# Leetcode

## Two-sum

Code：

class Solution(object):

def twoSum(self, nums, target):

if len(nums) <= 1:

return False

buff\_dict = {}

for i in range(len(nums)):

if nums[i] in buff\_dict:

return [buff\_dict[nums[i]], i]

else:

buff\_dict[target - nums[i]] = i

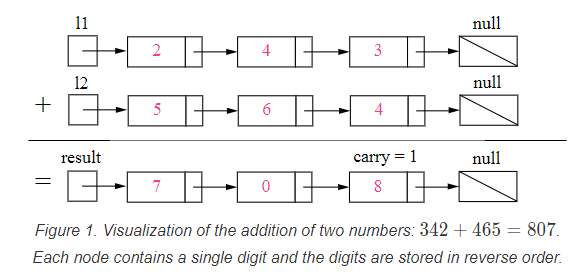
1. 这里运用了dict 的赋值：
   1. buff\_dict[target - nums[i]] = i ：如果 b\_d[3] = 0，则{3:0}
2. 这里只用到一次循环，原理是：
   1. 将dict中未有的元素与target的差取出来放入dict，并赋值（实际上是编号）
   2. 如果dict中有，说明前值 中有可以和该元素相加等于target的元素，则输出前值的序号（即dict中差值的序号） 和该元素的序号。

## Add Two Numbers

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.

You may assume the two numbers do not contain any leading zero, except the number 0 itself.





# 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):

carry = 0

root = n = ListNode(0)

while l1 or l2 or carry:

v1 = v2 = 0

if l1:

v1 = l1.val

l1 = l1.next

print v1

if l2:

v2 = l2.val

l2 = l2.next

print v2

carry, val = divmod(v1+v2+carry, 10)

n.next = ListNode(val)

n = n.next

return root.next

1. root = n = ListNode(0) : ListNode(0)意思是节点值是0的listnode赋值给root和n.
   1. 下面的程序中也是将节点val赋值 给n.next， 节点val的值是val。
2. Carry: 加法中的进位。
3. root.next：如果return root，得到[0，7，0，8]，如果return root.next，输出[7，0，8]。

## Longest Substring Without Repeating Characters

### 正则表达式解：

import re

class Solution(object):

def lengthOfLongestSubstring(self, s):

a = re.findall(r'.',s)

b = []

c = 0

for i in range (len(a)):

if a[i] in b:

x = b.index(a[i])

del b[0:x+1]

b.append(a[i])

else:

b.append(a[i])

c = max(c,len(b))

return c

1. re.findall(r'.',s): 可以将string所有的单个字符分隔开，便于后面字符串的操作

### 字典解

class Solution:

def lengthOfLongestSubstring(self, s):

start = maxLength = 0

usedChar = {}

for i in range(len(s)):

if s[i] in usedChar and start <= usedChar[s[i]]:

start = usedChar[s[i]] + 1

else:

maxLength = max(maxLength, i - start + 1)

usedChar[s[i]] = i

return maxLength

1. 不同之处：
   1. 这里使用字典赋值代替index。
   2. 正则解编辑字符串，并不处理字符串本身，而是选取start的点来计算substring的长度。（这样程序跑的速度稍快）

## [Median of Two Sorted Arrays](https://leetcode.com/problems/median-of-two-sorted-arrays/)

class Solution(object):

def findMedianSortedArrays(self, nums1, nums2):

l1 = sorted(nums1 + nums2)

leg = len(nums1)+len(nums2)

print leg,l1[leg/2],l1[(leg/2)-1]

if leg % 2 != 0:

return float(l1[(leg/2)])

else:

return float(l1[leg/2]+l1[(leg/2)-1])/2

1. 注意float使用的顺序：要先float再除，除完再float跟没float一样。

## Longest Palindromic Substring

class Solution:

def longestPalindrome(self, s):

maxLen=1

start=0

for i in range(len(s)):

