蓝桥杯软件大赛历届试题及答案

[核桃的数量 2](#_Toc431199523)

[打印十字图 2](#_Toc431199524)

[带分数 3](#_Toc431199525)

[剪格子 4](#_Toc431199526)

[错误票据 5](#_Toc431199527)

[翻硬币 7](#_Toc431199528)

[连号区间数 8](#_Toc431199529)

[买不到的数目 8](#_Toc431199530)

[大臣的旅费 9](#_Toc431199531)

[幸运数 11](#_Toc431199532)

[横向打印二叉树 12](#_Toc431199533)

[危险系数 13](#_Toc431199534)

[网络寻路 15](#_Toc431199535)

[高僧斗法 16](#_Toc431199536)

[格子刷油漆 17](#_Toc431199537)

[农场阳光 18](#_Toc431199538)

[约数倍数选卡片 18](#_Toc431199539)

[九宫重排 19](#_Toc431199540)

[公式求值 22](#_Toc431199541)

[回文数字 24](#_Toc431199542)

[国王的烦恼 25](#_Toc431199543)

[数字游戏 27](#_Toc431199544)

[邮局 27](#_Toc431199545)

[城市建设 28](#_Toc431199546)

[最大子阵 30](#_Toc431199547)

[蚂蚁感冒 31](#_Toc431199548)

[地宫取宝 32](#_Toc431199549)

[斐波那契 33](#_Toc431199550)

[波动数列 34](#_Toc431199551)

[小朋友排队 35](#_Toc431199552)

[分糖果 36](#_Toc431199553)

[矩阵翻硬币 37](#_Toc431199554)

[兰顿蚂蚁 38](#_Toc431199555)

### 核桃的数量

问题描述

小张是软件项目经理，他带领3个开发组。工期紧，今天都在加班呢。为鼓舞士气，小张打算给每个组发一袋核桃（据传言能补脑）。他的要求是：

1. 各组的核桃数量必须相同

2. 各组内必须能平分核桃（当然是不能打碎的）

3. 尽量提供满足1,2条件的最小数量（节约闹革命嘛）

输入格式

输入包含三个正整数a, b, c，表示每个组正在加班的人数，用空格分开（a,b,c<30）

输出格式

输出一个正整数，表示每袋核桃的数量。

样例输入1

2 4 5

样例输出1

20

样例输入2

3 1 1

样例输出2

3

import java.io.\*;

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[] arr = new int[3];

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

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

}

for (int a = 1; a < 27000; a++) {

if (a % arr[0] == 0 && a % arr[1] == 0 && a % arr[2] == 0) {

System.out.print(a);

break;

}

}

}

}

### 打印十字图

问题描述

小明为某机构设计了一个十字型的徽标（并非红十字会啊），如下所示：

..$$$$$$$$$$$$$..  
..$...........$..  
$$$.$$$$$$$$$.$$$  
$...$.......$...$  
$.$$$.$$$$$.$$$.$  
$.$...$...$...$.$  
$.$.$$$.$.$$$.$.$  
$.$.$...$...$.$.$  
$.$.$.$$$$$.$.$.$  
$.$.$...$...$.$.$  
$.$.$$$.$.$$$.$.$  
$.$...$...$...$.$  
$.$$$.$$$$$.$$$.$  
$...$.......$...$  
$$$.$$$$$$$$$.$$$  
..$...........$..  
..$$$$$$$$$$$$$..

对方同时也需要在电脑dos窗口中以字符的形式输出该标志，并能任意控制层数。

输入格式

一个正整数 n (n<30) 表示要求打印图形的层数。

输出格式

对应包围层数的该标志。

样例输入1

1

样例输出1

..$$$$$..  
..$...$..  
$$$.$.$$$  
$...$...$  
$.$$$$$.$  
$...$...$  
$$$.$.$$$  
..$...$..  
..$$$$$..

样例输入2

3

样例输出2

..$$$$$$$$$$$$$..  
..$...........$..  
$$$.$$$$$$$$$.$$$  
$...$.......$...$  
$.$$$.$$$$$.$$$.$  
$.$...$...$...$.$  
$.$.$$$.$.$$$.$.$  
$.$.$...$...$.$.$  
$.$.$.$$$$$.$.$.$  
$.$.$...$...$.$.$  
$.$.$$$.$.$$$.$.$  
$.$...$...$...$.$  
$.$$$.$$$$$.$$$.$  
$...$.......$...$  
$$$.$$$$$$$$$.$$$  
..$...........$..  
..$$$$$$$$$$$$$..

提示:请仔细观察样例，尤其要注意句点的数量和输出位置。



import java.util.Scanner;

public class Main{

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

fd(sc.nextInt());

}

public static void fd(int n){

int len=5+4\*n;

String a[][]=new String[len][len];

int b=len/2;

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

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

a[i][j]=".";

}}

for(int i=b-2;i<b+2;i++){

a[i][b]="$";

a[b][i]="$";

}

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

for(int i=b-2-2\*m;i<(b+3+2\*m);i++){

a[b - (4+m\*2)][i] = "$";

a[b + (4+m\*2)][i] = "$";

a[i][b - (4+m\*2)] = "$";

a[i][b + (4+m\*2)] = "$";

}

}

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

for (int i = b - (3+m\*2); i <= b + 3+m\*2; i++) {

a[b - (2+m\*2)][i] = "$";

a[b + 2+m\*2][i] = "$";

a[i][b + 2+m\*2] = "$";

a[i][b - (2+m\*2)]= "$";

}

a[b - (2+m\*2)][b - (1+m\*2)] = ".";

a[b - (2+m\*2)][b + 1+m\*2] = ".";

a[b + 2+m\*2][b - (1+m\*2)] = ".";

a[b + 2+m\*2][b + 1+m\*2] = ".";

a[b - (1+m\*2)][b - (2+m\*2)] = ".";

a[b + 1+m\*2][b - (2+m\*2)] = ".";

a[b - (1+m\*2)][b + 2+m\*2] = ".";

a[b + 1+m\*2][b + 2+m\*2] = ".";

}

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

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

System.out.print(a[i][j]);

}

System.out.println();

}

}

}

### 带分数

问题描述

100 可以表示为带分数的形式：100 = 3 + 69258 / 714。

还可以表示为：100 = 82 + 3546 / 197。

注意特征：带分数中，数字1~9分别出现且只出现一次（不包含0）。

类似这样的带分数，100 有 11 种表示法。

输入格式

从标准输入读入一个正整数N (N<1000\*1000)

输出格式

程序输出该数字用数码1~9不重复不遗漏地组成带分数表示的全部种数。

注意：不要求输出每个表示，只统计有多少表示法！

样例输入1

100

样例输出1

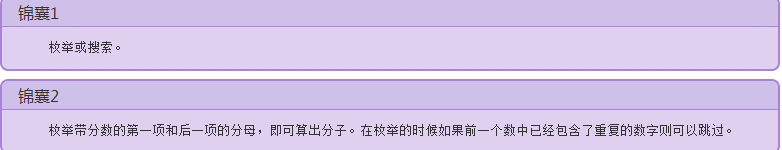
11

样例输入2

105

样例输出2

6



import java.util.Scanner;

/\*\*

\* http://lx.lanqiao.org/problem.page?gpid=T26

\* Created by revintec on 14-5-5.

\*/

public class Main{

static int getBits(int k){

int bits=0;

while(k>0){

int a=k%10;k/=10;

if(a==0) return -1;

int b=1<<a;

if((bits&b)!=0) return -1;

bits|=b;

}return bits;

}

static int getBits2(int b){

int lo=9;

for(int i=1;i<1<<10;i<<=1)

if((b&i)!=0) --lo;

return lo;

}

public static void main(String[] args){

// System.clearProperty("m.TEST");

Scanner sc=System.getProperty("m.TEST")==null?new Scanner(System.in):

new Scanner("999988");

int sx=sc.nextInt();

int[] ts={0,

10,

100,

1000,

10000,

100000,

1000000,

10000000,

100000000,

1000000000,

};

int so=0;

for(int a=1;a<sx;++a){

int bits=getBits(a);

if(bits==-1) continue;

for(int b=1;;++b){

int bts=getBits(b);

if((bits&bts)!=0) continue;

bts|=bits;

long c=b\*(sx-a);

if(c>=ts[getBits2(bts)]) break;

if((getBits((int)c)^bts)==1022)

++so;

}

}System.out.println(so);

}

}

### 剪格子

问题描述

如下图所示，3 x 3 的格子中填写了一些整数。

+--\*--+--+  
|10\* 1|52|  
+--\*\*\*\*--+  
|20|30\* 1|  
\*\*\*\*\*\*\*--+  
| 1| 2| 3|  
+--+--+--+

我们沿着图中的星号线剪开，得到两个部分，每个部分的数字和都是60。

本题的要求就是请你编程判定：对给定的m x n 的格子中的整数，是否可以分割为两个部分，使得这两个区域的数字和相等。

如果存在多种解答，请输出包含左上角格子的那个区域包含的格子的最小数目。

如果无法分割，则输出 0。

输入格式

程序先读入两个整数 m n 用空格分割 (m,n<10)。

表示表格的宽度和高度。

接下来是n行，每行m个正整数，用空格分开。每个整数不大于10000。

输出格式

输出一个整数，表示在所有解中，包含左上角的分割区可能包含的最小的格子数目。

样例输入1

3 3  
10 1 52  
20 30 1  
1 2 3

样例输出1

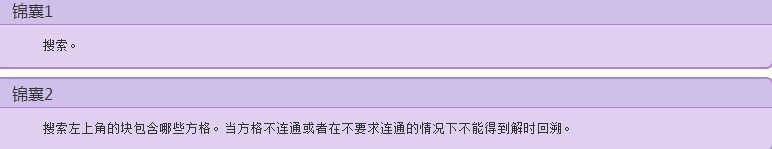
3

样例输入2

4 3  
1 1 1 1  
1 30 80 2  
1 1 1 100

样例输出2

10



import java.util.Scanner;

public class Main{

static int sum,half,m,n;

static int[][] map;

static boolean[][] flag;

static int count=100;

public static void main(String[] args) {

// TODO Auto-generated method stub

Scanner sc=new Scanner(System.in);

n=sc.nextInt();

m=sc.nextInt();

map=new int[m][n];

flag=new boolean[m][n];

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

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

map[i][j]=sc.nextInt();

sum+=map[i][j];

}

if(sum%2!=0)

System.out.println(0);

else{

half=sum/2;

dfs(0,0,1,0);

System.out.println(count==100?0:count);

}

}

private static void dfs(int i, int j, int step,int s) {

if(i<0 || i>=m || j<0 || j>=n ){

return;

}

if(flag[i][j]==true)

return;

if(s>half){

flag[i][j]=false;

return;

}

else{

s+=map[i][j];

flag[i][j]=true;

if(s==half){

if(step<count)

count=step;

}

else{

dfs(i+1,j,step+1,s);

dfs(i,j+1,step+1,s);

dfs(i-1,j,step+1,s);

dfs(i,j-1,step+1,s);

flag[i][j]=false;

