Universidad del Valle de Guatemala
Análisis y Diseño de Algoritmos - Sección 10
Christopher Sandoval 13660
13/05/2020

Tarea 5
Análisis amortizado

1. **Agregamos al contador binario una operación de decremento: *Decrement*. Identifique el worst-case scenario realista para una secuencia de n operaciones *Increment* y *Decrement* combinadas, y provea una cota superior para su tiempo de ejecución. Suponga que el contador trabaja con una lista de k bits.**
   La cota superior de este tiempo de ejecución es de $O(nk)$. Si se tiene una lista de k bits, el peor escenario es que se modifiquen todos los k bits con cada operación de incremento o decremento. Entonces se tendría que para cada k bit, se aplicarían n operaciones intercaladas que modificarían cada bit.

2. **Supongamos que a una pila con multipop agregamos la operación multipush(k, A) que hace una secuencia de operaciones push con los primeros k elementos en el arreglo A. En este caso, ¿se mantiene el costo amortizado de $O(1)$ por operación? Explique.**
   Si realizamos una secuencia de n operaciones multipop y multipush, el costo de esta secuencia sería ahora $O(n^2)$ y ya no de $O(n)$. Considerando esto, el costo amortizado por cada operación sería $\frac{O(n^2)}{n} = O(n)$, es decir lineal.

3. **Considere una pila común y corriente cuyo tamaño nunca excede k. Luego de k operaciones push y/o pop se efectúa un backup automático, copiando toda la pila. El costo de copiar m elementos es m. Demuestre con el accounting method que cualquier secuencia de n operaciones entre push y pop (con backups cada k operaciones) costará $O(n)$.**
   Necesitamos que ya sea push o pop tengan un costo amortizado mayor al costo original que tienen, ya que debemos poder pagarle a la operación de backup de cada elemento. Tomando esto en cuenta, se asigna un costo amortizado de 2 a push para que existan créditos con los cuales pagar el backup. Sin embargo, no queremos que pop consuma estos créditos extra, por lo que su costo permanecerá igual, es decir de 1. Luego de cualquier secuencia de k operaciones tendremos k créditos para pagar cualquier backup. Si tenemos una secuencia de n operaciones, ésta costará como máximo (si todas son push) de 2n, por ende $O(n)$.

4. **Suponga que al contador binario de los ejemplos se le agrega la operación reset que busca y convierte todos los 1 en 0, uno por uno a partir del bit menos significativo. Demuestre con el accounting method que cualquier secuencia de n operaciones entre increment y reset toma un tiempo de ejecución de $O(n)$. Considere que el contador inicia desde 0 y que cada revisión y cada modificación de un bit toma $\Theta(1)$.**

   Costo 2 increment: debemos incluir un crédito para pagar la operación de increment. Para pagar todos los 1 que pasan a 0, hacemos que increment almacene un crédito extra por aplicar su operación. De esta forma garantizamos que por cada 1 exista un crédito extra que pague por su regreso a 0.

   Costo 3 increment: debemos incluir un crédito extra por cada bit que haya sido modificado desde el último reset para poder tener crédito acumulado destinado a para el siguiente reset. Es decir que como reset recorre todo el arreglo, necesita un crédito por cada posición que visita. De esta forma tendremos dos créditos por cada bit que se convierte en 1, suficiente para pagar una operación de reset.

   Costo 4 increment: no podemos garantizar que siempre habrán k créditos acumulados para pagar una operación reset, por lo que no es bueno que recorra todo el arreglo hasta llegar al bit más significativo. Se debe agregar una operación a increment de costo 1 que verifique si la última posición modificada es más grande que la anterior. Entonces se actualizará la posición del bit más significativo con costo 1. Hay que tener suficientes créditos para esto también.

   Costo 1 reset: debemos llevar registro de cuál es el bit más significativo para que reset lo cambie a 0 cuando sea ejecutado.

   Si increment tiene costo 4 y reset 1, cualquier secuencia de n operaciones tendrá costo amortizado máximo de 4n, por ende $O(n)$.

5. **Tenemos una función de potencial Φ tal que $\Phi(D_i) \geq \Phi(D_0)$ para todo i, pero $\Phi(D_0) \neq 0$. Defina una función Φ' tal que $\Phi'(D_0) = 0$, $\Phi'(D_i) \geq \Phi'(D_0)$, $\forall_i \geq 1$, y que los costos amortizados obtenidos con Φ' sean los mismos que con Φ.**

   Si solamente modificamos la función de potencial, se podrán mantener los costos amortizados de manera que la nueva función asigne los mismos valores a cada cambio de potencial. Es decir que debemos igualar las funciones $\Phi(D_j) - \Phi(D_i) = \Phi'(D_j) - \Phi'(D_i)$ para que se mantengan los costos amortizados. Si trabajamos a partir de esta ecuación, tenemos que $\Phi'(D_j) - \Phi'(D_i) = (\Phi(D_j) - \Phi(D_0)) - (\Phi(D_i) - \Phi(D_0))$. Si consideramos que $\Phi'(D_i) = \Phi(D_i) - \Phi(D_0)$, terminamos con que $\Phi'(D_0) = \Phi(D_0) - \Phi(D_0) = 0, \Phi'(D_i) \geq \Phi'(D_0)$. Notamos que los cambios de potencial son iguales a la función original para cada operación.

6. **Un min-heap binario es un árbol binario completo (todos sus niveles están llenos y sus hojas se llenan de izquierda a derecha) en donde cada nodo es menor que todos sus hijos. El min-heap tiene una operación de inserción llamada insert, pero consideremos además la función extract-min, que reemplaza la raíz por el último nodo del árbol y luego la intercambia con el menor de sus hijos. Supongamos que ambas operaciones tienen un tiempo de ejecución real $O(\log_2 n)$. Provea una función de potencial según la cual el costo amortizado de insert sea $O(\log_2 n)$ y el de extract-min sea $O(1)$. Demuestre que su función de potencial funciona.**

   Tenemos que costo real de las dos operaciones es de $O(log_2 n)$, y también conocemos que el costo amortizado de inserción debe ser $O(log_2 n)$ y de extract-min $O(1)$. En base a esto podemos armar las ecuaciones de costo amortizado para determinar las funciones de potencial que las cumplan. Las ecuaciones de costo amortizado nos dan la siguiente información:

   $a_i = O(log_2 n) + (\Phi(D_i) - \Phi(D_{i-1})) = O(log_2 n)$ nos indica que el cambio de potencial de inserción tiene orden $O(log_2 n)$ o menor.

   $a_i = O(log_2 n) + (\Phi(D_i) - \Phi(D_{i-1})) = O(1)$ nos indica que el cambio de potencial de extract-min tiene que ser orden $O(log_2 n)$ y negativo.

   Por otro lado, sabemos que el potencial del árbol debe reducirse $O(log_2 n)$ al perder el último nodo, entonces el nivel del nodo eliminado es $O(log_2 n)$. Basándonos en esto, para determinar la función del cambio de potencial en todo el árbol hay que sumar:

$$\Phi(D_i) - \Phi(D_{i-1}) = \sum_{i=1}^{n-1} p_i - (\sum_{i=1}^{n-1} p_i + O(log_2 n)) = - O(log_2 n)$$

   Finalmente para comprobar que esta función de potencial funciona, reemplazamos en las ecuaciones antes mencionadas. En la ecuación de extract-min, el resultado al reemplazar es $O(1)$ y en la ecuación de inserción es $O(log_2 n)$, comprobando que sí es correcta la función de potencial.