Q1. 什么是树

Q2. 度的概念/什么是树的度

Q3. 为什么结点数=总度数+1

Q4. 度为m/m叉树的区别

Q5. 度为m的树第i层最多有多少个节点

Q6. 高度为h的m叉树节点最多有?

Q7. 具有n个节点的m叉树的最小高度为$log_m(n(m-1)-1)$

Q8. 满二叉树的定义/性质

Q9. 什么是二叉排序树

Q10. 什么是平衡二叉树

Q11. 如何推出$n_0 = n_2 + 1$

Q12. 完全二叉树的高度范围

Q13. 树的顺序存储/链式存储

Q14. 树的遍历

Q1.
树是n(n>=0)个节点的有限集。当n=0的时候称为空树，在任意一棵非空树应满足

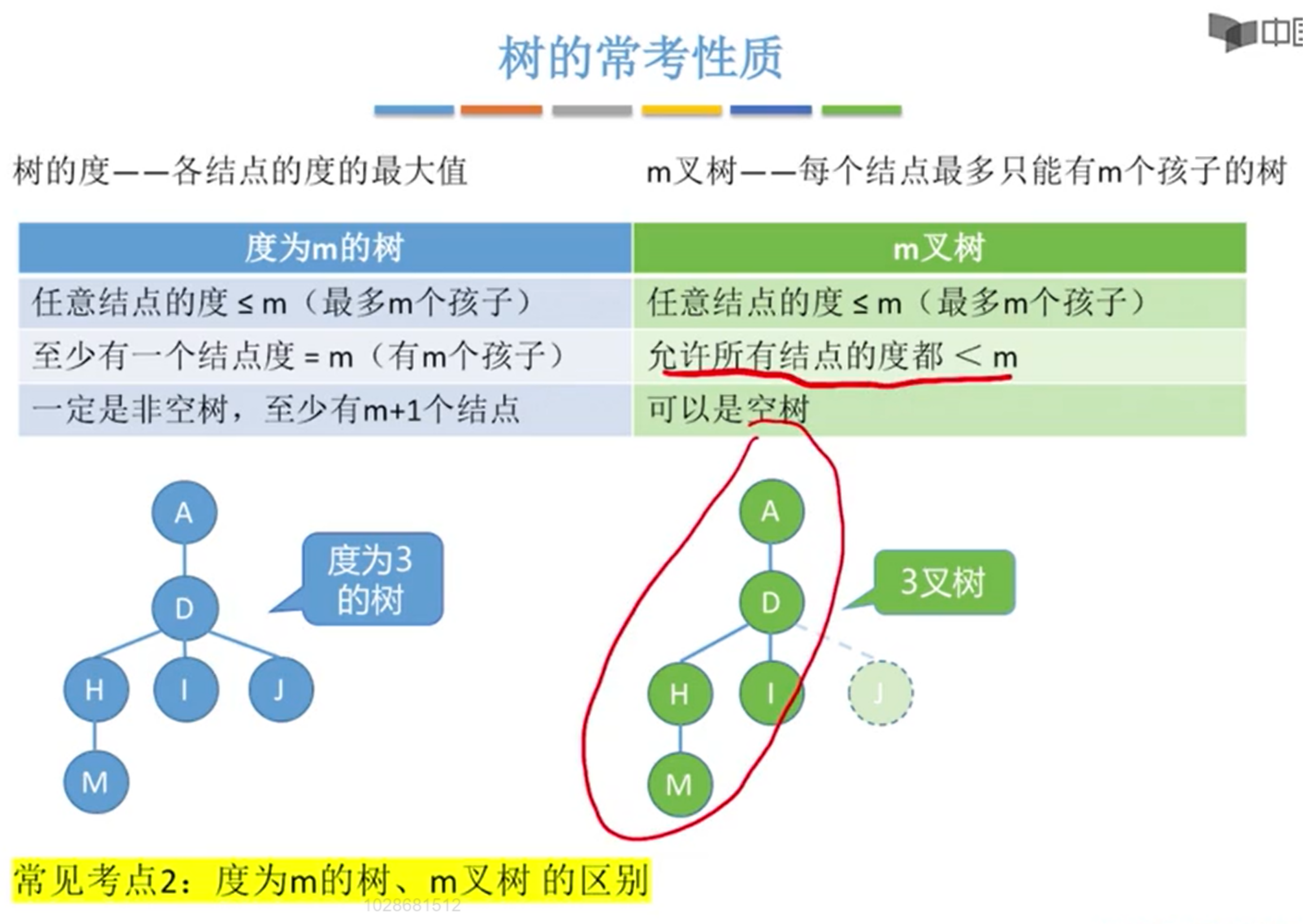
- 有且仅有一个特定的称为跟的节点
- 当n>1时，其余节点可以分为m(m>0)个互不相交的有限集 $T_1, T_2, T_3, \cdots$,其中每个集合本身又是一棵树，并且称为根的子树

