Robot Contest

AI researchers at the University of Szeged are holding a robot programming contest. Your friend, Hanga, has decided to take part in the contest. The objective is to program the ultimate *Pulibot*, admiring the great intelligence of the famous Hungarian herding dog breed, the Puli.

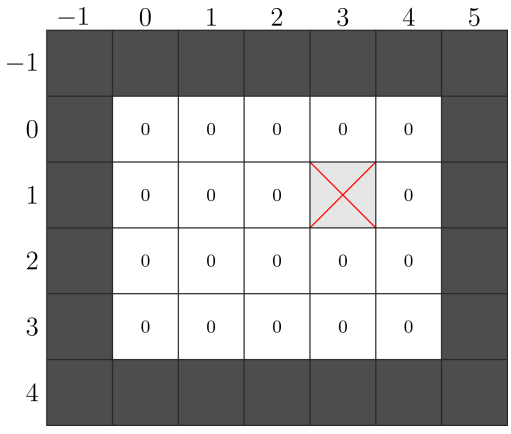
Pulibot will be tested on a maze consisting of a $(H+2) \times (W+2)$ grid of cells. The rows of the grid are numbered from -1 to H from north to south and the columns of the grid are numbered from -1 to W from west to east. We refer to the cell located at row r and column c of the grid ($-1 \leq r \leq H$, $-1 \leq c \leq W$) as cell (r, c).

Consider a cell (r, c) such that $0 \leq r < H$ and $0 \leq c < W$. There are 4 cells **adjacent** to cell (r, c):

- cell $(r, c-1)$ is referred to as the cell **west** of cell (r, c);
- cell $(r+1, c)$ is referred to as the cell **south** of cell (r, c);
- cell $(r, c+1)$ is referred to as the cell **east** of cell (r, c);
- cell $(r-1, c)$ is referred to as the cell **north** of cell (r, c).

Cell (r, c) is called a **boundary** cell of the maze if $r = -1$ or $r = H$ or $c = -1$ or $c = W$ holds. Each cell that is not a boundary cell of the maze is either an **obstacle** cell or an **empty** cell. Additionally, each empty cell has a **color**, represented by a nonnegative integer between 0 and Z_{MAX}, inclusive. Initially, the color of each empty cell is 0.

For example, consider a maze with $H = 4$ and $W = 5$, containing a single obstacle cell $(1, 3)$:



The only obstacle cell is denoted by a cross. Boundary cells of the maze are shaded. The number in each empty cell represents its color.

A **path** of length ℓ ($\ell > 0$) from cell (r_0, c_0) to cell (r_ℓ, c_ℓ) is a sequence of pairwise distinct *empty* cells $(r_0, c_0), (r_1, c_1), \ldots, (r_\ell, c_\ell)$ in which for each i ($0 \leq i < \ell$) the cells (r_i, c_i) and (r_{i+1}, c_{i+1}) are adjacent.

Note that a path of length ℓ contains exactly $\ell + 1$ cells.

At the contest, the researchers set up a maze in which there exists at least one path from cell $(0, 0)$ to cell $(H-1, W-1)$. Note that this implies that cells $(0, 0)$ and $(H-1, W-1)$ are guaranteed to be empty.

Hanga does not know which cells of the maze are empty and which cells are obstacles.

Your task is to help Hanga to program Pulibot so that it is capable of finding a *shortest path* (that is, a path of minimum length) from cell $(0, 0)$ to cell $(H-1, W-1)$ in the unknown maze set up by the researchers. The specification of Pulibot and the rules of the contest are described below.

Note that the last section of this problem statement describes a display tool you can use to visualize Pulibot.

Pulibot's Specification

Define the **state** of a cell (r, c) for each $-1 \leq r \leq H$ and $-1 \leq c \leq W$ as an integer so that:

- if cell (r, c) is a boundary cell then its state is -2;
- if cell (r, c) is an obstacle cell then its state is -1;
- if cell (r, c) is an empty cell then its state is the color of the cell.

