Pulibot

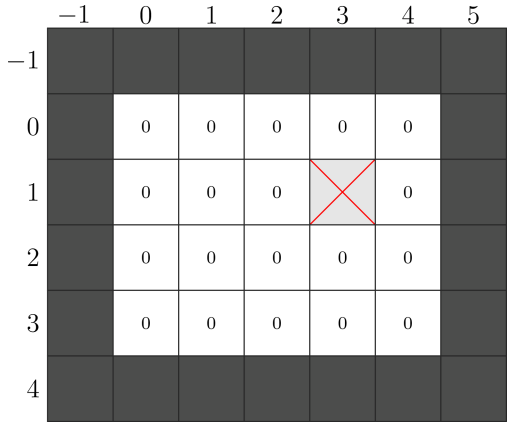
L'Università di Szeged sta organizzando una gara di programmazione di robot! Devi programmare il *Pulibot* definitivo, in grado di trovare percorsi minimi in un labirinto rettangolare.

Il Pulibot dovrà muoversi in un labirinto di $(H+2) \times (W+2)$ celle, con righe numerate da -1 a H da nord a sud e colonne numerate da -1 a W da ovest a est. Indichiamo con (r, c) la cella in riga r e colonna c ($-1 \leq r \leq H$, $-1 \leq c \leq W$). Una cella (r, c) tale che $0 \leq r < H$ e $0 \leq c < W$ ha 4 celle **adiacenti**:

- la cella $(r, c-1)$ è quella a **ovest** di (r, c);
- la cella $(r+1, c)$ è quella a **sud** di (r, c);
- la cella $(r, c+1)$ è quella a **est** di (r, c);
- la cella $(r-1, c)$ è quella a **nord** di (r, c).

Una cella (r, c) è detta di **bordo** se $r = -1$, $r = H$, $c = -1$ o $c = W$. Ogni cella non di bordo può essere o un **ostacolo** o **vuota**. Inoltre, le celle vuote possono essere colorate da Pulibot: il **colore** è rappresentato come un intero tra 0 e Z_{MAX} inclusi. All'inizio le celle vuote hanno tutte colore 0.

Ad esempio, considera il labirinto con $H = 4$ e $W = 5$, con un unico ostacolo nella cella $(1, 3)$:



L'ostacolo è indicato da una croce, le celle di bordo sono annerite, e i numeri nelle celle vuote corrispondono ai loro colori.

Un **percorso** di lunghezza $\ell > 0$ da (r_0, c_0) a (r_ℓ, c_ℓ) è una sequenza di celle *vuote* distinte $(r_0, c_0), (r_1, c_1), \ldots, (r_\ell, c_\ell)$ in cui due celle consecutive sono adiacenti. *Nota che un percorso di lunghezza ℓ contiene $\ell + 1$ celle.*

Al momento della gara, il tuo Pulibot verrà posto nella cella $(0, 0)$ di un labirinto in cui esiste almeno un percorso dalla cella $(0, 0)$ alla cella $(H - 1, W - 1)$ (quindi entrambe queste celle sono necessariamente vuote). *Non ti è noto quali celle siano vuote e quali siano ostacoli.*

Programma Pulibot di modo che possa trovare un *percorso minimo* (di minima lunghezza) dalla cella $(0, 0)$ alla cella $(H - 1, W - 1)$ nel labirinto, seguendo le specifiche di programmazione del Pulibot che seguono.

Nota che l'ultima sezione di questo testo descrive un tool che puoi usare per visualizzare il comportamento di Pulibot.

