Concurso de Robots

Investigadores de IA en la Universidad de Szeged celebran un concurso de programación de robots. Tu amigo Hanga ha decidido participar en el concurso. El objetivo es programar el mejor *Pulibot*, cuyo nombre admira la gran inteligencia de la raza Puli de perros pastores húngaros.

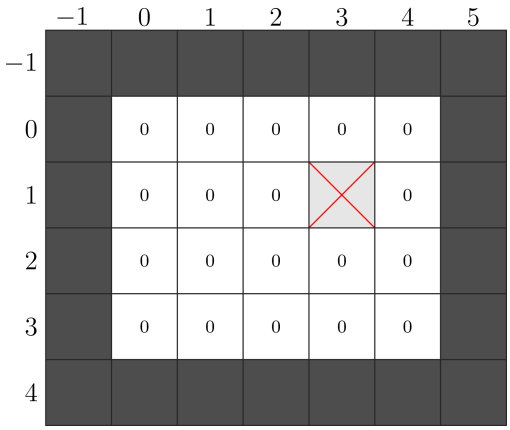
Pulibot será probado en un laberinto consistente en una cuadrícula de $(H+2) \times (W+2)$ casillas. Las filas de la cuadrícula están numeradas de -1 a H de norte a sur y las columnas de la cuadrícula están numeradas de -1 a W de oeste a este. Nos referimos a la casilla en la fila r y la columna c de la cuadrícula ($-1 \leq r \leq H$, $-1 \leq c \leq W$) como casilla (r, c).

Considera una casilla (r, c) tal que $0 \leq r < H$ y $0 \leq c < W$. Hay 4 casillas **adyacentes** a la casilla (r, c):

- A la casilla $(r, c-1)$ se la llama la casilla al **oeste** de la casilla (r, c);
- A la casilla $(r+1, c)$ se la llama la casilla al **sur** de la casilla (r, c);
- A la casilla $(r, c+1)$ se la llama la casilla al **este** de la casilla (r, c);
- A la casilla $(r-1, c)$ se la llama la casilla al **norte** de la casilla (r, c).

La casilla (r, c) se dice que es una casilla **frontera** del laberinto si $r = -1$ o $r = H$ o $c = -1$ o $c = W$. Cada casilla que no es una casilla frontera del laberinto es una casilla **obstáculo** o una casilla **vacía**. Adicionalmente, cada casilla vacía tiene un **color**, representado por un entero entre 0 y Z_{MAX}, inclusive. Inicialmente, el color de cada casilla vacía es 0.

Por ejemplo, considera un laberinto con $H = 4$ y $W = 5$ que contiene una única casilla obstáculo $(1, 3)$:



La casilla obstáculo está marcada con una cruz. Las casillas frontera del laberinto están sombreadas. Los números escritos en cada casilla vacía representan su color.

Un **camino** de longitud ℓ ($\ell > 0$) desde la casilla (r_0, c_0) a la casilla (r_ℓ, c_ℓ) es una secuencia de casillas *vacías* distintas $(r_0, c_0), (r_1, c_1), \ldots, (r_\ell, c_\ell)$ en la que para todo i ($0 \leq i < \ell$) las casillas (r_i, c_i) y (r_{i+1}, c_{i+1}) son adyacentes.

Fíjate en que un camino de longitud ℓ contiene exactamente $\ell + 1$ casillas.

En el concurso, los investigadores montan un laberinto en el que existe al menos un camino de la casilla $(0, 0)$ a la casilla $(H - 1, W - 1)$. Fíjate en que esto implica que se garantiza que las casillas $(0, 0)$ y $(H - 1, W - 1)$ están vacías.

Hanga no sabe qué casillas del laberinto están vacías y cuáles son obstáculos.

Tu tarea es ayudar a Hanga a programar el Pulibot de forma que sea capaz de encontrar un *camino más corto* (es decir, un camino de longitud mínima) desde la casilla $(0, 0)$ a la casilla $(H - 1, W - 1)$ en el laberinto desconocido que monten los investigadores. La especificación de Pulibot y las reglas del concurso están descritas abajo.

Nótese que la última sección de este enunciado describe una herramienta de visualización que puedes usar para ver el Pulibot.

