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

Auxiliar/Repaso/Consulta 6 - "Amortizado y Flashback"

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

5 de Mayo del 2017

P1. Move-to-Front (MF)

En el siguiente problema "online" tenemos una lista con n elementos, en la cual accesar al i-ésimo elemento tiene costo i y al accesar a un elemento se puede además intercambiarlo con elementos adyacentes a costo 1 cada intercambio.

El algoritmo **Move-to-Front** propone para enfrentar este problema que al accesar a un elemento este se intercambia con todos los elementos anteriores a el en la lista, de modo que si se accesa al k-ésimo elemento, a **Move-to-Front** le cuesta $2k - 1$.

Muestre que el costo de **Move-to-Front** es siempre menor o igual a 4 veces costo de cualquier otro algoritmo que enfrente el problema.

P2. Colas de Prioridad en Memoria Secundaria

Muestre los algoritmos de inserción y borrado para memoria secundaria vistos en cátedras. Realice un análisis amortizado de la operación de inserción y de la operación de eliminación por separado. Concluya que el análisis realizado no rompe la cota de ordenación en memoria secundaria.

P3. Consultas Tarea 1

Soluciones

P1. Sea OPT el algoritmo que dada una secuencia de accesos hace los intercambios que minimizan el costo total de accesos e intercambios, este algoritmo es irrealizable en la práctica, pues este es un problema "online" y los Algoritmos que lo resuelven NO conocen el input de antemano[1]😈. Sin embargo, si mostramos que MF (costo de Move-to-Front) es menor o igual a 4 veces OPT (costo del OPT), mostraremos que es menor o igual a 4 veces el costo de cualquier algoritmo que resuelva el problema. Para mostrar lo anterior veremos que el costo amortizado de MF es menor o igual al costo de OPT.

Definimos el potencial ϕ como el número de inversiones entre la lista de MF y la de OPT (🎩), en donde una inversión es un par de elementos (x, y) que aparecen en distinto orden relativo en cada uno de las listas[2].
Al comienzo el potencial es 0, pues ambas listas parten iguales y este potencial es siempre no negativo, por lo que cumple con los requisitos para ser potencial ($\phi_i \geq \phi_0$).
Para analizar el cambio de potencial $\Delta\phi$ lo separaremos en el cambio que se produce luego de que MF mueve el elemento accesado al principio $\Delta_1\phi$ y el cambio producido luego de que OPT hace sus intercambios $\Delta_2\phi$. ($\Delta\phi = \Delta_1\phi + \Delta_2\phi$).

Sea x el elemento accesado, k su posición en la lista de MF e i su posición en la lista de OPT. Entonces, al mover el elemento x al principio de la lista de MF se crean a lo más $i-1$ inversiones (los que están adelante de la lista de OPT) y como habían $k-1$ delante de x al menos se destruyen $k-1-(i-1)$ inversiones. De este modo se concluye que $\Delta_1\phi \leq 2 \times ((i-1) - (k-1-(i-1))) = 4i - 2k - 2$. Por otro lado, si OPT hace w intercambios estos crearán a lo más w inversiones, y por lo tanto $\Delta_2\phi \leq 2w$.

De este modo el costo amortizado de MF estará acotado por:

$$\begin{aligned} ca &= cr + \Delta\phi \\ &= (2k-1) + \Delta_1\phi + \Delta_2\phi \\ &\leq 2k - 1 + 4i - 2k - 2 + 2w \\ &= 4i + 2w - 3 \\ &\leq 4(i+w) \\ &= 4cr_{OPT} \end{aligned}$$

Finalmente como esto se cumple para todos los accesos tenemos que $MF_{amor} \leq 4 \cdot OPT$.

[1] Se puede mostrar que cualquier algoritmo "online" tiene un peor caso con la técnica de Adversario.
[2] En una aparece x antes que y, y en la otra al revés y viceversa.

P2. La cota inferior para el costo de las operaciones de una cola de prioridad en memoria secundaria es $\Omega\left(\frac{1}{B}\log_m n\right)$, pues si fuese asintóticamente menor podríamos insertar los elementos a la cola, después extraerlos y así ordenar en tiempo asintóticamente menor a $n\log_m n$, lo que es una contradicción.

Se propone tener dos heaps en memoria interna:

- Un heap de $M/2$ elementos denominado heap de inserción. Los elementos se insertan en este heap, si este se llena se ordenan sus elementos y envían a un archivo de tamaño $M/2$. Si hay k archivos de $M/2$ se les hace un $k-merge$ en un archivo de tamaño $kM/2$. Del mismo modo, si hay k de estos archivos, se convierten en uno de tamaño $k^2M/2$ y así en adelante.
  Supongamos que al insertar N elementos se llega a tener elementos de tamaño $k^rM/2$. Por lo tanto:

$$N \geq k^rM/2 \Rightarrow r \leq \log_k(2N/M) \tag{1}$$

- Tenemos un buffer de tamaño B de cada uno de los archivos que tenemos en memoria secundaria y formamos otro heap con los mínimos de estos buffers. Al eliminar extraemos el mínimo entre los dos heaps, si al hacer esto extraemos del segundo heap debemos insertar el siguiente elemento del buffer del archivo de donde venía el elemento extraído, si el buffer está vacío, simplemente traemos otro buffer a memoria principal del archivo correspondiente.
  De lo anterior se concluye que estos buffers no pueden llenar lo que nos queda de memoria, es decir:

$$rkB \leq M/2 \tag{2}$$

  Con esto tenemos lo siguiente:
  - Una secuencia de N eliminaciones tiene costo $\mathcal{O}(N/B)$ y un costo individual de $\mathcal{O}(1/B)$, pues las únicas operaciones que se hacen en disco son reponer los buffers de los archivos.
  - En el caso de una secuencia de inserciones, un elemento insertado se mueve por a lo más r archivos, por lo que cada inserción tiene un costo de $\mathcal{O}(r/B)$. De (1) se concluye que $k \in \mathcal{O}(m)$ y usando (2) concluímos que $r \in \mathcal{O}(\log_m n)$ y por lo tanto la inserción tiene costo $\mathcal{O}\left(\frac{1}{B}\log_m n\right)$.

  Si bien las eliminaciones rompen con la cota dada al principio 😨, esto no es un problema, pues para ordenar primero se deben insertar los elementos (y luego eliminar) de la cola y estas con asintóticamente iguales a la cota anteriormente impuesta.