A - Group Commands and Wall Planning

Time Limit: 2 sec / Memory Limit: 1024 MiB

Problem Statement

There is an $N \times N$ grid. The coordinate of the top-left cell is $(0, 0)$, and the coordinate of the cell i rows down and j columns to the right is (i, j). There are walls between some adjacent cells.

There are K robots on the grid. The initial position of the k-th robot is (i_k, j_k), and its destination is (i'_k, j'_k). Takahashi wants to operate these robots to bring all of them to their respective destinations.

Before making any moves, the following two types of preparations can be made:

1. In addition to the existing walls, you may add walls between any adjacent cells.
2. Divide the robots into groups. Each robot belongs to exactly one group, and all robots in the same group can be operated simultaneously.

These preparations must be completed before the first move. After that, no additional walls can be placed and the group assignments cannot be changed.

Next, the robots can be moved by repeatedly performing the following two types of operations:

1. **Group Command**: Specify a group and a direction (up, down, left, or right). All robots in that group will attempt to move one cell in the specified direction. If there is a wall between the current and target cells, or if another robot is occupying the target cell, the robot does not move. Among the robots belonging to the group, those farthest in the direction of movement will attempt to move first. For example, if robots are at $(1, 0)$ and $(2, 0)$ and the direction is up, the robot at $(1, 0)$ will move to $(0, 0)$ first (since it has a smaller i coordinate), and then the robot at $(2, 0)$ will move to the now-empty $(1, 0)$.
2. **Individual Command**: Specify a robot and a direction (up, down, left, or right). The specified robot will attempt to move one cell in the specified direction. If there is a wall between the current and target cells, or if another robot is occupying the target cell, the robot does not move.

Even if a robot reaches its destination once, if it moves away from the destination due to subsequent operations, it is not considered to have reached its destination.

You may perform at most KN^2 operations. Guide all robots to their destinations using as few operations as possible.

Scoring

Let T be the number of operations performed, and let d_k be the Manhattan distance between the final position and the destination of robot k. Then, you obtain the following absolute score:

$$T + 100 \times \sum_k d_k$$

The lower the absolute score, the better.

For each test case, we compute the **relative score** $\mathrm{round}(10^9 \times \frac{\mathrm{MIN}}{\mathrm{YOUR}})$, where YOUR is your absolute score and MIN is the lowest absolute score among all competitors obtained on that test case. The score of the submission is the sum of the relative scores.

The final ranking will be determined by the system test with more inputs which will be run after the contest is over. In both the provisional/system test, if your submission produces illegal output or exceeds the time limit for some test cases, only the score for those test cases will be zero, and your submission will be excluded from the MIN calculation for those test cases.

The system test will be performed only for **the last submission which received a result other than** `CE`. Be careful not to make a mistake in the final submission.

Number of test cases

- Provisional test: 50
- System test: 2000. We will publish seeds.txt (https://img.atcoder.jp/awtf2025heuristic/seeds.txt) (sha256=063a84b1c0dc9388b0996eed0bc645529c931c139bee6b8e0e84a4faf3e06c40) after the contest is over.

About relative evaluation system

In both the provisional/system test, the standings will be calculated using only the last submission which received a result other than `CE`. Only the last submissions are used to calculate the MIN for each test case when calculating the relative scores.

The scores shown in the standings are relative, and whenever a new submission arrives, all relative scores are recalculated. On the other hand, the score for each submission shown on the submissions page is the sum of the absolute score for each test case, and the relative scores are not shown. In order to know the relative score of submission other than the latest one in the current standings, you need to resubmit it. If your submission produces illegal output or exceeds the time limit for some test cases, the score shown on the submissions page will be 0, but the standings show the sum of the relative scores for the test cases that were answered correctly.

About execution time

Execution time may vary slightly from run to run. In addition, since system tests simultaneously perform a large number of executions, it has been observed that execution time increases by several percent compared to provisional tests. For these reasons, submissions that are very close to the time limit may result in `TLE` in the system test. Please measure the execution time in your program to terminate the process, or have enough margin in the execution time.

Input

Input is given from Standard Input in the following format.

$$
\begin{array}{l}
N \quad K \\
i_0 \quad j_0 \quad i'_0 \quad j'_0 \\
\vdots \\
i_{K-1} \quad j_{K-1} \quad i'_{K-1} \quad j'_{K-1} \\
v_{0,0} \cdots v_{0,N-2} \\
\vdots \\
v_{N-1,0} \cdots v_{N-1,N-2} \\
h_{0,0} \cdots h_{0,N-1} \\
\vdots \\
h_{N-2,0} \cdots h_{N-2,N-1}
\end{array}
$$

- In all test cases, $N = 30$ is fixed.
- $10 \leq K \leq 100$
- (i_k, j_k) represents the initial position of the k-th robot.
- (i'_k, j'_k) represents the destination of the k-th robot.
- All initial positions and all destinations are each mutually distinct, but the initial position of robot k may coincide with the destination of robot k'.
- Each $v_{i,0} \cdots v_{i,N-2}$ is a binary string of length $N-1$. The j-th character $v_{i,j}$ indicates whether there is a wall (`1`) or not (`0`) between cells (i, j) and $(i, j+1)$.
- Each $h_{i,0} \cdots h_{i,N-1}$ is a binary string of length N. The j-th character $h_{i,j}$ indicates whether there is a wall (`1`) or not (`0`) between cells (i, j) and $(i+1, j)$.
- It is guaranteed that all cells are mutually reachable.

