## 1. 哈希Two Sum

Giv**en an array of integers, return indices of the two numbers such that they add up to a specific target.**

**You may assume that each input would have exactly one solution.**

**Example:**

**Given nums = [2, 7, 11, 15], target = 9,**

**Because nums[0] + nums[1] = 2 + 7 = 9,**

**return [0, 1].**

**-----------------------------------------Code--------------------------------------------------**

class Solution(object):

def twoSum(self, nums, target):

#hash用于建立数值到下标的映射

hash = {}

#循环nums数值，并添加映射

for i in range(len(nums)):

if target - nums[i] in hash:

return [hash[target - nums[i]], i]

hash[nums[i]] = i

#无解的情况

return [-1, -1]

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

## 8. String to Integer (atoi)

Hint: Carefully consider all possible input cases. If you want a challenge, please do not see below and ask yourself what are the possible input cases.

Notes: It is intended for this problem to be specified vaguely (ie, no given input specs). You are responsible to gather all the input requirements up front.

-----------------------------------------Algorithm-------------------------------------------------

The key is the function strip()

-----------------------------------------Code--------------------------------------------------

class Solution(object):

def myAtoi(self, str):

if not len(str):

return 0

str = str.strip()

sign = -1 if str[0] == "-" else 1

ans,i,length = 0,0,len(str)

if str[0] in ["+","-"]:

i += 1

while i < length and str[i].isdigit():

ans = ans\*10 + int(str[i])

i += 1

print(ans)

return max(-2\*\*31,min(2\*\*31-1,sign\*ans))

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

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

obj = Solution()

str = "+-111"

print(obj.myAtoi(str))

## 15. 双指针 3Sum

Given an array S of n integers, are there elements a, b, c in S 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.

For example, given array S = [-1, 0, 1, 2, -1, -4],

A solution set is:

[

[-1, 0, 1],

[-1, -1, 2]

]

**-----------------------------------------Code--------------------------------------------------**

#include <stdlib.h>

#include <cstring>

#include<vector>

#include<iostream>

#include<algorithm>

class Solution {

public:

std::vector<std::vector<int> > threeSum(std::vector<int>& nums) {

sort(nums.begin(),nums.end());

int len=nums.size(),target=0,st,en;

std::vector<std::vector<int> >Return;

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

target=-nums[i];

st=i+1,en=len-1;

while(st<en){

if(nums[st]+nums[en]==target){

std::vector<int> ans;

ans.push\_back(nums[i]);

ans.push\_back(nums[st]);

ans.push\_back(nums[en]);

st++,en--;

Return.push\_back(ans);

while(st<en&&nums[en]==nums[en+1])en--;

while(st<en&&nums[st]==nums[st-1])st++;

}

else if(nums[st]+nums[en]>target){

en--;

}

else{

st++;

}

}

while(nums[i]==nums[i+1])i++;

}

return Return;

}

};

int main(){

std::vector<int>v;

v.push\_back(0);

v.push\_back(2);

v.push\_back(0);

v.push\_back(-2);

v.push\_back(2);

Solution obj;

std::vector<std::vector<int> >ans(obj.threeSum(v));

for(std::vector<std::vector<int> >::iterator i=ans.begin();i<ans.end();i++){

for(std::vector<int>::iterator j=(\*i).begin();j<(\*i).end();j++){

std::cout<<\*j<<" ";

}

std::cout<<std::endl;

}

return 0;

}

class Solution(object):

def largestRectangleArea(self, heights):

stack=[0]

heights.append(0)

width=0

i=0

ans=-1

Len=len(heights)

while (i < Len) :

if( not stack or heights[i] > heights[stack[-1]] ):

stack.append(i)

i += 1

else:

index = stack.pop()

if(stack):

width = i - stack[-1] - 1

else:

width = i

ans=max(ans , width\*heights[index])

return ans

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

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

## 21. 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.

-----------------------------------------Code--------------------------------------------------

class Solution(object):

def mergeTwoLists(self, l1, l2):

ans = ListNode(0)

tmp = ans

while l1 and l2:

if l1.val <= l2.val:

tmp.next = l1

tmp = l1

l1= l1.next

else:

tmp.next = l2

tmp = l2

l2 = l2.next

if l1:

tmp.next = l1

else:

tmp.next = l2

return ans.next

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

## 29. Divide Two Integers

Divide two integers without using multiplication, division and mod operator.

If it is overflow, return MAX\_INT.

-----------------------------------------Code--------------------------------------------------

**class Solution(object):**

**def divide(self, dividend, divisor):**

**negative = (dividend > 0 and divisor <0) or (dividend < 0 and divisor > 0)**

**dividend,divisor = abs(dividend),abs(divisor)**

**ans,tmp = 0, 0**

**while(dividend >= divisor):**

**'''begin with subtracting one divisor each time'''**

**tmp = divisor**

**i = 0**

**while(dividend >= tmp):**

**dividend -= tmp**

**'''1<<i reveals how many divisors are subtracted this loop'''**

**ans += (1<<i)**

**tmp <<= 1**

**'''i reveals how many bits c has been right shift'''**

**i += 1**

**if(negative):**

**ans = -ans**

**return min((1 << 31) -1,max( -(1 << 31),ans))**

**obj = Solution()**

**print( obj.divide(-100,11) )**

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

## 53. Maximum Subarray

Find the contiguous subarray within an array (containing at least one number) which has the largest sum.

For example, given the array [-2,1,-3,4,-1,2,1,-5,4],

the contiguous subarray [4,-1,2,1] has the largest sum = 6.

