目录

第 1 章 Top 100 liked

1.1 Leetcode 1.Two Sum

Fig 1.1. Given an array of integers, return indices of the two numbers such that they add up to a specific target.

思路：用 *sub_dict* 存储每一个元素与 target 之差的值。

```
class Solution(object):
    def twoSum(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: List[int]
        """
        sub_dict = {}
        for i in range(0, len(nums)):
            sub_item = target -nums[i]
            if nums[i] not in sub_dict.keys():
                sub_dict[sub_item] = i
            else:
                return sub_dict[nums[i]],i
```

图 1.1 two sum

1.2 Leetcode 2. Add Two Numbers

Fig 1.2.You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order and each of their nodes contain a single digit. Add the two numbers and return it as a linked list.

思路：链表的操作, 且从低位数到高位数累加的技巧。

1.3 Leetcode 3. Longest Substring Without Repeating Characters

Fig 1.3.Given a string, find the length of the longest substring without repeating characters.

思路：与 two sum 1.1有点类似，不过需要遍历全部的数组，并且注意 *used_char* 的序号一定要在每一次的 start 之后，不然就不必更改 start。

```python
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution(object):
    def addTwoNumbers(self, l1, l2):
        """
        :type l1: ListNode
        :type l2: ListNode
        :rtype: ListNode
        """
        r = head = ListNode(0)
        carry = 0
        while l1 or l2 or carry:
            v1,v2 = 0, 0
            if l1:
                v1 = l1.val
                l1 = l1.next
            if l2:
                v2 = l2.val
                l2 =l2.next
            tmp = v1 + v2 + carry
            r.next = ListNode(tmp%10)
            r = r.next
            carry = tmp//10
        return head.next
```

图 1.2 Add Two Nums

```python
class Solution(object):
    def lengthOfLongestSubstring(self, s):
        """
        :type s: str
        :rtype: int
        """
        max_len = start = 0
        used_char = {}
        for i in range(len(s)):
            if s[i] in used_char and start<= used_char[s[i]]:
                start = used_char[s[i]] + 1
            else:
                max_len = max(max_len, i-start+1)
            used_char[s[i]] = i
        return max_len
```

图 1.3 Longest Substring Without Repeating Characters

1.4 Leetcode 5. Longest Palindromic Substring

Fig 1.4Given a string s, find the longest palindromic substring in s. You may assume that the maximum length of s is 1000.

思路：寻找回文字符串的子函数，然后遍历数组。寻找回文字符串的子函数中的边界条件需要注意。

```
class Solution(object):
    def longestPalindrome(self, s):
        """
        :type s: str
        :rtype: str
        """
        longest_str = ''
        for i in range(0, len(s)):
            tmp_str = self.findPalindrome(i, i, s)
            if len(tmp_str)> len(longest_str):
                longest_str = tmp_str
            tmp_str = self.findPalindrome(i, i+1, s)
            if len(tmp_str)> len(longest_str):
                longest_str = tmp_str
        return longest_str


    def findPalindrome(self, l, r, s):
        while l>=0 and r<len(s) and s[l] == s[r]:
            l-=1
            r+=1
        return s[l+1:r]
```

图 1.4 Longest Palindromic Substring

1.5 Leetcode 11. Container With Most Water

Fig 1.5Given n non-negative integers a1, a2, ..., an , where each represents a point at coordinate (i, ai). n vertical lines are drawn such that the two endpoints of line i is at (i, ai) and (i, 0). Find two lines, which together with x-axis forms a container, such that the container contains the most water.

Note: You may not slant the container and n is at least 2.

思路：两边取短的高度为宽，乘以间隔是此刻的水容量。谁短谁往中间靠一步。

```
class Solution(object):
    def maxArea(self, height):
        """
        :type height: List[int]
        :rtype: int
        """
        max_vol = 0
        l, r = 0, len(height)-1
        while l< r:

            if height[l]<height[r]:
                max_vol = max(max_vol, height[l]*(r-l))
                l+=1
            else:
                max_vol = max(max_vol, height[r]*(r-l))
                r-=1
        return max_vol
```

图 1.5 Longest Palindromic Substring

1.6 Leetcode 15. 3Sum

Fig 1.6Given an array nums of n integers, are there elements a, b, c in nums such that a + b + c = 0? Find all unique triplets in the array which gives the sum of zero.

Note: The solution set must not contain duplicate triplets.

思路：利用递归的思路去解题。用 sum 的 target 逐一减掉数组中的一个数，然后递归同样的操作。同时注意消除可能出现同样元素的情况。各边界情况值得注意。

```
class Solution(object):
    def threeSum(self, nums):
        """
        :type nums: List[int]
        :rtype: List[List[int]]
        """
        results = []
        def findNSum(l, target, N, result):
            r = len(nums)-1
            if r+1-l < N or nums[l]*N > target or nums[r]*N < target:
                return
            if N == 2:
                while l < r:
                    s = nums[l] + nums[r]
                    if s == target:
                        results.append(result +[nums[l], nums[r]])
                        l += 1
                        while l< r and nums[l] == nums[l-1]:
                            l +=1
                    elif s< target:
                        l += 1
                    else:
                        r -= 1
            else:
                for i in range(l, len(nums)):
                    if i==l or (i>l and nums[i]!=nums[i-1]):
                        findNSum(i+1, target-nums[i], N-1, result + [nums[i]])

        nums.sort()
        target = 0
        findNSum(0, target, 3, [])
        return results
```

图 1.6 3Sum

1.7 Leetcode 17. Letter Combinations of a Phone Number

Fig 1.7.Given a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent.

A mapping of digit to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.

思路：*cur_combs* = result + s in *map_dict* of *str_now*.

```
class Solution(object):
    def letterCombinations(self, digits):
        """
        :type digits: str
        :rtype: List[str]
        """
        phone_map = {'2':'abc', '3':'def', '4':'ghi', '5':'jkl', '6':'mno', '7':'pqrs', '8':'tuv', '9':'wxyz'}
        result = [''] if len(digits)>0 else []
        for digit in digits:
            map_str = phone_map[digit]
            cur_map = []
            for s in map_str:
                for map_item in result:
                    cur_map.append(map_item+s)
            result = cur_map
        return result
```

图 1.7 Letter Combinations of a Phone Number

1.8 Leetcode 19. Remove Nth Node From End of List

Fig 1.8.Given a linked list, remove the n-th node from the end of list and return its head.

思路：设置 fast 和 slow 两个指针，先让 fast 走 n 步，然后 fast 和 slow 同时走到 fast 是末尾的时候，这个时候 slow 就到了倒数 n 位，然后直接链表的操作. 注意 fast 走完以后是否到了末尾。

1.9 Leetcode 20. Valid Parentheses

Fig 1.9.Given a string containing just the characters '(', ')', '', '', '[' and ']', determine if the input string is valid.

An input string is valid if:

Open brackets must be closed by the same type of brackets. Open brackets must be closed in the correct order. Note that an empty string is also considered valid.

思路: 栈的操作。

```python
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution(object):
    def removeNthFromEnd(self, head, n):
        """
        :type head: ListNode
        :type n: int
        :rtype: ListNode
        """
        fast = slow = head
        for i in range(n):
            fast = fast.next
        if not fast:
            return head.next
        while fast.next:
            fast = fast.next
            slow = slow.next
        slow.next = slow.next.next
        return head
```

图 1.8 Remove Nth Node From End of List

```python
class Solution(object):
    def isValid(self, s):
        """
        :type s: str
        :rtype: bool
        """
        saved_stack = []
        map_dict = {'(':')', '{':'}', '[':']'}
        for item in s:
            if item in map_dict.keys():
                saved_stack.append(item)
            else:
                if saved_stack ==[] or item != map_dict[saved_stack.pop()]:
                    return False
        return saved_stack == []
```

图 1.9 Remove Nth Node From End of List

1.10 Leetcode 21. Merge Two Sorted Lists

Fig 1.10.Merge two sorted linked lists and return it as a new list. The new list should be made by splicing together the nodes of the first two lists.

思路: 链表的操作。

```
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution(object):
    def mergeTwoLists(self, l1, l2):
        """
        :type l1: ListNode
        :type l2: ListNode
        :rtype: ListNode
        """
        result = head = ListNode(0)
        while l1 and l2:
            if l1.val < l2.val:
                result.next = l1
                l1 = l1.next
            else:
                result.next = l2
                l2 = l2.next
            result = result.next
        if l1:
            result.next = l1
        if l2:
            result.next = l2
        return head.next
```

图 1.10 Merge Two Sorted Lists

1.11 Leetcode 33. Search in Rotated Sorted Array

Fig 1.11.Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand.

(i.e., [0,1,2,4,5,6,7] might become [4,5,6,7,0,1,2]).

You are given a target value to search. If found in the array return its index, otherwise return -1.

You may assume no duplicate exists in the array.

Your algorithm's runtime complexity must be in the order of O(log n).

思路: 循环数组的排序问题。循环数组是前后两段升序序列连接在一起，前一段序列的元素永远大于后一段序列的元素。主要先看 median 落在前一段升序序列，还是后一段升序序列。然后再看 target 在特定升序序列中的位置。

```
class Solution(object):
    def search(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: int
        """
        if len(nums) == 0:
            return -1
        left, right = 0, len(nums)-1
        while left<= right:
            median = (left + right)//2
            if target == nums[median]:
                return median
            if nums[left] <= nums[median]:
                if nums[left]<= target < nums[median]:
                    right = median-1
                else:
                    left = median + 1
            else:
                if nums[median] < target <= nums[right]:
                    left = median + 1
                else:
                    right = median -1
        return -1
```

图 1.11 Search in Rotated Sorted Array

1.12 Leetcode 34. Find First and Last Position of Element in Sorted Array

Fig 1.12.Given an array of integers nums sorted in ascending order, find the starting and ending position of a given target value. Your algorithm's runtime complexity must be in the order of O(log n). If the target is not found in the array, return [-1, -1].

思路: 二分法的活用。

```
class Solution(object):
    def searchRange(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: List[int]
        """
        if len(nums) == 0:
            return [-1, -1]
        left, right = 0, len(nums)-1
        while left <=right:
            median = (left + right)//2
            if nums[median] == target:
                left, right = median, median
                while left-1 >=0 and nums[left-1] == target:
                    left = left -1
                while right + 1 < len(nums) and nums[right+1] == target:
                    right = right + 1
                return left, right
            elif nums[median] < target:
                left = median +1
            else:
                right = median -1
        return [-1, -1]
```

图 1.12 Find First and Last Position of Element in Sorted Array

1.13 Leetcode 39. Combination Sum

Fig 1.13.Given a set of candidate numbers (candidates) (without duplicates) and a target number (target), find all unique combinations in candidates where the candidate numbers sums to target. The same repeated number may be chosen from candidates unlimited number of times. Note: All numbers (including target) will be positive integers. The solution set must not contain duplicate combinations.

思路: 和 nsum 思路差不多，递归。注意最后的终止条件，注意剪枝，注意元素是否可以重复。

```
class Solution(object):
    def combinationSum(self, candidates, target):
        """
        :type candidates: List[int]
        :type target: int
        :rtype: List[List[int]]
        """
        results = []
        def findSum(l, target, result):
            if target == 0:
                results.append(result)
            for i in range(l, len(candidates)):
                if candidates[i] > target:
                    return
                findSum(i, target-candidates[i], result +[candidates[i]])
        candidates.sort()
        findSum(0, target, [])
        return results
```

图 1.13 Combination Sum

1.14 Leetcode 46. Permutations

Fig 1.14.Given a collection of distinct integers, return all possible permutations.

思路: 递归的用法。注意不重复元素。

```
class Solution:
    def permute(self, nums):
        """
        :type nums: List[int]
        :rtype: List[List[int]]
        """
        results = []
        def dfs(result, nums):
            if len(nums) == 0:
                return results.append(result)
            for i in range(len(nums)):
                dfs(result+[nums[i]], nums[:i]+nums[i+1:])
        dfs([],nums)
        return results
```

图 1.14 Permutations

1.15 Leetcode 48. Rotate Image

Fig 1.15.You are given an n x n 2D matrix representing an image.

Rotate the image by 90 degrees (clockwise).

思路: 先上下对折颠倒，再对角线转换。

```
class Solution(object):
    def rotate(self, matrix):
        """
        :type matrix: List[List[int]]
        :rtype: None Do not return anything, modify matrix in-place instead.
        """
        n = len(matrix)
        for i in range(n//2):
            for j in range(n):
                matrix[i][j], matrix[n-i-1][j] = matrix[n-i-1][j],matrix[i][j]

        for i in range(n):
            for j in range(i+1, n):
                matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j]
```

图 1.15 Rotate Image

1.16 Leetcode 49. Group Anagrams

Fig 1.16.Given an array of strings, group anagrams together.

思路: 利用字典。

```
class Solution:
    def groupAnagrams(self, strs: 'List[str]') -> 'List[List[str]]':
        group_dict = {}
        for item in strs:
            key = ''.join(sorted(item))
            if key not in group_dict.keys():
                group_dict[key] = []
            group_dict[key].append(item)
        return list(group_dict.values())
```

图 1.16 Group Anagrams

1.17 Leetcode 53. Maximum Subarray

Fig 1.17.Given an integer array nums, find the contiguous subarray (containing at least one number) which has the largest sum and return its sum.

思路:dp 思路。

```
class Solution:
    def maxSubArray(self, nums: 'List[int]') -> 'int':
        dp= [nums[0]]*len(nums)
        for i in range(1, len(nums)):
            if dp[i-1]>0:
                dp[i] = dp[i-1]+nums[i]
            else:
                dp[i] = nums[i]
        return max(dp)
```

图 1.17 max sub

1.18 Leetcode 55. Jump Game

算法如 Fig 1.18.Given an array of non-negative integers, you are initially positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Determine if you are able to reach the last index.

思路:dp 思路。算法如 Fig 1.18.

1.19 Leetcode 62. Unique Paths

算法如 Fig 1.19.A robot is located at the top-left corner of a m x n grid (marked 'Start' in the diagram below).

```
class Solution:
    def canJump(self, nums: 'List[int]') -> 'bool':
        max_reach = 0
        for i in range(len(nums)):
            if max_reach< i : return False
            if max_reach >=len(nums)-1: return True
            max_reach = max(max_reach, i+nums[i])
```

图 1.18 Jump Game

The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked 'Finish' in the diagram below).

How many possible unique paths are there?

思路:dp 思路。

```
class Solution:
    def uniquePaths(self, m: 'int', n: 'int') -> 'int':
        dp = [[0]*n]*m
        for i in range(m):
            for j in range(n):
                if i == 0 and j == 0:
                    dp[i][j] = 1
                elif i == 0 and j !=0:
                    dp[i][j] = dp[i][j-1]
                elif j == 0 and i !=0:
                    dp[i][j] = dp[i-1][j]
                else:
                    dp[i][j] = dp[i][j-1] + dp[i-1][j]
        return dp[m-1][n-1]
```

图 1.19 Unique Paths

1.20 Leetcode 64. Minimum Path Sum

算法如 fig 1.20. Given a m x n grid filled with non-negative numbers, find a path from top left to bottom right which minimizes the sum of all numbers along its path.

