Robot Contest

Investigadores de AI da Universidade de Szeged estão a organizar um concurso de programação de robôs. A tua amiga Hanga decidiu participar neste concurso. O objetivo do concurso é programar o derradeiro *Pulibot*, chamado assim em honra da inteligência da famosa raça de cão pastor húngaro, o Puli.

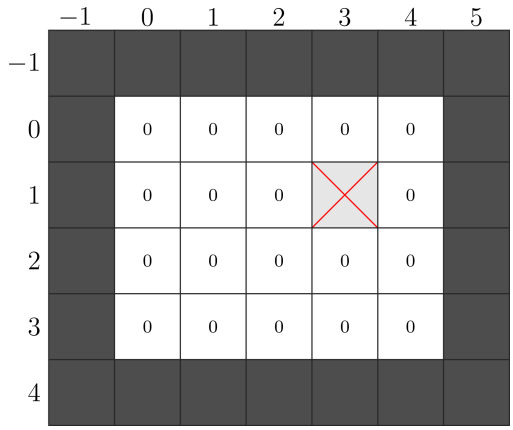
Pulibot vai ser testado num labirinto que consiste numa grelha de $(H+2) \times (W+2)$ células. As linhas da grelha são numeradas de -1 a H de norte para sul e as colunas da grelha são numeradas de -1 a W de oeste para este. Referimo-nos à célula na linha r e na coluna c da grelha ($-1 \leq r \leq H$, $-1 \leq c \leq W$) como célula (r, c).

Considera uma célula (r, c) tal que $0 \leq r < H$ e $0 \leq c < W$. Há 4 células **adjacentes** à célula (r, c):

- a célula $(r, c-1)$ é referida como a célula a **oeste** da célula (r, c);
- a célula $(r+1, c)$ é referida como a célula a **sul** da célula (r, c);
- a célula $(r, c+1)$ é referida como a célula a **este** da célula (r, c);
- a célula $(r-1, c)$ é referida como a célula a **norte** da célula (r, c);

Uma célula (r, c) é chamada de célula de **fronteira** do labirinto se $r = -1$ ou $r = H$ ou $c = -1$ ou $c = W$ se verifica. Cada célula que não seja uma célula de fronteira ou é uma célula **obstáculo** ou uma célula **vazia**. Para além disso, cada célula vazia tem uma **cor**, representada por um inteiro não negativo entre 0 e Z_{MAX}, inclusive. No início, a cor de cada célula vazia é 0.

Por exemplo, considera o labirinto com $H = 4$ e $W = 5$, contendo uma única célula obstáculo $(1, 3)$:



A única célula obstáculo está desenhada com uma cruz. As células de fronteira do labirinto estão a cinzento. O números em cada célula vazia representa a sua respetiva cor.

Um **caminho** de comprimento ℓ ($\ell > 0$) da cécula (r_0, c_0) à célula (r_ℓ, c_ℓ) é uma sequência de células *vazias* $(r_0, c_0), (r_1, c_1), \ldots, (r_\ell, c_\ell)$, distintas dois a dois, em que para cada i ($0 \leq i < \ell$) as células (r_i, c_i) e (r_{i+1}, c_{i+1}) são adjacentes.

Nota que um caminho de comprimento ℓ tem exatamente $\ell + 1$ células.

No concurso, os investigadores montaram um labirinto onde existe pelo menos um caminho da célula $(0, 0)$ até à célula $(H-1, W-1)$. Nota que as células $(0, 0)$ e $(H-1, W-1)$ são garantidamente vazias.

A Hanga não sabe quais das células do labirinto estão vazias e quais têm obstáculos.

A tua tarefa é ajudar a Hanga a programar o Pulibot tal que este seja capaz de encontrar um *caminho mínimo* (isto é, um dos caminhos de comprimento mínimo) da célula $(0, 0)$ até á célula $(H-1, W-1)$ deste labirinto desconhecido construído pelos investigadores. A especificação do Pulibot e as regras do concurso são descritas de seguida.

Nota que a última secção do enunciado deste problema descreve uma ferramenta de visualização que podes usar para visualizar o Pulibot.

