Algoritmos y Estructuras de Datos II

Ordenación

11 de marzo de 2015

Contenidos

Número de operaciones de un comando

- Una vez que uno sabe qué **operación** quiere contar, debe imaginar una ejecución arbitraria, genérica del comando intentando contar el número de veces que esa ejecución arbitraria realizará **dicha operación**.
- Ése es el verdadero método para contar.
- Es imprescindible comprender **cómo** se ejecuta el comando.
- A modo de ayuda, en las filminas que siguen se da un método imperfecto para ir aprendiendo.
- El método supone que ya sabemos cuál **operación** queremos contar.

Número de operaciones de un comando
Secuencia de comandos

- Una secuencia de comandos se ejecuta de manera secuencial, del primero al último.
- Para contar cuántas veces se ejecuta **la operación**, entonces, se cuenta cuántas veces se ejecuta en el primero, cuántas en el segundo, etc. y luego se suman los números obtenidos:
- $\text{ops}(C_1;C_2;\ldots;C_n) = \text{ops}(C_1) + \text{ops}(C_2) + \ldots + \text{ops}(C_n)$
- $\text{ops}(C_1;C_2;\ldots;C_n) = \sum_{i=1}^{n} \text{ops}(C_i)$

Número de operaciones de un comando
Comando **skip**

- El comando **skip** equivale a una secuencia vacía:
- ops(**skip**) $= 0$

Número de operaciones de un comando
Comando **for**

- El comando **for** k:= n **to** m **do** C(k) **od** "equivale" también a una secuencia:
- **for** k:= n **to** m **do** C(k) **od** "equivale" a C(n);C(n+1);...;C(m)
- ops(**for** k:= n **to** m **do** C(k) **od**) = ops(C(n)) + ops(C(n+1)) + ... + ops(C(m))
- ops(**for** k:= n **to** m **do** C(k) **od**) $= \sum_{k=n}^{m} ops(C(k))$
- Esta ecuación solamente vale cuando **no se quieren contar las operaciones que involucran el índice k** implícitas en el **for**: inicialización, comparación con la cota m, incremento; ni el cómputo de los límites n y m. Por eso escribimos "equivale" entre comillas. En los apuntes hay otras ecuaciones posibles para el caso en que sí quieran contarse.

Número de operaciones de un comando

Comando condicional **if**

- El comando **if** b **then** C **else** D **fi** se ejecuta evaluando la condición b y luego, en función del valor de verdad que se obtenga, ejecutando C (caso verdadero) o D (caso falso).
- Para contar cuántas veces se ejecuta **la operación**, entonces, se cuenta cuántas veces se la ejecuta durante la evaluación de b y luego cuántas en la ejecución de C o D
- $\text{ops}(\textbf{if b then C else D fi}) = \begin{cases} \text{ops(b)+ops(C)} & b \text{ es V} \\ \text{ops(b)+ops(D)} & b \text{ es F} \end{cases}$

Número de operaciones de un comando
Asignación

- El comando x:=e se ejecuta evaluando la expresión e y modificando la posición de memoria donde se aloja la variable x con el valor de e.
-

$$\text{ops(x:=e)} = \begin{cases} \text{ops(e)+1} & \text{si se quiere contar la asignación} \\ & \text{o las modificaciones de memoria} \\ \text{ops(e)} & \text{caso contrario} \end{cases}$$

Número de operaciones de un comando
El ciclo **do**

- El ciclo **do** b $\rightarrow$ C **od** (o equivalente **while** b **do** C **od**) se ejecuta evaluando la condición b, y dependiendo de si su valor es V o F se continúa de la siguiente manera:
  - si su valor fue F, la ejecución termina ahí
  - si su valor fue V, la ejecución continúa con la ejecución del cuerpo C del ciclo, y luego de eso vuelve a ejecutarse todo el ciclo nuevamente.
- Es decir que su ejecución es una secuencia de evaluaciones de la condición b y ejecuciones del cuerpo C que finaliza con la primera evaluación de b que dé F.