Note: You can only move either down or right at any point in time.

思路:dp 思路。

1.21 Leetcode 70. Climbing Stairs

算法如 Fig 1.21.

```
class Solution:
    def minPathSum(self, grid: 'List[List[int]]') -> 'int':
        m, n = len(grid), len(grid[0])
        dp = [[0]*n]*m
        for i in range(m):
            for j in range(n):
                if i == 0 and j == 0:

                    dp[i][j] = grid[i][j]
                elif i==0 and j !=0:

                    dp[i][j] = dp[i][j-1] + grid[i][j]
                elif i!=0 and j ==0 :

                    dp[i][j] = dp[i-1][j] + grid[i][j]
                else:

                    dp[i][j] = min(dp[i-1][j], dp[i][j-1]) + grid[i][j]
        return dp[m-1][n-1]
```

图 1.20 Minimum Path Sum

You are climbing a stair case. It takes n steps to reach to the top. Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top? Note: Given n will be a positive integer.

思路:dp 思路。

1.22 Leetcode 75. Sort Colors

算法如 Fig 1.22.

Given an array with n objects colored red, white or blue, sort them in-place so that objects of the same color are adjacent, with the colors in the order red, white and blue.

Here, we will use the integers 0, 1, and 2 to represent the color red, white, and blue respectively.

Note: You are not suppose to use the library's sort function for this problem.

思路: 利用两个指针计数，一个技术 0 的个数，一个技术 0，1 的个数，2 的个数为全部。并利用先后顺序来 paint，非常巧妙。

1.23 Leetcode 78. Subsets

算法如 Fig 1.23.

Given a set of distinct integers, nums, return all possible subsets(the power set).

Note: The solution set must not contain duplicate subsets.

```python
class Solution:
    def climbStairs(self, n):
        """
        :type n: int
        :rtype: int
        """
        if n == 0:
            return 0
        if n == 1:
            return 1
        if n== 2 :
            return 2
        prepre = 1
        pre = 2
        for i in range(2, n):
            tmp = pre
            pre = prepre + pre
            prepre = tmp
        return pre
```

图 1.21 Climbing Stairs

```python
class Solution(object):
    def sortColors(self, nums):
        """
        :type nums: List[int]
        :rtype: None Do not return anything, modify nums in-place instead.
        """
        i,j = 0, 0
        for k in range(len(nums)):
            tmp = nums[k]
            nums[k] = 2
            if tmp<2:
                nums[i] = 1
                i += 1
            if tmp ==0:
                nums[j] = 0
                j += 1
```

图 1.22 Sort Colors

思路: 递归的思路。

```
class Solution:
    def subsets(self, nums):
        """
        :type nums: List[int]
        :rtype: List[List[int]]
        """
        results = []
        nums.sort()
        def dfs(l, result):
            results.append(result)
            for i in range(l, len(nums)):
                dfs(i+1, result+[nums[i]])
        dfs(0, [])
        return results
```

图 1.23 Subsets

1.24 Leetcode 79. Word Search

算法如 Fig 1.24.

Given a 2D board and a word, find if the word exists in the grid. The word can be constructed from letters of sequentially adjacent cell, where ”adjacent” cells are those horizontally or vertically neighboring. The same letter cell may not be used more than once.

思路: 递归的思路。

```
class Solution:
    def exist(self, board, word):
        """
        :type board: List[List[str]]
        :type word: str
        :rtype: bool
        """
        if not board:
            return False
        for i in range(len(board)):
            for j in range(len(board[0])):
                if self.dfs(board, i, j, word):
                    return True
        return False

        # check whether can find word, start at (i,j) position
    def dfs(self, board, i, j, word):
        if len(word) == 0: # all the characters are checked
            return True
        if i<0 or i>=len(board) or j<0 or j>=len(board[0]) or word[0]!=board[i][j]:
            return False
        tmp = board[i][j]
        board[i][j] = "#"
        res = self.dfs(board, i+1, j, word[1:]) or self.dfs(board, i-1, j, word[1:]) \
        or self.dfs(board, i, j+1, word[1:]) or self.dfs(board, i, j-1, word[1:])
        board[i][j] = tmp
        return res
```

图 1.24 Word Search

1.25 Leetcode 94. Binary Tree Inorder Traversal

算法如 Fig 1.25.

Given a binary tree, return the inorder traversal of its nodes' values.

思路: 利用栈进行中序遍历。

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def inorderTraversal(self, root):
        """
        :type root: TreeNode
        :rtype: List[int]
        """
        stack, result = [], []
        while root or stack:
            while root:
                stack.append(root)
                root = root.left
            node = stack.pop()
            result.append(node.val)
            root = node.right
        return result
```

图 1.25 Binary Tree Inorder Traversal

1.26 Leetcode 96. Unique Binary Search Trees

算法如 Fig 1.26.

Given n, how many structurally unique BST's (binary search trees) that store values 1 ... n?

思路: 利用 dp 思路，当前节点独立树的个数 = 左子树的个数 * 右子树的个数。

1.27 Leetcode 98. Validate Binary Search Tree

算法如 Fig 1.27.

```python
class Solution:
    def numTrees(self, n: int) -> int:
        dp = [1,1,2]
        if n <= 2: return dp[n]
        else:
            #dp = dp + [0 for _ in range(n-2)]
            dp.extend([0 for _ in range(n-2)])
            for i in range(3, n+1):
                for j in range(i):
                    dp[i] += dp[j]*dp[i-j-1]
            return dp[~0]
```

图 1.26 Unique Binary Search Trees

Given a binary tree, determine if it is a valid binary search tree (BST).

Assume a BST is defined as follows:

The left subtree of a node contains only nodes with keys less than the node's key. The right subtree of a node contains only nodes with keys greater than the node's key. Both the left and right subtrees must also be binary search trees.

思路: 二叉搜索树的中序遍历是有序的，二叉搜索树的每一个节点的左子树的节点的值小于当前节点的值，当前节点的值小于右子树的节点的值。

1.28 Leetcode 102. Binary Tree Level Order Traversal

算法如 Fig 1.28.

Given a binary tree, return the level order traversal of its nodes' values. (ie, from left to right, level by level).

思路: 按序将每一层的 node 读取出来。

1.29 Leetcode 104. Maximum Depth of Binary Tree

算法如 Fig 1.29.

Given a binary tree, find its maximum depth.

The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

Note: A leaf is a node with no children.

思路: 利用广度优先搜索遍历树，来探索树到底有多少层。

```
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution(object):
    def isValidBST(self, root):
        """
        :type root: TreeNode
        :rtype: bool
        """
        stack, result = [], []
        while root or stack:
            while root:
                stack.append(root)
                root = root.left
            node = stack.pop()
            result.append(node.val)
            root = node.right
        for i in range(1, len(result)):
            if result[i]<=result[i-1]:
                return False
        return True
```

图 1.27 Validate Binary Search Tree

1.30 Leetcode 105. Construct Binary Tree from Preorder and Inorder Traversal

算法如 Fig 1.30.

Given preorder and inorder traversal of a tree, construct the binary tree.

Note: You may assume that duplicates do not exist in the tree.

思路: 前序遍历的元素在中序遍历的数组中，会出现左边的全部是其左子树，右边全部是其右子树。

1.31 Leetcode 121. Best Time to Buy and Sell Stock

算法如 Fig 1.31.

Say you have an array for which the ith element is the price of a given stock on day i.

```
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution(object):
    def inorderTraversal(self, root):
        """
        :type root: TreeNode
        :rtype: List[int]
        """
        stack, result = [], []
        while stack or root:
            while root:
                stack.append(root)
                root = root.left
            node = stack.pop()
            result.append(node.val)
            root = node.right
        return result
```

图 1.28 Binary Tree Level Order Traversal

If you were only permitted to complete at most one transaction (i.e., buy one and sell one share of the stock), design an algorithm to find the maximum profit.

Note that you cannot sell a stock before you buy one.

思路: 利用遍历的思想找最小 price 和最大 profit。

1.32 Leetcode 136. Single Number

算法如 Fig 1.32.

Given a non-empty array of integers, every element appears twice except for one. Find that single one.

Note:

Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?

思路: 异或操作。

```python
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution(object):
    def maxDepth(self, root):
        """
        :type root: TreeNode
        :rtype: int
        """
        if not root:
            return 0
        level, depth =[root], 1
        while level:
            tmp = []
            for node in level:
                if node.left:
                    tmp.append(node.left)
                if node.right:
                    tmp.append(node.right)
            if len(tmp)!=0:
                depth += 1
            level = tmp
        return depth
```

图 1.29 Maximum Depth of Binary Tree

```python
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution(object):
    def buildTree(self, preorder, inorder):
        """
        :type preorder: List[int]
        :type inorder: List[int]
        :rtype: TreeNode
        """
        if inorder:
            ind = inorder.index(preorder.pop(0))
            root = TreeNode(inorder[ind])
            root.left = self.buildTree(preorder, inorder[:ind])
            root.right = self.buildTree(preorder, inorder[ind+1:])
            return root
```

图 1.30 Construct Binary Tree from Preorder and Inorder Traversal

```
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        max_profit, min_price = 0, float('inf')
        for price in prices:
            min_price = min(price, min_price)
            profit = price - min_price
            max_profit = max(max_profit, profit)
        return max_profit
```

图 1.31 Best Time to Buy and Sell Stock

```
class Solution(object):
    def singleNumber(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        result = nums[0]
        for i in range(1,len(nums)):
            result ^= nums[i]
        return result
```

图 1.32 Single Number

1.33 Leetcode 139. Word Break

算法如 Fig 1.33.

Given a non-empty string s and a dictionary wordDict containing a list of non-empty words, determine if s can be segmented into a space-separated sequence of one or more dictionary words.

Note:

The same word in the dictionary may be reused multiple times in the segmentation. You may assume the dictionary does not contain duplicate words.

思路: 利用 dp 的思路，遍历 s，用 dp 数组记录到遍历某一个字符时，该字符之前的字符串是否在 wordDict 中。

```
class Solution:
    def wordBreak(self, s: str, wordDict: List[str]) -> bool:
        dp = [False for i in range(len(s)+1)]
        dp[0] = True
        for i in range(1,len(s)+1):
            for k in range(0,i):
                if dp[k] and s[k:i] in wordDict:
                    dp[i] = True
        return dp[len(s)]
```

图 1.33 Word Break

1.34 Leetcode 141. Linked List Cycle

算法如 Fig 1.34.

Given a non-empty string s and a dictionary wordDict containing a list of non-empty words, determine if s can be segmented into a space-separated sequence of one or more dictionary words.

Note:

The same word in the dictionary may be reused multiple times in the segmentation. You may assume the dictionary does not contain duplicate words.

思路: 首先使用快慢指针技巧，如果 fast 指针和 slow 指针相遇，则说明链表存在环路。

```
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution(object):
    def hasCycle(self, head):
        """
        :type head: ListNode
        :rtype: bool
        """
        slow = fast = head
        while fast and fast.next:
            fast = fast.next.next
            slow = slow.next
            if slow == fast:
                return True
        return False
```

图 1.34 Linked List Cycle

1.35 Leetcode 142. Linked List Cycle II

算法如 Fig 1.35.

Given a non-empty string s and a dictionary wordDict containing a list of non-empty words, determine if s can be segmented into a space-separated sequence of one or more dictionary words.

Note:

The same word in the dictionary may be reused multiple times in the segmentation. You may assume the dictionary does not contain duplicate words.

思路: 首先使用快慢指针技巧，如果 fast 指针和 slow 指针相遇，则说明链表存在环路。

在 fast 指针和 slow 指针相遇后，fast 指针不动，slow 指针回到 head，然后 slow 指针和 fast 指针同时向前走，只不过这一次两个指针都是一步一步向前走。两个指针相遇的节点就是环路的起点。

```python
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution(object):
    def detectCycle(self, head):
        """
        :type head: ListNode
        :rtype: ListNode
        """
        try:
            fast = head.next
            slow = head
            while fast is not slow:
                fast = fast.next.next
                slow = slow.next
        except:
            # if there is an exception, we reach the end and there is no cycle
            return None

        slow = slow.next
        while head is not slow:
            head = head.next
            slow = slow.next

        return head
```

图 1.35 Linked List Cycle II

1.36 Leetcode 148. Sort List

算法如 Fig 1.36.

Sort a linked list in O(n log n) time using constant space complexity.

思路: 这个题要求用 O(nlongn) 的时间复杂度和 O(1) 的空间复杂度。所以可以使用 merge 排序，但是如果是链表可以修改指针，把两个有序链表进行原地的合并。

Merge 排序就是先划分成一前一后等分的两块，然后对两块分别进行排序，然后再合并两个有序序列。

第一步，如何等分地划分，可以使用快慢指针的方式，当快指针到达结尾，那么慢指针到了中间位置，把链表进行截断分成了两个。

第二步，合并有序的序列，对于单链表来说，正好用到了 Merge Two Sorted Lists 里的把两个链表合并的方法。

事实上，这个答案里面并不是 O(1) 的空间，因为，第一，添加了新的链表头的个数会随着递归的次数而不断增加，并不是常量个；第二，递归本身就不是常量空间。

```python
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def sortList(self, head: ListNode) -> ListNode:
        if not head or not head.next:
            return head
        fast,slow = head.next, head
        while fast and fast.next:
            fast = fast.next.next
            slow = slow.next
        second = slow.next
        slow.next = None

        l = self.sortList(head)
        r = self.sortList(second)
        return self.merge(l, r)

    def merge(self, l, r):
        dummy = ListNode(None)
        node = dummy
        while l and r:
            if l.val<r.val:
                node.next = l
                l = l.next
            else:
                node.next = r
                r = r.next
            node = node.next
        node.next = l or r
        return dummy.next
```

图 1.36 Sort List

1.37 Leetcode 152. Maximum Product Subarray

算法如 Fig 1.37.

Given an integer array nums, find the contiguous subarray within an array (containing at least one number) which has the largest product

思路: 如果是连续子数组的和的问题我们肯定能想到虫取法之类的，但是求积就比较麻烦了，因为某个位置可能出现了 0 或者负数。。当遇到 0 的时候，整个乘积会变成 0；当遇到负数的时候，当前的最大乘积会变成最小乘积，最小乘积会变成最大乘积。当前的最大值等于已知的最大值、最小值和当前值的乘积，当前值，这三个数的最大值。当前的最小值等于已知的最大值、最小值和当前值的乘积，当前值，这三个数的最小值。结果是最大值数组中的最大值。

```python
class Solution(object):
    def maxProduct(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if len(nums)==0:
            return []
        max_pro = 1
        f = [0]*len(nums)
        g = [0]*len(nums)
        f[0] = g[0] = res = nums[0]
        for i in range(1, len(nums)):
            f[i] = min(f[i-1]*nums[i], nums[i], g[i-1]*nums[i])
            g[i] = max(f[i-1]*nums[i], nums[i], g[i-1]*nums[i])
            res = max(g[i], res)
        return res
```

图 1.37 Maximum Product Subarray

1.38 Leetcode 155. Min Stack

算法如 Fig 1.38.

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

push(x) – Push element x onto stack. pop() – Removes the element on top of the stack. top() – Get the top element. getMin() – Retrieve the minimum element in the stack.

思路: 定义栈里每个元素为一个元组 (x, cur_min)，表示当前元素和截至当前元素时的最小值。不管栈怎么变化，总可以常数时间执行四种操作。本思路也可以用两个栈来实现，一个栈存放正常的元素，一个栈存放当前最小值，两个栈同步操作即可。

1.39 Leetcode 160. Intersection of Two Linked Lists

算法如 Fig 1.39.

Write a program to find the node at which the intersection of two singly linked lists begins.

思路: 假设两个链表有交叉，那么交叉后的长度是一样的，而交叉前的长度可能不一致。如果我们将两个链表的交叉前的长度截成一致，那么就可以同时遍历两个链表，判断是否有相交节点。截断交叉前的不同长度等价于截断两个完整链表的不同长度。此方法需要计算两个链表长度。

1.40 Leetcode 169. Majority Element

算法如 Fig 1.40.

