ACM Pacific NW Region Programming Contest
15 November 2003

Problem H: Audit Time

Action Comic Magazine's budget for their new Intranet order entry site was pretty low, so they decided to build their own database engine. It stores a set of customer accounts and is able to retrieve balances and update with a new balance. The new balance is calculated and provided to the database by the client application. This is the good news!

The **bad** news is that ACM really didn't understand transactions or how databases protect the integrity of their data. In fact, customer's accounts can be reviewed and updated by multiple users at the same time, without the transactional controls normally found in a database engine. The result is a set of transactions having steps mixed together (interleaved), with no assurance that the final result is correct.

A sample history of a set of two interleaved transactions illustrates the issue:

Transaction ID	Action	CustomerID	Step #
1	Retrieve	A	1
1	Retrieve	B	2
2	Update	B	3
2	Update	A	4
1	Update	A	5

As you can see, Transaction 2's update of Customer A will be overwritten by Transaction 1's update to the same customer. (This is an example of a "lost update.")

You have been hired to determine which sets of transactions need to be subjected to a detailed audit. You will do this by determining which transaction sets would have provided the same ending data had they been serialized in a properly−written database.

Definitions:

Conflicting pairs: A pair of steps are "in conflict" if they are not in the same transaction, they access the same customer account, and at least one step is an update. In the example above, conflicts exist between steps 1 and 4, 2 and 3, and 4 and 5.

Serial History: A history is serial if for any two transactions T_i and T_j in it, where i <> j, all operations from T_i are ordered in `s` before all operations from T_j, or vice versa. In the example above, there are two possible serial histories (either all steps from transaction #1 followed by all steps from transaction #2, or the reverse).

Problem Requirement:

For each interleaved transaction set history `s`, you must determine if there is an equivalent serial history `s'` that contains the same steps and where all conflicting pairs of steps occur in the same order in both histories. Transaction sets meeting **both** these criteria need not be subjected to further audit.

The example transaction above does **not** meet the requirements of this problem since situating Transaction 1 before Transaction 2 would reverse the order of the conflicting pair {step 4, step 5}, while situating Transaction 2 first would reverse the orders in the conflicting pair {step 1, step 4} as well as the order in the conflicting pair {step 2, step 3}.

Input: Read from file h.in

The input to your program will be several transaction set histories. The first line will indicate the number of histories to analyze.

Each history's first line is 1 digit h indicating the number of steps in the history. (Note: while the input will not specify the number of transactions in a history, you may assume 0 < number of transactions < 20.) The next h lines of input contain the transaction number, 'R' (retrieve) or 'U' (update), and the associated Customer ID as shown below. One space separates each item in the row. An entry of "1 R 55" would indicate *Transaction #1, 'Retrieve,' Customer #55.*

SAMPLE INPUT:

```
2
4
1 R 55
2 R 55
2 U 55
1 U 55
8
1 R 55
2 R 55
1 R 70
1 U 55
2 U 80
3 R 70
3 U 80
3 U 70
```

Output

Your output will list each history and whether it requires investigation, as seen below. There are four spaces between the colon and the result for each history.

SAMPLE OUTPUT

```
History #1:    Needs Investigation
History #2:    OK
```

Note: in the case of History #2 in the example above, the valid serial history would be to perform Transaction #2 first, followed by Transaction #1, followed by Transaction #3.