EI1022, Algoritmia

Problemas 2 y 3: Divide y vencerás

Método maestro

Subproblemas del mismo tamaño

	$T(n) = aT(n/b) + \mathcal{O}(n^k)$	$T(n) = aT(n/b) + \mathcal{O}(n^k \log^p n)$
$a < b^k$	$T(n) \in \mathcal{O}(n^k)$	$T(n) \in \mathcal{O}(n^k \log^p n)$
$a = b^k$	$T(n) \in \mathcal{O}(n^k \log n)$	$T(n) \in \mathcal{O}(n^k \log^{p+1} n)$
$a > b^k$	$T(n) \in \mathcal{O}(n^{\log_b a})$	$T(n) \in \mathcal{O}(n^{\log_b a})$

Subproblemas de distinto tamaño

	$T(n) = \sum_{1 \leq i \leq m} T(\alpha_i n) + \mathcal{O}(n)$
$\sum_{1 \leq i \leq m} \alpha_i < 1$	$T(n) \in \mathcal{O}(n)$

Ejercicio

Utiliza el método maestro para acotar asintóticamente las funciones $T(n)$ definidas por las siguientes ecuaciones:

a) $T(n) = \begin{cases} 1, & \text{si } n \leq 1; \\ 9T(n/3) + n, & \text{si } n > 1. \end{cases}$

b) $T(n) = \begin{cases} 1, & \text{si } n \leq 1; \\ 3T(n/4) + \log n, & \text{si } n > 1. \end{cases}$

c) $T(n) = \begin{cases} 1, & \text{si } n \leq 1; \\ 5T(n/4) + n, & \text{si } n > 1. \end{cases}$

d) $T(n) = \begin{cases} 1, & \text{si } n \leq 1; \\ T(n/2) + T(n/4) + n, & \text{si } n > 1. \end{cases}$

e) $T(n) = \begin{cases} 1, & \text{si } n \leq 1; \\ T(n/2) + n, & \text{si } n > 1. \end{cases}$

f) $T(n) = \begin{cases} 1, & \text{si } n \leq 1; \\ 4T(n/2) + n^2, & \text{si } n > 1. \end{cases}$

g) $T(n) = \begin{cases} 1, & \text{si } n \leq 1; \\ 4T(n/2) + n^3, & \text{si } n > 1. \end{cases}$

h) $T(n) = \begin{cases} 1, & \text{si } n \leq 1; \\ T(n/2) + 1, & \text{si } n > 1. \end{cases}$

i) $T(n) = \begin{cases} 1, & \text{si } n \leq 1; \\ 2T(n/2) + 1, & \text{si } n > 1. \end{cases}$

j) $T(n) = \begin{cases} 1, & \text{si } n \leq 1; \\ T(2n/3) + 1, & \text{si } n > 1. \end{cases}$

k) $T(n) = \begin{cases} 1, & \text{si } n \leq 1; \\ 2T(n/2) + n \log n, & \text{si } n > 1. \end{cases}$

l) $T(n) = \begin{cases} 1, & \text{si } n \leq 1; \\ 2T(n/2) + n, & \text{si } n > 1. \end{cases}$

Punto fijo de un vector (RyV)

- Si tenemos un vector v de enteros, decimos que i es un *punto fijo* de v si $v[i] = i$
- Si v está ordenado de menor a mayor, podemos usar "reduce y vencerás" para encontrar un punto fijo con un coste $\mathcal{O}(\log n)$
- Implementa el programa `punto_fijo.py` que encuentra el punto fijo de un vector de enteros

Implementación

- Los tipos serán:

```
type Data = list[int]
type Result = int | None
```

- La función `read_data` leerá un entero por línea y los devolverá en una lista
- La función `process` buscará el punto fijo. Si lo encuentra, devolverá el índice del punto fijo, en caso contrario, devolverá `None`
- La función `show_result` escribirá el índice del punto fijo o la cadena `"No hay punto fijo"`

Pistas

- ¿Cuáles de los siguientes vectores serían entradas válidas para `punto_fijo.py`?
  - `[2, 3, 3, 4, 5, 10]`
  - `[100, 101, 102, 103]`
  - `[1, -10, 5, 6]`
  - `[2, 3, 4, 5, 6, 7]`
  - `[-6, -5, -4, -3]`

Pistas

- ¿Cuáles de los siguientes vectores serían entradas válidas para `punto_fijo.py`?
  - `[2, 3, 3, 4, 5, 10]`
  - `[100, 101, 102, 103]`
  - `[1, -10, 5, 6]`
  - `[2, 3, 4, 5, 6, 7]`
  - `[-6, -5, -4, -3]`
- ¿Cómo puedes averiguar, mirando solo una posición, que los vectores válidos del punto anterior no tienen solución?

