University of Arkansas
Department of Computer Science and Computer Engineering
2018 High School Programming Contest
Problems

Wing Ning Li Hung Nguyen Adam Higgins Paul Martin Tyler Moore
Matthew Luther

Revision Date: March 9, 2018

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 $2^{32} - 1$, or $4,294,967,295$.

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 Comparing two integers

The ability to compare the values stored in two variables is a fundamental operation in any useful Apps.

Given two integer a and b, output `<` if a is less than b, `=` if a is the same as b, and `>` if a is greater than b.

input Each test case a a pair of of integers separated by a single space. *There is always a blank line after every test case.* Input may contain multiple test cases.

output For each test case, output two lines: the first line containing one of `<`, `=`, and `>`; *and a blank line as second*

Example:

Input

```
1 2

1 1

2 1

```

Output

```
<

=

>
```

2 Rearranging numbers

Given a sequence of non negative integers, rearrange the values so that the first element is the smallest, the second element is the largest, the third element is the second smallest, and the fourth element is the second largest and so on.

input Each test case is a sequence of non negative integers separated by a single space in a line. The number of integers in the sequence is at most 1000. *There is always a blank line after every test case.* Input may contain multiple test cases.

output For each test case, output two lines: the first line containing the rearranged sequence where each number is separated by a single space; *and a blank line as second*

Example:

Input

```
1 2 3 4

4 1 3 8 7

```

Output

```
1 4 2 3

1 8 3 7 4
```

3 Converting time interval to hours and minutes

A day has 24 hours. We may specify the times of a day using hh:mm format. For example, 10:15 (for 10:15 am) and 14:30 (for 2:30 pm). The time interval from 10:15 to 14:30 indicates a duration of 4 hours and 15 minutes.

Write a program to convert a time interval to a duration of hours and minutes.

input Each test case contains a pair of times in hh:mm (00:00 to 24:00) format with a space separating them. *There is always a blank line after every test case.* Input may contain multiple test cases.

output For each test case, print the duration of the time interval as shown in the examples *and a blank line.*

Example:

Input

```
10:15 14:30

22:01 23:01

```

Output

```
4 hours and 15 minutes.

1 hour and 0 minute.
```

4 Finding the longest common free time in two schedules

A time interval of the previous problem may be used in a schedule specifying an activity, which means that the time slot is not available or free. Here is an example of a schedule:

```
00:00 06:00 sleep
06:45 07:00 breakfast
07:30 08:00 trip to school
08:00 15:30 school time
16:00 17:30 practice
18:20 18:45 dinner
19:30 20:45 homework
22:00 23:59 sleep
```

Given the schedules of two persons, find the longest common free time between the two schedule so the two persons may spend the most time together.

input Each test case contains two schedules. The first line of a schedule contains a single integer n that specifies the number of activities of the day ($n \leq 30$). The next n line has the format of `time time appointment`, where time is in the format of hh:mm and appointment is a string. A space is used to separated each element. *There is always a blank line after every test case.* Input may contain multiple test cases.

output For each test case, output one line *and a blank line.* The line begins with `The longest free time starts at hh:mm and will last for this-long`. Note that verb+hh:mm+ should be replaced by that actual time and `this-long` should be replaced by actual hours and minutes. See example below. In case no common free time can be found, the first line of output should be `No common free time exists!`

Example:

Input

```
8
00:00 06:00 sleep
06:45 07:00 breakfast
07:30 08:00 trip to school
08:00 15:30 school time
16:00 17:30 practice
18:20 18:45 dinner
19:30 20:45 homework
22:00 23:59 sleep
7
00:00 06:00 sleep
06:45 07:00 breakfast
07:30 08:00 trip to school
08:00 15:30 school time
18:20 18:45 dinner
19:30 20:30 homework
22:45 23:59 sleep

2
00:00 07:00 play games
07:30 15:00 sleep
7
00:00 06:00 sleep
06:45 07:00 breakfast
07:30 08:00 trip to school
08:00 15:30 school time
18:30 18:45 dinner
19:30 20:30 homework
22:45 23:59 sleep

1
00:00 07:00 play games
1
07:00 24:00 sleep
```

Output

```
The longest free time starts at 20:45 and will last for 1 hour and 15 minutes.

The longest free time starts at 15:30 and will last for 3 hours and 0 minute.

No common free time exists!
```

5 Finding the missing pieces

A binary string is sequence of 0's and 1's. For example, `0011010101`. There are many ways to break a string into two pieces. For example, 0011010101 may be broken into 0 and 011010101 or 0011010 and 101.

