|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 题号 | 题目 | 知识点 | 难度 | 通过率 |
| MT1 | 最大差值 | 贪心 | 简单 | 37.67% |
| MT2 | 棋子翻转 | 数组模拟 | 中等 | 27.11% |
| MT3 | 拜访 | 动态规划 | 较难 | 21.28% |
| MT4 | 直方图内最大矩形 | 动态规划分治 | 中等 | 25.77% |
| MT5 | 字符串计数 | 动态规划字符串 | 中等 | 26.74% |
| MT6 | 平均年龄 | 数组 | 困难 | 13.74% |
| MT7 | 字符编码 | 字符串贪心 | 简单 | 29.33% |
| MT8 | 奇数位丢弃 | 数组模拟链表 | 简单 | 25.72% |
| MT9 | 二维数组打印 | 数组模拟 | 简单 | 30.44% |
| MT10 | 股票交易日 | 动态规划穷举 | 简单 | 30.67% |
| MT11 | 丢失的三个数 | 哈希 | 中等 | 21.55% |
| MT12 | 网格走法数目 | 动态规划数学 | 中等 | 36.89% |
| MT13 | 拼凑面额 | 动态规划 | 中等 | 26.24% |
| MT14 | 整数加法 | 数组 | 中等 | 19.79% |
| MT15 | 病毒传播 | 图搜索 | 中等 | 19.97% |
| MT16 | 公交车 | 图动态规划 | 中等 | 27.71% |
| MT17 | 共享单车 | 图搜索 | 中等 | 18.63% |
| MT18 | 重要节点 |  | 中等 | 27.93% |
| MT19 | K | 穷举 | 中等 | 19.56% |
| MT20 | 抽牌 | 模拟 | 较难 | 31.77% |
| MT21 | 公约数 | 数学 | 较难 | 22.28% |
| MT22 | 双袋购物 | 动态规划 | 较难 | 35.19% |
| MT23 | 友好城市 | 图dfs | 较难 | 28.15% |
| MT24 | 吃鱼 |  | 简单 | 31.88% |
| MT25 | 超链接 | 哈希 | 中等 | 44.71% |
| MT26 | 改考卷 | 模拟 | 中等 | 44.88% |
| MT27 | 交错序列 | 字符串 | 简单 | 45.62% |
| MT28 | 手机号 | 字符串 | 中等 | 34.25% |
| MT29 | 题目列表 |  | 简单 | 40.53% |
| MT30 | 序列操作 | 队列模拟 | 较难 | 21.57% |
| MT31 | 硬币兑换 |  | 简单 | 37.99% |
| MT32 | 图的闭包 | 图 | 中等 | 33.57% |
| MT33 | 最小排列 |  | 简单 | 23.79% |
| MT34 | 数字构造 | 贪心 | 中等 | 41.11% |
| MT35 | 字符串距离 | 字符串 | 较难 | 13.63% |
| MT36 | 连线游戏 |  | 简单 | 51.90% |
| MT37 | 魔法表 | 模拟 | 简单 | 33.25% |
| MT38 | 贪心的小博 | 数学 | 简单 | 37.67% |
| MT39 | 被 | 数学 | 中等 | 22.56% |
| MT40 | 关灯游戏 | 模拟 | 中等 | 46.70% |
| MT41 | 射击训练 | 动态规划 | 中等 | 27.16% |
| MT42 | bit位数 | 位运算 | 简单 | 45.88% |
| MT43 | 数字字符 | 搜索贪心 | 中等 | 44.50% |
| MT44 | 图的遍历 | 递归图贪心树 | 中等 | 15.82% |
| MT45 | 最长全1串 | 数组贪心 | 中等 | 22.43% |
| MT46 | 外卖满减 | 动态规划 | 中等 | 26.74% |
| MT47 | 种花 | 数组贪心 | 中等 | 33.54% |
| MT48 | 考试策略 | 动态规划 | 中等 | 33.67% |
| MT49 | 路由器 | 数组穷举 | 中等 | 23.44% |
| MT50 | D塔2 | 模拟 | 中等 | 28.54% |

# MT1 最大差值

题目描述

有一个长为n的数组A，求满足0≤a≤b<n的A[b]-A[a]的最大值。

给定数组A及它的大小n，请返回最大差值。

测试样例：

[10,5],2

返回：0

import java.util.\*;

public class LongestDistance {

public int getDis(int[] A, int n) {

int max = 0;

int cur = A[0];

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

if(A[i]-cur >max){

max = A[i] - cur;

}

if(A[i]<cur){

cur = A[i];

}

}

return max;

}

}

# MT2 棋子翻转

题目描述

在4x4的棋盘上摆满了黑白棋子，黑白两色的位置和数目随机其中左上角坐标为(1,1),右下角坐标为(4,4),现在依次有一些翻转操作，要对一些给定支点坐标为中心的上下左右四个棋子的颜色进行翻转，请计算出翻转后的棋盘颜色。