}

}

}

}

### 错误票据

问题描述

某涉密单位下发了某种票据，并要在年终全部收回。

每张票据有唯一的ID号。全年所有票据的ID号是连续的，但ID的开始数码是随机选定的。

因为工作人员疏忽，在录入ID号的时候发生了一处错误，造成了某个ID断号，另外一个ID重号。

你的任务是通过编程，找出断号的ID和重号的ID。

假设断号不可能发生在最大和最小号。

输入格式

要求程序首先输入一个整数N(N<100)表示后面数据行数。

接着读入N行数据。

每行数据长度不等，是用空格分开的若干个（不大于100个）正整数（不大于100000），请注意行内和行末可能有多余的空格，你的程序需要能处理这些空格。

每个整数代表一个ID号。

输出格式

要求程序输出1行，含两个整数m n，用空格分隔。

其中，m表示断号ID，n表示重号ID

样例输入1

2  
5 6 8 11 9   
10 12 9

样例输出1

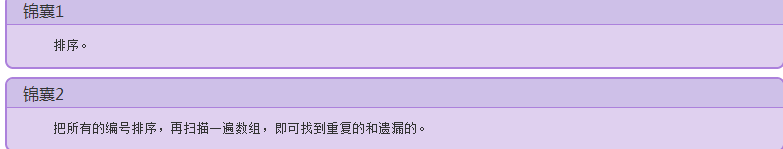
7 9

样例输入2

6  
164 178 108 109 180 155 141 159 104 182 179 118 137 184 115 124 125 129 168 196  
172 189 127 107 112 192 103 131 133 169 158   
128 102 110 148 139 157 140 195 197  
185 152 135 106 123 173 122 136 174 191 145 116 151 143 175 120 161 134 162 190  
149 138 142 146 199 126 165 156 153 193 144 166 170 121 171 132 101 194 187 188  
113 130 176 154 177 120 117 150 114 183 186 181 100 163 160 167 147 198 111 119

样例输出2

105 120



import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.ArrayList;

import java.util.List;

public class Main {

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

int duan = 0;

int chong = 0;

List<Integer> list = new ArrayList();

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

String line = br.readLine();

int len = Integer.parseInt(line);

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

{

String l = br.readLine();

String[] s = l.split(" ");

for(int j = 0; j < s.length; j++)

{

list.add(Integer.parseInt(s[j]));

}

}

for(int i = 0 ; i < list.size(); i++)

{

for(int j = 0; j < list.size(); j++)

{

int a = list.get(i);

int b = list.get(j);

int temp = 0;

if(a < b)

{

temp = a;

// a = b;

// b = temp;

list.set(i, b);

list.set(j, temp);

}

}

}

for(int i = 0 ; i < list.size() - 1; i++)

{

int a = list.get(i);

int b = list.get(i + 1);

if(a == b)

chong = a;

if(b - a > 1)

duan = b -1;

}

System.out.println(duan + " " + chong);

}

}

### 翻硬币

问题描述

小明正在玩一个“翻硬币”的游戏。

桌上放着排成一排的若干硬币。我们用 \* 表示正面，用 o 表示反面（是小写字母，不是零）。

比如，可能情形是：\*\*oo\*\*\*oooo

如果同时翻转左边的两个硬币，则变为：oooo\*\*\*oooo

现在小明的问题是：如果已知了初始状态和要达到的目标状态，每次只能同时翻转相邻的两个硬币,那么对特定的局面，最少要翻动多少次呢？

我们约定：把翻动相邻的两个硬币叫做一步操作，那么要求：

输入格式

两行等长的字符串，分别表示初始状态和要达到的目标状态。每行的长度<1000

输出格式

一个整数，表示最小操作步数。

样例输入1

\*\*\*\*\*\*\*\*\*\*  
o\*\*\*\*o\*\*\*\*

样例输出1

5

样例输入2

\*o\*\*o\*\*\*o\*\*\*  
\*o\*\*\*o\*\*o\*\*\*

样例输出2

1



import java.io.BufferedReader;

import java.io.InputStreamReader;

public class Main{

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

BufferedReader bu = new BufferedReader(

new InputStreamReader(System.in));

String s = bu.readLine();

char[] c1 = s.toCharArray();

s = bu.readLine();

char[] c2 = s.toCharArray();

int i = 0;

int c = 0;

boolean[] b = new boolean[c1.length];

boolean x = false;

while(!x){

if(c1[i] == c2[i]){

i++;

b[i] =true;

}else{

if(i < c1.length -1){

c1[i] = fz(c1[i]);

c1[i+1] = fz(c1[i+1]);

c++;

i++;

}

}

for(int i1 = 0;i1 < b.length;i1++){

if(b[i1] == true){

x = true;

} else{

x = false;

}

}

}

System.out.println(c);

}

public static char fz(char c ){

if(c == '\*'){

return 'o';

}else{

return '\*';

}

}

}

### 连号区间数

问题描述

小明这些天一直在思考这样一个奇怪而有趣的问题：

在1~N的某个全排列中有多少个连号区间呢？这里所说的连号区间的定义是：

如果区间[L, R] 里的所有元素（即此排列的第L个到第R个元素）递增排序后能得到一个长度为R-L+1的“连续”数列，则称这个区间连号区间。

当N很小的时候，小明可以很快地算出答案，但是当N变大的时候，问题就不是那么简单了，现在小明需要你的帮助。

输入格式

第一行是一个正整数N (1 <= N <= 50000), 表示全排列的规模。

第二行是N个不同的数字Pi(1 <= Pi <= N)， 表示这N个数字的某一全排列。

输出格式

输出一个整数，表示不同连号区间的数目。

样例输入1

4  
3 2 4 1

样例输出1

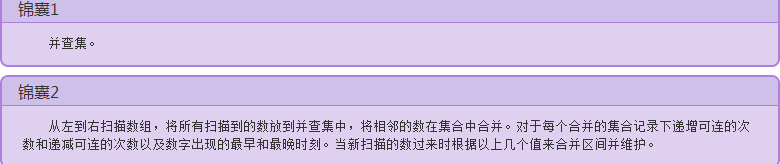
7

样例输入2

5  
3 4 2 5 1

样例输出2

9



import java.io.\*;

public class Main{

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

//long start=System.currentTimeMillis();

fun();

//System.out.println("\nTime:"+(System.currentTimeMillis()-start)+"ms");

}

public static void fun()throws IOException{

//String filename="test.txt";

//FileInputStream file=new FileInputStream(filename);

//System.setIn(file);

int[] a;

StreamTokenizer st=new StreamTokenizer(new InputStreamReader(System.in));

st.nextToken();

int n=(int)st.nval;

a=new int[n];

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

st.nextToken();

a[i]=(int)st.nval;

}

int min;

int max;

int num=0;

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

min=max=a[l];

for(int r=l;r<n;r++){

if(a[r]<min)min=a[r];

if(a[r]>max)max=a[r];

//

if(max-min==r-l)

num++;

}

}

System.out.print(num);

}

}

### 买不到的数目

问题描述

小明开了一家糖果店。他别出心裁：把水果糖包成4颗一包和7颗一包的两种。糖果不能拆包卖。

小朋友来买糖的时候，他就用这两种包装来组合。当然有些糖果数目是无法组合出来的，比如要买 10 颗糖。

你可以用计算机测试一下，在这种包装情况下，最大不能买到的数量是17。大于17的任何数字都可以用4和7组合出来。

本题的要求就是在已知两个包装的数量时，求最大不能组合出的数字。

输入格式

两个正整数，表示每种包装中糖的颗数(都不多于1000)

输出格式

一个正整数，表示最大不能买到的糖数

样例输入1

4 7

样例输出1

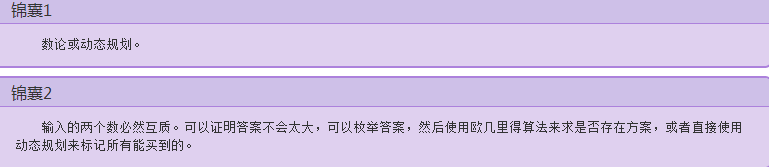
17

样例输入2

3 5

样例输出2

7



import java.io.BufferedReader;

import java.io.InputStreamReader;

public class Main {

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

BufferedReader buf = new BufferedReader(

new InputStreamReader(System.in));

String strNum = buf.readLine();

String[] num = strNum.split(" ");

buf.close();

int a, b;

a = Integer.valueOf(num[0]);

b = Integer.valueOf(num[1]);

if (a > b) {

int tem = a;

a = b;

b = tem;

}

int c = a \* b;

int tem = c;

while (tem > 0) {

if (tem % a == 0)

tem = --c;

else if (tem % b == 0)

tem = --c;

else

tem -= b;

}

System.out.println(c);

}

}

### 大臣的旅费

问题描述

很久以前，T王国空前繁荣。为了更好地管理国家，王国修建了大量的快速路，用于连接首都和王国内的各大城市。

为节省经费，T国的大臣们经过思考，制定了一套优秀的修建方案，使得任何一个大城市都能从首都直接或者通过其他大城市间接到达。同时，如果不重复经过大城市，从首都到达每个大城市的方案都是唯一的。

J是T国重要大臣，他巡查于各大城市之间，体察民情。所以，从一个城市马不停蹄地到另一个城市成了J最常做的事情。他有一个钱袋，用于存放往来城市间的路费。

聪明的J发现，如果不在某个城市停下来修整，在连续行进过程中，他所花的路费与他已走过的距离有关，在走第x千米到第x+1千米这一千米中（x是整数），他花费的路费是x+10这么多。也就是说走1千米花费11，走2千米要花费23。

J大臣想知道：他从某一个城市出发，中间不休息，到达另一个城市，所有可能花费的路费中最多是多少呢？

输入格式

输入的第一行包含一个整数n，表示包括首都在内的T王国的城市数

城市从1开始依次编号，1号城市为首都。

接下来n-1行，描述T国的高速路（T国的高速路一定是n-1条）

每行三个整数Pi, Qi, Di，表示城市Pi和城市Qi之间有一条高速路，长度为Di千米。

输出格式

输出一个整数，表示大臣J最多花费的路费是多少。

样例输入1

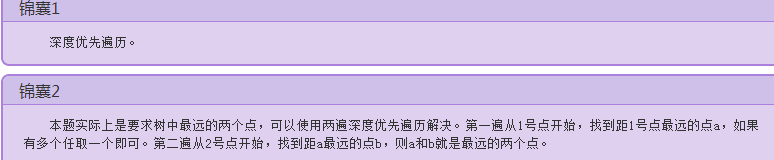
5  
1 2 2  
1 3 1  
2 4 5  
2 5 4

样例输出1

135

输出格式

大臣J从城市4到城市5要花费135的路费。



import java.io.BufferedInputStream;

import java.io.IOException;

import java.util.ArrayList;

public class Main

{

private static BufferedInputStream in = new BufferedInputStream(System.in);

private static ArrayList<Node> n = new ArrayList<Node>();

private static Integer dis = 0;

private static Integer pow = 0;

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

{

int size = readInt();

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

{

n.add(null);

}

for(int i=1; i<size; i++)

{

int x = readInt()-1;

int y = readInt()-1;

int d = readInt();

Node node = new Node();

node.nextSide = n.get(x);

node.con = y;

node.power = d;

n.set(x, node);

node = new Node();

node.nextSide = n.get(y);

node.con = x;

node.power = d;

n.set(y, node);

}

away(0, 0, -1);

pow = 0;

away(dis, 0, -1);

System.out.println(pow \* 10 + (1 + pow) \* pow /2);

}

private static void away(int index, int power, int from)

{

if(pow < power)

{

pow = power;

dis = index;

}

Node node = n.get(index);

while(node != null)

{

if(node.con == from)

{

node = node.nextSide;

continue;

}

away(node.con, power + node.power, index);

node = node.nextSide;

}

}

private static int readInt() throws IOException

{

int i,sum=0;

while(((i=in.read())&48) != 48 || i>57);

for(;(i&56) == 48 || (i&62) == 56; i=in.read())

sum = sum\*10 + (i&15);

return sum;

}

private static class Node

{

int power;

int con;

Node nextSide;

}

}

### 幸运数

问题描述

幸运数是波兰数学家乌拉姆命名的。它采用与生成素数类似的“筛法”生成

。

首先从1开始写出自然数1,2,3,4,5,6,....

1 就是第一个幸运数。

我们从2这个数开始。把所有序号能被2整除的项删除，变为：

1 \_ 3 \_ 5 \_ 7 \_ 9 ....

把它们缩紧，重新记序，为：

1 3 5 7 9 .... 。这时，3为第2个幸运数，然后把所有能被3整除的序号位置的数删去。注意，是序号位置，不是那个数本身能否被3整除!! 删除的应该是5，11, 17, ...

