[#A] 2.2 Ordenamiento en memoria secundaria: Mergesort. Cota inferior.

Jérémy Barbay

09 September 2010

Contents

1. Un modelo mas fino
   (a) Cuantos paginas quedan en memoria local?
      - no tan importante para busqueda
      - muy importante para applicaciones de computacion con mucho datos.

   (b) Nuevas notaciones
      - B = Tamano pagina
      - N = cantidad de elementos en total
      - n = cantidad de paginas con elementos = N/B
      - M = cantidad de memoria local
      - m = cantidad de paginas locales = M/B
      - mnemotechnique:
        - N,M,B en # palabras maquinas (=bytes?)
        - n,m en # paginas
        - $n << N$, $m << M$

   (c) En estas notaciones, usando resultos previos:
      - Insertion Sort (en un B-Arbol)
        - usa dictionarios en memoria externa
        - $N \lg N / \lg B = N \log_B N$

- Heap Sort
  - usa colas de prioridades en memoria externa
  - $N \lg N / \lg B = N \log_B N$
  - Eso es optimo o no?

2. Cotas Inferiores en Memoria Secundaria

   - para buscar en un diccionario?
     - en modelo RAM? (de comparaciones)
       - $\lg N$
     - en modele Memoria Externa? (de comparaciones)
       - $\lg N / \lg B = \log_B N$ (tight)
       - para fusionar dos arreglos ordenados?
     - en modelo RAM?
       - N
     - en modelo Memoria Externa con paginas de tamaño B?
       - $N/B = n$ (tight)
       - para fusionar k arreglos ordenados?
     - en modelo RAM?
       - N
     - en modelo de Memoria Externa con M paginas de tamaño B?
       - $N/B = n$ (si $M > kB$)
       - para Ordenar
   - en modelo RAM de comparaciones
     - $n \lg n$
   - en modelo Memoria Externa con n/B paginas de tamaño B
     - $\Omega(N/B \frac{\lg(N/B)}{\lg(M/B)})$
     - que se puede notar mas simplamente $\Omega(n \lg_m n)$
   - Prueba:
     - en vez de considerar el problema de ordenamiento, supponga que el arregla sea una permutacion y considera el problema (equivalente en ese caso) de identificar cual permutacion sea.
     - inicialemente, pueden ser n! permutaciones.

        * supponga que cada bloque de B elementos sea ya ordenado (impliqua un costo de al maximo n=N/B accessos a la memoria externa).
        * queda $N!/((B!)^n)$ permutaciones posibles.
        * para cada accesso a una pagina de memoria externo,
    - con M entradas en memoria primaria
    - B nuevas entradas se pueden quedar de $\binom{M}{B} = \frac{M!}{B!(M-B)!}$ maneras distintas
    - calcular la union de los $M+B$ elementos reduce la cuantidad de permtuaciones por un factor de $1/\binom{M}{B}$
    - despues de t accessos (distintos) a la memoria externa, se reduci la cuantidad de permutaciones a $N!/((B!)^n\binom{M}{B}^t)$
    - cuanto accessos a la memoria sean necesarios para que queda al maximo una permutacion?
        * $N!/((B!)^n\binom{M}{B}^t)$ debe ser al maximo uno.
        * usamos las formulas siguientes:
            · $\log(x!) \approx x \log x$
            · $\log \binom{M}{B} \approx B \lg \frac{M}{B}$
    - BONUS: Para ordenar strings, un caso particular (donde la
- $\Omega(N_1/B \log_{M/B}(N_1/B) + K_2 \lg_{M/B} K_2 + N/B)$
- donde
    - N_1 es la suma de los tamanos de las caldenas mas cortas que B
    - K_2 es la cuantidad de caldenas mas largas que B

3. Ordenar en Memoria Externa N elementos (en $n = N/B$ paginas) http://en.wikipedia.org/wiki/External$_sorting$

    - Usando dictionarios o colas de prioridades en memoria externa
        - $N \lg N / \lg B = N \log_B N$
        - No es "tight" con la cota inferior
        - impliqua
            * o que hay un mejor algoritmo
            * o que hay una mejor cota inferior
            * Queda un algoritmo de ordenamiento: MergeSort

- usa la fusion de $m-1$ arreglos ordenados en memoria externa:
  (a) carga en memoria principal $m-1$ paginas, cada una la primera de su arreglo.
  (b) calcula la union de estas paginas en la pagina m de memoria principal,
    * botando la pagina cuando llena
    * cargando una nueva pagina (del mismo arreglo) cuando vacilla
  (c) La complejidad es n accessos.
- Algoritmo:
  (a) ordena cada de las n paginas -> n accessos
  (b) Cada nodo calcula la union de m arreglos y escribe su resultado, pagina por pagina, en la memoria externa.
- Analisis:
  * Cada nivel de recurencia costa n accessos
  * Cada nivel reduce por $m-1$ la cantidad de arreglos
  * la complejidad total es de orden $n \log_m n$ accessos. (tight)

4. **BONUS** Ahora, cual es la cota inferior para una cola de prioridad?

  - una cola de prioridad se puede usar para ordenar (con N accessos)
  - hay una cota inferior para ordernar de $n \log_m n$
  - entonces????