Especificação do Pulibot

Definimos o **estado** de uma célula (r, c), para cada $-1 \leq r \leq H$ and $-1 \leq c \leq W$, como um inteiro tal que :

- se a célula (r, c) é uma célula de fronteira então o seu estado é -2;
- se a célula (r, c) é uma célula obstaculo então o seu estado é -1;
- se a célula (r, c) é uma célula vazia então o seu estado é a cor da célula.

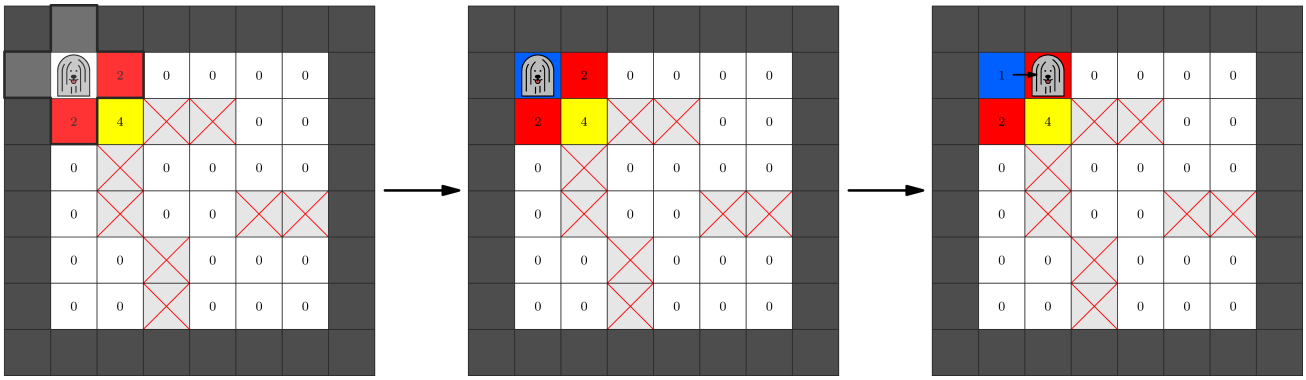
O programa do Pulibot é executado como uma sequência da passos. Em cada passo, o Pulibot vê o estado das células das células proximas dele e executa uma instrução. A instrução realizada depende dos estados vistos. Uma descrição mais precisa segue-se.

Supõe que no início do passo atual, o Pulibot está na célula (r, c), que é uma célula vazia. Um passo consiste no seguinte:

1. Primeiro, o Pulibot vê o atual **array de estado**, isto é, o array $S = [S[0], S[1], S[2], S[3], S[4]]$, que consiste no estado da célula (r, c) e de todas as células adjacentes:
    - $S[0]$ é o estado da célula (r, c).
    - $S[1]$ é o estado da célula a oeste.
    - $S[2]$ é o estado da célula a sul.
    - $S[3]$ é o estado da célula a este.

    - $S[4]$ é o estado da célula a norte.
2. De seguida, o Pulibot executa a **instrução** (Z, A) que corresponde ao array de estado visualizado.
3. Por fim, o Pulibot executa essa instrução: muda a cor da célula (r, c) para a cor Z e de seguida executa a ação A, que é uma das seguintes ações:
    - *fica* na célula (r, c);
    - *mover* para uma das 4 células adjacentes;
    - *terminada o programa*.

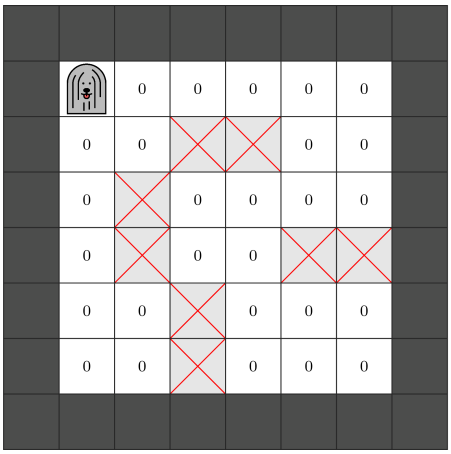
Por exemplo, considera o cenário representado do lado esquerdo da figura seguinte. O Pulibot está na célula $(0, 0)$ com a cor 0. O Pulibot vê o array de estado $S = [0, -2, 2, 2, -2]$. O Pulibot pode ter um programa, que ao ver este array de estado, muda a cor da célula atual para $Z = 1$ e depois move-se para este, como representado no meio e na direita da figura seguinte:

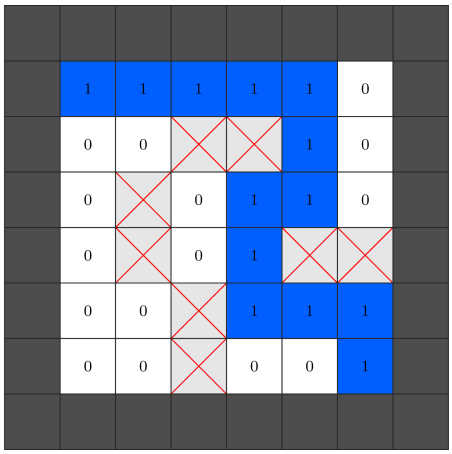


Regras do Concurso de Robôs

- No início, o Pulibot é colocado na célula $(0, 0)$ e começa a correr o seu programa.
- O Pulibot não se pode mover para uma célula não vazia.
- O programa do Pulibot deve terminar ao fim de no máximo $500\,000$ passos.
- Após o programa do Pulibot terminar, as células vazias do labirinto devem ser coloridas de forma a que:
    - Existe um caminho mínimo de $(0, 0)$ a $(H - 1, W - 1)$ tal que a cor de cada célula pertencente a esse caminho é 1.
    - Todas as outras células vazias têm a cor 0.
- O Pulibot pode terminar o seu programa em qualquer célula vazia.

Por exemplo, a figura seguinte exemplifica um possível labirinto com $H = W = 6$. A configuração inicial está representada à esquerda e uma coloração aceitável das células vazias após o programar terminar está representada à direita:





Detalhes de Implementação

Deves implementar a seguinte função.

```
void program_pulibot()
```

- Esta função deve construir o programa do Pulibot. Este programa deve funcionar corretamente para todos os valores de H e W e para todo o labirinto que verifica as condições do problema.
- Esta função é chamada exatamente uma vez por cada caso de teste.

Esta função pode fazer chamada à seguinte função para construir o programa do Pulibot:

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

- S: array de tamanho 5 que descreve o array de estado.
- Z: um inteiro não negativo que representa uma cor.
- A: um único caracter que representação a ação do Pulibot da seguinte forma:
  - `H`: fica na mesma célula;
  - `W`: move-te para oeste;
  - `S`: move-te para sul;
  - `E`: move-te para este;
  - `N`: move-te para norte;
  - `T`: termina o programa.
- Chamar esta função ensina o Pulibot que ao vêr o array de estado S deve executar a instrução (Z, A).

Chamar esta função múltiplas vezes com o mesmo array de estado S vai resultar em `Output isn't correct`.

Não é necessário chamar a função `set_instruction` para cada possível array de estado S. No entanto, se o Pulibot vir um array de estado para o qual uma instrução não foi definida, vai

receber um `Output isn't correct`.

Após o `program_pulibot` ser completado, o avaliador vai experimentar o programa do Pulibot em pelo menos um labirinto. Estas invocações *não* contam para o limite de tempo da tua solução. O avaliador *não* é adaptativo, isto é, o conjunto de labirinto já está pré-definido para casa caso de teste.

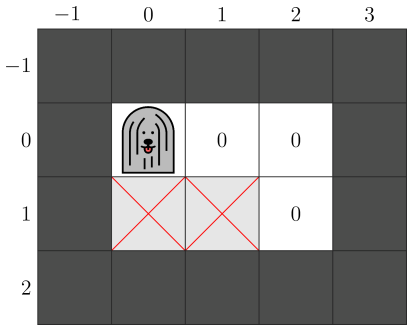
Se o Pulibot violar qualquer Regra do Concurso de Robôs durante a sua execução vais receber um `Output isn't correct`.

