Odometro de piedras

Leonardo inventó un Odómetro original: un carro que podía medir distancias dejando caer piedras mientras las rueddas del carro giraban. Contando las piedras obtenía el número de giros de la rueda, lo que le permitía al usuario computar la distancia recorrida por el odómetro. 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 una tablero cuadrado imaginario de 256x256 celdas. Cada celda contiene como maximo 15 piedras y se identifica por un par de coordenadas (fila, columna), donde cada coordenada 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 "border". El odómetro siempre empieza en la celda (0, 0) (la esquina noroeste), mirando al norte.

Comandos básicos

El odometro puede ser programado usando los siguientes comandos

`left` — gire 90 grados a la izquierda (sentido contrahorario) y permanezca en la celda corriente (e.g. si estaba mirando al sur antes, entonces mirará al este después del comando).

`right` — gire 90 grados a la derecha (sentido horario) y permanezca en la celda corriente (e.g. si estaba mirando al oeste antes, entonces mirará al norte después del comando).

`move` — mueva una unidad hacia adelante (en la dirección en la que apunta el odómetro) hacia la celda adyacente. Si no existe tal celda (i.e. el border en tal dirección ha sido ya alcanzado) entonces este comando no tiene efecto.

`get` — remueva una piedra de la celda corriente. Si la celda corriente no tiene piedras, entonces el comando no tiene efecto.

•`put` — agregue una piedra a la celda corriente. Si la celda corriente ya contiene 15 piedras, entonces el comando no tiene efecto. El odómetro nunca agota sus piedras.

`halt` — termina la ejecución.

El odómetro ejecuta los comandos en el orden en que están dados en el programa. El programa debe contener a lo sumo un comando por línea. Líneas vacías son ignoradas. El símbolo # indica un comentario; cualquier texto que sigue, hasta el fin de la línea, es ignorado. Si el odómetro llega al fin del programa, la ejecución es terminada.

Ejemplo 1

Considere el siguiente programa para el odómetro. Lleva al odómetro a la celda (0, 2), mirando al Este. (Observe que el primer `move` es ignorado, porque el odómetro está sobre la esquina NorOeste mirando al Norte.)

```
move  # no efecto
right
# ahora el odómetro esta mirando al Este
move
move
```

Rótulos, borders y piedras

Para alterar el flujo del programa dependiendo del estado corriente, puedes usar rótulos, que son cadenas sensibles a mayúscula de a lo más 128 símbolos elegidos de `a`, …, `z`, `A`, …, `Z`, `0`, …, `9`. Los nuevos comandos relativos al uso de rótulos están listados abajo. En las descripciones abajo, L denota cualquier rótulo válido.

•*`L`*`:` (i.e. *`L`* seguido por dos puntos '`:`') — declara la posición de un rótulo *`L`* dentro del programa. Todos los rótulos deben ser únicos. Declarar un rótulo no tiene efecto sobre el odómetro.

`jump` *`L`* — continúe la ejecución tras saltar incondicionalmente a la línea con rótulo *`L`*.

•`border` *`L`* — si el odómetro está en un border mirando una arista de la grilla (i.e. una instrucción `move` no tendría efecto) continúe la ejecución saltando a la línea con rótulo *`L`*; de otra forma, la ejecución continúa normalmente y este comando no tiene efecto.

• `pebble` *`L`* — si la celda corriente contiene al menos una piedra, continúe la ejecución saltando a la línea con rótulo *`L`*; de otro modo, la ejecución continúa normalmente y este comando no tiene efecto.

Ejemplo 2

El siguiente programa ubica la primera (la más al oeste) piedra en la fila 0 y allí se detiene; si no hay piedras en fila 0, para en el border al final de la fila. Usa dos rótulos `leonardo` y `davinci`.

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

El odómetro inicia girando a la derecha. El ciclo comienza con la declaración del rótulo

`leonardo:` y termina con el comando jump leonardo. En el ciclo, el odómetro verifica la presencia de una piedra o del border al final de la fila; si no es así, el odómetro efectúa un move de la celda corriente (0, j) a la celda adyacente (0, j + 1) siempre que esta última exista. (El comando halt no es estrictamente necesario aquí ya que el programa termina de todos modos.)

Enunciado

Usted debe enviar un programa en el lenguage propio del odometro, como esta descripto arriba, que marca el comportamiento esperado del odometro. Cada subtarea (ver abajo) especifica un comportamiento que es requierido

- *Tamaño del Programa* — el programa debe ser suficientemente corto. El tamaño es el numero de comandos que hay en el. Rotulos, declaraciones,comentarios y lineas vacias is "no seran tenidos en cuenta" en el tamañno.

- *Longitud de Ejecucion* — Elprograma debe terminar suficientemente rapido. La longitud de ejecucion es el numero de pasos realizados, cada comando de ejecucion simple cuenta como un paso independientemente que el comando tenga efecto o no; rotulos, comentarios y lineas en blanco, no cuentan como un paso.

En el Ejemplo 1, el tamano delprograma es 4 y la longitud de Ejecucion es 4. En el Ejemplo 2 el tamano del programa es 6, cuando se ejecuta con una grilla con una pidra en la celda (0, 10), el largo de ejecucion es 43 pasos: `right`, 10 iteraciones del loop, cada iteracion tiene 4 pasos (`pebble davinci`;`border davinci`;`move`;`jump leonardo`), and finally, `pebble davinci` and `halt`.

