Concours de robotique

Les chercheurs en IA à l'Université de Szeged organisent un concours de programmation de robot. Votre amie, Hanga, a décidé de participer à ce concours. L'objectif est de programmer le *Pulibot* en s'inspirant de la grande intelligence d'une celèbre race de chien, le Puli Berger Hongrois.

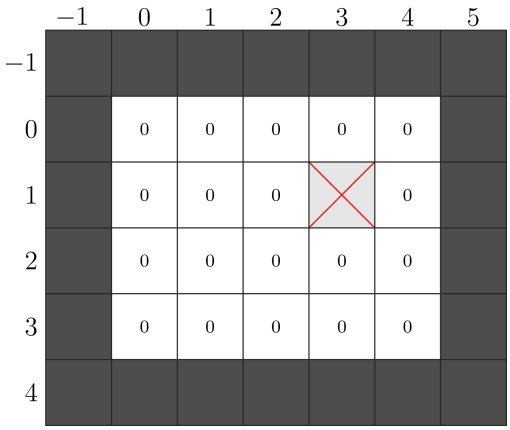
Pulibot sera testé sur un labyrinthe constitué d'une grille de $(H+2) \times (W+2)$ cases. Les lignes de la grille sont numérotées de -1 à H du nord au sud et les colonnes sont numérotées de -1 à W d'ouest en est. On parlera de la case située à la ligne r et à la colonne c ($-1 \leq r \leq H$, $-1 \leq c \leq W$) comme de la case (r, c).

Considérons une case (r, c) telle que $0 \leq r < H$ et $0 \leq c < W$. Il y a 4 cases **adjacentes** à la case (r, c) :

- la case $(r, c-1)$ est appelée la case **ouest** de la case (r, c);
- la case $(r+1, c)$ est appelée la case **sud** de la case (r, c);
- la case $(r, c+1)$ est appelée la case **est** de la case (r, c);
- la case $(r-1, c)$ est appelée la case **nord** de la case (r, c).

On dit que la case (r, c) est une case **bordure** du labyrinthe si $r = -1$ ou $r = H$ ou $c = -1$ ou $c = W$. Chaque case qui n'est pas une case bordure du labyrinthe est soit une case **d'obstacle**, soit une case **vide**. De plus, chaque case vide est d'une certaine **couleur**, représentée par un entier positif ou nul entre 0 et Z_{MAX} inclus. Initialement, la couleur de chaque case vide est 0.

Par exemple, considérons un labyrinthe avec $H = 4$ et $W = 5$, contenant une unique case d'obstacle $(1, 3)$:



L'unique case d'obstacle est représentée par une croix. Les cases bordure du labyrinthe sont grisées. Le nombre écrit dans chaque case vide représente sa couleur.

Un **chemin** de taille ℓ ($\ell > 0$) de la case (r_0, c_0) à la case (r_ℓ, c_ℓ) est une suite de cases *vides* distinctes $(r_0, c_0), (r_1, c_1), \ldots, (r_\ell, c_\ell)$ dans laquelle pour chaque i ($0 \leq i < \ell$) les cases (r_i, c_i) et (r_{i+1}, c_{i+1}) sont adjacentes.

Notez qu'un chemin de longueur ℓ contient exactement $\ell + 1$ cases.

Pour le concours, les chercheurs ont conçu un labyrinthe dans lequel il existe au moins un chemin de la case $(0, 0)$ à la case $(H - 1, W - 1)$. Notez qu'il est donc garanti que les cases $(0, 0)$ et $(H - 1, W - 1)$ sont vides.

Hanga ne sait pas quelles cases du labyrinthe sont des cases vides et lesquelles sont des cases d'obstacle.

Votre tâche est d'aider Hanga à programmer Pulibot afin qu'il soit capable de trouver un *plus court chemin* (c'est-à-dire un chemin de longueur minimale) de la case $(0, 0)$ à la case $(H - 1, W - 1)$ dans le labyrinthe inconnu conçu par les chercheurs. Les spécifications de Pulibot et les règles du concours sont décrites ci-dessous.

Notez que la dernière section de cet énoncé décrit un **outil de visualisation** que vous pouvez utiliser pour visualiser Pulibot.

Spécifications de Pulibot

Définissons l'**état** d'une case (r, c) pour chaque $-1 \leq r \leq H$ et $-1 \leq c \leq W$:

- si la case (r, c) est une case bordure alors son état est -2;
- si la case (r, c) est une case d'obstacle alors son état est -1;
- si la case (r, c) est une case vide alors son état est la couleur de la case.

