树，总共包含4节内容。具体如下：

1.树、二叉树

2.二叉查找树

3.平衡二叉树、红黑树

4.递归树

章节	内容
23	树、二叉树
24	二叉查找树
25	平衡二叉查找树、红黑树
26	递归树

一、树

1.树的常用概念

根节点、叶子节点、父节点、子节点、兄弟节点，还有节点的高度、深度以及层数，树的高度。

2.概念解释

节点：树中的每个元素称为节点

父子关系：相邻两节点的连线，称为父子关系

根节点：没有父节点的节点

叶子节点：没有子节点的节点

父节点：指向子节点的节点

子节点：被父节点指向的节点

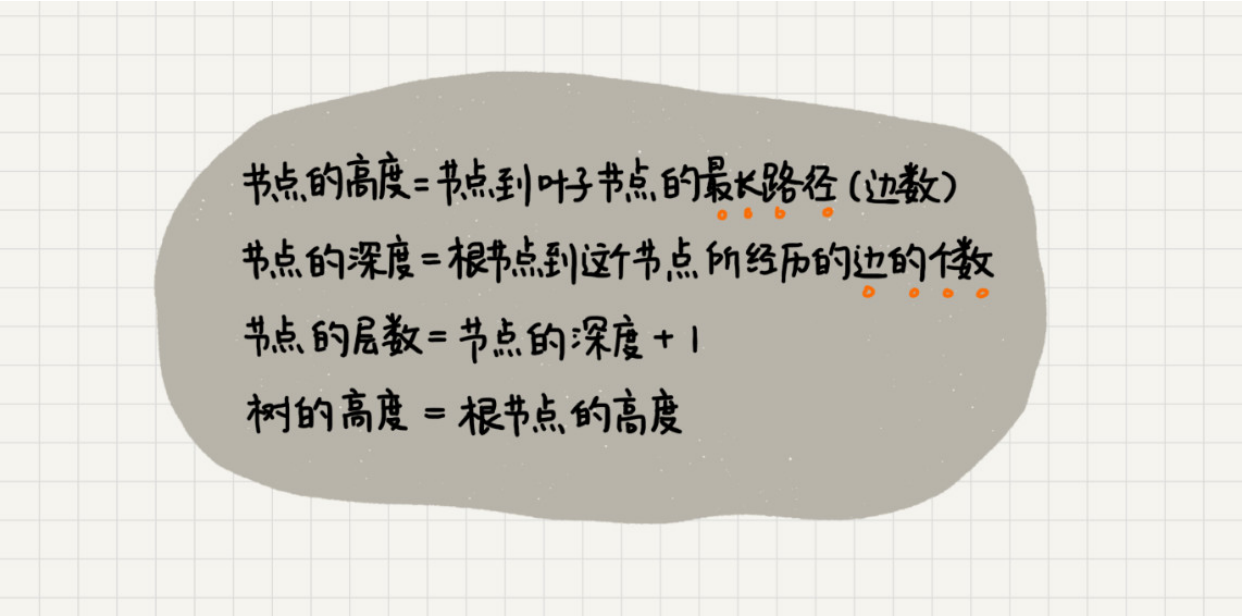
兄弟节点：具有相同父节点的多个节点称为兄弟节点关系

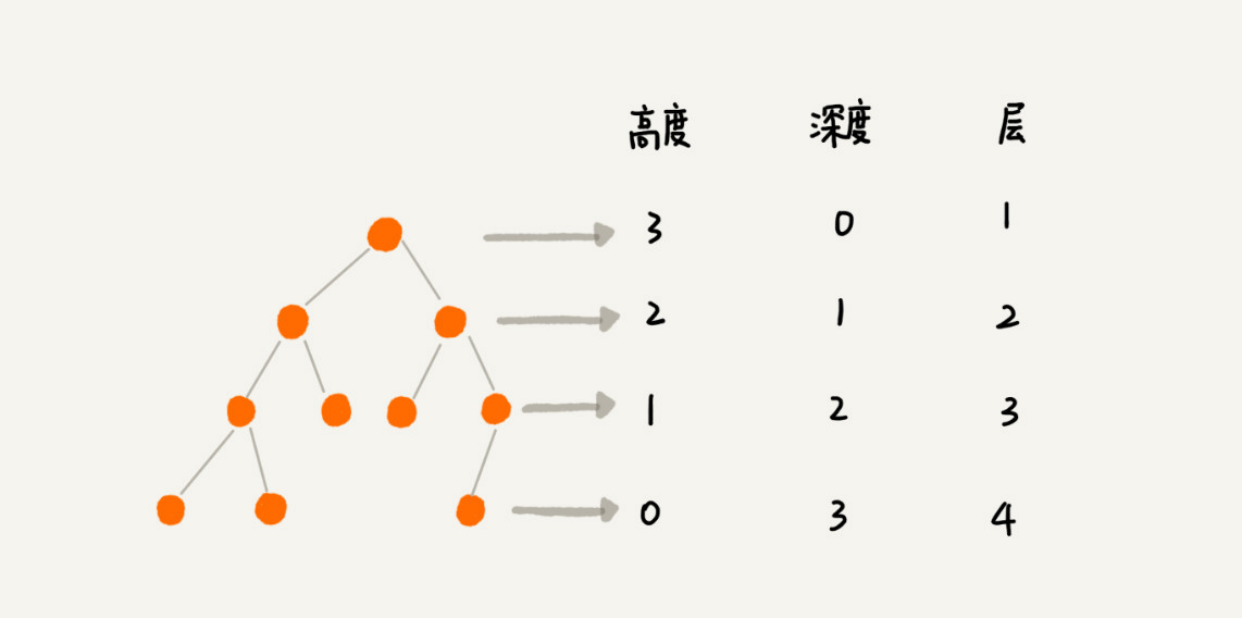
节点的高度：节点到叶子节点的最长路径所包含的边数

节点的深度：根节点到节点的路径所包含的边数

节点的层数：节点的深度+1（根节点的层数是1）

树的高度：等于根节点的高度





二、二叉树

1.概念

①什么是二叉树?

