```
public class ListNode {
public var val: Int
public var next: ListNode?
public init(_ val: Int) {
self.val = val
self.next = nil
}
}

public class TreeNode {
public var val: Int
public var left: TreeNode?
public var right: TreeNode?
public init(_ val: Int) {
self.val = val
self.left = nil
self.right = nil
}
}

// 题目选自牛客网《名企高频面试题》https://www.nowcoder.com/ta/job-code-high-rd?company=665
// 企业选择<字节跳动>、职位选择<客户端>
// 总共79题

/**
1. 反转链表
2. 排序数组
3. LRU缓存机制
4. 二叉树的前序遍历
4. 二叉树的中序遍历
4. 二叉树的后序遍历
5. 最小的k个数
6. 二叉树的层序遍历
7. 数组中的第k个最大元素
8. 两数之和
9. 合并两个有序链表
10. 用两个栈实现队列、用栈实现队列
11. 青蛙跳台阶问题、爬楼梯
```

12. 最大子序和
13. k个一组翻转链表
14. 无重复字符的最长子串
15. 环形链表
16. 合并两个有序数组
17. 环形链表 II
18. 有效的括号
19. 删除链表的倒数第 N 个结点
20. 字符串相加
21. 二叉树的锯齿形层次遍历
22. 最长公共子序列（跳过）
23. 相交链表
24. 两数相加
25. 二叉树的最近公共祖先
26. 反转字符串
27. 螺旋矩阵（跳过）
28. 斐波那契数
29. 最长回文子串（跳过）
30. 从前序与中序遍历序列构造二叉树、重建二叉树
31. 三数之和
32. 最长上升子序列
33. 搜索旋转排序数组
34. x 的平方根
35. 最小栈
36. 买卖股票的最佳时机
37. 合并K个升序链表
38. 全排列
39. 盛最多水的容器
40. 二叉树的右视图
41. 岛屿问题（跳过）
42. 二叉树的最大深度
43. 排序链表（跳过）
44. 平衡二叉树（跳过）
45. 数组中出现次数超过一半的数字
46. 表达式求值（跳过）
47. 回文链表
48. 最小编辑代价（跳过）
49. 路径总和 II、二叉树中和为某一值的路径
50. 合并区间（跳过）

51. 在两个长度相等的排序数组中找到上中位数（跳过）
52. 判断该二叉树是否为搜索二叉树和完全二叉树（跳过）
53. 二叉树的完全性检验（完全二叉树）
54. 不同路径
55. 二叉树的所有路径、删除链表中重复的节点
56. 整数反转
57. 矩阵元素查找（跳过）
58. 缺失的第一个正数
59. 0～n-1中缺失的数字（跳过）
60. 将字符串转化为整数（跳过）
61. 对称二叉树
62. LFU缓存结构设计（跳过）
63. 有重复项数字的所有排列（跳过）
64. 路径总和
68. 二叉树的直径
71. 链表中倒数第k个节点
72. 寻找峰值

1. 回文数
2. 翻转字符串里的单词
3. 颠倒二进制位
4. 只出现一次的数字
5. 长度最小的子数组
6. 旋转图像
7. 最长连续序列
8. 零钱兑换
9. 买卖股票的最佳时机 II
10. 二叉树的镜像、翻转二叉树
11. 字符串解码 */

/* 1-- /
/
题号：206. 反转链表
NC78
出现频率：189
难度：简单
重复出现：leetcode

输入: 1->2->3->4->5->NULL
输出: 5->4->3->2->1->NULL

*/

func reverseList(_ head: ListNode?) -> ListNode? {

```
var newHead: ListNode? = nil
var cur = head

while cur != nil {
    // 1 记录即将断开的节点
    let tmp = cur!.next
    // 2 翻转
    cur!.next = newHead
    // 3 重制
    newHead = cur!
    cur = tmp
}
return newHead
```

}

/* 2-- /
/
题号：912. 排序数组
NC140
出现频率：141
难度：中等

输入：nums = [5,2,3,1]
输出：[1,2,3,5]
*/

func sortArray(_ nums: [Int]) -> [Int] {
var arr = nums
quickSort(&arr, 0, arr.count - 1)
return arr
}

private func quickSort(_ nums: inout [Int], _ left: Int, _ right: Int) {
if left >= right {
return
}

```
 var l = left;
 var r = right;

 let key = nums[l];

 while l < r {

     while l<r && nums[r] >= key{
         r -= 1;
     }

     nums[l] = nums[r];

     while l<r && nums[l] <= key{
         l += 1;
     }

     nums[r] = nums[l]
 }

 nums[l] = key;
 quickSort(&nums, left, l-1)
 quickSort(&nums, l+1, right)
```

}

/* 3-- */

/*
题号：146.LRU缓存机制
NC93
出现频率：134
难度：中等
*/

// 1.创建双向链表类
class DLikedNode {
var pre: DLikedNode?
var next: DLikedNode?
var value: Int?
var key: Int?

```
 // 2. 构造函数
```

```
    init(pre: DLikedNode? = nil, next: DLikedNode? = nil, value: Int, key: Int?
    = nil) {
        self.pre = pre
        self.next = next
        self.value = value
        self.key = key
    }
```

}

class LRUCache {

```
    // 3. 属性构建
    var capacity = 0
    var count = 0
    let first = DLikedNode.init(value: -1)
    let last = DLikedNode.init(value: -1)

    var hash = Dictionary<Int, DLikedNode>()

    //4. 实现构造函数
    init(_ capacity: Int) {
        self.capacity = capacity
        first.next = last
        first.pre = nil

        last.pre = first
        last.next = nil
    }

    // 7
    // 获取
    func get(_ key: Int) -> Int {
        if let node = hash[key] {
            moveToHead(node: node)
            return node.value!
        } else {
            return -1
        }
    }

    // 8
    // 写入
    func put(_ key: Int, _ value: Int) {

    }
```

```
    // 6
    // 删除尾节点
    func removeLastNode() {
        //移除倒数第二个node
        let theNode = last.pre
        theNode?.pre?.next = last
        last.pre = theNode?.pre

        count -= 1
        hash[theNode!.key!] = nil

        theNode?.pre = nil
        theNode?.next = nil
    }

    // 5
    // 移动到头节点
    func moveToHead(node: DLikedNode) {

        //取出节点
        let pre = node.pre
        let next = node.next
        pre?.next = next
        next?.pre = pre

        //置入头节点
        node.pre = first
        node.next = first.next

        first.next = node
        node.next?.pre = node
    }
```

}

/* 4---------- */

// 实现二叉树先序，中序和后序遍历
// NC45
// 出现频率：120

/*
题号：144.二叉树的前序遍历
难度：简单
*/
func preorderTraversal(_ root: TreeNode?) -> [Int] {

var result: [Int] = []

```
guard let root = root else { return result }

result.append(root.val)
result += preorderTraversal(root.left)
result += preorderTraversal(root.right)

return result
```

}

func preorderTraversal2(_ root: TreeNode?) -> [Int] {
var result: [Int] = []
guard let root = root else { return result }
var queue = [root]

```
while !queue.isEmpty {
    let node = queue.popLast()
    result.append(node!.val)

    if node?.right != nil {queue.append(node!.right!)}
    if node?.left != nil {queue.append(node!.left!)}
}
return result
```

}

/*
题号：94. 二叉树的中序遍历
难度：中等

给定一个二叉树的根节点 root ， 返回它的 中序 遍历。
输入：root = [1,null,2,3]
输出：[1,3,2]

*/

func inorderTraversal(_ root: TreeNode?) -> [Int] {
var result: [Int] = []
guard let root = root else { return result }
result += inorderTraversal(root.left)

result.append(root.val)
result += inorderTraversal(root.right)
return result
}

func inorderTraversal2(_ root: TreeNode?) -> [Int] {
var result: [Int] = []
var queue = TreeNode
var cur: TreeNode? = root

```
while cur != nil || !queue.isEmpty {
    while cur != nil {
        queue.append(cur!)
        cur = cur!.left
    }

    cur = queue.popLast()
    result.append(cur!.val)
    cur = cur?.right
}
return result
```

}

/*
题号：145. 二叉树的后序遍历
难度：困难
*/

func postorderTraversal(_ root: TreeNode?) -> [Int] {
var result: [Int] = []
guard let root = root else { return result }
result += postorderTraversal(root.left)
result += postorderTraversal(root.right)
result.append(root.val)
return result
}

/* 5-- /
/
题号：剑指 Offer 40. 最小的k个数
NC119

难度：简单
出现频率：82

输入：arr = [3,2,1], k = 2
输出：[1,2] 或者 [2,1]
*/

func getLeastNumbers(_ arr: [Int], _ k: Int) -> [Int] {
var nums = arr
quickSort(&nums, 0, nums.count - 1)
// 2. 获取倒数第k个元素
return Array(nums[0..<k])
}

/* 6-- */

/*
题号：102. 二叉树的层序遍历
NC15
出现频率：62
难度：中等
*/

func levelOrder(_ root: TreeNode?) -> [[Int]] {
guard let root = root else { return [] }

```
var result = [[Int]]()
var queue: [TreeNode] = [root]

while !queue.isEmpty {
    var current = [Int]()

    for _ in 0 ..< queue.count {
        let node = queue.removeFirst()
        current.append(node.val)

        if node.left != nil {
            queue.append(node.left!)
        }

        if node.right != nil {
            queue.append(node.right!)
        }
    }
```

```
        result.append(current)
    }
    return result
```

