Entregable 1
Acortar el camino a la salida del laberinto

EI1022/MT1022 - Algoritmia 2022/2023

Fecha de entrega: miércoles 19 de octubre de 2022

Contenido

1. El problema

Disponemos de un laberinto de R filas y C columnas, en el que puede existir más de un camino entre un par de celdas cualquiera. Consideraremos que en este laberinto la entrada está siempre en la celda superior izquierda, $(0, 0)$, y la salida en la celda inferior derecha, $(R - 1, C - 1)$.

Queremos averiguar qué pared debemos eliminar del laberinto para que la longitud del camino más corto entre la entrada y la salida sea mínima y que sea menor que la longitud original.

Quitar una pared equivale a añadir una arista al grafo que representa al laberinto. Así pues, la solución será una arista (u, v). Para obtener todos el mismo resultado en caso de empate entre varias aristas, deberá elegirse la menor según el orden de Python para las tuplas. Antes de comparar la arista (u, v) la reescribiremos si es necesario para que u sea menor que v, según el orden de Python.

Por último, deberemos tener en cuenta el caso de que no exista ninguna pared que permita mejorar el camino más corto existente.

Implementa un programa de línea de órdenes, `entregable1.py`, que reciba, por la entrada estándar, un laberinto en el formato que se especifica en el apartado 1.1. Como resultado de su ejecución, el programa mostrará sus resultados por la salida estándar en el formato que se detalla en el apartado 1.2.

El coste temporal asintótico del programa deberá ser lineal con el tamaño del laberinto (grafo), entendido como el número total de celdas (vértices).

PISTA: Si al grafo le añadimos la arista (u, v) estamos añadiendo un camino de $(0, 0)$ a $(R - 1, C - 1)$ de longitud $L((0,0), u) + 1 + L(v, (R - 1, C - 1))$, donde $L(a, b)$ es la longitud del camino más corto entre a y b. Así pues, tu algoritmo puede recorrer todas las posibles aristas nuevas y quedarse con la que minimice la longitud del camino añadido. Para ser eficiente tendrá que precalcular $L((0,0), u)$ y $L(v, (R - 1, C - 1))$.

1.1. Formato de la entrada

Una instancia de laberinto consistirá en cuatro líneas de texto, la primera con el número de filas del laberinto, la segunda con el número de columnas, la tercera con un entero con el número de aristas adicionales y la cuarta con un entero que se

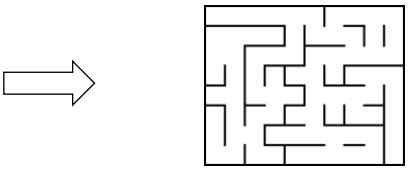
utilizará como semilla para generar el laberinto. Para que podáis obtener un laberinto a partir de estos cuatro enteros, se os proporciona la función:

```
create_labyrinth(rows: int, cols: int, n: int, s: int) -> UndirectedGraph[Vertex]
```

Esta función utiliza un MFSet para crear un laberinto, pero le añade n aristas adicionales que provocan que el laberinto (grafo) tenga ciclos. La función está implementada en el fichero `entregable1.py` que se os proporciona como punto de partida.

Veamos un ejemplo de instancia. A la izquierda se muestran las cuatro líneas del fichero de texto '`lab_008x010.i`' y, a la derecha, la representación en forma de laberinto del grafo que genera la función `create_labyrinth(8, 10, 4, 1)`:

```
8
10
4
1
```



1.2. Formato de la salida

El programa deberá mostrar siempre tres lineas de texto por la salida estándar:

- El contenido de la primera línea dependerá de si la pared existe o no. Si no existe, deberá contener el texto, en mayúsculas 'NO VALID WALL'. Si existe, deberá contener cuatro enteros separados por un espacio en blanco, indicando la pared que hay que tirar:

  ```
  fila1 columna1 fila2 columna2
  ```

  donde `(fila1, columna1)` y `(fila2, columna2)` son las coordenadas de las dos celdas entre las que está la pared que queremos eliminar. Es decir, son las coordenadas de la arista que hay que añadir al grafo. IMPORTANTE: Recuerda que `(fila1, columna1)` debe ser menor que `(fila2, columna2)`, según el operador '<' de Python.

