University of Arkansas
Department of Computer Science and Computer Engineering
2013 High School Programming Contest
Problems

Wing Ning Li Hung Nguyen Adam Higgins Paul Martin
Sawyer Anderson Wes Deneke

Revision Date: March 7, 2013

Please carefully read the output specification. Submitted solutions are graded by a differential program, i.e., the output produced by a submission will be compared *byte-for-byte* with the output of the judges' solutions. Thus, even though submitted solutions may produce output that is *technically* correct, it will be counted as incorrect if these specifications are not followed exactly.

For all problems, solutions receive input from standard input (*stdin*) and produce output on standard output (*stdout*).

Restricting input and output to stdin and stdout does not prevent a program from processing files. Let `Main` be the program name, `Input` be the input file name, and `Output` be the output file name. The following command at the terminal allows the program to read and write the input and output files respectively:

```
./Main < Input > Output      # for C or C++, assuming Main is the executable
java Main < Input > Output   # for Java, assuming Main is a class file
```

For all problems, an *integer* is defined as an optional minus sign followed by one digit from the set 1..9 followed by zero or more digits from the set 0..9. `3`, `-21`, and `309` are integers; `0123`, `- 4`, and `+38` are not.

A *positive integer* is an integer whose value is greater than 0. You may assume the value of a positive integer is no more than $2^{31} - 1$, or $2,147,483,647$.

A *string*'s maximum length is 16,384.

For all problems, a *line* is defined as zero or more characters followed by a newline ('\n') character. Unless otherwise directed, a line of output should have no leading or trailing whitespace. A program producing the integer 27 on one line of output should consist of the character '2' followed by the character '7' followed by the newline ('\n') character.

Input for a problem consists of multiple test cases; each test case is followed by a blank line. You must read input until the *end-of-file*. Refer to the code for the Warm Up problem for hints and ideas for handling input properly.

For each test case, your program must output the result following the specified format in each problem. Output a blank line after the output of each test case. Refer to the code for the Warm Up problem again for hints and ideas for handling output properly.

1 Counting Inversions: a Special Case

Given a list of *unique* integers, an inversion occurs if two elements in the list are out of order, with respect to increasing order. For example, the list 3 1 4 2 has a total of three inversions: the first two elements (3,1) are out of order, the first and last elements (3,2) are out of order, and the next to last and the last elements (4,2) are out of order. Note that the number of inversions is 0 if and only if the list is sorted.

Write a program that computes the number of inversions for a list of two integers.

input Each test case has only one line containing two different integers separated by a space. *There is always a blank line after every test case.*

output For each test case, output two lines: the first line contains the inversion count, *and a blank line as second.*

Example:

Input

```
1 3

5 3

```

Output

```
0

1
```

2 Counting Inversions: the General Case

Please refer to the definition of inversion in the previous problem. Write a program that computes the number of inversions for a list of *unique* integers.

input Each test case has two lines: the first line has only a positive integer $N \leq 1,000$, the number of integers in the list; the second line contains N integers separated by spaces. The list of integers are distinct. *There is always a blank line after every test case.*

output For each test case, output two lines: the first line is the inversion count, *and a blank line as second.*

Example:

Input

```
6
1 2 3 4 5 6

4
4 3 2 1

9
5 99 1 8 2 6 4 7 3
```

Output

```
0

6

20
```

3 Conversion Between Base 3 and Base 5

Decimal numbers, which are the numbers we typically use, are called base 10 numbers, and 10 is the base. The following example illustrates the notion of base: we write base ten number **2012** as $2 \times 10^3 + 0 \times 10^2 + 1 \times 10^1 + 2 \times 10^0$ (note the power of 10 and the position of each digit).

On the other hand, binary numbers, which are the numbers computers typically use, are called base 2 numbers, and 2 is the base. In base 2, we only have two digits 0 and 1. So we write a base two number **10111** as $1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$ (again note the power of 2 and the position of each digit), which is 23 in base ten since $16 + 0 + 4 + 2 + 1 = 23$.

Note that base ten has 10 digits: $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$; and base two has 2 digits: $0, 1$.

