K-esimo elemento y Quicksort randomizado

Teoría de Algoritmos I (75.29 / 95.06)

Ing. Víctor Daniel Podberezski
vpodberezski@fi.uba.ar

Calculo de la mediana

Sea

un set de números n = $\{a_1, a_2, ..., a_n\}$

La mediana

es el numero que queda en la posición del medio si se presentan ordenados

Ej:

10,6,2,15,1 → 1,2,6,10,15

Formalmente

El k-esimo numero más grande de S tal que

k=(n+1)/2, si n es par

K=n/2 ,si n es impar

Solución determinística

Podemos calcularlo mediante:

Sort → O(n log n)

Podemos hacerlo mejor?

Usaremos una algoritmo randomizado + división y conquista

Encontrar el k-esimo elemento

Sea

El set S de n números,

Un número k entre 1 y n

La función SELECT(S,k)

Retorna el k-esimo mayor elemento.

Casos particulares:

- Mediana: $k = n/2$ o $(n+1)/2$
- Minimo $k=1$
- Máximo $k=n$

División del problema: El pivot

Seleccionar un elemento $a_i \in S$ como pivot Y formar 2 sets

S- = {aj : aj < ai}

S+ = {aj : aj > ai}

Pueden ocurrir 3 cosas:

Si |S-|=k-1 entonces ai es el k-esimo elemento.

Si |S-|>k-1 nos quedamos con S- y repetimos el Proceso

Si |S-|<k-1 nos quedamos con S+ y repetimos, buscando el k-1-|s-| elemento

Algoritmo

```
select(S,k)
   Sl={} Sr={}
   p = calcularpivot(k)
   Desde j=1 a k
      Si sj < p
         Sl += {sj}
      Si sj > p
         Sr+={sj}

   si size(sl) = k-1
      return p;
   Sino si size(sl) > k-1
      select (sl , k)
   sino
      select (sr, k – 1 - size(sl))
```

Análisis del pivot

Si pudiésemos seleccionar el pivot justo como el valor medio

- Siempre nos quedaríamos con la mitad de los elementos.
- Nos quedaría una recurrencia como: $T(n) = T(n/2) + cn$
- Aplicando el teorema maestro: $O(n)$

En el peor de los casos si selecciono el menor (o mayor de los elementos

- La recurrencia me queda $T(n) = T(n-1) + cn$
- Nos quedaría una complejidad de $O(n^2)$

Tenemos que intentar seleccionar un pivot “central”!

Pivot “centrado”

Podemos intentar seleccionar un pivot que al menos ε * n elementos menores y mayores que él (ε>0)

- La recurrencia me quedaría como: $T(n) = T((1-\varepsilon)n) + cn$
- La complejidad nos quedaría lineal.
- Cuanto mas central, mas rápido achicamos el conjunto analizado.

Una elección al azar

Proponemos seleccionar un $a_i \in S$ como pivot uniformemente al azar

- Consideramos centrales a los elementos que al menos dejan ¼ de los elementos del lado izquierdo o a la derecha
- La mitad de los elementos son centrales (ε=1/4)
- La probabilidad de seleccionar un pivot central es de ½

Al dividir en S+ y S- verificamos que la división cumpla el requisito

- Si no cumple, volvemos a seleccionar al azar otro pivot
- Probabilisticamente tendría que repetir a lo sumo 2 veces la elección
- Este proceso es O(n)

Análisis de cada fase

En cada fase j del algoritmo

Reduzco al menos en ¼ el tamaño del problema

El tamaño del conjunto que estoy analizando esta acotado por

$$n\left(\frac{3}{4}\right)^{j+1} \leqslant |S'| \leqslant n\left(\frac{3}{4}\right)^{j}$$

La cantidad de pruebas de pivot esperado es a lo sumo 2.

Análisis global

En cada fase X_j

la cantidad de elementos es a lo sumo $n(3/4)^j$

La cantidad de operaciones en cada fase son lineales $c^*n(3/4)^j$

Se espera que la cantidad de repeticiones de cada fase sea 2, entonces $E[Xj] \leq 2c^*n(3/4)^j$ (esperanza de la fase X_j)

El algoritmos esta conformado por una sucesión de fases

$X = X0 + X1 + X2 + \ldots$

La esperanza total es $E[X] \leqslant \sum_j E[X_j] \leqslant \sum_j 2cn\left(\frac{3}{4}\right)^j = 2cn\sum_j \left(\frac{3}{4}\right)^j \leqslant 8cn$

QuickSort

Es un algoritmo de ordenamiento

creado por C. A. R. Hoare.

Utiliza

Division y conquista

Divide en cada paso

En 2 subproblemas utilizando un valor pivot

Por un lado se procesan los valores menores al pivot y por el otro los mayores

Pseudocódigo

```
QuickSort(S)
  Si |S|<=3
    Ordenar S
    Retornar S
  Sino
    p = seleccionarPivot(S)
    Por cada elemento de S
      Ponerlo en S- si es menor a p
      Ponerlo en S+ si es mayor a p


    S- = QuickSort(S-)
    S+ = QuickSort(S+)

  Retornar S-, p, S+
```

Análisis de la solución

La eficiencia de la solución

Depende de la selección del pivot

Si el pivot es el valor medio

Divide los problemas en partes iguales

Queda una recurrencia $T(n) = 2T(n/2) + O(n)$

Que es $O(nlogn)$

Si el pivot es "malo",

Deja separado en tamaños de subrpoblemas muy dispares

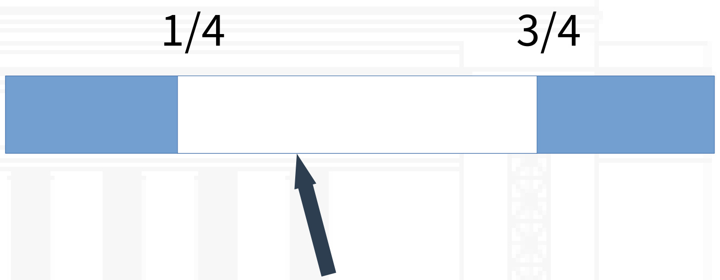
En el peor caso nos queda una recurrecncia $T(n) = T(n-1) – O(n)$

Que es $O(n^2)$

Quicksort randomizado

Modificaremos Quicksort

Intentaremos elegir el pivot aleatoriamente



Queremos que sea “central”

Ni entre ¼ *|S| inicial ni final

Al dividir en S+ y S- verificamos que la división cumpla el requisito

Si no cumple, volvemos a seleccionar al azar otro pivot

Probabilisticamente tendría que repetir a lo sumo 2 veces la elección

Pseudocódigo randomizado

```
QuickSort(S)
  Si |S|<=3
    Ordenar S
    Retornar S
  Sino
    Repetir
      p = seleccionarAleatoriamentePivot(S)
      Por cada elemento de S
        Ponerlo en S- si es menor a p
        Ponerlo en S+ si es mayor a p
    Hasta que |S-|>=1/4*|S| y |S+|>=1/4*|S|

    S- = QuickSort(S-)
    S+ = QuickSort(S+)

  Retornar S-, p, S+
```

Complejidad

En cada fase iteración j

la cantidad de elementos es a lo sumo $|S|(3/4)^j$

La cantidad de operaciones en cada fase son lineales $c^*|S|(3/4)^j$

Se espera que la cantidad de repeticiones de cada fase sea 2.

Hay a lo sumo O(log|S|)

Iteraciones internas

Por lo tanto

El proceso total es $O(|S|\log|S|)$

.UBA fiuba

FACULTAD DE INGENIERÍA

Presentación realizada en Junio de 2020