Exemplo

A função `program_pulibot` pode fazer chamadas a `set_instruction` da seguinte forma:

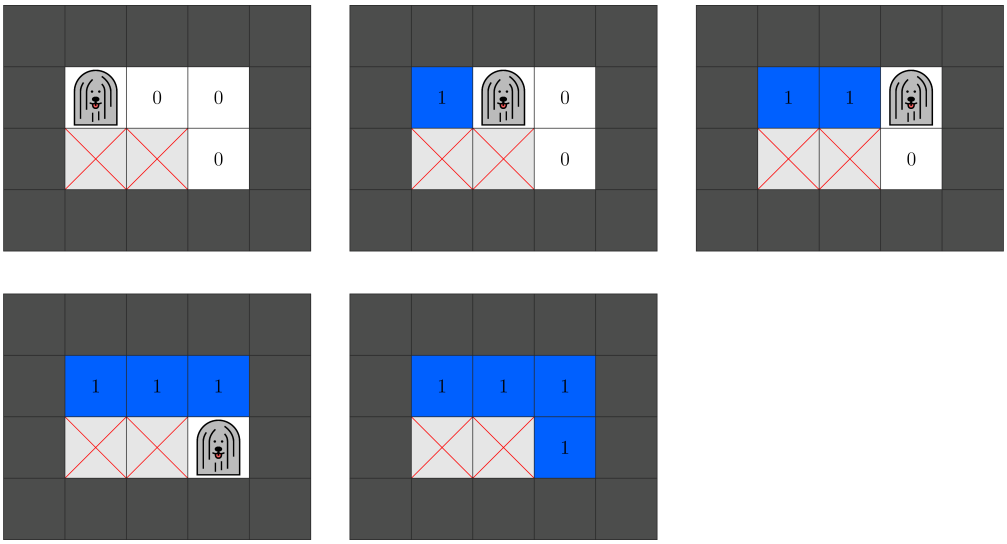
Chamada	Instrução para o array de estado S
`set_instruction([0, -2, -1, 0, -2], 1, E)`	Muda a cor para 1 e move-te para este
`set_instruction([0, 1, -1, 0, -2], 1, E)`	Muda a cor para 1 e move-te para este
`set_instruction([0, 1, 0, -2, -2], 1, S)`	Muda a cor para 1 e move-te para sul
`set_instruction([0, -1, -2, -2, 1], 1, T)`	Muda a cor para 1 e termina o programa

Considera o cenário onde $H = 2$ e $W = 3$, e o labirinto está representado na seguinte figura.



Para este labirinto em particular o programa do Pulibot corre em quatro passos. Os arrays de estado visto pelo Pulibot e as instruções realizadas corresponde exatamente às quatro chamadas a `set_instruction` feitas anteriormente, pela mesmo ordem. A última destas instruções termina o programa.

A seguinte figura mostra o labirinto antes de cada um dos quatro passos e as cores finais após o programa terminar.



No entanto, nota que este programa de 4 instruções pode não encontrar um caminho mínimo noutros labirintos válidos. Consequemente, se submetido, receberia um `Output isn't correct`.

Restrições

$Z_{MAX} = 19$. Assim, o Pulibot pode usar cores de 0 a 19, inclusive.

Para cada labirinto usado para testar o Pulibot:

- $2 \leq H, W \leq 15$
- Existe pelo menos um caminho da célula $(0, 0)$ para a célula $(H - 1, W - 1)$.

Subtarefas

1. (6 pontos) Não existe nenhuma célula obstáculo no labirinto
2. (10 pontos) $H = 2$
3. (18 pontos) Existe exatamente um caminho entre cada par de células vazias.
4. (20 pontos) Cada caminho mínimo da $(0, 0)$ para a célula $(H - 1, W - 1)$ tem comprimento $H + W - 2$.
5. (46 pontos) Nenhuma restrição adicional.

Se, em qualquer um dos casos de teste, chamadas à função `set_instruction` ou o programa do Pulibot não respeitam as resctrições descritas nos Detalhes de Implementação, a tua pontuação para essa subtarefa será 0.

Em cada subtarefa, podes obter uma pontuação parcial ao construir uma coloração que está quase correta.

Formalmente:

- A solução a um caso de teste é **completa** se a coloração final das células vazias respeita as Regras do Concurso de Robôs.

- A solução de um caso de teste é **parcial** se a coloração final tem o seguinte aspeto:
    - Existe um caminho mínimo de $(0, 0)$ a $(H - 1, W - 1)$ tal que a cor de cada célula pertencente a esse caminho é 1.
        - Não existe nenhuma outra célula vazia na grelha com a cor 1.
        - Pelo menos uma célula vazia da grelha tem uma cor diferente de 0 ou 1.

Se a tua solução para um caso de teste não for nem completa nem parcial, a tua pontuação no caso de teste correspondente será 0.

