Entregable 3
Briker

EI1022/MT1022 - Algoritmia 2022/2023

Fecha de entrega: miércoles 23 de noviembre de 2022

Contenido

1. **El problema** 1
2. **Ficheros de partida** 2
3. **Entrega en el aula virtual** 6
4. **Calificación del entregable** 7

1. El problema

Briker es un juego clásico de tipo puzle. Podéis ver su mecánica de juego en este vídeo:

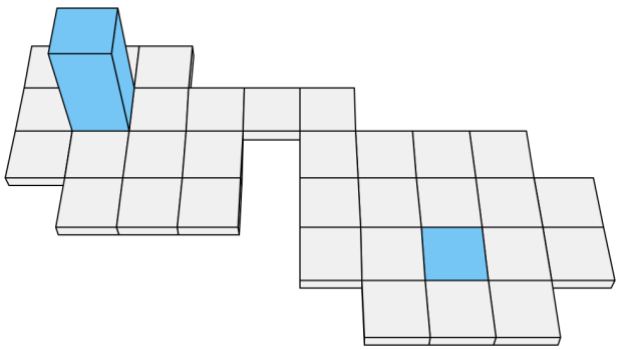
https://youtu.be/ftX59-g6MOk

El juego consiste en mover un bloque (prisma rectangular, ladrillo o, en inglés, *brick*) de tamaño 1x1x2 sobre un tablero para llevarlo desde su posición de partida a una posición objetivo. El bloque se mueve volteándolo y debe estar en todo momento apoyado sobre baldosas.

Un puzle (una instancia) consiste en:

- Un tablero rectangular con baldosas en algunas casillas.
- Una baldosa de partida, donde se coloca inicialmente el bloque.
- Una baldosa objetivo, a la que deberemos llevar el bloque para resolver el puzle. El bloque debe quedar de pie sobre la baldosa

Un ejemplo:



Vuestro objetivo es encontrar la solución que resuelve el puzle con el mínimo número de movimientos. Si hay más de una solución óptima, podéis devolver una cualquiera de ellas. También deberéis tener en cuenta el caso de que un puzle no tenga solución.

Se pide:

- Implementar un programa de línea de órdenes, `entregable3.py`, que reciba, por la entrada estándar, un puzle en el formato que se especifica en el apartado 1.1. Si el puzle tiene solución, el programa deberá mostrar por la salida estándar la secuencia de movimientos más corta que lo resuelve. Si el puzle no tiene solución, el programa deberá mostrar por la salida estándar la línea de texto `"INSTANCE WITHOUT SOLUTION"`. En el apartado 1.2 se detalla el formato de la salida.

- El programa deberá utilizar el esquema de búsqueda con retroceso con optimización.

1.1. Formato de la entrada

Una instancia del problema (un puzle) consistirá en un fichero de líneas de texto.

Veamos un ejemplo. El puzle que aparece al principio del entregable es la instancia del fichero 'puzzle_01_9.i', cuyo contenido es:

```
ooo-------
oSoooo----
oooo-oooo-
-ooo-ooooo
-----ooToo
------ooo-
```

Cada carácter tiene un significado concreto: 'o' indica que hay una baldosa en esa posición, 'S' indica la baldosa de partida del bloque y 'T' la baldosa objetivo. Por último, el carácter '-' aparece en las posiciones que no tienen baldosa.

1.2. Formato de la salida

Si el puzle tiene solución, la salida consistirá en una única linea de texto compuesta exclusivamente con los caracteres 'L', 'R', 'U' y 'D', donde cada caracter representa una dirección del movimiento: *Left*, *Right*, *Up* y *Down*, respectivamente.

Por ejemplo, para el puzle que aparece al principio del entregable (`puzzle_01_9.i`), una salida válida sería la cadena:

```
DRURRDRRD
```

Si un puzle no tiene solución, la salida consistirá exactamente en la siguiente línea de texto:

```
INSTANCE WITHOUT SOLUTION
```

2. Ficheros de partida

En el aula virtual disponéis del fichero `entregable3-briker.zip`, que al descomprimirlo crea una carpeta con el siguiente contenido:

- Ficheros para la implementación:

  - **`direction.py`:** Ya implementado. Define el tipo enumerado `Direction` y la función `directions2string` que convierte secuencias del tipo enumerado en cadenas de texto con el formato indicado en el apartado 1.2.

  - **`board.py`:** Ya implementado. Define la clase `Board`, que nos permite representar puzles a partir de las líneas de texto con su descripción.

  - **`brick.py`:** Define la clase `Brick`, que representa el bloque y permite moverlo en las cuatro direcciones. La clase tiene un método `move` que debéis implementar.

  - **`entregable3.py`**: El programa principal. Debéis implementar las funciones `read_data`, `process` y `show_results`.