Q2.
树中孩子的个数称为该节点的度，树中节点的最大度数称为树的度

Q3.
以归纳的思想来看，叶子节点度为0可以归纳给父节点，逐层网上递推发现只有根节点无代表(根结点的度代表其左右孩子)，因此

$$\text{结点数} = \text{总度数} + 1 \qquad (1\text{代表了根节点})$$

Q4.



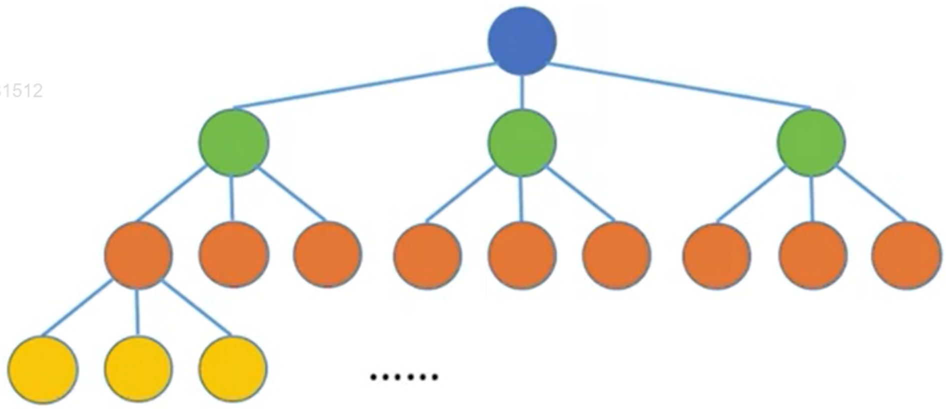
Q5.

度为m的树，每层最多可以有m^{i-1}个节点

常见考点3：度为m的树第 i 层至多有 m^{i-1} 个结点（i≥1）

m叉树第 i 层至多有 m^{i-1} 个结点（i≥1）

28681512



第 1 层：m^0

第 2 层：m^1 3^1

第 3 层：m^2 3^2

第 4 层：m^3

Q6.

利用等比数列求和的知识 最终结果是$\frac{m^h-1}{m-1}$个节点

Q7.

