Programación 3
Año 2022

Práctico 7

Ejercicio 1 (Kleinberg & Tardos, Ex. 5.3)**.** Queremos analizar n tarjetas bancarias que fueron confiscadas por sospecha de fraude. Cada tarjeta está asociada a una única cuenta de banco, aunque cada cuenta puede estar asociada a más de una tarjeta; dos tarjetas son *equivalentes* si están asociadas a una misma cuenta.

Leer y comparar los datos de cada tarjeta es un proceso costoso, pero el banco cuenta con un dispositivo al que se ingresan dos tarjetas y responde rápidamente si son equivalentes o no.

Nos interesa saber si entre las n tarjetas confiscadas hay algún conjunto de más de $n/2$ tarjetas que son todas equivalentes entre sí.

(a) Dé un algoritmo de tipo *divide y vencerás* para solucionar el problema realizando $O(n \log n)$ pruebas de equivalencia con el dispositivo mencionado.
(b) Demuestre la corrección de su algoritmo.
(c) Demuestre que la cantidad de pruebas de equivalencia que realiza su algoritmo es $O(n \log n)$ como se pide. Enuncie los resultados teóricos del libro de referencia que utilice.

Sugerencia: Note que no puede haber más de una clase de equivalencia (conjunto de tarjetas equivalentes entre sí) de tamaño mayor a $n/2$. Utilice este hecho para definir un algoritmo que devuelve, si existe, un representante de dicha clase de equivalencia.

Ejercicio 2 (Kleinberg & Tardos, Ex. 5.2)**.** Sea $L_1, L_2, \ldots, L_n$ una secuencia de números. Decimos que un par de índices i, j, $1 \leq i < j \leq n$, es una *inversión significativa* si se cumple $L_i > 2L_j$.

(a) Dé un algoritmo para calcular la cantidad de inversiones significativas. Su algoritmo debe admitir una implementación con tiempo de ejecución $O(n \log n)$.
Sugerencia: Adapte la función Merge-and-count de la sección 5.3 del libro; observe que el orden relativo de los elementos de una secuencia no se altera si todos se multiplican por dos.
(b) Demuestre la corrección de su algoritmo.

Sugerencia: Demuestre que para n mayor que una constante (determinada por la condición del paso base de su algoritmo), la cantidad de inversiones en L, $R(L)$, se puede expresar como

$$R(L) = R(A) + r + R(B)\,, \tag{2.1}$$

donde A y B son las partes en que se subdivide el problema original, de tamaños n_A y n_B, respectivamente, y

$$r = \sum_{j=1}^{n_B} \sum_{i=1}^{n_A} \mathbb{1}_{\{A_i > 2B_j\}}\,. \tag{2.2}$$

Luego demuestre por inducción que su algoritmo calcula $R(L)$.

(c) Demuestre que su algoritmo admite una implementación con tiempo de ejecución que es $O(n \log n)$. Enuncie los resultados teóricos del libro de referencia que utilice.

Ejercicio 3. El algoritmo de ordenación llamado *Quicksort* utiliza un método con el cual se redistribuyen los elementos de un segmento de un arreglo en torno a un *separador*, también llamado *pivote*, dejando el segmento *parcialmente ordenado*. Tras esa redistribución el pivote queda en una posición, s, los elementos menores que el pivote quedan en las posiciones anteriores a s, y los elementos mayores que el pivote quedan en las posiciones posteriores a s. El segmento del arreglo queda parcialmente ordenado en el sentido de que los elementos que quedan en las posiciones anteriores a s, aunque no están ordenados entre sí, quedan correctamente ordenados con respecto al pivote y a los que quedan en las posiciones posteriores. Además el pivote queda exactamente en la posición que quedaría si el segmento se ordenara totalmente.

El algoritmo de la figura 3.1, `Particion`, es una posible versión del método, en el cual se elige el primer elemento del segmento como pivote. El tiempo de ejecución es $\Theta(n)$, donde n es la cantidad de elementos del segmento. Es un algoritmo `on place`, lo cual significa que no usa un arreglo auxiliar, sino que el espacio adicional usado es $O(1)$.

Como se puede ver, en cada iteración del ciclo de la línea 4 la distancia entre las posiciones s e i disminuye en uno, por lo que el ciclo termina en $\Theta(n)$ iteraciones. Además se cumple que cada elemento en el rango $inf\,..\,s$ es menor o igual que el pivote, y cada elemento en el rango $i+1\,..\,sup$ es mayor que el pivote. Al empezar cada iteración el rango $s+1\,..\,i$ no fue explorado.

Data: Un segmento de arreglo: $A[inf\,..\,sup], inf \leq sup$
Result: El pivote es $A[inf]$; los elementos del arreglo se redistibuyen: el pivote queda en la posición s; cada elemento en $\{A[inf], \ldots, A[s]\}$ es menor o igual al pivote; cada elemento en $\{A[s+1], \ldots, A[sup]\}$ es mayor que el pivote; devuelve la posición en que queda el pivote.

```
1  pivote ← A[inf]
2  s ← inf
3  i ← sup
4  while i > s do
5      if A[i] ≤ pivote then
6          s ← s + 1
7          Intercambiar A[s] y A[i]
8      else
9          i ← i − 1
10 Intercambiar A[inf] y A[s]
11 return s
```

Figura 3.1: Particion

Dato:		5	2	8	6	4	3
Ciclo:	$i > s$						
		5	2	8	6	4	3
		5	3	8	6	4	2
		5	3	2	6	4	8
		5	3	2	6	4	8
		5	3	2	4	6	8
	$i = s$	5	3	2	4	6	8
Intercambio:		→			←		
Resultado:		4	3	2	5	6	8

Figura 3.2: Aplica `Particion` a `Dato` obteniendo `Resultado` En cada paso del ciclo se indica en verde la región que ya fue explorada y contiene elementos menores o iguales al pivote, y en amarillo la región explorada que contiene elementos mayores al pivote. El cursor s está en el extremo derecho de la región verde. El cursor i, que referencia el próximo elemento que se compara con el pivote, se muestra en rojo. En el resultado se indica en gris la posición en que queda el pivote.

Al terminar el ciclo s es igual a i por lo que todo el segmento fue explorado. Entonces se intercambia el pivote con el elemento que quedó en la posición s.

En la figura 3.2 se ve un ejemplo de ejecución de `Particion`.

(a) Asuma que se dispone del algoritmo `Mediana` que dado un segmento de arreglo (el arreglo y las posiciones que determinan el rango del segmento) devuelve la posición en que se encuentra la mediana de los elementos de ese rango[1]. El tiempo de ejecución de `Mediana` es $\Theta(n)$, donde n es la cantidad de elementos del segmento.

Usando los algoritmos `Particion` y `Mediana` diseñe un algoritmo que mediante la estrategia *dividir y conquistar* ordena un arreglo. Exprese el tiempo de ejecución con una relación de recurrencia indicando la procedencia de cada término. Determine el tiempo de ejecución del algoritmo.

[1]La mediana de un conjunto es el valor que quedaría en la posición central si se ordenara el conjunto.

Asuma que en el arreglo no hay elementos repetidos.

(b) Modifique el algortimo que ordena el arreglo usando `Particion` pero no `Mediana`. Analice el tiempo de ejecución. Describa una configuración de la entrada a la que le corresponda el peor caso del tiempo de ejecución. Escriba la relación de recurrencia para el peor caso y resuelvela.