掉石里程計

李安納度發明了第一部里程計：是利用輪子轉動掉下小石頭來測量距離的一輛車子。只要數算小石頭的數量就可得知輪子的轉動次數，從而得知走過的距離。身為電腦科學家，我們在里程計加入軟件的控制，以擴充它的功能。你的任務就是依下列的規則設計里程計的程式。

操作網格範圍

里程計在一假想以 256 x 256 個單位格子組成的方形網格上移動。每個格子最多可容納 15 粒小石頭，且以〔 横列, 直行 〕 座標標示，每座標值範圍於 0, ..., 255。格子 (i, j) 相鄰的格子〔 如存在的話 〕為 (i - 1, j), (i + 1, j), (i, j - 1) 和 (i, j + 1)。任何位於第一横列或最後横列，第一直行或最後直行的格子，稱之為"邊界"。里程計必定從格子 (0, 0) 〔 西北角 〕開動，面向北方。

基本指令

里程計程式可使用下列指令。

- `left` — 向左轉 90 度〔 逆時針方向 〕 並留在目前格子〔 例如：如本來向南，指令完成後將向東 〕。
- `right` — 向右轉 90 度〔 順時針方向 〕 並留在目前格子〔 例如：如本來向西，指令完成後將向北 〕。
- `move` — 向前移動一格〔 往里程計的面向 〕到相鄰的格子。如果此格子不存在〔 也就是已在該方向的邊界 〕，則此指令沒有作用。
- `get` — 從目前格子移走一粒小石頭。如果目前格子沒有小石頭，則此指令沒有作用。
- `put` — 在目前格子增加一粒小石頭。如果目前格子已有 15 粒小石頭，則此指令沒有作用。里程計的小石頭永不會用完。
- `halt` — 終止執行。

里程計依程式中的順序執行指令，程式中必須每一行只有一個指令，空白行可忽略，`#` 表示註解；任何在後面的文字，直到該行結尾，都可忽略。如果里程計到達程式結尾，則終止執行。

範例 1

考慮以下里程計程式。它把里程計帶到 (0, 2)，面向東。〔注意第一個 `move` 被忽略，因為里程計位於西北角，面向北方〕。

```
move  # 沒有作用
right
# 現在里程計朝向東
move
move
```

標籤，邊界和小石頭

如要根據現時的狀況改變程式的流程的話，你可以使用標籤，標籤是最多包含 128 個由 `a`，…，`z`，`A`，…，`Z`，`0`，…，`9`中選取的符號，而區分大小寫的字符串。以下是關於標籤的新指令。在以下的描述中，*L* 代表任何有效的標籤。

- *L*`:`〔即 *L* 後為冒號 '`:`'〕 — 宣告程式中標籤 *L* 的位置。所有已宣告的標籤必要是唯一的。宣告標籤對於里程計並無作用。
- `jump` *L* — 無條件跳到標籤 *L* 的那一行繼續執行。
- `border` *L* — 如果里程計位於邊界，且面向網格邊緣〔即 `move` 指令將沒有作用〕，則跳到標籤 `L` 的那一行繼續執行；否則，程式繼續正常執行，而此指令沒有作用。
- `pebble` *L* — 如果目前的格子含有至少一粒小石頭，則跳到標籤 `L` 的那一行繼續執行；否則，程式繼續正常執行，而此指令沒有作用。

範例 2

下列程式找到橫列 0 的第一顆〔最西方的〕小石頭，並停在該處；如果橫列 0 沒有小石頭，它則停在該橫列末尾的邊界上。它使用了 `leonardo` 和 `davinci` 兩個標籤。

```
right
leonardo:
pebble davinci  # 找到小石頭
border davinci  # 橫列末尾
move
jump leonardo
davinci:
halt
```

里程計開始時先向右轉。迴圈由宣告 `leonardo:` 標籤處開始，在 `jump leonardo` 指令處結束。在迴圈中，里程計檢查小石頭是否存在，或者是否在橫列末尾的邊界；若不是，里程計就從格子 (0, j) 作出 `move` 移動到相鄰的格子 (0, j + 1)，因為後者確實存在。〔`halt` 指令在此不一定需要，因為程式無論如何都會終止。〕

陳述

你必須遞交如上面描述的里程計語言的程式，使里程計按照預期的行動。每個子任務〔如下〕規定了一項里程計必須達成的行動，以及所遞交的解答所必須滿足的限制。限制和以下事項有關：

- *程式大小* — 程式要足夠短。程式大小是裏面指令的數目。標籤宣告、註解、和空白行 *不算在* 程式大小裏。
- *執行長度* — 程式必須終止得足夠快。執行長度是執行步驟 *的數目*：每個單一指令的執行算做一個步驟，不論指令是否有作用；標籤宣告、註解、和空白行不算做步驟。

在範例 1，程式大小為 4，執行長度為 4。在範例 2，程式大小為 6，且在只有一粒小石頭在格子 (0, 10) 的網格執行時，執行長度為43：`right`，迴圈經過 10 次，每次 4 個步驟〔`pebble davinci`；`border davinci`；`move`；`jump leonardo`〕，和最後的 `pebble davinci` 與 `halt`。

子任務 1 [9 分]