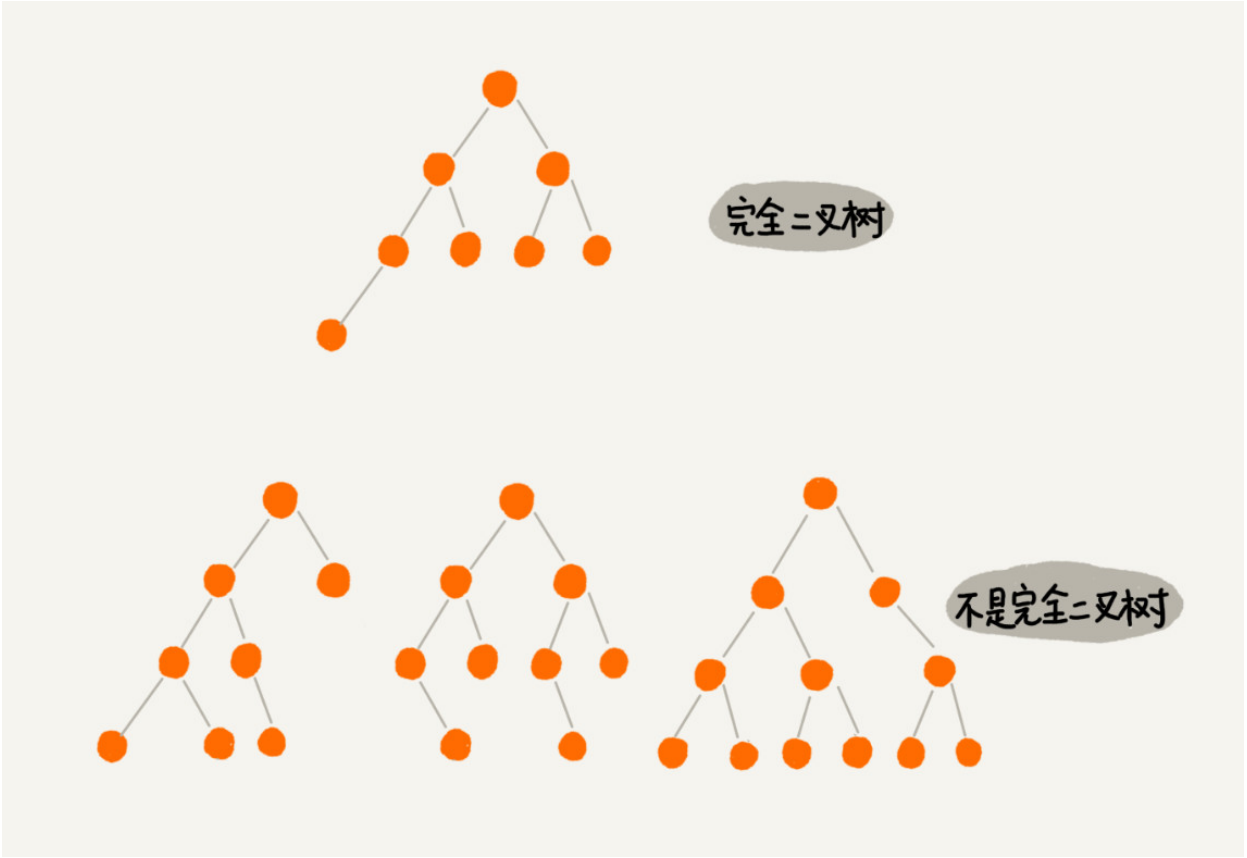
每个节点最多只有2个子节点的树，这两个节点分别是左子节点和右子节点。

②什么是满二叉树?

有一种二叉树，除了叶子节点外，每个节点都有左右两个子节点，这种二叉树叫做满二叉树。

