Odometro a ciottoli

Leonardo ha inventato l'*odometro* originale: un carrello che misurava le distanze facendo cadere dei ciottoli ad ogni giro delle ruote del carrello. Il numero dei ciottoli dava il numero di giri delle ruote, che a sua volta permetteva l'utente di calcolare la distanza percorsa dall'odometro. Da bravi informatici, abbiamo aggiunto un controllo software all'odometro, estendendo le sue funzionalità. Il tuo compito è programmare l'odometro seguendo le regole specificate sotto.

Griglia di funzionamento

L'odometro si muove su una griglia quadrata immaginaria di 256 × 256 celle unitarie. Ogni cella può contenere al più 15 ciottoli ed è identificata da una coppia di coordinate (riga, colonna), dove ogni coordinata è compresa nell'intervallo 0, …, 255. Data una cella (i, j), le celle adiacenti ad essa sono (se esistono) (i - 1, j), (i + 1, j), (i, j - 1), e (i, j + 1). Ogni cella giacente sulla prima o ultima riga, o sulla prima o ultima colonna, è chiamata un *bordo*. L'odometro parte sempre alla cella (0, 0) (l'angolo nord-ovest), rivolto a nord.

Comandi di base

L'odometro può essere programmato usando i seguenti comandi.

- `left` — gira di 90 gradi verso sinistra (in senso antiorario) e resta nella cella corrente (ad esempio se era rivolto a sud prima, sarà rivolto a est dopo l'esecuzione del comando).
- `left` — gira di 90 gradi verso destra (in senso orario) e resta nella cella corrente (ad esempio se era rivolto a ovest prima, sarà rivolto a nord dopo l'esecuzione del comando).
- `move` — spostati in avanti di una unità (nella direzione verso cui è rivolto l'odometro) in una cella adiacente. Se non esiste tale cella (cioè il bordo in quella direzione è già stato raggiunto) questo comando non ha effetto.
- `get` — rimuovi un ciottolo dalla cella corrente. Se la cella non contiene ciottoli, questo comando non ha effetto.
- `put` — aggiungi un ciottolo alla cella corrente. Se la cella corrente contiene già 15 ciottoli, questo comando non ha effetto. L'odometro ha una riserva illimitata di ciottoli.
- `halt` — termina l'esecuzione.

L'odometro esegue i comandi nell'ordine in cui sono dati nel programma. Il programma deve contenere al più un comando per linea. Le linee vuote sono ignorate. Il simbolo `#` indica un commento; il testo a seguire, fino alla fine della linea, viene ignorato. Se l'odometro raggiunge la fine del programma, l'esecuzione è terminata.

Esempio 1

Considera il seguente programma per l'odometro. Esso porta l'odometro alla cella (0, 2), rivolto verso est. (Nota che la prima `move` viene ignorata, perché l'odometro è sull'angolo nord-ovest rivolto a nord.)

```
move  # nessun effetto
right
# adesso l'odometro è rivolto a est
move
move
```

Etichette, bordi e ciottoli

Per alterare il flusso del programma a seconda dello stato corrente, puoi usare delle etichette, che sono stringhe case-sensitive di al più 128 simboli scelti tra `a`, …, `z`, `A`, …, `Z`, `0`, …, `9`. I nuovi comandi riguardanti le etichette sono elencati sotto. Nelle descrizioni sotto, *`L`* denota una qualunque etichetta valida.

- *`L`*`:` (cioè *`L`* seguito da due punti '`:`') — dichiara la locazione all'interno di un programma di una etichetta *`L`*. Tutte le etichette dichiarate devono essere distinte. La dichiarazione di una etichetta non ha effetto sull'odometro.
- `jump` *`L`* — continua l'esecuzione incondizionatamente saltando alla linea con etichetta *`L`*.
- `border` *`L`* — continua l'esecuzione saltando alla linea con etichetta *`L`*, se l'odometro è su un bordo rivolto all'esterno della griglia (cioè una istruzione `move` non avrebbe effetto); altrimenti, l'esecuzione continua normalmente e questo comando non ha effetto.
- `pebble` *`L`* — continua l'esecuzione saltando alla linea con etichetta *`L`*, se la cella corrente contiene almeno un ciottolo; altrimenti, l'esecuzione continua normalmente e il comando non ha effetto.