# 用来检测非连续重复的palindromic，比如 bcb

if i-maxLen >=1 and s[i-maxLen-1:i+1]==s[i-maxLen-1:i+1][::-1]:

print i,start,maxLen,s[start:start+maxLen]

start=i-maxLen-1

maxLen+=2

continue

# 用来检测连续重复的palindromic，比如bbb

if i-maxLen >=0 and s[i-maxLen:i+1]==s[i-maxLen:i+1][::-1]:

print i,start,maxLen,s[start:start+maxLen]

start=i-maxLen

maxLen+=1

return s[start:start+maxLen]

1. 算法：不管是多长的对称substring, 一定是从中间开始向外对称，中间的对称只有3个元素。eg：bacab
   1. 首先检测到aca（三个元素），再设aca == \*;
   2. 检测到b\*b（三个元素）。

## Reverse Integer

Reverse digits of an integer. Example1: x = 123, return 321; x = -123, return -321

解法一：my own

class Solution(object):

def reverse(self, x):

if "-" not in str(x):

if int(str(x)[::-1]) > 2147483648 or int(str(x)[::-1]) < -2147483648:

return 0

else:

return int(str(x)[::-1])

else:

if int("-"+str(x)[1:][::-1]) > 2147483648 or int("-"+str(x)[1:][::-1]) < -2147483648:

return 0

else:

return int("-"+str(x)[1:][::-1])

解法二：三行简洁解

class Solution(object):

def reverse(self, x):

s=cmp(x,0)

r=int(`s\*x`[::-1])

return(r<2\*\*31)\*s\*r

1. cmp(x,y): 用于比较两个对象，如果 x < y 返回 -1, 如果 x == y 返回 0, 如果 x > y 返回 1。
2. int(`s\*x`[::-1]): `\*\*\*`表示这是string，可以代替str()。
3. return(r<2\*\*31): 括号表示True/False，which the multiplication then treats as 1 or 0.
4. 可以写成一行，用封号连接： s=cmp(x,0)；r=int(`s\*x`[::-1])；return(r<2\*\*31)\*s\*r

解法三：

**def** **reverse**(self, x):

n = cmp(x, 0) \* int(str(abs(x))[::-1])

**return** n **if** n.bit\_length() < 32 **else** 0

1. Return n if n.bit\_length() < 32:bit\_length() 表示位数。

## 10. Regular Expression Matching

import re

class Solution(object):

def isMatch(self, s, p):

p1 = '^'+p+'$'

# print p1

pattern = re.compile(p1)

return bool(re.match(pattern, s))

1. Re.match(pattern, s): 这里是在s字符串里找满足pattern的substring。
2. 需要克服的问题：在sample1 中pattern是”a”，s 是”aa”，match的意思是从s第一字符开始满足pattern就好，显然”aa”满足”a”。但题目要求**完全cover，**所以需要”^a$”（不许比a多，不许比a少）来保证完全cover的要求。

## 11. Container With Most Water

class Solution(object):

def maxArea(self, height):

prod = 0

i, j = 0, len(height)-1

while j > i:

tp = (j-i)\*min(height[i],height[j])

prod = max(prod, tp)

if height[i] < height[j]:

i += 1

else:

j -= 1

return prod

1. 算法：
   1. 取初始值；
   2. 使用循环，取该情况最优面积值—〉与初始值比较—〉判断是否更换初始值（越大越好）
   3. 返回最优面积值

## 15. 3Sum

解法一：my own---time exceeded

b = []

for i in list(it.combinations(nums,3)):

if sum(i) == 0 and sorted(i) not in b:

b.append(sorted(i))

return b

解法二：

res = []

nums.sort()

for i in xrange(len(nums)-2):

# 如果现初始值等于前初始值，输出的结果将是一样的，可以忽略。

if i > 0 and nums[i] == nums[i-1]:

continue

l, r = i+1, len(nums)-1

while l < r:

s = nums[i] + nums[l] + nums[r]

if s < 0:

l +=1

elif s > 0:

r -= 1

else:

res.append((nums[i], nums[l], nums[r]))

