Competencia de robots

Los investigadores de Inteligencia Artificial en la Universidad de Szeged han organizado una competencia de programación de robots. Tu amigo, Hanga, ha decidido tomar parte en la competencia. El objetivo es programar *Pulibot*, simulando la inteligencia de una raza húngara de perros de caza famosa, el Puli

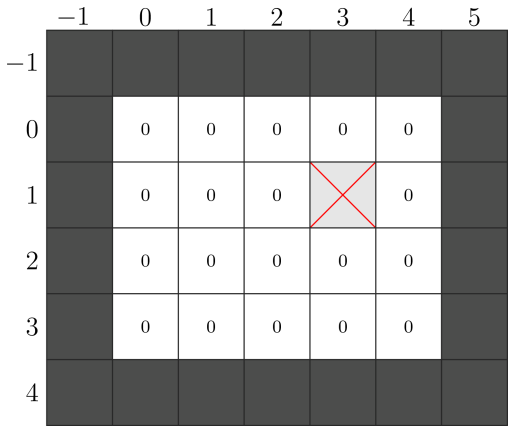
Se probará Pulibot en un laberinto representado por una retícula de $(H+2) \times (W+2)$ celdas. Las filas de la retícula están numeradas de -1 a H de norte a sur y las columnas de la retícula están numeradas de -1 a W de oeste a este. Llamamos (r, c) a la celda ubicada en la fila r y columna c de la retìcula ($-1 \leq r \leq H$, $-1 \leq c \leq W$).

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

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

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

Por ejemplo, considera un laberinto con $H = 4$ y $W = 5$, con una sola celda obstáculo $(1, 3)$:



La única celda obstáculo se marca con una cruz. Las celdas borde están sombreadas. Los números escritos en cada celda vacía repesentan sus colores respectivos.

Un **camino** de longitud ℓ ($\ell > 0$) de la celda (r_0, c_0) a la celda (r_ℓ, c_ℓ) es una sucesión de celdas *vacías* distintas dos a dos $(r_0, c_0), (r_1, c_1), \ldots, (r_\ell, c_\ell)$ en donde para cada i ($0 \leq i < \ell$) las celdas (r_i, c_i) y (r_{i+1}, c_{i+1}) son adyacentes.

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

En la competencia, los investigadores construyen un laberinto en el cual existe al menos un camino de la celda $(0, 0)$ a la celda $(H - 1, W - 1)$. Nota que esto garantiza que las celdas $(0, 0)$ y $(H - 1, W - 1)$ están vacías.

Hanga no sabe cuáles celdas del laberinto son vacías y cuáles son obstáculos.

Tu tarea es ayudar a Hanga a programar Pulibot de manera que sea capaz de encontrar un *camino más corto* (esto es, de longitud mínima) de la celda $(0, 0)$ a la celda $(H - 1, W - 1)$ del laberinto desconocido proporcionado por los investigadores. La especificación de Pulibot y las reglas de la competencia se describen a continuación. Nota que la última sección de este problema describe una herramienta de visualización que puedes usar para proyectar a Pulibot.

Especificación de Pulibot

Se define el **estado** de una celda (r, c) para cada $-1 \leq r \leq H$ y $-1 \leq c \leq W$ como un entero de manera que:

- Si la celda (r, c) es un borde, entonces su estado es -2;
- Si la celda (r, c) es un obstáculo entonces su estado es -1;
- Si la celda (r, c) es una celda vacía entonces su estado es el color de la celda.

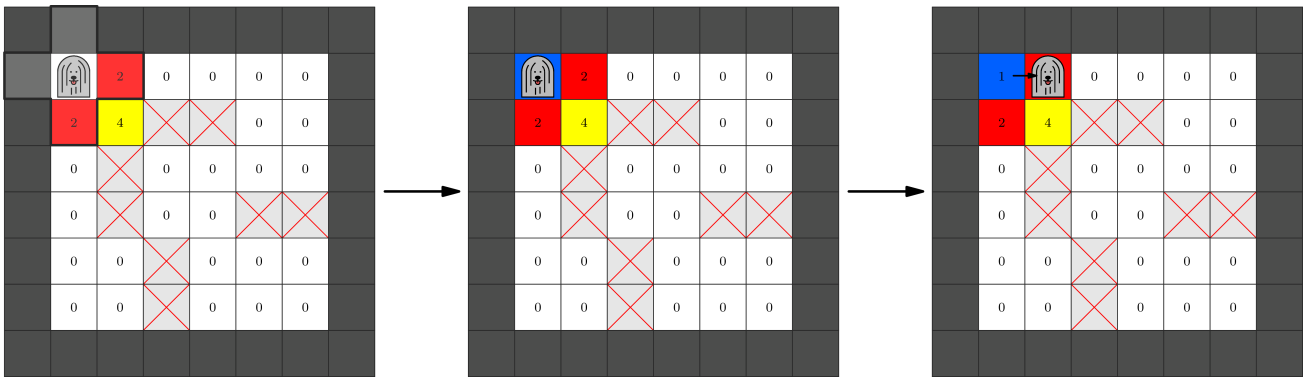
El programa de Pulibot se ejecuta como una sucesión de pasos. En cada paso, Pulibot reconoce los estados de las celdas vecinas y luego ejecuta una instrucción. La instrucción que ejecuta se determina por los estados reconocidos. Una descripción más precisa se da a continuación.