- Contenido adicional:
    - **public_test**: carpeta con cinco instancias de prueba.
    - **entregable3_test.py**: programa que importa vuestro entregable3.py y lo utiliza para obtener la solución de una instancia. Luego valida la solución obtenida y el tiempo que ha necesitado vuestra función process para obtenerla.
    - **briker_viewers**: carpeta con tres programas que permiten visualizar las soluciones en 3D. Hay un programa para Linux, otro para Mac OS y otro para MS Windows.

2.1. El fichero direction.py (ya implementado)

Define el tipo enumerado Direction, que tiene cuatro valores posibles: Direction.Right, Direction.Left, Direction.Up y Direction.Down.

El fichero también contiene la implementación de la función

```
def directions2string(solution: Iterable[Direction]) -> str:
```

que, dada una secuencia del tipo enumerado Direction, devuelve una cadena en el formato de salida del apartado 1.2.

Ejemplo:

```
ds = [Direction.Right, Direction.Up, Direction.Down, Direction.Left]
sol = directions2string(ds)
print(sol)  # Muestra por pantalla la cadena 'RUDL'
```

2.2. El fichero board.py (ya implementado)

El fichero contiene dos definiciones de tipo: RowCol y Board.

RowCol es un tipo inmutable muy simple que permite manejar coordenadas en el tablero. Ejemplo de uso:

```
pos = RowCol(3, 5) # Crea un objeto
print(pos.row)     # Accede al campo row del objeto
```

Por otro lado, la clase Board nos permite representar puzles a partir de las líneas de texto con su descripción.

Su constructor construye el objeto a partir de las líneas de texto con la descripción del puzle (formato del apartado 1.1):

```
lines = ['ooo-------', 'oSoooo----', ...]
my_board = Board(lines)
```

Tiene tres métodos públicos:

```
def has_tile(self, pos: RowCol) -> bool:
    # Devuelve True si hay baldosa en la posición indicada

def start_pos(self) -> RowCol:
    # Devuelve la posición de partida del bloque

def target_pos(self) -> RowCol:
    # Devuelve la posición objetivo del bloque
```

2.3. El fichero `brick.py`

`Brick` es la clase que utilizaremos para representar el bloque (prisma rectangular, ladrillo o *brick*) y para moverlo en las cuatro direcciones. Por decisión de diseño, los objetos de esta clase son inmutables.

Podemos considerar que el bloque está formado por dos cubos cuyo lado es la unidad. Así pues, la clase `Brick` sólo necesita dos atributos, `b1` y `b2`, ambos del tipo `RowCol`, con las coordenadas de los dos cubos.

La clase `Brick` tiene un método que debéis implementar (si lo necesitáis podéis añadir métodos adicionales):

```
def move(self, d: Direction) -> Brick:
    # Mueve el bloque. Como el objeto es inmutable, la función 'move'
    # devuelve un nuevo bloque
```

IMPORTANTE: Para simplificar la implementación del método `move` hemos añadido dos restricciones que fijan a qué cubo del bloque corresponden `b1` y `b2`:

- Cuando el bloque esté tumbado sobre una fila (en horizontal), `b1` será el cubo de menor columna.
- Cuando el bloque esté tumbado sobre una columna (en vertical), `b1` será el cubo de menor fila.

Implementar el método `move` es lo primero que debéis hacer del entregable. Podéis utilizar el validador `entregable3_test.py`, que presentaremos en el apartado 2.6, para depurar vuestra implementación.

2.4. El fichero `entregable3.py`

Fichero con la estructura del programa. Debéis utilizarlo con las siguientes restricciones:

- Podéis añadir funciones o clases adicionales, pero **no debéis modificar nada del programa principal**. Modificar el programa principal supondrá un cero en la calificación del entregable. Tampoco podéis modificar la firma (el tipo) de las tres funciones que se utilizan en el programa principal.
- Podéis utilizar la biblioteca `algoritmia`.
- También podéis importar módulos adicionales, pero sólo si están en la *Python Standard Library* (p.ej. `numpy` no lo está).

Veamos la estructura de `entregable3.py` y las funciones que debéis implementar.

Para empezar, y por comodidad, el programa define los tipos `Decision` y `Solution`, que utilizaréis internamente en vuestro método `proces` al seguir el esquema de búsqueda con retroceso:

```
Decision = Direction        # Un alias del tipo enumerado Direction
Solution = tuple[Decision, ...]
```

Como siempre, el programa principal utiliza la estructura de tres funciones vista en clase:

```
board = read_data(sys.stdin)
solution = process(board)
show_results(solution)
```

Este entregable consiste en implementar las tres funciones que aparecen en el programa principal:

- **`read_data`**`(f: TextIO) -> Board`

  Entrada: El fichero con la instancia (ojo, `f` no es un nombre de fichero, es un fichero).