此时7为第3个幸运数，然后再删去序号位置能被7整除的(19,39,...)

最后剩下的序列类似：

1, 3, 7, 9, 13, 15, 21, 25, 31, 33, 37, 43, 49, 51, 63, 67, 69, 73, 75, 79, ...

输入格式

输入两个正整数m n, 用空格分开 (m < n < 1000\*1000)

输出格式

程序输出 位于m和n之间的幸运数的个数（不包含m和n）。

样例输入1

1 20

样例输出1

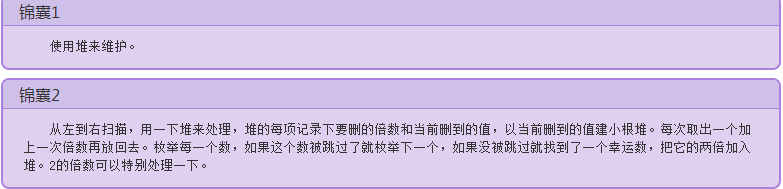
5

样例输入2

30 69

样例输出2

8



import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.util.ArrayList;

import java.util.StringTokenizer;

class Reader11{

static BufferedReader reader;

static StringTokenizer tokenizer;

static void init(InputStream input){

reader=new BufferedReader(new InputStreamReader(input));

tokenizer=new StringTokenizer("");

}

static String next() throws IOException{

while (!tokenizer.hasMoreElements()) {

tokenizer =new StringTokenizer(reader.readLine());

}

return tokenizer.nextToken();

}

static int nextInt() throws IOException{

return Integer.parseInt(next());

}

}

public class Main{

/\*\*

\* @param args

\* @throws IOException

\*/

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

// TODO Auto-generated method stub

Reader11.init(System.in);

int m=Reader11.nextInt();

int n=Reader11.nextInt();

ArrayList<Integer> a=new ArrayList<Integer>();

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

a.add(i);

}

int k=1;

int t=a.get(k);

while (t<=a.size()) {

int t0=a.get(k+1);

for (int j = 1; t\*j < a.size()+j; j++) {

a.remove(t\*j-j);

}

if (a.get(k)==t0) {

}else {

k++;

}

t=a.get(k);

}

int Mj = 0;

for (int i = 0; i < a.size(); i++) {

if (a.get(i)>m) {

Mj=i;

break;

}else if(a.get(i)==m){

Mj=i+1;

break;

}

}

System.out.println(a.size()-Mj);

}

}

### 横向打印二叉树

问题描述

二叉树可以用于排序。其原理很简单：对于一个排序二叉树添加新节点时，先与根节点比较，若小则交给左子树继续处理，否则交给右子树。

当遇到空子树时，则把该节点放入那个位置。

比如，10 8 5 7 12 4 的输入顺序，应该建成二叉树如下图所示，其中.表示空白。

...|-12  
10-|  
...|-8-|  
.......|...|-7  
.......|-5-|  
...........|-4

本题目要求：根据已知的数字，建立排序二叉树，并在标准输出中横向打印该二叉树。

输入格式

输入数据为一行空格分开的N个整数。 N<100，每个数字不超过10000。

输入数据中没有重复的数字。

输出格式

输出该排序二叉树的横向表示。为了便于评卷程序比对空格的数目，请把空格用句点代替：

样例输入1

10 5 20

样例输出1

...|-20  
10-|  
...|-5

样例输入2

5 10 20 8 4 7

样例输出2

.......|-20  
..|-10-|  
..|....|-8-|  
..|........|-7  
5-|  
..|-4



import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.StringTokenizer;

public class Main {

static int n;

static int[] a=new int[108];

static int[] left;

static int[] right;

static int[] level;

static int maxl=0;

static int M,N;

static char[][] map;

static boolean[] flag;

static boolean[] flag2;

static void add(int root,int index)

{

if(a[index]<a[root])

if(left[root]==0) {

left[root]=index; level[index]=level[root]+1;

if(level[index]>maxl) maxl=level[index]; }

else add(left[root],index);

else

if(right[root]==0) { right[root]=index; level[index]=level[root]+1; if(level[index]>maxl) maxl=level[index]; }

else add(right[root],index);

}

static void Fill(int root,int x,int y)

{

int lvl=level[root]+1;

String num=String.valueOf(a[root]);

int len=num.length();

flag[x]=true;

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

{

map[x][y-i]=num.charAt(i);

int h=2<<(maxl-lvl);

for(int j=1;j<h;j++) { map[x+j][y-i]='.'; map[x-j][y-i]='.'; }

}

if(left[root]==0 && right[root]==0 && level[root]<maxl) flag2[x]=true;

if(left[root]!=0)

Fill(left[root],x+(2<<(maxl-lvl))/2,y-8);

if(right[root]!=0)

Fill(right[root],x-(2<<(maxl-lvl))/2,y-8);

}

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

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

StringTokenizer tok=new StringTokenizer(bfr.readLine());

// long begin=System.currentTimeMillis();

int i,j;

for(i=1;tok.hasMoreTokens();i++) a[i]=Integer.parseInt(tok.nextToken());

n=i;

left =new int[n];

right =new int[n];

level=new int[n];

left[0]=1; right[0]=1; level[0]=0; level[1]=1;

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

add(1,i);

M=(2<<(maxl-1))-1;

N=8\*maxl-3;

map=new char[M][N];

flag=new boolean[M];

flag2=new boolean[M];

for(int l=1;l<maxl;l++)

{

int ty=l\*8-1;

int start=(2<<(l-1))-1,d=(2<<l);

for(i=0;i<M;i++) map[i][ty]=map[i][ty-1]=map[i][ty-2]='.';

for(i=start;i<M;i+=d)

{

map[i][ty]='-';

int hei=(2<<(l-1))/2;

for(int ii=0;ii<=hei;ii++)

{

map[i+ii][ty-1]='|';

map[i-ii][ty-1]='|';

}

map[i+hei][ty-2]='-';

map[i-hei][ty-2]='-';

}

}

Fill(1,M/2,N-1);

for(i=0;i<M;i++)

{

if(flag[i])

{

for(j=0;;j++)

if(map[i][j]!=0 && map[i][j]!='.') break;

int jj;

for(jj=N-1;jj>j+1;jj--)

if(map[i][jj]!=0)

System.out.print(map[i][jj]);

if(!flag2[i])

System.out.print(map[i][jj]+""+map[i][jj-1]);

System.out.println();

}

}

// System.out.println(System.currentTimeMillis()-begin);

}

}

### 危险系数

问题描述

抗日战争时期，冀中平原的地道战曾发挥重要作用。

地道的多个站点间有通道连接，形成了庞大的网络。但也有隐患，当敌人发现了某个站点后，其它站点间可能因此会失去联系。

我们来定义一个危险系数DF(x,y)：

对于两个站点x和y (x != y), 如果能找到一个站点z，当z被敌人破坏后，x和y不连通，那么我们称z为关于x,y的关键点。相应的，对于任意一对站点x和y，危险系数DF(x,y)就表示为这两点之间的关键点个数。

本题的任务是：已知网络结构，求两站点之间的危险系数。

输入格式

输入数据第一行包含2个整数n(2 <= n <= 1000), m(0 <= m <= 2000),分别代表站点数，通道数；

接下来m行，每行两个整数 u,v (1 <= u, v <= n; u != v)代表一条通道；

最后1行，两个数u,v，代表询问两点之间的危险系数DF(u, v)。

输出格式

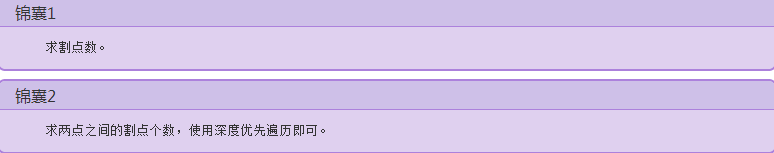
一个整数，如果询问的两点不连通则输出-1.

样例输入

7 6  
1 3  
2 3  
3 4  
3 5  
4 5  
5 6  
1 6

样例输出

2



import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.ArrayList;

import java.util.StringTokenizer;

public class Main {

static int n,m;

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

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

StringTokenizer tok=new StringTokenizer(bfr.readLine());

// long begin=System.currentTimeMillis();

n=Integer.parseInt(tok.nextToken());

m=Integer.parseInt(tok.nextToken());

ArrayList<Integer>[] way=new ArrayList[n];

int count=0;

int i;

for(i=0;i<n;i++) way[i]=new ArrayList<Integer>();

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

{

tok=new StringTokenizer(bfr.readLine());

int x=Integer.parseInt(tok.nextToken())-1, y=Integer.parseInt(tok.nextToken())-1;

way[x].add(y); way[y].add(x);

}

tok=new StringTokenizer(bfr.readLine());

int g1=Integer.parseInt(tok.nextToken())-1, g2=Integer.parseInt(tok.nextToken())-1;

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

{

if(i==g1 || i==g2) continue;

boolean[] flag=new boolean[n];

int[] que=new int [n];

int front=0,rear=1;

que[0]=g1;

while(front<rear)

{

for(int p=0;p<way[que[front]].size();p++)

{

if(way[que[front]].get(p)==i) continue;

if (!flag[way[que[front]].get(p)])

{

que[rear] = way[que[front]].get(p);

flag[que[rear]]=true;

if(que[rear]==g2) break;

rear++;

}

}

if(flag[g2]) { count++; break; }

front++;

}

}

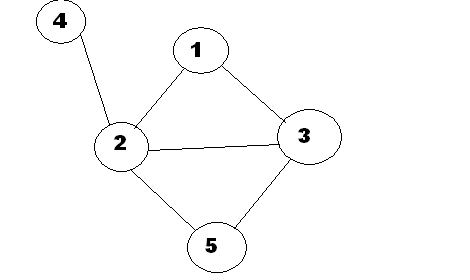
System.out.println(n-2-count);

// System.out.println(System.currentTimeMillis()-begin);

}

}

### 网络寻路



问题描述

X 国的一个网络使用若干条线路连接若干个节点。节点间的通信是双向的。某重要数据包，为了安全起见，必须恰好被转发两次到达目的地。该包可能在任意一个节点产生，我们需要知道该网络中一共有多少种不同的转发路径。

源地址和目标地址可以相同，但中间节点必须不同。

如下图所示的网络。

1 -> 2 -> 3 -> 1 是允许的

1 -> 2 -> 1 -> 2 或者 1 -> 2 -> 3 -> 2 都是非法的。

输入格式

输入数据的第一行为两个整数N M，分别表示节点个数和连接线路的条数(1<=N<=10000; 0<=M<=100000)。

接下去有M行，每行为两个整数 u 和 v，表示节点u 和 v 联通(1<=u,v<=N , u!=v)。

输入数据保证任意两点最多只有一条边连接，并且没有自己连自己的边，即不存在重边和自环。

输出格式

输出一个整数，表示满足要求的路径条数。

样例输入1

3 3  
1 2  
2 3  
1 3

样例输出1

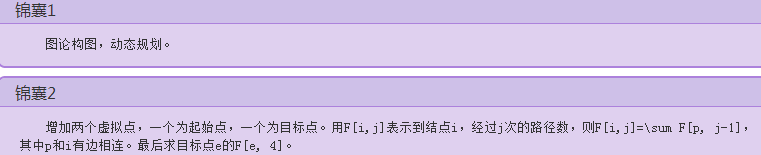
6

样例输入2

4 4  
1 2  
2 3  
3 1  
1 4

样例输出2

10



import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.ArrayList;

import java.util.StringTokenizer;

public class Main {

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

BufferedReader bfr = new BufferedReader(

new InputStreamReader(System.in));

StringTokenizer tok = new StringTokenizer(bfr.readLine());

int n = Integer.parseInt(tok.nextToken());

int m = Integer.parseInt(tok.nextToken());

int i, count = 0;