③什么是完全二叉树?

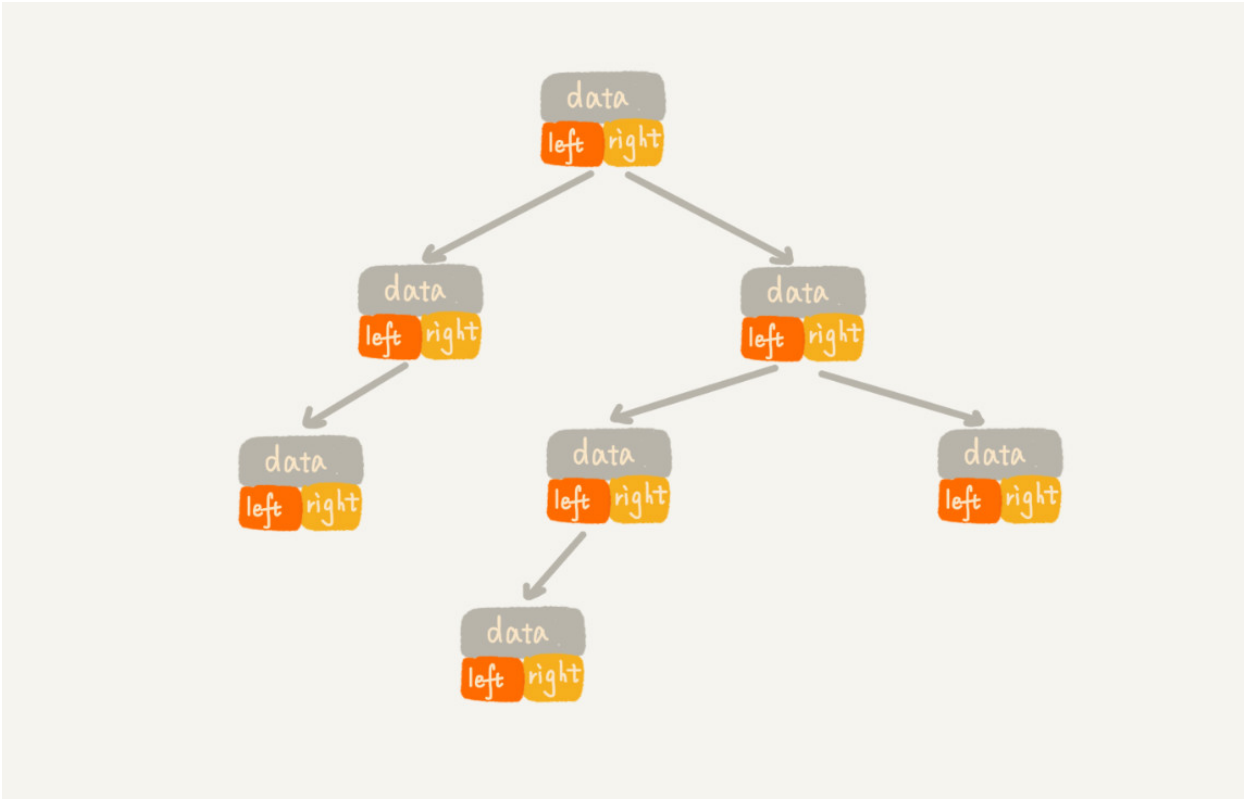
有一种二叉树，叶子节点都在最底下两层，最后一层叶子节都靠左排列，并且除了最后一层，其他层的节点个数都要达到最大，这种二叉树叫做完全二叉树。