**-----------------------------------------Algorithm-------------------------------------------------**

**Dynamic Programming**

**dp[i] represents the maximum sum among the first i number including dp[i] contiguously.**

**For nums[i], you have two operations toward it, that is adding it to the sum or not.**

**then the equation of transfer: dp[i] = max(nums[i], dp[i-1]+nums[i])**

**in which nums[i] means not adding it to the sum, dp[i-1] + nums[i] means adding it to the sum**

**-----------------------------------------Code--------------------------------------------------**

**class Solution(object):**

**def maxSubArray(self, nums):**

**ans = nums[0]**

**dp = [0]\*len(nums)**

**dp[0] = nums[0]**

**for i in range(1,len(nums)):**

**dp[i] = max(nums[i], dp[i-1]+nums[i])**

**ans = max(ans, dp[i])**

**print(dp)**

**return ans**

**obj=Solution();**

**nums=[1,2]**

**print(obj.maxSubArray(nums))**

**-------------------------------------------------------------------------------------------------**

## 56. Merge Intervals

**Given a collection of intervals, merge all overlapping intervals.**

**For example,**

**Given [1,3],[2,6],[8,10],[15,18],**

**return [1,6],[8,10],[15,18].**

**-----------------------------------------Code--------------------------------------------------**

**class Interval(object):**

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

**self.start = s**

**self.end = e**

**class Solution(object):**

**def merge(self, intervals):**

**ans = []**

**for obj in sorted(intervals,key = lambda a: a.start):**

**if ans and ans[-1].end >= obj.start:**

**ans[-1].end = max(ans[-1].end,obj.end)**

**else:**

**ans.append(obj)**

**return ans**

**intervals = []**

**obj1 = Interval()**

**obj1.\_init\_(3,4)**

**obj2 = Interval()**

**obj2.\_init\_(2,6)**

**obj3 = Interval()**

**obj3.\_init\_(3,10)**

**obj4 = Interval()**

**obj4.\_init\_(10,18)**

**intervals = [obj1,obj2,obj3,obj4]**

**obj = Solution()**

**tmp = obj.merge(intervals)**

**for e in tmp:**

**print("#",e.start,e.end)**

**-------------------------------------------------------------------------------------------------**

## 74. 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.**

**For example,**

**Consider the following matrix:**

**[**

**[1, 3, 5, 7],**

**[10, 11, 16, 20],**

**[23, 30, 34, 50]**

**]**

**Given target = 3, return true.**

**-----------------------------------------Code--------------------------------------------------**

**class Solution:**

**def searchMatrix(self, matrix, target):**

**m,n = len(matrix),len(matrix[0])**

**L,R = 0,m-1**

**mid = (L+R)//2**

**row = matrix[0]**

**while L <= R:**

**if matrix[mid][0] < target:**

**L = mid + 1**

**elif matrix[mid][0] > target:**

**R = mid - 1**

**else:**

**return True**

**mid = (L+R)//2**

**print(mid)**

**row = matrix[mid]**

**print(row)**

**L,R = 0,n - 1**

**mid = (L+R)//2**

**while L <= R:**

**if row[mid] > target:**

**R = mid - 1**

**elif row[mid] < target:**

**L = mid + 1**

**else:**

**return True**

**mid = (L+R)//2**

**return False**

**obj = Solution()**

**nums = [**

**[1, 3, 5, 7],**

**[10, 11, 16, 20],**

**[23, 30, 34, 50]**

**]**

**print( obj.searchMatrix(nums,2))**

**-------------------------------------------------------------------------------------------------**

**-------------------------------------------------------------------------------------------------**

## 75. Sort Colors

**Given an array with n objects colored red, white or blue, sort them 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.**

**click to show follow up.**

**Follow up:**

**A rather straight forward solution is a two-pass algorithm using counting sort.**

**First, iterate the array counting number of 0's, 1's, and 2's, then overwrite array with total number of 0's, then 1's and followed by 2's.**

**Could you come up with an one-pass algorithm using only constant space?**

**-----------------------------------------Algorithm-------------------------------------------------**

**p0/p2 is index where next 0/2 should be**

**-----------------------------------------Code--------------------------------------------------**

**class Solution:**

**def sortColors(self, nums):**

**length = len(nums)**

**p0,p2,pivot = 0,length-1,0**

**while pivot <= p2:**

**if nums[pivot] == 2:**

**nums[pivot],nums[p2] = nums[p2],nums[pivot]**

**p2 -= 1**

**elif nums[pivot] == 0:**

**nums[pivot],nums[p0] = nums[p0],nums[pivot]**

**p0 += 1**

**#because nums[pivot] is 0 or 1, so pivot should increase by 1**

**#when it's 0, it should be in the left of pivot**

**#when it's 1, we should ignore it**

**pivot += 1**

**else:**

**pivot += 1**

**return nums**

**obj = Solution()**

**nums= [2,1,0,1,0]**

**print( obj.sortColors(nums))**

## 76. 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).**

**For example,**

**S = "ADOBECODEBANC"**

**T = "ABC"**

**Minimum window is "BANC".**

**Note:**

**If there is no such window in S that covers all characters in T, return the empty string "".**

**If there are multiple such windows, you are guaranteed that there will always be only one unique minimum window in S.**

**-----------------------------------------Algorithm-------------------------------------------------**

**-----------------------------------------Code--------------------------------------------------**

**class Solution:**

**# @return a string**

**def minWindow(self, S, T):**

**t\_cnt={}**

**start=0; minSize=100000; minStart=0**

**for char in T:**

**if char not in t\_cnt:**

**t\_cnt[char]=1**

**else: t\_cnt[char]+=1**

**count=len(T)**

**for end in range(len(S)):**

**if S[end] in T:**

**t\_cnt[S[end]]-=1**

**if t\_cnt[S[end]]>=0:**

**count-=1**

**if count==0:**

**while True:**

**if S[start] in T:**

**#t\_cnt[i]<0 means the character i in S[start:end] outnumbers that in T**

**if t\_cnt[S[start]]<0:**

**t\_cnt[S[start]]+=1**

**else:**

**break**

**start+=1**

**if minSize>end-start+1:**

**minSize=end-start+1**

**minStart=start**

**if minSize==100000:**

**return ''**

**else:**

**return S[minStart:minStart+minSize]**

**obj = Solution()**

**s="ADOBECODEBANC"**

**t="ABC"**

**print( obj.minWindow(s,t) ) obj = Solution()**

**s="ADOBECODEBANC"**

**t="ABC"**

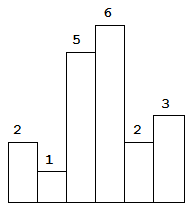
**print( obj.minWindow(s,t) )**

**#http://www.cnblogs.com/zuoyuan/p/3785421.html**

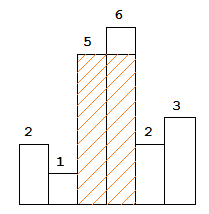
**-------------------------------------------------------------------------------------------------**

## 84. 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.



Above is a histogram where width of each bar is 1, given height = [2,1,5,6,2,3].



The largest rectangle is shown in the shaded area, which has area = 10 unit.

For example,

Given heights = [2,1,5,6,2,3],

return 10.

**-----------------------------------------Code--------------------------------------------------**

class Solution(object):

def largestRectangleArea(self, heights):

stack=[0]

heights.append(0)

width=0

i=0

ans=-1

Len=len(heights)

while (i < Len) :

if( not stack or heights[i] > heights[stack[-1]] ):

stack.append(i)

i += 1

else:

index = stack.pop()

if(stack):

width = i - stack[-1] - 1

else:

width = i

ans=max(ans , width\*heights[index])

return ans

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

## 85. 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.

For example, given the following matrix:

1 0 1 0 0

1 0 1 1 1

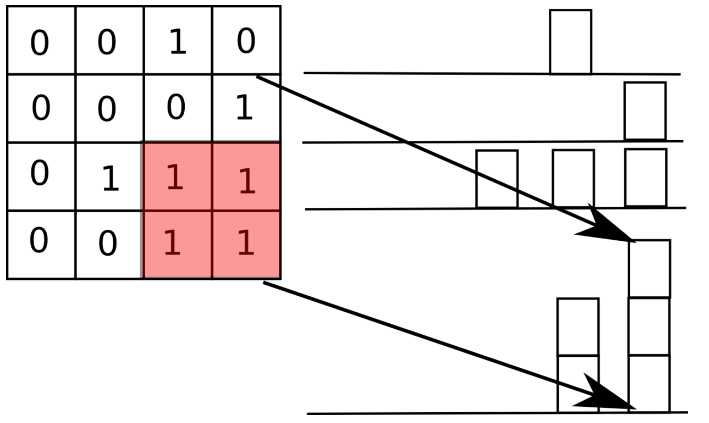
1 1 1 1 1

1 0 0 1 0

Return 6.

