Metodología y Tecnología de la Programación

Ingeniería en Informática

Curso 2008-2009

Análisis amortizado

Yolanda García Ruiz	**D228**	**ygarciar@fdi.ucm.es**
Jesús Correas	**D228**	**jcorreas@fdi.ucm.es**

Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid

(elaborado a partir de [COR01] y [BB97])

Bibliografía

- **Importante**: Estas transparencias son un material de apoyo a las clases presenciales y no sustituyen a la bibliografía básica ni a las propias clases presenciales para el estudio de la asignatura
- Bibliografía básica:
  - [COR01] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, *Introduction to Algorithms*, McGraw-Hill, New York, NY, 2ª edición, 2001. Secciones 17.1 a 17.3
  - [BB97] Sección 4.6
- Bibliografía complementaria:

  `http://www.cs.utexas.edu/~vlr/s06.357/notes/lec16.pdf`

Análisis amortizado

1. Introducción
2. Método de la contabilidad
3. Método de la función potencial
4. Tablas dinámicas

Introducción

- En algunas ocasiones el análisis de caso peor es demasiado pesimista
- Normalmente se produce cuando un proceso P tiene tiempos de ejecución diferentes para llamadas sucesivas
- Por ejemplo, puede deberse a que P tiene efectos laterales, que hace que diferentes llamadas a P tengan diferentes comportamientos
- En estos casos, el análisis de caso medio tampoco es el más adecuado, pues no hay que considerar la media de todas las entradas, sino la media entre *llamadas sucesivas*
- En el análisis amortizado **el tiempo requerido para una secuencia de operaciones es promediado entre todas las operaciones realizadas**

Introducción (cont.)

- En el análisis amortizado no intervienen probabilidades
- El análisis amortizado garantiza (una cota superior d)el rendimiento medio de una operación en el caso peor
- Este tipo de análisis suele estar asociado a estructuras de datos dinámicas (como `ArrayList` de Java), gestión dinámica de memoria, operaciones de bases de datos, etc.
- Estas estructuras de datos mantienen un **estado** que persiste entre las distintas llamadas a operaciones sobre la estructura de datos
- Aunque nos referimos genéricamente a un único proceso P, puede ser un conjunto de procesos. Por ejemplo, distintas operaciones sobre una estructura de datos
- Existen dos técnicas fundamentales:
  - Método de la contabilidad
  - Método de la función potencial

Análisis amortizado

1. Introducción
2. $\boxed{\text{Método de la contabilidad}}$
3. Método de la función potencial
4. Tablas dinámicas

Método de la contabilidad

- Se asignan distintos costes amortizados a distintas operaciones, mayores o menores al coste real
- Cuando el coste es mayor que el coste real, la diferencia se asigna a objetos específicos de la estructura de datos en forma de "crédito"
- Este crédito se puede usar posteriormente para ayudar a pagar operaciones cuyo coste amortizado es menor a su coste real
- El coste amortizado debe elegirse de forma que:
  a) el coste total amortizado sea una cota superior del coste total real
  b) esta relación se cumpla para todas las secuencias de llamadas
  c) el crédito total de la estructura de datos sea no negativo en todo momento (no se permite que el coste total real sea mayor al amortizado por anticipado)

Ejemplo: Método de la contabilidad

- Ejemplo: Pila con operación `pop` múltiple

```
proc push(S,d)
   ...
fin proc
fun pop(S)
   ...
fin fun
fun length(S)
   ...
fin fun
```

```
fun multipop(S,k)
   res ← 0
   mientras length(S)>0 Y k>0 hacer
      res ← pop(S)
      k ← k-1
   fin mientras
   devolver res
fin fun
```

- En un análisis de caso peor o caso medio, la complejidad de `multipop` está en $\mathcal{O}(n)$.