Especificación del Pulibot

Definimos el **estado** de la casilla (r, c) para cada $-1 \leq r \leq H$ y $-1 \leq c \leq W$ como un entero que:

- si la casilla (r, c) es una casilla de la frontera su estado es -2;
- si la casilla (r, c) es una casilla obstáculo su estado es -1;
- si la casilla (r, c) es una casilla vacía entonces el estado es el color de la casilla.

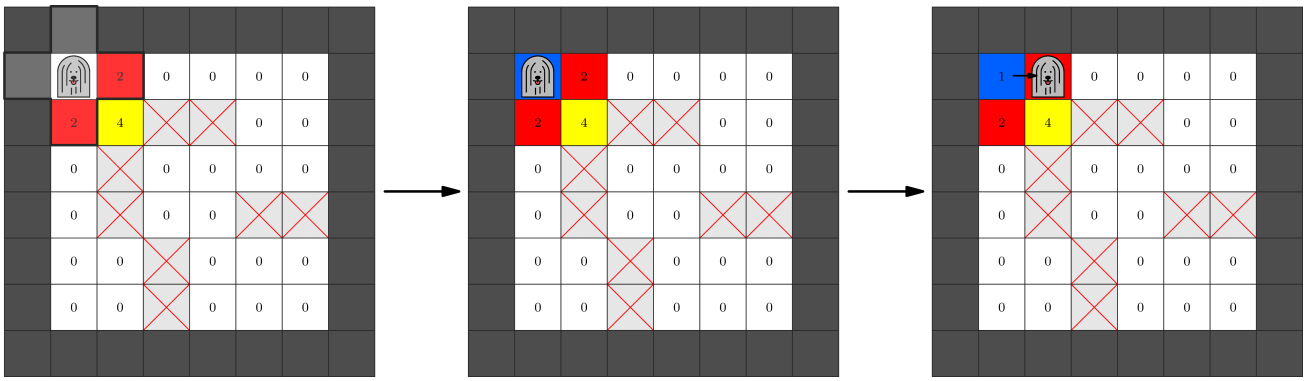
El programa de Pulibot se ejecuta como una secuencia de pasos. En cada paso, Pulibot reconoce el estado de las casillas cercanas y después ejecuta una instrucción. La instrucción que ejecuta está determinada por los estados reconocidos. A continuación hay una descripción más precisa.

Supongamos que Pulibot está en la casilla vacía (r, c) justo antes de ejecutar un paso. El paso se realiza como sigue:

1. Primero, Pulibot reconoce el **array de estado** actual, es decir, el array $S = [S[0], S[1], S[2], S[3], S[4]]$, conteniendo el estado de las casillas (r, c) y de todas las casillas adyacentes:
    - $S[0]$ es el estado de la casilla (r, c).
    - $S[1]$ es el estado de la casilla al oeste.
    - $S[2]$ es el estado de la casilla al sur.
    - $S[3]$ es el estado de la casilla al este.
    - $S[4]$ es el estado de la casilla al norte.
2. Después, Pulibot determina la **instrucción** (Z, A) que se corresponde con el array de estado que ha reconocido.

3. Finalmente, Pulibot ejecuta la instrucción: pone la casilla (r, c) de color Z y después ejecuta la acción A, que es una de las siguientes acciones:
    - *quedarse* en la casilla (r, c);
    - *moverse* a una de las 4 casillas adyacentes;
    - *terminar el programa*.

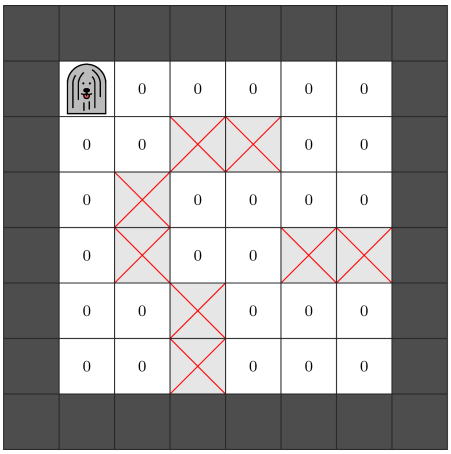
Por ejemplo, considera la situación representada en la siguiente figura a la izquierda. Pulibot está actualmente en la casilla $(0, 0)$, que tiene el color 0. Pulibot reconoce el array de estado $S = [0, -2, 2, 2, -2]$. Pulibot puede tener un programa en el que, tras reconocer este array, pone el color de la casilla actual a $Z = 1$ y después se mueve hacia el este, tal como se muestra en la figura en el medio y a la derecha:

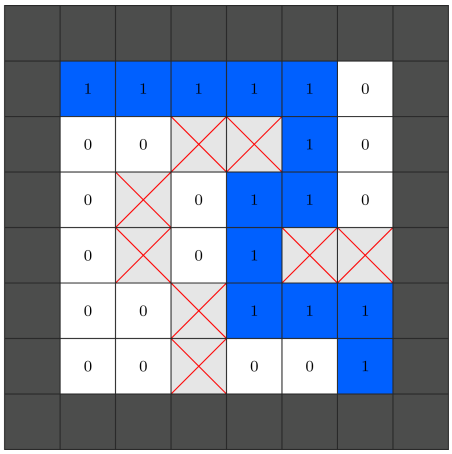


Reglas del Concurso de Robots

- Al principio, Pulibot está situado en la casilla $(0, 0)$ y comienza a ejecutar su programa.
- No está permitido que Pulibot se mueva a una casilla no vacía.
- El programa de Pulibot debe terminar tras como mucho $500\,000$ pasos.
- Cuando el programa de Pulibot termine, las casillas vacías del laberinto deberían estar coloreadas de forma que:
    - Exista un camino de longitud mínima de $(0, 0)$ a $(H - 1, W - 1)$ tal que el color de cada casilla incluída en el camino sea 1.
    - El color de todas las otras casillas vacías sea 0.
- Pulibot puede terminar su programa en cualquier casilla vacía.

Por ejemplo, la siguiente figura muestra un laberinto con $H = W = 6$. La disposición inicial se muestra a la izquierda y una coloración aceptable de las casillas vacías después de la terminación se muestra a la derecha:





Detalles de Implementación

Debes implementar el siguiente procedimiento:

```
void program_pulibot()
```

- El procedimiento debe generar el programa del Pulibot. Este programa debería funcionar correctamente para todos los valores de H y W y cualquier laberinto que satisfaga las restricciones del problema.
- Este procedimiento se llama exactamente una vez por cada caso de prueba.

Este procedimiento puede hacer llamadas al siguiente procedimiento para generar el programa del Pulibot:

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

- S: array de longitud 5 describiendo un array de estado.
- Z: un entero no negativo representando un color.
- A: un único carácter representando una acción de Pulibot:
  - `H`: quedarse;
  - `W`: moverse hacia el oeste;
  - `S`: moverse hacia el sur;
  - `E`: moverse hacia el este;
  - `N`: moverse hacia el norte;
  - `T`: terminar el programa.
- Llamar a este procedimiento ordena al Pulibot a que al reconocer el array de estados S debería realizar la instrucción (Z, A).

Llamar al procedimiento más de una vez con el mismo array de estados S resultará en el veredicto de `Output isn't correct`.

No es necesario llamar a `set_instruction` con todos y cada uno de los posibles arrays de estado S. Sin embargo, si el Pulibot después reconoce un array de estados para el cual no se ha determinado una instrucción, obtendrás el veredicto `Output isn't correct`.

Después de que el procedimiento `program_pulibot` acabe, el grader ejecutará el programa del Pulibot sobre uno o más laberintos. Estas ejecuciones *no* cuentan para el límite de tiempo de tu solución. El grader *no* es adaptivo, es decir, el conjunto de laberintos está predefinido en cada caso de prueba.

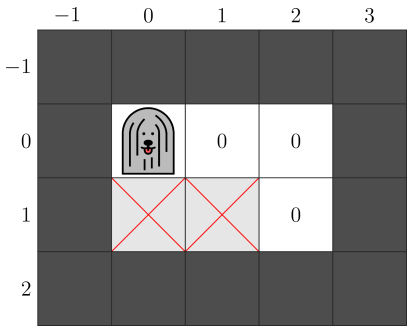
Si el Pulibot viola alguna de las Reglas del Concurso de Robots antes de terminar su programa, obtendrás el veredicto de `Output isn't correct`.

