Auxiliar 4: Análisis Amortizado

CC4102 - Diseño y Análisis de Algoritmos
Profesor: Gonzalo Navarro
Auxiliar: Jorge Bahamonde
Ayudantes : Sebastián Ferrada, Willy Maikowski

24 de Octubre de 2016

1. Suponga que tenemos un contador de modo que el costo de incrementar el contador es igual al número de dígitos (en el caso binario, bits) que deben ser modificados. No es difícil mostrar que si el contador comienza en 0, el costo amortizado de operaciones de incremento es de $O(1)$ por operación. En este problema buscamos implementar un contador que permite *incrementos* y *decrementos*. Sólo consideraremos secuencias de operaciones que mantienen el contador en un valor no negativo.

   (a) Muestre que incluso en secuencias que mantienen el contador con un valor no negativo, es posible que una secuencia de n operaciones que comience con el contador en 0 y que permita incrementos y decrementos tenga un costo amortizado de $\Omega(\log n)$ por operación (equivalentemente, un costo total de $\Omega(n \log n)$).

   (b) Para arreglar este problema, considere el siguiente *sistema redundante de dígitos ternarios*. Un número se representa como una secuencia de *trits*, que, como su nombre lo indica, pueden tomar tres posibles valores: 0, +1 o −1. El valor de un número $t_{k-1}, ..., t_0$ (donde cada t_i es un trit) se define como

   $$\sum_{i=0}^{k-1} t_i 2^i$$

   El proceso de incrementar un número de este tipo es análogo al de la operación en números binarios. Se añade 1 al trit menos significativo; si el resultado es 2, se cambia el trit a 0 y se propaga una *reserva* hacia el siguiente trit. Se repite el proceso hasta que no exista carry.

   El proceso de decrementar es similar: se resta 1 al trit menos significativo; si el resultado es −2, se reemplaza por 0 y se "pide" al siguiente trit.

   Mediremos el costo de un incremento o decremento como el número de trits que se ve alterado. Comenzando de 0, se realiza una secuencia de n incrementos y decrementos (sin un orden particular).

   Demuestre que, utilizando esta representación, el costo amortizado por operación es $O(1)$ (equivalentemente, que el costo total para las n operaciones es $O(n)$).

2. Un *quack* es una mezcla entre queues y stacks. Puede ser visto como una lista de elementos, escrita de derecha a izquierda, que soporta las siguientes operaciones:

   - QPush(x) agrega x en el extremo izquierdo.
   - QPop(x) remueve y retorna el elemento más a la izquierda.
   - QPull(x) remueve y retorna el elemento más a la derecha.

Implemente un quack usando tres stacks de modo que cada operación tenga un costo amortizado constante, considerando que estos stacks son capaces de ejecutar las operaciones SPush y SPop en tiempo constante. Almacene los elementos sólo una vez en cualquiera de los 3 stacks, y utilice a lo más $O(1)$ de memoria adicional.

3. Un *multistack* consiste en una serie (potencialmente infinita) de stacks $S_0, ..., S_{t-1}$ donde el j-ésimo stack puede almacenar hasta 3^j elementos. Todas las operaciones de *push* se realizan inicialmente sobre S_0.Cuando se desea *pushear* un elemento en un stack lleno S_j, se vacía este stack en el siguiente, S_{j+1} (posiblemente repitiéndose la operación de forma recursiva). Considere que las operaciones de *push* y *pop* en los stacks individuales tiene costo 1.

   (a) En el peor caso, ¿cuánto cuesta una operación de *push* en esta estructura?

   (b) Demuestre que el costo amortizado de una secuencia de n operaciones de *push* en un multistack inicialmente vacío es de $O(n \log n)$. Utilice la siguiente función de potencial:

$$\Phi = 2 \sum_{j=0}^{t-1} N_j \cdot (\log_3 n - j)$$

   donde N_j es el número de elementos en el j-ésimo stack.

   (c) Demuestre lo mismo para cualquier secuencia de *pushes* y *pops*.