Supón que al inicio del paso actual, Pulibot está en la celda (r, c), la cual está vacia. El paso se ejecuta como sigue:

1. Primero, Pulibot reconoce el **arreglo de estados** actual, esto es, el arreglo $S = [S[0], S[1], S[2], S[3], S[4]]$, que consiste de los estados de la celda(r, c) y de los estados de todas las celdas adyacentes:
    - $S[0]$ es el estado de la celda (r, c).
    - $S[1]$ es el estado de la celda al oeste.
    - $S[2]$ es el estado de la celda al sur.
    - $S[3]$ es el estado de la celda al este.
    - $S[4]$ es el estado de la celda al norte.
2. Luego, Pulibot determina la **instrucción** (Z, A) que corresponde al arreglo de estados reconocido.

3. Finalmente, Pulibot ejecuta dicha instrucción: colorea la celda (r, c) con color Z y luego ejecuta la acción A, la cual es una de las siguientes:
    - *permanecer* en la celda (r, c);
    - *moverse* a una de las 4 celdas adyacentes;
    - *terminar el programa*.

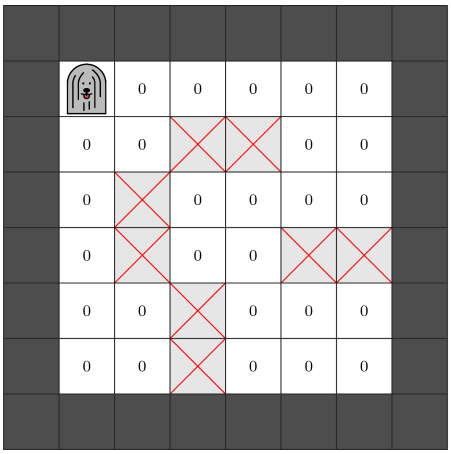
Por ejemplo, considera el escenario que se muestra a la izquierda de la siguiente figura. Pulibot está actualmente en la celda $(0, 0)$ con color 0. Pulibot reconoce el arreglo de estados $S = [0, -2, 2, 2, -2]$. Pulibot podría tener un programa en el cual, después de reconocer este arreglo, colorea la celda actual con color $Z = 1$ y luego se mueve al este, como se muestra en el medio y a la derecha de la figura.

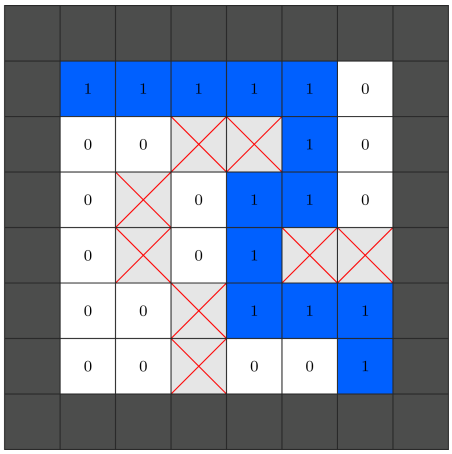


Reglas de la competencia de robots

- Al comienzo, Pulibot está en la celda $(0, 0)$ y comienza a ejecutar su programa.
- Pulibot no tiene permitido moverse a una celda que no está vacía.
- El programa de Pulibot debe terminar en a lo más $500\,000$ pasos.
- Después de que termina el programa de Pulibot, las celdas vacias del laberinto deben estar coloreadas de modo que:
    - Existe un camino más corto de $(0, 0)$ to $(H - 1, W - 1)$ en el cual, el color de cada celda que hace parte del camino es 1.
    - El color de cualquier otra celda vacía es 0.
- Pulibot termina su programa en alguna celda vacía.

Por ejemplo, la siguiente figura muestra un posible laberinto con $H = W = 6$. La configuración inicial se muestra a la izquierda y la coloración esperada de las celdas vacías despúes de que termine el programa de Pulibot se muestra a la derecha:





Detalles de implementación

Debes implementar el procedimientos siguiente.

```
void program_pulibot()
```

- Este procedimiento debe producir el programa de Pulibot. Este programa debe funcionar correctamente para todos los valores de H y W para cualquier laberinto que cumpla los requerimientos de la tarea.
- Se llama a este procedimiento exactamente una vez para cada caso de prueba.

