Odómetro de Piedras

Leonardo inventó originalmente el *odometro*: un carro que podía medir distancias dejando caer piedras a medida que las ruedas del carro giran. Contando las piedras se puede obtener el número de vueltas, lo que permite al usuario calcular la distancia recorrida por el odómetro. Como informáticos, hemos agregado un software de control al odómetro, extendiendo sus funcionalidades. Tu tarea es programar el odómetro bajo las reglas especificadas a continuación.

Grilla de Operación

El odómetro se mueve en una grilla cuadrado imaginaria de 256 x 256 celdas. Cada celda puede contener a lo más 15 piedras y está identificada por un par de coordenadas (fila, columna), donde cada coordenada está en el rango 0, …, 255. Dada una celda (i, j), las celdas adyacentes a esta son (si es que existen) (i - 1, j), (i + 1, j), (i, j - 1) y (i, j + 1). Cualquier celda que se encuentre en la primera o última fila, o en la primera o última columna, es denominada un *borde*. El odómetro siempre parte en la celda (0, 0) (la esquina nor-oeste), mirando hacia el norte.

Comandos Básicos

El odómetro puede ser programado utilizando los siguientes comandos.

- `left` — gira 90 grados a la izquierda (contrario a las manecillas del rejos) y permanece en la celda actual (p.e. si estaba mirando hacia el sur antes del comando, entonces mirará hacia el este después del comando).
- `right` — gira 90 grados a la derecha (como las manecillas del reloj) y permanece en la celda actual (p.e. si estaba mirando hacia el oeste antes del comando, entonces mirará hacia el norte después del comando).
- `move` — mueve una unidad hacia adelante (en la dirección en la que mira el odómetro) hacia la celda adyacente. Si tal celda no existe (es decir, el borde en esa dirección ya se ha alcanzado) entonces este comando no tiene efecto.
- `get` — remueve una piedra de la celda actual. Si la celda actual no tiene piedras, entonces el comando no tiene efecto.
- `put` — agrega una piedra a la celda actual. Si la celda actual ya contiene 15 piedras, entonces el comando no tiene efecto. Al odómetro nunca se le acaban las piedras.
- `halt` — termina la ejecución.

El odómetro ejecuta los comandos en el order en que están dados en el programa, un comando por linea. El programa debe tener a lo más un comando por linea. Las lineas vacias son ignoradas. El símbolo `#` indica un comentario; cualquier texto que le sigue, hasta el final de la linea, es ignorado. Si el odómetro llega al final del programa, la ejecución se termina.

Ejemplo 1

Considera el siguiente programa para el odómetro. Éste lleva al odómetro a la celda (0, 2), mirando hacia el este. (Nota que el primer `move` es ignorado, ya que el odómetro esta en la esquina nor-oeste mirando hacia el norte).

```
move  # sin efecto
right
# ahora el odometro esta mirando hacia el este
move
move
```

Etiquetas, bordes y piedras

Para alterar el flujo del programa dependiendo en el estado actual, puedes usar etiquetas, que son cadenas sensibles a mayúsculas y minúsculas de a lo más 128 símbolos elegidos entre `a`, …, `z`, `A`, …, `Z`, `0`, …, `9`. Los nuevos comandos relevantes para las etuiquetas se describe a continuación. En la siguiente descripción *`L`* denota cualquier etuiqueta válida.

- *`L`*`:` (es decir *`L`* seguido de dos puntos‘`:`’) — declara la localización dentro del programa de la etiqueta*`L`*. Todas las etiquetas decladaras deben ser únicas. Declarar una etiqueta no tiene efecto para el odómetro.
- `jump` *`L`* — continua la ejecución saltando incondicionalmente a la linea con la etiqueta *`L`*.
- `border` *`L`* — continua la ejecución saltando a la linea con la etiqueta *`L`*, si el odómetro se encuentra en un borde enfrentando hacia afuera de la grilla (es decir la instrucción `move` no tendría efecto); en otro caso, la ejecución continua normalmente y el comando no tiene efecto.
- `pebble` *`L`* — continua la ejecución saltando hacia la linea con la etiqueta *`L`*, si la celda actual contiene al menos una piedra; en otro caso, la ejecución continua normalmente y el comando no tiene efecto.

Ejemplo 2

El siguiente programa encuentra la primera (de oeste a este) piedra en la fila 0 y se detiene ahí; si no hay piedras en la fila , se detiene en el borde al final de la fila. El programa usa dos etiquetas `leonardo` and `davinci`.

```
right
leonardo:
pebble davinci  # piedra encontrada
border davinci  # fin de la fila
move
jump leonardo
davinci:
halt
```

El odómetro comienza por girar a su derecha. El ciclo comienza con la declaración de la etiqueta `leonardo:` y termina con el comando `jump leonardo`. En el ciclo, el odómetro comprueba la presencia de piedras o del borde al final de la fila; si ninguna de las dos se cumple, el odómetro hace un `move` desde la celda actual (0, j) a la celda adyacente (0, j + 1) ya que la esta existe. (El comando `halt` no es estrictamente necesario aquí ya que el programa terminaría en cualquier caso).

Enunciado

Tú debes enviar un programa en el lenguaje propio del odómetro, como se describió anteriormente, que haga que el odómetro se comporte como se espera. Cada subtarea (ver más abajo) especifica el comportamiento que se requiere que el odómetro cumpla y las restricciones que el programa enviado debe satisfacer. Las restriciones corresponden a los siguientes dos materias.

