Segundo examen parcial

AA 361075 Carlos Eduardo Sánchez Torres

14 de octubre de 2021

1. El ordenamiento por inserción puede ser expresado como un procedimiento recursivo. Para ordenar A[1..n], se ordena recursivamente A[1..n–1] y después se inserta A[n] en el arreglo ordenado A[1..n–1]. (20 puntos)

```
insert(sequence, i)
    key = sequence[i+1]
    while i >= 0 and sequence[i] >  key
        sequence[i+1] = sequence[i]
        i = i - 1
    sequence[i+1] = key

insertion_sort_recursive_stack(sequence, n)
     if n > 2:
         insertion_sort_recursive_stack(sequence, n-1)
         insert(sequence, n-2)
     return sequence
```

Listing 1: Ordenamiento por inserción recursivo.

a. Argumente por qué esta versión sigue un esquema Divide y Vencerás.

Todos los algoritmos diseñados con divide y vencerás tienen los siguientes 3 pasos: dividir el problema en subproblemas, vencer los subproblemas recursivamente hasta el caso base, combinar las sub-soluciones en una solución el cual resuelve el problema original y el ordenamiento por inserción recursivo tiene los 3 pasos: divide la secuencia en ordenados A[n-1..n] y no ordenados A[1..n-1], se ordena recursivamente la no ordenada utilizando insertion sort y se combinan la secuencia ordenada con la no ordenada, esto es, se inserta A[n-1] en la secuencia ordenada, entonces el ordenamiento por inserción recursivo sigue un esquema Divide y Vencerás.

b. Exprese la recurrencia que corresponde al tiempo de ejecución de esta versión.
Dividir. Calcula el arreglo no ordenado en tiempo constante.
Vencer. Recursivamente se resuelve un subproblema de tamaño $n-1$, contribuyendo T(n-1) al tiempo de ejecución.
Combinar. El procedimiento *insert* toma $O(n)$ para insertar.

$$T(n) = \begin{cases} \Theta(1) & n \leq 2 \\ T(n-1) + O(n) & n > 2 \end{cases} \tag{1}$$

c. Obtenga su solución en notación asintótica para el mejor y peor caso.
Para n pequeñas sabemos por el inciso anterior que es T(n) constante. Por consecuencia, y sin pérdidas de generalidad

$$T(n) \leq T(n-1) + cn \tag{2}$$

Mejor caso. La secuencia está ordenada, esto significa que *insert* toma $\Theta(1)$ en insertar:

$$T(n) = T(n-1) + \Theta(1) \tag{3}$$

$$T(n) = T(n-1) + c \tag{4}$$

Por árbol de recurrencias:

$$T(n) = cn + c_1 = \Theta(n) \tag{5}$$

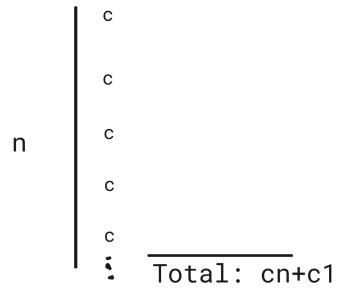

Figura 1: Árbol de recurrencias. c_1 el tiempo de pila de la recursividad.

Peor caso. La secuencia está invertida, esto significa que *insert* toma $\Theta(n)$ en insertar:

$$T(n) = T(n-1) + \Theta(n) \tag{6}$$

$$T(n) = T(n-1) + cn \tag{7}$$

Por árbol de recurrencias:

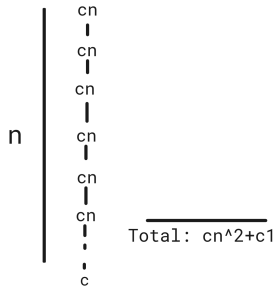

Figura 2: Árbol de recurrencias. c_1 el tiempo de pila de la recursividad.

$$T(n) = cn^2 + c_1 = \Theta(n^2) \tag{8}$$

4. Considere el algoritmo *LCS-LENGTH* del libro de texto (sección 15.4, página 394) para calcular la longitud de la subsecuencia común más larga. Modifique el algoritmo tal que solamente requiera min(m,n) más una cantidad constante de espacio adicional de almacenamiento (siendo m y n las longitudes de las cadenas de entrada). Analice el T(n) del nuevo algoritmo para su mejor y peor caso. (20 puntos)
Observaciones:

- Dado que no necesitamos determinar LCS, podemos eliminar la tabla b sin problemas y dejar de usar el método de impresión. El espacio asintótico no disminuye $S(n) = \Theta(mn)$.
- Nuestro índice inicial es 0 y no 1, es decir, sea un arreglo $a = [a_0, ..., a_n]$, $a[0] = a_0$. Cormen usa a conveniencia uno u otro estándar (véase el comentario de la línea 3 del algoritmo [1]), solo usaré el mencionado anteriormente.

```
1  LCSL(X, Y)
2    m = X.length
3    n = Y.length
4    a = min(m,n) + 1
5    b = max(m,n) + 1
6    c = zeros(a) // Sea c una secuencia de a=(min(m,n)+1) ceros
7    for i=1 to b
8       for j=1 to a
9          if X[i-1] == Y[j-1]
10           c[j] = c[j-1] + 1
11         elif c[j] < c[j-1]
12           c[j] = c[j-1]
13   return c[a-1]
```

Listing 2: LCS-LENGHT

La suma de k las constantes $c_1 + c_2 + c_3 + ... + c_k = c$.

$$S(m,n) = c_1 + c_2 + c_3 + c_4 + min(m,n) + c_5 + c_6 + c_7 = min(m,n) + c \tag{9}$$

El tiempo de inicialización del vector de ceros es $T(m,n) = (min(m,n) + 1)$, las estructuras de control condicionales de la línea 9 a la 12 claramente se ejecutan una o la otra así T(m,n)=(mn-1).

$$T(m,n) = c_1 + c_2 + c_3 + c_4 + min(m,n) + 1 + max(m,n) + mn + (mn-1) + c5 \tag{10}$$

$$T(m,n) = 2mn + max(m,n) + min(m,n) + c \tag{11}$$

Sabemos que $max(m,n) + min(m,n) = m + n$ porque si $max(m,n) = m, min(m,n) = n \implies max(m,n) + min(m,n) = m + n$ y $max(m,n) = n, min(m,n) = m \implies max(m,n) + min(m,n) = n + m$ entonces $m + n = n + m$, pudiendo sustituirlo:

$$T(m,n) = 2mn + m + n + c \tag{12}$$

Dado que no existen instrucciones que no se cumplan podemos decir que es el mismo tiempo de ejecución T(n):

$$T(m,n) = 2mn + m + n + c = \Theta(mn) \tag{13}$$

Sea $c_1, c_2, n_0, m_0 \geq 0$, $\forall n \geq n_0$ y $\forall m \geq m_0$

$$c_1 mn \leq 2mn + m + n + c \leq c_2 mn \tag{14}$$

$$c_1 \leq 2 + \frac{1}{n} + \frac{1}{m} + \frac{1}{mn} \leq c_2 \tag{15}$$

Tenemos que

$$lim_{(m,n)\to(\infty,\infty)} 2 + \frac{1}{n} + \frac{1}{m} + \frac{1}{mn} = 2 \implies c_1 = 2 \tag{16}$$

Para $n \geq n_0 = 1$ y $m \geq m_0 = 1$, $c_2 = 5$. Concluyendo, podemos elegir $n_0 = m_0 = 1, c_1 = 2, c_2 = 5$ tal que $2mn + m + n + c = \Theta(mn)$

3. Considere el problema de dar cambio de n centavos (siendo n un entero positivo) utilizando la menor cantidad de monedas. Suponga que los valores de las monedas son enteros positivos. (25 puntos)

a. Proponga un algoritmo voraz iterativo que otorgue el cambio suponiendo que posee monedas de 25, 10, 5 y 1 centavo. Utilice la técnica de invariante de lazo para demostrar que su algoritmo es correcto.

Entrada. $n \in \mathbb{N}$ centavos.

Salida. Minimizar $(c_1 + c_2 + c_3 + c_4) \in \mathbb{N}$ tal que $25c_1 + 10c_2 + 5c_3 + 1c_4 = n$.

La estructura sub-óptima consiste en almacenar los restos $(n/coeficiente)$ y guardar los cocientes $(c_1, c_2, ...)$, donde la opción voraz es empezar por el coeficiente más grande, dejando como subproblema el resto.

```
makeChange(n)
  change = {}
  coins = [25,10,5,1]
  coin = 1 # coins[1] = 25
  while n > 0
    c = floor(n / coins[coin])
    n = n % coins[coin]
    change[coins[coin]] = c # c1,c2,...
    coin = coin + 1
  return change
```