ArrayList<Integer>[] way = new ArrayList[n];

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

way[i] = new ArrayList<Integer>();

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

tok = new StringTokenizer(bfr.readLine());

int x = Integer.parseInt(tok.nextToken()) - 1;

int y = Integer.parseInt(tok.nextToken()) - 1;

way[x].add(y);

way[y].add(x);

}

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

int a = i, ll = way[i].size();

for (int ii = 0; ii < ll; ii++) {

int aa = way[a].get(ii);

int lll = way[aa].size();

for (int iii = 0; iii < lll; iii++) {

int aaa = way[aa].get(iii);

if (aaa == a)

continue;

count += way[aaa].size();

count--;

}

}

}

System.out.println(count);

}

}

### 高僧斗法

问题描述

　　古时丧葬活动中经常请高僧做法事。仪式结束后，有时会有“高僧斗法”的趣味节目，以舒缓压抑的气氛。  
　　节目大略步骤为：先用粮食（一般是稻米）在地上“画”出若干级台阶（表示N级浮屠）。又有若干小和尚随机地“站”在某个台阶上。最高一级台阶必须站人，其它任意。(如图1所示)  
　　两位参加游戏的法师分别指挥某个小和尚向上走任意多级的台阶，但会被站在高级台阶上的小和尚阻挡，不能越过。两个小和尚也不能站在同一台阶，也不能向低级台阶移动。  
　　两法师轮流发出指令，最后所有小和尚必然会都挤在高段台阶，再也不能向上移动。轮到哪个法师指挥时无法继续移动，则游戏结束，该法师认输。  
　　对于已知的台阶数和小和尚的分布位置，请你计算先发指令的法师该如何决策才能保证胜出。

输入格式

　　输入数据为一行用空格分开的N个整数，表示小和尚的位置。台阶序号从1算起，所以最后一个小和尚的位置即是台阶的总数。（N<100, 台阶总数<1000）

输出格式

　　输出为一行用空格分开的两个整数: A B, 表示把A位置的小和尚移动到B位置。若有多个解，输出A值较小的解，若无解则输出-1。

样例输入

1 5 9

样例输出

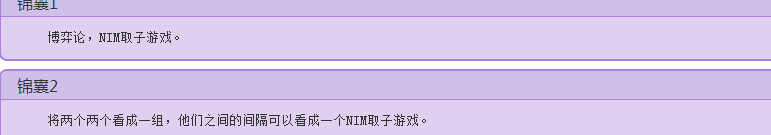
1 4

样例输入

1 5 8 10

样例输出

1 3



import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.StringTokenizer;

public class Main {

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

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

StringTokenizer tok= new StringTokenizer(bfr.readLine());

int i=0,j,flag=1;

int[] monk=new int[108];

while(tok.hasMoreTokens()) monk[i++]=Integer.parseInt(tok.nextToken());

int N=i-1;

int[] A=new int[N];

for(i=0;i<N;i++) A[i]=monk[i+1]-monk[i]-1;

int sum=A[0];

for(i=2;i<N;i++,i++) sum^=A[i];

if(sum==0) System.out.println(-1);

else

{

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

{

for(j=1;j<=A[i];j++)

{

int s=sum;

A[i]-=j;

if(i>0) A[i-1]+=j;

if(i%2==0) { s^=(A[i]+j); s^=A[i]; }

else { s^=(A[i-1]-j); s^=A[i-1]; }

if(s==0) { System.out.println(monk[i]+" "+(monk[i]+j)); flag=0; break; }

else { A[i]+=j; if(i>0) A[i-1]-=j; }

}

if(flag==0) break;

}

}

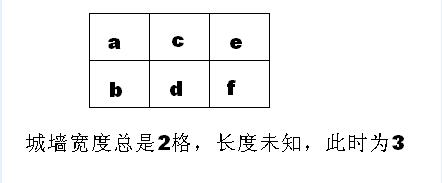
}

}

### 格子刷油漆

问题描述

　　X国的一段古城墙的顶端可以看成 2\*N个格子组成的矩形（如下图所示），现需要把这些格子刷上保护漆。



你可以从任意一个格子刷起，刷完一格，可以移动到和它相邻的格子（对角相邻也算数），但不能移动到较远的格子（因为油漆未干不能踩！）  
　　比如：a d b c e f 就是合格的刷漆顺序。  
　　c e f d a b 是另一种合适的方案。  
　　当已知 N 时，求总的方案数。当N较大时，结果会迅速增大，请把结果对 1000000007 (十亿零七) 取模。

输入格式

　　输入数据为一个正整数（不大于1000）

输出格式

　　输出数据为一个正整数。

样例输入

2

样例输出

24

样例输入

3

样例输出

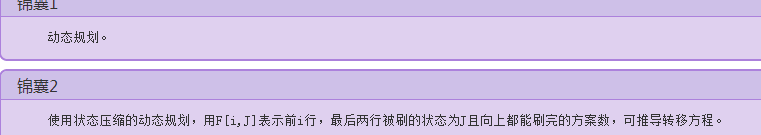
96

样例输入

22

样例输出

359635897



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

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

long sum=0,mod=1000000007;

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

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

A[1]=1; A[2]=2;

B[1]=1; B[2]=6;

int i,j;

for(i=3;i<=n;i++)

{

A[i]=2\*A[i-1]%mod;

B[i]=(2\*B[i-1]%mod+2\*A[i-2]%mod+B[i-2]\*2%mod+2\*A[i-2]%mod+2\*B[i-2]%mod)%mod;

}

sum=4\*B[n]%mod;

for(j=2;j<n;j++)

{

sum+=(2\*(A[j-1]\*2\*B[n-j]\*2%mod+2\*A[n-j]\*2\*B[j-1]%mod));

sum%=mod;

}

if(n>1) System.out.println(sum);

else System.out.println(2);

}

}

### 农场阳光

问题描述

　　X星球十分特殊，它的自转速度与公转速度相同，所以阳光总是以固定的角度照射。  
　　最近，X星球为发展星际旅游业，把空间位置出租给Y国游客来晒太阳。每个租位是漂浮在空中的圆盘形彩云（圆盘与地面平行）。当然，这会遮挡住部分阳光，被遮挡的土地植物无法生长。  
　　本题的任务是计算某个农场宜于作物生长的土地面积有多大。

输入格式

　　输入数据的第一行包含两个整数a, b，表示某农场的长和宽分别是a和b，此时，该农场的范围是由坐标(0, 0, 0), (a, 0, 0), (a, b, 0), (0, b, 0)围成的矩形区域。  
　　第二行包含一个实数g，表示阳光照射的角度。简单起见，我们假设阳光光线是垂直于农场的宽的，此时正好和农场的长的夹角是g度，此时，空间中的一点(x, y, z)在地面的投影点应该是(x + z \* ctg(g度), y, 0)，其中ctg(g度)表示g度对应的余切值。  
　　第三行包含一个非负整数n，表示空中租位个数。  
　　接下来 n 行，描述每个租位。其中第i行包含4个整数xi, yi, zi, ri，表示第i个租位彩云的圆心在(xi, yi, zi)位置，圆半径为ri。

输出格式

　　要求输出一个实数，四舍五入保留两位有效数字，表示农场里能长庄稼的土地的面积。

样例输入

10 10  
90.0  
1  
5 5 10 5

样例输出

21.46

样例输入

8 8  
90.0  
1  
4 4 10 5

样例输出

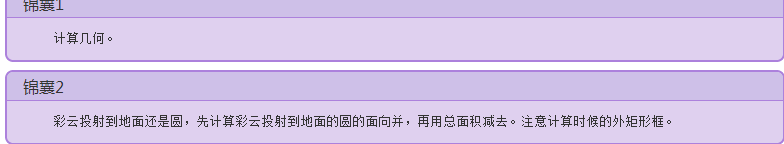
1.81

样例输入

20 10  
45.0  
2  
5 0 5 5  
8 6 14 6

样例输出

130.15



### 约数倍数选卡片

问题描述

　　闲暇时，福尔摩斯和华生玩一个游戏：  
　　在N张卡片上写有N个整数。两人轮流拿走一张卡片。要求下一个人拿的数字一定是前一个人拿的数字的约数或倍数。例如，某次福尔摩斯拿走的卡片上写着数字“6”，则接下来华生可以拿的数字包括：  
　　1，2，3, 6，12，18，24 ....  
　　当轮到某一方拿卡片时，没有满足要求的卡片可选，则该方为输方。  
　　请你利用计算机的优势计算一下，在已知所有卡片上的数字和可选哪些数字的条件下，怎样选择才能保证必胜！  
　　当选多个数字都可以必胜时，输出其中最小的数字。如果无论如何都会输，则输出-1。

输入格式

　　输入数据为2行。第一行是若干空格分开的整数（每个整数介于1~100间），表示当前剩余的所有卡片。  
　　第二行也是若干空格分开的整数，表示可以选的数字。当然，第二行的数字必须完全包含在第一行的数字中。

输出格式

　　程序则输出必胜的招法！！

样例输入

2 3 6  
3 6

样例输出

3

样例输入

1 2 2 3 3 4 5  
3 4 5

样例输出

4

import java.util.Arrays;

import java.util.Scanner;

public class Main{

static int[] cnt=new int[101];

static int[] end=new int[101];

public static boolean f(int[][] table,int x){

for(int i=end[x];i>=0;i--) {

int j=table[x][i];

if(cnt[j]>0){

cnt[j]--;

if(f(table,j)){

cnt[j]++;

return false;

}

cnt[j]++;

}

}

return true;

}

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

String[] s1=sc.nextLine().split(" ");

String[] s2=sc.nextLine().split(" ");

int[] m=new int[s2.length];

int[][] table=new int[101][100];

for (int i = 0; i < s1.length; i++) {

int x=Integer.parseInt(s1[i]);

cnt[x]++;

}

for (int i = 1; i < 101; i++) {

if(cnt[i]>0){

int t=0;

for (int j = 1; j <=100; j++) {

if(cnt[j]>0 && (i%j==0||j%i==0))

{

table[i][t]=j;

t++;

}

}

end[i]=t-1;

}

}

for (int i = 0; i < s2.length; i++) {

m[i]=Integer.parseInt(s2[i]);

}

Arrays.sort(m);

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

cnt[m[i]]--;

if(f(table,m[i])){

System.out.println(m[i]);

return;

}

cnt[m[i]]++;

}

System.out.println(-1);

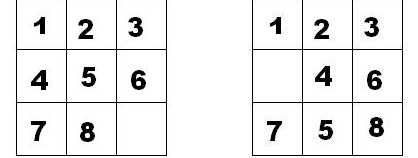
}

}

### 九宫重排

问题描述

　　如下面第一个图的九宫格中，放着 1~8 的数字卡片，还有一个格子空着。与空格子相邻的格子中的卡片可以移动到空格中。经过若干次移动，可以形成第二个图所示的局面。



我们把第一个图的局面记为：12345678.  
　　把第二个图的局面记为：123.46758  
　　显然是按从上到下，从左到右的顺序记录数字，空格记为句点。  
　　本题目的任务是已知九宫的初态和终态，求最少经过多少步的移动可以到达。如果无论多少步都无法到达，则输出-1。

输入格式

　　输入第一行包含九宫的初态，第二行包含九宫的终态。

输出格式

　　输出最少的步数，如果不存在方案，则输出-1。

样例输入

12345678.  
123.46758

样例输出

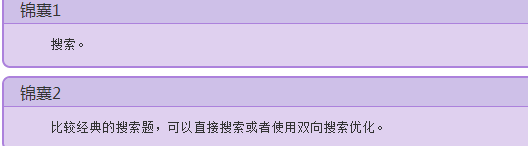
3

样例输入

13524678.  
46758123.

