Concurso de Robots

Algunos investigadores en inteligencia artificial de la Universidad de Szeged están organizando un concurso de programación de robots y tu amiga Hanga quiere participar. El objetivo es programar el mejor *Pulibot* que el mundo ha visto, en honor a la famosa raza de perro pastor húngaro, los Puli.

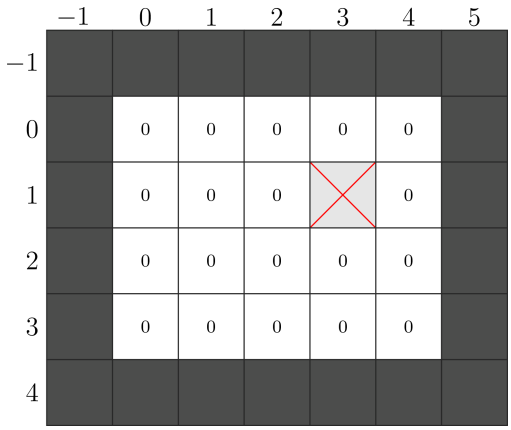
Pulibot será puesto a prueba con un laberinto, representado por una cuadrícula de $(H+2) \times (W+2)$. Las filas de la cuadrícula están numeradas del -1 al H de norte a sur y las columnas de la cuadrícula están numeradas del -1 al W de oeste a este. Nos referimos a la casilla que está en la fila r y en la columna c de la cuadrícula ($-1 \leq r \leq H$, $-1 \leq c \leq W$) como la casilla (r, c).

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

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

Llamamos a la casilla (r, c) como una casilla **borde** del laberinto si se cumple que $r = -1$ o $r = H$ o $c = -1$ o $c = W$. Cada casilla vacía que no es una casilla borde del laberinto, es ya sea una casilla con un **obstáculo** o una casilla **vacía**. Adicionalmente, cada casilla vacía tiene un **color**, representado por un entero no negativo entre 0 y Z_{MAX} inclusivo. Inicialmente, el color de cada casilla vacía es 0.

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



La única casilla con un obstáculo se muestra con una cruz. Las casillas borde del laberinto están sombreadas. El número escrito en cada casilla vacía representa su color.

Un **camino** de longitud ℓ ($\ell > 0$) de 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 cual, para toda i ($0 \leq i < \ell$) las casillas (r_i, c_i) y (r_{i+1}, c_{i+1}) son adyacentes.

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

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

Hanga no sabe cuáles casillas del laberinto están vacías ni cúales contienen obstáculos.

Tu tarea es ayudar a Hanga a programar su Pulibot tal que sea capaz de encontrar un *camino más corto* (es decir, un camino con longitud mínima) de la casilla $(0, 0)$ a la casilla $(H - 1, W - 1)$ en el laberinto desconocido escogido por los investigadores. Las especificaciones del Pulibot y las reglas del concurso se describen a continuación.

Nota que la última sección de este problema describe una herramienta que puedes usar para visualizar a Pulibot.

Especificaciones de Pulibot

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

- si la casilla (r, c) es un borde entonces su estado es -2;
- si la casilla (r, c) contiene un obstáculo entonces su estado es -1;
- si la casilla (r, c) está vacía entonces su estado es el color de la casilla.

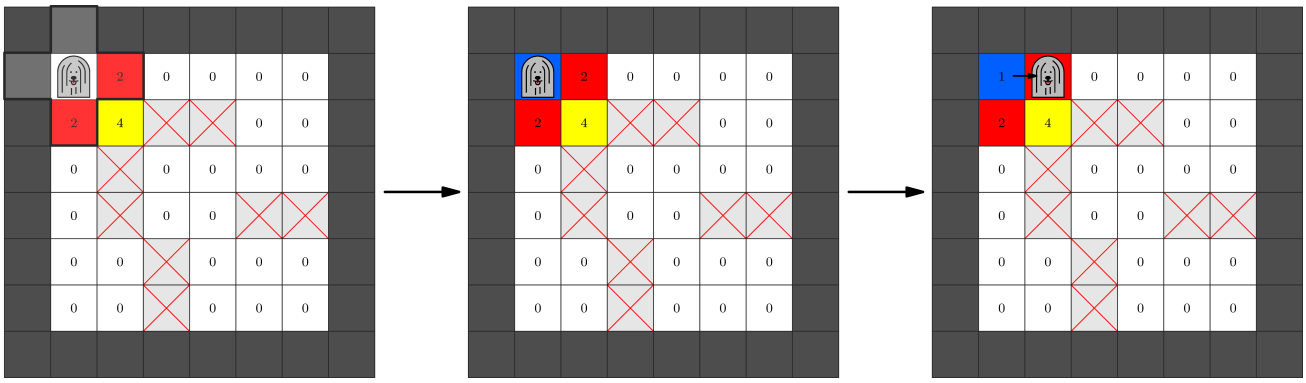
El programa de Pulibot se ejecuta como una secuencia de pasos. En cada paso, Pulibot reconoce el estado de las casillas cercanas y luego ejecuta una instrucción. La instrucción que ejecuta se

determina utilizando los estados reconocidos. Te damos una descripción más precisa a continuación:

Supón que al principio del paso actual, Pulibot está en la casilla (r, c), que está vacía. Procedemos así:

1. Primero, Pulibot reconoce el actual **arreglo de estados**, es decir, el arreglo $S = [S[0], S[1], S[2], S[3], S[4]]$, que consiste del estado de la casilla (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. Segundo, Pulibot determina la **instrucción** (Z, A) que corresponde al arreglo de estados que reconoció.
3. Finalmente, Pulibot ejecuta esa instrucción: colorea la casilla (r, c) de color Z y luego ejecuta la acción A, que es alguna de las siguientes acciones:
   - *quedarse* en la casilla (r, c);
   - *moverse* a alguna de las 4 casillas adyacentes;
   - *terminar el programa*.

Por ejemplo, considera el caso de la izquierda en la siguiente figura. Pulibot actualmente se encuentra en la casilla $(0, 0)$ que tiene color (0). Pulibot reconoce el arreglo de estados $S = [0, -2, 2, 2, -2]$. Pulibot podría tener un programa que, en cuanto reconozca este arreglo, coloree la casilla actual a $Z = 1$ y luego se mueva al este, como se muestra en el resto de la figura:

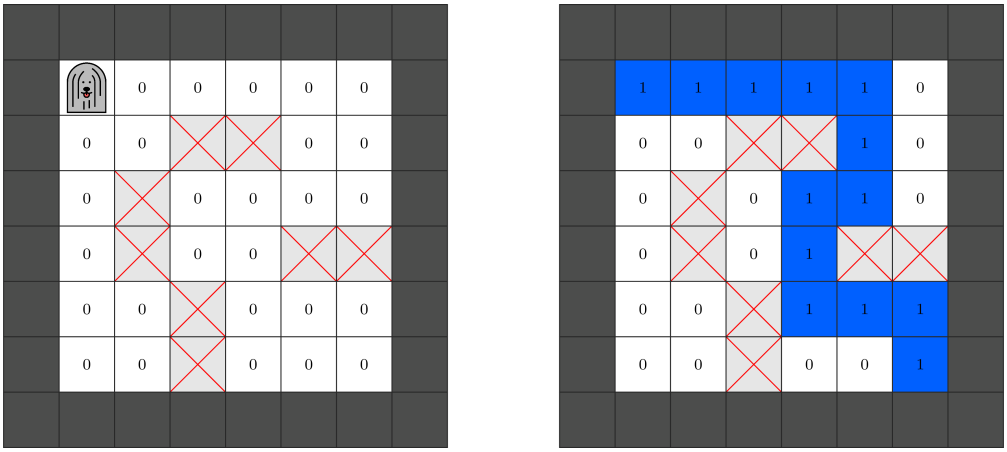


Reglas del Concurso de Robots

- Al principio, Pulibot es colocado en la casilla $(0, 0)$ y empieza a ejecutar su programa.
- No está permitido que Pulibot se mueva a una casilla no vacía.
- El programa de Pulibot deberá terminar en a lo más $500\,000$ pasos.
- Después de que el programa de Pulibot termine, las casillas vacías del laberinto deben estar coloreadas tal que:
  - Exista un camino más corto de $(0, 0)$ a $(H - 1, W.1)$ para el cual el color de todas las casillas en el camino sea 1.

    - Todas las demás casillas vacías tengan color 0.
- Pulibot puede terminar su programa en cualquier casilla vacía.

Por ejemplo, la siguiente figura muestra un posible laberinto con $H = W = 6$. La configuración incial se muestra a la izquierda y una posible coloración aceptable de casillas vacias después de que Pulibot termine de ejecutarse se muestra a la derecha:



Detalles de Implementación

Debes implementar la siguiente función:

```
void program_pulibot()
```

- Esta función deberá de crear el programa de Pulibot. Este programa debe funcionar correctamente para todos los valores de H y W y para cualquier laberinto que cumpla con los límites del problema.
- Esta función se llama exactamente una vez en cada caso de prueba.

Esta función puede hacer llamadas al siguiente procedimiento para crear el programa de Pulibot:

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

- S: un arreglo de longitud 5 que describe un arreglo de estados.
- Z: un entero no negativo que representa un color.
- A: un caracter que representa una acción de Pulibot, como se describe a continuación:
    - `H`: quedarse;
    - `W`: moverse al oeste;
    - `S`: moverse al sur;
    - `E`: moverse al este;
    - `N`: moverse al norte;
    - `T`: terminar el programa.
- Llamar a este procedimiento le indica a Pulibot que cuando reconozca el arreglo de estados S, deberá de ejecutar la instrucción (Z, A).