}

/* 7-- */

/*
题号：215.数组中的第k个最大元素
NC88
出现频率：62
难度：中等

在未排序的数组中找到第 k 个最大的元素。请注意，你需要找的是数组排序后的第 k 个最大的元素，而不是第 k 个不同的元素

输入: [3,2,1,5,6,4] 和 k = 2
输出: 5
*/

// 小码哥快速排序讲解: https://juejin.im/post/6844904095250120718

func findKthLargest(_ nums: [Int], _ k: Int) -> Int {

```
    var arr = nums
    quickSort(&arr, 0, arr.count - 1)
    return arr[arr.count - k]
```

}

/* 8-- */

/*
题号：1. 两数之和
NC61
出现频率：59
难度：简单

给定一个整数数组 nums 和一个目标值 target，请你在该数组中找出和为目标值的那 两个 整数，并返回他们的数组下标。
你可以假设每种输入只会对应一个答案。但是，数组中同一个元素不能使用两遍。

给定 nums = [2, 7, 11, 15], target = 9
因为 nums[0] + nums[1] = 2 + 7 = 9
所以返回 [0, 1]
*/

func twoSum(_ nums: [Int], _ target: Int) -> [Int] {
var dic = Dictionary()

```
for (i, v) in nums.enumerated() {
    if dic[target - v] != nil{
        return [i, dic[target - v]!]
    }
    dic[v] = i
}
return [-1,-1]
```

}

/* 9-- */

/*
题号：21.合并两个有序链表
NC33
出现频率：57
难度：简单

将两个升序链表合并为一个新的 升序 链表并返回。新链表是通过拼接给定的两个链表的所有节点组成的
输入：1->2->4, 1->3->4
输出：1->1->2->3->4->4

*/

// 双指针法
func mergeTwoLists(_ l1: ListNode?, _ l2: ListNode?) -> ListNode? {
var l1 = l1
var l2 = l2
var cur: ListNode?
// 1. 创建虚拟头节点。
let newList = ListNode.init(-1)
cur = newList
// 2. 当l1和l2不为空时，比较大小，拼接。

```
while l1 != nil && l2 != nil {
if l1!.val <= l2!.val {
cur!.next = l1
l1 = l1!.next
} else {
cur!.next = l2
l2 = l2!.next
}
cur = cur!.next
}
// 3. 当l1或l2为空时，拼接另一个链表剩余元素。
cur!.next = l1 ?? l2
return newList.next
}

// 递归法
func mergeTwoLists2(_ l1: ListNode?, _ l2: ListNode?) -> ListNode? {
if (l1 == nil) {
return l2
} else if (l2 == nil) {
return l1
} else if (l1!.val < l2!.val) {
l1!.next = mergeTwoLists(l1!.next, l2)
return l1
} else {
l2!.next = mergeTwoLists(l1, l2!.next)
return l2
}
}

/* 10---------------------------------------------- */

/*
题号：剑指 Offer 09. 用两个栈实现队列
NC76
出现频率：57
难度：简单

用两个栈实现一个队列。队列的声明如下，请实现它的两个函数 appendTail 和 deleteHead ，分别完成在队列尾部插入整数和在队列头部删除整数的功能。(若队列中没有元素，deleteHead 操
```

作返回 -1)

*/

var stack1 = Int
var stack2 = Int

func appendTail(_ value: Int) {
stack1.append(value)
}

func deleteHead() -> Int {
if stack2.isEmpty {
while !stack1.isEmpty {
stack2.append(stack1.popLast()!) //不能removelast，此函数不会改变stack1的count
}
}
return stack2.isEmpty ? -1 : stack2.popLast()! //记住-1
}

/*
题号：232. 用栈实现队列
难度：简单
*/
class MyQueue {

```
var stack1: [Int]
var stack2: [Int]

/** Initialize your data structure here. */
init() {
    stack1 = [Int]()
    stack2 = [Int]()
}

/** Push element x to the back of queue. */
func push(_ x: Int) {
    stack1.append(x)
}

/** Removes the element from in front of queue and returns that element. */
func pop() -> Int {
    if stack2.isEmpty {
        while !stack1.isEmpty {
```

```
                stack2.append(stack1.popLast()!)
            }
        }
        return stack2.isEmpty ? -1 : stack2.popLast()!
    }

    /** Get the front element. */
    func peek() -> Int {
        if !stack2.isEmpty {
            return stack2.last!
        }
        return stack1.isEmpty ? -1 : stack1.first!
    }

    /** Returns whether the queue is empty. */
    func empty() -> Bool {
        return stack1.isEmpty && stack2.isEmpty
    }
```

}

/* 11-- */

/*
题号：剑指 Offer 10- II. 青蛙跳台阶问题
NC68
出现频率：55
难度：简单

一只青蛙一次可以跳上1级台阶，也可以跳上2级台阶。求该青蛙跳上一个 n 级的台阶总共有多少种跳法。
答案需要取模 1e9+7（1000000007），如计算初始结果为：1000000008，请返回 1。

示例 1：
输入：n = 2
输出：2

示例 2：
输入：n = 7
输出：21

*/

func numWays(_ n: Int) -> Int {

var a = 1
var b = 1
var sum = 0

```
for _ in 0..<n {
    sum = (a + b) % 1000000007
    (a, b) = (b, sum)
}
return a
```

}

func numWays2(_ n: Int) -> Int {
if n < 2 { return 1 }

```
var nums = [Int].init(repeating: 0, count: n + 1)
nums[0] = 1
nums[1] = 1

for i in 2...n {
    nums[i] = (nums[i - 1] + nums[i - 2]) % 1000000007
}
return nums[n]
```

}

/*
题号：70. 爬楼梯
出现频率：2
难度：简单

假设你正在爬楼梯。需要 n 阶你才能到达楼顶。
每次你可以爬 1 或 2 个台阶。你有多少种不同的方法可以爬到楼顶呢?
注意：给定 n 是一个正整数。
*/

func climbStairs(_ n: Int) -> Int {
var dp = Array.init(repeating: 0, count: n + 1)
dp[0] = 1
dp[1] = 1

```
    for i in 1..<n {
        dp[i+1] = dp[i] + dp[i-1]
    }
    return dp[n]
```

}

// 空间复杂度 O(1)
func climbStairs2(_ n: Int) -> Int {
var a = 1
var b = 1
var sum = 0

```
    for _ in 0..<n {
        sum = a + b
        (a, b) = (b, sum)
    }
    return a
```

}

/* 12-- */

/*
题号：53.最大子序和
NC19
出现频率：52
难度：简单

给定一个整数数组 nums ，找到一个具有最大和的连续子数组（子数组最少包含一个元素），返回其最大和。

输入: [-2,1,-3,4,-1,2,1,-5,4]
输出: 6
解释: 连续子数组 [4,-1,2,1] 的和最大，为 6。

*/

func maxSubArray(_ nums: [Int]) -> Int {

```
    // 注意：result的初始值为第一个元素。
    // 1. 创建变量
```

```
    var result = nums[0] // result：连续子数组的最大和
    var sum = 0 // sum：遍历数组元素和

    for i in nums {
        // 2. 如果 sum > 0，则说明 sum 对结果有增益效果，则 sum 保留并加上当前遍历数字
        if sum > 0 {
            sum += i
        }
        // 3. 如果 sum <= 0，则说明 sum 对结果无增益效果，需要舍弃，则 sum 直接更新为当前
    遍历数字
        else {
            sum = i
        }
        // 3. 每次比较 sum 和 ans的大小，将最大值置为result，遍历结束返回结果
        result = result >= sum ? result : sum
    }
    return result
```

}

/* 13---------- */

/*
题号：25.k个一组翻转链表
NC50
出现频率：52
难度：困难
*/

//pre、end、start、next四个游标，移动到group头尾 -> 头尾断开 -> 反转 -> 合并 -> 准备next group

func reverseKGroup(_ head: ListNode?, _ k: Int) -> ListNode? {
//1. 虚拟头节点
let newHead = ListNode.init(-1)
newHead.next = head

```
    //2. pre end
    var pre: ListNode? = newHead
    var end: ListNode? = newHead

    //3. 位移k
    while end != nil {
        for _ in 0..<k {
```

```
            if end != nil {
                end = end!.next
            }
        }
        if end == nil { break }

        //4. start next
        let start = pre?.next
        let next = end?.next

        //5. 断开
        end?.next = nil

        //6. 反转
        pre?.next = reverse(head: start!)

        //7. 合并
        start?.next = next

        //8. 重制
        pre = start
        end = start
    }

    return newHead.next
```

}

// 反转链表
func reverse(head: ListNode) -> ListNode{
var newHead: ListNode? = nil
var cur: ListNode? = head

```
    while cur != nil {
        // 5.1 记录即将断开的节点
        let next = cur!.next
        // 5.2 翻转
        cur!.next = newHead
        // 5.3 重制
        newHead = cur
        cur = next
    }
    return newHead! //反转后的头节点
```

}

/* 14-- */

/*
题号：3. 无重复字符的最长子串
NC41
出现频率：52
难度：中等

输入: "abcabcbb"
输出: 3
解释: 因为无重复字符的最长子串是 "abc"，所以其长度为 3。
*/

func lengthOfLongestSubstring(_ s: String) -> Int {
var max = 0
var arr = Character

```
for i in s {
    while arr.contains(i) {
        arr.removeFirst()
    }
    arr.append(i)

    max = arr.count > max ? arr.count : max
}
return max
```

