Práctica 1: Complejidad, contratos e invariantes

Ejercitación básica

Ejercicio 1. Dar algoritmos para resolver los siguientes problemas y determinar su complejidad temporal y espacial en peor caso:

a) Hallar el máximo de un arreglo de n enteros.

b) Hallar los k números más grandes de un arreglo de n enteros.

c) Dados dos arreglos de enteros A y B, determinar cuál es más largo.

d) Dados dos arreglos de enteros A y B, determinar cuántos elementos de A aparecen también en B.

e) Dado un arreglo A de enteros, producir un arreglo B de enteros tal que $B[i]$ contenga el número $A[i]^2$.

f) Dado un arreglo A de enteros, producir un arreglo B de enteros que resulte de borrar todos los números negativos de A.

g) Dado un arreglo A de enteros cuyos valores están entre 0 y $k-1$, producir un arreglo B de tamaño k tal que $B[i]$ contenga el número de veces que aparece el entero i en A.

h) Dada un cadena de caracteres S de longitud n, determinar si S es un palíndromo, es decir, si la inversa de S es igual a S. Por ejemplo, la cadena "`abcbbcba`" es un palíndromo.

i) Dada un cadena de caracteres S de longitud n y una cadena de caracteres T de longitud m, determinar si T aparece como *subcadena* de S. Por ejemplo, la cadena "`abc`" aparece como subcadena de "`ababc`" pero no de "`abacb`".

j) Dada un cadena de caracteres S de longitud n, determinar cuántas de sus subcadenas son palíndromos. Por ejemplo, la cadena "`abcb`" tiene 5 subcadenas palíndromas: "`a`", "`b`", "`c`", "`b`" y "`bcb`".

Ejercicio 2. Usamos las letras f, g, h para denotar funciones $f, g, h : \mathbb{N} \to \mathbb{R}^+$.

a) Demostrar que si $f \in O(g)$ y $g \in O(h)$, entonces $f \in O(h)$.

b) Sean $b, c > 1$ dos números reales. Demostrar que $O(\log_b n) = O(\log_c n)$. Recordar que $\log_b n = \frac{\log_c n}{\log_c b}$.

c) Sea $n! = 1 \cdot 2 \cdot 3 \cdots n$ el factorial de n. Demostrar que $2^n \in O(n!)$.

d) Demostrar que $n \leq 2^n$ para todo $n \in \mathbb{N}$, y concluir que $n \in O(2^n)$.

e) Notamos $\lfloor x \rfloor$ a la parte entera de $x \in \mathbb{R}$. Sean $n, p \in \mathbb{N}$. Como resultado del ítem anterior, observemos que $\frac{n}{p} \leq \lfloor \frac{n}{p} \rfloor + 1 \leq 2^{\lfloor \frac{n}{p} \rfloor + 1} \leq 2^{\frac{n}{p}+1}$. Usando este hecho, demostrar que $n^p \in O(2^n)$.

f) Demostrar que $f(n) + g(n) = O(\max\{f(n), g(n)\})$ y que $\max\{f(n), g(n)\} \in O(f(n) + g(n))$.

Ejercicio 3. Diseñar un tipo de datos `VectorSumable` con la siguiente interfaz, respetando las complejidades temporales en peor caso indicadas:

- `inicializar`(A) — crea una estructura que representa el arreglo A recibido. Precondición: se asume que los elementos de A son números enteros. Complejidad: $O(n)$, asumiendo que $|A| = n$.
- `valor`(i) — devuelve el elemento en la posición i del arreglo A. Precondición: se asume que $0 \leq i \leq n$. Complejidad: $O(1)$.
- `suma`(i, j) — devuelve la suma de los elementos del arreglo A en el intervalo $[i, j)$.[1] Es decir, devuelve $A[i] + A[i+1] + \ldots + A[j-1]$. Precondición: se asume que $0 \leq i \leq j \leq n$. Complejidad: $O(1)$.

¿Cuál es el invariante de la estructura de datos?

[1] Usamos $[i, j)$ para referirnos al intervalo desde i inclusive hasta j exclusive.

Ejercitación adicional

Ejercicio 4. Diseñar un tipo de datos `VectorConUndo` con la siguiente interfaz, respetando las complejidades temporales en peor caso indicadas:

- `inicializar`(n) — crea un vector de tamaño n, cuyos elementos son todos 0.
  Precondición: se asume que $n \geq 0$. Complejidad: $O(n)$.
- `leer`(i) — devuelve el valor en la posición i.
  Precondición: se asume que $0 \leq i < n$. Complejidad: $O(1)$.
- `escribir`(i, x) — sobreescribe el elemento en la posición i con el valor x.
  Precondición: se asume que $0 \leq i < n$. Complejidad: $O(1)$.
- `ctrlZ`() — deshace la última operación de escritura. Si no hubo escrituras, esta operación no tiene efecto.
  Complejidad: $O(1)$.

¿Cuál es el invariante de la estructura de datos?