Ejemplo: Método de la contabilidad (cont.)

```
crear S //S inicialmente vacía
desde i ← 1 hasta n hacer
   ...
   seleccionar operacion
      1 :
        push(S,d)
      2 :
        d ← pop(S)
      3 :
        l ← length(S)
      4 :
        multipop(S,k)
   fin seleccionar
   ...
fin desde
```

- Un algoritmo que realice diversas operaciones en la pila con una entrada de tamaño n estaría fácilmente en $\mathcal{O}(n^2)$
- Sin embargo, el número de iteraciones del bucle en `multipop` depende del estado de la pila en cada momento.

Ejemplo: Método de la contabilidad (cont.)

- Los costes reales de las operaciones sobre la pila son (utilizando el método de la instrucción característica, por ejemplo):

`push`	1
`pop`	1
`multipop`	$min(k, length(S))$

- En concreto, en una secuencia de n operaciones sobre la pila, se introducen en la pila como máximo n elementos
- Independientemente del número de veces que se llame a `multipop`, el número total de veces que se repite el cuerpo del bucle de `multipop` será menor a n
- Podemos contabilizar el número de elementos que se introducen en la pila añadiendo un crédito extra de 1 por cada operación `push`, para pagar la operación `pop` o `multipop` que elimine el elemento introducido
- Las operaciones `pop` y `multipop` no necesitan tener coste amortizado, pues utilizan el crédito proporcionado por `push`
- Los costes quedan por tanto:

`push`	2
`pop`	0
`multipop`	0

Análisis amortizado

1. Introducción
2. Método de la contabilidad
3. $\boxed{\text{Método de la función potencial}}$
4. Tablas dinámicas

Método de la función potencial

- Se asocia a la estructura de datos una *función potencial* Φ que representa la noción del nivel de organización de la estructura (su "energía potencial")
- Valores más elevados de la función potencial se asocian a estados más desorganizados de la estructura
- Sean:

  D_0 la estructura de datos inicial,

  c_i el coste real de la operación i-ésima sobre la estructura de datos, y

  D_i la estructura de datos después de realizar la operación i-ésima sobre D_{i-1}
- $\Phi(D_i)$ hace corresponder D_i con un número real que es el potencial asociado a D_i (Se puede convenir que $\Phi(D_0) = 0$)

Método de la función potencial (cont.)

- El coste amortizado $\hat{c}_i$ se define como:
$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$
- El coste total amortizado de n operaciones es por tanto:
$$\hat{T}_n = \sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1})) =$$
$$\sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0) = T_n + \Phi(D_n) - \Phi(D_0)$$
donde T_n es el coste total real de n operaciones
- El coste total real es: $T_n = \hat{T}_n - (\Phi(D_n) - \Phi(D_0))$
- **Por tanto, para que se cumpla que $T_n \leq \hat{T}_n$ debe verificarse** $\Phi(D_n) \geq \Phi(D_0), \forall n > 0$
- Es decir, $\hat{T}_n$ es una cota superior del tiempo total necesario para hacer una secuencia de operaciones (si la estructura de datos no queda más organizada que inicialmente)
- Lo más complicado es determinar la función potencial

Ejemplo: Método de la función potencial

- Utilizamos el mismo ejemplo de la pila
- Se puede definir Φ como el número de elementos en la pila en cada momento. $\Phi(D_0) = 0$, y $\Phi(D_i) \geq 0, \forall i > 0$
- e_i es número de elementos de la pila después de realizar la operación i-ésima
- Coste de las distintas operaciones:

$$
\begin{aligned}
\text{push:} \quad & c_i = 1 \\
& \hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1 + (e_{i-1} + 1) - e_{i-1} = 2 \\
\text{pop:} \quad & c_i = 1 \\
& \hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1 + (e_{i-1} - 1) - e_{i-1} = 0 \\
\text{multipop:} \quad & c_i = k' \text{ (donde } k' = min(k, e_{i-1})) \\
& \hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = k' + (e_{i-1} - k') - e_{i-1} = 0
\end{aligned}
$$

Análisis amortizado

Tablas dinámicas

- Algunos lenguajes tienen mecanismos para utilizar tablas (*arrays* dinámicos) para los que no se conoce de antemano el número de objetos que se utilizarán (por ejemplo, `ArrayList` o `Vector` de Java).
- Vamos a ver un ejemplo sencillo de análisis de tablas dinámicas.
- En el caso que vamos a analizar, vamos a considerar que la tabla dinámica se implementa mediante un *array*, y se utilizan dos operaciones básicas:
  - `insertar`, que introduce en la tabla dinámica un elemento que ocupa una posición libre del *array* sobre el que se implementa.
  - `eliminar`, que elimina un elemento en la tabla dinámica, liberando una posición del *array* sobre el que se implementa.

