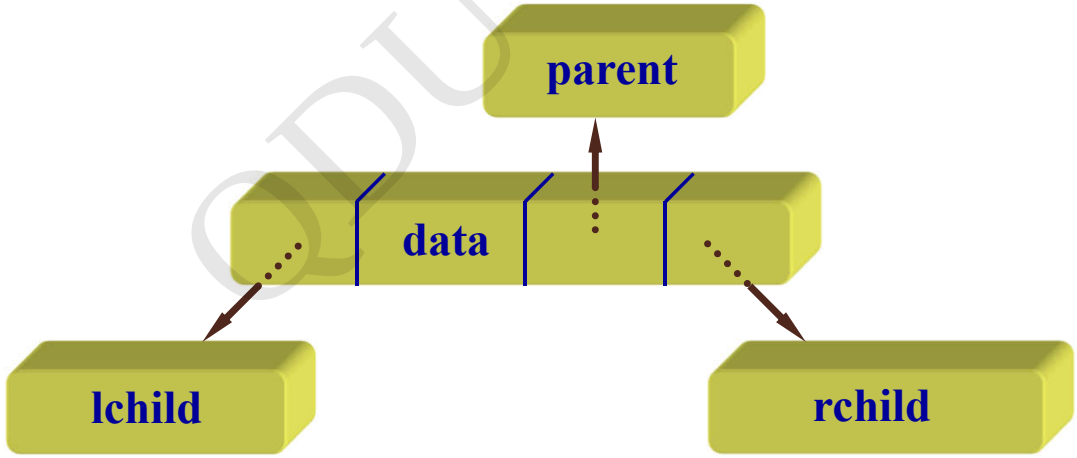
- 二叉树的二叉链表表示



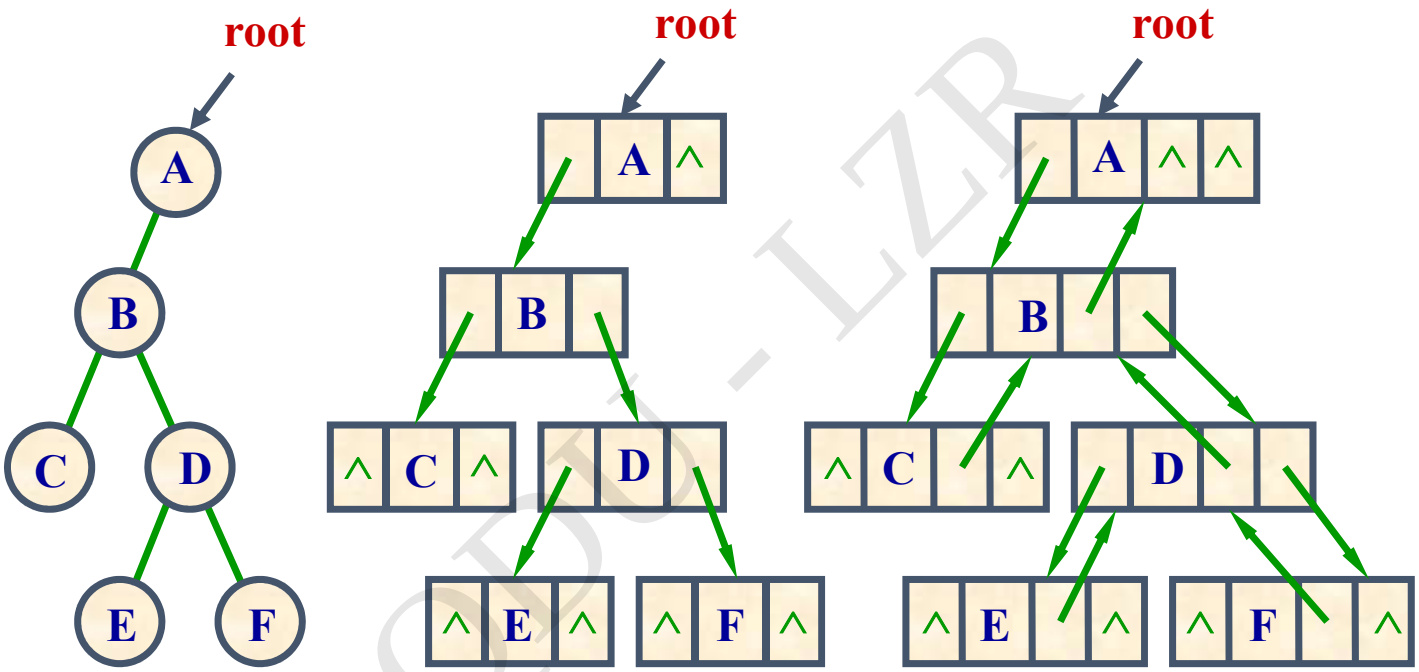
2．链式存储结构

- 二叉树的三叉链表表示

左子女指针　　双亲指针　　右子女指针

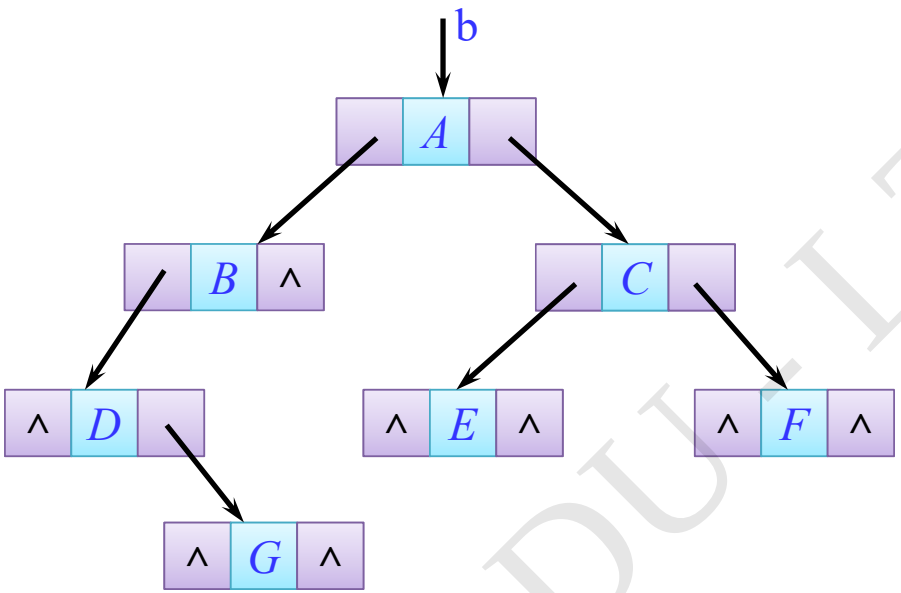


二叉树链表表示的示例



二叉树　　二叉链表　　三叉链表

在二叉链中，空指针的个数？



n=7

空指针域个数=8

- ✓ n个结点 ⇨ $2n$个指针域
- ✓ 分支数为$n-1$ ⇨ 非空指针域有$n-1$个
- ✓ 空指针域个数 = $2n-(n-1) = n+1$

在二叉链存储中，结点的类型声明如下：

```
// ----- 二叉树的二叉链表存储表示 ------
typedef struct BiTNode {
    TElemType data;
    BiTNode *lchild, *rchild;  //左右孩子指针
} BiTNode, *BiTree;
```

- data表示数据域，用于存储放入结点值（默认情况下为单个字母）。
- lchild和rchild分别表示左指针域和右指针域，分别存储左孩子和右孩子结点（即左、右子树的根结点）的存储地址。
- 当某结点不存在左或右孩子时，其lchild或rchild指针域取特殊值NULL。

END