# LeetCode总结一: 双指针与滑动窗口

Two Pointers and Sliding Window

002-Add Two Numbers  
003-Longest Substring Without Repeating Characters  
005-Longest Palindromic Substring  
011-Container With Most Water  
015-3Sum  
016-3Sum Closet  
018-4Sum  
026-Remove Duplicate FromSorted Array  
027-Remove Elements  
028-Implement strStr()  
042-Trapping Rain Water  
064-Text Justification  
076-Minimum Window Substring  
080-Remove Duplicates from Sorted Array II  
125-Valid Palindrome  
159-Longest Substring with At Most Two Distinct Characters  
340-Longest Substring with At Most K Distinct Characters  
395-Longest Substring with At Least K Repeating Characters  
161-One Edit Distance  
167-Two Sum II - Input array is sorted  
209-Minimum Size Subarray Sum  
239-Sliding Window Maximum  
244-Shortest Word Distance II  
245-Shortest Word Distance III  
259-3Sum Smaller  
274-H-Index  
283-Move Zeroes  
344-Reverse String  
345-Reverse Vowels of a String  
438-Find All Anagrams in a String  
482-License Key Formatting

### 002-Add Two Number

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.

Example:

Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)

Output: 7 -> 0 -> 8

Explanation: 342 + 465 = 807.

# 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

"""

carry = 0

dummy = ListNode(-1)

cur = dummy

while l1 or l2:

a, b = 0, 0

if l1:

a = l1.val

l1 = l1.next

if l2:

b = l2.val

l2 = l2.next

tot = a + b + carry

carry = tot / 10

cur.next = ListNode(tot % 10)

cur = cur.next

if carry != 0:

cur.next = ListNode(carry)

return dummy.next

### 003 - Longest Substring Without Repeating Characters

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

Example 1:

Input: "abcabcbb"

Output: 3

Explanation: The answer is "abc", which the length is 3.

Example 2:

Input: "bbbbb"

Output: 1

Explanation: The answer is "b", with the length of 1.

Example 3:

Input: "pwwkew"

Output: 3

Explanation: The answer is "wke", with the length of 3.

Note that the answer must be a substring, "pwke" is a subsequence and not a substring.

class Solution(object):

def lengthOfLongestSubstring(self, s):

"""

:type s: str

:rtype: int

"""

dic = {}

i, j = 0, 0

n = len(s)

res = 0

while j < n:

char = s[j]

if char in dic and i <= dic[char]:

res = max(res, j - i)

i = dic[char] + 1

dic[char] = j

j += 1

return max(res, j - i)

### 005 - Longest Palindromic Substring

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

Example 1:

Input: "babad"

Output: "bab"

Note: "aba" is also a valid answer.

Example 2:

Input: "cbbd"

Output: "bb"

class Solution(object):

def longestPalindrome(self, s):

"""

:type s: str

:rtype: str

"""

res = ''

maxLen = 0

n = len(s)

start, end = 0, 0

for i in range(n):

oddLen = self.palindromeLength(i, i, n, s)

evenLen = self.palindromeLength(i, i + 1, n, s)

curMax = max(oddLen, evenLen)

if curMax > maxLen:

start = i - (curMax - 1) / 2

end = i + curMax / 2

res = s[start : end + 1]

maxLen = curMax

return res

def palindromeLength(self, i, j, n, s):

while i >= 0 and j < n and s[i] == s[j]:

i -= 1

j += 1

return j - i - 1

### 011-Container With Most Water

Given 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.

Example:

Input: [1,8,6,2,5,4,8,3,7]

Output: 49

class Solution(object):

def maxArea(self, height):

"""

:type height: List[int]

:rtype: int

"""

n = len(height)

res = 0

l, r = 0, n - 1

while l < r:

area = min(height[l], height[r]) \* (r - l)

res = max(res, area)

if height[l] < height[r]:

l += 1

else:

r -= 1

return res

### 015-3Sum

Given 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.

Example:

Given array nums = [-1, 0, 1, 2, -1, -4],

A solution set is:

[

[-1, 0, 1],

[-1, -1, 2]

]

class Solution(object):

def threeSum(self, nums):

"""

:type nums: List[int]

:rtype: List[List[int]]

"""

nums.sort()

n = len(nums)

res = []

for i in range(n - 2):

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

continue

j = i + 1

k = n - 1

while j < k:

tot = nums[i] + nums[j] + nums[k]

if tot == 0:

res.append([nums[i], nums[j], nums[k]])

while j < k and nums[j] == nums[j + 1]:

j += 1

while j < k and nums[k] == nums[k - 1]:

k -= 1

j += 1

k -= 1

elif tot < 0:

j += 1

else:

k -= 1

return res

### 16. 3Sum Closest

Given an array nums of n integers and an integer target, find three  
integers in nums such that the sum is closest to target.  
Return the sum of the three integers. You may assume that each  
input would have exactly one solution.

Example:

Given array nums = [-1, 2, 1, -4], and target = 1.

The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).

class Solution(object):

def threeSumClosest(self, nums, target):

"""

:type nums: List[int]

:type target: int

:rtype: int

"""

nums.sort()

n = len(nums)

diff = 1 << 31

res = 0

for i in range(n - 2):

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

continue

j = i + 1

k = n - 1

while j < k:

tot = nums[i] + nums[j] + nums[k]

if tot == target:

return tot

else:

curDiff = abs(tot - target)

if curDiff < diff:

res = tot

diff = curDiff

if tot < target:

j += 1

else:

k -= 1

return res

### 018-4Sum

Given an array nums of n integers and an integer target, are there elements a, b, c, and d in nums  
such that a + b + c + d = target? Find all unique quadruplets in the array which gives the sum of target.

Note:

The solution set must not contain duplicate quadruplets.

Example:

Given array nums = [1, 0, -1, 0, -2, 2], and target = 0.

A solution set is:

[

[-1, 0, 0, 1],

[-2, -1, 1, 2],

[-2, 0, 0, 2]

]

class Solution(object):

def fourSum(self, nums, target):

"""

:type nums: List[int]

:type target: int

:rtype: List[List[int]]

"""

nums.sort()

n = len(nums)

res = []

for i in range(n - 3):

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

continue

for j in range(i + 1, n - 2):

if j > i + 1 and nums[j] == nums[j - 1]:

continue

head = j + 1

tail = n - 1

while head < tail :

sum = nums[i] + nums[j] + nums[head] + nums[tail]

if sum == target:

res.append([nums[i], nums[j], nums[head], nums[tail]])

head += 1

tail -= 1

while head < tail and nums[head] == nums[head - 1]:

head += 1

while head < tail and nums[tail] == nums[tail + 1]:

tail -= 1

elif sum < target:

head += 1

else:

tail -= 1

return res

### 26. Remove Duplicates from Sorted Array

Given a sorted array nums, remove the duplicates in-place such that  
each element appear only once and return the new length.

Do not allocate extra space for another array, you must do this  
by modifying the input array in-place with O(1) extra memory.

Example 1:

Given nums = [1,1,2],

Your function should return length = 2, with the first two elements of nums being 1 and 2 respectively.

It doesn't matter what you leave beyond the returned length.

Example 2:

Given nums = [0,0,1,1,1,2,2,3,3,4],

Your function should return length = 5, with the first five elements of nums

being modified to 0, 1, 2, 3, and 4 respectively.

It doesn't matter what values are set beyond the returned length.

class Solution(object):

def removeDuplicates(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

if not nums:

return 0

idx = 0

n = len(nums)

for i in range(1, n):

if nums[i] != nums[idx]:

idx += 1

nums[idx] = nums[i]

return idx + 1

### 027-Remove Elements

Given an array nums and a value val, remove all instances of that value  
in-place and return the new length.  
Do not allocate extra space for another array, you must do this  
by modifying the input array in-place with O(1) extra memory.  
The order of elements can be changed.  
It doesn't matter what you leave beyond the new length.

Example 1:

Given nums = [3,2,2,3], val = 3,

Your function should return length = 2, with the first two elements of nums being 2.

It doesn't matter what you leave beyond the returned length.

Example 2:

Given nums = [0,1,2,2,3,0,4,2], val = 2,

Your function should return length = 5, with the first five elements of nums containing 0, 1, 3, 0, and 4.

Note that the order of those five elements can be arbitrary.

It doesn't matter what values are set beyond the returned length.

class Solution(object):

def removeElement(self, nums, val):

"""

:type nums: List[int]

:type val: int

:rtype: int

"""

idx = 0

n = len(nums)

for i in range(n):

if nums[i] != val:

nums[idx] = nums[i]

idx += 1

return idx

### 028-Implement strStr()

Return the index of the first occurrence of needle in haystack,  
or -1 if needle is not part of haystack.

Example 1:

Input: haystack = "hello", needle = "ll"

Output: 2

Example 2:

Input: haystack = "aaaaa", needle = "bba"

Output: -1

class Solution(object):

def strStr(self, haystack, needle):

"""

:type haystack: str

:type needle: str

:rtype: int

"""

m = len(haystack)

n = len(needle)

if n == 0:

return 0

for i in range(m - n + 1) :

if haystack[i : i + n] == needle :

return i

return -1

### 042-Trapping Rain Water

Given n non-negative integers representing an elevation map where the width of each bar is 1,  
compute how much water it is able to trap after raining.  
Example:

Input: [0,1,0,2,1,0,1,3,2,1,2,1]

Output: 6

class Solution(object):

def trap(self, height):

"""

:type height: List[int]

:rtype: int

"""

n = len(height)

if n == 0:

return 0

res = 0

hmax = 0

for i in range(n):

if height[i] > height[hmax]:

hmax = i

t = 0

for i in range(hmax + 1):

if height[i] < height[t]:

res += (height[t] - height[i])

else:

t = i

t = n - 1

for i in range(n - 1, hmax - 1, -1):

if height[i] < height[t]:

res += (height[t] - height[i])

else:

t = i

return res

### 064-Text Justitification

Given an array of words and a width maxWidth, format the text such that each line has exactly  
maxWidth characters and is fully (left and right) justified.

You should pack your words in a greedy approach; that is, pack as many words as you can in each line.  
Pad extra spaces ' ' when necessary so that each line has exactly maxWidth characters.

Extra spaces between words should be distributed as evenly as possible. If the number of spaces on a line  
do not divide evenly between words, the empty slots on the left will be assigned more spaces than the  
slots on the right.

For the last line of text, it should be left justified and no extra space is inserted between words.

Note:

A word is defined as a character sequence consisting of non-space characters only.  
Each word's length is guaranteed to be greater than 0 and not exceed maxWidth.  
The input array words contains at least one word.

Example 1:

Input:

words = ["This", "is", "an", "example", "of", "text", "justification."]

maxWidth = 16

Output:

[

"This is an",

"example of text",

"justification. "

]

Example 2:

Input:

words = ["What","must","be","acknowledgment","shall","be"]

maxWidth = 16

Output:

[

"What must be",

"acknowledgment ",

"shall be "

]

Explanation: Note that the last line is "shall be " instead of "shall be",

because the last line must be left-justified instead of fully-justified.

Note that the second line is also left-justified becase it contains only one word.

Example 3:

Input:

words = ["Science","is","what","we","understand","well","enough","to","explain",

"to","a","computer.","Art","is","everything","else","we","do"]

maxWidth = 20

Output:

[

"Science is what we",

"understand well",

"enough to explain to",

"a computer. Art is",

"everything else we",

"do "

]

class Solution(object):

def fullJustify(self, words, maxWidth):

"""

:type words: List[str]

:type maxWidth: int

:rtype: List[str]

"""

res = []

idx = 0

length = len(words)

while idx < length:

cnt = len(words[idx])

last = idx + 1

while last < length:

if cnt + len(words[last]) + 1 > maxWidth:

break

cnt += (1 + len(words[last]))

last += 1

line = ""

diff = last - idx - 1

if last == length or diff == 0:

# in the case of the last line or single word situation

for i in range(idx, last):

if i < last - 1:

line += words[i] + " "

else:

line += words[i]

curLineLen = len(line)

for i in range(curLineLen, maxWidth):

line += " "

else:

space = (maxWidth - cnt) / diff

remain = (maxWidth - cnt) % diff

for i in range(idx, last):

line += words[i]

extraspace = 1 if i - idx < remain else 0

if i < last - 1:

for j in range(space + 1 + extraspace):

line += " "

res.append(line)

idx = last

return res

### 076-Minimum Window Substring

Given a string S and a string T, find the minimum window in S which will  
contain all the characters in T in complexity O(n).

Example:

Input: S = "ADOBECODEBANC", T = "ABC"

Output: "BANC"

Note:  
If there is no such window in S that covers all characters in T,  
return the empty string "". If there is such window, you are guaranteed that there will always be only one unique minimum window in S.

from collections import defaultdict

class Solution(object):

def minWindow(self, s, t):

"""

:type s: str

:type t: str

:rtype: str

"""

lens = len(s)

lent = len(t)

count = {}

dic = defaultdict(list)

window = []

res = ''

for c in t:

if c in count:

count[c] += 1

else:

count[c] = 1

for i in range(lens):

char = s[i]

if char not in count:

continue

if dic.get(char) is not None and len(dic[char]) == count[char]:

idx = dic[char].pop(0)

window.remove(idx)

dic[char].append(i)

window.append(i)

if (res == '' or window[-1] - window[0] < len(res)) and len(window) == lent:

res = s[window[0] : window[-1] + 1]

return res

### 080-Remove Duplicates from Sorted Array II

Given a sorted array nums, remove the duplicates in-place such that duplicates appeared at most  
twice and return the new length.

Do not allocate extra space for another array, you must do this by modifying the input array in-place  
with O(1) extra memory.

Example 1:

Given nums = [1,1,1,2,2,3],

Your function should return length = 5, with the first five elements of

nums being 1, 1, 2, 2 and 3 respectively.

It doesn't matter what you leave beyond the returned length.

Example 2:

Given nums = [0,0,1,1,1,1,2,3,3],

Your function should return length = 7, with the first seven elements of

nums being modified to 0, 0, 1, 1, 2, 3 and 3 respectively.

It doesn't matter what values are set beyond the returned length.

class Solution(object):

def removeDuplicates(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

return self.helper(nums, 2)

def helper(self, nums, k):

idx = 1

cnt = 1

n = len(nums)

if n < k:

return n

for i in range(1, n):

if nums[i] == nums[i - 1]:

if cnt < k:

nums[idx] = nums[i]

idx += 1

cnt += 1

else:

nums[idx] = nums[i]

cnt = 1

idx += 1

return idx

### 125-Valid Palindrome

Given a string, determine if it is a palindrome, considering only  
alphanumeric characters and ignoring cases.

Note: For the purpose of this problem, we define empty string as valid palindrome.

Example 1:

Input: "A man, a plan, a canal: Panama"

Output: true

Example 2:

Input: "race a car"

Output: false

class Solution(object):

def isPalindrome(self, s):

"""

:type s: str

:rtype: bool

"""

n = len(s)

l, r = 0, n - 1

while l <= r :

while l < r and not (s[l].isdigit() or s[l].isalpha()) :

l += 1

while l < r and not (s[r].isdigit() or s[r].isalpha()) :

r -= 1

if s[l].lower() != s[r].lower() :

return False

l += 1

r -= 1

return True

### 159-Longest Substring with At Most Two Distinct Characters

See 340 Below

### 340-Longest Substring with At Most K Distinct Characters

Given a string, find the length of the longest substring T  
that contains at most k distinct characters.

Example 1:

Input: s = "eceba", k = 2

Output: 3

Explanation: T is "ece" which its length is 3.

Example 2:

Input: s = "aa", k = 1

Output: 2

Explanation: T is "aa" which its length is 2.

class Solution(object):

def lengthOfLongestSubstringKDistinct(self, s, k):

"""

:type s: str

:type k: int

:rtype: int

"""

cnt = 0

dic = {}

n = len(s)

res = 0

i = idx = 0

while i < n:

c = s[i]

if c in dic:

dic[c] += 1

else:

dic[c] = 1

while len(dic) > k:

res = max(res, i - idx)

char = s[idx]

if dic[char] > 1:

dic[char] -= 1

else:

del dic[char]

idx += 1

i += 1

res = max(res, i - idx)

return res

### 395-Longest Substring with At Least K Repeating Characters

Find the length of the longest substring T of a given string (consists of  
lowercase letters only) such that every character in T appears no  
less than k times.

Example 1:

Input:

s = "aaabb", k = 3

Output:

3

The longest substring is "aaa", as 'a' is repeated 3 times.

Example 2:

Input:

s = "ababbc", k = 2

Output:

5

The longest substring is "ababb", as 'a' is repeated 2 times and 'b' is repeated 3 times.

Example 3: (more general)

Input:

s = "gababbcfckikiii", k = 2

output:

6

We have two substring in which every character repeats at least 2 times, 'ababb' and 'kikiii'.

The second one is longer than the first one, so the answer is 6

Java **Code**:

public class Solution {

public static void main(String[] args) {

String s = "gababbcfckikiii";

int k = 2;

int res = longestSubstring(s, k);

System.out.println(res);

}

public static int longestSubstring(String s, int k) {

return helper(s, k, 0, s.length() - 1);

}

private static int helper(String s, int k, int start, int end){

if(start > end) return 0;

int[] count = new int[26];

for(int i = start; i <= end; i++){

count[s.charAt(i) - 'a']++;

}

for(int i = 0; i < 26; i++){

if(count[i] > 0 && count[i] < k){

char chr = (char)(i + 'a');

int pos = s.indexOf(chr, start);

int left = helper(s, k, start, pos - 1);

int right = helper(s, k, pos + 1, end);

return Math.max(left, right);

}

}

System.out.println(s.substring(start, end + 1));

return end - start + 1;

}

}

Python **code**:

class Solution(object):

def longestSubstring(self, s, k):

"""

:type s: str

:type k: int

:rtype: int

"""

return self.helper(s, k, 0, len(s) - 1)

def helper(self, s, k, start, end):

if start > end:

return 0

count = [0] \* 26

for i in range(start, end + 1):

count[ord(s[i]) - ord('a')] += 1

for i in range(26):

if 0 < count[i] < k:

char = chr(i + ord('a'))

pos = s.index(char, start)

left = self.helper(s, k, start, pos - 1)

right = self.helper(s, k, pos + 1, end)

return max(left, right)

return end - start + 1

### 161-One Edit Distance

Given two strings s and t, determine if they are both one edit distance apart.

Note:

There are 3 possiblities to satisify one edit distance apart:

Insert a character into s to get t  
Delete a character from s to get t  
Replace a character of s to get t

Example 1:

Input: s = "ab", t = "acb"

Output: true

Explanation: We can insert 'c' into s to get t.

Example 2:

Input: s = "cab", t = "ad"

Output: false

Explanation: We cannot get t from s by only one step.

Example 3:

Input: s = "1203", t = "1213"

Output: true

Explanation: We can replace '0' with '1' to get t.

class Solution(object):

def isOneEditDistance(self, s, t):

"""

:type s: str

:type t: str

:rtype: bool

"""

m = len(s)

n = len(t)

if s == t or abs(m - n) > 1:

return False

if m > n:

return self.isOneEditDistance(t, s)

cnt = 0

i = j = 0

while i < m:

if s[i] != t[j]:

if cnt == 1:

return False

if m < n:

i -= 1

cnt += 1

i += 1

j += 1

return True

### 167-Two Sum II - Input array is sorted (Too Easy, skip)

### 209-Minimum Size Subarray Sum \*

Given an array of n positive integers and a positive integer s,  
find the minimal length of a contiguous  
subarray of which the sum ≥ s. If there isn't one, return 0 instead.

Example:

Input: s = 7, nums = [2,3,1,2,4,3]

Output: 2

Explanation: the subarray [4,3] has the minimal length under the problem constraint.

class Solution(object):

def minSubArrayLen(self, s, nums):

"""

:type s: int

:type nums: List[int]

:rtype: int

"""

if not nums:

return 0

n = len(nums)

i, j = 0, 0

sum = 0

res = n + 1

while i < n and j < n:

while j < n and sum < s:

sum += nums[j]

j += 1

while i < n and sum >= s:

res = min(res, j - i)

sum -= nums[i]

i += 1

return res if res < n + 1 else 0

### 239-Sliding Window Maximum

Given an array nums, there is a sliding window of size k which is moving from  
the very left of the array to the very right.  
You can only see the k numbers in the window.  
Each time the sliding window moves right by one position. Return the max sliding window.

Example:

Input: nums = [1,3,-1,-3,5,3,6,7], and k = 3

Output: [3,3,5,5,6,7]

Explanation:

Window position Max

--------------- -----

[1 3 -1] -3 5 3 6 7 3

1 [3 -1 -3] 5 3 6 7 3

1 3 [-1 -3 5] 3 6 7 5

1 3 -1 [-3 5 3] 6 7 5

1 3 -1 -3 [5 3 6] 7 6

1 3 -1 -3 5 [3 6 7] 7

Note:  
You may assume k is always valid, 1 ≤ k ≤ input array's size for non-empty array.

Follow up:  
Could you solve it in linear time?

Solution 1: Using Priority Queue O(nlogn)

class Solution {

public int[] maxSlidingWindow(int[] nums, int k) {

int len = nums.length;

if (len == 0) {

return new int[0];

}

PriorityQueue<Integer> queue = new PriorityQueue<>(new Comparator<Integer>() {

public int compare(Integer a, Integer b) {

return b - a;

}

});

for (int i = 0; i < k; i++) {

queue.offer(nums[i]);

}

int[] res = new int[len - k + 1];

res[0] = queue.peek();

for (int j = k; j < len; j++) {

int idx = j - k;

queue.remove(nums[idx]);

queue.offer(nums[j]);

res[idx + 1] = queue.peek();

}

return res;

}

}

# Solution 2: Using dqueue O(n)

class Solution(object):

def maxSlidingWindow(self, nums, k):

"""

:type nums: List[int]

:type k: int

:rtype: List[int]

"""

res = []

queue = []

n = len(nums)

for i in range(n):

while queue and queue[-1] < nums[i] :

queue.pop()

queue.append(nums[i])

j = i - k

if j >= 0 and queue[0] == nums[j] :

queue.pop(0)

if j >= -1:

res.append(queue[0])

return res

### 244-Shortest Word Distance II

Design a class which receives a list of words in the constructor, and implements a method that takes two words  
word1 and word2 and return the shortest distance between these two words in the list.  
Your method will be called repeatedly many times with different parameters.

Example:

Assume that words = ["practice", "makes", "perfect", "coding", "makes"].

Input: word1 = “coding”, word2 = “practice”

Output: 3

Input: word1 = "makes", word2 = "coding"

Output: 1

class WordDistance(object):

def \_\_init\_\_(self, words):

"""

:type words: List[str]

"""

self.dic = {}

self.length = len(words)

for i, s in enumerate(words):

if s in self.dic:

self.dic[s].append(i)

else:

self.dic[s] = [i]

def shortest(self, word1, word2):

"""

:type word1: str

:type word2: str

:rtype: int

"""

pos1 = self.dic[word1]

pos2 = self.dic[word2]

n1 = len(pos1)

n2 = len(pos2)

i, j = 0, 0

res = self.length + 1

while i < n1 and j < n2:

res = min(res, abs(pos2[j] - pos1[i]))

if pos1[i] < pos2[j]:

i += 1

else:

j += 1

return res

# Your WordDistance object will be instantiated and called as such:

# obj = WordDistance(words)

# param\_1 = obj.shortest(word1,word2)

### 245-Shortest Word Distance III

Given a list of words and two words word1 and word2, return the shortest distance between these two words in the list.

word1 and word2 may be the same and they represent two individual words in the list.

Example:

Assume that words = ["practice", "makes", "perfect", "coding", "makes"].

Input: word1 = “makes”, word2 = “coding”

Output: 1

Input: word1 = "makes", word2 = "makes"

Output: 3

class Solution(object):

def shortestWordDistance(self, words, word1, word2):

"""

:type words: List[str]

:type word1: str

:type word2: str

:rtype: int

"""

idx = -1

n = len(words)

res = n + 1

for i in range(n):

w = words[i]

if w == word1 or w == word2:

if idx != -1 and (word1 == word2 or words[idx] != w):

res = min(res, i - idx)

idx = i

return res

### 259-3Sum Smaller

Given an array of n integers nums and a target, find the number of index triplets i, j, k  
with 0 <= i < j < k < n that satisfy the condition nums[i] + nums[j] + nums[k] < target.

Example:

Input: nums = [-2,0,1,3], and target = 2

Output: 2

Explanation: Because there are two triplets which sums are less than 2:

[-2,0,1]

[-2,0,3]

Follow up: Could you solve it in O(n2) runtime?

class Solution(object):

def threeSumSmaller(self, nums, target):

"""

:type nums: List[int]

:type target: int

:rtype: int

"""

n = len(nums)

nums.sort()

cnt = 0

for i in range(n - 2):

j = i + 1

k = n - 1

while j < k:

tot = nums[i] + nums[j] + nums[k]

if tot < target:

cnt += (k - j)

j += 1

else:

k -= 1

return cnt

### 274-H-Index

Given an array of citations (each citation is a non-negative integer) of a researcher,  
write a function to compute the researcher's h-index.

According to the definition of h-index on Wikipedia:  
"A scientist has index h if h of his/her N papers have at least h citations each, and the other N − h papers  
have no more than h citations each."

Example:

Input: citations = [3,0,6,1,5]

Output: 3

Explanation:

[3,0,6,1,5] means the researcher has 5 papers in total and each of them had

received 3, 0, 6, 1, 5 citations respectively.

Since the researcher has 3 papers with at least 3 citations each and the remaining

two with no more than 3 citations each, her h-index is 3.

class Solution(object):

def hIndex(self, citations):

"""

:type citations: List[int]

:rtype: int

"""

citations.sort()

n = len(citations)

for i in range(n):

h = n - i

if citations[i] >= h:

return h

return 0

### 283-Move Zeroes

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.

Example:

Input: [0,1,0,3,12]

Output: [1,3,12,0,0]

Note:  
You must do this in-place without making a copy of the array.  
Minimize the total number of operations.

class Solution(object):

def moveZeroes(self, nums):

"""

:type nums: List[int]

:rtype: void Do not return anything, modify nums in-place instead.

"""

n = len(nums)

idx = 0

for i in range(n):

if nums[i] != 0:

nums[idx], nums[i] = nums[i], nums[idx]

idx += 1

### 344-Reverse String (skip)

### 345-Reverse Vowels of a String (skip)

### 346-Moving Average from Data Stream

Given a stream of integers and a window size, calculate the moving average of all integers in the sliding window.

Example:

MovingAverage m = new MovingAverage(3);

m.next(1) = 1

m.next(10) = (1 + 10) / 2

m.next(3) = (1 + 10 + 3) / 3

m.next(5) = (10 + 3 + 5) / 3

class MovingAverage(object):

def \_\_init\_\_(self, size):

"""

Initialize your data structure here.

:type size: int

"""

self.size = size

self.total = 0

self.q = []

def next(self, val):

"""

:type val: int

:rtype: float

"""

if len(self.q) < self.size :

self.q.append(val)

self.total = float(sum(self.q))

return self.total / len(self.q)

else:

head = self.q.pop(0)

self.q.append(val)

self.total = self.total - head + val

return self.total / self.size

# Your MovingAverage object will be instantiated and called as such:

# obj = MovingAverage(size)

# param\_1 = obj.next(val)

### 438-Find All Anagrams in a String

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.

Example 1:

Input:

s: "cbaebabacd" p: "abc"

Output:

[0, 6]

Explanation:

The substring with start index = 0 is "cba", which is an anagram of "abc".

The substring with start index = 6 is "bac", which is an anagram of "abc".

class Solution(object):

def findAnagrams(self, s, p):

"""

:type s: str

:type p: str

:rtype: List[int]

"""

count = [0] \* 26

for c in p:

count[ord(c) - ord('a')] += 1

ns = len(s)

np = len(p)

cnt = np

idx = 0

res = []

for i in range(ns):

char = s[i]

posi = ord(char) - ord('a')

if count[posi] >= 1:

cnt -= 1

count[posi] -= <

### 482. License Key Formatting

Now you are given a string S, which represents a software license key which we would like to format. The string S is composed of alphanumerical characters and dashes. The dashes split the alphanumerical characters within the string into groups. (i.e. if there are M dashes, the string is split into M+1 groups). The dashes in the given string are possibly misplaced.

We want each group of characters to be of length K (except for possibly the first group, which could be shorter, but still must contain at least one character). To satisfy this requirement, we will reinsert dashes. Additionally, all the lower case letters in the string must be converted to upper case.

So, you are given a non-empty string S, representing a license key to format, and an integer K. And you need to return the license key formatted according to the description above.

Example 1:

Input: S = "2-4A0r7-4k", K = 4

Output: "24A0-R74K"

Explanation: The string S has been split into two parts, each part has 4 characters.

Example 2:

Input: S = "2-4A0r7-4k", K = 3

Output: "24-A0R-74K"

Explanation: The string S has been split into three parts, each part has 3 characters except the first part as it could be shorter as said above.

Note:

The length of string S will not exceed 12,000, and K is a positive integer.

String S consists only of alphanumerical characters (a-z and/or A-Z and/or 0-9) and dashes(-).

String S is non-empty.

题目是将一个序列号按照一定格式输出。首先要将原序列号中的分隔符‘-’删除，并把字母都换成大写。然后每隔K个加一个分隔符‘-’，分隔成若干个子序列，注意第一个子序列字符个数可以小于或等于K个。

# LeetCode总结二: 排序相关题目

### 排序及相关(三路快排, 归并, 快排)

021-Merge Two Sorted Lists  
023-Merge k Sorted Lists  
056-Merge Intervals  
057-Insert Interval  
075-Sort Colors  
088-Merge Sorted Array  
163-Missing Ranges  
189-Rotate Array  
215-Kth Largest Element in an Array  
228-Summary Ranges  
252-Meeting Rooms  
280-Wiggle Sort  
334-Increasing Triplet Subsequence  
414-Third Maximum Number

### Quick Sort implementation

quick sort implementation using partition and recurssion  
  
In worse case (when the array is already sorted), this algorithm takes O(n^2)  
On average, it's time complexity is O(nlogn)

class QuickSort(object):

def sort(self, nums, left, right):

if not nums or left > right:

return

pos = self.partition(nums, left, right)

self.sort(nums, left, pos - 1)

self.sort(nums, pos + 1, right)

def partition(self, nums, left, right):

pivot = nums[left]

l, r = left, right

while l <= r:

while l <= r and nums[l] <= pivot:

l += 1

while l <= r and nums[r] > pivot:

r -= 1

if l > r:

break

nums[l], nums[r] = nums[r], nums[l]

nums[left], nums[r] = nums[r], nums[left]

return r

test = QuickSort()

# nums = [3, 9, 1, 1, 4, 9, 8, 10, 7, 6, 2]

# nums = [1, 1, 1]

# nums = [0, 1, 2]

# nums = [1]

n = len(nums)

test.sort(nums, 0, n - 1)

print nums

### Merge Sort implementation

Similar to quick sort, in worse case the time complexity is O(n^2)

class MergeSort(object):

def sort(self, nums, left, right):

if left >= right:

return

mid = left + (right - left) / 2

self.sort(nums, left, mid)

self.sort(nums, mid + 1, right)

self.merge(nums, left, mid, right)

# The merge process is the same as LeeCode 88

def merge(self, nums, left, mid, right):

tmp = nums[mid + 1 : right + 1]

idx = right

i = mid

j = len(tmp) - 1

while i >= left and j >= 0:

if nums[i] < tmp[j]:

nums[idx] = tmp[j]

j -= 1

else:

nums[idx] = nums[i]

i -= 1

idx -= 1

while j >= 0:

nums[idx] = tmp[j]

j -= 1

idx -= 1

### 021-Merge Two Sorted Lists

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.

Example:

Input: 1->2->4, 1->3->4

Output: 1->1->2->3->4->4

Solution:

# 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

"""

dummy = ListNode(-1)

cur = dummy

while l1 and l2:

if l1.val < l2.val:

cur.next = l1

l1 = l1.next

else:

cur.next = l2

l2 = l2.next

cur = cur.next

if l1:

cur.next = l1

else:

cur.next = l2

return dummy.next

### 023-Merge k Sorted Lists

Merge k sorted linked lists and return it as one sorted list.  
Analyze and describe its complexity.

Example:

Input:

[

1->4->5,

1->3->4,

2->6

]

Output: 1->1->2->3->4->4->5->6

Solution:

Java **Code**:

/\*\*

\* Definition for singly-linked list.

\* public class ListNode {

\* int val;

\* ListNode next;

\* ListNode(int x) { val = x; }

\* }

\*/

class Solution {

public ListNode mergeKLists(ListNode[] lists) {

PriorityQueue<ListNode> pq = new PriorityQueue<>(new Comparator<ListNode>() {

public int compare(ListNode n1, ListNode n2) {

return n1.val - n2.val;

}

});

for (ListNode n : lists) {

if (n != null) {

pq.offer(n);

}

}

ListNode dummy = new ListNode(-1);

ListNode cur = dummy;

while (!pq.isEmpty()) {

ListNode node = pq.poll();

cur.next = node;

cur = cur.next;

if (cur.next != null) {

pq.offer(cur.next);

}

}

return dummy.next;

}

}

Python **code**:

# Definition for singly-linked list.

# class ListNode(object):

# def \_\_init\_\_(self, x):

# self.val = x

# self.next = None

class Solution(object):

def mergeKLists(self, lists):

"""

:type lists: List[ListNode]

:rtype: ListNode

"""

queue = []

for head in lists:

if head:

heapq.heappush(queue, [head.val, head])

dummy = ListNode(-1)

cur = dummy

while queue:

tup = heapq.heappop(queue)

node = tup[1]

cur.next = node

cur = cur.next

if node.next is not None:

heapq.heappush(queue, [node.next.val, node.next])

return dummy.next

### 056-Merge Intervals

Given a collection of intervals, merge all overlapping intervals.

Example 1:

Input: [[1,3],[2,6],[8,10],[15,18]]

Output: [[1,6],[8,10],[15,18]]

Explanation: Since intervals [1,3] and [2,6] overlaps, merge them into [1,6].

Example 2:

Input: [[1,4],[4,5]]

Output: [[1,5]]

Explanation: Intervals [1,4] and [4,5] are considerred overlapping.

Solution:

# Definition for an interval.

# class Interval(object):

# def \_\_init\_\_(self, s=0, e=0):

# self.start = s

# self.end = e

class Solution(object):

def merge(self, intervals):

"""

:type intervals: List[Interval]

:rtype: List[Interval]

"""

res = []

if not intervals:

return res

n = len(intervals)

intervals.sort(key=lambda x : x.start)

res.append(intervals[0])

for i in range(1, n):

prev = res[-1]

cur = intervals[i]

if cur.start > prev.end:

res.append(cur)

else:

s = min(prev.start, cur.start)

e = max(prev.end, cur.end)

res.pop()

res.append(Interval(s, e))

return res

### 252-Meeting Rooms

Given an array of meeting time intervals consisting of start and end times  
[[s1,e1],[s2,e2],...] (si < ei), determine if a person could attend all meetings.

Example 1:

Input: [[0,30],[5,10],[15,20]]

Output: false

Example 2:

Input: [[7,10],[2,4]]

Output: true

Solution:

# Definition for an interval.

# class Interval(object):

# def \_\_init\_\_(self, s=0, e=0):

# self.start = s

# self.end = e

class Solution(object):

def canAttendMeetings(self, intervals):

"""

:type intervals: List[Interval]

:rtype: bool

"""

n = len(intervals)

if n < 2:

return True

intervals.sort(key = lambda x : x.start)

for i in range(1, n):

prev = intervals[i - 1]

cur = intervals[i]

if prev.end > cur.start:

return False

return True

### 057-Insert Interval

Given a set of non-overlapping intervals, insert a new interval  
into the intervals (merge if necessary).

You may assume that the intervals were initially sorted according  
to their start times.

