Entregable 4
Rellenando valles

EI(MT)1022 - Algoritmia 2022/2023

Fecha de entrega: 14 de diciembre de 2022

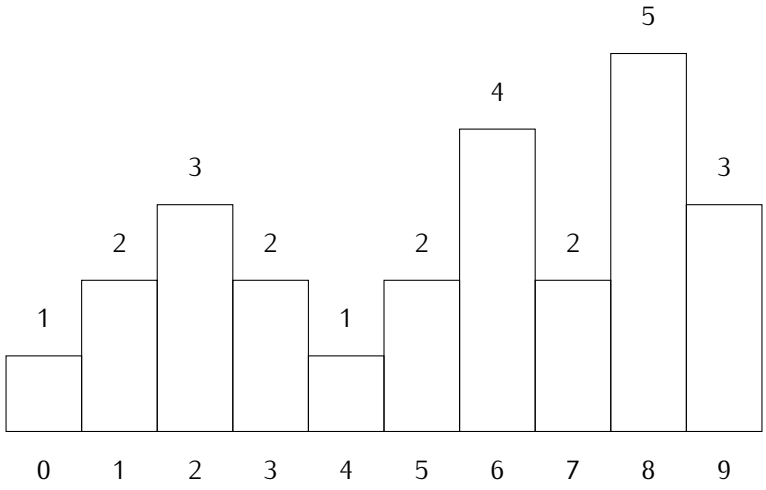
Contenido

1. Introducción

Sea v un vector, decimos que el par (l, r) es un *valle* en v si todos los elementos entre las posiciones l y r, excluidas, son menores que el menor de $v[l]$ y $v[r]$, al que llamaremos *altura del valle*. Es decir

$$(l, r) \text{ es un valle } \iff \forall\, l < i < r : v[i] < \min(v[l], v[r]). \tag{1}$$

Por ejemplo, en el vector `[1, 2, 3, 2, 1, 2, 4, 2, 5, 3]`, el par $(2, 6)$ es un valle como puedes ver en la figura siguiente:



También son valles los pares $(3, 5)$ y $(6, 8)$. Sin embargo, el par $(2, 5)$ no es un valle porque $v[3]$ no es menor que $v[5]$ y por lo tanto no se cumple (1). Tampoco es valle el par $(2, 8)$ porque $v[6] > v[2]$.

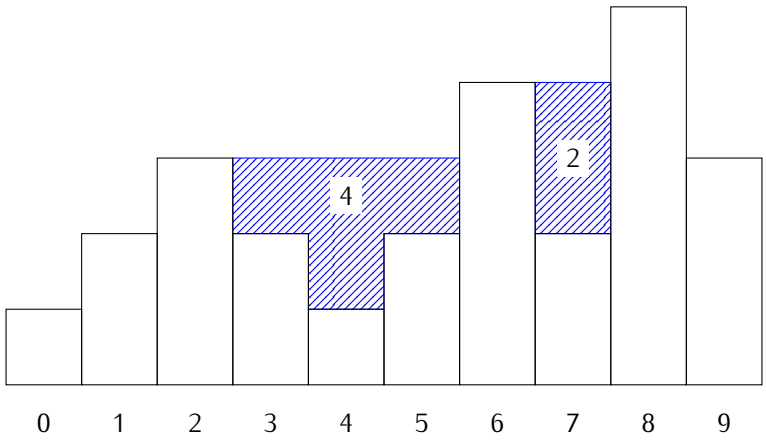
Rellenar un valle (l, r) es hacer que todos los elementos entre ambas posiciones tengan el mismo valor que la altura del valle. El coste de rellenar un valle es igual a la suma de lo que hay que añadir en cada posición, es decir el coste de rellenar el valle (l, r) es

$$c_v(l, r) = \sum_{i=l+1}^{r-1} \big(\text{mín}(v[l], v[r]) - v[i]\big). \tag{2}$$

que se puede calcular más fácilmente si definimos $h = \text{mín}(v[l], v[r])$ y $d = r - l - 1$ con lo que queda

$$c_v(l, r) = dh - \sum_{i=l+1}^{r-1} v[i]. \tag{3}$$

Por ejemplo, en el caso del vector anterior el coste de rellenar el valle $(2, 6)$ es $3 \times 3 - (2 + 1 + 2) = 4$ y el de rellenar el valle $(6, 8)$ es $1 \times 4 - 2 = 2$, como puedes ver en esta figura:



Consideraremos que $(l, l + 1)$ es un valle vacío con un coste de rellenado igual a cero.

