Programando el Robot

Los investigadores de inteligencia artificial en la Universidad de Szeged van a armar un torneo de programar robots, y tu amigo Hanga quiere participar. El objetivo es programar el magnífico *Pulibot*, tal que refleje el inigualable intelecto de la famosa raza perruna Húngara, los *Puli*.

El Pulibot se testeará en un laberinto que consiste en una grilla de $(H+2) \times (W+2)$ celdas.

Las filas de la grilla se numeran de -1 a H de norte a sur y las columnas se numeran de -1 a W de oeste a este. Denotamos (r, c) (con $-1 \leq r \leq H$, $-1 \leq c \leq W$) a la celda en la fila r y columna c.

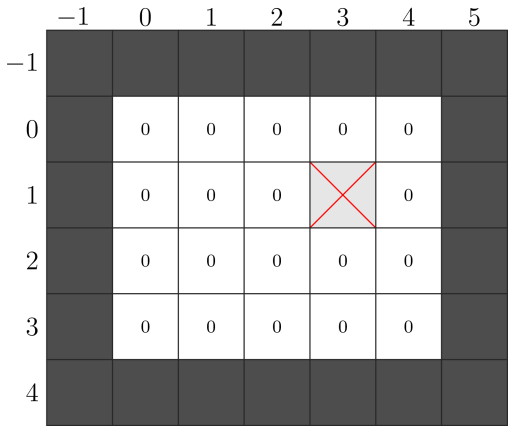
Considerá una celda (r, c) con $0 \leq r < H$ y $0 \leq c < W$. Hay 4 celdas **adyacentes** a la celda (r, c):

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

La celda (r, c) se dice que está en el **borde** si $r = -1$ o $r = H$ o $c = -1$ o $c = W$.

Cada celda que no está en el borde es o bien un **obstáculo** o bien está **libre**. Además, cada celda libre tiene un **color** representado por un entero no negativo entre 0 y Z_{MAX}, inclusive. Al principio, todas las celdas libres tienen el color 0.

Por ejemplo, considerá el laberinto con $H = 4$ y $W = 5$, y un solo obstáculo en la celda $(1, 3)$:



El obstáculo se muestra con una cruz, las celdas del borde se muestran sombreadas, y el color de cada celda libre está escrito en la celda.

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

Notá que un camino de longitud ℓ tiene exactamente $\ell + 1$ celdas.

En el torneo, los organizadores construyen el laberinto de tal forma que exista el menos un camino desde la celda $(0, 0)$ hasta la celda $(H - 1, W - 1)$. En particular, esto implica que tanto $(0, 0)$ como $(H - 1, W - 1)$ son celdas libres.

Hanga *no* sabe cuáles celdas del laberinto son libres y cuáles no.

Tu tarea es ayudar a Hanga a que programe al Pulibot para que pueda encontrar el *camino más corto* (el decir, el camino de longitud mínima) desde la celda $(0, 0)$ hasta la celda $(H - 1, W - 1)$ en el laberinto desconocido construido por los organizadores.

Las reglas y especificación para Pulibot están a continuación.

Notá que la última sección del enunciado de este problema describe una herramienta de visualización que podés usar para visualizar a Pulibot.

La Especificación de Pulibot

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

- si la celda (r, c) está en el borde, entonces tiene estado -2;
- si la celda (r, c) es un obstáculo, entonces tiene estado -1;
- si la celda (r, c) es una celda libre, entonces su estado es igual a su color.

