Having studied mutual exclusion protocols in the previous year's competition you are now facing a more challenging problem. You have a big enterprise system with a number concurrently running processes. The system has several resources — databases, message queues, etc. Each concurrent process works with two resources at a time. For example, one process might copy a job from a particular database into the message queue, the other process might take a job from the message queue, perform the job, and then put the result into some other message queue, etc.

All resources are protected from concurrent access by mutual exclusion protocols also known as locks. For example, to access a particular database process acquires the lock for this database, then performs its work, then releases the lock. No two processes can hold the same lock at the same time (that is the property of mutual exclusion). Thus, the process that tries to acquire a lock *waits* if that lock is taken by some other process.

The main loop of the process that works with resources P and Q looks like this:

```
loop forever
  DoSomeNonCriticalWork()
  P.lock()
  Q.lock()
  WorkWithResourcesPandQ()
  Q.unlock()
  P.unlock()
end loop
```

The order in which locks for resources P and Q are taken is important. Consider a case where process c had acquired lock P with `P.lock()` and is waiting for lock Q in `Q.lock()`. It means that lock Q is taken by some other process d. If the process d is working (not waiting), then we say that there is a *wait chain* of length 1. If d had acquired lock Q and is waiting for another lock R, which is acquired by a working process e, then we say that there is a wait chain of length 2, etc. If any process in this *wait chain* waits for lock P that is already taken by process c, then we say that the wait chain has infinite length and the system *deadlocks*.

For this problem, we are interested only in alternating wait chains where processes hold their first locks and wait for the second ones. Formally:

> *Alternating wait chain* of length n ($n \geq 0$) is an alternating sequence of resources R_i ($0 \leq i \leq n+1$) and distinct processes c_i ($0 \leq i \leq n$): $R_0\ c_0\ R_1\ c_1 \ldots R_n\ c_n\ R_{n+1}$, where process c_i acquires locks for resources R_i and R_{i+1} in this order. Alternating wait chain is a deadlock when $R_0 = R_{n+1}$.

You are given a set of resources each process works with. Your task is to decide the order in which each process has to acquire its resource locks, so that the system never deadlocks and the maximum length of any possible alternating wait chain is minimized.

Input

The first line of the input file contains a single integer n ($1 \leq n \leq 100$) — the number of processes.

The following n lines describe resources that each process needs. Each resource is designated with an uppercase English letter from `L` to `Z`, so there are at most 15 resources. Each line describing process contains two different resources separated by a space.

Output

On first line of the output file write a singe integer number m — the minimally possible length of the maximal alternating wait chain.

Then write n lines — one line per process. On each line write two resources in the order they should be taken by the corresponding process to ensure this minimal length of the maximal alternating wait chain. Separate resources on a line by a space. If there are multiple satisfying orderings, then write any of them. The order of the processes in the output should correspond to their order in the input.

Sample Input

```
2
P Q
R S
6
P Q
Q R
R S
S T
T U
U P
4
P Q
P Q
P Q
P Q
3
P Q
Q R
R P
6
P Q
Q S
S R
R P
P S
R Q
```

Sample Output

```
0
P Q
R S
0
P Q
R Q
R S
T S
T U
P U
0
P Q
P Q
P Q
P Q
1
P Q
Q R
P R
2
P Q
Q S
R S
P R
P S
R Q
```