MUTEXES (ESTONIA)

Modern programming languages allow writing programs that consist of several threads of execution. This is as if several programs are running in parallel in the same address space, accessing the same variables. Often the threads need to be synchronized with each other. For instance, one thread may need to wait for another to complete some computation and store the result into some variable.

The simplest tool for thread synchronization is *mutex*. A mutex is a special object that can be in *locked* or *unlocked* state. A locked mutex is always owned by exactly one thread. There are two operations that a thread can apply to a mutex: LOCK and UNLOCK.

When a thread applies LOCK to a mutex that is currently unlocked, the mutex becomes locked and the thread acquires ownership of the mutex. If a thread tries to apply LOCK to a mutex that is already locked by some other thread, the thread is blocked until the mutex is unlocked.

When a thread applies UNLOCK to a mutex owned by the thread, the mutex becomes unlocked. If there were other threads waiting to LOCK the mutex, one of them is granted ownership of the mutex. If there were several threads waiting, one is selected arbitrarily.

One of the common problems in multithreaded programs are deadlocks. A deadlock occurs when two or more threads are waiting for each other to release a mutex and none of them can continue. A deadlock occurs also when a thread is waiting for a mutex that was locked by another thread that has terminated without releasing the mutex.

Task: You are provided descriptions of some threads and your task is to decide whether deadlocks can occur.

Each of the threads is a sequence of instructions of the following form:

```
LOCK <mutex>
UNLOCK <mutex>
```

You may assume the following about the commands:

- names of *mutexes* are uppercase letters $A \ldots Z$;
- no thread attempts to lock a mutex it already owns;
- no thread attempts to unlock a mutex it does not own.

Input data: The first line of input file `MUT.IN` contains the number of threads M, ($1 \leq M \leq 5$) and is followed by M blocks describing each thread. The first line of a block describing thread i contains the number of instructions in this thread N_i, ($1 \leq N_i \leq 10$) and is followed by N_i lines with instructions. Instructions do not contain extraneous whitespace.

Output data: The first line of output file `MUT.OUT` must contain one number: D. D must be 1 if deadlocks are possible, or 0 if not.

If deadlocks are possible, the second line must describe a state of program in which a deadlock occurs. If there are several states with a deadlock, output any of them. In this case we are looking for complete deadlock, in which none of the threads can continue execution – a thread must be either terminated or blocked by a mutex. If deadlocks are not possible, the line must be empty.

State of program is described by specifying the zero-based index of current instruction for each thread in the order in which the threads are presented in input file. For a terminated thread, output −1 as the index. The indexes must be on a single line and separated by spaces.

Sample:

MUT.IN	MUT.OUT
2	1
1	-1 1
LOCK X	
2	
LOCK Y	
LOCK X	

The solution will not receive points for test cases, where there are no deadlocks, if it has not solved any test case, where deadlocks are possible.