Example 1:

Input: intervals = [[1,3],[6,9]], newInterval = [2,5]

Output: [[1,5],[6,9]]

Example 2:

Input: intervals = [[1,2],[3,5],[6,7],[8,10],[12,16]], newInterval = [4,8]

Output: [[1,2],[3,10],[12,16]]

Explanation: Because the new interval [4,8] overlaps with [3,5],[6,7],[8,10].

Solution:

# Definition for an interval.

# class Interval(object):

# def \_\_init\_\_(self, s=0, e=0):

# self.start = s

# self.end = e

class Solution(object):

def insert(self, intervals, newInterval):

"""

:type intervals: List[Interval]

:type newInterval: Interval

:rtype: List[Interval]

"""

res = []

n = len(intervals)

for i in range(n):

curInterval = intervals[i]

if curInterval.end < newInterval.start:

res.append(curInterval)

elif curInterval.start > newInterval.end:

res.append(newInterval)

newInterval = curInterval

else:

s = min(curInterval.start, newInterval.start)

e = max(curInterval.end, newInterval.end)

newInterval = Interval(s, e)

res.append(newInterval)

return res

### 075-Sort Colors

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.

Example:

Input: [2,0,2,1,1,0]

Output: [0,0,1,1,2,2]

Solution:

class Solution(object):

def sortColors(self, nums):

"""

:type nums: List[int]

:rtype: void Do not return anything, modify nums in-place instead.

"""

l, r = 0, len(nums) - 1

idx = 0

while idx <= r :

if nums[idx] == 0:

nums[idx], nums[l] = nums[l], nums[idx]

l += 1

idx += 1

elif nums[idx] == 2:

nums[idx], nums[r] = nums[r], nums[idx]

r -= 1

else:

idx += 1

### 088-Merge Sorted Array

Given two sorted integer arrays nums1 and nums2, merge nums2 into nums1 as one sorted array.

Note:

The number of elements initialized in nums1 and nums2 are m and n respectively.  
You may assume that nums1 has enough space (size that is greater or equal to m + n)  
to hold additional elements from nums2.  
Example:

Input:

nums1 = [1,2,3,0,0,0], m = 3

nums2 = [2,5,6], n = 3

Output: [1,2,2,3,5,6]

Solution:

class Solution(object):

def merge(self, nums1, m, nums2, n):

"""

:type nums1: List[int]

:type m: int

:type nums2: List[int]

:type n: int

:rtype: void Do not return anything, modify nums1 in-place instead.

"""

i = m - 1

j = n - 1

idx = m + n - 1

while i >= 0 and j >= 0:

if nums1[i] < nums2[j]:

nums1[idx] = nums2[j]

j -= 1

else:

nums1[idx] = nums1[i]

i -= 1

idx -= 1

while j >= 0:

nums1[idx] = nums2[j]

j -= 1

idx -= 1

### 163-Missing Ranges

Given a sorted integer array nums, where the range of elements are in the inclusive range [lower, upper],  
return its missing ranges.

Example:

Input: nums = [0, 1, 3, 50, 75], lower = 0 and upper = 99,

Output: ["2", "4->49", "51->74", "76->99"]

Solution:

class Solution(object):

def findMissingRanges(self, nums, lower, upper):

"""

:type nums: List[int]

:type lower: int

:type upper: int

:rtype: List[str]

"""

if not nums:

return [self.makeRange(lower, upper)]

res = []

if lower < nums[0]:

res.append(self.makeRange(lower, nums[0] - 1))

n = len(nums)

for i in range(1, n):

if nums[i] - nums[i - 1] > 1 :

res.append(self.makeRange(nums[i - 1] + 1, nums[i] - 1))

if upper > nums[-1] :

res.append(self.makeRange(nums[-1] + 1, upper))

return res

def makeRange(self, a, b):

if a == b :

return str(a)

else:

return str(a) + "->" + str(b)

### 228-Summary Ranges

Given a sorted integer array without duplicates, return the summary of its ranges.

Example 1:

Input: [0,1,2,4,5,7]

Output: ["0->2","4->5","7"]

Explanation: 0,1,2 form a continuous range; 4,5 form a continuous range.

Example 2:

Input: [0,2,3,4,6,8,9]

Output: ["0","2->4","6","8->9"]

Explanation: 2,3,4 form a continuous range; 8,9 form a continuous range.

Solution:

class Solution(object):

def summaryRanges(self, nums):

"""

:type nums: List[int]

:rtype: List[str]

"""

res = []

if not nums:

return res

n = len(nums)

start = end = nums[0]

for i in range(1, n):

if nums[i] - end > 1:

res.append(self.makeRange(start, end))

start = end = nums[i]

else:

end = nums[i]

res.append(self.makeRange(start, end))

return res

def makeRange(self, a, b):

if a == b:

return str(a)

else:

return str(a) + "->" + str(b)

### 189-Rotate Array

Given an array, rotate the array to the right by k steps, where k is non-negative.

Example 1:

Input: [1,2,3,4,5,6,7] and k = 3

Output: [5,6,7,1,2,3,4]

Explanation:

rotate 1 steps to the right: [7,1,2,3,4,5,6]

rotate 2 steps to the right: [6,7,1,2,3,4,5]

rotate 3 steps to the right: [5,6,7,1,2,3,4]

Example 2:

Input: [-1,-100,3,99] and k = 2

Output: [3,99,-1,-100]

Explanation:

rotate 1 steps to the right: [99,-1,-100,3]

rotate 2 steps to the right: [3,99,-1,-100]

Solution:

class Solution(object):

def rotate(self, nums, k):

"""

:type nums: List[int]

:type k: int

:rtype: void Do not return anything, modify nums in-place instead.

"""

n = len(nums)

k = k % n

if k == 0:

return

self.reverse(nums, 0, n - 1)

self.reverse(nums, 0, k - 1)

self.reverse(nums, k, n - 1)

def reverse(self, nums, i, j):

while i < j:

nums[i], nums[j] = nums[j], nums[i]

i += 1

j -= 1

### 215-Kth Largest Element in an Array

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.

Example 1:

Input: [3,2,1,5,6,4] and k = 2

Output: 5

Example 2:

Input: [3,2,3,1,2,4,5,5,6] and k = 4

Output: 4

PS: 面苹果的时候死在了这道题目上, 刷过原题直接给出了最优解, 之后对方follow up要求我写暴力解! WTF, 没想过...

Solution:

解法一: 暴力, 复杂度O(n \* k^2)

class Solution(object):

def findKthLargest(self, nums, k):

"""

:type nums: List[int]

:type k: int

:rtype: int

"""

n = len(nums)

if n == 0:

return 0

klarges = [-(1 << 31)] \* k

klarges[k - 1] = nums[0]

for i in range(1, n):

cur = nums[i]

if cur <= klarges[0]:

continue

if cur >= klarges[-1]:

klarges.pop(0)

klarges.append(cur)

else:

for j in range(1, k):

if klarges[j - 1] <= cur < klarges[j]:

klarges.insert(j, cur)

klarges.pop(0)

return klarges[0]

解法二: 优先队列优化, 复杂度 O(n \* logn)

import heapq

class Solution(object):

def findKthLargest(self, nums, k):

h = []

n = len(nums)

for x in nums:

heapq.heappush(h, x)

res = 0

for i in range(n):

value = heapq.heappop(h)

if i == n - k:

res = value

break

return res

解法三: 自己进行排序, 快排或者归并排序平均复杂度为O(nlogn)

class Solution(object):

def findKthLargest(self, nums, k):

n = len(nums)

self.sort(nums, 0, n - 1)

return nums[n - k]

def sort(self, nums, left, right):

if left >= right:

return

mid = left + (right - left) / 2

self.sort(nums, left, mid)

self.sort(nums, mid + 1, right)

self.merge(nums, left, mid, right)

# The merge process is the same as LeeCode 88

def merge(self, nums, left, mid, right):

tmp = nums[mid + 1 : right + 1]

idx = right

i = mid

j = len(tmp) - 1

while i >= left and j >= 0:

if nums[i] < tmp[j]:

nums[idx] = tmp[j]

j -= 1

else:

nums[idx] = nums[i]

i -= 1

idx -= 1

while j >= 0:

nums[idx] = tmp[j]

j -= 1

idx -= 1

最优解: 利用快速排序中的partition方法, 复杂度 O(logn)

class Solution(object):

def findKthLargest(self, nums, k):

"""

:type nums: List[int]

:type k: int

:rtype: int

"""

n = len(nums)

k = n - k

l, r = 0, n - 1

while l < r:

p = self.partition(nums, l, r)

if p == k:

break

elif p < k:

l = p + 1

else:

r = p - 1

return nums[k]

def partition(self, nums, left, right):

pivot = nums[left]

n = len(nums)

l, r = left, right

while l <= r:

while l <= r and nums[l] <= pivot:

l += 1

while l <= r and nums[r] >= pivot:

r -= 1

if l > r:

break

nums[l], nums[r] = nums[r], nums[l]

nums[left], nums[r] = nums[r], nums[left]

return r

### 414-Third Maximum Number

Given a non-empty array of integers, return the third maximum number in this array.  
If it does not exist, return the maximum number. The time complexity must be in O(n).

Example 1:

Input: [3, 2, 1]

Output: 1

Explanation: The third maximum is 1.

Example 2:

Input: [1, 2]

Output: 2

Explanation: The third maximum does not exist, so the maximum (2) is returned instead.

Example 3:

Input: [2, 2, 3, 1]

Output: 1

Explanation: Note that the third maximum here means the third maximum distinct number.

Both numbers with value 2 are both considered as second maximum.

Solution:

class Solution(object):

def thirdMax(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

n = len(nums)

a = b = c = None

cnt = 0

for i in range(n):

if nums[i] == a or nums[i] == b or nums[i] == c:

continue

if not c or nums[i] > c:

a = b

b = c

c = nums[i]

cnt += 1

elif not b or b < nums[i] < c:

a = b

b = nums[i]

cnt += 1

elif not a or a < nums[i] < b:

a = nums[i]

cnt += 1

else:

continue

return a if cnt > 2 else c

### 280-Wiggle Sort

Given an unsorted array nums, reorder it in-place such that  
nums[0] <= nums[1] >= nums[2] <= nums[3]....

Example:

Input: nums = [3,5,2,1,6,4]

Output: One possible answer is [3,5,1,6,2,4]

Solution:

class Solution(object):

def wiggleSort(self, nums):

"""

:type nums: List[int]

:rtype: void Do not return anything, modify nums in-place instead.

"""

n = len(nums)

for i in range(n - 1):

if i % 2 == 0:

if nums[i] > nums[i + 1]:

self.swap(nums, i, i + 1)

else:

if nums[i] < nums[i + 1]:

self.swap(nums, i, i + 1)

def swap(self, nums, i, j):

nums[i], nums[j] = nums[j], nums[i]

### 334-Increasing Triplet Subsequence

Given an unsorted array return whether an increasing subsequence of length 3 exists or not in the array.

Formally the function should:

Return true if there exists i, j, k

such that arr[i] < arr[j] < arr[k] given 0 ≤ i < j < k ≤ n-1 else return false.

Note: Your algorithm should run in O(n) time complexity and O(1) space complexity.

Example 1:

Input: [1,2,3,4,5]

Output: true

Example 2:

Input: [5,4,3,2,1]

Output: false

Solution

class Solution(object):

def increasingTriplet(self, nums):

"""

:type nums: List[int]

:rtype: bool

"""

n = len(nums)

if n < 3:

return False

a = b = (1 << 31)

for i in range(n):

if nums[i] <= a:

a = nums[i]

elif a < nums[i] <= b:

b = nums[i]

else:

return True

return False

# LeetCode总结三: 查找表相关(Map, Set, Array记忆化查找)

**查找表 (Map, Set, Array)**

**454-4Sum II**

Given four lists A, B, C, D of integer values, compute how many tuples (i, j, k, l) t  
here are such that A[i] + B[j] + C[k] + D[l] is zero.

To make problem a bit easier, all A, B, C, D have same length of N where 0 ≤ N ≤ 500.  
All integers are in the range of -228 to 228 - 1 and the result is guaranteed to be at most 231 - 1.

Example:

Input:

A = [ 1, 2]

B = [-2,-1]

C = [-1, 2]

D = [ 0, 2]

Output:

2

Explanation:

The two tuples are:

1. (0, 0, 0, 1) -> A[0] + B[0] + C[0] + D[1] = 1 + (-2) + (-1) + 2 = 0

2. (1, 1, 0, 0) -> A[1] + B[1] + C[0] + D[0] = 2 + (-1) + (-1) + 0 = 0

Solution:

class Solution(object):

def fourSumCount(self, A, B, C, D):

"""

:type A: List[int]

:type B: List[int]

:type C: List[int]

:type D: List[int]

:rtype: int

"""

dic = {}

lenA = len(A)

lenB = len(B)

lenC = len(C)

lenD = len(D)

cnt = 0

for i in range(lenA):

for j in range(lenB):

twoSum = A[i] + B[j]

if twoSum in dic:

dic[twoSum] += 1

else:

dic[twoSum] = 1

for i in range(lenC):

for j in range(lenD):

twoSum = C[i] + D[j]

if 0 - twoSum in dic:

cnt += dic[-twoSum]

return cnt

**128-Longest Consecutive Sequence**

Given an unsorted array of integers, find the length of the longest consecutive elements sequence.  
Your algorithm should run in O(n) complexity.

Example:

Input: [100, 4, 200, 1, 3, 2]

Output: 4

Explanation: The longest consecutive elements sequence is [1, 2, 3, 4].

Therefore its length is 4.

Solution:

class Solution(object):

def longestConsecutive(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

myset = set()

for n in nums:

myset.add(n)

maxlen = 0

for n in nums:

curlen = 0

k = n

while k in myset:

myset.remove(k)

k -= 1

curlen += 1

k = n + 1

while k in myset:

myset.remove(k)

k += 1

curlen += 1

maxlen = max(curlen, maxlen)

return maxlen

**217-Contain Duplicate**

Given an array of integers, find if the array contains any duplicates.  
Your function should return true if any value appears at least twice in the array,  
and it should return false if every element is distinct.

Example 1:

Input: [1,2,3,1]

Output: true

Example 2:

Input: [1,2,3,4]

Output: false

Example 3:

Input: [1,1,1,3,3,4,3,2,4,2]

Output: true

Solution:

class Solution(object):

def containsDuplicate(self, nums):

"""

:type nums: List[int]

:rtype: bool

"""

myset = set()

for n in nums:

if n in myset:

return True

myset.add(n)

return False

**219-Contains Duplicate II**

Given an array of integers and an integer k, find out whether there are two distinct  
indices i and j in the array such that nums[i] = nums[j] and the absolute difference  
between i and j is at most k.

Example 1:

Input: nums = [1,2,3,1], k = 3

Output: true

Example 2:

Input: nums = [1,0,1,1], k = 1

Output: true

Example 3:

Input: nums = [1,2,3,1,2,3], k = 2

Output: false

Solution:

class Solution(object):

def containsNearbyDuplicate(self, nums, k):

"""

:type nums: List[int]

:type k: int

:rtype: bool

"""

dic = {}

for i, n in enumerate(nums):

if n in dic:

if abs(i - dic[n]) <= k:

return True

dic[n] = i

return False

**220-Contains Duplicate III**

Given an array of integers, find out whether there are two distinct indices i and j in the array  
such that the absolute difference between nums[i] and nums[j] is at most t and the absolute  
difference between i and j is at most k.

Example 1:

Input: nums = [1,2,3,1], k = 3, t = 0

Output: true

Example 2:

Input: nums = [1,0,1,1], k = 1, t = 2

Output: true

Example 3:

Input: nums = [1,5,9,1,5,9], k = 2, t = 3

Output: false

Solution 1: brute force (Time Limit Exceeded)

class Solution(object):

def containsNearbyAlmostDuplicate(self, nums, k, t):

"""

:type nums: List[int]

:type k: int

:type t: int

:rtype: bool

"""

n = len(nums)

for i in range(n):

for j in range(i + 1, n):

if abs(nums[i] - nums[j]) <= t and abs(i - j) <= k:

return True

return False

Solution: use tree set

|a - b| <= t

means:

-t <= (a - b ) <= t

means:

b - t <= a <= b + t

In this problme,

we need to find nums[i] and nums[j] within a window of k size that satisfy :

|nums[i] - nums[j]| <= t

so that

nums[i] - t <= nums[j] <= nums[i] + t

A TreeSet data structure in Java allow us to implement this, the ceiling(E e) API of TreeSet

can return the least element, in the set, that is grater than the given element. That is:

if we have

Integer a = TreeSet.ceiling(nums[i] - t)

Then a satisfy a >= nums[i] - t

Jave **Code**:

public class Solution {

public boolean containsNearbyAlmostDuplicate(int[] nums, int k, int t) {

/\*

The ceiling(E e) method is used to return the least element in this set

greater than or equal to the given element, or null if there is no such element.

\*/

TreeSet<Long> treeSet = new TreeSet<>();

for(int i = 0 ; i < nums.length ; i ++){

Long leastGraterThan = treeSet.ceiling((long)nums[i] - (long)t);

if(leastGraterThan != null && leastGraterThan <= (long)nums[i] + (long)t) {

return true;

}

treeSet.add((long)nums[i]);

// This maintain a sliding window with sieze <= k

if(treeSet.size() == k + 1)

treeSet.remove((long)nums[i-k]);

}

return false;

}

}

**246-Strobogrammatic Number**

A strobogrammatic number is a number that looks the same when rotated 180 degrees (looked at upside down).  
Write a function to determine if a number is strobogrammatic. The number is represented as a string.

Example 1:

Input: "69"

Output: true

Example 2:

Input: "88"

Output: true

Example 3:

Input: "962"

Output: false

Solution:

class Solution(object):

def isStrobogrammatic(self, num):

"""

:type num: str

:rtype: bool

"""

dic = {'6': '9', '8': '8', '9': '6', '0': '0', '1': '1'}

l, r = 0, len(num) - 1

while l <= r:

left = num[l]

right = num[r]

if left not in dic or right not in dic or dic[left] != right:

return False

l += 1

r -= 1

return True

**249-Group Shifted Strings**

Given a string, we can "shift" each of its letter to its successive letter,  
for example: "abc" -> "bcd". We can keep "shifting" which forms the sequence:  
"abc" -> "bcd" -> ... -> "xyz"  
Given a list of strings which contains only lowercase alphabets,  
group all strings that belong to the same shifting sequence.

Example:

Input: ["abc", "bcd", "acef", "xyz", "az", "ba", "a", "z"],

Output:

[

["abc","bcd","xyz"],

["az","ba"],

["acef"],

["a","z"]

]

Solution:

class Solution(object):

def groupStrings(self, strings):

"""

:type strings: List[str]

:rtype: List[List[str]]

"""

dic = {}

res = []

for s in strings:

key = self.encode(s)

if key in dic:

dic[key].append(s)

else:

dic[key] = [s]

for k in dic:

res.append(dic[k])

return res

def encode(self, s):

n = len(s)

key = ''

for i in range(1, n):

diff = (ord(s[i]) - ord(s[i - 1])) % 26

key += str(diff)

return key

**266-Palindrome Permutation**

Given a string, determine if a permutation of the string could form a palindrome.

Example 1:

Input: "**code**"

Output: false

Example 2:

Input: "aab"

Output: true

Example 3:

Input: "carerac"

Output: true

Solution:

class Solution(object):

def canPermutePalindrome(self, s):

"""

:type s: str

:rtype: bool

"""

dic = {}

cnt = 0

for char in s:

if char in dic:

dic[char] += 1

else:

dic[char] = 1

for char in dic:

if dic[char] % 2 != 0:

cnt += 1

return cnt < 2

**290-Word Pattern**

Given a pattern and a string str, find if str follows the same pattern.  
Here follow means a full match, such that there is a bijection between a  
letter in pattern and a non-empty word in str.

Example 1:

Input: pattern = "abba", str = "dog cat cat dog"

Output: true

Example 2:

Input:pattern = "abba", str = "dog cat cat fish"

Output: false

Example 3:

Input: pattern = "aaaa", str = "dog cat cat dog"

Output: false

Example 4:

Input: pattern = "abba", str = "dog dog dog dog"

Output: false

Solution:

class Solution(object):

def wordPattern(self, pattern, str):

"""

:type pattern: str

:type str: str

:rtype: bool

"""

words = str.split(" ")

n = len(words)

dicA = {}

dicB = {}

if n != len(pattern):

return False

for i in range(n):

char = pattern[i]

word = words[i]

if char in dicA:

if dicA[char] != word:

return False

else:

dicA[char] = word

if word in dicB:

if dicB[word] != char:

return False

else:

dicB[word] = char

return True

**299-Bulls and Cows**

You are playing the following Bulls and Cows game with your friend:  
You write down a number and ask your friend to guess what the number is.  
Each time your friend makes a guess, you provide a hint that indicates  
how many digits in said guess match your secret number exactly in both  
digit and position (called "bulls") and how many digits match the secret  
number but locate in the wrong position (called "cows"). Your friend  
will use successive guesses and hints to eventually derive the secret number.  
Write a function to return a hint according to the secret number and friend's guess,  
use A to indicate the bulls and B to indicate the cows.

Please note that both secret number and friend's guess may contain duplicate digits.

Example 1:

Input: secret = "1807", guess = "7810"

Output: "1A3B"

Explanation: 1 bull and 3 cows. The bull is 8, the cows are 0, 1 and 7.

Example 2:

Input: secret = "1123", guess = "0111"

Output: "1A1B"

Explanation: The 1st 1 in friend's guess is a bull, the 2nd or 3rd 1 is a cow.

Note: You may assume that the secret number and your friend's guess only  
contain digits, and their lengths are always equal.

Solution:

class Solution(object):

def getHint(self, secret, guess):

"""

:type secret: str

:type guess: str

:rtype: str

"""

nums = [0] \* 10

n = len(secret)

A = B = 0

for i in range(n):

s = int(secret[i])

g = int(guess[i])

if s == g:

A += 1

else:

if nums[s] < 0:

B += 1

if nums[g] > 0:

B += 1

nums[s] += 1

nums[g] -= 1

return str(A) + 'A' + str(B) + 'B'

**311-Sparse Matrix Multiplication**

Given two sparse matrices A and B, return the result of AB.

You may assume that A's column number is equal to B's row number.

Example:

Input:

A = [

[ 1, 0, 0],

[-1, 0, 3]

]

B = [

[ 7, 0, 0 ],

[ 0, 0, 0 ],

[ 0, 0, 1 ]

]

Output:

| 1 0 0 | | 7 0 0 | | 7 0 0 |

AB = | -1 0 3 | x | 0 0 0 | = | -7 0 3 |

| 0 0 1 |

Solution: 这道题可以用hashmap来做, 由于稀疏矩阵有大量的行或者列全部是非零元素, 因此用map记录那些有非零元素的行和列,  
然后只计算这些包含非零元素的行和列. 这样做代码比较麻烦, 另一种方法是先把暴力求解的代码写出来, 然后做出小的修改.

直接暴力求矩阵相乘的代码:

class Solution(object):

def multiply(self, A, B):

"""

:type A: List[List[int]]

:type B: List[List[int]]

:rtype: List[List[int]]

"""

rowA = len(A)

colA = len(A[0])

colB = len(B[0])

C = [[0 for j in range(colB)] for i in range(rowA)]

for i in range(rowA):

for j in range(colB):

for k in range(colA):

C[i][j] += A[i][k] \* B[k][j]

return C

上面代码在leetcode上提交会超时. 对于系数矩阵, 可以对上面代码做小修改, 如下:

class Solution(object):

def multiply(self, A, B):

"""

:type A: List[List[int]]

:type B: List[List[int]]

:rtype: List[List[int]]

"""

rowA = len(A)

colA = len(A[0])

colB = len(B[0])

C = [[0 for j in range(colB)] for i in range(rowA)]

for i in range(rowA):

for k in range(colA):

if A[i][k] != 0:

for j in range(colB):

if B[k][j] != 0:

C[i][j] += A[i][k] \* B[k][j]

return C

利用hashmap的解法:

class Solution(object):

def multiply(self, A, B):

"""

:type A: List[List[int]]

:type B: List[List[int]]

:rtype: List[List[int]]

"""

dicA = {}

dicB = {}

rowA = len(A)

colA = len(A[0])

rowB = len(B)

colB = len(B[0])

for i in range(rowA) :

for j in range(colA) :

if A[i][j] != 0 :

dicA[i] = A[i]

break

for j in range(colB):

tmp = []

nonZero = False

for i in range(rowB):

tmp.append(B[i][j])

if B[i][j] != 0 :

nonZero = True

if nonZero:

dicB[j] = tmp

res = [[0 for j in range(colB)] for i in range(rowA)]

for r in dicA :

for c in dicB :

res[r][c] = self.dotProduct(dicA[r], dicB[c])

return res

def dotProduct(self, lst1, lst2) :

res = 0

for i in range(len(lst1)) :

res += lst1[i] \* lst2[i]

return res

**325-Maximum Size Subarray Sum Equals k**

Given an array nums and a target value k, find the maximum length of a subarray that sums to k.  
If there isn't one, return 0 instead.

Note:  
The sum of the entire nums array is guaranteed to fit within the 32-bit signed integer range.

Example 1:

Input: nums = [1, -1, 5, -2, 3], k = 3

Output: 4

Explanation: The subarray [1, -1, 5, -2] sums to 3 and is the longest.

Example 2:

Input: nums = [-2, -1, 2, 1], k = 1

Output: 2

Explanation: The subarray [-1, 2] sums to 1 and is the longest.

Follow Up:  
Can you do it in O(n) time?

class Solution(object):

def maxSubArrayLen(self, nums, k):

"""

:type nums: List[int]

:type k: int

:rtype: int

"""

dic = {0: -1}

summ = 0

maxlen = 0

n = len(nums)

for i in range(n):

summ += nums[i]

if summ not in dic:

dic[summ] = i

if summ - k in dic:

maxlen = max(maxlen, i - dic[summ - k])

return maxlen

**349-Intersection of Two Arrays**

Given two arrays, write a function to compute their intersection.

Example 1:

Input: nums1 = [1,2,2,1], nums2 = [2,2]

Output: [2]

Example 2:

Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4]

Output: [9,4]

Note:  
Each element in the result must be unique.  
The result can be in any order.

Solution:

class Solution(object):

def intersection(self, nums1, nums2):

"""

:type nums1: List[int]

:type nums2: List[int]

:rtype: List[int]

"""

res = []

myset = set()

for n in nums1:

myset.add(n)

for n in nums2:

if n in myset:

res.append(n)

myset.remove(n)

return res

**383-Ransom Note**

Given an arbitrary ransom note string and another string containing letters from all the magazines,  
write a function that will return true if the ransom note can be constructed from the magazines;  
otherwise, it will return false.  
Each letter in the magazine string can only be used once in your ransom note.

Note:  
You may assume that both strings contain only lowercase letters.

canConstruct("a", "b") -> false

canConstruct("aa", "ab") -> false

canConstruct("aa", "aab") -> true

Solution:

class Solution(object):

def canConstruct(self, ransomNote, magazine):

"""

:type ransomNote: str

:type magazine: str

:rtype: bool

"""

counts = [0] \* 26

for char in magazine:

counts[ord(char) - ord('a')] += 1

for char in ransomNote:

if counts[ord(char) - ord('a')] == 0:

return False

counts[ord(char) - ord('a')] -= 1

return True

**387-First Unique Character in a String**

Given a string, find the first non-repeating character in it and return it's index.  
If it doesn't exist, return -1.

Examples:

s = "leetcode"

return 0.

s = "loveleetcode",

return 2.

Note: You may assume the string contain only lowercase letters.

class Solution(object):

def firstUniqChar(self, s):

"""

:type s: str

:rtype: int

"""

counts = [0] \* 26

n = len(s)

res = -1

for char in s:

counts[ord(char) - ord('a')] += 1

for i in range(n):

c = s[i]

if counts[ord(c) - ord('a')] == 1:

res = i

break

return res

**447-Number of Boomerangs**

Given n points in the plane that are all pairwise distinct, a "boomerang" is a tuple of points (i, j, k)  
such that the distance between i and j equals the distance between i and k (the order of the tuple matters).

Find the number of boomerangs. You may assume that n will be at most 500 and coordinates of points are  
all in the range [-10000, 10000] (inclusive).

Example:

Input:

[[0,0],[1,0],[2,0]]

Output:

2

Explanation:

The two boomerangs are [[1,0],[0,0],[2,0]] and [[1,0],[2,0],[0,0]]

Solution: 制作一个查找表, 对于某一个点, 记录所有其他点到改点的距离, 如果有两个点以上到这个点的距离相同, 则可以组合成两个以上的 Boomerangs.  
例如对于点a, 其余个点中有个3点: b, c, d到它的距离相等, 那么能组成 Permutation(2, 3) = 6个 Boomerangs, 分别为:  
abc, abd, acd, acb, adb, adc. 推广开去, 如果有 n 个点到该点距离相等, 结果为 Permutation(2, n) = n \* (n - 1) 个

class Solution(object):

def numberOfBoomerangs(self, points):

"""

:type points: List[List[int]]

:rtype: int

"""

dic = {}

n = len(points)

res = 0

for i in range(n):

pi = points[i]

dic = {}

for j in range(n):

if i == j:

continue

pj = points[j]

distance = self.dist(pi, pj)

if distance in dic:

dic[distance] += 1

else:

dic[distance] = 1

for k in dic:

if dic[k] > 1:

res += dic[k] \* (dic[k] - 1)

return res

def dist(self, p1, p2):

x1, y1 = p1[0], p1[1]

x2, y2 = p2[0], p2[1]

return (x1 - x2) \*\* 2 + (y1 - y2) \*\* 2

**523-Continuous Subarray Sum**

Given a list of non-negative numbers and a target integer k, write a function to check if the array has a continuous  
subarray of size at least 2 that sums up to the multiple of k, that is, sums up to n \* k where n is also an integer.

Example 1:

Input: [23, 2, 4, 6, 7], k=6

Output: True

Explanation: Because [2, 4] is a continuous subarray of size 2 and sums up to 6.

Example 2:

Input: [23, 2, 6, 4, 7], k=6

Output: True

Explanation: Because [23, 2, 6, 4, 7] is an continuous subarray of size 5 and sums up to 42.

Note:  
The length of the array won't exceed 10,000.  
You may assume the sum of all the numbers is in the range of a signed 32-bit integer.

class Solution(object):

def checkSubarraySum(self, nums, k):

"""

:type nums: List[int]

:type k: int

:rtype: bool

"""

dic = {0: -1}

n = len(nums)

sum = 0

for i in range(n):

sum += nums[i]

if k != 0:

sum %= k

if sum in dic:

if i - dic[sum] >= 2:

return True

else:

dic[sum] = i

return False

**525-Contiguous Array**

Given a binary array, find the maximum length of a contiguous subarray with equal number of 0 and 1.

Example 1:

Input: [0,1]

Output: 2

Explanation: [0, 1] is the longest contiguous subarray with equal number of 0 and 1.

Example 2:

Input: [0,1,0]

Output: 2

Explanation: [0, 1] (or [1, 0]) is a longest contiguous subarray with equal number of 0 and 1.

Note: The length of the given binary array will not exceed 50,000.

Solution:

class Solution(object):

def findMaxLength(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

n = len(nums)

if n == 0:

return 0

summ = [0] \* n

summ[0] = 1 if nums[0] == 1 else -1

for i in range(1, n):

if nums[i] == 0:

summ[i] = summ[i - 1] - 1

else:

summ[i] = summ[i - 1] + 1

dic = {0: -1}

res = 0

for i, x in enumerate(summ):

if x in dic:

res = max(res, i - dic[x])

else:

dic[x] = i

return res

**560-Subarray Sum Equals K**

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

Example 1:

Input:nums = [1,1,1], k = 2

Output: 2

Note:  
The length of the array is in range [1, 20,000].  
The range of numbers in the array is [-1000, 1000] and the range of the integer k is [-1e7, 1e7].

Solution:

class Solution(object):

def subarraySum(self, nums, k):

"""

:type nums: List[int]

:type k: int

:rtype: int

"""

cursum = 0

n = len(nums)

dic = {0: 1}

res = 0

for i, x in enumerate(nums):

cursum += x

if cursum - k in dic:

res += dic[cursum - k]

if cursum in dic:

dic[cursum] += 1

else:

dic[cursum] = 1

return res

**001-2sum (skip)**

# LeetCode总结四: LinkedList

01**9**-Remove Nth Node From End of List  
024-Swap Node in Pairs  
025-Reverse Nodes in k-Group  
061-Rotate List  
082-Remove Duplicates from Sorted List II  
083-Remove Duplicates from Sorted List  
0**9**2-Reverse LinkedList II  
138-Copy List with Random Pointer  
141-Linked List Cycle  
142-Linked List Cycle II  
143-Reorder List  
146-LRU Cache  
160-Intersection of Two Linked Lists  
203-Remove Linked List Elements  
206-Reverse Linked List  
234-Palindrome Linked List  
237-Delete Node in a Linked list  
287-Find the Duplicate Number  
328-Odd Even Linklist List

### 019-Remove Nth Node From End of List

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

Example:

Given linked list: 1->2->3->4->5, and n = 2.

After removing the second node from the end, the linked list becomes 1->2->3->5.

Solution:

# 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

"""

dummy = ListNode(0)

dummy.next = head

slow = fast = dummy

for i in range(n):

fast = fast.next

while fast.next:

fast = fast.next

slow = slow.next

slow.next = slow.next.next

return dummy.next

### 024-Swap Node in Pairs

Given a linked list, swap every two adjacent nodes and return its head.

Example:

Given 1->2->3->4, you should return the list as 2->1->4->3.

Solution:

# Definition for singly-linked list.

class ListNode(object):

def \_\_init\_\_(self, x):

self.val = x

self.next = None

class Solution(object):

def swapPairs(self, head):

"""

:type head: ListNode

:rtype: ListNode

"""

dummy = ListNode(0)

dummy.next = head

cur = head

prev = dummy

while cur and cur.next:

post = cur.next

cur.next = post.next

prev.next = post

post.next = cur

cur = cur.next

prev = post.next

return dummy.next

def test(self):

node1 = ListNode(1)

cur = node1

for i in range(2, 3):

cur.next = ListNode(i)

cur = cur.next

newNode = self.swapPairs(node1)

self.printList(newNode)

def printList(self, head):

res = ''

cur = head

while cur:

res += str(cur.val) + ' -> '

cur = cur.next

res += 'NULL'

print res

test = Solution()

test.test()

### 025-Reverse Nodes in k-Group

Given a linked list, reverse the nodes of a linked list k at a time and return its modified list.  
k is a positive integer and is less than or equal to the length of the linked list.  
If the number of nodes is not a multiple of k then left-out nodes in the end should remain as it is.

Example:

Given this linked list: 1->2->3->4->5

For k = 2, you should return: 2->1->4->3->5

For k = 3, you should return: 3->2->1->4->5

class Solution(object):

def reverseKGroup(self, head, k):

"""

:type head: ListNode

:type k: int

:rtype: ListNode

"""

if not head or not head.next or k < 2:

return head

dummy = ListNode(0)

dummy.next = head

begin = dummy

idx = 0

while head:

idx += 1

if idx % k == 0:

begin = self.reverse(begin, head.next)

head = begin.next

else:

head = head.next

return dummy.next

def reverse(self, begin, end):

cur = begin.next

prev = begin

post = cur.next

while post and post != end:

cur.next = post.next

post.next = prev.next

prev.next = post

post = cur.next

return cur

### 206-Reverse Linked List

Reverse a singly linked list.

Example:

Input: 1->2->3->4->5->NULL

Output: 5->4->3->2->1->NULL

Follow up:  
A linked list can be reversed either iteratively or recursively.  
Could you implement both?

class Solution(object):

def reverseList(self, head):

"""

:type head: ListNode

:rtype: ListNode

"""

dummy = ListNode(0)

dummy.next = head

cur = head

post = cur.next

while post is not None:

cur.next = post.next

post.next = dummy.next

dummy.next = post

post = cur.next

return dummy.next

递归解:

class Solution(object):

def reverseList(self, head):

"""

:type head: ListNode

:rtype: ListNode

"""

return self.reverseHelper(head, None)

def reverseHelper(self, head, newHead):

if not head:

return newHead

nextHead = head.next

head.next = newHead

return self.reverseHelper(nextHead, head)

### 083-Remove Duplicates from Sorted List

Given a sorted linked list, delete all duplicates such that each element appear only once.