Esempio 2

Il programma seguente localizza il primo (più a ovest) ciottolo nella riga 0 e termina lì; se non ci sono ciottoli nella riga 0, termina sul bordo alla fine della riga. Usa due etichette `leonardo` e `davinci`.

```
right
leonardo:
pebble davinci  # ciottolo trovato
border davinci  # fine della riga
move
jump leonardo
davinci:
halt
```

L'odometro inizia girando alla propria destra. Il ciclo comincia con la dichiarazione di etichetta `leonardo:` e finisce con il comando `jump leonardo`. Nel ciclo, l'odometro controlla la presenza di un ciottolo o del bordo alla fine della riga; in caso contrario, l'odometro esegue una `move` dalla cella corrente (0, j) alla cella adiacente (0, j + 1) dal momento che la destinazione esiste. (Il comando `halt` non è strettamente necessario qui visto che il programma termina in ogni caso.)

Descrizione del problema

Devi sottoporre un programma nel linguaggio dell'odometro, come descritto sopra, che faccia comportare l'odometro come richiesto. Ogni subtask (vedi sotto) specifica un comportamento che l'odometro deve osservare e i vincoli che la soluzione deve soddisfare. I vincoli riguardano le due seguenti questioni.

- *Dimensione del programma* — il programma deve essere abbastanza breve. La dimensione del programma è il numero di comandi presenti in esso. Dichiarazioni di etichette, commenti e linee vuote *non sono contate* nella dimensione.
- *Lunghezza di esecuzione* — il programma deve terminare abbastanza velocemente. La lunghezza di esecuzione è il numero di *passi* eseguiti: ogni singola esecuzione di un comando conta come passo, sia che il comando abbia un effetto o no; dichiarazioni di etichette, commenti e linee vuote non contano come passi.

Nell'Esempio 1, la dimensione del programma è 4 e la lunghezza di esecuzione è 4. Nell'Esempio 2, la lunghezza del programma è 6 e, se eseguito su una griglia con un singolo ciottolo nella cella (0, 10), la lunghezza di esecuzione è 43 passi: `right`, 10 iterazioni del ciclo, ogni iterazione esegue 4 passi (`pebble davinci`; `border davinci`; `move`; `jump leonardo`), e alla fine, `pebble davinci` e `halt`.

Subtask 1 [9 punti]

All'inizio ci sono x ciottoli nella cella (0, 0) e y nella cella (0, 1), invece tutte le altre celle sono vuote. Scrivi un programma che termina con l'odometro nella cella (0, 0) se $x \leq y$, e nella cella (0, 1) altrimenti. (Non è importante la direzione in cui l'odometro è rivolto alla fine; allo stesso modo non è importante quanti ciottoli sono presenti alla fine nella griglia, o dove sono collocati.)

Limiti: dimensione del programma ≤ 100, lunghezza di esecuzione ≤ 1 000.

Subtask 2 [12 punti]

Stesso task come sopra ma quando il programma termina, la cella (0, 0) deve contenere esattamente x ciottoli e la cella (0, 1) deve contenere esattamente y ciottoli.

Limiti: dimensione del programma ≤ 200, lunghezza di esecuzione ≤ 2 000.

Subtask 3 [19 punti]

Ci sono esattamente due ciottoli da qualche parte nella riga 0: uno è nella cella (0, x), l'altro nella cella (0, y); x e y sono distingi, e x + y è pari. Scrivi un programma che termina con l'odometro nella cella (0, (x + y) / 2), cioè, esattamente nel punto medio tra le due celle che contengono i ciottoli. Lo stato finale della griglia non è rilevante.

Limiti: dimensione del programma ≤ 100, lunghezza di esecuzione ≤ 200 000.

Subtask 4 [fino a 32 punti]

Nella griglia ci sono al più 15 ciottoli, ognuno in una cella diversa. Scrivi un programma che li raccoglie tutti nell'angolo a nord-ovest; più precisamente, se c'erano x ciottoli nella griglia all'inizio, alla fine ci devono essere esattamente x ciottoli nella cella (0, 0) e nessun ciottolo altrove.

