Universidad Nacional de Ingeniería
Ciencias de la Computación
Medians and order Statistics

Yuri Nuñez Medrano *
ynunezm@gmail.com

Resumen

El Orden estadístico i-esimo de un conjunto de n elementos es el i-esimo menor elemento, por ejemplo el minimo de un conjunto de elementos es el primer orden estadístico ($i = 1$), y el máximo es el nesimo orden estadístico (i=n), la mediana, es la mitad dependiendo si es par o impar, $i = \lfloor (n+1)/2 \rfloor$ o $\lceil (n+1)/2 \rceil$ dependiendo del caso.

1. Medianas y Orden Estadístico

Podemos encontrar el i-esimo elemento mediante el ordenamiento previo de n elementos con un tiempo de ejecución $\Omega(nlgn)$, mediante un algoritmo usando heapsort o mergesort, y luego encontrando el indice del i-esimo, veremos casos donde se puede encontrar el i-esimo elemento en un tiempo de ejecución de $O(nlgn)$ en el peor de los casos. .

1.1. Mínimo y Maximo

Cuantas comparacione son necesarias para determinar el minimo de n elementos. Facilmente podemos tener un limite superior de $n-1$ comparaciones, donde $A.lenght = n$.

Algorithm 1: MINIMUN(A)

```
1 min=A[1]
2 for i = 2 to A.lenght do
3     if min >A[i] then
4         min=A[i]
5 return min
```

Simultaneamente mínimo y máximo.
En algunas aplicaciones se nos hace necesario encontrar el maximo como tambien el mínimo. Con el caso anterior podemos evaluar $2n-2$ comparaciones. Nosotros podemos el mínimo y el máximo con $3\lfloor n/2 \rfloor$ comparaciones, teniendo auxiliares para el mínimo y máximo que comparan de dos indices y los siguientes dos indices siguientes y con los auxiliares y nos quedamos tanto con el mínimo y el máximo.

1.2. Seleccion en tiempo lineal esperado

En general el problema de selección parece más dificultoso que encontrar el mínimo. Pero el tiempo de ejecución $O(n)$ es sorprendente para ambos problemas.
El algorítmo 2 tiene modelado despues del QUICKSORT. Como un QUICKSORT partimos el input array, pero con la diferencia que evaluamos sólo una partición en vez de dos, esta diferencia nos da un tiempo de ejecución de $\Theta(n)$ asumiendo que los elementos son distintos.

Algorithm 2: RANDOMIZED_SELECT(A,p,r,i)

```
1 if p==r then
2     return A[p]
3 q=RANDOMIZED_PARTITION(A,p,r)
4 k=q-p+1
5 if i==k then
6     return A[q]
7 else if i<k then
8 return RANDOMIZED_SELECT(A,p,q-1,i) ;
9 else  return RANDOMIZED_SELECT(A,q+1,r,i-k) ;
```

Intuición para el Análisis
Asumimos que todos los elementos son diferentes.
Caso lucky(suertudo) $\frac{1}{10} : \frac{9}{10}$.
$T(n) \leq T(\frac{9n}{10}) + \Theta(n)$
Teorema Maestro caso 3
$\Theta(n)$
Caso unlucky(sinsuerte) $n : n-1$
$T(n) = T(n-1) + \Theta(n)$
$\Theta(n^2)$
Análisis de tiempo Esperado

*Escuela de Ciencias de la Computación, 27-08-15

- T(n) es una variable aleatoria para el tiempo de ejecución RANDOMIZED_SELECT input de tamaño n, asumiendo que las variables aleatórias son independientes.
- Definimos el indicador aleatório X_k para $k = 0, 1, ..., n-1$.