**-----------------------------------------Algorithm-------------------------------------------------**

Proof without words:



Then the problem is transferred to “84. Largest Rectangle in Histogram”

**-----------------------------------------Code--------------------------------------------------**

class Solution(object):

def maximalRectangle(self, matrix):

if (not matrix):

return 0;

height = [0 for i in range(len(matrix[0]))]

ans=-1

for row in matrix:

for i in range(len(row)):

if( row[i] == '0'):

height[i] = 0

else:

height[i] += 1

print(height)

ans = max (ans, self.cal(height) )

return ans

def cal(self, heights):

if (not heights):

return 0

ans = -1

heights.append(0)

stack = [0]

length = len(heights)

i = 0

while i < length:

cur\_num = heights[i]

if( not stack or cur\_num > heights[ stack[-1] ] ):

stack.append(i)

i += 1

else:

index = stack.pop()

if (stack):

width = i - stack[-1] - 1

else:

width = i

ans = max( ans, width\*heights[index])

return ans

obj=Solution();

nums1=[[1,0,1,0,0],[1,0,1,1,1],[1,1,1,1,1],[1,0,0,1,0] ]

nums2=["10100","10111","11111","10010"]

'''

pay attenton to the difference between

["10100","10111","11111","10010"]

and

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

The difference are :

print(type('0') )

print(type(0) )

'''

print(obj.maximalRectangle(nums2))

88. Merge Sorted Array

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

Note:

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

-----------------------------------------Code--------------------------------------------------

class Solution(object):

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

while n > 0:

if m <= 0 or nums2[n-1] >= nums1[m-1]:

nums1[m+n-1] = nums2[n-1]

n -= 1

else:

nums1[m+n-1] = nums1[m-1]

m -= 1

print(nums1)

obj = Solution()

nums1 = [0]

nums2 = [1]

print( obj.merge(nums1,0,nums2,1) )

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

## 89. Gray Code

The gray code is a binary numeral system where two successive values differ in only one bit.

Given a non-negative integer n representing the total number of bits in the code, print the sequence of gray code. A gray code sequence must begin with 0.

For example, given n = 2, return [0,1,3,2]. Its gray code sequence is:

00 - 0

01 - 1

11 - 3

10 - 2

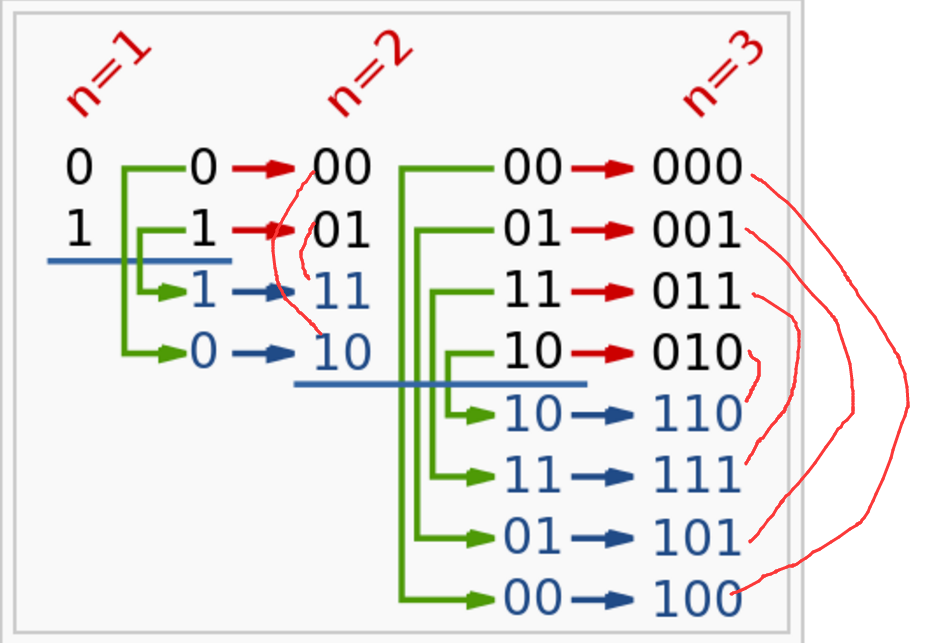
Note:

For a given n, a gray code sequence is not uniquely defined.

For example, [0,2,3,1] is also a valid gray code sequence according to the above definition.

For now, the judge is able to judge based on one instance of gray code sequence. Sorry about that.

-----------------------------------------Algorithm-------------------------------------------------



**Just find the law.**

-----------------------------------------Code--------------------------------------------------

class Solution:

def grayCode(self,n):

if not n:

return [0]

ans = [0]\*2\*\*n

ans[0],ans[1] = 0,1

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

tmp = 2\*\*(i-1)

for j in range(tmp):

ans[tmp+j] = ans[tmp-j-1] + tmp

return ans

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

## 137. Single Number II

**Given an array of integers, every element appears three times except for one. Find that single one.**

**Note:**

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

出现3次，找出1次的

我的：统计每个位在所有数中出现总次数，最后模3余1的数就是

缺点：有用额外内存

**-----------------------------------------Code--------------------------------------------------**

public class Solution {

public int singleNumber(int[] nums) {

int [] a = new int[35];

int ans=0,len=nums.length,tmp;

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

tmp=1;

for(int j=0;j<32&&nums[i]!=0;j++){

if((nums[i]&tmp)!=0){

a[j]+=1;

nums[i]-=tmp;

}

tmp<<=1;

}

}

tmp=1;

for(int i=0;i<32;i++){

if(a[i]%3==1){

ans+=tmp;

}

tmp<<=1;

}

return ans;

}

}

网友版本1：分别从第0到32位的每一位进行处理，每一位出现总次数模3

public class Solution {

public int singleNumber(int nums[]) {

int res = 0;

for (int i = 0; i < 32; ++i) {

int sum = 0;

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

sum += (nums[j] >> i) & 1;

}

res += (sum % 3) << i;

}

return res;

}

}

网友版本2： 考虑每一位的统计值，如果累加到3就归为0，则只会有0/1/2三种情况，所以将大小为32的int数组改为只用两个int即可。

第一个int的第i位为0/1代表第i位当前累加有0/1个 1 ，第二个int的第i位为1代表第i位当前累加有2。

当int1和int2中的第i位均为1时，我们将他们都清零。

例如：数组 [4 7 4 4](这里为了方便，我们取int的最后四位)

4　　0　　1　　0　　0

　 one 0　　1　　0　　0

two 0 0　　0　　0

=====================

　　　7　　0　　1　　1　　1

one 0　　0　　1　　1

two 0 1　　0　　0

=====================

4　　0　　1　　0　　0

one 0　　0　　0　　0

two 0 0　　1　　1

=====================

4　　0　　1　　0　　0

　　one 0　　1　　0　　0

two 0 0　　0　　0

=====================

python代码：

def SingleNumber2\_2(Array):

one = 0

two = 0

three = 0

for i in range(0, len(Array)):

two |= (one & Array[i]) //假设one某位是1，two某位必定为0（否则此位出现3次），1^1=1，two某位=1|1=1；1^0=0,two=0|0=0

one ^= Array[i] //假设one某位是1，1^1=0，出现两次后归0；1^0=1，新的为0则还是只出现1次；

three = one & two //以下三行用于归0(two\one都减去同为1的那个位，同为1则出现了3次）

two -= three

one -= three

return one

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

## 152. Maximum Product Subarray

Find the contiguous subarray within an array (containing at least one number) which has the largest product.

For example, given the array [2,3,-2,4],

the contiguous subarray [2,3] has the largest product = 6.