样例输出

22



 import java.io.\*;

import java.util.\*;

public class Main{

static Map<String,Integer> hm1=new HashMap<String,Integer>();

static Map<String,Integer> hm2=new HashMap<String,Integer>();

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

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

String start=bf.readLine();

String end=bf.readLine();

char[][] a=new char[3][3];

char[][] b=new char[3][3];

int c=0,x1=0,y1=0,x2=0,y2=0;

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

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

a[i][j]=start.charAt(c);

b[i][j]=end.charAt(c);

c++;

if(a[i][j]=='.'){

x1=i;

y1=j;

}

if(b[i][j]=='.'){

x2=i;

y2=j;

}

}

}

Node node1=new Node(0,x1,y1,a);

Node node2=new Node(0,x2,y2,b);

Queue<Node> qnode1=new LinkedList<Node>();

Queue<Node> qnode2=new LinkedList<Node>();

qnode1.add(node1);

qnode2.add(node2);

hm1.put(node1.gettu(), 0);

hm2.put(node2.gettu(), 0);

System.out.println(bfs(qnode1,qnode2));

}

public static int bfs(Queue<Node> q1,Queue<Node> q2){

while(!q1.isEmpty()||!q2.isEmpty()){

if(!q1.isEmpty()){

Node node=q1.poll();

int x=node.getX();

int y=node.getY();

if(hm2.containsKey(node.gettu())){

return node.getSum()+hm2.get(node.gettu());

}

if(x>0){

char[][] c=node.getCopy();

c[x][y]=c[x-1][y];

c[x-1][y]='.';

Node node2=new Node(node.sum+1,x-1,y,c);

String s=node2.gettu();

if(hm2.containsKey(s)){

return node2.getSum()+hm2.get(node2.gettu());

}

if(!hm1.containsKey(s)){

hm1.put(s,node2.getSum());

q1.add(node2);

}

}

if(x<2){

char[][] c=node.getCopy();

c[x][y]=c[x+1][y];

c[x+1][y]='.';

Node node2=new Node(node.sum+1,x+1,y,c);

String s=node2.gettu();

if(hm2.containsKey(s)){

return node2.getSum()+hm2.get(s);

}

if(!hm1.containsKey(s)){

hm1.put(s,node2.getSum());

q1.add(node2);

}

}

if(y>0){

char[][] c=node.getCopy();

c[x][y]=c[x][y-1];

c[x][y-1]='.';

Node node2=new Node(node.sum+1,x,y-1,c);

String s=node2.gettu();

if(hm2.containsKey(s)){

return node2.getSum()+hm2.get(s);

}

if(!hm1.containsKey(s)){

hm1.put(s,node2.getSum());

q1.add(node2);

}

}

if(y<2){

char[][] c=node.getCopy();

c[x][y]=c[x][y+1];

c[x][y+1]='.';

Node node2=new Node(node.sum+1,x,y+1,c);

String s=node2.gettu();

if(hm2.containsKey(s)){

return node2.getSum()+hm2.get(s);

}

if(!hm1.containsKey(s)){

hm1.put(s,node2.getSum());

q1.add(node2);

}

}

}

if(!q2.isEmpty()){

Node node=q2.poll();

int x=node.getX();

int y=node.getY();

if(hm1.containsKey(node.gettu())){

return node.getSum()+hm1.get(node.gettu());

}

if(x>0){

char[][] c=node.getCopy();

c[x][y]=c[x-1][y];

c[x-1][y]='.';

Node node2=new Node(node.sum+1,x-1,y,c);

String s=node2.gettu();

if(hm1.containsKey(s)){

return node2.getSum()+hm1.get(s);

}

if(!hm2.containsKey(s)){

hm2.put(s,node2.getSum());

q2.add(node2);

}

}

if(x<2){

char[][] c=node.getCopy();

c[x][y]=c[x+1][y];

c[x+1][y]='.';

Node node2=new Node(node.sum+1,x+1,y,c);

String s=node2.gettu();

if(hm1.containsKey(s)){

return node2.getSum()+hm1.get(s);

}

if(!hm2.containsKey(s)){

hm2.put(s,node2.getSum());

q2.add(node2);

}

}

if(y>0){

char[][] c=node.getCopy();

c[x][y]=c[x][y-1];

c[x][y-1]='.';

Node node2=new Node(node.sum+1,x,y-1,c);

String s=node2.gettu();

if(hm1.containsKey(s)){

return node2.getSum()+hm1.get(s);

}

if(!hm2.containsKey(s)){

hm2.put(s,node2.getSum());

q2.add(node2);

}

}

if(y<2){

char[][] c=node.getCopy();

c[x][y]=c[x][y+1];

c[x][y+1]='.';

Node node2=new Node(node.sum+1,x,y+1,c);

String s=node2.gettu();

if(hm1.containsKey(s)){

return node2.getSum()+hm1.get(s);

}

if(!hm2.containsKey(s)){

hm2.put(s,node2.getSum());

q2.add(node2);

}

}

}

}

return -1;

}

}

class Node{

int sum,x,y;

char[][] c=null;

public char[][] getCopy(){

char[][] copy=new char[3][3];

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

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

copy[i][j]=c[i][j];

}

}

return copy;

}

public String gettu(){

StringBuffer s=new StringBuffer();

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

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

s.append(c[i][j]);

}

}

return s.toString();

}

public Node(int sum, int x, int y, char[][] c) {

super();

this.sum = sum;

this.x = x;

this.y = y;

this.c = c;

}

public int getSum() {

return sum;

}

public void setSum(int sum) {

this.sum = sum;

}

public int getX() {

return x;

}

public void setX(int x) {

this.x = x;

}

public int getY() {

return y;

}

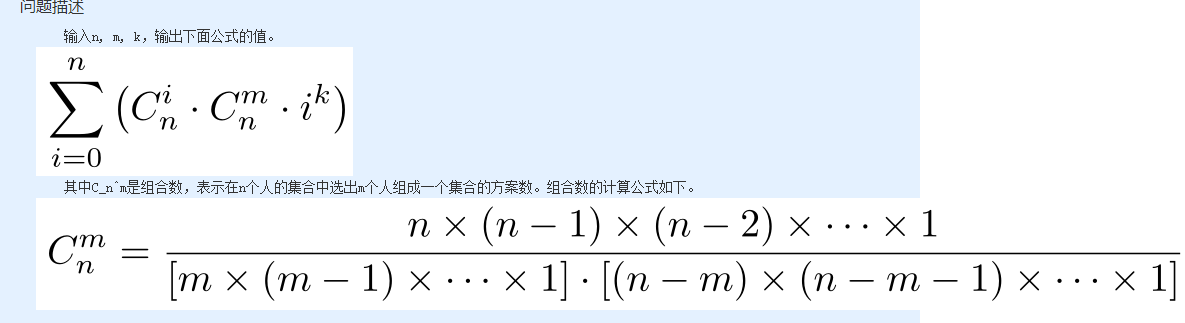
public void setY(int y) {

this.y = y;

}

}

### 公式求值



输入格式

　　输入的第一行包含一个整数n；第二行包含一个整数m，第三行包含一个整数k。

输出格式

　　计算上面公式的值，由于答案非常大，请输出这个值除以999101的余数。

样例输入

3  
1  
3

样例输出

162

样例输入

20  
10  
10

样例输出

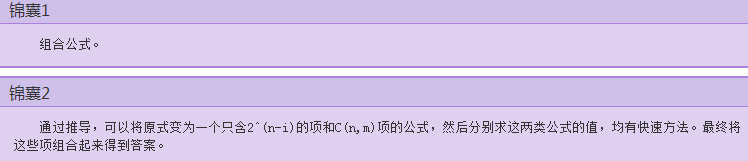
359316

数据规模和约定

　　对于10%的数据，n≤10，k≤3；  
　　对于20%的数据，n≤20，k≤3；  
　　对于30%的数据，n≤1000，k≤5；  
　　对于40%的数据，n≤10^7，k≤10；  
　　对于60%的数据，n≤10^15，k ≤100；  
　　对于70%的数据，n≤10^100，k≤200；  
　　对于80%的数据，n≤10^500，k ≤500；  
　　对于100%的数据，n在十进制下不超过1000位，即1≤n<10^1000，1≤k≤1000，同时0≤m≤n，k≤n。

提示

　　999101是一个质数；  
　　当n位数比较多时，绝大多数情况下答案都是0，但评测的时候会选取一些答案不是0的数据；



import java.math.\*;

import java.util.\*;

public class Main {

final long mod = 999101l;

final int maxk = 1005;

long[][]dp = new long[maxk][maxk];

long[] fac = new long[ (int) mod];

BigInteger n,m,Mod = BigInteger.valueOf(mod);

int k;

long ans;

Main()

{

Scanner jin = new Scanner(System.in);

n = jin.nextBigInteger();

m = jin.nextBigInteger();

k = jin.nextInt();

if(n.equals(new BigInteger("7349813")) && m.equals(new BigInteger("3590741")) && k == 9)

{

System.out.println(591101);

return;

}

getfac();

long lc = lucas(n,m);

if(lc == 0l)

{

System.out.println(0);

return;

}

getdp();

ans = 0l;

int i;

long p = qpow(2l,n.subtract(BigInteger.valueOf(k)));

for(i=k;i>=0;i--,p=(p+p)%mod)

ans = (ans + dp[k][i] \* p % mod) % mod;

ans = ans \* lc % mod;

System.out.println(ans);

}

void getdp()

{

int i,j;

dp[0][0] = 1l;

long N = n.mod(Mod).longValue();

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

for(j=0;j<k;j++)

{

dp[i+1][j] += (long)j \* dp[i][j] % mod;

dp[i+1][j+1] += (N + mod - (long)j) % mod \* dp[i][j] % mod;

}

}

long qpow(long a,BigInteger b)

{

long ans;

for(ans=1l;!b.equals(BigInteger.ZERO);b=b.shiftRight(1),a=a\*a%mod)

if(b.and(BigInteger.ONE).equals(BigInteger.ONE))

ans = ans \* a % mod;

return ans;

}

long qpow(long a,long b)

{

long ans;

for(ans=1l;b>0l;b>>=1l,a=a\*a%mod)

if((b&1l) == 1l)

ans = ans \* a % mod;

return ans;

}

void getfac()

{

int i;

fac[0] = 1l;

for(i=1;i<mod;i++)

fac[i] = fac[i - 1] \* (long)i % mod;

}

long lucas(BigInteger n,BigInteger m)

{

long ret = 1l;

while(!n.equals(BigInteger.ZERO) && !m.equals(BigInteger.ZERO))

{

int a = n.mod(Mod).intValue(),b = m.mod(Mod).intValue();

if(a < b)return 0l;

ret = ret \* fac[a] % mod \* qpow(fac[b] \* fac[a - b] % mod,mod - 2l) % mod;

n = n.divide(Mod);

m = m.divide(Mod);

}

return ret;

}

public static void main(String[] args) {

// TODO Auto-generated method stub

new Main();

}

}

### 回文数字

问题描述

　　观察数字：12321，123321 都有一个共同的特征，无论从左到右读还是从右向左读，都是相同的。这样的数字叫做：回文数字。  
  
　　本题要求你找到一些5位或6位的十进制数字。满足如下要求：  
　　该数字的各个数位之和等于输入的整数。

输入格式

　　一个正整数 n (10<n<100), 表示要求满足的数位和。

输出格式

　　若干行，每行包含一个满足要求的5位或6位整数。  
　　数字按从小到大的顺序排列。  
　　如果没有满足条件的，输出：-1

样例输入

44

样例输出

99899  
499994  
589985  
598895  
679976  
688886  
697796  
769967  
778877  
787787  
796697  
859958  
868868  
877778  
886688  
895598  
949949  
958859  
967769  
976679  
985589  
994499

样例输入

60

样例输出

-1

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class Main

{

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

{

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

String str1 = bf.readLine();

int n = Integer.parseInt(str1);

int t = 0;

for (int i = 1; i <= 9; i++)

for (int j = 0; j <= 9; j++)

for (int k = 0; k <= 9; k++)