```
class MinStack(object):

    def __init__(self):
        """
        initialize your data structure here.
        """
        self.stack = []

    def push(self, x):
        """
        :type x: int
        :rtype: None
        """
        pre_min = 2147483647 if len(self.stack) == 0 else self.stack[-1][-1]
        cur_min = min(x, pre_min)
        self.stack.append((x, cur_min))


    def pop(self):
        """
        :rtype: None
        """
        return self.stack.pop()


    def top(self):
        """
        :rtype: int
        """
        return self.stack[-1][0]


    def getMin(self):
        """
        :rtype: int
        """
        return self.stack[-1][1]
```

图 1.38 Min Stack

```
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution(object):
    def getIntersectionNode(self, headA, headB):
        """
        :type head1, head1: ListNode
        :rtype: ListNode
        """
        curA,curB = headA,headB
        lenA,lenB = 0,0
        while curA is not None:
            lenA += 1
            curA = curA.next
        while curB is not None:
            lenB += 1
            curB = curB.next
        curA,curB = headA,headB
        if lenA > lenB:
            for i in range(lenA-lenB):
                curA = curA.next
        elif lenB > lenA:
            for i in range(lenB-lenA):
                curB = curB.next
        while curB != curA:
            curB = curB.next
            curA = curA.next
        return curA
```

图 1.39 Intersection of Two Linked Lists

Given an array of size n, find the majority element. The majority element is the element that appears more than ⌊n/2⌋ times.

You may assume that the array is non-empty and the majority element always exist in the array. 思路: “投票算法”，设定两个变量 candidate 和 count。candidate 保存当前可能的候选众数，count 保存该候选众数的出现次数。遍历数组 num。如果当前的数字 e 与候选众数 candidate 相同，则将计数 count + 1 否则，如果当前的候选众数 candidate 为空，或者 count 为 0，则将候选众数 candidate 的值置为 e，并将计数 count 置为 1。否则，将计数 count - 1

最终留下的候选众数 candidate 即为最终答案。

```
class Solution(object):
    def majorityElement(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        count = 0
        for num in nums:
            if count == 0:
                key = num
                count = 1
            else:
                if key == num:
                    count += 1
                else:
                    count -= 1
        return key
```

图 1.40 Majority Element

1.41 Leetcode 198. House Robber

算法如 Fig 1.41.

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security system connected and it will automatically contact the police if two adjacent houses were broken into on the same night.

Given a list of non-negative integers representing the amount of money of each house,

determine the maximum amount of money you can rob tonight without alerting the police. 思路: 整体的思路是当前房间偷和不偷两个状态，如果偷就加上前面第二个偷的商品的状态，如果不偷就是前面一个房间的状态。

```
class Solution(object):
    def rob(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        left = left_left = 0
        for i in range(len(nums)):
            left, left_left =  max(nums[i] + left_left, left), left
        return left
```

图 1.41 House Robber

1.42 Leetcode 200. Number of Islands

算法如 Fig 1.42.

Given a 2d grid map of '1's (land) and '0's (water), count the number of islands. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water. 思路: 如果不懂方法还是很难做的，这个题在《挑战程序设计竞赛》书的前面就讲了，我觉得还是挺经典的题目。

做法是，我们对每个有“1” 的位置进行 dfs，把和它四联通的位置全部变成“0”，这样就能把一个点推广到一个岛。

所以，我们总的进行了 dfs 的次数，就是总过有多少个岛的数目。

注意理解 dfs 函数的意义：已知当前是 1，把它周围相邻的所有 1 全部转成 0.

1.43 Leetcode 206. Reverse Linked List

算法如 Fig 1.43.

Reverse a singly linked list.

思路: 通过迭代将节点重组，前面的节点转移到重组链表的后面。实际上就是头结点倒插操作。

```python
class Solution(object):
    def numIslands(self, grid):
        """
        :type grid: List[List[str]]
        :rtype: int
        """
        count = 0
        for i in range(len(grid)):
            for j in range(len(grid[0])):
                if grid[i][j] == "1":
                    count+=1
                    self.dfs(i, j, grid)
        return count

    def dfs(self, i, j, grid):
        grid[i][j] = "0"
        if i-1>=0 and grid[i-1][j] == "1":
            self.dfs(i-1, j, grid)
        if i+1<len(grid) and grid[i+1][j] == "1":
            self.dfs(i+1, j, grid)
        if j-1>=0 and grid[i][j-1] == "1":
            self.dfs(i, j-1, grid)
        if j+1<len(grid[0]) and grid[i][j+1] == "1":
            self.dfs(i, j+1, grid)
```

图 1.42 Number of Islands

```python
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution(object):
    def reverseList(self, head):
        """
        :type head: ListNode
        :rtype: ListNode
        """
        prev = None
        cur = head
        while cur:
            next = cur.next
            cur.next = prev
            prev = cur
            cur = next
        return prev
```

图 1.43 Reverse Linked List

1.44 Leetcode 207. Course Schedule

算法如 Fig 1.44.

There are a total of n courses you have to take, labeled from 0 to n-1. Some courses may have prerequisites, for example to take course 0 you have to first take course 1, which is expressed as a pair: [0,1] Given the total number of courses and a list of prerequisite pairs, is it possible for you to finish all courses?

思路: 使用 DFS。这个方法是，我们每次找到一个新的点，判断从这个点出发是否有环。具体做法是使用一个 visited 数组，当 visited[i] 值为 0，说明还没判断这个点；当 visited[i] 值为 1，说明当前的循环正在判断这个点；当 visited[i] 值为 2，说明已经判断过这个点，含义是从这个点往后的所有路径都没有环，认为这个点是安全的。那么，我们对每个点出发都做这个判断，检查这个点出发的所有路径上是否有环，如果判断过程中找到了当前的正在判断的路径，说明有环；找到了已经判断正常的点，说明往后都不可能存在环，所以认为当前的节点也是安全的。如果当前点是未知状态，那么先把当前点标记成正在访问状态，然后找后续的节点，直到找到安全的节点为止。最后如果到达了无路可走的状态，说明当前节点是安全的。

```
class Solution(object):
    def canFinish(self, numCourses, prerequisites):
        """
        :type numCourses: int
        :type prerequisites: List[List[int]]
        :rtype: bool
        """
        graph = collections.defaultdict(list)
        for u, v in prerequisites:
            graph[u].append(v)
        visited = [0]*numCourses
        for i in range(numCourses):
            if not self.dfs(graph, i, visited):
                return False
        return True

    def dfs(self, graph, i, visited):
        if visited[i] == 1: return False
        if visited[i] == 2: return True
        visited[i] = 1
        for j in graph[i]:
            if not self.dfs(graph, j, visited):
                return False
        visited[i] = 2
        return True
```

图 1.44 Course Schedule

1.45 Leetcode 208. Implement Trie (Prefix Tree)

算法如 Fig 1.45.

Implement a trie with insert, search, and startsWith methods.

思路: 定义数据结构的孩子的时候，可以使用字典，也可以使用数组。下面的 python 实现用的字典，而 C++ 实现用的数组。每个节点的子孩子都是一个字典，根据字典查找下一个位置的节点，就像字典一样。同时用 isword 保存当前是不是一个词（也可能是路径中的点）。

```
    def __init__(self):
        self.children = collections.defaultdict(Node)
        self.isword = False

class Trie(object):

    def __init__(self):
        """
        Initialize your data structure here.
        """
        self.root = Node()

    def insert(self, word):
        """
        Inserts a word into the trie.
        :type word: str
        :rtype: void
        """
        current = self.root
        for w in word:
            current = current.children[w]
        current.isword = True

    def search(self, word):
        """
        Returns if the word is in the trie.
        :type word: str
        :rtype: bool
        """
        current = self.root
        for w in word:
            current = current.children.get(w)
            if current == None:
                return False
        return current.isword

    def startsWith(self, prefix):
        """
        Returns if there is any word in the trie that starts with the given prefix.
        :type prefix: str
        :rtype: bool
        """
        current = self.root
        for w in prefix:
            current = current.children.get(w)
            if current == None:
                return False
        return True
```

图 1.45 Implement Trie

1.46 Leetcode 215. Kth Largest Element in an Array

算法如 Fig 1.46.

Find the kth largest element in an unsorted array. Note that it is the kth largest element in the sorted order, not the kth distinct element.

思路: 注意是从后往前数，还是从前往后数第 k 个值。快排的思路。

```
class Solution:
    def findKthLargest(self, nums, k):
        pos = self.partition(nums, 0, len(nums)-1)
        if pos > len(nums) - k:
            return self.findKthLargest(nums[:pos], k-(len(nums)-pos))
        elif pos < len(nums) - k:
            return self.findKthLargest(nums[pos+1:], k)
        else:
            return nums[pos]

# Lomuto partition scheme
    def partition(self, nums, l, r):
        pivot = nums[l]
        lo = l+1
        for i in range(l+1, r+1):
            if nums[i] < pivot:
                nums[i], nums[lo] = nums[lo], nums[i]
                lo += 1
        nums[lo-1], nums[l] = nums[l], nums[lo-1]
        return lo-1
```

图 1.46 Kth Largest Element in an Array

1.47 Leetcode 221. Maximal Square

算法如 Fig 1.47.

Given a 2D binary matrix filled with 0's and 1's, find the largest square containing only 1's and return its area.

思路: 设这个 DP[i][j] 数组为以 i, j 位置为右下角顶点的能够成的最大正方形的边长。数组如果是第一行或者第一列，显然 dp 和 matrix 相等。如果是其他位置，当 matrix[i][j] = 1 时，能够成的正方形等于左边、上边、左上能够成的正方形边长的最小值 +1. 为什么是最小值? 因为只要存在一个 0，那么就没法构成更大的正方形，这个是很保守的策略。

```
class Solution(object):
    def maximalSquare(self, matrix):
        """
        :type matrix: List[List[str]]
        :rtype: int
        """
        if not matrix or not matrix[0]: return 0
        M, N, sideLen = len(matrix), len(matrix[0]), [[1 if ch == '1' else 0 for ch in row] for row in matrix]
        for i in range(1, M):
            for j in range(1, N):
                if matrix[i][j] == '1':
                    sideLen[i][j] = 1 + min(sideLen[i - 1][j], sideLen[i][j - 1], sideLen[i - 1][j - 1])
        return max(max(row) for row in sideLen) ** 2
```

图 1.47 Maximal Square

1.48 Leetcode 226. Invert Binary Tree

算法如 Fig 1.48.

Invert a binary tree.

思路: （BFS）除了上面的用栈的解法，用队列也可以解决该问题。先将根节点入队，交换左右节点并将非空的节点加入队列，再将根节点出队，这样循环下去即可。

```
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution(object):
    def invertTree(self, root):
        """
        :type root: TreeNode
        :rtype: TreeNode
        """
        if root == None:
            return None
        stack = [root]
        while stack:
            node = stack.pop()
            node.left, node.right = node.right, node.left
            if node.left:
                stack.append(node.left)
            if node.right:
                stack.append(node.right)
        return root
```

图 1.48 Invert Binary Tree

1.49 Leetcode 234. Palindrome Linked List

算法如 Fig 1.49.

Given a singly linked list, determine if it is a palindrome.

思路: 判断回文主要是前半部分和后半部分的比较，若能将前半部分压栈，再依次出栈与后半部分比较，则可判断是否回文。

1.50 Leetcode 236. Lowest Common Ancestor of a Binary Tree

算法如 Fig 1.50.

Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree.

According to the definition of LCA on Wikipedia: “The lowest common ancestor is defined between two nodes p and q as the lowest node in T that has both p and q as descendants (where we allow a node to be a descendant of itself).”

```
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution(object):
    def isPalindrome(self, head):
        """
        :type head: ListNode
        :rtype: bool
        """
        if not head or not head.next:
            return True

        new_list = []

        # 快慢指针法找链表的中点
        slow = fast = head
        while fast and fast.next:
            new_list.append(slow.val)
            slow = slow.next
            fast = fast.next.next

        if fast: # 链表有奇数个节点
            slow = slow.next

        while new_list:
            val = new_list.pop()
            if val != slow.val:
                return False
            slow = slow.next
        return True
```

图 1.49 Palindrome Linked List

Given the following binary tree: root = [3,5,1,6,2,0,8,null,null,7,4]

思路: 利用先序遍历。若 root 为空或者 root 为 A 或者 root 为 B，说明找到了 A 和 B 其中一个。若左子树找到了 A，右子树找到了 B，说明此时的 root 就是公共祖先。若左子树是 none 右子树不是，说明右子树找到了 A 或 B。同理若右子树是 none 左子树不是，说明左子树找到了 A 或 B。

```
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution(object):
    def lowestCommonAncestor(self, root, p, q):
        """
        :type root: TreeNode
        :type p: TreeNode
        :type q: TreeNode
        :rtype: TreeNode
        """
        if root == p or root == q or not root:
            return root
        left = self.lowestCommonAncestor(root.left, p, q)
        right = self.lowestCommonAncestor(root.right, p, q)
        if left and right:
            return root
        if not left:
            return right
        if not right:
            return left
        return None
```

图 1.50 Lowest Common Ancestor of a Binary Tree

1.51 Leetcode 238. Product of Array Except Self

算法如 Fig 1.51.

Given an array nums of n integers where n > 1, return an array output such that output[i] is equal to the product of all the elements of nums except nums[i].

思路: 这个题巧妙的地方在于，结果数组不算作空间复杂度里，所以可以用在结果数组中遍历的方式去做。第一次遍历在结果数组里保存每个数字左边的数字乘积，第二个遍历保存的是左边乘积和这个数字右边的乘积的乘积。

1.52 Leetcode 240. Search a 2D Matrix II

算法如 Fig 1.52.

