

Programming Problems Solutions

16th Annual Computer Science Programming Contest

Department of Mathematics and Computer Science
Western Carolina University
April 5, 2005

Problem 1: Encryption

A company wants to transmit data over the telephone line, but they are concerned that their lines are tapped. All of their data is transmitted as four-digit integers. They have asked you to write a program that encrypts their data so that it may be transmitted more securely. Your program should read a four-digit integer and encrypt it as follows:

Replace each digit by $((\text{digit} + 7) \bmod 10)$. Then, swap the first digit with the third, swap the second digit with the fourth, and print the encrypted integer.

An example session (input is italicized):

Enter a Four Digit Number: *6254*

The Encrypted Number is: *2139*

Solution

```
// 1992 WCU Computer Science High School Programming Contest
// Solution to Problem One, Copyright 2005, all rights reserved
// Mark Holliday
// Western Carolina University
// Department of Mathematics and Computer Science
import java.io.*;
import java.util.*;

class Encrypt
{
    BufferedReader br;

    public Encrypt()
    {
        try {
            br = new BufferedReader(new InputStreamReader(
                System.in));
        } catch (Exception e) {}
    }

    private void start()
    {
        StringTokenizer line = new StringTokenizer("");
    }
}
```

```

int[] digit = new int[4];
try {
    line = new StringTokenizer(br.readLine());
} catch (Exception e) {}
int input = Integer.parseInt(line.nextToken());
for (int i = 0; i < 4; i++)
{
    digit[3 - i] = (input % 10 + 7) % 10;
    input = input / 10;
}
int tmp = digit[0];
digit[0] = digit[2];
digit[2] = tmp;
tmp = digit[1];
digit[1] = digit[3];
digit[3] = tmp;
System.out.println(digit[0] + "" + digit[1] + "" + digit[2]
    + "" + digit[3]);
}

public static void main(String[] args)
{
    (new Encrypt()).start();
}
}

```

Problem 2: Degree of Inversion

Compute the extent to which an array of four integers is out of sorted order with sorted order meaning that the smallest index has the smallest value, the second index has the second smallest value, and so on until the largest index has the largest value. The metric for computing the degree of inversion is defined as follows. For each array index i count how many elements at larger indexes have values strictly smaller than the value of the element at index i . The degree of inversion is the sum of this count over all indexes. Notice that if the array is sorted, then the degree of inversion is 0. Your program output must look like the example output shown below. User input is shown in italics and program output is shown in boldface.

Example Program Run 1

Please enter the array values: *4 3 2 1*

The degree of inversion is: **6**

Example Program Run 2

Please enter the array values: *1 2 3 4*

The degree of inversion is: **0**

```
// 2002 WCU Computer Science High School Programming Contest
// Solution to Problem Two, Copyright 2005, all rights reserved
// Mark Holliday
// Western Carolina University
// Department of Mathematics and Computer Science
import java.io.*;
import java.util.*;

public class DegreeInversion
{
    private static final int NUM_ELTS = 4;

    public static void main(String[] args)
    {
        int i, j;
        int count;
        int[] array = new int[NUM_ELTS];
        BufferedReader br = null;

        try {
            br = new BufferedReader(new InputStreamReader(
                System.in));
        } catch (Exception e) {}

        System.out.println("Please enter the array values: ");
        StringTokenizer line = new StringTokenizer("");
        try {
            line = new StringTokenizer(br.readLine());
        } catch (Exception e) {}
        for (i = 0; i < NUM_ELTS; i++)
```

```
        array[i] = Integer.parseInt(line.nextToken());

count = 0;
for (i = 0; i < NUM_ELTS; i++)
    for (j = i; j < NUM_ELTS; j++)
        if (array[i] > array[j])
            count++;
System.out.println("The degree of inversion is: " + count);
    }
}
```

Problem 3: Pseudo-Random Numbers

Computers normally cannot generate really random numbers, but frequently are used to generate sequences of pseudo-random numbers. These are generated by some algorithm, but appear for all practical purposes to be really random. Random numbers are used in many applications, including simulation.

