مسابقه ربات

متخصصان هوش‌مصنوعی در دانشگاه سگد در حال برگزاری یک مسابقه برنامه‌نویسی روبات هستند. دوست شما، هانگا، تصمیم به شرکت در این مسابقه گرفته‌است. هدف، برنامه‌نویسی برای روبات پیشرفته *پولیبات* است، که نام آن از روی *پولی*، نژاد سگ باهوش مجارستانی گرفته شده‌است.

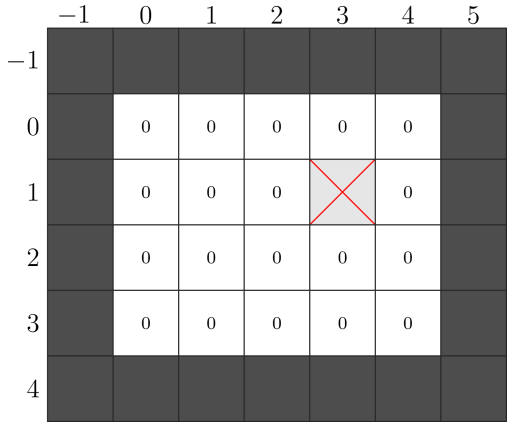
پولیبات بر روی یک ماز تشکیل شده از $(H+2) \times (W+2)$ خانه جدولی قرار دارد. سطرهای جدول از -1 تا H از شمال تا جنوب و ستون‌های آن از -1 تا W از غرب تا شرق شماره‌گذاری شده‌اند. خانه قرار گرفته در سطر r و ستون c از جدول را به شکل (r, c) نمایش می‌دهیم. $(-1 \leq r \leq H, -1 \leq c \leq W)$

خانه‌ی (r, c) را در نظر بگیرید به طوری که $0 \leq r < H$ and $0 \leq c < W$ برقرار باشد. 4 خانه به عنوان همسایه‌ی این خانه محسوب می‌شوند:

- خانه‌ی $(r, c-1)$ نشان‌دهنده‌ی همسایه‌ی **غربی** خانه‌ی (r, c) می‌باشد.
- خانه‌ی $(r+1, c)$ نشان‌دهنده‌ی همسایه‌ی **جنوبی** خانه‌ی (r, c) می‌باشد.
- خانه‌ی $(r, c+1)$ نشان‌دهنده‌ی همسایه‌ی **شرقی** خانه‌ی (r, c) می‌باشد.
- خانه‌ی $(r-1, c)$ نشان‌دهنده‌ی همسایه‌ی **شمالی** خانه‌ی (r, c) می‌باشد.

به خانه‌ی (r, c) می‌گوییم **مرزی** اگر حداقل یکی از شروط $r = -1$ یا $r = H$ یا $c = -1$ یا $c = W$ را داشته باشد. هر خانه‌ای که مرزی نباشد، یا **مانع** است و یا **خالی**. علاوه بر آن، هر خانه‌ی خالی یک **رنگ** دارد که با یک عدد صحیح نامنفی نشان داده می‌شود و مقدار آن از 0 تا Z_{MAX} می‌باشد.

برای مثال، یک ماز با $H = 4$ و $W = 5$ را در نظر بگیرید که تنها یک مانع در خانه $(1, 3)$ وجود دارد.



تک خانه‌ی مانع با ضربدر نشان داده شده است. خانه‌های مرزی محو شده‌اند. شماره‌ی داخل هر خانه نشان دهنده‌ی رنگ آن خانه می‌باشد.

یک **مسیر** به طول ℓ $(\ell > 0)$ از (r_0, c_0) به (r_ℓ, c_ℓ) یک دنباله متمایز از خانه‌های خالی $(r_0, c_0), (r_1, c_1), \ldots, (r_\ell, c_\ell)$ می‌باشد که در آن به ازای هر i $(0 \leq i < \ell)$ خانه‌های (r_i, c_i) و (r_{i+1}, c_{i+1}) مجاورند.

