```

题号 题目 知识点 难度 通过率

QQ1 生成格雷码 递归 简单 22.61%

QQ2 微信红包 模拟 简单 25.61%

QQ3 编码 字符串模拟 中等 26.60%

QQ4 游戏任务标记 模拟 中等 37.55%

QQ5 素数对 模拟 中等 31.58%

QQ6 geohash编码 模拟 中等 24.82%

QQ7 拼凑硬币 动态规划 中等 22.25%

QQ8 数字转换机 贪心 中等 34.40%

QQ9 魔法阵 计算几何 简单 39.06%

QQ10 石子合并 中等 49.23%

QQ11 小Q的排序 模拟 简单 48.22%

```

### QQ1 生成格雷码

#### 题目描述

```

在一组数的编码中，若任意两个相邻的代码只有一位二进制数不同， 则称这种编码为格雷码(Gray Code)，请编写一个函数，使用递归的方法生成N位的格雷码。

给定一个整数n，请返回n位的格雷码，顺序为从0开始。

```

#### 测试样例：

```

1

返回：["0","1"]

```

```java

import java.util.Scanner;

public class Main {

public static String[] getGray(int n) {

//总共需要的位数

String[] grayCode = new String[(int) Math.pow(2, n)];

if (n == 1) {

//位数为1为递归结束情况，规律:

/\*

1、除了最高位（左边第一位），格雷码的位元完全上下对称（看下面列表）。

比如第一个格雷码与最后一个格雷码对称（除了第一位），第二个格雷码与倒数第二个对称，以此类推。

2、最小的重复单元是 0 , 1。

000

001

011

010

110

111

101

100

\*/

grayCode[0] = "0";

grayCode[1] = "1";

return grayCode;

}

//递归前n-1的格雷码

String[] last = getGray(n - 1);

//格雷码计算方式，前一次格雷码分两部分，假设01

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

grayCode[i] = "0" + last[i];//前半部分的二进制码前加 0

grayCode[grayCode.length - i - 1] = "1" + last[i];//后半部分的二进制码前加1

}

return grayCode;

}

public static void main(String[] args) {

Scanner in = new Scanner(System.in);

int b;

while (in.hasNext()) {

b = in.nextInt();

String[] strings = getGray(b);

System.out.print("[");

int c = strings.length;

for (int i = 0; i < c - 1; i++) {

System.out.print("\"" + strings[i] + "\",");

}

System.out.print("\"" + strings[c - 1] + "\"");

System.out.println("]");

}

}

}

```

### QQ2 微信红包

#### 题目描述

```

春节期间小明使用微信收到很多个红包，非常开心。在查看领取红包记录时发现，某个红包金额出现的次数超过了红包总数的一半。请帮小明找到该红包金额。写出具体算法思路和代码实现，要求算法尽可能高效。

给定一个红包的金额数组gifts及它的大小n，请返回所求红包的金额。

若没有金额超过总数的一半，返回0。

```

#### 测试样例：

```

[1,2,3,2,2],5

```

#### 返回

```

2

```

```java

import java.util.\*;

public class Gift {

public int getValue(int[] gifts, int n) {

int count = 0;

int res = gifts[0];

for (int num : gifts) {

if (num == res) count++;

else count--;

if (count == 0) res = num;

}

if (count > 0) return res;

else return 0;

}

}

```

### QQ3 编码

#### 题目描述

```

假定一种编码的编码范围是a ~ y的25个字母，从1位到4位的编码，如果我们把该编码按字典序排序，形成一个数组如下： a, aa, aaa, aaaa, aaab, aaac, … …, b, ba, baa, baaa, baab, baac … …, yyyw, yyyx, yyyy 其中a的Index为0，aa的Index为1，aaa的Index为2，以此类推。 编写一个函数，输入是任意一个编码，输出这个编码对应的Index.