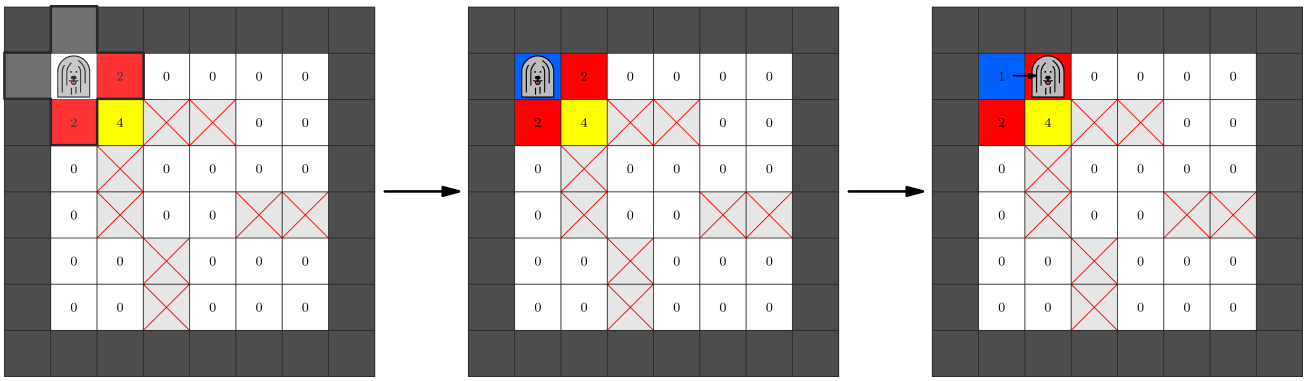
Le programme de Pulibot est exécuté comme une succession d'étapes. À chaque étape, Pulibot reconnaît les états des cases aux alentours et exécute une instruction. L'instruction qu'il exécute est déterminée par les états reconnus. Une description plus précise suit.

Supposons qu'au début de l'étape actuelle, Pulibot est à la case (r, c) qui est une case vide. L'étape est exécutée ainsi :

1. Premièrement, Pulibot reconnaît le **tableau d'états** actuel, c'est-à-dire le tableau $S = [S[0], S[1], S[2], S[3], S[4]]$ constitué de l'état de la case (r, c) et des cases adjacentes :
   - $S[0]$ est l'état de la case (r, c).
   - $S[1]$ est l'état de la case à l'ouest.
   - $S[2]$ est l'état de la case au sud.
   - $S[3]$ est l'état de la case à l'est.
   - $S[4]$ est l'état de la case au nord.
2. Ensuite, Pulibot détermine l'**instruction** (Z, A) qui correspond au tableau d'états reconnu.

3. Finalement, Pulibot exécute cette instruction : il change la couleur de la case (r, c) en la couleur Z puis réalise l'action A qui est l'une des actions suivantes :
    - *rester* sur la case (r, c);
    - *se déplacer* sur l'une des 4 cases adjacentes;
    - *terminer le programme*.

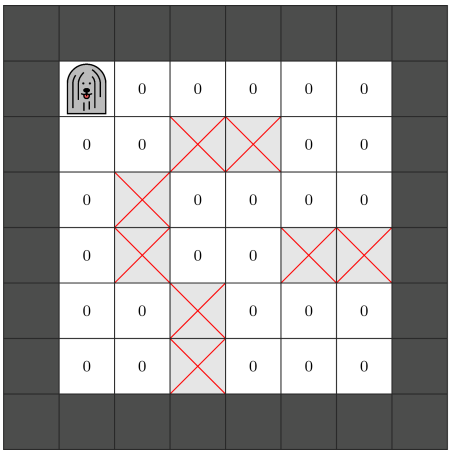
Par exemple, considérons le scénario représenté à gauche de la figure suivante. Pulibot est actuellement sur la case $(0, 0)$ avec la couleur 0. Pulibot reconnaît le tableau d'états $S = [0, -2, 2, 2, -2]$. Pulibot pourrait avoir un programme qui, après avoir reconnu ce tableau, change la couleur de la case actuelle en $Z = 1$ et se déplace vers l'est, comme montré au milieu et à droite de la figure :