توجه کنید که یک مسیر به طول ℓ شامل دقیقا $\ell + 1$ خانه است.

هنگام مسابقه، متخصصان یک ماز طراحی می‌کنند که در آن حداقل یک مسیر از خانه $(0, 0)$ به خانه $(H-1, W-1)$ وجود دارد. توجه کنید که این یعنی تضمین می‌شود خانه‌های $(0, 0)$ و $(H-1, W-1)$ خالی هستند.

هانگا نمی‌داند کدام خانه‌ها مانع و کدام خانه‌ها خالی هستند.

شما باید به هانگا کمک کنید طوری پولیبات را برنامه ریزی کند که قادر به پیدا کردن یک **کوتاه‌ترین مسیر** (یعنی مسیر با کمترین طول ممکن) از خانه $(0, 0)$ به $(H-1, W-1)$ در ماز ناشناخته طراحی شده توسط متخصصین باشد. مشخصات پولیبات و قوانین مسابقه در زیر توضیح داده شده است.

توجه کنید که آخرین بخش این سوال توضیحات برنامه نمایشی برای ترسیم پولیبات می‌باشد.

مشخصات پولیبات

وضعیت خانه‌ی (r, c) را به ازای هر $-1 \leq r \leq H$ و $-1 \leq c \leq W$ با یک عدد صحیح با صورت زیر تعریف می‌شود:

- اگر خانه‌ی (r, c) مرزی باشد، وضعیتش -2 است.
- اگر خانه‌ی (r, c) مانع باشد، وضعیتش -1 است.
- اگر خانه‌ی (r, c) خالی باشد، وضعیتش همان رنگ خانه است.

برنامه پولیبات به صورت دنباله‌ای از مراحل اجرا می‌شود. در هر مرحله، پولیبات وضعیت خانه‌های مجاور را مشاهده می‌کند و سپس دستوری را انجام می‌دهد. دستورالعملی که انجام می‌دهد بسته به وضعیت مشاهده شده تعیین می‌شود. توضیحات بیشتر در ادامه آمده‌است.

فرض کنید در ابتدای مرحله فعلی، پولیبات در خانه (r, c) قرار دارد، که این خانه خالی است. اجرای این مرحله به شکل زیر است:

1- در ابتدا، پولیبات **state array** فعلی را مشاهده میکند. یعنی یک آرایه $S = [S[0], S[1], S[2], S[3], S[4]]$ متشکل از وضعیت خانه (r, c) و خانه‌های مجاور آن:

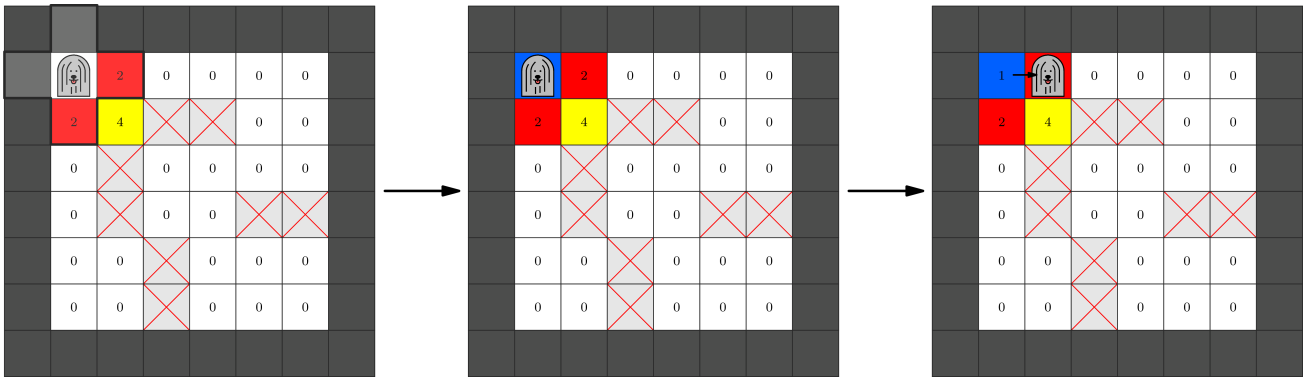
- $S[0]$ وضعیت خانه (r, c) است.
- $S[1]$ وضعیت خانه همسایه غربی است.
- $S[2]$وضعیت خانه همسایه جنوبی است.
- $S[3]$وضعیت خانه همسایه شرقی است.
- $S[4]$وضعیت خانه همسایه شمالی است.