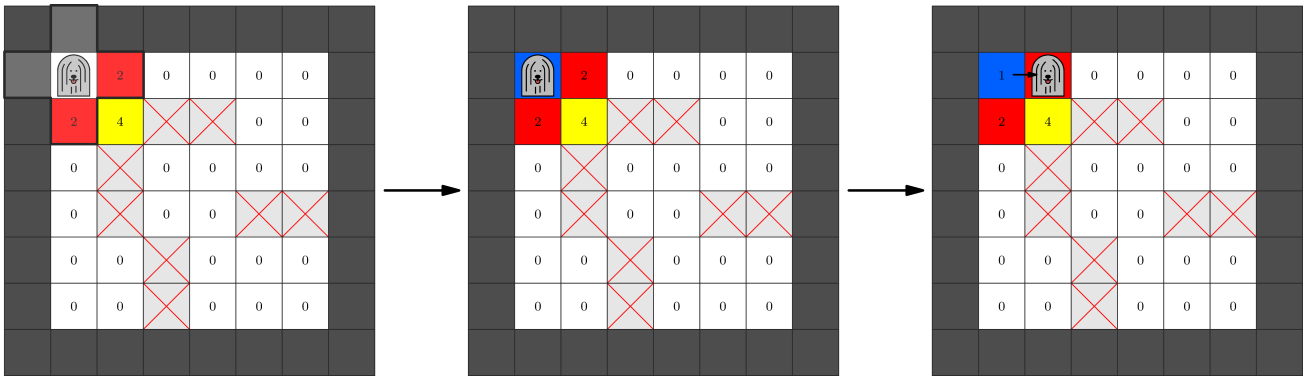
El programa de Pulibot se ejecuta como una secuencia de pasos. En cada paso, Pulibot ve los estados de la celdas cercanas y ejecuta cierta instrucción que depende de dichos estados. Más precisamente:

Suponé que al principio del paso actual de ejecución, Pulibot está en la celda (r, c), que es una celda libre. Entonces el paso de ejecución procede de la siguiente manera:

1. Primero, Pulibot computa el **arreglo de estados**, que es un arreglo $S = [S[0], S[1], S[2], S[3], S[4]]$ que consiste en el estado de la celda (r, c) y todas sus celdas adyacentes:
    - $S[0]$ es el estado de la celda (r, c).
    - $S[1]$ es el estado de la celda al oeste de (r, c).
    - $S[2]$es el estado de la celda al sur de (r, c).
    - $S[3]$ es el estado de la celda al este de (r, c).
    - $S[4]$ ies el estado de la celda al norte de (r, c).
2. Luego, Pulibot determina qué **instrucción** (Z, A) ejecutar en base al arreglo de estados S.

3. Finalmente, Pulibot ejecuta la instrucción: pinta la celda (r, c) de color Z y hace la acción A, que puede ser:
    - *quedarse* en la celda (r, c);
    - *moverse* a una de las 4 celdas adyacentes;
    - *terminar el programa*.

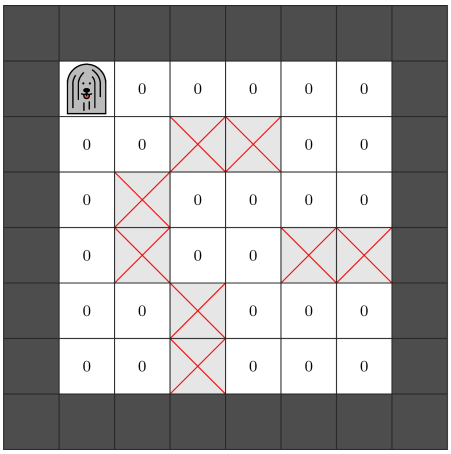
Por ejemplo, considerá la situación que se muestra a la izquiera de la figura siguiente. Pulibot se encuentra en la celda $(0, 0)$, que está pintada con el color 0. El arreglo de estados computado por Pulibot será entonces $S = [0, -2, 2, 2, -2]$. En un programa hipotético de Pulibot, al ver este arreglo de estados, puede decidir pintar la celda actual de color $Z = 1$ y moverse al este. Este proceso se muestra en la figura del medio y la de la derecha.

