Odómetro de piedras

Leonardo inventó el odómetro original: un carro que podía medir distancias dejando caer piedras mientras las ruedas del carro giraban. Contando las piedras obtenía el número de giros de la rueda, lo que le permitía computar la distancia recorrida. Como informáticos, hemos añadido 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 por un par de coordenadas (fila, columna), donde cada columna 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 al norte.

Comandos básicos

El odómetro 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 una unidad hacia delante (en la dirección que mira el odómetro) a la celda adyacente. Si no existe dicha celda (o sea ya estamos en el borde en esa dirección) el comando no hace nada.
- `get` - elimina una piedra de la celda actual. Si en la celda actual no hay piedras, entonces el comando no tiene ningún efecto.
- `put` - añade una piedra en la celda actual. Si la celda actual ya tiene 15 piedras, entonces el comando no hace nada. El odómetro nunca se queda sin piedras.
- `halt` - acaba la ejecución.

El odómetro ejecuta comandos en el orden en que se los da el programa. El programa tiene que contener como mucho un comando por línea. Líneas vacías serán ignoradas. El símbolo `#` indica un comentario; cualquier texto que lo sigue, hasta el final de línea, es ignorado. Si el odómetro llega hasta el final del programa, la ejecución se da por terminada.

Ejemplo 1

Considera el siguiente programa para el odómetro. Lleva al odómetro a la celda (0, 2), mirando al este. (Nótese que el primer `move` es ignorado, ya que el odómetro está en la esquina noroeste mirando hacia el norte.)

```
move  # sin efecto
right
# ahora el odómetro mira hacia el este.
move
move
```

Etiquetas, bordes y piedras

Para alterar el curso del programa dependiendo del status actual, puedes usar etiquetas, que son strings de como mucho 128 símbolos elegidos de `a`, ..., `z`, `A`, ..., `Z`, `0`, ..., `9`. Mayúsculas y minúsculas se consideran diferentes. Los nuevos comandos relacionados con las etiquetas se listan a continuación. En la descripción, *`L`* denota cualquier etiqueta válida.

- *`L`*`:` (o sea *`L`* seguido por dos puntos '`:`') - declara la localización dentro del programa de una etiqueta *`L`*. Todas las etiquetas declaradas tienen que ser únicas. Declarar una etiqueta no tiene ningún efecto sobre el odómetro.
- `jump` *`L`* - continua con la ejecución de manera incondicional saltando a la línea con etiqueta *`L`*.
- `border` *`L`* - continúa la ejecución saltando a la línea con etiqueta *`L`*, si el odómetro está en un borde mirando el final del tablero (o sea la instrucción `move` no tendría ningún efecto); en otro caso, la ejecución continuaría normalmente y este comando no tendría ningún efecto.
- `pebble` *`L`* - continua la ejecución saltando a la línea con etiqueta *`L`*, si la celda actual contiene como mínimo una piedra; en caso contrario, la ejecución sigue con normalidad y este comando no tiene ningún efecto.

Ejemplo 2

El programa siguiente localiza la primera piedra (la más al oeste) en la fila 0 y se para allí. Si no hay piedras en la fila 0, se para en el borde al final de la fila. Usa dos etiquetas `leonardo` y `davinci`.

```
right
leonardo:
pebble davinci  # Piedra encontrada
border davinci  # final de la fila
move
jump leonardo
davinci:
halt
```

El odómetro empieza girando a su derecha. El bucle empieza con la declaración de la etiqueta `leonardo:` y acaba con el comando `jump leonardo`. En el bucle, el odómetro busca la presencia de una piedra o borde al final de una fila; si no es el caso, el odómetro hace un `move` de la celda actual (0,j) a la celda adyacente (0, j+1) ya que esta última existe. (El comando `halt` no es estrictamente necesario ya que este programa acabará igualmente.)

Enunciado

Deberías enviar un programa en el lenguaje propio del odómetro, como se ha descrito, que haga realizar la tarea que se le pida al odómetro. Cada subtarea (ver a continuación) especifica el comportamiento que se requiere del odómetro. Las restricciones afectan las dos cuestiones siguientes:

- *tamaño del programa* — el programa ha de ser suficienteme corto. El tamaño del programa es su número de comandos. La declaración de etiquetas, los comentarios y las líneas en blanco no se cuentan.

- *pasos de ejecución* — el programa tiene que acabar dentro de un cierto número de pasos. La ejecución se mide en los *pasos*: cada comando ejecutado cuenta como un paso, indiferentemente de si tiene efecto o no. Las declaraciones de etiquetas, comentarios y líneas en blanco no cuentan como paso.