2- سپس، پولیبات دستور (Z, A) که متناظر state array است را تعیین میکند.

3- در نهایت، پولیبات آن دستور را اجرا می‌کند: یعنی رنگ خانه (r, c) را به رنگ Z تغییر داده، و سپس عملیات A را که یکی از انواع زیر است، اجرا می‌کند:

- **stay** در خانه (r, A) بمان و کاری نکن.
- **move** به یکی از 4 خانه همسایه برو.
- **terminate** اجرای برنامه را متوقف کن.

برای مثال، سناریویی که تصویر چپ نشان می‌دهد را فرض کنید. پولیبات در حال حاضر در خانه‌ی $(0, 0)$ قرار دارد و رنگ آن 0 است. پولیبات state array را $S = [0, -2, 2, 2, -2]$ مشاهده می‌کند. پولیبات ممکن است برنامه‌ای داشته باشد که با تشخیص این آرایه، رنگ خانه‌ی کنونی را به $Z = 1$ تغییر دهد و سپس به سمت شرق حرکت کند. در تصویر راست و وسط نمایش داده شده است.

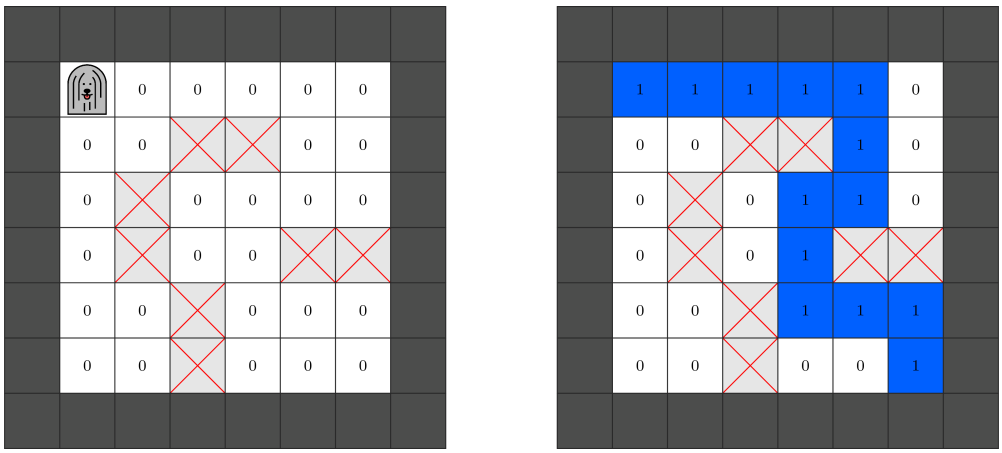


قوانین مسابقه ربات

- در ابتدا، پولیبات در خانه‌ی $(0, 0)$ قرار دارد و اجرا کردن برنامه‌اش را شروع می‌کند.
- پولیبات حق ندارد به خانه‌ی خالی حرکت کند.
- برنامه‌ی پولیبات بعد از حداکثر $500\,000$ حرکت باید متوقف شود.
- بعد از متوقف شدن برنامه‌ی پولیبات، خانه‌های خالی ماز باید به صورت زیر رنگ شده باشند:

- مسیری وجود داشته باشد که کوتاه ترین مسیر ممکن باشد و از خانه‌ی $(0, 0)$ به خانه‌ی $(H - 1, W - 1)$ برود و رنگ تمام خانه‌های این مسیر 1 باشد.
  - سایر خانه‌ها، همه‌شان رنگ 0 باشند.
  - پولیبات برنامه‌ی خود را ممکن است در هر خانه‌ی خالی‌ای متوقف کند.