```python
class Solution(object):
    def productExceptSelf(self, nums):
        """
        :type nums: List[int]
        :rtype: List[int]
        """
        if nums == []:
            return []
        result = [None]*len(nums)
        result[0] = 1
        for i in range(1, len(nums)):
            result[i] = result[i-1]*nums[i-1]
        tmp = 1
        for i in range(len(nums)-2, -1, -1):
            tmp *= nums[i+1]
            result[i] *= tmp
        return result
```

图 1.51 Product of Array Except Self

Write an efficient algorithm that searches for a value in an m x n matrix. This matrix has the following properties:

Integers in each row are sorted in ascending from left to right. Integers in each column are sorted in ascending from top to bottom. 思路: 方法是从右上角向左下角进行遍历，根据比较的大小决定向下还是向左查找。剑指 offer 的解释是我们从矩阵的左下角或者右上角开始遍历，这样知道了比较的结果是大还是小，就知道了对应的前进方向。

```python
class Solution(object):
    def searchMatrix(self, matrix, target):
        """
        :type matrix: List[List[int]]
        :type target: int
        :rtype: bool
        """
        if not matrix:
            return False

        row, col = 0, len(matrix[0])-1
        while row<len(matrix) and col>=0:
            if matrix[row][col] == target:
                return True
            elif matrix[row][col]<target:
                row += 1
            else:
                col -= 1
        return False
```

图 1.52 Search a 2D Matrix II

1.53 Leetcode 279. Perfect Squares

算法如 Fig 1.53.

Given a positive integer n, find the least number of perfect square numbers (for example, 1, 4, 9, 16, ...) which sum to n.

思路: 如果一个数 x 可以表示为一个任意数 a 加上一个平方数 b x b，也就是 x=a+b x b，那么能组成这个数 x 最少的平方数个数，就是能组成 a 最少的平方数个数加上 1（因为 b*b 已经是平方数了）。

```
class Solution(object):
    def numSquares(self, n):
        """
        :type n: int
        :rtype: int
        """
        dp = [n]*(n+1)
        dp[0] = 0
        dp[1] = 1
        for i in range(2, n+1):
            j = 1
            while j*j <=i:
                dp[i] = min(dp[i], dp[i-j*j]+1)
                j += 1
        return dp[-1]
```

图 1.53 Perfect Squares

1.54 Leetcode 283. Move Zeroes

算法如 Fig 1.54.

Given an array nums, write a function to move all 0's to the end of it while maintaining the relative order of the non-zero elements.

思路: 先将所有的非零数移动到前面，再一次性将后面所有数置为 0。这样需要用到两个循环。

1.55 Leetcode 287. Find the Duplicate Number

算法如 Fig 1.55.

Given an array nums containing n + 1 integers where each integer is between 1 and n (inclusive), prove that at least one duplicate number must exist. Assume that there is only

```python
class Solution(object):
    def moveZeroes(self, nums):
        """
        :type nums: List[int]
        :rtype: None Do not return anything, modify nums in-place instead.
        """
        i = 0
        for num in nums:
            if num!=0:
                nums[i] = num
                i += 1

        for j in range(i, len(nums)):
            nums[j] = 0
        return nums
```

图 1.54 Move Zeroes

one duplicate number, find the duplicate one. 思路: 找到中间数 mid, 然后遍历数组，如果大于中间数的数的个数，那么说明重复的数字在前半段，我们类似递归的思想，再次在前半段执行，同样如果小于说明在后半段出现。这里注意我们是和 mid 比较而不是 nums[mid]，因为数字固定是在 0 n 之间的数字。

```python
class Solution(object):
    def findDuplicate(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        l, r = 0, len(nums)-1
        while l<=r:
            m = l+(r-l)//2
            count = 0
            for i in range(len(nums)):
                if nums[i]<=m:
                    count +=1
            if count>m:
                r = m-1
            else:
                l = m + 1
        return l
```

图 1.55 Find the Duplicate Number

1.56 Leetcode 300. Longest Increasing Subsequence

算法如 Fig 1.56.

Given an unsorted array of integers, find the length of longest increasing subsequence.

思路: 我们维护一个一维 dp 数组，其中 dp[i] 表示以 nums[i] 为结尾的最长递增子串的长度，对于每一个 nums[i]，我们从第一个数再搜索到 i，如果发现某个数小于 nums[i]，

我们更新 dp[i]，更新方法为 dp[i] = max(dp[i], dp[j] + 1)，即比较当前 dp[i] 的值和那个小于 num[i] 的数的 dp 值加 1 的大小，我们就这样不断的更新 dp 数组，到最后 dp 数组中最大的值就是我们要返回的 LIS 的长度。

```python
class Solution(object):
    def lengthOfLIS(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if not nums: return 0
        dp = [0] * len(nums)
        dp[0] = 1
        for i in range(1, len(nums)):
            tmax = 1
            for j in range(0, i):
                if nums[i] > nums[j]:
                    tmax = max(tmax, dp[j] + 1)
            dp[i] = tmax
        return max(dp)
```

图 1.56 Longest Increasing Subsequence

1.57 Leetcode 309. Best Time to Buy and Sell Stock with Cooldown

算法如 Fig 1.57.

Say you have an array for which the ith element is the price of a given stock on day i.

Design an algorithm to find the maximum profit. You may complete as many transactions as you like (ie, buy one and sell one share of the stock multiple times) with the following restrictions:

You may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again). After you sell your stock, you cannot buy stock on next day. (ie, cooldown 1 day)

思路: 引入辅助数组 sells 和 holds

sell[i] 表示在第 i 天不持有股票所能获得的最大累计收益 hold[i] 表示在第 i 天持有股票所能获得的最大累计收益

状态转移方程：sell[i] = max(sell[i - 1], hold[i - 1] + prices[i])–今天卖了股票，或者啥也没干 hold[i] = max(hold[i - 1], sell[i - 2] - prices[i])–今天持有了股票或者啥也没干

```python
class Solution(object):
    def maxProfit(self, prices):
        """
        :type prices: List[int]
        :rtype: int
        """
        if not prices: return 0
        sell = [0] * len(prices)
        hold = [0] * len(prices)
        hold[0] = -prices[0]
        for i in range(1, len(prices)):
            sell[i] = max(sell[i - 1], hold[i - 1] + prices[i])
            hold[i] = max(hold[i - 1], (sell[i - 2] if i >= 2 else 0) - prices[i])
        return sell[-1]
```

图 1.57 Best Time to Buy and Sell Stock with Cooldown

1.58 Leetcode 322. Coin Change

算法如 Fig 1.58.

You are given coins of different denominations and a total amount of money amount. Write a function to compute the fewest number of coins that you need to make up that amount. If that amount of money cannot be made up by any combination of the coins, return -1.

思路: 有一堆不同面额的硬币，问最少取多少枚硬币，可以凑够想要的面值，每种硬币数无限。假设 dp[i] 表示凑够 i 元所需要的最少硬币数，一共有 n 种面值硬币，那么 dp[i]=min(dp[i−coins[0]],dp[i−coins[1]],...dp[i−coins[k])+1，其中 coins[k]<=i 。

```python
class Solution(object):
    def coinChange(self, coins, amount):
        """
        :type coins: List[int]
        :type amount: int
        :rtype: int
        """
        n = len(coins)
        dp = [float("inf")]*(amount+1)
        dp[0] = 0
        for i in range(1, amount+1):
            for n in coins:
                if n<=i:
                    dp[i] = min(dp[i], dp[i-n]+1)
        return dp[amount] if dp[amount]<=amount else -1
```

图 1.58 Coin Change

1.59 Leetcode 338. Counting Bits

算法如 Fig 1.59.

Given a non negative integer number num. For every numbers i in the range 0 i num calculate the number of 1's in their binary representation and return them as an array.

思路: 将最低位的 1 去掉得到较小的数, 一直减到 0，可以得到所有的小于等于 num 的数的 1 的个数。

```python
class Solution(object):
    def countBits(self, num):
        """
        :type num: int
        :rtype: List[int]
        """
        dp = [0]
        for i in range(1, num+1):
            dp.append(dp[i&(i-1)]+1)
        return dp
```

图 1.59 Counting Bits

1.60 Leetcode 347. Top K Frequent Elements

算法如 Fig 1.60.

Given a non-empty array of integers, return the k most frequent elements.

思路: 先用 dict 得到所有不同数的个数；再对个数排序，取前 k 个个数最多的对应的数即可。

```python
class Solution(object):
    def topKFrequent(self, nums, k):
        """
        :type nums: List[int]
        :type k: int
        :rtype: List[int]
        """
        used_char = {}
        for num in nums:
            used_char[num] = used_char.get(num, 0) + 1
        res = list(used_char.items())
        res.sort(key=lambda x:x[1], reverse=True)
        return [res[i][0] for i in range(k)]
```

图 1.60 Top K Frequent Elements

1.61 Leetcode 416. Partition Equal Subset Sum

算法如 Fig 1.61.

Given a non-empty array containing only positive integers, find if the array can be partitioned into two subsets such that the sum of elements in both subsets is equal.

Note:

Each of the array element will not exceed 100. The array size will not exceed 200.

思路: 和 target sum 的方法一样。

```
class Solution(object):
    def canPartition(self, nums):
        """
        :type nums: List[int]
        :rtype: bool
        """
        numSum=sum(nums)
        if numSum%2!=0:
            return False
        dp=[0]*(numSum+1)
        dp[0]=1

        for num in nums:
            for i in range(numSum,num-1,-1):
                dp[i] = dp[i] or dp[i-num]
            if dp[int(numSum/2)]==1:
                return True
        return False
```

图 1.61 Partition Equal Subset Sum

1.62 Leetcode 437. Path Sum III

算法如 Fig 1.62.

You are given a binary tree in which each node contains an integer value.

Find the number of paths that sum to a given value.

The path does not need to start or end at the root or a leaf, but it must go downwards (traveling only from parent nodes to child nodes).

The tree has no more than 1,000 nodes and the values are in the range -1,000,000 to 1,000,000.

思路: 使用 DFS 解决。dfs 函数有两个参数，一个是当前的节点，另一个是要得到的值。当节点的值等于要得到的值的时候说明是一个可行的解。再求左右的可行的解的个数，求和之后是所有的。

1.63 Leetcode 438. Find All Anagrams in a String

算法如 Fig 1.63.

```python
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution(object):
    def pathSum(self, root, sum):
        """
        :type root: TreeNode
        :type sum: int
        :rtype: int
        """
        if not root: return 0
        return self.dfs(root, sum) + self.pathSum(root.left, sum) + self.pathSum(root.right, sum)

    def dfs(self, root, sum):
        res = 0
        if not root: return res
        sum -= root.val
        if sum == 0:
            res += 1
        res += self.dfs(root.left, sum)
        res += self.dfs(root.right, sum)
        return res
```

图 1.62 Path Sum III

Given a string s and a non-empty string p, find all the start indices of p's anagrams in s.

Strings consists of lowercase English letters only and the length of both strings s and p will not be larger than 20,100.

The order of output does not matter.

思路: 判断两个字符串是否是排列组合直接统计词频然后 == 判断即可. 使用的是双指针的解法, 用了一个滑动窗口，每次进入窗口的字符的个数 +1，超出滑动窗口的字符个数-1.

1.64 Leetcode 448. Find All Numbers Disappeared in an Array

算法如 Fig 1.64.

Given an array of integers where 1 a[i] n (n = size of array), some elements appear twice and others appear once.

Find all the elements of [1, n] inclusive that do not appear in this array.

Could you do it without extra space and in O(n) runtime? You may assume the returned list does not count as extra space.

思路: 注意看这题数组元素的值全都在数组索引的范围内，那么肯定就是套用那个经典的给元素值赋正负来表达额外信息的套路了。而这题就是遍历每个元素，把与元素值相等（实际差个 1）的那个索引对应的元素赋值为负数，然后遍历一遍完成后，再次遍历，如果某个索引对应的元素值是正数，就说明与这个索引相同的值在数组中不存在，就是结果了。

```python
class Solution(object):
    def findAnagrams(self, s, p):
        """
        :type s: str
        :type p: str
        :rtype: List[int]
        """
        count = collections.Counter()
        m, n = len(s), len(p)
        l, r = 0, 0
        p_count = collections.Counter(p)
        result = []
        while r< m:
            count[s[r]] += 1
            if r-l+1 == n:
                if count == p_count:
                    result.append(l)
                count[s[l]] -= 1
                if count[s[l]] == 0:
                    del count[s[l]]
                l += 1
            r+=1
        return result
```

图 1.63 Find All Anagrams in a String

```python
class Solution(object):
    def findDisappearedNumbers(self, nums):
        """
        :type nums: List[int]
        :rtype: List[int]
        """
        for num in nums:
            ind = num if num>0 else -num
            ind -= 1
            if nums[ind]>0:
                nums[ind] = -nums[ind]
        result = []
        for i in range(len(nums)):
            if nums[i]>0:
                result.append(i+1)
        return result
```

图 1.64 Find All Numbers Disappeared in an Array

1.65 Leetcode 494. Target Sum

算法如 Fig 1.65.

You are given a list of non-negative integers, a1, a2, ..., an, and a target, S. Now you have 2 symbols + and -. For each integer, you should choose one from + and - as its new symbol.

Find out how many ways to assign symbols to make sum of integers equal to target S.

思路: 这道题中我们加正负号无非是将 nums 分为两个子集 p,n，p 中元素全部加正号，n 中元素全部加负号，使得 sum(p) - sum(n) = S，而本身又有 sum(p) + sum(n) = sum(nums)，故两式相加化简得 sum(p) = (sum(nums)+S) / 2 那么这个式子直接给出了一个信息，也就是如果能找到 p，则必有 sum(nums)+S% 2 == 0 这个条件，这个条件可以帮我们快速判断是否有解。那么此时题目就变成给定一个数组 nums，求有多少组不同的 p，使得 sum(p) = target，直接 d. 建立 dp，dp[i] = j 代表数组 nums 中有 j 组子集的和为 i，初始 dp[0] = 1 如 nums = [1,1,1,1,1]，按照如下步骤分析对 nums[0] 分析，则得 dp[1] = 1(因为 dp[0] = 1) 对 nums[1] 分析，则得 dp[1] = 2,dp[2] = 1(因为 dp[0] = 1,dp[1] = 1) 对 nums[2] 分析，则得 dp[1] = 3,dp[2] = 2,d[3] = 1, 依次类推

```
class Solution(object):
    def findTargetSumWays(self, nums, S):
        """
        :type nums: List[int]
        :type S: int
        :rtype: int
        """
        if sum(nums)<S:
            return 0
        if (S+sum(nums))%2==1:
            return 0
        target = (S+sum(nums))/2
        dp = [0]*(target+1)
        dp[0] = 1
        for n in nums:
            i = target
            while(i>=n):
                dp[i] = dp[i] + dp[i-n]
                i = i-1
        return dp[target]
```

图 1.65 Target Sum

1.66 Leetcode 538. Convert BST to Greater Tree

算法如 Fig 1.66.

Given a Binary Search Tree (BST), convert it to a Greater Tree such that every key of the original BST is changed to the original key plus sum of all keys greater than the original key in BST. 思路: 递归。但是如何递归呢。看叶子节点的左右子树的深度都是 0，那么，它的深度是 0，一个数的深度是其左右子树的最大值 +1。树总的最大宽度是其左右子树高度的和中的最大值。求最大距离的过程需要在递归里面写，所以这个步骤比较巧妙，一个递归实现了两个作用。