}

/* 15-- */

/*
题号：141. 环形链表
NC4
出现频率：50
难度：简单

给定一个链表，判断链表中是否有环。
*/
func hasCycle(_ head: ListNode?) -> Bool {

```
let newHead = ListNode.init(-1)
```

```
    newHead.next = head

    // 1. 声明快慢指针
    var slow: ListNode? = newHead
    var fast: ListNode? = newHead

    // 2. 快慢指针开始移动
    while fast != nil {
        slow = slow?.next
        fast = fast?.next?.next

        // 3. 找到环，重置慢指针
        if slow === fast {
            return true
        }
    }
    return false
```

}

/* 16-- */

/*
题号：88. 合并两个有序数组
NC22
出现频率：49
难度：简单

给你两个有序整数数组 nums1 和 nums2，请你将 nums2 合并到 nums1 中，使 nums1 成为一个有序数组。

说明：

初始化 nums1 和 nums2 的元素数量分别为 m 和 n 。
你可以假设 nums1 有足够的空间（空间大小大于或等于 m + n）来保存 nums2 中的元素。

示例：

输入：
nums1 = [1,2,3,0,0,0], m = 3
nums2 = [2,5,6], n = 3

输出：[1,2,2,3,5,6]
*/

func merge(_ nums1: inout [Int], _ m: Int, _ nums2: [Int], _ n: Int) {

```
var cur = m + n - 1
var i1 = m - 1
var i2 = n - 1

while i2 >= 0 {
    if i1 >= 0 && nums1[i1] >= nums2[i2] {
        nums1[cur] = nums1[i1]
        cur -= 1
        i1 -= 1
    } else {
        nums1[cur] = nums2[i2]
        cur -= 1
        i2 -= 1
    }
}
```

}

/* 17-- */

/*
题号：142. 环形链表 II
NC3
出现频率：46
难度：中等

给定一个链表，返回链表开始入环的第一个节点。 如果链表无环，则返回 null。
*/

func detectCycle(_ head: ListNode?) -> ListNode? {
let newHead = ListNode.init(-1)
newHead.next = head

```
var fast: ListNode? = newHead
var slow: ListNode? = newHead

while fast?.next?.next != nil {
    fast = fast?.next?.next
    slow = slow?.next

    if fast === slow {
        fast = newHead
```

```
            while fast !== slow {
                fast = fast?.next
                slow = slow?.next
            }
            return slow
        }
    }

    return nil
```

}

/* 18-- */

/*
20. 有效的括号
NC52
出现频率：46
难度：中等

给定一个只包括 '('，')'，'{'，'}'，'['，']' 的字符串 s ，判断字符串是否有效。

有效字符串需满足：

左括号必须用相同类型的右括号闭合。
左括号必须以正确的顺序闭合。
*/

func isValid(_ s: String) -> Bool {
// 利用堆栈 思想
if s.count % 2 != 0 { return false }

```
    var tempArr = [Character]()
    for c in s {
        switch c {
        case "(","{","[": tempArr.append(c)
        case ")": if tempArr.popLast() != "("  { return false }
        case "}": if tempArr.popLast() != "{"  { return false }
        case "]": if tempArr.popLast() != "["  { return false }
        default: break
        }
    }
    return tempArr.count == 0 ? true : false
```

}

/* 19-- */

/*
19. 删除链表的倒数第 N 个结点
NC53
出现频率：39
难度：中等

给你一个链表，删除链表的倒数第 n 个结点，并且返回链表的头结点。

输入：head = [1,2,3,4,5], n = 2
输出：[1,2,3,5]
*/

func removeNthFromEnd(_ head: ListNode?, _ n: Int) -> ListNode? {

```
let newHead = ListNode.init(-1)
newHead.next = head

var fast: ListNode? = newHead
var slow: ListNode? = newHead

for _ in 0..<n {
    fast = fast?.next
}

while fast?.next != nil {
    fast = fast?.next
    slow = slow?.next
}

slow?.next = slow?.next?.next
return newHead.next
```

}

/* 20-- */

/*
415. 字符串相加
NC1 大数相加
出现频率：39

难度：中等

给定两个字符串形式的非负整数 num1 和num2 ， 计算它们的和。
*/
// zero.asciiValue == 48

```
func addStrings(_ num1: String, _ num2: String) -> String {
let num1Array = Character
let num2Array = Character
var res = ""
var i = num1.count - 1
var j = num2.count - 1
var carry: UInt8 = 0
let zero: Character = "0"
while i>=0 || j>=0 {
let n1 = i >= 0 ? num1Array[i].asciiValue! - zero.asciiValue! : 0
let n2 = j >= 0 ? num2Array[j].asciiValue! - zero.asciiValue! : 0
let temp = n1 + n2 + carry
carry = temp / 10
res = String(temp % 10) + res
i -= 1
j -= 1
}
if carry == 1 {
res = "1" + res
}
return res
}
```

/* 21-- */

/*
题号：103. 二叉树的锯齿形层次遍历
NC14
出现频率：2
难度：中等
*/

/*
题号：剑指 Offer 32 - III. 从上到下打印二叉树 III

出现频率：37
难度：中等
*/

func zigzagLevelOrder(_ root: TreeNode?) -> [[Int]] {

```
//1. 空判断
guard let root = root else { return [[Int]]() }

var result = [[Int]]()
var queue = [root]
var isZ = false

//2. 创建队列，加入root
while !queue.isEmpty {
    var cur = [Int]()
    //3. 遍历一次队列
    for _ in 0..<queue.count {
        let node = queue.removeFirst()
        //4. 将队列的值保存
        cur.append(node.val)
        //5. 队列中添加新值
        if node.left != nil {queue.append(node.left!)}
        if node.right != nil {queue.append(node.right!)}
    }
    if isZ { cur.reverse() }
    result.append(cur)
    isZ = !isZ
}
return result
```

}

/* 22 跳过 -- */

/*
题号：1143. 最长公共子序列
NC127
出现频率：37
难度：中等

给定两个字符串 text1 和 text2，返回这两个字符串的最长 公共子序列 的长度。如果不存在 公共子序列 ，返回 0 。

一个字符串的 子序列 是指这样一个新的字符串：它是由原字符串在不改变字符的相对顺序的情

况下删除某些字符（也可以不删除任何字符）后组成的新字符串。

例如，"ace" 是 "abcde" 的子序列，但 "aec" 不是 "abcde" 的子序列。
两个字符串的 公共子序列 是这两个字符串所共同拥有的子序列。
*/

```
func longestCommonSubsequence(_ text1: String, _ text2: String) -> Int {
let n1 = text1.count
let n2 = text2.count
let t1 = Array(text1)
let t2 = Array(text2)
var dp = Array(repeating: Array(repeating: 0, count: n2 + 1), count: n1 + 1)
for i in 1...n1 {
for j in 1...n2 {
let c1 = t1[i-1]
let c2 = t2[j-1]
if c1 == c2 {
dp[i][j] = dp[i-1][j-1] + 1
}else{
dp[i][j] = max(dp[i-1][j], dp[i][j-1])
}
}
}
return dp[n1][n2]
}
```

/* 23-- */

/*
题号：160.相交链表
NC66
出现频率：33
难度：简单

编写一个程序，找到两个单链表相交的起始节点。
*/

/*
题号：剑指 Offer 52. 两个链表的第一个公共节点
出现频率：1
难度：简单

*/

func getIntersectionNode(_ headA: ListNode?, _ headB: ListNode?) -> ListNode? {
var A = headA
var B = headB
// 1. A和B要么相等，要么同时为nil，=表示对象相等。
while A ! B {
A = A == nil ? headB : A!.next
B = B == nil ? headA : B!.next
}
return A
}

/* 24-- */

/*
题号：2. 两数相加
NC40
出现频率：32
难度：中等

给出两个 非空 的链表用来表示两个非负的整数。其中，它们各自的位数是按照 逆序 的方式存储的，并且它们的每个节点只能存储 一位 数字。
如果，我们将这两个数相加起来，则会返回一个新的链表来表示它们的和。
您可以假设除了数字 0 之外，这两个数都不会以 0 开头。

示例：

输入：(2 -> 4 -> 3) + (5 -> 6 -> 4)
输出：7 -> 0 -> 8
原因：342 + 465 = 807

*/

func addTwoNumbers(_ l1: ListNode?, _ l2: ListNode?) -> ListNode? {
var l1 = l1
var l2 = l2
var needAdd = 0

```
let newHead = ListNode.init(-1)
var cur = newHead
```

```
    while l1 != nil || l2 != nil {
        let new = (l1?.val ?? 0) + (l2?.val ?? 0) + needAdd
        needAdd = new / 10
        cur.next = ListNode.init(new % 10)

        cur = cur.next!
        l1 = l1?.next
        l2 = l2?.next
    }

    if needAdd == 1 {
        cur.next = ListNode.init(1)
    }

    return newHead.next
```

}

/* 25-- */

/*
题号：236. 二叉树的最近公共祖先
NC102
出现频率：32
难度：中等

输入: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1
输出: 3
解释: 节点 5 和节点 1 的最近公共祖先是节点 3。