To generalize, base three number has 3 digits: $0, 1, 2$; and base 5 number has 5 digits: $0, 1, 2, 3, 4$. A base 3 number **21221** is equivalent to $2 \times 3^4 + 1 \times 3^3 + 2 \times 3^2 + 2 \times 3^1 + 1 \times 3^0$ in decimal, which is $2 \times 81 + 1 \times 27 + 2 \times 9 + 2 \times 3 + 1 \times 1 = 214$. A base 5 number **1324** is equivalent to $1 \times 5^3 + 3 \times 5^2 + 2 \times 5^1 + 4 \times 5^0$ in decimal, which is $1 \times 125 + 3 \times 25 + 2 \times 5 + 4 \times 1 = 125 + 75 + 10 + 4 = 214$.

Since the value (in base 10) of base 3 number **21221** and the value of base 5 number **1324** are the same, base 3 number **21221** can be converted to base 5 number **1324**, and vice versa.

Write a program that converts between base 3 and base 5 numbers.

input Each test case has only one line containing two "numbers". The first is the base, and the second is a positive number N in that base where the *decimal* (base 10) value of N is at most 100,000,000. *There is always a blank line after every test case.*

output For each test case, output two lines: the first line contain the converted number of the input, that is if the input number is in base 3 the output in base 5 and if the input in base 5 the output in base 3, *and a blank line as second.*

Example:

Input

```
3 21221

5 1324

5 4

```

Output

```
1324

21221

11
```

4 Arbitrary Base Conversions

In addition to the bases we have seen in the previous problem, which is $2, 3, 5, 10$. In theory any value could be the base. Let us extend the *digits* for bases up to 62 as follows:

- Characters $0, 1, \ldots, 9$ have decimal values $0, 1, \ldots, 9$ as usual.
- Lowercase characters $a, b, c, \ldots, z$ have decimal values $10, 11, 12, \ldots, 35$.
- Uppercase characters $A, B, C, \ldots, Z$ have decimal values $36, 37, 38, \ldots, 61$.

With these new *digits*, we can represent numbers in bases $2, 3, 4, \ldots, 62$. Here are few examples of numbers in various bases:

1. In base 2, **101100110** is equivalent to $1\times2^8+0\times2^7+1\times2^6+1\times2^5+0\times2^4+0\times2^3+1\times2^2+1\times2^1+0\times2^0$ in decimal, which is 358.
2. In base 11, **2a6** is equivalent to $2 \times 11^2 + 10 \times 11^1 + 6 \times 11^0$ in decimal, which is 358.
3. In base 36, **9y** is equivalent to $9 \times 36^1 + 34 \times 36^0$ in decimal, which is 358.
4. In base 62, **5M** is equivalent to $5 \times 62^1 + 48 \times 62^0$, which is 358.

Write a program to convert numbers between any two bases among the bases $2, 3, 4, 5, \ldots, 62$ using the *digits* as defined previously.

input Each test case has only one line containing three "numbers". The first is a base, the second is a positive number N in that base, and the third is the base that the second number needs to be converted to. The value of N in base 10 is at most 100,000,000. *There is always a blank line after every test case.*

output For each test case, output two lines: the first line containing the converted number, *and a blank line as second.*

Example:

Input

```
10 358 2

10 358 36

10 358 62

2 101100110 11

2 11011111000010101110111100 36

62 Hspc 20
```

Output

```
101100110

9y

5M

2a6

hello

34ed4i
```

5 Friendship

A friendship is a mutual relationship that is symmetric, that is, if Alice is a friend of Bob, then Bob is also a friend of Alice. Such a relationship could be represented as one pair (Alice, Bob), or (Bob, Alice), or as two pairs (Alice, Bob) and (Bob, Alice). Note that a person *cannot* be a friend of himself or herself. Thus, if a person is not friends with anyone else, then he has 0 friends instead of 1 friend (himself).

Given a set of friendships, we are interested in finding those individuals that has the most friends. Instead of using their names, we will use integers 1 to N as their ID's. Thus, in case there are two or more persons with the same maximum number of friends, we will order them based on the increasing order of their ID's.

For a given relationship set, write a program to identify those that have the most friends.

input Each test case may have one or more lines. The first line contains an integer N, $1 \leq N \leq 20$, indicating the total number of individuals. The second line contains an integer M, $0 \leq M \leq 1,000$, indicating the number of relationship lines that follow. A relationship line contains one integer p_i, followed by a space, followed by another integer p_j to represent friendship between p_i and p_j, where $p_i \neq p_j$. *There is always a blank line after every test case.*

output For each test case, output two lines: the first line containing those individuals that have the most friends in increasing order of their ID's with a comma separating the ID's, *and a blank line as second.*

Example:

Input

```
3
1
1 2

3
2
1 3
3 2

3
3
1 3
3 1
2 3

3
4
1 3
3 1
2 3
1 2

3
0
```

Output

```
1,2

3

3

1,2,3

1,2,3
```

6 Maximum Mutual Friendship

Refer to the previous problem for the definition of friendship and how it can be specified.

