French-Australian Regional Informatics Olympiad

Friday 7th March, 2014

Duration: 4 hours

3 questions

All questions should be attempted

Problem 1
Network System Administration

Input File: *standard input*
Output File: *standard output*

Time and Memory Limits: 0.5 seconds, 128 MB

You are the Network System Administrator (NSA) for a local school. Recently hired on a huge salary after a catastrophic school website defacement, you are responsible for monitoring the school network to keep the systems safe.

The school administration had agreed to give you access to a stream of student e-mail *meta-data* (the time, sender and recipient of every e-mail sent through the network) although this decision has recently come under fire from the student body over privacy concerns.

In order to prove to the school administration that your complete access to the email meta-data is justified, you will write a program to show how the meta-data would be useful in the case of a *trojan mule* attack on the network. From your expert NSA training, you know that a trojan mule is a computer virus a lot like a trojan horse except that it does not reproduce.

In particular, a trojan mule starts on some initial computer then attaches itself to the first e-mail sent from that computer. Upon transmission to the e-mail's recipient, the virus removes itself from the sender's computer and infects the recipient, then waits for the next outgoing e-mail to attach to and so on. Thus, only one copy of the virus exists in the network at any time.

Your program must be able to answer queries of the form "if a trojan mule was initally on computer X, what computer would it be on now?". These queries may be interspersed with new e-mail meta-data.

Input

- The first line of input will contain one line with two space-separated integers N and M, representing the number of computers and the number of input lines.
- The following M lines of input will each take one of the two forms:
  1. `E` x y ($1 \leq x, y \leq N$ and $x \neq y$), indicating a new e-mail sent from computer x to computer y; or,
  2. `Q` x ($1 \leq x \leq N$), a query of the virus' current host given that it was initially on computer x.

Output

Your program must output one line for each query (`Q` x line of input), containing one integer representing the computer that the virus would be on at that point in time if it started on computer x.

(continued over . . .)

Sample Input

```
5 8
E 1 2
E 3 4
Q 4
E 2 3
E 3 5
Q 1
E 5 2
Q 2
```

Sample Output

```
4
5
2
```

Explanation

There are 5 computers connected through the school network.

1. An email is sent from 1 to 2.
2. An email is sent from 3 to 4.
3. We are asked, if the trojan mule was initially on computer 4, where is it now? As no emails have been sent from 4 yet, the trojan mule would still be on 4.
4. An email is sent from 2 to 3.
5. An email is sent from 3 to 5.
6. We are asked, if the trojan mule was initially on computer 1, where is it now? The trojan mule was transmitted from 1 to 2 (#1) then from 2 to 3 (#4) then from 3 to 5 (#5) so it would now be on computer 5.
7. An email is sent from 5 to 2.
8. We are asked, if the trojan mule was initially on computer 2, where is it now? The trojan mule was transmitted from 2 to 3 (#4) then from 3 to 5 (#5) then from 5 back to 2 (#7) so it would now be back on computer 2.

Subtasks & Constraints

For all subtasks, $1 \leq N \leq 100,000$ and $1 \leq M \leq 100,000$.

- For Subtask 1 (10 points), $1 \leq N \leq 100$ and $1 \leq M \leq 100$.
- For Subtask 2 (25 points), $1 \leq N \leq 100$ and $1 \leq M \leq 50,000$.
- For Subtask 3 (30 points), no computer receives or sends any more e-mails after it has sent one e-mail. That is, once a computer appears as an x in an `E` line, it will not appear in any future `E` lines.
- For Subtask 4 (35 points), no further constraints apply.

Problem 2
Connect

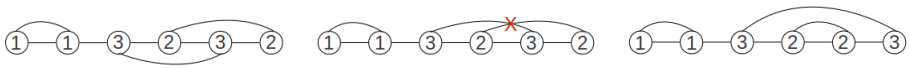
Input File: *standard input*
Output File: *standard output*

Time and Memory Limits: 1 second, 128 MB

You are building a critical electronic circuit for the xPhone 6, a new phone that will revolutionise the entire industry with its 2.2mm thinner body and 5.4% less curve on its corners than the obsolete and useless xPhone 5.