*/

/*
题解：https://leetcode-cn.com/problems/lowest-common-ancestor-of-a-binary-tree/solution/236-er-cha-shu-de-zui-jin-gong-gong-zu-xian-hou-xu/
解题思路如下：
1、当root为空时，返回nil
2、如果root的值和p或者q的值相等时，直接返回root
3、递归左右子树，用left和right表示递归求出的结果
4、如果left是空，说明p和q节点都不在左子树，那么结果就在右子树，返回right
5、如果right是空，说明p和q节点都不在右子树，那么结果就在左子树，返回left
6、如果left和right都不为空，说明p和q节点分别在左右子树，那么结果就是root
*/

func lowestCommonAncestor(_ root: TreeNode?, _ p: TreeNode?, _ q: TreeNode?) ->
TreeNode? {
if root == nil || root === p || root === q { return root }

```
let left = lowestCommonAncestor(root?.left, p, q)
let right = lowestCommonAncestor(root?.right, p, q)

if left == nil { return right } //left为空 || left & right 都为空
if right == nil { return left } //right为空
return root // 公共祖先
```

}

/* 26---------- */

/*
题号：344.反转字符串
NC103
出现频率：32
难度：简单

输入：["h","e","l","l","o"]
输出：["o","l","l","e","h"]
*/

// 双指针，原地交换
func reverseString(_ s: inout [Character]) {
var l = 0
var r = s.count - 1

```
while l < r {
    let cur = s[r]
    s[r] = s[l]
    s[l] = cur

    l += 1
    r -= 1
}
```

}

/* 27 跳过---------- */

/*
题号：54. 螺旋矩阵
NC38
出现频率：31
难度：中等

给你一个 m 行 n 列的矩阵 matrix ， 请按照 顺时针螺旋顺序 ， 返回矩阵中的所有元素。
输入：matrix = [[1,2,3],[4,5,6],[7,8,9]]
输出：[1,2,3,6,9,8,7,4,5]
*/

/* 28-- */

/*
题号：509. 斐波那契数
NC65
出现频率：31
难度：中等

斐波那契数，通常用 F(n) 表示，形成的序列称为 斐波那契数列 。该数列由 0 和 1 开始，后面的每一项数字都是前面两项数字的和。也就是：
*/

func fib(_ n: Int) -> Int {
if n == 0 {return 0}

```
var arr = [Int](repeating: 0, count: n+1)
arr[0] = 0
arr[1] = 1

for i in 2..<arr.count{
    arr[i] = arr[i-1] + arr[i-2]
}
return arr[n]
```

}

func fib2(_ n: Int) -> Int {
var a = 0
var b = 1
var sum = 0
for _ in 0..<n {

```
sum = a + b
(a,b) = (b,sum)
}
return a
}

/* 29 跳过---------------------------------------------- */

/*
题号：5. 最长回文子串
NC17
出现频率：30
难度：中等

给你一个字符串 s，找到 s 中最长的回文子串。

输入：s = "babad"
输出："bab"
解释："aba" 同样是符合题意的答案。
*/

//中心拓展
func longestPalindrome(_ s: String) -> String {
if s.isEmpty {
return ""
}
var start = 0
var end = 0
let chars = Character
for (i,_) in chars.enumerated() {
// 当 回文子串 长度为奇数；例如“aba”
let length1 = helper(chars: chars, left: i, right: i)
// 当 回文子串 长度为偶数；例如“abba”
let length2 = helper(chars: chars, left: i, right: i + 1)
let length = max(length1, length2)
// 判断是否大于之前的最大长度 如果大于 则更新
if length > end - start + 1 {
start = i - (length - 1) / 2
end = i + length / 2
}
```

```
}
return String(chars[start...end])
}

/*
辅助方法 主要是为了计算出 回文子串 长度
*/
func helper(chars: [Character], left: Int, right: Int) -> Int {
var l = left, r = right
while l >= 0 && r < chars.count && chars[l] == chars[r] {
l -= 1
r += 1
}
// 这里是 -1 而不是 +1 因为在满足边界条件时 l和r都多+1和-1了一次；所以实际公式是 r - l + 1 - 2
return r - l - 1
}

/* 30---------------------------------------------- */

/*
题号：105. 从前序与中序遍历序列构造二叉树
NC12
出现频率：29
难度：中等

根据一棵树的前序遍历与中序遍历构造二叉树。

例如，给出

前序遍历 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 20
/ \
15 7
```

题解： https://leetcode-cn.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/solution/cong-qian-xu-yu-zhong-xu-bian-li-xu-lie-gou-zao-9/
*/

/*
题号：剑指 Offer 07. 重建二叉树
出现频率：1
难度：中等
*/

var indexMap: [Int: Int] = [:]
// 构造哈希映射，帮助我们快速定位根节点
func buildTree(_ preorder: [Int], _ inorder: [Int]) -> TreeNode? {
for (index, value) in inorder.enumerated() {
indexMap[value] = index
}

```
return myBuildTree(preorder, inorder, 0, preorder.count - 1, 0)
```

}

func myBuildTree(_ preorder: [Int], _ inorder: [Int], _ preleft: Int, _ preright: Int, _ inleft: Int) -> TreeNode? {

```
guard preleft <= preright else {
    return nil
}

// 1.1 前序遍历中的第一个节点就是根节点
let preRootIndex = preleft
// 1.2 在中序遍历中定位根节点
let inroot = indexMap[preorder[preRootIndex]] ?? 0

// 2. 把根节点建立出来
let root = TreeNode(preorder[preRootIndex])

// 3.1 得到左子树中的节点数目
let leftsize = inroot - inleft

// 3.2 递归地构造左子树，并连接到根节点
// 前序遍历中 从 左边界+1 开始的leftsize个元素就对应了中序遍历中 从左边界开始到根节点定位 -1 的元素
root.left = myBuildTree(preorder, inorder, preleft + 1, preleft + leftsize,
```

```
  inleft)
  // 3.3 递归地构造右子树，并连接到根节点
  root.right = myBuildTree(preorder, inorder, preleft + leftsize + 1, preright
  , inroot + 1)
  return root
```

}

func buildTree2(_ preorder: [Int], _ inorder: [Int]) -> TreeNode? {
if preorder.count == 0 || inorder.count == 0 { return nil }

```
  //构建二叉树根结点
  let root: TreeNode? = TreeNode.init(preorder[0])

  //对中序序列进行遍历
  for (index, num) in inorder.enumerated() {
      // 如果找到根节点
      if num == preorder[0] {
          root?.left = buildTree(Array(preorder[1..<index+1]), Array(inorder[0
  ..<index]))
          root?.right = buildTree(Array(preorder[index+1..<preorder.endIndex])
  , Array(inorder[index+1..<inorder.endIndex]))
      }
  }
  return root
```

}

/* 31-- */

/*
题号：15. 三数之和
NC54
出现频率：29
难度：中等

给你一个包含 n 个整数的数组 nums，判断 nums 中是否存在三个元素 a，b，c ，使得 a + b + c = 0 ？请你找出所有满足条件且不重复的三元组。

给定数组 nums = [-1, 0, 1, 2, -1, -4],
满足要求的三元组集合为：
[
[-1, 0, 1],

[-1, -1, 2]
]

题解：https://leetcode-cn.com/problems/3sum/solution/hua-jie-suan-fa-15-san-shu-zhi-he-by-guanpengchn/
*/

func threeSum(_ nums: [Int]) -> [[Int]] {
if nums.count < 3 { return [] }

```
// 0. 排序
let nums = nums.sorted()
var result = Set<[Int]>()


for i in 0 ..< nums.count {

    // 1.最小的数大于0直接跳出循环
    if nums[i] > 0 { break }

    // 2.跳过起点相同的
    if i != 0 && nums[i] == nums[i - 1] { continue }

    // 3. 初始化左右指针
    var l = i + 1
    var r = nums.count - 1

    // 4. 比较
    while l < r {
        let sum = nums[i] + nums[l] + nums[r]
        if sum == 0 {
            result.insert([nums[i], nums[l], nums[r]])
            l += 1
            r -= 1
        } else if sum < 0 {
            l += 1
        } else {
            r -= 1
        }
    }
}
return Array(result)
```

}

/* 32-- */

/*
题号：300. 最长上升子序列
NC91
出现频率：29
难度：中等

输入: [10,9,2,5,3,7,101,18]
输出: 4
解释: 最长的上升子序列是 [2,3,7,101]，它的长度是 4。

题解：https://leetcode-cn.com/problems/longest-increasing-subsequence/solution/zui-chang-shang-sheng-zi-xu-lie-by-leetcode-soluti/
*/

func lengthOfLIS(_ nums: [Int]) -> Int {
// O($n^{2)}$ O(n)
guard nums.count > 0 else { return 0 }

```
var dp = [Int](repeating: 1, count: nums.count) //记录的是数组中比该元素小的元素数量。
for i in 0..<nums.count {
    for j in 0..<i {
        if nums[i] > nums[j] {
            dp[i] = max(dp[i], dp[j] + 1)
        }
    }
}
return dp.max()!
```

}

/* 33-- */

/*
题号：33. 搜索旋转排序数组
NC48
出现频率：27
难度：中等

给你一个升序排列的整数数组 nums ，和一个整数 target 。

假设按照升序排序的数组在预先未知的某个点上进行了旋转。（例如，数组 [0,1,2,4,5,6,7] 可能变为 [4,5,6,7,0,1,2] ）。
请你在数组中搜索 target ，如果数组中存在这个目标值，则返回它的索引，否则返回 -1 。

