Count 1

191. Number of 1 Bits

Write a function that takes an unsigned integer and returns the number of ’1' bits it has

For example, the 32-bit integer ’11' has binary representation 00000000000000000000000000001011, so the function should return 3.

**S1 Bit**

这里需要注意的点是，需要把n看作unsigned int, 所以应该是n = n >>> 1.

注意无符号右移是 >>>

public int hammingWeight(int n) {

int count = 0;

while (n != 0) {

count += n & 1;

n >>>= 1;

}

return count;

}

190. Reverse Bits

Reverse bits of a given 32 bits unsigned integer. For example, given input 43261596

(represented in binary as 00000010100101000001111010011100),

return 964176192 (represented in binary as 00111001011110000010100101000000).

题解：

其实也不难啦，注意一下细节

public int reverseBits(int n) {

int result = 0;

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

result = result << 1;

if ((n & (1 << i)) != 0) result += 1;

}

return result;

}

public int reverseBits(int n) {

int result = 0;

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

result = result << 1;

result += (n >> i) & 1;

}

return result;

}

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.

For num = 5 you should return [0,1,1,2,1,2].

It is very easy to come up with a solution with run time O(n\*sizeof(integer)). But can you do it in linear time O(n) /possibly in a single pass?

Space complexity should be O(n).

**S1 Math trick**

这道题其实是利用了一个trick：

假设dp[i] = # of bits of i, 那么满足一个公式，dp[i] = dp[i / 2] + i%2

public int[] countBits(int num) {

int[] dp = new int[num + 1];

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

dp[i] = dp[i / 2] + i % 2;

}

return dp;

}

477. Total Hamming Distance

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

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.

**S1. Maths**

在每一位的时候，查看所有的nums。

注意，假设一共有n个num，其中k个在某一位的bit是1，其他(n-k)个是0，那么一共可以增加

k\*(n-k) 个。

public int totalHammingDistance(int[] nums) {

int total = 0, n = nums.length;

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

int bitCount = 0; *// # of '1' at this bit.*

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

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

}

total += bitCount\*(n - bitCount);

}

return total;

}

Power of X

231. Power of Two

Given an integer, write a function to determine if it is a power of two.

**S1. bit**

根据位运算就好了，如果是power of 2, 一定>0 && 二进制一定只有一位1.

public boolean isPowerOfTwo(int n) {

return n > 0 && Integer.bitCount(n) == 1;

}

342. Power of Four

Given an integer (signed 32 bits), write a function to check whether it is a power of 4.

Given num = 16, return true. Given num = 5, return false.

Follow up: Could you solve it without loops/recursion?

**S1 Bit maniputation**

Power of 4的三个性质：

1. > 0
2. 转换成二进制后只有一个1
3. 1出现在奇数位。（如果出现在偶数位，就会是power of 2, 但不是power of 4.）

public boolean isPowerOfFour(int num) {

return num > 0 && (Integer.bitCount(num) == 1) && ((num & 0x55555555) != 0);

}

326. Power of Three

Given an integer, write a function to determine if it is a power of three.

Follow up: Could you do it without using any loop / recursion?

**S1 Maths**

有一个相当取巧的方法：

public boolean isPowerOfThree(int n) {

*// 1162261467 is 3^19, 3^20 is bigger than int*

return (n > 0 && 1162261467 % n == 0);

}

476. Number Complement

Given a positive integer, output its complement number. The complement strategy is to flip the bits of its binary representation.

Input: 5, Output: 2

Explanation: The binary representation of 5 is 101 (no leading zero bits), and its complement is 010. So you need to output 2.

public int findComplement(int num) {

int mask = Integer.highestOneBit(num) - 1;

return num ^ mask;

*//return (~num) & mask;*

}

43. Multiply Strings

Given two non-negative integers num1 and num2 represented as strings, return the product of num1 and num2.

Note:

The length of both num1 and num2 is < 110.

Both num1 and num2 contains only digits 0-9.

Both num1 and num2 does not contain any leading zero.