2. Objetivo

Debéis escribir un problema que, dado un vector encuentre el valle con el máximo coste de rellenado. En caso de que haya más de un valle con el máximo coste, el programa devolverá aquel situado más a la izquierda, esto es, aquel cuya primera componente sea menor. Si todos los valles son vacíos (por ejemplo, si la entrada es el vector `[1, 2, 3]`), el valle devuelto será el $(0, 1)$.

La entrada del programa será un fichero que se leerá por la entrada estándar y contendrá una línea por cada elemento del vector. Todos los elementos serán enteros y puedes asumir que habrá al menos dos elementos.

La salida del programa consistirá en una única línea con tres números enteros: si el valle encontrado es (l, r) y tiene un coste de rellenado c, la salida tendrá los números l, r y c en este orden separados por espacios.

El programa empleará el esquema de *divide y vencerás* y deberá tener un coste temporal $\mathcal{O}(n \log n)$.

3. Ficheros de partida

En el aula virtual tenéis el fichero `entregable4_valle.zip` que contiene lo siguiente:

- `entregable4.py` el fichero base para vuestra implementación.
- `entregable4_test.py` un programa para comprobar vuestra implementación.
- `public_tests` un directorio con las instancias de prueba.

El fichero `entregable4.py` es el que tenéis que modificar y entregar. Al modificarlo, debéis tener en cuenta las siguientes condiciones:

- 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).

En `entregable4.py` se define el tipo `Solution` para representar las soluciones. Es una tupla con tres componentes de tipo entero: los dos extremos del vector y el coste de rellenar ese valle. La declaración es:

```
Solution = tuple[int, int, int]  # l, r, coste
```

A continuación, se definen las tres funciones que debéis escribir. Seguimos la estructura vista en clase por lo que el cuerpo del programa es:

```
if __name__ == "__main__":
    v = read_data(sys.stdin)
    sol = process(v)
    show_results(sol)
```

En este caso, lo que hace cada función es lo siguiente. La función `read_data`, con la cabecera

```
def read_data(f: TextIO) -> list[int]:
```

lee del fichero `f` una lista de enteros que representa el vector v. El formato es el especificado en la sección 2.

La función `process`, con la cabecera

```
def process(v: list[int]) -> Solution:
```

recibe la lista leída por `process` y devuelve la solución correspondiente, esto es la tupla $\big(l, r, c_v(l, r)\big)$ tal que (l, r) es el valle que maximiza $c_v(l, r)$ y está más a la izquierda.

Finalmente, `show_results`, con la cabecera

```
def show_results(sol: Solution):
```

muestra por la salida estándar la solución en el formato indicado en la sección 2.

4. Prueba de la implementación

Para probar vuestra implementación podéis utilizar el programa `entregable4_test.py` junto con las instancias de prueba contenidas en el directorio `public_tests`. La forma de ejecutarlo es análoga a los de entregables anteriores. Si queréis probar, por ejemplo, la instancia `public_tests/public_5_0_2_4.vec.vec`, tenéis que escribir en la línea de órdenes:

```
python3.10 entregable4_test.py public_tests/public_5_0_2_4.vec.vec
```

Si queréis probar todas las instancias del directorio `public_tests`, escribid:

```
python3.10 entregable4_test.py public_tests/*.vec
```