输入：nums = [4,5,6,7,0,1,2], target = 0
输出：4
*/

func search(_ nums: [Int], _ target: Int) -> Int {

```
if nums.count == 0 { return -1 }

var l = 0
var r = nums.count - 1

// 二分查找
while l <= r { //如果没有 = 那么当输入[1] 1 时，会return -1
    let mid = (l + r) / 2

    if nums[mid] == target { return mid }

    //判断哪一边有序，在有序的一边判断target的位置。
    if nums[l] <= nums[mid] {
        //通过有序边缩小范围
        if target >= nums[l] && target < nums[mid] {
            r = mid - 1
        } else {
            l = mid + 1
        }
    } else {
        if target > nums[mid] && target <= nums[r] {
            l = mid + 1
        } else {
            r = mid - 1
        }
    }
}

return
    -1
```

}

/* 34-- */

/*
题号：69. x 的平方根
NC32
出现频率：27
难度：简单

计算并返回 x 的平方根，其中 x 是非负整数。

由于返回类型是整数，结果只保留整数的部分，小数部分将被舍去。

示例 1:

输入: 4
输出: 2

示例 2:

输入: 8
输出: 2
说明: 8 的平方根是 2.82842...,
由于返回类型是整数，小数部分将被舍去。
*/

/*
4
mid = 2
l = 3 r = 4
result = 2

mid = 3
l = 3 r = 3

mid = 4
l = 3 r = 2
退出
*/

func mySqrt(_ x: Int) -> Int {
var l = 0
var r = x
var result = 0

```
while l <= r {
    let mid =  (l + r) / 2

    if mid * mid <= x {
        l = mid + 1
        result = mid
    } else {
        r = mid - 1
    }
}
return result
```

}

/* 35-- */

/*
题号：155. 最小栈
NC90
出现频率：26
难度：简单

设计一个支持 push ， pop ， top 操作，并能在常数时间内检索到最小元素的栈。

push(x) —— 将元素 x 推入栈中。
pop() —— 删除栈顶的元素。
top() —— 获取栈顶元素。
getMin() —— 检索栈中的最小元素。
*/

class MinStack {

```
var array = [Int]()
var minArray = [Int]()
/** initialize your data structure here. */
init() {

}

func push(_ x: Int) {
    array.append(x)

    if minArray.count == 0 {
        minArray.append(x)
```

```
        } else {
            minArray.append(min(x, minArray.last!))
        }
    }

    func pop() {
        array.popLast()
        minArray.popLast()
    }

    func top() -> Int {
        return array.last!
    }

    func getMin() -> Int {
        return minArray.last!
    }
```

}

/* 36---------- */

/*
题号：121. 买卖股票的最佳时机
NC7
出现频率：25
难度：简单

给定一个数组，它的第 i 个元素是一支给定股票第 i 天的价格。

如果你最多只允许完成一笔交易（即买入和卖出一支股票一次），设计一个算法来计算你所能获取的最大利润。

输入: [7,1,5,3,6,4]
输出: 5
解释: 在第 2 天（股票价格 = 1）的时候买入，在第 5 天（股票价格 = 6）的时候卖出，最大利润 = 6-1 = 5 。
注意利润不能是 7-1 = 6, 因为卖出价格需要大于买入价格；同时，你不能在买入前卖出股票。

*/

//遍历数组，更新最小值
//根据遍历值与最小值的差，更新最大值
func maxProfit(_ prices: [Int]) -> Int {

```
    if prices.count == 0 { return 0 }

    // min & max
    var min = prices[0]
    var max = 0

    for i in prices {
        if i < min {
            min = i
        }

        if i - min > max {
            max = i - min
        }
    }
    return max
```

}

/* 37-- */

/*
题号：23. 合并K个升序链表
NC51
出现频率：25
难度：困难

给你一个链表数组，每个链表都已经按升序排列。

请你将所有链表合并到一个升序链表中，返回合并后的链表。

示例 1：

输入：lists = [[1,4,5],[1,3,4],[2,6]]
输出：[1,1,2,3,4,4,5,6]
解释：链表数组如下：
[
1->4->5,
1->3->4,
2->6
]
将它们合并到一个有序链表中得到。
1->1->2->3->4->4->5->6

*/

//思路：遍历k个排序链表，记录到字典中，字典的key是链表中的val，字典的value是k个链表中val出现的次数。然后通过字典再生成新的链表。算法的时间复杂度应该是O(nk),n是排序列表中元素的个数。

func mergeKLists(_ lists: [ListNode?]) -> ListNode? {

```
var result: ListNode?
var dic = [Int: Int]()
var cur: ListNode?

// 1.遍历k个排序链表，记录到字典中，字典的key是链表中的val，字典的value是k个链表中val出现的次数
for node in lists {
    cur = node
    while cur != nil {
        dic.updateValue(dic[cur!.val] ?? 0 + 1, forKey: cur!.val)
        cur = cur!.next
    }
}

// 2.通过字典再生成新的链表
for key in dic.keys.sorted() {
    for _ in 1...dic[key]! {
        if result == nil {
            result = ListNode.init(key)
            cur = result
            continue
        }

        cur?.next = ListNode.init(key)
        cur = cur?.next
    }
}

return result
```

}

/* 38 跳过---------- /
/
题号：剑指 Offer 38. 字符串的排列
出现频率：25
难度：中等

输入一个字符串，打印出该字符串中字符的所有排列。
你可以以任意顺序返回这个字符串数组，但里面不能有重复元素。

示例:

输入：s = "abc"
输出：["abc","acb","bac","bca","cab","cba"]

*/

/* 38-- */

/*
题号：46. 全排列
出现频率：25
难度：中等

给定一个 没有重复 数字的序列，返回其所有可能的全排列。

示例:

输入: [1,2,3]
输出:
[
[1,2,3],
[1,3,2],
[2,1,3],
[2,3,1],
[3,1,2],
[3,2,1]
]
*/
var result3 = [Int]
var path3 = Int
var used = Int: Bool

func permute(_ nums: [Int]) -> [[Int]] {

```
guard nums.count != 0 else {
    return []
}

dfs(nums: nums, depth: 0)
```

```
    return result3
```

}

func dfs(nums: [Int], depth: Int) {

```
    if nums.count == depth {
        result3.append(path3)
        return
    }

    for i in 0..<nums.count {
        if used[nums[i]] ?? false == false {
            path3.append(nums[i])
            used[nums[i]] = true

            dfs(nums: nums, depth: depth + 1)

            path3.removeLast()
            used[nums[i]] = false
        }
    }
```

}

/* 39-- */

/*
题号：11. 盛最多水的容器
NC128
出现频率：25
难度：中等

给你 n 个非负整数 a1，a2，...，an，每个数代表坐标中的一个点 (i, ai) 。在坐标内画 n 条垂直线，垂直线 i 的两个端点分别为 (i, ai) 和 (i, 0) 。找出其中的两条线，使得它们与 x 轴共同构成的容器可以容纳最多的水。

输入：[1,8,6,2,5,4,8,3,7]
输出：49
解释：图中垂直线代表输入数组 [1,8,6,2,5,4,8,3,7]。在此情况下，容器能够容纳水（表示为蓝色部分）的最大值为 49。

https://leetcode-cn.com/problems/container-with-most-water/solution/shuang-zhi-zhen-fa-zh

eng-ming-jian-dan-yi-dong-bu-/
*/
func maxArea(_ height: [Int]) -> Int {
var l = 0
var r = height.count - 1
var result = 0

```
while l < r {
    if height[l] > height[r] {
        result = max(result, height[r] * (r - l))
        r -= 1
    } else {
        result = max(result, height[l] * (r - l))
        l += 1
    }
}
return result
```

}

/* 40-- */

/*
题号：199. 二叉树的右视图
NC136
出现频率：25
难度：中等
*/

func rightSideView(_ root: TreeNode?) -> [Int] {
guard let root = root else { return [] }

```
var result = [Int]()
var queue: [TreeNode] = [root]

while !queue.isEmpty {
    let count = queue.count

    for index in 0 ..< count {
        let node = queue.removeFirst()
        if index == count - 1{
            result.append(node.val)
        }
```

```
            if node.left != nil {
                queue.append(node.left!)
            }

            if node.right != nil {
                queue.append(node.right!)
            }
        }
    }
    return result
```

}

/* 41-- /
/
题号：NC109 岛屿问题
出现频率：23
难度：中等
*/

/* 42-- */

/*
题号：104.二叉树的最大深度
NC13
出现频率：23
难度：简单
*/
func maxDepth(_ root: TreeNode?) -> Int {
guard let root = root else { return 0 }

```
    let leftDepth = maxDepth(root.left)
    let rightDepth = maxDepth(root.right)

    return max(leftDepth, rightDepth) + 1
```

}

/* 43 跳过-- */

/*
题号：148. 排序链表

NC70
出现频率：22
难度：中等
*/

func sortList(_ head: ListNode?) -> ListNode? {
if head == nil || head?.next == nil {
return head
}

```
var slow = head, fast = head?.next
while fast != nil && fast?.next != nil {
    slow = slow?.next
    fast = fast?.next?.next
}

let temp = slow?.next
slow?.next = nil
var left = sortList(head)
var right = sortList(temp)
var h = ListNode(0)
let res = h

while left != nil && right != nil {
    if left!.val < right!.val {
        h.next = left
        left = left?.next
    } else {
        h.next = right
        right = right?.next
    }
    h = h.next!
}

h.next = left != nil ? left : right
return res.next
```

