

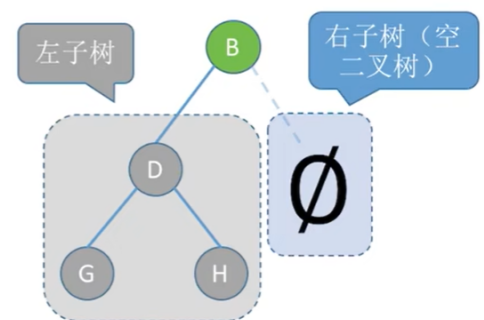
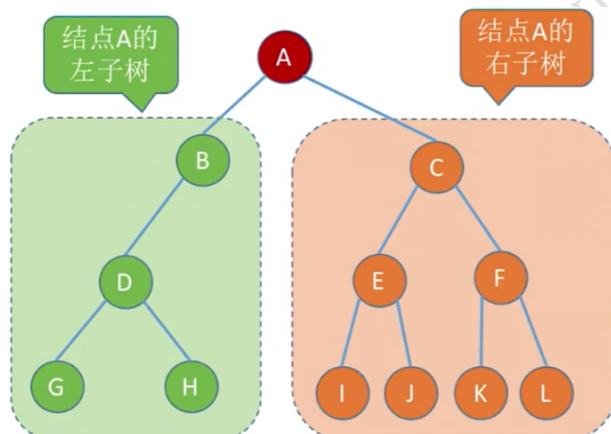
Lecture 16: 二叉树常考性质

16.1 二叉树的定义和基本术语

16.1.1 二叉树的基本概念

基本概念

- 二叉树是 $n(n \geq 0)$ 个结点的有限集合：
- 两种二叉树
 - 1. 或者空二叉树，即 $n = 0$
 - 2. 或者由一个根节点和两个互不相交的被称为根的左子树和右子树分成两棵二叉树
- 特点
 - 1. 每个结点至多只有两棵子树；
 - 2. 左右子树不能颠倒 (二叉树是有序树)；
 - 3. 度为 2 的有序树：在这棵树里面至少有一个结点它的度为 2，而二叉树无要求；
- 补充：
 - 二叉树为空，只是代表二叉树没有结点，不代表此二叉树不存在；
- 图示：

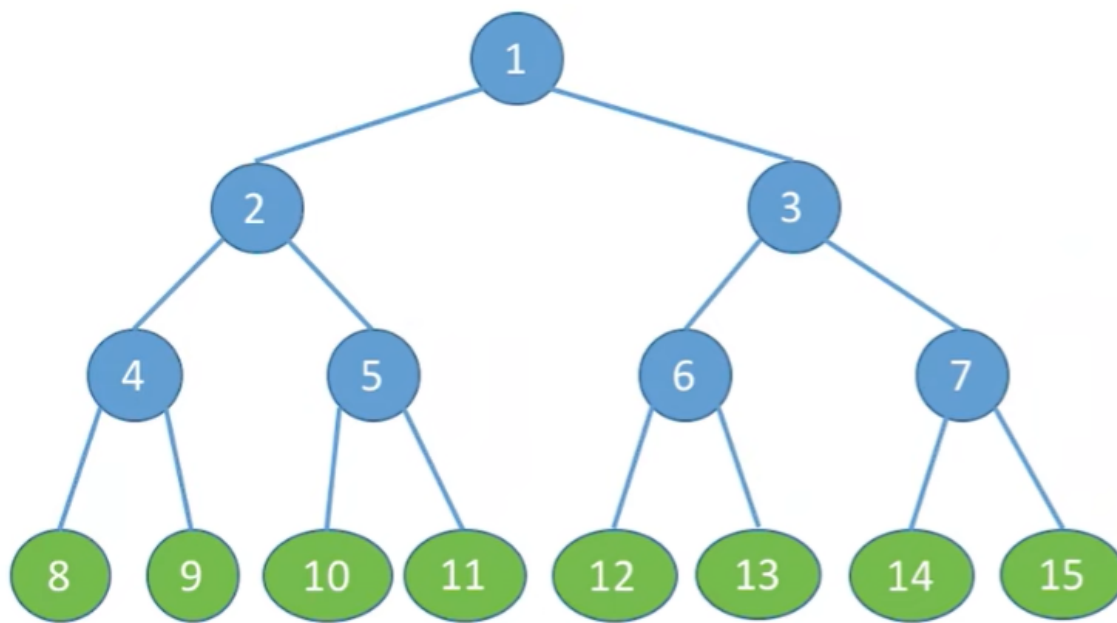


二叉树是递归定义的数据结构

16.1.2 常见二叉树

满二叉树

- 概念：
 - 一颗高度为 h ，且含有 $2^h - 1$ 个结点的二叉树
- 特点
 - 1. 只有最后一层有叶子节点；
 - 2. 不存在度为 1 的结点；
 - 3. 按层序从 1 开始编号，结点 i 的左孩子为 $2i$ ，右孩子为 $2i+1$ 。结点 i 的父节点为 $\lfloor \frac{i}{2} \rfloor$ (向下取整) (如果有的话)；
- 图示：