El programa hace unas comprobaciones preliminares de los métodos `read_data` y `show_results` y después utiliza `process` para resolver cada una de las instancias. Tras llamar a `process` compara la solución obtenida con la esperada, que está codificada en el nombre del fichero[1]. Si todo ha ido bien, se muestra un mensaje como el siguiente:

```
------------------------------------------------------------
INSTANCIA: public_tests/public_5_0_2_4.vec.vec
RESULT: OK
  USEDTIME: OK - 0.00 sec (<= 1 sec)
------------------------------------------------------------
```

Si el resultado es correcto pero el tiempo total supera un segundo, se muestra un mensaje como este:

```
------------------------------------------------------------
INSTANCIA: public_tests/public_5_0_2_4.vec.vec
RESULT: ERROR_TIMEOUT
  USEDTIME: ERROR - 3.01 sec (> 1 sec)
------------------------------------------------------------
```

Si el resultado es erróneo, se muestra un mensaje como:

```
------------------------------------------------------------
INSTANCIA: public_tests/public_5_0_2_4.vec.vec
RESULT: ERROR_INVALID_SOLUTION
  Tu función 'process()' ha devuelto [0, 4, 5] en lugar de [0, 4, 4].
------------------------------------------------------------
```

Si se produce alguna excepción, el programa se detiene y la muestra junto a la traza de pila.

Si `process` entra en un bucle infinito, tendrás que interrumpir manualmente la ejecución.

Los ficheros de prueba se dividen en dos grupos. Los de depuración, que tiene el prefijo `debug` en el nombre y que debéis utilizar para probar que vuestra implementación es correcta. Son el mínimo aceptable, no admitiremos ninguna entrega en la que falle alguno de estos ficheros. El segundo grupo, los ficheros con prefijo `public` en el nombre, son aquellos destinados a evaluar el tiempo de ejecución de la implementación (y, secundariamente, la presencia de posibles *bugs* más sutiles). Se admitirán entregas aunque fallen en alguno de estos ficheros, aunque eso indicará que posiblemente habrá fallos en las pruebas privadas.

5. Entrega en el aula virtual

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

- `entregable4.py`: El fichero con el código del entregable.
- `alxxxxxx_alyyyyyy.txt` o `alxxxxxx.txt`: El fichero para identificar el grupo. Debe ser un fichero de texto con el nombre, número de DNI y dirección de correo de cada miembro del grupo. El nombre del fichero será el nombre de usuario de cada miembro del grupo (el *al*), separados por guión bajo.

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

- Los nombres de los ficheros deben utilizar únicamente minúsculas.
- El separador utilizado en fichero de identificación del grupo es el guión bajo (_), no un menos (-) ni ningún otro carácter similar.

[1] Los cuatro números del nombre del fichero son el número de elementos de la instancia, el valor de l, el de r y el de $c_v(l, r)$.

- Debéis poner correctamente las extensiones de los tres ficheros (`.py` y `.txt`). Si utilizáis Windows y tenéis las extensiones de fichero 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 ficheros conocidos.
- No subáis ningún fichero ni directorio adicional.
- No comprimáis los ficheros.

6. Calificación

El entregable se calificará utilizando un conjunto de veinte pruebas privadas que se publicarán junto con las calificaciones. Estas pruebas tendrán las mismas tallas que las que tienen prefijo `public` en la carpeta `public_tests`. La nota sobre la que se podrán aplicar las penalizaciones que se comentan en las secciones siguientes será simplemente el número de pruebas superadas multiplicado por 0,5.

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 `entregable4_test.py` que os proporcionamos.

Para considerar superada una instancia la solución debe ser correcta y obtenerse en menos de un segundo. Así pues, el programa puede estar mal de dos formas diferentes:

- Teniendo uno o más errores y funcionar mal con algunas instancias (o con todas).
- Devolviendo la solución correcta, pero sobrepasando 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.

6.1. Errores graves

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 la sección 2.

Finalmente, se calificará con cero cualquier programa que no supere las pruebas con prefijo `debug` de la carpeta `public_tests`.

6.2. Errores leves

Se penalizará que la salida no respete el formato que se especifica en la sección 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 la sección 5).

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

6.3. Penalización por copia

A las copias detectadas en la entrega, se les 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.