برای مثال، تصویر زیر یک ماز امکان پذیر با $H = W = 6$ را نشان می‌دهد. وضعیت اولیه در تصویر چپ و یکی از رنگ آمیزی‌های قابل قبول که بعد از توقف شکل گرفته، در سمت راست نشان داده می‌شود.



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)
```

array of length 5 describing a state array. * Z: a nonnegative integer representing a color. * A: :S * :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 * .(Z, A) the instruction

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

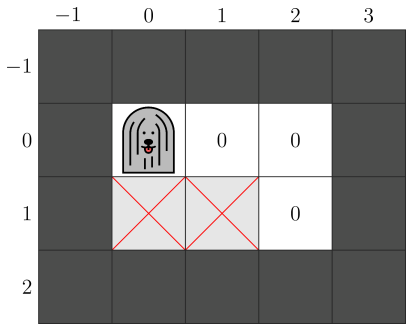
Example

: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

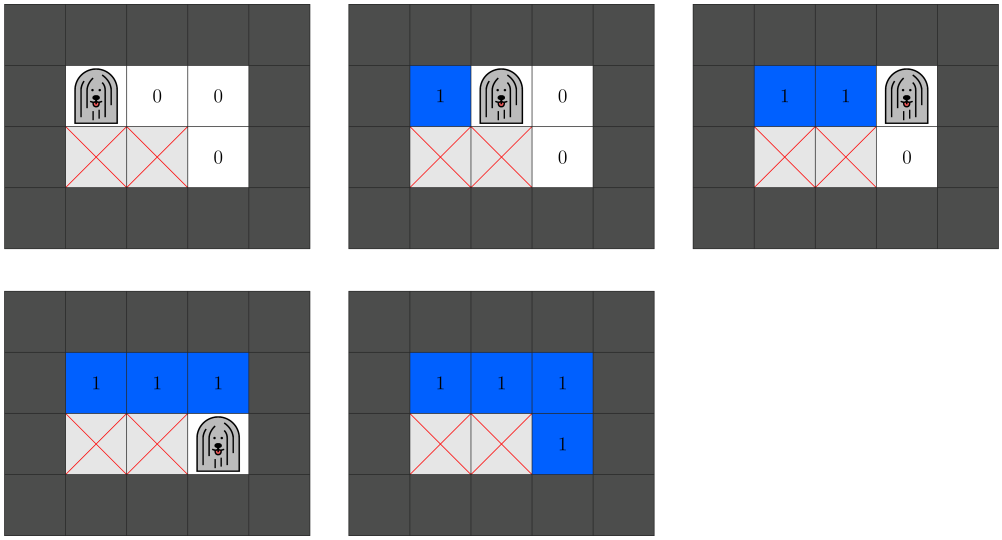
`set_instruction([0, 1, -1, 0, -2], 1, E)` | Set color to 1 and move east
`set_instruction([0, 1, 0, -2, -2], 1, S)` | Set color to 1 and move south
`set_instruction([0, -1, -2, -2, 1], 1, T)` | Set color to 1 and terminate program

.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

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

For each maze used to test Pulibot: * $2 \leq H, W \leq 15$ * There is at least one path from cell $(0, 0)$.$(H - 1, W - 1)$ to cell

Subtasks

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

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 .0 your solution for that subtask will be

.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 .1 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

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

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

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

Sample Grader

:$(0 \leq r < H)$ $2 + r$ The sample grader reads the input in the following format: * line 1: H W * line $m[r][0]$ $m[r][1]$ $\ldots$ $m[r][W-1]$

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 :$(0 \leq r < H)$ $1 + r$ grader prints H lines in the following format: * Line $e[r][0]$ $e[r][1]$ $\ldots$ $e[r][W-1]$

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