**-----------------------------------------Algorithm-------------------------------------------------**

**Dynamic Programming**

max\_dp[i] represents the the maximum product among the first i number ***including*** nums[i] contiguously.

min\_dp[i] represents the the minimum product among the first i number ***including*** nums[i] contiguously.

then if nums[i] > 0:

max\_dp[i] = max(nums[i],max\_dp[i-1]\*nums[i])

min\_dp[i] = min(nums[i],min\_dp[i-1]\*nums[i])

else:

max\_dp[i] = max(nums[i],min\_dp[i-1]\*nums[i])

min\_dp[i] = min(nums[i],max\_dp[i-1]\*nums[i])

**-------------------------------------------Code---------------------------------------------------**

class Solution(object):

def maxProduct(self, nums):

max\_dp = [0]\*len(nums)

min\_dp = [0]\*len(nums)

max\_dp[0] = nums[0]

min\_dp[0] = nums[0]

ans = nums[0]

for i in range(1,len(nums)):

if nums[i] > 0:

max\_dp[i] = max(nums[i],max\_dp[i-1]\*nums[i])

min\_dp[i] = min(nums[i],min\_dp[i-1]\*nums[i])

else:

max\_dp[i] = max(nums[i],min\_dp[i-1]\*nums[i])

min\_dp[i] = min(nums[i],max\_dp[i-1]\*nums[i])

ans = max(ans, max\_dp[i])

print(max\_dp)

print(min\_dp)

return ans

obj=Solution();

nums=[4]

print(obj.maxProduct(nums))

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

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.

-----------------------------------------Algorithm-------------------------------------------------

**The key is to push (x,curMin). And the curMin means the minimum of those below the x(included)**

-------------------------------------------Code---------------------------------------------------

class MinStack(object):

min = 1 << 31

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

"""

initialize your data structure here.

"""

self.array = []

def push(self, x):

"""

:type x: int

:rtype: void

"""

curMin = self.getMin()

if curMin == None or x < curMin:

curMin = x

self.array.append((x, curMin));

def pop(self):

"""

:rtype: void

"""

self.array.pop()

def top(self):

"""

:rtype: int

"""

if not self.array:

return None

return self.array[len(self.array)-1][0]

def getMin(self):

"""

:rtype: int

"""

if not self.array:

return None

return self.array[len(self.array)-1][1]

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

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

## 167. Two Sum II - Input array is sorted

**Given an array of integers that is already sorted in ascending order, find two numbers such that they add up to a specific target number.**

**The function twoSum should return indices of the two numbers such that they add up to the target, where index1 must be less than index2. Please note that your returned answers (both index1 and index2) are not zero-based.**

**You may assume that each input would have exactly one solution.**

**Input: numbers={2, 7, 11, 15}, target=9**

**Output: index1=1, index2=2**

**-----------------------------------------Algorithm-------------------------------------------------**

**Binary Search**

**-------------------------------------------Code---------------------------------------------------**

我的解法：

#include <stdlib.h>

#include <cstring>

#include<vector>

#include<iostream>

#include<algorithm>

class Solution {

public:

std::vector<int> twoSum(std::vector<int>& numbers, int target) {

std::vector<int>::iterator L,R,M;

std::vector<int>ans;

for(std::vector<int>::iterator i=numbers.begin();i<numbers.end();i++){

L=i+1,R=numbers.end(),M=L+(R-L)/2; //迭代器类似指针，不能写成（L+R)/2

while(L<=R){ //如果没有=，则4个元素时最小的可能取不到

if(target>\*i+\*M){ //迭代器类似指针，访问需要用\*

L=M+1;

M=L+(R-L)/2;

}

else if(target<\*i+\*M){

R=M-1;

M=L+(R-L)/2;

}

else{

ans.push\_back(distance(numbers.begin(),i)+1); //题目中数组下标从1开始

ans.push\_back(distance(numbers.begin(),M)+1);

return ans;

}

}

}，

return ans; //不加这一行，提交无法通过

}

};

int main(){

std::vector<int>v;

v.push\_back(0);

v.push\_back(0);

v.push\_back(3);

v.push\_back(4);

Solution obj;

std::vector<int>ans(obj.twoSum(v,4));

for(std::vector<int>::iterator i=ans.begin();i<ans.end();i++){

std::cout<<\*i<<std::endl;

}

return 0;

}

python 版本

class Solution(object):

def twoSum(self, numbers, target):

for i in range(len(numbers)):

l, r = 0, len(numbers)-1 # 覆盖全部的index

tmp = target - numbers[i]

while l < r:

sum = numbers[l] + numbers[r]

if sum < target: # 最左往右移动

l += 1

elif sum > target: # 最右往左移动

r -= 1

else:

return l + 1, r + 1 # 根据题意，返回list的index+1

obj=Solution()

nums=[0,0,1,4]

print(obj.twoSum(nums,5))

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

## 89. Rotate Array

Rotate an array of n elements to the right by k steps.

For example, with n = 7 and k = 3, the array [1,2,3,4,5,6,7] is rotated to [5,6,7,1,2,3,4].

Note:

Try to come up as many solutions as you can, there are at least 3 different ways to solve this problem.

-----------------------------------------Algorithm-------------------------------------------------

time:O(n) ------- space:O(1)------------ reverse 3 times

-----------------------------------------Code--------------------------------------------------

class Solution:

def rotate(self, nums, k):

n = len(nums)

k = k%n

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

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

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

print("ans",nums)

def reverse(self,nums,L,R):

mid = (L + R)//2

i = 0

while i <= mid - L:

nums[L + i],nums[R - i] = nums[R - i], nums[L + i]

i += 1

print(nums)

obj = Solution()

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

k = 11

print( obj.rotate(nums,k) )

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

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

## 198. House Robber

Algorithm: dp

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.

**-----------------------------------------Algorithm-------------------------------------------------**

设dp[i]为到i时偷盗的最大数目。

每到一个house，可以选择偷或不偷。选择偷的话dp[i]=dp[i-2]+nums[i];选择不偷的话dp[i]=dp[i-1]。

dp[i] = max(num[i] + dp[i - 2], dp[i - 1]), 由此看出我们需要初始化dp[0]和dp[1]，其中dp[0]即为num[0]，dp[1]此时应该为max(num[0], num[1])，

**-----------------------------------------Code--------------------------------------------------**

class Solution(object):

def rob(self, nums):

size=len(nums)

if(size==1):

return nums[0]

if(size==0):

return 0

dp=[nums[0]]

if(nums[1]>nums[0]):

dp.append(nums[1])

else:

dp.append(nums[0])

for index in range(2,size):

dp.append(max(dp[index-1],dp[index-2]+nums[index]))

return dp[size-1]

obj=Solution();

nums=[1,2,3,4,5]

print(obj.rob(nums))

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

## 200. Number of Islands

Given a 2d grid map of '1's (land) and '0's (water), count the number of islands. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Example 1:

11110

11010

11000

00000

Answer: 1

Example 2:

11000

11000

00100

00011

Answer: 3

**-----------------------------------------Algorithm-------------------------------------------------**

**dfs**

**key: grid is immutable so there must be a visited[[]].**

**-----------------------------------------Code-------------------------------------------------**

**class Solution(object):**

**def numIslands(self, grid):**

**if not grid:**

**return 0**

**m,n=len(grid),len(grid[0])**

**visited = [[False]\*n for x in range(m)]**

**ans = 0**

**for i in range(m):**

**for j in range(n):**

**if( grid[i][j] == '1' and not visited[i][j]):**

**print("i=",i,"j=",j,"ans=",ans+1)**

**ans += 1**

**self.dfs(m,n,i,j,grid, visited)**

**return ans**

**def dfs(self,m,n,x,y,grid,visited):**

**if( x > 0 and grid[x-1][y] == '1' and visited[x-1][y] == False):**

**visited[x-1][y] = True**

**self.dfs(m,n,x-1,y,grid,visited)**

**if( x < m - 1 and grid[x+1][y] == '1' and visited[x+1][y] == False):**

**visited[x+1][y] = True**

**self.dfs(m,n,x+1,y,grid,visited)**

**if( y > 0 and grid[x][y-1] == '1' and visited[x][y-1] == False):**

**visited[x][y-1] = True**

**self.dfs(m,n,x,y-1,grid,visited)**

**if( y < n - 1 and grid[x][y+1] == '1' and visited[x][y+1] == False):**

**visited[x][y+1] = True**

**self.dfs(m,n,x,y+1,grid,visited)**

**obj = Solution()**

**board=['11000','11000','00100','00011']**

**print( obj.numIslands(board) )**

**-------------------------------------------------------------------------------------------------**

**-----------------------------------------Algorithm-------------------------------------------------**

**-----------------------------------------Code-------------------------------------------------**

**-------------------------------------------------------------------------------------------------**

## 213. House Robber II

Note: This is an extension of House Robber.

After robbing those houses on that street, the thief has found himself a new place for his thievery so that he will not get too much attention. This time, all houses at this place are arranged in a circle. That means the first house is the neighbor of the last one. Meanwhile, the security system for these houses remain the same as for those in the previous street.

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.

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

**-----------------------------------------Algorithm-------------------------------------------------**

**Dynamic Programming**

**There are two cases here:**

**1) 1st element is included and last is not included.**

**2) 1st is not included and last is included.**

**Therefore, we can use the similar dynamic programming approach to scan the array twice and get the larger value.**

**What is interesting is how that algorithm comes to be?**

**Well, the key point is to handle the first and last house:**

1. **Now support the house[1] is robbed, so the house[n] should not be robbed. That is equivalent to considering only the house[1] – house[n-1]**
2. **Support the house[1] is not robbed and :**
3. **the house[n-1] is not robbed either, then the house[n] should be robbed. That is equivalent to considering only the house[2]-house[n]**
4. **the house[n-1] is robbed, so the house[n] shouldn’t be robbed, which is equivalent to considering only house[2]-house[n-1], that is included both by considering house[1]-house[n-1] and by considering house[2]-house[n]**

**------------------------------------------Code----------------------------------------------------**

class Solution(object):

def rob(self, nums):

if(not nums):

return 0

if(len(nums) == 1):

return nums[0]

if(len(nums) ==2 ):

return max(nums[0], nums[1])

return max(self.rob\_do(nums,0,len(nums)-2),self.rob\_do(nums,1,len(nums)-1))

def rob\_do(self,nums,i,j):

print("i""j",i,j)

dp = [0]\*len(nums)

dp[i] = nums[i]

dp[i+1] = max(nums[i], nums[i+1])

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

dp[index] = max(dp[index-1], dp[index-2]+nums[index])

print(dp)

return dp[j]

obj=Solution();

nums=[1,2]

print(obj.rob(nums))

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

## 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.

For example,

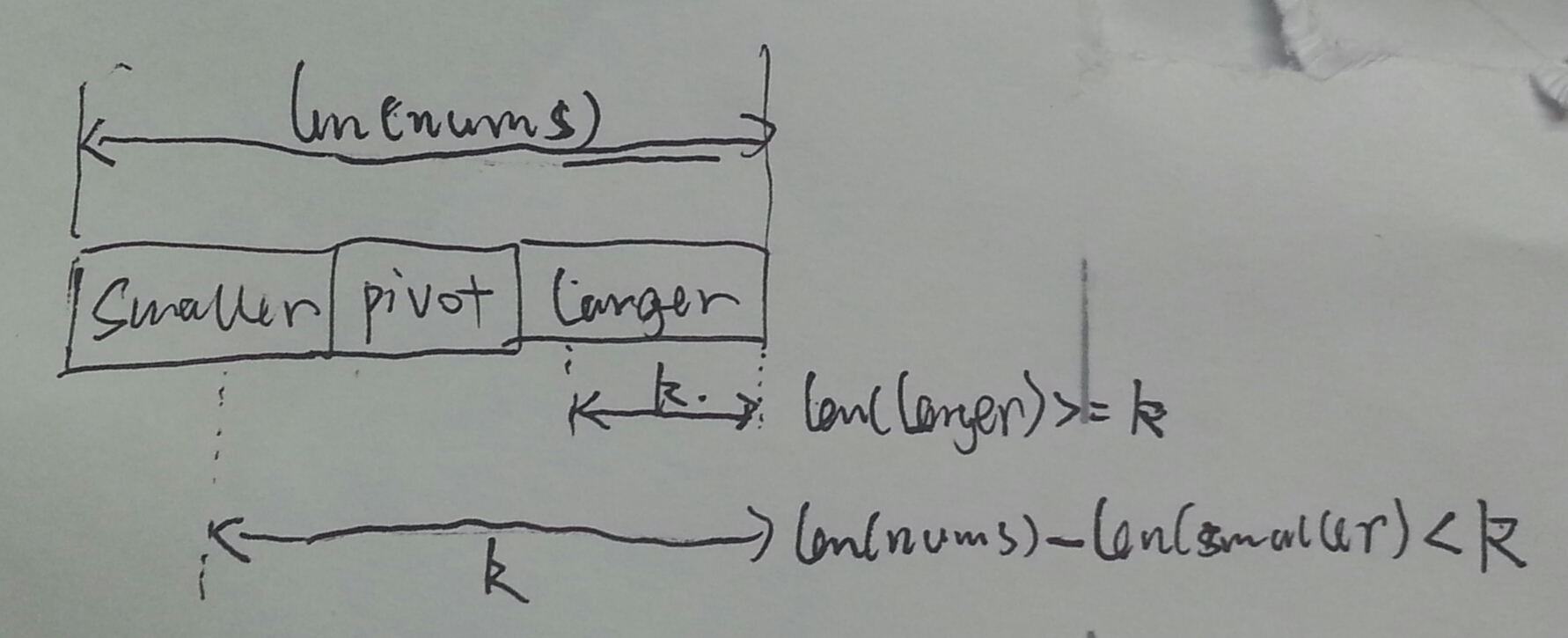
Given [3,2,1,5,6,4] and k = 2, return 5.

Note:

You may assume k is always valid, 1 ≤ k ≤ array's length.

-----------------------------------------Algorithm-------------------------------------------------

split the array into 3 parts: just as shown in the picture



if pivot is not random, is nums[0] for example, the performance is bad is the array is sorted.

------------------------------------------Code----------------------------------------------------

import random

class Solution:

def findKthLargest(self, nums, k):

pivot = random.choice(nums)

larger = [i for i in nums if i > pivot]

smaller = [i for i in nums if i < pivot]

if len(larger) >= k:

return self.findKthLargest(larger,k)

elif len(nums)-len(smaller) < k:

return self.findKthLargest(smaller,k-(len(nums)-len(smaller)))

return pivot

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

## 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.

For example, given the following matrix:

1 0 1 0 0

1 0 1 1 1

1 1 1 1 1

1 0 0 1 0

Return 4.

