Odómetro de piedras

Leonardo inventó el "odómetro" original: un auto que podía medir distancias a partir de ir dejando piedras cada que las llantas giraban. Contando las piedras se obtenía la cantidad de vueltas que daba la rueda, esto permitía al usuario calcular la distancia recorrida por el odómetro. Como computólogos, hemos añadido control vía software al odómetro, extendiendo sus funcionalidades. Tu tarea es programar el odómetro bajo las reglas especificadas abajo.

Rejilla de operación

El odómetro se mueve en una rejilla cuadrada de 256 x 256 celdas, cada celda puede contener a lo más 15 piedritas y es identificada por un par de coordenadas (fila, columna), donde cada coordenada se encuentra en el rango 0, …, 255. Dada una celda (i, j), las celdas adyacentes a ella son (si es qué existen): (i – 1, j), (i + 1, j), (i, j - 1) y (i, j + 1). Cualquier celda sobre la primera o ultima fila, o en la primera o ultima columna, es llamada un "borde". El odómetro siempre inicia en la celda (0,0) (la esquina noroeste), mirando al norte.

Comandos Básicos

El odómetro puede ser programado usando los siguientes comandos.

- `left` — gira 90 grados a la izquierda (sentido inverso de las manecillas del reloj) y permanece en la misma celda (por ejemplo si estaba mirando al sur antes, entonces mirará al este después del comando).
- `right` — gira 90 grados a la derecha (sentido de las manecillas del reloj) y permanece en la misma celda. (Por ejemplo si estaba mirando al oeste antes, entonces mirará al norte después del comando).
- `move` — mueve la unidad hacia adelante (en la dirección que esta mirando el odómetro) hacia una celda adyacente. Si dicha celda no existe (por ejemplo en el borde en la dirección que lo hizo llegar al borde) entonces este comando no tiene efecto.
- `get` — quita una piedra de la celda actual. Si la celda no tiene piedras, entonces el comando no tiene efecto.
- `put` — agrega una piedra a la celda actual. Si la celda contiene 15 piedras, entonces el comando no tiene efecto. El odómetro nunca se quedará sin piedras.
- `halt` — Termina la ejecución.

El odómetro ejecuta los comandos en el orden que son dados por el programa. El programa debe contener a lo más un comando por línea. Líneas vacías son ignoradas. El símbolo # indica un comentario, cualquier texto que le siga, hasta el fin de línea, es ignorado. Si el odómetro alcanza el fin del programa entonces la ejecución termina.

Ejemplo 1

Considera el siguiente programa para el odómetro. Este lleva al odómetro a la celda (0,2), mirando al este. (note que el primer `move` es ignorado, dado que el odómetro es en la esquina noroeste mirando al norte.)

```
move  # Sin efecto.
right
# Ahora el odómetro mira al éste.
move
move
```

Etiquetas, bordes y piedras.

Para alterar el flujo del programa dependiendo del estado actual, puedes usar etiquetas, las cuales son cadenas sensibles a mayúsculas de a lo más 128 símbolos elegidos de entre `a`, …, `z`, < code>A</code>, …, `Z`, < code>0</code>, …, `9`. Los nuevos comandos concernientes a las etiquetas son listados debajo. En las descripciones de abajo, `"L"` denota cualquier etiqueta válida.

- *`L`*`:` (es decir *`L`* seguido por dos puntos '`:`') — declara el lugar dentro de un programa para una etiqueta *`L`*. Todas las etiquetas declaradas deben ser únicas. Declarar una etiqueta no tiene ningún efecto en el odómetro.

- `jump` *`L`* — continua la ejecución a partir de un salto incondicionado a la línea con la etiqueta *`L`*.

- `border` *`L`* — continua la ejecución saltando a la línea con la etiqueta *`L`*, si el odómetro está en el borde mirando hacia la orilla de la rejilla (es decir una instrucción < code>move</code> no tendría efecto); en otro caso, la ejecución continua normalmente y este comando no tiene efecto.

- `pebble` *`L`* — continua la ejecución saltando a la línea con la etiqueta *`L`*, si la celda actual contiene al menos una piedra, de otro modo, la ejecución continua normalmente y este comando no tiene efecto.

