Analisis y diseño de algoritmos. Análisis Amortizado

Juan Gutiérrez

November 26, 2021

1 Dos problemas de motivación

1.1 Problema de operaciones en pilas

Recordemos que las pilas son estructuras de datos que admiten dos operaciones.

- PUSH(S,X): adiciona el objeto x en el tope de la pila S.
- POP(S): devuelve el objeto en el tope de la pila, y además lo retira. Solo se puede usar si $S \neq \emptyset$.

  Ambas operaciones consumen tiempo $O(1)$. Por simplicidad, suponga que tienen *costo de ejecución* igual a 1.

  Considere una nueva operación:

  MULTIPOP(S, k)
  while not STACK-EMPTY(S) and $k > 0$
  POP(S)
  $k = k - 1$

MULTIPOP(S, k) recibe una pila S y un entero positivo k y retira $\min\{k, |S|\}$ elementos de la pila. (Invoca a la función auxiliar STACK-EMPTY(S), la cual devuelve true si $S = \emptyset$ y false en caso contrario.)

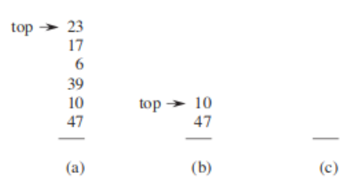

Figure 1: Tomada del libro Cormen, introduction to algorithms. Aplico MULTIPOP con $k = 4$ y luego con $k = 10$.

Tiempo de ejecución: $O(\min\{k, |S|\})$.

Problema Pilas. Dada una pila vacía S a la cual se le aplican una secuencia de n operaciones Push, Pop, o Multipop. ¿Cual es el tiempo de ejecución total que toman las n operaciones?

Primer análisis: Tendríamos n operaciones, cada una de las cuales tiene tiempo de ejecución $O(n)$ (porque el tamaño de la pila es como máximo n), entonces el tiempo de ejecución total es $O(n^2)$.

Podemos hacer un análisis más fino, usando análisis amortizado, y demostrar que el tiempo de ejecución es $O(n)$.

1.2 Problema del contador binario

Decimos que un número x está representado en un arreglo $A[0..k-1]$ de ceros y unos si $x = \sum_{i=0}^{k-1} 2^i \cdot A[i]$. Dicho arreglo es llamado *contador*.

Considere la siguiente operación.

Increment(A)
$i = 0$
while $i < k$ and $A[i] == 1$
    $A[i] = 0$
    $i = i + 1$
if $i < A.length$
    $A[i] = 1$

El algoritmo Increment recibe un arreglo $A[0..k-1]$ de ceros y unos que representa a un número x. Modifica A de manera tal que ahora A representa al número $(x+1) \mod 2^k$.

¿Cual es el tiempo de ejecución de Increment?. Un primer análisis nos da un tiempo de ejecución $O(k)$. Sin embargo, el bucle termina luego de encontrar el primer cero. Entonces el tiempo de ejecución es $O($"posición del primer cero más uno"$)$. O también, $O($"número de bits cambiados"$)$.

Problema Contador. Dado un contador A que representa a 0 inicialmente, encontrar el tiempo de ejecución total luego de hacer n llamadas a Increment.

Counter value	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	Total cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19
12	0	0	0	0	1	1	0	0	22
13	0	0	0	0	1	1	0	1	23
14	0	0	0	0	1	1	1	0	25
15	0	0	0	0	1	1	1	1	26
16	0	0	0	1	0	0	0	0	31

Figure 2: Tomada del libro Cormen, introduction to algorithms.

Un primer análisis nos da un tiempo de ejeución de $O(nk)$, ya que cada llamada a increment es $O(k)$. Sin embargo, utilizando análisis amortizado, veremos que el tiempo de ejecución es $O(n)$.