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

Subtarea 1 [9 puntos]

Al principio hay x piedras en la celda (0, 0) e y piedras en la celda (0, 1), mientras que todas las otras celdas estan vacías. Recuerda que como mucho puede haber 15 piedras en cada casilla. Escribe un programa que acaba con el odómetro en la celda (0, 0) si $x \leq y$, y en la celda (0, 1) en caso contrario. (No nos importa la dirección del odómetro al acabar; tampoco importa cuantas

piedras haya en el tablero al final, o donde se encuentren.)

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

Subtask 2 [12 points]

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

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

Subtask 3 [19 points]

Hay exactamente 2 piedras en alguna celda de la fila 0: una en la celda (0, x), la otra en la celda (0, y); x e y son diferentes y x + y es par. Escribe un programa que deje el odómetro en la celda (0, (x + y) / 2), o sea, exactamente en el punto medio entre las dos celdas que contienen las piedras. El estado final del tablero no es relevante.

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

Subtask 4 [hasta 32 puntos]

Como mucho hay 15 piedras en el tablero, todas en posiciones diferentes. Escribe un programa que las ponga todas en la esquina noroeste. Más precisamente, si al inicio había x piedras en el tablero, al final tiene que haber x piedras en la celda (0, 0) y ninguna piedra en ninguna otra celda.

La puntuación de esta subtarea depende del número de pasos en la ejecución del programa enviado. Más precisamente, si L es el maximo número de pasos en los distintos juegos de prueba, tu puntuación será:

- 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 $L \geq 2\,000\,000$.

Límites: tamaño del programa ≤ 200.

Subtask 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 encuentre el mínimo, o sea acabe con el odómetro en la celda (i, j) tal que cualquier otra celda contenga como mínimo tantas piedras como (i, j). Después de ejecutar el programa, el número de piedras en cada celda tiene que ser el mismo que antes de ejecutar el programa.

La puntuación para esta subtarea depende del tamaño P del programa enviado. En concreto, la puntuación será:

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

Límites: pasos de ejecución ≤ 44 400 000.

Detalles de implementación

Tienes que enviar exactamente un archivo para cada subtarea, escritos segun la sintaxi especificada en las normas anteriores. Cada archivo enviado tiene un tamaño máximo de 5MiB. Para cada subtarea, el código de tu odómetro tiene que testearse en distintos casos, y recibiras cierto feedback. En el caso de que el código no sea sintácticamente correcto y por lo tanto imposible de testear, recibirás información sobre el error específico.

No es necesario que tus envíos contengan programas para todas tus subtareas. Si envías una envio que no contenga el programa que resuelve la subtarea X, tu envio más reciente de X será automáticamente incluida; si no existe tal programa, esa subtarea valdrá 0 para esa entrega.

Como es habitual, la puntuación de un envío es la suma de las puntuaciones de cada subtarea, y la puntuación final de la tarea es la máxima puntuación entre los release-tested envíos y el último envío.

Simulador

Para testear, se te proporciona un simulador del odómetro, al cual le puedes colocar tus programas y tableros de entrada. Los programas para el odómetro serán escritos en el mismo formato que el usado para los envíos (o sea el descrito anteriormente)

Se dará el tablero siguiendo el formato siguiente: cada línea del archivo contendrá 3 números R, C y P que indican que la fila R y columna C tienen P piedras. Se supondrá que todas las celdas no especificadas en el tablero no contienen piedras. Por ejemplo, considera el siguiente archivo:

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

El tablero descrito por este fichero contendría 15 piedras: 3 en la celda (0, 10) y 12 en la celda (4, 5).

Puedes invocar el simulador llamando al programa `simulator.py` en la carpeta de tu tarea, pasando el nombre del fichero del programa como argumento. El simulador aceptará las siguientes opciones de línea de comandos:

- `-h` da un breve resumen de las opciones posibles;

- `-g GRID_FILE` carga la descripción del tablero del fichero `GRID_FILE` (por defecto: tablero vacío);
- `-s GRID_SIDE` fija el tamaño del tablero a `GRID_SIDE x GRID_SIDE` (por defecto: 256, usado en la especificación del problema); el uso de tableros más pequeños puede ser útil para debuggar el programa;
- `-m STEPS` limita el número de pasos en la ejecución de la simulación a como mucho `STEPS`;
- `-c` entra en modo de compilación. En este modo, el simulador devuelve exactamente la misma salida, pero en vez de hacer la simulación en Python, genera y compila un pequeño programa en C. Esto tiene un gran coste al arrancar, pero después da resultados significativamente más rápidos. Se te recomienda usarlo cuando tu programa se espera que se ejecute durante más de alrededor de 10 000 000 pasos.

Número de envíos

El máximo número de envíos permitido para este problema es 128.