Example 1:

Input: 1->1->2

Output: 1->2

Example 2:

Input: 1->1->2->3->3

Output: 1->2->3

Solution:

class Solution(object):

def deleteDuplicates(self, head):

"""

:type head: ListNode

:rtype: ListNode

"""

cur = head

while cur and cur.next:

if cur.val == cur.next.val:

cur.next = cur.next.next

else:

cur = cur.next

return head

### 082-Remove Duplicates from Sorted List II

Given a sorted linked list, delete all nodes that have duplicate numbers,  
leaving only distinct numbers from the original list.

Example 1:

Input: 1->2->3->3->4->4->5

Output: 1->2->5

Example 2:

Input: 1->1->1->2->3

Output: 2->3

Solution:

class Solution(object):

def deleteDuplicates(self, head):

"""

:type head: ListNode

:rtype: ListNode

"""

dummy = ListNode(0)

dummy.next = head

prev = dummy

cur = head

while cur and cur.next:

if cur.val == cur.next.val:

while cur and cur.next and cur.val == cur.next.val:

cur = cur.next

cur = cur.next

prev.next = cur

else:

cur = cur.next

prev = prev.next

return dummy.next

### 061-Rotate List

Given a linked list, rotate the list to the right by k places, where k is non-negative.

Example 1:

Input: 1->2->3->4->5->NULL, k = 2

Output: 4->5->1->2->3->NULL

Explanation:

rotate 1 steps to the right: 5->1->2->3->4->NULL

rotate 2 steps to the right: 4->5->1->2->3->NULL

Example 2:

Input: 0->1->2->NULL, k = 4

Output: 2->0->1->NULL

Explanation:

rotate 1 steps to the right: 2->0->1->NULL

rotate 2 steps to the right: 1->2->0->NULL

rotate 3 steps to the right: 0->1->2->NULL

rotate 4 steps to the right: 2->0->1->NULL

Solution:

class Solution(object):

def rotateRight(self, head, k):

"""

:type head: ListNode

:type k: int

:rtype: ListNode

"""

if not head:

return None

dummy = ListNode(0)

dummy.next = head

fast = slow = head

size = 0

while fast:

fast = fast.next

size += 1

fast = head

k %= size

for i in range(k):

fast = fast.next

while fast.next:

fast = fast.next

slow = slow.next

fast.next = dummy.next

dummy.next = slow.next

slow.next = None

return dummy.next

### 092-Reverse LinkedList II

Reverse a linked list from position m to n. Do it in one-pass.

Note: 1 ≤ m ≤ n ≤ length of list.

Example:

Input: 1->2->3->4->5->NULL, m = 2, n = 4

Output: 1->4->3->2->5->NULL

Solution:

class Solution(object):

def reverseBetween(self, head, m, n):

"""

:type head: ListNode

:type m: int

:type n: int

:rtype: ListNode

"""

dummy = ListNode(0)

dummy.next = head

fast = head

prev = slow = dummy

for i in range(n):

fast = fast.next

for i in range(m):

if i > 0:

prev = prev.next

slow = slow.next

post = slow.next

while post != fast:

slow.next = post.next

post.next = prev.next

prev.next = post

post = slow.next

return dummy.next

### 138-Copy List with Random Pointer

A linked list is given such that each node contains an additional  
random pointer which could point to any node in the list or null.  
Return a deep copy of the list.

Solution:

class Solution(object):

def copyRandomList(self, head):

"""

:type head: RandomListNode

:rtype: RandomListNode

"""

dic = {}

cur = head

while cur:

copy = RandomListNode(cur.label)

dic[cur] = copy

cur = cur.next

cur = head

while cur:

dic[cur].next = dic.get(cur.next)

dic[cur].random = dic.get(cur.random)

cur = cur.next

return dic.get(head)

### 141-Linked List Cycle

Given a linked list, determine if it has a cycle in it.

Follow up:  
Can you solve it without using extra space?

Solution:

class Solution(object):

def hasCycle(self, head):

"""

:type head: ListNode

:rtype: bool

"""

fast = slow = head

while fast and fast.next:

fast = fast.next.next

slow = slow.next

if fast == slow:

return True

return False

### 142-Linked List Cycle II

Given a linked list, return the node where the cycle begins. If there is no cycle, return null.

Note: Do not modify the linked list.

Follow up:  
Can you solve it without using extra space?

Solution:

class Solution(object):

def detectCycle(self, head):

"""

:type head: ListNode

:rtype: ListNode

"""

fast = slow = head

while fast and fast.next:

fast = fast.next.next

slow = slow.next

if slow == fast:

fast = head

while fast != slow:

fast = fast.next

slow = slow.next

return slow

return None

### 143-Reorder List

Given a singly linked list L: L0 -> L1 -> ... -> Ln-1 -> Ln  
reorder it to: L0 -> Ln -> L1 -> Ln-1 -> L2 -> Ln-2 -> ...

You may not modify the values in the list's nodes, only nodes itself may be changed.

Example 1:

Given 1->2->3->4, reorder it to 1->4->2->3.

Example 2:

Given 1->2->3->4->5, reorder it to 1->5->2->4->3.

Solution:

class Solution(object):

def reorderList(self, head):

if not head or not head.next:

return

p1 = p2 = head

while p2.next and p2.next.next:

p1 = p1.next

p2 = p2.next.next

tmp\_head = p1.next

p1.next = None

new\_tmp\_head = self.reverse(tmp\_head)

p1.next = new\_tmp\_head

middle = p1

p2 = middle.next

p1 = head

while p1 != middle:

middle.next = p2.next

p2.next = p1.next

p1.next = p2

p1 = p2.next

p2 = middle.next

def reverse(self, head):

dummy = ListNode(0)

dummy.next = head

cur = head

post = head.next

while post:

cur.next = post.next

post.next = dummy.next

dummy.next = post

post = cur.next

return dummy.next

### 160-Intersection of Two Linked Lists

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

a1 - a2

|

b1 - b2 - c1 - c2 - c3

Notes:

If the two linked lists have no intersection at all, return null.  
The linked lists must retain their original structure after the function returns.  
You may assume there are no cycles anywhere in the entire linked structure.  
Your **code** should preferably run in O(n) time and use only O(1) memory.

Solution:

class Solution(object):

def getIntersectionNode(self, headA, headB):

"""

:type head1, head1: ListNode

:rtype: ListNode

"""

lenA = lenB = 0

cur = headA

while cur:

cur = cur.next

lenA += 1

cur = headB

while cur:

cur = cur.next

lenB += 1

curA = headA

curB = headB

if lenA > lenB:

for i in range(lenA - lenB):

curA = curA.next

else:

for i in range(lenB - lenA):

curB = curB.next

while curA and curB:

if curA == curB:

return curA

curA = curA.next

curB = curB.next

return None

### 237-Delete Node in a Linked list

Write a function to delete a node (except the tail) in a singly linked list, given only access to that node.

Given linked list -- head = [4,5,1,**9**], which looks like following:

4 -> 5 -> 1 -> **9**

Example 1:

Input: head = [4,5,1,**9**], node = 5

Output: [4,1,**9**]

Explanation: You are given the second node with value 5, the linked list

should become 4 -> 1 -> **9** after calling your function.

Example 2:

Input: head = [4,5,1,**9**], node = 1

Output: [4,5,**9**]

Explanation: You are given the third node with value 1, the linked list

should become 4 -> 5 -> **9** after calling your function.

Note:

The linked list will have at least two elements.  
All of the nodes' values will be unique.  
The given node will not be the tail and it will always be a valid node of the linked list.  
Do not return anything from your function.

class Solution(object):

def deleteNode(self, node):

"""

:type node: ListNode

:rtype: void Do not return anything, modify node in-place instead.

"""

node.val = node.next.val

node.next = node.next.next

### 146-LRU Cache

Design and implement a data structire for Least Recently Used (LRU) cache. it should support  
the follwing operations: get and put.

get(key) Get the value (will always be positive) of the key if the key exists in the cache, otherwise return -1  
put(key, value) Set or insert the value if the key is not already present. When the cache reached its capacity,  
it should invalidate the least recently used item before inserting a new item

Follow up: Could you do both operations in O(1) time complexity ?

Example:

LRUCache cache = new LRUCache( 2 ); // 2 is capacity

cache.put(1, 1);

cache.put(2, 2);

cache.get(1); // returns 1

cache.put(3, 3); // evicts key 2

cache.get(2); // returns -1 (not found)

cache.put(4, 4); // evicts key 1

cache.get(1); // returns -1 (not found)

cache.get(3); // returns 3

cache.get(4); // returns 4

Solution: 用一个双向链表和 HashMap 可以实现全 O(1) 复杂度, 如果用单向链表则两种操作都需要O(n)时间复杂度  
利用HashMap储存cache, 每次get或者put都很**简**单第从 HashMap 中找到缓存的值. 利用双向链表记录每个被访问的顺序,  
Least Recent Used 的缓存放在链表尾部, 最近被访问的缓存放在链表头部, 每次链表长度到达上线, 删除尾部的缓存元素即可  
同时也要删除掉存储在 HashMap 中相应缓存 (的引用 / 指针)

class LRUCache(object):

def \_\_init\_\_(self, capacity):

"""

:type capacity: int

"""

self.capacity = capacity

self.cache = {}

self.dlist = DoubleLinkedList()

def get(self, key):

"""

:type key: int

:rtype: int

"""

if key not in self.cache:

return -1

node = self.cache[key]

self.dlist.set\_head(node)

return node.val

def put(self, key, value):

"""

:type key: int

:type value: int

:rtype: void

"""

if key in self.cache:

node = self.cache[key]

node.val = value

self.dlist.set\_head(node)

else:

if self.capacity == self.dlist.size:

tailNode = self.dlist.tail

del self.cache[tailNode.key]

self.dlist.remove(tailNode)

node = ListNode(key, value)

self.dlist.insert(node)

self.cache[key] = node

# 以下是双向链表的实现

class ListNode(object):

def \_\_init\_\_(self, key, val):

self.key = key

self.val = val

self.prev = None

self.next = None

class DoubleLinkedList(object):

def \_\_init\_\_(self):

self.head = None

self.tail = None

self.size = 0

def insert(self, node):

# insert a node to the head

node.next = self.head

if self.head is not None:

self.head.prev = node

else:

self.tail = node

self.head = node

self.size += 1

def set\_head(self, node):

# Move a node to the head

prev\_node = node.prev

post\_node = node.next

if prev\_node is not None:

prev\_node.next = post\_node

if post\_node is not None:

post\_node.prev = prev\_node

else:

self.tail = prev\_node

node.next = self.head

self.head.prev = node

node.prev = None

self.head = node

def remove(self, node):

prev\_node = node.prev

post\_node = node.next

if prev\_node is not None:

prev\_node.next = post\_node

else:

self.head = post\_node

if post\_node is not None:

post\_node.prev = prev\_node

else:

self.tail = prev\_node

self.size -= 1

# Your LRUCache object will be instantiated and called as such:

# obj = LRUCache(capacity)

# param\_1 = obj.get(key)

# obj.put(key,value)

解法二: 上面的双向链表属于比较的详细实现. 其实对于这道题, 双向链表中的remove方法可以**简**单第变成 removeTail , 因为这道题只需要**简**单但删除掉为节点.

class LRUCache(object):

def \_\_init\_\_(self, capacity):

"""

:type capacity: int

"""

self.capacity = capacity

self.cache = {}

self.dlist = DoubleLinkedList()

def get(self, key):

"""

:type key: int

:rtype: int

"""

if key not in self.cache:

return -1

node = self.cache[key]

self.dlist.set\_head(node)

return node.val

def put(self, key, value):

"""

:type key: int

:type value: int

:rtype: void

"""

if key in self.cache:

node = self.cache[key]

node.val = value

self.dlist.set\_head(node)

else:

if self.capacity == self.dlist.size:

tailNode = self.dlist.tail

del self.cache[tailNode.key]

self.dlist.removeTail()

node = ListNode(key, value)

self.dlist.insert(node)

self.cache[key] = node

class ListNode(object):

def \_\_init\_\_(self, key, val):

self.key = key

self.val = val

self.prev = None

self.next = None

class DoubleLinkedList(object):

def \_\_init\_\_(self):

self.head = None

self.tail = None

self.size = 0

def insert(self, node):

# insert a node to the head

node.next = self.head

if self.head is None:

self.tail = node

else:

self.head.prev = node

self.head = node

self.size += 1

def set\_head(self, node):

# Move a node to the head

prevNode = node.prev

postNode = node.next

if prevNode:

prevNode.next = postNode

if postNode:

postNode.prev = prevNode

else:

self.tail = prevNode

node.next = self.head

self.head.prev = node

node.prev = None

self.head = node

def removeTail(self):

prev = self.tail.prev

if prev:

prev.next = None

self.tail.prev = None

self.tail = prev

self.size -= 1

### 203-Remove Linked List Elements

Remove all elements from a linked list of integers that have value val.

Example:

Input: 1->2->6->3->4->5->6, val = 6

Output: 1->2->3->4->5

Solution:

class Solution(object):

def removeElements(self, head, val):

"""

:type head: ListNode

:type val: int

:rtype: ListNode

"""

dummy = ListNode(0)

dummy.next = head

prev = dummy

cur = head

while cur:

if cur.val == val:

prev.next = cur.next

else:

prev = prev.next

cur = cur.next

return dummy.next

### 234-Palindrome Linked List

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

Example 1:

Input: 1->2

Output: false

Example 2:

Input: 1->2->2->1

Output: true

Follow up:  
Could you do it in O(n) time and O(1) space?

Solution: **简**单的解法是利用数组把所有节点的值记录下来, 再遍历整个数组判断回文性质  
Follow up可以翻转后半段的链表再进行判断

class Solution(object):

def isPalindrome(self, head):

"""

:type head: ListNode

:rtype: bool

"""

if not head:

return True

fast = slow = head

if not fast or not fast.next:

return True

if not fast.next.next:

return fast.val == fast.next.val

while fast.next and fast.next.next:

fast = fast.next.next

slow = slow.next

cur = slow.next

post = slow.next.next

while post:

cur.next = post.next

post.next = slow.next

slow.next = post

post = cur.next

fast = slow.next

slow = head

while fast:

if fast.val != slow.val:

return False

fast = fast.next

slow = slow.next

return True

### 287-Find the Duplicate Number

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 one duplicate  
number, find the duplicate one.

Example 1:

Input: [1,3,4,2,2]

Output: 2

Example 2:

Input: [3,1,3,4,2]

Output: 3

Solution: 利用链表的定义. 把数组的index想象着地址, 数组的元素为指向的下一个地址. 这样如果链表有Cycle,  
那么必然有连个元素指向同一个地址. 这样问题就变成了在一条有cycle链表中找出循环的起点.

class Solution(object):

def findDuplicate(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

slow = nums[0]

fast = nums[nums[0]]

while fast != slow:

fast = nums[nums[fast]]

slow = nums[slow]

fast = 0

while fast != slow:

fast = nums[fast]

slow = nums[slow]

return fast

### 328-Odd Even Linklist List

Given a singly linked list, group all odd nodes together followed by the even nodes.  
Please note here we are talking about the node number and not the value in the nodes.

You should try to do it in place. The program should run in O(1) space complexity  
and O(nodes) time complexity.

Example 1:

Input: 1->2->3->4->5->NULL

Output: 1->3->5->2->4->NULL

Example 2:

Input: 2->1->3->5->6->4->7->NULL

Output: 2->3->6->7->1->5->4->NULL

Solution:

class Solution(object):

def oddEvenList(self, head):

"""

:type head: ListNode

:rtype: ListNode

"""

if not head:

return None

p1 = head

p2 = head.next

if not p2:

return head

while p2 and p2.next:

p3 = p2.next

p2.next = p3.next

p3.next = p1.next

p1.next = p3

p1 = p3

p2 = p2.next

return head

# LeetCode总结五: 栈, 队列 & 优先队列

[](https://www.jianshu.com/u/e34517ea5dc1)

**[安全地带](https://www.jianshu.com/u/e34517ea5dc1)** 关注

2018.09.20 12:11 字数 1411 阅读 2评论 0喜欢 0

### 栈, 队列, 优先队列

020-Valid Parentheses  
071-Simplify Path  
150-Evaluate Reverse Polish Notation  
032-Longest Valid Parentheses  
084-Largest Rectangle in Histogram  
224-Basic Calculator  
227-Basic Calculator II  
394-Decode String  
716-Max Stack  
157-Read N Characters Given Read4  
158-Read N Characters Given Read4 II-Call multiple times  
346-Moving Average from Data Stream  
347-Top K Frequent Elements  
621-Task Scheduler

### 020-Valid Parentheses

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.  
Example 4:

Input: "([)]"

Output: false

Example 5:

Input: "{[]}"

Output: true

Solution:

class Solution(object):

def isValid(self, s):

"""

:type s: str

:rtype: bool

"""

dic = {'{': '}', '[': ']', '(': ')'}

stack = []

for c in s:

if c == "{" or c == "[" or c == "(":

stack.append(c)

else:

if not stack:

return False

left = stack.pop()

if dic[left] != c:

return False

return True if len(stack) == 0 else False

### 071-Simplify Path

Given an absolute path for a file (Unix-style), simplify it.

For example,

path = "/home/", => "/home"

path = "/a/./b/../../c/", => "/c"

path = "/a/../../b/../c//.//", => "/c"

path = "/a//b////c/d//././/..", => "/a/b/c"

In a UNIX-style file system, a period ('.') refers to the current directory, so it can be ignored  
in a simplified path. Additionally, a double period ("..") moves up a directory, so it cancels  
out whatever the last directory was.

Corner Cases:

Did you consider the case where path = "/../"?  
In this case, you should return "/".  
Another corner case is the path might contain multiple slashes '/' together, such as "/home//foo/".  
In this case, you should ignore redundant slashes and return "/home/foo".

class Solution(object):

def simplifyPath(self, path):

"""

:type path: str

:rtype: str

"""

dirnames = path.split("/")

res = ""

stack = []

for d in dirnames:

if d != "" and d != "." and d != "..":

stack.append(d)

else:

if stack and d == "..":

stack.pop()

while stack:

res = "/" + stack.pop() + res

return res if res != "" else "/"

### 150-Evaluate Reverse Polish Notation

Evaluate the value of an arithmetic expression in Reverse Polish Notation.

Valid operators are +, -, \* , /. Each operand may be an integer or another expression.

Note:

Division between two integers should truncate toward zero.  
The given RPN expression is always valid. That means the expression would  
always evaluate to a result and there won't be any divide by zero operation.

Solution:

class Solution(object):

def evalRPN(self, tokens):

"""

:type tokens: List[str]

:rtype: int

"""

stack = []

for i, x in enumerate(tokens):

if x == "\*" or x == '-' or x == "+" or x == "/":

a = stack.pop()

b = stack.pop()

if x == "+":

stack.append(b + a)

elif x == "-":

stack.append(b - a)

elif x == "\*":

stack.append(b \* a)

else:

stack.append(self.divide(b, a))

else:

stack.append(eval(x))

return stack.pop()

def divide(self, n1, n2):

if n1 == 0:

return 0

sign = (n1 \* n2) / abs(n1 \* n2)

n1 = abs(n1)

n2 = abs(n2)

res = n1 / n2

return sign \* res

### 032-Longest Valid Parentheses

Given a string containing just the characters '(' and ')', find the length  
of the longest valid (well-formed) parentheses substring.

Example 1:

Input: "(()"

Output: 2

Explanation: The longest valid parentheses substring is "()"

Example 2:

Input: ")()())"

Output: 4

Explanation: The longest valid parentheses substring is "()()"

Solution:

class Solution(object):

def longestValidParentheses(self, s):

"""

:type s: str

:rtype: int

"""

stack = []

n = len(s)

res = 0

for i, char in enumerate(s):

if char == '(':

stack.append(i)

elif stack and s[stack[-1]] == '(':

stack.pop()

else:

stack.append(i)

if not stack:

return len(s)

l, r = 0, len(s) - 1

while stack:

l = stack.pop()

res = max(res, r - l)

r = l - 1

res = max(res, r + 1)

return res

### 084-Largest Rectangle in Histogram

Given n non-negative integers representing the histogram's bar height  
where the width of each bar is 1, find the area of largest rectangle in the histogram.

Example:

Input: [2,1,5,6,2,3]

Output: 10

Solution:

class Solution(object):

def largestRectangleArea(self, heights):

"""

:type heights: List[int]

:rtype: int

"""

res = 0

heights.append(0)

stack = []

n = len(heights)

for i in range(n):

while stack and heights[i] < heights[stack[-1]]:

h = heights[stack.pop()]

idx = -1

if stack:

idx = stack[-1]

res = max(res, h \* (i - 1 - idx))

stack.append(i)

return res

### 224-Basic Calculator

Implement a basic calculator to evaluate a simple expression string.

The expression string may contain open ( and closing parentheses ), the plus + or minus sign -, non-negative integers and empty spaces .

Example 1:

Input: "1 + 1"

Output: 2

Example 2:

Input: " 2-1 + 2 "

Output: 3

Example 3:

Input: "(1+(4+5+2)-3)+(6+8)"

Output: 23

Solution:

class Solution(object):

def calculate(self, s):

"""

:type s: str

:rtype: int

"""

n = len(s)

res = 0

sign = 1

stack = []

i = 0

while i < n:

if s[i].isdigit():

cursum = eval(s[i])

while i + 1 < n and s[i + 1].isdigit():

cursum = cursum \* 10 + eval(s[i + 1])

i += 1

res = res + sign \* cursum

elif s[i] == '+':

sign = 1

elif s[i] == '-':

sign = -1

elif s[i] == '(':

stack.append(res)

stack.append(sign)

res = 0

sign = 1

elif s[i] == ')':

res = res \* stack.pop() + stack.pop()

else:

pass

i += 1

return res

### 227-Basic Calculator II

Implement a basic calculator to evaluate a simple expression string.

The expression string contains only non-negative integers, +, -, \* , / operators and empty spaces.  
The integer division should truncate toward zero.

Example 1:

Input: "3 + 2 \* 2"

Output: 7

Example 2:

Input: " 3/2 "

Output: 1

Example 3:

Input: " 3 + 5 / 2 "

Output: 5

Solution:

class Solution(object):

def calculate(self, s):

"""

:type s: str

:rtype: int

"""

stack = []

n = len(s)

res = 0

sign = '+'

num = 0

for i in range(n):

if s[i].isdigit():

num = num \* 10 + int(s[i])

if (not s[i].isdigit() and s[i] != " ") or i == n - 1:

if sign == "+":

stack.append(num)

elif sign == "-":

stack.append(-num)

elif sign == "\*":

stack.append(stack.pop() \* num)

else:

a = stack.pop()

b = 0

if a < 0:

b = (-1) \* (abs(a) / num)

else:

b = a / num

stack.append(b)

sign = s[i]

num = 0

return sum(stack)

### 394-Decode String

Given an encoded string, return it's decoded string.  
The encoding rule is: k[encoded\_string], where the encoded\_string inside the square brackets  
is being repeated exactly k times. Note that k is guaranteed to be a positive integer.  
You may assume that the input string is always valid; No extra white spaces, square brackets  
are well-formed, etc.  
Furthermore, you may assume that the original data does not contain any digits and that digits  
are only for those repeat numbers, k. For example, there won't be input like 3a or 2[4].

Examples:

s = "3[a]2[bc]", return "aaabcbc".

s = "3[a2[c]]", return "accaccacc".

s = "2[abc]3[cd]ef", return "abcabccdcdcdef".

Solution:

class Solution(object):

def decodeString(self, s):

"""

:type s: str

:rtype: str

"""

# strs = list(s)

stack = []

res = ''

for i, c in enumerate(s):

if c == ']':

tmp = ''

while stack and not stack[-1].isdigit():

char = stack.pop()

if char.isalpha():

tmp = char + tmp

num = ''

while stack and stack[-1].isdigit():

num = stack.pop() + num

cur = ''

for i in range(int(num)):

cur += tmp

stack.append(cur)

else:

stack.append(c)

while stack:

res = stack.pop() + res

return res

def test(self):

s1 = "3[a]2[bc]" #return "aaabcbc".

s2 = "3[a2[c]]" #return "accaccacc".

s3 = "2[abc]3[cd]ef" # return "abcabccdcdcdef".

s4 = "10[leetcode]"

print self.decodeString(s4)

### 716-Max Stack

Design a max stack that supports push, pop, top, peekMax and popMax.

push(x) -- Push element x onto stack.

pop() -- Remove the element on top of the stack and return it.

top() -- Get the element on the top.

peekMax() -- Retrieve the maximum element in the stack.

popMax() -- Retrieve the maximum element in the stack, and remove it.

If you find more than one maximum elements, only remove the top-most one.

class MaxStack(object):

def \_\_init\_\_(self):

"""

initialize your data structure here.

"""

self.maxStack = []

self.dataStack = []

def push(self, x):

"""

:type x: int

:rtype: void

"""

self.dataStack.append(x)

if self.maxStack:

if self.maxStack[-1] < x:

self.maxStack.append(x)

else:

self.maxStack.append(self.maxStack[-1])

else:

self.maxStack.append(x)

def pop(self):

"""

:rtype: int

"""

res = self.dataStack.pop()

self.maxStack.pop()

return res

def top(self):

"""

:rtype: int

"""

return self.dataStack[-1]

def peekMax(self):

"""

:rtype: int

"""

return self.maxStack[-1]

def popMax(self):

"""

:rtype: int

"""

tmp = []

while self.dataStack[-1] != self.maxStack[-1]:

tmp.append(self.dataStack.pop())

self.maxStack.pop()

res = self.dataStack.pop()

self.maxStack.pop()

while tmp:

self.push(tmp.pop())

return res

def test():

maxstack = MaxStack()

nums = [1, 6, 4, 5, 2]

for n in nums:

maxstack.push(n)

print maxstack.peekMax()

print maxstack.popMax()

print maxstack.peekMax()

print maxstack.top()

### 157-Read N Characters Given Read4

The API: int read4(char \* buf) reads 4 characters at a time from a file.  
The return value is the actual number of characters read.  
For example, it returns 3 if there is only 3 characters left in the file.

By using the read4 API, implement the function int read(char \* buf, int n)  
that reads n characters from the file.

Solution: 这道题的题例有问题.   
  
\* buf is destination buffer, so when ever we call read4(buf), 4 consecutive characters from the file are copied into the \* buf array.  
Solution is to initialize a temporary buff tmpbuf = ["", "", "", ""] with size of 4.  
Then continuously invoke read4 against tmpbuf : read4(tmpbuf). So that every time we do this,  
we fill the tmpbuf with 4 character  
For example, the first time you invoke read4(tmpbuf), results in the tmpbuf = ['a', 'b', 'c', 'd'].  
Then we move these character into the destination buffer buf.  
Repeat this until the read4(tmpbuf) we reach the end of the file, in which case the tmpbuf wont be filled out.  
It may end up with ['h', 'i', '', ''] or ['x', '', '', '']

# The read4 API is already defined for you.

# @param buf, a list of characters

# @return an integer

# def read4(buf):

class Solution(object):

def read(self, buf, n):

"""

:type buf: Destination buffer (List[str])

:type n: Maximum number of characters to read (int)

:rtype: The number of characters read (int)

"""

eof = False

cur\_num\_of\_char = 0 # current number of characters we have copied from the file into the buffer

while not eof and cur\_num\_of\_char < n:

tmp = ["" for i in range(4)]

cnt = read4(tmp)

if cnt < 4:

eof = True

number\_of\_characters\_left = n - cur\_num\_of\_char

cnt = min(cnt, number\_of\_characters\_left)

for i in range(cnt):

buf[cur\_num\_of\_char] = tmp[i]

cur\_num\_of\_char += 1

return cur\_num\_of\_char

### 158-Read N Characters Given Read4 II-Call multiple times

The API: int read4(char \* buf) reads 4 characters at a time from a file.  
The return value is the actual number of characters read. For example, it returns 3 if there is only 3 characters left in the file.  
By using the read4 API, implement the function int read(char \* buf, int n) that reads n characters from the file.

Note:  
The read function may be called multiple times.

Example 1:

Given buf = "abc"

read("abc", 1) // returns "a"

read("abc", 2); // returns "bc"

read("abc", 1); // returns ""

Example 2:

Given buf = "abc"

read("abc", 4) // returns "abc"

read("abc", 1); // returns ""

Solution:

# The read4 API is already defined for you.

# @param buf, a list of characters

# @return an integer

# def read4(buf):

class Solution(object):

def \_\_init\_\_(self):

self.queue = []

def read(self, buf, n):

"""

:type buf: Destination buffer (List[str])

:type n: Maximum number of characters to read (int)

:rtype: The number of characters read (int)

"""

cur\_num\_of\_chars = 0

while True:

tmp = ['' for i in range(4)]

read4(tmp)

for i in range(4):

self.queue.append(tmp[i])

num\_of\_chars\_left = n - cur\_num\_of\_chars

if not self.queue or num\_of\_chars\_left == 0:

break

length = min(num\_of\_chars\_left, len(self.queue))

for i in range(length):

buf[cur\_num\_of\_chars] = self.queue.pop(0)

cur\_num\_of\_chars += 1

return cur\_num\_of\_chars

### 346-Moving Average from Data Stream

Given a stream of integers and a window size,  
calculate the moving average of all integers in the sliding window.

Example:

MovingAverage m = new MovingAverage(3);

m.next(1) = 1

m.next(10) = (1 + 10) / 2

m.next(3) = (1 + 10 + 3) / 3

m.next(5) = (10 + 3 + 5) / 3

Solution:

class MovingAverage(object):

def \_\_init\_\_(self, size):

"""

Initialize your data structure here.

:type size: int

"""

self.queue = []

self.curval = 0

self.size = size

def next(self, val):

"""

:type val: int

:rtype: float

"""

if len(self.queue) == self.size:

self.curval -= self.queue.pop(0)

self.queue.append(val)

self.curval += val

if len(self.queue) < self.size:

return float(self.curval) / len(self.queue)

else:

return float(self.curval) / self.size

### 347-Top K Frequent Elements

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

Example 1:

Input: nums = [1,1,1,2,2,3], k = 2

Output: [1,2]

Example 2:

Input: nums = [1], k = 1

Output: [1]

Note:  
You may assume k is always valid, 1 ≤ k ≤ number of unique elements.  
Your algorithm's time complexity must be better than O(n log n), where n is the array's size.

import heapq

class FreqObj(object):

def \_\_init\_\_(self, n, freq):

self.n = n

self.freq = freq

def \_\_cmp\_\_(self, other):

return cmp(other.freq, self.freq)

class Solution(object):

def topKFrequent(self, nums, k):

"""

:type nums: List[int]

:type k: int

:rtype: List[int]

"""

pq = []

res = []

freq = {}

for n in nums:

if n in freq:

freq[n] += 1

else:

freq[n] = 1

for num in freq:

heapq.heappush(pq, FreqObj(num, freq[num]))

for i in range(k):

element = heapq.heappop(pq)

res.append(element.n)

return res

### 388-Longest Absolute File Path

Suppose we abstract our file system by a string in the following manner:

The string "dir\n\tsubdir1\n\tsubdir2\n\t\tfile.ext" represents:

dir

subdir1

subdir2

file.ext

The directory dir contains an empty sub-directory subdir1 and a sub-directory subdir2 containing a file file.ext.

The string "dir\n\tsubdir1\n\t\tfile1.ext\n\t\tsubsubdir1\n\tsubdir2\n\t\tsubsubdir2\n\t\t\tfile2.ext" represents:

dir

subdir1

file1.ext

subsubdir1

subdir2

subsubdir2

file2.ext

The directory dir contains two sub-directories subdir1 and subdir2. subdir1 contains a file file1.ext  
and an empty second-level sub-directory subsubdir1. subdir2 contains a second-level sub-directory subsubdir2  
containing a file file2.ext.

We are interested in finding the longest (number of characters) absolute path to a file within our file system.  
For example, in the second example above, the longest absolute path is "dir/subdir2/subsubdir2/file2.ext",  
and its length is 32 (not including the double quotes).

Given a string representing the file system in the above format, return the length of the longest absolute  
path to file in the abstracted file system. If there is no file in the system, return 0.

Note:  
The name of a file contains at least a . and an extension.  
The name of a directory or sub-directory will not contain a ..  
Time complexity required: O(n) where n is the size of the input string.

Notice that a/aa/aaa/file1.txt is not the longest file path, if there is another path aaaaaaaaaaaaaaaaaaaaa/sth.png.

Solution:

class Solution(object):

def lengthLongestPath(self, input):

"""

:type input: str

:rtype: int

"""

dirs = input.split("\n")

stack = [0]

maxlen = 0

for s in dirs:

dir\_name = s.strip("\t")

indent = len(s) - len(dir\_name)

while indent < len(stack) - 1:

stack.pop() # find the parent dir

length = stack[-1] + len(dir\_name) + 1 # the 1 stands for a slash "/"

stack.append(length)

if "." in dir\_name:

# if is a file, remove the last "/" from the end of the path, we have length - 1

maxlen = max(maxlen, length - 1)

return maxlen

### 621-Task Scheduler

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.

Example 1:

Input: tasks = ["A","A","A","B","B","B"], n = 2

Output: 8

Explanation: A -> B -> idle -> A -> B -> idle -> A -> B.

Note:  
The number of tasks is in the range [1, 10000].  
The integer n is in the range [0, 100].

Solution: 经典任务调度算法

import heapq

from collections import Counter

class Task(object):

def \_\_init\_\_(self, name, freq):

self.name = name

self.freq = freq

def \_\_cmp\_\_(self, task):

return cmp(task.freq, self.freq)

class Solution(object):

def leastInterval(self, tasks, n):

"""

:type tasks: List[str]

:type n: int

:rtype: int

"""

counts = Counter(tasks)

num = 0

queue = []

taskInterVal = []

for t in counts:

task = Task(t, counts[t])

heapq.heappush(queue, task)

while queue:

can = []

k = n + 1

while k > 0 and queue:

task = heapq.heappop(queue)

task.freq -= 1

taskInterVal.append(task.name)

can.append(task)

k -= 1

num += 1

for i in range(k):

taskInterVal.append('idle')

for task in can:

if task.freq > 0:

heapq.heappush(queue, task)

if not queue:

break

num += k

print taskInterVal

return num

def test(self):

tasks = ['A', 'A', 'A', 'B', 'B', 'B']

print self.leastInterval(tasks, 2)

### 358-Rearrange String k Distance Apart

Given a non-empty string s and an integer k, rearrange the string such that the same  
characters are at least distance k from each other.

All input strings are given in lowercase letters. If it is not possible to rearrange  
the string, return an empty string "".

Example 1:

Input: s = "aabbcc", k = 3

Output: "abcabc"

Explanation: The same letters are at least distance 3 from each other.

Example 2:

Input: s = "aaabc", k = 3

Output: ""

Explanation: It is not possible to rearrange the string.

Example 3:

Input: s = "aaadbbcc", k = 2

Output: "abacabcd"

Explanation: The same letters are at least distance 2 from each other.

Solution: 和621题相似, 但是提交的时候有些Corner case非常恶心

import heapq

from collections import Counter

class Char(object):

def \_\_init\_\_(self, c, freq):

self.c = c

self.freq = freq

def \_\_cmp\_\_(self, char):

return cmp(char.freq, self.freq)

class Solution(object):

def rearrangeString(self, s, k):

"""

:type s: str

:type k: int

:rtype: str

"""

counts = Counter(s)

pq = []

candidate = []

waitQueue = []

for c in counts:

char = Char(c, counts[c])

heapq.heappush(pq, char)

while pq:

char = heapq.heappop(pq)

char.freq -= 1

candidate.append(char.c)

waitQueue.append(char)

if len(waitQueue) < k:

continue

headChar = waitQueue.pop(0)

if headChar.freq > 0:

heapq.heappush(pq, headChar)

return ''.join(candidate) if len(candidate) == len(s) else ""

# LeetCode总结六: 二分查找

033-Search in Rotated Sorted Array  
081-Search in Rotated Sorted Array II  
034-Search for a Range  
035-Search Insertion Position  
074-Search a 2D Matrix  
153-Find Minimum in Rotated Sorted Array  
154-Find Minimum in Rotated Sorted Array II  
162-Find Peak Element  
275-H-Index II  
278-First Bad Version  
540-Single Element in a Sorted Array  
744-Find Smallest Letter Greater Than Target

### 033-Search in Rotated Sorted Array

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).