Listing 3: Algoritmo voraz iterativo

Invariante. Al inicio de cada iteración, c_i (monedas) es la división redondeada hacia abajo del $resto_{i-1}$ entre $denominación_i$ (la cual es única) y $resto_i$ convergiendo a 0 y su suma a n. El cambio contiene $\{denominación_0 : c_0, ..., denominación_i : c_i\}$:

$$25c_1 + 10c_2 + 5c_3 + 1c_4 = n \tag{17}$$

$$\sum_{i=1}^{m=4} denominación_i * c_i = n \tag{18}$$

$$\sum_{i=1}^{m} resto_{i-1} = n \tag{19}$$

$$\sum_{i=1}^{m} [resto_{i-1}]_{denominacion_i} = n \tag{20}$$

$$[resto_0]_{denominacion_1} + [resto_1]_{denominacion_2} + ... + [resto_{m-1}]_{denominacion_m} = n \tag{21}$$

$$[n]_{denominacion_1} + [resto_1]_{denominacion_2} + ... + [resto_{m-1}]_1 = n \tag{22}$$

$$[n]_{denominacion_1} + [resto_1]_{denominacion_2} + ... + [resto_{m-1}]_1 = n \tag{23}$$

Inicialización. El $resto_0 = n$ entre $denominación_i$ es n, esto claramente significa que el cambio total es vació o nulo. Cumpliéndose la invariante al inicio.

Mantenimiento. Suponiendo que la invariante de lazo es correcta, $c_1 = \lfloor \frac{resto_0}{denominación_1} \rfloor, c_2 = \lfloor \frac{resto_1}{denominación_2} \rfloor, ...,$ $c_i = \lfloor \frac{resto_{i-1}}{denominación_i} \rfloor$. Si al inicio de la iteración i la invariante se cumple, entonces el cambio contiene las monedas c_i por denominación[i] (instrucción 8): $\{denominación_i : c_i\}$. Al iniciar la iteración i+1 la $denominación_{i+1}$ -gracias a la instrucción 9 tiende a m denominaciones- y $resto_{i+1} = resto_{i-1} \, \%denominacion_i = [resto_{i-1}]_{denominacion_i}$ cumpliéndose la invariante.

Terminación. El algoritmo termina cuando $resto_j = 0$, sustituyendo en la invariante $\{denominación_0 : c_0, ..., denominación_i : c_i\}, , denominación_{j-1} : c_{j-1}\}$, claramente

$$[n]_{denominacion_1} + [resto_1]_{denominacion_2} + ... + [resto_{j-1}]_{denominacion_j} = n \tag{24}$$

Cumpliéndose la invariante.

b.Indique un conjunto de denominaciones de monedas tal que su algoritmo voraz no genere una solución óptima. El conjunto debe incluir siempre a la unidad (un centavo) para asegurar que siempre existe una solución.

Para el conjunto de denominaciones $(25, 20, 10, 5, 1)$ y $n = 40$, el algoritmo produce: 25+10+5 (3 monedas), sin embargo, la solución óptima es 20+20 (2 monedas). El problema fue optar por el coeficiente más grande, cuando existe un coeficiente menor tal que con $n/20 == 2$.

Nota. El problema puede resolverse mediante programación lineal, el cual siempre genera una solución óptima. Por ejemplo, para la última situación:
Sea c_i la cantidad de monedas por denominación, entonces

$$MINZ = c_1 + c_2 + c_3 + c_4 + c_5 \tag{25}$$

$$25c_1 + 20c_2 + 10c_3 + 5c_4 + c_5 = 40 \tag{26}$$

$$c_i \in \mathbb{N} \tag{27}$$

4. Sea $A[1..n]$ un arreglo de n números distintos. Si $i < j$ y $A[i] > A[j]$, al par (i, j) se le conoce como una inversión de A. Suponga que los elementos de entrada, A, son una permutación uniformemente aleatoria de los valores $< 1, 2, ..., n >$. Utilice variables aleatorias indicadoras para calcular el número esperado de inversiones.

Sea X_{ij} una variable aleatoria indicadora tal que

$$X_{ij} = I\{A[i] > A[j]\} \text{ para } 1 \leq i \leq j \leq n \tag{28}$$

Sabemos que

$$P(X_{ij}) = \frac{\sum_{k=0}^{n-1} k}{(n-1)(n)} = \frac{1}{2} \tag{29}$$

Por Lema 5.1

$$E[X_{ij}] = \frac{1}{2} \tag{30}$$

Sea X la variable aleatoria que denota el número de inversiones en A:

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij} \tag{31}$$

Podemos calcular ahora el número esperado de inversiones.

