Algorítmica y Lenguajes de Programación

Ordenación (ii)

Ordenación. Introducción

- En la lección anterior se vieron dos métodos de ordenación:
  - Método de la burbuja.
  - Método de la burbuja con señal.
- El primero de ellos tenía una complejidad de $O(n^2)$ para todos los casos.
- El segundo tenía una complejidad $O(n^2)$ para todos los casos excepto el mejor, vector ordenado; en éste la complejidad era O(n).
- **Hoy estudiaremos otros dos algoritmos:**
  - **El método de la sacudida.**
  - **El método rápido (o *quicksort*).**

Ordenación. Método de la sacudida (i)

- El método de la sacudida es una variación del método de la burbuja.
- En cada iteración se realizan dos etapas:
  - En la primera etapa (de derecha a izquierda) se trasladan los elementos más pequeños hacia la parte izquierda del vector, almacenando en una variable la posición del último elemento intercambiado.
  - En la segunda (de izquierda a derecha) se trasladan los elementos más grandes hacia la parte derecha del vector, almacenando en otra variable la posición del último elemento intercambiado.
- Los nuevos barridos se hacen entre los límites marcados por ambas variables.
- El algoritmo termina:
  - Cuando en una etapa no se producen intercambios, o bien…
  - Cuando el contenido de la variable "izquierda" ha sobrepasado a la variable "derecha".
- A continuación se muestra el algoritmo de este método.

Ordenación. Método de la sacudida (ii)

```
subrutina sacudida (v Î vector(MAX) de elemento)
variables
  i,izq,dcha,k Î entero
  temp Î elemento

inicio
  izq←2
  dcha←MAX
  k←MAX

  repetir
    desde i←dcha hasta izq con paso -1 hacer
      si v(i-1)>v(i) entonces
        temp←v(i-1)
        v(i-1)←v(i)
        v(i)←temp
        k←i
      fin si
    fin desde
    izq←k+1

    desde i←izq hasta dcha hacer
      si v(i-1)>v(i) entonces
        temp←v(i-1)
        v(i-1)←v(i)
        v(i)←temp
        k←i
      fin si
    fin desde
    dcha←k-1
  hasta que izq>dcha
fin
```

Ordenación. Método de la sacudida (iii)

- **La complejidad del método de la sacudida es $O(n^2)$ en el peor caso.**
- El bucle más externo se ejecutará, como máximo, n veces y en el interior del mismo hay dos bucles con complejidad O(n).
- **Sin embargo, el método de la sacudida se aproxima a O(n) para los casos intermedios.**
- Una buena forma de ver la diferente eficiencia de los métodos de la burbuja, sacudida y rápido es a través de la página web:

http://java.sun.com/applets/jdk/1.0/demo/SortDemo/example1.html

5

Ordenación. Método de la sacudida (iv)

```
subroutine sacudida (vector)
implicit none
  integer vector(max)
  integer i,izq,dcha,k
  integer temp

  izq=2
  dcha=max
  k=max

  do while (izq<=dcha)
    do i=dcha,izq,-1
       if (vector(i-1)>vector(i)) then
         temp=vector(i-1)
              vector(i-1)=vector(i)
              vector(i)=temp
              k=i
       end if
     end do
     izq=k+1

    do i=izq,dcha
       if (vector(i-1)>vector(i)) then
         temp=vector(i-1)
              vector(i-1)=vector(i)
              vector(i)=temp
              k=i
       end if
     end do
     dcha=k-1
  end do
end subroutine
```

6

Ordenación. Método *quicksort* (i)

- El método rápido de ordenación es un claro ejemplo de los algoritmos “divide y vencerás” (que serán vistos en lecciones posteriores).
- Básicamente un algoritmo divide y vencerás se limita a resolver subproblemas más sencillos e integrar las soluciones parciales hasta obtener una solución global.
- En el caso de la ordenación el algoritmo básico sería el siguiente:

```
si el vector tiene tamaño 1 entonces
  el vector ya está ordenado
si no
  dividir el vector en dos vectores A y B tales que los elementos de A
   sean menores que los elementos de B

  ordenar A y B mediante quicksort

  retornar la concatenación de A y B (el vector ordenado)
fin
```