给定两个数组A和f,分别为初始棋盘和翻转位置。其中翻转位置共有3个。请返回翻转后的棋盘。

测试样例：

[[0,0,1,1],[1,0,1,0],[0,1,1,0],[0,0,1,0]],[[2,2],[3,3],[4,4]]

返回：[[0,1,1,1],[0,0,1,0],[0,1,1,0],[0,0,1,0]]

import java.util.\*;

public class Flip {

public int[][] flipChess(int[][] a, int[][] b) {

// write code here

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 2; j++) {

b[i][j]--;

}

}

for (int i = 0; i < 3; i++) {

// a[b[i][0]][b[i][1]]

// 上

if (b[i][0] - 1 >= 0 && b[i][0] - 1 <= 3)

a[b[i][0] - 1][b[i][1]] = a[b[i][0] - 1][b[i][1]] == 1 ? 0 : 1;

// 下

if (b[i][0] + 1 >= 0 && b[i][0] + 1 <= 3)

a[b[i][0] + 1][b[i][1]] = a[b[i][0] + 1][b[i][1]] == 1 ? 0 : 1;

// 左

if (b[i][1] - 1 >= 0 && b[i][1] - 1 <= 3)

a[b[i][0]][b[i][1] - 1] = a[b[i][0]][b[i][1] - 1] == 1 ? 0 : 1;

// 右

if (b[i][1] + 1 >= 0 && b[i][1] + 1 <= 3)

a[b[i][0]][b[i][1] + 1] = a[b[i][0]][b[i][1] + 1] == 1 ? 0 : 1;

}

return a;

}

}

# MT3 拜访

题目描述

现在有一个城市销售经理，需要从公司出发，去拜访市内的商家，已知他的位置以及商家的位置，但是由于城市道路交通的原因，他只能在左右中选择一个方向，在上下中选择一个方向，现在问他有多少种方案到达商家地址。

给定一个地图map及它的长宽n和m，其中1代表经理位置，2代表商家位置，-1代表不能经过的地区，0代表可以经过的地区，请返回方案数，保证一定存在合法路径。保证矩阵的长宽都小于等于10。

测试样例：

[[0,1,0],[2,0,0]],2,3

返回：2

import java.util.\*;

public class Visit {

public int countPath(int[][] map, int n, int m) {

// write code here

int[] s = new int[2];

int[] e = new int[2];

int[][] plan = new int[n][m];

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

for(int j = 0; j < map[0].length; j++) {

if (map[i][j] == 1) {

s[0] = i;

s[1] = j;

}

if (map[i][j] == 2) {

e[0] = i;

e[1] = j;

}

}

}

int dx = s[0] > e[0] ? -1 : 1;

int dy = s[1] > e[1] ? -1 : 1;

for(int i = s[0]; i != e[0] + dx; i += dx) {

for(int j = s[1]; j != e[1] + dy; j += dy) {

if (i == s[0] && j == s[1]) plan[i][j] = 1;

else if(i == s[0]){ //X外边界初始化(X固定,方案等于上次Y方案)

plan[i][j] = (map[i][j] == -1) ? 0 : plan[i][j - dy];

}else if(j == s[1]){ //Y外边界初始化

plan[i][j] = (map[i][j] == -1) ? 0 : plan[i - dx][j];

}else{

//当前可达时,方案数=上一点(注意是-)沿x方向走到当前点的方案数+上一点沿y方向走到当前点的方案数

plan[i][j] = (map[i][j] == -1) ? 0 : plan[i - dx][j] + plan[i][j-dy];

//等于-1:当前点不可达,方案数为0;否则可达,方案数为之前两个方向方案和

}

}

}

return plan[e[0]][e[1]];

}

}

# MT4 直方图内最大矩形

题目描述

有一个直方图，用一个整数数组表示，其中每列的宽度为1，求所给直方图包含的最大矩形面积。比如，对于直方图[2,7,9,4],它所包含的最大矩形的面积为14(即[7,9]包涵的7x2的矩形)。

给定一个直方图A及它的总宽度n，请返回最大矩形面积。保证直方图宽度小于等于500。保证结果在int范围内。

测试样例：

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

返回：14

import java.util.\*;

import java.util.stream.Collectors;

//思路：对于【2，7，9，4，1】，分别求出2\*4，7\*2，9\*1，4\*3，1\*5.