Este procedimiento puede hacer llamadas al procedimiento siguiente para producir el programa de Pulibot.

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

- S: arreglo de longitud 5 describiendo un arreglos de estados.
- Z: un entero no negativo representando un color.
- A: un solo carácter representado una acción de Pulibot como sigue:
  - `H`: permanecer;
  - `W`: moverse al oeste;
  - `S`: moverse al sur;
  - `E`: moverse al este;
  - `N`: moverse al norte;
  - `T`: terminar el programa.
- Llamar a este procedimiento instruye a Polibot que después de reconocer el estado S debe ejecutar la instrucción (Z, A).

Llamar a este procedimiento varias veces con el mismo estado S resultará en un veredicto `Output isn't correct`.

No se requiere llamar a `set_instruction` con cada arreglo de estados S posible. Sin embargo si Pulibot reconoce un estado para el cual no se ha establecido una instrucciòn, usted obtendrá un veredicto `Output isn't correct`.

Después que termina `program_pulibot`, el grader invoca el programa Pulibot con uno o más laberintos. Esas invocaciones *no* cuentan para el tiempo límite de su solución. El grader *no* es adaptativo, esto es, el conjunto de laberintos está pre-definido para cada caso de pruebas.

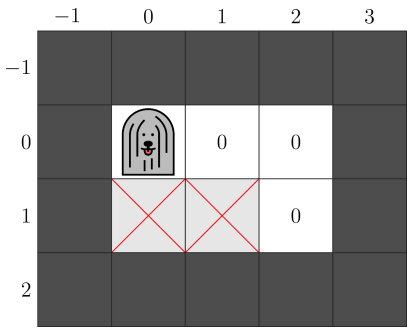
Si Pulibot viola cualquiera de las Reglas de la competencia de robots, usted obtendrá un veredicto `Output isn't correct`.

Ejemplo

El `program_pulibot` puede hacer llamadas a `set_instruction` como sigue:

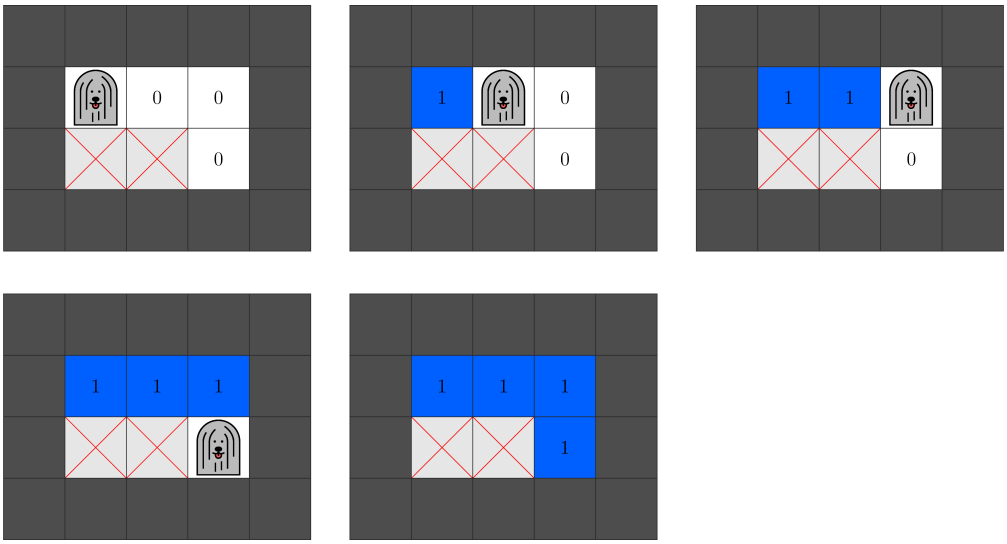
Llamado	Instrucción para el arreglo de estados S
`set_instruction([0, -2, -1, 0, -2], 1, E)`	Poner el color en 1 y moverse al este
`set_instruction([0, 1, -1, 0, -2], 1, E)`	Poner el color en 1 y moverse al este
`set_instruction([0, 1, 0, -2, -2], 1, S)`	Poner el color en 1 y moverse al sur
`set_instruction([0, -1, -2, -2, 1], 1, T)`	Poner el color en 1 y terminar el programa

Considera un escenario donde $H = 2$ y $W = 3$, y el laberinto se muestra en la siguiente figura.



Para este laberinto en particular el programa Pulibot corre en cuatro pasos. Los arreglos de estados que Pulibot reconoce y las instrucciones que ejecuta corresponden exactamente a los cuatro llamados a `set_instruction` hechos anteriormente, en orden. La última de estas instrucciones termina el programa.

