Examen de Programación 3
4 de febrero de 2019

En recuadros con este formato aparecerán aclaraciones que cumplen una función explicativa pero que no eran requeridos como parte de la solución.

Ejercicio 1 (35 puntos)

Sea A un arreglo con n elementos, donde n es una potencia de 2 mayor que 1. Los elementos de A pertenecen a un tipo abstracto de datos, T, donde se encuentra definida una relación de orden total; la función booleana `EsMenor`(a, b) toma dos elementos del tipo T y devuelve **true** si y solo si a es menor que b. Los elementos de tipo T pueden ser copiados a variables auxiliares directamente mediante operaciones de asignación. Se puede asumir también que existe una primitiva, denotada $B[i : j]$, $i \leq j$, que construye un subarreglo de B con los elementos en posiciones $i, i+1, \ldots, j$, donde B es un arreglo de elementos de tipo T.

Queremos obtener el mínimo y el máximo del conjunto de elementos contenidos en A. Una implementación directa, recorriendo el arreglo A de principio a fin, requeriría $2(n-1)$ invocaciones a `EsMenor`. En este ejercicio construimos una solución que requiere solo $3n/2 - 2$ invocaciones, usando la técnica *divide y vencerás*.

(a) Dé un algoritmo de tipo *divide y vencerás* para obtener el mínimo y el máximo de A. La cantidad de invocaciones a `EsMenor` para un arreglo de tamaño n, $C(n)$, debe satisfacer la siguiente relación de recurrencia

$$C(2) = 1, \tag{1}$$
$$C(n) = 2C(n/2) + 2, \quad n > 2. \tag{2}$$

(b) Muestre que, para el algoritmo propuesto, $C(n)$ satisface la relación de recurrencia pedida en la parte a y demuestre que se cumple $C(n) = 3n/2 - 2$, $n \geq 2$.

(c) Demuestre la corrección del algoritmo propuesto.

Solución:

(a) El problema se resuelve mediante el algoritmo de la figura 1.

```
1  Algorithm MinMax(A)
       Input: A es un arreglo de n elementos, n potencia de 2, n > 1.
2      if n = 2 then
3          if EsMenor(A[1], A[2]) then
4              min = A[1]
5              max = A[2]
6          else
7              min = A[2]
8              max = A[1]
9          end
10     else
11         (min_1, max_1) = MinMax(A[1 : n/2])
12         (min_2, max_2) = MinMax(A[n/2 + 1 : n])
13         if EsMenor(min_1, min_2) then min = min_1 else min = min_2
14         if EsMenor(max_1, max_2) then max = max_2 else max = max_1
15     end
16     return (min, max)
17 end
```

Figura 1: Algoritmo de tipo *divide y vencerás* para obtener el mínimo y el máximo de A.

(b) Para $n = 2$, por la condición en el paso 2, se realiza solo una invocación a `EsMenor` en el paso 3, por lo cual se satisface (1). En otro caso, n es una potencia de 2 mayor que 2, y los pasos 11 y 12

realizan llamadas recursivas sobre arreglos de tamaño $n/2$, cada una de las cuales realiza $C(n/2)$ invocaciones a `EsMenor`. Adicionalmente, los pasos 13 y 14 realizan una invocación cada uno. En total, para $n > 2$, se realizan entonces $2C(n/2) + 2$ invocaciones a `EsMenor`, en acuerdo con (2). Esto demuestra que, para el algoritmo de la figura 1, $C(n)$ satisface la relación de recurrencia pedida.

Ahora usamos esta relación recurrencia para mostrar que $C(n) = 3n/2 - 2$. Sea $n = 2^i$ una potencia de 2 mayor que 1, es decir, con $i \geq 1$; la demostración es por inducción en i. Para $i = 1$ tenemos $n = 2$, de modo que $3n/2 - 2 = 1$ y también, por (1), $C(n) = 1$. Supongamos ahora que $i > 1$ y la tesis de inducción se cumple para valores menores de i. Como tenemos $n/2 = 2^{i-1}$, la hipótesis de inducción implica que $C(n/2) = 3n/4 - 2$, por lo cual, reemplazando en (2), obtenemos

$$C(n) = 2\Big(3n/4 - 2\Big) + 2\,, \tag{3}$$

que, simplificando, se reduce a $C(n) = 3n/2 - 2$.