Given one binary string s and another two binary strings x and y, we would like to know if s may be broken into x and y.

Write a program to help us answering the above question.

input Each test case has three lines. The first line is s, the second line is x, and the third line is y. The length of the binary string is at most 256. *There is always a blank line after every test case.* Input may contain multiple test cases.

output For each test case, output two lines: the first line containing `YES` or `NO` *and a blank line as second*

Example:

Input

```
0011010101
0
011010101

0011010101
101
0011010

11
0
1
```

Output

```
YES

YES

NO
```

6 Recovering from broken pieces[1]

In the previous problem, we discover that there are many ways to break a binary string into two pieces. Given a set of binary strings obtained as the result of broken the same binary string into two pieces, we would like to recover the original string. For example, given 0, 011010101, 0011010, and 101, the original string to be recovered must be 0011010101.

input Each test case has even number of lines. Each line contains a binary string resulted from broken the string to be recovered into two pieces, and the other piece of the two also appears in some other line of the input. The length of the string to be recovered is at most 256. *There is always a blank line after every test case.* Input may contain multiple test cases.

output For each test case, output two lines: the first line contains the original string recovered. *and a blank line as second.* In case multiple solutions are possible, the first such answer in the lexicographical ordering shall be the answer.

Example:

Input

```
0
101
011010101
0011010

1
11
00
0
001
011
0
011
```

Output

```
0011010101

0011
```

[1]This problem is adopted from a problem called File Fragmentation in the book Programming Challenges by Skiena and Revilla

7 Compute the penalty

A web page designer has N job orders from their customers. Each month, the system uses a first-come-first-serve schedule to arrange the jobs. Customers require their jobs to be started the first day of the month. Every day a job is delayed incurs a penalty. Each job takes at least on day and may require several days. Once a job is started, the designer will not start a different job before completing the current job.

Each job has two integers associated with it: d (days needed to complete the job) and p (cents per day penalty). For example, for the following job order: (3,3), (2,5), (4,3) the penalty is $0 \times 3 + 3 \times 5 + 5 \times 3 = 30$.

Write a program to compute the penalty incurred for the job order maintained by the system

input Each test case has several lines. The first line is an integer N ($1 \leq N \leq 1000$), indicating the number of jobs. There are N lines after that. Each line contains two integers separated by a space, specifying a job length (from 1 to 100 days) and a job penalty (from 1 to 100 cents). The order of the jobs is given by the order of the lines. *There is always a blank line after every test case.* The input may contain multiple test cases.

output For each test case, output two lines: the penalty incurred for the ordering as the first line *and a blank line as second*

Example:

Input

```
3
3 3
2 5
4 3

1
4 1

```

Output

```
30

0
```

8 Minimizing the penalty[2]

It seems that in the previous problem the ordering kept by the system may not result in the minimum penalty. Write a program to rearrange the job order so that the penalty is minimized.

input Each test case has several lines. The first line is an integer N ($1 \leq N \leq 1000$), indicating the number of jobs. The are N lines after that. Each line contains two integers separated by a space, specifying a job length (from 1 to 100 days) and a job penalty (from 1 to 100 cents). The order of the jobs is given by the order of the lines. *There is always a blank line after every test case.* Input may contain multiple test cases. Note the input specification is identical to the previous problem.

output For each test case, output three lines: the first line is the rearranged order with the minimum total penalty; the second line is the total penalty for the order; *the third line is blank.* If multiple orders have the minimum total penalty, output the order that maintains the original order of the jobs (the input order) as much as possible.

Example:

Input

```
3
3 3
2 5
4 3

1
4 1

```

Output

```
2 1 3
21

1
0
```

[2]This problem is adopted from a problem called Shoemaker's problem in the book Programming Challenges by Skiena and Revilla