Ejercicio 5. Sea A un arreglo de tamaño n que contiene los elementos $0, 1, \ldots, n-1$ sin repeticiones. Recordemos que los índices de los elementos de A van de 0 a $n-1$.

a) Diseñar un algoritmo que reciba A como entrada y construya un arreglo B de tamaño n, de tal modo que $B[i]$ contenga la posición del elemento i en A. Por ejemplo, si $A = [4, 0, 1, 2, 3]$ entonces $B = [1, 2, 3, 4, 0]$, ya que (por ejemplo) el valor 0 aparece en la posición 1 de A.

b) Determinar la complejidad temporal en peor caso del algoritmo diseñado.

c) Si el algoritmo no es $O(n)$ —es decir, *lineal*— modificarlo para que sí lo sea.

¿Cuál es el invariante que cumplen los ciclos del algoritmo?

Ejercicio 6. Sea M una matriz de $n \times n$ entradas, cada una de las cuales es un número entero entre 0 y 10. Diseñar un algoritmo para determinar si hay dos filas que suman lo mismo en tiempo $O(n^2)$ en peor caso.

Ejercicio 7. Sean A y B arreglos de enteros ordenados crecientemente, sin repetidos.

- La *unión* de A y B es un arreglo $A \cup B$ ordenado crecientemente que contiene a un entero x si y sólo si x aparece en A **o** x aparece en B. Por ejemplo, $[1, 3, 5, 7] \cup [1, 2, 3, 4, 5] = [1, 2, 3, 4, 5, 7]$.
- La *intersección* de A y B es un arreglo $A \cap B$ que contiene a un entero x si y sólo si x aparece en A **y** x aparece en B. Por ejemplo, $[1, 3, 5, 7] \cap [1, 2, 3, 4, 5] = [1, 3, 5]$.

Asumiendo que los tamaños de los arreglos son $|A| = n$ y $|B| = m$:

a) Diseñar un algoritmo para calcular $A \cup B$ en tiempo $O(\max\{n, m\})$ en peor caso.

b) Diseñar un algoritmo para calcular $A \cap B$ en tiempo $O(\max\{n, m\})$ en peor caso.

Ejercicio 8. Sea A un arreglo de n números enteros. Decimos que el elemento en la posición i está *fuera de lugar* si el arreglo A no está ordenado pero queda ordenado si se elimina el elemento en la posición i. Por ejemplo, el arreglo $[1, 3, 5, 6, 8, 4, 9]$ no está ordenado. El 4 se encuentra fuera de lugar, porque si se lo borra resulta el arreglo $[1, 3, 5, 6, 8, 9]$, que sí está ordenado. Diseñar un algoritmo que recibe un arreglo A con un elemento fuera de lugar y lo ordena en tiempo lineal. ¿Cuáles son los invariantes?

Ejercicio 9. Sea A un arreglo de tamaño n que contiene los elementos $0, 1, \ldots, n-1$ sin repeticiones. Notamos $A^k[i]$ al resultado de hacer $\underbrace{A[A[\ldots A[}_{k \text{ veces}} i]]]$. Por ejemplo, en el arreglo $[5, 3, 2, 4, 1, 0]$ tenemos:

$$A^1[1] = A[1] = 3 \qquad A^2[1] = A[A[1]] = A[3] = 4 \qquad A^3[1] = A[A[A[1]]] = A[4] = 1$$

en tal caso decimos que tenemos un *ciclo* $1 \to 3 \to 4 \to 1$ de largo 3. En general un ciclo de largo n está dado por un número i tal que $i \to A^1[i] \to A^2[i] \to \ldots \to A^n[i]$ siempre y cuando el último elemento sea i (es decir, $A^n[i] = i$)

pero i no aparezca antes en la secuencia (es decir, $A^k[i] \neq i$ cuando k es tal que $1 \leq k < n$). Diseñar un algoritmo para calcular el tamaño del ciclo más largo en tiempo lineal.

Ejercicio 10. Sea A un arreglo de n números enteros, y sean p y q dos números enteros tales que $p \leq q$. Se quiere reordenar el arreglo A en tres partes, de tal modo que aparezcan primero todos los números menores que p, a continuación los números en el intervalo $[p, q]$ y a continuación los números mayores que q. Por ejemplo, si $A = [12, 2, -1, 1, -2, 5, 3, 11, 13]$, $p = 0$ y $q = 10$, la salida podría ser:

$$[\underbrace{-1, -2}_{<p}, \underbrace{2, 1, 5, 3}_{[p,q]}, \underbrace{12, 11, 13}_{>q}]$$

Es importante respetar el orden relativo de las tres partes, pero **no** es necesario respetar el orden de los números dentro de cada una de las partes. Por ejemplo, otra salida válida posible podría ser:

$$[\underbrace{-2, -1}_{<p}, \underbrace{1, 2, 3, 5}_{[p,q]}, \underbrace{13, 12, 11}_{>q}]$$

Diseñar un algoritmo para resolver este problema en tiempo lineal, usando sólo $O(1)$ de espacio extra. Por simplicidad, se puede asumir que p, q no aparecen en el arreglo A. *Sugerencia:* mantener como invariante la siguiente configuración:

elementos $< p$	elementos en $[p, q]$	elementos aún no procesados	elementos $> q$