(c) Sea $n = 2^i$ una potencia de 2, $i \geq 1$. La demostración de corrección es por inducción en i. Para $i = 1$, tenemos $n = 2$, de modo que la condición del paso 2 se satisface y el resultado, calculado en las variables min, max, queda determinado por los pasos 3–9. El problema de encontrar el mínimo y el máximo en un arreglo de tamaño 2 se reduce a identificar uno de los dos elementos, $A[1]$ o $A[2]$, que es mayor o igual al otro. Este elemento es el máximo y el otro es el mínimo. Esto es precisamente lo que se resuelve mediante la comparación del paso 3, que es verdadera si $A[2]$ es máximo y falsa si $A[1]$ es máximo. Esto muestra que el algoritmo funciona correctamente para $i = 1$.

Sea ahora $i > 1$ y supongamos que el algoritmo funciona correctamente para arreglos de tamaño 2^{i-1}. En este caso la condición del paso 2 no se satisface y el resultado queda determinado por los pasos 10–15. Las llamadas recursivas de los pasos 11 y 12 se realizan sobre subarreglos de tamaño $n/2$, que es igual a 2^{i-1}. Por lo tanto, por hipótesis de inducción, las variables min_1, max_1 y min_2, max_2 contienen el mínimo y el máximo de los subarreglos $A[1 : n/2]$ y $A[n/2 + 1 : n]$, respectivamente. Como estos subarreglos cubren el arreglo completo A, el mínimo de los elementos de A es el mínimo entre min_1 y min_2, tal como se calcula en el paso 13. De forma similar, el máximo de los elementos de A es el máximo entre max_1 y max_2, como se calcula en el paso 14. Esto completa la demostración de corrección.

Ejercicio 2 (40 puntos)

Dadas dos cadenas de caracteres, $x = (x_1, x_2, \ldots, x_n)$, $y = (y_1, y_2, \ldots, y_m)$, definimos la *distancia de edición* de x a y como el menor número de cambios que son necesarios para transformar x en y. Un cambio consiste en alguna de las siguientes operaciones:

- Insertar un carácter: $(x_1, \ldots, x_n) \mapsto (x_1, \ldots, x_i, c, x_{i+1}, \ldots, x_n)$.
- Eliminar un carácter: $(x_1, \ldots, x_n) \mapsto (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$.
- Sustituir un carácter: $(x_1, \ldots, x_n) \mapsto (x_1, \ldots, x_{i-1}, c, x_{i+1}, \ldots, x_n)$.

Modelamos una secuencia de operaciones que transforma x en y como un emparejamiento, S, (no necesariamente perfecto) entre el conjunto de posiciones de caracteres de x, $\{1, 2, \ldots, n\}$, y el de y, $\{1, 2, \ldots, m\}$. Este emparejamiento representa una *alineación* entre x, y, por lo cual no contiene ningún cruce, es decir, en S no existen parejas (i, j), (i', j'), tales que $i < i'$ y $j' < j$. La figura 2 ilustra un emparejamiento entre las cadenas $x = aniversario$, $y = universos$. Una pareja (i, j) de S corresponde a una coincidencia entre los caracteres x_i, y_j (como $(2, 2)$ y $(11, 8)$ en la figura) o una operación de sustitución del carácter x_i por y_j (como $(1, 1)$ en la figura). Una posición j de y que no está emparejada corresponde a una operación de inserción de y_j (como la posición 9 de y en la figura). Una posición i de x que no está emparejada corresponde a una operación de eliminación de x_i (como las posiciones 8, 9 y 10 de x en la figura).

