Estadísticas de orden

Dada una secuencia de n números A,

¿Cómo podemos encontrar el menor elemento de A?

¿Y el segundo menor?

Estadísticas de orden

¿Y el i-ésimo menor?

¿Qué complejidad tiene esto?

¿Con que valor de i el problema se hace más difícil?

Buscando la mediana

Concentrémonos en la mediana ya que es el más difícil

Podemos encontrarla ordenando en tiempo $\Omega(n \cdot \log n)$

¿Se puede encontrar la mediana sin ordenar los datos?

Buscando la mediana

Si tomamos un elemento cualquiera de la secuencia,

¿Cómo podemos comprobar si este es la mediana?

Dado un elemento arbitrario $x \in A$, podemos contar cuantos son mayores y cuantos son menores a x para saber su posición.

Adicionalmente podemos hacer una lista con los menores y con los mayores.

Buscando la mediana

Secuencia de datos:

7 3 5 2 1 9 7 8 6 4 3

Valor escogido: 6

Menores: 3 5 2 1 4 3

Mayores: 7 9 7 8

¿Donde está la mediana?

Buscando la mediana

Repitamos el proceso recursivamente

Valor escogido: 2

Menores: 1

Mayores: 3 5 4 3

Buscando la mediana

Repitamos el proceso nuevamente

1 2 3 5 4 3 6 7 9 7 8

Valor escogido: 5

Menores: 3 4 3

Mayores:

Buscando la mediana

¿Como sabemos cuando encontramos la mediana?

1 2 3 4 3 5 6 7 9 7 8

$\boldsymbol{partition}(\boldsymbol{A},\ \boldsymbol{i},\ \boldsymbol{f})$:

$\boldsymbol{p}$ = un pivote aleatorio en $\boldsymbol{A}[\boldsymbol{i},\ \boldsymbol{f}]$

$\boldsymbol{m},\ \boldsymbol{M}$ = listas vacías

for $\boldsymbol{x} \in \boldsymbol{A}[\boldsymbol{i},\ \boldsymbol{f}]$:

if $\boldsymbol{x} < \boldsymbol{p}$:

insertar $\boldsymbol{x}$ al final de $\boldsymbol{m}$

else if $\boldsymbol{x} > \boldsymbol{p}$:

insertar $\boldsymbol{x}$ al final de $\boldsymbol{M}$

$\boldsymbol{A}[\boldsymbol{i},\ \boldsymbol{f}]$ = concatenar $\boldsymbol{m}$ con $\boldsymbol{p}$ con $\boldsymbol{M}$

return $\boldsymbol{i} + |\boldsymbol{m}|$

Luego de hacer partition podemos fijarnos en que índice quedó finalmente el pivote.

Si el pivote cayó a la izquierda de la mediana, debemos revisar los datos a la derecha (los mayores)

Si el pivote cayó a la derecha de la mediana, debemos revisar los datos de la izquierda (los menores)

```
median(A):
    i = 0
    f = n − 1
    p = partition(A, i, f)
    while p ≠ n/2:
        if p < n/2:
            i = p + 1
        else:
            f = p − 1
        p = partition(A, i, f)
    return A[p]
```

Analicemos el algoritmo median

¿Cuál es su complejidad?

¿Cómo podemos usarlo para encontrar el i-ésimo menor?

```
quickSelect(A, j):
    i = 0
    f = n − 1
    p = partition(A, i, f)
    while p ≠ j:
        if p < j:
            i = p + 1
        else:
            f = p − 1
        p = partition(A, i, f)
    return A[p]
```

Analicemos el algoritmo *quickSelect*

¿Cuál es la complejidad? ¿Cuándo se da el peor caso?

¿Es la misma complejidad si hay elementos repetidos?

¿Se puede aprovechar que hayan elementos repetidos?

El peor caso se da cuando el pivote escogido hace que una de las dos listas, m o M, quedan vacías. Si esto pasa en cada paso, el algoritmo toma tiempo cuadrático.

Muchas implementaciones de partition escogen como pivote el primer elemento. Esto hace que el peor caso se de cuando los datos vienen en orden.

Usar un pivote aleatorio permite que el peor caso sea impredecible, por lo que no es posible causar el peor caso intencionalmente.

En el caso de que hayan elementos repetidos, es posible separar los datos en menores, iguales y mayores al pivote.

En este caso es necesario que partition retorne los índices donde empieza y termina la zona de iguales para no volverla a considerar.

Esta versión se conoce como partition 3-way

Ordenando los datos

¿Cómo podemos utilizar lo que hemos visto para ordenar?

Ordenando los datos

1 2 3 4 3 5 6 7 9 7 8

En cada iteración, el pivote queda en su posición ordenada

¿Por qué?

¿Se puede usar esto para ordenar?

$$quicksort(A, i, f):$$

$$\quad if\ i \leq f:$$

$$\quad\quad p = partition(A,\ i,\ f)$$

$$\quad\quad quicksort(A,\ i,\ p-1)$$

$$\quad\quad quicksort(A,\ p+1,\ f)$$

Para ordenar todo llamamos $quicksort(A,\ 0,\ n-1)$

Analicemos quicksort

¿Cual es su complejidad en el mejor caso?

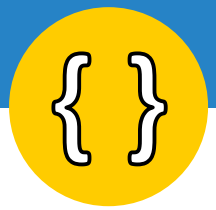
¿Y en el peor caso?

¿Cuándo se da el peor caso? ¿Se puede evitar?

El peor caso es $O(n^2)$, y la razón y el análisis son el mismo que con quickSelect.

La mejor forma de evitarlo sigue siendo escoger un pivote aleatorio.

Quicksort en arreglos



En general, usar arreglos es más conveniente

Pero la operación de concatenar es muy cara en arreglos

Debemos reformular $\boldsymbol{partition}$ para que funcione en arreglos

$\mathit{partition}(A,\ i,\ f)$:

$x =$ un indice aleatorio en $[i,\ f]$

$p = A[x]$

$A[x] \rightleftarrows A[f]$

$j = i$

$\mathit{for}\ k \in [i,\ f-1]$:

$\mathit{if}\ A[k] < p$:

$A[j] \rightleftarrows A[k]$

$j = j + 1$

$A[j] \rightleftarrows A[f]$

$\mathit{return}\ j$

Resumen de algoritmos de ordenación

Algoritmo	Mejor caso	Caso promedio	Peor caso	Memoria adicional
SelectionSort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
InsertionSort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
QuickSort	$O(n \cdot \log n)$	$O(n \cdot \log n)$	$O(n^2)$	$O(1)$
MergeSort	$O(n \cdot \log n)$	$O(n \cdot \log n)$	$O(n \cdot \log n)$	$O(n)$
?????	$O(n \cdot \log n)$	$O(n \cdot \log n)$	$O(n \cdot \log n)$	$O(1)$

Las demostraciones formales de complejidad de QuickSort se verán en ayudantía.

Y un último algoritmo que veremos este miércoles