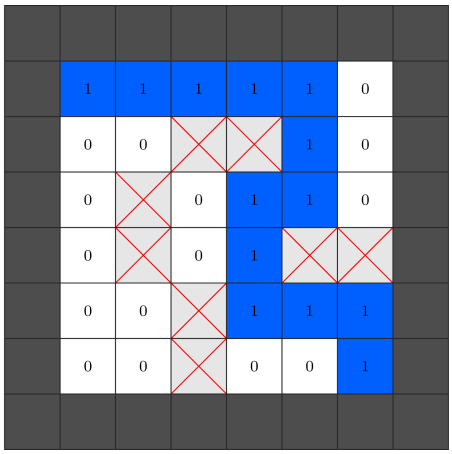


Reglas del Torneo

- Al principio, Pulibot se encuentra en la celda $(0, 0)$ y comienza a ejecutar su programa.
- No se le permite a Pulibot moverse a una celda que no esté libre.
- El programa de Pulibot debe terminar en a lo sumo $500\,000$ pasos.
- Luego de que termine el programa, los colores de las celdas libres deben ser tales que:
    - Exista un camino mínimo desde $(0, 0)$ hasta $(H - 1, W - 1)$ tal que el color de todas las celdas del camino sea 1.
    - Toda otra celda libre tiene color 0.
- Pulibot puede terminar el programa en cualquier celda libre.

Por ejemplo, en la figura siguiente se ve un laberinto posible con $H = W = 6$. La configuración inicial se muestra en la izquierda y una coloración final válida de las celdas libres se muestra a la derecha.





Detalles de Implementación

Tenés que implementar la función siguiente

```
void program_pulibot()
```

- La función debe crear el programa que ejecutará Pulibot. El programa producido debe funcionar para todos los valores de H y W y cualquier laberinto que acaten a las restricciones del problema.
- Esta función se llamará exactamente una vez por cada caso de prueba.

Para especificar el programa de Pulibot, desde `program_pulibot` podés llamar a la siguiente función:

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

- S: arreglo de longitud 5 que describe un arreglo de estados.
- Z: entero no negativo que representa un color.
- A: un caracter que representa una acción de Pulibot:
  - `H`: quedarse quieto;
  - `W`: moverse al oeste (west);
  - `S`: moverse al sur;
  - `E`: moverse al este;
  - `N`: movere al norte;
  - `T`: terminar el programa.
- Llamar a esta función le dice a Pulibot que cada vez que ve el arreglo de estados S, debe ejecutar la instrucción (Z, A).

Si llamás esta función varias veces con el mismo arreglo de estados S, el veredicto será `Output isn't correct`.

No es necesario que llames a `set_instruction` para todos los posibles arreglos de estado S. Sin embargo, si Pulibot ve un arreglo de estados en el cual no se especificó qué instrucción ejecutar, el veredicto será `Output isn't correct`.

Luego de que retorne la función `program_pulibot`, el evaluador probará el programa producido en uno o más laberintos.

El tiempo de testear el programa contra los laberintos **no** cuenta para el límite de tiempo de tu solución. El evaluador **no** es adaptativo; es decir, el conjunto de laberintos está predeterminado para cada caso de prueba.

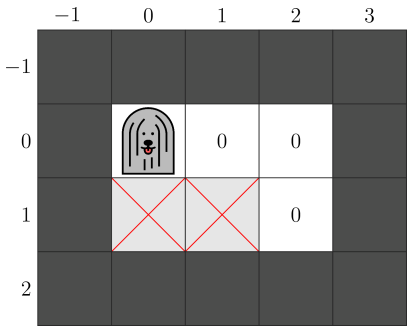
Si Pulibot viola cualquiera de las Reglas del Torneo antes de terminar su programa, el veredicto será `Output isn't correct`.

Ejemplo

La llamada a `program_pulibot` puede realizar las siguientes llamadas a `set_instruction`:

Llamada	Instrucción correspondiente al arreglo de estados S
`set_instruction([0, -2, -1, 0, -2], 1, E)`	Pintar casilla actual 1 y moverse al este
`set_instruction([0, 1, -1, 0, -2], 1, E)`	Pintar casilla actual 1 y moverse al este
`set_instruction([0, 1, 0, -2, -2], 1, S)`	Pintar casilla actual 1 y moverse al sur
`set_instruction([0, -1, -2, -2, 1], 1, T)`	Pintar casilla actual 1 y terminar el programa.

