Odómetro de piedras

Leonardo inventó el primer *odómetro*: un carro que podía medir distancias al dejar caer piedras a medida que las ruedas del carro giraban. Al contar las piedras se obtenía el número de giros de la rueda, lo que permitía al usuario calcular la distancia recorrida por el odómetro. Como programadores, le hemos añadido un programa de control al odómetro, extendiendo sus funcionalidades. Su tarea es programar el odómetro usando las reglas especificadas a continuación.

Cuadrícula de operación

El odómetro se mueve en una cuadrícula imaginaria de 256 x 256 casillas unitarias. Cada casilla contiene a lo mucho 15 piedras y se identifica por un par de coordenadas (fila, columna), donde cada coordenada está en el rango 0, ..., 255. Dada una casilla (i, j), las casillas adyacentes a ella (si existen) son (i - 1, j), (i + 1, j), (i, j - 1) y (i, j + 1). Cualquier casilla que esté en la primera o la última fila, o en la primera o la última columna, se conoce como un *borde*. El odómetro siempre empieza en la casilla (0, 0) (la esquina nor-occidental), mirando hacia el norte.

Comandos básicos

El odómetro puede ser programado usando los siguientes comandos.

- `left` — girar 90 grados a la izquierda (en sentido contrario a las manecillas del reloj) y quedarse en la casilla actual (es decir, si estaba mirando hacia el sur antes, mirará hacia el este después del comando).
- `right` — girar 90 grados hacia la derecha (en sentido de las manecillas del reloj) y quedarse en la casilla actual (es decir si estaba mirando hacia el oeste antes, mirará hacia el norte después del comando).
- `move` — moverse una casilla hacia adelante (en la dirección que el odómetro esté mirando). Si tal casilla no existe (es decir si el borde en esa dirección ya ha sido alcanzado) este comando no tiene efecto.
- `get` — remover una piedra de la casilla actual. Si la casilla actual no tiene piedras, el comando no tiene efecto.
- `put` — añadir una piedra a la casilla actual. Si la casilla ya contiene 15 piedras, entonces el comando no tiene efecto. El odómetro nunca se queda sin piedras.
- `halt` — terminar la ejecución.

El odómetro ejecuta los comandos en el orden en que están dados en el programa. El programa debe contener por lo menos un comando por línea. Las líneas vacías son ignoradas. El símbolo # indica un comentario; cualquier texto que le siga, hasta el final de la línea, se ignora. Si el odómetro alcanza el final del programa, la ejecución es terminada.

Ejemplo 1

Considere el siguiente programa para el odómetro. Este lleva al odómetro a la casilla (0, 2), mirando hacia el este. (Note que el primer `move` se ignora, puesto que el odómetro está en la esquina nor-occidental mirando hacia el norte.)

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

Rótulos, bordes y piedras

Para alterar el flujo del programa dependiendo del estado actual, puede usar rótulos, que son cadenas en las cuales las mayúsculas se interpretan distinto a las minúsculas de a lo mucho 128 símbolos elegidos de `a`, …, `z`, `A`, …, `Z`, `0`, …, `9`. Los nuevos comandos que se refieren a los rótulos se muestran a continuación. En la descripción, *`L`* se refiere a un rótulo válido.

- *`L`*`:` (es decir *`L`* seguido de dos puntos '`:`') — declara el lugar dentro de un programa con un rótulo *`L`*. Todos los rótulos declarados deben ser únicos. Declarar un rótulo no afecta al odómetro.
- `jump` *`L`* — continúa la ejecución saltando incondicionalmente a la línea con el rótulo *`L`*.
- `border` *`L`* — continúa la ejecución saltando a la línea con el rótulo *`L`* si el odómetro está en un borde mirando hacia el borde de la cuadrícula (es decir si la instrucción `move` no tendría efecto); en caso contrario, la ejecución continúa normalmente y este comando no tiene efecto.
- `pebble` *`L`* — continua la ejecución saltando a la línea con rótulo *`L`* si la casilla actual contiene una piedra; en caso contrario, la ejecución continúa normalmente y este comando no tiene efecto.

Ejemplo 2

El siguiente programa encuentra la primera piedra (la de más al oeste) en la fila 0 y se detiene ahí; si no hay piedras en la fila 0, se detiene en el borde al final de la fila. Usa dos rótulos `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 ciclo comienza con la declaración del rótulo `leonardo:` y termina con el comando `jump leonardo`. En el ciclo, el odómetro revisa la presencia de una piedra o de un borde al final de la fila; si no hay, el odómetro hace un `move` de la celda actual (0, j) a la celda adyacente (0, j + 1) ya que dicha celda existe. (El comando `halt` no es estrictamente necesario aquí ya que el programa termina de cualquier forma.)

Enunciado

Usted debe enviar un programa en el lenguaje del odómetro, como fue descrito anteriormente, que haga que el odómetro se comporte como se espera. Cada subtarea (ver abajo) especifica un comportamiento que se requiere que el odómetro siga y unas restricciones que la solución enviada debe satisfacer. Las restricciones se refieren a los siguientes dos hechos.

