Algoritmos y Estructuras de Datos II

Ordenación

March 11, 2013

Contenidos

Algoritmos y Estructuras de Datos

- Introducción a los Algoritmos
  Algoritmos y Estructuras de Datos I
  - pre- y post- condiciones
  - “qué” hace un algoritmo
- Algoritmos y Estructuras de Datos II
  - “cómo” hace el algoritmo

Ejemplo de “qué” y “cómo”

- “Qué”: cuenta las letras ’a’ de un arreglo.
- “Cómo”: recorre el arreglo de izquierda a derecha incrementando un contador cada vez que lee ’a’.

Análisis de algoritmos

Analizar el “cómo” permite

- predecir el tiempo de ejecución (eficiencia en tiempo)
- predecir el uso de memoria (eficiencia en espacio)
- predecir el uso de otros recursos
- comparar distintos algoritmos para un mismo problema

Problema del pintor

Un pintor tarda una hora y media en pintar una pared de 3 metros de largo. ¿Cuánto tardará en pintar una de 5 metros de largo?

$$\begin{aligned} 3 \text{ metros} &\longleftrightarrow 90 \text{ minutos} \\ 1 \text{ metro} &\longleftrightarrow 30 \text{ minutos} \\ 5 \text{ metros} &\longleftrightarrow 150 \text{ minutos} \end{aligned}$$

Solución: dos horas y media.
El trabajo de pintar la pared es **proporcional** a su longitud.

Problema del bibliotecario

Un bibliotecario tarda un día en ordenar alfabéticamente una biblioteca con 1000 expedientes. ¿Cuánto tardará en ordenar una con 2000 expedientes?

Razonamiento similar

$$\begin{array}{rcl} \text{1000 expedientes} & \longleftrightarrow & \text{1 día} \\ \text{2000 expedientes} & \longleftrightarrow & \text{2 días} \end{array}$$

Solución: dos días.
¿Es el trabajo de ordenar expedientes **proporcional** a la cantidad?

Otros problemas
del pintor

Un pintor tarda una hora y media en pintar una pared ***cuadrada*** *de 3 metros de lado. ¿Cuánto tardará en pintar una de 5 metros de lado?*

$$\begin{aligned} 9 \text{ metros cuadrados} &\longleftrightarrow 90 \text{ minutos} \\ 1 \text{ metro cuadrado} &\longleftrightarrow 10 \text{ minutos} \\ 25 \text{ metros cuadrados} &\longleftrightarrow 250 \text{ minutos} \end{aligned}$$

Solución: cuatro horas y 10 minutos.
El trabajo de pintar la pared cuadrada es **proporcional** a su superficie, que es proporcional al cuadrado del lado.

Otros problemas
el del globo esférico

Si lleva cinco horas inflar un globo aerostático esférico de 2 metros de diámetro, ¿cuánto llevará inflar uno de 4 metros de diámetro?

El trabajo de inflar el globo es **proporcional** a su volumen, que es proporcional al cubo del diámetro ($V = \frac{\pi d^3}{6}$).

$$\begin{array}{rcrcr} \text{diámetro} = 2 & \longleftrightarrow & k \text{ metros cúbicos} & \longleftrightarrow & 5 \text{ horas} \\ \text{diámetro} = 4 & \longleftrightarrow & 8k \text{ metros cúbicos} & \longleftrightarrow & 40 \text{ horas} \end{array}$$

Solución: cuarenta horas.

Algoritmos de ordenación

Para resolver el problema del bibliotecario, es necesario

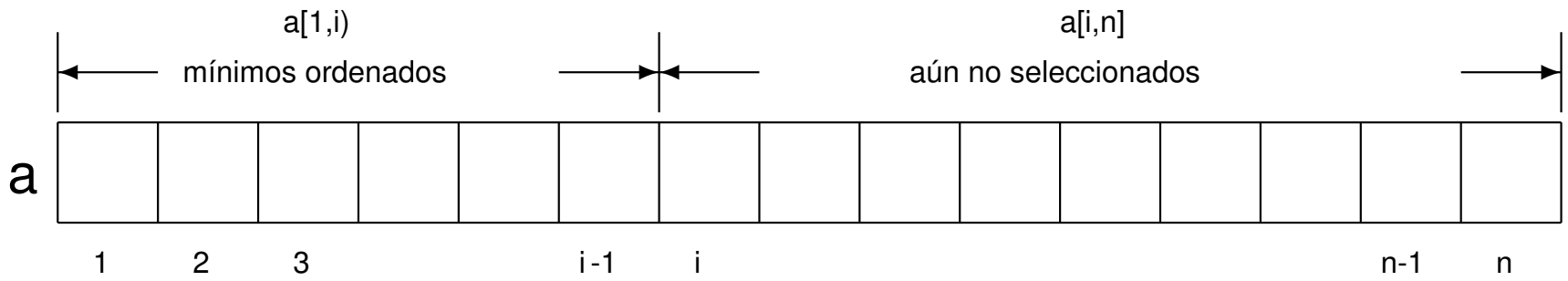
- establecer a qué es proporcional la tarea de ordenar expedientes,
- estudiar métodos de ordenación,
- asumiremos la existencia de elementos o items a ordenar,
- relacionados por un orden total,
- que deben ordenarse de menor a mayor y
- que no necesariamente son diferentes entre sí.

