Entregable 2
Predicción de ocupación

EI(MT)1022 - Algoritmia 2023/2024

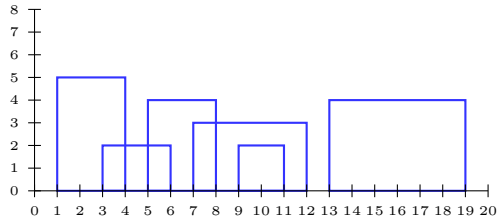
Contenido

1. Introducción

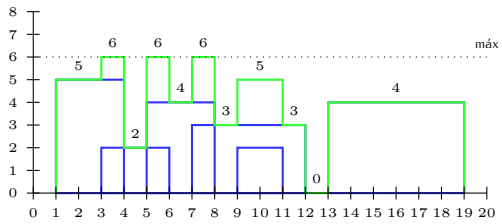
Supongamos que un hotel tiene una predicción de las reservas que van a producirse en los próximos días. El hotel tiene un número limitado de habitaciones y quiere simular su ocupación durante esos días. Vuestro objetivo es escribir un programa que, a partir de una lista de reservas y un número total de habitaciones, genere un listado de ocupaciones de habitaciones.

Para simplificar el problema, asumiremos que hay una lista de reservas que determina para cada una de ellas tres valores: el día de inicio, el número de habitaciones reservadas y la duración.

Por ejemplo, supongamos un (pequeño) hotel de seis habitaciones con la siguiente lista de reservas: `[(1, 5, 3), (3, 2, 3), (5, 4, 3), (7, 3, 5), (9, 2, 2), (13, 4, 6)]`. Podemos representar la lista así:



Podemos calcular el número de habitaciones necesario cada día como la suma de las de las habitaciones que estarán reservadas en ese día limitadas por el número de habitaciones del hotel:



La previsión de ocupación es una lista de enteros $[d_0, h_0, d_1, h_1, \ldots, d_{n-1}, h_{n-1}, d_n]$ en la que d_0 es el primer día en el que hay una reserva, h_0 es el número de habitaciones que se reservaran desde el día d_0 hasta el día d_1 (excluido) y así sucesivamente. Esto quiere decir que para cada $0 \leq i < n$ el valor de h_i es el número de habitaciones reservadas desde el día d_i hasta el día d_{i+1}, limitado por el número de habitaciones del hotel. Por lo tanto, desde el día d_n no hay previstas reservas. Para minimizar la longitud de la representación, debe cumplirse que $h_{i-1} \neq h_i$ para $0 < i < n$.

En nuestro ejemplo, la previsión resultante es: `[1, 5, 3, 6, 4, 2, 5, 6, 6, 4, 7, 6, 8, 3, 9, 5, 11, 3, 12, 0, 13, 4, 19]`.

Observa que el último 19 es el día en que termina la última reserva.

2. Objetivo

Debéis escribir un programa que, dado un fichero de texto que contiene el número de habitaciones del hotel y la lista de reservas, muestre la correspondiente previsión de ocupación. El fichero tendrá una primera línea con el número de habitaciones y, a continuación, una linea por cada reserva que estará formada por tres enteros separados por espacio: día inicial, número de habitaciones y duración. De este modo, el fichero correspondiente al ejemplo de la sección anterior es:

```
6
1 5 3
3 2 3
5 4 3
7 3 5
9 2 2
13 4 6
```

Para mostrar la previsión de ocupaciones, el programa escribirá una línea de texto formada por enteros separados por blancos. En nuestro ejemplo:

```
1 5 3 6 4 2 5 6 6 4 7 6 8 3 9 5 11 3 12 0 13 4 19
```

El programa deberá tener un coste temporal $\mathcal{O}(n \log n)$ y seguir el esquema "divide y vencerás". Divide la lista de reservas en dos partes, calcula recursivamente una previsión para cada una y combínalas.

3. Ficheros de partida

En el aula virtual tenéis una carpeta con los siguientes ficheros:

- `public_test.zip`: un archivo que contiene la carpeta `public_test` con las instancias de prueba y sus soluciones.
- `e2.py`: el programa que debéis completar y entregar.
- `e2_test.py`: un programa para probar el funcionamiento de `e2.py`.

3.1. Ficheros de prueba

En el fichero `public_test.zip` están las instancias de prueba (ficheros con extensión `.i`) y sus soluciones (ficheros con extensión `.o`).