**-----------------------------------------Algorithm-------------------------------------------------**

**A little difference from the Problem 85**

**------------------------------------------Code----------------------------------------------------**

class Solution(object):

def maximalSquare(self, matrix):

if (not matrix):

return 0;

height = [0 for i in range(len(matrix[0]))]

ans=-1

for row in matrix:

for i in range(len(row)):

if( row[i] == '0'):

height[i] = 0

else:

height[i] += 1

print(height)

ans = max (ans, self.cal(height) )

return ans

def cal(self, heights):

if (not heights):

return 0

ans = -1

heights.append(0)

stack = [0]

length = len(heights)

i = 0

while i < length:

cur\_num = heights[i]

if( not stack or cur\_num > heights[ stack[-1] ] ):

stack.append(i)

i += 1

else:

index = stack.pop()

if (stack):

width = i - stack[-1] - 1

else:

width = i

if(width >= heights[index]): '''here is the difference from the code of Problem 85'''

ans = max( ans, heights[index]\*heights[index]) '''here is the difference from the code of Problem 85'''

return ans

obj=Solution();

nums1=[[1,0,1,0,0],[1,0,1,1,1],[1,1,1,1,1],[1,0,0,1,0] ]

nums2=["10100","10111","11111","10010"]

'''

pay attenton to the difference between

["10100","10111","11111","10010"]

and

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

The difference are :

print(type('0') )

print(type(0) )

'''

print(obj.maximalSquare(nums2))

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

## 238. Product of Array Except Self

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

Solve it without division and in O(n).

For example, given [1,2,3,4], return [24,12,8,6].

Follow up:

Could you solve it with constant space complexity? (Note: The output array does not count as extra space for the purpose of space complexity analysis.)

**-----------------------------------------Algorithm-------------------------------------------------**

**Dynamic Programming**

**dp[i] = nums[0]\*nums[1]\*…\*nums[i-1]\*nums[i+1]\*…\*nums[n-1]**

**Firstly, for each nums[i], we refer to the product of nums[j] (j<i) as dp[i]**

**Secondly, for each nums[i], we refer to the product of nums[k] (i < k) as right.**

**Lastly, dp[i] = dp[i]\*right**

**-----------------------------------------Code--------------------------------------------------**

**O(m\*log(n))**

class Solution(object):

def productExceptSelf(self, nums):

dp = [0]\*len(nums)

dp[0] = 1

dp[1] = nums[0]

for i in range(2,len(nums)):

dp[i] = dp[i-1]\*nums[i-1]

print("@",dp)

right = nums[len(nums)-1]

for i in range(0,len(nums)-1)[::-1]:

dp[i] = right\*dp[i]

right \*= nums[i]

return dp

obj=Solution();

nums=[1,0]

print(obj.productExceptSelf(nums))

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

## 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.

For example,

Given nums = [1,3,-1,-3,5,3,6,7], and k = 3.

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

Therefore, return the max sliding window as [3,3,5,5,6,7].

Note:

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

Follow up:

Could you solve it in linear time?

-----------------------------------------Algorithm-------------------------------------------------

from left to right, not increased double-end queue

-----------------------------------------Code--------------------------------------------------

from collections import deque

class Solution(object):

def maxSlidingWindow(self, nums, k):

dq = deque()

ans = []

for i in range(len(nums)):

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

dq.pop()

dq.append(nums[i])

if i - k >= 0 and dq[0] == nums[ i - k ]:

dq.popleft()

if i >= k - 1:

ans.append(dq[0])

return ans

obj = Solution()

nums=[1,-1]

print( obj.maxSlidingWindow(nums,1) )

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

## 240. Search a 2D Matrix II

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

Integers in each row are sorted in ascending from left to right.

Integers in each column are sorted in ascending from top to bottom.

For example,

Consider the following matrix:

[

[1, 4, 7, 11, 15],

[2, 5, 8, 12, 19],

[3, 6, 9, 16, 22],

[10, 13, 14, 17, 24],

[18, 21, 23, 26, 30]

]

Given target = 5, return true.

Given target = 20, return false.

-----------------------------------------Code--------------------------------------------------

**O(m\*log(n))**

class Solution:

def searchMatrix(self, matrix, target):

for row in matrix:

if row[0] > target:

continue

elif row[0] == target:

return True

elif self.bin\_search(row,target):

return True

return False

def bin\_search(self,row,target):

L,R = 0,len(row)-1

mid = (L+R)//2

while L <= R:

if target < row[mid]:

R = mid - 1

elif target > row[mid]:

L = mid + 1

else:

return True

mid = (L+R)//2

return False

obj = Solution()

nums= [

[1, 4, 7, 11, 15],

[2, 5, 8, 12, 19],

[3, 6, 9, 6, 22],

[10, 13, 14, 17, 24],

[18, 21, 23, 26, 30]

]

print( obj.searchMatrix(nums,16))

**O(m+n)**

class Solution:

def searchMatrix(self, matrix, target):

col,row = len(matrix[0])- 1,0

while row < len(matrix) and col >=0:

if target < matrix[row][col]:

col -= 1

elif target > matrix[row][col]:

row += 1

else:

return True

return False

obj = Solution()

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

print( obj.searchMatrix(nums,2))

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

## 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.

**-----------------------------------------Algorithm-------------------------------------------------**

**Dynamic Programming**

**cost[i][j] represents the minimum cost when painting the ith house with color j**

**Then the transfer equation is :**

**cost[i][0] = min(cost[i-1][1], cost[i-1][2]) + costs[i][0]**

**cost[i][1] = min(cost[i-1][0], cost[i-1][2]) + costs[i][1]**

**cost[i][2] = min(cost[i-1][0], cost[i-1][1]) + costs[i][2]**

**-----------------------------------------Code--------------------------------------------------**

**class Solution(object):**

**def minCost(self, costs):**

**if len(costs) == 0:**

**return 0**

**cost = [[0 for i in range(3)] for j in range(len(costs))]**

**cost[0][0] = costs[0][0]**

**cost[0][1] = costs[0][1]**

**cost[0][2] = costs[0][2]**

**i = 1**

**while i < len(costs):**

**cost[i][0] = min(cost[i-1][1], cost[i-1][2]) + costs[i][0]**

**cost[i][1] = min(cost[i-1][0], cost[i-1][2]) + costs[i][1]**

**cost[i][2] = min(cost[i-1][0], cost[i-1][1]) + costs[i][2]**

**i += 1**

**return min(cost[i-1][0], min(cost[i-1][1], cost[i-1][2]))**

**-------------------------------------------------------------------------------------------------**

## 258. Add Digits

**Algorithm:递归&数字根**

**Given a non-negative integer num, repeatedly add all its digits until the result has only one digit.**

**For example:**

**Given num = 38, the process is like: 3 + 8 = 11, 1 + 1 = 2. Since 2 has only one digit, return it.**

**Follow up:**

**Could you do it without any loop/recursion in O(1) runtime?**

**-----------------------------------------Code--------------------------------------------------**

递归：

class Solution(object):

def addDigits(self,num):

sum=0

if(num<10):

return num

while(num>0):

sum+=num%10

num=num//10 #整除是//

return self.addDigits(sum)