{

if (n == 2 \* (i + j) + k)

{

System.out.println(i \* 10000 + j \* 1000 + k \* 100 + j

\* 10 + i);

t = 1;

}

}

for (int i = 1; i <= 9; i++)

for (int j = 0; j <= 9; j++)

for (int k = 0; k <= 9; k++)

{

if (n == 2 \* (i + j + k))

{

System.out.println(i \* 100000 + j \* 10000 + k \* 1000

+ k \* 100 + j \* 10 + i);

t = 1;

}

}

if (t == 0)

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

}

}

### 国王的烦恼

问题描述

C国由n个小岛组成，为了方便小岛之间联络，C国在小岛间建立了m座大桥，每座大桥连接两座小岛。两个小岛间可能存在多座桥连接。然而，由于海水冲刷，有一些大桥面临着不能使用的危险。  
  
　　如果两个小岛间的所有大桥都不能使用，则这两座小岛就不能直接到达了。然而，只要这两座小岛的居民能通过其他的桥或者其他的小岛互相到达，他们就会安然无事。但是，如果前一天两个小岛之间还有方法可以到达，后一天却不能到达了，居民们就会一起抗议。

现在C国的国王已经知道了每座桥能使用的天数，超过这个天数就不能使用了。现在他想知道居民们会有多少天进行抗议。

输入格式

　　输入的第一行包含两个整数n, m，分别表示小岛的个数和桥的数量。  
　　接下来m行，每行三个整数a, b, t，分别表示该座桥连接a号和b号两个小岛，能使用t天。小岛的编号从1开始递增。

输出格式

　　输出一个整数，表示居民们会抗议的天数。

样例输入

4 4  
1 2 2  
1 3 2  
2 3 1  
3 4 3

样例输出

2

样例说明

　　第一天后2和3之间的桥不能使用，不影响。  
　　第二天后1和2之间，以及1和3之间的桥不能使用，居民们会抗议。  
　　第三天后3和4之间的桥不能使用，居民们会抗议。

数据规模和约定

　　对于30%的数据，1<=n<=20，1<=m<=100；  
　　对于50%的数据，1<=n<=500，1<=m<=10000；  
　　对于100%的数据，1<=n<=10000，1<=m<=100000，1<=a, b<=n， 1<=t<=100000。

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.util.\*;

public class Main {

static int n;

static int sum = 0;

static int a[];

static Edge p[];

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

BufferedReader buf = new BufferedReader(

new InputStreamReader(System.in));

String read[] = buf.readLine().split("\\s+");

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

int m = Integer.parseInt(read[1]);

p = new Edge[m];

a = new int[n];

int s, e, w, t;

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

read = buf.readLine().split("\\s+");

s = Integer.parseInt(read[0]) - 1;

e = Integer.parseInt(read[1]) - 1;

w = Integer.parseInt(read[2]);

p[i] = new Edge(s, e, w);

}

java.util.Arrays.sort(p);

boolean flag = false;

init();

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

flag = false;

s = p[i].s;

e = p[i].e;

if (!isConnect(s, e)) {

flag = true;

union(s, e);

}

while (i < m - 1 && p[i].w == p[i + 1].w) {

s = p[i + 1].s;

e = p[i + 1].e;

if (flag && !isConnect(s, e))

union(s, e);

if (!flag && !isConnect(s, e)) {

flag = true;

union(s, e);

}

i++;

}

if (flag)

sum++;

}

System.out.println(sum);

}

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

if (find(i) == find(j))

return true;

return false;

}

private static void init() {

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

a[i] = i;

}

private static void union(int x, int y) {

a[find(x)] = find(y);

}

private static int find(int x) {

if (a[x] == x)

return x;

a[x] = find(a[x]);

return a[x];

}

}

class Edge implements Comparable<Edge> {

int s;

int e;

int w;

public Edge(int s, int e, int w) {

super();

this.s = s;

this.e = e;

this.w = w;

}

@Override

public int compareTo(Edge p) {

if (this.w < p.w)

return 1;

else if (this.w > p.w)

return -1;

return 0;

}

}

### 数字游戏

问题描述

　　栋栋正在和同学们玩一个数字游戏。  
  
　　游戏的规则是这样的：栋栋和同学们一共n个人围坐在一圈。栋栋首先说出数字1。接下来，坐在栋栋左手边的同学要说下一个数字2。再下面的一个同学要从上一个同学说的数字往下数两个数说出来，也就是说4。下一个同学要往下数三个数，说7。依次类推。  
  
　　为了使数字不至于太大，栋栋和同学们约定，当在心中数到 k-1 时，下一个数字从0开始数。例如，当k=13时，栋栋和同学们报出的前几个数依次为：  
　　1, 2, 4, 7, 11, 3, 9, 3, 11, 7。  
  
　　游戏进行了一会儿，栋栋想知道，到目前为止，他所有说出的数字的总和是多少。

输入格式

　　输入的第一行包含三个整数 n,k,T，其中 n 和 k 的意义如上面所述，T 表示到目前为止栋栋一共说出的数字个数。

输出格式

　　输出一行，包含一个整数，表示栋栋说出所有数的和。

样例输入

3 13 3

样例输出

17

样例说明

　　栋栋说出的数依次为1, 7, 9，和为17。

数据规模和约定

　　1 < n,k,T < 1,000,000；

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner scan = new Scanner(System.in);

long n = scan.nextInt();

long k = scan.nextInt();

long t = scan.nextInt();

long sum = 1;

long now = 1;

long v1 = (1+n)\*n/2;

long v2 = n\*n;

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

now = (v1+i\*v2+now)%k;

sum += now;

}

System.out.println(sum);

scan.close();

}

}

### 邮局

问题描述

　　C村住着n户村民，由于交通闭塞，C村的村民只能通过信件与外界交流。为了方便村民们发信，C村打算在C村建设k个邮局，这样每户村民可以去离自己家最近的邮局发信。  
　　现在给出了m个备选的邮局，请从中选出k个来，使得村民到自己家最近的邮局的距离和最小。其中两点之间的距离定义为两点之间的直线距离。

输入格式

　　输入的第一行包含三个整数n, m, k，分别表示村民的户数、备选的邮局数和要建的邮局数。  
　　接下来n行，每行两个整数x, y，依次表示每户村民家的坐标。  
　　接下来m行，每行包含两个整数x, y，依次表示每个备选邮局的坐标。  
　　在输入中，村民和村民、村民和邮局、邮局和邮局的坐标可能相同，但你应把它们看成不同的村民或邮局。

输出格式

　　输出一行，包含k个整数，从小到大依次表示你选择的备选邮局编号。（备选邮局按输入顺序由1到m编号）

样例输入

5 4 2  
0 0  
2 0  
3 1  
3 3  
1 1  
0 1  
1 0  
2 1  
3 2

样例输出

2 4

数据规模和约定

　　对于30%的数据，1<=n<=10，1<=m<=10，1<=k<=5；  
　　对于60%的数据，1<=m<=20；  
　　对于100%的数据，1<=n<=50，1<=m<=25，1<=k<=10。

### 城市建设

问题描述

　　栋栋居住在一个繁华的C市中，然而，这个城市的道路大都年久失修。市长准备重新修一些路以方便市民，于是找到了栋栋，希望栋栋能帮助他。  
  
　　C市中有n个比较重要的地点，市长希望这些地点重点被考虑。现在可以修一些道路来连接其中的一些地点，每条道路可以连接其中的两个地点。另外由于C市有一条河从中穿过，也可以在其中的一些地点建设码头，所有建了码头的地点可以通过河道连接。  
  
　　栋栋拿到了允许建设的道路的信息，包括每条可以建设的道路的花费，以及哪些地点可以建设码头和建设码头的花费。  
  
　　市长希望栋栋给出一个方案，使得任意两个地点能只通过新修的路或者河道互达，同时花费尽量小。

输入格式

　　输入的第一行包含两个整数n, m，分别表示C市中重要地点的个数和可以建设的道路条数。所有地点从1到n依次编号。  
　　接下来m行，每行三个整数a, b, c，表示可以建设一条从地点a到地点b的道路，花费为c。若c为正，表示建设是花钱的，如果c为负，则表示建设了道路后还可以赚钱（比如建设收费道路）。  
　　接下来一行，包含n个整数w\_1, w\_2, …, w\_n。如果w\_i为正数，则表示在地点i建设码头的花费，如果w\_i为-1，则表示地点i无法建设码头。  
　　输入保证至少存在一个方法使得任意两个地点能只通过新修的路或者河道互达。

输出格式

　　输出一行，包含一个整数，表示使得所有地点通过新修道路或者码头连接的最小花费。如果满足条件的情况下还能赚钱，那么你应该输出一个负数。

样例输入

5 5  
1 2 4  
1 3 -1  
2 3 3  
2 4 5  
4 5 10  
-1 10 10 1 1

样例输出

9

样例说明

　　建设第2、3、4条道路，在地点4、5建设码头，总的花费为9。

数据规模和约定

　　对于20%的数据，1<=n<=10，1<=m<=20，0<=c<=20，w\_i<=20；  
　　对于50%的数据，1<=n<=100，1<=m<=1000，-50<=c<=50，w\_i<=50；  
　　对于70%的数据，1<=n<=1000；  
　　对于100%的数据，1 <= n <= 10000，1 <= m <= 100000，-1000<=c<=1000，-1<=w\_i<=1000，w\_i≠0。

import java.util.\*;

public class Main {

public static void main(String []arge){

Scanner sc = new Scanner(System.in);

int n=Integer.parseInt(sc.next()),m=Integer.parseInt(sc.next());

City city = new City(n,m);

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

city.edge[i] = new Edge(Integer.parseInt(sc.next()),Integer.parseInt(sc.next()),Integer.parseInt(sc.next()));

}

int wharf = 0;

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

int num = Integer.parseInt(sc.next());

if(num!=-1){

wharf++;

city.edge[m+wharf] = new Edge(0,i,num);

}

}

city.wharf = wharf;

city.tree();

sc.close();

}

}

class Edge{

int start,destination,value;

Edge(int start,int destination,int value){

this.start = start;

this.destination = destination;

this.value = value;

}

}

class City{

Edge[] edge;

int n,m,wharf;

int[] father;

City(int n,int m){

this.n = n;

this.m = m;

wharf = 0;

edge = new Edge[m+n+1];

father = new int[n+1];

}

void tree(){

init();

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

int a = unionsearch(edge[i].start);

int b = unionsearch(edge[i].destination);

if(a!=b){

father[a] = b;

}

}

int i;

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

if(unionsearch(1)!=unionsearch(i)){

break;

}

}

init();

QuickSort(1,m+wharf+1);

if(i==n+1){

int sum1 = kruskal(false);

init();

int sum2 = kruskal(true);

System.out.println(sum1<sum2 ? sum1:sum2);

}

else{

System.out.println(kruskal(true));

}

}

int kruskal(boolean flag){

int sum = 0;

int num = m+wharf;

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

if(edge[i].start==0 && flag==false) continue;

int a = unionsearch(edge[i].start);

int b = unionsearch(edge[i].destination);

if(a!=b || edge[i].value<0){

sum += edge[i].value;

father[a] = b;

}

}

return sum;

}

void init(){

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

father[i] = i;

}

}

int unionsearch(int x){

if(x == father[x]) return x;

father[x] = unionsearch(father[x]);

return father[x];

}

void QuickSort(int left,int right){

if(left<right){

int middle = Partition(left,right);

QuickSort(left,middle);

QuickSort(middle+1,right);

}

}

int Partition(int left,int right){

Edge key = edge[left];

int i = left;

for(int j=left+1;j<right;j++){

if(edge[j].value < key.value){

i++;

Edge news = edge[j];

edge[j] = edge[i];

edge[i] = news;

}

}

Edge news = edge[i];

edge[i] = edge[left];

edge[left] = news;

return i;