Example 1:

Input: nums = [4,5,6,7,0,1,2], target = 0

Output: 4

Example 2:

Input: nums = [4,5,6,7,0,1,2], target = 3

Output: -1

Solution:

class Solution(object):

def search(self, nums, target):

"""

:type nums: List[int]

:type target: int

:rtype: int

"""

n = len(nums)

if n == 0:

return -1

l, r = 0, n - 1

while l < r:

mid = l + (r - l) / 2

if nums[mid] > nums[r]:

l = mid + 1

else:

r = mid

pivot = l

res = self.binarySearch(nums, 0, pivot - 1, target)

if res != -1:

return res

return self.binarySearch(nums, pivot, n - 1, target)

def binarySearch(self, nums, l, r, target):

while l <= r:

mid = l + (r - l) / 2

if nums[mid] == target:

return mid

elif nums[mid] < target:

l = mid + 1

else:

r = mid - 1

return -1

def test(self):

nums = [4,5,6,7,0,1,2]

print self.search(nums, 0)

### 081-Search in Rotated Sorted Array II

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

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

You are given a target value to search. If found in the array return true, otherwise return false.

Example 1:

Input: nums = [2,5,6,0,0,1,2], target = 0

Output: true

Example 2:

Input: nums = [2,5,6,0,0,1,2], target = 3

Output: false

Solution: 这道题用**二**分查找我没能想明白, 只能用递归的方法做到 O(nlogn)

class Solution(object):

def search(self, nums, target):

"""

:type nums: List[int]

:type target: int

:rtype: bool

"""

if not nums:

return False

l, r = 0, len(nums) - 1

return self.helper(nums, target, l, r)

def helper(self, nums, target, l, r):

if l > r:

return False

if l == r:

return nums[l] == target

mid = l + (r - l) / 2

if nums[mid] == target:

return True

return self.helper(nums, target, l, mid) or self.helper(nums, target, mid + 1, r)

def test(self):

nums = [2,5,6,0,0,1,2]

# nums = [2, 1]

print self.search(nums, 1)

### 034-Find First and Last Position of Element in Sorted Array

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].

Example 1:

Input: nums = [5,7,7,8,8,10], target = 8

Output: [3,4]

Example 2:

Input: nums = [5,7,7,8,8,10], target = 6

Output: [-1,-1]

Solution:

class Solution(object):

def searchRange(self, nums, target):

"""

:type nums: List[int]

:type target: int

:rtype: List[int]

"""

if not nums:

return [-1, -1]

l, r = 0, len(nums) - 1

lowerRange = self.lower(nums, target, l, r)

upperRange = self.upper(nums, target, l, r)

return [lowerRange, upperRange]

def upper(self, nums, target, l, r):

while l < r:

mid = l + (r - l + 1) / 2

if nums[mid] <= target:

l = mid

else:

r = mid - 1

return l if nums[l] == target else -1

def lower(self, nums, target, l, r):

while l < r:

mid = l + (r - l) / 2

if nums[mid] < target:

l = mid + 1

else:

r = mid

return r if nums[r] == target else -1

### 035-Search Insertion Position

Given a sorted array and a target value, return the index if the target is found.  
If not, return the index where it would be if it were inserted in order.

You may assume no duplicates in the array.

Example 1:

Input: [1,3,5,6], 5

Output: 2

Example 2:

Input: [1,3,5,6], 2

Output: 1

Example 3:

Input: [1,3,5,6], 7

Output: 4

Example 4:

Input: [1,3,5,6], 0

Output: 0

Solution:

class Solution(object):

def searchInsert(self, nums, target):

"""

:type nums: List[int]

:type target: int

:rtype: int

"""

l, r = 0, len(nums) - 1

if target > nums[r]:

return r + 1

while l < r:

mid = l + (r - l) / 2

if nums[mid] < target :

l = mid + 1

else:

r = mid

return r

### 074-Search a 2D Matrix

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 from left to right.  
The first integer of each row is greater than the last integer of the previous row.  
Example 1:

Input:

matrix = [

[1, 3, 5, 7],

[10, 11, 16, 20],

[23, 30, 34, 50]

]

target = 3

Output: true

Example 2:

Input:

matrix = [

[1, 3, 5, 7],

[10, 11, 16, 20],

[23, 30, 34, 50]

]

target = 13

Output: false

Solution1:

class Solution(object):

def searchMatrix(self, matrix, target):

"""

:type matrix: List[List[int]]

:type target: int

:rtype: bool

"""

if not matrix or not matrix[0]:

return False

m = len(matrix)

n = len(matrix[0])

l, r = 0, m \* n - 1

while l <= r :

mid = l + (r - l) / 2

val = matrix[mid / n][mid % n]

if val == target:

return True

elif val < target:

l = mid + 1

else:

r = mid - 1

return False

Solution2:

class Solution(object):

def searchMatrix(self, matrix, target):

"""

:type matrix: List[List[int]]

:type target: int

:rtype: bool

"""

if not matrix or not matrix[0]:

return False

row = len(matrix)

col = len(matrix[0])

searchRows = None

for i in range(row):

r = matrix[i]

head, tail = r[0], r[col - 1]

if target == head or target == tail:

return True

if head < target < tail:

searchRows = r

break

elif target < head:

return False

else:

continue

if not searchRows:

return False

i, j = 0, col - 1

while i <= j:

mid = i + (j - i) / 2

if target == searchRows[mid]:

return True

elif target < searchRows[mid]:

j = mid - 1

else:

i = mid + 1

return False

### 153-Find Minimum in Rotated Sorted Array

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]).

Find the minimum element.

You may assume no duplicate exists in the array.

Example 1:

Input: [3,4,5,1,2]

Output: 1

Example 2:

Input: [4,5,6,7,0,1,2]

Output: 0

Solution:

class Solution(object):

def findMin(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

l, r = 0, len(nums) - 1

if nums[l] < nums[r]:

return nums[0]

while l < r:

mid = l + (r - l) / 2

if nums[mid] > nums[r]:

l = mid + 1

else:

r = mid

return min(nums[l], nums[r])

### 154-Find Minimum in Rotated Sorted Array II

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]).

Find the minimum element.

The array may contain duplicates.

Example 1:

Input: [1,3,5]

Output: 1

Example 2:

Input: [2,2,2,0,1]

Output: 0

This is a follow up problem to Find Minimum in Rotated Sorted Array.  
Would allow duplicates affect the run-time complexity? How and why?

Solution:

class Solution(object):

def findMin(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

l, r = 0, len(nums) - 1

if nums[l] < nums[r]:

return nums[l]

while l < r:

mid = l + (r - l) / 2

if nums[mid] > nums[r]:

l = mid + 1

elif nums[mid] < nums[r]:

r = mid

else:

r -= 1

return min(nums[l], nums[r])

### 162-Find Peak Element

A peak element is an element that is greater than its neighbors.

Given an input array nums, where nums[i] ≠ nums[i+1], find a peak element and return its index.

The array may contain multiple peaks, in that case return the index to any one of the peaks is fine.

You may imagine that nums[-1] = nums[n] = -∞.

Example 1:

Input: nums = [1,2,3,1]

Output: 2

Explanation: 3 is a peak element and your function should return the index number 2.

Example 2:

Input: nums = [1,2,1,3,5,6,4]

Output: 1 or 5

Explanation: Your function can return either index number 1 where the peak element is 2,

or index number 5 where the peak element is 6.

Solution:

class Solution(object):

def findPeakElement(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

l, r = 0, len(nums) - 1

while l < r:

mid = l + (r - l) / 2

if nums[mid] < nums[mid + 1]:

l = mid + 1

else:

r = mid

return r

### 275-H-Index II

Given an array of citations sorted in ascending order (each citation is a non-negative integer) of  
a researcher, write a function to compute the researcher's h-index.

According to the definition of h-index on Wikipedia: "A scientist has index h if h of his/her N  
papers have at least h citations each, and the other N − h papers have no more than h citations each."

Example:

Input: citations = [0,1,3,5,6]

Output: 3

Explanation: [0,1,3,5,6] means the researcher has 5 papers in total and each of them had

received 0, 1, 3, 5, 6 citations respectively.

Since the researcher has 3 papers with at least 3 citations each and the remaining

two with no more than 3 citations each, her h-index is 3.

Solution:

class Solution(object):

def hIndex(self, citations):

"""

:type citations: List[int]

:rtype: int

"""

if not citations:

return 0

n = len(citations)

l, r = 0, n - 1

while l <= r:

mid = l + (r - l) / 2

if citations[mid] == n - mid:

return n - mid

elif citations[mid] < n - mid:

l = mid + 1

else:

r = mid - 1

return n - l

### 278-First Bad Version

You are a product manager and currently leading a team to develop a new product.  
Unfortunately, the latest version of your product fails the quality check.  
Since each version is developed based on the previous version,  
all the versions after a bad version are also bad.

Suppose you have n versions [1, 2, ..., n] and you want to find out the first bad one,  
which causes all the following ones to be bad.

You are given an API bool isBadVersion(version) which will return whether version is bad.  
Implement a function to find the first bad version. You should minimize the number of calls to the API.

Example:

Given n = 5, and version = 4 is the first bad version.

call isBadVersion(3) -> false

call isBadVersion(5) -> true

call isBadVersion(4) -> true

Then 4 is the first bad version.

Solution:

# The isBadVersion API is already defined for you.

# @param version, an integer

# @return a bool

# def isBadVersion(version):

class Solution(object):

def firstBadVersion(self, n):

"""

:type n: int

:rtype: int

"""

l = 1

r = n

while l < r:

mid = l + (r - l) / 2

if isBadVersion(mid):

r = mid

else:

l = mid + 1

return l

### 540-Single Element in a Sorted Array

Given a sorted array consisting of only integers where every element appears  
twice except for one element which appears once. Find this single element that appears only once.

Example 1:

Input: [1,1,2,3,3,4,4,8,8]

Output: 2

Example 2:

Input: [3,3,7,7,10,11,11]

Output: 10

Note: Your solution should run in O(log n) time and O(1) space.

Solution:

class Solution(object):

def singleNonDuplicate(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

l, r = 0, len(nums) - 1

while l < r:

mid = l + (r - l) / 2

if mid % 2 == 1:

mid -= 1

if nums[mid] == nums[mid + 1] :

l = mid + 2

else:

r = mid

return nums[l]

### 744-Find Smallest Letter Greater Than Target

Given a list of sorted characters letters containing only lowercase letters,  
and given a target letter target, find the smallest element in the list that is larger than the given target.  
Letters also wrap around. For example, if the target is target = 'z' and letters = ['a', 'b'], the answer is 'a'.

Examples:

Input:

letters = ["c", "f", "j"]

target = "a"

Output: "c"

Input:

letters = ["c", "f", "j"]

target = "c"

Output: "f"

Input:

letters = ["c", "f", "j"]

target = "d"

Output: "f"

Input:

letters = ["c", "f", "j"]

target = "g"

Output: "j"

Input:

letters = ["c", "f", "j"]

target = "j"

Output: "c"

Input:

letters = ["c", "f", "j"]

target = "k"

Output: "c"

Solution:

class Solution(object):

def nextGreatestLetter(self, letters, target):

"""

:type letters: List[str]

:type target: str

:rtype: str

"""

l, r = 0, len(letters) - 1

if ord(target) >= ord(letters[r]):

return letters[0]

while l < r:

mid = l + (r - l) / 2

if ord(target) < ord(letters[mid]):

r = mid

else:

l = mid + 1

return letters[r]

# LeetCode总结七: 设计类题目

**155-Min Stack**

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.  
  Example:

MinStack minStack = new MinStack();

minStack.push(-2);

minStack.push(0);

minStack.push(-3);

minStack.getMin(); --> Returns -3.

minStack.pop();

minStack.top(); --> Returns 0.

minStack.getMin(); --> Returns -2.

Solution:

class MinStack(object):

def \_\_init\_\_(self):

"""

initialize your data structure here.

"""

self.data = []

self.minval = []

def push(self, x):

"""

:type x: int

:rtype: void

"""

self.data.append(x)

if self.minval:

if x < self.minval[-1]:

self.minval.append(x)

else:

self.minval.append(self.minval[-1])

else:

self.minval.append(x)

def pop(self):

"""

:rtype: void

"""

self.data.pop()

self.minval.pop()

def top(self):

"""

:rtype: int

"""

return self.data[-1]

def getMin(self):

"""

:rtype: int

"""

return self.minval[-1]

**716-Max Stack**

Design a max stack that supports push, pop, top, peekMax and popMax.

push(x) -- Push element x onto stack.

pop() -- Remove the element on top of the stack and return it.

top() -- Get the element on the top.

peekMax() -- Retrieve the maximum element in the stack.

popMax() -- Retrieve the maximum element in the stack, and remove it.

If you find more than one maximum elements, only remove the top-most one.

class MaxStack(object):

def \_\_init\_\_(self):

"""

initialize your data structure here.

"""

self.maxStack = []

self.dataStack = []

def push(self, x):

"""

:type x: int

:rtype: void

"""

self.dataStack.append(x)

if self.maxStack:

if self.maxStack[-1] < x:

self.maxStack.append(x)

else:

self.maxStack.append(self.maxStack[-1])

else:

self.maxStack.append(x)

def pop(self):

"""

:rtype: int

"""

res = self.dataStack.pop()

self.maxStack.pop()

return res

def top(self):

"""

:rtype: int

"""

return self.dataStack[-1]

def peekMax(self):

"""

:rtype: int

"""

return self.maxStack[-1]

def popMax(self):

"""

:rtype: int

"""

tmp = []

while self.dataStack[-1] != self.maxStack[-1]:

tmp.append(self.dataStack.pop())

self.maxStack.pop()

res = self.dataStack.pop()

self.maxStack.pop()

while tmp:

self.push(tmp.pop())

return res

def test():

maxstack = MaxStack()

nums = [1, 6, 4, 5, 2]

for n in nums:

maxstack.push(n)

print maxstack.peekMax()

print maxstack.popMax()

print maxstack.peekMax()

print maxstack.top()

**232-Implement Queue using Stacks**

Implement the following operations of a queue using stacks.

push(x) -- Push element x to the back of queue.  
pop() -- Removes the element from in front of queue.  
peek() -- Get the front element.  
empty() -- Return whether the queue is empty.

Example:

MyQueue queue = new MyQueue();

queue.push(1);

queue.push(2);

queue.peek(); // returns 1

queue.pop(); // returns 1

queue.empty(); // returns false

Notes:

You must use only standard operations of a stack -- which means only push to top, peek/pop from top, size, and is empty operations are valid.  
Depending on your language, stack may not be supported natively.  
You may simulate a stack by using a list or deque (double-ended queue),  
as long as you use only standard operations of a stack.  
You may assume that all operations are valid (for example,  
no pop or peek operations will be called on an empty queue).

Solution:

class MyQueue(object):

def \_\_init\_\_(self):

"""

Initialize your data structure here.

"""

self.stackA = []

self.stackB = []

self.front = None

def push(self, x):

"""

Push element x to the back of queue.

:type x: int

:rtype: void

"""

self.stackA.append(x)

if not self.front:

self.front = x

# print self.stackA, self.stackB

def pop(self):

"""

Removes the element from in front of queue and returns that element.

:rtype: int

"""

res = self.front

while self.stackA:

self.stackB.append(self.stackA.pop())

self.stackB.pop()

if not self.stackB:

self.front = None

else:

self.front = self.stackB[-1]

while self.stackB:

self.stackA.append(self.stackB.pop())

return res

def peek(self):

"""

Get the front element.

:rtype: int

"""

return self.front

def empty(self):

"""

Returns whether the queue is empty.

:rtype: bool

"""

return self.front is None

**251-Flatten 2D Vector**

Implement an iterator to flatten a 2d vector.

Example:

Input: 2d vector =

[

[1,2],

[3],

[4,5,6]

]

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

Explanation: By calling next repeatedly until hasNext returns false,

the order of elements returned by next should be: [1,2,3,4,5,6].

Solution:

class Vector2D(object):

def \_\_init\_\_(self, vec2d):

"""

Initialize your data structure here.

:type vec2d: List[List[int]]

"""

self.vec = []

for r in vec2d:

self.vec = self.vec + r

self.idx = 0

self.end = len(self.vec) - 1

def next(self):

"""

:rtype: int

"""

res = self.vec[self.idx]

self.idx += 1

return res

def hasNext(self):

"""

:rtype: bool

"""

return self.idx <= self.end

**341-Flatten Nested List Iterator**

Given a nested list of integers, implement an iterator to flatten it.  
Each element is either an integer, or a list -- whose elements may also be integers or other lists.

Example 1:

Input: [[1,1],2,[1,1]]

Output: [1,1,2,1,1]

Explanation: By calling next repeatedly until hasNext returns false,

the order of elements returned by next should be: [1,1,2,1,1].

Example 2:

Input: [1,[4,[6]]]

Output: [1,4,6]

Explanation: By calling next repeatedly until hasNext returns false,

the order of elements returned by next should be: [1,4,6].

Solution:

# """

# This is the interface that allows for creating nested lists.

# You should not implement it, or speculate about its implementation

# """

#class NestedInteger(object):

# def isInteger(self):

# """

# @return True if this NestedInteger holds a single integer, rather than a nested list.

# :rtype bool

# """

#

# def getInteger(self):

# """

# @return the single integer that this NestedInteger holds, if it holds a single integer

# Return None if this NestedInteger holds a nested list

# :rtype int

# """

#

# def getList(self):

# """

# @return the nested list that this NestedInteger holds, if it holds a nested list

# Return None if this NestedInteger holds a single integer

# :rtype List[NestedInteger]

# """

class NestedIterator(object):

def \_\_init\_\_(self, nestedList):

"""

Initialize your data structure here.

:type nestedList: List[NestedInteger]

"""

self.intlist = self.flattern(nestedList)

self.idx = 0

self.end = len(self.intlist) - 1

def flattern(self, nestedList):

res = []

self.flattern\_helper(nestedList, res)

return res

def flattern\_helper(self, nestedList, res):

n = len(nestedList)

for i in range(n):

nestedElement = nestedList[i]

if nestedElement.getList():

self.flattern\_helper(nestedElement.getList(), res)

else:

if nestedElement.getInteger() is not None:

res.append(nestedElement.getInteger())

def next(self):

"""

:rtype: int

"""

res = self.intlist[self.idx]

self.idx += 1

return res

def hasNext(self):

"""

:rtype: bool

"""

return self.idx <= self.end

# Your NestedIterator object will be instantiated and called as such:

# i, v = NestedIterator(nestedList), []

# while i.hasNext(): v.append(i.next())

PS: 面试时候遇到一道类似的题, 一个List其中元素有可能是整数,也有课能是nested list, 写一个函数falttern it

class Solution():

def flattern(self, vec2d):

res = []

self.flatternHelper(vec2d, res)

return res

def flatternHelper(self, vec2d, res):

i = 0

n = len(vec2d)

while i < n:

nested = vec2d[i]

if type(nested) is list:

self.flatternHelper(nested, res)

else:

res.append(nested)

i += 1

def test(self):

vec2d = [

[1,2],

[3],

[[4,5],6], 1, 2

]

print self.flattern(vec2d)

**271-Encode and Decode Strings**

Design an algorithm to encode a list of strings to a string.  
The encoded string is then sent over the network and is decoded back to the original list of strings.

Machine 1 (sender) has the function:

string encode(vector<string> strs) {

// ... your **code**

return encoded\_string;

}

Machine 2 (receiver) has the function:

vector<string> decode(string s) {

//... your **code**

return strs;

}

So Machine 1 does:

string encoded\_string = encode(strs);

and Machine 2 does:

vector<string> strs2 = decode(encoded\_string);

strs2 in Machine 2 should be the same as strs in Machine 1.

Implement the encode and decode methods.

class Codec:

def encode(self, strs):

"""Encodes a list of strings to a single string.

:type strs: List[str]

:rtype: str

"""

if not strs or len(strs) == 0:

return ""

res = ""

for s in strs:

n = len(s)

res += str(n) + "/" + s

return res

def decode(self, s):

"""Decodes a single string to a list of strings.

:type s: str

:rtype: List[str]

"""

res = []

if not s or len(s) == 0:

return res

i = 0

n = len(s)

while i < n:

slash = s.index("/", i)

length = int(s[i:slash])

res.append(s[slash + 1: slash + 1 + length])

i = slash + length + 1

return res

**281-Zigzag Iterator**

Given two 1d vectors, implement an iterator to return their elements alternately.

Example:

Input:

v1 = [1,2]

v2 = [3,4,5,6]

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

Explanation: By calling next repeatedly until hasNext returns false,  
the order of elements returned by next should be: [1,3,2,4,5,6].

Follow up: What if you are given k 1d vectors? How well can your **code** be extended to such cases?  
Clarification for the follow up question:  
The "Zigzag" order is not clearly defined and is ambiguous for k > 2 cases.  
If "Zigzag" does not look right to you, replace "Zigzag" with "Cyclic". For example:

Input:

[1,2,3]

[4,5,6,7]

[8,9]

Output: [1,4,8,2,5,9,3,6,7].

Solution:

class ZigzagIterator(object):

def \_\_init\_\_(self, v1, v2):

"""

Initialize your data structure here.

:type v1: List[int]

:type v2: List[int]

"""

self.lst = []

i, j = 0, 0

len1 = len(v1)

len2 = len(v2)

while i < len1 or j < len2:

if i < len1:

self.lst.append(v1[i])

i += 1

if j < len2:

self.lst.append(v2[j])

j += 1

self.idx = 0

self.size = len1 + len2

def next(self):

"""

:rtype: int

"""

res = self.lst[self.idx]

self.idx += 1

return res

def hasNext(self):

"""

:rtype: bool

"""

return self.idx < self.size

# Your ZigzagIterator object will be instantiated and called as such:

# i, v = ZigzagIterator(v1, v2), []

# while i.hasNext(): v.append(i.next())

Follow up: look at merge K sorted List

**284-Peeking Iterator**

Given an Iterator class interface with methods: next() and hasNext(),  
design and implement a PeekingIterator that support the peek() operation --  
it essentially peek() at the element that will be returned by the next call to next().

Example:

Assume that the iterator is initialized to the beginning of the list: [1,2,3].

Call next() gets you 1, the first element in the list.

Now you call peek() and it returns 2, the next element. Calling next() after that still return 2.

You call next() the final time and it returns 3, the last element.

Calling hasNext() after that should return false.

Follow up: How would you extend your design to be generic and work with all types, not just integer?

Solution:

# Below is the interface for Iterator, which is already defined for you.

#

# class Iterator(object):

# def \_\_init\_\_(self, nums):

# """

# Initializes an iterator object to the beginning of a list.

# :type nums: List[int]

# """

#

# def hasNext(self):

# """

# Returns true if the iteration has more elements.

# :rtype: bool

# """

#

# def next(self):

# """

# Returns the next element in the iteration.

# :rtype: int

# """

class PeekingIterator(object):

def \_\_init\_\_(self, iterator):

"""

Initialize your data structure here.

:type iterator: Iterator

"""

self.iter = iterator

self.cached = None

self.peaked = False

def peek(self):

"""

Returns the next element in the iteration without advancing the iterator.

:rtype: int

"""

if not self.peaked:

self.cached = self.iter.next()

self.peaked = True

return self.cached

def next(self):

"""

:rtype: int

"""

if self.peaked:

self.peaked = False

return self.cached

return self.iter.next()

def hasNext(self):

"""

:rtype: bool

"""

if self.peaked:

return True

return self.iter.hasNext()

# Your PeekingIterator object will be instantiated and called as such:

# iter = PeekingIterator(Iterator(nums))

# while iter.hasNext():

# val = iter.peek() # Get the next element but not advance the iterator.

# iter.next() # Should return the same value as [val].

**348-Design Tic-Tac-Toe**

Design a Tic-tac-toe game that is played between two players on a n x n grid.

You may assume the following rules:

A move is guaranteed to be valid and is placed on an empty block.  
Once a winning condition is reached, no more moves is allowed.  
A player who succeeds in placing n of their marks in a horizontal, vertical, or diagonal row wins the game.  
Example:

Given n = 3, assume that player 1 is "X" and player 2 is "O" in the board.

TicTacToe toe = new TicTacToe(3);

toe.move(0, 0, 1); -> Returns 0 (no one wins)

|X| | |

| | | | // Player 1 makes a move at (0, 0).

| | | |

toe.move(0, 2, 2); -> Returns 0 (no one wins)

|X| |O|

| | | | // Player 2 makes a move at (0, 2).

| | | |

toe.move(2, 2, 1); -> Returns 0 (no one wins)

|X| |O|

| | | | // Player 1 makes a move at (2, 2).

| | |X|

toe.move(1, 1, 2); -> Returns 0 (no one wins)

|X| |O|

| |O| | // Player 2 makes a move at (1, 1).

| | |X|

toe.move(2, 0, 1); -> Returns 0 (no one wins)

|X| |O|

| |O| | // Player 1 makes a move at (2, 0).

|X| |X|

toe.move(1, 0, 2); -> Returns 0 (no one wins)

|X| |O|

|O|O| | // Player 2 makes a move at (1, 0).

|X| |X|

toe.move(2, 1, 1); -> Returns 1 (player 1 wins)

|X| |O|

|O|O| | // Player 1 makes a move at (2, 1).

|X|X|X|

Solution:

class TicTacToe(object):

def \_\_init\_\_(self, n):

"""

Initialize your data structure here.

:type n: int

"""

self.n = n

self.row = [[0 for j in range(n)] for i in range(2)]

self.col = [[0 for j in range(n)] for i in range(2)]

self.diag = [[0, 0], [0, 0]]

def move(self, r, c, player):

"""

Player {player} makes a move at ({row}, {col}).

@param r The row of the board.

@param c The column of the board.

@param player The player, can be either 1 or 2.

@return The current winning condition, can be either:

0: No one wins.

1: Player 1 wins.

2: Player 2 wins.

:type row: int

:type col: int

:type player: int

:rtype: int

"""

player -= 1

self.row[player][r] += 1

if self.row[player][r] == self.n:

return player + 1

self.col[player][c] += 1

if self.col[player][c] == self.n:

return player + 1

if r == c:

self.diag[player][0] += 1

if self.diag[player][0] == self.n:

return player + 1

if r + c == self.n - 1:

self.diag[player][1] += 1

if self.diag[player][1] == self.n:

return player + 1

return 0

**355-Design Twitter**

Design a simplified version of Twitter where users can post tweets, follow/unfollow another user and is able to see the 10 most recent tweets in the user's news feed. Your design should support the following methods:

1. postTweet(userId, tweetId): Compose a new tweet.
2. getNewsFeed(userId): Retrieve the 10 most recent tweet ids in the user's news feed. Each item in the news feed must be posted by users who the user followed or by the user herself. Tweets must be ordered from most recent to least recent.
3. follow(followerId, followeeId): Follower follows a followee.
4. unfollow(followerId, followeeId): Follower unfollows a followee.

Example:

Twitter twitter = new Twitter();

// User 1 posts a new tweet (id = 5).

twitter.postTweet(1, 5);

// User 1's news feed should return a list with 1 tweet id -> [5].

twitter.getNewsFeed(1);

// User 1 follows user 2.

twitter.follow(1, 2);

// User 2 posts a new tweet (id = 6).

twitter.postTweet(2, 6);

// User 1's news feed should return a list with 2 tweet ids -> [6, 5].

// Tweet id 6 should precede tweet id 5 because it is posted after tweet id 5.

twitter.getNewsFeed(1);

// User 1 unfollows user 2.

twitter.unfollow(1, 2);

// User 1's news feed should return a list with 1 tweet id -> [5],

// since user 1 is no longer following user 2.

twitter.getNewsFeed(1);

结合系统设计真题: Design Twitter来分析

// Tweet link to next Tweet so that we can save a lot of time

// when we execute getNewsFeed(userId)

class Tweet{

public int id;

public int time;

public Tweet next;

public Tweet(int id){

this.id = id;

time = Twitter.timeStamp;

Twitter.timeStamp += 1;

next = null;

}

}

class User{

public int id;

public Set<Integer> followed;

public Tweet tweet\_head;

public User(int id){

this.id=id;

followed = new HashSet<>();

addFollowee(id); // first follow itself

tweet\_head = null;

}

public void addFollowee(int id){

followed.add(id);

}

public void removeFollowee(int id){

followed.remove(id);

}

// everytime user post a new tweet, add it to the head of tweet list.

public void post(int id){

Tweet tw = new Tweet(id);

tw.next = tweet\_head;

tweet\_head = tw;

}

}

public class Twitter {

public static int timeStamp;

// easy to find if user exist

private Map<Integer, User> userMap;

/\*\* Initialize your data structure here. \*/

public Twitter() {

timeStamp = 0;

userMap = new HashMap<Integer, User>();

}

/\*\* Compose a new tweet. \*/

public void postTweet(int userId, int tweetId) {

if(!userMap.containsKey(userId)){

User u = new User(userId);

userMap.put(userId, u);

}

userMap.get(userId).post(tweetId);

}

/\*\* Follower follows a followee. If the operation is invalid, it should be a no-op. \*/

public void follow(int followerId, int followeeId) {

if(!userMap.containsKey(followerId)){

User u = new User(followerId);

userMap.put(followerId, u);

}

if(!userMap.containsKey(followeeId)){

User u = new User(followeeId);

userMap.put(followeeId, u);

}

userMap.get(followerId).addFollowee(followeeId);

}

/\*\* Follower unfollows a followee. If the operation is invalid, it should be a no-op. \*/

public void unfollow(int followerId, int followeeId) {

if(!userMap.containsKey(followerId) || followerId == followeeId)

return;

userMap.get(followerId).removeFollowee(followeeId);

}

// first get all tweets lists from one user including itself and all people it followed.

// Second add all heads into a max heap. Every time we poll a tweet with

// largest time stamp from the heap, then we add its next tweet into the heap.

// So after adding all heads we only need to add 9 tweets at most into this

// heap before we get the 10 most recent tweet.

public List<Integer> getNewsFeed(int userId) {

List<Integer> res = new LinkedList<>();

if(!userMap.containsKey(userId)) {

return res;

}

Set<Integer> followees = userMap.get(userId).followed;

PriorityQueue<Tweet> maxHeap = new PriorityQueue<Tweet>(new Comparator<Tweet>() {

public int compare(Tweet a, Tweet b) {

return b.time - a.time;

}

});

for(int followee: followees){

Tweet t = userMap.get(followee).tweet\_head;

// very imporant! If we add null to the head we are screwed.

if(t!=null){

maxHeap.add(t);

}

}

int n=0;

while(!maxHeap.isEmpty() && n<10){

Tweet t = maxHeap.poll();

res.add(t.id);

n++;

if(t.next!=null)

maxHeap.add(t.next);

}

return res;

}

}

/\*

\* Your Twitter object will be instantiated and called as such:

\* Twitter obj = new Twitter();

\* obj.postTweet(userId,tweetId);

\* List<Integer> param\_2 = obj.getNewsFeed(userId);

\* obj.follow(followerId,followeeId);

\* obj.unfollow(followerId,followeeId);

\*/

**380-Insert Delete GetRandom O(1)**

Design a data structure that supports all following operations in average O(1) time.

1. insert(val): Inserts an item val to the set if not already present.
2. remove(val): Removes an item val from the set if present.
3. getRandom(): Returns a random element from current set of elements.  
   Each element must have the same probability of being returned.

from random import randint

class RandomizedSet(object):

def \_\_init\_\_(self):

"""

Initialize your data structure here.

"""

self.myset = set()

self.lst = []

def insert(self, val):

"""

Inserts a value to the set. Returns true if the set did not already contain the specified element.

:type val: int

:rtype: bool

"""

if val in self.myset:

return False

self.myset.add(val)

self.lst.append(val)

return True

def remove(self, val):

"""

Removes a value from the set. Returns true if the set contained the specified element.

:type val: int

:rtype: bool

"""

if val not in self.myset:

return False

self.myset.remove(val)

self.lst.remove(val)

return True

def getRandom(self):

"""

Get a random element from the set.

:rtype: int

"""

size = len(self.lst)

idx = randint(0, size - 1)

return self.lst[idx]

**384-Shuffle an Array**

Shuffle a set of numbers without duplicates.

from random import randint

class Solution(object):

def \_\_init\_\_(self, nums):

"""

:type nums: List[int]

"""

self.nums = nums

def reset(self):

"""

Resets the array to its original configuration and return it.

:rtype: List[int]

"""

return self.nums

def shuffle(self):

"""

Returns a random shuffling of the array.

:rtype: List[int]

"""

res = []

if not self.nums:

return res

myset = set()

size = len(self.nums)

while True:

idx = randint(0, size - 1)

if self.nums[idx] in myset:

continue

res.append(self.nums[idx])

myset.add(self.nums[idx])

if len(res) == size:

break

return res

**398-Random Pick Index**

Given an array of integers with possible duplicates, randomly output the index of  
a given target number. You can assume that the given target number must exist in the array.

from random import randint

class Solution(object):

def \_\_init\_\_(self, nums):

"""

:type nums: List[int]

"""

self.nums = nums

self.size = len(nums)

def pick(self, target):

"""

:type target: int

:rtype: int

"""

res = 0

while True:

idx = randint(0, self.size - 1)

if self.nums[idx] != target:

continue

res = idx

break

return res

**460-LFU Cache**

**535-Encode and Decode TinyURL**

该题目可和系统设计真题 TinyURL结合分析

from random import randint

class Codec:

def \_\_init\_\_(self):

self.dic = {}

def encode(self, longUrl):

"""Encodes a URL to a shortened URL.

:type longUrl: str

:rtype: str

"""

key = 'http://tinyurl.com/'

for i in range(6):

key += chr(randint(0, 255))

# dic = {key: longUrl}

self.dic[key] = longUrl

return key

def decode(self, shortUrl):

"""Decodes a shortened URL to its original URL.

:type shortUrl: str

:rtype: str

"""

return self.dic[shortUrl]

# LeetCode总结八: 二叉树

**094-Binary Tree Inorder Traversal**

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

Example:

Input: [1,null,2,3]

1

\

2

/

3

Output: [1,3,2]

Solution1: iterative

class Solution(object):

def inorderTraversal(self, root):

"""

:type root: TreeNode

:rtype: List[int]

"""

stack = []

res = []

cur = root

while True:

if cur:

stack.append(cur)

cur = cur.left

else:

if not stack:

break

node = stack.pop()

res.append(node.val)

cur = node.right

return res

Solution2: recursive

class Solution(object):

def inorderTraversal(self, root):

"""

:type root: TreeNode

:rtype: List[int]

"""

res = []

if not root:

return res

self.helper(root, res)

return res

def helper(self, node, res):

if not node:

return

self.helper(node.left, res)

res.append(node.val)

self.helper(node.right, res)

**095-Unique BST II**

Given an integer n, generate all structurally unique BST's (binary search trees) that store values 1 ... n.

