611 Parallel Deadlock

A common problem in parallel computing is establishing proper communication patterns so that processors do not deadlock while either waiting to receive messages from other processors, or waiting for the sending of messages to other processors to complete. That is, one processor will not complete sending a message until it is received by the destination processor. Likewise, a receive cannot complete until a message is actually sent.

There are two modes of communication: blocking and non-blocking. A blocking send will not complete until a matching receive is performed at the destination processor. Likewise, a blocking receive will not complete until the matching send is performed by the source processor. Non- blocking actions will "return" immediately (i.e., allow the program to continue), but will not actually complete until the matching action is performed at the target. The matching action of a send is a receive (either blocking or non-blocking), and similarly, the matching action of a receive is a send (either blocking or non-blocking).

At the start of each timestep, each processor that is not blocked starts to run its next instruction. Processors that execute blocking instructions become blocked. Messages can be received at the end of the timestep in which they are sent, but may need to wait several timesteps until the recipient performs a matching receive. If the recipient of a message is waiting to receive from the sender, then the message is received in the same timestep. Messages are received in the order that they are sent. If all of the actions for a particular blocking instruction complete at the end of the timestep, then the processor that ran the instruction will be unblocked before the next timestep.

A correct program will terminate only when all of its actions have completed. Pending non-blocking operations must be completed before a program can terminate.

Your program will take in a list of processors and actions (no more than 100 for each processor), and determine if each processor finishes its program. If a given processor does not finish, it must print out which other processors are preventing it from finishing.

Input

The first line will be a single positive integer that tells how many processors will be listed. For each processor there will be one line containing the name of the processor (a single capital letter) followed by a positive integer, N . The following N lines will contain the instructions that comprise the program for that processor.

An instruction is of the form "*Mode Action Target(s)*" where "*Mode*" can be "`B`" or "`N`", for blocking or non-blocking, respectively. "*Action*" can be "`S`" or "`R`", for send or receive, respectively. "*Target(s)*" will be one or more processor names to which the action is to be addressed. No processor will be listed twice and a processor will never attempt any sort of communication with itself . A send to multiple targets will not complete until matching receives have been performed by all of the targets (and vice versa).

Output

Given that instruction 1 occurs at $t = 1$, your program will output at which timestep each processor finishes. If a processor does not finish, you must output which processor are preventing that processor from finishing. Processors should be listed in alphabetical order, both for the list of processors and the

sets of processors that prevent a processor from finishing. The list of processors preventing termination should list processors at most once and separate multiple processors with "`and`".

Sample Input

```
4
I 5
B S B P C
N S B P C
N R B
B R P
B R C
B 2
B R I
B S I
P 3
N S I
N R I
B R I
C 4
N S I
B R I
B S P
B R I
```

Sample Output

```
B finishes at t=4
C never finishes because of P
I never finishes because of B and C
P finishes at t=5
```

Notice how C's final blocking receive would be matched by a send on I if both instructions were executed. However, it never gets executed because it is stuck in the blocking send to P (that has no matching receive on P), therefore causing deadlock on I.