Número de operaciones de un comando

El ciclo **do**

$$\text{ops}(\textbf{do b} \rightarrow \text{C } \textbf{od}) = \text{ops(b)} + \sum_{\text{k=1}}^{n} d_k$$

donde

- n es el número de veces que se ejecuta el cuerpo del **do**
- d_k es el número de operaciones que realiza la k-ésima ejecución del cuerpo C del ciclo y la subsiguiente evaluación de la condición o guarda b

Número de operaciones de una expresión

- Similares ecuaciones se pueden obtener para la evaluación de expresiones.
- Por ejemplo, para evaluar la expresión e<f, primero se evalúa la expresión e, luego se evalúa la expresión f y luego se comparan dichos valores.
-

$$\text{ops(e<f)} = \begin{cases} \text{ops(e)+ops(f)+1} & \text{si se cuentan comparaciones} \\ \text{ops(e)+ops(f)} & \text{caso contrario} \end{cases}$$

Ejemplo: número de comparaciones de la ordenación por selección

$$
\begin{aligned}
\text{ops(selection\_sort(a))} &= \sum_{i=1}^{n-1} \text{ops(min\_pos\_from(a,i))} \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \text{ops(a[j]} < \text{a[minp])} \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 1 \\
&= \sum_{i=1}^{n-1} (n-i) \\
&= \sum_{i=1}^{n-1} i \\
&= \frac{n*(n-1)}{2} \\
&= \frac{n^2}{2} - \frac{n}{2}
\end{aligned}
$$

Ejemplo: número de intercambios de la ordenación por selección

$$\begin{aligned} \text{ops(selection\_sort(a))} &= \sum_{i=1}^{n-1} \text{ops(swap(a,i,minp))} \\ &= \sum_{i=1}^{n-1} 1 \\ &= n\text{-}1 \end{aligned}$$

Conclusión del ejemplo

- Número de comparaciones de la ordenación por selección: $\frac{n^2}{2} - \frac{n}{2}$
- Número de intercambios de la ordenación por selección: n-1
- Esto significa que la operación de **intercambio no es representativa** del comportamiento de la ordenación por selección, ya que el número de comparaciones crece más que proporcionalmente respecto a los intercambios.
- Por otro lado, pudimos contar las operaciones de manera **exacta**.

Ordenación por inserción

- **No siempre** es posible contar el **número exacto** de operaciones.
- Un ejemplo de ello lo brinda otro algoritmo de ordenación: la ordenación por inserción.
- Es un algoritmo que se utiliza por ejemplo en juegos de cartas, cuando es necesario mantener un gran número de cartas en las manos, en forma ordenada.
- Cada carta que se levanta de la mesa, se inserta en el lugar correspondiente entre las que ya están en las manos, manteniendolas ordenadas.

Ordenación por inserción

Ejemplo

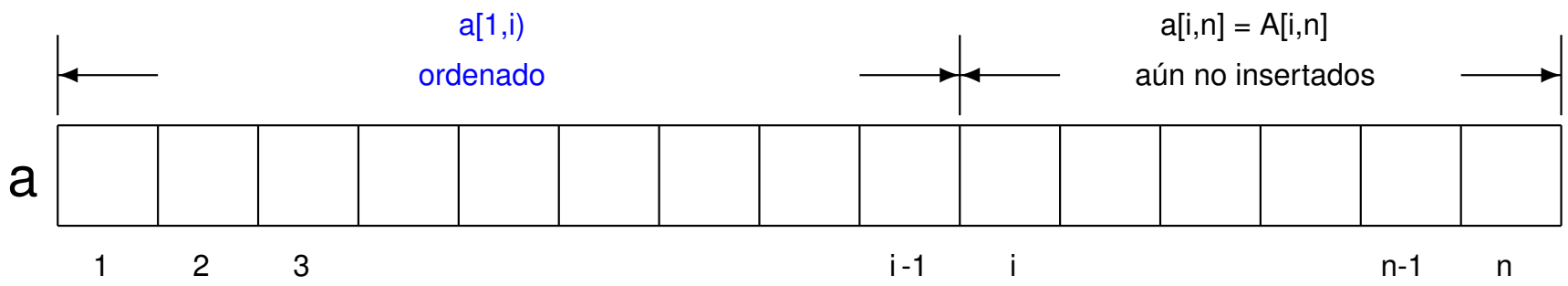
9	3	1	3	5	2	7
9	3	1	3	5	2	7
3	9	1	3	5	2	7
3	1	9	3	5	2	7
1	3	9	3	5	2	7
1	3	3	9	5	2	7
1	3	3	5	9	2	7

1	3	3	5	2	9	7
1	3	3	2	5	9	7
1	3	2	3	5	9	7
1	2	3	3	5	9	7
1	2	3	3	5	7	9

Demo (www.sorting-algorithms.com)