Tablas dinámicas (cont.)

- Si se intentan insertar más elementos de los que caben en la tabla dinámica, se debe reservar un *array* con más capacidad, copiar los elementos del *array* antiguo sobre el nuevo, y por último liberar el *array* antiguo.
- De la misma forma, si se elimina un número suficiente de objetos de la tabla dinámica, se debe reservar un *array* con menos capacidad, copiar los elementos no eliminados todavía y liberar el espacio ocupado por el *array* antiguo.
- Dada una tabla dinámica T no vacía, el **factor de carga** $\alpha(T)$ es la razón entre el número de elementos almacenados y el número de posiciones del *array* sobre el que se implementa.
- Para una tabla vacía el factor de carga es 1.
- Primero vamos a analizar el caso en el que se producen inserciones sobre una tabla dinámica y se necesita realizar la **expansión de la tabla**

Tablas dinámicas – Expansión

- Una tabla dinámica se llena cuando todas las posiciones del *array* sobre el que se implementa están ocupadas: cuando el factor de carga es 1.
- Para que se pueda expandir una tabla dinámica supondremos que podemos reservar regiones de memoria cuando sea necesario (como se hace en C o C++).
- Cuando se inserta un nuevo elemento en una tabla llena, se debe expandir la tabla reservando un *array* con más elementos que el anterior y copiar los elementos del *array* anterior al nuevo
- Una heurística habitual consiste en reservar una tabla con tamaño doble de la tabla anterior.
  - Si solamente se realizan inserciones sobre la tabla, el factor de carga será siempre igual o mayor a $1/2$

Tablas dinámicas – Expansión (cont.)

- Utilizaremos una estructura de datos `tabla` para almacenar la información de la tabla, con tres atributos:
  - **`datos`** contiene una referencia al *array* donde se almacenan los elementos de la tabla
  - **`num`** contiene el número de elementos actualmente en la tabla
  - **`tamaño`** guarda el número total de posiciones en el *array*
- El algoritmo que implementa el método `insertar` es el siguiente:

```
proc insertar(T,x) // T es de tipo tabla ; x es el elemento a insertar
   si T.tamaño = 0 entonces crear T.datos[1..1] ; T.tamaño ← 1
   si T.num = T.tamaño entonces
      crear nuevo_array[1..2*T.tamaño]
      desde i ← 1 hasta T.tamaño hacer nuevo_array[i] ← T.datos[i]
      liberar T.datos
      T.datos ← nuevo_array ; T.tamaño ← 2*T.tamaño
   fin si
   T.datos[T.num+1] ← x ; T.num ← T.num + 1
fin proc
```

Tablas dinámicas – Expansión (cont.)

- En el algoritmo anterior se realizan asignaciones de elementos en el *array* en las líneas 5 y 9.
- Podemos analizar este algoritmo teniendo en cuenta el número de asignaciones que se realizan (se puede considerar que la reserva y liberación de memoria de las líneas 4 y 6 están en un orden de complejidad igual o inferior al del bucle de la línea 5).
- De esta forma, la complejidad de `insertar` es lineal en el caso peor si se considera la insercion de elementos individualmente.
- Vamos a analizar una secuencia de n operaciones de inserción sobre una tabla inicialmente vacía. Calculamos el coste c_i de la inserción i-ésima:
    - Si en el *array* sobre el que se implementa la tabla quedan posiciones disponibles, el coste c_i es constante.
    - Si el *array* está lleno y se produce una expansión de la tabla, entonces el coste es $c_i = i$.

Tablas dinámicas – Expansión (cont.)