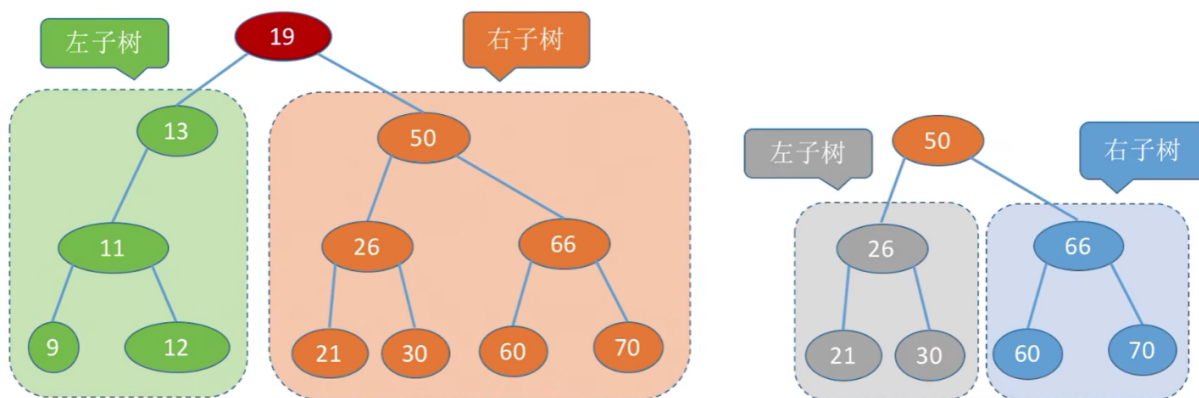


完全二叉树

- 概念：
 - 当且仅当每个结点都与高度为 h 的满二叉树中编号为 $1 \sim n$ 的结点一一对应时，称为完全二叉树
- 特点
 - 1. 只有最后两层可能有叶子结点
 - 2. 最多只有一个度为 1 的结点
 - 3. 按层序从 1 开始编号，结点 i 的左孩子为 $2i$ ，右孩子为 $2i+1$ 。结点 i 的父节点为 $\lfloor \frac{i}{2} \rfloor$ （向下取整）（如果有的话）；
 - 4. $i \leq \lfloor \frac{n}{2} \rfloor$ 为**分支结点**， $i > \lfloor \frac{n}{2} \rfloor$ 为**叶子节点**
 - 5. 如果某个结点只有一个孩子那么一定是**左孩子（因为顺序编号）（偶数）**
- 解释：即如果按照满二叉树排列后，按照从大到小的数字进行删除，删除后的树为完全二叉树；

二叉排序树

- 用途：
 - 二叉树可以用于元素的排序，搜索
- 特征：
 - 1. **左子树**上所有结点的**关键字**均小于根节点的关键字；
 - 2. **右子树**上所有结点的**关键字**均大于根节点的关键字；
 - 3. **左子树和右子树又各是一棵二叉排序树**；
- 图示：



概念：平衡二叉树

- 树上任一结点的**左子树**和**右子树**的深度之差不超过 1；

16.2 二叉树的性质

最常用性质

- 二叉树第 k 层：

$$\text{第}k\text{层} = 2^{k-1}\text{个结点}$$

- 二叉树结点数量：

$$n_0 = n_2 + 1$$

- 二叉树结点最大数量：

$$\text{高为}h, \text{最多结点数} = 2^h - 1$$

- 二叉树高度范围

$$\text{高度范围} : \lfloor \log_2 n \rfloor + 1 \sim n$$

- 二叉树空指针总数：

$$\text{空指针数} = 2 * n_0 + 1 * n_1 = n_0 + n_0 + n_1 = n_0 + n_2 + 1 + n_1 = n + 1$$

16.2.1 二叉树的常考性质

考点一：设非空二叉树度为 0, 1 和 2 的结点个数分别为 n_0, n_1 和 n_2 ,

- 结论：

$$n_0 = n_2 + 1$$

- 解释：

- 因为叶子结点比二分支结点多一个，所以 $n_0 = n_2 + 1$

- 推导：

$$n = n_0 + n_1 + n_2$$

$$n = n_1 + 2n_2 + 1$$

- 第一个公式: $n = n_0 + n_1 + n_2 \rightarrow$ 节点个数 = 度为0节点数 + 度为1节点数 + 度为2节点数;
- 第二个公式: $n = n_1 + 2n_2 + 1 \rightarrow$ 节点个数 = 总度数 + 1 = (度为1节点数 + 与之相连的度为2的节点数 + 1 个根节点);
- 带入解出: $n_0 = n_2 + 1$