Example:

Input: 3

Output:

[

[1,null,3,2],

[3,2,null,1],

[3,1,null,null,2],

[2,1,3],

[1,null,2,null,3]

]

Explanation:

The above output corresponds to the 5 unique BST's shown below:

1 3 3 2 1

\ / / / \ \

3 2 1 1 3 2

/ / \ \

2 1 2 3

Solution:

# 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 generateTrees(self, n):

"""

:type n: int

:rtype: List[TreeNode]

"""

if n == 0:

return []

return self.helper(1, n)

def helper(self, start, end):

res = []

if start > end:

res.append(None)

return res

if start == end:

res.append(TreeNode(start))

return res

for i in range(start, end + 1):

left = self.helper(start, i - 1)

right = self.helper(i + 1, end)

for l in left:

for r in right:

root = TreeNode(i)

root.left = l

root.right = r

res.append(root)

return res

**098-Validate Binary Search Tree**

Solution1: iterative

class Solution(object):

def isValidBST(self, root):

"""

:type root: TreeNode

:rtype: bool

"""

stack = []

cur = root

prev = None

while True:

if cur:

stack.append(cur)

cur = cur.left

else:

if not stack:

break

node = stack.pop()

if prev and node.val <= prev.val:

return False

cur = node.right

prev = node

return True

Solution2: recursive

class Solution(object):

def isValidBST(self, root):

"""

:type root: TreeNode

:rtype: bool

"""

if not root:

return True

if not root.left and not root.right:

return True

INF = (1 << 32) - 1

large = INF

small = -INF

return self.helper(root, large, small)

def helper(self, node, large, small):

if not node:

return True

if node.val >= large or node.val <= small:

return False

return self.helper(node.left, node.val, small) and self.helper(node.right, large, node.val)

**108-Convert Sorted Array to Binary Search Tree**

Given an array where elements are sorted in ascending order, convert it to a height balanced BST.

For this problem, a height-balanced binary tree is defined as a binary tree in which  
the depth of the two subtrees of every node never differ by more than 1.

Example:

Given the sorted array: [-10,-3,0,5,9],

One possible answer is: [0,-3,9,-10,null,5], which represents the following height balanced BST:

0

/ \

-3 9

/ /

-10 5

Solution:

class Solution(object):

def sortedArrayToBST(self, nums):

"""

:type nums: List[int]

:rtype: TreeNode

"""

l, r = 0, len(nums) - 1

return self.helper(nums, l, r)

def helper(self, nums, l, r):

if l > r:

return None

mid = l + (r - l) / 2

root = TreeNode(nums[mid])

left = self.helper(nums, l, mid - 1)

right = self.helper(nums, mid + 1, r)

root.left = left

root.right = right

return root

**230-Kth Smallest Element in a BST**

Solution1: iterative

class Solution(object):

def kthSmallest(self, root, k):

"""

:type root: TreeNode

:type k: int

:rtype: int

"""

stack = []

cur = root

count = k

prev = None

res = 0

while True:

if cur:

stack.append(cur)

cur = cur.left

else:

if not stack:

break

node = stack.pop()

count -= 1

if count == 0:

res = node.val

break

cur = node.right

prev = node

return res

Solution2: recursive

class Solution(object):

def kthSmallest(self, root, k):

"""

:type root: TreeNode

:type k: int

:rtype: int

"""

res = []

count = [k]

self.helper(root, res, count)

return res[0]

def helper(self, node, res, count):

if not node:

return

self.helper(node.left, res, count)

count[0] -= 1

if count[0] == 0:

res.append(node.val)

return

self.helper(node.right, res, count)

**235-Lowest Common Ancestor of Binary Search Tree**

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

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).”

Given binary search tree: root = [6,2,8,0,4,7,9,null,null,3,5]

\_\_\_\_\_\_\_6\_\_\_\_\_\_

/ \

\_\_\_2\_\_ \_\_\_8\_\_

/ \ / \

0 \_4 7 9

/ \

3 5

Example 1:

Input: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 8

Output: 6

Explanation: The LCA of nodes 2 and 8 is 6.

Example 2:

Input: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 4

Output: 2

Explanation: The LCA of nodes 2 and 4 is 2, since a node can be a descendant of itself

according to the LCA definition.

Solution1:

class Solution(object):

def lowestCommonAncestor(self, root, p, q):

"""

:type root: TreeNode

:type p: TreeNode

:type q: TreeNode

:rtype: TreeNode

"""

if not root:

return None

minVal = min(p.val, q.val)

maxVal = max(p.val, q.val)

if minVal <= root.val <= maxVal:

return root

if root.val > maxVal:

return self.lowestCommonAncestor(root.left, p, q)

else:

return self.lowestCommonAncestor(root.right, p, q)

Solution2:

class Solution(object):

def lowestCommonAncestor(self, root, p, q):

"""

:type root: TreeNode

:type p: TreeNode

:type q: TreeNode

:rtype: TreeNode

"""

if not root:

return None

minVal = min(p.val, q.val)

maxVal = max(p.val, q.val)

res = None

stack = []

cur = root

while cur:

if minVal <= cur.val <= maxVal:

res = cur

break

elif cur.val > maxVal:

cur = cur.left

else:

cur = cur.right

return res

**236-Lowest Common Ancestor of a Binary Tree**

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).”

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

\_\_\_\_\_\_\_3\_\_\_\_\_\_

/ \

\_\_\_5\_\_ \_\_\_1\_\_

/ \ / \

6 \_2 0 8

/ \

7 4

Example 1:

Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1

Output: 3

Explanation: The LCA of of nodes 5 and 1 is 3.

Example 2:

Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 4

Output: 5

Explanation: The LCA of nodes 5 and 4 is 5, since a node can be a descendant of itself

according to the LCA definition.

Solution:

class Solution(object):

def lowestCommonAncestor(self, root, p, q):

"""

:type root: TreeNode

:type p: TreeNode

:type q: TreeNode

:rtype: TreeNode

"""

parents = {root: None}

queue = [root]

while p not in parents or q not in parents:

node = queue.pop(0)

if node.left:

queue.append(node.left)

parents[node.left] = node

if node.right:

queue.append(node.right)

parents[node.right] = node

myset = set()

while p:

myset.add(p)

p = parents[p]

while q not in myset:

q = parents[q]

return q

**450-Delete Node in a BST**

Given a root node reference of a BST and a key, delete the node with the given key in the BST.  
Return the root node reference (possibly updated) of the BST.

Basically, the deletion can be divided into two stages:

Search for a node to remove.  
If the node is found, delete the node.  
Note: Time complexity should be O(height of tree).

Example:

root = [5,3,6,2,4,null,7]

key = 3

5

/ \

3 6

/ \ \

2 4 7

Given key to delete is 3. So we find the node with value 3 and delete it.

One valid answer is [5,4,6,2,null,null,7], shown in the following BST.

5

/ \

4 6

/ \

2 7

Another valid answer is [5,2,6,null,4,null,7].

5

/ \

2 6

\ \

4 7

Solution:

1. if its a leaf node, simply remove it
2. the node has only one child (left or right), attach its child's subtree to it
3. The node has both left and right child

* find minimun of the right subtree
* copy the value of it min to the targeted node (the one to be deleted)
* remove the duplicated

class Solution(object):

def deleteNode(self, root, key):

"""

:type root: TreeNode

:type key: int

:rtype: TreeNode

"""

if not root:

return None

if root.val > key:

root.left = self.deleteNode(root.left, key)

elif root.val < key:

root.right = self.deleteNode(root.right, key)

else:

if not root.left:

return root.right

if not root.right:

return root.left

minNode = self.findMin(root.right)

root.val = minNode.val

root.right = self.deleteNode(root.right, root.val)

return root

def findMin(self, node):

cur = node

while cur.left:

cur = cur.left

return cur

**099-Recover Binary Search Tree**

Two elements of a binary search tree (BST) are swapped by mistake.  
Recover the tree without changing its structure.

Example 2:

Input: [3,1,4,null,null,2]

3

/ \

1 4

/

2

Output: [2,1,4,null,null,3]

2

/ \

1 4

/

3

Solution:

class Solution(object):

def recoverTree(self, root):

"""

:type root: TreeNode

:rtype: void Do not return anything, modify root in-place instead.

"""

prev = p = q = None

stack = []

cur = root

while True:

if cur:

stack.append(cur)

cur = cur.left

else:

if not stack:

break

node = stack.pop()

if prev and node.val < prev.val:

if not p:

p = prev

q = node

cur = node.right

prev = node

tmp = p.val

p.val = q.val

q.val = tmp

**100-Same Tree**

Given two binary trees, write a function to check if they are the same or not.

Two binary trees are considered the same if they are structurally identical and the nodes have the same value.

Example 1:

Input: 1 1

/ \ / \

2 3 2 3

[1,2,3], [1,2,3]

Output: true

Example 2:

Input: 1 1

/ \

2 2

[1,2], [1,null,2]

Output: false

Example 3:

Input: 1 1

/ \ / \

2 1 1 2

[1,2,1], [1,1,2]

Output: false

Solution1:

class Solution(object):

def isSameTree(self, p, q):

"""

:type p: TreeNode

:type q: TreeNode

:rtype: bool

"""

if not p and not q:

return True

if p and not q or not p and q:

return False

if p.val != q.val:

return False

return self.isSameTree(p.left, q.left) and self.isSameTree(p.right, q.right)

Solution2:

class Solution(object):

def isSameTree(self, p, q):

"""

:type p: TreeNode

:type q: TreeNode

:rtype: bool

"""

stack = []

curp = p

curq = q

while True:

if curp and not curq or not curp and curq:

return False

if curp and curq:

stack.append(curp)

stack.append(curq)

curp = curp.left

curq = curq.left

else:

if not stack:

break

nq = stack.pop()

np = stack.pop()

if np.val != nq.val:

return False

curp = np.right

curq = nq.right

return True

**102-Binary Tree Level Order Traversal**

Solution: BFS

class Solution(object):

def levelOrder(self, root):

"""

:type root: TreeNode

:rtype: List[List[int]]

"""

res = []

if not root:

return res

queue = [root]

while queue:

n = len(queue)

tmp = []

for i in range(n):

node = queue.pop(0)

if node.left:

queue.append(node.left)

if node.right:

queue.append(node.right)

tmp.append(node.val)

res.append(tmp)

return res

**103-Binary Tree Zigzag Level Order Traversal**

For example:

Given binary tree [3,9,20,null,null,15,7],

3

/ \

9 20

/ \

15 7

return its zigzag level order traversal as:

[

[3],

[20,9],

[15,7]

]

Solution (BFS):

class Solution(object):

def zigzagLevelOrder(self, root):

"""

:type root: TreeNode

:rtype: List[List[int]]

"""

res = []

if not root:

return res

queue = [root]

layer = 1

while queue:

n = len(queue)

tmp = []

for i in range(n):

node = queue.pop(0)

if node.left:

queue.append(node.left)

if node.right:

queue.append(node.right)

if layer % 2 == 0:

tmp.insert(0, node.val)

else:

tmp.append(node.val)

layer += 1

res.append(tmp)

return res

**144-Binary Tree Preorder Traversal**

Example:

Input: [3, 5, null, null, 9, 10, 20]

3

/ \

5 9

/ \

10 20

output: [3, 5, 9, 10, 20]

Solution:

class Solution(object):

def preorderTraversal(self, root):

"""

:type root: TreeNode

:rtype: List[int]

"""

res = []

if not root:

return res

stack = [root]

while stack:

node = stack.pop()

if node.right:

stack.append(node.right)

if node.left:

stack.append(node.left)

res.append(node.val)

return res

**145-Binary Tree Postorder Traversal**

Input: [3, 5, null, null, 9, 10, 20]

3

/ \

5 9

/ \

10 20

Output: [5, 10, 20, 9, 3]

Solution:

class Solution(object):

def postorderTraversal(self, root):

"""

:type root: TreeNode

:rtype: List[int]

"""

res = []

if not root:

return res

stack = [root]

while stack:

node = stack.pop()

if node.left:

stack.append(node.left)

if node.right:

stack.append(node.right)

res.insert(0, node.val)

return res

**104-Maximum Depth of Binary Tree**

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.

Example:

Given binary tree [3,9,20,null,null,15,7],

3

/ \

9 20

/ \

15 7

Solution:

# 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

if not root.left and not root.right:

return 1

left = 1 + self.maxDepth(root.left)

right = 1 + self.maxDepth(root.right)

return max(left, right)

**105-Construct Binary Tree from Preorder and Inorder Traversal**

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

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

For example, given

inorder = [9,3,15,20,7]

postorder = [9,15,7,20,3]

Return the following binary tree:

3

/ \

9 20

/ \

15 7

Solution:

class Solution(object):

def buildTree(self, inorder, postorder):

"""

:type inorder: List[int]

:type postorder: List[int]

:rtype: TreeNode

"""

if not inorder or not postorder:

return None

root = TreeNode(postorder.pop())

idx = inorder.index(root.val)

right = self.buildTree(inorder[idx + 1:], postorder)

left = self.buildTree(inorder[:idx], postorder)

root.left = left

root.right = right

return root

**106-Construct Binary Tree from Inorder and Postorder Traver**

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

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

For example, given

preorder = [3,9,20,15,7]

inorder = [9,3,15,20,7]

Return the following binary tree:

3

/ \

9 20

/ \

15 7

Solution:

class Solution(object):

def buildTree(self, preorder, inorder):

"""

:type preorder: List[int]

:type inorder: List[int]

:rtype: TreeNode

"""

if not preorder or not inorder:

return None

root = TreeNode(preorder.pop(0))

idx = inorder.index(root.val)

left = self.buildTree(preorder, inorder[:idx])

right = self.buildTree(preorder, inorder[idx + 1:])

root.left = left

root.right = right

return root

**110-Balanced Binary Tree**

Given a binary tree, determine if it is height-balanced.

For this problem, a height-balanced binary tree is defined as:

a binary tree in which the depth of the two subtrees of every node never differ by more than 1.

Example 1:

Given the following tree [3,9,20,null,null,15,7]:

3

/ \

9 20

/ \

15 7

Return true.

Example 2:

Given the following tree [1,2,2,3,3,null,null,4,4]:

1

/ \

2 2

/ \

3 3

/ \

4 4

Solution:

class Solution(object):

def isBalanced(self, root):

"""

:type root: TreeNode

:rtype: bool

"""

res = [True]

if not root:

return True

self.helper(root, res)

return res[0]

def helper(self, node, res):

if not node:

return 0

l = 1 + self.helper(node.left, res)

r = 1 + self.helper(node.right, res)

if abs(l - r) > 1:

res[0] = False

return max(l, r)

return max(l, r)

**111-Minimum Depth of Binary Tree**

Given a binary tree, find its minimum depth.  
The minimum depth is the number of nodes along the shortest  
path from the root node down to the nearest leaf node.

Note: A leaf is a node with no children.

Example:

Given binary tree [3,9,20,null,null,15,7],

3

/ \

9 20

/ \

15 7

return its minimum depth = 2.

Solution:

class Solution(object):

def minDepth(self, root):

"""

:type root: TreeNode

:rtype: int

"""

if not root:

return 0

queue = [root]

layer = 1

while queue:

n = len(queue)

for i in range(n):

node = queue.pop(0)

if not node.left and not node.right:

return layer

if node.left:

queue.append(node.left)

if node.right:

queue.append(node.right)

layer += 1

return layer

**112-Path Sum**

Given a binary tree and a sum, determine if the tree has a root-to-leaf path such  
that adding up all the values along the path equals the given sum.

Note: A leaf is a node with no children.

Example:

Given the below binary tree and sum = 22,

5

/ \

4 8

/ / \

11 13 4

/ \ \

7 2 1

return true, as there exist a root-to-leaf path 5->4->11->2 which sum is 22.

Recursive Solution:

class Solution(object):

def hasPathSum(self, root, sum):

"""

:type root: TreeNode

:type sum: int

:rtype: bool

"""

if not root:

return False

if not root.left and not root.right:

return root.val == sum

return self.hasPathSum(root.left, sum - root.val) or self.hasPathSum(root.right, sum - root.val)

BFS Solution:

class KeyNode(object):

def \_\_init\_\_(self, node, val):

self.node = node

self.val = val

class Solution(object):

def hasPathSum(self, root, sum):

"""

:type root: TreeNode

:type sum: int

:rtype: bool

"""

if not root:

return False

queue = [KeyNode(root, root.val)]

while queue:

keynode = queue.pop(0)

node = keynode.node

curval = keynode.val

if not node.left and not node.right and curval == sum:

return True

if node.left:

queue.append(KeyNode(node.left, curval + node.left.val))

if node.right:

queue.append(KeyNode(node.right, curval + node.right.val))

return False

**113-Path Sum II**

Given a binary tree and a sum, find all root-to-leaf paths where each path's sum equals the given sum.

Note: A leaf is a node with no children.

Example:

Given the below binary tree and sum = 22,

5

/ \

4 8

/ / \

11 13 4

/ \ / \

7 2 5 1

Return:

[

[5,4,11,2],

[5,8,4,5]

]

DFS solution:

class Solution(object):

def pathSum(self, root, sum):

"""

:type root: TreeNode

:type sum: int

:rtype: List[List[int]]

"""

res = []

can = []

if not root:

return res

self.helper(root, sum, res, can)

return res

def helper(self, node, sum, res, can):

if not node:

return

if not node.left and not node.right:

if node.val == sum:

can.append(node.val)

res.append(can[:])

can.pop()

return

can.append(node.val)

self.helper(node.left, sum - node.val, res, can)

self.helper(node.right, sum - node.val, res, can)

can.pop()

BFS Solution:

class Keynode(object):

def \_\_init\_\_(self, node, val, path):

self.node = node

self.val = val

self.path = path

class Solution(object):

def pathSum(self, root, sum):

"""

:type root: TreeNode

:type sum: int

:rtype: List[List[int]]

"""

res = []

if not root:

return res

queue = [Keynode(root, root.val, str(root.val))]

while queue:

keynode = queue.pop(0)

node = keynode.node

curval = keynode.val

path = keynode.path

if not node.left and not node.right and curval == sum:

can = []

strs = path.split(',')

for s in strs:

can.append(int(s))

res.append(can)

if node.left:

pathl = path + ',' + str(node.left.val)

newKeynode = Keynode(node.left, curval + node.left.val, pathl)

queue.append(newKeynode)

if node.right:

pathr = path +',' + str(node.right.val)

newKeynode = Keynode(node.right, curval + node.right.val, pathr)

queue.append(newKeynode)

return res

**437-Path Sum III**

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.

Example:

root = [10,5,-3,3,2,null,11,3,-2,null,1], sum = 8

10

/ \

5 -3

/ \ \

3 2 11

/ \ \

3 -2 1

Return 3. The paths that sum to 8 are:

1. 5 -> 3

2. 5 -> 2 -> 1

3. -3 -> 11

Solution: DFS + backtracking

class Solution(object):

def pathSum(self, root, sum):

"""

:type root: TreeNode

:type sum: int

:rtype: int

"""

dic = {0: 1}

res = [0]

self.helper(root, 0, sum, dic, res)

return res[0]

def helper(self, node, cursum, target, dic, res):

if not node:

return 0

cursum += node.val

if cursum - target in dic:

res[0] += dic[cursum - target]

if cursum in dic:

dic[cursum] += 1

else:

dic[cursum] = 1

self.helper(node.left, cursum, target, dic, res)

self.helper(node.right, cursum, target, dic, res)

dic[cursum] -= 1

**124-Binary Tree Maximum Path Sum**

Given a non-empty binary tree, find the maximum path sum.

For this problem, a path is defined as any sequence of nodes from some starting node  
to any node in the tree along the parent-child connections.  
The path must contain at least one node and does not need to go through the root.

Example 1:

Input: [1,2,3]

1

/ \

2 3

Output: 6

Example 2:

Input: [-10,9,20,null,null,15,7]

-10

/ \

9 20

/ \

15 7

Output: 42

Solution:

class Solution(object):

def maxPathSum(self, root):

"""

:type root: TreeNode

:rtype: int

"""

MIN\_INT = ((1 << 32) - 1) \* (-1)

res = [MIN\_INT]

self.helper(root, res)

return res[0]

def helper(self, node, res):

if not node:

return 0

left = max(0, self.helper(node.left, res))

right = max(0, self.helper(node.right, res))

right = self.helper(node.right, res)

res[0] = max(res[0], left + right + node.val)

return max(left, right) + node.val

**116-Populating Next Right Pointers in Each Node**

Solution 1: O(n) space complexity solution - BFS

# class TreeLinkNode:

# def \_\_init\_\_(self, x):

# self.val = x

# self.left = None

# self.right = None

# self.next = None

class Solution:

# @param root, a tree link node

# @return nothing

def connect(self, root):

if not root:

return

queue = [root]

while queue:

n = len(queue)

dummy = TreeLinkNode(0)

cur = dummy

for i in range(n):

node = queue.pop(0)

cur.next = node

cur = cur.next

if node.left:

queue.append(node.left)

if node.right:

queue.append(node.right)

dummy.next = None

Solution 2: O(1) space complexity

# class TreeLinkNode:

# def \_\_init\_\_(self, x):

# self.val = x

# self.left = None

# self.right = None

# self.next = None

class Solution:

# @param root, a tree link node

# @return nothing

def connect(self, root):

if not root:

return

head = root

cur = None

while head.left:

cur = head

while cur:

cur.left.next = cur.right

if cur.next:

cur.right.next = cur.next.left

cur = cur.next

head = head.left

**129-Summ Root to Leaf Numbers**

Given a binary tree containing digits from 0-9 only, each root-to-leaf path could represent a number.  
An example is the root-to-leaf path 1->2->3 which represents the number 123.  
Find the total sum of all root-to-leaf numbers.

Note: A leaf is a node with no children.

Example:

Input: [1,2,3]

1

/ \

2 3

Output: 25

Explanation:

The root-to-leaf path 1->2 represents the number 12.

The root-to-leaf path 1->3 represents the number 13.

Therefore, sum = 12 + 13 = 25.

Example 2:

Input: [4,9,0,5,1]

4

/ \

9 0

/ \

5 1

Output: 1026

Explanation:

The root-to-leaf path 4->9->5 represents the number 495.

The root-to-leaf path 4->9->1 represents the number 491.

The root-to-leaf path 4->0 represents the number 40.

Therefore, sum = 495 + 491 + 40 = 1026.

Solution:

class Solution(object):

def sumNumbers(self, root):

"""

:type root: TreeNode

:rtype: int

"""

if not root:

return 0

nums = []

self.helper(root, nums, root.val)

return sum(nums)

def helper(self, node, nums, curval):

if not node.left and not node.right:

nums.append(curval)

return

if node.left:

self.helper(node.left, nums, curval \* 10 + node.left.val)

if node.right:

self.helper(node.right, nums, curval \* 10 + node.right.val)

**156-Binary Tree Upside Down**

Given a binary tree where all the right nodes are either leaf nodes with a sibling (a left node that shares the same parent node) or empty, flip it upside down and turn it into a tree where the original right nodes turned into left leaf nodes. Return the new root.

Example:

Input: [1,2,3,4,5]

1

/ \

2 3

/ \

4 5

Output: return the root of the binary tree [4,5,2,#,#,3,1]

4

/ \

5 2

/ \

3 1

Clarification:

Confused what [4,5,2,#,#,3,1] means? Read more below on how binary tree is serialized on OJ.

The serialization of a binary tree follows a level order traversal, where '#' signifies a path terminator where no node exists below.

Here's an example:

1

/ \

2 3

/

4

\

5

The above binary tree is serialized as [1,2,3,#,#,4,#,#,5].

Solution:

class Solution(object):

def upsideDownBinaryTree(self, root):

"""

:type root: TreeNode

:rtype: TreeNode

"""

tmp = None

next = None

prev = None

cur = root

while cur:

next = cur.left

cur.left = tmp

tmp = cur.right

cur.right = prev

prev = cur

cur = next

return prev

**173-Binary Search Tree Iterator**

Implement an iterator over a binary search tree (BST).  
Your iterator will be initialized with the root node of a BST.  
Calling next() will return the next smallest number in the BST.  
Note: next() and hasNext() should run in average O(1) time and  
uses O(h) memory, where h is the height of the tree.

Solution 1: O(n) memory

class BSTIterator(object):

def \_\_init\_\_(self, root):

"""

:type root: TreeNode

"""

vals = []

self.helper(vals, root)

self.vals = vals

self.start = 0

self.end = len(self.vals) - 1

def helper(self, vals, node):

if not node:

return

self.helper(vals, node.left)

vals.append(node.val)

self.helper(vals, node.right)

def hasNext(self):

"""

:rtype: bool

"""

return self.start <= self.end

def next(self):

"""

:rtype: int

"""

val = self.vals[self.start]

self.start += 1

return val

Solution 2: O(h) memory

class BSTIterator(object):

def \_\_init\_\_(self, root):

"""

:type root: TreeNode

"""

self.stack = []

self.helper(self.stack, root)

def helper(self, stack, node):

cur = node

while cur:

stack.append(cur)

cur = cur.left

def hasNext(self):

"""

:rtype: bool

"""

return len(self.stack) > 0

def next(self):

"""

:rtype: int

"""

node = self.stack.pop()

self.helper(self.stack, node.right)

return node.val

**199-Binary Tree Right Side View**

class Solution(object):

def rightSideView(self, root):

"""

:type root: TreeNode

:rtype: List[int]

"""

res = []

if not root:

return res

queue = [root]

while queue:

n = len(queue)

for i in range(n):

node = queue.pop(0)

if i == n - 1:

res.append(node.val)

if node.left:

queue.append(node.left)

if node.right:

queue.append(node.right)

return resß

**222-Count Complete Tree Nodes**

Given a complete binary tree, count the number of nodes.

Note:

Definition of a complete binary tree from Wikipedia:  
In a complete binary tree every level, except possibly the last,  
is completely filled, and all nodes in the last level are as far left as possible.  
It can have between 1 and 2h nodes inclusive at the last level h.

Example:

Input:

1

/ \

2 3

/ \ /

4 5 6

Output: 6

Solution:

class Solution(object):

def countNodes(self, root):

"""

:type root: TreeNode

:rtype: int

"""

if not root:

return 0

lh = 1 + self.leftHeight(root.left)

rh = 1 + self.rightHeight(root.right)

if lh == rh:

return 2 \*\* lh - 1

return 1 + self.countNodes(root.left) + self.countNodes(root.right)

def leftHeight(self, node):

if not node:

return 0

return 1 + self.leftHeight(node.left)

def rightHeight(self, node):

if not node:

return 0

return 1 + self.rightHeight(node.right)

**226-Invert Binary Tree**

class Solution(object):

def invertTree(self, root):

"""

:type root: TreeNode

:rtype: TreeNode

"""

if not root:

return None

left = self.invertTree(root.left)

right = self.invertTree(root.right)

root.left = right

root.right = left

return root

**255-Verify Preorder Sequence in Binary Search Tree**

Given an array of numbers, verify whether it is the correct preorder traversal sequence of a binary search tree.

You may assume each number in the sequence is unique.

Consider the following binary search tree:

5

/ \

2 6

/ \

1 3

Example 1:

Input: [5,2,6,1,3]

Output: false

Example 2:

Input: [5,2,1,3,6]

Output: true

Solution:

class Solution(object):

def verifyPreorder(self, preorder):

"""

:type preorder: List[int]

:rtype: bool

"""

stack = []

prev = - (1 << 31)

for n in preorder:

if n < prev:

return False

while stack and stack[-1] < n:

prev = stack.pop()

stack.append(n)

return True

**250-Count Univalue Subtrees**

Given a binary tree, count the number of uni-value subtrees.

A Uni-value subtree means all nodes of the subtree have the same value.

Example :

Input: root = [5,1,5,5,5,null,5]

5

/ \

1 5

/ \ \

5 5 5

Output: 4

Solution:

class Solution(object):

def countUnivalSubtrees(self, root):

"""

:type root: TreeNode

:rtype: int

"""

count = [0]

self.helper(count, root)

return count[0]

def helper(self, count, node):

if not node:

return True

left = self.helper(count, node.left)

right = self.helper(count, node.right)

if left and right:

if node.left and node.val != node.left.val:

return False

if node.right and node.val != node.right.val:

return False

count[0] += 1

return True

return False

**257-Binary Tree Paths**

Given a binary tree, return all root-to-leaf paths.

Note: A leaf is a node with no children.

Example:

Input:

1

/ \

2 3

\

5

Output: ["1->2->5", "1->3"]

Explanation: All root-to-leaf paths are: 1->2->5, 1->3

Solution1: DFS

class Solution(object):

def binaryTreePaths(self, root):

"""

:type root: TreeNode

:rtype: List[str]

"""

res = []

if not root:

return res

can = ''

self.helper(root, res, can)

return res

def helper(self, node, res, can):

if not node.left and not node.right:

can += str(node.val)

res.append(can)

return

can = can + str(node.val) + '->'

if node.left:

self.helper(node.left, res, can)

if node.right:

self.helper(node.right, res, can)

Solution2: BFS

class StructNode(object):

def \_\_init\_\_(self, node, path):

self.node = node

self.path = path

class Solution(object):

def binaryTreePaths(self, root):

"""

:type root: TreeNode

:rtype: List[str]

"""

if not root:

return []

res = []

queue = [StructNode(root, str(root.val))]

while queue:

structNode = queue.pop(0)

node = structNode.node

path = structNode.path

if not node.left and not node.right:

res.append(path)

if node.left:

queue.append( StructNode(node.left, path + '->' + str(node.left.val)) )

if node.right:

queue.append( StructNode(node.right, path + '->' + str(node.right.val)) )

return res

**285-Inorder Successor in BST**

Solution1: O(n)

class Solution(object):

def inorderSuccessor(self, root, p):

"""

:type root: TreeNode

:type p: TreeNode

:rtype: TreeNode

"""

if not root:

return None

stack = []

prev = None

cur = root

while True:

if cur:

stack.append(cur)

cur = cur.left

else:

if not stack:

break

node = stack.pop()

if prev:

return node.val

if node == p:

prev = node

cur = node.right

return None

Solution2: O(logn)

class Solution(object):

def inorderSuccessor(self, root, p):

"""

:type root: TreeNode

:type p: TreeNode

:rtype: TreeNode

"""

if not root:

return None

cur = root

suc = None

while cur:

if cur.val > p.val:

suc = cur

cur = cur.left

else:

cur = cur.right

return suc

**449-Serialize and Deserialize BST**

Serialization is the process of converting a data structure or object into a sequence of bits  
so that it can be stored in a file or memory buffer, or transmitted across a network connection  
link to be reconstructed later in the same or another computer environment.

Design an algorithm to serialize and deserialize a binary search tree. There is no restriction  
on how your serialization/deserialization algorithm should work. You just need to ensure that  
a binary search tree can be serialized to a string and this string can be deserialized to the original tree structure.

The encoded string should be as compact as possible.

Note: Do not use class member/global/static variables to store states.  
Your serialize and deserialize algorithms should be stateless.

Solution:

class Codec:

def serialize(self, root):

"""Encodes a tree to a single string.

:type root: TreeNode

:rtype: str

"""

data = ''

if not root:

return data

stack = [root]

while stack:

node = stack.pop()

data += str(node.val) + ','

if node.right:

stack.append(node.right)

if node.left:

stack.append(node.left)

return data[:-1]

# O(nlogn)

def deserialize(self, data):

"""Decodes your encoded data to tree.

:type data: str

:rtype: TreeNode

"""

if not data or len(data) == 0:

return None

values = data.split(',')

n = len(values)

root = TreeNode(int(values[0]))

for i in range(1, n):

val = int(values[i])

node = TreeNode(val)

self.insertBST(root, node)

return root

def insertBST(self, root, node):

if not root:

root = node

if root.val > node.val:

if not root.left:

root.left = node

else:

self.insertBST(root.left, node)

else:

if not root.right:

root.right = node

else:

self.insertBST(root.right, node)

**297-Serialize and Deserialize Binary Tree**

Serialization is the process of converting a data structure or object into a sequence of bits so that it can be stored in a file or memory buffer, or transmitted across a network connection link to be reconstructed later in the same or another computer environment.

Design an algorithm to serialize and deserialize a binary tree. There is no restriction on how your serialization/deserialization algorithm should work. You just need to ensure that a binary tree can be serialized to a string and this string can be deserialized to the original tree structure.

Example:

You may serialize the following tree:

1

/ \

2 3

/ \

4 5

as "[1,2,3,null,null,4,5]"

Solution:

class Codec:

def serialize(self, root):

"""Encodes a tree to a single string.

:type root: TreeNode

:rtype: str

"""

if not root:

return ''

can = []

queue = [root]

while queue:

node = queue.pop(0)

if not node:

can.append('NULL')

else:

can.append(str(node.val))

queue.append(node.left)

queue.append(node.right)

return '.'.join(can)

def deserialize(self, data):

"""Decodes your encoded data to tree.

:type data: str

:rtype: TreeNode

"""

if not data:

return None

values = data.split('.')

n = len(values)

nodes = [None for i in range(n)]

nums = [0 for i in range(n)]

for i in range(1, n):

nums[i] = nums[i - 1]

if values[i] == 'NULL':

nums[i] += 1

for i in range(n):

if values[i] != 'NULL':

nodes[i] = TreeNode(int(values[i]))

for i in range(n):

if nodes[i]:

k = nums[i]

l = 2 \* (i - k) + 1

r = 2 \* (i - k) + 2

nodes[i].left = nodes[l]

nodes[i].right = nodes[r]

return nodes[0]

**298-Binary Tree Longest Consecutive Sequence**

Given a binary tree, find the length of the longest consecutive sequence path.

The path refers to any sequence of nodes from some starting node to any node in the tree along the parent-child connections. The longest consecutive path need to be from parent to child (cannot be the reverse).

Example 1:

Input:

1

\

3

/ \

2 4

\

5

Output: 3

Explanation: Longest consecutive sequence path is 3-4-5, so return 3.

Example 2:

Input:

2

\

3

/

2

/

1

Output: 2

Explanation: Longest consecutive sequence path is 2-3, not 3-2-1, so return 2.

Solution:

class Solution(object):

def longestConsecutive(self, root):

"""

:type root: TreeNode

:rtype: int

"""

if not root:

return 0

maxlen = [1]

curlen = 1

self.helper(root, maxlen, curlen)

return maxlen[0]

def helper(self, node, maxlen, curlen):

if not node or (not node.left and not node.right):

return

curlen\_left = curlen

curlen\_right = curlen

if node.left and node.left.val - node.val == 1:

curlen\_left += 1

else:

curlen\_left = 1

if node.right and node.right.val - node.val == 1:

curlen\_right += 1

else:

curlen\_right = 1

maxlen[0] = max(maxlen[0], curlen\_left, curlen\_right)

self.helper(node.left, maxlen, curlen\_left)

self.helper(node.right, maxlen, curlen\_right)

**314-Binary Tree Vertical Order Traversal**

Given a binary tree, return the vertical order traversal of its nodes' values. (ie, from top to bottom, column by column).

If two nodes are in the same row and column, the order should be from left to right.

Examples 1:

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

3

/\

/ \

9 20

/\

/ \

15 7

Output:

[

[9],

[3,15],

[20],

[7]

]

Examples 2:

Input: [3,9,8,4,0,1,7]

3

/\

/ \

9 8

/\ /\

/ \/ \

4 01 7

Output:

[

[4],

[9],

[3,0,1],

[8],

[7]

]

Examples 3:

Input: [3,9,8,4,0,1,7,null,null,null,2,5] (0's right child is 2 and 1's left child is 5)

3

/\

/ \

9 8

/\ /\

/ \/ \

4 01 7

/\

/ \

5 2

Output:

[

[4],

[9,5],

[3,0,1],

[8,2],

[7]

]

Solution:

class Solution(object):

def verticalOrder(self, root):

"""

:type root: TreeNode

:rtype: List[List[int]]

"""

res = []

if not root:

return res

queueA = [root]

queueB = [0]

dic = {}

minIdx = 0

maxIdx = 0

while queueA:

idx = queueB.pop(0)

node = queueA.pop(0)

if idx in dic:

dic[idx].append(node.val)

else:

dic[idx] = [node.val]

if node.left:

queueA.append(node.left)

queueB.append(idx - 1)

minIdx = min(minIdx, idx - 1)

if node.right:

queueA.append(node.right)

queueB.append(idx + 1)

maxIdx = max(maxIdx, idx + 1)

for i in range(minIdx, maxIdx + 1):

res.append(dic[i])

return res

**331-Verify Preorder serialization of a binary tree**

One way to serialize a binary tree is to use pre-order traversal. When we encounter a non-null node,  
we record the node's value. If it is a null node, we record using a sentinel value such as #.

\_9\_

/ \

3 2

/ \ / \

4 1 # 6

/ \ / \ / \

# # # # # #

For example, the above binary tree can be serialized to the string "9,3,4,#,#,1,#,#,2,#,6,#,#", where # represents a null node.