Si la función llama a este procedimiento multiples veces con el mismo arreglo de estados S, recibirás el veredicto `Output isn't correct`.

No se requiere que llames a `set_instruction` con todos los posibles arreglos de estados S. Sin embargo, si Pulibot luego reconoce un estado para el cual nunca le indicaste una instrucción, recibirás el veredicto `Output isn't correct`.

Despues de que `program_pulibot` termine de ejecutarse, el evaluador invocará el programa de Pulibot en uno o más laberintos. Estas invocaciones *no* se considerarán para calcular el tiempo de ejecución de tu solución. El evaluador *no* es adaptivo, es decir, el conjunto de laberintos está predefinido para cada caso de prueba.

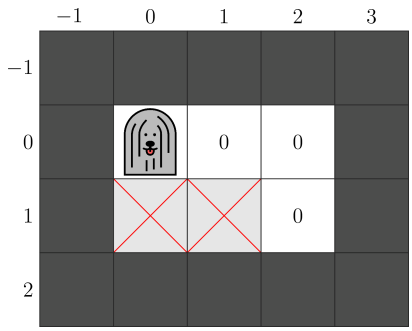
Si Pulibot rompe cualquiera de las reglas del concurso de robots antes de terminar su programa, recibirás el veredicto `Output isn't correct`.

Ejemplo

La función `program_pulibot` puede llamar al procedimiento `set_instruction` como se muestra a continuación:

Llamada	Instrucción para el arreglo de estados S
`set_instruction([0, -2, -1, 0, -2], 1, E)`	Colorear con 1 y moverse al este
`set_instruction([0, 1, -1, 0, -2], 1, E)`	Colorear con 1 y moverse al este
`set_instruction([0, 1, 0, -2, -2], 1, S)`	Colorear con 1 y moverse al sur
`set_instruction([0, -1, -2, -2, 1], 1, T)`	Colorear con 1 y terminar el programa

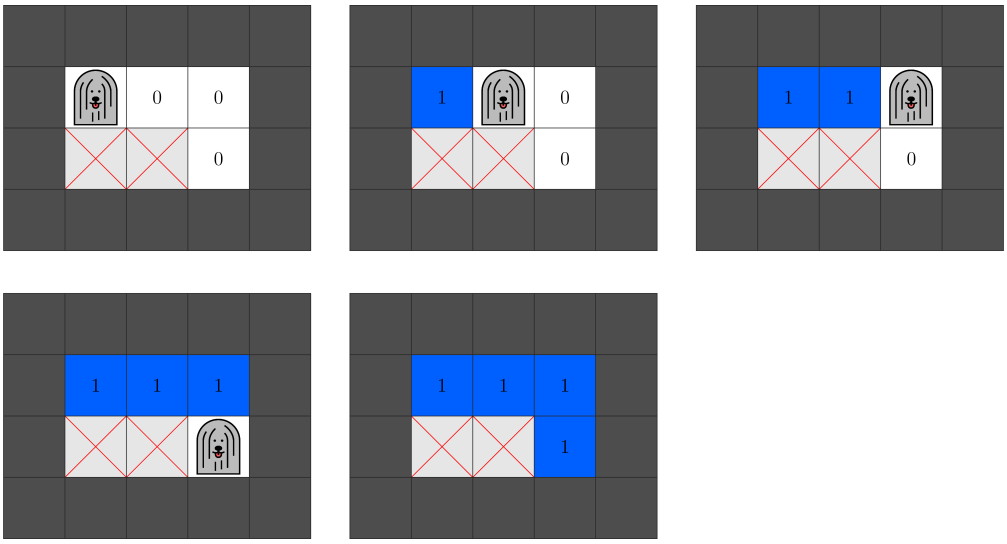
Consider a scenario where $H = 2$ and $W = 3$, and the maze is displayed in the following figure.



Para este laberinto en particular, el programa de Pulibot se ejecuta en cuatro pasos. Los arreglos de estados que Pulibot reconoce y las instrucciones que ejecuta ecorresponden exactamente a las

cuatro llamadas a `set_instruction`, en orden. La última 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 terminar.



Sin embargo, nota que este programa de 4 puede ser que no encuentre un camino más corto en otros laberintos válidos. Por lo tanto, si fueses a subirlo, recibirías el veredicto `Output isn't correct`.

Límites

$Z_{MAX} = 19$. Por lo tanto, Pulibot puede utilizar los colores del 0 al 19 inclusivo.