}

}

### 最大子阵

问题描述

　　给定一个n\*m的矩阵A，求A中的一个非空子矩阵，使这个子矩阵中的元素和最大。  
  
　　其中，A的子矩阵指在A中行和列均连续的一块。

输入格式

　　输入的第一行包含两个整数n, m，分别表示矩阵A的行数和列数。  
　　接下来n行，每行m个整数，表示矩阵A。

输出格式

　　输出一行，包含一个整数，表示A中最大的子矩阵中的元素和。

样例输入

3 3  
-1 -4 3  
3 4 -1  
-5 -2 8

样例输出

10

样例说明

　　取最后一列，和为10。

数据规模和约定

　　对于50%的数据，1<=n, m<=50；  
　　对于100%的数据，1<=n, m<=500，A中每个元素的绝对值不超过5000。

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class Main {

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

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

String[] s = sc.readLine().split(" ");

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

int m = Integer.parseInt(s[1]);

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

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

s = sc.readLine().split(" ");

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

f[i][j] = Integer.parseInt(s[j]);

}

}

System.out.print(cc(f));

}

static int csum(int[] s) {

int b = 0;

int max = Integer.MIN\_VALUE;

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

if (b > 0)

b += s[i];

else

b = s[i];

if (b > max)

max = b;

}

return max;

}

static int cc(int[][] s) {

int[] c = new int[s[0].length];

int max = Integer.MIN\_VALUE;

int n = s.length, m = s[0].length;

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

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

c[k] = 0;

}

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

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

c[k] += s[j][k];

}

int d = csum(c);

if (d > max)

max = d;

}

}

return max;

}

}

### 蚂蚁感冒

问题描述

　　长100厘米的细长直杆子上有n只蚂蚁。它们的头有的朝左，有的朝右。  
  
　　每只蚂蚁都只能沿着杆子向前爬，速度是1厘米/秒。  
  
　　当两只蚂蚁碰面时，它们会同时掉头往相反的方向爬行。  
  
　　这些蚂蚁中，有1只蚂蚁感冒了。并且在和其它蚂蚁碰面时，会把感冒传染给碰到的蚂蚁。  
  
　　请你计算，当所有蚂蚁都爬离杆子时，有多少只蚂蚁患上了感冒。

输入格式

　　第一行输入一个整数n (1 < n < 50), 表示蚂蚁的总数。  
  
　　接着的一行是n个用空格分开的整数 Xi (-100 < Xi < 100), Xi的绝对值，表示蚂蚁离开杆子左边端点的距离。正值表示头朝右，负值表示头朝左，数据中不会出现0值，也不会出现两只蚂蚁占用同一位置。其中，第一个数据代表的蚂蚁感冒了。

输出格式

　　要求输出1个整数，表示最后感冒蚂蚁的数目。

样例输入

3  
5 -2 8

样例输出

1

样例输入

5  
-10 8 -20 12 25

样例输出

3

import java.util.\*;

public class Main{

public static void main(String args[]){

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

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

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

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

a[i]=sc.nextInt();

if(a[i]<0)

b[i]=0;

else {

b[i]=1;

}

}

int k=a[1]>0?1:0;

int x=0,y=0,q=0,w=0;

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

if(Math.abs(a[1])<Math.abs(a[i])&&a[1]>0&&b[i]==0)

x++;

if(Math.abs(a[1])>Math.abs(a[i])&&a[1]<0&&b[i]==1)

y++;

if(Math.abs(a[1])>Math.abs(a[i])&&a[1]>0&&b[i]==1)

q++;

if(Math.abs(a[1])<Math.abs(a[i])&&a[1]<0&&b[i]==0)

w++;

}

if(k==1)

if(x!=0)

System.out.println(x+q+1);

else

System.out.println(1);

else {

if(y!=0)

System.out.println(y+w+1);

else {

System.out.println(1);

}

}

}

}

### 地宫取宝

问题描述

　　X 国王有一个地宫宝库。是 n x m 个格子的矩阵。每个格子放一件宝贝。每个宝贝贴着价值标签。  
  
　　地宫的入口在左上角，出口在右下角。  
  
　　小明被带到地宫的入口，国王要求他只能向右或向下行走。  
  
　　走过某个格子时，如果那个格子中的宝贝价值比小明手中任意宝贝价值都大，小明就可以拿起它（当然，也可以不拿）。  
  
　　当小明走到出口时，如果他手中的宝贝恰好是k件，则这些宝贝就可以送给小明。  
  
　　请你帮小明算一算，在给定的局面下，他有多少种不同的行动方案能获得这k件宝贝。

输入格式

　　输入一行3个整数，用空格分开：n m k (1<=n,m<=50, 1<=k<=12)  
  
　　接下来有 n 行数据，每行有 m 个整数 Ci (0<=Ci<=12)代表这个格子上的宝物的价值

输出格式

　　要求输出一个整数，表示正好取k个宝贝的行动方案数。该数字可能很大，输出它对 1000000007 取模的结果。

样例输入

2 2 2  
1 2  
2 1

样例输出

2

样例输入

2 3 2  
1 2 3  
2 1 5

样例输出

14

import java.io.InputStreamReader;

import java.io.OutputStreamWriter;

import java.io.PrintWriter;

import java.io.StreamTokenizer;

import java.util.Arrays;

public class Main

{

private static StreamTokenizer tokenizer = new StreamTokenizer(

new InputStreamReader(System.in));

private static PrintWriter outWriter = new PrintWriter(

new OutputStreamWriter(System.out));

private static int n, m, k;

private static int[][] table;

private static final int MOD = 1000000007;

private static long[][][][] state;

private static long dfs(int i, int j, int num, int max)

{

if (state[i][j][num][max] != -1)

return state[i][j][num][max];

long currentAns = 0;

if (i == n - 1 && j == m - 1)

{

if (num == k || max < table[i][j] && num + 1 == k)

currentAns++;

state[i][j][num][max] = currentAns;

return currentAns;

}

if (i + 1 < n)

{

currentAns += dfs(i + 1, j, num, max);

if (max < table[i][j] && num + 1 <= k)

currentAns += dfs(i + 1, j, num + 1, table[i][j]);

}

if (j + 1 < m)

{

currentAns += dfs(i, j + 1, num, max);

if (max < table[i][j] && num + 1 <= k)

currentAns += dfs(i, j + 1, num + 1, table[i][j]);

}

state[i][j][num][max] = currentAns;

return currentAns;

}

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

{

tokenizer.nextToken();

n = (int) tokenizer.nval;

tokenizer.nextToken();

m = (int) tokenizer.nval;

tokenizer.nextToken();

k = (int) tokenizer.nval;

table = new int[n][m];

state = new long[n][m][k + 1][14];

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

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

for (int t = 0; t <= k; t++)

Arrays.fill(state[i][j][t], -1);

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

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

{

tokenizer.nextToken();

table[i][j] = (int) tokenizer.nval;

table[i][j]++;

}

long ret = dfs(0, 0, 0, 0);

outWriter.println(ret % MOD);

outWriter.flush();

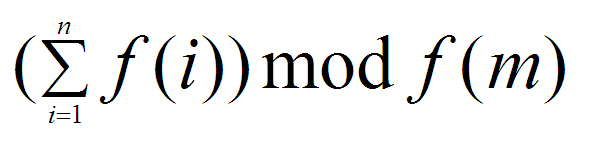
}

}

### 斐波那契

问题描述

　　斐波那契数列大家都非常熟悉。它的定义是：  
  
　　f(x) = 1 .... (x=1,2)  
　　f(x) = f(x-1) + f(x-2) .... (x>2)  
  
　　对于给定的整数 n 和 m，我们希望求出：  
　　f(1) + f(2) + ... + f(n) 的值。但这个值可能非常大，所以我们把它对 f(m) 取模。  
　　公式如下  
  
　　但这个数字依然很大，所以需要再对 p 求模。



输入格式

　　输入为一行用空格分开的整数 n m p (0 < n, m, p < 10^18)

输出格式

　　输出为1个整数，表示答案

样例输入

2 3 5

样例输出

0

样例输入

15 11 29

样例输出

25

import java.math.BigInteger;

import java.util.Scanner;

public class Main{

public static void main(String[] args) {

// TODO Auto-generated method stub

Scanner sc=new Scanner(System.in);

long n,m;

n=sc.nextLong();

m=sc.nextLong();

BigInteger p=sc.nextBigInteger(),fn,fm;

if(n+2>m)

{

fm=think(m,null);

fn=think(n+2,fm).subtract(new BigInteger("1"));

System.out.println(fn.remainder(fm).remainder(p));

}

else

{

fn=think(n+2,p).subtract(new BigInteger("1"));

System.out.println(fn.remainder(p));

}

}

private static BigInteger think(long m,BigInteger mod) {

// TODO Auto-generated method stub

BigInteger a1=new BigInteger("1"),a2=new BigInteger("1"),x[][];

if(m==1)return a1;

else if(m==2)return a2;

else

{

x=new BigInteger[2][2];

x[0][0]=new BigInteger("1");

x[0][1]=new BigInteger("1");

x[1][0]=new BigInteger("1");

x[1][1]=new BigInteger("0");

x=doublex(x,m-2,mod);

return x[0][0].add(x[0][1]);

}

}

private static BigInteger[][] doublex(BigInteger[][] x, long n,BigInteger mod) {

// TODO Auto-generated method stub

BigInteger x2[][];

x2=new BigInteger[2][2];

if(n==1)return x;

else

{

if(n%2==1)return cheng(doublex(cheng(x,x,mod),n/2,mod),x,mod);

else return doublex(cheng(x,x,mod),n/2,mod);

}

}

private static BigInteger[][] cheng(BigInteger[][] x, BigInteger[][] y,BigInteger mod) {

// TODO Auto-generated method stub

BigInteger z[][];

z=new BigInteger[2][2];

if(mod!=null)

{

z[0][0]=x[0][0].multiply(y[0][0]).add(x[1][0].multiply(y[0][1])).remainder(mod);

z[0][1]=x[0][0].multiply(y[0][1]).add(x[0][1].multiply(y[1][1])).remainder(mod);

z[1][0]=x[1][0].multiply(y[0][0]).add(x[1][1].multiply(y[1][0])).remainder(mod);

z[1][1]=x[1][0].multiply(y[0][1]).add(x[1][1].multiply(y[1][1])).remainder(mod);

return z;

}

z[0][0]=x[0][0].multiply(y[0][0]).add(x[1][0].multiply(y[0][1]));

z[0][1]=x[0][0].multiply(y[0][1]).add(x[0][1].multiply(y[1][1]));

z[1][0]=x[1][0].multiply(y[0][0]).add(x[1][1].multiply(y[1][0]));

z[1][1]=x[1][0].multiply(y[0][1]).add(x[1][1].multiply(y[1][1]));

return z;

}

}

### 波动数列

问题描述

　　观察这个数列：  
　　1 3 0 2 -1 1 -2 ...  
　　这个数列中后一项总是比前一项增加2或者减少3。  
　　栋栋对这种数列很好奇，他想知道长度为 n 和为 s 而且后一项总是比前一项增加a或者减少b的整数数列可能有多少种呢？

输入格式

　　输入的第一行包含四个整数 n s a b，含义如前面说述。

输出格式

　　输出一行，包含一个整数，表示满足条件的方案数。由于这个数很大，请输出方案数除以100000007的余数。

样例输入

4 10 2 3

样例输出

2

样例说明

　　这两个数列分别是2 4 1 3和7 4 1 -2。

数据规模和约定

　　对于10%的数据，1<=n<=5，0<=s<=5，1<=a,b<=5；  
　　对于30%的数据，1<=n<=30，0<=s<=30，1<=a,b<=30；  
　　对于50%的数据，1<=n<=50，0<=s<=50，1<=a,b<=50；  
　　对于70%的数据，1<=n<=100，0<=s<=500，1<=a, b<=50；  
　　对于100%的数据，1<=n<=1000，-1,000,000,000<=s<=1,000,000,000，1<=a, b<=1,000,000。