The electronic circuit you are building has the following form:

- There are $2N$ nodes on a main wire (a "bus"), electrically connected to each other in a line.
- Each node also needs to be connected to exactly one other node with a new wire you will place. The $2N$ nodes are labelled by integers between 1 and N. There will be exactly two nodes with each label, and these N pairs of nodes need to be connected to each other with N new wires.
- The circuit must be printed on to a flat circuit board, and **no wires can overlap**. Each placed wire between two nodes will either be entirely above the main wire, or entirely below.



The left image illustrates a valid wiring configuration. The middle image illustrates an invalid wiring configuration. The right image illustrates a valid wiring configuration that only places new wires above the main wire. Note that for the test case in the left and middle images, there is no valid wiring configuration that only places new wires above the main wire.

You must write a program that determines if it is possible to build the circuit with these constraints. If so, you must output for each node whether its new wire connects to it from above or below the main wire.

Input

- The first line of input will contain one integer N, the number of pairs of nodes.
- The following $2N$ lines will each contain one integer L_i between 1 and N, the label of the ith node. Each value of L_i will appear twice.

Output

- If your program determines that there is no way to connect all pairs of nodes, it should output one line containing `0`.
- Otherwise, your program should output $2N$ lines describing the way it connects the pairs. The ith line should contain either `1` or `2`. A `1` means the node is connected from above, and `2` means it is connected from below to the other node of the same label.

Sample Input 1

```
3
1
1
3
2
3
2
```

Sample Output 1

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

Sample Input 2

```
3
1
1
3
2
2
3
```

Sample Output 2

```
1
1
1
1
1
1
```

Explanation

The first sample case corresponds to the case in the left and middle diagrams above. The second sample case corresponds to the case in the right diagram.

Subtasks & Constraints

For all subtasks, $1 \leq N \leq 100\,000$.

- For Subtask 1 (20 points), $1 \leq N \leq 100$ and the circuit is either impossible to connect or only requires new wires above the main wire (that is, either `0` will be the correct output or $2N$ lines each containing `1` will be a correct outpt).
- For Subtask 2 (20 points), $1 \leq N \leq 100$.
- For Subtask 3 (20 points), $1 \leq N \leq 1\,000$.
- For Subtask 4 (20 points), the circuit is either impossible to connect or only requires new wires above the main wire (that is, either `0` will be the correct output or $2N$ lines each containing `1` will be a correct outpt).
- For Subtask 5 (20 points), no further constraints apply.

Problem 3
Super Maria II: Another Castle

Time and Memory Limits: 1 second, 128 MB

The Fungus Republic had been a peaceful place thanks to the brave deeds of the humble electrician, Maria, but this all changed when her arch-nemesis Wowser invaded. Wowser kidnapped Prince Nectarine (again) and hid him somewhere he thought Maria would never find.

Maria was undaunted, however, and using the signal from Nectarine's mushroom phone was able to narrow down his location to one of N castles. The castles are arranged in a line and are numbered from 0 to $N-1$ where castle 0 is the leftmost castle, castle 1 is second from the left, etc.

Maria is very hasty about finding the prince, so she has rented a carriage to be pulled by Shoyi the friendly dinosaur. Whenever Maria wants to get out of the carriage or change direction, she must pay Shoyi a tip.

Maria begins at castle 0, facing to the right. When Maria is at castle i, she can do one of three things.

- Move to any castle in the direction she is facing without spending any coins. That is, if she is facing left, she can move to any castle j at no cost providing that $j < i$. If she is facing right then she can move to any castle j at no cost providing that $j > i$.
- Pay Shoyi C_1 coins to turn the carriage around and face the opposite direction.
- Pay Shoyi C_2 coins to let Maria out so that she can open the door of castle i. If Maria chooses to do this, one of two things happen.
  - She finds Prince Nectarine, defeats Wowser, and returns the Fungus Republic to safety.
  - Shce finds Frog, who kindly tells her that the prince is in another castle and in which direction (left or right) she must go to find him.

Maria wants to minimise the total number of coins she spends in finding the prince. Maria whips out her trusty laptop and sends you an e-mail, asking you to write a program to help find the prince while minimising the total number of coins she pays Shoyi.