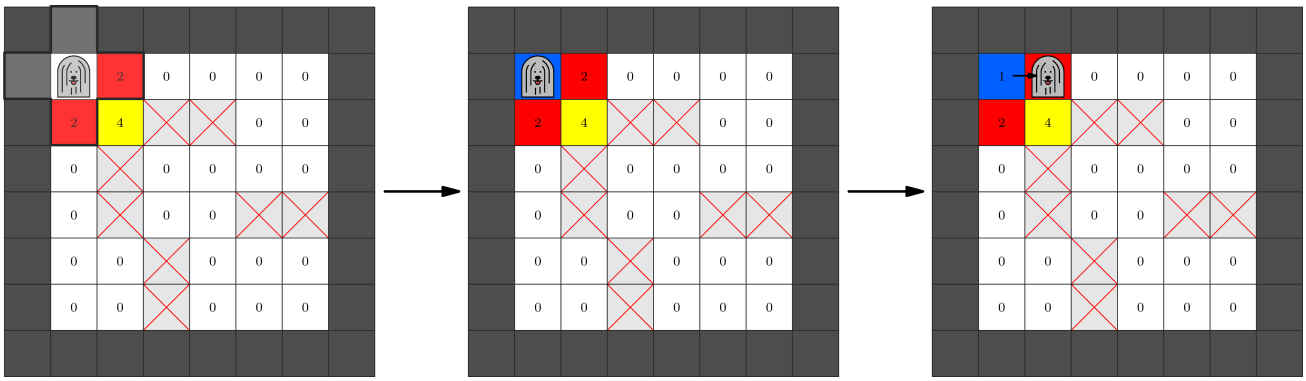
Pulibot's program is executed as a sequence of steps. In each step, Pulibot recognizes the states of nearby cells and then performs an instruction. The instruction it performs is determined by the recognized states. A more precise description follows.

Suppose that at the beginning of the current step, Pulibot is at cell (r, c), which is an empty cell. The step is performed as follows:

1. First, Pulibot recognizes the current **state array**, that is, the array $S = [S[0], S[1], S[2], S[3], S[4]]$, consisting of the state of cell (r, c) and of all adjacent cells:
    - $S[0]$ is the state of cell (r, c).
    - $S[1]$ is the state of the cell to the west.
    - $S[2]$ is the state of the cell to the south.
    - $S[3]$ is the state of the cell to the east.
    - $S[4]$ is the state of the cell to the north.
2. Then, Pulibot determines the **instruction** (Z, A) which corresponds to the recognized state array.
3. Finally, Pulibot performs that instruction: it sets the color of cell (r, c) to color Z and then it performs action A, which is one of the following actions:
    - *stay* at cell (r, c);

- *move* to one of the 4 adjacent cells;
- *terminate the program*.

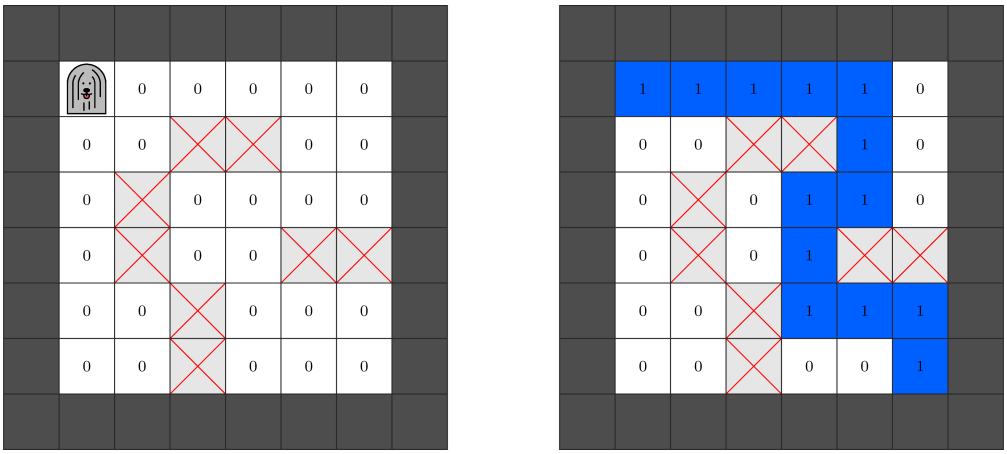
For example, consider the scenario displayed on the left of the following figure. Pulibot is currently at cell $(0, 0)$ with the color 0. Pulibot recognizes the state array $S = [0, -2, 2, 2, -2]$. Pulibot may have a program which, upon recognizing this array, sets the color of the current cell to $Z = 1$ and then moves to the east, as displayed in the middle and on the right of the figure:



Robot Contest Rules

- At the start, Pulibot is placed at cell $(0, 0)$ and begins to execute its program.
- Pulibot is not allowed to move to a cell which is not empty.
- Pulibot's program must terminate after at most $500\,000$ steps.
- After the termination of Pulibot's program, empty cells in the maze should be colored such that:
  - There exists a shortest path from $(0, 0)$ to $(H - 1, W - 1)$ for which the color of each cell included in the path is 1.
  - All other empty cells have color 0.
- Pulibot may terminate its program at any empty cell.

For example, the following figure shows a possible maze with $H = W = 6$. The starting configuration is displayed on the left and one acceptable coloring of empty cells after termination is displayed on the right:



Implementation Details

You should implement the following procedure.

```
void program_pulibot()
```

- This procedure should produce Pulibot's program. This program should work correctly for all values of H and W and any maze which meets the task constraints.
- This procedure is called exactly once for each test case.