```
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution(object):
    def convertBST(self, root):
        """
        :type root: TreeNode
        :rtype: TreeNode
        """
        self.tsum = 0
        def dfs(root):
            if not root:
                return
            dfs(root.right)
            root.val += self.tsum
            self.tsum = root.val
            dfs(root.left)
        dfs(root)
        return root
```

图 1.66 Convert BST to Greater Tree

1.67 Leetcode 543. Diameter of Binary Tree

算法如 Fig 1.67.

Given a binary tree, you need to compute the length of the diameter of the tree. The diameter of a binary tree is the length of the longest path between any two nodes in a tree.

This path may or may not pass through the root. 思路: “右 - 根 - 左”顺序遍历 BST。

```
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution(object):
    def diameterOfBinaryTree(self, root):
        """
        :type root: TreeNode
        :rtype: int
        """
        self.ans = 0

        def depth(p):
            if not p:
                return 0
            left = depth(p.left)
            right = depth(p.right)
            self.ans = max(self.ans, left+right)
            return 1+max(left, right)

        depth(root)
        return self.ans
```

图 1.67 Diameter of Binary Tree

1.68 Leetcode 560. Subarray Sum Equals K

算法如 Fig 1.68.

Given an array of integers and an integer k, you need to find the total number of continuous subarrays whose sum equals to k.

思路: 使用一个字典保存数组某个位置之前的数组和，然后遍历数组求和，这样当我们求到一个位置的和的时候，向前找 sum-k 是否在数组中，如果在的话，更新结果为之前的结果 +(sum-k 出现的次数)。同时，当前这个 sum 出现的次数就多了一次。这个字典的意义是什么呢？其意义就是我们在到达 i 位置的时候，前 i 项的和出现的次数的统计。我们想找的是在 i 位置向前的连续区间中，有多少个位置的和是 k。有了这个统计，我们就不用向前一一遍历找 sum - k 在哪些位置出现了，而是直接得出了前面有多少个区间。所以，在每个位置我们都得到了以这个位置为结尾的并且和等于 k 的区间的个数，所以总和就是结果。这个题的解法不难想出来，因为如果要降低时间复杂度，应该能想到增加空间复杂度，那么要么使用数组，要么就是用字典之类的，保留之前的结果。

```python
class Solution(object):
    def subarraySum(self, nums, k):
        """
        :type nums: List[int]
        :type k: int
        :rtype: int
        """
        d = collections.defaultdict(int)
        d[0] = 1
        ans = 0
        sum = 0
        for i in range(len(nums)):
            sum += nums[i]
            if sum - k in d:
                ans += d[sum-k]
            d[sum] += 1
        return ans
```

图 1.68 Subarray Sum Equals K

1.69 Leetcode 581. Shortest Unsorted Continuous Subarray

算法如 Fig 1.69.

Given an integer array, you need to find one continuous subarray that if you only sort this subarray in ascending order, then the whole array will be sorted in ascending order, too. You need to find the shortest such subarray and output its length. 思路: 按照我们自己寻找最小子数组的思路来解决。首先分别设置左、右指针来指示子数组的首和尾，并设置最大值和最小值，（最小值赋大值，最大值赋小值）；分别左右两方向寻找不符合递增规则的边界索引，期间若左指针 l 最后等于数组最大索引，则证明数组全部元素均按升序排序；在找到的界限内寻找内部的最小值和最大值；左指针向左走，右指针向右走，直到小于界限内最小值后的位置，及大于界限内最大值前的位置。注意处理好最后返回值。

1.70 Leetcode 617. Merge Two Binary Trees

算法如 Fig 1.70.

Given two binary trees and imagine that when you put one of them to cover the other, some nodes of the two trees are overlapped while the others are not.

You need to merge them into a new binary tree. The merge rule is that if two nodes overlap, then sum node values up as the new value of the merged node. Otherwise, the NOT null node will be used as the node of new tree. 思路: 如果两个树都有节点的话就把两个相加，左右孩子为两者的左右孩子。否则选不是空的节点当做子节点。

```python
class Solution(object):
    def findUnsortedSubarray(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        left = 0
        right = 0
        for i in range(len(nums)-1):
            if nums[i] > nums[i+1]:
                left = i
                break
        for j in range(len(nums)-1, -1, -1):
            if nums[j] < nums[j-1]:
                right = j
                break
        min_num = min(nums[left:right+1])
        max_num = max(nums[left:right+1])
        new_left = left
        new_right = right
        for i in range(0, left):
            if nums[i]>min_num:
                new_left = i
                break
        for j in range(len(nums)-1, right, -1):
            if nums[j]<max_num:
                new_right = j
                break
        if new_right == new_left:
            return 0
        return max(new_right-new_left+1, 0)
```

图 1.69 Shortest Unsorted Continuous Subarray

```python
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution(object):
    def mergeTrees(self, t1, t2):
        """
        :type t1: TreeNode
        :type t2: TreeNode
        :rtype: TreeNode
        """
        if t1 is None and t2 is None:
            return
        # 只有一个结点为空时
        if t1 is None:
            return t2
        if t2 is None:
            return t1
        # 结点重叠时
        t1.val += t2.val
        # 进行迭代
        t1.right = self.mergeTrees(t1.right, t2.right)
        t1.left = self.mergeTrees(t1.left, t2.left)
        return t1
```

图 1.70 Merge Two Binary Trees

1.71 Leetcode 621. Task Scheduler

算法如 Fig 1.71.

Given a char array representing tasks CPU need to do. It contains capital letters A to Z where different letters represent different tasks. Tasks could be done without original order. Each task could be done in one interval. For each interval, CPU could finish one task or just be idle. However, there is a non-negative cooling interval n that means between two same tasks, there must be at least n intervals that CPU are doing different tasks or just be idle. You need to return the least number of intervals the CPU will take to finish all the given tasks.

思路: 但是真正做这个题目的时候,发现单个 CPU 做调度其实很简单。只要知道出现最多的那个(或几个)task 就行了,其他的任务往缝隙里面塞。比如:如给定:AAABBCD,n=2。那么我们满足个数最多的任务所需的数量,即可以满足任务间隔要求,即:AXXAXXA;(其中,X 表示需要填充任务或者 idle 的间隔)如果有两种或两种以上的任务具有相同的最多的任务数,如:AAAABBBBCCDE,n=3。那么我们将具有相同个数的任务 A 和 B 视为一个任务对,最终满足要求的分配为:ABXXABXXABXXAB,剩余的任务在不违背要求间隔的情况下穿插进间隔位置即可,空缺位置补 idle。由上面的分析我们可以得到最终需要最少的任务时间:(最多任务数-1)*(n + 1) + (相同最多任务的任务个数)。有上面的例子来说就是:(num(A)-1) * (3+1) + (2)。其中,(最多任务数-1)*(n + 1)代表的是 ABXXABXXABXX,(相同最多任务的任务个数)代表的是最后的 AB. 最后,别忘了要对任务数求最大值,毕竟每个任务都是要调度一遍的。

```
class Solution(object):
    def leastInterval(self, tasks, n):
        """
        :type tasks: List[str]
        :type n: int
        :rtype: int
        """
        c = collections.defaultdict(int)
        for t in tasks:
            c[t] += 1
        m = max(c.itervalues())
        l = len([k for k in c if c[k] == m])
        return max(len(tasks), (m - 1) * (n + 1) + l)
```

图 1.71 Task Scheduler

1.72 Leetcode 647. Palindromic Substrings

算法如 Fig 1.72.

Given a string, your task is to count how many palindromic substrings in this string. The substrings with different start indexes or end indexes are counted as different substrings even they consist of same characters.

思路: index 从 0 到 len 进行遍历。对于每个单个的字符，其本身是一个回文。然后对回文长度是奇数的情况进行遍历：使用 left 和 right 双指针，往两边走，判断总长度是 3,5,7……的子串是不是回文（left 指针和 right 指针指向的字符相等）。再对回文是偶数的情况同样的进行遍历。最后求和即可。

```python
class Solution(object):
    def countSubstrings(self, s):
        """
        :type s: str
        :rtype: int
        """
        leftrights = [(x,x) for x in range(len(s))]+[(x, x+1) for x in range(len(s)-1)]
        count = 0
        for left, right in leftrights:
            while left>=0 and right < len(s) and s[left] == s[right]:
                count += 1
                left -= 1
                right += 1
        return count
```

图 1.72 Palindromic Substrings

1.73 Leetcode 739. Daily Temperatures

算法如 Fig 1.73.

Given a list of daily temperatures T, return a list such that, for each day in the input, tells you how many days you would have to wait until a warmer temperature. If there is no future day for which this is possible, put 0 instead. For example, given the list of temperatures T = [73, 74, 75, 71, 69, 72, 76, 73], your output should be [1, 1, 4, 2, 1, 1, 0, 0]. Note: The length of temperatures will be in the range [1, 30000]. Each temperature will be an integer in the range [30, 100].

思路: 如果正序遍历的话需要一个栈，栈的操作是这样的：如果栈是空或者栈顶的元素小于当前元素，那么说明前面的这天的温度小于今天的，所以直接弹出前面这天，并且把他这天的结果设置为和今天的位置差。需要注意的是，无论当天的温度是高是低，它的结果的确定需要根据后面确定，所以要入栈。

```python
class Solution(object):
    def dailyTemperatures(self, T):
        """
        :type T: List[int]
        :rtype: List[int]
        """
        res = [0]*len(T)
        stack = []
        for i in range(len(T)):
            while stack and T[i] > T[stack[-1]]:
                cur = stack.pop()
                res[cur] = i -cur
            stack.append(i)
        return res
```

图 1.73 Daily Temperatures

第 2 章 剑指 offer

2.1 二维数组的查找

算法如图 2.1 在一个二维数组中（每个一维数组的长度相同），每一行都按照从左到右递增的顺序排序，每一列都按照从上到下递增的顺序排序。请完成一个函数，输入这样的一个二维数组和一个整数，判断数组中是否含有该整数。

思路：选择右上角的数字开始查找。如果等于要查找的数字，则查找过程结束。如果该数字大于要查找的数字，则剔除这个数字所在的列。如果该数字小于要查找的数字，则剔除这个数字所在的行。

```
class Solution:
    # array 二维列表
    def Find(self, target, array):
        # write code here
        if array == [[]]:
            return False
        row, col = 0, len(array[0])-1
        while row < len(array) and col >=0:
            if array[row][col] == target:
                return True
            elif array[row][col]>target:
                col -= 1
            else:
                row += 1
        return False
```

图 2.1 二维数组的查找

2.2 替换空格

算法如图 2.2 请实现一个函数，将一个字符串中的每个空格替换成“%20”。例如，当字符串为 We Are Happy. 则经过替换之后的字符串为 We%20Are%20Happy。

思路：直接替换即可，注意边界条件。

```python
class Solution:
    # s 源字符串
    def replaceSpace(self, s):
        # write code here
        if len(s) ==0:
            return s
        len_s = len(s)
        s = list(s)
        for i in range(len_s):
            if s[i] == " ":
                s[i] = "%20"
        return ''.join(s)
```

图 2.2 替换空格

2.3 从尾到头打印链表

算法如图 2.3 输入一个链表，按链表值从尾到头的顺序返回一个 ArrayList。

思路：利用 stack 的后进先出的特性存储链表的节点，然后再打印出来。

2.4 重建二叉树

算法如图 2.4 输入某二叉树的前序遍历和中序遍历的结果，请重建出该二叉树。假设输入的前序遍历和中序遍历的结果中都不含重复的数字。例如输入前序遍历序列 1,2,4,7,3,5,6,8 和中序遍历序列 4,7,2,1,5,3,8,6，则重建二叉树并返回。

思路：前序遍历中的节点在中序遍历中的左边节点序列是它自己的左子树，右边节点序列是它自己的右子树。

2.5 用两个栈实现队列

算法如图 2.5 用两个栈来实现一个队列，完成队列的 Push 和 Pop 操作。队列中的元素为 int 类型。

思路：利用一个栈专门 push 数据，在 pop 数据时当另一个栈为空的时候，把 push 数据栈里的数据全部存在另一个空栈。不为空时，直接 pop。

```python
# -*- coding:utf-8 -*-
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    # 返回从尾部到头部的列表值序列，例如[1,2,3]
    def printListFromTailToHead(self, listNode):
        # write code here
        if not listNode:
            return []
        h = listNode
        stack = []
        while h:
            stack.append(h.val)
            h = h.next
        result = []
        while stack:
            result.append(stack.pop())
        return result
```

图 2.3 从尾到头打印链表

```python
class Solution:
    # 返回构造的TreeNode根节点
    def reConstructBinaryTree(self, pre, tin):
        # write code here
        if tin:
            ind = tin.index(pre.pop(0))
            root = TreeNode(tin[ind])
            root.left = self.reConstructBinaryTree(pre,tin[:ind])
            root.right = self.reConstructBinaryTree(pre,tin[ind+1:])
            return root
```

图 2.4 重建二叉树

```python
# -*- coding:utf-8 -*-
class Solution:
    def __init__(self):
        self.stack1 = []
        self.stack2 = []

    def push(self, node):
        # write code here
        self.stack1.append(node)

    def pop(self):
        # return xx
        if not self.stack2:
            while self.stack1:
                self.stack2.append(self.stack1.pop())
        return self.stack2.pop()
```

图 2.5 用两个栈实现队列

2.6 旋转数组的最小数字

算法如图 2.6 把一个数组最开始的若干个元素搬到数组的末尾，我们称之为数组的旋转。输入一个非减排序的数组的一个旋转，输出旋转数组的最小元素。例如数组 3,4,5,1,2 为 1,2,3,4,5 的一个旋转，该数组的最小值为 1。NOTE：给出的所有元素都大于 0，若数组大小为 0，请返回 0。

思路：注意旋转数组的概念。前半截的数都大于后半截的数。因此，当 medium 的数小于 left 的数的时候，说明前半截的数很少。要找最小值的话，就需要把 right 移到 medium 处。medium 的数大于 left 的数的时候，说明后半截的数比较少。要找最小值的话，需要把 left 移到 medium 处。然后终止条件在最小值附近两个指针差 1。同时还要注意出现相同元素的情况。比如 1，0，1，1，1

```python
class Solution:
    def minNumberInRotateArray(self, rotateArray):
        # write code here
        if len(rotateArray) == 0:
            return 0
        l, r = 0, len(rotateArray) -1
        while l<=r:
            m = (l+r)//2
            if r-l == 1:
                m = r
                return rotateArray[m]
            if rotateArray[l] == rotateArray[m] and rotateArray[m] == rotateArray[r]:
                return min(rotateArray[l:r+1])
            if rotateArray[l]<=rotateArray[m]:
                l = m
            else:
                r = m
```

图 2.6 旋转数组的最小数字

2.7 斐波那契数列

算法如图 2.7 大家都知道斐波那契数列，现在要求输入一个整数 n，请你输出斐波那契数列的第 n 项（从 0 开始，第 0 项为 0）。n<=39

思路：可以用动态规划的思路解题。

2.8 跳台阶

算法如图 2.8 一只青蛙一次可以跳上 1 级台阶，也可以跳上 2 级。求该青蛙跳上一个 n 级的台阶总共有多少种跳法（先后次序不同算不同的结果）。

思路：可以用动态规划的思路解题。