Ejemplo 2

El siguiente programa encuentra la primer piedra en la fila 0 y se detiene ahí, si no hay piedras en la fila 0 entonces se detiene en el borde al final de la fila. Además usa dos etiquetas: `leonardo` y < code>davinci</code>.

```
right
leonardo:
pebble davinci  # Piedra encontrada.
border davinci  # Fin de la línea.
move
jump leonardo
davinci:
halt
```

El odómetro inicia girando a su derecha. El ciclo inicia con la declaración de la etiqueta `leonardo:` y termina con el comando `jump leonardo`. En el ciclo el odómetro hecha por la presencia de una piedra o por el borde al final de la fila, si no sucede nada de esto, el odómetro hace un `move` a partir de la celda actual (0, j) hacia la celda adyacente (0, j + 1) mientras que la celda destino existe. (El comando `halt` no es estrictamente necesario aquí dado que el programa termina de cualquier modo.)

Problema

Debes enviar un programa en el lenguaje del odómetro, descrito arriba, que haga que el odómetro se comporte como se espera. Cada sub-tarea (ver abajo) especifica un comportamiento del odómetro que es necesario cumplir y las restricciones que la solución enviada debe satisfacer. Las restricciones conciernen los siguientes dos tipos:

- *Tamaño del programa* — El programa debe ser lo suficientemente corto, el tamaño de un programa está dado por la cantidad de comandos que contiene. Declaraciones de etiquetas y líneas en blanco "no cuentan" en el tamaño.

- *Longitud de ejecución* — el programa debe terminar lo suficientemente rápido. La longitud de ejecución es el numero de pasos hechos: cada comando simple cuenta como un paso, sin importar si el comando tiene efecto o no; declaraciones de etiquetas, comentarios y líneas en blanco no cuentan como un paso.

En el Ejemplo 1, el tamaño del programa es 4 y la longitud de ejecución es 4. En el Ejemplo 2, el tamaño del programa es 6 y, cuando se ejecuta en una rejilla con una simple piedra en la celda (0, 10), la longitud de ejecución es de 43 pasos: `right`, 10 iteraciones del ciclo, cada iteración toma 4 pasos (`pebble davinci`; < code>border davinci</code>; `move`; < code>jump leonardo</code>), y finalmente , `pebble davinci` y `halt`.

Sub-tarea 1 [9 puntos]

Al inicio hay `x` piedras en la celda (0,0) y `y` en la celda (0,1), mientras que todas las demás celdas están libres. Recuerda que puede haber a lo más 15 piedras en cada celda. Escribe un programa que termine con el odómetro en la celda (0,0) si $x \leq y$, y en la celda (0,1) en caso contrario. (No tomamos en consideración la dirección del odómetro al final de la ejecución, y tampoco cuantas piedras quedan al final en la rejilla, o donde es que están localizadas.)

Limites: tamaño del programa ≤ 100, longitud de ejecución ≤ 1,000.

Sub-tarea 2 [12 puntos]

La misma tarea que arriba, pero cuando el programa termine, la celda (0,0) debe contener exactamente x piedras y la celda (0,1) debe contener exactamente y piedras.

Limites: tamaño del programa ≤ 200, longitud de ejecución ≤ 2,000.

Sub-tarea 3 [19 puntos]

Hay exactamente dos piedras en algún lugar de la fila 0, una es en la celda (0, x) y la otra en la celda (0, y), `x` y `y` son distintas, y `x + y` es par. Escribe un programa que deje al odómetro en la celda (0, (x+y) / 2), es decir exactamente en el punto medio entre las dos celdas que contienen piedras. El estado final de la rejilla no es relevante.

Limites: tamaño del programa ≤ 100, longitud de ejecución ≤ 200,000.

Sub-tarea 4 [hasta 32 puntos]

Hay a lo más 15 piedras en la rejilla y no hay dos de ellas en la misma celda. Escribe un programa que las junte a todas ellas en la esquina noroeste, más precisamente, si hay x piedras en la rejilla al inicio, al final debe haber exactamente x piedras en la celda (0,0) y no debe haber piedras en ningún otro lugar.