Il punteggio per questo subtask dipende dalla lunghezza di esecuzione del programma sottoposto. Più precisamente, se L è il massimo delle lunghezze di esecuzione sui vari test case, il tuo punteggio sarà:

- 32 punti se $L \leq 200\,000$;
- $32 - 32 \log_{10} (L / 200\,000)$ punti se $200\,000 < L < 2\,000\,000$;
- 0 punti se $L \geq 2\,000\,000$.

Limiti: dimensione del programma ≤ 200.

Subtask 5 [fino a 28 punti]

Ci può essere un numero qualunque di ciottoli in ogni cella della griglia (ovviamente, compreso tra 0 e 15). Scrivi un programma che trova il minimo, cioè, che termina con l'odometro in una cella (i, j) tale che ogni altra cella contiene almeno il numero di ciottoli in (i, j). Alla fine dell'esecuzione del programma, il numero di ciottoli in ogni cella deve essere lo stesso che prima di eseguire il programma.

Il punteggio per questo subtask dipende dalla dimensione P del programma sottoposto. Più precisamente, il tuo punteggio sarà:

- 28 punti se $P \leq 444$.
- $28 - 28 \log_{10} (P / 444)$ punti se $444 < P < 4\,440$;
- 0 punti se $P \geq 4\,440$.

Limiti: lunghezza di esecuzione ≤ 44 400 000.

Dettagli implementativi

Devi sottoporre esattamente un file per ogni subtask, scritto rispettando le regole di sintassi specificate sopra. Ogni file sottoposto può avere una dimensione massima di 5 MiB. Per ogni subtask, il tuo codice per l'odometro verrà testato su alcuni test cases, e riceverai del feedback sulle risorse usate dal tuo programma. Nel caso in cui il tuo codice non sia sintatticamente corretto e quindi impossibile da testare, riceverai informazioni sullo specifico errore di sintassi.

Non è necessario che la tua sottoposizione contenga programmi per odometro per tutti i subtask. Se la tua sottoposizione corrente non contiene il programma per odometro per il subtask X, la tua più recente sottoposizione per il subtask X è automaticamente inclusa; se non c'è un tale programma, il subtask avrà punteggio zero per quella sottoposizione.

Come di consueto, il punteggio di una sottoposizione è la somma dei punteggi ottenuti in ogni subtask, e il punteggio finale del task è il punteggio massimo tra le sottoposizioni release-tested e l'ultima sottoposizione.

Simulatore

Per scopi di test, ti è fornito un simulatore di odometro, con cui puoi provare i tuoi programmi e le tue griglie di input. I programmi per l'odometro saranno scritti nello stesso formato utilizzato per le sottoposizioni (ad esempio, quello sopra descritto).

Le descrizioni delle griglie saranno date secondo il seguente formato: ogni linea del file deve contenere tre numeri, R, C e P, ad indicare che la cella della griglia alla riga R e colonna C contiene P ciottoli. Si assume che tutte le celle non specificate nella descrizione della griglia non contengano ciottoli. Ad esempio, si consideri il file:

```
0 10 3
4 5 12
```

La griglia descritta da questo file conterrebbe 15 ciottoli: 3 nella cella (0, 10) e 12 nella cella (4, 5).

Puoi eseguire il simulatore di test chiamando il programma `simulator.py` nella tua cartella del task, passando come argomento il nome del file del programma. Il simulatore accetta le seguenti opzioni da riga di comando:

- `-h` fornisce una breve panoramica delle opzioni disponibili;
- `-g GRID_FILE` carica la descrizione della griglia dal file `GRID_FILE` (default: griglia vuota);
- `-s GRID_SIDE` imposta la dimensione della griglia a `GRID_SIDE x GRID_SIDE` (default: 256, come nella specifica del problema); l'impiego di griglie più piccole può essere utile per il debug dei programmi;
- `-m STEPS` limita il numero di passi di esecuzione nella simulazione ad al più `STEPS`;

- `-c` usa la modalità compilazione; in modalità compilazione, il simulatore restituisce esattamente lo stesso output, ma invece di eseguire la simulazione in Python, genera e compila un piccolo programma C. Questo causa un maggiore overhead all'avvio, ma fornisce risultati considerevolmente più veloci; ti è consigliato di usare questa modalità quando ti aspetti che il tuo programma esegua più di 10 000 000 di passi.

Numero di sottoposizioni

Il massimo numero di sottoposizioni permesse per questo task è 128.