## 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]

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

## 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

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

## 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();**

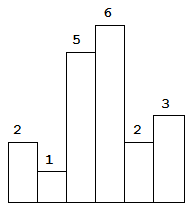
**nums=[1,2]**

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

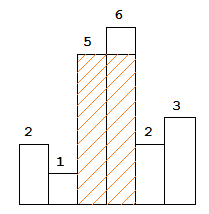
**-------------------------------------------------------------------------------------------------**

## 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

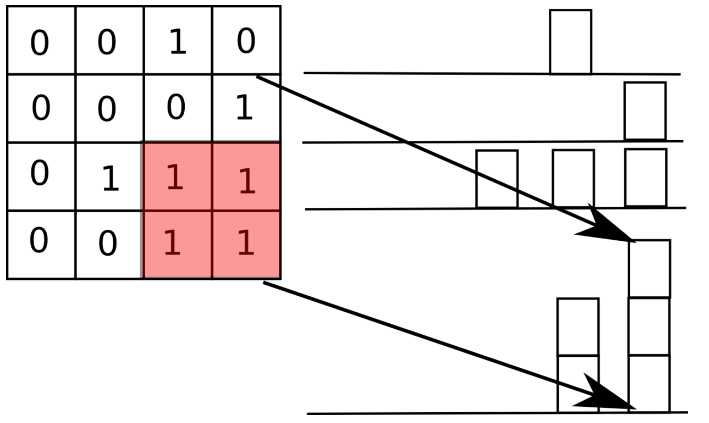
1 1 1 1 1

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))

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

## 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))

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

## 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))

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

## 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))

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

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

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

## 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))

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

## 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--------------------------------------------------**

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))

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

## 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]

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

## 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;

}

}