TRAINING COMPETITION FOR THE BULGARIAN EXTENDED NATIONAL TEAM
Bankya, 20 June 2025
Group A

Problem AT32. Sync

Author: Emil Indzhev

A group of N robots standing in a line are trying to fire a signal at the same time. The robots are quite limited – the only thing they remember is what state they are in. The only other thing they see is what states their neighbors' states. The only thing they can do is updating their state. The robots all run on the same clock, i.e. all of them update their states at the same time.

Formally, at each time step robot i ($1 \leq i \leq N$) is in some state S_i. Then it has to choose what state to transition to for the next time step. This can depend only on S_{i-1}, S_i and S_{i+1}. The robot may choose to stay in state S_i or transition to any other valid state. All robots share the same rule-set (i.e. the robot does not know its index i). All robots choose their next state based on the states of the robots at the current time step and then they all update their states simultaneously. The leftmost and rightmost robots see the fictitious state `X` on their left and right, respectively. I.e. we set S_0 and S_{N+1} to `X`. The robots cannot enter state `X`.

The robots are normally kept in the special built-in state `Wait`. They have hard-coded rules such that, if a robot is in state `Wait` and only sees `Wait` and/or `X` on each of its sides, it will stay in `Wait`.

Notice that, if all robots start in `Wait`, no progress will happen. Thus, at time step 0, all robots are in `Wait` except the leftmost one (i.e. robot 1), which is put in the built-in state `Init`. That state has no restrictions on how it is used (unlike `Wait`).

The robots will then keep updating their states on each timestep until some robot enters the special built-in state `Fire` (or until the time limit is up). The goal is for all robots to enter `Fire` on that same timestep. They win, if they do this, and lose, if only some robots enter `Fire`, or no robots enter `Fire` before the time is up.

Your job is to design a set of states and a rule-set for the robots so that they can synchronize and fire the signal at the same time. The rule-set should work for all allowed values of N within the time steps limit, which is $5N$. Additionally, you may not use more than 40 distinct states (not counting `X` and `Fire`). Finally, you should also try to further minimize the number of states used, as you will be scored based on that.

You will need to submit a `.txt` file with a list of transition rules to the grading system. You do not need to "declare" states – all states mentioned in the rules you list are implicitly declared. State names can be arbitrary sequences of non-whitespace printable ASCII characters except `/` and `#`. The file should have one rule per line (lines without rules are also allowed). Additionally, everything on a line after a `#` is considered a comment and is ignored. The rule format is as follows:

```
L M R -> E
```

This means that a robot in state `M` who sees their left neighbor in state `L` and their right neighbor in `R` transitions to state `E`. For convenience, for `L`, `M` or `R`, you can list multiple states separated by `/` (without spaces) or use `?` as a wildcard that matches with every possible state. Additionally, `L`, `M` and `R` may not be `Fire` (since the run ends as soon as a robot enters `Fire`) and `M` and `E` may not be `X` (since no robot may be in state `X`). Example:

```
? A_0/Y^/10+ X -> Fire
```

Here, the rightmost robot (since its right neighbor is in `X`), when in states `A_0`, `Y^` or `10+` will enter `Fire` regardless of its left neighbor (since the rule has `?`); this will end the run.

When there are multiple matching rules for some robot's situation, the top-most one (in the order in the rule-set) takes precedence. Example:

```
Wait ? B -> Wait
? C B/Init -> A
```

If the actual configuration is `Wait C B`, both rules would match, but the middle robot will enter state `Wait`, since that rule is first.

Note the built-in rule about `Wait`, which is implicitly listed before the rules you write in your rule set (i.e. has higher precedence):

```
X/Wait Wait X/Wait -> Wait
```

Also note that you do not need to define transition rules for all triples of states. Not defining a rule is only an issue, if that configuration actually occurs (in which case the verdict is Wrong Answer).

Here is an example of a full (but unsuccessful) rule-set ran with $N = 3$:

```
? Init ? -> Wait # Init always turns into a Wait
? Wait ? -> A' # Wait with a non-Wait neighbor turns into an A'

# Next are the transitions from A'
Wait A' Wait -> Wait
? A' X -> Init
Wait/X A' ? -> _B2
? A' ? -> A'

# Finally _B2 fires
X/_B2 _B2 A'/_B2 -> Fire
```

Running this rule-set yields the following sequence of robot states:

```
Init Wait Wait
Wait A' Wait
A' Wait A'
_B2 A' Init
Fire A' Wait
```

The run ends since a robot entered `Fire`. However, the robots lose since not all of them did.

Local testing

You are provided with a simulator that will read N and then your list of rules (until the end of input, which you can "type" with Ctrl+Z on Windows). The simulator will then run your rule-set on the given N until the run ends (due to a robot firing or due to the number of time steps exceeding the limit). It will print the sequence of states on each time step as well as the total number of time steps and the total number of states. You may modify the simulator's code however you want.

Constraints

- $2 \leq N \leq 2500$
- Number of time steps $\leq 5N$
- Number of states ≤ 40

Subtasks

Subtask	Points	N
1	7	≤ 5
2	6	≤ 15
3	14	$= 2^K$
4	14	$= 2^K + 1$
5	14	$= 2^K - 1$
6	14	$= 2^K - 2$
7	31	No restrictions

You get points for a given subtask only if your program passes all tests in it and all other subtasks included in its constraints.

Scoring

Subtasks 1 and 2 will not have partial scoring – you either get the full points or not. For all other subtasks, your score (the fraction of the points you receive) depends on the number of states you use (not counting `X` and `Fire`). Let's call that number M. Then your score S is:

- If $M \leq 7$: $S = 1$
- Otherwise: $S = \left(\frac{7}{M}\right)^{0.6}$