2 Análisis agregado

2.1 Problema de operaciones en pila

Recordemos la definición de costos:

- Costo de Push(S, x): 1
- Costo de Pop(S, x): 1
- Costo de Multipop(S, k): $\min\{|S|, k\}$

Teorema 2.1. *El costo de una secuencia de n operaciones* Push*,* Pop *o* Multipop *a partir de una pila vacía es $O(n)$.*

Proof. Sea a el costo total de las operaciones Push. Sea b el costo total de las operaciones Pop. Sea c el costo total de las operaciones Multipop.

Como POP() nunca es llamado con la pila vacía, además cada elemento que ha sido puesto en la pila, es quitado como máximo una vez. Luego $a \geq b + c$. Luego, el costo total es $a + b + c \leq 2a \leq 2n$. $\square$

Luego, del Teorema 2.1, el costo amortizado de cada operación es $O(1)$.

2.2 Problema de contador binario

Teorema 2.2. *El costo total en n llamadas a* INCREMENTA(A) *a partir de un arreglo A que representa a 0, es $O(n)$.*

Proof. Sea $A = A_0, A_1, A_2, \ldots, A_n$ la secuencia que representa los cambios hechos en A.

Para cada i, sea $costo(A_i) =$"número de bits que cambian de A_i a A_{i+1}". Luego, el costo total es

$$\begin{aligned}
\sum_{i=0}^{n-1} costo(A_i) &= \sum_{i=0}^{n-1} (\text{“número de bits que cambian de } A_i \text{ a } A_{i+1}\text{”}) \\
&= \sum_{j=0}^{k-1} (\text{“número de veces que cambia el bit } j\text{”}) \\
&= \sum_{j=0}^{k-1} |\{i : \text{“el bit } j \text{ cambia de } A_i \text{ a } A_{i+1}\text{”}\}| \\
&= \sum_{j=0}^{k-1} \lfloor n/2^j \rfloor \\
&\leq n \sum_{j=0}^{\infty} 1/2^j \\
&= 2n.
\end{aligned}$$

□

Del Teorema 2.2, el costo amortizado de cada operación es $O(1)$.

3 Método de recargas

A cada operación le asignamos un costo amortizado, que excede su costo real ("credito") o es menor que su costo real.

Si $\hat{c_i}$ es el costo amortizado y c_i es el costo real. Se debe cumplir que, para una secuencia $c_1, c_2, \ldots, c_n$ de tareas,

$$\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{c_i}$$

3.1 Problema de operaciones en pila

Asignamos los siguientes costos amortizados: PUSH=2, POP=0, MULTIPOP=0.

Teorema 3.1. *Sea una secuencia de n operaciones PUSH, POP y MULTIPOP comenzando en una pila vacía. El costo total es menor o igual que $2n$.*

Proof. Sea

$$\hat{c}_i = \begin{cases} 2 & \text{si la } i\text{-ésima operación es PUSH} \\ 0 & \text{caso contrario} \end{cases}$$

Sea c_i el costo real de la i-ésima operación.

Note la siguiente invariante: "al final de la i-ésima operación, se tiene que

$$\sum_{j=1}^{i} \hat{c}_j - \sum_{j=1}^{i} c_j \geq |S_i|'',$$

donde S_i es el contenido de la pila luego de la i-ésima operación.

Demostración por inducción.

Si $i = 1$ entonces, como la primera operación es PUSH, tenemos que $\hat{c}_1 = 2$, $c_1 = 1$ y $|S_1| = 1$ por lo tanto la invariante se cumple.

Si $i > 1$, por hipótesis de inducción asumimos que $\sum_{j=1}^{i-1} \hat{c}_j - \sum_{j=1}^{i-1} c_j \geq |S_{i-1}|$. Tenemos tres casos:

- Caso 1: la i-ésima operación es Push. En ese caso, $\hat{c}_i = 2$ y $c_i = 1$. Luego

$$\begin{aligned} \sum_{j=1}^{i} \hat{c}_j - \sum_{j=1}^{i} c_j &= \sum_{j=1}^{i-1} \hat{c}_j - \sum_{j=1}^{i-1} c_j + \hat{c}_i - c_i \\ &\geq |S_{i-1}| + \hat{c}_i - c_i \\ &= |S_{i-1}| + 1 \\ &= |S_i| \end{aligned}$$

- Caso 2: la i-ésima operación es Pop.