# 意思是等于了之后必须大的变小，小的变大才有可能再次等于，除非有相同的值出现。

while l < r and nums[l] == nums[l+1]:

l += 1

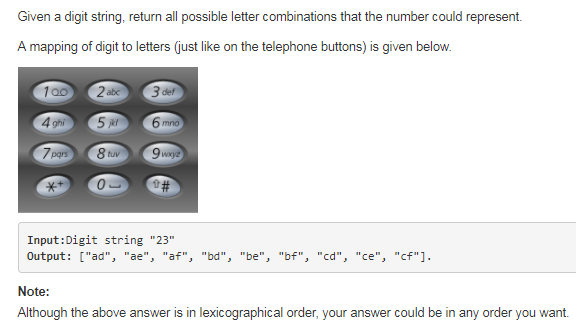
while l < r and nums[r] == nums[r-1]:

r -= 1

l += 1; r -= 1

return res

## 17. Letter Combinations of a Phone Number



解法一：my own

import re

import itertools as it

class Solution(object):

def letterCombinations(self, digits):

b = []

dict = {2: ["a","b","c"],3:["d","e","f"],4:["g","h","i"],5:["j","k","l"],6:["m","n","o"],7:["p","q","r","s"],8:["t","u","v"],9:["w","x","y","z"]}

if digits != "":

digreg = re.findall(r'.',digits)

dig1 = [dict[int(i)] for i in digreg]

a = list(it.product(\*dig1))

for i in range (len(a)):

a[i] = ''.join(a[i])

return a

else:

return []

1. 运用了数组，字典，itertools中的product（迭代），正则表达式：
   1. 先用字典对数字进行赋值；
   2. 再用正则表达式分隔输入的数字字符串；
   3. 再用字典和product找出数字对应的字符串的所有组合；
   4. 最后用join把输出格式化，再return。

## 21. Merge Two Sorted Lists

解法一：my—own 36.41%

class Solution(object):

def mergeTwoLists(self, l1, l2):

root = n = ListNode(0)

while l1 or l2:

if l1 and l2 and l1.val <= l2.val:

n.next = ListNode(l1.val)

n = n.next

l1 = l1.next

elif l1 and l2 and l1.val > l2.val:

n.next = ListNode(l2.val)

n = n.next

l2 = l2.next

elif l1:

n.next = ListNode(l1.val)

n = n.next

l1 = l1.next

elif l2:

n.next = ListNode(l2.val)

n = n.next

l2 = l2.next

return root.next

解法二：最快解

class Solution(object):

def mergeTwoLists(self, l1, l2):

head = n = ListNode(0)

while l1 or l2:

# 使用not来表示否定

if not l1 or not l2:

# 此处l1或者l2的赋值是把还未赋值的逻辑点全赋值了。

n.next = l1 or l2

break

if l1.val < l2.val:

n.next = l1

l1 = l1.next

else:

n.next = l2

l2 = l2.next

n = n.next

return head.next

1. 使用not来表示否定。
2. 此处l1或者l2的赋值是把还未赋值的逻辑点全赋值了。
3. 注意：赋值的时候其实是赋值整条逻辑链。逻辑链的val其实是逻辑链的起始点，next是下一个点。上题其实是用val，next使head在两条逻辑链间转换，直到有一个逻辑链的val用光了，再赋值给另一个逻辑链的val就可以不用管了，因为已经附上了另一个逻辑链剩下的所有val。

## 22. Generate Parentheses

解法一：time exceeded

import itertools as it

class Solution(object):

def generateParenthesis(self, n):

b = []

x = []

for \_ in range (0,n):

b.extend("()")

b = sorted(b)

c = set(it.permutations(b))

print c

for i in c:

if i[0] == ")" or i[-1] == "(":

continue

sum1 = 0

sum2 = 0

for j in i:

if j == "(":

sum1 += 1

else:

sum2 += 1

if sum2 > sum1:

break

if sum1 == sum2:

x.append(i)

for i in range (len(x)):

x[i] = ''.join(x[i])

return x

解法二：generate

class Solution(object):

def generateParenthesis(self, n):

def generate(p, left, right):

if right >= left >= 0:

if not right:

yield p

for q in generate(p + '(', left-1, right): yield q

for q in generate(p + ')', left, right-1): yield q

return list(generate('', n, n))