- *Tamaño del programa* — el programa debe ser lo suficientemente corto. El tamaño del programa es el número de comandos que tiene. Declaraciones de rótulos, comentarios y líneas en blanco *no se cuentan en el tamaño*.
- *Longitud de ejecución* — el programa debe terminar lo suficientemente rápido. La longitud de ejecución es el número de *pasos* realizados: cada ejecución de un comando cuenta como un paso, independiente de si el comando haya tenido algún efecto o no; declaraciones de rótulos, 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 cuadrícula con una sola piedra en la casilla (0, 10), la longitud de ejecución es de 43 pasos: `right`, 10 iteraciones del ciclo, donde 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 a lo mucho x piedras en la casilla (0, 0), y en la casilla (0, 1) y el resto de casillas está vacía. Recuerde que puede haber a lo mucho 15 piedras en una casilla. Escriba un programa que termine con el odómetro en la casilla (0, 0) si $x \leq y$, y en la casilla (0,1) en caso contrario. (No nos importa la dirección en la que esté mirando el odómetro al final; tampoco nos importa cuántas

piedras quedan al final sobre la cuadrícula, o dónde estén ubicadas.)

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

Subtarea 2 [12 puntos]

Misma tarea que en el punto anterior pero cuando el programa termine la casilla (0, 0) debe tener exactamente x piedras y la casilla (0,1) debe tener exactamente y piedras.

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

Subtarea 3 [19 puntos]

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

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

Subtarea 4 [hasta 32 puntos]

Hay a lo mucho 15 piedras en la cuadrícula, no hay dos de ellas en la misma casilla. Escriba un programa que las recoja todas y las ponga en la esquina nor-occidental; más precisamente, si hubieran x piedras en la cuadrícula al principio, al final debe haber exactamente x piedras en la casilla (0, 0) y ninguna piedra en cualquier otra casilla.

El puntaje para esta subtarea depende de la longitud de ejecución del programa enviado. Más precisamente, si L es el máximo de las longitudes de ejecución de diversos casos de prueba, su 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.

Límites: tamaño del programa ≤ 200.

Subtarea 5 [hasta 28 puntos]

Puede haber cualquier cantidad de piedras en cada casilla de la cuadrícula (por supuesto, entre 0 y 15). Escriba un programa que encuentre el mínimo, es decir, que termine con el odómetro en la casilla (i, j) tal que cada otra casilla contiene al menos la misma cantidad de piedras que (i, j). Después de correr el programa, la cantidad de piedras en cada casilla debe ser la misma que antes de correr el programa.

El puntaje de esta subtarea depende del tamaño del programa P del programa enviado. Más precisamente, su 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: longitud de ejecución ≤ 44 400 000.

Detalles de implementación

Usted tiene que enviar exactamente un archivo por cada subtarea, escrito de acuerdo a las reglas de sintaxis especificadas anteriormente. Cada archivo enviado puede tener un tamaño máximo de 5 MiB. Por cada subtarea, el código de su odómetro será probado en algunos casos de prueba, y usted recibirá algo de retroalimentación sobre los recursos que use su código. En caso de que el código no sea sintácticamente correcto y que por lo tanto sea imposible de probar, usted recibirá información específica sobre el error de sintaxis.

No es necesario que sus envíos contengan programas de odómetro para todas las subtareas. Si su envío actual no contiene el programa odómetro para la subtarea X, su más reciente envío de la subtarea X será automáticamente incluido; si no hay ningún envío, la subtarea obtendrá cero por ese envío.

Como de costumbre, el puntaje de un envío es la suma de puntajes obtenidos en cada subtarea, el puntaje final del problema será el máximo puntaje entre los envíos probados por un release y el último envío.

Simulador

Para propósitos de prueba, a usted se le proveerá con un simulador de odómetro, usted lo podrá alimentar con sus propios programas y cuadrículas. Los programas de odómetro serán escritos en el mismo formato que el formato usado para el envío (es decir, el descrito anteriormente).

Las descripciones de la cuadrícula serán dadas usando el siguiente formato: cada línea del archivo debe contener tres números, R, C y P, que representan que la casilla en la fila R y columna C contiene P piedras. Se asumirá que en todas las casillas no especificadas en la descripción de la cuadrícula no se encontrará ninguna piedra. Por ejemplo, considere el archivo:

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

La cuadrícula descrita por este archivo contendrá 15 pierdras: 3 en la casilla (0, 10) y 12 en la casilla (4, 5).

Usted puede invocar el simulador de prueba llamando el programa `simulator.py` en su directorio de prueba, pasando el nombre del programa como argumento. El programa simulador aceptará las siguientes opciones de línea de comando:

- `-h` dará una breve descripción de las opciones disponibles;
- `-g GRID_FILE` carga la descripción de la cuadrícula desde el archivo `GRID_FILE` (por defecto la cuadrícula estará vacía);
- `-s GRID_SIDE` fija el tamaño de la cuadrícula a `GRID_SIDE x GRID_SIDE` (por defecto es 256 como especifica el problema); el uso de cuadrículas más pequeñas puede ser útil para depurar;
- `-m STEPS` limita el número de pasos de ejecución de la simulación a máximo `STEPS`;
- `-c` entra en modo de compilación; en modo de compilación, el simulador retorna exactamente la misma salida, en lugar de hacer la simulación con Python, genera y compila un pequeño programa en C. Esto causa una mayor sobrecarga al inicio, pero genera resultados más rápidos; se le recomienda usar esto si espera que su programa corra más de 10 000 000 de pasos.

Número de envíos

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