ロボットコンテスト (Robot Contest)

セゲド大学の AI 研究者たちはロボットプログラミングコンテストを開催している． あなたの友人である Hanga はこのコンテストに参加することにした．このコンテストの目的は，究極の *Pulibot* を作ることである．とても賢いハンガリーの牧羊犬，Puli に敬意を評してこの名が名付けられている．

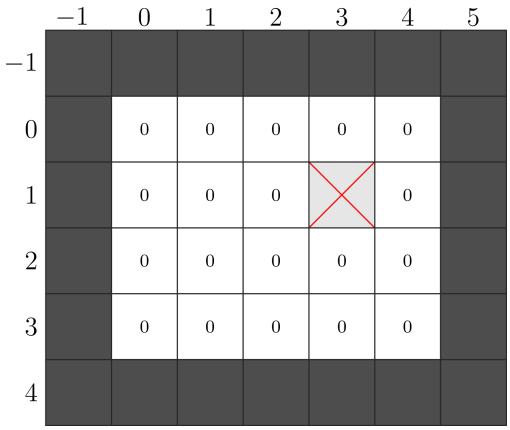
Pulibot は $(H+2) \times (W+2)$ マスのグリッド上にある迷路を使ってテストされる． グリッドの行には，北から順に -1 から H までの番号が付けられており，列には，西から順に -1 から W までの番号が付けられている．行 r と列 c が交わるマス $(-1 \leq r \leq H, -1 \leq c \leq W)$ をマス (r, c) と書くことにする．

$0 \leq r < H$ かつ $0 \leq c < W$ なるマス (r, c) について考える．マス (r, c) には **隣接する** 4 個のマスが存在する：

- マス $(r, c-1)$ をマス (r, c) の **西** のマスと呼ぶ．
- マス $(r+1, c)$ をマス (r, c) の **南** のマスと呼ぶ．
- マス $(r, c+1)$ をマス (r, c) の **東** のマスと呼ぶ．
- マス $(r-1, c)$ をマス (r, c) の **北** のマスと呼ぶ．

$r = -1, r = H, c = -1, c = W$ のいずれかが満たされているとき，マス (r, c) を **境界** マスと呼ぶ．境界マスではないマスは **障害物** マスか **空白** マスのいずれかである．加えて，それぞれの空白マスには **色** が塗られており，0 以上 Z_{MAX} 以下の非負整数で表される．はじめ，すべての空白マスの色は 0 である．

例えば，$H = 4$ かつ $W = 5$ であり，障害物マスがマス $(1, 3)$ の 1 箇所であるような迷路について考える：



唯一の障害物マスはバツ印で，境界マスは黒色で表されている． 空白マスに書かれている数字は各々のマスの色である．

マス (r_0, c_0) からマス (r_ℓ, c_ℓ) への長さ ℓ $(\ell > 0)$ の **パス** とは， $0 \leq i < \ell$ なる整数 i についてマス (r_i, c_i) とマス (r_{i+1}, c_{i+1}) が隣接している， 相異なる *空白* マス $(r_0, c_0), (r_1, c_1), \ldots, (r_\ell, c_\ell)$ の列のことである．

長さ ℓ のパスにはちょうど $\ell + 1$ 個のマスが含まれることに注意せよ．

コンテストでは，研究者たちはマス $(0, 0)$ からマス $(H - 1, W - 1)$ へのパスが少なくとも 1 つ存在する迷路を用意する． マス $(0, 0)$ とマス $(H - 1, W - 1)$ が空白マスであることが保証されることに注意せよ．

Hanga はどのマスが空白マスであり，どのマスが障害物マスであるかを知らない．

あなたの課題は，研究者たちによってどのように迷路が用意されているかわからないながらも，マス $(0, 0)$ からマス $(H - 1, W - 1)$ までの *最短のパス*（長さが最小であるパス）を見つけることができるように Pulibot をプログラムし，Hanga を助けることである． Pulibot の仕様とコンテストのルールは以下に記述されている．

この問題文の最後のセクションに Pulibot のビジュアライザについての説明があることに注意せよ．

Pulibot の仕様

マス (r, c) $(-1 \leq r \leq H, -1 \leq c \leq W)$ について，マスの **状態** を以下のように 1 つの整数として定義する：

- マス (r, c) が境界マスであるとき，マス (r, c) の状態は -2 である．
- マス (r, c) が障害物マスであるとき，マス (r, c) の状態は -1 である．
- マス (r, c) が空白マスであるとき，マス (r, c) の状態はそのマスの色である．