Specifiche di Pulibot

Diciamo che lo **stato** di una cella (r, c) è:

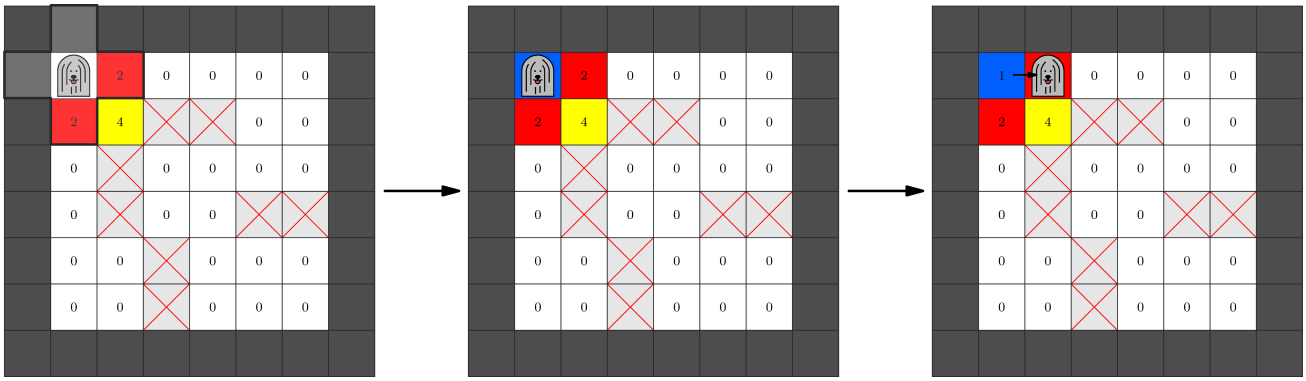
- -2 se la cella è di bordo;
- -1 se la cella è un ostacolo;
- il colore della cella se è vuota.

Il programma di Pulibot viene eseguito come una sequenza di passi. In ogni passo, Pulibot percepisce lo stato delle celle adiacenti, e poi esegue un'istruzione che dipende da questo stato.

Più precisamente, assumendo che Pulibot si trovi nella cella vuota (r, c):

1. Innanzitutto, Pulibot costruisce il **vettore di stato** $S = [S[0], S[1], S[2], S[3], S[4]]$ con lo stato di (r, c) e delle sue celle adiacenti:
    - $S[0]$ è lo stato di (r, c).
    - $S[1]$ è lo stato della cella a ovest (*W*).
    - $S[2]$ è lo stato della cella a sud (*S*).
    - $S[3]$ è lo stato della cella a est (*E*).
    - $S[4]$ è lo stato della cella a nord (*N*).
2. Quindi, Pulibot recupera dalla sua memoria l'**istruzione** (Z, A) che corrisponde a S.
3. Infine, Pulibot esegue l'istruzione: imposta il colore di (r, c) a Z e poi esegue A, che è un'azione tra
    - *H (hold)*: rimanere fermo in (r, c);
    - muoversi in una cella adiacente (*W*, *S*, *E*, *N*);
    - *T*: terminare il programma.

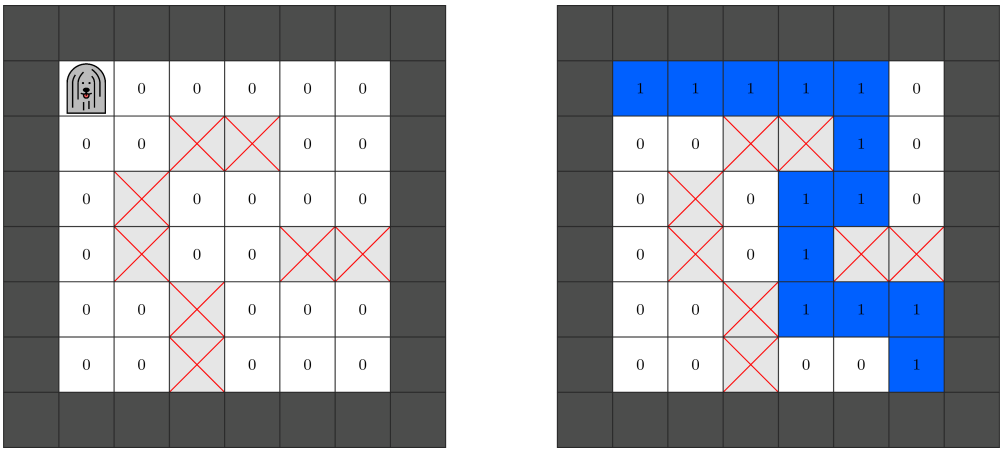
Per esempio, considera lo scenario mostrato a sinistra nella figura seguente. Pulibot è nella cella $(0, 0)$ di colore 0, e costruisce il vettore di stato $S = [0, -2, 2, 2, -2]$. Pulibot potrebbe avere un programma per cui l'istruzione che corrisponde ad S imposta il colore della cella corrente a $Z = 1$ e *successivamente* si sposta a est, come mostrato nelle immagini al centro e a destra della figura.



