Algoritmos y Estructuras de Datos II

Ordenación elemental

11 y 13 de marzo de 2019

Contenidos

Generalidades

Toda la información sobre la materia se encuentra en la wiki, accesible desde

wiki.cs.famaf.unc.edu.ar

Algoritmos y Estructuras de Datos

Programación imperativa:

- 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" de un algoritmo

Ejemplo:

un algoritmo para contar los ceros de una secuencia finita de enteros.

- ¿**Qué** hace?

  devuelve (calcula, computa) el número de ocurrencias del cero en la secuencia dada.

- ¿**Cómo** lo hace? Hay varias posibilidades, por ejemplo:

  recorre la secuencia de izquierda a derecha incrementando un contador cada vez que observa un cero.

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

Organización de la materia

La materia está organizada en tres partes:

- Análisis de algoritmos.
  - Cómo se ejecutan los algoritmos y estimar cuánto trabajo realiza.
- Estructuras de datos.
  - Tipos de datos concretos y abstractos.
- Algoritmos avanzados.
  - Algunas técnicas para resolver problemas algorítmicos.

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?

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

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

Problema del profe de Algoritmos 2

El profe de esta materia tarda media hora en ordenar alfabéticamente 100 exámenes. ¿Cuánto tardará en ordenar 200 exámenes?

Razonamiento similar

$$\begin{array}{rcr} \text{100 exámenes} & \longleftrightarrow & 1/2 \text{ hora} \\ \text{200 exámenes} & \longleftrightarrow & 1 \text{ hora} \end{array}$$

Solución: una hora.
¿Está bien? ¿Es el trabajo de ordenar exámenes **proporcional** a la cantidad de exámenes a ordenar?

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?*

9 metros cuadrados	$\longleftrightarrow$	90 minutos
1 metro cuadrado	$\longleftrightarrow$	10 minutos
25 metros cuadrados	$\longleftrightarrow$	250 minutos

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 ($V = \frac{\pi d^3}{6}$).

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

Solución: cuarenta horas.

Algoritmos de ordenación

Para resolver el problema del profe de esta materia, es necesario

- establecer a qué es proporcional la tarea de ordenar exámenes,
- estudiar/inventar 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.

¿Cómo?

Reflexionemos sobre lo siguiente:

- ¿Qué significa que una secuencia de exámenes, números, palabras, etc. esté ordenada?
- ¿Cómo hacen para controlar si una secuencia de números está ordenada?
  - (a esta pregunta la vamos a continuar en el práctico y en el laboratorio)
- ¿Cómo harían para ordenar de menor a mayor ciertos datos o ciertas cosas físicas que están desordenados/as?
  - números
  - cartas de un juego,
  - palabras,
  - exámenes.

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 **los restantes**, lo coloca en el segundo lugar apartándolo del resto,
- **selecciona** el menor de todos **los restantes**, lo coloca en el tercer lugar apartándolo del resto,
- …(*en cada uno de estos pasos ordena un elemento*) …
- hasta terminar.

Ordenación por selección
Ejemplo

- ssort.ejemplo
- ssort.hs
- cronometrar su ejecución con las secuencias
  - 5000..1
  - 10000..1

  y comparar los tiempos obtenidos.

Ordenación por selección

Ejemplo: ahora con arreglos

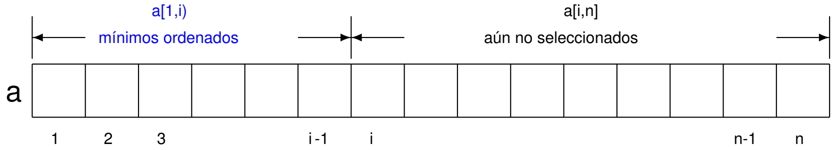
9	3	1	3	5	2	7
9	3	1	3	5	2	7
1	3	9	3	5	2	7
1	3	9	3	5	2	7
1	2	9	3	5	3	7
1	2	9	3	5	3	7
1	2	3	9	5	3	7

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

Demo (www.sorting-algorithms.com)

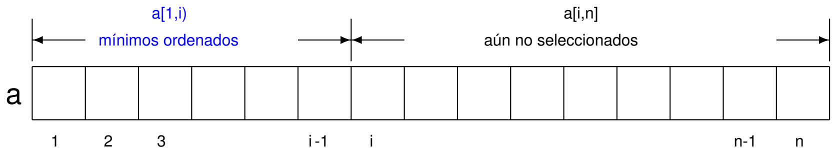
Ordenación por selección

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 ≥ 0 ∧ a = A}
proc selection_sort (in/out a: array[1..n] of T)
      var i, minp: nat
      i:= 1                                   {Inv: Invariante de recién}
      do i < n → 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