Ordenación por selección
Idea

- Es el algoritmo de ordenación más sencillo (pero no el más rápido),
- selecciona el menor de todos, lo coloca en el primer lugar apartándolo del resto,
- selecciona el menor de todos, lo coloca en el segundo lugar apartándolo del resto,
- selecciona el menor de todos, lo coloca en el tercer lugar apartándolo del resto,
- ...
- hasta terminar.

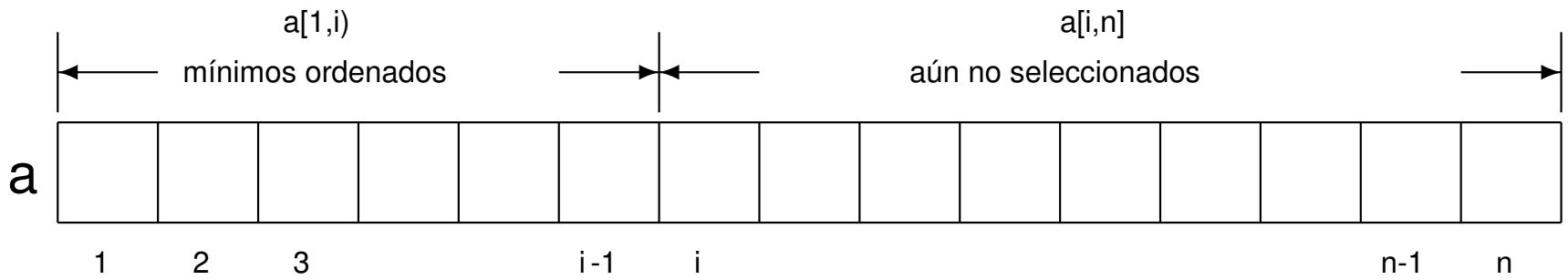
Ordenación por selección

En un arreglo



Ordenación por selección

Invariante



- Invariante:
  - el arreglo a es una permutación del original,
  - un segmento inicial a[1,i) del arreglo está ordenado, y
  - dicho segmento contiene los elementos mínimos del arreglo.

Ordenación por selección

Pseudocódigo

{Pre: $n \geq 0 \wedge a = A$}
proc selection_sort (**in**/**out** a: **array**[1..n] **of T**)
    var i, minp: **nat**
    i:= 1 {Invariante}
    do $i < n \rightarrow$ minp:= min_pos_from(a,i)
        swap(a,i,minp)
        i:= i+1
    od
end proc
{Post: a está ordenado y es permutación de A}

Ordenación por selección

Swap o intercambio

{Pre: a = A $\wedge$ 1 $\leq$ i,j $\leq$ n }
proc swap (**in**/**out** a: **array**[1..n] **of T**, **in** i,j: **nat**)
    var tmp: **T**
    tmp:= a[i]
    a[i]:= a[j]
    a[j]:= tmp
end proc
{Post: a[i] = A[j] $\wedge$ a[j] = A[i] $\wedge$ $\forall$ k. k $\notin${i,j} $\Rightarrow$ a[k]=A[k]}

¡Garantiza permutación!

Ordenación por selección

Función de selección

{Pre: $0 < i \leq n$}
fun min_pos_from (a: **array**[1..n] **of T**, i: **nat**) **ret** minp: **nat**
    var j: **nat**
    minp:= i
    j:= i+1 {Inv: a[minp] es el mínimo de a[i,j)}
    do $j \leq n \rightarrow$ **if** $a[j] < a[minp]$ **then** minp:= j **fi**
        j:= j+1
    od
end fun
{Post: a[minp] es el mínimo de a[i,n]}

Ordenación por selección

Ejemplo

	7	4	1	3	5	2	6
	1	4	7	3	5	2	6
	1	2	7	3	5	4	6
	1	2	3	7	5	4	6
	1	2	3	4	5	7	6
	1	2	3	4	5	7	6
	1	2	3	4	5	6	7
	1	2	3	4	5	6	7

Comando **for**

k:= n
do k $\leq$ m $\rightarrow$ C
k:= k+1
od

escribiremos

for k:= n **to** m **do** C **od**

siempre que k no se modifique en C.