```python
class Solution:
    def Fibonacci(self, n):
        # write code here
        if n == 0:
            return 0
        result = [0]*(n+1)
        result[0] = 0
        result[1] = 1
        for i in range(2, n+1):
            result[i] = result[i-1] + result[i-2]
        return result[-1]
```

图 2.7 斐波那契数列

```python
class Solution:
    def jumpFloor(self, number):
        # write code here
        if number == 0:
            return 0
        if number == 1:
            return 1
        dp = [0]*number
        dp[0] = 1
        dp[1] = 2
        for i in range(2,number):
            dp[i] = dp[i-1] + dp[i-2]
        return dp[-1]
```

图 2.8 跳台阶

2.9 变态跳台阶

算法如图 2.9 一只青蛙一次可以跳上 1 级台阶，也可以跳上 2 级。求该青蛙跳上一个 n 级的台阶总共有多少种跳法（先后次序不同算不同的结果）。

思路：数学归纳法。

```
# -*- coding:utf-8 -*-
class Solution:
    def jumpFloor(self, number):
        # write code here
        if number == 0:
            return 0
        if number == 1:
            return 1
        dp = [0]*number
        dp[0] = 1
        dp[1] = 2
        for i in range(2,number):
            dp[i] = dp[i-1] + dp[i-2]
        return dp[-1]
```

图 2.9 变态跳台阶

2.10 矩形覆盖

算法如图 2.10 我们可以用 2*1 的小矩形横着或者竖着去覆盖更大的矩形。请问用 n 个 2*1 的小矩形无重叠地覆盖一个 2*n 的大矩形，总共有多少种方法?

思路：当我们选择第一个小矩形去覆盖大矩形时，有横着放和竖着放两种选择。横着放剩下的就变成填充 n-2 的大矩形。竖着放就变成填充 n-1 的大矩形

2.11 二进制中 1 的个数

算法如图 2.11 输入一个整数，输出该数二进制表示中 1 的个数。其中负数用补码表示。

```python
class Solution:
    def rectCover(self, number):
        # write code here
        if number == 0:
            return 0
        if number == 1:
            return 1
        f = [0]*number
        f[0] = 1
        f[1] = 2
        for i in range(2, number):
            f[i] = f[i-1] + f[i-2]
        return f[number-1]
```

图 2.10 矩形覆盖

思路：把一个整数减去 1，再和原来整数做与运算，会把该整数最右边的 1 变成 0，那么一个整数的二进制表示中有多少个 1，就可以进行多少次这样的操作。注意把负号去掉，以免掉入死循环。

```python
class Solution:
    def NumberOf1(self, n):
        # write code here
        count = 0
        if n < 0:
            n = n & 0xffffffff
        while n:
            count += 1
            n = (n - 1) & n
        return count
```

图 2.11 二进制中 1 的个数

2.12 数值的整数次方

算法如图 2.12 给定一个 double 类型的浮点数 base 和 int 类型的整数 exponent。求 base 的 exponent 次方。

思路：当指数为负数的时候，要分 base 是 0 和非 0 的时候。0 的 0 次方可以输出 0 也可以输出 1。先求指数／ 2 时候的 result，然后再迭代。我们用右移运算符代替了除以 2。

用位与运算 &0x1 代替了求余运算。

```python
class Solution:
    def Power(self, base, exponent):
        # write code here

        if base == 0 and exponent < 0:
            return 0
        flag = False
        if exponent < 0:
            exponent = -exponent
            flag = True
        result = self.UnsignedPower(base, exponent)
        if flag:
            result = 1.0/result
        return result

    def UnsignedPower(self, base, absExponent):
        if absExponent == 0:
            return 1
        if absExponent == 1:
            return base
        result = self.UnsignedPower(base, absExponent>>1)
        result *= result
        if absExponent&0x1 == 1:
            result *= base
        return result
```

图 2.12 数值的整数次方

2.13 调整数组顺序使奇数位于偶数前面

算法如图 2.13 输入一个整数数组，实现一个函数来调整该数组中数字的顺序，使得所有的奇数位于数组的前半部分，所有的偶数位于数组的后半部分，并保证奇数和奇数，偶数和偶数之间的相对位置不变。

思路：用两个指针指代头尾，尾指针从尾部出发，每遇到一个偶数（且自己是奇数的时候）就和偶数替换位置。当尾指针遇到头指针以后算一个循环，头指针再往前走一步，加 1。每一个内部替换都保证头尾指针的顺序。

```python
class Solution:
    def reOrderArray(self, array):
        # write code here
        for i in range(0,len(array)):
            for j in range(len(array)-1,i,-1):
                if array[j-1]%2 ==0 and array[j]%2==1:
                    array[j], array[j-1] = array[j-1],array[j]
        return array
```

图 2.13 调整数组顺序使奇数位于偶数前面

2.14 链表中倒数第 k 个结点

算法如图 2.14 输入一个链表，输出该链表中倒数第 k 个结点。

思路：要考虑链表为空，k 大于链表长度，k 等于链表长度，k 小于链表长度各种情况。其余的可以按照 leetcode 中的思路来。

```
class Solution:
    def FindKthToTail(self, head, k):
        # write code here
        if not head:
            return head
        fast = slow = head
        for i in range(k):
            if not fast:
                return None
            fast = fast.next
        if not fast:
            return head
        while fast.next:
            fast= fast.next
            slow = slow.next

        return slow.next
```

图 2.14 链表中倒数第 k 个结点

2.15 反转链表

算法如图 2.15 输入一个链表，反转链表后，输出新链表的表头。

思路：注意分输入的链表是空链表，一个节点的链表，多个节点的链表的情况，其余就是链表的操作。

2.16 合并两个排序的链表

算法如图 2.16 输入两个单调递增的链表，输出两个链表合成后的链表，当然我们需要合成后的链表满足单调不减规则。

思路：链表的操作。

```python
class Solution:
    # 返回ListNode
    def ReverseList(self, pHead):
        # write code here
        if not pHead or not pHead.next:
            return pHead

        last = None

        while pHead:
            tmp = pHead.next
            pHead.next = last
            last = pHead
            pHead = tmp
        return last
```

图 2.15 反转链表

```python
class Solution:
    # 返回合并后列表
    def Merge(self, pHead1, pHead2):
        # write code here
        r = h = ListNode(0)
        while pHead1 and pHead2:
            if pHead1.val <= pHead2.val:
                r.next = ListNode(pHead1.val)
                pHead1 = pHead1.next
            else:
                r.next = ListNode(pHead2.val)
                pHead2 = pHead2.next
            r = r.next
        if pHead1:
            r.next = pHead1
        if pHead2:
            r.next = pHead2
        return h.next
```

图 2.16 合并两个排序的链表

2.17 树的子结构

算法如图 2.17 输入两棵二叉树 A，B，判断 B 是不是 A 的子结构。（ps：我们约定空树不是任意一个树的子结构）

思路：利用递归的思路去做。

2.18 二叉树的镜像

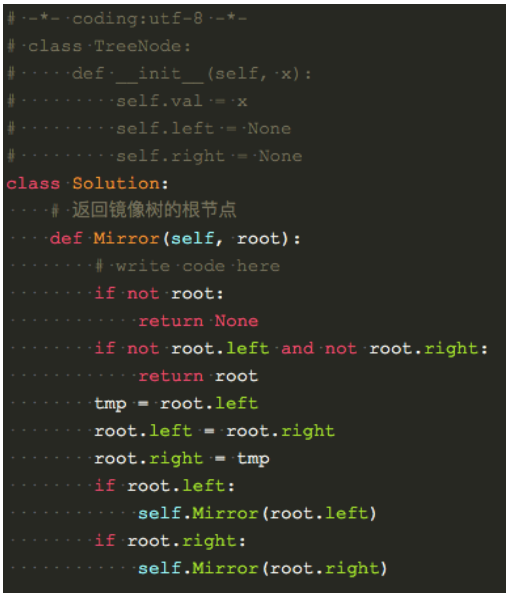
算法如图 2.18 操作给定的二叉树，将其变换为源二叉树的镜像。

思路：利用递归的思路去做，注意输入的树有空树，只有左节点的树，只有右节点的树，和正常树。

```
# -*- coding:utf-8 -*-
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution:
    def HasSubtree(self, pRoot1, pRoot2):
        # write code here
        result = False
        if pRoot1 and pRoot2:
            if pRoot1.val == pRoot2.val:
                result = self.isSubtree(pRoot1, pRoot2)
            if not result:
                result = self.HasSubtree(pRoot1.left, pRoot2)
            if not result:
                result = self.HasSubtree(pRoot1.right, pRoot2)
        return result

    def isSubtree(self, pRoot1, pRoot2):
        if not pRoot2:
            return True
        if not pRoot1:
            return False
        if pRoot1.val != pRoot2.val:
            return False
        return self.isSubtree(pRoot1.left, pRoot2.left)and self.isSubtree(pRoot1.right,
                                                                          pRoot2.right)
```

图 2.17 树的子结构



```
# -*- coding:utf-8 -*-
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution:
    # 返回镜像树的根节点
    def Mirror(self, root):
        # write code here
        if not root:
            return None
        if not root.left and not root.right:
            return root
        tmp = root.left
        root.left = root.right
        root.right = tmp
        if root.left:
            self.Mirror(root.left)
        if root.right:
            self.Mirror(root.right)
```

图 2.18 二叉树的镜像

2.19 顺时针打印矩阵

算法如图 2.19 输入一个矩阵，按照从外向里以顺时针的顺序依次打印出每一个数字，例如，如果输入如下 4 X 4 矩阵：1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 则依次打印出数字 1,2,3,4,8,12,16,15,14,13,9,5,6,7,11,10.

思路：每一圈最开始的点的横坐标和纵坐标都是一样的。

```python
class Solution:
    # matrix类型为二维列表，需要返回列表
    def printMatrix(self, matrix):
        # write code here
        if not matrix:
            return None
        rows = len(matrix)
        cols = len(matrix[0])
        start = 0
        result = []
        while rows > 2*start and cols > 2*start:
            endx = rows-1-start
            endy = cols-1-start
            for i in range(start,endy+1):
                result.append(matrix[start][i])
            if start < endx:
                for i in range(start+1,endx+1):
                    result.append(matrix[i][endy])
            if start < endx and start < endy:
                for i in range(endy-1,start-1,-1):
                    result.append(matrix[endx][i])
            if start < endx-1 and start < endy:
                for i in range(endx-1,start,-1):
                    result.append(matrix[i][start])

            start += 1
        return result
```

图 2.19 顺时针打印矩阵

2.20 包含 min 函数的栈

算法如图 2.20 定义栈的数据结构，请在该类型中实现一个能够得到栈中所含最小元素的 min 函数（时间复杂度应为 O（1））。

思路：利用一个辅助栈存储每一次压入某个元素时的最小值。

2.21 栈的压入、弹出序列

算法如图 2.21 输入两个整数序列，第一个序列表示栈的压入顺序，请判断第二个序列是否可能为该栈的弹出顺序。假设压入栈的所有数字均不相等。例如序列 1,2,3,4,5 是某栈

```python
class Solution:
    def __init__(self):
        self.stack = []
        self.min_stack = []

    def push(self, node):
        # write code here
        if self.min_stack ==[] or node < self.min_stack[-1]:
            self.min_stack.append(node)
        else:
            self.min_stack.append(self.min_stack[-1])
        self.stack.append(node)

    def pop(self):
        # write code here
        self.stack.pop()
        self.min_stack.pop()

    def top(self):
        # write code here
        return self.stack[-1]

    def min(self):
        # write code here
        return self.min_stack[-1]
```

图 2.20 包含 min 函数的栈

的压入顺序，序列 4,5,3,2,1 是该压栈序列对应的一个弹出序列，但 4,3,5,1,2 就不可能是该压栈序列的弹出序列。（注意：这两个序列的长度是相等的）

思路：利用辅助栈按压入序列压入元素，如果当前元素等于弹出序列的元素之首，那弹出当前元素。到最终的压入序列尾部，辅助栈中存储的序列弹出顺序和弹出栈序列是否一致。

```python
class Solution:
    def IsPopOrder(self, pushV, popV):
        # write code here
        stack = []
        j = 0
        for i in range(len(pushV)):
            stack.append(pushV[i])
            if pushV[i] == popV[j]:
                stack.pop()
                j+=1
        while stack:
            if stack.pop()!= popV[j]:
                return False
            j +=1
        return True
```

图 2.21 栈的压入、弹出序列

2.22 从上往下打印二叉树

算法如图 2.22

从上往下打印出二叉树的每个节点，同层节点从左至右打印。

思路：广度优先搜索二叉树。

```
# -*- coding:utf-8 -*-
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution:
    # 返回从上到下每个节点值列表，例：[1,2,3]
    def PrintFromTopToBottom(self, root):
        # write code here
        if not root:
            return []
        level, result = [root], []
        while level:
            for node in level:
                result.append(node.val)
            tmp = []
            for node in level:
                if node.left:
                    tmp.append(node.left)
                if node.right:
                    tmp.append(node.right)
            level = tmp
        return result
```

图 2.22 从上往下打印二叉树

2.23 二叉搜索树的后序遍历序列

算法如图 2.23

输入一个整数数组，判断该数组是不是某二叉搜索树的后序遍历的结果。如果是则输出 Yes, 否则输出 No。假设输入的数组的任意两个数字都互不相同。

思路：二叉搜索树的左子树永远小于父节点，父节点永远小于右子树上的节点。后序遍历序列代表最后边的节点就是父节点。前面的序列节点能分成全部小于它和全部大于它的两个序列。

2.24 二叉树中和为某一值的路径

算法如图 2.24

输入一颗二叉树的跟节点和一个整数，打印出二叉树中结点值的和为输入整数的所有路径。路径定义为从树的根结点开始往下一直到叶结点所经过的结点形成一条路径。(注意: 在返回值的 list 中，数组长度大的数组靠前)

```python
class Solution:
    def VerifySquenceOfBST(self, sequence):
        # write code here
        if not sequence:
            return False
        root = sequence[-1]
        i = 0
        while sequence[i]< root:
            i += 1

        for j in range(i, len(sequence)-1):
            if sequence[j]<root:
                return False
        left_seq = sequence[:i]
        right_seq = sequence[i:len(sequence)-1]
        left = True
        if len(left_seq) >0:
            left = self.VerifySquenceOfBST(left_seq)
        right = True
        if len(right_seq) >0:
            right = self.VerifySquenceOfBST(right_seq)
        return left and right
```

图 2.23 二叉搜索树的后序遍历序列

思路：从父节点到子节点的节点序列称为树的路径，因此可以采用中序遍历。注意返回路径集合时候的返回值。tr

```python
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution:
    # 返回二维列表，内部每个列表表示找到的路径
    def FindPath(self, root, expectNumber):
        # write code here
        if not root:
            return []
        if (not root.left and not root.right) and expectNumber == root.val:
            return [[root.val]]
        res = []
        left = self.FindPath(root.left, expectNumber-root.val)
        right = self.FindPath(root.right, expectNumber-root.val)
        for i in left+right:
            res.append([root.val]+i)
        return res
```

图 2.24 二叉树中和为某一值的路径

2.25 字符串的排列

算法如图 2.25

输入一个字符串, 按字典序打印出该字符串中字符的所有排列。例如输入字符串 abc, 则打印出由字符 a,b,c 所能排列出来的所有字符串 abc,acb,bac,bca,cab 和 cba。

思路：字符串的全排列，与 nsum 一样的递归思路，然后注意输出的格式。

2.26 数组中出现次数超过一半的数字

算法如图 2.26