数字根：如题的叠加结果就是数字根。小于10的正数的树根是它本身，大于9且不是9的倍数的树根都是对9取余，9的倍数的树根是9

class Solution(object):

def addDigits(self,num):

if(num<10):

return num

if(num%9==0):

return 9

return num%9

obj=Solution()

print(obj.addDigits(18))

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

## 260. Single Number III

**-----------------------------------------Algorithm-------------------------------------------------**

**位运算**

**找出两个出现一次的数**

**以a、b最低的不相同的位，记作bits,将所有数分成两组**

**所有数异或=a^b**

**A组bits跟a相同；B组bits跟b相同。分别对A、B进行异或得到a,b.**

bits=a^b&(~a^b+1)

**-----------------------------------------Code--------------------------------------------------**

public class Solution {

public int[] singleNumber(int[] nums) {

int []ans=new int[2];

int len=nums.length,XOR=nums[0],bits;

for(int i=1;i<len;i++){

XOR^=nums[i];

}

bits=XOR&(~XOR+1);

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

if((nums[i]&bits)==0) ans[0]^=nums[i];

else ans[1]^=nums[i];

}

return ans;

}

}

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

## 264. Ugly Number II Question

**Write a program to find the n-th ugly number.**

**Ugly numbers are positive numbers whose prime factors only include 2, 3, 5. For example, 1, 2, 3, 4, 5, 6, 8, 9, 10, 12 is the sequence of the first 10 ugly numbers.**

**Note that 1 is typically treated as an ugly number.**

**-----------------------------------------Algorithm-------------------------------------------------**

**Dynamic Programming**

**set**

**-----------------------------------------Code--------------------------------------------------**

#include <cstdlib>

#include <cstdio>

#include <cstring>

#include<set>

#include<iostream>

class Solution {

public:

int nthUglyNumber(int n) {

std::set<long long> order;

order.insert(1);

int count = 0;

long long curmin = 0;

while(count < n)

{

curmin = \*(order.begin());

order.erase(order.begin());

order.insert(curmin \* 2);

order.insert(curmin \* 3);

order.insert(curmin \* 5);

count ++;

}

return (int)curmin;

}

};

int main(){

Solution obj;

std::cout<<obj.nthUglyNumber(1352);

return 0;

}

一个因子是2，3，5中的一个，另一个因子在 uglys 数组中，该因子下标存储在 ids 数组中：

a b c 分别是2 3 5的倍数，且ids0,ids1,ids2分别都会遍历uglys中的所有数。这样， uglys 数组

中每个数的2 3 5倍数都被考虑到

public class Solution {

public int nthUglyNumber(int n) {

int[] ids = new int[3];

int[] uglys = new int[n];

uglys[0] = 1;

for(int i = 1; i < n; i++) {

int a = uglys[ids[0]] \* 2;

int b = uglys[ids[1]] \* 3;

int c = uglys[ids[2]] \* 5;

int min = Math.min(a, Math.min(b, c));

if(a == min) ids[0]++;

if(b == min) ids[1]++;

if(c == min) ids[2]++;

uglys[i] = min;

}

return uglys[n-1];

}

}

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

## 276. Paint Fence

There is a fence with n posts, each post can be painted with one of the k colors.

You have to paint all the posts such that no more than two adjacent fence posts have the same color.

Return the total number of ways you can paint the fence.

Note: n and k are non-negative integers.

**-----------------------------------------Algorithm-------------------------------------------------**

Consider posts from 1 ~ n. Now we look at last post, marked n:

S(n) means: last 2 fence posts have same color.

Note: S(n) will equal to whatever that's on n-1 position，

Also, just because n and n-1 are same, that means n-2 and n-1 have to be differnet.

SO: S(n) = D(n - 1)

D(n) means: last 2 fence posts have different color.

Note: for n - 1, and n-2 positions, we have 2 different conditions: For example: xxy, or wxy, same 2 x's or different w vs. x.

So: D(n) = (k - 1) \* (D(n - 1) + S(n - 1))

We can also create dp(n) = S(n) + D(n); // dp (n) is our totoal results. Will need to return dp (n);

Use above equations to figure out dp (n)

dp (n) = S(n) + D(n) = D(n - 1) + (k - 1) \* (D(n - 1) + S(n - 1))

= D(n - 1) + (k - 1)( dp (n - 1))

= (k - 1) \* (D(n - 2) + S(n - 2)) + (k - 1)( dp (n - 1))

= (k - 1)( dp (n - 1) + dp (n - 2))

Since n-2 >=1, so n>=3. We need fiture out cases for n = 0,1,2,3

**-----------------------------------------Algorithm-------------------------------------------------**

For any continuous 3 posts, the only not allowed permutation is “aaa” in which “a” represents one kind of color.

That means the color of the third post is the same as the color of the first and the second post, that is:

“aaa” = “third = second AND third = first”

Since that is the only not allowed permutation, the oppose of it is allowed, according to the De Morgan's laws:

the oppose of “aaa”= “third !=second OR third !=first”

“third != second” means (k-1)\* dp (n-1) ; “third != first”means (k-2)\* dp (n-2) ; “OR” means the plus of 2 permutation, then the transfer equation is:

dp (n) = (k-1)\* dp (n-1) + (k-1)\* dp (n-2)

**-----------------------------------------Code--------------------------------------------------**

public class Solution {

public int numWays(int n, int k) {

if (n <= 1 || k <= 0) {

return n \* k;

}

int[] dp = new int[n + 1]; //index based : 1

dp[1] = k;

dp[2] = k\*k;

for (int i = 3; i <= n; i++) {

dp[i] = (k - 1) \* (dp[i - 1] + dp[i - 2]);

}

return dp[n];

}

}-------------------------------------------------------------------------------------------------

## 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.

Follow up:

Could you solve it in O(nk) runtime?

**-----------------------------------------Algorithm-------------------------------------------------**

The idea is similar to the problem Paint House I, for each house and each color, the minimum cost of painting the house with that color should be the minimum cost of painting previous houses, and make sure the previous house doesn't paint with the same color.

We can use min1 and min2 to track the indices of the 1st and 2nd smallest cost till previous house, if the current color's index is same as min1, then we have to go with min2, otherwise we can safely go with min1.

The code below modifies the value of costs[][] so we don't need extra space.

**-----------------------------------------Code--------------------------------------------------**

def minCostII(self, costs):

if not costs or len(costs) == 0:

return 0

n, k = len(costs), len(costs[0])

min1 = min2 = -1

for i in range(n):

last1, last2 = min1, min2

min1 = min2 = -1

for j in range(k):

if j != last1:

costs[i][j] += 0 if last1 < 0 else costs[i - 1][last1]

else:

costs[i][j] += 0 if last2 < 0 else costs[i - 1][last2]

if min1 < 0 or costs[i][j] < costs[i][min1]:

min2, min1 = min1, j

elif min2 < 0 or costs[i][j] < costs[i][min2]:

min2 = j

return costs[n - 1][min1]

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

## 280: Wiggle Sort

Problem Description:

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

For example, given nums = [3, 5, 2, 1, 6, 4], one possible answer is [1, 6, 2, 5, 3, 4].

The final sorted nums needs to satisfy two conditions:

If i is odd, then nums[i] >= nums[i - 1];

If i is even, then nums[i] <= nums[i - 1].