El puntaje para esta sub-tarea depende de la longitud de ejecución del programa enviado. Más precisamente, si L es el máximo de las longitudes de ejecución en los varios casos de prueba, tu puntaje será:

- 32 puntos si L ≤ 200,000;
- 32 - 32 $\log_{10}$ (L / 200,000) puntos si 200,000 < L < 2,000,000;
- 0 puntos si L ≥ 2,000,000.

Limites: tamaño del programa ≤ 200.

Sub-tarea 5 [hasta 28 puntos]

Hay un número arbitrario de piedras en cada celda de la rejilla, (por supuesto, entre 0 y 15). Escribe un programa que encuentre el mínimo, es decir, que termine con el odómetro en la celda (i, j) tal que cada una de las otras celdas contiene al menos tantas piedras como (i, j). Después de correr el programa el número de piedras debe ser el mismo que antes de haberlo corrido.

El puntaje para esta sub-tarea depende del tamaño P del programa enviado. Más precisamente, tu puntaje será:

- 28 puntos si P ≤ 444;

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

Limites: longitud de ejecución ≤ 44 400 000.

Detalles de implementación

Debes enviar un solo archivo por sub-tarea, escrito de acuerdo a las reglas de sintaxis descritas arriba. Cada archivo enviado puede tener un tamaño máximo de 5 MiB. Por cada sub-tarea tu código para el odómetro será evaluado con un pequeño grupo de casos, y recibirás retroalimentación de los recursos usados por tu código. En el caso de que el código no sea sintácticamente correcto y por ende sea imposible de evaluar, recibirás información específica acerca del error de sintaxis.

No es necesario que tus envíos contengan programas para todas las sub-tareas. Si tu envío actual no contiene el programa para la sub-tarea X, tu envió más reciente para la sub-tarea X es automáticamente incluido, si no existe dicho programa, el puntaje de la sub-tarea será cero para ese envío.

Como de costumbre, el puntaje de un envió es la suma de los puntajes obtenidos en cada sub-tarea, y el puntaje final de la tarea es el máximo puntaje de entre los envíos liberados y el último envío.

Simulador

Para propósitos de pruebas, se te provee de un simulador de odómetro, el cual puede ser alimentado con tus programas y rejillas de entrada. Los programas para el odómetro deben ser escritos en el mismo formato usado para el envío (es decir el descrito arriba).

La descripción de las rejillas deben ser dadas usando el siguiente formato: cada línea del archivo debe contener tres números, R, C y P, significando que la celda en la fila R y columna C contiene P piedras. Para todas las celdas no especificadas en la descripción de la rejilla se asume que no contienen piedras. Por ejemplo considera el archivo:

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

La rejilla descrita en este archivo contiene 15 piedras: 3 en la celda (0, 10) y 12 en la celda (4, 5).

Puedes invocar al simulador de pruebas llamando al programa `simulator.py` en el directorio de la tarea, pasando el nombre del programa como argumento. El programa de simulación aceptará las siguientes opciones de línea de comando:

- `-h` dará una pequeña descripción de las opciones disponibles;
- `-g ARCHIVO_REJILLA` carga la descripción de la rejilla del archivo `ARCHIVO_REJILLA` (Por defecto: Rejilla vacia);

- `-s LADO_REJILLA` establece el tamaño del lado de la rejilla a < code> LADO_REJILLA x LADO_REJILLA </code> (por defecto: 256, como se usa en la especificación del problema); el uso de rejillas más pequeñas puede ser útil para propósitos de depuración;

- `-m PASOS` limita el numero máximo de pasos durante la simulacion a `PASOS`;

- `-c` entra en modo compilación; en modo compilacion, el simulador regresa exactamente la misma salida, pero en vez de hacerlo con Pyton, genera y compila un pequeño programa de C. Esto causa que el inicio sea más lento, pero después genera resultados significativamente más rápidos, se te recomienda usarlo cuando esperes que tu programa corra por más de 10,000,000 de pasos.

Número de envíos

La cantidad máxima de envíos permitidos para esta tarea es 128.