一開始有 x 粒小石頭在格子 (0, 0) 和 y 粒小石頭在格子 (0, 1) 而其他的格子都是空的。記住任何一個格子裡面最多只能有 15 粒小石頭。寫一個程式使如果 $x \le y$，里程計會終止於 (0, 0)，但如是其他的狀況，則終止於 (0, 1)。〔我們不關心最後里程計向著哪一個方向；我們也不關心最後格子裡面有多小粒小石頭或者是這些小石頭落在何處〕。

限制： 程式大小 ≤ 100，執行長度 $\le 1\,000$。

子任務 2 [12 分]

與前一個任務相同，但當程式結束執行，格子 (0,0) 必須包含 x 粒小石頭而且 (0,1) 必須包含 y 粒小石頭。

限制： 程式大小 ≤ 200，執行長度 $\le 2\,000$。

子任務 3 [19 分]

現只有兩顆小石頭在橫列 0 的某處：一粒在格子 (0, x)，另一粒在格子 (0, y)；x 和 y 不相同，並且 x + y 是雙數。寫一個程式留下里程計於格子 $(0, (x + y) / 2)$，即是在兩個含有小石頭的格子的中點。格子最後的狀況是無關。

限制： 程式大小 ≤ 100，執行長度 $\le 200\,000$。

子任務 4 [32 分]

在網格中最多只能有 15 粒小石頭，而且沒有兩粒小石頭落在同一個格子內。寫一個程式將小石頭全部收集在西北角；更精確一點來說，如果一開始網格中有 x 顆小石頭，在結束的時候，格子 (0, 0) 必須要有 x 粒小石頭，而其他格子都沒有小石頭。

這個子任務的得分將依所遞交程式的執行長度而定。更精確一點來說，如果 L 是在各個測試中的的最大執行長度，你的得分將是：

- 32 分 如 $L \leq 200\,000$；
- $32 - 32 \log_{10}(L / 200\,000)$ 分 如 $200\,000 < L < 2\,000\,000$；
- 如果 $L \geq 2\,000\,000$ 的話則 0 分。

限制： 程式大小 ≤ 200。

子任務 5 [28 分]

每一格可以有任意數目的小石頭〔當然仍在 0 與 15 之間〕。寫個找出最小值的程式，也就是在格子 (i, j) 上終止，使得其它的格子都有至少和格子 (i, j) 一樣多粒小石頭。程式完成執行後，每個格子裡的小石頭數目必須和執行前相同。

這個子任務的得分將依所遞交的程式大小而定。更精確一點來說，你的得分將是：

- 如果 $P \leq 444$ 的話則 28 分；
- 如果 $444 < P < 4\,440$ 的話則 $28 - 28 \log_{10}(P / 444)$ 分；
- 如果 $P \geq 444$ 的話則 0 分；

限制： 執行長度 ≤ 44 400 000。

實現細節

你必須根據上述語法規則，對每個子任務撰寫並遞交剛好一個檔案。每個遞交的檔案最大為 5 MiB。對於每個子任務，你的里程計程式碼將被用於多筆測試資料測試，而你會收到你的程式碼用到哪些資源的回饋。如果程式碼因語法不正確而無法測試時，你將收到特定的語法錯誤的資訊。

你所遞交的里程計程式可以不必包含全部的子任務。如果你這次的遞交未包含子任務 X 的里程計程式，則你最近一次提遞的子任務 X 會自動加入；如果沒有這個程式，則此子任務得分為零。

如常，每次遞交的分數是每項子任務的得分總和，本題最後總分是已發布的遞交所得的分數和最後一次遞交的分數的中的最大值。

模擬器

作為測試用途，給你提供了一個里程計的模擬器，你可用來回饋程式以及輸入網格。里程計程式將會使用和遞交相同的格式來進行編寫（即是如上所描述的）。

網格的描述是使用以下的格式：檔案中的每行必須包含三個數字，R，C 和 P，以代表在橫列 R 及 直行 C 的格子含有 P 粒小石頭。所有未在網格描述中指明的格子可假設沒有包含小石頭。例如這個檔案：

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

這檔案描述的網格包含 15 顆小石頭：3 粒在格子 (0,10) 和 12 粒在格子 (4,5)。

在你的任務目錄中你可呼叫 `simulator.py` 及傳遞程式檔案名稱為參數來引用測試模擬器。模擬器可接受以下的指令行選項：

- `-h` 會為可用的選項提供一個簡要的概述；
- `-g GRID_FILE` 會從檔案 `GRID_FILE` 讀取網格的描述`-h` （預設：empty grid）
- `-s GRID_SIDE` 會設定網格的大小為 GRID_SIDE x GRID_SIDE（預設：256，一如問題說明中所用的），使用小一點的網格可有助程式除錯。
- `-m STEPS` 限制模擬執行步驟數量最多為 `STEPS`;
- `-c` 進入編譯模式；在編譯模式中，模擬器會回傳一樣的輸出結果，但不會利用 Python 進行模擬，反而產生並編譯成一個小的 C 程式。這會在程式啟動時會產生更大的開支，但期後會明顯的較快速給予結果；如果你的程式執行超過 10 000 000 個步驟時，建議你使用這模式。

遞交次數

本任務的最大允許遞交次數為 128 。