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

int mod = 100000007;

int n, s, a, b, i, j, t;

int x[][] = new int[1001][1001];

Scanner sc = new Scanner(System.in);

n = sc.nextInt();

s = sc.nextInt();

a = sc.nextInt();

b = sc.nextInt();

b %= n;

b \*= -1;

while (b < 0)

b += n;

a %= n;

s %= n;

while (s < 0)

s += n;

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

for (j = 0; j < n; j++)

x[i][j] = 0;

x[1][a] = x[1][b] = 1;

for (i = 1; i < n - 1; i++)

for (j = 0; j < n; j++) {

t = (j + a \* (i + 1)) % n;

x[i + 1][t] += x[i][j];

x[i + 1][t] %= mod;

t = (j + b \* (i + 1)) % n;

t %= n;

x[i + 1][t] += x[i][j];

x[i + 1][t] %= mod;

}

System.out.printf("%d\n", x[n - 1][s]);

}

}

### 小朋友排队

问题描述

　　n 个小朋友站成一排。现在要把他们按身高从低到高的顺序排列，但是每次只能交换位置相邻的两个小朋友。  
  
　　每个小朋友都有一个不高兴的程度。开始的时候，所有小朋友的不高兴程度都是0。  
　　如果某个小朋友第一次被要求交换，则他的不高兴程度增加1，如果第二次要求他交换，则他的不高兴程度增加2（即不高兴程度为3），依次类推。当要求某个小朋友第k次交换时，他的不高兴程度增加k。  
　　请问，要让所有小朋友按从低到高排队，他们的不高兴程度之和最小是多少。  
  
　　如果有两个小朋友身高一样，则他们谁站在谁前面是没有关系的。

输入格式

输入的第一行包含一个整数n，表示小朋友的个数。  
　　第二行包含 n 个整数 H1 H2 … Hn，分别表示每个小朋友的身高。

输出格式

　　输出一行，包含一个整数，表示小朋友的不高兴程度和的最小值。

样例输入

3  
3 2 1

样例输出

9

样例说明

　　首先交换身高为3和2的小朋友，再交换身高为3和1的小朋友，再交换身高为2和1的小朋友，每个小朋友的不高兴程度都是3，总和为9。

数据规模和约定

　　对于10%的数据， 1<=n<=10；  
　　对于30%的数据， 1<=n<=1000；  
　　对于50%的数据， 1<=n<=10000；  
　　对于100%的数据，1<=n<=100000，0<=Hi<=1000000。

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class Main{

static int N = 100010;

static int MAX = 1000100;

static int[] C = new int[MAX];

static int[] S = new int[MAX];

static int[] b = new int[N];

static long[] total = new long[N];

static long ans;

static int[] num = new int[N];

static int T, s, t, i, j;

static int Lowbit(int x) {

return x & (-x);

}

static void add(int pos, int num, int[] P) {

while (pos <= MAX) {

P[pos] += num;

pos += Lowbit(pos);

}

}

static int Sum(int end, int[] P) {

int cnt = 0;

while (end > 0) {

cnt += P[end];

end -= Lowbit(end);

}

return cnt;

}

static void init() {

total[0] = 0;

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

total[i] = total[i - 1] + i;

}

}

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

init();

BufferedReader buf = new BufferedReader(

new InputStreamReader(System.in));

T = Integer.parseInt(buf.readLine());

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

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

num[j] = Integer.parseInt(str[j]);

add(num[j] + 1, 1, C);

b[j] = j - Sum(num[j], C);

b[j] -= Sum(num[j] + 1, C) - Sum(num[j], C) - 1;

}

ans = 0;

for (int j = T - 1; j > -1; --j) {

add(num[j] + 1, 1, S);

b[j] += Sum(num[j], S);

ans += total[b[j]];

}

System.out.println(ans);

}

}

### 分糖果

问题描述

　　有n个小朋友围坐成一圈。老师给每个小朋友随机发偶数个糖果，然后进行下面的游戏：  
　　每个小朋友都把自己的糖果分一半给左手边的孩子。  
　　一轮分糖后，拥有奇数颗糖的孩子由老师补给1个糖果，从而变成偶数。  
  
　　反复进行这个游戏，直到所有小朋友的糖果数都相同为止。  
  
　　你的任务是预测在已知的初始糖果情形下，老师一共需要补发多少个糖果。

输入格式

　　程序首先读入一个整数N(2<N<100)，表示小朋友的人数。  
　　接着是一行用空格分开的N个偶数（每个偶数不大于1000，不小于2）

输出格式

　　要求程序输出一个整数，表示老师需要补发的糖果数。

样例输入

3  
2 2 4

样例输出

4

import java.io.\*;

public class Main {

static int count2;

static int[] arr;

static boolean bl;

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

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

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

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

arr = new int[n];

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

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

func();

}

public static void func() {

if (ifSame()) {

System.out.print(count2);

return;

}

ifEven();

if (ifSame()) {

System.out.print(count2);

return;

}

int temp = arr[arr.length - 1];

for (int a = arr.length - 1; a >= 0; a--) {

if (a > 0) {

arr[a] = arr[a] / 2 + arr[a - 1] / 2;

}

if (a == 0) {

arr[0] = arr[0] / 2 + temp / 2;

}

}

func();

}

public static void ifEven() {

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

if (arr[a] % 2 == 0) {

continue;

} else {

arr[a] += 1;

count2++;

}

}

}

public static boolean ifSame() {

int temp = arr[0];

int count1 = 0;

for (int a = arr.length - 1; a > 0; a--) {

if (temp == arr[a]) {

count1++;

}

}

if (count1 == arr.length - 1) {

bl = true;

}

return bl;

}

}

### 矩阵翻硬币

小明先把硬币摆成了一个 n 行 m 列的矩阵。  
　随后，小明对每一个硬币分别进行一次 Q 操作。  
　　对第x行第y列的硬币进行 Q 操作的定义：将所有第 i\*x 行，第 j\*y 列的硬币进行翻转。  
　　其中i和j为任意使操作可行的正整数，行号和列号都是从1开始。  
　　当小明对所有硬币都进行了一次 Q 操作后，他发现了一个奇迹——所有硬币均为正面朝上。  
　　小明想知道最开始有多少枚硬币是反面朝上的。于是，他向他的好朋友小M寻求帮助。  
　　聪明的小M告诉小明，只需要对所有硬币再进行一次Q操作，即可恢复到最开始的状态。然而小明很懒，不愿意照做。于是小明希望你给出他更好的方法。帮他计算出答案。

输入格式

　　输入数据包含一行，两个正整数 n m，含义见题目描述。

输出格式

　　输出一个正整数，表示最开始有多少枚硬币是反面朝上的。

样例输入

2 3

样例输出

1

数据规模和约定

　　对于10%的数据，n、m <= 10^3；  
　　对于20%的数据，n、m <= 10^7；  
　　对于40%的数据，n、m <= 10^15；  
　　对于10%的数据，n、m <= 10^1000（10的1000次方）。

import java.math.\*;

import java.util.\*;

public class Main{

public static void main(String[] args){

Scanner sc = new Scanner(System.in);

String n = sc.next();

String m = sc.next();

BigInteger sum = new BigInteger("1");

BigInteger bn = new BigInteger(n);

BigInteger bm = new BigInteger(m);

bn = kaifang(bn,new BigInteger(bn.toString().substring((int)(n.length()/2))),1);

bm = kaifang(bm,new BigInteger(bm.toString().substring((int)(m.length()/2))),1);

sum = sum.multiply(bn).multiply(bm);

System.out.println(sum);

}

private static BigInteger kaifang(BigInteger b,BigInteger bi,int i) {

if( (bi.multiply(bi).add(b)).divide(bi).divide(new BigInteger("2")).equals(bi) ){

System.out.println(i);

return bi;

}

else{

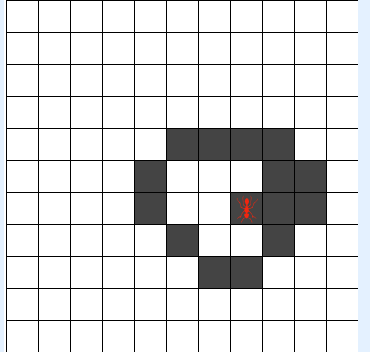
return kaifang(b,(bi.multiply(bi).add(b)).divide(bi).divide(new BigInteger("2")),i+1 );

}

}

}

### 兰顿蚂蚁



兰顿蚂蚁，是于1986年，由克里斯·兰顿提出来的，属于细胞自动机的一种。  
　　平面上的正方形格子被填上黑色或白色。在其中一格正方形内有一只“蚂蚁”。  
　　蚂蚁的头部朝向为：上下左右其中一方。  
　　蚂蚁的移动规则十分简单：  
　　若蚂蚁在黑格，右转90度，将该格改为白格，并向前移一格；  
　　若蚂蚁在白格，左转90度，将该格改为黑格，并向前移一格。  
　　规则虽然简单，蚂蚁的行为却十分复杂。刚刚开始时留下的路线都会有接近对称，像是会重复，但不论起始状态如何，蚂蚁经过漫长的混乱活动后，会开辟出一条规则的“高速公路”。  
　　蚂蚁的路线是很难事先预测的。  
　　你的任务是根据初始状态，用计算机模拟兰顿蚂蚁在第n步行走后所处的位置。

输入格式

　　输入数据的第一行是 m n 两个整数（3 < m, n < 100），表示正方形格子的行数和列数。  
　　接下来是 m 行数据。  
　　每行数据为 n 个被空格分开的数字。0 表示白格，1 表示黑格。  
　　接下来是一行数据：x y s k, 其中x y为整数，表示蚂蚁所在行号和列号（行号从上到下增长，列号从左到右增长，都是从0开始编号）。s 是一个大写字母，表示蚂蚁头的朝向，我们约定：上下左右分别用：UDLR表示。k 表示蚂蚁走的步数。

输出格式

　　输出数据为两个空格分开的整数 p q, 分别表示蚂蚁在k步后，所处格子的行号和列号。

样例输入

5 6  
0 0 0 0 0 0  
0 0 0 0 0 0  
0 0 1 0 0 0  
0 0 0 0 0 0  
0 0 0 0 0 0  
2 3 L 5

样例输出

1 3

样例输入

3 3  
0 0 0  
1 1 1  
1 1 1  
1 1 U 6

样例输出

0 0

import java.io.\*;

public class Main {

static int n, m;

static int s, e;

static char[] chs = { 'L', 'U', 'R', 'D', 'L' };

static int count = 0;

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

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

String s1[] = br.readLine().split(" ");

n = Integer.parseInt(s1[0]);

m = Integer.parseInt(s1[1]);

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

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

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

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

arr[a][b] = Integer.parseInt(str[b]);

}

}

int x, y;

String s2[] = br.readLine().split(" ");

x = Integer.parseInt(s2[0]);

y = Integer.parseInt(s2[1]);

char dec = s2[2].charAt(0);

int z;

z = Integer.parseInt(s2[3]);

s = x;

e = y;

while (count < z) {

if (arr[s][e] == 1) {

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

if (dec == chs[i]) {

dec = chs[i + 1];

break;

}

}

arr[s][e] = 0;

func(dec, s, e);

}

if (arr[s][e] == 0) {

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

if (dec == chs[j]) {

dec = chs[j - 1];

break;

}

}

arr[s][e] = 1;

func(dec, s, e);

}

}

System.out.print(s + " " + e);

}

public static void func(char dec, int x, int y) {

if (dec == 'L') {

e -= 1;

count++;

}

if (dec == 'U') {

s -= 1;

count++;

}

if (dec == 'R') {

e += 1;

count++;

}

if (dec == 'D') {

s += 1;

count++;

}

}

}