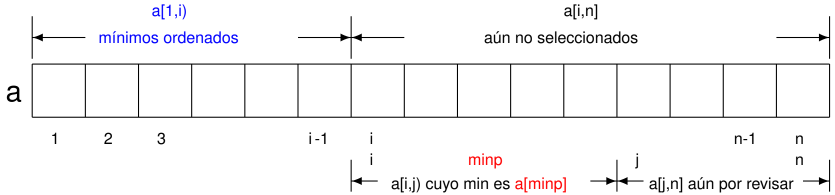
Swap o intercambio

{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

Invariante de la función de selección



- Invariante:
  - invariante anterior, y
  - el mínimo del segmento a[i,j) está en la posición minp.

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]}

Comando **for**

Fragmentos de la siguiente forma aparecen con frecuencia:

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

Por simplicidad, lo reemplazaremos por

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

siempre que k no se modifique en C.
Además, asumiremos que el **for** declara la variable k, cuya vida dura sólo durante la ejecución del ciclo.

Comando **for**

Reemplazo 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 $j \leq n \rightarrow$ **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
end fun

Comando **for**

Reemplazo en selection_sort

proc selection_sort (**in**/**out** a: **array**[1..n] **of T**)
    var i, minp: **nat**
    i:= 1
    do i $<$ n $\rightarrow$ 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

Comando **for . . . downto**

Fragmentos de la siguiente forma también aparecen con cierta frecuencia:

k:= m
do k $\geq$ n $\rightarrow$ C
k:= k-1
od

Por simplicidad, lo reemplazaremos por

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

siempre que k no se modifique en C.

Ordenación por selección en lenguajes de programación

- En python.
- En c.

Problema del profe

Cuando el algoritmo es la ordenación por selección

- ¿Cómo se respondería el problema del profe si el algoritmo utilizado por él fuera el de ordenación por selección?
- ¿Cuánto más trabajo resulta ordenar 200 exámenes que 100 con este algoritmo?
- ¿Cuánto trabajo es ordenar 200 exámenes (con este algoritmo)?
- ¿Cuánto trabajo es ordenar 100 exámenes (con este algoritmo)?
- ¿Cuánto trabajo es ordenar n exámenes (con este algoritmo)?

Problema del profe

Análisis

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

Analizando el procedimiento selection_sort

- 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**,
- nuevamente **allí** debe estar la operación que más se repite,
- encontramos **una comparación** entre elementos de a, y **una asignación** (condicionada al resultado de la comparación).

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, **la comparación** entre elementos de a **es representativa** del trabajo de la ordenación por selección.
- Esto es habitual: para medir la eficiencia de los algoritmos de ordenación es habitual considerar el número de comparaciones entre elementos del arreglo.
- Veremos luego que acceder (o modificar) una celda de un arreglo es **constante**: su costo no depende de cuál es la celda, ni de la longitud 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 $i \in \{1, 2, \dots, n-1\}$.
- 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 profe

Con la fórmula obtenida

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

100 exámenes	$\longleftrightarrow$	4950 comparaciones	$\longleftrightarrow$	1/2 hora
200 exámenes	$\longleftrightarrow$	19900 comparaciones	$\longleftrightarrow$	2 horas

Solución: 2 horas.

Resolviendo el problema del profe

Con una fórmula simplificada

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

100 exámenes	$\longleftrightarrow$	10000 comparaciones	$\longleftrightarrow$	1/2 hora
200 exámenes	$\longleftrightarrow$	40000 comparaciones	$\longleftrightarrow$	2 horas

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

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.
- La secuencia se puede escribir horizontalmente: $C_1;C_2;\ldots;C_n$
- o verticalmente

$$\begin{array}{l} C_1 \\ C_2 \\ \vdots \\ C_n \end{array}$$

Número de operaciones de un comando

Secuencia de comandos

- 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:
- $\text{ops}(C_1;C_2;\ldots;C_n) = \text{ops}(C_1) + \text{ops}(C_2) + \ldots + \text{ops}(C_n)$
- $\text{ops}\begin{pmatrix} C_1 \\ C_2 \\ \vdots \\ C_n \end{pmatrix} = \text{ops}(C_1) + \text{ops}(C_2) + \ldots + \text{ops}(C_n)$