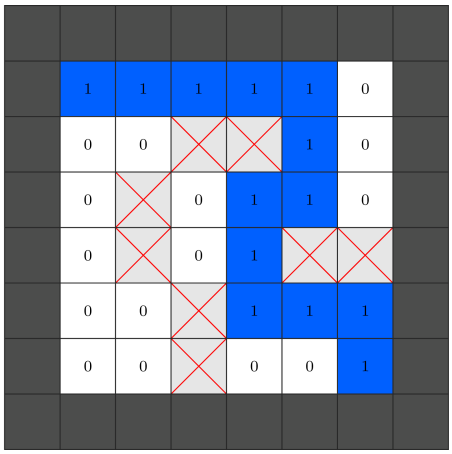


Règles du concours de robotique

- Au début, Pulibot est placé sur la case $(0, 0)$ et commence à exécuter son programme.
- Pulibot n'est pas autorisé à se déplacer vers une case qui n'est pas vide.
- Le programme de Pulibot doit terminer après au plus $500\,000$ étapes.
- Après la terminaison du programme de Pulibot, les cases vides du labyrinthe doivent être coloriées de sorte à ce que :
    - Il existe un plus court chemin de $(0, 0)$ à $(H - 1, W - 1)$ pour lequel la couleur de chaque case du chemin est 1.
    - Toutes les autres cases vides sont de couleur 0.
- Pulibot peut terminer son programme sur n'importe quelle case vide.

Par exemple, la figure suivante illustre un labyrinthe possible avec $H = W = 6$.
La configuration de départ est représentée à gauche et un coloriage acceptable des cases vides (après la terminaison du programme) est représenté à droite :





Détails d'implémentation

Vous devez implémenter la fonction suivante.

```
void program_pulibot()
```

- Cette fonction doit produire le programme de Pulibot. Ce programme doit fonctionner correctement pour toutes les valeurs de H et W et pour tout labyrinthe satisfaisant les contraintes de l'énoncé.
- Cette fonction est appelée exactement une fois dans chaque test.

Cette fonction peut faire des appels à la fonction suivante pour produire le programme de Pulibot :

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

- S: tableau de taille 5 décrivant un tableau d'états.
- Z: un entier positif ou nul représentant une couleur.
- A: un unique caractère représentant une action de Pulibot comme suit :
  - `H`: rester immobile;
  - `W`: se déplacer vers l'ouest;
  - `S`: se déplacer vers le sud;
  - `E`: se déplacer vers l'est;
  - `N`: se déplacer vers le nord;
  - `T`: terminer le programme.
- Appeler cette fonction indique à Pulibot qu'après avoir reconnu le tableau d'états S il doit exécuter l'instruction (Z, A).

Si vous appelez cette fonction plusieurs fois avec le même tableau d'états S, vous obtiendrez le verdict `Sortie incorrecte`.

Il n'est pas demandé d'appeler `set_instruction` avec chaque tableau d'états S possible. Cependant, si à un moment Pulibot reconnaît un tableau d'états pour lequel aucune instruction n'a

été choisie, vous obtiendrez le verdict `Sortie incorrecte`.

Une fois que l'appel à votre fonction `program_pulibot` est terminé, l'évaluateur (grader) invoque le programme de Pulibot sur un ou plusieurs labyrinthes. Ces invocations ne comptent *pas* dans la limite de temps pour votre solution. L'évaluateur (grader) n'est *pas* adaptatif, l'ensemble de labyrinthes est prédéfini dans chaque test.

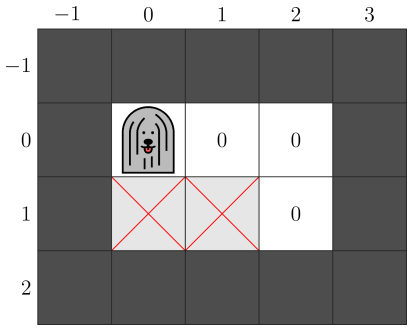
Si Pulibot viole n'importe quelle règle du concours de robotique avant de terminer son programme, vous obtiendrez le verdict `Sortie incorrecte`.

Exemple

La fonction `program_pulibot` peut faire des appels à `set_instruction` comme suit :