Three individuals 1,2, and 3 are mutual friends if 1 and 2 are friends, 1 and 3 are friends, and 2 and 3 are friends. In general, a group of people are mutual friends if *every* pair of people in the group are friends. Given a set of friendships, write a program to find the size of the largest mutual friend group. As in the previous problem, instead of using their names, we will use integers 1 to N to represent the N individuals.

input Each test case may have one or more lines. The first line contains an integer N, $1 \leq N \leq 20$, indicating the total number of individuals. The second line contains an integer M, $0 \leq M \leq 1,000$, indicating the number of relationship lines that follow. A relationship line contains one integer p_i, followed by a space, followed by another integer p_j to represent friendship between p_i and p_j, where $p_i \neq p_j$. *There is always a blank line after every test case.*

output For each test case, output two lines: the first line is the size of the largest mutual friend group, *and a blank line as second*

Example:

Input

```
3
1
1 2

3
2
1 3
3 2

3
3
1 3
3 1
2 3

3
4
1 3
3 1
2 3
1 2

3
0
```

Output

```
2

2

2

3

0
```

7 Is This Square Magic?

An $n \times n$ **magic square** is a square filled with integers such that the sums of the n rows, n columns, and two diagonals are all the same. The sums are called the **magic number** of the magic square. Following is an example of a 3×3 magic square whose sums of all rows, columns, and diagonals are 15.

2	7	6
9	5	1
4	3	8

The numbers for magic squares need *not* be positive, however. Here is another 3×3 magic square whose magic number is -42:

0	-39	-3
-17	-14	-11
-25	11	-28

In addition, the numbers of a magic square need *not* be unique, so the following is a trivial magic square with magic number 3:

1	1	1
1	1	1
1	1	1

Given a 3×3 square, write a program to determine if it is a magic square or not. If it is a magic square, print out its magic number.

input Each test case contains three lines describing the numbers in the three rows of the square. On each line has three integers for the three values in the respective columns of the row. All of the entries n of the squares are in the range $-100,000 \leq n \leq 100,000$. *There is always a blank line following each test case.*

output For each test case, output two lines. The first line is either the string `MAGIC` and magic number, both of which are separated by one space, if the square is magic; or the string `NORMAL` if it is not. *The second line is a blank line.*

Example:

Input

```
2 7 6
9 5 1
4 3 8

0 -39 -3
-17 -14 -11
-25 11 -28

1 1 1
1 1 1
1 1 1

1 2 3
4 5 6
7 8 9
```

Output

```
MAGIC 15

MAGIC -42

MAGIC 3

NORMAL
```

8 Magic-Square Sequence

A sequence of nine integers is called a **magic-square sequence** if a 3×3 magic square can be formed using these numbers. For example, all of the following sequences of integers:

- $1, 2, 3, 4, 5, 6, 7, 8, 9$
- $-25, 0, -39, -3, -14, -11, -28, 11, -17$
- $1, 1, 1, 1, 1, 1, 1, 1, 1$

are magic-square sequences since a magic square can be formed from each sequence by arranging the numbers as shown in the examples of the previous problem. On the other hand, the following sequences *cannot* form magic squares no matter how one arranges their numbers:

- $-4, -5, 3, 2, 1, 6, -14, 7, 8$
- $1, 1, 1, 1, 1, 2, 1, 1, 1$

Thus, they are *not* magic-square sequences.

Write a program to determine if a nine integer sequence is a **magic-square sequence** or not.

input Each test case contains only one line with nine space-separated integers n of the sequence, and $-100,000 \leq n \leq 100,000$. *There is always a blank line following each test case.*

output For each test case, output two lines. The first line is either the string `YES` if the input sequence is magic-square, or `NO` if it is not. *The second line is a blank line.*

Example:

Input

```
1 2 3 4 5 6 7 8 9

-25 0 -39 -3 -14 -11 -28 11 -17

1 1 1 1 1 1 1 1 1

-4 -5 3 2 1 6 -14 7 8

1 1 1 1 1 2 1 1 1
```

Output

```
YES

YES

YES

NO

NO
```