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

$$\begin{array}{l} C(n) \\ C(n+1) \\ \vdots \\ C(m) \end{array}$$

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

- De esta “equivalencia” resulta

$$\begin{aligned}&\text{ops}(\textbf{for } \text{k:= n } \textbf{to} \text{ m } \textbf{do} \text{ C(k) } \textbf{od}) = \\ &= \text{ops}(\text{C(n)}) + \text{ops}(\text{C(n+1)}) + \ldots + \text{ops}(\text{C(m)})\end{aligned}$$

- que también se puede escribir

$$\text{ops}(\textbf{for } \text{k:= n } \textbf{to} \text{ m } \textbf{do} \text{ C(k) } \textbf{od}) = \sum_{\text{k=n}}^{\text{m}} \text{ops}(\text{C(k)})$$

Número de operaciones de un comando

Comando **for** (una salvedad importante)

La ecuación

$$\text{ops}(\textbf{for}\ \text{k:= n}\ \textbf{to}\ \text{m}\ \textbf{do}\ \text{C(k)}\ \textbf{od}) = \sum_{k=n}^{m} \text{ops}(\text{C(k)})$$

solamente vale cuando **no hay interés en 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.

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
- ops(**if** b **then** C **else** D **fi**) $= \begin{cases} \text{ops(b)+ops(C)} & \text{caso } b \text{ V} \\ \text{ops(b)+ops(D)} & \text{caso } b \text{ F} \end{cases}$

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.
- $$\text{ops(x:=e)} = \begin{cases} \text{ops(e)+1} & \text{si se desea contar la asignación} \\ & \text{o las modificaciones de memoria} \\ \text{ops(e)} & \text{en caso contrario} \end{cases}$$
- Tener en cuenta que la evaluación de e puede implicar la llamada a funciones auxiliares cuyas operaciones deben ser también contadas.

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.
-

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

Ejemplo: número de comparaciones 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
```

Ejemplo: número de comparaciones de la ordenación por selección

$$
\begin{array}{ll}
 & \text{ops(selection\_sort(a))} \\
= & \text{ops(}\textbf{for}\text{ i:= 1 }\textbf{to}\text{ n-1 }\textbf{do}\text{ minp:= min\_pos\_fr}\ldots\text{;swap}\ldots\textbf{od}\text{)} \\
= & \sum_{\text{i=1}}^{\text{n-1}} \text{ops(minp:= min\_pos\_from(a,i);swap(a,i,minp))} \\
= & \sum_{\text{i=1}}^{\text{n-1}} \text{(ops(minp:= min\_pos\_from(a,i))} + \text{ops(swap(a,i,minp)))} \\
= & \sum_{\text{i=1}}^{\text{n-1}} \text{ops(minp:= min\_pos\_from(a,i))} \\
= & \sum_{\text{i=1}}^{\text{n-1}} \text{ops(min\_pos\_from(a,i))} \\
= & \sum_{\text{i=1}}^{\text{n-1}} \text{ops(minp:= i;}\textbf{for}\text{ j:= i+1 }\textbf{to}\text{ n }\textbf{do if}\ldots\textbf{fi od}\text{)}
\end{array}
$$

Ejemplo: número de comparaciones de la ordenación por selección

$$
\begin{aligned}
& \text{ops(selection\_sort(a))} \\
= & \sum_{\text{i=1}}^{\text{n-1}} \text{ops(minp:= i;}\textbf{for}\text{ j:= i+1 }\textbf{to}\text{ n }\textbf{do if} \ldots \textbf{fi od}\text{)} \\
= & \sum_{\text{i=1}}^{\text{n-1}} \text{(ops(minp:= i)} + \text{ops(}\textbf{for}\text{ j:= i+1 }\textbf{to}\text{ n }\textbf{do if} \ldots \textbf{fi od}\text{))} \\
= & \sum_{\text{i=1}}^{\text{n-1}} \text{ops(}\textbf{for}\text{ j:= i+1 }\textbf{to}\text{ n }\textbf{do if} \ldots \textbf{fi od}\text{)} \\
= & \sum_{\text{i=1}}^{\text{n-1}} \sum_{\text{j=i+1}}^{\text{n}} \text{ops(}\textbf{if}\text{ a[j] < a[minp] }\textbf{then}\text{ minp:= j }\textbf{if}\text{)} \\
= & \sum_{\text{i=1}}^{\text{n-1}} \sum_{\text{j=i+1}}^{\text{n}} \text{(ops(a[j]} < \text{a[minp])} + \text{ops(minp:= j)) o ops(}\textbf{skip}\text{)} \\
= & \sum_{\text{i=1}}^{\text{n-1}} \sum_{\text{j=i+1}}^{\text{n}} \text{ops(a[j]} < \text{a[minp])}
\end{aligned}
$$

Ejemplo: número de comparaciones de la ordenación por selección

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