3.2. Fichero principal

El fichero `e2.py` contiene la estructura del programa. Debéis modificarlo teniendo en cuenta lo siguiente:

- Podéis añadir funciones o clases adicionales, pero no debéis modificar nada del programa principal. Tampoco podéis modificar la firma (el tipo) de las tres funciones que se utilizan en el programa principal. Hacer cualquiera de esas modificaciones supondrá un cero en la calificación del entregable.
- Podéis importar módulos de la biblioteca estándar de Python, pero la única biblioteca externa permitida es algoritmia. Por ejemplo, NumPy es una biblioteca externa y, por lo tanto, no está permitida
- La utilización de `dac_scheme` es opcional: podéis implementar vuestra solución directamente o utilizar la función `div_solve` y la clase `IDivideAndConquerProblem`.

Al comienzo del fichero se definen los tipos `Reservation`, `Problem` y `Forecast`:

```
Reservation = tuple[int, int, int]
Problem = tuple[int, list[Reservation]]
Forecast = list[int]
```

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

```
if __name__ == "__main__":
    problem = read_data(sys.stdin)
    forecast = process(problem)
    show_results(forecast)
```

La función `read_data` tiene la firma

```
def read_data(f: TextIO) -> Problem:
```

Debéis implementar el cuerpo de modo que acepte ficheros con el formato explicado en la sección 2.
La función `process` tiene la firma

```
def process(problem: Problem) -> Forecast:
```

Debéis implementar el cuerpo de modo que genere la previsión correspondiente a la lista de reservas.

3.3. Validador de soluciones

El programa `e2_test.py` os permite probar vuestra implementación. Debéis ejecutarlo en una terminal dentro del directorio donde tengáis el programa `e2.py` ya que lo importa para probarlo.

Se asume que los ficheros de entrada tienen la extensión `.i` y que por cada uno de ellos hay un fichero con extensión `.o` con la solución correspondiente. Así, basta con especificar los ficheros de entrada que se quieren probar. Esto se puede hacer pasando tantos argumentos como ficheros se quieran probar. Por ejemplo, para probar los ficheros `public_test/problem_20_1.i` y `public_test/problem_20_2.i` se usaría la orden

```
python -O e2_test.py public_test/problem_20_1.i public_test/problem_20_2.i
```

Adicionalmente, se puede usar la opción `-d` para probar todos los ficheros `.i` de un directorio. Así, para pasar todas las pruebas del directorio `public_test` se usaría la orden

```
python -O e2_test.py -d public_test
```

Si todo va bien, el programa responde con

```
Todas las pruebas de process() han sido correctas
```

Si se especifica la opción `-v`, se puede ver información sobre cada prueba. Así, con

```
python -O e2_test.py -v public_test/problem_20_1.i public_test/problem_20_2.i
```

se obtiene

```
Probando: public_test/problem_20_1.i --> OK, tiempo: 0.0000
Probando: public_test/problem_20_2.i --> OK, tiempo: 0.0000
Todas las pruebas de process() han sido correctas
```

Observad que `-v` se pasa a `e2_test.py`, no a python.

Cuando hay una prueba que genera un *timeout*, el proceso se detiene. Durante el desarrollo podéis incrementar el valor del *timeout* con la opción `-t`.

4. Entrega

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

- `e2.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 lo que corresponda.

Vuestra 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 utilicéis un guión bajo ("_") ni ningún otro carácter similar.
- Las extensiones de los ficheros deben ser las que corresponden (`.py` y `.txt`). Si utilizáis Windows es posible que tengáis las extensiones de archivo ocultas (es la configuración por defecto de Windows) por lo que es posible que enviéis ficheros con doble extensión, evitadlo: configurad vuestro Windows para que muestre las extensiones de los archivos.
- No subáis ningún fichero ni directorio adicional.

5. Calificación

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

Las pruebas privadas consistirán en un grupo de instancias de tamaño similar a los de las pruebas públicas.

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 `e2_test.py` que os proporcionamos.

Para considerar una instancia superada, vuestro método `process` deberá tardar menos de un segundo en terminar. Con problemas pequeños es posible conseguirlo incuso si vuestro algoritmo es ineficiente. Para instancias más grandes, debéis tener un coste $\mathcal{O}(n \log n)$, donde n es el número de tareas.

Vuestro programa puede tener los siguientes problemas:

- Tiene uno o más errores y funciona mal con algunas instancias (o con todas).
- No tiene errores, pero tarda más de un segundo en obtener la solución de alguna instancia.

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.

5.1. Errores graves en el entregable

Obtendréis directamente una puntuación de cero en el entregable si modificáis el programa principal de e1.py o los tipos de cualquiera de las funciones que utiliza.

En caso de que no hayáis seguido el esquema "divide y vencerás", la nota se multiplicará por $0{,}4$.

Se os penalizará también con un cero si vuestro programa no lee las instancias desde la entrada estándar en el formato que se especifica en la sección 2, también se penalizará que la salida no respete el formato que se especifica en esa sección. 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.
- Algún `print` olvidado en el código y 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 4).

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

5.2. 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.