思路：最小高度意味着是每一个节点尽可能有更多的孩子，也就是满m叉树，因此满足

常见考点6：具有n个结点的m叉树的最小高度为$\lceil \log_m(n(m-1)+1) \rceil$

高度最小的情况——所有结点都有m个孩子

前h-1层最多有几个结点

$$\frac{m^{h-1}-1}{m-1} < n \le \frac{m^h-1}{m-1}$$

前h层最多有几个结点

$$m^{h-1} < n(m-1)+1 \le mh$$

$$h-1 < \log_m(n(m-1)+1) \le h$$

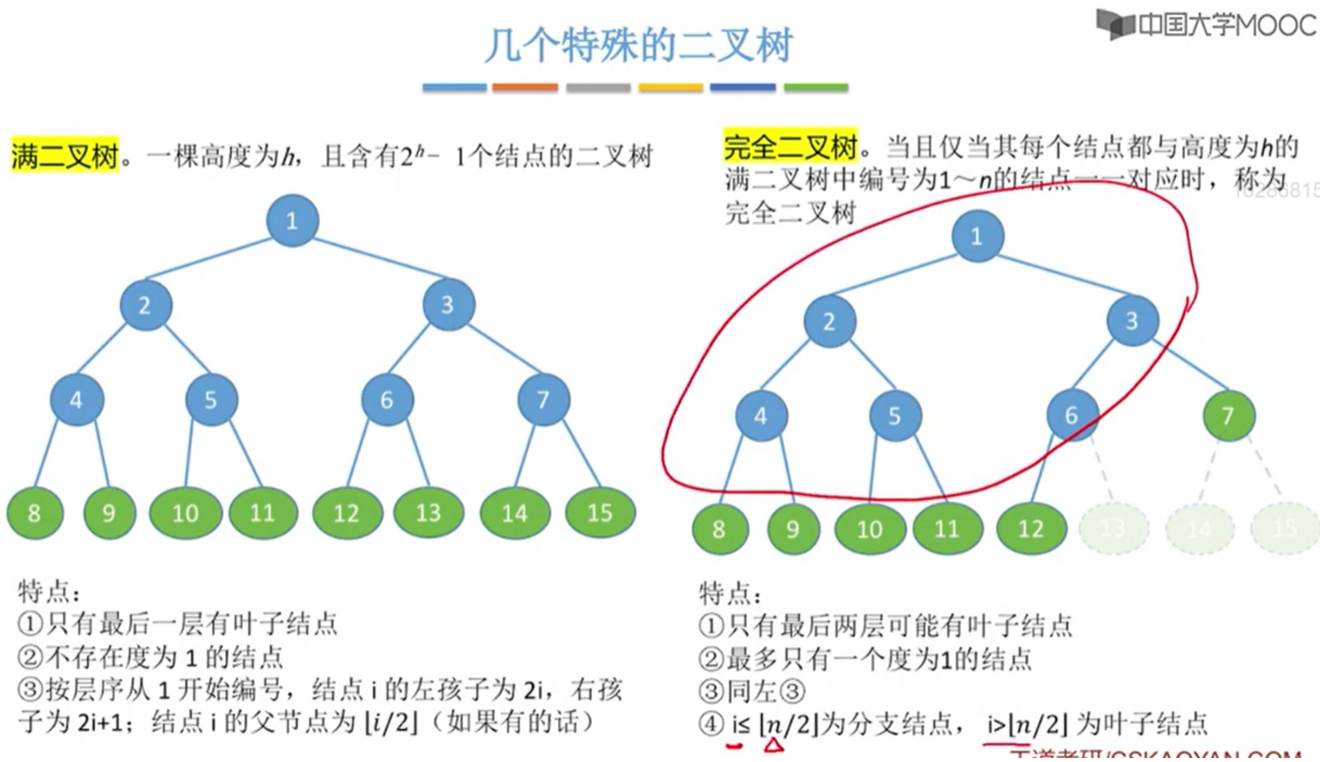
$$h_{min} = \lceil \log_m(n(m-1)+1) \rceil$$

Q8.

满二叉树：高度为h，且含有2^h-1个节点的二叉树

完全二叉树： 高度为h，当且仅当每个节点都与高度为h的满二叉树编号为1~n的节点一一对应。
区别：

- 满二叉树不存在度为1的节点，完全二叉树只存在于一个度为1的节点
- 满二叉树只在最后一层有叶子节点，完全二叉树在最后两层可能有叶子节点
- 完全二叉树和满二叉树都符合编号规范



Q9.
左子树上的所有关键字都小于根节点的关键字，右子树上的所有关键都大于根节点上的关键字。(递归地，左右子树也满足上述性质
)

Q10.
树上任一结点的左子树和右子树的深度之查不超过1(平衡二叉树有更好的搜索效率)

Q11.
联立$n = n_0 + n_1 + n_2$,
$n = n_1 + 2n_2 + 1 \qquad (\text{结点数} = \text{总度数} + 1)$

Q12.
h为向上取整，方法参考m叉树的高度求法

常见考点1：具有n个（$n > 0$）结点的完全二叉树的高度h为$\lceil \log_2(n+1) \rceil$或$\lfloor \log_2 n \rfloor + 1$

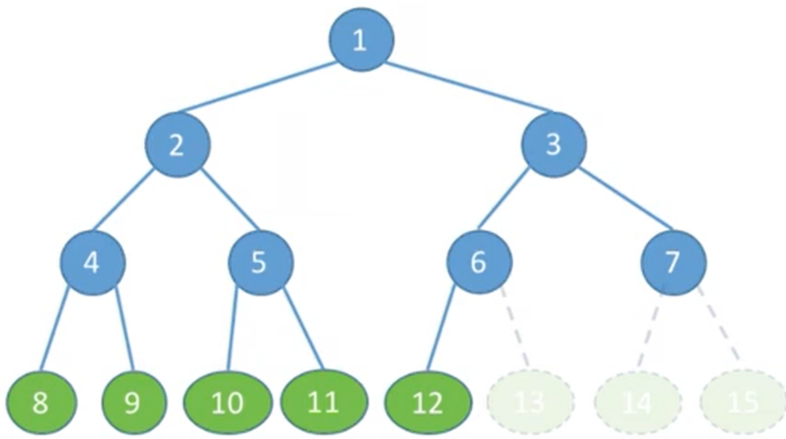
高为 h 的满二叉树共有 $2^h - 1$ 个结点
高为 h-1 的满二叉树共有 $\boxed{2^{h-1} - 1}$个结点