$$X_k = \left\{ \begin{array}{ll} 1 & si\ genera\ part.\ k : n-k-1 \\ 0 & para\ los\ demas \end{array} \right\}$$

$$T(n) = \left\{ \begin{array}{ll} T(max\{0, n-1\}) + \Theta(n) & si\ 0 : n-1 \\ T(max\{1, n-2\}) + \Theta(n) & si\ 1 : n-2 \\ ... & ... \\ T(max\{n-1, 0\}) + \Theta(n) & si\ n-1 : 0 \end{array} \right\}$$

$E[T(n)] = \sum_{k=0}^{n-1} E[X_k(T(max\{k, n-k-1\}) + \Theta(n))]$
$= \sum_{k=0}^{n-1} E[X_k]E[T(max\{k, n-k-1\}) + \Theta(n)]$
$= \frac{1}{n}\sum_{k=0}^{n-1} E[T(max\{k, n-k-1\})] + \frac{1}{n}\sum_{k=0}^{n-1}\Theta(n)$
$= \frac{2}{n}\sum_{k=\lfloor n/2 \rfloor}^{n-1} E[T(k)] + \Theta(n)$
Necesita
$E[T(n)] \leq cn$ para una cte>0
Probar por el Método de Sustitución
$= \frac{2}{n}\sum_{k=\lfloor n/2 \rfloor}^{n-1} \underbrace{E[T(k)]}_{\leq cn} + \Theta(n)$
$= \frac{2}{n}\sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + \Theta(n)$
$= \frac{2c}{n}\sum_{k=\lfloor n/2 \rfloor}^{n-1} k + \Theta(n)$
$= \frac{2c}{n} \underbrace{\sum_{k=\lfloor n/2 \rfloor}^{n-1} k}_{<\frac{3}{8}n^2} + \Theta(n)$
$= cn - (\underbrace{\frac{1}{4}cn - \Theta(n)}_{\geq 0})$
$\leq cn$

1.3. Seleccion en el caso peor tiempo lineal

Se evaluará el algoritmo SELECT parecido al algoritmo 2 en un tiempo de ejecución de $O(n)$ en el caso peor, Aqui sin embargo se garantiza una buena partición del array, SELECT usa el algoritmo determinista PARTITION del Quicksort.

- Dividir los n elementos del arreglo, en $\lfloor \frac{n}{5} \rfloor$ grupos de 5 elementos y un grupo de n mod 5 elementos.
- Calcular la mediana de cada uno de los $\lceil \frac{n}{5} \rceil$ grupos
- Usar SELECT recursivamente para calcular las medianas de x, de las $\lceil \frac{n}{5} \rceil$ medianas calculadas anteriormente.
- Dividir el arreglo de entrada en 2 subarray utilizando la version modificada de PARTITION, para que el pivote sea x. Sea k el número de elementos del lado de abajo de la particion y n-k del otro lado.
- Si $i = k$ retornar x
  Si $i < k$ usar SELECT recursivamente para el iesimo elemento menor elemento en la parte inferior.

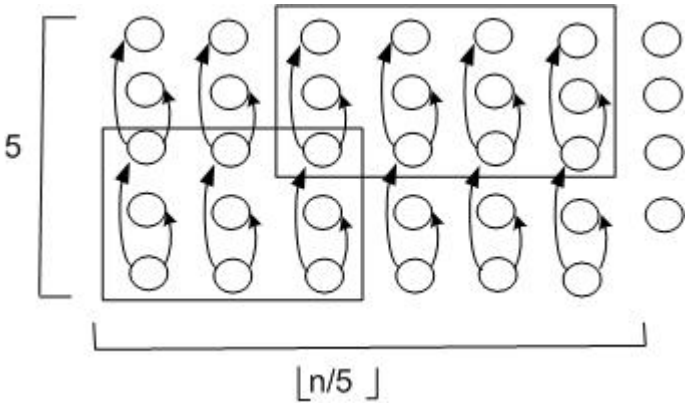

Figura 1: Select

Sino usar SELECT recursivamente al (i-k)esimo menor elemento de la parte superior.