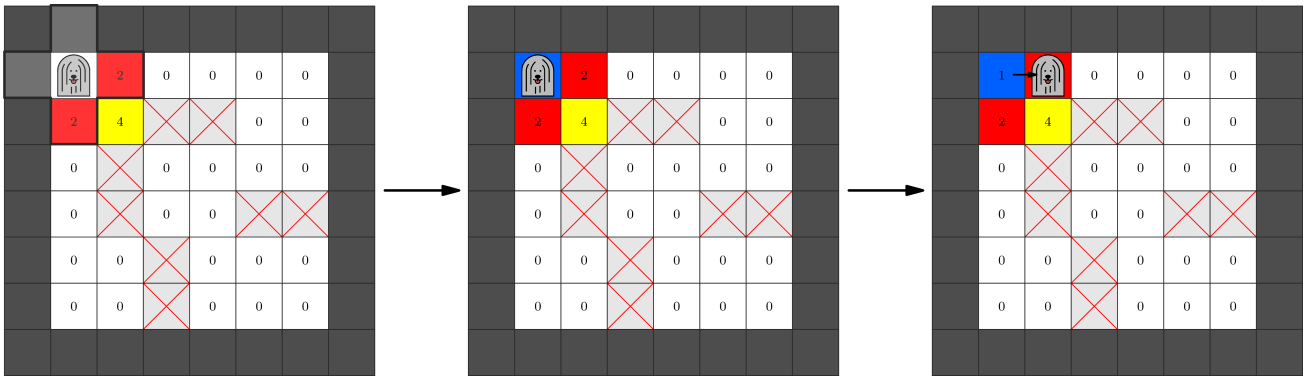
Pulibot のプログラムは連続したステップからなる． それぞれのステップでは，Pulibot は近隣のマスの状態を認識し，その後に命令を実行する． 実行される命令は認識されたマスの状態によって決定される． より詳細な説明は以下に記す．

現在のステップの開始時点で，Pulibot が空白マス (r, c) にいるとする． このステップは以下のように実行される：

1. 最初， Pulibot は現在の **状態の配列** を認識する． 状態の配列とは， 配列 $S = [S[0], S[1], S[2], S[3], S[4]]$ であり，マス (r, c) の状態と隣接するすべてのマスの状態からなる配列である：
    - $S[0]$ はマス (r, c) の状態である．
    - $S[1]$ は西のマスの状態である．
    - $S[2]$ は南のマスの状態である．
    - $S[3]$ は東のマスの状態である．
    - $S[4]$ は北のマスの状態である．
2. 次に，Pulibot は取得した状態の配列に応じて **命令** (Z, A) を決める．

3. 最後に，Pulibot は上で決めた命令を実行する：マス (r, c) の色を Z に変更し，行動 A を実行する．行動 A は以下のいずれかである：
    - マス (r, c) に *留まる*．
    - 4 個の隣接するマスのうちのいずれかに *移動する*．
    - *プログラムを終了する*．

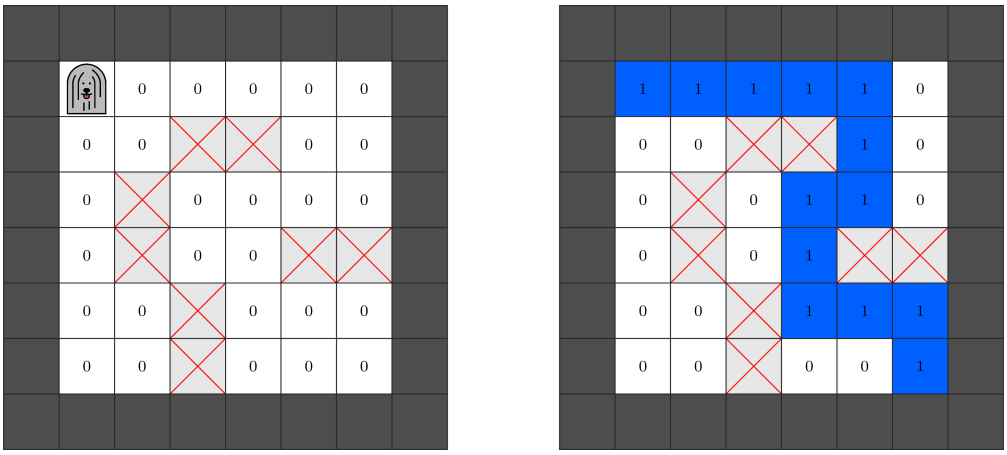
例えば，以下の図の左に示されているシナリオについて考える． Pulibot は現在，色が 0 であるマス $(0, 0)$ にいる． Pulibot は状態の配列 $S = [0, -2, 2, 2, -2]$ を認識する． この配列を取得したとき，図の中央と右で示されているように，現在いるマスの色を $Z = 1$ に変更し，東に移動するように Pulibot をプログラムすることができる．