考点二: 每层节点数

- 二叉树第 i 层至多有 2^{i-1} 个结点 ($i \geq 1$)
- m 叉数第 i 层至多有 m^{i-1} 个结点 ($i \geq 1$)

考点三

- 高度为 h 的二叉树至多有 $2^h - 1$ 个结点 (满二叉树)
- 高度为 h 的 m 叉树至多有 $\frac{m^h - 1}{m - 1}$ 个结点

16.2.2 完全二叉树的常考性质

考点一: 具有 n 个 ($n > 0$) 结点的完全二叉树的高度 h 为

- 高度 $h = \lceil \log_2(n + 1) \rceil$ 或 $\lfloor \log_2 n \rfloor + 1$;
- 注意: 一个是和上层比较, 一个和下次比较, 所以不同

考点二: 对于完全二叉树, 可以由结点数 n 推出为 0, 1 和 2 的结点个数 n_0, n_1 和 n_2

- 完全二叉树最多只有一个度为 1 的结点, 即:
 - $n_1 = 0$ 或 1
 - $n_0 = n_2 + 1 \rightarrow n_0 + n_2$ 一定是奇数
- 推导
 - 若完全二叉树有 $2k$ 个 (偶数) 个结点, 则必有 $n_1 = 1, n_0 = k, n_2 = k - 1$
 - 若完全二叉树有 $2k - 1$ 个 (奇数) 个结点, 则必有 $n_1 = 0, n_0 = k, n_2 = k - 1$

完全二叉树性质补充

- 1. 当 $i > 1$ 时, 结点 i 的双亲编号为 $\lfloor \frac{i}{2} \rfloor$
 - 当 i 为偶数时, 其双亲的编号为 $\frac{i}{2}$, 它是双亲的左孩子
 - 当 i 为奇数时, 其双亲的编号为 $\frac{i-1}{2}$, 它是双亲的右孩子
- 2. 当 $2i \leq n$ 时, 结点 i 的左孩子编号为 $2i$, 否则无左孩子。
- 3. 当 $2i + 1 \leq n$ 时, 结点 i 的右孩子编号为 $2i + 1$, 否则无右孩子。
- 4. 结点 i 所在层次 (深度) 为 $\lfloor \log_2 i \rfloor + 1$ 。

16.3 二叉树的存储结构

16.3.1 顺序存储

什么是顺序存储

- 结论:

- 二叉树的顺序存储结构，只适合存储**完全二叉树**；

- 概念：

- 定义一个长度为 `MaxSize` 的数组 `t`，按照**从上至下，从左至右**的顺序依次存储中的各个结点；

- 注意：

- 非完全二叉树不能通过 `i` 和 `n` 的比较来判断是否为左右结点 (因为不满)，只能用 `isEmpty` 来判断

- 初始化：

- `for(int i=0;i<Maxsize;i++)`



`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

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

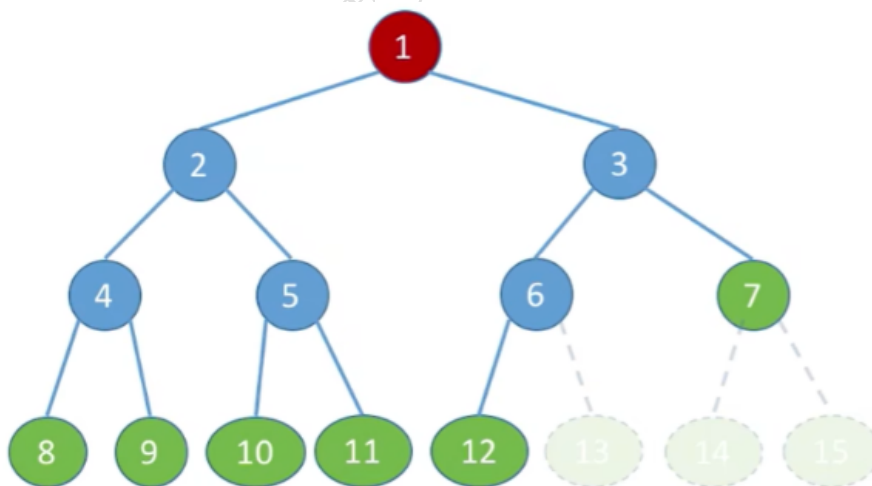
- 基本操作：

- `i` 的左孩子：`2i`
- `i` 的右孩子：`2i+1`
- `i` 的父节点：`[i/2]`
- `i` 所在的层次：`⌈log2(n + 1)⌉` 或 `⌊log2 n⌋ + 1`

- 基本判断操作：

- 判断 `i` 是否有左孩子：`2i ≤ n`
- 判断 `i` 是否有右孩子：`2i+1 ≤ n`
- 判断 `i` 是否是叶子/分支结点：`i > [n/2]`

