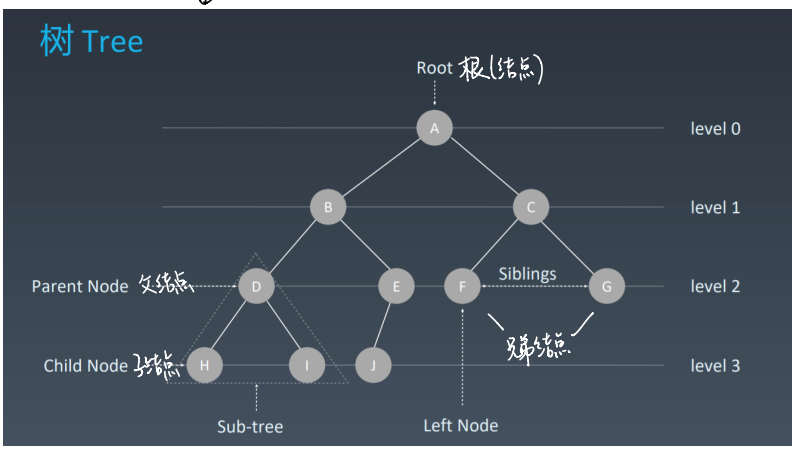
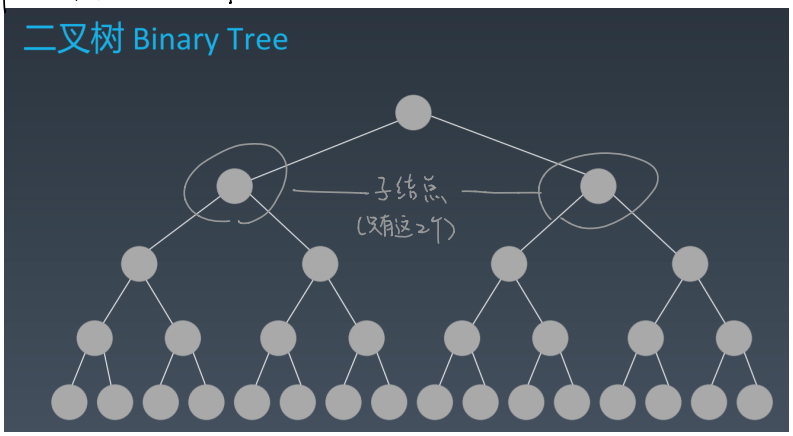
链表：一维　　加速的关键在于：升维

二叉树：二维

指向了节点的话 → 变成



现实中用的最多的：二叉树



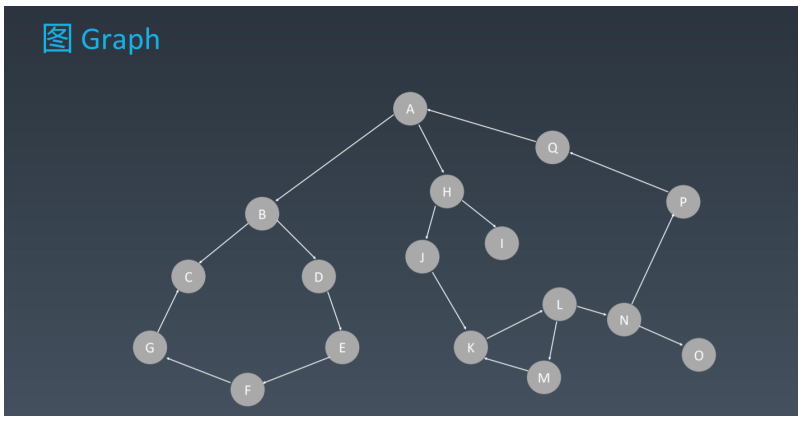
子结点又有了

树 & 图最关键的差别：

看有没有环

(如果它这个结点本身的话，

又连到新的子结点，永远都不会返回去



(有指向) 形成环 → 图

Listed List是特殊化的Tree

Tree是特殊化的Graph

(没有环的图就是树)

树结点：

```java
public class TreeNode {
  public int val;
  public TreeNode left, right;
  public TreeNode(int val) {
    this.val = val;
    this.left = null;
    this.right = null;
  }
}
```

为什么会出来树这个结构

二叉树遍历 Pre-order/In-order/Post-order

O(n)

1.前序（Pre-order）：根-左-右

2.中序（In-order）：左-根-右

3.后序（Post-order）：左-右-根

示例代码

```python
# 前序
def preorder(self, root):

  if root:
    self.traverse_path.append(root.val)   # 根
    self.preorder(root.left)              # 左
    self.preorder(root.right)             # 右

# 中序
def inorder(self, root):

  if root:
    self.inorder(root.left)               # 左
    self.traverse_path.append(root.val)   # 根
    self.inorder(root.right)              # 右

# 后序
def postorder(self, root):

  if root:
    self.postorder(root.left)             # 左
    self.postorder(root.right)            # 右
    self.traverse_path.append(root.val)   # 根
```

基于递归

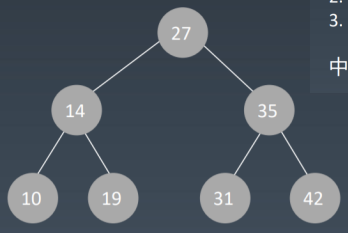
树：

没法有效地进行所谓的循环

（写循环相对比较麻烦

而递归调用相对比较简单

二叉搜索树 Binary Search Tree



二叉搜索树，也称二叉排序树、有序二叉树（Ordered Binary Tree）、排序二叉树（Sorted Binary Tree），是指一棵空树或者具有下列性质的二叉树：

1. 左子树上所有结点的值均小于它的根结点的值；
2. 右子树上所有结点的值均大于它的根结点的值；
3. 以此类推：左、右子树也分别为二叉查找树。（这就是 重复性！）

中序遍历：升序排列

二叉搜索树常见操作

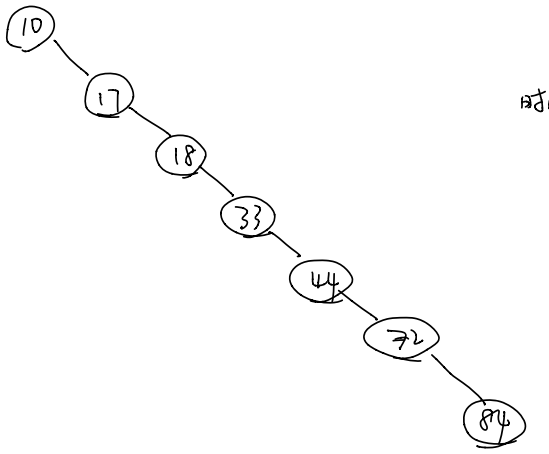
O(logn)

1. 查询
2. 插入新结点（创建）
3. 删除

Demo: https://visualgo.net/zh/bst

特殊情况

for example：



变成了链表

时间复杂度 O(n)