Apuntes de Teórico

PROGRAMACIÓN 3

Sorting

Versión 1.0

Apuntes de Teórico de Programación 3 – Sorting

Índice

Introducción: Sorting

Sorting es el proceso de ordenar una secuencia de elementos de acuerdo a un orden lineal.

Se clasifican los diferentes tipos de ordenamiento en :

- Internos –En la memoria de la computadora-
  - Basados en estructura de los elementos
  - Comparación de claves
- Externos – En memoria secundaria-
    - ejs.: Bases de datos, archivos.

Nota: En este curso se estudiará ordenamiento interno basado en comparación de elementos.

Definición del problema de sorting

Elementos a ser ordenados: producto cartesiano con un componente clave. La clave pertenece a un tipo de datos para el cual está definido la relación de orden lineal "**≤**"

Ejemplo:

persona

nombre

cédula ← CLAVE (se considera numérica)

dirección

Definición del problema:

Dados n elementos $r_1, r_2, r_3, \ldots, r_n$, cada elemento r_i tiene una clave a_i, se quiere ordenar los elementos por su clave de acuerdo a un orden lineal $\leq$

Entonces, dadas las claves a_1, a_2, a_n, se quiere determinar una permutación de las mismas de forma que

$a_{p1} \leq a_{p2} \leq a_{p3} \leq \ldots \leq a_{pn}$

o sea $\mathbf{a_{pi} \leq a_{pi+1}}$ $1 \leq i \leq n-1$

Algoritmos basados en comparaciones

Clase de algoritmos que están definidos por una operación básica que consiste en comparar elementos de la secuencia dada 2 a 2

Análisis del problema sorting

Se trata de determinar la complejidad mínima de cualquier algoritmo que ordene por comparaciones. Se expresa en cantidad de comparaciones realizadas al ordenar secuencias de largo n.

Sea D_n el dominio de definición de los algoritmos de la clase o sea que D_n estará compuesto por secuencias, de largo ***n***, de enteros.

Sea C la clase de algoritmos que ordena por comparación de claves 2 a 2 secuencias de D_n:

$C = \{A_1, A_2, \ldots, A_q\}$

En otras palabras: Se quiere hallar una cota inferior, en cantidad de comparaciones, que valga para cualquier algoritmo de la clase, para:

$$\begin{cases} \text{Peor Caso:} & F_W(n) \\ \text{Caso Medio:} & F_A(n) \end{cases}$$

Peor Caso

Sea $T_i(n)$ la cantidad de comparaciones que realiza un algoritmo A_i, para ordenar una secuencia de n elementos. Se denotará $T_{Wi}(n)$ a la cantidad de comparaciones en el peor caso de A_i.
Por definición de peor caso, será: $T_i(n) \leq T_{Wi}(n)$

Cada Algoritmo de la clase tendrá asociado el costo en su peor caso, se define el conjunto de estas expresiones:

$T_W = \{T_{W1}(n), T_{W2}(n), \ldots, T_{Wq}(n)\}$

El problema a resolver consiste en determinar $F_W(n)$ tal que:

$F_W(n) \leq \min T_W$

O sea que se podrá decir:

$T_{Wi}(n) \in \Omega (F_W(n))$

Cualquier algoritmo de ordenación realizará al menos $F_W(n)$ comparaciones en su peor caso.

Cálculo de Fw(n)

- Cualquier algoritmo A (genérico de la clase C) tiene asociado su árbol de decisión (AD).
- Sea k = prof(AD)

De lo anterior se concluye que $T_w(n) = k$

Si ***m*** es la cantidad de **nodos externos**, entonces se sabe por ***el lema 1*** que $\mathbf{m \leq 2^k}$

Se quiere razonar sobre los árboles de decisión en forma independiente de los algoritmos correspondientes.

Para esto debe tomarse un árbol genérico y acotar inferiormente su profundidad **k.** De esta forma se podrá asegurar que la cota que se encuentre sirva para **todo** AD (AD de un algoritmo que resuelva el problema de sorting)**.**

Entonces deberá buscarse un AD hipotético que tenga el menor **k** posible. Esas condiciones se dan en un AD con pocos nodos externos y que sea completo (o cercano a completo) como se consideraba en el lema 1. Bajo estas condiciones se tiende a reducir el árbol, en particular su profundidad. Sin embargo el AD debe seguir correspondiendo a un cierto algoritmo que **resuelva** el problema de sorting y por lo tanto debe tener una mínima cantidad de nodos externos.

¿Cuál es el mínimo valor para **m**?

Obs. 1) Las instancias de D_n son infinitas, pero se puede observar que dado cualquier algoritmo y su AD asociado, todas las instancias que mantengan el mismo orden relativo entre sus elementos terminarán en el mismo nodo externo.

Obs. 2) En particular D_n contiene al conjunto D'_n formado sólo por las secuencias sin elementos repetidos. Agrupando en este caso aquellas instancias que tienen el mismo orden relativo se tienen ***n!*** posibles grupos de secuencias. Cada una de estas agrupaciones terminará en el mismo nodo externo y se tendrá, para D'_n , que **m** debe ser al menos **n!**.

Obs. 3)Al considerar D_n en lugar de D'_n lo que puede suceder es que se agreguen hojas al AD y tener un valor de **m** mayor (nunca puede disminuir m agrandando el conjunto de entradas). Como lo que se busca es el menor valor de **m** posible se tomará **n!** como dicho valor.