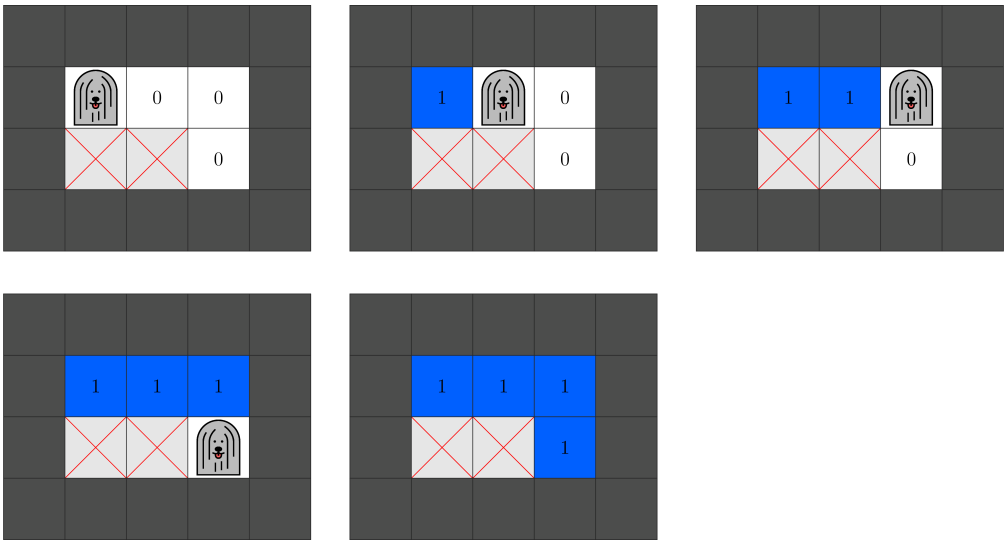
Appel	Instruction pour le tableau d'états S
`set_instruction([0, -2, -1, 0, -2], 1, E)`	Changer la couleur en 1 et se déplacer vers l'est
`set_instruction([0, 1, -1, 0, -2], 1, E)`	Changer la couleur en 1 et se déplacer vers l'est
`set_instruction([0, 1, 0, -2, -2], 1, S)`	Changer la couleur en 1 et se déplacer vers le sud
`set_instruction([0, -1, -2, -2, 1], 1, T)`	Changer la couleur en 1 et terminer le programme

Considérons un scénario où $H = 2, W = 3$ et le labyrinthe est illustré dans la figure suivante.



Pour ce labyrinthe en particulier, le programme de Pulibot s'exécute en quatre étapes. Les tableaux d'états que Pulibot reconnaît et les instructions qu'il effectue correspondent exactement aux quatre appels à `set_instruction` fait plus haut, dans l'ordre. La dernière de ces instructions termine le programme.

La figure suivante montre le labyrinthe avant chacune des quatre étapes et les couleurs finales après terminaison.



Cependant, notez que ce programme de 4 instructions pourrait ne pas trouver un plus court chemin dans d'autres labyrinthes valides. Ainsi, s'il est soumis, il obtiendra le verdict `Sortie incorrecte`.

Contraintes

$Z_{MAX} = 19$. Ainsi, Pulibot peut utiliser les couleurs de 0 à 19 inclus.

Pour chaque labyrinthe utilisé pour tester Pulibot :

- $2 \leq H, W \leq 15$
- Il y a au moins un chemin de la case $(0, 0)$ à la case $(H - 1, W - 1)$.

Sous-tâches

1. (6 points) Il n'y a pas de case d'obstacle dans le labyrinthe.
2. (10 points) $H = 2$
3. (18 points) Il y a exactement un chemin entre chaque paire de cases vides.
4. (20 points) Chaque plus court chemin de la case $(0, 0)$ à la case $(H - 1, W - 1)$ est de longueur $H + W - 2$.
5. (46 points) Aucune contrainte supplémentaire.

Si, dans un des tests, les appels à la fonction `set_instruction` ou le programme de Pulibot (au cours de son exécution) ne sont pas conformes aux contraintes décrites dans les Détails d'implémentation, le score de votre solution pour cette sous-tâche sera 0.

Dans chaque sous-tâche, vous pouvez obtenir un score partiel en produisant un coloriage qui est presque correct.

Formellement :

- La solution d'un test est **complète** si le coloriage final des cases vides satisfait les règles du concours de robotique.
- La solution d'un test est **partielle** si le coloriage final satisfait les propriétés suivantes :
  - Il existe un plus court chemin de $(0, 0)$ à $(H-1, W-1)$ pour lequel la couleur de chaque case du chemin est 1.
  - Il n'y a aucune autre case de la grille ayant la couleur 1.
  - Une ou plusieurs cases vides dans la grille ont une couleur autre que 0 ou 1.

Si la solution à un test est ni complète ni partielle, votre score pour le test correspondant sera 0.

Dans les sous-tâches 1-4, le score pour une solution complète est 100% et le score pour une solution partielle est 50% des points de sa sous-tâche.

