Odómetro de piedras

Leonardo fue el inventor original del Odómetro: usando carrito que podía medir distancias dejando caer piedras mientras las ruedas del carrito giraban. Contando las piedras daba el número de giros de la rueda, lo que le permitía al usuario calcular la distancia recorrida por el odómetro. Como cientificos informáticos, hemos añadido un software para controlar el odómetro, y aumentar sus funcionalidades. Tu tarea es programar el odómetro bajo las reglas especificadas a continuación.

Tablero de operaciones

El odómetro se mueve en un tablero cuadrado imaginario de 256x256 celdas. Cada celda contiene como mucho 15 piedras y se identifica cada celda por un par de coordenadas (fila, columna), donde cada una se encuentra en el rango 0, ..., 255. Dada una celda (i, j), sus celdas adyacentes (si existen) son (i - 1, j), (i + 1, j), (i, j - 1) y (i, j + 1). Cualquier celda de la primera o última fila o de la primera o última columna se llama borde. El odómetro siempre empieza en la celda (0, 0) (la esquina noroeste), mirando el norte.

Comandos básicos

El odometro puede ser programado usando los siguientes comandos:

- `left` — gira 90 grados a la izquierda (en el sentido contrario a las agujas del reloj) y se queda en la misma celda (por ejemplo si miraba hacia el Sur, después mirará hacia el Este).
- `right` — gira 90 grados a la derecha (en el sentido de las agujas del reloj) y se queda en la misma celda (por ejemplo si miraba hacia el Oeste, después mirará hacia el Norte).
- `move` — se mueve desde la celda actual hacia adelante (en la dirección en la cual mira el odómetro) hacia la celda adyacente. Si no existe dicha celda (o sea ya está en el borde de esa dirección) el comando no hace nada.
- `get` — remueve una piedra de la celda actual. Si en la celda actual no hay piedres, entonces el comando no tiene ningun efecto.
- `put` — agrega una piedra en la celda actual. Si en la celda actual ya hay 15 piedras, entonces este comando no tiene efecto. El odómetro nunca rueda fuera de piedras.
- `halt` — termina el programa.

El odómetro ejecuta los comandos en el mismo orden en que son dados en el programa, un comando por linea. El programa debe contener al menos un comando. Lineas vacias son ignoradas. El simbolo `#` indica un comentario; todo texto en la misma linea de este simbolo es ignorado. Si el odómetro llega la final del programa la corrida termina.

Ejemplo 1

Considera el siguiente programa para ser ejecutado por el odómetro. Tomando el odómetro en la celda (0, 2), mirando al Oeste. (Observe que el primer`move` es ignorado porque el odómetro está ubicado en la noroeste y mirando hacia el norte).

```
move  # sin efecto
right
#  ahora el odométro está mirando hacia el este
move
move
```

Etiquetas, bordes y piedras

Para alterar el flujo de programa dependiendo de un estado actual, puedes usar Etiquetas, las cuales son strings "case-sensitive" de maximo 128 caracteres de `a`, …, `z`, `A`, …, `Z`, `0`, …, `9`. Los comandos asociados a una etiqueta se muestrasn a continuacion. En la siguiente descripcion *`L`* denota una etiqueta valida.

- *`L`*`:` (es decir: *`L`* seguido por '`:`') — declara la ubicacion dentro del programa de la etiqueta *`L`*. Toda declaración de etiqueta debe ser única. Declarando una etiqueta no se afecta el odómetro.
- `jump` *`L`* — continua la ejecucion del programa saltando incondicionalmente a la linea de la etiqueta *`L`*.
- `border` *`L`* — continúa la ejecución saltando a la línea con la etiqueta L, si el odómetro se encuentra en un borde enfrentando hacia afuera del tablero (es decir la instrucción `move` no tendría efecto); en otro caso, la ejecución continúa normalmente y el comando no tiene efecto.
- `pebble` *`L`* — continúa la ejecución saltando hacia la línea con la etiqueta *`L`*, si la celda actual contiene al menos una piedra; en otro caso, la ejecución continúa 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` y `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 ésta existe. (El comando `halt` no es estrictamente necesario aquí ya que el programa terminaría en cualquier caso).

Restricciones

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 topicos.

- *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 líneas en blanco no son contadas en el tamaño.
- 'Largo de la 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.*

In Example 1, the program size is 4 and the execution length is 4. In Example 2, the program size is 6 and, when executed on a grid with a single pebble in cell (0, 10), the execution length is 43 steps: `right`, 10 iterations of the loop, each iteration taking 4 steps (`pebble davinci`; `border davinci`; `move`; `jump leonardo`), and finally, `pebble davinci` and `halt`.

Sub-tarea 1 [9 puntos]

Al comienzo hay X piedras en la celda (0, 0) y 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 cualquier 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.

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

Sub-tarea 3 [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.

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

Sub-tarea 4 [hasta 32 puntos]

Hay a lo más 15 piedras en el tablero, 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 $L \geq 2\,000\,000$.

Limites: tamaño del programa ≤ 200.

Sub-tarea 5 [hasta 28 puntos]

Puede haber cualquier número de piedras en cada celda del tablero (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 del tamaño P del programa enviado. Mas 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 $P \geq 4\,440$.

Limites: largo de la ejecución ≤ 44 400 000.

Detalles para la implementación

Debes enviar exactamente un archivo por cada sub-tarea, escrito con las reglas de syntaxis especificadas arriba. Cada archivo sub-tarea debe tener un maximo de 5 MB. Para cada sub-tarea, tu código para el odómetro será probado con unos cuantos casos de prueba, y recibiras alguna retroalimentación con respecto a los recursos utilizados por tu código. En caso de que el código sea syntacticamente 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 tablero de entrada. Los programas de odómetro serán escritos en el mismo formato utilizado para envío (es decir, como se describió anteriormente).

Las descripciones del tablero 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 del tablero no contienen piedras. Por ejemplo, considera el archivo:

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

El tablero descrito 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 línea de comandos:

- `-h` entrega una breve descripción de las opciones disponibles;
- `-g GRID_FILE` carga la descripcion del tablero desde el archivo `GRID_FILE` (default: empty grid);
- `-s GRID_SIDE` fija el tamaño del tablero `GRID_SIDE x GRID_SIDE` (por defecto: 256, como se usa en la especificación del problema); el uso de tableros más pequeños puede ser útil para corregir errores del programa;
- `-m STEPS` limita el número de pasos de la ejecución a lo sumo `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 cuando esperes que tu programa correrá más de 10 000 000 pasos.

Número de envíos

El número máximo de envíos para este problemas es 128.