Es decir: $n! \leq m$

Y por lo tanto: $n! \leq m \leq 2^k$

Como se trata de hallar una cota inferior para el peor caso, se trata de encontrar una cota inferior para ***k*** y entonces se debe hallar **Fw(n) ≤ k para cualquier AD.**

Entonces: $\mathbf{n! \leq 2^k}$

Tomando logaritmos en base 2:

- $k \geq \log (n!)$
- $\log(\ n!\ ) = \sum_{i=1}^{n} \log\ \ i$

Se dará por conocido que

- $\sum_{i=1}^{n} \log i \geq n \log n - 1.5n$

Entonces: $k \geq n \log n - 1.5n$

Y por lo tanto se podrá considerar que la **Fw(n) buscada es:**

$$Fw(n) = n \log n - 1.5n \in \theta(n \log n) \Rightarrow$$
$$Fw(n) = n \log n - 1.5n \in \Omega(n \log n)$$

Usualmente se dice que la función que se buscaba en un principio es $\boldsymbol{F_w(n) = n \log n}$

Esto vale para todo algoritmo de la clase C, entonces **todo algoritmo** que ordene por comparación de elementos, realizará al menos ***n log n*** comparaciones en **su peor caso**.

Caso promedio

En forma similar al peor caso, sea $T_i(n)$ la cantidad de comparaciones que realiza un algoritmo A_i de la clase C, para ordenar una secuencia de n elementos. Se denotará $T_{Ai}(n)$ a la cantidad de comparaciones en el caso medio de A_i.

Cada Algoritmo de la clase tendrá asociado el costo en su caso medio, definimos el conjunto de estos valores:

$T_A = \{ T_{A1}(n), T_{A2}(n),\ldots, T_{Aq}(n)\}$

El problema a resolver consiste en determinar $F_A(n)$ tal que:

$F_A(n) \leq \min T_A$

O sea que se podrá decir: $T_{Ai}(n) \in \Omega (F_A(n))$

Cualquier algoritmo de ordenación realizará al menos $F_A(n)$ comparaciones en su caso medio.

Consideraciones previas

a) Como se trata de un caso promedio, deben especificarse las condiciones bajo las cuales se considerará dicho promedio, estas son:

- Todas las secuencias tienen la misma probabilidad de ocurrir
- Las secuencias consideradas no tienen elementos repetidos (al igual que en el peor caso se denota D'_n a este dominio)

b) Las secuencias de D'n también son infinitas, por lo que se deberá obtener un punto de vista útil para trabajar con las probabilidades. Efectivamente existen infinitas secuencias, sin embargo es posible agruparlas en grupos equivalentes desde el punto de vista de la ejecución del algoritmo como en el caso anterior (peor caso).
Como todas las secuencias son equiprobables los grupos de secuencias también son equiprobables y considerando cualquier árbol de decisión, todos los nodos externos resultan equiprobables (como fin de ejecución).

c) Teniendo en cuenta lo anterior, análogamente a lo visto para el peor caso, se tiene que la cantidad de nodos externos es exactamente **n!**.

Determinación del costo en el caso medio

Por definición se tiene **para todo** algoritmo de la clase C:

$$T_A(n) = \sum_{S \in D_n'} p(S) * T(S)$$

Donde

p(S) es la probabilidad que se de la secuencia S∈ D'n y
T(S) es la cantidad de comparaciones que realiza el algoritmo para S

Utilizando las consideraciones del apartado anterior, se puede plantear esta sumatoria en términos de nodos externos del árbol de decisión:

$$T_A(n) = \sum_{iexterno} p_i k_i$$

Donde:

p_i es la probabilidad de alcanzar el nodo externo i o sea **p_i = 1/n!**
k_i es el largo del camino de la raíz al nodo **i** o sea cantidad de comparaciones en ese caso.

Se tiene entonces: $T_A(n) = \sum_{iexterno} p_i k_i = \frac{1}{n!} \sum k_i$

Por el lema 2 se sabe que la sumatoria de profundidades de los nodos externos del AD es mayor o igual a ***m log m*** , siendo m la cantidad de nodos externos.

la función F_A buscada cumplirá**:** $F_A(n) \leq T_A(n) = \frac{1}{n!} \sum_{iexterno} k_i$

Como **m = n!,** y por **Lema 2** se tiene: $F_A(n) \leq \frac{1}{n!} n! \log(n!)$

Se concluye:

$$F_A(n) = n \log n - 1.5\, n$$

$$F_A(n) \in \Omega(n \log n)$$

Usualmente se dice que la función que se buscaba en un principio es: ***$F_w(n) = n \log n$***

Esto vale para todo algoritmo de la clase C, entonces **todo algoritmo** que ordene por comparación de elementos, realizará al menos ***n log n*** comparaciones en **su caso medio (en las condiciones establecidas)**.

Observaciones finales:

Se han encontrado las funciones

$F_W(n) = n \log n$ que será el costo mínimo para cualquier algoritmo de la clase C en su peor caso.

$F_A(n) = n \log n$ ídem para el caso medio en las condiciones establecidas.

def) Óptimo: cualquier algoritmo A_i que cumpla $T_W(n) = F_W(n)$ comparaciones se dirá que es óptimo en el peor caso. Análogamente vale para el caso medio. (Esta definición de optimalidad vale para cualquier problema cambiando la palabra "comparaciones" por "operaciones básicas")

Obs.) Si se encuentra un algoritmo que haga $F_W(n)$ comparaciones en el peor caso, implicará que la cota inferior encontrada es la mejor posible (no podría ser mayor que $F_W(n)$ porque habría un algoritmo que hace menos comparaciones). Análogamente sucede con el caso medio.