Ordenación. Método *quicksort* (ii)

```
acción quicksort (v Î vector(MAX) de
elemento, izq,dcha Î entero)
  i,j Î entero
  pivote, temp Î elemento

  i←izq
  j←dcha
  pivote←v((izq+dcha)/2)

  mientras(i<j)
    mientras (pivote>v(i))
      i←i+1
    fin mientras
    mientras (pivote<v(j))
     j←j-1
    fin mientras
    si (i<j) entonces
      temp←v(i)
      v(i)←v(j)
      v(j)←temp
    fin si
    si (i<dcha) entonces
      i←i+1
    fin si
    si (j>izq) entonces
      j←j-1
    fin si
  fin mientras

  si i=j y i=(izq+dcha)/2 entonces
    i←i+1
    j←j+1
  fin si

  si (izq<j) entonces
    llamar quicksort(v,izq,j)
  fin si
  si (dcha>i) entonces
    llamar quicksort(v,i,dcha)
  fin si
fin acción
```

Ordenación. Método *quicksort* (iii)

```
recursive subroutine quicksort (vector,izq,dcha)
  implicit none

  integer vector(max)
  integer izq,dcha
  integer i,j
  integer pivote,temp

  i=izq
  j=dcha
  pivote=vector((izq+dcha)/2)
  do while (i<j)
    do while (pivote>vector(i))
      i=i+1
    end do

    do while (pivote<vector(j))
      j=j-1
    end do

[Continúa...]
```

9

Ordenación. Método *quicksort* (iv)

```
[Continúa...]
    if (i < j) then
      temp=vector(i)
      vector(i)=vector(j)
      vector(j)=temp
      if (i<dcha) then
        i=i+1
      end if
      if (j>izq) then
        j=j-1
      end if
    end if
  end do
  if ((i==j).and.(i==(izq+dcha)/2)) then
    i=i+1
    j=j-1
  end if
  if (izq<j) then
    call quicksort(vector,izq,j)
  end if
  if (dcha>i) then
    call quicksort(vector,i,dcha)
  end if
end subroutine
```

10

Ordenación. Método *quicksort* (v)

- Como se puede ver, el algoritmo es bastante complicado... Debe ser muy eficiente entonces para merecer el esfuerzo...
- En el **caso peor** el pivote siempre es el mayor (o el menor) de todos los elementos del vector y en cada llamada recursiva el vector es dividido en una parte que contiene todos los elementos del vector menos el pivote y otra vacía. En esta situación la complejidad del algoritmo es **$O(n^2)$.**
- En el **caso mejor** el pivote es siempre la media de todos los elementos; de esta forma el vector se divide en dos partes equilibradas siendo la complejidad del algoritmo **$O(n \cdot \log n)$**.
- El **caso medio** es bastante difícil de probar de una manera formal pero puede apreciarse de forma intuitiva que es muy poco probable que cada vez que se tome el pivote, éste sea el máximo o el mínimo; por tanto estará más o menos próximo a la media y se dividirá el vector en dos partes aproximadamente iguales. Así, la complejidad para el caso medio también es **$O(n \cdot \log n)$**.

11

Ordenación. Resumen

- Hoy hemos estudiado dos nuevos algoritmos de ordenación:
  - Método de la sacudida.
  - Método rápido (quicksort).
- El primero es una variación del método de la burbuja que, como éste, tiene una complejidad de **$O(n^2)$** para el caso peor aunque en los casos promedios tiende hacia **$O(n)$**.
- El segundo es el método de ordenación más utilizado, se trata de un algoritmo de los denominados “divide y vencerás” que tiene para los casos mejor y medio una complejidad **$O(n \cdot \log n)$**.
- En la próxima lección veremos algoritmos de búsqueda en vectores ordenados.

12