ロボットコンテストのルール

- 最初，Pulibot はマス $(0, 0)$ にいる．そして，プログラムの実行を始める．
- Pulibot は空白マスでないマスへの移動は許されていない．
- Pulibot のプログラムは $500\,000$ 回以下のステップで終了しなければならない．
- Pulibot のプログラムの終了後，空白マスの色は以下のように塗られていなければならない：
    - 色 1 で塗られたマスからなるマス $(0, 0)$ からマス $(H - 1, W - 1)$ までの最短のパスが 1 つ存在し，
    - パスに含まれない空白マスの色はすべて 0 でなければならない．
- Pulibot はいずれかの空白マス上でプログラムを終了しなければならない．

例えば，以下の図は $H = W = 6$ である迷路を示している． 最初の状態は左図に示されており，プログラムの終了後に認められる空白マスの色の塗り方のひとつが右図に示されている．



実装の詳細

以下の関数を実装せよ．

```
void program_pulibot()
```

- この関数は Pulibot のプログラムを生成しなければならない．Pulibot のプログラムはいかなる H, W についても，そして制約をみたすいかなる迷路についても，正しく動作をしなければならない．
- この関数はそれぞれのテストケースについてちょうど 1 回呼び出される．

この関数は Pulibot のプログラムを作成するために以下の関数を呼び出すことができる．

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

- S：長さ 5 の配列であり，状態の配列を表す．
- Z：非負整数であり，色を表す．
- A：1 文字であり，以下のように Pulibot の行動を表す：
  - `H`：留まる
  - `W`：西に移動する
  - `S`：南に移動する
  - `E`：東に移動する
  - `N`：北に移動する
  - `T`：プログラムを終了する
- この関数の呼び出しにより，状態の配列 S を認識した場合は命令 (Z, A) を実行する，ということを Pulibot に指示できる．

同じ状態の配列 S に対して複数回この関数を呼び出すと，結果は `Output isn't correct` と評価される．

`set_instruction` をすべての存在しうる状態の配列 S について呼び出す必要はない．しかしながら，もし Pulibot が命令が指示されていない状態の配列を認識した際には `Output isn't correct` として評価される．

`program_pulibot` の終了後，採点プログラムは 1 つ以上の迷路について，Pulibot のプログラムを実行する． これらの Pulibot のプログラムの実行は，あなたの解答の実行制限時間として *含まれない*．採点プログラムは適応的 (adaptive) *ではない*．それぞれのテストケースについての迷路は事前に決められている．

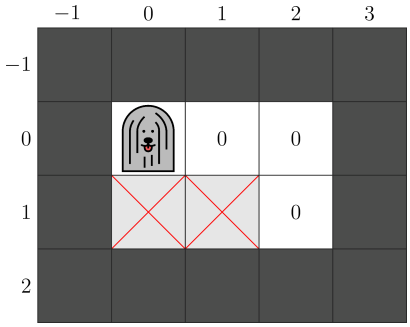
もし Pulibot が Pulibot のプログラムの実行の終了前にいずれかのロボットコンテストのルールを破った際には，`Output isn't correct` として評価される．