```

#### 输入描述:

```

输入一个待编码的字符串,字符串长度小于等于100.

```

#### 输出描述:

```

输出这个编码的index

```

#### 示例1

#### 输入

```

baca

```

#### 输出

```

16331

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

String str;

while ((str = reader.readLine()) != null) {

char[] chars = str.trim().toCharArray();

int temp = 0, sum = 0;

for (int i = 0; i < 4; i++) {

temp \*= 25;

if (i < chars.length) temp += chars[i] - 'a';

sum += temp;

if (i < chars.length - 1) sum += 1;

}

System.out.println(sum);

}

}

}

```

### QQ4 游戏任务标记

#### 题目描述

```

游戏里面有很多各式各样的任务，其中有一种任务玩家只能做一次，这类任务一共有1024个，任务ID范围[1,1024]。请用32个unsigned int类型来记录着1024个任务是否已经完成。初始状态都是未完成。 输入两个参数，都是任务ID，需要设置第一个ID的任务为已经完成；并检查第二个ID的任务是否已经完成。 输出一个参数，如果第二个ID的任务已经完成输出1，如果未完成输出0。如果第一或第二个ID不在[1,1024]范围，则输出-1。

```

#### 输入描述:

```

输入包括一行,两个整数表示人物ID.

```

#### 输出描述:

```

输出是否完成

```

#### 示例1

#### 输入

```

1024 1024

```

#### 输出

```

1

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.lang.Math;

public class Main {

private static int[] s = new int[32];

private static int testId(int set, int check) {

//ID检查

if (set > 1024 || check > 1024) return -1;

// index 为int数组的下标（0~31） bit为int变量的第几位（0~31） 例如 1--s[1]=2^31

int set\_index = (set - 1) / 32;

int set\_bit = set % 32;

int check\_index = (check - 1) / 32;

int check\_bit = 32 \* (check\_index + 1) - check;

//设置第一个ID任务完成

s[set\_index] = s[set\_index] | (int) Math.pow(2, set\_bit);

// 检查第二个任务是否完成

if ((s[check\_index] & (int) Math.pow(2, check\_bit)) != 0) return 1;

else return 0;

}

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String[] str = br.readLine().split(" ");

int set = Integer.parseInt(str[0]);

int check = Integer.parseInt(str[1]);

System.out.println(testId(set, check));

}

}

```

### QQ5 素数对

#### 题目描述

```

给定一个正整数，编写程序计算有多少对质数的和等于输入的这个正整数，并输出结果。输入值小于1000。

如，输入为10, 程序应该输出结果为2。（共有两对质数的和为10,分别为(5,5),(3,7)）

```

#### 输入描述:

```

输入包括一个整数n,(3 ≤ n < 1000)

```

#### 输出描述:

```

输出对数

```

#### 示例1

#### 输入

```

10

```

#### 输出

```

2

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class Main {

public static void main(String[] args) throws NumberFormatException, IOException {

BufferedReader beader = new BufferedReader(new InputStreamReader(System.in));

int target = Integer.parseInt(beader.readLine());

int[] numbs = new int[target];

int count = 0;

int result = 0;

for (int i = 2; i <= target; i++) {

int judge = 0;

for (int j = 2; j < i; j++) {

if (i % j == 0) {

judge = 1;

break;

}

}

if (judge == 0) {

numbs[count] = i;

count++;

}

}

for (int i = 0; i < count - 1; i++) {

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

if ((numbs[i] + numbs[j]) == target) {

result++;

break;

}

}

}

System.out.println(result);

}

}

```

### QQ6 geohash编码

#### 题目描述

```

geohash编码：geohash常用于将二维的经纬度转换为字符串，分为两步：第一步是经纬度的二进制编码，第二步是base32转码。

此题考察纬度的二进制编码：算法对纬度[-90, 90]通过二分法进行无限逼近（取决于所需精度，本题精度为6）。注意，本题进行二分法逼近过程中只采用向下取整来进行二分，针对二分中间值属于右区间。算法举例如下： 针对纬度为80进行二进制编码过程：

1) 区间[-90, 90]进行二分为[-90, 0),[0, 90]，成为左右区间，可以确定80为右区间，标记为1；

2) 针对上一步的右区间[0, 90]进行二分为[0, 45),[45, 90]，可以确定80是右区间，标记为1；

3) 针对[45, 90]进行二分为[45, 67),[67,90],可以确定80为右区间，标记为1；

4) 针对[67,90]进行二分为[67, 78),[78,90]，可以确定80为右区间，标记为1；

5) 针对[78, 90]进行二分为[78, 84),[84, 90]，可以确定80为左区间，标记为0；

6) 针对[78, 84)进行二分为[78, 81), [81, 84)，可以确定80为左区间，标记为0；