Comando **for**
En min_pos_from

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

```
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
```

Comando **for**
En selection_sort

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

Comando **for**
En selection_sort

```
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
```

Problema del bibliotecario

- ¿Cómo se respondería el problema del bibliotecario si el algoritmo utilizado por él fuera el de ordenación por selección?
- ¿Cuánto más trabajo resulta ordenar 2000 expedientes que 1000 con este algoritmo?
- ¿Cuánto trabajo es ordenar 2000 expedientes?
- ¿Cuánto trabajo es ordenar 1000 expedientes?
- ¿Cuánto trabajo es ordenar n expedientes?

Problema del bibliotecario

- Para contestar estas preguntas habría que contar cuántas operaciones elementales realiza.
- Cuantas sumas, asignaciones, llamadas a funciones, comparaciones, intercambios, etc.
- En vez de eso, se elige una operación suficientemente representativa.
- ¿Qué es una operación representativa?
- Una tal que se repite más o tanto como cualquier otra.
- Hay que buscar la que más se repite.

Analizando ordenación por selección

- selection_sort contiene un ciclo,
- allí debe estar la operación que más se repite,
- encontramos una llamada a la función min_pos_from y una llamada al procedimiento swap,
- el procedimiento swap es constante (siempre realiza 3 asignaciones elementales),
- la función min_pos_from, en cambio, tiene un ciclo,
- allí debe estar la operación que más se repite,
- encontramos una comparación entre elementos de a, y una asignación.

Analizando ordenación por selección
Conclusión

- La operación que más se repite es la comparación entre elementos de a,
- toda otra operación se repite a lo sumo de manera proporcional a esa,
- por lo tanto, se toma esa operación como representativa del trabajo de la ordenación por selección.
- Para medir la eficiencia de los algoritmos de ordenación es habitual considerar el número de comparaciones entre elementos del arreglo.

¿Cuántas comparaciones realiza la ordenación por selección?

- Al llamarse a min_pos_from(a,i) se realizan n-i comparaciones.
- selection_sort llama a min_pos_from(a,i) para $i \in \{1, 2, \ldots, n-1\}$.
- por lo tanto, en total son (n-1) + (n-2) + . . . + (n-(n-1)) comparaciones.
- es decir, (n-1) + (n-2) + . . . + 1 = $\frac{n*(n-1)}{2}$ comparaciones.

Resolviendo el problema del bibliotecario

Para un arreglo de tamaño n, son $\frac{n*(n-1)}{2}$ comparaciones.

1000 expedientes	$\longleftrightarrow$	499500 comparaciones	$\longleftrightarrow$	1 día
2000 expedientes	$\longleftrightarrow$	1999000 comparaciones	$\longleftrightarrow$	4 días

Solución: 4 días.

Resolviendo el problema del bibliotecario

Como $\frac{n*(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2}$, el número de comparaciones es proporcional n^2.

1000 expedientes	$\longleftrightarrow$	1000000 comparaciones	$\longleftrightarrow$	1 día
2000 expedientes	$\longleftrightarrow$	4000000 comparaciones	$\longleftrightarrow$	4 días

Solución: 4 días.
Conviene utilizar la expresión n^2 para contestar la pregunta; es más sencilla y da el mismo resultado.

Número de operaciones de un comando

$$\begin{aligned}
\text{ops}(C_1;C_2;\ldots;C_n) &= \text{ops}(C_1) + \text{ops}(C_2) + \ldots + \text{ops}(C_n) \\
&= \sum_{i=1}^{n} \text{ops}(C_i) \\
\text{ops}(\textbf{skip}) &= 0 \\
\text{ops}(\textbf{for k:= n to m do C(k) od}) &= \sum_{k=n}^{m} \text{ops}(C(k)) \\
\text{ops}(\textbf{if b then C else D fi}) &= \begin{cases} \text{ops(b)+ops(C)} & b = \text{V} \\ \text{ops(b)+ops(D)} & b = \text{F} \end{cases}
\end{aligned}$$

La ecuación del **for** solamente vale cuando no se quieren contar las operaciones que involucran el índice k: inicialización, comparación con la cota m, incremento; ni el cómputo de los límites n y m. En los apuntes hay otras ecuaciones posibles para el caso en que sí quieran contarse.

Número de operaciones de un comando o expresión

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

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

Número de operaciones de un 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

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

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

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.