Given a string of comma separated values, verify whether it is a correct preorder traversal serialization of a binary tree.  
Find an algorithm without reconstructing the tree.

Each comma separated value in the string must be either an integer or a character '#' representing null pointer.

You may assume that the input format is always valid, for example it could never contain two consecutive commas such as "1,,3".

Example 1:

Input: "9,3,4,#,#,1,#,#,2,#,6,#,#"

Output: true

Example 2:

Input: "1,#"

Output: false

Example 3:

Input: "9,#,#,1"

Output: false

Solution:

class Solution(object):

def isValidSerialization(self, preorder):

"""

:type preorder: str

:rtype: bool

"""

nodes = preorder.split(',')

d = -1

for n in nodes:

d += 1

if d > 0:

return False

if n != '#':

d -= 2

return d == 0

**333-Largest BST Subtree**

Given a binary tree, find the largest subtree which is a Binary Search Tree (BST), where largest means subtree with largest number of nodes in it.

Note:  
A subtree must include all of its descendants.

Example:

Input: [10,5,15,1,8,null,7]

10

/ \

5 15

/ \ \

1 8 7

Output: 3

Explanation:

The Largest BST Subtree in this case

5

/ \

1 8

The return value is the subtree's size, which is 3.

Solution:

class Solution(object):

def \_\_init\_\_(self):

self.num = 0

self.inf = (1 << 32) - 1

def largestBSTSubtree(self, root):

"""

:type root: TreeNode

:rtype: int

"""

if not root:

return 0

self.helper(root)

return self.num

def helper(self, node):

if not node:

return {'size': 0, 'max': (-1) \* self.inf, 'min': self.inf}

left = self.helper(node.left)

right = self.helper(node.right)

if left['size'] == -1 or right['size'] == -1 or node.val >= right['min'] or node.val <= left['max']:

# That means this tree is not a BST

return {'size': -1, 'min': 0, 'max': 0}

# If this is a valid BST, update the max

size = left['size'] + right['size'] + 1

self.num = max(self.num, size)

minVal = min(node.val, left['min'])

maxVal = max(node.val, right['max'])

return {'size': size, 'min': minVal, 'max': maxVal}

**404-Sum of Left Leaves**

Find the sum of all left leaves in a given binary tree.

Example:

3

/ \

9 20

/ \

15 7

There are two left leaves in the binary tree, with values 9 and 15 respectively. Return 24.

Solution:

class Solution(object):

def sumOfLeftLeaves(self, root):

"""

:type root: TreeNode

:rtype: int

"""

if not root:

return 0

queue = [root]

res = 0

while queue:

node = queue.pop(0)

if node.left:

if not node.left.left and not node.left.right:

res += node.left.val

queue.append(node.left)

if node.right:

queue.append(node.right)

return res

**426-Convert Binary Search Tree to Sorted Doubly Linked List**

[Problem description](https://leetcode.com/problems/convert-binary-search-tree-to-sorted-doubly-linked-list/description/)

"""

# Definition for a Node.

class Node(object):

def \_\_init\_\_(self, val, left, right):

self.val = val

self.left = left

self.right = right

"""

class Solution(object):

def treeToDoublyList(self, root):

"""

:type root: Node

:rtype: Node

"""

if not root:

return None

dummy = Node(0, None, None)

prev = dummy

cur = root

stack = []

while True:

if cur:

stack.append(cur)

cur = cur.left

else:

if not stack:

break

node = stack.pop()

prev.right = node

node.left = prev

prev = node

cur = node.right

prev.right = dummy.right

dummy.right.left = prev

return dummy.right

**572-Subtree of Another Tree**

Given two non-empty binary trees s and t, check whether tree t has exactly the same structure and node values with a subtree of s. A subtree of s is a tree consists of a node in s and all of this node's descendants. The tree s could also be considered as a subtree of itself.

Example 1:  
Given tree s:

3

/ \

4 5

/ \

1 2

Given tree t:

4

/ \

1 2

Return true, because t has the same structure and node values with a subtree of s.

Example 2:

Given tree s:

3

/ \

4 5

/ \

1 2

/

0

Given tree t:

4

/ \

1 2

Return false.

Solution:

class Solution(object):

def isSubtree(self, s, t):

"""

:type s: TreeNode

:type t: TreeNode

:rtype: bool

"""

if s and not t or not s and t:

return False

if self.sameTree(s, t):

return True

return self.isSubtree(s.left, t) or self.isSubtree(s.right, t)

def sameTree(self, p, q):

if not p and not q:

return True

if not p and q or p and not q or p.val != q.val:

return False

return self.sameTree(p.left, q.left) and self.sameTree(p.right, q.right)

**617-Merge two binary trees**

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.

Example 1:

Input:

Tree 1 Tree 2

1 2

/ \ / \

3 2 1 3

/ \ \

5 4 7

Output:

Merged tree:

3

/ \

4 5

/ \ \

5 4 7

Solution:

class Solution(object):

def mergeTrees(self, t1, t2):

"""

:type t1: TreeNode

:type t2: TreeNode

:rtype: TreeNode

"""

if not t1 and not t2:

return None

n1 = n2 = 0

left1 = left2 = None

right1 = right2 = None

if t1:

n1 = t1.val

left1 = t1.left

right1 = t1.right

if t2:

n2 = t2.val

left2 = t2.left

right2 = t2.right

root = TreeNode(n1 + n2)

root.left = self.mergeTrees(left1, left2)

root.right = self.mergeTrees(right1, right2)

return root

**654-Maximum Binary Tree**

Given an integer array with no duplicates. A maximum tree building on this array is defined as follow:

The root is the maximum number in the array.  
The left subtree is the maximum tree constructed from left part subarray divided by the maximum number.  
The right subtree is the maximum tree constructed from right part subarray divided by the maximum number.  
Construct the maximum tree by the given array and output the root node of this tree.

Example 1:

Input: [3,2,1,6,0,5]

Output: return the tree root node representing the following tree:

6

/ \

3 5

\ /

2 0

\

1

Solution1: brute dorce recursion

class Solution(object):

def constructMaximumBinaryTree(self, nums):

"""

:type nums: List[int]

:rtype: TreeNode

"""

if not nums:

return None

if len(nums) == 1:

return TreeNode(nums[0])

l, r = 0, len(nums) - 1

return self.helper(nums, l, r)

def helper(self, nums, l, r):

maxNum = max(nums[l : r + 1])

idx = nums.index(maxNum)

root = TreeNode(maxNum)

left, right = None, None

if idx != l :

left = self.helper(nums, l, idx - 1)

if idx != r:

right = self.helper(nums, idx + 1, r)

root.left = left

root.right = right

return root

Solution2: 大神的解法

class Solution(object):

def constructMaximumBinaryTree(self, nums):

"""

:type nums: List[int]

:rtype: TreeNode

"""

n = len(nums)

stack = []

for i in range(n):

cur = TreeNode(nums[i])

while stack and stack[-1].val < nums[i]:

cur.left = stack.pop()

if stack:

stack[-1].right = cur

stack.append(cur)

if not stack:

return None

return stack[0]

**671-Second Minimum Node In a Binary Tree**

Given a non-empty special binary tree consisting of nodes with the non-negative value,  
where each node in this tree has exactly two or zero sub-node.  
If the node has two sub-nodes, then this node's value is the smaller value among its two sub-nodes.

Given such a binary tree, you need to output the second minimum value in the set made of all the nodes' value in the whole tree.

If no such second minimum value exists, output -1 instead.

Example 1:

Input:

2

/ \

2 5

/ \

5 7

Output: 5

Explanation: The smallest value is 2, the second smallest value is 5.

Example 2:

Input:

2

/ \

2 2

Output: -1

Explanation: The smallest value is 2, but there isn't any second smallest value.

Solution:

class Solution(object):

def findSecondMinimumValue(self, root):

"""

:type root: TreeNode

:rtype: int

"""

queue = [root]

firstMin = root.val

inf = (1 << 32) - 1

secondMin = inf

while queue:

node = queue.pop(0)

if node.val > firstMin and node.val <= secondMin:

secondMin = node.val

continue

if node.left:

queue.append(node.left)

if node.right:

queue.append(node.right)

return secondMin if secondMin < inf else -1

# Leetcode总结10: 图论相关题目以及BFS专题

### connected components

利用DFS找出图的连通分量数目

Example:

class Component(object):

# take egdes list and construct a graph using adjacency lsit

def \_\_init\_\_(self, edges, n, directed):

self.graph = { i : set() for i in range(n) }

self.count = 0

self.n = n

for u, v in edges:

self.graph[u].add(v)

if not directed:

self.graph[v].add(u)

def components(self):

visited = [False for i in range(self.n)]

for n in self.graph:

if not visited[n]:

self.components\_aux(n, visited)

self.count += 1

return self.count

# DFS

def components\_aux(self, u, visited):

visited[u] = True

lst = self.graph[u]

for n in lst:

if not visited[n]:

self.components\_aux(n, visited)

edges = [

[0, 1], [0, 2], [1, 4], [1, 3],

[4, 3], [3, 5], [5, 6], [5, 7],

[6, 7], [8, 9], [9, 10]]

n = 11

com = Component(edges, n, False)

print com.components()

### Find path

Looks for a path from start to end

class GraphPath(object):

def \_\_init\_\_(self, edges, n, directed):

self.graph = { i : set() for i in range(n) }

self.count = 0

self.n = n

for u, v in edges:

self.graph[u].add(v)

if not directed:

self.graph[v].add(u)

def path(self, start, end):

assert start in self.graph and end in self.graph

path = [-1 for i in range(self.n)]

visited = [False for i in range(self.n)]

self.path\_aux(start, visited, path)

stack = []

res = []

p = end

while p != -1:

stack.append(p)

p = path[p]

if len(stack) == -1 and stack[0] == end:

return [] # no path

while stack:

res.append(stack.pop())

return res

def path\_aux(self, u, visited, path):

visited[u] = True

lst = self.graph[u]

for n in lst:

if not visited[n]:

path[n] = u

self.path\_aux(n, visited, path)

edges = [

[0, 1], [0, 2], [1, 4], [1, 3],

[4, 3], [3, 5], [5, 6], [5, 7],

[6, 7], [8, 9], [9, 10]]

n = 11

g = GraphPath(edges, 11, False)

print g.path(0, 5)

### Un-weighted graph

#### shortest path - BFS

# shortest path, unweighted

class ShortestPathUW(object):

def \_\_init\_\_(self, edges, n, directed):

self.graph = { i : set() for i in range(n) }

self.n = n

for u, v in edges:

self.graph[u].add(v)

if not directed:

self.graph[v].add(u)

def shortestPath(self, start, end):

visited = [False for i in range(self.n)]

path = [-1 for i in range(self.n)]

dist = [-1 for i in range(self.n)]

queue = []

self.shortestPath\_aux(start, visited, path, dist, queue)

if dist[end] == -1:

return [], -1 # no path

stack = []

res = []

p = end

while p != -1:

stack.append(p)

p = path[p]

while stack:

res.append(stack.pop())

return res, dist[end]

# BFS iteration

def shortestPath\_aux(self, start, visited, path, dist, queue):

queue.append(start)

visited[start] = True

dist[start] = 0

while queue:

v = queue.pop()

lst = self.graph[v]

for n in lst:

if not visited[n]:

queue.append(n)

visited[n] = True

path[n] = v

dist[n] = dist[v] + 1

### 有向带权图最短路径 - Dijstra算法Python实现

import heapq

import collections

class Dijskastra(object):

def shortestPath\_dijkstra(self, n, edges, start, end):

graph = collections.defaultdict(dict)

inf = (1 << 32) - 1

dist = [inf for i in range(n)]

dist[0] = 0

comeFrom = [-1 for i in range(n)]

for u, v, weight in edges:

graph[u][v] = weight

queue = [(start, 0)]

while queue:

u, weight = heapq.heappop(queue)

for v in graph[u]:

if weight + graph[u][v] < dist[v]:

dist[v] = weight + graph[u][v]

comeFrom[v] = u

heapq.heappush(queue, (v, dist[v]))

# 如果不需要求路径信息, 代码可以更**简**化

def shortestPath\_dijkstra\_v2(self, n, edges, start, end):

graph = collections.defaultdict(dict)

inf = (1 << 32) - 1

for u, v, weight in edges:

graph[u][v] = weight

queue = [(start, 0)]

while queue:

u, weight = heapq.heappop(queue)

if u == end:

return weight

for v in graph[u]:

heapq.heappush(queue, (v, weight + graph[u][v]))

return -1

def test(self):

edges = [

(0, 1, 5), (0, 2, 2), (0, 3, 6), (1, 4, 1),

(2, 4, 5), (2, 3, 3), (2, 1, 1), (3, 4, 1),

(3, 7, 4), (4, 6, 6), (4, 5, 1), (5, 6, 1),

(7, 5, 2)]

n = 8

src = 1

end = 6

shortestDist, path = self.shortestPath\_dijkstra(n, edges, src, end)

print shortestDist

print path

edges = [

Edge(0, 1, 5), Edge(0, 2, 2), Edge(0, 3, 6), Edge(1, 4, 1),

Edge(2, 4, 5), Edge(2, 3, 3), Edge(2, 1, 1), Edge(3, 4, 1),

Edge(3, 7, 4), Edge(4, 6, 6), Edge(4, 5, 1), Edge(5, 6, 1),

Edge(7, 5, 2)]

n = 8

g = WeightedGraph(edges, n, True)

g.printGraph()

print g.shortestDist\_Dijkstra\_V1(0, 6)

### 279-Perfect Square

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

Example 1:

Input: n = 12

Output: 3

Explanation: 12 = 4 + 4 + 4.

Example 2:

Input: n = 13

Output: 2

Explanation: 13 = 4 + 9.

Solution 1: BFS 和 Coin change 同样的原理

class Solution(object):

def numSquares(self, n):

"""

:type n: int

:rtype: int

"""

queue = [0]

visited = [False] \* n

cnt = 0

while queue:

length = len(queue)

for i in range(length):

num = queue.pop(0)

k = 1

tmp = 0

while k \*\* 2 <= n:

tmp = num + k \*\* 2

if tmp == n:

return cnt + 1

if tmp < n and not visited[tmp]: # 剪枝

queue.append(tmp)

visited[tmp] = True

k += 1

cnt += 1

return cnt

Solution2: 动态规划 Python 代码 4812ms, 比剪枝BFS慢得多.

class Solution(object):

def numSquares(self, n):

"""

:type n: int

:rtype: int

"""

\_LARGE = 1 << 31

dp = [ \_LARGE for i in range(n + 1)]

dp[0] = 0

for i in range(1, n + 1):

j = 1

while j \*\* 2 <= i:

dp[i] = min(dp[i], dp[i - j \*\* 2] + 1)

j += 1

return dp[-1]

Solution 3 : 通过动态规划的递推关系可以使用记忆化递归 - 和 dp 一样慢

class Solution(object):

def \_\_init\_\_(self):

self.large = 1 << 31

def numSquares(self, n):

"""

:type n: int

:rtype: int

"""

if n == 1:

return 1

memo = [-1] \* (n + 1)

return self.helper(n, memo)

def helper(self, n, memo):

# print n, memo[n]

if n == 1 or n == 0:

return n

if memo[n] != -1 and memo[n] != self.large:

return memo[n]

k = 1

res = self.large

while k \*\* 2 <= n:

res = min(res, 1 + self.helper(n - k \*\* 2, memo))

k += 1

memo[n] = res

return res

### 127-Word Ladder

Given two words (beginWord and endWord), and a dictionary's word list,  
find the length of shortest transformation sequence from beginWord to endWord, such that:

Only one letter can be changed at a time.  
Each transformed word must exist in the word list. Note that beginWord is not a transformed word.  
Note:

Return 0 if there is no such transformation sequence.  
All words have the same length.  
All words contain only lowercase alphabetic characters.  
You may assume no duplicates in the word list.  
You may assume beginWord and endWord are non-empty and are not the same.  
Example 1:

Input:

beginWord = "hit",

endWord = "cog",

wordList = ["hot","dot","dog","lot","log","cog"]

Output: 5

Explanation: As one shortest transformation is "hit" -> "hot" -> "dot" -> "dog" -> "cog",

return its length 5.

Example 2:

Input:

beginWord = "hit"

endWord = "cog"

wordList = ["hot","dot","dog","lot","log"]

Output: 0

Explanation: The endWord "cog" is not in wordList, therefore no possible transformation.

Solution1:   
  
比较直观但是较麻烦的的办法, 直接使用邻接表构建一幅图, 相差一个字母的两个单词组成一个边. 然后对图进行广度优先遍历找出两个节点的最短路径  
  
这样做提交会超时, 但是很直观, 因此也把代码写出来.

class Solution(object):

def ladderLength(self, beginWord, endWord, wordList):

"""

:type beginWord: str

:type endWord: str

:type wordList: List[str]

:rtype: int

"""

graph = {}

wordList.append(beginWord)

n = len(wordList)

for w in wordList:

graph[w] = set()

for i in range(n):

wi = wordList[i]

for j in range(n):

if i == j:

continue

wj = wordList[j]

if self.differByOne(wi, wj):

graph[wi].add(wj)

graph[wj].add(wi)

# print graph

queue = [beginWord]

visited = set()

visited.add(beginWord)

layer = 1

while queue:

length = len(queue)

for i in range(length):

w = queue.pop(0)

neighbors = graph[w]

for s in neighbors:

if s == endWord:

return layer + 1

if s not in visited:

visited.add(s)

queue.append(s)

layer += 1

return 0

def differByOne(self, s1, s2):

assert len(s1) == len(s2)

cnt = 0

i = 0

n = len(s1)

while i < n:

if s1[i] != s2[i]:

cnt += 1

if cnt > 1:

return False

i += 1

return True

Solution 2:   
  
利用 BFS 其实并不一定要构建一幅图, 就如 coin change 以及 perfect square一样  
直接使用队列也就可以实现了, 这样省去了构造图需要的计算. Python提交通过

class Solution(object):

def ladderLength(self, beginWord, endWord, wordList):

"""

:type beginWord: str

:type endWord: str

:type wordList: List[str]

:rtype: int

"""

wordList = set(wordList)

visited = set()

visited.add(beginWord)

if endWord not in wordList:

return 0

dist = 1

queue = [beginWord]

while queue:

length = len(queue)

for i in range(length):

w = queue.pop(0)

if w == endWord:

return dist

for i in range(len(w)):

chars = list(w)

for j in range(ord('a'), ord('z') + 1):

chars[i] = chr(j)

tmp = ''.join(chars)

if tmp in wordList:

wordList.remove(tmp)

queue.append(tmp)

dist += 1

return 0

### 322-Coin Change

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.

Example 1:

Input: coins = [1, 2, 5], amount = 11

Output: 3

Explanation: 11 = 5 + 5 + 1

Example 2:

Input: coins = [2], amount = 3

Output: -1

Solution: BFS - 和 Perfect Square 一样的原理

class Solution(object):

def coinChange(self, coins, amount):

"""

:type coins: List[int]

:type amount: int

:rtype: int

"""

if amount == 0:

return 0

queue = [0]

visited = [False for i in range(amount + 1)]

visited[0] = True

cnt = 0

while queue:

length = len(queue)

for i in range(length):

num = queue.pop(0)

for c in coins:

tmp = num + c

if tmp == amount:

return cnt + 1

if tmp < amount and not visited[tmp]:

visited[tmp] = True

queue.append(tmp)

cnt += 1

return -1

Solution2: DP

class Solution(object):

def coinChange(self, coins, amount):

"""

:type coins: List[int]

:type amount: int

:rtype: int

"""

if amount == 0:

return 0

n = len(coins)

inf = 1 << 31

dp = [inf for i in range(amount + 1)]

dp[0] = 0

for i in range(1, amount + 1):

for j in range(n):

c = coins[j]

if i - c < 0:

continue

dp[i] = min(dp[i], dp[i - c] + 1)

return dp[-1] if dp[-1] < inf else -1

### 301-Remove Invalid Parentheses

Remove the minimum number of invalid parentheses in order to make the input string valid. Return all possible results.

Note: The input string may contain letters other than the parentheses ( and ).

Example 1:

Input: "()())()"

Output: ["()()()", "(())()"]

Example 2:

Input: "(a)())()"

Output: ["(a)()()", "(a())()"]

Example 3:

Input: "')("

Output: [""]

Solution:

class Solution(object):

def removeInvalidParentheses(self, s):

"""

:type s: str

:rtype: List[str]

"""

queue = [s]

res = []

visited = set()

stop = False

while queue:

string = queue.pop(0)

if self.isValid(string):

res.append(string)

stop = True

if stop:

continue

for i in range(len(string)):

if string[i] == '(' or string[i] == ')':

temp = string[:i] + string[i + 1:]

if temp not in visited:

queue.append(temp)

visited.add(temp)

return res

def isValid(self, s) :

count = 0

for i, c in enumerate(s):

if c == '(' :

count += 1

elif c == ')' :

count -= 1

if count < 0 :

return False

return count == 0

### 433-Minimum Genetic Mutation

A gene string can be represented by an 8-character long string, with choices from "A", "C", "G", "T".

Suppose we need to investigate about a mutation (mutation from "start" to "end"), where ONE mutation is defined as ONE single character changed in the gene string.

For example, "AACCGGTT" -> "AACCGGTA" is 1 mutation.

Also, there is a given gene "bank", which records all the valid gene mutations. A gene must be in the bank to make it a valid gene string.

Now, given 3 things - start, end, bank, your task is to determine what is the minimum number of mutations needed to mutate from "start" to "end". If there is no such a mutation, return -1.

Example 1:

start: "AACCGGTT"

end: "AACCGGTA"

bank: ["AACCGGTA"]

return: 1

Example 2:

start: "AACCGGTT"

end: "AAACGGTA"

bank: ["AACCGGTA", "AACCGCTA", "AAACGGTA"]

return: 2

Example 3:

start: "AAAAACCC"

end: "AACCCCCC"

bank: ["AAAACCCC", "AAACCCCC", "AACCCCCC"]

return: 3

Solution: 和 word ladder一样

class Solution(object):

def minMutation(self, start, end, bank):

"""

:type start: str

:type end: str

:type bank: List[str]

:rtype: int

"""

bank = set(bank)

if end not in bank:

return -1

queue = [start]

steps = 0

choices = ['A', 'C', 'G', 'T']

cnt = 0

while queue:

length = len(queue)

for i in range(length):

g = queue.pop(0)

if g == end:

return cnt

for j in range(len(g)):

for ch in choices:

newG = g[:j] + ch + g[j + 1:]

if newG in bank:

queue.append(newG)

bank.remove(newG)

cnt += 1

return -1

### 126-Word Ladder II

Complicated see sorce **code** in Solution folder

### 490-The Maze

[Problem description](https://leetcode.com/problems/the-maze/description/)

Solution:

class Position(object):

def \_\_init\_\_(self, x, y):

self.x = x

self.y = y

class Solution(object):

def hasPath(self, maze, start, destination):

"""

:type maze: List[List[int]]

:type start: List[int]

:type destination: List[int]

:rtype: bool

"""

queue = []

row = len(maze)

col = len(maze[0])

visited = [[False for j in range(col)] for i in range(row)]

starPos = Position(start[0], start[1])

dirs = [[1, 0], [-1, 0], [0, 1], [0, -1]]

queue = [starPos]

while queue:

pos = queue.pop(0)

for d in dirs:

r = pos.x

c = pos.y

while r >= 0 and r < row and c >= 0 and c < col and maze[r][c] == 0:

r += d[0]

c += d[1]

r -= d[0]

c -= d[1]

if visited[r][c]:

continue

visited[r][c] = True

if r == destination[0] and c == destination[1]:

return True

queue.append(Position(r, c))

return False

### 505-The Maze II

[Problem description](https://leetcode.com/problems/the-maze-ii/description/)

Solution:

import heapq

class Position(object):

def \_\_init\_\_(self, r, c, dist):

self.r = r

self.c = c

self.dist = dist

def \_\_cmp\_\_(self, other):

return self.dist - other.dist

class Solution(object):

def shortestDistance(self, maze, start, destination):

"""

:type maze: List[List[int]]

:type start: List[int]

:type destination: List[int]

:rtype: int

"""

row = len(maze)

col = len(maze[0])

MAX\_INT = (1 << 32) - 1

distance = [[MAX\_INT for j in range(col)] for i in range(row)]

queue = [Position(start[0], start[1], 0)]

dirs = [[1, 0], [-1, 0], [0, 1], [0, -1]]

while queue:

pos = heapq.heappop(queue)

if distance[pos.r][pos.c] <= pos.dist:

continue

distance[pos.r][pos.c] = pos.dist

for d in dirs:

r = pos.r

c = pos.c

dist = pos.dist

while r >= 0 and c >= 0 and r < row and c < col and maze[r][c] == 0:

r += d[0]

c += d[1]

dist += 1

r -= d[0]

c -= d[1]

dist -= 1

if dist != pos.dist:

heapq.heappush(queue, Position(r, c, dist))

res = distance[destination[0]][destination[1]]

return res if res < MAX\_INT else -1

### 787-Cheapest Flights Within K Stops

[Problem description](https://leetcode.com/problems/cheapest-flights-within-k-stops/description/)

Solution: Dijkstra算法的**简**化

import heapq

import collections

class Triple(object):

def \_\_init\_\_(self, u, weight, stop):

self.u = u

self.weight = weight

self.stop = stop

def \_\_cmp\_\_(self, other):

return self.weight - other.weight

class Solution(object):

def findCheapestPrice(self, n, flights, src, dst, k):

"""

:type n: int

:type flights: List[List[int]]

:type src: int

:type dst: int

:type K: int

:rtype: int

"""

graph = collections.defaultdict(dict)

for u, v, w in flights:

graph[u][v] = w

queue = [Triple(src, 0, k + 1)]

while queue:

triple = heapq.heappop(queue)

u, weight, stop = triple.u, triple.weight, triple.stop

if u == dst:

return weight

if stop > 0:

for v in graph[u]:

heapq.heappush(queue, Triple(v, weight + graph[u][v], stop - 1))

return -1

### 133-Clone Graph

Given the head of a graph, return a deep copy (clone) of the graph. Each node in the graph contains a  
label (int) and a list (List[UndirectedGraphNode]) of its neighbors.  
There is an edge between the given node and each of the nodes in its neighbors.

# Definition for a undirected graph node

# class UndirectedGraphNode:

# def \_\_init\_\_(self, x):

# self.label = x

# self.neighbors = []

class Solution:

# @param node, a undirected graph node

# @return a undirected graph node

def cloneGraph(self, node):

if not node:

return None

cloneHead = UndirectedGraphNode(node.label)

queue = [node]

dic = {node: cloneHead}

while queue:

cur = queue.pop(0)

for n in cur.neighbors:

if n not in dic:

nCopy = UndirectedGraphNode(n.label)

dic[n] = nCopy

dic[cur].neighbors.append(nCopy)

queue.append(n)

else:

dic[cur].neighbors.append(dic[n])

return dic[node]

### 207-Course Schedule

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?

Example 1:

Input: 2, [[1,0]]

Output: true

Explanation: There are a total of 2 courses to take.

To take course 1 you should have finished course 0. So it is possible.

Example 2:

Input: 2, [[1,0],[0,1]]

Output: false

Explanation: There are a total of 2 courses to take.

To take course 1 you should have finished course 0, and to take course 0 you should

also have finished course 1. So it is impossible.

Solution 1: 利用BFS找出拓扑排序

class Solution(object):

def canFinish(self, numCourses, prerequisites):

"""

:type numCourses: int

:type prerequisites: List[List[int]]

:rtype: bool

"""

graph = {i : set() for i in range(numCourses)}

parent = {i : 0 for i in range(numCourses)}

for u, v in prerequisites:

graph[u].add(v)

parent[v] += 1

queue = []

res = []

for u in parent:

if parent[u] == 0:

queue.append(u)

while queue:

u = queue.pop(0)

res.insert(0, u)

for v in graph[u]:

parent[v] -= 1

if parent[v] == 0:

queue.append(v)

if len(res) != numCourses:

return False

return True

def test(self):

prerequisites = [[1,0],[2,0],[3,1],[3,2]]

# prerequisites = [[0,1],[2,0],[1,3],[3,2]]

numCourses = 4

print self.canFinish(numCourses, prerequisites)

Solution2: 一开始想到的是最笨的办法, 利用三个集合来检查有向图中是否存在环. 如果存在, 则无法找到拓扑排序.

from random import sample

class Solution(object):

def canFinish(self, numCourses, prerequisites):

return not self.hasCycle(numCourses, prerequisites)

def constructGraph(self, num, edgeList):

graph = {x: set() for x in range(num)}

for u, v in edgeList:

graph[u].add(v)

return graph

def topoSort(self, num, edgeList):

graph = self.constructGraph(num, edgeList)

stack = []

visited = set()

for node in graph:

if node in visited:

continue

self.helper(node, stack, visited, graph)

return stack

def helper(self, node, stack, visited, graph):

visited.add(node)

for n in graph[node]:

if n in visited:

continue

self.helper(n, stack, visited, graph)

stack.insert(0, node)

def hasCycle(self, num, edgeList):

graph = self.constructGraph(num, edgeList)

whiteSet = set()

graySet = set()

blackSet = set()

for node in graph:

whiteSet.add(node)

while len(whiteSet) > 0:

curNode = sample(whiteSet, 1)[0]

if self.dfs(curNode, whiteSet, graySet, blackSet, graph) :

return True

return False

# Time complexity is O(v + e), where v is number of vertices, and e is number of edges

# so it is O(numCourses + len(prerequisites))

def dfs(self, node, whiteSet, graySet, blackSet, graph):

self.moveNode(node, whiteSet, graySet)

for n in graph[node]:

if n in blackSet:

continue

if n in graySet:

return True

if self.dfs(n, whiteSet, graySet, blackSet, graph):

return True

self.moveNode(node, graySet, blackSet)

return False

def moveNode(self, node, fromSet, toSet):

fromSet.remove(node)

toSet.add(node)

# test case

prerequisites = [[1,0],[2,0],[3,1],[3,2]]

# prerequisites = [[0,1],[2,0],[1,3],[3,2]]

numCourses = 4

### 210-Course Schedule II

same as 207

class Solution(object):

def findOrder(self, numCourses, prerequisites):

"""

:type numCourses: int

:type prerequisites: List[List[int]]

:rtype: List[int]

"""

graph = {i : set() for i in range(numCourses)}

parent = {i : 0 for i in range(numCourses)}

for u, v in prerequisites:

graph[u].add(v)

parent[v] += 1

# print graph

# print parent

queue = []

res = []

for u in parent:

if parent[u] == 0:

queue.append(u)

while queue:

u = queue.pop(0)

res.append(u)

for v in graph[u]:

parent[v] -= 1

if parent[v] == 0:

queue.append(v)

if len(res) != numCourses:

return []

res.reverse()

return res

### 269-Alien Dictionary

There is a new alien language which uses the latin alphabet. However, the order among letters are unknown to you. You receive a list of non-empty words from the dictionary, where words are sorted lexicographically by the rules of this new language. Derive the order of letters in this language.

Example 1:

Input:

[

"wrt",

"wrf",

"er",

"ett",

"rftt"

]

Output: "wertf"

Example 2:

Input:

[

"z",

"x"

]

Output: "zx"

Example 3:

Input:

[

"z",

"x",

"z"

]

Output: ""

Explanation: The order is invalid, so return "".

Solution:  
和course schedule一样, 利用广度优先搜索找出拓扑排序. 但这题首先要想到是拓扑排序的问题有点难.

class Solution(object):

def alienOrder(self, words):

"""

:type words: List[str]

:rtype: str

"""

if not words:

return ""

n = len(words)

graph = {}

numOfParents = {}

for w in words:

for c in w:

numOfParents[c] = 0

graph[c] = set()

for i in range(n - 1):

w1, w2 = words[i], words[i + 1]

length = min(len(w1), len(w2))

for j in range(length):

c1, c2 = w1[j], w2[j]

if c1 != c2:

if c2 not in graph[c1]:

graph[c1].add(c2)

numOfParents[c2] += 1

break

print graph

print numOfParents

queue = []

res = []

for node in numOfParents:

if numOfParents[node] == 0:

queue.append(node)

while queue:

node = queue.pop(0)

res.append(node)

neighbors = graph[node]

for charNode in neighbors:

numOfParents[charNode] -= 1

if numOfParents[charNode] == 0:

queue.append(charNode)

if len(res) != len(graph):

return ""

return "".join(res)

### 277-Find the Celebrity

Suppose you are at a party with n people (labeled from 0 to n - 1) and among them, there may exist one celebrity. The definition of a celebrity is that all the other n - 1 people know him/her but he/she does not know any of them.

Now you want to find out who the celebrity is or verify that there is not one. The only thing you are allowed to do is to ask questions like: "Hi, A. Do you know B?" to get information of whether A knows B. You need to find out the celebrity (or verify there is not one) by asking as few questions as possible (in the asymptotic sense).

You are given a helper function bool knows(a, b) which tells you whether A knows B. Implement a function int findCelebrity(n), your function should minimize the number of calls to knows.

Note: There will be exactly one celebrity if he/she is in the party. Return the celebrity's label if there is a celebrity in the party. If there is no celebrity, return -1.

Solution:

if a knows b, a must not be the celebrity  
if a does not knows b, b must not be the celebrity

# The knows API is already defined for you.

# @param a, person a

# @param b, person b

# @return a boolean, whether a knows b

# def knows(a, b):

class Solution(object):

def findCelebrity(self, n):

"""

:type n: int

:rtype: int

"""

candidate = 0

for i in range(1, n):

# if a knows b, a must not be the celebrity

if knows(candidate, i):

candidate = i

for i in range(n):

if i != candidate:

# if a does not knows b, b must not be the celebrity

if not knows(i, candidate) or knows(candidate, i):

return -1

return candidate

### 332-Reconstruct Itinerary

Given a list of airline tickets represented by pairs of departure and arrival airports [from, to], reconstruct the itinerary in order. All of the tickets belong to a man who departs from JFK. Thus, the itinerary must begin with JFK.

Note:

If there are multiple valid itineraries, you should return the itinerary that has the smallest lexical order when read as a single string. For example, the itinerary ["JFK", "LGA"] has a smaller lexical order than ["JFK", "LGB"].  
All airports are represented by three capital letters (IATA **code**).  
You may assume all tickets form at least one valid itinerary.  
Example 1:

Input: [["MUC", "LHR"], ["JFK", "MUC"], ["SFO", "SJC"], ["LHR", "SFO"]]

Output: ["JFK", "MUC", "LHR", "SFO", "SJC"]

Example 2:

Input: [["JFK","SFO"],["JFK","ATL"],["SFO","ATL"],["ATL","JFK"],["ATL","SFO"]]

Output: ["JFK","ATL","JFK","SFO","ATL","SFO"]

Explanation: Another possible reconstruction is ["JFK","SFO","ATL","JFK","ATL","SFO"].

But it is larger in lexical order.

