Chapter 1

一． 数据结构的存储方式

\*数组（顺序存储）Array： 紧凑连续存储，可以随机访问，可通过index快速找到对应元素，相对节约存储空间。但是如果要扩容，需要重新分配，复制之前的数据过去，时间复杂度为O(n)；插入或者删除，需要搬移后面数据保持连续，时间复杂度为O(n)

\*链表（链式存储）Linked List:元素不连续，而是靠指针指向下一个元素的位置，所以不需要扩容；如果已知某一元素的前后驱，操作指针即可删除或插入新元素，时间复杂度O(1)。但无索引，不能随机访问，每个元素需要存储指向前后的指针，消耗更多空间。

二． 数据结构：Hash Table，Stack, Queue, Set, Tree, Graph

\*Queue 和 Stack 既可以用数组也可以用链表实现，用数组需要处理扩容缩容，用链表需要更多的内存空间储存节点指针

\*Graph: 邻接表就是用链表（省空间，但很多操作效率不如邻接链表）；邻接矩阵就是用二维数组（判断连通性迅速，可以进行矩阵运算，但图稀疏耗费空间）

\*Hash Table: 通过散列函数将Key映射到一个大的数组里。对于hash collision, 拉链法需要链表特性，操作简单，但需要额外空间存储指针；线性探查法需要数组特性，用于连续寻址，不需要额外内存，操作复杂。

\*Tree: 用Array实现的是Set,set是一个完全二叉树，不需要节点指针，操作简单；用linkedList实现就是常见Tree, 不是完全二叉树不适合用数组，但衍生出二叉搜索树BST,AVL Tree, 红黑树，区间树，B树等。

三． 数据结构的基本操作

遍历+访问：增删查改（高效）

线性（迭代for/while）和非线性(递归Recursive)

数组：迭代

Linked List: 迭代和递归都可以



Binary Tree: 递归左右节点



N叉树：递归每一个child root



Graph: graph是好几个N叉树的结合体，出现环就用Boolean数组visited做标记。

\*解题思路：

先刷二叉树：前序，中序，后序遍历

前序遍历：root 🡺 root.left 🡺 root.right 中左右

中序遍历：root.left 🡺 root 🡺 root.right 左中右

后序遍历：root.left 🡺 root.right 🡺 root 左右中

理解树，再进行专题：回溯，动态规划，分治等

Chapter 2

一． 二叉树的重要性

Quick Sort: 二叉树前序遍历（中左右）

对nums[lo…hi]排序：

1.找到一个分界点p

2. 通过交换元素，使nums[lo..p-1]<=nums[p],且nums[p+1..hi]都大于nums[p]

3. 递归地去nums[lo..p-1]和nums[p+1..hi]中寻找新的分界点

先构造分界点，再去左右子数组构造分界点，类似BT的前序遍历

Merge Sort：二叉树后序遍历 （右左中）

对nums[lo…hi]排序：

1.先对nums[lo..mid]排序

2. 对nums[mid+1..hi]排序

3. 将这两个有序的子数组合并

先对左右子数组排序，再合并（类似合并有序链表），类似BT的后序遍历框架，分治算法

二． 递归算法的秘诀

**key: 明确函数的【定义】是什么， 相信定义，利用定义推到最终结果，不能跳入递归的细节**

Example: count the number of nodes of BT?

定义：count(root)返回以root为根的树有多少节点

int count(TreeNode root){

//base case

if(root == null) return 0;

//自己加上子树的节点数就是整棵树的节点数

return 1+ count(root.left)+ count(root.right);

}

method:清楚当前root节点该做什么，根据函数定义递归调用子节点，让节点做相同的事情

三． 算法实践

**# 226 invert BT (翻转)**

🡺only need to switch the left and right node of each root

TreeNode invertTree(TreeNode root){

//base case

if (root ==null) return null;

/\*\*\*\*\*前序遍历位置\*\*\*\*\*/

//root 节点需要交换其左右节点

TreeNode temp = root.left;

root.left = root.right;

root.right= temp;

//让左右子节点继续翻转它们的子节点

invertTree(root.left);

invertTree(root.right);

return root;

}

！！二叉树难点就：将题目的要求细化成每一个节点需要做的事情

**#116填充每个节点的下一个右侧节点指针**

Populating next right pointers in each node

Given a perfect BT where all leaves are on the same level and every parent has two children.

