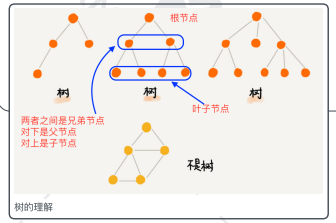
14.二叉树

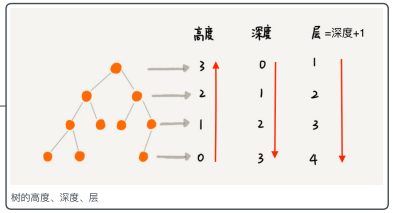
1. 概念与理解

常见的树类型

树的理解

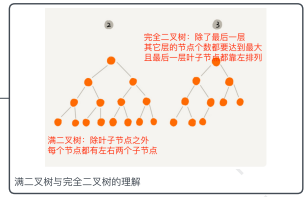


树的高度、深度、层



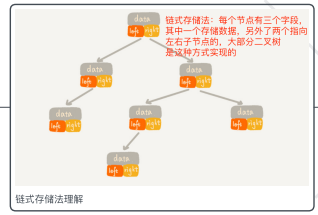
2. 二叉树类型和存储方式

2.1 二叉树类型
*满二叉树
*完全二叉树

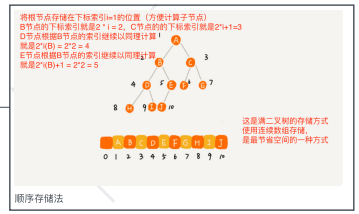


满二叉树与完全二叉树的理解

2.2 存储方式
*链式存储法
*顺序存储法



链式存储法理解



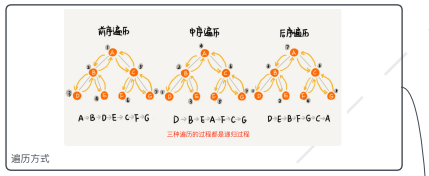
顺序存储法

3.二叉树遍历

遍历方式
*前序遍历
对于树中的任意节点，先打印自身，再打印左子树，最后打印右子树

*中序遍历
对于树中的任意节点，先打印左子树，再打印自身，最后打印右子树

*后续遍历
对于树中的任意节点，先打印左子树，再打印右子树，最后打印自身



遍历方式

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
```

```
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
}
```

遍历过程伪代码

4.遍历时间复杂度

从我前面画的前、中、后序遍历的顺序图，可以看出来，每个节点最多会被访问2次，所以遍历操作的时间复杂度，跟节点的个数 n 成正比，也就是说二叉树遍历的时间复杂度是 O(n)

二叉树
满二叉树的高度不会超过log2(n),高度用作很多复杂度的分析

二叉查找树
（支持动态数据集合的快速插入、删除、查找操作）
左子树<父节点<右子树

平衡二叉查找树
(主要讲红黑树)
为了解决二叉查找树因为动态更新导致的性能退化问题