The code is just to fix the orderings of nums that do not satisfy 1 and 2.

-----------------------------------------Code--------------------------------------------------

for (int i = 1; i < n; i++)

if (((i & 1) && nums[i] < nums[i - 1]) || (!(i & 1) && nums[i] > nums[i - 1]))

swap(nums[i], nums[i - 1]);

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

-----------------------------------------Code--------------------------------------------------

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

## 287. Find the Duplicate Number

**-----------------------------------------Algorithm-------------------------------------------------**

**Binary Search**

**n+1个1-n的数，只有1个是重复的，找出这个数**

**二分查找**

**-----------------------------------------Code--------------------------------------------------**

package test\_leetcode;

public class Solution {

public int findDuplicate(int[] nums) {

int len=nums.length,sum=0;

int L=1,R=len-1,mid=(L+R)/2;

while(L<R){

for(int i=0;i<len&&sum<=(R-L)/2+1;i++){

if(nums[i]<=mid&&nums[i]>=L)sum++;

}

if(sum>(R-L)/2+1){

R=mid;

}

else{

L=mid+1;

}

mid=(L+R)/2;

sum=0;

}

return mid;

}

}

## 324. Wiggle Sort II

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

Example:

(1) Given nums = [1, 5, 1, 1, 6, 4], one possible answer is [1, 4, 1, 5, 1, 6].

(2) Given nums = [1, 3, 2, 2, 3, 1], one possible answer is [2, 3, 1, 3, 1, 2].

Note:

You may assume all input has valid answer.

Follow Up:

Can you do it in O(n) time and/or in-place with O(1) extra space?

-----------------------------------------Code--------------------------------------------------

time-O(nlog(n))-space-O(n)

class Solution:

def wiggleSort(self, nums):

length = int(len(nums))

odd = length

even = (length+1)//2

tmp=sorted(nums)

for i in range(length):

if i&1:

odd -= 1

nums[i] = tmp[odd]

else:

even -= 1

nums[i] = tmp[even]

print(nums)

obj = Solution()

nums= [4,5,5,6]

print( obj.wiggleSort(nums))

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

## 376. Wiggle Subsequence

A sequence of numbers is called a wiggle sequence if the differences between successive numbers strictly alternate between positive and negative. The first difference (if one exists) may be either positive or negative. A sequence with fewer than two elements is trivially a wiggle sequence.

For example, [1,7,4,9,2,5] is a wiggle sequence because the differences (6,-3,5,-7,3) are alternately positive and negative. In contrast, [1,4,7,2,5] and [1,7,4,5,5] are not wiggle sequences, the first because its first two differences are positive and the second because its last difference is zero.

Given a sequence of integers, return the length of the longest subsequence that is a wiggle sequence. A subsequence is obtained by deleting some number of elements (eventually, also zero) from the original sequence, leaving the remaining elements in their original order.

Examples:

Input: [1,7,4,9,2,5]

Output: 6

The entire sequence is a wiggle sequence.

Input: [1,17,5,10,13,15,10,5,16,8]

Output: 7

There are several subsequences that achieve this length. One is [1,17,10,13,10,16,8].

Input: [1,2,3,4,5,6,7,8,9]

Output: 2

Follow up:

Can you do it in O(n) time?

DYNAMIC PROGRAMMING

class Solution:

def wiggleMaxLength(self,nums):

length = len(nums)

if length <= 1:

return length

down,up = 1,1

for i in range(1,length):

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

up = down + 1

elif nums[i] < nums[i-1]:

down = up + 1

return max(down,up)

obj = Solution()

nums=[3,3,3,2,5]

print( obj.wiggleMaxLength(nums))

GREEDY

class Solution:

def wiggleMaxLength(self,nums):

length = len(nums)

if length <= 1:

return length

ans,flag = 1,None

for i in range(1,length):

if( (not flag or flag == None) and nums[i-1] < nums[i]):

print(i," ",nums[i])

ans += 1

flag = True

elif((flag or flag == None) and nums[i-1] > nums[i]):

print(i," ",nums[i])

ans += 1

flag = False

return ans

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

## 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".

-----------------------------------------Algorithm-------------------------------------------------

the use of stack

**prevString curK [ curSting ] ….**

-----------------------------------------Code--------------------------------------------------

class Solution(object):

def decodeString(self, s):

stack = []

curK = 0

curString = ""

for ch in s:

if ch.isdigit():

curK = curK \* 10 + int(ch)

elif ch.isalpha():

curString += ch

elif ch == "[":

stack.append(curK)

stack.append(curString)

curK = 0

curString = ""

else:

prevString = stack.pop()

k = stack.pop()

curString = prevString + k\*curString

return curString

obj = Solution()

print( obj.decodeString("3[3[3[a]]]") )

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

## 448. Find All Numbers Disappeared in an Array

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

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

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

Example:

Input:

[4,3,2,7,8,2,3,1]

Output:

[5,6]

-----------------------------------------Code--------------------------------------------------

**O(n) mark**

class Solution(object):

def findDisappearedNumbers(self, nums):

for i in range(len(nums)):

index = abs(nums[i]) - 1

nums[index] = - abs(nums[index])

return [i + 1 for i in range(len(nums)) if nums[i] > 0]

**O(n) swap**

class Solution(object):

def findDisappearedNumbers(self, nums):

ans,length,i = [],len(nums),0

while i < length:

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

a = nums[i]

nums[i],nums[a-1] = nums[a-1],a

continue

i += 1

for i in range(length):

if i+1 != nums[i]:

ans.append(i+1)

return ans

obj = Solution()

nums =[4,3,2,7,8,2,3,1]

print("ans",obj.findDisappearedNumbers(nums))

## 461. Hamming Distance

The Hamming distance between two integers is the number of positions at which the corresponding bits are different.

Given two integers x and y, calculate the Hamming distance.

Note:

0 ≤ x, y < 231.

Example:

Input: x = 1, y = 4

Output: 2

Explanation:

1 (0 0 0 1)

4 (0 1 0 0)

↑ ↑

The above arrows point to positions where the corresponding bits are different.

the use of bin() and count()

-----------------------------------------Code-------------------------------------------------------

class Solution(object):

def hammingDistance(self, x, y):

return bin(x^y).count("1")

obj = Solution()

print(obj.hammingDistance(10,4))

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

## 477. Total Hamming Distance

The Hamming distance between two integers is the number of positions at which the corresponding bits are different.

Now your job is to find the total Hamming distance between all pairs of the given numbers.

Example:

Input: 4, 14, 2

Output: 6

Explanation: In binary representation, the 4 is 0100, 14 is 1110, and 2 is 0010 (just

showing the four bits relevant in this case). So the answer will be:

HammingDistance(4, 14) + HammingDistance(4, 2) + HammingDistance(14, 2) = 2 + 2 + 2 = 6.

Note:

Elements of the given array are in the range of 0 to 10^9

Length of the array will not exceed 10^4.

-----------------------------------------Code------------------------------------------------------class Solution(object):

def totalHammingDistance(self, nums):

ans,length,bit\_1 = 0,len(nums),[0]\*32

for i in range(32):

for j in range(length):

bit\_1[i] += 1 if nums[j]&1 else 0

nums[j] //= 2

for i in range(32):

ans += (length-bit\_1[i])\*bit\_1[i]

return ans

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

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

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

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