Regole di gara

- Inizialmente, Pulibot viene posizionato in $(0, 0)$ e inizia a eseguire il suo programma.
- Pulibot non si può muovere in celle non vuote.
- Il programma di Pulibot deve terminare entro al massimo $500\,000$ passi.
- Terminato il programma, le celle vuote devono essere colorate di modo che:
  - Esiste un percorso minimo da $(0, 0)$ a $(H - 1, W - 1)$ per cui il colore di tutte le sue celle è 1.
  - Il colore di tutte le altre celle vuote è 0.
- Pulibot può terminare l'esecuzione in una qualunque cella vuota.

Per esempio, la figura che segue mostra un possibile labirinto con $H = W = 6$. La configurazione di partenza è mostrata a sinistra, mentre una colorazione finale valida è mostrata a destra:



Note di implementazione

Devi implementare la seguente funzione:

```
void program_pulibot()
```

- Questa funzione deve memorizzare un programma per Pulibot, che deve funzionare per tutti i labirinti che rispettano i limiti del problema.
- Questa funzione è chiamata esattamente una volta per ogni testcase.

Questa funzione deve chiamare la seguente funzione per memorizzare il programma di Pulibot:

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

- S: un array di lunghezza 5 con il vettore di stato.
- Z: un colore (intero non-negativo).
- A: un'azione di Pulibot interpretata come segue:
  - `H`: rimanere fermo (hold);
  - `W`: muoversi ad ovest (west);
  - `S`: muoversi a sud (south);
  - `E`: muoversi ad est (east);
  - `N`: muoversi a nord (north);
  - `T`: terminare il programma.
- Chiamando questa funzione fai memorizzare a Pulibot l'istruzione (Z, A) per il vettore di stato S.

Chiamare questa funzione più volte con lo stesso S causerà un risultato di `Output isn't correct`.

Non è richiesto di chiamare `set_instruction` per ogni possibile valore di S, se però durante l'esecuzione Pulibot incontrasse un S per cui non c'è un'istruzione memorizzata, otterrai un risultato di `Output isn't correct`.

Dopo che `program_pulibot` termina, il grader esegue il programma di Pulibot su uno o più labirinti. Il tempo richiesto da queste esecuzioni *non conta* per il time limit della tua soluzione, e il grader *non è adattivo* (l'insieme di labirinti è predefinito in ogni testcase).

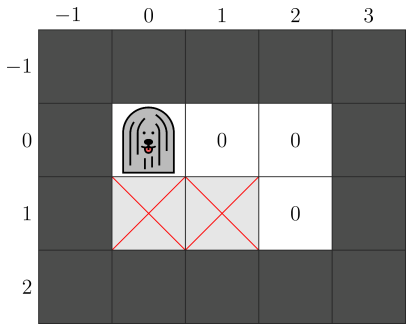
Se Pulibot viola una delle regole di gara prima di terminare il programma, otterrai un risultato di `Output isn't correct`.

Esempio

La funzione `program_pulibot` potrebbe fare chiamate a `set_instruction` come segue:

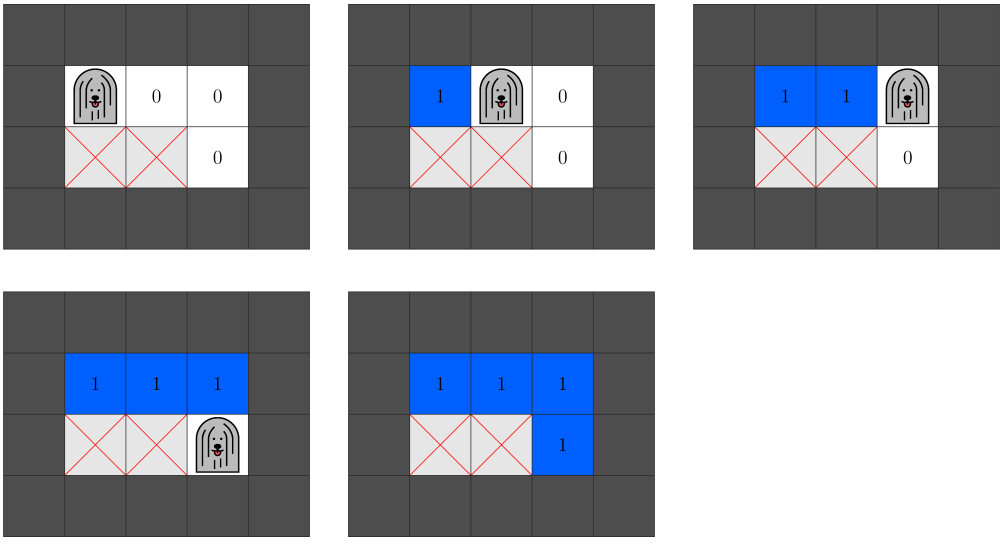
Chiamata	Istruzione per lo stato S
`set_instruction([0, -2, -1, 0, -2], 1, E)`	Imposta il colore a 1 e vai a est
`set_instruction([0, 1, -1, 0, -2], 1, E)`	Imposta il colore a 1 e vai a est
`set_instruction([0, 1, 0, -2, -2], 1, S)`	Imposta il colore a 1 e vai a sud
`set_instruction([0, -1, -2, -2, 1], 1, T)`	Imposta il colore a 1 e termina

Considera uno scenario con $H = 2$ e $W = 3$, dove il labirinto è mostrato in figura:



Per questo labirinto, il programma di Pulibot esegue quattro step. I vettori di stato che Pulibot costruisce e le corrispondenti istruzioni che esegue corrispondono esattamente a quelli nella tabella sopra, nell'ordine dato. L'ultima istruzione termina il programma.

La figura seguente mostra il labirinto prima di ciascuno dei quattro step e i colori finali dopo l'arresto.



Nota però che questo programma di 4 istruzioni potrebbe non trovare un percorso minimo in altri labirinti validi. Pertanto, se venisse sottoposto, riceverebbe il verdetto `Output isn't correct`.

Assunzioni

- $Z_{MAX} = 19$. Quindi, Pulibot può usare i colori da 0 a 19, estremi inclusi.
- $2 \leq H, W \leq 15$ per ogni labirinto usato per testare Pulibot.
- C'è sempre almeno un percorso dalla cella $(0, 0)$ alla cella $(H - 1, W - 1)$.

Subtask

1. (6 punti) Il labirinto non contiene ostacoli,.
2. (10 punti) $H = 2$.
3. (18 punti) C'è esattamente un percorso tra ogni coppia di celle vuote.
4. (20 punti) Ogni percorso minimo da $(0, 0)$ a $(H - 1, W - 1)$ ha lunghezza $H + W - 2$.
5. (46 punti) Nessuna limitazione aggiuntiva.

Se, in un qualunque testcase, la chiamata a `set_instruction` o l'esecuzione del programma non rispetta le assunzioni nel testo, il tuo punteggio per quel subtask è 0. Puoi ottenere punteggi parziali producendo una colorazione che sia quasi corretta. Più precisamente:

- La soluzione di un testcase è **completa** se la colorazione finale soddisfa le regole di gara.
- La soluzione di un testcase è **parziale** se la colorazione finale è come segue:
  - Esiste un percorso minimo da $(0, 0)$ a $(H - 1, W - 1)$ di celle di colore 1.
  - Non ci sono altre celle vuote nella griglia di colore 1.
  - Alcune celle vuote nella griglia hanno colore ≥ 2.

Se la tua soluzione di un testcase non è completa né parziale, il punteggio per quel subtask è 0.