Input / Output

This task has no input or output files. Instead it must use functions from the header file "`carriage_lib.h`" to gain information. These provided functions are described in detail on the next page.

Library

Your program must interact with functions in "`carriage_lib.h`", as follows:

- Your program must begin by calling `carriage_nb_cells()`, `carriage_reverse_cost()`, and `carriage_open_cost()`
  - `carriage_nb_cells()` returns N, the number of castles which the prince may be in.
  - `carriage_reverse_cost()` returns C_1, the number of coins Maria must pay Shoyi to change direction.
  - `carriage_open_cost()` returns C_2, the number of coins Maria must pay Shoyi to stop so that she can open a castle door.
- Your program should then repeatedly call `carriage_move(int k)` and `carriage_open()` in order to find the castle which holds the prince.
  - `carriage_move(int k)` moves the carriage a distance of $|k|$ to the left or right depending on whether k is positive or negative. For instance calling this function with -5 as an argument will cause Maria's carriage to move 5 cells to the left. If Maria is not already facing in the direction she is made to move, she must pay Shoyi C_1 coins to change direction. If the argument given to this function causes Maria to move to a position which is negative or is not less than N, then your program will be terminated and will be awarded no marks.
  - `carriage_open()` causes Maria to pay Shoyi C_2 coins to stop the carriage so that she can open the castle door. If Maria is at the castle where Prince Nectarine has been hidden then your program will be terminated and you will be awarded marks according to the "Scoring" section on the next page. Otherwise, this function will return 1 if the prince is in a castle to the right of the current castle, or -1 if the prince is in a castle to the left of the current castle.

Library Prototypes

The signatures for these functions in C and C++ are as follows:

```
int carriage_nb_cells();
int carriage_reverse_cost();
int carriage_open_cost();
void carriage_move(int k);
int carriage_open();
```

Experimentation

In order to experiment with your code on your own machine, first download `carriage_lib.h` to the same folder as your code file and add `#include "carriage_lib.h"` to the top of your code.

The compiled executable will take four whitespace-separated integers from standard input on the first call to a provided function from the header file. The integers should represent, respectively, N, C_1, C_2 and p where the castle with index p (between 0 and $N-1$) contains Prince Nectarine.

The executable will print one line to standard output, either an error message due to invalid input or an invalid move, or an indication of success including the number of coins used.

Sample Input

```
10 3 4 7
```

Sample Session

Function Call	Explanation
`carriage_nb_cells()`	Returns 10, the number of castles.
`carriage_reverse_cost()`	Returns 3, the cost of changing direction.
`carriage_open_cost()`	Returns 4, the cost of opening a castle door.
`carriage_move(5)`	Moves to castle 5, at no cost.
`carriage_open()`	Returns 1, as the Prince is in castle 7, to the right of castle 5. This costs 4 coins.
`carriage_move(4)`	Moves to castle 9, at no cost.
`carriage_open()`	Returns -1, as the Prince is in castle 7, to the left of castle 9. This costs 4 coins.
`carriage_move(-3)`	Moves to castle 6. This costs 3 coins, as it involved a change in direction.
`carriage_open()`	Returns 1, as the Prince is in castle 7, to the right of castle 6. This costs 4 coins.
`carriage_move(1)`	Moves to castle 7. This costs 3 coins, as it involved a change in direction.
`carriage_open()`	Finds the prince! This costs 4 coins. The program outputs that the prince has been found with the use of 22 coins and exits.

Scoring & Constraints

- For all cases, N, C_1 and C_2 are integers such that $1 \leq N \leq 1,000,000$ and $0 \leq C_1, C_2 \leq 10,000$.
- For 30% of cases, $N \leq 1000$ and $C_1, C_2 \leq 1000$.
- For each test case, your score will be 100 if your program finds the Prince with the minimum number of coins. Otherwise, if your program successfully finds the Prince your score will be a value between 0 and 60, based on a linear function of the number of coins your program used.
- Your total score for the task will be the average of your scores for each individual test case.
- Note that the judges use a modified version of the library which will force your program to use as many coins as possible in each case.