This procedure can make calls to the following procedure to produce Pulibot's program:

```
void set_instruction(int[] S, int Z, char A)
```

- S: array of length 5 describing a state array.
- Z: a nonnegative integer representing a color.
- A: a single character representing an action of Pulibot as follows:
  - `H`: stay;
  - `W`: move to the west;
  - `S`: move to the south;
  - `E`: move to the east;
  - `N`: move to the north;
  - `T`: terminate the program.
- Calling this procedure instructs Pulibot that upon recognizing the state array S it should perform the instruction (Z, A).

Calling this procedure multiple times with the same state array S will result in an `Output isn't correct` verdict.

It is not required to call `set_instruction` with each possible state array S. However, if Pulibot later recognizes a state array for which an instruction was not set, you will get an `Output isn't correct` verdict.

After `program_pulibot` completes, the grader invokes Pulibot's program over one or more mazes. These invocations do *not* count towards the time limit for your solution. The grader is *not* adaptive, that is, the set of mazes is predefined in each test case.

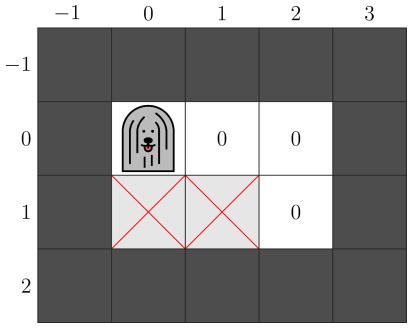
If Pulibot violates any of the Robot Contest Rules before terminating its program, you will get an `Output isn't correct` verdict.

Example

The procedure `program_pulibot` may make calls to `set_instruction` as follows:

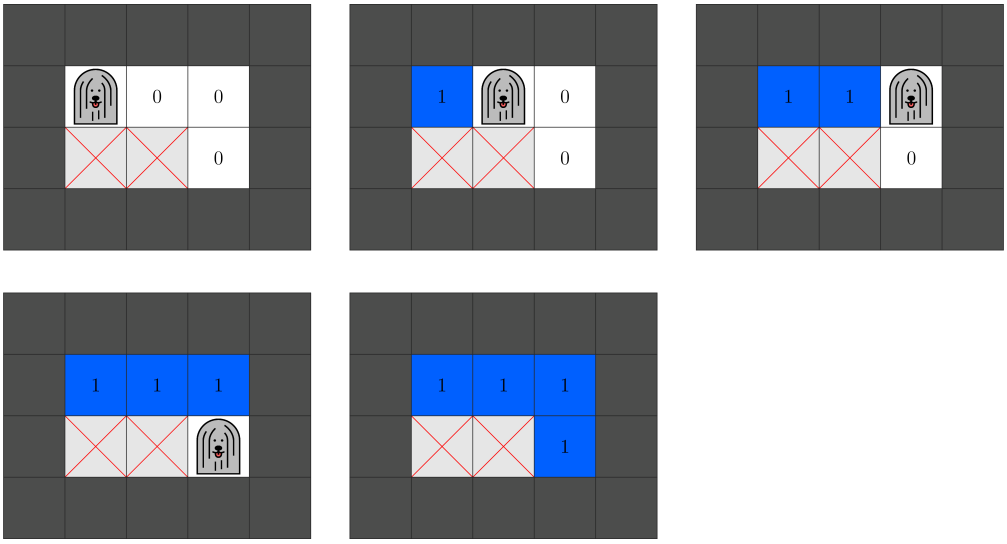
Call	Instruction for state array S
`set_instruction([0, -2, -1, 0, -2], 1, E)`	Set color to 1 and move east
`set_instruction([0, 1, -1, 0, -2], 1, E)`	Set color to 1 and move east
`set_instruction([0, 1, 0, -2, -2], 1, S)`	Set color to 1 and move south
`set_instruction([0, -1, -2, -2, 1], 1, T)`	Set color to 1 and terminate program

Consider a scenario where $H = 2$ and $W = 3$, and the maze is displayed in the following figure.



For this particular maze Pulibot's program runs in four steps. The state arrays Pulibot recognizes and the instructions it performs correspond exactly to the four calls to `set_instruction` made above, in order. The last of these instructions terminates the program.

The following figure shows the maze before each of the four steps and the final colors after termination.



However, do note that this program of 4 instructions might not find a shortest path in other valid mazes. Therefore, if submitted, it will receive an `Output isn't correct` verdict.

Constraints

$Z_{MAX} = 19$. Hence, Pulibot can use colors from 0 to 19, inclusive.

For each maze used to test Pulibot:

- $2 \leq H, W \leq 15$
- There is at least one path from cell $(0, 0)$ to cell $(H-1, W-1)$.