Dans la sous-tâche 5, votre score dépend du nombre de couleurs utilisées dans le programme de Pulibot. Plus précisément, notons $Z^\star$ la valeur maximale de Z parmi tous les appels à `set_instruction`. Le score de chaque test est calculé selon le tableau suivant :

Condition	Score (complete)	Score (partial)
$11 \leq Z^\star \leq 19$	$20 + (19 - Z^\star)$	$12 + (19 - Z^\star)$
$Z^\star = 10$	31	23
$Z^\star = 9$	34	26
$Z^\star = 8$	38	29
$Z^\star = 7$	42	32
$Z^\star \leq 6$	46	36

Le score pour chaque sous-tâche est le score minimum parmi les tests de la sous-tâche.

Évaluateur d'exemple (grader)

L'évaluateur d'exemple (grader) lit l'entrée dans le format suivant :

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

Ici, m est un tableau de H tableaux de W entiers, décrivant les cases vides et les cases d'obstacle du labyrinthe (toutes les cases sauf les cases bordure). $m[r][c] = 0$ si la case (r, c) est une case vide et $m[r][c] = 1$ si la case (r, c) est une case d'obstacle.

L'évaluateur d'exemple commence par appeler `program_pulibot()`. Si l'évaluateur d'exemple (grader) détecte une violation de protocole, il affiche `Protocol Violation: <MSG>` et termine, où `<MSG>` est un des messages d'erreur suivant :

- `Invalid array`: $-2 \leq S[i] \leq Z_{MAX}$ n'est pas satisfait pour un certain i ou la taille de S n'est pas 5.
- `Invalid color`: $0 \leq Z \leq Z_{MAX}$ n'est pas satisfait
- `Invalid action`: le caractère A n'est pas égal à `H`, `W`, `S`, `E`, `N` ou `T`.
- `Same state array`: `set_instruction` a été appelé avec le même tableau S au moins deux fois.

Sinon, quand l'appel à `program_pulibot` est terminé, l'évaluateur d'exemple (grader) exécute le programme de Pulibot dans le labyrinthe décrit par l'entrée.

L'évaluateur d'exemple produit deux sorties.

Premièrement, l'évaluateur d'exemple écrit un log (une trace) des actions de Pulibot dans le fichier `robot.bin` dans le dossier courant. Ce fichier sert d'entrée à **l'outil de visualisation** décrit dans la section suivante.

Deuxièmement, si le programme de Pulibot ne termine pas avec succès, l'évaluateur d'exemple affiche l'un des messages d'erreur suivant :

- `Unexpected state`: Pulibot reconnaît un tableau d'états avec lequel `set_instruction` n'a jamais été appelée.
- `Invalid move`: effectuer une action déplace Pulibot vers une case non-vide.
- `Too many steps`: Pulibot effectue $500\,000$ étapes sans terminer son programme.

Sinon, soit $e[r][c]$ l'état de la case (r, c) après que le programme de Pulibot a terminé. L'évaluateur d'exemple affiche H lignes dans le format suivant :

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

Outil de visualisation

L'archive jointe pour ce sujet contient un fichier nommé `display.py`. S'il est exécuté, ce programme Python affichera les étapes de Pulibot dans le labyrinthe décrit par l'entrée de l'évaluateur. Pour que cela fonctionne, le fichier binaire `robot.bin` doit être présent dans le dossier depuis lequel vous lancez la commande.

Pour exécuter le script, utilisez la commande suivante :

```
python3 display.py
```

Une interface graphique simple s'affiche. Les fonctionnalités principales sont les suivantes :

- Vous pouvez observer l'état de tout le labyrinthe. La position actuelle de Pulibot est indiquée par un rectangle.

- Vous pouvez naviguer à travers les actions de Pulibot en cliquant sur les flèches ou en appuyant sur les touches correspondantes. Vous pouvez aussi aller directement à une action spécifique.
- L'action suivante du programme de Pulibot est affichée en bas. Cela montre le tableau d'états actuels et l'instruction qu'il va exécuter. Après l'action finale, il affiche soit l'un des messages d'erreur de l'évaluateur, ou `Terminated` si le programme s'est terminé avec succès.
- Pour chaque nombre qui représente une couleur, vous pouvez associer une couleur de fond, ainsi qu'un texte. Le texte est une courte chaîne de caractères qui apparaîtra dans chaque case ayant cette couleur. Vous pouvez associer les couleurs de fonds et les textes de l'une des manières suivantes :
    - En les paramétrant dans une fenêtre de dialogue après avoir cliqué sur le bouton `Colors`.
    - En éditant le contenu du fichier `colors.txt`.
- Pour recharger `robot.bin`, utilisez le bouton `Reload`. Ceci est utile si le contenu de `robot.bin` a changé.