**S1 Maths**

就是平时计算的方法，只是用代码写出来而已，

4 3 2 1 5

2 8 3

总的长度不会超过 m + n, 并且计算3和上面每一位分别相乘时，涉及两个位置，当前位和进位。当前位的index=i+j+1, 因为m, n的index都是从0开始，所以需要+1

进位在当前位的前面一位，所以是i+j

public String multiply(String num1, String num2) {

int m = num1.length(), n = num2.length();

int[] nums = new int[m + n];

for (int i = m-1;i >= 0;i--) {

for (int j = n-1;j >= 0;j--) {

int mul = (num1.charAt(i) - '0') \* (num2.charAt(j) - '0');

int cur = i + j + 1;

int prev = i + j;

int sum = mul + nums[cur]; *// add previous carry number*

nums[prev] += sum / 10;

nums[cur] = sum % 10; *// carry number*

}

}

StringBuilder sb = new StringBuilder();

for (int num:nums) {

if (!(sb.length() == 0 && num == 0)) sb.append(num);

}

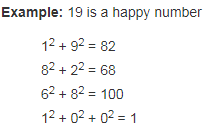
return sb.length() == 0 ? "0" : sb.toString();

}

202. Happy Number

Write an algorithm to determine if a number is "happy".

A happy number is a number defined by the following process: Starting with any positive integer, replace the number by the sum of the squares of its digits, and repeat the process until the number equals 1 (where it will stay), or it loops endlessly in a cycle which does not include 1. Those numbers for which this process ends in 1 are happy numbers.



**S1 Maths**

终止条件：n == 1, 或者n = 某个processed 过的number – 会无限循环

public boolean isHappy(int n) {

Set<Integer> set = new HashSet<>();

while (n != 1) {

if (set.contains(n)) return false;

set.add(n);

n = getSum(n);

}

return true;

}

private int getSum(int n) {

int sum = 0;

while (n > 0) {

sum += Math.pow((n % 10), 2);

n /= 10;

}

return sum;

}

357. Count Numbers with Unique Digits

Given a non-negative integer n, count all numbers with unique digits, x, where 0 ≤ x < 10n.

Example: Given n = 2, return 91. (The answer should be the total numbers in the range of 0 ≤ x < 100,

excluding [11,22,33,44,55,66,77,88,99])

这个题的关键在于怎样分解问题，用dp来表示什么，找出规律：

n == 0: 1

n == 1: 10

n == 2: 9\*9 // 注意！这里存在不连续。因为0，所以第一个乘的不是10，是9

n == 3: 9\*9\*8 // 所以这里的循环需要从9开始，不能从10 开始

...

同时需要注意，当i >= 10之后，count不会再增加，因为一定会有重复数字

public int countNumbersWithUniqueDigits(int n) {

if (n == 0) {

return 1;

}

int ans = 10, base = 9;

for (int i = 2; i <= n && i <= 10; i++) {

base = base \* (9 - (i - 2));

ans += base;

}

return ans;

}

**Math**

149. Max Points on a Line

Given n points on a 2D plane, find the maximum number of points that lie on the same straight line.

**S Math + Map**

on a line: on the same point OR have same slope.

外循环i从0~n-1, 内循环j只要从0~i-1即可，因为后面的会在后面计算，不需要重复算。

计算每个点的时候都new Map，<slope, # of points that match>, 然后对same point和same slope分开计数。

public int maxPoints(Point[] points) {

if (points == null || points.length < 2) return points.length;

int max = 0;

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

*// <slope, # of points that match>*

Map<Double, Integer> map = new HashMap<>();

Point p1 = points[i];

int samePoint = 1;

int localMax = 0;

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

Point p2 = points[j];

double slope = 0.0;

if (p1.y == p2.y && p1.x == p2.x) {

samePoint++;

continue;

} else if (p1.x == p2.x) slope = Double.MAX\_VALUE;

else slope = (double) (p1.y - p2.y) / (p1.x - p2.x);

if (map.containsKey(slope)) {

int count = map.get(slope) + 1;

map.put(slope, count);

localMax = Math.max(localMax, count);

} else {

map.put(slope, 1);

localMax = Math.max(localMax, 1);

}

}

max = Math.max(max, localMax + samePoint);

}

return max;

}

Houzz – Remove any number contains 9

Remove any number containing 9 like 9, 19, 29, ... 91, 92, ... 99, 109...

Write a function that returns the nth number. E.g. newNumber(1) = 1 newNumber(8) = 8, newNumber(9) = 10

基本方针就是写九进制，9-based，可以用recursion, iteration实现。

S1.Recursion

public static int removeNumber9(int n) {

return n != 0 ? removeNumber9(n / 9)\*10 + n % 9 : 0;

}

**S2.Iteration, using stack**

因为从左到右总会先取到高位的值，但是加的时候需要先从低位开始加，所以先放进stack里。

public static int removeNumber9nd(int n) {

int base = 9;

Stack<Integer> s = new Stack<>();

while (n != 0) {

s.push(n % base);

n /= base;

}

int res = 0;

while (!s.isEmpty()) {

res = 10\*res + s.pop();