$$2^{h-1} - 1 < n \le 2^h - 1$$

$$2^{h-1} < n + 1 \le 2^h$$

$$h - 1 < \log_2(n+1) \le h$$

$$h = \lceil \log_2(n+1) \rceil$$

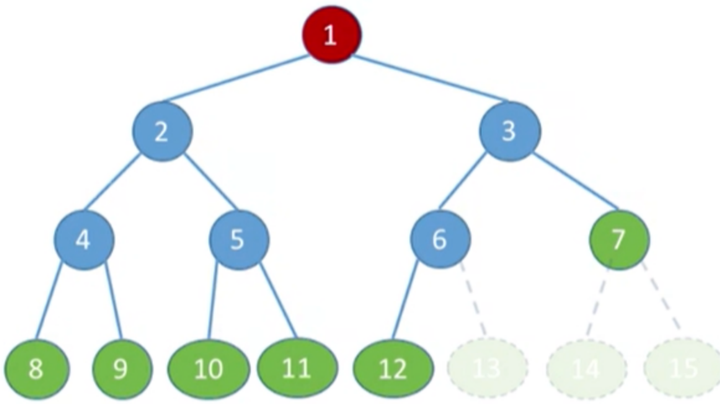


Q13.

```
#define MaxSize 100
struct TreeNode{
    Elemtype value;
    bool isEmpty;
}
TreeNode t[MaxSize];
```

中国大学MOOC

二叉树的顺序存储




```c
#define MaxSize 100
struct TreeNode {
    ElemType value; //结点中的数据元素
    bool isEmpty;   //结点是否为空
};
```

```c
TreeNode t[MaxSize];
```

定义一个长度为 MaxSize 的数组 t，按照从上至下、从左至右的顺序依次存储完全二叉树中的各个结点

t[0] t[1] t[2] ......

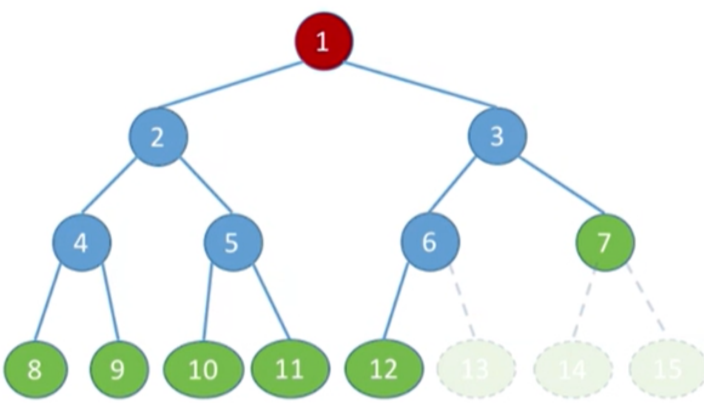
可以让第一个位置空缺，保证数组下标和结点编号一致

初始化时所有结点标记为空

```c
for (int i=0; i<MaxSize; i++){
    t[i].isEmpty=true;
}
```

中国大学MOOC

二叉树的顺序存储

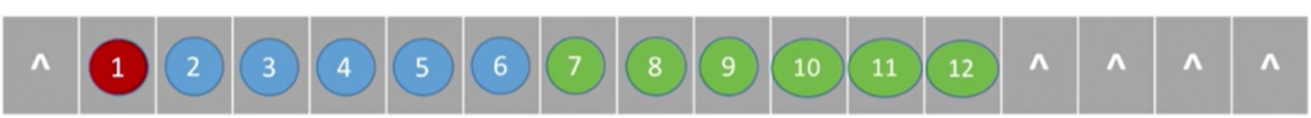


几个重要常考的基本操作：

- i 的左孩子 ——2i
- i 的右孩子 ——2i+1
- i 的父节点 ——$\lfloor i/2 \rfloor$
- i 所在的层次 ——$\lceil \log_2(n+1) \rceil$或$\lfloor \log_2 n \rfloor + 1$