//就是求每个元素所能组成最大矩形的大小，拿4举例子，先向左查找大于4的元素，

//如果>4,width++，否则break，同理向右查找。

public class MaxInnerRec {

public int countArea(int[] A, int n) {

int max = 0;

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

int width = 1;//初始为1，算自己

//向左查

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

if(A[right] >= A[i])

width++;

else

break;

}

//向右查

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

if(A[left] >= A[i])

width++;

else

break;

}

if(max < A[i] \* width)

max = A[i] \* width;

}

return max;

}

}

# MT5 字符串计数

题目描述

求字典序在s1和s2之间的，长度在len1到len2的字符串的个数，结果mod 1000007。

输入描述:

每组数据包涵s1（长度小于100），s2（长度小于100），len1（小于100000），len2（大于len1，小于100000）

输出描述:

输出答案。

示例1

输入

复制

ab ce 1 2

输出

复制

56

import java.io.\*;

import java.util.\*;

public class Main{

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

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

String s = br.readLine();

while(s != null){

System.out.println(countTest(s.split(" ")));

s = br.readLine();

}

}

private static int countTest(String[] str){

char[] ch1 = str[0].toCharArray();

char[] ch2 = str[1].toCharArray();

int len1 = Integer.parseInt(str[2]);

int len2 = Integer.parseInt(str[3]);

long res = 0;

for(int i = len1;i<=len2;i++){

long sum1 = 0;

long sum2 = 0;

int min =ch1.length > i ? i : ch1.length;

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

sum1 = sum1 + (ch1[j] - 'a') \* (long)Math.pow(26,i-1-j);

}

min = ch2.length > i ? 1 : ch2.length;

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

sum2 = sum2 + (ch2[j] - 'a') \* (long)Math.pow(26,i-1-j);

}

res = res+sum2-sum1;

}

res--;

res = res%1000007;

return (int)res;

}

}

# MT6 平均年龄

题目描述

已知某公司总人数为W，平均年龄为Y岁(每年3月末计算，同时每年3月初入职新人)，假设每年离职率为x，x>0&&x<1,每年保持所有员工总数不变进行招聘，新员工平均年龄21岁。

从今年3月末开始，请实现一个算法，可以计算出第N年后公司员工的平均年龄。(最后结果向上取整)。

输入描述:

输入W Y x N

输出描述:

输出第N年后的平均年龄

示例1

输入

复制

5 5 0.2 3

输出

复制

15

import java.util.\*;

import java.io.\*;

public class Main{

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

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

String str = null;

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

String[] ss = str.split(" ");

int W = Integer.parseInt(ss[0]);

int Y = Integer.parseInt(ss[1]);

double x = Double.parseDouble(ss[2]);

int N = Integer.parseInt(ss[3]);

System.out.println(cal(W, Y, x, N));

}

bf.close();

}

public static int cal (int W, int Y, double x, int N){

double y = Y;

double w = W \* x;

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

y++;

y = (double)((W - w) \* y + 21 \* w) / W;

}

return (int)(y % 1 != 0 ? y / 1 + 1 : y / 1);

}

}

# MT7 字符编码

题目描述

请设计一个算法，给一个字符串进行二进制编码，使得编码后字符串的长度最短。

输入描述:

每组数据一行，为待编码的字符串。保证字符串长度小于等于1000。

输出描述:

一行输出最短的编码后长度。

示例1

输入

复制

MT-TECH-TEAM

输出

复制

33

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.HashMap;

import java.util.PriorityQueue;

/\*\*

哈夫曼编码

1.将字符串转化为字符数组，遍历统计每个字符出现的次数，放入hash表中。

2.有序队列中存入map中的值

\*/

public class Main {

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

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

String line = "";

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

HashMap<Character,Integer> map = new HashMap<>();

for (char c : line.toCharArray()) map.put(c,map.getOrDefault(c,0)+1);

PriorityQueue<Integer> pq = new PriorityQueue<>();

for (int v : map.values()) pq.offer(v);

int ans = 0;

while (pq.size() > 1){

int a = pq.poll(),b = pq.poll();

ans += a + b;

pq.offer(a + b);

}

System.out.println(ans);

}

}

}

# MT8 奇数位丢弃

题目描述

对于一个由0..n的所有数按升序组成的序列，我们要进行一些筛选，每次我们取当前所有数字中从小到大的第奇数位个的数，并将其丢弃。重复这一过程直到最后剩下一个数。请求出最后剩下的数字。

输入描述:

每组数据一行一个数字，为题目中的n(n小于等于1000)。

输出描述:

一行输出最后剩下的数字。

示例1

输入

复制

500

输出

复制

255

import java.util.\*;

import java.io.\*;

public class Main{

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

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

String s = br.readLine();

while(s != null){

System.out.println(solution(s));

s = br.readLine();

}

}

static int solution(String s){

int num = Integer.parseInt(s);

int k = 0;

while(Math.pow(2,k) < num + 1){

k++;

}

return (int)(Math.pow(2,k - 1) - 1);

}

}

# MT9 二维数组打印

题目描述

有一个二维数组(n\*n),写程序实现从右上角到左下角沿主对角线方向打印。

给定一个二位数组arr及题目中的参数n，请返回结果数组。

测试样例：

[[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16]],4

返回：[4,3,8,2,7,12,1,6,11,16,5,10,15,9,14,13]

import java.util.\*;

public class Printer {

public int[] arrayPrint(int[][] arr, int n) {

int[] A = new int[n\*n];

int p = 0;

for(int p1=n-1;p1>-1;p1--){

for(int p2=0;p2<n-p1;p2++){

A[p] = arr[p2][p1+p2];