}

/* 44 跳过---------- */

/*
题号：110. 平衡二叉树
NC62

出现频率：21
难度：简单
*/
func isBalanced(_ root: TreeNode?) -> Bool {
guard let root = root else {
return true
}
let left = helper(root.left)
let right = helper(root.right)
return abs(left - right) <= 1 && isBalanced(root.left) && isBalanced(root.right)
}

private func helper(_ root: TreeNode?) -> Int {
guard let root = root else {
return 0
}
let left = helper(root.left)
let right = helper(root.right)
return max(left, right) + 1
}

/* 45-- /
/
题号：剑指 Offer 39. 数组中出现次数超过一半的数字
NC73
出现频率：21
难度：简单

数组中有一个数字出现的次数超过数组长度的一半，请找出这个数字。

你可以假设数组是非空的，并且给定的数组总是存在多数元素。

输入: [1, 2, 3, 2, 2, 2, 5, 4, 2]
输出: 2
*/

func majorityElement(_ nums: [Int]) -> Int {

```
var count = 0
var candidate = nums[0]
```

```
  for num in nums {
      if count == 0 {
          candidate = num
      }
      count += (num==candidate) ? 1:-1
  }
  return candidate
```

}

func majorityElement2(_ nums: [Int]) -> Int {
return nums.sorted()[nums.count / 2]
}

/* 46-- /
/
题号：NC137 表达式求值
出现频率：21
难度：中等
*/

/* 47-- */

/*
题号：234. 回文链表
NC96
出现频率：17
难度：简单

请判断一个链表是否为回文链表。

示例 1:

输入: 1->2
输出: false
示例 2:

输入: 1->2->2->1
输出: true
进阶：
你能否用 O(n) 时间复杂度和 O(1) 空间复杂度解决此题?
*/

func isPalindrome(_ head: ListNode?) -> Bool {
var newHead = head
var list = Int

```
// 1.将链表加入到数组中
while newHead != nil {
    list.append(newHead!.val)
    newHead = newHead?.next
}

// 2.双指针比较数组两端是否相等
var start = 0
var end = list.count - 1
while start < end {
    if list[start] != list[end] {
        return false
    }
    start += 1
    end -= 1
}
return true
```

}

/* 48-- /
/
题号：NC35 最小编辑代价
出现频率：16
难度：困难
*/

/* 49-- */

/*
题号：113. 路径总和 II
NC8
出现频率：2
难度：中等

题号：剑指 Offer 34. 二叉树中和为某一值的路径
出现频率：1
难度：中等

给定一个二叉树和一个目标和，找到所有从根节点到叶子节点路径总和等于给定目标和的路径。

给定如下二叉树，以及目标和 sum = 22，
5
/ \
4 8
/ / \
11 13 4
/ \ / \
7 2 5 1

返回:

[
[5,4,11,2],
[5,8,4,5]
]
*/

var path = Int
var res = [Int]

func pathSum(_ root: TreeNode?, _ sum: Int) -> [[Int]] {

```
dfs(root, sum)
return res
```

}

func dfs(_ root: TreeNode?, _ sum: Int) {
guard let root = root else { return }

```
path.append(root.val)
let tmp = sum - root.val
if tmp == 0 && root.left == nil && root.right == nil {
    res.append(path)
}

dfs(root.left, tmp)
dfs(root.right, tmp)

path.removeLast() // 重点，遍历完后，需要把当前节点remove出去，因为用的是同一个list对
```

```
象来存所有的路径
```

}

/* 50 跳过-- */

/*
题号：56. 合并区间
NC37
出现频率：15
难度：中等
*/

/*
输入: intervals = [[1,3],[2,6],[8,10],[15,18]]
输出: [[1,6],[8,10],[15,18]]
解释: 区间 [1,3] 和 [2,6] 重叠, 将它们合并为 [1,6].
*/

func merge(_ intervals: [[Int]]) -> [[Int]] {
if intervals.count <= 1 { return intervals }

```
// 1. 排序
var list = intervals.sorted{ $0[0] < $1[0] }
var index = 1

while index < list.count {
    // 2. 拿到比较对象
    let pre = list[index - 1]
    let aft = list[index]

    // 3.分情况处理

    // 3.1 前一个区间包含后一个区间
    if pre.last! > aft.last! {
        list.remove(at: index)
    }
    // 3.2 前一个区间和后一个区间相交
    else if aft.first! <= pre.last! {
        list[index - 1] = [pre.first!, aft.last!]
        list.remove(at: index)
    }
    // 3.3 比较下一位
    else {
```

```
            index = index + 1
        }
    }
    return list
```

}

/* 51-- /
/
题号：NC36 在两个长度相等的排序数组中找到上中位数
出现频率：15
难度：困难
*/

/* 52-- /
/
题号：NC60 判断该二叉树是否为搜索二叉树和完全二叉树
出现频率：15
难度：中等
*/

//搜索二叉树

/*
题号：958.二叉树的完全性检验（完全二叉树）
难度：中等
*/

func isCompleteTree(_ root: TreeNode?) -> Bool {
guard let root = root else { return false }
var queue = [root]
var leaf = false

```
while !queue.isEmpty {
    let node = queue.removeFirst()

    // 1）如果某个节点的右子树不为空，则它的左子树必须不为空
    if node.left == nil && node.right != nil {
        return false
    }
    // 2）如果某个节点的右子树为空，则排在它后面的节点必须没有子节点
    if leaf && (node.left != nil || node.right != nil) {
        return false
```

```
        }
        // 3）叶子节点
        if node.left == nil || node.right == nil {
            leaf = true
        }
        // 4）队列添加元素
        if node.left != nil {
            queue.append(node.left!)
        }
        if node.right != nil {
            queue.append(node.right!)
        }
    }
    return true
```

}

/* 53-- */

/*
题号：62. 不同路径
NC34
出现频率：15
难度：中等

一个机器人位于一个 m x n 网格的左上角 （起始点在下图中标记为“Start” ）。
机器人每次只能向下或者向右移动一步。机器人试图达到网格的右下角（在下图中标记为“Finish”）。
问总共有多少条不同的路径?

*/

//https://leetcode-cn.com/problems/unique-paths/solution/bu-tong-lu-jing-dong-tai-gui-hua-zu-he-fen-xi-java/
//每一个方格可以由上一个向右或者上一个向下到达
//dp[i][j] = dp[i][j-1] + dp[i-1][j]
func uniquePaths(_ m: Int, _ n: Int) -> Int {

```
    var dp = Array(repeating: Array(repeating: 0, count: n), count: m)

    for i in 0..<m { dp[i][0] = 1 }
    for j in 0..<n { dp[0][j] = 1 }

    for i in 1..<m {
```

```
        for j in 1..<n {
            dp[i][j] = dp[i - 1][j] + dp[i][j-1]
        }
    }
    return dp[m-1][n-1]
```

}

/* 54-- */

/*
题号：257. 二叉树的所有路径
NC5
出现频率：15
难度：中等

给定一个二叉树，返回所有从根节点到叶子节点的路径。

说明: 叶子节点是指没有子节点的节点。

示例:

输入:

```
  1
```

/ \
2 3
\
5

输出: ["1->2->5", "1->3"]

解释: 所有根节点到叶子节点的路径为: 1->2->5, 1->3
*/

var res3 = String

func binaryTreePaths(_ root: TreeNode?) -> [String] {
bfs(root, "")
return res3
}

func bfs(_ root:TreeNode? ,_ s:String) {

```
guard let root = root else { return }

let result = s + "\(root.val)"

if root.left == nil && root.right == nil {
    res3.append(result)
}

bfs(root.left, result + "->")
bfs(root.right, result + "->")
```

}

/* 55---------- */
//题目二：删除链表中重复的节点
//NC24
//出现频率：15
func deleteDuplicates(_ head: ListNode?) -> ListNode? {
let newHead = ListNode.init(-1)
newHead.next = head

```
var head = head

while head != nil {
    if head?.val == head?.next?.val {
        head?.next = head?.next?.next
    } else {
        head = head?.next
    }
}
return newHead.next}
```

/* 56---------- */

/*
题号：7. 整数反转
NC57
出现频率：14
难度：简单

给出一个 32 位的有符号整数，你需要将这个整数中每位上的数字进行反转。

示例 1:

输入: 123
输出: 321

示例 2:

输入: -123
输出: -321

示例 3:

输入: 120
输出: 21
注意:

假设我们的环境只能存储得下 32 位的有符号整数，则其数值范围为 [–231，231 – 1]。请根据这个假设，如果反转后整数溢出那么就返回 0。
*/

func reverse(_ x: Int) -> Int {

```
var x = x //旧的值
var result = 0 //新的值

while x != 0 {
    // curX % 10 取模，即最后一位的值
    // curX / 10 除以10，即去掉最后一位
    result = result * 10 + x % 10
    x = x / 10

    // 每次转变后检查是否溢出
    if result > Int32.max || result < Int32.min {
        return 0
    }
}
return result
```

}

/* 57-- */
//矩阵元素查找
//NC86

/* 58-- */

/*
题号：41. 缺失的第一个正数
NC30
出现频率：14
难度：困难

给你一个未排序的整数数组 nums ， 请你找出其中没有出现的最小的正整数。
进阶：你可以实现时间复杂度为 O(n) 并且只使用常数级别额外空间的解决方案吗?