- Si se realizan n operaciones de inserción, el coste en el caso peor de cada operación está en $\mathcal{O}(n)$.
- Por tanto, la complejidad de las n operaciones estará en $\mathcal{O}(n^2)$.
- Sin embargo, esta aproximación de la complejidad es muy pesimista, porque las operaciones de inserción de complejidad lineal ocurren pocas veces.
- Podemos utilizar el método de la función potencial.
- definimos una función potencial Φ que valga 0 en la operación que realice una expansión, y que vaya creciendo según se va llenando la expansión realizada.
- Por ejemplo, podemos utilizar la función

$$\Phi(T) = 2 \cdot T.num - T.tamaño$$

Tablas dinámicas – Expansión (cont.)

- La función $\Phi(T) = 2 \cdot T.num - T.tamaño$ cumple las características de una función potencial, pues $\Phi(T) = 0$ cuando T está vacía, y $\Phi(T) \geq 0$ en cualquier otra situación de la tabla dinámica:
  - Inmediatamente **después** de una expansión, $T.num = T.tamaño/2$, por lo que $\Phi(T) = 0$.
  - Inmediatamente **antes** de una expansión, $T.num = T.tamaño$, y por tanto $\Phi(T) = T.num$.
- Para analizar el coste amortizado de la i-ésima operación de inserción, utilizaremos num_i, $tamaño_i$ y ϕ_i para referirnos a los valores correspondientes de los atributos y la función potencial **después** de la i-ésima operación.

Tablas dinámicas – Expansión (cont.)

- Inicialmente, $num_0 = 0$, $size_0 = 0$ y $\Phi_0 = 0$.
- Si la inserción i-ésima no realiza una expansión de la tabla, entonces $tamaño_i = tamaño_{i-1}$ y el coste amortizado es

$$\begin{aligned}\hat{c}_i &= c_i + \phi_i - \phi_{i-1} \\ &= 1 + (2 \cdot num_i - tamaño_i) - (2 \cdot num_{i-1} - tamaño_{i-1}) \\ &= 1 + (2 \cdot num_i - tamaño_i) - (2 \cdot (num_i - 1) - tamaño_i) = 3\end{aligned}$$

- Si la inserción i-ésima realiza una expansión de la tabla, entonces $tamaño_i/2 = tamaño_{i-1} = num_i - 1$ y el coste amortizado es

$$\begin{aligned}\hat{c}_i &= c_i + \phi_i - \phi_{i-1} \\ &= num_i + (2 \cdot num_i - tamaño_i) - (2 \cdot num_{i-1} - tamaño_{i-1}) \\ &= num_i + (2 \cdot num_i - (2 \cdot num_i - 2)) - (2(num_i - 1) - (num_i - 1)) \\ &= num_i + 2 - (num_i - 1) = 3\end{aligned}$$

- En ambos casos el coste amortizado de cada inserción es constante, y por tanto **la complejidad de realizar n inserciones es lineal.**

Tablas dinámicas – Expansión y Contracción

- La operación `eliminar` realiza el borrado de un elemento de una tabla dinámica.
- Si se elimina un número suficiente de elementos de una tabla dinámica, es conveniente realizar la **contracción de la tabla** para evitar mantener una gran cantidad de espacio asignado y sin utilizarse.
- La contracción consiste en reservar un *array* con menos capacidad para la tabla, copiar los elementos no eliminados todavía sobre este *array* y liberar el espacio ocupado por el *array* antiguo.
- Es importante que la tabla dinámica verifique dos propiedades:
  - El factor de carga debe estar acotado inferiormente por una constante,
  - el coste amortizado debe estar acotado superiormente por una constante.
- Antes hemos visto que una estrategia habitual para la expansión de tablas dinámicas consiste en doblar el tamaño de la tabla cuando se llena.

Tablas dinámicas – Expansión y Contracción (cont.)

