卵石里程表

李奥纳多发明了初始的"里程表"：一辆能够在车轮转动时通过抛洒卵石测量距离的马车。计数卵石的数量可以知道车轮的转数，就可以计算出车轮行走的距离。作为计算机科学家，我们已经把软件控制加到了里程表里，扩展了它的功能。你的任务就是根据下述规则对里程表编程。

操作网格

里程表在一个想象的 256 × 256 个单元的网格上移动。每个单元最多包含 15 块卵石，并且由一对坐标(行、列) 标记，其中每一个坐标的取值范围都是 0 到 255。给定一个单元(i,j)，其邻接单元为(i - 1, j), (i + 1, j), (i, j - 1) 和 (i, j + 1)，(如果它们存在的话) 。任何位于第一行或最后一行，以及第一列或最后一列的单元被称为边界。里程表总是从 (0, 0) 单元(西北角)开始，并面向北方。

基本命令

里程表可以用下面的命令编程。

left — 左转 90 度(逆时针) 并保留在当前单元中(例如，如果它此前朝南，则执行此命令后就朝东)。

right — 右转 90 度(顺时针) 并保留在当前单元中(例如，如果它此前朝西，则执行此命令后就朝北)。

move — 向前移动一个单元(沿里程表所面向的方向)。如果该单元不存在 (即已经到达该方向的边界)该命令无效。

get — 从当前单元中去掉一块卵石。如果当前单元中没有卵石，该命令无效。

put —在当前单元中增加一块卵石。如果当前单元中已经有了 15 块卵石，则此命令无效。里程表有用不完的卵石。

halt — 结束执行。

里程表按程序中给定命令的顺序执行，每个命令一行。空行被忽略。符号#表示其后直至行尾的内容是注释并被忽略。如果里程表到达了程序的结尾，程序终止。

例 1

考虑下面的里程表程序，它使里程表移到（0，2）单元，面向东。（注意：第一个命令 move 被忽略，因为里程表位于西北角并且面向东）。

```
move  # 无效
right
# 现在里程表朝东
move
move
```

标号，边界和卵石

为根据当前状态改变程序流，你可以使用标号，标号是最多包含 128 个取自 a-z，A-Z，0-9 的符号。包含标号的新命令在下面列出，“L”,表示任何合法的标号。

"L"：（即"L"后紧跟：冒号）-在程序中声明标号位置“L”。所有被声明的标号必须唯一。声明标号对里程表没有影响。

jump "L"-无条件跳转到标号所在行继续执行。

border “L”- 如果在边界上的里程表面向网格上的边缘（即：move 命令无效）则转到标号“L”所在行继续执行；否则，执行正常继续，并且此命令无效。

pebble “L”- 如果当前单元至少包含一块卵石，则转到标号“L”所在行继续执行；否则，执行正常继续，并且此命令无效。

例 2

下面的程序把第一块卵石放在第 0 行并停在那里；如果第 0 行没有卵石它就停在该行的末尾边界上。它使用两个标号 leonardo 和 davinci。

```
right
leonardo:
pebble davinci  # 发现了卵石
border davinci  # 行尾
move
jump leonardo
```

davinci:
halt

里程表从向右转开始。循环从标号声明 leonardo:开始到 jump leonardo 命令结束。在循环中里程表检查卵石或者在行末检查边界的存在；如果不存在，里程表执行从当前单元（0，j）move 到单元（0，j+1）因为后者存在。（命令 halt 在这里并不是必须的，因为程序无论如何都会停止）。

Statement

你需要使用上面所描述的里程表的语言提交程序，以便使里程表如期望的那样行动。每一个子任务（见下文）描述里程表需要完成的一个动作，以及所提交的解必须满足的限制。限制涉及如下两方面。

Program size-程序必须足够短。程序的大小是它所包含的命令的数量。标号声明，注释和空行不记入程序的大小。

Execution length — 程序必须结束得足够快. 执行的长度是所执行的步数: 每个单条命令计为一步，而不管该命令是否有效; 注释和空行不计步数。

在例 1 中，程序的大小是 4，执行长度也是 4。在例 2 中，程序的大小是 6, 当在网格中执行且单元(0, 10)中只有一块卵石时, 执行长度为 43 步: 右转，循环 10 次，每次 4 步 (pebble davinci; border davinci; move; jump leonardo), 最后是 pebble davinci and halt。

Subtask 1 [9 points]