}

return res;

}

Yelp:

Given 4 points, decide whether they are square, Rectangle, Diamond or 平行四边形

判断正方形其实有两种情况，难度有区别：

1. 正方形一定是水平放置的
2. 正方形怎么放都可以，可以斜着放

第二种就比第一种要更复杂一点。

先看第一种：首先sort，确定四个点，再保证有两个相邻的直角，并且这两个直角涉及的三条边边长相等即可。

public static boolean isSquare(Point[] points) {

Arrays.sort(points, new Comparator<Point>() {

@Override

public int compare(Point p1, Point p2) {

if (p1.x != p2.x) return p1.x - p2.x;

else return p1.y - p2.y;

}

});

if (points[1].x != points[0].x || points[2].y != points[0].y

|| points[3].x != points[2].x) return false;

int len1 = points[1].y - points[0].y;

int len2 = points[2].x - points[0].x;

int len3 = points[3].y - points[2].y;

if (len1 != len2 || len2 != len3) return false;

return true;

}

第二种：

类似，也是证明前面的两个条件，不过边长不能直接相减得到，直角也要根据向量相乘为0得到。

*// determine shape given 4 points*

public static boolean isSquareHard(Point[] points) {

Arrays.sort(points, new Comparator<Point>() {

@Override

public int compare(Point p1, Point p2) {

return p1.x - p2.x;

}

});

if (helper(points, 0, 1, 2) && helper(points, 2, 0, 3)) return true;

return false;

}

*// i is center, j and k are nodes next to i.(i->j)\*(i->k) vector should be 0*

static boolean helper(Point[] points, int i, int j, int k) {

double edge1 = (Math.pow(points[j].x - points[i].x, 2)

+ Math.pow(points[j].y - points[i].y, 2));

double edge2 = (Math.pow(points[k].x - points[i].x, 2)

+ Math.pow(points[k].y - points[i].y, 2));

if (edge1 != edge2) return false;

if ((points[i].x - points[j].x)\*(points[i].x - points[k].x)

+ (points[i].y - points[j].y)\*(points[i].y - points[k].y) == 0) return true;

return false;

}

29. Divide Two Integers

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

If it is overflow, return MAX\_INT.

题解：

1. 首先考虑overflow有几种情况 - 两种，Integer.MIN\_VALUE / -1 || divisor == 0

2. 很明显要用bit manipulation, 那么一次bit manipulation的循环只能算出2^n，那么具体的除数只能先算最大的x, for x > 2^n, 然后减掉2^n, 再继续算.... 最后把所有的n加起来

dividend = divisor\*(m1 + m2 + m3 + ...) ANY number could be expressed in this way.

eg: 3\*(8 + 4 + 2 + 1) and m should always be 1 << n, n is int，也就是说n是2的某次方，

Hence we only need to find out all the m1, m2, m3... and get the sum.

eg: 15 / 3, 第一次依次比较15与(3\*1), 15与(3\*2), 15与(3\*4)......直到某个值，15 < (3\* 2^n),

然后multiple += n, dividend -= (3\* 2^n)...

till dividend < divisor. the sum of multiple is the result.

public int divide(int dividend, int divisor) {

if(divisor == 0 || (dividend == Integer.MIN\_VALUE && divisor == -1))

return Integer.MAX\_VALUE;

int res = 0;

int sign = (dividend < 0) ^ (divisor < 0) ? -1 : 1;

long dvd = Math.abs((long)dividend);

long dvs = Math.abs((long)divisor);

while(dvs <= dvd) {

long temp = dvs,mul = 1;

while(dvd >= (temp << 1)){

temp <<= 1;

mul <<= 1;

}

dvd -= temp;

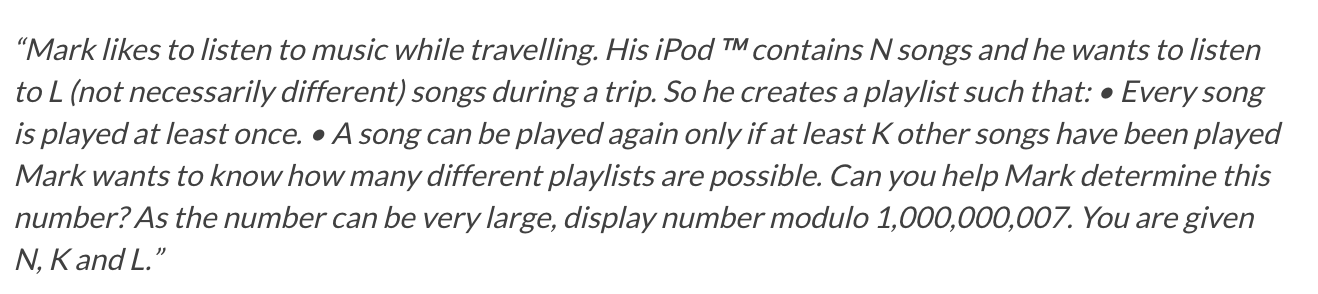
res += mul;