A[n\*n-1-p] = arr[n-1-p2][n-1-p1-p2];

p++;

}

}

return A;

}

}

# MT10 股票交易日

题目描述

在股市的交易日中，假设最多可进行两次买卖(即买和卖的次数均小于等于2)，规则是必须一笔成交后进行另一笔(即买-卖-买-卖的顺序进行)。给出一天中的股票变化序列，请写一个程序计算一天可以获得的最大收益。请采用实践复杂度低的方法实现。

给定价格序列prices及它的长度n，请返回最大收益。保证长度小于等于500。

测试样例：

[10,22,5,75,65,80],6

返回：87

import java.util.\*;

public class Stock {

public int maxProfit(int[] prices, int n) {

// write code here

int firstBuy = Integer.MIN\_VALUE,firstSell = 0;

int secondBuy = Integer.MIN\_VALUE,secondSell = 0;

for (int price : prices) {

firstBuy = Math.max(firstBuy, -price); //3

firstSell = Math.max(firstSell, firstBuy + price);//6

secondBuy = Math.max(secondBuy, firstSell - price);//3

secondSell = Math.max(secondSell, secondBuy+price);//9

}

return secondSell;

}

}

# MT11 丢失的三个数

题目描述

现在有一个数组，其值为从1到10000的连续增长的数字。出于某次偶然操作，导致这个数组中丢失了某三个元素，同时顺序被打乱，现在需要你用最快的方法找出丢失的这三个元素，并且将这三个元素根据从小到大重新拼接为一个新数字，计算其除以7的余数。 例：丢失的元素为336，10，8435，得到的新数字为103368435，除以七的余数为2。

输入描述:

输入数据为一行，包含9997个数字，空格隔开。

输出描述:

输出为一行，包含一个数字。

示例1

输入

复制

同题设例子输入

输出

复制

2

import java.util.\*;

import java.io.\*;

public class Main{

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

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

String str = bf.readLine();

String[] arr = str.split(" ");

int[] nums = new int[arr.length];

StringBuffer sb = new StringBuffer("");

int carry = 0;

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

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

Arrays.sort(nums);

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

if (i + 1 + carry != nums[i]) {

sb.append(i + 1 + carry);

carry++;

}

}

long res = Long.valueOf(sb.toString());

System.out.println(res % 7);

}

}

# MT12 网格走法数目

题目描述

有一个X\*Y的网格，小团要在此网格上从左上角到右下角，只能走格点且只能向右或向下走。请设计一个算法，计算小团有多少种走法。给定两个正整数int x,int y，请返回小团的走法数目。

输入描述:

输入包括一行，逗号隔开的两个正整数x和y，取值范围[1,10]。

输出描述:

输出包括一行，为走法的数目。

示例1

输入

复制

3 2

输出

复制

10

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.io.IOException;

public class Main {

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

BufferedReader br = new BufferedReader(

new InputStreamReader(System.in));

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

int x = Integer.parseInt(strs[0]);

int y = Integer.parseInt(strs[1]);

int[][] dp = new int[x + 1][y + 1];

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

dp[i][0] = 1;

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

dp[0][j] = 1;

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

for (int j = 1; j <= y; j++) {

dp[i][j] = dp[i - 1][j] + dp[i][j - 1];

}

}

System.out.println(dp[x][y]);

}

}

# MT13 拼凑面额

题目描述

给你六种面额1、5、10、20、50、100元的纸币，假设每种币值的数量都足够多，编写程序求组成N元（N为0-10000的非负整数）的不同组合的个数。

输入描述:

输入为一个数字N，即需要拼凑的面额

输出描述:

输出也是一个数字，为组成N的组合个数。

示例1

输入

复制

5

输出

复制

2

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.io.IOException;

import java.util.Arrays;

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().trim());

int[] money = new int[]{1, 5, 10, 20, 50, 100};

long[] dp = new long[n + 1];

Arrays.fill(dp, 1);

for(int i = 1; i < 6; i++){

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

if(j >= money[i])

dp[j] += dp[j - money[i]];

}

}

System.out.println(dp[n]);

}

}

# MT14 整数加法

题目描述

请设计一个算法能够完成两个用字符串存储的整数进行相加操作，对非法的输入则返回error

输入描述:

输入为一行，包含两个字符串，字符串的长度在[1,100]。

输出描述:

输出为一行。合法情况输出相加结果，非法情况输出error

示例1

输入

复制

123 123

abd 123

输出

复制

246

Error

import java.io.\*;

import java.util.\*;

public class Main{

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

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

String str = br.readLine();

while (str!=null){

String[] strs = str.split(" ");

int n1 = strs[0].length();

int n2 = strs[1].length();

int add = 0;

String res = "";

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

int a = 0;

int b = 0;

if (i >= n1 && i >= n2 && add == 0) break;

if (i <= n1 - 1){

a = strs[0].charAt(n1-1-i) - '0';

if (!(a>=0 && a<=9)){

res = "error";

break;