```python
class Solution:
    def Permutation(self, ss):
        # write code here
        if len(ss) == 0:
            return []
        results = []
        def dfs(ss, result):
            if len(ss)==0:
                return results.append(result)
            for i in range(len(ss)):
                dfs(ss[:i] + ss[i+1:], result+ss[i])
        dfs(ss,'')
        results = list(set(results))
        return sorted(results)
```

图 2.25 字符串的排列

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。例如输入一个长度为 9 的数组 1,2,3,2,2,2,5,4,2。由于数字 2 在数组中出现了 5 次，超过数组长度的一半，因此输出 2。如果不存在则输出 0。

思路：可利用快排的思路，也可用多数人投票法则类似 sort color 的解法。

```python
class Solution:
    def MoreThanHalfNum_Solution(self, numbers):
        # write code here
        n = len(numbers)
        if n == 0:
            return 0
        max_num = numbers[0]
        count = 1
        for i in range(1, n):
            if count == 0:
                max_num = numbers[i]
                count = 1
            elif numbers[i] != max_num:
                count -=1
            else:
                count += 1
        count = 0;
        for i in range(n):
            if (numbers[i] == max_num):
                count += 1

        return max_num if count*2>n else 0
```

图 2.26 数组中出现次数超过一半的数字

2.27 最小的 K 个数

算法如图 2.27

输入 n 个整数，找出其中最小的 K 个数。例如输入 4,5,1,6,2,7,3,8 这 8 个数字，则最小的 4 个数字是 1,2,3,4,。

思路：可利用快排的思路，也可用一个 k 长度的数组存储最小的 k 个数 (最大堆)。

```python
# -*- coding:utf-8 -*-
class Solution:
    def GetLeastNumbers_Solution(self, tinput, k):
        # write code here
        if k == 0 or k >len(tinput):
            return []
        self.qsort(tinput,0, len(tinput))
        return tinput[:k]
    def qsort(self, tinput, l, r):
        if l< r:
            pos = self.Partition(tinput, l)
            self.qsort(tinput,l, pos)
            self.qsort(tinput, pos+1, r)

    def Partition(self, tinput, l):
        pivot = tinput[l]
        lo = l+1
        for i in range(l+1, len(tinput)):
            if tinput[i]<pivot:
                tinput[i],tinput[lo] = tinput[lo],tinput[i]
                lo += 1
        tinput[l], tinput[lo-1] = tinput[lo-1], tinput[l]
        return lo-1
```

图 2.27 最小的 K 个数

2.28 连续子数组的最大和

算法如图 2.28

HZ 偶尔会拿些专业问题来忽悠那些非计算机专业的同学。今天测试组开完会后, 他又发话了: 在古老的一维模式识别中, 常常需要计算连续子向量的最大和, 当向量全为正数的时候, 问题很好解决。但是, 如果向量中包含负数, 是否应该包含某个负数, 并期望旁边的正数会弥补它呢? 例如:6,-3,-2,7,-15,1,2,2, 连续子向量的最大和为 8(从第 0 个开始, 到第 3 个为止)。给一个数组，返回它的最大连续子序列的和，你会不会被他忽悠住? (子向量的长度至少是 1)。

思路：动态规划的思路，就是前一个元素的最大和大于 0，就可以加上目前的元素做目前元素的最大和。否则就从目前元素算起。

```python
class Solution:
    def FindGreatestSumOfSubArray(self, array):
        # write code here

        dp = [0]*(len(array))
        dp[0] = array[0]
        for i in range(1, len(array)):
            if dp[i-1]>0:
                dp[i] = dp[i-1]+array[i]
            else:
                dp[i] = array[i]
        return max(dp)
```

图 2.28 连续子数组的最大和

2.29 整数中 1 出现的次数（从 1 到 n 整数中 1 出现的次数）

算法如图 2.29

求出 1 13 的整数中 1 出现的次数, 并算出 100 1300 的整数中 1 出现的次数? 为此他特别数了一下 1 13 中包含 1 的数字有 1、10、11、12、13 因此共出现 6 次, 但是对于后面问题他就没辙了。ACMer 希望你们帮帮他, 并把问题更加普遍化, 可以很快的求出任意非负整数区间中 1 出现的次数（从 1 到 n 中 1 出现的次数）。

思路：没看懂，再说吧。

```python
# -*- coding:utf-8 -*-
class Solution:
    def NumberOf1Between1AndN_Solution(self, n):
        # write code here
        digits = str(n)
        N = len(digits)
        count = 0
        for i in range(N):
            if digits[0] == '0':
                pass
            elif i < N - 1:
                count += int(digits[0]) * 10 ** (N - 2 - i) * (N - 1 - i)
                if digits[0] == '1':
                    count += int(digits[1:]) + 1
                else:
                    count += 10 ** (N - 1 - i)
            else:
                count += 1
            digits = digits[1:]
        return count
```

图 2.29 整数中 1 出现的次数

2.30 把数组排成最小的数

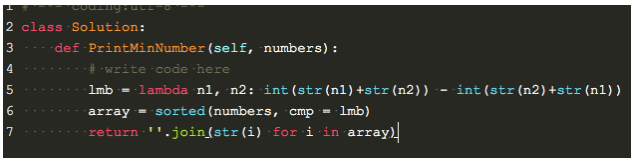
算法如图 2.30

输入一个正整数数组，把数组里所有数字拼接起来排成一个数，打印能拼接出的所有数字中最小的一个。例如输入数组 3，32，321，则打印出这三个数字能排成的最小数字为 321323。

思路：把数字转化为字符串，然后两两连接之后的数字按从小到大排序。

2.31 丑数

算法如图 2.31



```
class Solution:
    def PrintMinNumber(self, numbers):
        # write code here
        lmb = lambda n1, n2: int(str(n1)+str(n2)) - int(str(n2)+str(n1))
        array = sorted(numbers, cmp = lmb)
        return ''.join(str(i) for i in array)
```

图 2.30 把数组排成最小的数

把只包含质因子 2、3 和 5 的数称作丑数（Ugly Number）。例如 6、8 都是丑数，但 14 不是，因为它包含质因子 7。习惯上我们把 1 当做是第一个丑数。求按从小到大的顺序的第 N 个丑数。

思路：每一个丑数是前面的丑数乘以 2，3，5 得到的。已有的丑数是按顺序存放在数组中的。对于乘以 2 而言，肯定存在某个丑数 T_2，排在它之前的每个丑数乘以 2 得到的结果都小于已有最大的丑数，在它之后的每个丑数乘以 2 得到的结果都太大。我们只需记下这个丑数的位置，同时每次生成新的丑数的时候去更新这个 T_2 即可。

```
# -*- coding:utf-8 -*-
class Solution:
    def GetUglyNumber_Solution(self, index):
        # write code here

        def min3(x,y,z):
            return x if x < y and x < z else y if y < z else z

        if index == 0:
            return 0
        if index == 1:
            return 1
        cache = [1]
        seed2 = seed3 = seed5 = 0
        i = 1
        while i < index:
            while cache[seed2] * 2 <= cache[-1]:
                seed2 += 1
            while cache[seed3] * 3 <= cache[-1]:
                seed3 += 1
            while cache[seed5] * 5 <= cache[-1]:
                seed5 += 1
            cache.append(min3(cache[seed2]*2,cache[seed3]*3,cache[seed5]*5))
            i += 1
        return cache[index-1]
```

图 2.31 丑数

2.32 第一个只出现一次的字符

算法如图 2.32

在一个字符串 (0<= 字符串长度 <=10000，全部由字母组成) 中找到第一个只出现一次的字符, 并返回它的位置, 如果没有则返回 -1（需要区分大小写）.

思路：和 two sum 类似的思路。

```python
# -*- coding:utf-8 -*-
class Solution:
    def FirstNotRepeatingChar(self, s):
        # write code here
        used_char = {}
        for item in s:
            if item not in used_char.keys():
                used_char[item] = 0
            used_char[item] += 1
        for i in range(len(s)):
            if used_char[s[i]] == 1:
                return i
        return -1
```

图 2.32 第一个只出现一次的字符

2.33 数字在排序数组中出现的次数

算法如图 2.33

统计一个数字在排序数组中出现的次数。

思路：从前往后搜索找 start 的 index，从后往前搜索找 end 的 index。

```python
class Solution:
    def GetNumberOfK(self, data, k):
        # write code here
        start, end = -1, -1
        for i in range(0, len(data)):
            if data[i] == k:
                start = i
                break
        if start == -1:
            return 0
        for i in range(len(data)-1, -1, -1):
            if data[i] == k:
                end = i
                break
        count = end - start + 1
        return count
```

图 2.33 数字在排序数组中出现的次数

2.34 二叉树的深度

算法如图 2.34

输入一棵二叉树，求该树的深度。从根结点到叶结点依次经过的结点（含根、叶结点）形成树的一条路径，最长路径的长度为树的深度

思路：利用层次遍历的思路 tr。

```python
# -*- coding:utf-8 -*-
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution:
    def TreeDepth(self, pRoot):
        # write code here
        if not pRoot:
            return 0
        level = [pRoot]
        depth = 1
        while level:
            tmp = []
            for node in level:
                if node.left:
                    tmp.append(node.left)
                if node.right:
                    tmp.append(node.right)
            level = tmp
            if len(level) >0:
                depth += 1
        return depth
```

图 2.34 二叉树的深度

2.35 平衡二叉树

算法如图 2.35

输入一棵二叉树，判断该二叉树是否是平衡二叉树。

思路：平衡二叉树是指左右子树高度差不超过 1，并且左右子树都是平衡二叉树。不断的判断每个父节点的左右节点树的高度是不超过 1。

2.36 和为 S 的连续正数序列

算法如图 2.36

```
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution:
    def IsBalanced_Solution(self, pRoot):
        # write code here
        if pRoot == None:
            return True
        if abs(self.TreeDepth(pRoot.left)-self.TreeDepth(pRoot.right)) > 1:
            return False
        return self.IsBalanced_Solution(pRoot.left) and self.IsBalanced_Solution(pRoot.rig

    def TreeDepth(self, pRoot):
        # write code here
        if pRoot == None:
            return 0
        nLeft = self.TreeDepth(pRoot.left)
        nRight = self.TreeDepth(pRoot.right)
        return max(nLeft+1,nRight+1)#(nLeft+1 if nLeft > nRight else nRight +1)
```

图 2.35 平衡二叉树

小明很喜欢数学, 有一天他在做数学作业时, 要求计算出 9 16 的和, 他马上就写出了正确答案是 100。但是他并不满足于此, 他在想究竟有多少种连续的正数序列的和为 100(至少包括两个数)。没多久, 他就得到另一组连续正数和为 100 的序列:18,19,20,21,22。现在把问题交给你, 你能不能也很快的找出所有和为 S 的连续正数序列? Good Luck! 输出所有和为 S 的连续正数序列。序列内按照从小至大的顺序，序列间按照开始数字从小到大的顺序

思路: 取连续正数序列的前两位 small = 1，big =2，curSum = small + big 当 curSum <sum 时，big = big + 1 curSum += big. 当 curSum >sum 时，减去 small,small 自行加 1. 当 curSum==sum 时，继续 big+1 操作，然后加 big 找下一个连续正数序列。

```
class Solution:
    def FindContinuousSequence(self, tsum):
        # write code here
        if tsum<3:
            return []
        small = 1
        big = 2
        middle = (tsum + 1)>>1
        result = []
        curSum = small + big
        while small < middle:
            if curSum == tsum:
                result.append([i for i in range(small, big+1)])
                big += 1
                curSum += big
            elif curSum > tsum:
                curSum -= small
                small += 1
            else:
                big += 1
                curSum += big
        return result
```

图 2.36 和为 S 的连续正数序列

2.37 和为 S 的两个数字

算法如图 2.37

输入一个递增排序的数组和一个数字 S，在数组中查找两个数，使得他们的和正好是 S，如果有多对数字的和等于 S，输出两个数的乘积最小的。对应每个测试案例，输出两个数，小的先输出。

思路：从两边向中间靠拢，相等的刚好是乘积最小的。如果 l+r 对应的两个数之和大于 sum，说明 r 得减 1. 否则就是 l 需要加 1。直到两个 sum 刚好等于 target。注意和 twosum 的联系与区别。

```
-*-coding:utf-8-*-
class Solution:
    def FindNumbersWithSum(self, array, tsum):
        # write code here
        l, r = 0, len(array)-1

        result = []
        while l< r:
            curSum = array[l] + array[r]
            if curSum == tsum:
                result = [array[l], array[r]]
                break
            elif curSum > tsum:
                r-=1
            else:
                l+=1
        return result
```

图 2.37 和为 S 的两个数字

2.38 翻转单词顺序列

算法如图 2.38

牛客最近来了一个新员工 Fish，每天早晨总是会拿着一本英文杂志，写些句子在本子上。同事 Cat 对 Fish 写的内容颇感兴趣，有一天他向 Fish 借来翻看，但却读不懂它的意思。例如，“student. a am I”。后来才意识到，这家伙原来把句子单词的顺序翻转了，正确的句子应该是“I am a student.”。Cat 对一一的翻转这些单词顺序可不在行，你能帮助他么?

思路：先翻转每一个单词，再翻转全部句子的顺序。

```python
class Solution:
    def ReverseSentence(self, s):
        # write code here
        start, end = 0, 1
        s = list(s)
        while end < len(s)-1:
            if s[end] == " ":
                s[start:end] = s[start:end][::-1]
                start = end + 1
                end = start + 1
            else:
                end += 1
        if start < len(s)-1:
            s[start:] = s[start:][::-1]
        return ''.join(s[::-1])
```

图 2.38 翻转单词顺序列

2.39 扑克牌顺子

算法如图 2.39

现在, 要求你使用这幅牌模拟上面的过程, 然后告诉我们 LL 的运气如何， 如果牌能组成顺子就输出 true， 否则就输出 false。为了方便起见, 你可以认为大小王是 0。

思路: 明确规则就好, 然后用程序语言表达规则。五张牌中有对子就不可能是顺子。五张牌中的 0 多于顺子间的间隔的话也能是顺子。

```python
class Solution:
    def IsContinuous(self, numbers):
        # write code here
        if len(numbers) == 0 or len(numbers) ==1:
            return False
        numbers.sort()
        gap_count = 0
        zero_count = 0
        for i in range(len(numbers)):
            if numbers[i] != 0:
                break
            zero_count += 1
        for i in range(zero_count+1, len(numbers)):
            if numbers[i] == numbers[i-1]:
                return False
            if numbers[i]-numbers[i-1]>1:
                gap_count += numbers[i] - numbers[i-1] - 1
        return True if zero_count >= gap_count else False
```

图 2.39 扑克牌顺子

2.40 求 1+2+3+...+n

算法如图 2.40

求 1+2+3+...+n，要求不能使用乘除法、for、while、if、else、switch、case 等关键字及条件判断语句。思路：直接用递归好了。