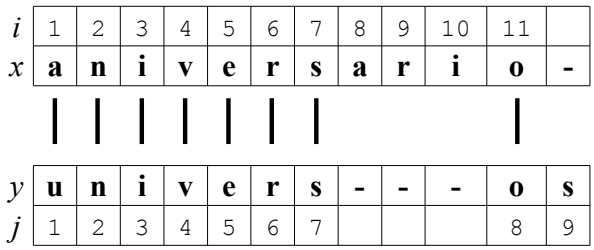

Figura 2: Alineación de las secuencias $x = aniversario$, $y = universos$ representada por el emparejamiento $S = \{(1, 1), (2, 2), \ldots, (7, 7), (11, 8)\}$.

Definimos $OPT(i, j)$, $0 \leq i \leq n$, $0 \leq j \leq m$, como la distancias de edición del prefijo $x_1, \ldots, x_i$ de x al prefijo $y_1, \ldots, y_j$ de y.

(a) ¿Cuánto vale $OPT(0, j)$, $0 \leq j \leq m$? ¿Cuánto vale $OPT(i, 0)$, $0 \leq i \leq n$?

(b) Supongamos que n y m son positivos y sea S un emparejamiento que corresponde a una secuencia óptima de operaciones para transformar x en y. Argumente que o bien n está emparejada con una posición de y o m está emparejada con una posición de x (o ambas), pero no puede ocurrir que ninguna de las dos esté emparejada. Argumente también que si ambas están emparejadas entonces deben estar emparejadas entre sí.

(c) Especifique una relación de recurrencia para OPT. Explique el origen de cada término.

(d) Dé un algoritmo **iterativo** de programación dinámica que permita calcular la distancia de edición de x a y usando la recurrencia de la parte anterior. El algoritmo debe admitir una implementación cuyo tiempo de ejecución es polinomial en n y m (no es necesario demostrarlo).

Solución:

Este problema es un caso particular del de *alineación de secuencias* que se presenta en la sección 6.6 del libro de Kleinberg & Tardos. En este caso particular tenemos $\delta = 1$ y $\alpha_{x_i, y_j} = 1$ si $x_i \neq y_j$ y $\alpha_{x_i, y_j} = 0$ en otro caso.

(a) Si x tiene largo 0, podemos transformarla en y insertando j caracteres, donde j es el largo de y. Evidentemente no es posible realizar una cantidad menor de operaciones, por lo cual tenemos $OPT(0, j) = j$.

Análogamente, tenemos $OPT(i,0) = i$, ya que podemos transformar una cadena x de largo i en una cadena vacía mediante i operaciones de eliminación.

(b) Si la posición n de x no está emparejada en S y la posición m de y tampoco, entonces podríamos disminuir la cantidad de operaciones, contradiciendo el hecho de que S es óptimo. En efecto, agregando (n,m) a S reemplazamos dos operaciones, la inserción de y_m y la eliminación de x_n, por una operación de sustitución de x_n por y_m. Esto muestra que al menos alguna de las dos posiciones finales de x, y, deben estar emparejadas. Si ambas lo están, debe ser entre sí, porque de lo contrario habría un cruce.

(c) Sea S un emparejamiento entre $\{1,2,\ldots,i\}$ y $\{1,2,\ldots,j\}$ que corresponde a una secuencia óptima de operaciones para transformar $x_1,\ldots,x_i$ en $y_1,\ldots,y_j$. Distinguimos tres casos:

1. Si $(i,j) \in S$, entonces $S \setminus \{(i,j)\}$ define una transformación de $x_1,\ldots,x_{i-1}$ en $y_1,\ldots,y_{j-1}$, que realiza una cantidad óptima de operaciones, $OPT(i-1,j-1)$, porque de lo contrario S no sería óptimo. La cantidad de operaciones implicadas en S es por lo tanto $OPT(i-1,j-1) + \alpha_{i,j}$, donde $\alpha_{i,j} = 1$ si $x_i \neq y_j$ y $\alpha_{i,j} = 0$ en otro caso.
2. Si la posición i de x no está emparejada en S, entonces S define una secuencia de operaciones que transforma $x_1,\ldots,x_{i-1}$ en $y_1,\ldots,y_j$; la transformación completa se obtiene agregando una operación de eliminación para x_i. Esta transformación de $x_1,\ldots,x_{i-1}$ en $y_1,\ldots,y_j$ realiza $OPT(i-1,j)$ operaciones, porque si hiciera más S no sería óptimo. En consecuencia, en este caso tenemos $OPT(i,j) = OPT(i-1,j) + 1$.
3. Si la posición j de y no está emparejada en S, entonces S define una secuencia de operaciones que transforma $x_1,\ldots,x_i$ en $y_1,\ldots,y_{j-1}$; la transformación completa se obtiene agregando una operación de inserción para y_j. Esta transformación de $x_1,\ldots,x_i$ en $y_1,\ldots,y_{j-1}$ realiza $OPT(i,j-1)$ operaciones, porque si hiciera más S no sería óptimo. En consecuencia, en este caso tenemos $OPT(i,j) = OPT(i,j-1) + 1$.

Por la parte anterior, estos tres casos son los únicos posibles, de donde concluimos que se cumple

$$OPT(i,j) = \text{mín}\left\{OPT(i-1,j-1) + \alpha_{i,j}, OPT(i-1,j) + 1, OPT(i,j-1) + 1\right\}, \quad i > 0, j > 0\,. \quad (4)$$

La relación de recurrencia se completa con los pasos base que se derivan de la parte a,

$$OPT(0,j) = j\,, \quad 0 \leq j \leq m\,, \quad (5)$$
$$OPT(i,0) = i\,, \quad 1 \leq i \leq n\,. \quad (6)$$

Alternativamente, se puede argumentar que si $x_n = y_m$ entonces cualquier emparejamiento óptimo, S, tal que $(n,m) \notin S$ se puede transformar en otro, S', también óptimo, tal que $(n,m) \in S'$: si $(n,m') \in S$ reemplazamos (n,m') por (n,m) y si en cambio $(n',m) \in S$ reemplazamos (n',m) por (n,m). Con esta formulación la recurrencia resulta

$$\begin{aligned}
OPT(0,j) &= j\,, \quad 0 \leq j \leq m\,,\\
OPT(i,0) &= i\,, \quad 1 \leq i \leq n\,,\\
OPT(i,j) &= OPT(i-1,j-1)\,, \quad i > 0, j > 0, x_i = y_j\,,\\
OPT(i,j) &= 1 + \text{mín}\left\{OPT(i-1,j-1), OPT(i-1,j), OPT(i,j-1)\right\}, \quad i > 0, j > 0, x_i \neq y_j\,.
\end{aligned}$$

(d) El problema se resuelve mediante el algoritmo de la figura 3.

```
1  Algorithm DistanciaEdición(x, y)
2      Hacer OPT[0, j] = j,    0 ≤ j ≤ m
3      Hacer OPT[i, 0] = i,    1 ≤ i ≤ n
4      for i = 1 to n do
5          for j = 1 to m do
6              if x_i = y_j then hacer α = 0 else hacer α = 1
7              Hacer OPT[i, j] = mín {OPT[i − 1, j − 1] + α, OPT[i − 1, j] + 1, OPT[i, j − 1] + 1}
8          end
9      end
10     return OPT[n, m]
11 end
```

Figura 3: Algoritmo para calcular la distancia de edición de x a y.

Ejercicio 3 (25 puntos)

(a) Defina el concepto de *certificador eficiente* para un problema de decisión.

(b) Defina la clase de complejidad $\mathcal{NP}$ para problemas de decisión.

(c) Defina la clase de problemas de decisión que son $\mathcal{NP}$-completos. Puede usar la notación $\leq_P$ para expresar la definición.

Solución:

(a) Ver sección 8.3 del libro de Kleinberg & Tardos.

(b) Ver sección 8.3 del libro de Kleinberg & Tardos.

(c) Ver sección 8.4 del libro de Kleinberg & Tardos.