- Aparentemente, la estrategia más natural en la contracción de tablas consistirá en reducir su tamaño a la mitad cuando su ocupación es menor al 50 %
- Sin embargo, supongamos el siguiente caso:
    1. Se realizan $n = 2^k$ inserciones en la tabla.
    2. A continuación se realiza la siguiente secuencia de inserciones y borrados: I, B, B, I, I, B, B, I, I, B, B, ...
- En este caso, después de (1) el *array* que implementa la tabla tendrá un tamaño de n elementos, todos ocupados.
- En la primera operación de (2) se inserta un elemento sobre una tabla llena, por lo que el tamaño pasa a ser $2n$.
- En las dos siguientes operaciones se eliminan dos elementos de la tabla, por lo que la carga es menor al 50 % y por tanto se reduce el tamaño de nuevo a n.
- En las dos siguientes operaciones se insertan dos elementos que vuelven a provocar la expansión de la tabla a un tamaño $2n$, etc.

Tablas dinámicas – Expansión y Contracción (cont.)

- No parece conveniente utilizar esta estrategia, pues en este escenario la mitad de las operaciones de inserción y borrado de (2) podrían tener un coste lineal.
- El problema de esta situación es que después de una expansión no se hace el número de borrados suficiente para compensar la contracción.
- La forma de solucionarlo consiste en permitir que el factor de carga sea inferior a $1/2$. Por ejemplo, realizando una contracción cuando el factor de carga esté por debajo de $1/4$.
- La contracción consiste en reducir el tamaño del *array* a la mitad: así, después de una contracción el factor de carga vuelve a ser $1/2$.

Tablas dinámicas – Expansión y Contracción (cont.)

- Vamos a utilizar el método de la función potencial para estudiar el coste de una secuencia de n inserciones y borrados sobre una tabla dinámica con estas dos estrategias de expansión y contracción.
- Definimos una función potencial que tome el valor 0 justo después de una expansión o una contracción, y vaya aumentando de valor según aumenta el factor de carga hacia 1, o bien disminuye hacia 1/4. Esta función puede ser:

$$\Phi(T) = \begin{cases} 2T.num - T.tamaño & si\ \alpha(T) \geq 1/2 \\ T.tamaño/2 - T.num & si\ \alpha(T) < 1/2 \end{cases}$$

- Se cumplen las condiciones de la función potencial, pues $\Phi(T) = 0$ cuando T está vacía, y $\Phi(T) \geq 0$ en cualquier otra situación.
- Si la tabla está vacía, $T.num = T.tamaño = 0$, y $\alpha(T) = 1$.
- Además, $T.num = \alpha(T) \cdot T.tamaño$ en cualquier situación de la tabla

Tablas dinámicas – Expansión y Contracción (cont.)

- Sea c_i el coste real de la operación i-ésima,y $\hat{c}_i$ el coste amortizado respecto de la función Φ.
- $num_i, tamaño_i, \alpha_i, \phi_i$ son los valores respectivos después de realizar la operación i-ésima.
- Inicialmente, $num_0 = 0, tamaño_0 = 0, \alpha_0 = 1, \phi_0 = 0$.
- Vemos primero el caso de **operaciones de inserción:**
- **Si** $\alpha_{i-1} \geq 1/2$, el análisis es el mismo que hemos visto antes solamente para expansión, que produce un coste amortizado $\hat{c}_i = 3$.
- **Si** $\alpha_{i-1} < 1/2$ **y** $\alpha_i < 1/2$, el coste amortizado es:

$$\begin{aligned}
\hat{c}_i &= c_i + \phi_i - \phi_{i-1} \\
&= 1 + (tamaño_i/2 - num_i) - (tamaño_{i-1}/2 - num_{i-1}) \\
&= 1 + (tamaño_i/2 - num_i) - (tamaño_i/2 - (num_i - 1)) = 0
\end{aligned}$$

Tablas dinámicas – Expansión y Contracción (cont.)