Subtarea 1 [9 puntos]

Al inicio, hay x piedras en la celda (0, 0) e y piedras en la celda (0, 1), y el resto de las celdas estan vacias. Recuerde que no puede haber mas de 15 piedras en cada celda. Escriba un programa que termine con el odometro en la celda (0, 0) si $x \leq y$, en la celda (0, 1) en caso contrario, no nos importa la direccion hacia donde esta mirando el odometro, al final ni tampoco cuantas piedras estan al finalen la grilla, ni donde estan ubicadas.

Limites: tamano del programa ≤ 100, longitud de ejecucion ≤ 1 000.

Subtarea 2 [12 puntos]

Misma tarea que elpunto anterior, pero cuando el programa termina, la celda (0, 0) debe contener exactamente x piedras y la (0, 1) debe contener exactamente y piedras

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

Subtarea 3 [19 puntos]

Hay exactamente dos piedras en alguna celda de la fila 0, una en la celda (0, x), otra en la celda (0, y);x e y son distintas, x+y es par. Escriba un programa que deje el odometro en la celda (0, (x + y) / 2) i.e., exactamente y en e lpunto medio de las dos celdas que contienen las piedras. El estado finsl de la Grilla no es relevante.

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

Subtarea 4 [hasta 32 puntos]

Hay como maximo 15 piedras en la grilla, no hay dos en una misma celda . Escriba un programa que las recoja a todas y las deposite en la celda Norte-Oeste ; mas precisamente si hay x piedras en la grilla al principio, al final debe haber 15 piedras en la celda Nor.Oeste

Elpuntaje de esta subtarea depende de la longitud de ejecucion del programa enviado. Mas precisamente, si L es la maxima longitud de ejecucion en varios casos de prueba, su puntaje sera:

- 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: tamano de programa ≤ 200.

Subtarea 5 (hasta 28 puntos)

Seguramente hay una cantidad de piedras en cada celda de la Grilla (por supuesto entre 0 y 15). Escriba un programa que encuentre el minimo, i.e., que termine con elodometro en la celda (i, j) tal que cuaquiera otra celda contenga al menos la misma cantidad de piedras que la celda (i, j). Despues de correr elp rograma el numero de piedras en cada celda, debe ser elmismo que antes de correr elpprograma.

El puntaje paara esta subtarea depende de la longitud del programa P del programa enviado. Mas precisamente, su puntaje sera:

- 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 ejeecucion ≤ 44 400 000.

detalles de Implementacion

Usted tiene que enviar exactamente un archivo por cada sub tarea, escrito de acuerdo con las reglas

de sintaxis especificadas anteriormente. Cada archivo enviado, puede tener un tamaño maximo de 5 MiB. para cada subtarea, el codigo de su odometro sera testeado con algunos casos de prueba y usted recibira alguna realimentacion sobre los recursos que use su codigo.En caso que su codigo no sea sintacticamente correcto, porlo tanto imposible de probar, usted recibira informacion especifica sobre su error de sintaxis.

No es necesrio que sus envios contengan programas de odometro para todas las subtareas. Si su envio actual no contiene el programa odometro para la subtarea X, su mas reciente enviopara la subtarea X, sera automaticamente incluido,si no hay ningun envio, la subtarea X obtendra cero por ese envio.

Como es usual, el puntaje del envío es la suma de los puntajes obtenidos en cada subtarea, y el puntaje final de la tarea es el máximo puntaje entre un release probado y el último envío.

Simulador

Para el propósito de prueba se le proveerá un simulador del odómetro, que podrá alimentar con sus propios programas y la grilla de entrada. Programas de odómetro serán escritos en el mismo formato utilizado para los envíos (i.e., descripto anteriormente).

Descripciones de la frilal serán dados utilizando el siguiente formato: cada línea del archivo debe contener tres números, R, C y P, representando que la celda está en la fila R, la columna C y contiene P piedras. De todas las celdas no especificadas en la grilla se asume que no contienen piedras. Por ejemplo, considere el archivo:

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

La grilla descripta por este file puewsde contener 15 piedras: 3 en la celda (0, 10) y 12 en la celda (4, 5).

Usted puede llamar al simulador de prueba llamando al programa `simulator.py` en su directorio de tareas,

- `-h` will give a brief overview of the available options;
- `-g GRID_FILE` cargha la descripcion de la grilla file `GRID_FILE` (default: empty grid);
- `-s GRID_SIDE` sets the size of the grid to `GRID_SIDE x GRID_SIDE` (default: 256, as used in the problem specification); usage of smaller grids can be useful for program debugging;
- `-m STEPS`limita en numero de pasos en la simulacion a como mnaximo `STEPS`;
- `-c` enters compilation mode; in compilation mode, the simulator returns exactly the same output, but instead of doing the simulation with Python, it generates and compiles a small C program. This causes a larger overhead when starting, but then gives significantly faster results; you are advised to use it when your program is expected to run for more than about 10 000 000 steps.

Numero de envios

La maxima cantidad de envios permitido para este problema es 128.