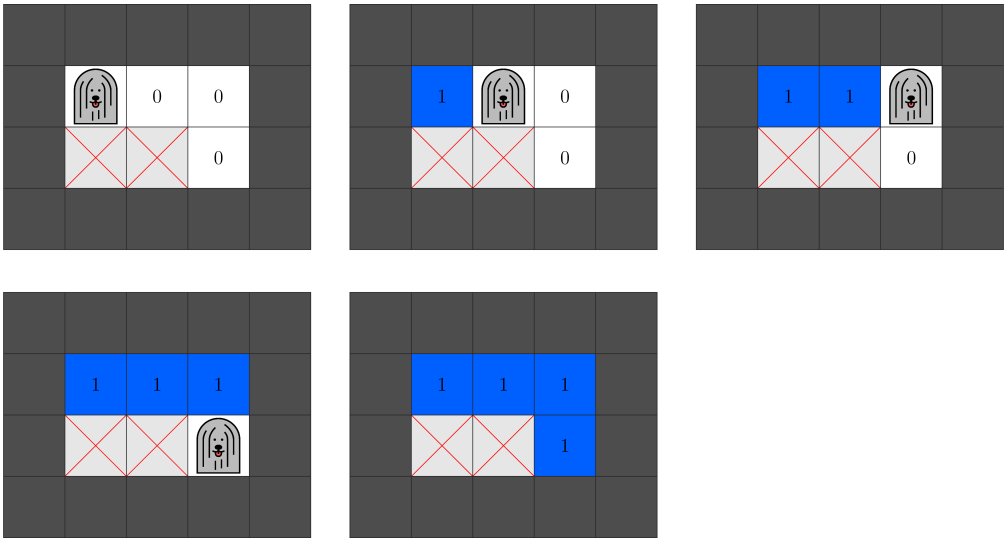
Considerá un caso donde $H = 2$ y $W = 3$, y el laberinto es el que se muestra en la figura:



Para este laberinto en particular, el programa producido se ejecuta en 4 pasos. Para este laberinto, Pulibot ve los arreglos de estados en el orden que se especificaron, y por lo tanto ejecuta las

acciones en el orden en el que se especificaron. Al ejecutar la última de estas instrucciones, se termina el programa.

La siguiente figura muestra el laberinto antes de cada una de los 4 pasos de ejecución, así como los colores finales luego de terminar el programa:



Notá que en otros laberintos válidos, este programa no necesariamente encuentra el camino mínimo. Por lo tanto, si mandás esto como solución recibirás el veredicto `Output isn't correct`.

Restricciones

$Z_{MAX} = 19$. Es decir, Pulibot puede usar colores entre 0 y 19, inclusive.

Para cada laberinto en el que se testea Pulibot,

- $2 \leq H, W \leq 15$.
- Existe al menos un camino desde $(0, 0)$ hasta $(H - 1, W - 1)$.

Subtareas

1. (6 points) No hay ningún obstáculo en el laberinto.
2. (10 points) $H = 2$
3. (18 points) Entre cada par de celdas libres hay exactamente un camino.
4. (20 points) El camino mínimo entre la celda $(0, 0)$ y la celda $(H - 1, W - 1)$ tiene longitud $H + W - 2$.
5. (46 points) Sin restricciones adicionales.

Si en cualquiera de los casos de prueba las llamadas a la función `set_instruction` o el programa de Pulibot no se comportan de acuerdo a las restricciones descritas en Detalles de Implementación, el puntaje de tu solución en la subtarea será 0.