2.完全二叉树的存储

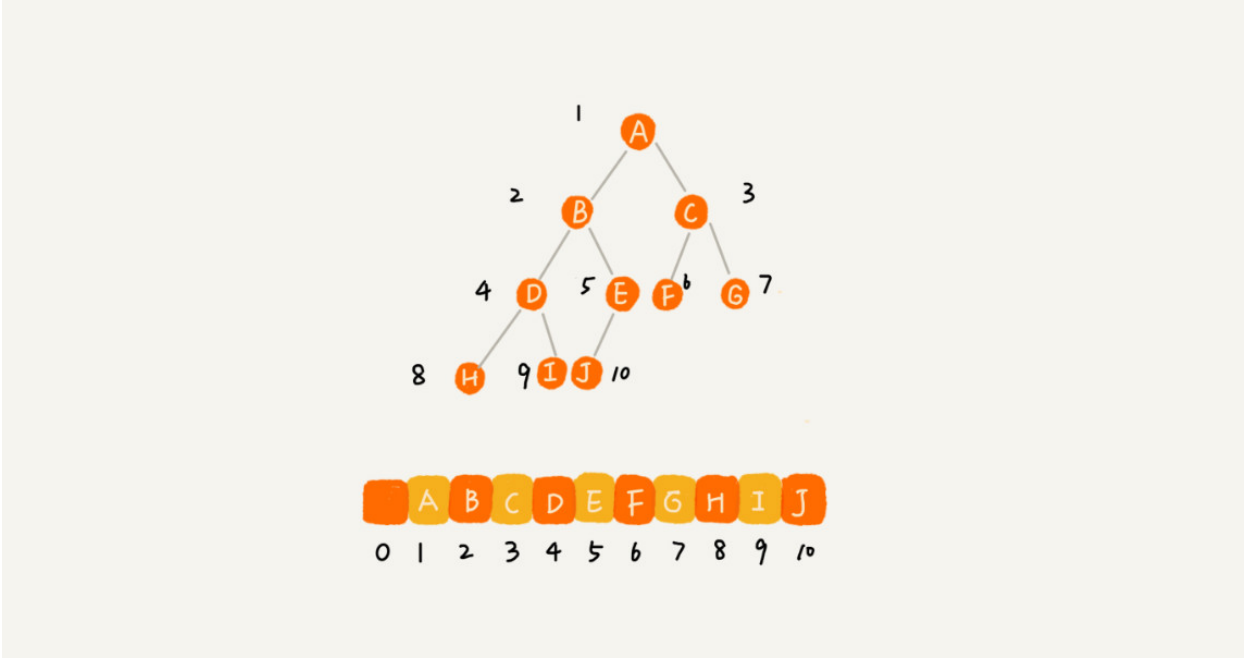
①链式存储

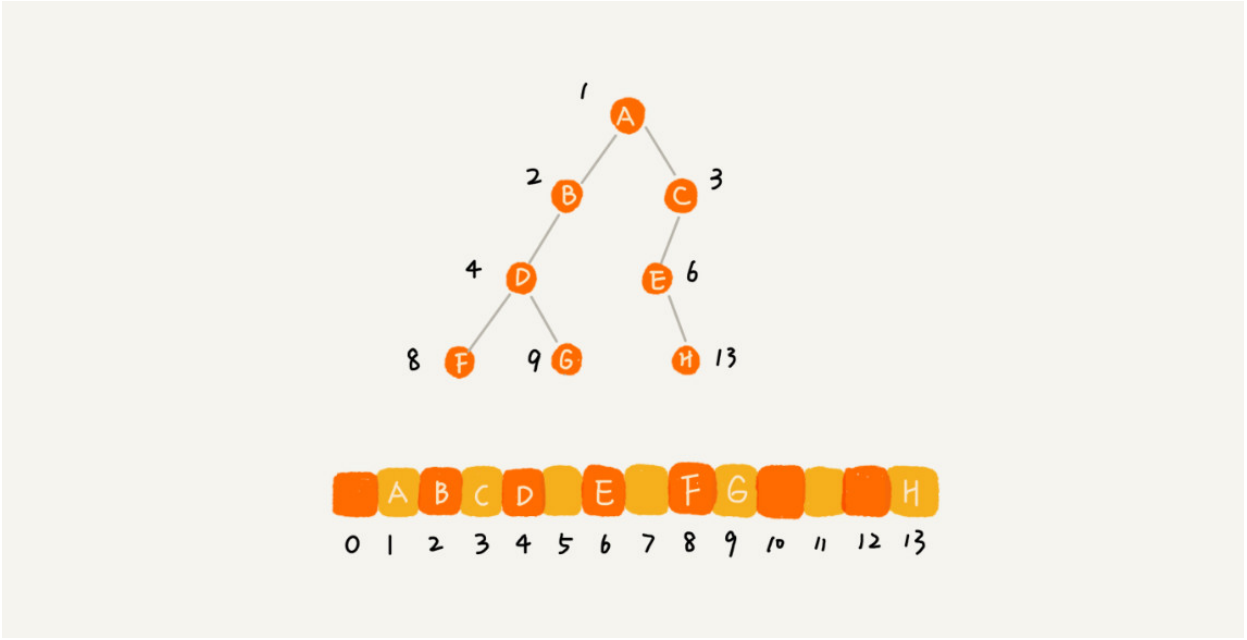
每个节点由3个字段，其中一个存储数据，另外两个是指向左右子节点的指针。我们只要拎住根节点，就可以通过左右子节点的指针，把整棵树都串起来。这种存储方式比较常用，大部分二叉树代码都是通过这种方式实现的。



②顺序存储

用数组来存储，对于完全二叉树，如果节点X存储在数组中的下标为i，那么它的左子节点的存储下标为2*i，右子节点的下标为2*i+1，反过来，下标i/2位置存储的就是该节点的父节点。注意，根节点存储在下标为1的位置。完全二叉树用数组来存储时最省内存的方式。