  Salida: Un objeto de la clase `Board`, que contiene toda la información del puzle.

- **process**(board: Board) -> Optional[Solution]

  Entrada: Un parámetro, el tablero que devuelve la función `read_data`.

  Salida: La solución del puzle como una tupla de `Direction`, o None, si el puzle no tiene solución.

- **show_results**(solution: Optional[Solution])

  Entrada: Un parámetro, el mismo que devuelve la función `process`.

  Salida: No devuelve nada. Sólo muestra texto por la salida estándar siguiendo el formato que se detalla en el apartado 1.2.

Seguid con detalle estos pasos para ejecutar el programa `entregable3.py` desde la línea de órdenes:

1. Si no está instalada ya, instalad la biblioteca `algoritmia`. Se ha explicado cómo hacerlo en la segunda sesión de prácticas (la información está disponibles en el aula virtual).

2. Abrid un terminal e id al directorio donde está el fichero `entregable3.py` y la carpeta `public_test`.

3. Ya podéis lanzar el programa con la orden:

   En Linux: `python3.10 entregable3.py < public_test/puzzle_01_9.i`

   En Windows: `python entregable3.py < public_test\puzzle_01_9.i`

   que deberá mostrar por pantalla la línea:

   ```
   DRURRDRRD
   ```

2.5. La carpeta con las instancias de prueba: `public_test`

La carpeta '`public_test`' contiene cinco puzles de prueba.

El nombre de cada instancia sigue el patrón '`puzzle_<n>_<t>.i`', donde `<n>` es el número de puzle y `<t>` es un entero que indica la longitud de la solución óptima. Si `<t>` es 0 significa que el puzle no tiene solución.

2.6. El validador de soluciones: `entregable3_test.py`

Una solución es válida si resuelve el puzle en el número óptimo de movimientos.

El validador de soluciones (`entregable3_test.py`) es un programa que importa tu fichero `entregable3.py` y lo utiliza para resolver una instancia del problema. Para utilizarlo, abre un terminal, ve al directorio donde están `entregable3_test.py`, la carpeta `public_test` y tu `entregable3.py`, y ejecuta la orden:

```
python3.10 entregable3_test.py public_test/puzzle_01_9.i
```

El resultado de la ejecución para una instancia puede ser:

- **OK**: La solución es válida y se ha obtenido dentro del límite de tiempo. Se muestra una salida como esta:

  ```
  INSTANCE: public_test/puzzle_01_9.i
  RESULT: OK
    LENGTH:   OK - 9 movements
    USEDTIME: OK - 0.00 sec (<= 1 sec)
  ```

- **ERROR_LENGTH**: La solución resuelve el puzle, pero no es óptima:

  ```
  INSTANCE: public_test/puzzle_01_9.i
  RESULT: ERROR_LENGTH
    LENGTH:   ERROR - 11 movements (> 9)
    USEDTIME: OK - 0.00 sec (<= 1 sec)
  ```

- **ERROR_TIMEOUT**: La solución es válida, pero se ha superado el límite de tiempo de un segundo. Se muestra una salida como esta:

```
INSTANCE: public_test/puzzle_01_9.i
RESULT: ERROR_TIMEOUT
  LENGTH:   OK - 9 movements
  USEDTIME: ERROR - 1.45 sec (> 1 sec)
```

- **ERROR_INVALID_SOLUTION**. No se ha producido ningún error de ejecución, pero la solución obtenida no es válida. Se muestra el motivo por el que no es válida.

- **ERROR_CHECK_FAILED**. No se ha producido ningún error de ejecución, pero tu programa no ha pasado alguna verificación intermedia. Se muestra el problema detectado.

- **ERROR_EXCEPTION_LAUNCHED**. Se ha producido un error de ejecución que ha lanzado una excepción. Se muestra la excepción que se ha lanzado e información de traza para depuración.

Hay otro resultado posible: que alguna llamada a tu programa no termine (en un plazo razonable) y tengas que pulsar `Ctrl-C` para cancelar la ejecución del validador.

2.7. La carpeta con el visor de soluciones: `briker_viewers`

La carpeta contiene tres compilaciones diferentes del visor de soluciones, según el sistema operativo: `briker_viewer_linux`, `briker_viewer_mac`, y `briker_viewer_win.exe`.