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.

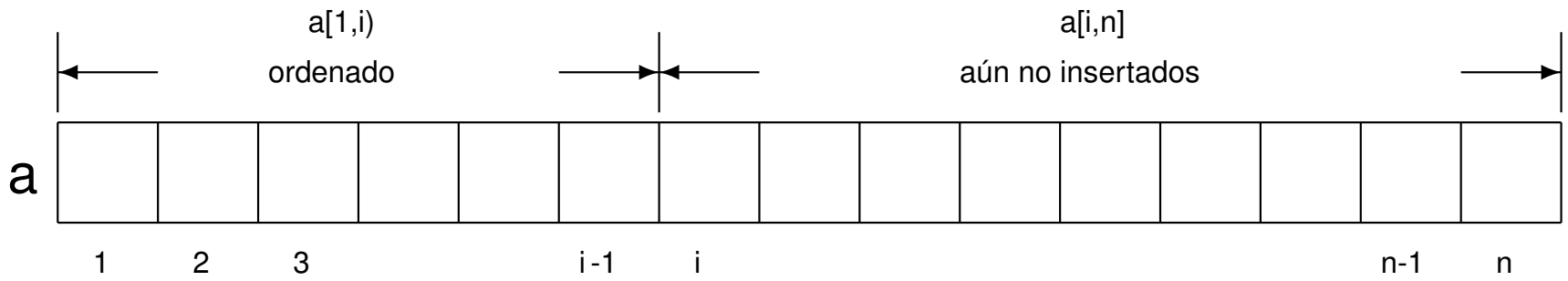
Ordenación por inserción

Ejemplo

	7	4	1	3	5	2	6
	4	7	1	3	5	2	6
	4	1	7	3	5	2	6
	1	4	7	3	5	2	6
	1	4	3	7	5	2	6
	1	3	4	7	5	2	6
	1	3	4	5	7	2	6
	1	3	4	5	2	7	6
	1	3	4	2	5	7	6
	1	3	2	4	5	7	6
	1	2	3	4	5	7	6
	1	2	3	4	5	6	7

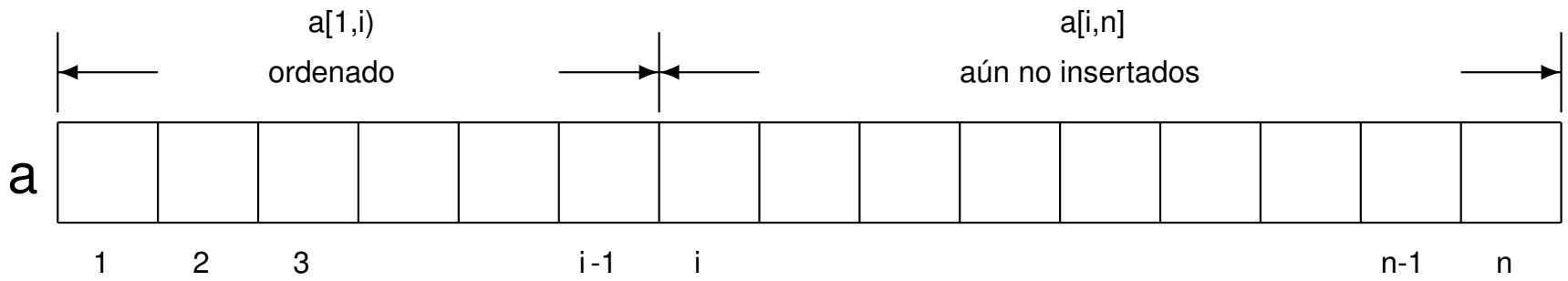
Ordenación por inserción

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

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**
        {Invariante: a[1,i) está ordenado $\wedge$ a es permutación de A}
        insert(a,i)
    od
end proc
{Post: a está ordenado y es permutación de A}

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**) **ret**
    j:= i
    {Inv: $a[1_{\not j} i]$ y a[j,i] están ordenados $\wedge$ a es permutación de A}
    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}

$a[1_{\not j} i]$ es el segmento a[1,i] sin el elemento de la celda j.

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) ret
        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)

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

Ordenación por inserción, casos

- mejor caso: arreglo ordenado, n comparaciones y 0 swaps.
- peor caso: arreglo ordenado al revés, $\frac{n^2}{2} - \frac{n}{2}$ comparaciones y swaps, 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
- ambos son del orden de n^2 en el peor caso,
- ordenación por inserción es del orden de n en el mejor caso,
- 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 casi ordenada, en cuyo caso:
  - ordenación por inserción es del orden de n, y la respuesta es 2 días.