Example

El procedimiento `program_pulibot` puede hacer llamadas a `set_instruction` de la siguiente forma:

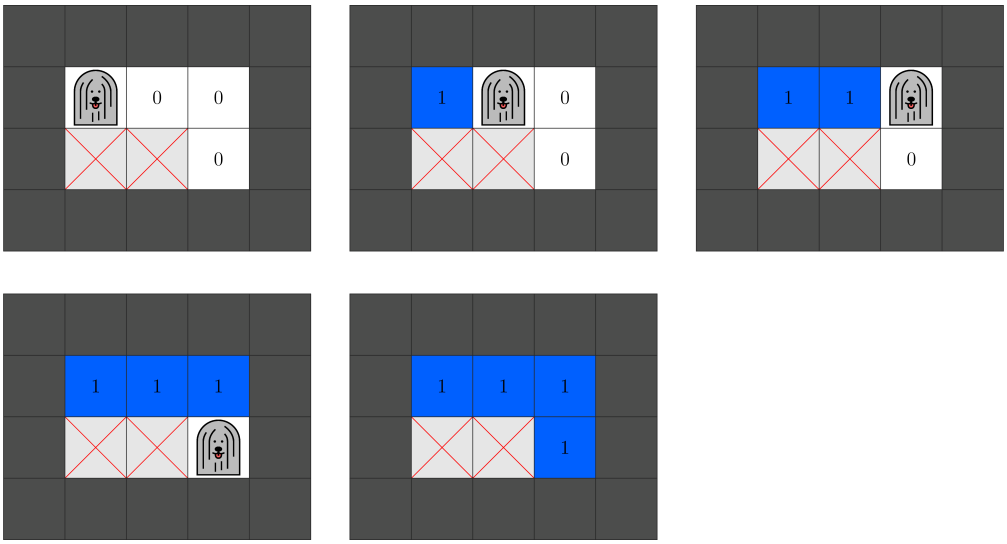
Llamada	Instrucción para el array de estado S
`set_instruction([0, -2, -1, 0, -2], 1, E)`	Poner color a 1 y moverse hacia el este
`set_instruction([0, 1, -1, 0, -2], 1, E)`	Poner color a 1 y moverse hacia el este
`set_instruction([0, 1, 0, -2, -2], 1, S)`	Poner color a 1 y moverse hacia el sur
`set_instruction([0, -1, -2, -2, 1], 1, T)`	Poner color a 1 y terminar el programa

Considera la situación donde $H = 2$ y $W = 3$, y el laberinto es como se muestra en la siguiente figura.



Para este laberinto el programa de Pulibot se ejecuta en cuatro pasos. Los arrays de estado que el Pulibot reconoce y las instrucciones que realiza corresponden exactamente a las cuatro llamadas a `set_instruction` hechas arriba, en orden. La última instrucción de estas termina el programa.

La siguiente figura muestra el laberinto antes de cada uno de los cuatro pasos y los colores finales después de que el programa termine.



Sin embargo, recalcamos que este programa de 4 instrucciones tal vez no encuentre un camino de longitud mínima en otros laberintos válidos. Por tanto, si se envía, recibirá el veredicto de `Output isn't correct`.

Constraints

$Z_{MAX} = 19$. Por tanto, el Pulibot puede usar colores de 0 a 19, inclusive.

Para cada laberinto usado para probar al Pulibot:

- $2 \leq H, W \leq 15$
- Hay al menos un camino desde la casilla $(0, 0)$ a la casilla $(H - 1, W - 1)$.

Subtasks

1. (6 puntos) No hay ninguna casilla obstáculo en el laberinto.
2. (10 puntos) $H = 2$
3. (18 puntos) Hay exactamente un camino entre cada par de casillas vacías.
4. (20 puntos) Los caminos más cortos de la casilla $(0, 0)$ a la casilla $(H - 1, W - 1)$ tienen longitud $H + W - 2$.
5. (46 puntos) Sin restricciones adicionales.

Si, en alguno de los casos de prueba, las llamadas al procedimiento `set_instruction` o la ejecución del programa de Pulibot no se adecuan a las restricciones descritas en Detalles de Implementación, la puntuación de tu solución para esa subtarea será 0.

En cada subtarea, puedes obtener una puntuación parcial si produces una coloración que es casi correcta.

Formalmente:

- La solución de un caso de prueba es **completa** si la coloración final de las casillas vacías satisface las Reglas del Concurso de Robots.
- La solución de un caso de prueba es **parcial** si la coloración final tiene el siguiente aspecto:
  - Exista un camino de longitud mínima de $(0, 0)$ a $(H - 1, W - 1)$ tal que el color de cada casilla incluída en el camino es 1.
  - No hay ninguna otra casilla vacía en la cuadrícula con color 1.
  - Alguna casilla vacía en la cuadrícula tiene un color diferente de 0 y 1.