Estos programas deben ejecutarse desde un terminal para poder pasarles los parámetros. Abrid un terminal y, desde el directorio que corresponda, podéis ejecutarlo con la siguiente orden:

- En Linux: `briker_viewers/briker_viewer_linux public_test/puzzle_01_9.i DRURRDRRD`
- En Mac OS: `briker_viewers/briker_viewer_mac public_test/puzzle_01_9.i DRURRDRRD`
- En MS Windows: `briker_viewers\briker_viewer_win.exe public_test\puzzle_01_9.i DRURRDRRD`

Se abrirá la ventana del visor de soluciones. Podéis controlarlo con las siguientes teclas:

- `<Return>`: Lanza la animación hasta el final.
- `<Space>`: Efectúa un paso de la animación.
- `<R> o <r>`: Reinicia la animación.
- `<+> o <->`: Acerca o aleja el tablero.
- `<Escape>`: Cierra la ventana y termina el programa.

Si lanzáis el visor sólo con el puzle, sin pasarle la solución, podréis mover el bloque con las teclas del cursor.

3. Entrega en el aula virtual

La entrega consistirá en un subir **tres** ficheros a la tarea correspondiente del aula virtual, **sólo debe subirlos uno de los miembros del grupo**. Estos son los tres ficheros:

- **`entregable3.py`**: El fichero con el código principal del entregable.

- **`brick.py`**: El fichero con la implementación de la clase `Brick`.

- **`alxxxxxx_alyyyyyy.txt`**: Un fichero de texto que deberá contener el nombre, el número de DNI y el `al######` de cada miembro del grupo (el formato del contenido es libre). Evidentemente, en el nombre del fichero tenéis que reemplazar alxxxxxx y alyyyyyy por los correspondientes a los dos miembros del grupo.

  Si el grupo es unipersonal, el fichero de texto deberá llamarse **`alxxxxxx.txt`**, reemplazando `alxxxxxx` por el que corresponda.

Vuestra entrega debe cumplir estas restricciones (cada restricción no cumplida quita un punto del entregable):

- Los nombres de los tres ficheros deben utilizar únicamente minúsculas.
- El separador utilizado en el nombre de fichero '`alxxxxxx_alyyyyyy.txt`' es el guion bajo ('_'), no utilices un menos ('-') ni ningún otro carácter similar.
- Debéis poner correctamente las extensiones de los tres archivos ('`.py`' y '`.txt`'). Si utilizáis Windows y tenéis las extensiones de archivo ocultas (que es la configuración por defecto de Windows) es posible que enviéis ficheros con doble extensión, evitadlo: configurad Windows para que muestre las extensiones de los archivos conocidos.
- No subáis ningún fichero ni directorio adicional.
- Nada de comprimir los archivos y subir un zip, rar o similar.

4. Calificación del entregable

El entregable se calificará utilizando unas pruebas privadas que se publicarán junto con las calificaciones.

Las pruebas privadas consistirán en cinco instancias similares a las de la carpeta `public_test`.

El ordenador con el que se medirán los tiempos de ejecución será `lynx.uji.es`. Un ordenador al que todos tenéis acceso y en el que podéis ejecutar el programa `entregable3_test.py` que os proporcionamos.

Para considerar superada una instancia '`puzzle_<n>_<t>.i`', vuestro método `process` no deberá tardar más de un segundo y la solución obtenida, además de resolver el puzle, deberá tener exactamente `<t>` movimientos.

Así pues, el programa puede estar mal de dos formas diferentes:

- Tener uno o más errores y funcionar mal con algunas instancias (o con todas). Esto incluye devolver una solución que no sea óptima (que utilice más de `<t>` movimientos).
- Funcionar correctamente, pero sobrepasar el tiempo máximo de un segundo al resolver la instancia.

Estos problemas pueden detectarse utilizando las pruebas públicas, aunque superar las pruebas públicas no garantiza superar las privadas, sobre todo si la implementación se ha 'ajustado' específicamente para superar las públicas.

4.1. Errores graves en el entregable

Obtendréis directamente una puntuación de cero en el entregable si modificáis el programa principal de `entregable3.py` o los tipos de cualquiera de las funciones que utiliza.

Se penalizará también con un cero si vuestro programa no lee las instancias de la entrada estándar, tal y como se indica en el apartado 1.1.

También se penalizará que la salida no respete el formato que se especifica en el apartado 1.2. Una salida errónea puede tener dos causas:

- Una implementación que no respeta el formato especificado: La penalización consistirá en quitar **dos puntos** a la nota del entregable.
- El programa tiene algún `print` olvidado en el código que se ejecuta durante las pruebas: La penalización consistirá en quitar **un punto** a la nota del entregable.

Por último, también se penaliza con un punto cada restricción incumplida en la entrega al aula virtual (ver el apartado 3).

Revisadlo con detenimiento antes de entregar (todos los miembros del grupo).

4.2. Penalización por copia

En caso de detectarse una copia entre grupos, se aplicará la normativa de la universidad: la calificación de la evaluación continua (60 % de la nota final) será de cero en la primera convocatoria para todos los miembros de los grupos involucrados.