```

#### 输入描述:

```

输入包括一个整数n,(-90 ≤ n ≤ 90)

```

#### 输出描述:

```

输出二进制编码

```

#### 示例1

#### 输入

```

80

```

#### 输出

```

111100

```

```java

import java.io.\*;

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String str;

final int TIMES = 6;

while ((str = br.readLine()) != null) {

if (str.equals("")) continue;

int num = Integer.parseInt(str);

StringBuilder ans = new StringBuilder();

int left = -90, right = 90;

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

int mid = (left + right) / 2;

if (num >= mid) {

left = mid;

ans.append('1');

} else {

right = mid;

ans.append('0');

}

}

System.out.println(ans);

}

}

}

```

### QQ7 拼凑硬币

#### 题目描述

```

小Q十分富有，拥有非常多的硬币，小Q拥有的硬币是有规律的，对于所有的非负整数K，小Q恰好各有两个面值为2^K的硬币，所以小Q拥有的硬币就是1,1,2,2,4,4,8,8,…。小Q有一天去商店购买东西需要支付n元钱，小Q想知道有多少种方案从他拥有的硬币中选取一些拼凑起来恰好是n元（如果两种方案某个面值的硬币选取的个数不一样就考虑为不一样的方案）。

```

#### 输入描述:

```

输入包括一个整数n(1≤n≤10^18)，表示小Q需要支付多少钱。注意n的范围。

```

#### 输出描述:

```

输出一个整数，表示小Q可以拼凑出n元钱放的方案数。

```

#### 示例1

#### 输入

```

6

```

#### 输出

```

3

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.HashMap;

import java.util.Map;

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

long n = Long.parseLong(br.readLine());

Map<Long, Long> map = new HashMap<>();

map.put(0L, 1L);

map.put(1L, 1L);

System.out.println(solution(n, map));

}

private static long solution(Long n, Map<Long, Long> map) {

if (map.containsKey(n)) return map.get(n);

long cnt;

//偶数

if ((n & 1) == 0) cnt = solution(n >> 1, map) + solution((n >> 1) - 1, map);

else cnt = solution(n >> 1, map);

map.put(n, cnt);

return cnt;

}

}

```

### QQ8 数字转换机

#### 题目描述

```

小Q从牛博士那里获得了一个数字转换机，这台数字转换机必须同时输入两个正数a和b，并且这台数字转换机有一个红色的按钮和一个蓝色的按钮：

当按下了红色按钮，两个数字同时加1。

当按下了蓝色按钮，两个数字同时乘2。

小Q现在手中有四个整数a，b，A，B，他希望将输入的两个整数a和b变成A，B（a对应A，b对应B）。因为牛博士允许小Q使用数字转换机的时间有限，所以小Q希望按动按钮的次数越少越好。请你帮帮小Q吧。

```

#### 输入描述:

```

输入包括一行，一行中有四个正整数a，b，A，B，（1≤a，b，A，B≤10^9）。

```

#### 输出描述:

```

如果小Q可以完成转换，输出最少需要按动按钮的次数，否则输出-1。

```

#### 示例1

#### 输入

```

100 1000 202 2002

```

#### 输出

```

2

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String[] str = br.readLine().split(" ");

int a = Integer.parseInt(str[0]);

int b = Integer.parseInt(str[1]);

int A = Integer.parseInt(str[2]);

int B = Integer.parseInt(str[3]);

System.out.println(solution(a, b, A, B));

}

private static int solution(int a, int b, int A, int B) {

if (a == A && b == B) return 0;

else if (a != A && b == B) return -1;

else if (a == A) return -1;

else if (a > A || b > B) return -1;

int res;

if ((A & 1) == 0 && (B & 1) == 0) res = solution(a, b, A / 2, B / 2);

else res = solution(a, b, A - 1, B - 1);

if (res == -1) return -1;

return res + 1;

}

}

```

### QQ9 魔法阵

#### 题目描述

```

小Q搜寻了整个魔法世界找到了四块魔法石所在地，当4块魔法石正好能构成一个正方形的时候将启动魔法阵，小Q就可以借此实现一个愿望。

现在给出四块魔法石所在的坐标，小Q想知道他是否能启动魔法阵

```

#### 输入描述:

```

输入的第一行包括一个整数（1≤T≤5）表示一共有T组数据

每组数据的第一行包括四个整数x[i](0≤x[i]≤10000)，即每块魔法石所在的横坐标

每组数据的第二行包括四个整数y[i](0≤y[i]≤10000),即每块魔法石所在的纵坐标