Subtasks

1. (6 points) There is no obstacle cell in the maze.
2. (10 points) $H = 2$
3. (18 points) There is exactly one path between each pair of empty cells.
4. (20 points) Each shortest path from cell $(0, 0)$ to cell $(H-1, W-1)$ has length $H + W - 2$.
5. (46 points) No additional constraints.

If, in any of the test cases, the calls to the procedure `set_instruction` or Pulibot's program over its execution do not conform to the constraints described in Implementation Details, the score of your solution for that subtask will be 0.

In each subtask, you can obtain a partial score by producing a coloring that is almost correct.

Formally:

- The solution of a test case is **complete** if the final coloring of the empty cells satisfies Robot Contest Rules.
- The solution of a test case is **partial** if the final coloring looks as follows:
  - There exists a shortest path from $(0, 0)$ to $(H-1, W-1)$ for which the color of each cell included in the path is 1.
  - There is no other empty cell in the grid with color 1.
  - Some empty cell in the grid has a color other than 0 and 1.

If your solution to a test case is neither complete nor partial, your score for the corresponding test case will be 0.

In subtasks 1-4, the score for a complete solution is 100% and the score for a partial solution to a test case is 50% of the points for its subtask.

In subtask 5, your score depends on the number of colors used in Pulibot's program. More precisely, denote by $Z^\star$ the maximum value of Z over all calls made to `set_instruction`. The score of the test case is calculated according to the following table:

Condition	Score (complete)	Score (partial)
$11 \leq Z^\star \leq 19$	$20 + (19 - Z^\star)$	$12 + (19 - Z^\star)$
$Z^\star = 10$	31	23
$Z^\star = 9$	34	26
$Z^\star = 8$	38	29
$Z^\star = 7$	42	32
$Z^\star \leq 6$	46	36

The score for each subtask is the minimum of the points for the test cases in the subtask.

Sample Grader

The sample grader reads the input in the following format:

- line 1: H W
- line $2 + r$ ($0 \leq r < H$): $m[r][0]$ $m[r][1]$ $\ldots$ $m[r][W-1]$

Here, m is an array of H arrays of W integers, describing the non-boundary cells of the maze. $m[r][c] = 0$ if cell (r, c) is an empty cell and $m[r][c] = 1$ if cell (r, c) is an obstacle cell.

The sample grader first calls `program_pulibot()`. If the sample grader detects a protocol violation, the sample grader prints `Protocol Violation: <MSG>` and terminates, where `<MSG>` is one of the following error messages:

- `Invalid array`: $-2 \leq S[i] \leq Z_{MAX}$ is not met for some i or the length of S is not 5.
- `Invalid color`: $0 \leq Z \leq Z_{MAX}$ is not met.
- `Invalid action`: character A is not one of `H`, `W`, `S`, `E`, `N` or `T`.
- `Same state array`: `set_instruction` was called with the same array S at least twice.

Otherwise, when `program_pulibot` completes, the sample grader executes Pulibot's program in the maze described by the input.

The sample grader produces two outputs.

First, the sample grader writes a log of Pulibot's actions to the file `robot.bin` in the working directory. This file serves as the input of the visualization tool described in the following section.

Second, if Pulibot's program does not terminate successfully, the sample grader prints one of the following error messages:

- `Unexpected state`: Pulibot recognized a state array which `set_instruction` was not called with.

- `Invalid move`: performing an action resulted in Pulibot moving to a nonempty cell.
- `Too many steps`: Pulibot performed $500\,000$ steps without terminating its program.

Otherwise, let $e[r][c]$ be the state of cell (r, c) after Pulibot's program terminates. The sample grader prints H lines in the following format:

- Line $1 + r$ ($0 \leq r < H$): $e[r][0]$ $e[r][1]$ $\ldots$ $e[r][W-1]$

Display Tool

The attachment package for this task contains a file named `display.py`. When invoked, this Python script displays Pulibot's actions in the maze described by the input of the sample grader. For this, the binary file `robot.bin` must be present in the working directory.

To invoke the script, execute the following command.

```
python3 display.py
```

A simple graphical interface shows up. The main features are as follows:

- You can observe the status of the full maze. The current location of Pulibot is highlighted by a rectangle.
- You can browse through the steps of Pulibot by clicking the arrow buttons or pressing their hotkeys. You can also jump to a specific step.
- The upcoming step in Pulibot's program is shown at the bottom. It shows the current state array and the instruction it will perform. After the final step, it shows either one of the error messages of the grader, or `Terminated` if the program successfully terminates.
- To each number that represents a color, you can assign a visual background color, as well as a display text. The display text is a short string that shall appear in each cell having that color. You can assign background colors and display texts in either of the following ways:
  - Set them in a dialog window after clicking on the `Colors` button.
  - Edit the contents of the `colors.txt` file.
- To reload `robot.bin`, use the `Reload` button. It is useful if the contents of `robot.bin` have changed.