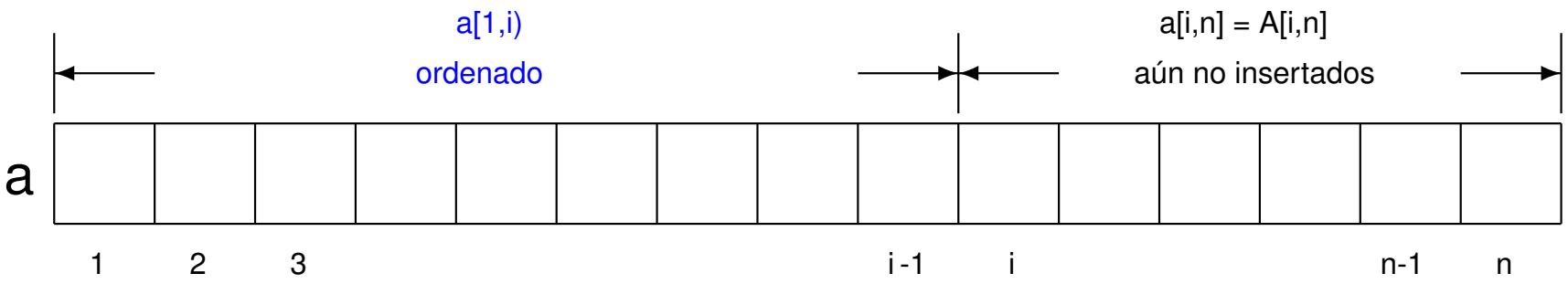
Ordenación por inserción

En un arreglo



Ordenación por inserción

Invariante



- Invariante:
  - el arreglo a es una permutación del original y
  - un segmento inicial a[1,i) del arreglo está ordenado.
  - (pero en general a[1,i) **no** contiene los mínimos del arreglo)

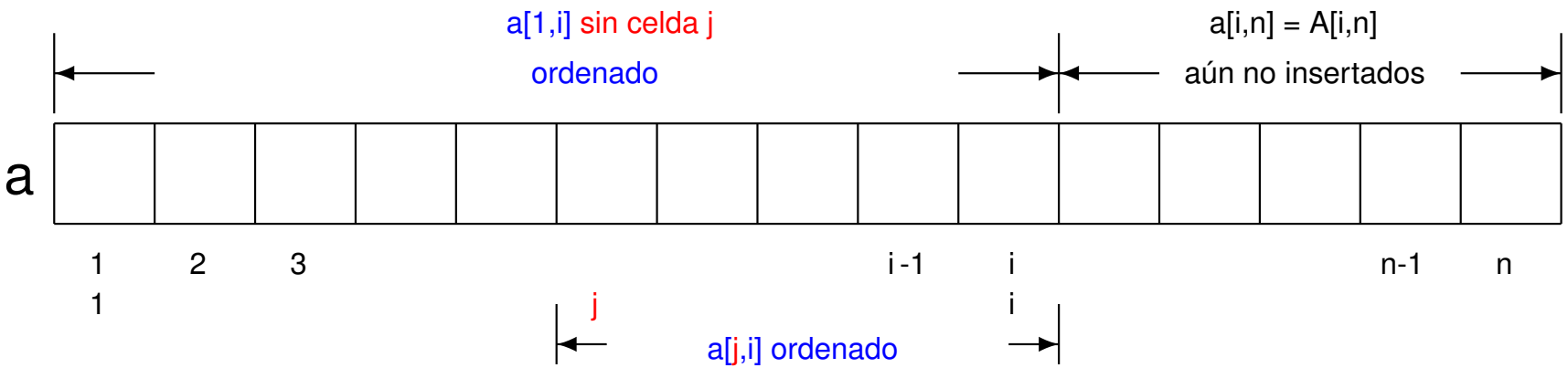
Ordenación por inserción

Pseudocódigo

{Pre: n $\geq$ 0 $\wedge$ a = A}
proc insertion_sort (**in/out** a: **array**[1..n] **of T**)
    for i:= 2 **to** n **do** {Inv: Invariante de recién}
        insert(a,i)
    od
end proc
{Post: a está ordenado y es permutación de A}

Ordenación por selección

Invariante del procedimiento de inserción



- Invariante:
  - el arreglo a es una permutación del original
  - a[1,i] sin celda j está ordenado, y
  - a[j,i] también está ordenado.