Si tu solución a un caso de prueba no es ni completa ni parcial, tu puntuación para el caso de prueba correspondiente será 0.

En las subtareas 1-4, la puntuación para un caso de prueba de una solución completa es el 100% y la de una solución parcial es el 50% de los puntos de la subtarea.

En la subtarea 5, tu puntuación depende del número de colores usados en el programa del Pulibot. Más precisamente, denotemos por $Z^\star$ el máximo valor de Z sobre todas las llamadas hechas a `set_instruction`. La puntuación del caso de prueba se calcula según la siguiente tabla:

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

La puntuación para cada subtarea es el mínimo de las puntuaciones de los casos de prueba de esa subtarea.

Grader de Ejemplo

El grader de ejemplo lee la entrada en el siguiente formato:

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

Aquí, m es un array de H arrays de W enteros, describiendo las celdas del laberinto. $m[r][c] = 0$ si la casilla (r, c) es una casilla vacía y $m[r][c] = 1$ si la casilla (r, c) es una casilla obstáculo.

El grader de ejemplo primero llama al `program_pulibot()`. Si el grader de ejemplo detecta una violación del protocolo, el grader de ejemplo imprime `Protocol Violation: <MSG>` y termina, donde `<MSG>` es uno de los siguientes mensajes de error:

- `Invalid array`: $-2 \leq S[i] \leq Z_{MAX}$ no se cumple para algún i o la longitud de S no es 5.
- `Invalid color`: $0 \leq Z \leq Z_{MAX}$ no se cumple.
- `Invalid action`: el carácter A no es `H`, `W`, `S`, `E`, `N` o `T`.
- `Same state array`: `set_instruction` fue llamado con la misma array S más de una vez.

Si no, cuando `program_pulibot` acabe, el grader de ejemplo ejecuta el programa del Pulibot en el laberinto descrito por la entrada.

El grader de ejemplo produce dos salidas.

Primero, el grader de ejemplo escribe una transcripción de las acciones del Pulibot al archivo `robot.bin` en el directorio de trabajo actual. Este archivo sirve como entrada de la herramienta de visualización descrita en la siguiente sección.

Segundo, si el programa del Pulibot no termina exitosamente, el grader de ejemplo imprime uno de los siguientes mensajes de error:

- `Unexpected state`: Pulibot ha reconocido un array de estado para el que `set_instruction` no ha sido llamado.
- `Invalid move`: Pulibot se ha movido a una casilla no vacía como resultado de una acción.
- `Too many steps`: Pulibot ha realizado $500\,000$ pasos sin terminar el programa.

Si no, sea $e[r][c]$ el estado de la casilla (r, c) después de que el programa de Pulibot termine. El grader de ejemplo imprime H líneas en el siguiente formato:

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

Herramienta de Visualización

El adjunto a este problema contiene un archivo llamado `display.py`. Cuando se ejecuta, este script de Python muestra las acciones del Pulibot en el laberinto descrito por la entrada del grader de ejemplo. Para esto, el archivo `robot.bin` debe estar presente en el directorio de trabajo actual.

Para usar el script, ejecuta el siguiente trabajo.

```
python3 display.py
```

Una interfaz gráfica simple aparecerá. Sus funcionalidades principales son las siguientes:

- Puedes observar el estado del laberinto completo. La localización actual del Pulibot está resaltada con un rectángulo.
- Puedes navegar en la reproducción de los pasos del Pulibot haciendo click en los botones de las flechas o pulsando los atajos de teclado correspondientes. También puedes saltar a un paso específico.

- El siguiente paso en el programa del Pulibot se muestra abajo. Muestra el actual array de estados y la instrucción que realizará. En el último paso, muestra alguno de los mensajes de error del grader, o `Terminated` si el programa termina exitosamente.
- Para cada número que representa un color, puedes asignar un color de fondo y un texto que mostrar visualmente. El texto es una cadena de caracteres corta que aparecerá en cada una de las casillas con ese color. Puedes asignar colores de fondo y textos en cualquiera de las siguientes maneras:
    - Ponlos en una ventana que se abrirá después de hacer click en el botón `Colors`.
    - Edita los contenidos del archivo `colors.txt`.
- Para recargar `robot.bin`, usa el botón `Reload`. Es útil si el contenido de `robot.bin` ha cambiado.