Pistas

- ¿Cuáles de los siguientes vectores serían entradas válidas para `punto_fijo.py`?
  - `[2, 3, 3, 4, 5, 10]`
  - `[100, 101, 102, 103]`
  - `[1, -10, 5, 6]`
  - `[2, 3, 4, 5, 6, 7]`
  - `[-6, -5, -4, -3]`
- ¿Cómo puedes averiguar, mirando solo una posición, que los vectores válidos del punto anterior no tienen solución?
- ¿Puedes descartar una parte del vector `[-10, -5, 1, 3, 6]` mirando solo la posición central?

Implementaciones

- Prepara tres implementaciones:
  - Utilizando el esquema `IReduceAndConquerProblem`
  - Recursiva sin utilizar el esquema
  - Iterativa
- Para no crear un fichero nuevo cada vez, puedes escribir tres funciones: `process_schema`, `process_rec` y `process_iter` y cambiar el programa principal:

```
if __name__ == "__main__":
    process = process_schema
    # process = process_rec
    # process = process_iter
    data0 = read_data(sys.stdin)
    result0 = process(data0)
    show_result(result0)
```

Pruebas

- En los ficheros auxiliares del aula virtual tienes el directorio `vectores_pf` con algunas pruebas para el problema de encontrar el punto fijo
- Los ficheros con extensión `vec` contienen vectores
- Los ficheros con extensión `sol` contienen las soluciones correspondientes

Pico de un vector (RyV)

- Un *pico* de un vector v es una posición i tal que:
  - Si $0 < i < |v| - 1$ entonces $v[i-1] \leq v[i]$ y $v[i] \geq v[i+1]$
  - Si $i = 0$ entonces $|v| = 1$ o $v[0] \geq v[1]$
  - Si $i = |v| - 1$ entonces $v[i-1] \leq v[i]$
- Ejemplo: el vector `[10, 20, 15, 2, 23, 90, 67]` tiene dos picos: 1 y 5
- Implementa el programa `pico.py` para encontrar un pico de un vector

Implementación

- Los tipos serán:

```
type Data = list[int]
type Result = int
```

- La función `read_data` será la misma que en `punto_fijo.py`
- La función `process` devolverá un pico cualquiera del vector
- La función `show_result` escribirá el valor del pico
- Usa una estrategia de "reduce y vencerás" para obtener un coste de $\mathcal{O}(\log n)$
- Implementa las versiones recursiva e iterativa

Pruebas

- En los ficheros auxiliares del aula virtual tienes el directorio `vectores_pico` con algunas pruebas para el problema de encontrar un pico
- Los ficheros con extensión `vec` contienen vectores
- El programa `es_pico.py` te permite comprobar tu solución
- Por ejemplo, uno de los picos de `v3.vec` es 990. La orden:

```
python3 es_pico.py 990 < v3.vec
```

escribe `OK`

Subvector de suma máxima (DyV)

- Dado un vector v, con números enteros (positivos y negativos), un subvector de suma máxima viene definido por dos valores b y e tales que

$$\forall b', e' : \sum_{b \leq i < e} v[i] \geq \sum_{b' \leq i < e'} v[i]$$

- Implementa el programa `suma_maxima.py` que recibe un vector de enteros y devuelve la suma máxima de un subvector y su posición

Implementación

- Los tipos serán:

```
type Data = list[int]
type Result = tuple[int, int, int]
```

- La función `read_data` vuelve a ser la de `punto_fijo.py`
- La función `process` usa una estrategia de "divide y vencerás" para encontrar el subvector de suma máxima
- La función `show_result` escribe por pantalla tres valores, uno por línea: la suma del subvector y los valores de b y e

Pruebas

- En los ficheros auxiliares del aula virtual tienes el directorio `vectores_sm` con algunas pruebas para el problema de encontrar el subvector de suma máxima
- Los ficheros con extensión `vec` contienen vectores
- Los ficheros con extensión `sol` contienen las soluciones correspondientes
- Ten en cuenta que puede haber más de un subvector con la misma suma máxima, por eso, debes comprobar la suma además de la posición