A common pseudo-random number generation technique is called the *linear congruential method*. If the last pseudo-random number generated was L , then the next number is generated by evaluating $(Z \times L + I) \bmod M$, where Z is a constant multiplier, I is a constant increment, and M is a constant modulus. For example, suppose Z is 7, I is 5, and M is 12. If the first random number (usually called the *seed*) is 4, then we can determine the next few pseudo-random numbers are follows:

Last Random Number, L	$(Z \times L + I)$	Next Random Number, $(Z \times L + I) \bmod M$
4	33	9
9	68	8
8	61	1
1	12	0
0	5	5
5	40	4

As you can see, the sequence of pseudo-random numbers generated by this technique repeats after six numbers. It should be clear that the longest sequence that can be generated using this technique is limited by the modulus, M .

In this problem you will be given sets of values for Z , I , M , and the seed, L . Each of these will have no more than four digits. For each such set of values you are to determine the length of the cycle of pseudo-random numbers that will be generated. But be careful -- the cycle might not begin with the seed!

Input

Each input line will contain four integer values, in order, for Z , I , M , and L . The last line will contain four zeroes, and marks the end of the input data. L will be less than M .

Output

For each input line, display the case number (they are sequentially numbered, starting with 1) and the length of the sequence of pseudo-random numbers before the sequence is repeated.

An example session:

Sample Input

```
7 5 12 4
5173 3849 3279 1511
9111 5309 6000 1234
```

```
1079 2136 9999 1237
0 0 0 0
```

Sample Output

```
Case 1: 6
Case 2: 546
Case 3: 500
Case 4: 220
```

Solution

```
// 1996 ACM North Central Region Programming Contest
// Solution to Problem Three, Copyright 2005, all rights reserved
// Mark Holliday
// Western Carolina University
// Department of Mathematics and Computer Science
import java.io.*;
import java.util.*;

class Rand
{
    private BufferedReader br;
    private int[] sequence;
    private int cycleLength;
    private int Z, I, M, L;

    public Rand()
    {
        try {
            br = new BufferedReader(new InputStreamReader(
                System.in));
            // file input would be done by
            // br = new BufferedReader(new InputStreamReader(
            //     new FileInputStream(new File
            //         ("C:\\holliday\\rand.in"))));
        } catch (Exception e) {e.printStackTrace();}
    }

    private void start()
    {
        int randCase = 1;
        while (anotherFourTuple())
        {
            System.out.println("Case " + randCase + ": " + findCycle());
            randCase++;
        }
    }

    private boolean anotherFourTuple()
    {
        String xline = "";
        StringTokenizer line = new StringTokenizer("");
        try {
            xline = br.readLine();
            line = new StringTokenizer(xline);
        }
    }
}
```

```

    } catch (Exception e) {e.printStackTrace();}
    System.out.println(xline);
    System.out.println(line.countTokens() + ", " +
        line.hasMoreTokens());
    Z = Integer.parseInt(line.nextToken());
    I = Integer.parseInt(line.nextToken());
    M = Integer.parseInt(line.nextToken());
    L = Integer.parseInt(line.nextToken());
    if ((Z == 0) && (I == 0) && (M == 0) && (L == 0))
        return false;
    sequence = new int[M];
    sequence[0] = L;
    return true;
}

private int nextNum(int current)
{
    return (Z * sequence[current] + I) % M;
}

private int findCycle()
{
    int current = 0;
    boolean found = false;
    while (true) {
        sequence[current + 1] = nextNum(current);
        for (int i = 0; i < current + 1; i++)
            if (sequence[i] == sequence[current + 1])
                return (current + 1) - i;
        current++;
    }
}

public static void main(String[] args)
{
    (new Rand()).start();
}
}

```

Problem 4: Zipper

Given three strings, you are to determine whether the third string can be formed by combining the characters in the first two strings. The first two strings can be mixed arbitrarily, but each must stay in its original order.

For example, consider forming "tcaete" from "cat" and "tree":

String A: cat
String B: tree
String C: tcaete

As you can see, we can form the third string by alternating characters from the two strings. As a second example, consider forming "catrtee" from "cat" and "tree":

String A: cat
String B: tree
String C: catrtee

Finally, notice that it is impossible to form "cttaree" from "cat" and "tree".