$$E[X] = E[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}] \tag{32}$$

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}] \tag{33}$$

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 1/2 \tag{34}$$

$$E[X] = \sum_{i=1}^{n-1} (n-i)/2 \tag{35}$$

$$E[X] = \frac{n(n-1)}{4} \tag{36}$$

5. Suponga que se proporciona un conjunto de n objetos, tal que el tamaño s_i del i-ésimo objeto satisface que $0 < s_i < 1$. Se requiere empacar todos los objetos en un conjunto de cajas de tamaño unitario. Cada caja puede contener cualquier subconjunto de objetos cuyo tamaño total no exceda 1. La heurística ***first-fit*** toma cada objeto y lo acomoda en la primera caja que pueda contenerlo. Finalmente, sea S la suma de los tamaños de todos los objetos:

a. Argumente a favor de que el número óptimo de cajas requeridas es al menos $\lceil S \rceil$.
Sea $m \in \mathbb{N}$ un conjunto de cajas ($m = optimo$) que contienen c_j donde $0 < c_j \leq 1$ (si fuera $c_j = 0$ no necesitaríamos dicha caja, por tanto se obvia este resultado), desde la suma de los contenidos de las cajas, podemos establecer que:

$$\sum_{j=1}^{m} c_j = S \tag{37}$$

También sabemos que:

$$sup\{c_j \in \mathbb{R} | 0 < c_j \leq 1\} = 1 \tag{38}$$

Esto significa el máximo contenido de cajas esta dado por:

$$\sum_{j=1}^{m} 1 = m \tag{39}$$

Así

$$m \geq \sum_{j=1}^{m} c_j \tag{40}$$

$$m \geq S \tag{41}$$

Para ser más precisos

$$min\{m \in \mathbb{N} | m \geq S\} \tag{42}$$

Sea $S \in \mathbb{R}^+$, por definición de la función techo:

$$\lceil S \rceil = min\{s \in \mathbb{N} | s \geq S\} \tag{43}$$

$$\lceil S \rceil \geq S \tag{44}$$

$$m \geq \lceil S \rceil \tag{45}$$

Concluyendo que el número óptimo de cajas óptimo es al menos $\lceil S \rceil$.

Se dice que al menos, por la siguiente situación: $s = \{0{,}7, 0{,}7, 0{,}6\}$, vemos $m = 3 > \lceil S \rceil = 2$

b. Demuestre que el número total de cajas utilizadas por la heurística first-fit nunca es mayor a $\lceil 2S \rceil$.
Si tenemos el paquete b_n y lo empacamos en la caja c_m, esto significa que no existe una caja c_j entre c_1 y c_{m-1} tal que $c_j + b_n \geq 1$ pero si $c_m + b_n \leq 1$; para que ocurra lo contrario, esto es que se puede empacar b_n en alguna c_j: $c_j + b_n \leq 1$ (vuelve innecesario abrir c_m).
Situaciones:

$$c_j + b_n \leq 1 \tag{46}$$

$$c_m + b_n \leq 1 \tag{47}$$

Para ejemplificar:
$b_n = 0{,}5$ $0 < c_j \leq 0{,}5$, sino $c_m = 0{,}5$
$b_n = 0{,}9$ $0 < c_j \leq 0{,}1$, sino $c_m = 0{,}9$
$b_n = 0{,}3$ $0 < c_j \leq 0{,}7$, sino $c_m = 0{,}7$
$b_n = 0{,}2$ $0 < c_j \leq 0{,}8$, sino $c_m = 0{,}2$

Podemos observar que siempre existen al menos $m^* - 1$ cajas completas a la mitad en todos los casos. De manera general, si tienes una caja a la mitad llénala sino empaca una nueva.

Obtenemos que

$$\sum_{i=1}^{n} b_i > \frac{m^* - 1}{2} \tag{48}$$

Por el inciso anterior

$$m \geq \sum_{i=1}^{n} b_i > \frac{m^* - 1}{2} \tag{49}$$

$$m > \frac{m^* - 1}{2} \tag{50}$$

$$2m > m^* - 1 \tag{51}$$

Porque son enteros

$$2m >= m^* \tag{52}$$

Aplicando propiedades de la función techo:

$$2m \geq 2\lceil S \rceil \geq \lceil 2S \rceil >= m^* \tag{53}$$

Concluyendo

$$\lceil 2S \rceil >= m^* \tag{54}$$

c. Demuestre que el factor de aproximación de la heurística first-fit es 2. Si

$$2m >= m^* \tag{55}$$

entonces por definición factor de aproximación:

$$max(\frac{m^*}{m}, \frac{m}{m^*}) <= 2 \tag{56}$$

Referencias

[1] Cormen, T. H., & Leiserson, C. E. (2009). In Introduction to algorithms (3rd Edition).