3.2 二叉树及存储结构

二叉树的定义

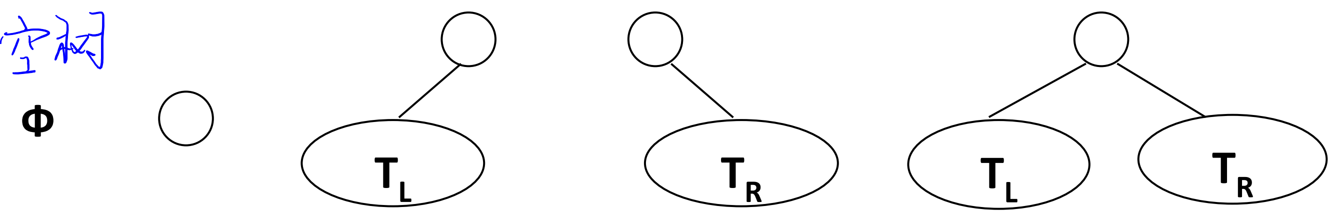
二叉树T：一个有穷的结点集合。

这个集合可以为空

若不为空，则它是由根结点和称为其左子树T_L和右子树T_R的两个不相交的二叉树组成。

- 二叉树具体五种基本形态

空树



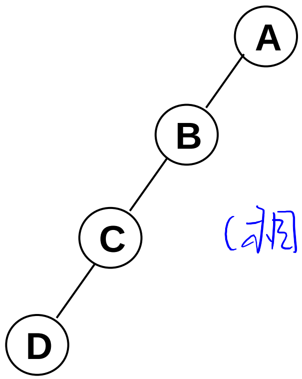
(a) (b) (c) (d) (e)

- 二叉树的子树有左右顺序之分

(与一般的度为2的树的区别)

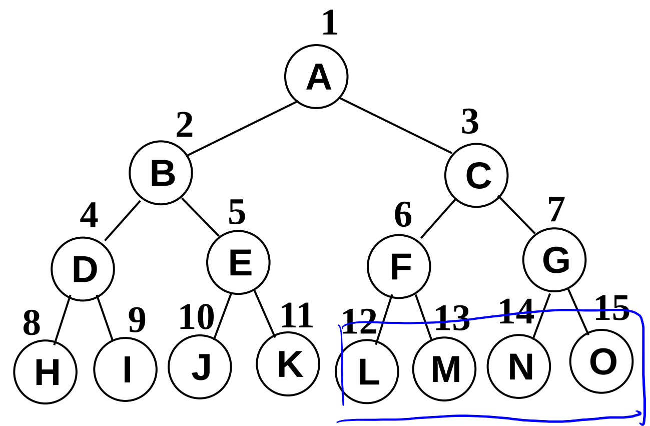
❖ 特殊二叉树

- 斜二叉树(Skewed Binary Tree)



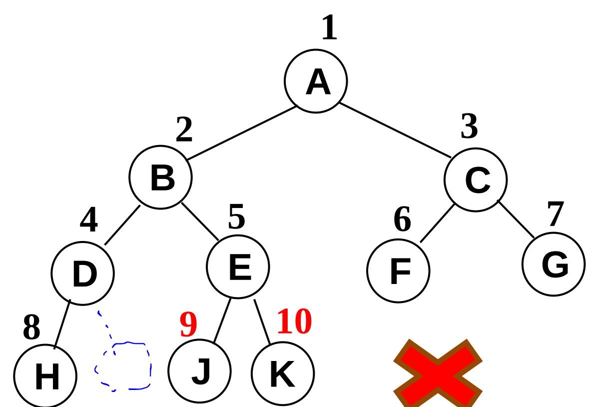
(相当于链表了)

- 完美二叉树(Perfect Binary Tree)
  满二叉树(Full Binary Tree)



- 完全二叉树 → 完美二叉树缺少最下面几个结点
  (Complete Binary Tree)