La siguiente figura muestra el estado del laberinto antes de cada uno de los cuatro pasos y su estado final después de terminar.



Sin embargo, nota que este programa de 4 instrucciones podría no encontrar un camino más corto en otros laberintos válidos. Por lo tanto, si se envía, recibirá un veredicto `Output isn't correct`.

Restricciones

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

Por cada laberinto usado para probar Pulibot:

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

Subtareas

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

Si, en alguno de los casos de prueba, las llamadas al procedimiento `set_instruction` o el programa de Pulibot durante su ejecución no está acorde a las restricciones descritas en los detalles de implementación, el puntaje de tu solución para esa subtarea será 0.

En cada subtarea, puede obtener un puntaje parcial produciendo un coloreado que sea casi correcto.

Formalmente:

- La solución de un caso de prueba es **completa** si el coloreado final de las celdas vacías satisface las reglas de competición de robots.
- La solución de un caso de prueba es **parcial** si el coloreado final luce como sigue:
  - Existe un camino mínimo de $(0, 0)$ a $(H - 1, W - 1)$ para el cual el color de cada celda incluida en el camino es 1.
  - No existe otra celda en la retícula con color 1.
  - Algunas celdas vacías tienen un color diferente de 0 y 1.

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

En las subtareas 1-4, obtendrás 50% de los puntos si tu solución es parcial.

En la subtarea 5, tu puntaje dependerá del número de colores usados en el programa de Pulibot. Más precisamente, sea $Z^\star$ el máximo valor de Z entre todas las llamadas a `set_instruction`. El puntaje del caso de prueba es calculado de acuerdo a la siguiente tabla:

Condición	Puntaje (completo)	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 de cada subtarea es el mínimo de los puntajes obtenidos para los casos de prueba en esa subtarea.

Grader de Ejemplo

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

- línea 1 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 celdas no borde en el laberinto. $m[r][c] = 0$ si la celda (r, c) está vacía y $m[r][c] = 1$ si la celda (r, c) es un obstáculo

El grader de ejemplo primero llama a `program_pulibot()`. Si el grader de ejemplo detecta una violación de 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 si la longitud 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 llamado con el mismo arreglo S al menos dos veces.

En otro caso, cuando `program_pulibot` termina, el grader de ejemplo ejecuta el programa Pullbot en el laberinto descrito por la entrada.

El grader de ejemplo produce dos salidas.

Primero, el grader de ejemplo escribe un registro de las acciones de Pulibot en el archivo `robot.bin` en la carpeta de trabajo. Este archivo sirve como entrada a la herramienta de visualización descrita en la siguiente sección.

Segundo. Si el programa de Pulibot no termina satisfactoriamente, el grader de ejemplo imprime uno de los siguientes mensajes de error:

- `Unexpected state`: Pulibot reconoció un arreglo de estados con el cual `set_instruction` no fue llamado.
- `Invalid move`: ejectuar una acción resultando en mover a Pulibot a una celda no vacía.
- `Too many steps`: Pulibot ejecutó $500\,000$ pasos sin terminar el programa.

En otro caso, sea $e[r][c]$ el estado de la celda (r, c) después que termina el programa Pulibot. 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 paquete adjunto para esta tarea contiene un archivo llamado `display.py`. Cuando se invoca, este script de Python muestra las acciones de Pulibot en el laberinto descrito por la entrada del grader de ejemplo. Para esto el archivo binario `robot.bin` debe estar presente en la carpeta de trabajo.

Para invocar este script, ejecuta el siguiente comando.

```
python3 display.py
```

Se muestra una interfaz gráfica simple. Las características principales son como siguen:

- Puedes observar el estado de todo el laberinto. La posición actual de Pulibot está resaltada con un rectángulo.
- Puedes recorrer los pasos de Pulibot haciendo click en los botones flecha o presionando una combinación de teclas que reemplace a las flechas. También puedes saltar a un paso

específico.

- El paso siguiente del programa Pulibot se muestra abajo. Muestra el arreglo de estados actual y la instrucción que ejecutará. Después del paso final, muestra o uno de los mensajes de error del grader, o `Terminated` si el programa termina satisfactoriamente.

- Para cada número que representa un color, puedes asignar un color de fondo visual, así como un texto. El texto es una cadena corta que puede ser escrita en cada celda teniendo el mismo color. Puedes asignar colores de fondo y textos en cualquiera de las siguientes maneras:

    - Fíjalos en una ventana de diálogo después de dar click en el botón `Colors`.
    - Editar los contenidos del archivo `colors.txt`.
- Para recargar `robot.bin`, usa el botón `Reload`. Es útil si el contenido de `robot.bin` ha cambiado.