South African
Computer Olympiad Training
DAY 1

Overview

Author	IOI 1994	Bruce	IOI 1995
Problem	**Castle**	**Change**	**Wires & switches**
Program name	`castle.exe`	`change.exe`	`wires.exe`
Source name	`castle.pas` `castle.java` `castle.cpp`	`change.pas` `change.java` `change.cpp`	`wires.pas` `wires.java` `wires.cpp`
Input file	`stdin`	`stdin`	`wires.in`
Output files	`stdout`	`stdout`	`stdout`
Time limit	1 second	1 second	1 second
Num. of tests	5	5	10
Points per test	20	20	10
Total points	**100**	**100**	**100**

The maximum total score for Day1 is 300 points.

* See text of Question.

South African Computer Olympiad Training DAY 1

The Castle

```
    1   2   3   4   5   6   7
 #############################
1 #   |   #   |   #   |   |   #
  #####---#####---#---#####---#
2 #   #   |   #   #   #   #   #
  #---#####---#####---#####---#
3 #   |   |   #   #   #   #   #
  #---#########---#####---#---#
4 # ->#   |   |   |   |   #   #
  #############################
```

(Figure 1)

#	= Wall
|	= No wall
_	= No wall
→	= It points to the wall to remove according to the example output.

Figure 1 shows the map of a castle. Write a program that calculates

1. how many rooms the castle has
2. how big the largest room is
3. which wall to remove from the castle to make as large a room as possible.

The castle is divided into m * n (m<=50, n<=50) square modules. Each such module can have between zero and four walls.

Input Data

Your program is to read from standard input. The castle is represented in the form of numbers, one for each module.

- The first two lines contain the number of modules in the north-south direction and the number of modules in the east-west direction.
- In the following lines each module is described by a number (0<=p<=15). This number is the sum of: 1 (= wall to the west), 2 (= wall to the north), 4 (= wall to the east), 8 (= wall to the south). Inner walls are defined twice; a wall to the south in module 1,1 is also indicated as a wall to the north in module 2,1.
- The castle always has at least two rooms.

Thus the representation for our example is:

```
4
7
11 6 11 6 3 10 6
7 9 6 13 5 15 5
1 10 12 7 13 7 5
13 11 10 8 10 12 13
```

Output Data

Write the following on three lines to standard output: First the number of rooms, then the area of the largest room (counted in modules) and a suggestion of which wall to remove (first the row and then the column of the module next to the wall and finally the compass direction that points to the wall). In our example ("4 1 E" is one of several possibilities, you need only produce one):

```
5
9
4 1 E
```

Constraints

- 2 <= m,n <= 50
- Time limit: 1 second
- Memory limit: 16384 Kb

Scoring

You receive 4 points for the number of rooms, 6 points for the area of the largest room, and 10 points for the wall to remove, for a maximum of 20 points per test-case over 5 test cases. Any form of invalid output data results in zero for that case.

Making Change

The cows have decided that they are tired of the strange American system of coins, being particularly annoyed by the large gap between the quarter ($0.25) and the $1 coins that force one to carry lots of quarters. They are going to create their own system of coins with sensible values.

However one feature of the American system that they like is that one can always make up a value with a "greedy" algorithm and get the minimal number of coins. The greedy algorithm says that one should always pick the largest value coin that is not greater than the amount still needed. So for example to make 83c in the US system (which has coins of values 1c, 2c, 5c, 10c, 25c and $1), one first picks three 25c coins to make 75c, then a 5c to make 80c, then a 2c and finally a 1c.
This results in six coins, and there is no way to make 83c with fewer coins. However if the coins had been 1c, 4c and 6c, then making 8c using a greedy algorithm takes 3 coins (6+1+1) even though it is possible to make 8c using 2 coins (4+4).

Task

You must write a program to determine whether the cow's proposed set of coins satisfies the greedy property, and if not compute the smallest value for which the greedy algorithm is not optimal.