Solution:

import heapq

class Solution(object):

def findItinerary(self, tickets):

"""

:type tickets: List[List[str]]

:rtype: List[str]

"""

graph = {}

res = []

for u, v in tickets:

if u not in graph:

graph[u] = []

heapq.heappush(graph[u], v)

self.helper('JFK', res, graph)

return res

def helper(self, s, res, graph):

queue = graph.get(s)

while queue:

u = heapq.heappop(queue)

self.helper(u, res, graph)

res.insert(0, s)

# LeetCode总结11: Union Find

### Union Find 完整实现

# coding: utf-8

class UnionFind\_V1(object):

def \_\_init\_\_(self, n):

self.size = n

self.parent = [i for i in range(n)]

def findRoot(self, p):

if p > self.size - 1:

return None

while p != self.parent[p]:

p = parent[p]

return p

def union(self, p, q):

if p > self.size - 1 or q > self.size - 1:

return

rootP = self.findRoot(p)

rootQ = self.findRoot(q)

self.parent[rootQ] = rootP

def isConnected(self, p, q):

if p > self.size - 1 or q > self.size - 1:

return False

return self.findRoot(p) == self.findRoot(q)

"""

Union find optimized version

基于集合内元素个数的优化: 永远把元素少的那棵树的根节点连接到元素多的那棵树的根节点

"""

class UnionFind\_V2(object):

def \_\_init\_\_(self, n):

self.size = n

self.parent = [i for i in range(n)]

self.numOfElementInSet = [1 for i in range(n)]

def findRoot(self, p):

if p > self.size - 1:

return None

while p != self.parent[p]:

p = parent[p]

return p

def union(self, p, q):

if p > self.size - 1 or q > self.size - 1:

return

rootP = self.findRoot(p)

rootQ = self.findRoot(q)

if self.numOfElementInSet[rootQ] < self.numOfElementInSet[rootP]:

self.parent[rootQ] = rootP

self.numOfElementInSet[rootP] = self.numOfElementInSet[rootP] + self.numOfElementInSet[rootQ]

else:

self.parent[rootP] = rootQ

self.numOfElementInSet[rootQ] = self.numOfElementInSet[rootQ] + self.numOfElementInSet[rootP]

def isConnected(self, p, q):

if p > self.size - 1 or q > self.size - 1:

return False

return self.findRoot(p) == self.findRoot(q)

"""

基于层数的优化: 把层数少的那组集合(树)连接到层数多的那颗树

"""

class UnionFind\_V3(object):

def \_\_init\_\_(self, n):

self.size = n

self.parent = [i for i in range(n)]

self.hight = [1 for i in range(n)] # 表示以 i 为根的树的层数

def findRoot(self, p):

if p > self.size - 1:

return None

while p != self.parent[p]:

p = parent[p]

return p

def union(self, p, q):

if p > self.size - 1 or q > self.size - 1:

return

rootP = self.findRoot(p)

rootQ = self.findRoot(q)

if self.hight[rootQ] < self.hight[rootP]:

self.parent[rootQ] = rootP

else if self.hight[rootQ] > self.hight[rootP]:

self.parent[rootP] = rootQ

else: # the two tree has the same height

self.parent[rootP] = rootQ

self.height[rootQ] = self.height[rootQ] + 1

def isConnected(self, p, q):

if p > self.size - 1 or q > self.size - 1:

return False

return self.findRoot(p) == self.findRoot(q)

"""

最终优化: 路径压缩

"""

class UnionFind\_V4(object):

def \_\_init\_\_(self, n):

self.size = n

self.parent = [i for i in range(n)]

self.hight = [1 for i in range(n)] # 表示以 i 为根的树的层数

def findRoot(self, p):

if p > self.size - 1:

return None

while p != self.parent[p]:

# 路径压缩优化 版本1

parent[p] = parent[parent[p]]

p = parent[p]

return p

def findRoot2(self, p):

if p > self.size - 1:

return None

# 路径压缩优化 版本2

if p != parent[p]:

parent[p] = self.findRoot2(parent[p])

return parent[p]

def union(self, p, q):

if p > self.size - 1 or q > self.size - 1:

return

rootP = self.findRoot(p)

rootQ = self.findRoot(q)

if self.hight[rootQ] < self.hight[rootP]:

self.parent[rootQ] = rootP

else if self.hight[rootQ] > self.hight[rootP]:

self.parent[rootP] = rootQ

else: # the two tree has the same height

self.parent[rootP] = rootQ

self.height[rootQ] = self.height[rootQ] + 1

def isConnected(self, p, q):

if p > self.size - 1 or q > self.size - 1:

return False

return self.findRoot(p) == self.findRoot(q)

### 261-Graph Valid Tree

Given n nodes labeled from 0 to n-1 and a list of undirected edges (each edge is a pair of nodes), write a function to check whether these edges make up a valid tree.

Example 1:

Input: n = 5, and edges = [[0,1], [0,2], [0,3], [1,4]]

Output: true

Example 2:

Input: n = 5, and edges = [[0,1], [1,2], [2,3], [1,3], [1,4]]

Output: false

Note: you can assume that no duplicate edges will appear in edges. Since all edges are undirected, [0,1] is the same as [1,0] and thus will not appear together in edges.

Solution:

class Solution(object):

def validTree(self, n, edges):

"""

:type n: int

:type edges: List[List[int]]

:rtype: bool

"""

if len(edges) != n - 1:

return False

vals = [i for i in range(n)]

for p, q in edges:

rootP = self.findRoot(p, vals)

rootQ = self.findRoot(q, vals)

if rootP == rootQ:

return False

vals[rootQ] = rootP

return True

def findRoot(self, n, vals):

while True:

if vals[n] == n:

return n

n = vals[vals[n]]

return 0

### 305-Number of Islands II

A 2d grid map of m rows and n columns is initially filled with water. We may perform an addLand operation which turns the water at position (row, col) into a land. Given a list of positions to operate, count the number of islands after each addLand operation. 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.

Example:

Input: m = 3, n = 3, positions = [[0,0], [0,1], [1,2], [2,1]]

Output: [1,1,2,3]

Explanation:

Initially, the 2d grid grid is filled with water. (Assume 0 represents water and 1 represents land).

0 0 0

0 0 0

0 0 0

Operation #1: addLand(0, 0) turns the water at grid[0][0] into a land.

1 0 0

0 0 0 Number of islands = 1

0 0 0

Operation #2: addLand(0, 1) turns the water at grid[0][1] into a land.

1 1 0

0 0 0 Number of islands = 1

0 0 0

Operation #3: addLand(1, 2) turns the water at grid[1][2] into a land.

1 1 0

0 0 1 Number of islands = 2

0 0 0

Operation #4: addLand(2, 1) turns the water at grid[2][1] into a land.

1 1 0

0 0 1 Number of islands = 3

0 1 0

Follow up:

Can you do it in time complexity O(k log mn), where k is the length of the positions?

Solution: 大神的解法, 自己是真的想不出来.

class Solution(object):

def numIslands2(self, m, n, positions):

"""

:type m: int

:type n: int

:type positions: List[List[int]]

:rtype: List[int]

"""

res = []

roots = [-1 for i in range(m \* n)]

dirs = [[0, 1], [-1, 0], [0, -1], [1, 0]]

count = 0

for p in positions:

v = n \* p[0] + p[1]

roots[v] = v

count += 1

for d in dirs:

x, y = p[0] + d[0], p[1] + d[1]

q = n \* x + y

if x < 0 or x >= m or y < 0 or y >= n or roots[q] == -1:

continue

rootQ = self.find(roots, q)

if v != rootQ:

roots[v] = rootQ

v = rootQ

count -= 1

res.append(count)

return res

def find(self, roots, q):

while q != roots[q]:

roots[q] = roots[roots[q]]

q = roots[q]

return q

### 323-Number of Connected Components in an Undirected Graph

Given n nodes labeled from 0 to n - 1 and a list of undirected edges (each edge is a pair of nodes), write a function to find the number of connected components in an undirected graph.

Example 1:

Input: n = 5 and edges = [[0, 1], [1, 2], [3, 4]]

0 3

| |

1 --- 2 4

Output: 2

Example 2:

Input: n = 5 and edges = [[0, 1], [1, 2], [2, 3], [3, 4]]

0 4

| |

1 --- 2 --- 3

Output: 1

Note:  
You can assume that no duplicate edges will appear in edges. Since all edges are undirected, [0, 1] is the same as [1, 0] and thus will not appear together in edges.

Solution: 经典Union Find问题

class Solution(object):

def countComponents(self, n, edges):

"""

:type n: int

:type edges: List[List[int]]

:rtype: int

"""

roots = [i for i in range(n)]

count = n

for p, q in edges:

rp = self.find(roots, p)

rq = self.find(roots, q)

if rp != rq:

roots[rq] = roots[rp]

count -= 1

return count

def find(self, roots, v):

while v != roots[v]:

roots[v] = roots[roots[v]] # 路径压缩

v = roots[v]

return v

### 547-Friend Circles

There are N students in a class. Some of them are friends, while some are not. Their friendship is transitive in nature. For example, if A is a direct friend of B, and B is a direct friend of C, then A is an indirect friend of C. And we defined a friend circle is a group of students who are direct or indirect friends.

Given a N \* N matrix M representing the friend relationship between students in the class. If M[i][j] = 1, then the ith and jth students are direct friends with each other, otherwise not. And you have to output the total number of friend circles among all the students.

Example 1:

Input:

[[1,1,0],

[1,1,0],

[0,0,1]]

Output: 2

Explanation:The 0th and 1st students are direct friends, so they are in a friend circle.

The 2nd student himself is in a friend circle. So return 2.

Example 2:

Input:

[[1,1,0],

[1,1,1],

[0,1,1]]

Output: 1

Explanation:The 0th and 1st students are direct friends, the 1st and 2nd students are direct friends,

so the 0th and 2nd students are indirect friends. All of them are in the same friend circle, so return 1.

Note:

N is in range [1,200].

M[i][i] = 1 for all students.

If M[i][j] = 1, then M[j][i] = 1.

Solution:

class Solution(object):

def findCircleNum(self, M):

"""

:type M: List[List[int]]

:rtype: int

"""

n = len(M)

roots = [i for i in range(n)]

count = n

for i in range(0, n):

for j in range(i + 1, n):

if M[i][j] == 1:

rp = self.find(roots, i)

rq = self.find(roots, j)

if rp != rq:

roots[rq] = rp

count -= 1

return count

def find(self, roots, v):

while v != roots[v]:

roots[v] = roots[roots[v]]

v = roots[v]

return v

# LeetCode总结12: 动态规划与记忆化递归

## 1. 01背包类问题:

### Knapsack01 Problem

从经典的0-1背包问题开始  
一个背包, 容量为C. 现在有n中不同的物品, 编号为 0, 1, 2, ..., n - 1, 其中每一件物品的重量为w(i), 价格为 v(i). 问可以想这个背包中方那些物品, 是的在不超过背包容量的基础上, 总物品的价格最大.

递推方程:

对于i个物体, 最大价值有两种可能:

第一种是放弃第i个, 只去前面i - 1个物体的最大值.

第二种是选择放进第i个, 然后加上前面 i - 1 个物体的最大值. 由于放进了第i个物品, 那么背包的容量就要减少w(i), 剩下 c - w(i) 的容量

因此递推方程如下:

F(i, c) = max(F(i - 1, c), v(i) + F(i - 1, c - w(i)))

Solution: 忆化递归

class Knapsack(object):

def maxValue(self, weights, values, capacity):

n = len(weights)

memo = [[-1 for j in range(capacity + 1)] for i in range(n)]

return self.helper(0, weights, values, capacity, memo)

def helper(self, i, weights, values, capacity, memo):

if i >= len(weights):

return 0

if memo[i][capacity] > -1:

return memo[i][capacity]

a = self.helper(i + 1, weights, values, capacity, memo)

b = 0

if capacity - weights[i] > 0:

b = values[i] + self.helper(i + 1, weights, values, capacity - weights[i], memo)

res = max(a, b)

memo[i][capacity] = res

return res

def test(self):

weights = [5, 4, 7, 2, 6]

values = [12, 3, 10, 3, 6]

capacity = 15

# output 25

print self.maxValue(weights, values, capacity)

Solution2: DP

class Knapsack01\_v2(object):

def maxValue(self, weights, values, capacity):

n = len(values)

dp = [[0 for j in range(capacity + 1)] for i in range(n)]

for j in range(1, capacity + 1):

if j >= weights[0]:

dp[0][j] = values[0]

for i in range(1, n):

for j in range(1, capacity + 1):

a = dp[i - 1][j]

b = 0

if j - weights[i] > 0:

b = values[i] + dp[i - 1][j - weights[i]]

dp[i][j] = max(a, b)

return dp[n - 1][capacity]

def test(self):

weights = [5, 4, 7, 2, 6]

values = [12, 3, 10, 3, 6]

capacity = 15

# output 25

print self.maxValue(weights, values, capacity)

### 322-Coin Change

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.

Example 1:

Input: coins = [1, 2, 5], amount = 11

Output: 3

Explanation: 11 = 5 + 5 + 1

Example 2:

Input: coins = [2], amount = 3

Output: -1

Note:  
You may assume that you have an infinite number of each kind of coin.

这道题最直观和最优解法其实是BFS解法 在图论专题已给出. 这里探讨DP 和 记忆化递归解  
有点类似01背包问题, 定义 F(amount) : fewest number of coins that you need to make up amount.  
递推方程为:

F(amount) = min(F(amount - c) + 1, ... )

where c = 1, 2, 5 (given by problem)

Solution 1: 记忆化递归

class Solution(object):

def \_\_init\_\_(self):

self.inf = (1 << 32) - 1

def coinChange(self, coins, amount):

"""

:type coins: List[int]

:type amount: int

:rtype: int

"""

if amount == 0:

return 0

memo = [-1 for i in range(amount + 1)]

res = self.helper(amount, coins, memo)

return res if res < self.inf else -1

def helper(self, amount, coins, memo):

if amount == 0:

return 0

if memo[amount] > -1:

return memo[amount]

res = self.inf

for c in coins:

if amount - c >= 0:

res = min(res, self.helper(amount - c, coins, memo) + 1)

memo[amount] = res

return res

Solution2: DP

class Solution(object):

def coinChange(self, coins, amount):

"""

:type coins: List[int]

:type amount: int

:rtype: int

"""

if amount == 0:

return 0

inf = (1 << 32) - 1

dp = [inf for i in range(amount + 1)]

dp[0] = 0

for i in range(1, amount + 1):

for c in coins:

if i - c >= 0:

dp[i] = min(dp[i], dp[i - c] + 1)

return dp[amount] if dp[amount] < inf else -1

### 416-Partition Equal Subset Sum

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.  
Example 1:

Input: [1, 5, 11, 5]

Output: true

Explanation: The array can be partitioned as [1, 5, 5] and [11].

Example 2:

Input: [1, 2, 3, 5]

Output: false

Explanation: The array cannot be partitioned into equal sum subsets.

变相的01背包问题: 一个数组中找出一个子集, 使得他们能够填满容量为 sum / 2 的背包. 其中sum是整个数组的和.

递推方程: F(n, c) = F(n - 1, c) or F(n - 1, c - nums[i])

F(n, c) : n个物体能否找到一组子集填满容量为c的背包  
F(n - 1, c) : n - 1 个物体中能否找找到一个子集填满容量为c的背包  
F(n - 1, c - nums[i]) : n - 1 个物体中能否找到一个子集填满容量为 c - nums[n] 的背包 (如果能的话, 那么把第n个物体放进去就最终填满容量为c的背包)

Solution 1: 记忆化递归

class Solution(object):

def canPartition(self, nums):

"""

:type nums: List[int]

:rtype: bool

"""

capacity = sum(nums)

if capacity % 2 != 0:

return False

capacity /= 2

n = len(nums)

memo = [[None for j in range(capacity + 1)] for i in range(n)]

return self.helper(0, nums, capacity, memo)

def helper(self, i, nums, capacity, memo):

if i >= len(nums) or capacity < 0:

return False

if capacity == 0:

return True

if memo[i][capacity] is not None:

return memo[i][capacity]

res = self.helper(i + 1, nums, capacity, memo) or self.helper(i + 1, nums, capacity - nums[i], memo)

memo[i][capacity] = res

return res

Solution2: DP解法

class Solution(object):

def canPartition(self, nums):

"""

:type nums: List[int]

:rtype: bool

"""

capacity = sum(nums)

if capacity % 2 != 0:

return False

capacity /= 2

n = len(nums)

dp = [[False for j in range(capacity + 1)] for i in range(n)]

for j in range(capacity + 1):

if nums[0] == j:

dp[0][j] = True

for i in range(1, n):

for j in range(1, capacity + 1):

a = dp[i - 1][j]

b = False

if j - nums[i] >= 0:

b = dp[i - 1][j - nums[i]]

dp[i][j] = a or b

return dp[n - 1][capacity]

Solution3: DP解法 - 空间压缩

class Solution(object):

def canPartition(self, nums):

"""

:type nums: List[int]

:rtype: bool

"""

capacity = sum(nums)

if capacity % 2 != 0:

return False

capacity /= 2

n = len(nums)

dp = [False for i in range(capacity + 1)]

for i in range(capacity + 1):

dp[i] = (nums[0] == i)

for i in range(1, n):

for j in range(capacity, nums[i] - 1, -1):

dp[j] = dp[j] or dp[j - nums[i]]

return dp[capacity]

### 518-Coin Change 2

You are given coins of different denominations and a total amount of money. Write a function to compute the number of combinations that make up that amount. You may assume that you have infinite number of each kind of coin.

Note: You can assume that

0 <= amount <= 5000  
1 <= coin <= 5000  
the number of coins is less than 500  
the answer is guaranteed to fit into signed 32-bit integer

Example 1:

Input: amount = 5, coins = [1, 2, 5]

Output: 4

Explanation: there are four ways to make up the amount:

5=5

5=2+2+1

5=2+1+1+1

5=1+1+1+1+1

Example 2:

Input: amount = 3, coins = [2]

Output: 0

Explanation: the amount of 3 cannot be made up just with coins of 2.

Example 3:

Input: amount = 10, coins = [10]

Output: 1

递推方程

F(amount) = F(amount - c1) + F(amount - c1 - c1) + ... F(0)

+ F(amount - c2) + F(amount - c2 - c2) + .. F(0)

+ ...

Where c1, c2, .. are coin values given by problem description

Solution1: 记忆化递归 Python超时, Java通过

class Solution {

public int change(int amount, int[] coins) {

if (amount == 0) {

return 1;

}

if (coins.length == 0) {

return 0;

}

int len = coins.length;

int[][] memo = new int[len][amount + 1];

for (int i = 0; i < len; i++) {

for (int j = 0; j <= amount; j++) {

memo[i][j] = -1;

}

}

return helper(0, memo, coins, amount);

}

private int helper(int idx, int[][] memo, int[] coins, int amount) {

if (amount < 0) {

return 0;

}

if (amount == 0) {

return 1;

}

if (memo[idx][amount] > -1) {

return memo[idx][amount];

}

int res = 0;

for (int i = idx; i < coins.length; i++) {

res += helper(i, memo, coins, amount - coins[i]);

}

memo[idx][amount] = res;

return res;

}

}

Solution 2: DP

class Solution {

public int change(int amount, int[] coins) {

int len = coins.length;

int[][] dp = new int[len + 1][amount + 1];

dp[0][0] = 1;

for (int i = 1; i <= len; i++) {

for (int j = 0; j <= amount; j++) {

if (j == 0) {

dp[i][j] = 1;

} else {

if (j - coins[i - 1] >= 0) {

dp[i][j] = dp[i - 1][j] + dp[i][j - coins[i - 1]];

} else {

dp[i][j] = dp[i - 1][j];

}

}

}

}

return dp[coins.length][amount];

}

}

## 2. 状态变换类问题

### 010-Regular Expression Matching

(problem description)[[https://leetcode.com/problems/regular-expression-matching/description/]](https://leetcode.com/problems/regular-expression-matching/description/%5D)  
二维DP问题, 个人觉得面试考到这样的题的可能性已经很低了.  
<https://www.youtube.com/watch?v=l3hda49XcDE> 讲得很好

伪码

if s[i] == p[j] or p[j] == '.'

dp[i][j] = dp[i - 1][j - 1] # example: abc & aec / abc & abc

else

if p[j] == ' \* ':

dp[i][j] = dp[i][j - 2] # by default: 0 occurence

if s[i] == p[j - 1] or p[j - 1] == '.':

dp[i][j] = dp[i - 1][j]

Solution:

class Solution(object):

def isMatch(self, s, p):

"""

:type s: str

:type p: str

:rtype: bool

"""

lens = len(s)

lenp = len(p)

dp = [[False for j in range(lenp + 1)] for i in range(lens + 1)]

dp[0][0] = True

for j in range(1, lenp + 1):

if p[j - 1] == '\*':

dp[0][j] = dp[0][j - 2]

for i in range(lens + 1):

for j in range(1, lenp + 1):

if s[i - 1] == p[j - 1] or p[j - 1] == '.':

dp[i][j] = dp[i - 1][j - 1]

else:

if p[j - 1] == '\*':

dp[i][j] = dp[i][j - 2]

if s[i - 1] == p[j - 2] or p[j - 2] == '.':

dp[i][j] = dp[i][j] or dp[i - 1][j]

return dp[-1][-1]

### 053-Maximum Subarray

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

Example:

Input: [-2,1,-3,4,-1,2,1,-5,4],

Output: 6

Explanation: [4,-1,2,1] has the largest sum = 6.

Follow up:

If you have figured out the O(n) solution, try coding another solution using the divide and conquer approach, which is more subtle.

Solution1: DP

递推关系:  
f(i) = max(f(i - 1) + nums[i], nums[i])  
到i为止的最大数组求和 等于在前面的最大数组和 加上 nums[i], 或者 nums[i] 本身.

class Solution(object):

def maxSubArray(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

n = len(nums)

dp = [0 for i in range(n)]

dp[0] = nums[0]

res = nums[0]

for i in range(1, n):

dp[i] = max(dp[i - 1] + nums[i], nums[i])

res = max(res, dp[i])

return res

Solution2: DP 空间压缩

class Solution(object):

def maxSubArray(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

n = len(nums)

dp = nums[0]

res = nums[0]

for i in range(1, n):

tmp = dp

dp = max(tmp + nums[i], nums[i])

res = max(res, dp)

return res

Solution3: 记忆化递归

class Solution(object):

def \_\_init\_\_(self):

self.maxv = 0

def maxSubArray(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

n = len(nums)

self.maxv = nums[0]

tmp = self.helper(n - 1, nums)

return self.maxv

def helper(self, i, nums):

if i == 0:

return nums[0]

if i < 0:

return 0

res = max(nums[i], nums[i] + self.helper(i - 1, nums))

self.maxv = max(self.maxv, res)

return res

### 062-Unique Paths

[题目](https://leetcode.com/problems/unique-paths/description/)

Solution1: DP

class Solution(object):

def uniquePaths(self, m, n):

"""

:type m: int

:type n: int

:rtype: int

"""

dp = [[0 for j in range(n)] for i in range(m)]

for i in range(m):

for j in range(n):

if i == 0 or j == 0:

dp[i][j] = 1

else:

dp[i][j] = dp[i - 1][j] + dp[i][j - 1]

return dp[m - 1][n - 1]

Solution2: DP空间压缩

class Solution(object):

def uniquePaths(self, m, n):

"""

:type m: int

:type n: int

:rtype: int

"""

dp = [1 for j in range(n)]

for i in range(1, m):

tmp = 1

for j in range(1, n):

dp[j] = dp[j] + tmp

tmp = dp[j]

return dp[n - 1]

### 063-Unique Paths II

[题目](https://leetcode.com/problems/unique-paths-ii/description/)

class Solution(object):

def uniquePathsWithObstacles(self, obstacleGrid):

"""

:type obstacleGrid: List[List[int]]

:rtype: int

"""

row = len(obstacleGrid)

col = len(obstacleGrid[0])

dp = [[0 for j in range(col)] for i in range(row)]

rflag = False

cflag = False

for j in range(col):

if obstacleGrid[0][j] == 1:

rflag = True

if not rflag:

dp[0][j] = 1

for i in range(row):

if obstacleGrid[i][0] == 1:

cflag = True

if not cflag:

dp[i][0] = 1

for i in range(1, row):

for j in range(1, col):

if obstacleGrid[i][j] != 1:

dp[i][j] = dp[i][j - 1] + dp[i - 1][j]

return dp[row - 1][col - 1]

### 064-Minimum Path Sum

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.

Example:

Input:

[

[1,3,1],

[1,5,1],

[4,2,1]

]

Output: 7

Explanation: Because the path 1→3→1→1→1 minimizes the sum.

Solution: DP

class Solution(object):

def minPathSum(self, grid):

"""

:type grid: List[List[int]]

:rtype: int

"""

if not grid or len(grid[0]) == 0:

return 0

row = len(grid)

col = len(grid[0])

dp = [[0 for j in range(col)] for i in range(row)]

dp[0][0] = grid[0][0]

for i in range(1, row):

dp[i][0] = dp[i - 1][0] + grid[i][0]

for j in range(1, col):

dp[0][j] = dp[0][j - 1] + grid[0][j]

for i in range(1, row):

for j in range(1, col):

dp[i][j] = min(dp[i - 1][j], dp[i][j - 1]) + grid[i][j]

return dp[-1][-1]

Solution2: DP 压缩空间 O(n)

class Solution(object):

def minPathSum(self, grid):

"""

:type grid: List[List[int]]

:rtype: int

"""

if not grid or len(grid[0]) == 0:

return 0

row = len(grid)

col = len(grid[0])

dp = grid[0][:]

for i in range(1, col):

dp[i] = dp[i] + dp[i - 1]

for i in range(1, row):

for j in range(col):

if j == 0:

dp[j] = dp[j] + grid[i][j]

else:

left = dp[j - 1]

up = dp[j]

res = min(dp[j], dp[j - 1]) + grid[i][j]

dp[j] = res

return dp[-1]

### 070-Climbing Stairs

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.

Example 1:

Input: 2

Output: 2

Explanation: There are two ways to climb to the top.

1. 1 step + 1 step

2. 2 steps

Example 2:

Input: 3

Output: 3

Explanation: There are three ways to climb to the top.

1. 1 step + 1 step + 1 step

2. 1 step + 2 steps

3. 2 steps + 1 step

Solution1: 递归解

class Solution(object):

def climbStairs(self, n):

"""

:type n: int

:rtype: int

"""

memo = [-1 for i in range(n + 1)]

return self.helper(memo, n)

def helper(self, memo, i):

if i == 1:

return 1

if i == 2:

return 2

if memo[i] != -1:

return memo[i]

res = self.helper(memo, i - 1) + self.helper(memo, i - 2)

memo[i] = res

return res

Solution2: DP解

class Solution(object):

def climbStairs(self, n):

"""

:type n: int

:rtype: int

"""

if n == 1:

return 1

if n == 2:

return 2

dp = [0 for i in range(n + 1)]

dp[1] = 1

dp[2] = 2

for i in range(3, n + 1):

dp[i] = dp[i - 1] + dp[i - 2]

return dp[n]

### 072-Edit Distance

Given two words word1 and word2, find the minimum number of operations required to convert word1 to word2.

You have the following 3 operations permitted on a word:

1. Insert a character
2. Delete a character
3. Replace a character

Example 1:

Input: word1 = "horse", word2 = "ros"

Output: 3

Explanation:

horse -> rorse (replace 'h' with 'r')

rorse -> rose (remove 'r')

rose -> ros (remove 'e')

Example 2:

Input: word1 = "intention", word2 = "execution"

Output: 5

Explanation:

intention -> inention (remove 't')

inention -> enention (replace 'i' with 'e')

enention -> exention (replace 'n' with 'x')

exention -> exection (replace 'n' with 'c')

exection -> execution (insert 'u')

Solution: DP

class Solution(object):

def minDistance(self, word1, word2):

"""

:type word1: str

:type word2: str

:rtype: int

"""

n1 = len(word1)

n2 = len(word2)

dp = [[0 for j in range(n2 + 1)] for i in range(n1 + 1)]

for i in range(n1 + 1):

for j in range(n2 + 1):

if i == 0 or j == 0:

if i == 0:

dp[0][j] = j

if j == 0:

dp[i][j] = i

else:

if word1[i - 1] == word2[j - 1]:

dp[i][j] = dp[i - 1][j - 1]

else:

dp[i][j] = min(dp[i - 1][j], dp[i][j - 1], dp[i - 1][j - 1]) + 1

return dp[n1][n2]

### 085-Maximal Rectangle

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

Example:

Input:

[

["1","0","1","0","0"],

["1","0","1","1","1"],

["1","1","1","1","1"],

["1","0","0","1","0"]

]

Output: 6

Solution: DP (利用84题 Largest Rectangle in Histogram 的结果来做)

class Solution(object):

def maximalRectangle(self, matrix):

"""

:type matrix: List[List[str]]

:rtype: int

"""

if not matrix:

return 0

row = len(matrix)

col = len(matrix[0])

heights = map(lambda x: eval(x), matrix[0])

res = self.maximalHistogram(heights)

for i in range(1, row):

for j in range(col):

if matrix[i][j] == '0':

heights[j] = 0

else:

heights[j] = heights[j] + 1

res = max(res, self.maximalHistogram(heights))

return res

def maximalHistogram(self, heights) :

res = 0

stack = [-1]

heights.append(-1)

n = len(heights)

for i in range(n):

while heights[i] < heights[stack[-1]] :

tmp = stack.pop()

res = max(res, heights[tmp] \* (i - 1 - stack[-1]))

stack.append(i)

return res

### 091-Decode Ways

A message containing letters from A-Z is being encoded to numbers using the following mapping:

'A' -> 1

'B' -> 2

...

'Z' -> 26

Given a non-empty string containing only digits, determine the total number of ways to decode it.

Example 1:

Input: "12"

Output: 2

Explanation: It could be decoded as "AB" (1 2) or "L" (12).

Example 2:

Input: "226"

Output: 3

Explanation: It could be decoded as "BZ" (2 26), "VF" (22 6), or "BBF" (2 2 6).

Solution:  
F(S) : Decode way of strinf S  
递推方程: F('1234') = F('234') + F('34')

记忆化递归:

class Solution(object):

def numDecodings(self, s):

"""

:type s: str

:rtype: int

"""

n = len(s)

memo = [-1 for i in range(n)]

return self.helper(0, s, memo)

def helper(self, i, s, memo):

if i == len(s):

return 1

if memo[i] > -1:

return memo[i]

res = 0

if 1 <= int(s[i]) <= 9:

res += self.helper(i + 1, s, memo)

if 10 <= int(s[i:i + 2]) <= 26:

res += self.helper(i + 2, s, memo)

memo[i] = res

return res

def test(self):

s = '01234'

print self.numDecodings(s)

### 096-Unique Binary Search Trees

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

Example:

Input: 3

Output: 5

Explanation:

Given n = 3, there are a total of 5 unique BST's:

1 3 3 2 1

\ / / / \ \

3 2 1 1 3 2

/ / \ \

2 1 2 3

大神的解释 - 卡特兰数列  
G(n) : number of unique BST with number n

G(n) = G(0) \* G(n - 1) + G(1) \* G(n - 2) + G(2) \* G(n - 3) + ... + G(n - 1) \* G(0)

有了递推关系式就好办了

Solution1: 记忆化递归

class Solution(object):

def numTrees(self, n):

"""

:type n: int

:rtype: int

"""

if n == 0:

return 0

memo = [-1 for i in range(n + 1)]

return self.helper(n, memo)

def helper(self, i, memo):

if i == 0 or i == 1:

return 1

if memo[i] > -1:

return memo[i]

res = 0

for j in range(i):

res += self.helper(j, memo) \* self.helper(i - 1 - j, memo)

memo[i] = res

return res

Solution2: DP (比记忆化递归慢一点)

class Solution(object):

def numTrees(self, n):

"""

:type n: int

:rtype: int

"""

if n < 2:

return 1

dp = [0 for i in range(n + 1)]

dp[0] = 1

dp[1] = 1

for i in range(2, n + 1):

for j in range(i):

dp[i] += dp[j] \* dp[i - j - 1]

return dp[-1]

### 118-Pascal's Triangle

Example:

Input: 5

Output:

[

[1],

[1,1],

[1,2,1],

[1,3,3,1],

[1,4,6,4,1]

]

Solution: (不是DP, 但有点像)

class Solution(object):

def generate(self, numRows):

"""

:type numRows: int

:rtype: List[List[int]]

"""

if numRows == 0:

return []

res = [[1]]

for i in range(numRows - 1):

nums = (res[-1])[:]

n = len(nums)

for j in range(n - 1, 0, -1):

nums[j] = nums[j] + nums[j - 1]

nums.append(1)

res.append(nums)

return res

### 119-Pascal's Triangle II

Given a non-negative index k where k ≤ 33, return the kth index row of the Pascal's triangle.

Note that the row index starts from 0.

class Solution(object):

def getRow(self, rowIndex):

"""

:type rowIndex: int

:rtype: List[int]

"""

nums = [1]

for i in range(rowIndex):

n = len(nums)

for j in range(n - 1, 0, -1):

nums[j] = nums[j - 1] + nums[j]

nums.append(1)

return nums

### 120-Triangle

Given a triangle, find the minimum path sum from top to bottom. Each step you may move to adjacent numbers on the row below.

For example, given the following triangle

[

[2],

[3,4],

[6,5,7],

[4,1,8,3]

]

The minimum path sum from top to bottom is 11 (i.e., 2 + 3 + 5 + 1 = 11).

Solution1: DP

class Solution(object):

def minimumTotal(self, triangle):

"""

:type triangle: List[List[int]]

:rtype: int

"""

n = len(triangle)

for i in range(n - 2, -1, -1):

row = triangle[i]

below = triangle[i + 1]

length = len(row)

for j in range(length):

row[j] = row[j] + min(below[j], below[j + 1])

return triangle[0][0]

### 121-Best Time to Buy and Sell Stock

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

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.

Example 1:

Input: [7,1,5,3,6,4]

Output: 5

Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.

Not 7-1 = 6, as selling price needs to be larger than buying price.

Example 2:

Input: [7,6,4,3,1]

Output: 0

Explanation: In this case, no transaction is done, i.e. max profit = 0.

实在忍不住要吐槽这道题, 今天哪能知道明天的股价?? 要是今天就能把接下来一段时间的股价做成array, 早就发财了.

于是, 把这道题的意思提炼一下就是:  
在一个数组中找出最大数和最小数的差值, 条件是最大数必须在最小数的后面

Solution:

class Solution(object):

def maxProfit(self, prices):

"""

:type prices: List[int]

:rtype: int

"""

n = len(prices)

if n == 0:

return 0

curmin = prices[0]

res = 0

for i in range(1, n):

curmin = min(curmin, prices[i])

res = max(res, prices[i] - curmin)

return res

### 122-Best Time to Buy and Sell Stock II

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 (i.e., buy one and sell one share of the stock multiple times).

Note: You may not engage in multiple transactions at the same time (i.e., you must sell the stock before you buy again).

Example 1:

Input: [7,1,5,3,6,4]

Output: 7

Explanation: Buy on day 2 (price = 1) and sell on day 3 (price = 5), profit = 5-1 = 4.

Then buy on day 4 (price = 3) and sell on day 5 (price = 6), profit = 6-3 = 3.

Example 2:

Input: [1,2,3,4,5]

Output: 4

Explanation: Buy on day 1 (price = 1) and sell on day 5 (price = 5), profit = 5-1 = 4.

Note that you cannot buy on day 1, buy on day 2 and sell them later, as you are

engaging multiple transactions at the same time. You must sell before buying again.

Example 3:

Input: [7,6,4,3,1]

Output: 0

Explanation: In this case, no transaction is done, i.e. max profit = 0.

Solution:

class Solution(object):

def maxProfit(self, prices):

"""

:type prices: List[int]

:rtype: int

"""

n = len(prices)

if n == 0:

return 0

res = 0

curmin = prices[0]

for i in range(1, n):

if prices[i] <= curmin:

curmin = prices[i]

else:

res += (prices[i] - curmin)

curmin = prices[i]

return res

另一种写法:

class Solution(object):

def maxProfit(self, prices):

"""

:type prices: List[int]

:rtype: int

"""

n = len(prices)

if n == 0:

return 0

res = 0

curmin = prices[0]

for i in range(1, n):

if prices[i] > curmin:

res += (prices[i] - curmin)

curmin = prices[i]

return res

### 123-Best Time to Buy and Sell Stock III

class Solution(object):

def maxProfit(self, prices):

"""

:type prices: List[int]

:rtype: int

"""

n = len(prices)

if n < 2:

return 0

dp1 = [0 for i in range(n)] # 记录当天卖出所得max profit

dp2 = [0 for i in range(n)] # 记录当天买入所得max profit

curmin = prices[0]

for i in range(1, n):

dp1[i] = max(dp1[i - 1], prices[i] - curmin)

curmin = min(curmin, prices[i])

curmax = prices[n - 1]

for i in range(n - 2, -1, -1):

dp2[i] = max(dp2[i + 1], curmax - prices[i])

curmax = max(curmax, prices[i])

res = 0

for i in range(n):

res = max(res, dp1[i] + dp2[i])

return res

另一种写法:

class Solution(object):

def maxProfit(self, prices):

"""

:type prices: List[int]

:rtype: int

"""

n = len(prices)

if n < 2:

return 0

dp1 = [0 for i in range(n)]

dp2 = [0 for i in range(n)]

curmin = prices[0]

curmax = prices[n - 1]

for i in range(1, n):

j = n - 1 - i

dp1[i] = max(dp1[i - 1], prices[i] - curmin)

curmin = min(curmin, prices[i])

dp2[j] = max(dp2[j + 1], curmax - prices[j])

curmax = max(curmax, prices[j])

res = 0

for i in range(n):

res = max(res, dp1[i] + dp2[i])

return res

### 152-Maximum Product Subarray

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

Example 1:

Input: [2,3,-2,4]

Output: 6

Explanation: [2,3] has the largest product 6.