}

}

if (i <= n2 - 1){

b = strs[1].charAt(n2-1-i) - '0';

if (!(b>=0 && b<=9)){

res = "error";

break;

}

}

res = ((add + a + b)%10) + res;

add = (add + a + b)/10;

}

System.out.println(res);

str = br.readLine();

}

}

}

# MT15 病毒传播

题目描述

给出一个图G(V,E)，图上有n个点，m条边，所有的边都是无向边。

最开始，也就是第0天的时候，这n个点中有一个点v感染了病毒，之后的每一天，凡是感染病毒的点都会向它的邻居点传播病毒。经过了t天之后，得到了感染病毒的点集S。要求找出第0天感染病毒的点v。如果v有很多不同的答案，把它们都找出来。

输入描述:

第一行两个数n，m，接下来有m行，每行两个数u，v，表示点u，v之间有一条无向边。接下来一行两个数k，t，其中k表示集合S的大小。最后一行k个数，集合S中的元素。输入的图可能有自环和重边，输入保证S中的数互不相同。（1≤n≤103,0≤m≤103,1≤t≤109,1≤u,v,k≤n,S中所有元素在区间[1,n]）

输出描述:

输出一行，如果不存在这样的v，输出-1。  
否则输出所有可能的v，按照从小到大的顺序输出，数字之间用空格隔开，不要在行末输出多余的空格。

示例1

输入

[复制](https://www.nowcoder.com/practice/javascript:void(0);)

4 3

3 2

1 2

1 4

3 2

4 2 1

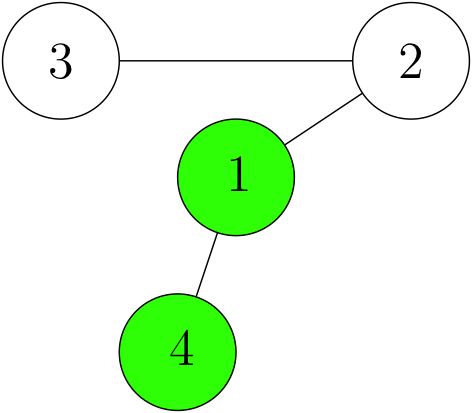
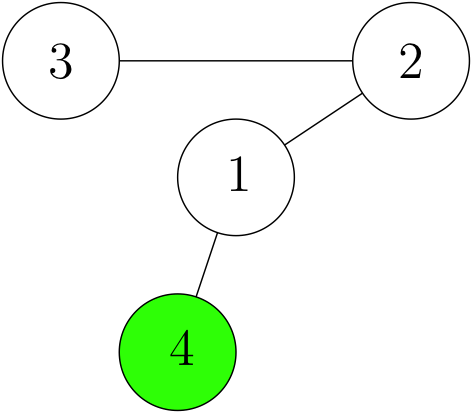
输出

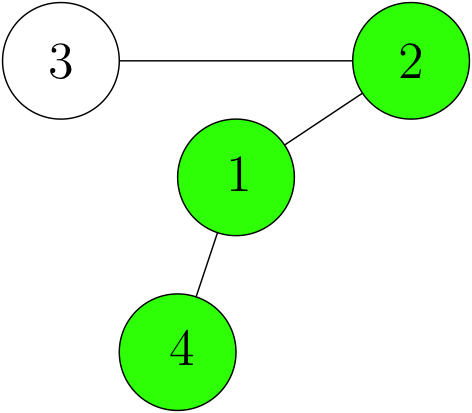
[复制](https://www.nowcoder.com/practice/javascript:void(0);)

4

说明

第0天，第1天，第2天感染病毒的点如图





import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.ArrayList;

import java.util.LinkedList;

public class Main {

//感染病毒的点为true, 未感染的为false

static boolean[] infected;

static ArrayList<Integer>[] graph;

static int n, m, k, t;

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

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

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

n = Integer.parseInt(strs[0]);

m = Integer.parseInt(strs[1]);

infected = new boolean[n + 1];

graph = new ArrayList[n + 1];

for (int i = 1; i <= n; i++) graph[i] = new ArrayList<>();

//建图

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

strs = br.readLine().split(" ");

int u = Integer.parseInt(strs[0]), v = Integer.parseInt(strs[1]);

graph[u].add(v);

graph[v].add(u);

}

strs = br.readLine().split(" ");

k = Integer.parseInt(strs[0]);

t = Integer.parseInt(strs[1]);

for (String s : br.readLine().split(" ")) {

infected[Integer.parseInt(s)] = true;

}

int res = 0;

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

if (infected[i] && bfs(i)) {

System.out.print((res++ > 0 ? " " : "") + i);

}

}

System.out.println((res == 0) ? "-1" : "");

}

//以x为起点传播t天的结果和实际结果比较是否相同

private static boolean bfs(int x) {

//每个点被传染需要的时间, 为0表明没有被传染

int[] cost = new int[n + 1];