输入：nums = [1,2,0]
输出：3

输入：nums = [3,4,-1,1]
输出：2
*/
func firstMissingPositive(_ nums: [Int]) -> Int {
if nums.count == 0 {return 1}
var dic = Dictionary()

```
for i in nums {
    if dic[i] == nil {
        dic[i] = 1
    }
}

for i in 1...dic.keys.count {
    if dic[i] == nil {
        return i
    }
}
return dic.keys.count + 1
```

}

/* 59-- */

/*
题号：剑指 Offer 53 - II. 0～n-1中缺失的数字
NC101
出现频率：14
难度：简单

一个长度为n-1的递增排序数组中的所有数字都是唯一的，并且每个数字都在范围0～n-1之内。在范围0～n-1内的n个数字中有且只有一个数字不在该数组中，请找出这个数字。

输入: [0,1,3]
输出: 2
*/

/* 60-- /
/
题号：NC100 将字符串转化为整数
出现频率：14
难度：困难
*/

/* 61-- */

/*
题号：101. 对称二叉树
NC16
出现频率：13
难度：简单

给定一个二叉树，检查它是否是镜像对称的。
例如，二叉树 [1,2,2,3,4,4,3] 是对称的。

```
  1
 / \
```

2 2
/ \ / \
3 4 4 3

*/

func isSymmetric(_ root: TreeNode?) -> Bool {
guard let root = root else { return true }
return dfs(left: root.left, right: root.right)
}

func dfs(left: TreeNode?, right: TreeNode?) -> Bool {
if left == nil && right == nil {

return true
}

```
    if left == nil || right == nil {
        return false
    }

    if left!.val != right!.val {
        return false
    }

    return dfs(left: left!.left, right: right!.right) && dfs(left: left!.right,
    right: right!.left)
```

}

/* 62-- /
/
题号：LFU缓存结构设计
NC94
出现频率：12
难度：困难
*/

/* 63-- /
/
题号：有重复项数字的所有排列
NC42
出现频率：11
难度：中等
*/

/* 64-- /
/
题号：112. 路径总和
NC9
出现频率：10
难度：简单

给定一个二叉树和一个目标和，判断该树中是否存在根节点到叶子节点的路径，这条路径上所有节点值相加等于目标和。

类似题目：113. 路径总和 II
*/

func hasPathSum(_ root: TreeNode?, _ sum: Int) -> Bool {
//1. 退出条件1
guard let root = root else { return false }

```
//2. 退出条件2
let tmp = sum - root.val
if tmp == 0 && root.left == nil && root.right == nil {
    return true
}

//3.递归
return hasPathSum(root.left, tmp) || hasPathSum(root.right, tmp)
```

}

/* 65-- /
/
题号：最长的括号子串
NC49
出现频率：10
难度：困难
*/

/* 66-- /
/
题号：序列化二叉树
NC123
出现频率：9
难度：困难
*/

/* 67-- /
/
题号：字符串变形
NC89
出现频率：9
难度：简单
*/

/* 68-- */

/*
题号：543.二叉树的直径
NC6
出现频率：9
难度：简单

给定一棵二叉树，你需要计算它的直径长度。一棵二叉树的直径长度是任意两个结点路径长度中的最大值。这条路径可能穿过也可能不穿过根结点。
*/

var currentDiameter = 0

// 通过计算二叉树的深度，获取二叉树的直径
func diameterOfBinaryTree(_ root: TreeNode?) -> Int {
let _ = maxDepth2(root)
return currentDiameter
}

func maxDepth2(_ root: TreeNode?) -> Int {
guard let root = root else { return 0 }

```
let leftDepth = maxDepth(root.left)
let rightDepth = maxDepth(root.right)

// 在计算深度的过程中，更新直径
currentDiameter = max(currentDiameter, leftDepth + rightDepth)

return max(leftDepth, rightDepth) + 1
```

}

/* 69-- /
/
题号：数字字符串转换成IP地址
NC20
出现频率：12
难度：中等
*/

/* 70-- /

/
题号：大数乘法
NC10
出现频率：9
难度：中等
*/

/* 71-- /
/
题号：剑指 Offer 22. 链表中倒数第k个节点
出现频率：8
难度：简单

输入一个链表，输出该链表中倒数第k个节点。为了符合大多数人的习惯，本题从1开始计数，即链表的尾节点是倒数第1个节点。

例如，一个链表有 6 个节点，从头节点开始，它们的值依次是 1、2、3、4、5、6。这个链表的倒数第 3 个节点是值为 4 的节点。

给定一个链表: 1->2->3->4->5, 和 k = 2.
返回链表 4->5.
*/
func getKthFromEnd(_ head: ListNode?, _ k: Int) -> ListNode? {
// 1. 常规解法
// var list = ListNode
// var tempHead = head
// while tempHead != nil {
// list.append(tempHead!)
// tempHead = tempHead?.next
// }

```
// return list[list.count-k]

// 2. 快慢
var fast : ListNode? = head
var slow : ListNode? = head
var tk = k
while tk > 0 {
    fast = fast?.next
    tk -= 1
}
```

```
    while fast != nil {
        fast = fast?.next
        slow = slow?.next
    }

    return slow
```

}

/* 72 跳过-- */

/*
题号：162. 寻找峰值
出现频率：8
难度：中等

峰值元素是指其值大于左右相邻值的元素。
给定一个输入数组 nums，其中 nums[i] ≠ nums[i+1]，找到峰值元素并返回其索引。
数组可能包含多个峰值，在这种情况下，返回任何一个峰值所在位置即可。
你可以假设 nums[-1] = nums[n] = -∞。

示例 1:

输入: nums = [1,2,3,1]
输出: 2
解释: 3 是峰值元素，你的函数应该返回其索引 2。

示例 2:

输入: nums = [1,2,1,3,5,6,4]
输出: 1 或 5
解释: 你的函数可以返回索引 1，其峰值元素为 2；
  或者返回索引 5， 其峰值元素为 6。

说明:

你的解法应该是 O(logN) 时间复杂度的。

思路 :
https://leetcode-cn.com/problems/find-peak-element/solution/hua-jie-suan-fa-162-xun-zhao-feng-zhi-by-guanpengc/

首先要注意题目条件，在题目描述中出现了 nums[-1] = nums[n] = -∞，这就代表着只要数组中存

在一个元素比相邻元素大，那么沿着它一定可以找到一个峰值
根据上述结论，我们就可以使用二分查找找到峰值
查找时，左指针 l，右指针 r，以其保持左右顺序为循环条件
根据左右指针计算中间位置 m，并比较 m 与 m+1 的值，如果 m 较大，则左侧存在峰值，r = m，如果 m + 1 较大，则右侧存在峰值，l = m + 1
*/
func findPeakElement(_ nums: [Int]) -> Int {
var l = 0
var r = nums.count - 1

```
while l < r {
    let mid = ( l + r ) / 2

    if nums[mid] > nums[mid + 1] {
        r = mid
    } else {
        l = mid + 1
    }
}
return l
```

}

/* 73-- /
/
题号：NC28 最小覆盖子串
出现频率：7
难度：中等
*/

/* 73-- /
/
题号：NC108 最大正方形
出现频率：7
难度：中等
*/

/* 74 跳过-- */

/*
题号：189. 旋转数组

NC110
出现频率：13
难度：中等

给定一个数组，将数组中的元素向右移动 k 个位置，其中 k 是非负数。

输入: [1,2,3,4,5,6,7] 和 k = 3
输出: [5,6,7,1,2,3,4]
解释:
向右旋转 1 步: [7,1,2,3,4,5,6]
向右旋转 2 步: [6,7,1,2,3,4,5]
向右旋转 3 步: [5,6,7,1,2,3,4]

*/

```
func rotate(_ nums: inout [Int], _ k: Int) {
for _ in 0..<k {
let last = nums.removeLast()
nums.insert(last, at: 0)
}
}
```

/* 75 跳过---------- /
/
题号：240. 搜索二维矩阵 II
NC29
出现频率：7
难度：中等

编写一个高效的算法来搜索 m x n 矩阵 matrix 中的一个目标值 target。该矩阵具有以下特性：

每行的元素从左到右升序排列。
每列的元素从上到下升序排列。
示例:

现有矩阵 matrix 如下：

```
[
[1, 4, 7, 11, 15],
[2, 5, 8, 12, 19],
[3, 6, 9, 16, 22],
[10, 13, 14, 17, 24],
```

[18, 21, 23, 26, 30]
]
给定 target = 5，返回 true。

给定 target = 20，返回 false。
*/

func findNumberIn2DArray(_ matrix: [[Int]], _ target: Int) -> Bool {
var row = matrix.count - 1 //行
var col = 0 //列

```
while row >= 0 && col <= matrix[0].count - 1 {
    if matrix[row][col] > target {
        row -= 1
    } else if matrix[row][col] < target{
        col += 1
    } else {
        return true
    }
}
return false
```

}

/* 76-- /
/
题号：随时找到数据流的中位数
NC131
出现频率：6
难度：中等
*/

/* 77-- /
/
题号：正则表达式匹配
NC122
出现频率：6
难度：困难
*/

/* 78-- /
/

题号：把数字翻译成字符串
NC116
出现频率：5
难度：困难
*/

/* 79-- /
/
题号：股票交易的最大收益（二）
NC135
出现频率：4
难度：中等
*/

// 以下题目为自添加重要题目
//--
//--
//--

/* 59-- */

/*
题号：9. 回文数
出现频率：23
难度：简单
*/
func isPalindrome(_ x: Int) -> Bool {
if x < 0 || x % 10 == 0 && x != 0 { return false }

```
var curX = x
var reverse = 0

while curX > reverse {
    reverse = reverse * 10 + curX % 10
    curX = curX / 10
}
return curX == reverse || curX == reverse / 10
```