Input data (stdin)

The first line is N, the number of coins. Each of the next N lines contains the value of a coin, in cents. Values may be repeated (e.g. for special editions). It is guaranteed that at least one coin will have the value of 1c.

Sample input:

```
3
1
6
4
```

Output data (stdout)

If the coin set does not have the greedy property, output the smallest value for which the greedy algorithm is non-optimal, followed by the minimum number of coins required to create this value (space separated).

If the coin set has the greedy property, output "-1 -1".

Sample output:

```
8 2
```

Constraints

1 <= N <= 100
1 <= each coin <= 10000

Scoring

An incorrectly formatted output file scores 0.
If the coin set has the greedy property, a correct answer scores 100% and an incorrect answer scores 0.
If the coin set does not have the greedy property, then 50% is awarded for getting the first output correct. If (and only if) this output is correct, 50% is awarded for getting the second field correct.

Time limit

1 second

South African Computer Olympiad Training DAY 1

Wires and Switches

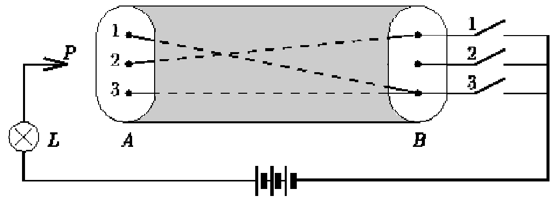

Figure 1:
Cable with three wires and three switches

In Figure 1, a cable with three wires connects side *A* to side *B*. On side *A*, the three wires are labeled 1, 2, and 3. On side *B*, wires 1 and 3 are connected to switch 3, and wire 2 is connected to switch 1.
In general, the cable contains *m* wires (1<=*m*<=90), labeled 1 through *m* on side *A*, and there are *m* switches on side *B*, labeled 1 through *m*. Each wire is connected to exactly one of the switches. Each switch can be connected to zero or more wires.

Measurements

Your program has to determine how the wires are connected to the switches by doing some measurements. Each switch can be made either conducting or non-conducting. Initially all switches are non-conducting. A wire can be tested on side *A* with probe *P*: Lamp *L* will light up if and only if the sensed wire is connected to a conducting switch.
Your program begins by reading one line with the number *m* from *standard input*. It then can give three kinds of commands by writing a line to *standard output*. Each command starts with a single uppercase letter: T (Test a wire), C (Change a switch), and D (Done). Command T is followed by a wire label, C by a switch label, and D by a list whose *i*-th element is the label of the switch to which wire *i* is connected.
After commands T and C, your program should read one line from standard input. Command T returns Y (Yes) when the wire's switch is conducting (the lamp lights up), otherwise it returns N (No). Command C returns Y if the new switch state is conducting, and N otherwise. The effect of command C is to change the state of the switch (if it was conducting then it will be non-conducting afterwards and vice versa); the result is returned just for feedback.
Your program may give commands T and C mixed in any order. Finally, it gives command D and terminates. Your program should give no more than nine hundred (900) commands in total.

Example

Figure 2 presents an example conversation involving 8 commands relating to Figure 1.

Std Output	Std Input
	3
C3	Y
T1	Y
T2	N
T3	Y
C3	N
C2	Y
T2	N
D3 1 3	

Figure 2: Example conversation

The file wires.in is fed to the opponent to initialise the state. So that you may test your program against the opponent with custom data, the format of the file is: first line is the number of wires (same as the number of switches), the next line is the switch the first wire is connected to, and so on. This is the wires.in for the example:

```
3
3
1
3
```

Constraints

- 1 <= m <= 90
- Time limit: 1 second.
- Memory limit: 16384 Kb

Scoring

There are 10 test cases. Each test case is worth 10 points, awarded if the correct 'done' command is issued in under 900 commands. Note that the evaluation program will exit immediately after the line for the 'done' command is output, after the 900th command has been issued, or after invalid output. No points are awarded if the program produces any invalid output (non-numeric data where a number should be, invalid commands, out-of range wire indices, and so forth), the answer is wrong, or the command limit is reached.