例

関数 `program_pulibot` は以下のように関数 `set_instruction` を呼び出すことができる．

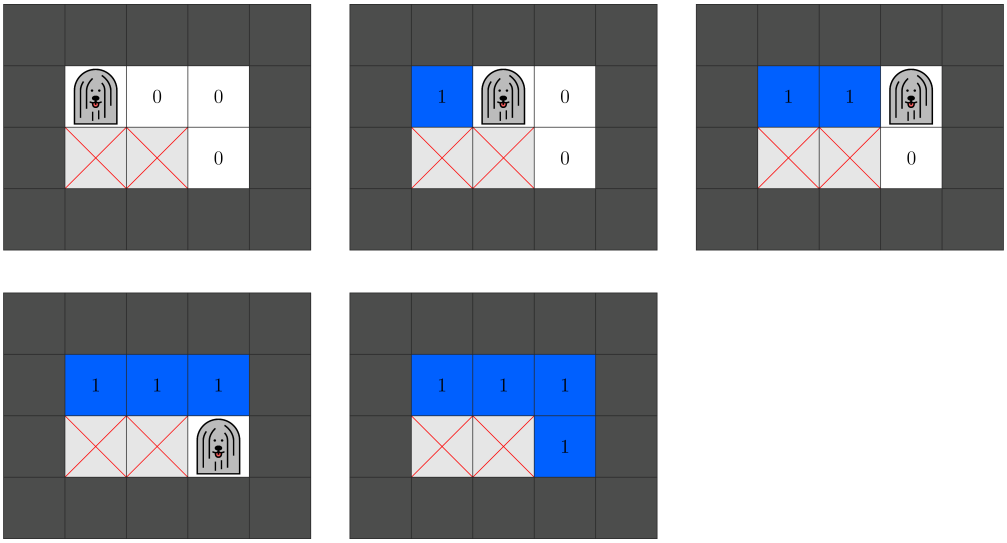
呼び出し	状態の配列 S についての命令
`set_instruction([0, -2, -1, 0, -2], 1, E)`	色を 1 に変更し，東に移動する
`set_instruction([0, 1, -1, 0, -2], 1, E)`	色を 1 に変更し，東に移動する
`set_instruction([0, 1, 0, -2, -2], 1, S)`	色を 1 に変更し，南に移動する
`set_instruction([0, -1, -2, -2, 1], 1, T)`	色を 1 に変更し，プログラムを終了する

$H = 2, W = 3$ であり，以下の図で示される迷路についてのシナリオを考える．



特にこの迷路について，Pulibot のプログラムは 4 回のステップを実行する． Pulibot が認識する状態の配列と，Pulibot が実行する命令は，上の 4 回の `set_instruction` の呼び出しとちょうど順番通り対応する． これらの命令のうち，最後の命令はプログラムを終了するものである．

以下の図は，4 回それぞれのステップの開始時点と，プログラム終了後の状態を表したものである．



しかしながら，このプログラムの 4 つの命令では他の迷路について，最短のパスを見つけることができないかもしれないことに注意せよ．そのため，もしこれを提出しても，`Output isn't correct` として評価される．

制約

$Z_{MAX} = 19$ である．したがって，Pulibot は 0 以上 19 以下の色を使用できる．

Pulibot をテストするそれぞれの迷路について：

- $2 \leq H, W \leq 15$
- マス $(0, 0)$ からマス $(H - 1, W - 1)$ へのパスが少なくとも 1 つ存在する．

小課題

1. (6 点) 迷路内に障害物マスが存在しない．
2. (10 点) $H = 2$．
3. (18 点) いかなる空白マスの組についても，ちょうど 1 つのパスがそれらの間に存在する．
4. (20 点) マス $(0, 0)$ からマス $(H - 1, W - 1)$ への最短のパスの長さは $H + W - 2$ である．
5. (46 点) 追加の制約はない．

もし，いずれかのテストケースについて，`set_instruction` の呼び出しあるいは Pulibot のプログラムの実行が，実装上の詳細に記されている制約に従っていなかった場合，あなたのその小課題の得点は 0 点となる．

それぞれの小課題について，色の塗り方がほとんど正しいときに，あなたは部分点を得られる．

正式には：

- 最終的な色の塗り方がロボットコンテストのルールを満たしているとき，テストケースについての解答が **完全** であるという．
- 最終的な色の塗り方が以下のようになっているとき，テストケースについての解答が **部分的** であるという：
  - 色 1 で塗られたマスからなるマス $(0, 0)$ からマス $(H - 1, W - 1)$ までの最短のパスが 1 つ存在し，
  - パスに含まれない空白マスの色はすべて 1 でない．
  - いくつかの空白マスの色が 0 でも 1 でもない．

もしあなたの解答があるテストケースについて完全でも部分的でもない場合，対応するテストケースについてのあなたの点数は 0 点となる．

小課題 1 から 4 について，解答が完全であるときは 100%，解答が部分的であるときは 50% の点数が得られる．

小課題 5 について，あなたの点数は Pulibot のプログラムが使った色の数に依存する． より正確には，すべての `set_instruction` についての Z の値の最大値を $Z^\star$ とすると，テストケースについての点数は以下の表のように計算される：

