Auxiliar 4 - "Memoria Externa"

Profesor: Pablo Barceló
Auxiliar: ~~Manuel~~ Ariel Cáceres Reyes

17 de Abril del 2017

P1. Relación

Dada una relación binaria $\mathcal{R} \subseteq \{1,\ldots,k\}\times\{1,\ldots,k\}$, representada como un archivo secuencial de sus pares (i,j), construya algoritmos eficientes (considerando que $N \gg M$) para determinar si la relación es:

a) Refleja

b) Simétrica

P2. Arreglos

Se tienen dos arreglos A y B de largo N almacenados en memoria externa. Si bien son diferentes, ambos contienen los enteros entre 1 y N. Se desea construir el arreglo C, dado por $C[i] = A[B[i]]$. Diseñe un algoritmo eficiente para esto, y analícelo.

P3. Mayoría

Considere una lista L de N elementos en memoria externa. Diseñe un algoritmo eficiente para encontrar un elemento que tenga la mayoría absoluta (es decir, que aparezca más de $N/2$ veces), y reportar su frecuencia. Si no existe tal elemento debe reportarse `no`.

P4. Skyline

Se nos entrega un conjunto P de N puntos en $\mathbb{R}^d$ en *posición general*; es decir, ningún par de puntos comparte el mismo valor en alguna dimensión. Considere que $N \gg M$. Dado un punto $p \in \mathbb{R}^d$, llamaremos $p[i]$ a su i-ésima coordenada. Un punto p_1 domina a otro punto p_2 (denotado como $p_1 \prec p_2$) si se cumple $p_1[i] < p_2[i], \forall i = 1,\ldots,d$.
Se nos pide calcular el *skyline* de P , denotado como $SKY(P)$, que incluye todos los puntos de P que no son dominados por ningún otro:

$$SKY(P) = \{p \in P | \nexists p' \in P, p' \prec p\}$$

a) Resuelva el problema para $d = 2$ usando $\mathcal{O}(n \log_m n)$ operaciones de disco.

b) Resuelva el problema, ahora para $d = 3$. Llegue a la misma cota para la cantidad de operaciones de disco que en el caso anterior.

Soluciones

P1. Relación

Para esta pregunta asumiremos que no se repiten pares.

a) Basta tener un contador iniciado en 0, escanear todo el input bloque por bloque y cuando encontramos un par de la forma (i, i) aumentamos el contador. Cuando terminamos de escanear respondemos que la relación es refleja si el contador es igual a k, no en caso contrario. Como solo escaneamos el input una vez hacemos $n = \frac{N}{B}$ llamadas a disco.

b) Con el siguiente algoritmo:

- Hacer una copia de los pares, pero con sus coordenadas invertidas. $\mathcal{O}(n)$.
- Ordenar tanto el original como la copia lexicográficamente (es decir, por la primera coordenada y en caso de empate mirar la segunda coordenada para decidir). $\mathcal{O}(n \log_m n)$, con $m = \frac{M}{B}$.
- Comparar que tanto la copia como el original sean iguales. $\mathcal{O}(n)$

Por lo tanto, el costo del algoritmo es $\mathcal{O}(n \log_m n)$ llamadas a disco.

P2. Arreglos

Con el siguiente algoritmo de costo $\mathcal{O}(n \log_m n)$:

- Generar archivo con los pares $(i, B[i])$. $\mathcal{O}(n)$
- Ordenar el archivo por la segunda coordenada. $\mathcal{O}(n \log_m n)$
- Reemplazar en el archivo la segunda coordenada por A. $\mathcal{O}(n)$
- Ordenar por la primera coordenada. $\mathcal{O}(n \log_m n)$
- Generar archivo con las segundas coordenadas, que es C. $\mathcal{O}(n)$

Este algoritmo hace $\mathcal{O}(n \log_m n)$ llamadas a disco.
Veamos que como los índices que ponemos y A y B contienen una permutación del 1 al N, siempre se cumple que para un i cualquiera:

- En la primera generación se ve: $\underbrace{\ldots (i, B[i])}_{\text{posición } i} \ldots$
- Después del primer orden: $\underbrace{\ldots (i, B[i])}_{\text{posición } B[i]} \ldots$
- Después de poner A: $\underbrace{\ldots (i, A[B[i]])}_{\text{posición } B[i]} \ldots$
- Después del segundo orden: $\underbrace{\ldots (i, A[B[i]])}_{\text{posición } i} \ldots$