- Una segunda línea con la longitud del camino más corto desde la entrada hasta la salida antes de eliminar la pared.

- Una tercera línea con la longitud del camino más corto desde la entrada hasta la salida después de eliminar la pared. Si no se elimina la pared, debe contener la longitud original.

Para las instancias de prueba puedes comparar tu salida con la salida correcta: para cada fichero de instancia '`.i`', hay un fichero de texto con el mismo nombre pero con la extensión '`.o`', que contiene la salida correcta.

Para las dos instancias de prueba más pequeñas, '`lab_004x006.i`' y '`lab_006x008.i`', no existe ninguna pared que mejore el camino. Para las otras ocho instancias hay al menos una pared.

2. Ficheros de partida

En el aula virtual disponéis de una carpeta con el siguiente contenido:

- `public_test.zip`: un archivo comprimido que contiene la carpeta `public_test` con un conjunto de instancias de prueba y sus soluciones.

- `entregable1.py`: el programa que debéis de utilizar como punto de partida. Incluye una función para generar un laberinto a partir de una instancia de prueba.

- `entregable1_viewer.py`: un programa que importa `entregable1.py` y lo utiliza para resolver una instancia y mostrar la solución de forma gráfica por pantalla.

- `entregable1_test.py`: programa que importa `entregable1.py` y lo utiliza para validar tanto la solución obtenida como el tiempo que la función `process` ha necesitado para obtenerla.

Los siguientes apartados explican con detalle estos contenidos.

2.1. Ficheros de prueba (public_test.zip)

Al descomprimir el fichero `public_test.zip` obtendremos la carpeta 'public_test', que contiene diez instancias de prueba y sus soluciones: los ficheros 'lab_<R>x<C>.i' y 'lab_<R>x<C>.o', respectivamente.

Las diez instancias podemos dividirlas en dos bloques según su tamaño:

- Seis pequeñas: 4x6, 6x8, 8x10, 15x20, 18x24 y 21x28.
- Cuatro grandes: 90x120, 120x160, 150x200 y 180x240.

2.2. Fichero principal (entregable1.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 importar módulos estándar de Python.
- Podéis importar módulos de la biblioteca `algoritmia`, pero sin cambiar la ruta al importarlos, por ejemplo, para importar la clase `Fifo`, debéis hacerlo así:

  ```
  from algoritmia.datastructures.queues import Fifo
  ```

  Lo que mostramos a continuación no es correcto y resta un punto al entregable:

  ```
  from misutils.algoritmia.datastructures.queues import Fifo
  ```

Veamos la estructura de `entregable1.py` y las funciones que debéis implementar.

Para empezar, y por comodidad, el programa define los tipos `Vertex` y `Edge`:

```
Vertex = tuple[int, int]
Edge   = tuple[Vertex, Vertex]
```

El programa principal utiliza la estructura de tres funciones vista en clase:

```
graph, rows, cols = read_data(sys.stdin)
edge_to_add, length_before, length_after = process(graph, rows, cols)
show_results(edge_to_add, length_before, length_after)
```

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

- `read_data(f: TextIO) -> tuple[UndirectedGraph[Vertex], int, int]`

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

  Salida: Una tupla de tres elementos, el primero es el grafo no dirigido que representa el laberinto y que se generará a partir de los datos de la instancia (con la función `create_labyrinth`); el segundo elemento de la tupla es el número de filas del laberinto y el tercero el número de columnas.

- `process(lab: UndirectedGraph[Vertex], rows: int, cols: int) -> tuple[Optional[Edge], int, int]`

  Entrada: Tres parámetros, los mismos que forman la tupla de salida de la función `read_data`.

Salida: Una tupla de tres elementos, donde:

- El primero es la arista que utilizaremos para acortar el camino (añadir la arista es lo que elimina la pared). Si no existe ninguna arista que acorte el camino deberá valer `None` (por eso el tipo es `Optional`).
- El segundo elemento es la longitud del camino más corto desde la entrada hasta la salida antes de eliminar la pared.
- El tercero elemento es la longitud del camino más corto desde la entrada hasta la salida después de eliminarla. Cuidado, si no se ha eliminado ninguna pared deberá devolver la longitud que corresponda.