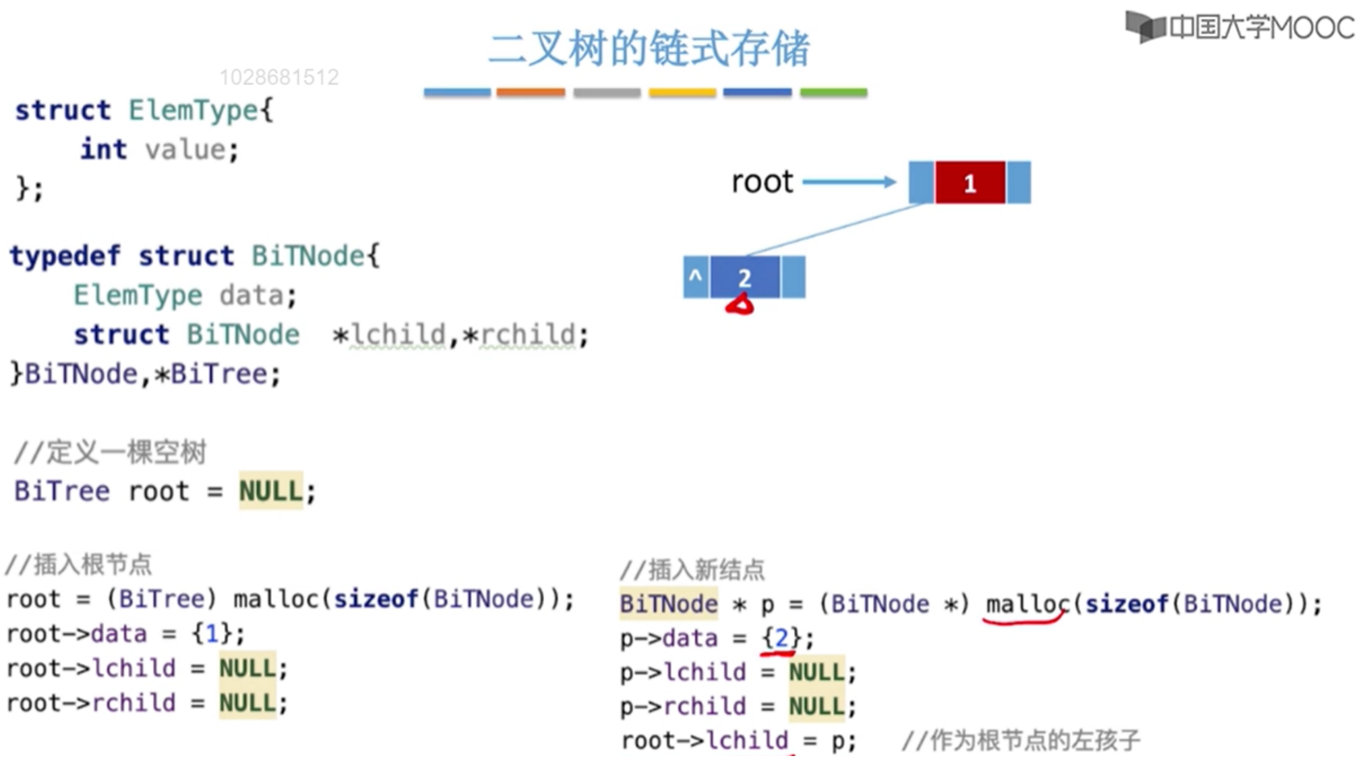
若完全二叉树中共有n个结点，则

- 判断 i 是否有左孩子？ ——$2i \le n$？
- 判断 i 是否有右孩子？ ——$2i+1 \le n$？
- 判断 i 是否是叶子/分支结点？



t[0] t[1] t[2] ......

```c
struct Elemtype {
    int value;
}
typedef struct BiTNode{
    ElemType data;
    struct BiTNode  *lchild ,*rchild; //pointer to lchild && rchild
}BiTNode,*BiTree;
```



Q14.

先序遍历：根左右(NLR)

中序遍历：左根右(LNR)

后序遍历：左右根(LRN)