The Input

The first line contains a single positive integer from 1 through 1000. It represents the number of data sets to follow. The processing for each data set is identical. The data sets appear on the following lines, one data set per line.

For each data set, the line of input consists of three strings, separated by a single space. All strings are composed of upper and lower case letters only. The length of the third string is always the sum of the lengths of the first two strings. The first two strings will have lengths between 1 and 200 characters, inclusive.

The Output

For each data set, print:

Data set n: yes

if the third string can be formed from the first two, or

Data set n: no

if it cannot. Of course n should be replaced by the data set number. See the sample output below for an example.

An example session:

Sample input

3
cat tree tcræte
cat tree catrtee
cat tree cttaree

Sample output

Data set 1: yes
Data set 2: yes
Data set 3: no

Solution

```
// 2004 ACM Pacific Northwest Region Programming Contest
// Solution to Problem Four, Copyright 2005, all rights reserved
// Mark Holliday
// Western Carolina University
// Department of Mathematics and Computer Science
import java.io.*;
import java.util.*;

class Zipper
{
    BufferedReader br;
    StringTokenizer line = new StringTokenizer("");
    private String stringA;
    private String stringB;
    private String stringC;

    public Zipper()
    {
        try {
            br = new BufferedReader(new InputStreamReader(
                System.in));
            // file input would be done by
            // br = new BufferedReader(new InputStreamReader(
            //     new FileInputStream(new File
            //         ("C:\\holliday\\b.in"))));
        } catch (Exception e) {}
    }

    private void start()
    {
        try {
            line = new StringTokenizer(br.readLine());
        } catch (Exception e) {}
        int numDataSets = Integer.parseInt(line.nextToken());
        for (int i = 0; i < numDataSets; i++)
        {
            readOneDataSet();
            System.out.println("Data set " + i + ": " + analyzeDataSet

```

```

());
    }
}

private void readOneDataSet()
{
    try {
        line = new StringTokenizer(br.readLine());
    } catch (Exception e) {}
    this.stringA = line.nextToken();
    this.stringB = line.nextToken();
    this.stringC = line.nextToken();
}

private String analyzeDataSet()
{
    if (stringA.length() + stringB.length() != stringC.length())
        return "no";
    return recAnalyze(0, 0, 0);
}

private String recAnalyze(int ptrA, int ptrB, int ptrC)
{
    if (stringC.length() == ptrC)
        return "yes";
    else if ((ptrA < stringA.length()) &&
             (ptrB == stringB.length()))
    {
        if (stringA.charAt(ptrA) == stringC.charAt(ptrC))
            return recAnalyze(ptrA + 1, ptrB, ptrC + 1);
        else
            return "no";
    }
    else if ((ptrA == stringA.length()) &&
             (ptrB < stringB.length()))
    {
        if (stringB.charAt(ptrB) == stringC.charAt(ptrC))
            return recAnalyze(ptrA, ptrB + 1, ptrC + 1);
        else
            return "no";
    }
    else if ((stringA.charAt(ptrA) != stringC.charAt(ptrC)) &&
             (stringB.charAt(ptrB) != stringC.charAt(ptrC)))
        return "no";
    else if ((stringA.charAt(ptrA) == stringC.charAt(ptrC)) &&
             (stringB.charAt(ptrB) != stringC.charAt(ptrC)))
        return recAnalyze(ptrA + 1, ptrB, ptrC + 1);
    else if ((stringA.charAt(ptrA) != stringC.charAt(ptrC)) &&
             (stringB.charAt(ptrB) == stringC.charAt(ptrC)))
        return recAnalyze(ptrA, ptrB + 1, ptrC + 1);
    else if (recAnalyze(ptrA + 1, ptrB, ptrC + 1).equals("yes")
            || recAnalyze(ptrA, ptrB + 1, ptrC + 1).equals("yes"))
        return "yes";
    else
        return "no";
}

public static void main(String[] args)

```

```
    {  
      (new Zipper()).start();  
    }  
}
```