}

/* 60-- */

/*
题号：151. 翻转字符串里的单词
出现频率：3
难度：中等

给定一个字符串，逐个翻转字符串中的每个单词。

示例 1：

输入："the sky is blue"
输出："blue is sky the"
示例 2：

输入：" hello world! "
输出："world! hello"
解释：输入字符串可以在前面或者后面包含多余的空格，但是反转后的字符不能包括。
示例 3：

输入："a good   example"
输出："example good a"
解释：如果两个单词间有多余的空格，将反转后单词间的空格减少到只含一个。
示例 4：

输入：s = " Bob Loves Alice "
输出："Alice Loves Bob"
示例 5：

输入：s = "Alice does not even like bob"
输出："bob like even not does Alice"
*/

func reverseWords(_ s: String) -> String {

```
var word = [Character]() // 装填每个字母
var words = [[Character]]() // 装填每个单词

for item in s {
    // 略过多余空格
    if item == " " && word.count == 0 {
        continue
    }
    // 装填一次单词
    else if item == " " && word.count != 0{
```

```
            words.append(word)
            word.removeAll()
        }

        // 添加单词
        else {
            word.append(item)
        }
    }
    // 添加最后一个单词
    if word.count != 0 { words.append(word) }

    var str = ""
    let count = words.count
    while words.count != 0 {
        // 非第一个单词，添加一个空格
        if words.count != count {str.append(" ")}
        // 倒叙添加单词
        str = str + words.popLast()!
    }

    return str
```

}

/* 61-- */

/*
题号：190. 颠倒二进制位
出现频率：2
难度：简单
*/
func reverseBits(_ n: Int) -> Int {
var result = 0
var index = 31
var n = n
while index >= 0 {
result += (n & 1) << index
n = n >> 1
index -= 1
}
return result
}

/* 62-- */

/*
题号：136. 只出现一次的数字
出现频率：1
难度：简单

给定一个非空整数数组，除了某个元素只出现一次以外，其余每个元素均出现两次。找出那个只出现了一次的元素。

说明：

你的算法应该具有线性时间复杂度。 你可以不使用额外空间来实现吗?

示例 1:

输入: [2,2,1]
输出: 1

*/

func singleNumber(_ nums: [Int]) -> Int {
// 第一反应用map
// 考虑到不适用额外空间，用异或
// a ^ 0 = a
// a ^ a = 0
// a ^ b ^ a = a ^ a ^ b = 0 ^ b = b
return nums.reduce(0) { 0 ^ 1 }
}

func singleNumber2(_ nums: [Int]) -> Int {
var dic = Dictionary()

```
for i in nums {
    var count = dic[i]
    count = count == nil ? 1 : count! + 1
    dic[i] = count
}

for item in dic.keys {
    let value = dic[item]
    if value == 1 { return item }
}
return -1
```

}

/* 63-- */

/*
题号：209. 长度最小的子数组
出现频率：1
难度：中等

给定一个含有 n 个正整数的数组和一个正整数 s ，找出该数组中满足其和 ≥ s 的长度最小的 连续 子数组，并返回其长度。如果不存在符合条件的子数组，返回 0。

输入：s = 7, nums = [2,3,1,2,4,3]
输出：2
解释：子数组 [4,3] 是该条件下的长度最小的子数组。
*/
func minSubArrayLen(_ s: Int, _ nums: [Int]) -> Int {
var minLen: Int = .max
var sum = 0
var left = 0

```
for right in 0 ..< nums.count {
    sum += nums[right]

    while sum >= s {
        minLen = min(right - left + 1, minLen)

        //移除最左边的数，再试图进行一次比较
        sum -= nums[left]
        left += 1
    }
}

return minLen == .max ? 0 : minLen
```

}

/* 67-- */

/*
题号：48. 旋转图像
出现频率：13

难度：中等

给定 matrix =
[
[1,2,3],
[4,5,6],
[7,8,9]
],

原地旋转输入矩阵，使其变为:
[
[7,4,1],
[8,5,2],
[9,6,3]
]
*/

// 先将数组反转，再进行对角线交换即可
func rotate(_ matrix: inout [[Int]]) {
let n = matrix.count
matrix.reverse()

```
for i in 0 ..< n { //第i行
    for j in i ..< n { //第i行的第j个元素
        if i == j { continue } //如果是对角线则直接跳过
        let tmp = matrix[i][j]
        matrix[i][j] = matrix[j][i]
        matrix[j][i] = tmp
    }
}
```

}

/* 68-- */

/*
题号：128. 最长连续序列
出现频率：9
难度：困难

输入: [100, 4, 200, 1, 3, 2]
输出: 4

解释: 最长连续序列是 [1, 2, 3, 4]。它的长度为 4。
*/

func longestConsecutive(_ nums: [Int]) -> Int {
// 1.先将数组插入到集合中
let set = Set(nums)

```
var maxLength = 0
for item in set {
    // 2. 遍历集合，如果集合不包含当前元素的上一个，则说明可以从这个元素开始计数(说明没有计过数)
    if !set.contains((item-1)) {
        // 3. 从该数开始计数，如果存在下一个，则+1，否则进入下一次循环
        var next = item + 1
        var curLength = 1 //此次循环最大长度
        while set.contains(next) {
            curLength += 1
            next += 1
        }
        maxLength = max(maxLength, curLength)
    }
}
// 4.返回结果即可
return maxLength
```

}

/* 70-- */

/*
题号：322. 零钱兑换
出现频率：8
难度：中等

给定不同面额的硬币coins和一个总金额amount。编写一个函数来计算可以凑成总金额所需的最少的硬币个数。如果没有任何一种硬币组合能组成总金额,返回-1。
f
示例 1：

输入：coins = [1, 2, 5], amount = 11
输出：3
解释：11 = 5 + 5 + 1

https://leetcode-cn.com/problems/coin-change/solution/322-ling-qian-dui-huan-by-leetcode-s

olution/
*/

func coinChange(_ coins: [Int], _ amount: Int) -> Int {
if coins.count == 0{
return -1
}
// dp[i][j] 使用i种硬币达到j的最小数量
var dp = Array(repeating: amount + 1, count: amount + 1)
// 0 需要 0的硬币
dp[0] = 0
for coin in coins {
var i = coin
while i <= amount {
// 假如: i - coin 合法, 只需要加上当前这一枚硬币即可
dp[i] = min(dp[i], dp[i - coin] + 1)
i += 1
}
}
return dp[amount] >= amount + 1 ? -1:dp[amount]

}

func coinChange2(_ coins: [Int], _ amount: Int) -> Int {
if amount == 0 { return 0 }

```
var dp = Array(repeating: amount + 1, count: amount + 1)
dp[0] = 0

for money in 1...amount {
    for coin in coins {
        if money >= coin {
            dp[money] = min(dp[money],dp[money - coin] + 1)
        }
    }
}

return dp[amount] == amount + 1 ? -1 : dp[amount]
```

}

/* 71-- */

/*
题号：122. 买卖股票的最佳时机 II
出现频率：5
难度：简单

给定一个数组，它的第 i 个元素是一支给定股票第 i 天的价格。

设计一个算法来计算你所能获取的最大利润。你可以尽可能地完成更多的交易（多次买卖一支股票）。
*/

//既然不限制交易次数, 那么把能赚钱的交易都累加上就是最大利润了.
func maxProfit2(_ prices: [Int]) -> Int {
var result = 0

```
for i in 1..<prices.count {
    let cur = prices[i] - prices[i - 1]
    result += cur > 0 ? cur : 0
}
return result
```

}

/* 72-- */

/*
题号：剑指 Offer 27. 二叉树的镜像
出现频率：4
难度：简单
*/

/*
题号：226. 翻转二叉树
出现频率：4
难度：简单
*/

func mirrorTree(_ root: TreeNode?) -> TreeNode? {
guard let root = root else { return nil }
let right = mirrorTree(root.right)
let left = mirrorTree(root.left)
root.right = left

```
root.left = right
return root
}

func mirrorTree2(_ root: TreeNode?) -> TreeNode? {
guard root != nil else {
return nil
}
var queue = TreeNode
queue.append(root!)
while !queue.isEmpty {
let node = queue.removeFirst()
let nodeLeft = node.left
node.left = node.right
node.right = nodeLeft
if node.left != nil {
queue.append(node.left!)
}
if node.right != nil {
queue.append(node.right!)
}
}
return root
}

/* 73---------------------------------------------- */

/*
题号：394. 字符串解码
出现频率：未知
难度：中等

题解：https://leetcode-cn.com/problems/decode-string/solution/decode-string-fu-zhu-zhan-fa-di-gui-fa-by-jyd/
*/
func decodeString(_ s: String) -> String {
var stack = (Int, String)
var words = ""
var nums = 0
for c in s {
```

```
if c == "[" {
stack.append((nums, words)) // 0 ""
nums = 0
words = ""
} else if c == "]" {
if let (curMutil, lastRes) = stack.popLast() {
words = lastRes + String(repeating: words, count: curMutil)
}
} else if c.isWholeNumber {
nums = nums * 10 + c.wholeNumberValue!
} else {
words.append(c)
}
}
return words
}
```