```
class Solution:

    def Sum_Solution(self, n):
        # write code here
        if n == 1:
            return 1
        result = self.Sum_Solution(n-1)
        result = result + n
        return result
```

图 2.40 求 1+2+3+...+n

2.41 不用加减乘除做加法

算法如图 2.41

写一个函数，求两个整数之和，要求在函数体内不得使用 +、-、*、/四则运算符号。思路：不考虑进位，对每一位相加（相当于二进制的异或）。计算进位值，得到 10. 如果这一步的进位值为 0，那么第一步得到的值就是最终结果。

```
2 class Solution:
3     def Add(self, num1, num2):
4         # write code here
5         while num2 != 0:
6             temp = num1 ^ num2
7             num2 = (num1 & num2) << 1
8             num1 = temp & 0xFFFFFFFF
9         return num1 if num1 >> 31 == 0 else num1 - 4294967296
```

图 2.41 不用加减乘除做加法

2.42 把字符串转换成整数

算法如图 2.42

将一个字符串转换成一个整数 (实现 Integer.valueOf(string) 的功能，但是 string 不符合数字要求时返回 0)，要求不能使用字符串转换整数的库函数。数值为 0 或者字符串不是一个合法的数值则返回 0。

输入描述:

输入一个字符串, 包括数字字母符号, 可以为空

输出描述:

如果是合法的数值表达则返回该数字，否则返回 0

思路：小心所有边界就好了包括符号，最前面的 0 什么的。

```python
class Solution:
    def StrToInt(self, s):
        # write code here
        if len(s) == 0:
            return 0
        valid_str = ['0', '1', '2','3', '4', '5','6', '7', '8', '9']
        s = list(s)
        start = 0
        abs_sign = 1
        if s[0] in ['+', '-']:
            abs_sign = -1 if s[0] == '-' else 1
            start = 1
        else:
            start = 0
        result = ''
        for i in range(start, len(s)):
            if s[i] in valid_str:
                result += s[i]
            else:
                return 0
        ind = 0
        for item in result:
            if result != '0':
                break
        if ind > len(result)-1:
            return 0
        else:
            return abs_sign*int(result[ind:])
```

图 2.42 把字符串转换成整数

2.43 数组中重复的数字

算法如图 2.43

在一个长度为 n 的数组里的所有数字都在 0 到 n-1 的范围内。数组中某些数字是重复的，但不知道有几个数字是重复的。也不知道每个数字重复几次。请找出数组中任意一个重复的数字。例如，如果输入长度为 7 的数组 2,3,1,0,2,5,3，那么对应的输出是第一个重复的数字 2。

思路：就太简单，没什么思路。

2.44 构建乘积数组

算法如图 2.46

```python
class Solution:
    # 这里要特别注意~找到任意重复的一个值并赋值到duplication[0]
    # 函数返回True/False
    def duplicate(self, numbers, duplication):
        # write code here
        if len(numbers)<=1:
            return False
        for i in range(1, len(numbers)):
            if numbers[i] in numbers[:i]:
                duplication[0] = numbers[i]
                return True
        return False
```

图 2.43 数组中重复的数字

给定一个数组 A[0,1,...,n-1], 请构建一个数组 B[0,1,...,n-1], 其中 B 中的元素 B[i]=A[0]*A[1]*...*A[i-1]*A[i+1]*...*A[n-1]。不能使用除法。

思路：巧妙地没有乘以 A[i]。

```python
class Solution:
    def multiply(self, A):
        # write code here
        if A ==[]:
            return []
        n = len(A)
        B = [None]*n
        B[0] = 1
        for i in range(1, n):
            B[i] = B[i-1]*A[i-1]
        tmp = 1
        for i in range(n-2, -1, -1):
            tmp *= A[i+1]
            B[i] *= tmp
        return B
```

图 2.44 构建乘积数组

2.45 正则表达式匹配

算法如图 2.46

请实现一个函数用来匹配包括'.' 和'*' 的正则表达式。模式中的字符'.' 表示任意一个字符，而'*' 表示它前面的字符可以出现任意次（包含 0 次）。在本题中，匹配是指字符串的

所有字符匹配整个模式。例如，字符串”aaa” 与模式”a.a” 和”ab*ac*a” 匹配，但是与”aa.a” 和”ab*a” 均不匹配

思路：emmmmmex。

```python
class Solution:
    # s, pattern都是字符串
    def match(self, s, pattern):
        # write code here
        s = list(s)
        p = list(pattern)
        m = len(s)
        n = len(p)
        dp = [[True] + [False] * m]
        for i in xrange(n):
            dp.append([False]*(m+1))

        for i in xrange(1, n + 1):
            x = p[i-1]
            if x == '*' and i > 1:
                dp[i][0] = dp[i-2][0]
            for j in xrange(1, m+1):
                if x == '*':
                    dp[i][j] = dp[i-2][j] or dp[i-1][j] or (dp[i-1][j-1] and p[i-2] == s[j-1]) or (dp[i][j-1] and p[i-2]=='.')
                elif x == '.' or x == s[j-1]:
                    dp[i][j] = dp[i-1][j-1]

        return dp[n][m]
```

图 2.45 正则表达式匹配

2.46 表示数值的字符串

算法如图 2.46

请实现一个函数用来判断字符串是否表示数值(包括整数和小数)。例如,字符串”+100”,”5e2”,”-123”,”3.1416” 和”-1E-16” 都表示数值。但是”12e”,”1a3.14”,”1.2.3”,”+-5” 和”12e+4.3” 都不是。

思路：主要是 e 和小数点的分界线，然后符号判断。

```python
# -*- coding:utf-8 -*-
class Solution:
    # s字符串
    def isNumeric(self, s):
        # write code here
        isAllowDot = True
        isAllowE = True
        for i in range(len(s)):
            if s[i] in "+-" and (i==0 or s[i-1] in "eE") and i < len(s)-1:
                continue
            elif isAllowDot and s[i] == ".":
                isAllowDot = False
                if i >= len(s)-1 or s[i+1] not in "0123456789":
                    return False
            elif isAllowE and s[i] in "Ee":
                isAllowDot = False
                isAllowE = False
                if i >= len(s)-1 or s[i+1] not in "0123456789+-":
                    return False
            elif s[i] not in "0123456789":
                return False
        return True
```

图 2.46 表示数值的字符串

2.47 字符流中第一个不重复的字符

算法如图 2.47

请实现一个函数用来找出字符流中第一个只出现一次的字符。例如，当从字符流中只读出前两个字符”go” 时，第一个只出现一次的字符是”g”。当从该字符流中读出前六个字符“google” 时，第一个只出现一次的字符是”l”。

思路：就是用 python 字典实现 hash 类似的思路 n。

```python
class Solution:
    # 返回对应char
    def __init__(self):
        self.s = ''
        self.s_dict = {}
    def FirstAppearingOnce(self):
        # write code here
        for char in self.s:
            if self.s_dict[char] == 1:
                return char
        return "#"

    def Insert(self, char):
        # write code here
        self.s = self.s + char
        if char not in self.s_dict.keys():
            self.s_dict[char] = 1
        else:
            self.s_dict[char] += 1
```

图 2.47 字符流中第一个不重复的字符

2.48 链表中环的入口结点

算法如图 2.48

给一个链表，若其中包含环，请找出该链表的环的入口结点，否则，输出 null。

思路：两者在环中相遇的节点离入口节点的距离等于链表首节点到入口节点的距离 r。

```python
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None
class Solution:
    def EntryNodeOfLoop(self, pHead):
        # write code here
        if pHead == None or pHead.next == None:
            return None
        slow = fast = pHead
        while fast and fast.next:
            slow = slow.next
            fast = fast.next.next
            if slow == fast:
                slow = pHead
                while slow != fast:
                    slow = slow.next
                    fast = fast.next
                return slow
        return None
```

图 2.48 链表中环的入口结点

2.49 删除链表中重复的结点

算法如图 2.49

在一个排序的链表中，存在重复的结点，请删除该链表中重复的结点，重复的结点不保留，返回链表头指针。例如，链表 1->2->3->3->4->4->5 处理后为 1->2->5

思路：考察链表的操作，很 6666。

```python
class Solution:
    def deleteDuplication(self, pHead):
        # write code here
        first = ListNode(-1)
        first.next = pHead
        last = first
        while pHead and pHead.next:
            if pHead.val == pHead.next.val:
                val = pHead.val
                while pHead and val==pHead.val:
                    pHead = pHead.next
                last.next = pHead
            else:
                last = pHead
                pHead = pHead.next
        return first.next
```

图 2.49 删除链表中重复的结点

2.50 二叉树的下一个结点

算法如图 2.50

给定一个二叉树和其中的一个结点，请找出中序遍历顺序的下一个结点并且返回。注意，树中的结点不仅包含左右子结点，同时包含指向父结点的指针。

思路：树的遍历。

```
class Solution:
    def GetNext(self, pNode):
        # write code here
        if pNode is None:
            return None

        while pNode.right is not None:
            pNode = pNode.right
            while pNode.left is not None:
                pNode = pNode.left
            return pNode

        while pNode.next:
            tmp=pNode.next
            if tmp.left==pNode:
                return tmp
            pNode=tmp
        return None
```

图 2.50 二叉树的下一个结点

2.51 对称的二叉树

算法如图 2.51

请实现一个函数，用来判断一颗二叉树是不是对称的。注意，如果一个二叉树同此二叉树的镜像是同样的，定义其为对称的。

思路：利用宽度优先遍历，利用栈同时存储镜像节点（操作特殊）。左节点的左和右节点的右一起先后入栈，然后左节点的右和右节点的左一起先后入栈。同时左右树节点比较。

```python
class Solution:
    def isSymmetrical(self, pRoot):
        # write code here
        if not pRoot:
            return True
        nodes_stack=[pRoot.left,pRoot.right]
        while nodes_stack:
            val_left,val_right=nodes_stack.pop(0),nodes_stack.pop(0)
            if not val_left and not val_right:
                continue
            if not val_left or not val_right:
                return False
            if val_left.val!=val_right.val:
                return False
            nodes_stack.append(val_left.left)
            nodes_stack.append(val_right.right)
            nodes_stack.append(val_left.right)
            nodes_stack.append(val_right.left)

        return True
```

图 2.51 对称的二叉树

2.52 按之字形顺序打印二叉树

算法如图 2.52

请实现一个函数按照之字形打印二叉树，即第一行按照从左到右的顺序打印，第二层按照从右至左的顺序打印，第三行按照从左到右的顺序打印，其他行以此类推。

思路：宽度优先遍历的变体。

2.53 把二叉树打印成多行

算法如图 2.53

从上到下按层打印二叉树，同一层结点从左至右输出。每一层输出一行。

思路：宽度优先遍历。

2.54 最长公共子串

算法如图 2.54

求两个字符串最长公共子串的问题思路：用一个矩阵来记录两个字符串中所有位置的两个字符之间的匹配情况，若是匹配则为 1，否则为 0。然后求出对角线最长的 1 的序列，其对应的位置就是最长匹配子串的位置。

```python
# -*- coding:utf-8 -*-
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution:
    def Print(self, pRoot):
        # write code here
        if not pRoot:
            return []
        printFlag = True
        level, result = [pRoot], []
        while level:
            tmp = []
            tmpList = []
            for node in level:
                tmpList.append(node.val)
                if node.left:
                    tmp.append(node.left)

                if node.right:
                    tmp.append(node.right)
            level = tmp

            if printFlag:
                result.append(tmpList)
                printFlag = False
            else:
                result.append(tmpList[::-1])
                printFlag = True
        return result
```

图 2.52 按之字形顺序打印二叉树

```python
# -*- coding:utf-8 -*-
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution:
    # 返回二维列表[[1,2],[4,5]]
    def Print(self, pRoot):
        # write code here
        if not pRoot:
            return []
        level, result = [pRoot], []
        while level:
            tmp = []
            tmp_list = []
            for node in level:
                tmp_list.append(node.val)
                if node.left:
                    tmp.append(node.left)
                if node.right:
                    tmp.append(node.right)
            level = tmp
            result.append(tmp_list)
        return result
```

图 2.53 把二叉树打印成多行

```python
def find_lcsubstr(s1, s2):
        m=[[0 for i in range(len(s2)+1)]  for j in range(len(s1)+1)]  #生成0矩阵，为方便后续计算，比字符串长度多了一
        mmax=0   #最长匹配的长度
        p=0  #最长匹配对应在s1中的最后一位
        for i in range(len(s1)):
                for j in range(len(s2)):
                        if s1[i]==s2[j]:
                                m[i+1][j+1]=m[i][j]+1
                                if m[i+1][j+1]>mmax:
                                        mmax=m[i+1][j+1]
                                        p=i+1
        return s1[p-mmax:p],mmax   #返回最长子串及其长度
```

图 2.54 最长公共子串

2.55 最长公共子序列

算法如图 2.55

子串要求字符必须是连续的，但是子序列就不是这样。最长公共子序列是一个十分实用的问题，它可以描述两段文字之间的“相似度”，即它们的雷同程度，从而能够用来辨别抄袭。对一段文字进行修改之后，计算改动前后文字的最长公共子序列，将除此子序列外的部分提取出来，这种方法判断修改的部分，往往十分准确。

思路：用动态回归的思想，一个矩阵记录两个字符串中匹配情况，若是匹配则为左上方的值加 1，否则为左方和上方的最大值。一个矩阵记录转移方向，然后根据转移方向，回溯找到最长子序列。

```
def find_lcseque(s1, s2):
        # 生成字符串长度加1的0矩阵，m用来保存对应位置匹配的结果
        m = [ [ 0 for x in range(len(s2)+1) ] for y in range(len(s1)+1) ]
        # d用来记录转移方向
        d = [ [ None for x in range(len(s2)+1) ] for y in range(len(s1)+1) ]

        for p1 in range(len(s1)):
                for p2 in range(len(s2)):
                        if s1[p1] == s2[p2]:            #字符匹配成功，则该位置的值为左上方的值加1
                                m[p1+1][p2+1] = m[p1][p2]+1
                                d[p1+1][p2+1] = 'ok'
                        elif m[p1+1][p2] > m[p1][p2+1]:  #左值大于上值，则该位置的值为左值，并标记回溯时的方向
                                m[p1+1][p2+1] = m[p1+1][p2]
                                d[p1+1][p2+1] = 'left'
                        else:                           #上值大于左值，则该位置的值为上值，并标记方向up
                                m[p1+1][p2+1] = m[p1][p2+1]
                                d[p1+1][p2+1] = 'up'
        (p1, p2) = (len(s1), len(s2))
        print numpy.array(d)
        s = []
        while m[p1][p2]:    #不为None时
                c = d[p1][p2]
                if c == 'ok':   #匹配成功，插入该字符，并向左上角找下一个
                        s.append(s1[p1-1])
                        p1-=1
                        p2-=1
                if c =='left':  #根据标记，向左找下一个
                        p2 -= 1
                if c == 'up':   #根据标记，向上找下一个
                        p1 -= 1
        s.reverse()
        return ''.join(s)
```

图 2.55 最长公共子串