- **`show_results`**`(edge_to_add: Optional[Edge], length_before: int, length_after: int)`

  Entrada: Tres parámetros, los mismos que forman la tupla de salida de la función `process`.

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

Sigue con detalle estos pasos para ejecutar entregable1.py desde la línea de órdenes:

1. Si no está instalada ya, instala la biblioteca `algoritmia`. Se ha explicado cómo hacerlo en la segunda sesión de prácticas (la información está disponibles en el aula virtual).
2. Abre un terminal y ve al directorio donde está el fichero `entregable1.py` y la carpeta `public_test` (usa `cd`).
3. Ya puedes lanzar tu programa:

   ```
   python3.10 entregable1.py < public_test/lab_008x010.i
   ```

   que deberá mostrar por pantalla:

   ```
   2 7 3 7
   22
   16
   ```

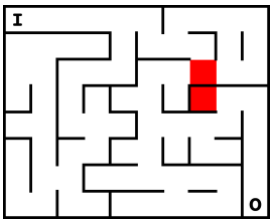
2.3. El visor de soluciones (`entregable1_viewer.py`)

Este programa importa tu fichero `entregable1.py` por lo que ambos deben estar en el mismo directorio.

Abre un terminal y ve al directorio donde están `entregable1_viewer.py`, `entregable1.py` y la carpeta `public_test`. Ejecuta la siguiente orden (en Windows utiliza la barra invertida, '\'):

```
python3.10 entregable1_viewer.py public_test/lab_008x010.i
```

Se abrirá una ventana con la siguiente imagen:



donde las dos celdas en rojo indican la pared que debería eliminarse para acortar el camino más corto desde la entrada (etiquetada con 'I') hasta la salida (etiquetada con 'O').

2.4. El validador de soluciones (`entregable1_test.py`)

Este programa importa tu fichero `entregable1.py` por lo que ambos deben estar en el mismo directorio.

Abre un terminal y ve al directorio donde están `entregable1_test.py`, `entregable1.py` y la carpeta `public_test`. Ejecuta la orden:

```
python3.10 entregable1_test.py public_test/lab_008x010.i public_test/lab_008x010.o
```

El resultado de la ejecución puede ser:

- SOLUTION OK - TIME OK (<num> sec)
- SOLUTION OK - TIME ERROR (<num> sec)
- SOLUTION ERROR - TIME OK (<num> sec)
- SOLUTION ERROR - TIME ERROR (<num> sec)
- El programa no termina en un plazo razonable y tienes que pulsar `Ctrl-C` para terminarlo.

3. Entrega en el aula virtual

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

- **`entregable1.py`**: El fichero con el código del entregable.
- **`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.

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

- Los nombres de los dos 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.
- Debes poner correctamente las extensiones de los dos archivos ('`.py`' y '`.txt`'). Si utilizas Windows y tienes las extensiones de archivo ocultas (que es la configuración por defecto de Windows) es posible que envíes ficheros con doble extensión, evítalo: configura tu Windows para que muestre las extensiones de los archivos conocidos.
- No subas 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 diez instancias de los mismos tamaños que los de las pruebas públicas:

- Seis pequeñas: 4x6, 6x8, 8x10, 15x20, 18x24 y 21x28.
- Cuatro grandes: 90x120, 120x160, 150x200 y 180x240.

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 `entregable1_test.py` que os proporcionamos.

Para considerar una instancia superada, vuestro método `process` deberá tardar **menos de un segundo** en terminar. Con los laberintos pequeños es posible conseguirlo incluso con un algoritmo cuadrático, pero con los grandes sólo es posible conseguirlo con un algoritmo lineal (con el número de celdas). Por lo tanto, con un algoritmo cuadrático, la nota máxima del entregable será de 6 sobre 10.

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).
- No tener errores, pero tardar más de un segundo en obtener la solución.

Ambos 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 `entregable1.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:

- 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.

Po 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.