卵石里程表

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，一個指令一行，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* — 無條件跳轉到標號 *L* 所在行繼續執行.
- `border` *L* — 如果在邊界上的里程表面向網格上的邊緣（即：`move`命令無效）則轉到標號 *L*所在行繼續執行；否則，執行正常繼續，並且此命令無效。
- `pebble` *L* — 如果當前單元至少包含一塊卵石，則轉到標號 *L*所在行繼續執行；否則，執行正常繼續，並且此命令無效。

樣例 2

下面的程式把第一塊卵石放在第0行並停在那裡；如果第0行沒有卵石它就停在該行的末尾邊界上。它使用兩個標號`leonardo`和`davinci`。

```
right
leonardo:
pebble davinci  # 發現了卵石
border davinci  # 行尾
move
jump leonardo
davinci:
halt
```

里程表從向右轉開始。迴圈從標號聲明leonardo:開始到jump leonardo命令結束。在迴圈中里程表檢查卵石或者在行末檢查邊界的存在；如果不存在，里程表執行從當前單元（0，j）move 到單元（0，j+1）因為後者存在。（命令halt在這裡並不是必須的，因為程式無論如何都會停止）。

Statement

你需要使用上面所描述的里程表的語言提交程式，以便使里程表如期望的那樣行動。每一個子任務（見下文）描述里程表需要完成的一個動作，以及所提交的解必須滿足的限制。限制涉及如下兩方面。

- *Program size* — 程式必須足夠短。程式的大小是它所包含的命令的數量。標號聲明，注釋和空行不記入程式的大小。
- *Execution length* — 程式必須結束得足夠快. 執行的長度是所執行的步數: 每個單條命令計為一步，而不管該命令是否有效; 注釋和空行不計步數。

在例1中，程式的大小是4，執行長度也是4。在例2中，程式的大小是6, 當在網格中執行且單元(0, 10)中只有一塊卵石時, 執行長度為43步: 右轉，迴圈10次，每次4步(`pebble davinci`;`border davinci`;`move`;`jump leonardo`), 最後是`pebble davinci` 和 `halt`。

Subtask 1 [9 points]

在開始時，在單元（0，0）中有X塊卵石，單元（0，1）中有Y塊卵石，其他單元都為空。記住，任意一個單元中最多只能有15塊卵石。寫一個程式，使得里程表在 $X \le Y$時停在單元（0，0）中，否則停在單元（0，1）中。（我們不在意里程表最後的朝向；我們也不在意最後網格中有多少塊卵石，以及它們在什麼位置）

限制：程式大小 ≤ 100,執行長度 ≤ 1000。

Subtask 2 [12 points]

與上面的任務相同，但是當程式結束時，單元（0，0）必須剛好包含X塊卵石，單元（0，1）中剛好有Y塊卵石。

限制：程式大小 ≤ 200,執行長度 ≤ 2000。

Subtask 3 [19 points]

剛好有兩塊卵石在第0行中：一塊在單元（0，x）中，另一塊在單元（0，y）中；x, y不相同並且x+y為偶數。寫一個程式，把里程表留在單元(0, (x + y) / 2)中，即：嚴格地位於包含卵石的兩個單元的中點。網格的最後狀態無關緊要。

限制：程式大小 ≤ 100,執行長度 ≤ 200000。

子任務 4 [32 分]

網格中最多有15塊卵石，沒有任何兩塊在同一個單元中。寫一個程式，把它們都收集到西北角，更準確地說，如果開始時網格中有x塊卵石，則結束時單元（0，0）中必須有x塊卵石，並且其他地方沒有卵石。

這個子任務的得分取決於程式的執行長度。如果L是在各種測試樣例下的最大執行長度，你的得分是:

- 32 points if L ≤ 200 000;

- 32 - 32 $\log_{10}$ (L / 200 000) points if 200 000 < L < 2 000 000;
- 0 points if L ≥ 2 000 000.

Limits: program size ≤ 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)終止，使得其它的格子都有至少和格子(i, j)一樣多顆小石頭。程式執行後，每個格子裡的小石頭數目必須和執行前相同。

本子任務的得分取決於程式大小P，更準確的說：

- 28 points if P ≤ 444;
- 28 - 28 $\log_{10}$ (P / 444) points if 444 < P < 4 440;
- 0 points if P ≥ 4 440.

限制: 執行長度 ≤ 44 400 000.

實現的細節

每個子任務必須繳交剛好一個檔案，根據上述語法規則撰寫。每個繳交的檔案最大5 MiB。於每個子任務，你的里程計程式碼將會用多筆測試資料測試， 而你會收到你的程式碼用到哪些資源的回饋。如果程式碼因語法錯誤而無法執行時，你將收到特定語法錯誤的資訊。

你所繳交的里程計程式可以不必包含全部的子任務。如果你這次的繳交未包含子任務 X 的里程計程式，則你最近一次的子任務 X 會自動列入; 如果沒有這個程式，則此子任務得分為零。

一如往常，每次繳交的分數是每項子任務得分的總和。本題最後總分是已知繳交的分數和最後一次分數的最大值。

模擬器

為了測試的目的，給你提供了一個里程計的模擬器，讓你可以用來餵你的程式以及輸入網格。里程計程式將使用上傳相同的格式進行撰寫(也就是如上所描述的).

網格的描述是使用下面的格式:檔案中的每一行必須包含三個數，R,C,以及P，代表在第R列C行的格子含有P個小石頭。沒有被描述到的格子代表該格子沒有包含任何小石頭。例如，請參考下面的檔案:

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

這個檔案描述的網格包含15顆小石頭: 3個在格子 (0,10) 另外，12顆在格子(4,5)。

在你的工作你可以呼叫測試模擬器藉由呼叫simulator.py 在你的工作目錄將程式檔案名稱當成參數，這個模擬器可以接受下列的指令行選項:

- `-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 。