- *Tamaño del programa* — el programa debe ser lo suficientemente corto. El tamaño del programa es el número de comandos en él. Declaraciones de etiquetas, comentarios y lineas en blanco no son contadas en el tamaño.
- *Largo de ejecución* — el programa debe terminar lo suficientemente rápido. El largo de la ejecución es el número de *pasos* realizados: cada una de las ejecuciones de un comando cuenta como un paso, independientemente de si el comando tuvo un efecto o no; declaraciones de etiquetas, comentarios y lineas en blanco no cuentan como pasos.

En el ejemplo 1, el tamaño del programa es 4 y el tiempo de ejecución es 4. En el ejemplo 2, el tamaño del programa es 6 y, cuando es ejecutado en una grilla con una sola piedra en la celda (0, 10), el largo de la ejecución es 43 pasos: `right`, 10 iteraciones del ciclo, cada iteración tomando 4 pasos (`pebble davinci`; `border davinci`; `move`; `jump leonardo`), y finalmente, `pebble davinci` y`halt`.

Sub-tarea 1 [9 puntos]

Al comienzo hay x piedras en la celda (0, 0) e y en la celda (0, 1), mientras que las demás otras celdas están vacias. Recuerda que pueden haber a lo mas 15 piedras en cualquier celda. Escribe un programa que termina con el odómetro en la celda (0, 0) si $x \leq y$, y en la celda (0, 1) en otro caso. (No importa cual es la dirección en la que mira el odómetro al final; tampoco importa cuantas

piedras están presentes en la grilla al final, o donde esten localizadas).

Límites: tamaño del programa ≤ 100, largo de ejecución ≤ 1 000.

Sub-tarea 2 [12 puntos]

Esta tarea es igual que la anterior, pero cuando el programa termina, la celda (0, 0) debe contener exactamente x piedras y la celda (0, 1) debe contener exactamente y piedras.

Límites: tamaño del programa ≤ 200, largo de ejecución ≤ 2 000.

Sub-tarea3 [19 puntos]

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

Límites: tamaño del programa ≤ 100, largo de ejecución ≤ 200 000.

Sub-tarea 4 [hasta 32 puntos]

Hay a lo más 15 piedras en la grilla, ningúna de ellas en la misma celda. Escribe un programa para juntarlas todas en la esquina nor-oeste; más precisamente, si al comienzo habían x piedras al principio, al final debe haber exactamente x piedras en la celda (0, 0) y ninguna piedra en otras celdas.

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

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

Límites: tamaño del programa ≤ 200.

Sub-tarea 5 [hasta 28 puntos]

Puede haber cualquier número de piedras en cada celda de la grilla (por supuesto, entre 0 y 15). Escribe un programa que encuentra el mínimo, es decir, que termina con el odómetro en la celda (i, j) de modo que cualquier otra celda contenga al menos tantas piedras como la celda (i, j). Luego de correr el programa, el número de piedras en cada celda debe ser el mismo que antes de correr el programa.

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

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

Límites: largo de ejecución ≤ 44 400 000.

Detalles de la Implementación

Tú debes enviar exactamente un archivo por cada sub-tarea, escrito de acuerdo a las reglas de sintaxis especificadas anteriormente. Cada tarea enviada puede pesar un máximo de 5MiB. Para cada sub-tarea, tu código del odómetro será probado en unos cuantos casos de prueba, y recibiras algún retroalimentación con respecto a los recursos utilizados por tu código. En caso de que el código sea sintacticamente incorrecto, y por lo tanto imposible de probar, recibirás información del error de sintaxis en específico.

No es necesario que tus envíos contengan programas de odómetro para todas las sub-tareas. Si tu envío actual no contiene un programa de odómetro para la sub-tarea X, tu envío mas reciente para la sub-tarea X será incluido automáticamente; si no existe tal programa, la sub-tarea tendrá un puntaje de cero para ese envío.

Como es usual, el puntaje de un envío es la suma de los puntajes objenidos en cada una de las sub-tareas, y el puntaje final de la tarea es el máximo puntaje entre las envíos liberados-probados y el último envío.

Simulador

Para que puedas probar tu programa se te proveerá de un simulador, que puedes utilizar con tus programas y grillas de entrada. Los programas de odometro serán escritos en el mismo formato utilizado para envío (es decir, como se describió anteriormente).

Las descripciones de la grilla deben ser entregadas usando el siguiente formato: cada linea del archivo debe contener tres números, R, C y P, que significa que la celda en la fila R y columna C contiene P piedras. Se asume que todas las celdas no especificadas en la descripción de la grilla no contienen piedras. Por ejemplo, considera el archivo:

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

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

Puedes llamar al simulador de prueba llamando al programa `simulator.py` en tu directorio de la tarea, pasando como argumento el nombre del archivo de tu programa. El programa del simulador acepta las siguientes opciones en la linea de comandos:

- `-h` entrega una breve descripción de las opciones disponibles;
- `-g GRID_FILE` carga la descripción de la grilla desde el archivo `GRID_FILE` (por defecto la grilla está vacia);
- `-s GRID_SIDE` fija el tamaño de la grilla a `GRID_SIDE x GRID_SIDE` (por defecto: 256, como se usa en la especificación del problema); el uso de grillas más pequeñas puede ser útil para corregir errores del programa;
- `-m STEPS` limita el número de pasos de la ejecución a lo más `STEPS`;
- `-c` activa el modo de compilación; en el modo de compilación, el simulador retorna exactamente el mismo resultado, pero en vez de hacer la simulación en Python, se genera y compila un pequeño programa en C. Esto genera un mayor sobrecosto al comenzar, pero entrega resultados significativamente más rápido después; se te recomienda utilizar esta opción esperes que tu programa correrá más que 10 000 000 de pasos.

Número de Envíos

El número máximo de envíos permitidos para esta tarea es 128.