- 完全二叉树：



非完全二叉树

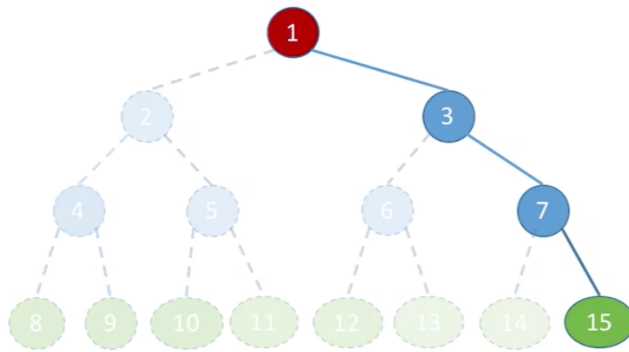
- 概念：

- 1. 非完全二叉树的节点编号无法和节点连续起来；
- 2. 所以在二叉树的顺序存储中，一定要把**二叉树的结点编号与完全二叉树对应**起来；

- 非完全二叉树的判断操作 `->` 无法判断，需要使用 `isEmpty` 属性来判断；

- 图示：

★ 二叉树的顺序存储中，一定要把二叉树的结点编号与完全二叉树对应起来



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

最坏情况：高度为 h 且只有 h 个结点的单支树（所有结点只有右孩子），也至少需要 2^h-1 个存储单元

结论：二叉树的顺序存储结构，只适合存储完全二叉树



- $t[0]$ $t[1]$ $t[2]$

代码实现

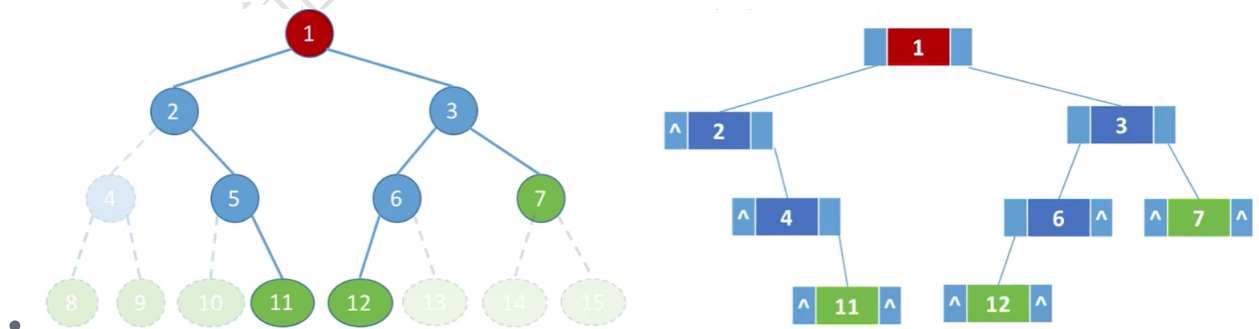
```
#define MaxSize 100
struct TreeNode{
    ElemType value; //结点中的数据元素
    bool isEmpty; //结点是否为空
}

//初始化数组 TreeNode t[MaxSize];
```

16.3.2 链式存储

什么是链式存储

- 注意： n 个结点，就有 $2n$ 个指针域， $n-1$ 个指针指向一个结点（根节点排除），即两者相减得到空链域（可以用于构造线索二叉树）
- 图示：



代码实现

- 节点定义：

```
//二叉树的结点(链式存储)
typedef struct BiTNode{
    ElemType data; //数据域
```

```

    struct BiTNode *lchild,*rchild; //左, 右孩子指针
}BiTNode,*BiTree;

```

- 链式树:

```

//二叉树的链式构建
struct ElemType{
    int value;
};
typedef struct BiTNode{
    ElemType data;
    struct BiTNode *lchild,*rchild;
}BiTNode,*BiTree;

//定义一棵空数(root指向根节点)
BiTree root = NULL;

//插入根节点
root = (BiTree)malloc(sizeof(BiTNode));
root->data = {1};
root->lchild = NULL;
root->rchild = NULL;

//插入新结点
BiTNode *p = (BiTNode *)malloc(sizeof(BiTNode));
p->data = {2};
p->lchild = NULL;
p->rchild = NULL;
root->lchild = p; //作为根节点的左孩子

```

三叉链表

- 概念: 除了左右子节点外, 还多一个指向父节点的指针;
- 目的: 方便找父结点;
- 注意: 但是考研中喜欢考不带 parent 的方式
- 代码:

```

//二叉树的结点(链式存储)
typedef struct BiTNode{
    ElemType data; //数据域
    struct BiTNode *lchild,*rchild; //左, 右孩子指针
    struct BiTNode *parent; //父节点指针
}BiTNode,*BiTree;

```