Output

First, output the wall placement information in the following format to Standard Output.

$$
\begin{array}{l}
v'_{0,0} \cdots v'_{0,N-2} \\
\vdots \\
v'_{N-1,0} \cdots v'_{N-1,N-2} \\
h'_{0,0} \cdots h'_{0,N-1} \\
\vdots \\
h'_{N-2,0} \cdots h'_{N-2,N-1}
\end{array}
$$

- Each $v'_{i,0} \cdots v'_{i,N-2}$ is a binary string of length $N-1$. The j-th character $v'_{i,j}$ indicates whether a wall is placed (`1`) or not (`0`) between cells (i,j) and $(i,j+1)$.
- Each $h'_{i,0} \cdots h'_{i,N-1}$ is a binary string of length N. The j-th character $h'_{i,j}$ indicates whether a wall is placed (`1`) or not (`0`) between cells (i,j) and $(i+1,j)$.
- For positions where a wall already exists, either `0` or `1` may be output.

Next, output the group assignment information to Standard Output in the following format.

$$
g_0 \quad \cdots \quad g_{K-1}
$$

- g_k is an integer between 0 and $K-1$ representing the group to which the k-th robot belongs. If $g_k = g_{k'}$, then robot k and robot k' belong to the same group.

Finally, output the sequence of operations to Standard Output in the following format.

$$
\begin{array}{l}
a_0 \quad b_0 \quad d_0 \\
\vdots \\
a_{T-1} \quad b_{T-1} \quad d_{T-1}
\end{array}
$$

- a_t is a single character specifying the type of operation on turn t. Use `g` for a group command and `i` for an individual command.
- b_t is an integer between 0 and $K-1$ representing the group number or robot number targeted by the operation on turn t. If a group with no robots is specified, nothing happens.
- d_t is a single character indicating the direction of the operation on turn t, as follows:
  - Up: `U`
  - Down: `D`
  - Left: `L`
  - Right: `R`

Show example (https://img.atcoder.jp/awtf2025heuristic/sJKH3KO4.html?lang=en&seed=0&output=sample)

Input Generation

Let $\mathrm{rand}(L, U)$ be a function that generates a uniformly random integer between L and U, inclusive.

Robot Generation

The number of robots K is determined by $K = \mathrm{rand}(10, 100)$.
The initial positions of the robots are chosen by selecting K distinct coordinates uniformly at random from the N^2 cells.
The destinations of the robots are similarly chosen by selecting K distinct coordinates uniformly at random from the N^2 cells.

Wall Generation

The number of wall segments W is determined by $W = \mathrm{rand}(0, 2)$.

Repeat the following W times:

1. Randomly choose the direction of the wall from up, down, left, or right.
2. Determine the wall length $L = \mathrm{rand}(10, 20)$.
3. For vertical walls, choose the starting point (i, j) by $i = \mathrm{rand}(5, N - 5),\ j = \mathrm{rand}(4, N - 6)$.
   If the chosen j is within an absolute distance of 4 from any j used in previously generated vertical walls, redo the direction selection.
   For upward walls, set $v_{i-L+1,j}, \cdots, v_{i,j}$ to 1. For downward walls, set $v_{i,j}, \cdots, v_{i+L-1,j}$ to 1.
   Ignore any part that goes out of bounds.
4. For horizontal walls, choose the starting point (i, j) by $i = \mathrm{rand}(4, N - 6),\ j = \mathrm{rand}(5, N - 5)$.
   If the chosen i is within an absolute distance of 4 from any i used in previously generated horizontal walls, redo the direction selection.
   For leftward walls, set $h_{i,j-L+1}, \cdots, h_{i,j}$ to 1. For rightward walls, set $h_{i,j}, \cdots, h_{i,j+L-1}$ to 1.
   Ignore any part that goes out of bounds.
5. After generating the wall, check whether all cells are still mutually reachable. If not, reset the wall and restart the W iterations.

Tools (Input generator and visualizer)

- Web version (https://img.atcoder.jp/awtf2025heuristic/sJKH3KO4.html?lang=en): This is more powerful than the local version providing animations and manual play.
- Local version (https://img.atcoder.jp/awtf2025heuristic/sJKH3KO4.zip): You need a compilation environment of Rust language (https://www.rust-lang.org/).
    - Pre-compiled binary for Windows (https://img.atcoder.jp/awtf2025heuristic/sJKH3KO4_windows.zip): If you are not familiar with the Rust language environment, please use this instead.

Please be aware that sharing visualization results or discussing solutions/ideas during the contest is prohibited.

Sample Input 1

```
30 59
8 18 11 17
8 23 17 15
1 12 11 7
19 17 24 22
23 9 8 20
21 17 26 10
4 19 12 29
19 29 7 7
19 24 12 19
```

Sample Output 1

```
00000000000000000000000000000
01000000000000000000000000000
00000000000000000000000000000
00000000000000000000001000000
00000000000100000000000000000
00000000000000000000000000001
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
```