```

#### 输出描述:

```

对于每组数据，如果能启动魔法阵输出“Yes”否则输出“No”。

```

#### 示例1

#### 输入

```

3

0022

0202

0156

1605

0077

0303

```

#### 输出

```

Yes

Yes

No

```

```java

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.Arrays;

import java.util.LinkedList;

import java.util.List;

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int n = Integer.parseInt(br.readLine());

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

char[] x = br.readLine().toCharArray();

char[] y = br.readLine().toCharArray();

System.out.println(solution(x, y));

}

}

private static String solution(char[] x, char[] y) {

int[] t = new int[3];

for (int i = 1; i < 4; i++) {

t[i - 1] = ((x[i] - x[0]) \* (x[i] - x[0]) + (y[i] - y[0]) \* (y[i] - y[0]));

}

Arrays.sort(t);

if (t[0] == t[1] && t[0] + t[1] == t[2]) return "yes";

return "No";

}

}

```

### QQ10 石子合并

#### 题目描述

```

小Q和牛博士在玩一个石子合并的游戏，初始一共有n堆石子，每堆石子有w[i]个石子。小Q和牛博士他们需要对石子堆进行合并，每次他们可以任意选择两堆石子进行合并。一堆有x个石子的石子堆和一堆有y个石子的石子堆合并将得到一堆x+y个石子的石子堆，这次合并得分为x\*y，当只剩下一堆石子的时候游戏结束。

、小Q和牛博士希望采取优秀的策略获得最大得分，希望你能来帮他们算算最大得分多少。

```

#### 输入描述:

```

输入包括两行，第一行一个正整数n(2≤n≤100)。

第二行包括n个正整数w[i](1≤w[i]≤100)，即每堆石子的个数。

```

#### 输出描述:

```

输出一个正整数，即小Q和牛博士最大得分是多少。

```

#### 示例1

#### 输入

```

3

1 2 3

```

#### 输出

```

11

```

```java

import java.io.\*;

import java.util.\*;

import java.lang.\*;

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader read = new BufferedReader(new InputStreamReader(System.in));

int n = Integer.parseInt(read.readLine());

String[] str = read.readLine().split(" ");

int[] num = new int[n];

for (int i = 0; i < n; i++) num[i] = Integer.parseInt(str[i]);

int grade;

int count = 0;

int sum = 0;

for (int i = num.length - 1; i >= 0; i--) {

grade = count \* num[i];

sum = sum + grade;

count += num[i];

}

System.out.println(sum);

}

}

```

### QQ11 小Q的排序

#### 题目描述

```

小Q在学习许多排序算法之后灵机一动决定自己发明一种排序算法，小Q希望能将n个不同的数排序为升序。小Q发明的排序算法在每轮允许两种操作：

1、 将当前序列中前n-1个数排为升序

2、 将当前序列中后n-1个数排为升序

小Q可以任意次使用上述两种操作，小Q现在想考考你最少需要几次上述操作可以让序列变为升序。

```

#### 输入描述:

```

输入包括两行，第一行包括一个正整数n(3≤n≤10^5)，表示数字的个数

第二行包括n个正整数a[i](1≤a[i]≤10^9)，即需要排序的数字，保证数字各不相同。

```

#### 输出描述:

```

输出一个正整数，表示最少需要的操作次数

```

#### 示例1

#### 输入

```

6

4 3 1 6 2 5

```

#### 输出

```

2

```

```java

import java.util.\*;

import java.io.\*;

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

String s;

while ((s = br.readLine()) != null) {

int N = Integer.parseInt(s);

String[] arr = br.readLine().split(" ");

int[] a = new int[N];

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

a[i] = Integer.parseInt(arr[i]);

}

int minIndex = 0;

int maxIndex = 0;

int min = 0;

int max = 0;

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

if (min > a[i]) {

min = a[i];

minIndex = i;

}

if (max < a[i]) {

max = a[i];

maxIndex = i;

}

}

if (minIndex == N - 1 && maxIndex == 0) System.out.println("3");

else if (minIndex != N - 1 && maxIndex != 0) System.out.println("2");

else System.out.println("1");

}

}

}

```