LinkedList<Integer> queue = new LinkedList<>();

cost[x] = 1;

queue.offer(x);

while (!queue.isEmpty()) {

int cur = queue.poll();

if (cost[cur] > t) break;

for (Integer e : graph[cur]) {

if (cost[e] == 0) {

cost[e] = cost[cur] + 1;

queue.offer(e);

}

}

}

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

if (!infected[i] && cost[i] != 0) return false;

if (infected[i] && cost[i] == 0) return false;

}

return true;

}

}

# MT16 公交车

题目描述

输入描述:

输出描述:

示例1

输入

复制

5 3

3 1 2 3

3 3 4 2

3 3 5 4

输出

复制

2

说明

样例解释：先坐第1班公交从1号站点到3号站点，再做第3班公交从3号站点到5号站点，一共花费2元。

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.\*;

public class Main {

static ArrayList<Integer>[] graph;

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

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

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

int n = Integer.parseInt(strs[0]), m = Integer.parseInt(strs[1]);

graph = new ArrayList[n + m + 1];

//建图, 将公交车抽象成一个站点, 只记录公交车到相应站点的边

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

strs = br.readLine().split(" ");

graph[i + n] = new ArrayList<>();

for (int j = 1; j < strs.length; j++) {

int station = Integer.parseInt(strs[j]);

if (graph[station] == null) graph[station] = new ArrayList<>();

graph[station].add(i + n);

graph[i + n].add(station);

}

}

//bfs求最短路,

int res = bfs(1, n, m);

//因为设立了抽象点,起点距离又设为1,所以每个点的距离是实际距离的两倍加一

System.out.println(res == 0 ? "-1" : res / 2);

}

private static int bfs(int x, int n, int m) {

int[] dist = new int[n + m + 1];

LinkedList<Integer> queue = new LinkedList<>();

dist[x] = 1;

queue.offer(x);

while (!queue.isEmpty()) {

int cur = queue.poll();

for (Integer e : graph[cur]) {

if (dist[e] == 0) {

dist[e] = dist[cur] + 1;

queue.offer(e);

}

}

}

return dist[n];

}

}

# MT17 共享单车

给出一张图，图上有n个节点，从1到n编号，和m条边，每条边有一个权重，表示小明走路通过这条边的时间，所有的边都是无向的。

小明从1号节点出发，他要去n号节点，他想用的总时间尽可能的短。小明有共享单车的APP，图上有些节点有共享单车，当他到达一个有车节点后，他就可以一直骑车。如果一条边走路通过的时间是t那么骑车通过的时间就是t/2，这里的除法是向下取整.如t=1时t/2=0，t=2时，t/2=1。

小明可以先走到一个节点取车再骑车过去，也可以直接走过去，问他在最优策路下，需要多少时间从1到n。

输入描述:

第一行两个数n.m（1sn.m≤105），接下夹有m行，每行三个数u.v.w（1≤u.V≤n.1s w≤10），表示u和v之间有一条边权为w的无向边。

接下一个数k（0sk≤n），表示有k个节点有车，最后输入k行表示有车节点的编号。

输入的图中可能有自环和重边。

输出描述:

输出一个数，从1到n的最少所需的时间。如哭1和n不连通，输出-1。

输入

复制

4 3

1 3 1

1 2 3

2 4 4

1

3

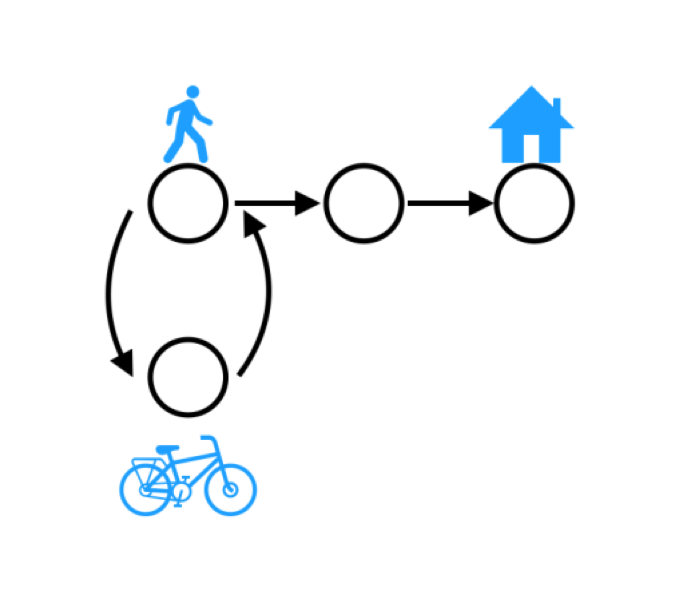
输出

复制

4

说明

小明走过去拿到车，然后骑车去目的地，路线入图：



import java.io.\*;

import java.util.\*;

public class Main {

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

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

String str = null;

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

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

String[] params = str.split(" ");

int n = Integer.parseInt(params[0]), m = Integer.parseInt(params[1]);

