Auxiliar 6

Análisis amortizado

Profesores: Benjamín Bustos, Gonzalo Navarro
Auxiliares: Sergio Rojas, Pablo Skewes

P1. [P1 - C1 - 2023-2]

Se ha implementado una cola común, con las operaciones **enqueue** y **dequeue**, usando dos pilas, S_1 y S_2. La idea es que el contenido de la cola sea S_1 leído de arriba hacia abajo, seguido de S_2 leído de abajo hacia arriba. Para encolar (enqueue(x)), se apila x en S_2 (push(S_2, x)). Para descolar ($x =$ dequeue()) se saca x de S_1 ($x =$ pop(S_1)). El problema es que S_1 puede estar vacía. En ese caso, antes del pop, se pasan todos los elementos de S_2 a S_1: while not empty(S_2) do push(S_1, pop(S_2)).

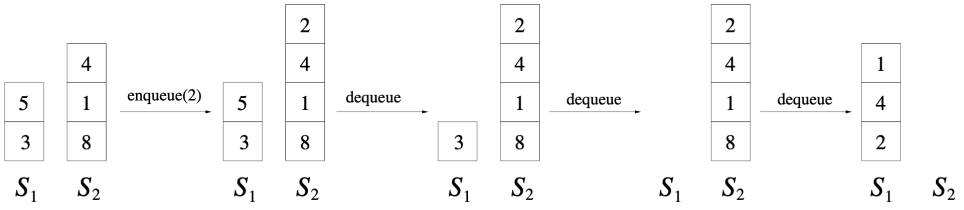

Figura 1: Representación de la cola implementada con dos pilas

Demuestre que esta implementación ofrece tiempo constante amortizado para **enqueue** y **dequeue**. Mida el costo de las operaciones como la cantidad de invocaciones a **push** y **pop**.

Solución:

Análisis global: Se puede argumentar que cada elemento entra a la cola por S_2, pasa una vez a S_1, y sale por S_1, lo que toma 4 operaciones de **push** y **pop**. Si hay n operaciones en total, entonces se insertan a lo sumo n elementos en la cola, por lo que el costo total no puede superar $4n$.

Contabilidad de costos: En el **enqueue** podemos cobrarle 3 a cada elemento que se inserta: 1 por el push(S_2) inicial, y 2 porque en el futuro puede moverse a S_1 con un pop(S_2) y push(S_1). Al **dequeue** podemos cobrarle 1. (O le podemos cobrar 1 al **enqueue** y 3 al **dequeue**.)

Función potencial: Podemos definir φ como $2 \cdot$ altura(S_2). Entonces **enqueue** tiene un costo amortizado de 3 y **dequeue** de 1. Cuando toca pasar todo S_2 a S_1, el costo real es $2 \cdot$ altura(S_2), pero $\Delta\varphi = -2 \cdot$ altura(S_2), por lo cual el costo amortizado es constante.

P2. [P1 - C2 - 2024-2]

Volviendo al problema del contador binario de k bits visto en clase, donde se demostró que el costo amortizado de la operación **INCREMENTAR** es constante, se desea agregar las operaciones **RESET**, que vuelve a dejar el número binario en 0, y **DECREMENTAR**, que decrementa en 1 el número en caso de no ser cero.

a) (**2pt**) La operación **RESET** se implementa manteniendo todo el tiempo la posición max del bit 1 más significativo en el número binario (la posición menos significativa es la 1 y la más significativa es k, con max $= 0$ si el número binario es cero). Cuando se realiza **INCREMENTAR**, se debe verificar si

es necesario incrementar max, y en ese caso hacerlo. En una operación **RESET**, se van cambiando los 1s a 0s desde la posición max a la posición 0, decrementando max en cada caso.

Considere que, al igual que los bit flips, cada incremento o decremento de max cuesta 1. Muestre que el costo amortizado de las operaciones **RESET** e **INCREMENTAR** es $O(1)$ cuando el contador binario parte en 0.

b) (**1pt**) Con respecto a la operación **DECREMENTAR**, muestre que una secuencia de operaciones **INCREMENTAR** y **DECREMENTAR** no tiene costo constante amortizado. Para ello, produzca una secuencia de largo m (donde m pueda ser arbitrariamente grande), cuyo costo no sea $O(m)$. Recuerde que consideramos que k no es una constante.

Solución:

Se puede demostrar de varias formas que el costo amortizado es 4. Con contabilidad de costos, en vez de cobrar 2 por cada operación de incremento, se cobrará 4: dos para cubrir el costo de las inversiones de bit al incrementar el número (como en el análisis visto en cátedra), uno por el posible incremento de max al poner el bit en 1, y finalmente uno para cubrir el paso por ese bit en un futuro **RESET**, decrementando max. Cada bit que está a la derecha del puntero tuvo que ser en algún momento el bit más significativo, dado que el contador parte en cero. En ese momento se guardó 1 de crédito para ese bit. Ese crédito guardado (que se mantiene incluso si el bit se modificó a 0 posteriormente, porque eso se paga con los primeros dos créditos que guarda **INCREMENTAR**) se usa para pagar el decremento de max cuando pasa por la posición. Con ello, el costo de **RESET** es cero, pues los incrementos pagan tanto los flips de 1 a 0 como los decrementos de max.

Una segunda solución es usar la función potencial $\varphi = \text{ones} + \text{max}$, donde ones es la cantidad de 1s en el número binario (como en clase). Un **INCREMENTAR** sobre un número terminado en l 1s realiza $l+1$ flips, y posiblemente incrementa max, con lo que $c_i \leq l+2$. Por otro lado, incrementa ones en $1-l$, y posiblemente incrementa max, con lo cual $\Delta\varphi_i \leq 2-l$. El costo amortizado es entonces $\hat{c_i} = c_i + \Delta\varphi_i \leq l+2+2-l = 4$. Por otro lado, la operación **RESET** cuesta exactamente $c_i = \text{ones} + \text{max}$, y deja el número binario y max en cero, por lo cual $\varphi_i = 0$ y $\Delta\varphi_i = -\text{ones} - \text{max}$. El costo amortizado de **RESET** es entonces $c_i + \Delta\varphi_i = 0$.

Consideremos números de k bits, que pueden representar hasta $2^k - 1$. Una secuencia de operaciones que primero realice 2^{k-1} **INCREMENTAR**, y luego una secuencia de $\frac{m}{2}$ pares consecutivos **DECREMENTAR** e **INCREMENTAR**, pasa de $2^{k-1} = 1000...0$ a $2^{k-1} - 1 = 0111...1$ y viceversa m veces, a costo k en cada caso. El costo total es entonces $2^{k-1} + km = O(n + m \log n)$, donde $n = 2^k$.