- **Si** $\alpha_{i-1} < 1/2$ **pero** $\alpha_i \geq 1/2$, el coste amortizado es:

$$
\begin{aligned}
\hat{c}_i &= c_i + \phi_i - \phi_{i-1} \\
&= 1 + (2 \cdot num_i - tamaño_i) - (tamaño_{i-1}/2 - num_{i-1}) \\
&= 1 + (2(num_{i-1} + 1) - tamaño_{i-1}) - (tamaño_{i-1}/2 - num_{i-1}) \\
&= 3num_{i-1} - \frac{3}{2} tamaño_{i-1} + 3 \qquad \text{(sabemos que } num_{i-1} = \alpha_{i-1} tamaño_{i-1}\text{)} \\
&= 3\alpha_{i-1} tamaño_{i-1} - \frac{3}{2} tamaño_{i-1} + 3 \\
&< \frac{3}{2} tamaño_{i-1} - \frac{3}{2} tamaño_{i-1} + 3 = 3
\end{aligned}
$$

- Por tanto, **el coste amortizado de una operación de inserción es a lo sumo 3.**

Tablas dinámicas – Expansión y Contracción (cont.)

- Si la operación i-ésima es una **operación de borrado:**
- En este caso, $num_i = num_{i-1} - 1$
- **Si** $\alpha_{i-1} < 1/2$, debemos considerar si se produce una contracción o no:
  - Si no se produce contracción, entonces $tamaño_i = tamaño_{i-1}$, y el coste amortizado es

$$
\begin{aligned}
\hat{c}_i &= c_i + \phi_i - \phi_{i-1} \\
&= 1 + (tamaño_i/2 - num_i) - (tamaño_{i-1}/2 - num_{i-1}) \\
&= 1 + (tamaño_i/2 - num_i) - (tamaño_i/2 - (num_i + 1) = 2
\end{aligned}
$$

  - Si se produce contracción, entonces $c_i = num_i + 1$ y el coste amortizado es

$$
\begin{aligned}
\hat{c}_i &= c_i + \phi_i - \phi_{i-1} \\
&= (num_i + 1) + (tamaño_i/2 - num_i) - (tamaño_{i-1}/2 - num_{i-1}) \\
&= (num_i + 1) + ((num_i + 1) - num_i) - ((2num_i + 2) - (num_i + 1) \\
&= 1
\end{aligned}
$$

Tablas dinámicas – Expansión y Contracción (cont.)

- **Si** $\alpha_{i-1} \geq 1/2$, se puede demostrar que $\hat{c}_i$ está acotado por una constante (*).
- Por tanto, **la complejidad de realizar n inserciones y/o borrados es lineal.**
- Ejercicio: ¿Cómo sería la demostración de (*)?

Tablas dinámicas – Expansión y Contracción (cont.)

- **Si** $\alpha_{i-1} \geq 1/2$, se puede demostrar que $\hat{c}_i$ está acotado por una constante (*).
- Por tanto, **la complejidad de realizar** n **inserciones y/o borrados es lineal.**
- Ejercicio: ¿Cómo sería la demostración de (*)?
- **Solución:** En este caso, $tamaño_i = tamaño_{i-1}, num_i = num_{i-1} - 1$. Se pueden dar dos situaciones:

a) Si $\alpha_{i-1} \geq 1/2$ y $\alpha_i \geq 1/2$,

$$\begin{aligned}\hat{c}_i = c_i + \phi_i - \phi_{i-1} &= 1 + (2num_i - tamaño_i) - (2num_{i-1} - tamaño_{i-1}) \\ &= 1 + (2num_i - tamaño_i) - (2num_i - 2 - tamaño_i) = 3\end{aligned}$$

b) Si $\alpha_{i-1} \geq 1/2$ y $\alpha_i < 1/2$,

$$\begin{aligned}\hat{c}_i = c_i + \phi_i - \phi_{i-1} &= 1 + (tamaño_i/2 - num_i) - (2num_{i-1} - tamaño_{i-1}) \\ &= 1 + (tamaño_i/2 - num_i) - (2(num_i - 1) - tamaño_i \\ &= 3 + 3/2tamaño_i - 3num_i = 3 + 3/2 \cdot num_i/\alpha_i - 3num_i \\ &> 3 + 3num_i - 3num_i = 3\end{aligned}$$