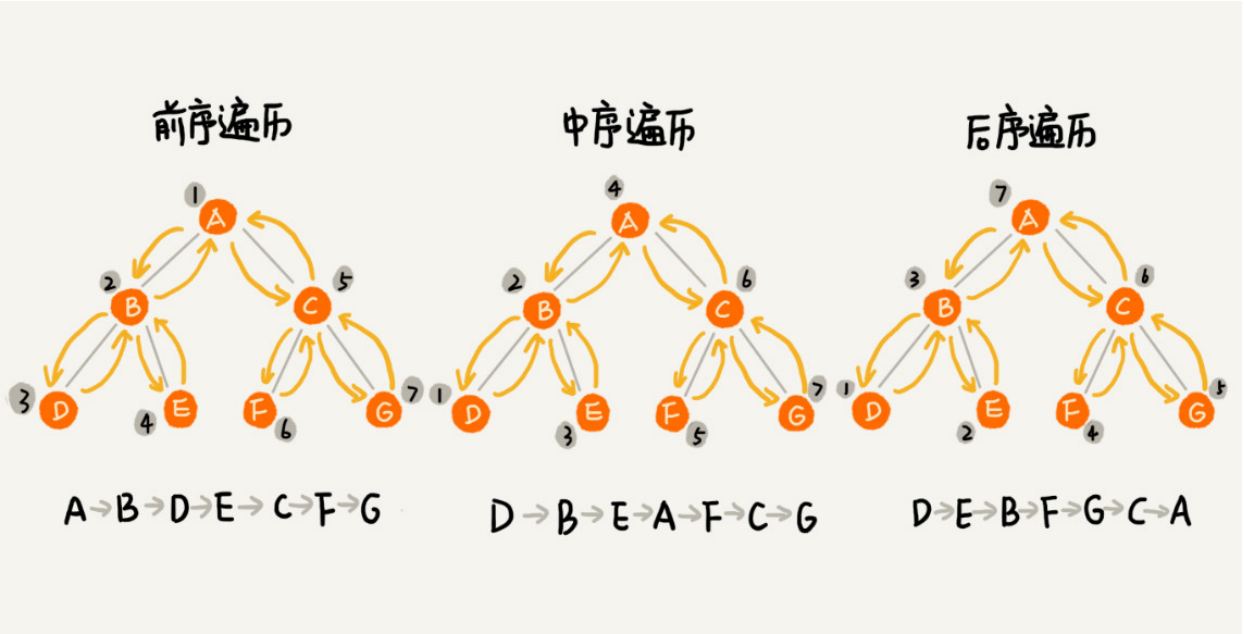


3.二叉树的遍历

①前序遍历：对于树中的任意节点来说，先打印这个节点，然后再打印它的左子树，最后打印它的右子树。

②中序遍历：对于树中的任意节点来说，先打印它的左子树，然后再打印它的本身，最后打印它的右子树。

③后序遍历：对于树中的任意节点来说，先打印它的左子树，然后再打印它的右子树，最后打印它本身。



前序遍历的递推公式：

`preOrder(r) = print r->preOrder(r->left)->preOrder(r->right)`

中序遍历的递推公式：

`inOrder(r) = inOrder(r->left)->print r->inOrder(r->right)`

后序遍历的递推公式：

`postOrder(r) = postOrder(r->left)->postOrder(r->right)->print r`

有了递推公式之后，将代码简单实现以下（有伪代码！！！！）

```
void preOrder(Node* root) {
  if (root == null) return;
  print root // 此处为伪代码，表示打印 root 节点
  preOrder(root->left);
  preOrder(root->right);
}

void inOrder(Node* root) {
  if (root == null) return;
  inOrder(root->left);
  print root // 此处为伪代码，表示打印 root 节点
  inOrder(root->right);
}

void postOrder(Node* root) {
  if (root == null) return;
  postOrder(root->left);
  postOrder(root->right);
  print root // 此处为伪代码，表示打印 root 节点
```

}

时间复杂度：

3种遍历方式中，每个节点最多会被访问2次，所以时间复杂度是O(n)。

三、思考

1.二叉树有哪几种存储方式？什么样的二叉树适合用数组来存储？

数组有链式存储和顺序存储，完全二叉树适合数组来存储，这样不会浪费内存空间，满二叉树也是一种特殊的完全二叉树

2.给定一组数据，比如1，3，5，6，9，10.你来算算，可以构建出多少种不同的二叉树？

确定两点：

1）n个数，即n个节点，能构造出多少种不同形态的树？

2）n个数，有多少种不同的排列？

当确定以上两点，将【1)的结果】乘以 【2)的结果】，即为最终的结果。

但是有一个注意的点： 如果n中有相等的数，产生的总排列数就不是n！了哟

【卡塔兰数】： https://en.wikipedia.org/wiki/Catalan_number

3.我们讲了三种二叉树的遍历方式，前、中、后序。实际上，还有另一种遍历方式，也就是按层遍历，你知道如何实现吗？

参照图的广度优先遍历

4.如何用循环实现二叉树的遍历？

极客时间文档： https://time.geekbang.org/column/article/67856