P3. Mayoría

El siguiente algoritmo que escanea el input 2 veces por bloques (por lo que cuesta $\mathcal{O}(n)$ llamadas a disco) resuelve el problema:

```
i ← 0
for x ∈ L do
  if i == 0 then
    m ← x
    i ← 1
  else
    if x == m then
      i ← i + 1
    else
      i ← i − 1
    end if
  end if
end for
i ← 0
for x ∈ L do
  if x == m then
    i ← i + 1
  end if
end for
if i > N/2 then
  return m, i
else
  return no
end if
```

. Veamos que el primer ciclo presenta a m como un candidato a ser mayoría absoluta y el segundo ciclo verifica que este elemento realmente lo sea.
Si no existe mayoría absoluta el algoritmo responde correctamente *no* pues el segundo ciclo desecha el candidato m presentado por el primero.
Si existe un elemento K que es mayoría absoluta, en este caso imaginemos un índice j que no es parte del algoritmo pero se actualiza a medida que este avanza. j aumenta en 1 si el elemento escaneado es igual a K y disminuye en 1 si es distinto a K. Observemos que cuando $j > 0$ se cumple que el i del algoritmo es > 0 y el $m = K$ (pues significa que se han visto más Ks que otro símbolo). Finalmente cuando se terminan las iteraciones $j > 0$, pues K es mayoría absoluta, y por lo tanto $m = K$, el algoritmo lo postula como candidato y luego lo corrobora.

P4. Skyline

a) $(d = 2)$.
El siguiente algoritmo computa el SKY:

$SKY \leftarrow \emptyset$
$y_{min} \leftarrow \infty$
$Sort_x(P)$
for i =1...N **do**
  $(x, y) \leftarrow p_i$
  if $y < y_{min}$ **then**
    $y_{min} \leftarrow y$
    $SKY.add(p_i)$
  end if
end for

El mayor costo del algoritmo se realiza cuando se ordena por la coordenada x, por lo que el número de llamadas a disco es $\mathcal{O}(n \log_m n)$. Veamos que cuando un punto es dominado por otro p se cumplirá que no es agregado a SKY, pues antes de analizarlo (dado que están ordenados por x) se debió haber seteado y_{min} como el y del punto o como algo menor que esto, por lo que no cumple la condición del **if**. Por otro lado si el punto no es dominado, ningún punto menor que el en la coordenada x debería ser menor también en la coordenada y, por lo que a l ser procesado cumple la condición del **if** y es agregado al SKY.

b) En el caso 3D, nuevamente ordenamos por la coordenada x y resolvemos el problema usando un algoritmo recursivo que:

- Pediremos que el resultado de la recursión sea calcular el SKY como una lista ordenada por coordenada y.
- El caso base ocurre cuando el input cabe en memoria principal, es decir $N < M$ y en este caso basta traer los datos a memoria en $\mathcal{O}(n)$ y computar el resultado en memoria interna.
- Para el paso recursivo lo que hacemos es particionar los puntos P en $\Theta(M/B) = \Theta(m)$ partes iguales $P_1, \ldots, P_{\Theta(m)}$. Luego obtenemos recursivamente $SKY(P_1), \ldots, SKY(P_{\Theta(m)}$. Finalmente lo que hacemos es escanear estos SKY según coordenada creciente de y, y manteniendo un buffer en memoria de $\Theta(B)$ de cada uno de ellos. Además mantenemos un arreglo Z_{min} que en $Z_{min}[i]$ tiene el menor valor de z observado para los puntos ya escaneados de $SKY(P_i)$. Entonces si analizo el punto $p_i \in SKY(P_i)$ este será parte del $SKY(P)$ ssi $\forall j < i,\ p_i[3] < Z_{min}[j]$. Lo anterior se cumple gracias a que comenzamos ordenando por la coordenada x y además estamos escaneando los puntos por la coordenada y. Este proceso toma tiempo $\mathcal{O}(n)$.

La ecuación de recurrencia correspondiente es:

$$T(N) = \begin{cases} \mathcal{O}(n) & N < M \\ mT(N/m) + \mathcal{O}(n) & si\ no \end{cases}$$

Si resolvemos obtenemos $T(N) \in \mathcal{O}(n \log_m n)$. 🙏