条件	点数（完全）	点数（部分的）
$11 \leq Z^\star \leq 19$	$20 + (19 - Z^\star)$	$12 + (19 - Z^\star)$
$Z^\star = 10$	31	23
$Z^\star = 9$	34	26
$Z^\star = 8$	38	29
$Z^\star = 7$	42	32
$Z^\star \leq 6$	46	36

それぞれの小課題についての点数は，それぞれの小課題に含まれるテストケースの得点の最小値である．

採点プログラムのサンプル

採点プログラムのサンプルは以下の形式でプログラムを読み込む：

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

ここで，m は 長さ W の整数の配列を要素に持つ長さ H の配列であり，迷路の境界マスでないマスの情報を表す．$m[r][c] = 0$ のとき，マス (r, c) が空白マスであることを表し，$m[r][c] = 1$ のとき，マス (r, c) が障害物マスであることを表す．

採点プログラムのサンプルは初めに `program_pulibot()` を呼び出す． もし採点プログラムのサンプルがプロコトル違反を検知したならば，採点プログラムのサンプルは `Protocol Violation: <MSG>` と表示し，終了する．`<MSG>` は以下のエラーメッセージのうちの 1 つである．

- `Invalid array`：$-2 \leq S[i] \leq Z_{MAX}$ がいくつかの i について満たされていない，あるいは配列 S の長さが 5 ではない．
- `Invalid color`：$0 \leq Z \leq Z_{MAX}$ が満たされていない．
- `Invalid action`：文字 A が `H`, `W`, `S`, `E`, `N`, `T` のいずれでもない．
- `Same state array`: `set_instruction` が同じ配列 S について少なくとも 2 回呼ばれている．

そうでない場合，`program_pulibot` が終了した後，採点プログラムのサンプルは入力によって表される迷路について，Pulibot のプログラムを実行する．

採点プログラムのサンプルは 2 つの出力を行う．

1 つ目の出力：採点プログラムのサンプルは Pulibot の行動を記録したファイル `robot.bin` をワーキングディレクトリに出力する．このファイルはビジュアライザの入力となる．詳細は次のセクションで記す．

2 つ目の出力：もしPulibot のプログラムの実行に問題があった場合，採点プログラムのサンプルは以下のエラーメッセージのうちの 1 つを出力する．

- `Unexpected state`：Pulibot は `set_instruction` が呼び出されていない状態の配列を認識した．
- `Invalid move`：空白マスでないマスに Pulibot が移動を試みた．
- `Too many steps`：Pulibot がプログラムを終了することなく $500\,000$ 回のステップを実行した．

そうでない場合，Pulibot のプログラムが終了したときのマス (r, c) の状態を $e[r][c]$ としたとき， 採点プログラムのサンプルは 以下の形式で H 行出力する：

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

ビジュアライザ

この問題の添付パッケージに，`display.py` という名前のファイルが含まれている． 実行すると，この Python スクリプトは採点プログラムのサンプルに入力された迷路についての Pulibot の行動を表示する． このとき，バイナリファイル `robot.bin` がワーキングディレクトリに存在していなければならない．

このスクリプトを実行するためには，以下のコマンドを実行すればよい．

```
python3 display.py
```

シンプルなグラフィカルインターフェースが表示される．主な機能は以下のとおりである：

- 迷路全体の状態を見ることができる．現在の Pulibot の位置は長方形で強調される．
- 矢印ボタンをクリックするあるいは矢印キーを押すことでそれぞれのステップについて見ることができる．特定のステップにジャンプすることもできる．
- 次の Pulibot のプログラムのステップは下に表示される． 現在の状態の配列と，Pulibot の次の行動が表示される． 最後のステップの後，エラーメッセージのうち 1 つ，あるいはプログラムが問題なく終了している場合は `Terminated` と表示される．
- 色を表すそれぞれの数字について，背景色，表示されるテキストを割り当てることができる．表示されるテキストは短い文字列であり，同じ色の各マスに書かれる．あなたは背景色と表示されるテキストを以下の方法で割り当てることができる：
  - `Colors` ボタンをクリックして，表示されるウィンドウから設定を行う．
  - `color.txt` の内容を編集する．
- `Reload` ボタンを押すことで `robot.bin` を再読込することができる．これは `robot.bin` の内容が変わったときに便利である．