Nei subtask 1-4, ottieni il 100% dei punti per una soluzione completa e il 50% dei punti per una soluzione parziale. Nel subtask 5, il tuo punteggio dipende dal numero di colori usato dal programma di Pulibot. Sia $Z^\star$ il massimo valore di Z tra tutte le chiamate a `set_instruction`, allora il tuo punteggio sarà calcolato come:

Condizione	Punteggio (completo)	Punteggio (parziale)
$11 \leq Z^\star \leq 19$	$20 + (19 - Z^\star)$	$12 + (19 - Z^\star)$
$Z^\star = 10$	31	23
$Z^\star = 9$	34	26
$Z^\star = 8$	38	29
$Z^\star = 7$	42	32
$Z^\star \leq 6$	46	36

Il punteggio di ogni subtask è il minimo numero di punti per un testcase in quel subtask.

Grader di esempio

Il grader di esempio legge l'input nel seguente formato:

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

Dove m è un array di H array di W interi, che descrive le celle non di bordo del labirinto. $m[r][c] = 0$ se la cella (r, c) è vuota, $m[r][c] = 1$ altrimenti.

All'avvio, il grader di esempio chiama `program_pulibot()`. Se rileva una violazione del protocollo, il grader di esempio stampa `Protocol Violation: <MSG>` e termina, dove `<MSG>` è uno dei seguenti messaggi di errore:

- `Invalid array`: $-2 \le S[i] \le Z_{MAX}$ non è rispettato per qualche i, oppure la lunghezza di S non è 5.
- `Invalid color`: $0 \le Z \le Z_{MAX}$ non è rispettato.
- `Invalid action`: il carattere A non è uno tra `H`, `W`, `S`, `E`, `N` e `T`.
- `Same state array`: `set_instruction` è stato chiamato almeno due volte con lo stesso array S.

Altrimenti, quando `program_pulibot` termina, il grader di esempio esegue il programma di Pulibot nel labirinto in input.

Il grader di esempio produce due output.

Prima, salva un log delle azioni effettuate da Pulibot nel file `robot.bin` nella cartella corrente. Questo file può essere usato come input del tool di visualizzazione descritto nella prossima sezione.

Poi, se il programma di Pulibot non termina correttamente, il grader stampa uno dei messaggi di errore:

- `Unexpected state`: Pulibot ha percepito uno stato col quale non era stato chiamato `set_instruction`.
- `Invalid move`: Qualche azione ha fatto spostare Pulibot in una cella non vuota.
- `Too many steps`: Pulibot ha eseguito $500\,000$ istruzioni senza terminare.

Altrimenti, chiamiamo $e[r][c]$ lo stato della cella (r, c) al termine del programma di Pulibot. Il grader di esempio stampa H righe nel seguente formato:

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

Tool di visualizzazione

Tra gli allegati a questo problema è presente un file chiamato `display.py`. Quando viene chiamato, questo script visualizza le azioni di Pulibot nel labirinto, leggendo il contenuto di `robot.bin` (che deve essere presente nella cartella corrente).

Per far partire il tool, esegui:

```
python3 display.py
```

che aprirà una semplice UI. Le feature principali sono:

- Puoi vedere lo stato del labirinto. La posizione corrente di Pulibot è segnata da un rettangolo.
- Puoi vedere le azioni di Pulibot cliccando sui tasti freccia o usando le hotkey corrispondendi. Puoi anche saltare a un'azione specifica.
- La prossima azione che verrà eseguita da Pulibot è mostrata in basso. Vengono mostrati l'array dello stato corrente e la mossa che verrà eseguita. Dopo la mossa finale, viene mostrato un errore del grader, o `Terminated` se il programma termina correttamente.
- A ogni numero che rappresenta un colore puoi assegnare un colore di background e un testo. Il testo è una breve stringa che verrà mostrata in ogni cella di quel colore. Puoi assegnare colori e testo in due modi:
  - dal dialogo che compare cliccando su `Colors`;
  - modificando il contenuto di `colors.txt`.
- Per ricaricare `robot.bin`, usa il tasto `Reload`.