Para cada laberinto con el que se evalúa Pulibot:

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

Subtareas

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

Si en cualquiera de los casos de prueba las llamadas al procedimiento `set_instruction` o el programa de Pulibot no sigue las restricciones descritas en Detalles de Implementación, tu puntaje para esa subtarea será 0.

En cada subtarea, podrás obtener un puntaje parcial si produces una coloración que está *casi* correcta.

Formalmente:

- La solución de un caso de prueba está **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 se ve como describimos a continuación:
  - Existe un camino más corto de $(0, 0)$ a $(H - 1, W.1)$ para el cual el color de todas las casillas en el camino sea 1.
  - No hay ninguna otra casilla vacía con el color 1.
  - Alguna casilla vacía tiene un color distinto de 0 o 1.

Si tu solución a un caso de prueba no está completa ni es parcial, entonces tu puntaje para ese caso de prueba sera 0.

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

En la subtarea 5, tu puntaje depende del número de colores utilizados en el programa de Pulibot. Más precisamente, sea $Z^\star$ el máximo valor de Z sobre todas las llamadas hechas a `set_instruction`. El resultado de un caso de prueba se calcula de acuerdo a la siguiente tabla:

Condición	Puntaje (completa)	Puntaje (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

El puntaje para cada subtarea es el mínimo puntaje de todos los casos de prueba en esa subtarea.

Evaluador de Ejemplo

El evaluador 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 arreglo de H arreglos de W enteros, describiendo las casillas que no forman parte del borde del laberinto. $m[r][c] = 0$ si la casilla (r, c) está vacía y $m[r][c] = 1$ si la casilla (r, c) contiene un obstáculo.

El evaluador de ejemplo primero llamará `program_pulibot()`. Si el evaluador de ejemplo detecta un incumplimiento de protocolo, el evaluador de ejemplo imprimirá `Protocol Violation: <MSG>` y terminará, donde `<MSG>` es uno de los siguientes mensajes de error:

- `Invalid array`: $-2 \leq S[i] \leq Z_{MAX}$ no se cumple para alguna i o el tamaño de S no es 5.
- `Invalid color`: $0 \leq Z \leq Z_{MAX}$ no se cumple.
- `Invalid action`: el caracter A no es uno de `H`, `W`, `S`, `E`, `N` o `T`.
- `Same state array`: `set_instruction` fue llamada con el mismo arreglo de estados S al menos dos veces.

De lo contrario, cuando `program_pulibot` termine, el evaluador de ejemplo ejecuta el programa de Pulibot en el laberinto descrito en la entrada.

El evaluador de ejemplo produce dos salidas:

Primero, el evaluador de ejemplo escribe un historial de las acciones de Pulibot al archivo `robot.bin` en el directorio activo. Este archivo funciona como la entrada a la herramienta de visualización descrita en la siguiente sección.

Segundo, si el programa de Pulibot no termina con éxito, el evaluador de ejemplo imprime uno de los siguientes mensajes de error:

- `Unexpected state`: Pulibot reconoció un arreglo de estados que no fue indicado con `set_instruction`.
- `Invalid move`: Pulibot se movió a una casilla no vacía.
- `Too many steps`: Pulibot hizo $500\,000$ pasos sin terminar su programa.

De lo contrario, sea $e[r][c]$ el estado de la casilla (r, c) después de que el programa de Pulibot termina. El evaluador 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 archivo adjunto para este problema contiene un archivo llamado `display.py`. Cuando se invoque, este script de Python mostrará las acciones de Pulibot en el laberinto según descritos por la entrada dada por el evaluador de ejemplo. Para esto, el archivo `robot.bin` deberá estar presente en el directorio activo.

Para invocar el script, ejecuta el siguiente comando:

```
python3 display.py
```

Una interfaz gráfica simple aparecerá. Las principales features se describen a continuación:

- Puedes observar el estatus del laberinto completo; la ubicación de Pulibot se resalta con un rectángulo.
- Puedes explorar los pasos de Pulibot haciendo click en las flechas o presionando sus atajos. También puedes saltar a un paso en específico.
- El siguiente paso en el programa de Pulibot se muestra abajo. Muestra el arreglo de estados actual y la instrucción que va a ejecutar. Después del último paso, muestra o el mensaje de error del evaluador o `Terminated` si el programa termina correctamente.
- Para cada número que representa un color, puedes asignarle un color de fondo, así como un texto. El texto es una cadena de caracteres corta que aparecerá en todas las casillas con ese color. Puedes asignar los colores de fondo y los textos usando alguno de estos métodos:
    - Configurándolos en la ventana que aparece después de darle click al botón de `Colors`.
    - Editando el archivo `colors.txt`.
- Para refrescar `robot.bin`, usa el botón de `Reload`. Es útil si el contenido de `robot.bin` cambió.