  En ese caso, $\hat{c}_i = 0$ y $c_i = 1$. Luego

$$\begin{aligned} \sum_{j=1}^{i} \hat{c}_j - \sum_{j=1}^{i} c_j &= \sum_{j=1}^{i-1} \hat{c}_j - \sum_{j=1}^{i-1} c_j + \hat{c}_i - c_i \\ &\geq |S_{i-1}| + \hat{c}_i - c_i \\ &= |S_{i-1}| - 1 \\ &= |S_i| \end{aligned}$$

- Caso 3: la i-ésima operación es Multipop.

  En ese caso, $\hat{c}_i = 0$ y $c_i = |S_i| - |S_{i-1}|$. Luego

$$\begin{aligned} \sum_{j=1}^{i} \hat{c}_j - \sum_{j=1}^{i} c_j &= \sum_{j=1}^{i-1} \hat{c}_j - \sum_{j=1}^{i-1} c_j + \hat{c}_i - c_i \\ &\geq |S_{i-1}| + \hat{c}_i - c_i \\ &= |S_{i-1}| - (|S_{i-1}| - |S_i|) \\ &= |S_i| \end{aligned}$$

Como la invariante se cumple, tenemos que, al final de la n-ésima operación. $\sum_{j=1}^{n} \hat{c_j} - \sum_{j=i}^{n} c_j \geq |S_n| \geq 0$. Luego

$$\sum_{j=i}^{n} c_j \leq \sum_{j=1}^{n} \hat{c_j} \leq 2n.$$

□

3.2 Problema de contador binario

Asignamos los siguientes costos amortizados a cada operación le asignamos costo 2.

Teorema 3.2. *Sea una secuencia de n operaciones INCREMENT a partir de un contaor binario que se encuentra inicialmente en 0. El costo total es menor o igual que $2n$.*

Proof. Sea $0 = A_0, A_1, A_2, \ldots, A_n$ la secuencia de valores del contador A.

Para cada i, sea $\hat{c_i} = 2$.

Sea c_i el costo real de la i-ésima operación. Denotamos por $r(A_i)$ a la posición del primer cero en A_i. Note que $c_i = r(A_{i-1})" + 1$.

Sea $p(A_i)$ la cantidad de bits prendidos (con valor igual a 1) en A_i. Note la siguiente invariante: al final de la i-ésima operación, se tiene que

$$\sum_{j=1}^{i} \hat{c_j} - \sum_{j=1}^{i} c_j \geq p(A_i).$$

Demostración por inducción.

Si $i = 1$ entonces, como A_0 guarda al número 0. Tenemos $c_0 = 1$ y $\hat{c_0} = 2$ y como existe un bit 1 en A_1, la propiedad se cumple.

Si $i > 1$, tenemos que

$$\begin{aligned}
\sum_{j=1}^{i} \hat{c_j} - \sum_{j=1}^{i} c_j &= \sum_{j=1}^{i-1} \hat{c_j} - \sum_{j=1}^{i-1} c_j + \hat{c_i} - c_i \\
&\geq p(A_{i-1}) + \hat{c_i} - c_i \\
&= p(A_{i-1}) + 2 - c_i \\
&= p(A_{i-1}) + 2 - (r(A_{i-1}) + 1) \\
&= (p(A_i) + r(A_{i-1}) - 1)) + 2 - (r(A_{i-1}) + 1) \\
&= p(A_i).
\end{aligned}$$

Por lo tanto, al finalizar las n operaciones, tendríamos que

$$\sum_{j=1}^{i} \hat{c_j} - \sum_{j=1}^{i} c_j \geq 0$$

Lo que implica que $\sum_{j=1}^{n} c_j \leq \sum_{j=1}^{n} \hat{c_j} \leq 2n$

□