ArrayList<int[]>[] graph = new ArrayList[n\*2+1];

for (int i = 1; i < graph.length; i++) graph[i] = new ArrayList<int[]>();

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

params = br.readLine().split(" ");

int u = Integer.parseInt(params[0]), v = Integer.parseInt(params[1]), w = Integer.parseInt(params[2]);

graph[u].add(new int[] {v, w});

graph[v].add(new int[] {u, w});

graph[u+n].add(new int[] {v+n, w/2});

graph[v+n].add(new int[] {u+n, w/2});

}

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

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

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

graph[v].add(new int[] {v+n, 0});

}

graph[2\*n].add(new int[] {n, 0});

Long[] reach = new Long[2\*n+1];

PriorityQueue<Node> minHeap = new PriorityQueue<Node>((a, b)->(a.dist==b.dist?0:(a.dist>b.dist?1:-1)));

reach[1] = 0L;

int index = 1;

while (reach[n] == null && index > 0) {

for (int[] next: graph[index]) {

if (reach[next[0]] != null) continue;

minHeap.add(new Node(next[0], reach[index] + next[1]));

}

while (!minHeap.isEmpty() && reach[minHeap.peek().index] != null) minHeap.poll();

if (minHeap.isEmpty()) index = 0;

else {

Node next = minHeap.poll();

index = next.index;

reach[index] = next.dist;

}

}

if (reach[n] == null) System.out.println(-1);

else System.out.println(reach[n]);

}

br.close();

}

private static class Node {

int index;

long dist;

Node (int index, long dist) {

this.index = index;

this.dist = dist;

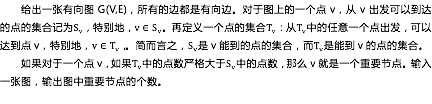
}

}

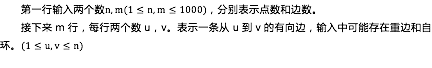
}

# MT18 重要节点

题目描述



输入描述:



输出描述:

IMG_256

示例1

输入

复制

4 3

2 1

3 1

1 4

输出

复制

2

说明

样例解释：重要节点是1，4。

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.ArrayList;

import java.util.LinkedList;

public class Main {

//arrive[i][j]为true表示从i可以到达j, false则不能

static boolean[][] arrive;

static int[] sv;

static int[] tv;

static ArrayList<Integer>[] graph;

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

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

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

int n = Integer.parseInt(strs[0]), m = Integer.parseInt(strs[1]);

arrive = new boolean[n + 1][n + 1];

sv = new int[n + 1];

tv = new int[n + 1];

graph = new ArrayList[n + 1];

for (int i = 1; i <= n; i++) graph[i] = new ArrayList<>();

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

strs = br.readLine().split(" ");

int u = Integer.parseInt(strs[0]), v = Integer.parseInt(strs[1]);

graph[u].add(v);

}

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

//根据arrive数组计算每个点能达到的点数和能达到每个点的点数

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

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

if (arrive[i][j]) {

sv[i]++;

tv[j]++;

}

}

}

int res = 0;

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

if (tv[i] > sv[i]) res++;

}

System.out.println(res);

}

//找出i这个点所能到达的所有点并在arrive数组中进行标记

private static void bfs(int i) {

LinkedList<Integer> queue = new LinkedList<>();

queue.offer(i);

arrive[i][i] = true;

while (!queue.isEmpty()) {

int cur = queue.poll();

for (Integer e : graph[cur]) {

if (!arrive[i][e]) {

arrive[i][e] = true;

queue.offer(e);

}

}

}

}

}

# MT19 K的倍数

题目描述

序列中任意个连续的元素组成的子序列称为该序列的子串。

现在给你一个序列P和一个整数K，询问元素和是K的倍数的子串的最大长度。

比如序列【1,2,3,4,5】，给定的整数K为 5，其中满足条件的子串为{5}、{2,3}、{1,2,3,4}、{1,2,3,4,5}，

那么答案就为 5，因为最长的子串为{1,2,3,4,5};如果满足条件的子串不存在，就输出 0。

输入描述:

第一行包含一个整数N, 1 ≤ 𝑁 ≤ 105。

第二行包含 N 个整数𝑝𝑖，𝑝𝑖表示序列P第i个元素的值。0 ≤ 𝑝𝑖 ≤ 105。第三行包含一个整数K，1 ≤ 𝐾 ≤ 105。

输出描述:

输出一个整数ANS，表示答案。

示例1

输入

复制

5

1 2 3 4 5

5

输出

复制

5

示例2

输入

复制

6

3 1 2 7 7 7

4

输出

复制

5

import java.util.\*;

import java.io.BufferedReader;

import java.io.InputStreamReader;

public class Main{

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

/\*

将之前输入得到的序列第i位变成包括他的前i位的和，

构建一个序列用于存储第一次出现某个余数对应的序号i，当之后再出现相同余数，

则之间的序列和能被k整除（对于余数为0的情况，只要出现整除情况就表示整个序列能被k整除）

\*/

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

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

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

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

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

num[0]=0;

int i,j;

for (i = 1; i <= N ; i++) {

num[i] = (Integer.parseInt(strings[i-1]) + num[i-1])%k;

