陈余 212241807213 计算机科学与技术

1-3最多约数问题。

问题描述:正整数x的约数是能整除x的正整数。正整数x的约数个数记为div(x)。例如，1、2、5、10都是正整数10的约数，且div(10)=4。设a和b是2个正整数，a≤b,，找出a和b之间约数个数最多的数x。

算法设计：对于给定的2个正整数a≤b，计算a和b之间约数个数最多的数。

数据输入：输入数据由文件名为input.txt的文本文件提供。文件的第1行有2个正整数a和b。

结果输出：若找到的a和b之间约数个数最多的数是x,则将div(x)输出到文件output.txt。

输入文件示例 输出文件示例

input.txt output.txt

1 36 9

Java：

import java.util.Scanner;

public class MaxDivisorsInRange {

public static void main(String[] args) {

int a, b, x = 0, i, k; // 输入ab

int n = 2, m = 1; // n：约数个数，m:最大约数个数

Scanner scanner = new Scanner(System.in);

System.out.print("请输入a和b的值：");

a = scanner.nextInt();

b = scanner.nextInt();

for (i = a; i <= b; i++) {

n = 2;

for (k = 2; k < i; k++) {

if (i % k == 0) {

n++;

}

if (n > m) {

m = n;

x = i;

}

}

}

System.out.println(a + "与" + b + "之间约数个数最多的为：" + x + "，约数有" + m"个。");

}

}

1-4金币阵列问题。

问题描述:有mxn (m≤100, n≤100)枚金币在桌面上排成一个m行n列的金币阵列。每枚金币或正面朝上或背面朝上。用数字表示金币状态，0表示金币正面朝上，1表示金币背面朝上。

金币阵列游戏的规则是:①每次可将任一行金币翻过来放在原来的位置上;②每次可任选2列，交换这2列金币的位置。

算法设计:给定金币阵列的初始状态和目标状态，计算按金币游戏规则，将金币阵列从初始状态变换到目标状态所需的最少变换次数。

数据输入:由文件input.txt给出输入数据。文件中有多组数据。文件的第1行有1个正整数k，表示有k组数据。每组数据的第1行有2个正整数m和n。以下m行是金币阵列的初始状态，每行有n个数字表示该行金币的状态，0表示正面朝上，1表示背面朝上。接着的m行是金币阵列的目标状态。

结果输出:将计算出的最少变换次数按照输入数据的次序输出到文件output.txt。相应数据无解时，输出-1。

Java：

public class gold {

private static int[][] source = {{1, 0, 1}, {0, 0, 0}, {1, 1, 0}, {1, 0, 1}};

private static int[][] target = {{1, 0, 1}, {1, 1, 1}, {0, 1, 1}, {1, 0, 1}};

// 临时矩阵，用于保存变换过程中的矩阵状态

private static int[][] tempMatrix = new int[4][3];

private static int rows = 4; // 矩阵行数

private static int columns = 3; // 矩阵列数

private static int count, best = 9999; // 变换步数和最大步数

private static void turnrow(int i) {//行反转函数

for (int j = 0; j < columns; j++)

tempMatrix[i][j] = 1 - tempMatrix[i][j]; // 翻转

count++;

}

// 列交换函数

private static void exchangecolumns(int i, int j) {

if (i == j)

return;

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

int tempValue = tempMatrix[k][i];

tempMatrix[k][i] = tempMatrix[k][j];

tempMatrix[k][j] = tempValue;

}

if (i != j)

count++;

}

// 检查两列是否相同

private static boolean samecolumns(int i, int j) {

boolean isSame = true;

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

if (target[k][i] != tempMatrix[k][j]) {

isSame = false;

break;

}

}

return isSame;

}

public static void main(String[] args) {

// 一、每一列运用列交换作为第1列

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

// 复制源矩阵到临时矩阵

for (int i = 0; i < rows; i++)

for (int j = 0; j < columns; j++)

tempMatrix[i][j] = source[i][j];

count = 0;

exchangecolumns(0, k); // 进行列交换

// 二、对每一行进行判断，如果元素与目标元素不相等，则进行行翻转处理

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

if (tempMatrix[i][0] != target[i][0])

turnrow(i); // 进行行翻转

}

// 三、向后判断后面列是否相同

boolean found = true;

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

found = false;

// 1 先判断tempMatrix当前列是否与目标矩阵的当前列相同

if (samecolumns(i, i)) {

found = true;

continue;

}

// 2 tempMatrix当前列与目标矩阵的当前列不相同时，看tempMatrix当前列之后的列是否与目标矩阵当前列相同

// 1）若之后有相同的列，则进行列交换操作，继续比较后面的列

// 2）若之后也没有相同的列，则说明此次变换不能变换成目标矩阵，进行下一种变换尝试

for (int j = i + 1; j < columns; j++) {

if (samecolumns(i, j)) {

exchangecolumns(i, j); // 进行列交换

found = true;

break;

}

}

if (!found)

break;

}

if (found)

best = (count < best) ? count : best;

}

if (best < 9999)

System.out.println("至少需要 " + best + " 步完成矩阵变换");

else

System.out.println("无解");

}

}