Universidad de Chile
Departamento de Ciencias de la Computación
CC4102 - Diseño y Análisis de Algoritmos

Auxiliar 3 - "Memoria Externa"

Profesores: Pablo Barceló
Gonzalo Navarro
Auxiliares: Matilde Rivas
Bernardo Subercaseaux

8 de Octubre del 2018

En este modelo, el costo de un algoritmo será el número de accesos al disco (Inputs/Outputs)

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. Desordenar
Se nos pide desordenar un archivo ordenado de números reales. Debemos leer un archivo de disco de largo N y producir otro de largo N, donde se encuentren los mismos elementos permutados.
Su programa debe producir cualquier permutación con la misma probabilidad. No describa cómo permutar uniformemente en memoria interna.

a) Resuelva el problema en $\mathcal{O}(n)$ I/Os para el caso en que $N \leq M^2/B$.

b) Resuelva el problema en el mismo tiempo de ordenar en disco, para el caso general.

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`.

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. Desordenar

a) Tendremos $N/M \leq M/B$ archivos y mandamos aleatoriamente elementos del input a algunos de estos archivos (tenemos buffers de B elementos que nos caben en memoria principal), si alguno de los archivos alcanza los M elementos no se le considera más en la repartición aleatoria del input. Luego uno por uno nos traemos estos archivos a memoria principal, les hacemos shuffle y los mandamos de vuelta a memoria externa. Finalmente vamos eligiendo al azar elementos de alguno de los archivos y los vamos mandando al output (todo esto con buffers). Notar que en términos de I/Os el algoritmo solo unas cuantas pasadas lineales por los datos, es decir $\mathcal{O}(N/B) = \mathcal{O}(n)$.

b) El problema con que $N > M^2/B$ es que ya no podemos dividir el input en N/M archivos, pues solo tenemos la capacidad de hacerlo en M/B archivos con lo que los archivos nos quedan de tamaño $N/(M/B) > M$ y no podemos hacer la segunda fase de traer los archivos a memoria y hacerles shuffle ahí. Para resolver lo anterior le haremos shuffle recursivamente a los archivos de tamaño $N/(M/B)$ lo que nos dará archivos de tamaño $N/(M/B)^2$ y así hasta que $N/(M/B)^i = M$ y nos quepan en memoria para hacerles el shuffle. Ahora la complejidad es un proceso lineal, pero que se hace $i = \log_m n$ veces por lo que la complejidad final es la de ordenar.

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.