Ordenación por inserción

Procedimiento de inserción

{Pre: 0 < i $\leq$ n $\wedge$ a = A}
proc insert (**in**/**out** a: **array**[1..n] **of T**, **in** i: **nat**)
    var j: **nat**
    j:= i   {Inv: Invariante de recién}
    do $j > 1 \wedge a[j] < a[j-1] \rightarrow$ swap(a,j-1,j)
                                         j:= j-1
    od
end proc
{Post: a[1,i] está ordenado $\wedge$ a es permutación de A}

Ordenación por inserción

Todo junto

```
proc insertion_sort (in/out a: array[1..n] of T)
        for i:= 2 to n do
            insert(a,i)
        od
end proc

proc insert (in/out a: array[1..n] of T, in i: nat)
        j:= i
        do j > 1 ∧ a[j] < a[j − 1] → swap(a,j-1,j)
                                     j:= j-1
        od
end proc
```

Número de Comparaciones e intercambios

Procedimiento insert(a,i)

	comparaciones		intercambios	
i	mín	máx	mín	máx
2	1	1	0	1
3	1	2	0	2
4	1	3	0	3
⋮	⋮	⋮	⋮	⋮
n	1	n-1	0	n-1
total insertion_sort	n - 1	$\frac{n^2}{2} - \frac{n}{2}$	0	$\frac{n^2}{2} - \frac{n}{2}$

Ordenación por inserción, casos

- mejor caso: arreglo ordenado, n comparaciones y 0 intercambios.
- peor caso: arreglo ordenado al revés, $\frac{n^2}{2} - \frac{n}{2}$ comparaciones e intercambios, es decir, del orden de n^2.
- caso promedio: del orden de n^2.

Resumen

- Hemos analizado dos algoritmos de ordenación
  - ordenación por selección
  - ordenación por inserción
- la ordenación por selección hace siempre el mismo número de comparaciones, del orden de n^2.
- la ordenación por inserción también es del orden de n^2 en el peor caso (arreglo ordenado al revés) y en el caso medio,
- la ordenación por inserción es del orden de n en el mejor caso (arreglo ordenado),
- la ordenación por inserción realiza del orden de n^2 swaps (contra n de la ordenación por selección) en el peor caso.

Problema del bibliotecario

- Con cualquiera de los dos algoritmos la respuesta es 4 días,
- salvo que se trate de una biblioteca ya ordenada o casi ordenada, en cuyo caso:
  - ordenación por inserción es del orden de n,
  - y por ello la respuesta sería: 2 días.

Repaso de la ordenación por selección

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j:= i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

Se lo puede abreviar omitiendo la función auxiliar.

Forma abreviada de la ordenación por selección

```
proc selection_sort (in/out a: array[1..n] of T)
    var minp: nat
    for i:= 1 to n-1 do
        minp:= i
        for j:= i+1 to n do
            if a[j] < a[minp] then minp:= j fi
        od
        swap(a,i,minp)
    od
end proc
```

Repaso de la ordenación por inserción

```
proc insertion_sort (in/out a: array[1..n] of T)
        for i:= 2 to n do
            insert(a,i)
        od
end proc

proc insert (in/out a: array[1..n] of T, in i: nat)
        j:= i
        do j > 1 ∧ a[j] < a[j − 1] → swap(a,j-1,j)
                                     j:= j-1
        od
end proc
```

También puede abreviarse omitiendo el procedimiento auxiliar.

Forma abreviada de la ordenación por inserción

```
proc insertion_sort (in/out a: array[1..n] of T)
        for i:= 2 to n do
            j:= i
            do j > 1 ∧ a[j] < a[j − 1] → swap(a,j-1,j)
                                         j:= j-1
            od
        od
end proc
```

Demo (www.sorting-algorithms.com)

- Ejecución de ordenación por selección
  - entrada aleatoria
  - casi ordenada
  - invertida
  - con repeticiones
- Ejecución de ordenación por inserción
  - entrada aleatoria
  - casi ordenada
  - invertida
  - con repeticiones
- Comparación y conclusiones.

Reflexión sobre paralelismo

¿Qué provecho podríamos sacar a los algoritmos que hemos visto si contáramos con varios o muchos procesadores capaces de cooperar entre ellos?