}

return sign == 1 ? res : -res;

}

Coursera - # of ways for playlist



真心是数学题，

比如假设N=20, L=9, k=2,

那么第一首有20种可能性，第二首有19种（只有第一个选过的不能选），第三首有18种可能性（第一首，第二首不能选），第四首又有20种可能性，因为

public int getNumber(int N, int L, int k) {

if (L > N) { *// not enough song, have to repeat.*

if (L >= K + 1) { *// not k too large to repeat.*

return 0;

}

}

int a = L / (K + 1); *// # of sequence that contains repeat*

int b = L % (K + 1);

long partA = 1;

long partB = 1;

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

partA \*= N-i;

}

partA = (long) Math.pow(partA, a);

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

partB \*= N-i;

}

return (long) (partA \* partB % (1e9 + 7));

}

319. Bulb Switcher

There are n bulbs that are initially off. You first turn on all the bulbs. Then, you turn off every second bulb. On the third round, you toggle every third bulb (turning on if it's off or turning off if it's on). For the ith round, you toggle every i bulb. For the nth round, you only toggle the last bulb. Find how many bulbs are on after n rounds.

第i轮能被i整除的灯状态翻转，所以要看的其实是从1到n，的每个数字能被多少数字整除，

更具体一点，是能被奇数的数字整除，还是偶数的数字整除，如果是奇数，那么最终就是on。

假设i=32，那么能被1,32 2,16 4,8 这些数字整除，是偶数，所以最后是off. 并且可以注意到，这些数字都是一对一对的。

那么由此看来，奇数只有一种情况：本身是平方数，比如36，存在6,6,但是6只会作用一次，所以是奇数。

所以 == 求1-n有多少平方数。the number of square numbers.

public int bulbSwitch(int n) {

return (int)Math.sqrt(n);

}

转换

8. String to Integer (atoi)

Implement atoi to convert a string to an integer.

Hint: Carefully consider all possible input cases.

题解：

重点是handle各种corner cases

1.whitespaces before integer

2.sign of #

3.overflow

4.invalid input - after all three, check whether each char is integer

其他很简单，这个解法巧妙的点就在于check overflow这一块，base > Integer.MAX\_VALUE / 10：

比如"30978977324..." (base == Integer.MAX\_VALUE / 10 && str.charAt(idx) > '7')：

可以handle positive & negative overflow at the same time.

public int myAtoi(String str) {

if (str == null || str.length() == 0) return 0;

int sign = 1, base = 0, idx = 0;

// 1.handle leading spaces

while (str.charAt(idx) == ' ') idx++;

// 2.handle sign +/-

if (str.charAt(idx) == '-' || str.charAt(idx) == '+') {

sign = str.charAt(idx++) == '-' ? -1 : 1;

}

// 遇到invalid char, 只把之前的valid返回

while (idx < str.length() && str.charAt(idx) >= '0' && str.charAt(idx) <= '9') {

// check overflow, and return Integer.MAX\_VALUE when overflow:

if (base > Integer.MAX\_VALUE / 10 || (base == Integer.MAX\_VALUE / 10 && str.charAt(idx) > '7')) {

return sign == 1 ? Integer.MAX\_VALUE : Integer.MIN\_VALUE;

}

base = base \* 10 + str.charAt(idx++) - '0';

}

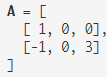
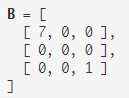
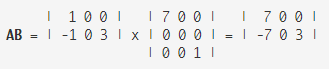
return sign \* base;

}

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.

**S1. Maths**

就是数学而已，按照矩阵乘法的方法来做

public int[][] multiply(int[][] A, int[][] B) {

int rowA = A.length, colA = A[0].length;

int rowB = B.length, colB = B[0].length;

int[][] result = new int[rowA][colB];

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

for (int j = 0; j < colA; j++) { *// colA=rowB*

if (A[i][j] == 0) continue;

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

if (B[j][k] == 0) continue;

result[i][k] += A[i][j] \* B[j][k];

}

}

}

return result;

}