有n个结点的二叉树，对树中结点按从上至下、从左到右顺序进行编号，编号为i（1 ≤ i ≤ n）结点与满二叉树中编号为 i 结点在二叉树中位置相同



少了这个

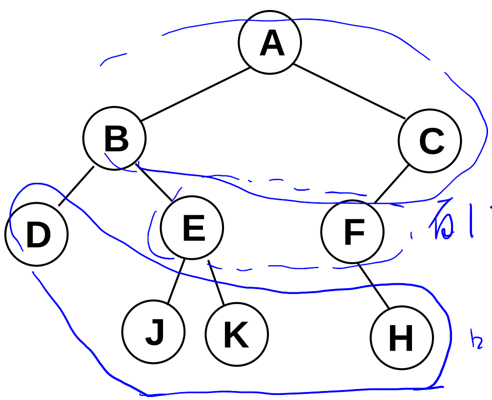
二叉树几个重要性质

- 一个二叉树第 *i* 层的最大结点数为：2^{i-1}，$i \geq 1$。

$1+2^1+2^2+\dots+2^{k-1} = 2^k-1$

- 深度为*k*的二叉树有最大结点总数为： 2^k-1，$k \geq 1$。

- 对任何非空二叉树 T，若n_0表示叶结点的个数、n_2是度为2的非叶结点个数，那么两者满足关系$n_0 = n_2 + 1$。



有2个子结点. $n_2=3$

有1个子结点. $n_1=2$

叶结点. $n_0=4$

边数 $= n_0+n_1+n_2-1 = 0n_0+1n_1+2n_2$

$\Rightarrow n_0 = n_2+1$

- $n_0 = 4$，$n_1 = 2$
- $n_2 = 3$；
- $n_0 = n_2 + 1$

二叉树的抽象数据类型定义

类型名称：二叉树

数据对象集：一个有穷的结点集合。

若不为空，则由根结点和其左、右二叉子树组成。

操作集： BT∈ BinTree, Item ∈ ElementType，重要操作有：

1、Boolean IsEmpty(BinTree BT)： 判别BT是否为空；

2、void Traversal(BinTree BT)：遍历，按某顺序访问每个结点；

3、BinTree CreatBinTree( )：创建一个二叉树。

常用的遍历方法有：

- void PreOrderTraversal(BinTree BT)：先序----根、左子树、右子树；
- void InOrderTraversal(BinTree BT)： 中序---左子树、根、右子树；
- void PostOrderTraversal(BinTree BT)：后序---左子树、右子树、根
- void LevelOrderTraversal(BinTree BT)：层次遍历，从上到下、从左到右

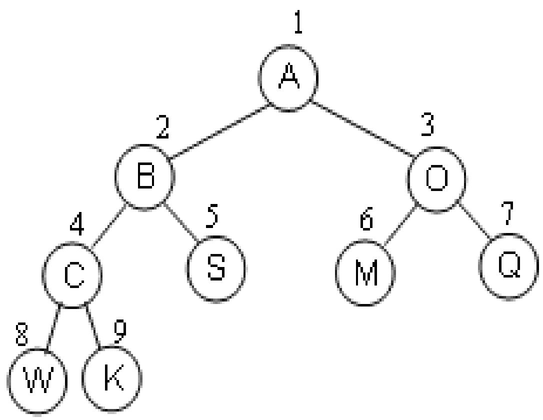
二叉树的存储结构

1. 顺序存储结构

可以用数组实现

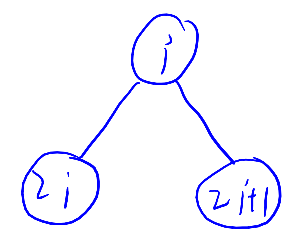
完全二叉树：按从上至下、从左到右顺序存储

n个结点的完全二叉树的结点父子关系：



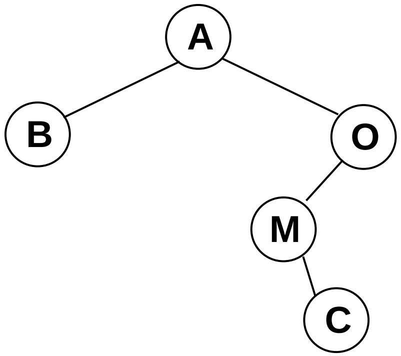
向下取整

- 非根结点（序号 i > 1）的父结点的序号是 $\lfloor i / 2 \rfloor$;
- 结点（序号为 i ）的左孩子结点的序号是 2i，
  （若2 i <= n，否则没有左孩子）；
- 结点（序号为 i ）的右孩子结点的序号是 2i+1，
  （若2 i +1<= n，否则没有右孩子）；

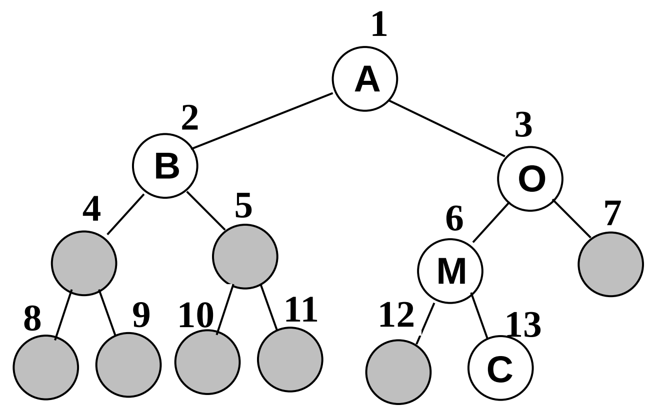


结点	A	B	O	C	S	M	Q	W	K
序号	1	2	3	4	5	6	7	8	9

□ 一般二叉树也可以采用这种结构，但会造成空间浪费……



(a)一般二叉树



(b) 对应的完全二叉树

结点	A	B	O	∧	∧	M	∧	∧	∧	∧	∧	∧	C
序号	1	2	3	4	5	6	7	8	9	10	11	12	13

造成空间浪费！

2. 链表存储

```
typedef  struct  TreeNode *BinTree;
typedef  BinTree Position;
struct  TreeNode{
        ElementType  Data;
        BinTree  Left;
        BinTree  Right;
}
```

指针

Left	Data	Right