Populate each next pointer to point to its next right node. if no next right node, the right next pointer should be set to null.

Initially, all next pointers are set to NULL.

🡺需要串联非同一个parent root下的nodes,所以只传入一个node的方法下迭代是不可行的。需要增加一个新的方法，传入两个节点：

//主函数

Node connect(Node root) {

if (root == null) return null;

connectTwoNode(root.left, root.right);

return root;

}

// 辅助函数

void connectTwoNode(Node node1, Node node2) {

if (node1 == null || node2 == null) {

return;

}

//\*\*\*\* 前序遍历位置 \*\*\*\*/

// 将传入的两个节点连接

node1.next = node2;

// 连接相同父节点的两个子节点

connectTwoNode(node1.left, node1.right);

connectTwoNode(node2.left, node2.right);

// 连接跨越父节点的两个子节点

connectTwoNode(node1.right, node2.left);

}

**#116 Flatten BT to a linked list**

Given the root of a BT, the return linked list should use the same TreeNode class where the right child pointer pointes to the next node in the list and the left child pointer is always null. And it should be in the same order as a preorder traversal of the binary tree.

**Input:** root = [1,2,5,3,4,null,6]

**Output:** [1,null,2,null,3,null,4,null,5,null,6]

🡺1. flat the left tree of the root

2. flat the right tree of the root

3. make the whole left tree as the root’s right

4. make the root’s left as null

5. the right flatten tree connect to the end of the right

public class flatten (TreeNode root){

//base case

if(root == null) return root;

flatten(root.left);

TreeNode leftTree = root.left;

flatten(root.right);

TreeNode rightTree = root.right;

root.left =null;

root.right = leftTree;

TreeNode p = root;

while(p.right !=null){

p=p.right;

}

p.right = rightTree;

}

**# 654 构造最大二叉树**

Given an int array with no duplicate to define a maximum BT

1. Create a root node whose value is the maximum value in nums.

2.Recursively build the left subtree on the subarray prefix to the left of the maximum value.

3.Recursively build the right subtree on the subarray suffix to the right of the maximum value.

🡺Logic is clear. The most important point is: to represent the subarray, we need to add parameters in the method. so we build another method: buildTree(int [] nums, int lo, int hi)

//main method

public TreeNode constructMaximumBinaryTree(int[] nums) {

return buildTree(nums, 0, nums.length-1);

}

//helper method

public TreeNode buildTree(int[] nums, int lo, int hi){

if(lo>hi){

return null;

}

//get the max value and the its index

int maxVal = nums[lo];

int maxIndex = lo;

for(int i = lo; i<=hi; i++){

if(nums[i]>maxVal){

maxVal = nums[i];

maxIndex = i;

}

}

TreeNode root = new TreeNode(maxVal);

root.left = buildTree(nums, lo, maxIndex-1);

root.right = buildTree(nums,maxIndex+1,hi);

return root;

}

**#** [**105. 从前序与中序遍历序列构造二叉树**](https://leetcode-cn.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/)

**Construct BT from preorder and inorder traversal**

根据BT的前序遍历与中序遍历构造二叉树。

注意:  
你可以假BT中没有重复的元素。

例如，

先序遍历preorder = [3,9,20,15,7] 中序遍历 inorder = [9,3,15,20,7]

返回如下的二叉树：[3, 9, null, null, 20, 15, 7]

🡺

根据preorder root=>left=>right， the first number in preorder array should be the root

inorder left=>root=>right, 根据找到的root, 将inorder分成左右子树, get root’s index

left is subarray: inorder[startIn…index-1], preorder[startPre +1,….]

right is subarray: inorder[index+1….endIn], pay attention to the startPre: 需要加上左侧子树上node的个数找startPre的位置： preStart+index-inStart+1

**#** [**106. 从中序与后序遍历序列构造二叉树**](https://leetcode-cn.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/)

**Construct BT from inorder and postorder traversal**

根据BT的中序遍历与后序遍历构造二叉树。

注意:  
你可以假BT中没有重复的元素。

例如，

Input: inorder = [9,3,15,20,7], postorder = [9,15,7,20,3]

Output: [3,9,20,null,null,15,7]

🡺类似的两题