Nas subtarefas 1-4, a pontuação para uma solução completa é 100% e a pontuação de solução parcial de cada caso teste é 50% dos pontos da respetiva subtarefa.

Na subtarefa 5, a tua pontuação depende do número de cores usadas pelo programa do Pulibot. Mais precisamente, seja $Z^\star$ o valor máximo de Z entre todas as chamadas a `set_instruction`. A pontuação de cada caso de teste é calculada de acordo com a seguinte tabela:

Condição	Pontuação (completa)	Pontuação (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

A pontuação para cada subtarefa é o mínimo das pontuações para cada caso de teste dessa subtarefa.

Avaliador Exemplo

O avaliador exemplo lê o input segundo o seguinte formato:

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

Aqui, m é um array de H arrays de W inteiros, descrevendo as células do labirinto que não as de fronteira. $m[r][c] = 0$ se a célula (r, c) é uma célula vazia e $m[r][c] = 1$ se a célula (r, c) é uma célula obstáculo.

O avaliador exemplo primeiro chama `program_pulibot()`. Se o avaliador exemplo detetar uma violação do protocolo, o avaliado exemplo imprime `Protocol Violation: <MSG>` e termina, onde `<MSG>` é uma das seguintes mensagens de erro:

- `Invalid array`: $-2 \leq S[i] \leq Z_{MAX}$ não se verifica para algum i ou o comprimento de S não é 5.
- `Invalid color`: $0 \leq Z \leq Z_{MAX}$ não se verifica.
- `Invalid action`: o caracter A não é um de `H`, `W`, `S`, `E`, `N` or `T`.
- `Same state array`: `set_instruction` foi chamado para o mesmo array de estado S pelo menos duas vezes.

Caso contrário, quando `program_pulibot` acabar, o avaliador exemplo irá correr o programa do Pulibot no labirinto descrito pelo input.

O avaliador exemplo produz dois outputs.

Primeiro, o avaliador exemplo escreve um registo das ações do Pulibot no ficheiro `robot.bin` no diretório de trabalho. Este ficheiro serve como input para a ferramenta de visualização descrita na secção seguinte.

Segundo, se o programa do Pulibot não terminar com sucesso, o avaliador exemplo imprime uma das seguintes mensagens de erro:

- `Unexpected state`: o Pulibot viu um array de estado par ao qual `set_instruction` não foi chamado.
- `Invalid move`: a execução da instrução resultou no Pulibot mover-se para uma célula não vazia.
- `Too many steps`: Pulibot executou $500\,000$ passos sem terminar o seu programa

Caso contrário, seja $e[r][c]$ o estado da célula (r, c) após o programa do Pulibot terminar. O avaliador exemplo imprime H linhas no seguinte formato:

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

Ferramenta de Visualização

O arquivo em anexo para este problema contém um ficheiro chamado `display.py`. Quando chamado, este script de Python mostra as ações do Pulibot no labirinto descrito pelo input no avaliador exemplo. Para tal, o ficheiro binário `robot.bin` deve estar no diretório de trabalho.

Para chamar este script, executa o seguinte comando.

```
python3 display.py
```

Um interface gráfico simples aparece. As suas principais qualidades são:

- Podes observar o estado do labirinto todo. A atual posição do Pulibot está destacada por um quadrado.

- Podes navegar pelos passos do Pulibot ao clicar nas setas ou pressionando as suas teclas de atalho. Também podes saltar para um passo específico.
- O próximo passo do programa do Pulibot pode ser visto em baixo. Mostra o qual array de estado e a instrução que vai ser executada. Após o passo final, ou mostra uma das mensagens de erro do avaliador ou mostra `Terminated` se o programa terminou com sucesso.
- Para cada número que representa uma cor, podes atribuir uma cor de fundo, tal como uma etiqueta. A etiqueta é uma pequena string que deverá aparecer em cada célula que tem essa cor. Podes atribuir cores de fundo e etiquetas numa das seguintes formas:
    - Atribuí-las na janela de diálogo depois de clicares no botão `Colors`.
    - Editar o conteúdo do ficheiro `colors.txt`.
- Para atualizar o `robot.bin`, usa o botão `Reload`. É útil se o conteúdo de `robot.bin` tiver sido alterado.