En cada subtarea, podés obtener puntaje parcial produciendo una coloración que es casi correcta. Formalmente:

- La solución de un caso de prueba es **completa** si la coloración final satisface las Reglas del torneo
- La solución en un caso de prueba es **parcial** si la coloración final tiene la siguiente forma:
  - Existe un camino mínimo de $(0, 0)$ a $(H - 1, W - 1)$ tal que el color de cada celda del camino es 1.
  - No hay ninguna otra celda del tablero con color 1.
  - Hay alguna celda libre que tiene un color que no es ni 0 ni 1.

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

En las subtareas 1-4, se otorgará el 100% del puntaje de la subtarea si tu solución es completa y el 50% del puntaje si tu solución es parcial.

En la subtarea 5, el puntaje depende en el número de colores que Pulibot usa en el programa. Más precisamente, denotamos $Z^\star$ como el máximo valor de Z sobre todas las llamadas a `set_instructions`. El puntaje de la subtarea se calcula con la siguiente tablita:

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 del puntaje de los casos de prueba de la subtarea.

Evaluador Local

El evaluador local lee la entrada en el siguiente formato:

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

Donde m es un arreglo de H arreglos de W enteros cada uno, que describen las celdas del laberinto que no son bordes. Si $m[r][c] = 0$ entonces la celda (r, c) está libre y si $m[r][c] = 1$ la celda (r, c) es un obstáculo.

El evaluador local primero llama a `program_pulibot()`. Si el evaluador local detecta una violación del protocolo, el evaluador local imprime `Protocol Violation: <MSG>` y termina, donde `<MSG>` es alguno 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 caracter A no está en el conjunto `H`, `W`, `S`, `E`, `N` o `T`.
- `Same state array`: `set_instruction` fue llamada con el mismo arreglo S al menos dos veces.

Si no pasa nada de eso, al terminar la llamada a `program_pulibot` el evaluador local ejecuta el programa de Pulibot en el laberinto descrito en la entrada.

El evaluador local produce dos resultados:

Primero, el evaluador local imprime la secuencia de acciones del robot al archivo `robot.bin` en el directorio de trabajo. Este archivo sirve como entrada a la herramienta de visualización descrita en la sección siguiente

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

- `Unexpected state`: Si Pulibot ve un arreglo de estados para el cual no se llamó `set_instruction`.
- `Invalid move`: Pulibot se intentó mover hacia una casilla no permitida.
- `Too many steps`: Pulibot hizo $500\,000$ pasos sin que el programa termine.

Si no pasa nada de eso, sea $e[r][c]$ el estado de la celda (r, c) al final de la ejecución del programa de Pulibot. El evaluador local 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

En el paquete adjunto al problema se encuentra un archivo llamado `display.py`. Cuando este archivo es ejecutado, este script de Python muestra las acciones de Pulibot en el laberinto descrito por la entrada al evaluador local. Para poder hacer esto, el archivo `robot.bin` tiene que existir en el directorio de trabajo.

Para ejecutar el script, usá el siguiente comando

```
python3 display.py
```

Se mostrará una simple interfaz gráfica. Las principales características son:

- Se puede ver el estado de todo el laberinto. La posición actual de Pulibot se muestra resaltada con un rectángulo.
- Podés avanzar y retroceder sobre el historial de pasos que hizo Pulibot apretando las flechas o apretando sus hotkeys. También podés ir a un paso específico de la ejecución.
- El paso de ejecución actual se muestra en la parte inferior. Se muestra el estado actual y la instrucción a ejecutar. Luego del útimo paso, se muestra o bien un mensaje de error del evaluador local, o bien `Terminated` si el programa terminó con éxito.
- A cada número que representa un color, podés asignarle un color de fondo y un texto. El texto asignado tiene que ser corto y aparecerá en cada celda con ese color. Podés asignar el fondo y el texto de un color de dos formas:
    - seleccionandolos en la ventana de diálogo luego de hacer click en el botón `Colors`.
    - Editando el contenido del archivo `colors.txt`.
- Para recargar el contenido de `robot.bin`, usá el botón `Reload`. Esto sirve si el contenido del archivo `robot.bin` cambió.