Tree

101：比较简单。判断对称看左结点的左/右子树和右结点的右/左子树

Linked List

83： 去重常用map做标记，本题注意sorted这个条件，不需要用map，只需一个值即可。

92：

reverse链表，需要三个指针p, q, r遍历要翻转的部分。对链表进行实际的翻转只有一步q->next=p。p=q, q=r等，实际的链表并没有变，只是p、q指向了链表中的下一个位置，从而继续翻转。翻转完后p指向刚刚翻转的部分，q指向翻转部分的下一个。p是翻转后链表的开头，没有向后遍历的p\_m反而由开头变成了末尾，将第一段末尾指向p，将p\_m指向第三段即q（翻转部分是第二段），则完成了链表中一部分的翻转。

新链表节点赋初值：ListNode\* dummy\_head = new ListNode(-1);

Hash Table

136. 按位异或：运算符^，位相同则该位结果为0，不同为1。

Back Tracking

89：本题比较巧妙。观察到格雷码按只变1位排好序后，倒序并分别在高1位加1接到原来后面即排列成新的格雷码。1<<i指将1按位向左移i位。

93：

IP地址四个数，每个数在0-255之间，两位数和三位数不能以0开头

回溯法（Backtracking）：常与递归结合，分成几种情况减小规模，每种情况都先往后试，如果不成功再向另一条路试，注意返回时要在递归后删去已经假设的情况。

递归一般要新开一个helper函数，原题的函数参量太少。

递归先写以return跳出和结束递归的部分。注意当pos=s.size()-1的时候相当于最后一位还没有分配，pos=s.size()才是结束递归的条件。

Two Pointers

86. 多指针时，随时注意考虑p是否存在，以及p->next是否存在等。注意p->next的改变会实际改变链表流向，p的改变只改变指向的位置。dummy\_head最后返回的时候不要返回head而要返回dummy\_head->next。

Breadth-first Search

102：对树进行层序遍历（广度优先搜索），用队列q暂存每一层的结点。由于queue是先进先出，在循环开始前n=q.size()，以n>0作为循环的条件，可以使新加入的下一层结点不影响上一层结点读入path中。

103：reverse(vector.begin(), vector.end());

Depth-first Search

104：深度优先搜索常与递归结合。

108：Binary Search Tree，即左结点<根节点<右结点，用二分法和递归。注意新建结点的时候要开空间，TreeNode\* root = new TreeNode();

109：与108类似，链表的话通过fast和slow指针来找到中间结点。

110：用-1这个不符合条件的深度来表示false，并且递归中有一层的return是-1那么其上所有层都return -1

111：注意最小深度必须从根节点到叶节点，中间结点不算，因此如果某结点的l或r为0，则递归返回r+1或l+1，即另一侧的深度。