}

int ANS = 0;

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

for (j = N; j - i > ANS; j--) {

if ((num[j]==num[i])){

ANS = j-i;

}

}

}

System.out.println(ANS);

}

}

# MT20 抽牌

题目描述

桌面上放了一堆牌，牌从上到下由1到n编号。每张牌上写有一个数字，第i张牌上的数字为a.小明和小方在玩轮流取牌的游戏，每次每个人只能取一张牌，小明先取牌。

取牌规则如下，当一个人取牌时，他只能取当前牌堆中的最上面一张或者最下面一张。小明和小方都采用随机取牌的策略，小明每次以概率p取最上面一张牌，以概率1-p取最下面一张牌：小方每次以概率q取最上面一张牌，以概率1-q取最下面一张牌.最后两人的得分为他们各自取到牌上的数字之和。问小明得分的数学期望。

输入描述:

第一行输入三个整数n，P，Q.（1sn≤1000，0≤P，Q≤100）。小明取最上面一张牌的概率p=P/100，小方取最上面一张牌的概率q=Q/100.

接下来一行n个数，每张牌上的数字。（1≤ai≤13）

输出描述:

输出答案，四舍五入保留三位小数。

示例1

输入

复制

2 10 90

1 2

输出

复制

1.900

示例2

输入

复制

5 10 20

1 3 4 5 13

输出

复制

18.038

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.Scanner;

public class Main {

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

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

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

int n = Integer.parseInt(strs[0]);

int P = Integer.parseInt(strs[1]), Q = Integer.parseInt(strs[2]);

double p = P / 100.0, q = Q / 100.0;

int[] a = new int[n];

strs = br.readLine().split(" ");

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

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

}

double[][] dp = new double[n][n];

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

dp[i][i] = a[i];

if (i < n - 1) {

dp[i][i + 1] = p \* a[i] + (1 - p) \* a[i + 1];

}

}

for (int i = n - 3; i >= 0; i--) {

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

dp[i][j] = p \* (q \* dp[i + 2][j] + (1 - q) \* dp[i + 1][j - 1] + a[i])

+ (1 - p) \* (q \* dp[i + 1][j - 1] + (1 - q) \* dp[i][j - 2] + a[j]);

}

}

System.out.printf("%.3f\n", dp[0][n-1]);

}

}

# MT21 公约数

题目描述

桌面上有n张牌，每张牌上写了一个数字，第i张牌的数字为a。现在从中选出K张牌把选出牌上的数字全部乘起来，得到一个数X。

问有多少种不同的选牌方案，使得X和A的最大公约数大于等于B。

输入描述:

第一行四个数n，K，A，B.（2≤50.2≤K≤n，1≤B≤A≤100000），接下夹一行n个数，每张牌上的数字（1≤ai≤100000）。

输出描述:

输出方案数。

示例1

输入

复制

5 2 12 6

4 4 1 2 3

输出

复制

3

import java.util.Scanner;

public class Main{

public static void main(String[] args){

//思路：

//显然是一道动态规划的题目

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int K = sc.nextInt();

int A = sc.nextInt();

int B = sc.nextInt();

int[] numbers = new int[n];

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

numbers[i] = sc.nextInt();

}

long count = 0;

long[][] demo = new long[K+1][A+1];

demo[1][isTrue(numbers[0],A)] = 1;//初始化

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

int j = i>K?K-1:i-1;

for(; j > 0; j--) // 选j个数

for(int mcd = A; mcd > 0; mcd--) // 最大公因数

if(demo[j][mcd] != 0)

demo[j+1][isTrue(A/mcd,numbers[i-1])\*mcd] += demo[j][mcd];

demo[1][isTrue(numbers[i-1],A)] += 1; // 只选择当前这个数

}

for(int i = B; i <= A; i++)

count += demo[K][i];

System.out.println(count);

}

static int isTrue(int x,int A){

int mid = 0;

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

mid=x%A;

if(mid==0)

break;

else{

x=A;

A=mid;

}

}

return A;

}

}

# MT22 双袋购物

# MT23 友好城市

# MT24 吃鱼

# MT25 超链接

# MT26 改考卷

# MT27 交错序列

# MT28 手机号

# MT29 题目列表

# MT30 序列操作

# MT31 硬币兑换

# MT32 图的闭包

# MT33 最小排列

# MT34 数字构造

# MT35 字符串距离

# MT36 连线游戏

# MT37 魔法表

# MT38 贪心的小博

# MT39 被

# MT40 关灯游戏

# MT41 射击训练

# MT42 bit位数

# MT43 数字字符

# MT44 图的遍历

# MT45 最长全1串

# MT46 外卖满减

# MT47 种花

# MT48 考试策略

# MT49 路由器

# MT50 D塔2