Example 2:

Input: [-2,0,-1]

Output: 0

Explanation: The result cannot be 2, because [-2,-1] is not a subarray.

Solution1:

class Solution(object):

def maxProduct(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

n = len(nums)

if n == 1:

return nums[0]

A = [0 for i in range(n)]

B = A[:]

A[0] = B[0] = nums[0]

res = nums[0]

for i in range(1, n):

A[i] = max(nums[i], nums[i] \* A[i - 1], nums[i] \* B[i - 1])

B[i] = min(nums[i], nums[i] \* A[i - 1], nums[i] \* B[i - 1])

res = max(res, A[i])

return res

Solution2: DP 压缩空间

class Solution(object):

def maxProduct(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

n = len(nums)

curMax, curMin = nums[0], nums[0]

res = nums[0]

for i in range(1, n):

tmpMax = curMax

tmpMin = curMin

curMax = max(tmpMax \* nums[i], tmpMin \* nums[i], nums[i])

curMin = min(tmpMax \* nums[i], tmpMin \* nums[i], nums[i])

res = max(res, curMax)

return res

### 174-Dungeon Game

[题目](https://leetcode.com/problems/dungeon-game/description/)

class Solution(object):

def calculateMinimumHP(self, dungeon):

"""

:type dungeon: List[List[int]]

:rtype: int

"""

row = len(dungeon)

col = len(dungeon[0])

maxint = (1 << 32) - 1

dp = [[maxint for j in range(col + 1)] for i in range(row + 1)]

dp[row][col - 1] = dp[row - 1][col] = 1

for i in range(row - 1, -1, -1):

for j in range(col - 1, -1, -1):

minval = min(dp[i][j + 1], dp[i + 1][j]) - dungeon[i][j]

dp[i][j] = minval if minval > 0 else 1

return dp[0][0]

### 198-House Robber

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.

Example 1:

Input: [1,2,3,1]

Output: 4

Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).

Total amount you can rob = 1 + 3 = 4.

Example 2:

Input: [2,7,9,3,1]

Output: 12

Explanation: Rob house 1 (money = 2), rob house 3 (money = 9) and rob house 5 (money = 1).

Total amount you can rob = 2 + 9 + 1 = 12.

Solution1: 记忆化递归  
基本上能使用动态规划的题目其实都可以用记忆化递归来做, 因为有相同子问题. 反之则不然, 动态规划的题目必须有最优子问题.  
对于这道题来说, 考虑偷取第0个房间开始和不考虑第0个房间开始两种选项:

第一种选项 - 从第0个房间开始考虑: Foo(n, 0, nums) = nums[0] + Foo(n, 2, nums)

第二种选项 - 从第1个房间开始考虑: Foo(n, 1, nums) = nums[1] + Foo(n, 3, nums)

res = max(Foo(n, 0, nums), Foo(n, 1, nums))

**code**:

class Solution(object):

def rob(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

if not nums:

return 0

n = len(nums)

memo = [-1 for i in range(n + 1)]

return self.helper(0, memo, nums, n)

def helper(self, i, memo, nums, n):

if i >= n:

return 0

if memo[i] > -1:

return memo[i]

a, b = 0, 0

if i < n:

a = nums[i] + self.helper(i + 2, memo, nums, n)

if i + 1 < n:

b = nums[i + 1] + self.helper(i + 3, memo, nums, n)

res = max(a, b)

memo[i] = res

return res

Solution2: DP  
对于动态规划的话比较抽象, 考虑偷第i个房子不代表一定要偷, 我理解了很久.

class Solution(object):

def rob(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

n = len(nums)

if n == 0:

return 0

if n == 1:

return nums[0]

rob = [0 for i in range(n)]

notRob = [0 for i in range(n)]

rob[0] = nums[0]

for i in range(1, n):

rob[i] = notRob[i - 1] + nums[i]

notRob[i] = max(rob[i - 1], notRob[i - 1])

return max(rob[n - 1], notRob[n - 1])

Solution2: DP 压缩空间

class Solution(object):

def rob(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

n = len(nums)

if n == 0:

return 0

if n == 1:

return nums[0]

rob = nums[0]

notRob = 0

for i in range(1, n):

a, b = rob, notRob

rob = b + nums[i]

notRob = max(a, b)

return max(rob, notRob)

### 213-House Robber II

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed. All houses at this place are arranged in a circle. That means the first house is the neighbor of the last one. Meanwhile, 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.

Example 1:

Input: [2,3,2]

Output: 3

Explanation: You cannot rob house 1 (money = 2) and then rob house 3 (money = 2),

because they are adjacent houses.

Example 2:

Input: [1,2,3,1]

Output: 4

Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).

Total amount you can rob = 1 + 3 = 4.

Solution: DP

1. 从第一个房子开始考虑, 有两种选择, 偷或者不偷第一个房子 -> 决定能不能偷最后一个房子
2. 从第二个开始考虑, 这样一定可以偷最后一个房子 (可以偷不意味一定要偷最后一个房子, 如果最大值是不偷最后一个房子所获得的话那就选择不偷最后一个)

class Solution(object):

def rob(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

n = len(nums)

if n == 0:

return 0

# A, B 代表从第一个房子开始考虑

A = [0 for i in range(n)]

B = A[:]

# C, D代表从第二个房子开始考虑

C = A[:]

D = A[:]

A[0] = nums[0]

for i in range(1, n):

if i == n - 1:

# A 代表偷第一个房子, 最后一个就不能偷

A[i] = max(A[i - 1], B[i - 1])

else:

A[i] = B[i - 1] + nums[i]

B[i] = max(A[i - 1], B[i - 1])

C[i] = D[i - 1] + nums[i]

D[i] = max(C[i - 1], D[i - 1])

return max(A[n - 1], B[n - 1], C[n - 1], D[n - 1])

Solution2: DP O(1) space complexity

class Solution(object):

def rob(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

n = len(nums)

if n == 0:

return 0

A = nums[0]

B = C = D = 0

for i in range(1, n):

ta, tb, tc, td = A, B, C, D

if i == n - 1:

A = max(ta, tb)

else:

A = tb + nums[i]

B = max(ta, tb)

C = td + nums[i]

D = max(tc, td)

return max(A, B, C, D)

### 221-Maximal Square

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

Example:

Input:

1 0 1 0 0

1 0 1 1 1

1 1 1 1 1

1 0 0 1 0

Output: 4

Solution:

class Solution(object):

def maximalSquare(self, matrix):

"""

:type matrix: List[List[str]]

:rtype: int

"""

row = len(matrix)

if row == 0:

return 0

col = len(matrix[0])

if col == 0:

return 0

dp = [[0 for j in range(col)] for i in range(row)]

maxline = 0

for i in range(row):

for j in range(col):

if i == 0 or j == 0:

dp[i][j] = int(matrix[i][j])

elif matrix[i][j] == '1':

dp[i][j] = min(dp[i - 1][j], dp[i][j - 1], dp[i - 1][j - 1]) + 1

maxline = max(maxline, dp[i][j])

return maxline \* maxline

### 238-Product of Array Except Self

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].

Example:

Input: [1,2,3,4]

Output: [24,12,8,6]

Note: Please solve it without division and in O(n).

Follow up:  
Could you solve it with constant space complexity? (The output array does not count as extra space for the purpose of space complexity analysis.)

class Solution(object):

def productExceptSelf(self, nums):

"""

:type nums: List[int]

:rtype: List[int]

"""

n = len(nums)

if n < 2:

return nums

A = [1 for i in range(n)]

B = A[:]

cur = 1

for i in range(1, n):

cur = cur \* nums[i - 1]

A[i] = cur

cur = 1

for i in range(n - 2, -1, -1):

cur = cur \* nums[i + 1]

B[i] = cur

for i in range(n):

A[i] = A[i] \* B[i]

return A

Solution2: 压缩空间

class Solution(object):

def productExceptSelf(self, nums):

"""

:type nums: List[int]

:rtype: List[int]

"""

n = len(nums)

if n < 2:

return nums

A = [1 for i in range(n)]

cur = 1

for i in range(1, n):

cur = cur \* nums[i - 1]

A[i] = cur

cur = 1

for i in range(n - 2, -1, -1):

cur = cur \* nums[i + 1]

A[i] = A[i] \* cur

return A

### 256-Paint House

There are a row of n houses, each house can be painted with one of the three colors: red, blue or green. The cost of painting each house with a certain color is different. You have to paint all the houses such that no two adjacent houses have the same color.

The cost of painting each house with a certain color is represented by a n x 3 cost matrix. For example, costs[0][0] is the cost of painting house 0 with color red; costs[1][2] is the cost of painting house 1 with color green, and so on... Find the minimum cost to paint all houses.

Note:  
All costs are positive integers.

Example:

Input: [[17,2,17],[16,16,5],[14,3,19]]

Output: 10

Explanation: Paint house 0 into blue, paint house 1 into green, paint house 2 into blue.

Minimum cost: 2 + 5 + 3 = 10.

Solution1:

class Solution(object):

def minCost(self, costs):

"""

:type costs: List[List[int]]

:rtype: int

"""

if not costs or not costs[0]:

return 0

n = len(costs)

dp = [[0 for j in range(3)] for i in range(n)]

for i in range(n):

if i == 0:

dp[i] = costs[i][:]

else:

dp[i][0] = min(dp[i - 1][1], dp[i - 1][2]) + costs[i][0]

dp[i][1] = min(dp[i - 1][0], dp[i - 1][2]) + costs[i][1]

dp[i][2] = min(dp[i - 1][0], dp[i - 1][1]) + costs[i][2]

res = min(dp[n - 1][0], dp[n - 1][1], dp[n - 1][2])

return res

Solution1:

class Solution(object):

def minCost(self, costs):

"""

:type costs: List[List[int]]

:rtype: int

"""

if not costs or not costs[0]:

return 0

n = len(costs)

dp = [[0 for j in range(3)] for i in range(n)]

for i in range(n):

if i == 0:

dp[i] = costs[i][:]

else:

for j in range(3):

minval = 1 << 31

for k in range(3):

if j != k:

minval = min(minval, dp[i - 1][k])

dp[i][j] = minval + costs[i][j]

res = min(dp[n - 1][0], dp[n - 1][1], dp[n - 1][2])

return res

Solution1: 压缩空间

class Solution(object):

def minCost(self, costs):

"""

:type costs: List[List[int]]

:rtype: int

"""

if not costs or not costs[0]:

return 0

n = len(costs)

dp = costs[0][:]

for i in range(1, n):

a, b, c = dp[0], dp[1], dp[2]

dp[0] = min(b, c) + costs[i][0]

dp[1] = min(a, c) + costs[i][1]

dp[2] = min(a, b) + costs[i][2]

return min(dp)

### 264-Ugly Number II

Write a program to find the n-th ugly number.

Ugly numbers are positive numbers whose prime factors only include 2, 3, 5.

Example:

Input: n = 10

Output: 12

Explanation: 1, 2, 3, 4, 5, 6, 8, 9, 10, 12 is the sequence of the first 10 ugly numbers.

Note:

1 is typically treated as an ugly number.  
n does not exceed 1690.

Solution: 最优解不好理解, 我使用基于优先队列的BFS解法. 复杂度为 O(nlogn)

import heapq

class Solution(object):

def nthUglyNumber(self, n):

"""

:type n: int

:rtype: int

"""

queue = [1]

visited = set([1])

res = 0

cnt = 0

while queue:

if cnt == n:

break

num = heapq.heappop(queue)

res = num

cnt += 1

for v in [2, 3, 5]:

val = num \* v

if val not in visited:

visited.add(val)

heapq.heappush(queue, val)

return res

### 265-Paint House II

There are a row of n houses, each house can be painted with one of the k colors. The cost of painting each house with a certain color is different. You have to paint all the houses such that no two adjacent houses have the same color.

The cost of painting each house with a certain color is represented by a n x k cost matrix. For example, costs[0][0] is the cost of painting house 0 with color 0; costs[1][2] is the cost of painting house 1 with color 2, and so on... Find the minimum cost to paint all houses.

Note:  
All costs are positive integers.

Example:

Input: [[1,5,3],[2,9,4]]

Output: 5

Explanation: Paint house 0 into color 0, paint house 1 into color 2. Minimum cost: 1 + 4 = 5;

Or paint house 0 into color 2, paint house 1 into color 0. Minimum cost: 3 + 2 = 5.

Follow up:  
Could you solve it in O(nk) runtime?

Solution1: 和256一样 - O(nk^2) 复杂度

class Solution(object):

def minCostII(self, costs):

"""

:type costs: List[List[int]]

:rtype: int

"""

if not costs or not costs[0]:

return 0

row = len(costs) # number of houses

col = len(costs[0]) # number of colors

dp = [[0 for j in range(col)] for i in range(row)]

dp[0] = costs[0][:]

maxint = (1 << 32) - 1

for i in range(1, row):

for j in range(col):

minv = maxint

for k in range(col):

if j != k:

minv = min(minv, dp[i - 1][k])

dp[i][j] = minv + costs[i][j]

return min(dp[row - 1])

Solution2: DP - improved to O(nk) time complexity

class Solution(object):

def minCostII(self, costs):

"""

:type costs: List[List[int]]

:rtype: int

"""

if not costs or not costs[0]:

return 0

row = len(costs) # number of houses

col = len(costs[0]) # number of colors

dp = [[0 for j in range(col)] for i in range(row)]

maxint = (1 << 32) - 1

first = second = maxint

minidx = 0

res = maxint

for i in range(row):

k = minidx

a1, a2 = first, second

first = second = maxint

for j in range(col):

if i == 0:

dp[i][j] = costs[i][j]

else:

if j != k:

dp[i][j] = a1 + costs[i][j]

else:

dp[i][j] = a2 + costs[i][j]

if dp[i][j] < first:

second = first

first = dp[i][j]

minidx = j

elif dp[i][j] < second:

second = dp[i][j]

if i == row - 1:

res = min(res, dp[i][j])

return res

### 300-Longest Increasing Subsequence

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

Example:

Input: [10,9,2,5,3,7,101,18]

Output: 4

Explanation: The longest increasing subsequence is [2,3,7,101], therefore the length is 4.

Solution: DP

class Solution(object):

def lengthOfLIS(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

n = len(nums)

if n == 0:

return 0

dp = [1 for i in range(n)]

res = 1

for i in range(1, n):

for j in range(i):

if nums[i] > nums[j]:

dp[i] = max(dp[i], dp[j] + 1)

res = max(res, dp[i])

return res

### 303-Range Sum Query - Immutable

Given an integer array nums, find the sum of the elements between indices i and j (i ≤ j), inclusive.

Example:

Given nums = [-2, 0, 3, -5, 2, -1]

sumRange(0, 2) -> 1

sumRange(2, 5) -> -1

sumRange(0, 5) -> -3

Note:  
You may assume that the array does not change.  
There are many calls to sumRange function.

class NumArray(object):

def \_\_init\_\_(self, nums):

"""

:type nums: List[int]

"""

self.dp = nums[:]

n = len(nums)

for i in range(1, n):

self.dp[i] = self.dp[i] + self.dp[i - 1]

def sumRange(self, i, j):

"""

:type i: int

:type j: int

:rtype: int

"""

if i == 0:

return self.dp[j]

return self.dp[j] - self.dp[i - 1]

# Your NumArray object will be instantiated and called as such:

# obj = NumArray(nums)

# param\_1 = obj.sumRange(i,j)

### 304-Range Sum Query 2D - Immutable

[题目](https://leetcode.com/problems/range-sum-query-2d-immutable/description/)

class NumMatrix(object):

def \_\_init\_\_(self, matrix):

"""

:type matrix: List[List[int]]

"""

self.dp = None

row = len(matrix)

if row == 0:

return

col = len(matrix[0])

if col == 0:

return

self.dp = [[0 for j in range(col + 1)] for i in range(row + 1)]

self.dp[1][1] = matrix[0][0]

for i in range(2, row + 1):

self.dp[i][1] = self.dp[i - 1][1] + matrix[i - 1][0]

for j in range(2, col + 1):

self.dp[1][j] = self.dp[1][j - 1] + matrix[0][j - 1]

for i in range(2, row + 1):

for j in range(2, col + 1):

self.dp[i][j] = self.dp[i - 1][j] + self.dp[i][j - 1] - self.dp[i - 1][j - 1] + matrix[i - 1][j - 1]

def sumRegion(self, row1, col1, row2, col2):

"""

:type row1: int

:type col1: int

:type row2: int

:type col2: int

:rtype: int

"""

if not self.dp:

return 0

r1 = row1 + 1

c1 = col1 + 1

r2 = row2 + 1

c2 = col2 + 1

return self.dp[r2][c2] - self.dp[r1 - 1][c2] - self.dp[r2][c1 - 1] + self.dp[r1 - 1][c1 - 1]

# Your NumMatrix object will be instantiated and called as such:

# obj = NumMatrix(matrix)

# param\_1 = obj.sumRegion(row1,col1,row2,col2)

### 309-Best Time to Buy and Sell Stock with Cooldown

(暂时理解不了 - 暂时放下)

### 343-Integer Break

Given a positive integer n, break it into the sum of at least two positive integers and maximize the product of those integers. Return the maximum product you can get.

Example 1:

Input: 2

Output: 1

Explanation: 2 = 1 + 1, 1 × 1 = 1.

Example 2:

Input: 10

Output: 36

Explanation: 10 = 3 + 3 + 4, 3 × 3 × 4 = 36.

Note: You may assume that n is not less than 2 and not larger than 58.

Solution 1: Memorized recursion  
这道题个人觉得递归更好理解一点, DP不好理解.

class Solution(object):

def integerBreak(self, n):

"""

:type n: int

:rtype: int

"""

memo = [-1 for i in range(n + 1)]

return self.helper(n, memo)

def helper(self, n, memo):

if n == 2:

return 1

if memo[n] > -1:

return memo[n]

res = 0

for i in range(1, n):

# i \* (n - i) 代表不在分割 n - i, 直接把 n **简**单第分成两份, 这是一种可能性

# i \* helper(n - i) 代表继续考虑把 n - i 进行分割, 求出最大值然后和 i 相乘, 这是另一种可能性

res = max(res, i \* (n - i), i \* self.helper(n - i, memo))

memo[n] = res

return res

Solution2: DP

class Solution(object):

def integerBreak(self, n):

"""

:type n: int

:rtype: int

"""

memo = [0 for i in range(n + 1)]

memo[2] = 1

for i in range(3, n + 1):

#求memo[i] : 代表分割 i 所得到的最大乘积

for j in range(1, i):

memo[i] = max(memo[i], j \* (i - j), j \* memo[i - j])

return memo[n]

### 338-Counting Bits

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.

Example 1:

Input: 2

Output: [0,1,1]

Example 2:

Input: 5

Output: [0,1,1,2,1,2]

Solution: 找规律 DP

class Solution(object):

def countBits(self, num):

"""

:type num: int

:rtype: List[int]

"""

dp = [0 for i in range(num + 1)]

for i in range(1, num + 1):

dp[i] = dp[i / 2] + i % 2

return dp

### 361-Bomb Enemy

Given a 2D grid, each cell is either a wall 'W', an enemy 'E' or empty '0' (the number zero), return the maximum enemies you can kill using one bomb.  
The bomb kills all the enemies in the same row and column from the planted point until it hits the wall since the wall is too strong to be destroyed.  
Note: You can only put the bomb at an empty cell.

Example:

Input: [["0","E","0","0"],["E","0","W","E"],["0","E","0","0"]]

Output: 3

Explanation: For the given grid,

0 E 0 0

E 0 W E

0 E 0 0

Placing a bomb at (1,1) kills 3 enemies.

Solution:

class Solution(object):

def maxKilledEnemies(self, grid):

"""

:type grid: List[List[str]]

:rtype: int

"""

if len(grid) == 0 or len(grid[0]) == 0:

return 0

row = len(grid)

col = len(grid[0])

eRow = 0

eCols = [0 for i in range(col)]

res = 0

for i in range(row):

for j in range(col):

if i == 0 or grid[i - 1][j] == 'W':

eCols[j] = self.getColCnt(grid, i, j)

if j == 0 or grid[i][j - 1] == 'W':

eRow = self.getRowCnt(grid, i, j)

if grid[i][j] == '0':

res = max(res, eCols[j] + eRow)

print eCols

return res

def getRowCnt(self, grid, i, j):

num = 0

while j < len(grid[0]) and grid[i][j] != 'W':

if grid[i][j] == 'E':

num += 1

j += 1

return num

def getColCnt(self, grid, i, j):

num = 0

while i < len(grid) and grid[i][j] != 'W':

if grid[i][j] == 'E':

num += 1

i += 1

return num

# LeetCode总结13 : Greedy

**253-Meeting Rooms II**

# Definition for an interval.

# class Interval(object):

# def \_\_init\_\_(self, s=0, e=0):

# self.start = s

# self.end = e

class Solution(object):

def minMeetingRooms(self, intervals):

"""

:type intervals: List[Interval]

:rtype: int

"""

starts = []

ends = []

for v in intervals:

starts.append(v.start)

ends.append(v.end)

starts.sort()

ends.sort()

i, j = 0, 0

n = len(intervals)

res = 0

while i < n and j < n:

if starts[i] < ends[j]:

res += 1

i += 1

else:

i += 1

j += 1

return res

**392-Is Subsequence**

Given a string s and a string t, check if s is subsequence of t.

You may assume that there is only lower case English letters in both s and t. t is potentially a very long (length ~ 500,000) string, and s is a short string (<= 100)

A subsequence of a string is a new string which is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (ie, "ace" is a subsequence of "abcde" while "aec" is not).

Example 1:  
s = "abc", t = "ahbgdc"

Return true.

Example 2:  
s = "axc", t = "ahbgdc"

Return false.

class Solution(object):

def isSubsequence(self, s, t):

"""

:type s: str

:type t: str

:rtype: bool

"""

n = len(t)

m = len(s)

i, j = 0, 0

while j < n and i < m:

if s[i] == t[j]:

i += 1

j += 1

else:

j += 1

return i == len(s)

**406-Queue Reconstruction by Height**

大神的解法:  
假设已经有了一个满足条件的队列, 例如 [[7, 0], [7, 1], [6, 1]],  
家下来有一个元素, 比队列中所有元素都小, 例如 [5, 0]. 我们只需要直接把它插入到0这个位置即可, 而队列后面所有元素顺序不变. 这是因为这个人比队列中所有人都矮, 根据条件, k这个数只和队列前面比自己高的人有关, 因此一个比所有人都矮的people元素插入到队列不影响其它队列元素.  
因此, 我们把input元素从高到矮挨个插入一个队列即可. 反之, 从矮到高插入则不可行, 因为比所有人都高的人插入队列要影响到整个队列前面的人.

Solution:

class Solution(object):

def reconstructQueue(self, people):

"""

:type people: List[List[int]]

:rtype: List[List[int]]

"""

people.sort(key=lambda (h, k) : (-h, k))

queue = []

for p in people:

pos = p[1]

queue.insert(pos, p)

return queue

**435-Non overlaping interval**

Given a collection of intervals, find the minimum number of intervals you need to remove to make the rest of the intervals non-overlapping.

Note:  
You may assume the interval's end point is always bigger than its start point.  
Intervals like [1,2] and [2,3] have borders "touching" but they don't overlap each other.  
Example 1:

Input: [ [1,2], [2,3], [3,4], [1,3] ]

Output: 1

Explanation: [1,3] can be removed and the rest of intervals are non-overlapping.

Example 2:

Input: [ [1,2], [1,2], [1,2] ]

Output: 2

Explanation: You need to remove two [1,2] to make the rest of intervals non-overlapping.

Example 3:

Input: [ [1,2], [2,3] ]

Output: 0

Explanation: You don't need to remove any of the intervals since they're already non-overlapping.

Solution:

class Interval(object):

def \_\_init\_\_(self, s=0, e=0):

self.start = s

self.end = e

class Solution(object):

def eraseOverlapIntervals(self, intervals):

"""

:type intervals: List[Interval]

:rtype: int

"""

if not intervals:

return 0

def comp(a, b):

if a.end == b.end:

return a.start - b.start

return a.end - b.end

intervals.sort(comp)

n = len(intervals)

res = 1

prev = intervals[0]

for i in range(1, n):

cur = intervals[i]

if cur.start >= prev.end:

res += 1

prev = cur

return n - res

**455-Assign Cookies**

Assume you are an awesome parent and want to give your children some cookies. But, you should give each child at most one cookie. Each child i has a greed factor gi, which is the minimum size of a cookie that the child will be content with; and each cookie j has a size sj. If sj >= gi, we can assign the cookie j to the child i, and the child i will be content. Your goal is to maximize the number of your content children and output the maximum number.

Note:  
You may assume the greed factor is always positive.  
You cannot assign more than one cookie to one child.

Example 1:

Input: [1,2,3], [1,1]

Output: 1

Explanation: You have 3 children and 2 cookies. The greed factors of 3 children are 1, 2, 3.

And even though you have 2 cookies, since their size is both 1, you could only make the child whose greed factor is 1 content.

You need to output 1.

Example 2:

Input: [1,2], [1,2,3]

Output: 2

Explanation: You have 2 children and 3 cookies. The greed factors of 2 children are 1, 2.

You have 3 cookies and their sizes are big enough to gratify all of the children,

You need to output 2.

Solution:

class Solution(object):

def findContentChildren(self, g, s):

"""

:type g: List[int]

:type s: List[int]

:rtype: int

"""

m = len(g)

n = len(s)

g.sort(reverse=True)

s.sort(reverse=True)

i = j = 0

res = 0

while j < m and i < n:

if s[i] >= g[j]:

res += 1

i += 1

j += 1

else:

j += 1

return res

def test(self):

s = [9, 5, 4, 1, 1]

g = [8, 7, 6, 2, 1, 1]

print self.findContentChildren(g, s)

**755-Pour Water**

We are given an elevation map, heights[i] representing the height of the terrain at that index. The width at each index is 1. After V units of water fall at index K, how much water is at each index?

Water first drops at index K and rests on top of the highest terrain or water at that index. Then, it flows according to the following rules:

If the droplet would eventually fall by moving left, then move left.  
Otherwise, if the droplet would eventually fall by moving right, then move right.  
Otherwise, rise at it's current position.  
Here, "eventually fall" means that the droplet will eventually be at a lower level if it moves in that direction. Also, "level" means the height of the terrain plus any water in that column.  
We can assume there's infinitely high terrain on the two sides out of bounds of the array. Also, there could not be partial water being spread out evenly on more than 1 grid block - each unit of water has to be in exactly one block.

Example 1:

Input: heights = [2,1,1,2,1,2,2], V = 4, K = 3

Output: [2,2,2,3,2,2,2]

Explanation:

# #

# #

## # ###

#########

0123456 <- index

The first drop of water lands at index K = 3:

# #

# w #

## # ###

#########

0123456

When moving left or right, the water can only move to the same level or a lower level.

(By level, we mean the total height of the terrain plus any water in that column.)

Since moving left will eventually make it fall, it moves left.

(A droplet "made to fall" means go to a lower height than it was at previously.)

# #

# #

## w# ###

#########

0123456

Since moving left will not make it fall, it stays in place. The next droplet falls:

# #

# w #

## w# ###

#########

0123456

Since the new droplet moving left will eventually make it fall, it moves left.

Notice that the droplet still preferred to move left,

even though it could move right (and moving right makes it fall quicker.)

# #

# w #

## w# ###

#########

0123456

# #

# #

##ww# ###

#########

0123456

After those steps, the third droplet falls.

Since moving left would not eventually make it fall, it tries to move right.

Since moving right would eventually make it fall, it moves right.

# #

# w #

##ww# ###

#########

0123456

# #

# #

##ww#w###

#########

0123456

Finally, the fourth droplet falls.

Since moving left would not eventually make it fall, it tries to move right.

Since moving right would not eventually make it fall, it stays in place:

# #

# w #

##ww#w###

#########

0123456

The final answer is [2,2,2,3,2,2,2]:

#

#######

#######

0123456

Example 2:

Input: heights = [1,2,3,4], V = 2, K = 2

Output: [2,3,3,4]

Explanation:

The last droplet settles at index 1, since moving further left would not cause it to eventually fall to a lower height.

Example 3:

Input: heights = [3,1,3], V = 5, K = 1

Output: [4,4,4]

Solution:

class Solution(object):

def pourWater(self, heights, V, K):

"""

:type heights: List[int]

:type V: int

:type K: int

:rtype: List[int]

"""

if not heights or V == 0:

return heights

n = len(heights)

while V > 0:

idx = K

for i in range(K - 1, -1, -1):

if heights[i] < heights[idx]:

idx = i

elif heights[i] > heights[idx]:

break

else: # heights[i] == heights[idx]

continue

if idx != K:

heights[idx] += 1

V -= 1

continue

for i in range(K + 1, n):

if heights[i] < heights[idx]:

idx = i

elif heights[i] > heights[idx]:

break

else:

continue

heights[idx] += 1

V -= 1

return heights

PS: 咬牙坚持住, 两个同事已经成功跳槽谷歌微软, 我也加油!!

# LeetCode总结14-线段树与Trie

### 208-Implement Trie

class Node(object):

def \_\_init\_\_(self):

self.isWord = False

self.next = {}

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

"""

cur = self.root

n = len(word)

for i in range(n):

c = word[i]

if not cur.next.get(c):

cur.next[c] = Node()

cur = cur.next[c]

cur.isWord = True

def search(self, word):

"""

Returns if the word is in the trie.

:type word: str

:rtype: bool

"""

cur = self.root

n = len(word)

for i in range(n):

c = word[i]

if not cur.next.get(c):

return False

cur = cur.next[c]

return cur.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

"""

cur = self.root

n = len(prefix)

for i in range(n):

c = prefix[i]

if not cur.next.get(c):

return False

cur = cur.next.get(c)

return True

Java 完整版实现

import java.util.HashMap;

public class Trie {

private class Node {

public boolean isWord;

public HashMap<Character, Node> next;

public Node(boolean isWord) {

this.isWord = isWord;

next = new HashMap<>();

}

public Node() {

this(false);

}

}

private Node root;

private int size;

public Trie() {

root = new Node();

size = 0;

}

public int getSize() {

return size;

}

public void add(String word) {

Node cur = root;

int len = word.length();

for (int i = 0; i < len; i++) {

char c = word.charAt(i);

if (cur.next.get(c) == null) {

cur.next.put(c, new Node());

}

cur = cur.next.get(c);

}

if (!cur.isWord) {

cur.isWord = true;

size += 1;

}

}

public boolean contains(String word) {

Node cur = root;

int len = word.length();

for (int i = 0; i < len; i++) {

char c = word.charAt(i);

if (cur.next.get(c) == null) {

return false;

}

cur = cur.next.get(c);

}

return cur.isWord;

}

public boolean isPrefix(String prefix) {

Node cur = root;

int len = prefix.length();

for (int i = 0; i < len; i++) {

char c = prefix.charAt(i);

if (cur.next.get(c) == null) {

return false;

}

cur = cur.next.get(c);

}

return true;

}

public boolean search(String word) {

Node cur = root;

return match(root, word, 0);

}

private boolean match(Node node, String word, int i) {

int len = word.length();

if (i == len) {

return node.isWord;

}

char c = word.charAt(i);

if (c != '.') {

if (node.next.get(c) == null) {

return false;

}

return match(node.next.get(c), word, i + 1);

} else { // c == '.'

for (char nextChar : node.next.keySet()) {

if (match(node.next.get(nextChar), word, i + 1)) {

return true;

}

}

return false;

}

}

}

### 211-Add and Search Word Data structure design

same as 208

class Node(object):

def \_\_init\_\_(self):

self.isWord = False

self.next = {}

class WordDictionary(object):

def \_\_init\_\_(self):

"""

Initialize your data structure here.

"""

self.root = Node()

def addWord(self, word):

"""

Adds a word into the data structure.

:type word: str

:rtype: void

"""

cur = self.root

n = len(word)

for i in range(n):

c = word[i]

if not cur.next.get(c):

cur.next[c] = Node()

cur = cur.next[c]

cur.isWord = True

def search(self, word):

"""

Returns if the word is in the data structure. A word could contain the dot character '.' to represent any one letter.

:type word: str

:rtype: bool

"""

return self.searchHelper(self.root, word, 0)

def searchHelper(self, node, word, i):

n = len(word)

if i == n:

return node.isWord

c = word[i]

if c != '.':

if not node.next.get(c):

return False

return self.searchHelper(node.next[c], word, i + 1)

else:

for char in node.next:

nextNode = node.next.get(char)

if self.searchHelper(nextNode, word, i + 1):

return True

return False

### 307-Range Sum Query - Mutable

Solution: 线段树

class SegmentTree(object):

def \_\_init\_\_(self, arr):

self.data = arr[:]

self.tree = [0 for i in range(4 \* len(arr))]

self.buildTree(0, 0, len(arr) - 1)

# O(n)

def buildTree(self, idx, l, r):

if l == r:

self.tree[idx] = self.data[l]

return

leftIdx = self.leftChild(idx)

rightIdx = self.rightChild(idx)

mid = l + (r - l) / 2

self.buildTree(leftIdx, l, mid)

self.buildTree(rightIdx, mid + 1, r)

self.tree[idx] = self.tree[leftIdx] + self.tree[rightIdx]

def query(self, leftBound, rightBound):

return self.queryHelper(0, 0, len(self.data) - 1, leftBound, rightBound)

def set(self, pos, val):

self.data[pos] = val

self.setHelper(0, 0, len(self.data) - 1, pos, val)

# O(logn)

def setHelper(self, treeIdx, l, r, pos, val):

if l == r:

self.tree[treeIdx] = val

return

mid = l + (r - l) / 2

leftChild = self.leftChild(treeIdx)

rightChild = self.rightChild(treeIdx)

if pos > mid:

self.setHelper(rightChild, mid + 1, r, pos, val)

else:

self.setHelper(leftChild, l, mid, pos, val)

self.tree[treeIdx] = self.tree[leftChild] + self.tree[rightChild]

# O(logn)

def queryHelper(self, treeIdx, l, r, leftBound, rightBound):

if l == leftBound and r == rightBound:

return self.tree[treeIdx]

mid = l + (r - l) / 2

leftChild = self.leftChild(treeIdx)

rightChild = self.rightChild(treeIdx)

if leftBound > mid:

return self.queryHelper(rightChild, mid + 1, r, leftBound, rightBound)

elif rightBound <= mid:

return self.queryHelper(leftChild, l, mid, leftBound, rightBound)

else:

leftRes = self.queryHelper(leftChild, l, mid, leftBound, mid)

rightRes = self.queryHelper(rightChild, mid + 1, r, mid + 1, rightBound)

return leftRes + rightRes

def leftChild(self, idx):

return idx \* 2 + 1

def rightChild(self, idx):

return idx \* 2 + 2

class NumArray(object):

def \_\_init\_\_(self, nums):

"""

:type nums: List[int]

"""

if not nums:

return

self.segTree = SegmentTree(nums)

def update(self, i, val):

"""

:type i: int

:type val: int

:rtype: void

"""

self.segTree.set(i, val)

def sumRange(self, i, j):

"""

:type i: int

:type j: int

:rtype: int

"""

return self.segTree.query(i, j)