在开始时，在单元（0，0）中有 X 块卵石，单元（0，1）中有 Y 块卵石，其他单元都为空。记住，任意一个单元中最多只能有 15 块卵石。写一个程序，使得里程表在 X ≤ Y 时停在单元（0，0）中，否则停在单元（0，1）中。（我们不在意里程表最后的朝向；我们也不在意最后网格中有多少块卵石，以及它们在什么位置）

限制：程序大小 ≤ 100,执行长度 ≤ 1000。

Subtask 2 [12 points]

与上面的任务相同，但是当程序结束时，单元（0，0）必须刚好包含 X 块卵石，单元（0，1）中刚好有 Y 块卵石。

限制：程序大小 ≤ 200,执行长度 ≤ 2000。

Subtask 3 [19 points]

刚好有两块卵石在第 0 行中：一块在单元（0，x）中，另一块在单元（0，y）中；x, y 不相同并且 x+y 为偶数。写一个程序，把里程表留在单元(0, (x + y) / 2)中，即：严格地位于包含卵石的两个单元的中点。网格的最后状态无关紧要。

限制：程序大小 ≤ 100,执行长度 ≤ 200000。

Subtask 4 [up to 32 points]

网格中最多有 15 块卵石，没有任何两块在同一个单元中。写一个程序，把它们都收集到西北角，更准确地说，如果开始时网格中有 x 块卵石，则结束时单元（0，0）中必须有 x 块卵石，并且其他地方没有卵石。

这个子任务的得分取决于程序的执行长度。如果 L 是在各种测试样例下的最大执行长度，你的得分是

32 points if $L \leq 200\,000$;

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

0 points if $L \geq 2\,000\,000$.

限制：程序大小 ≤ 200.

Subtask 5 [up to 28 points]

每个单元中有任意数量的卵石（当然在 0 和 15 之间），写一个程序，找到最小值，即程序当里程表位于单元(i, j)时停止，并且每个其他单元包括至少如（i，j）单元那么多的卵石。程序运行之后，每个单元当中的卵石数，必须与程序运行前一样。

本子任务的得分取决于程序大小 P，更准确的说：

28 points if $P \leq 444$;

$28 - 28 \log_{10}(P / 444)$ points if $444 < P < 4\,440$;

0 points if $P \geq 4\,440$.

限制: 执行长度 $\leq 44\ 400\ 000$.

实现的细节

你为每一个子任务提交一个文件，按照上面给定的语法书写。每个提交的文件最大为 5MB。对每一个子任务你的里程表代码将用一些用例测试，你会得到关于你的代码使用资源的反馈信息。如果代码的语法不正确因此不能测试，你将收到描述语法错误的信息，你不必为所有的子任务提交里程表程序。如果你当前提交的程序中不包含子任务 X 的里程表程序，你最近提交的关于子任务 X 的程序将被自动的包含；如果这样的程序不存在，对该提交的程序相应子任务得分为零。

一次提交的得分等于其所包含的各个子任务得分之和，该任务的最后得分是历次提交得分当中的最高得分。

模拟器

为进行测试，给你提供了一个里程表模拟器，你可以向它提交你的程序和输入网格。里程表程序将与提交程序的格式相同（即如上面所描述的那样）。

网格描述将使用下述格式给出：文件的每一行都包含 3 个数，R，C 和 P，表示该单元位于 R 行 C 列，包含 P 块卵石。网格中所有未说明的单元都包含卵石。

例如下面的文件

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

所描述的网格包含 15 块卵石：3 块在单元（0，10）中，12 块在单元（4，5）中。

你可以在你的任务目录中调用程序 simulator.py 来启动模拟器，把程序文件名作为参数。模拟器程序接受下列命令行选项：

-h 简单列出所有的命令行选项。

-g GRID_FILE 从文件 GRID_FILE 中加载网格描述 (默认中：空网格);

-s GRID_SIDE 把网格的大小设置为 GRID_SIDE x GRID_SIDE (默认值: 256, 如在程序说明当中使用); 使用较小的网格有助于程序调试;

-m STEPS 限制模拟执行步数最多为 STEPS;

-c 进入编译模式；在编译模式；模拟器返回相同的输出，但是，它生成并编译一个小的 C 程序，而不是使用 Python 进行模拟。这使得在开始时，开销较大，但是，程序运行很快；当你的程序预期运行超过 10 000 000 steps 时最好使用这种模式。

提交次数

本任务允许的最大提交次数为 128。