estudios de postgrado
en computación

Análisis y Diseño de Algoritmos (AyDA)

Isabel Besembel Carrera

ANÁLISIS AMORTIZADO
TABLAS DINÁMICAS

Análisis amortizado

- Garantiza el rendimiento promedio de cada operación en el peor de los casos.
- Es el tiempo requerido para desarrollar una secuencia de operaciones de un TAD promediado sobre todas sus operaciones.
- Métodos:
  - Agregado: En el peor de los casos el costo promedio o costo amortizado por operación es T(n)/n, donde T(n) es el costo total de una secuencia de n operaciones.
  - Este costo amortizado se aplica a cada operación, así existan varios tipos de operaciones en la secuencia. Los costos de cada operación son iguales.

Análisis amortizado

- Ejemplo: Pila[TipoEle: e]
- meterElePila(Pila, TipoEle) y sacarElePila(Pila) cada una con O(1)
- El costo total de una secuencia de n operaciones de ellas es $\theta(n)$
- Si se agrega al TAD Pila una operación sacarElePila(Pila, Entero) que elimina k elementos del tope de la pila o saca todos los elementos si k > |Pila|

12/3/98

Especificación Pila[TipoEle]

1	**Especificación sintáctica:** creaPila() Pila, meterElePila(Pila,TipoEle) Pila, sacarElePila(Pila) Pila, conTopePila(Pila) TipoEle, vacíaPila(Pila) Lógico, destruyePila(Pila) .	**-creaPila()**: Crea una pila vacía. **-meterElePila()**: Ingresa un nuevo elemento a la pila por el tope de la misma. **-sacarElePila()**: Elimina el elemento que está actualmente en el tope de la pila. Si la pila está vacía no hace ninguna eliminación. **-vacíaPila()**: Regresa Verdadero si la pila está vacía. **-destruyePila()**: Destruye la pila. **-conTopePila()**: Devuelve el elemento que se encuentra actualmente en el tope de la pila. Si la pila está vacía devuelve un valor especial que lo indica.
2	**Declaraciones:** TipoEle: e, {TipoEleNoDef}	
3	**Especificación semántica:** vacíaPila(creaPila())=Verdadero vacíaPila(meterElePila(creaPila(),e))=Falso conTopePila(creaPila())=TipoEleNoDef sacarElePila(creaPila())=creaPila()	

Implementación secuencial

<table>
<tr><td colspan="3">PilaSec[TipoEle]
Clases: Entero, Lógico, TipoEle, ArregloDe TipoEle</td></tr>
<tr><td>1</td><td>Superclases: ninguna</td><td rowspan="3">-max: Número máximo de elementos.
-tope: Número actual de elementos.
-p: Arreglo que contiene los elementos actuales en la pila.
-PilaSec(). Constructor. Crea una pila vacía.
-meterElePila(). Transformador. Ingresa un nuevo elemento a la pila.
-sacarElePila(). Transformador. Elimina el elemento que está actualmente en el tope de la pila, si no está vacía.
-conPila(). Observador. Devuelve el elemento que se encuentra actualmente en el tope de la pila, si existe.
-vacíaPila(). Observador. Regresa Verdadero si la pila está vacía, de lo contrario regresa Falso.
-numEle(). Observador. Regresa el tope.
-despliegue(). Observador. Despliega el contenido de la pila.</td></tr>
<tr><td>2</td><td>Estructura: privado:
max : Entero+ = 100
tope : Entero+ = 0
p : Arreglo[100]De TipoEle</td></tr>
<tr><td>3</td><td>Operaciones: público:
PilaSec()
meterElePila(TipoEle: e)
sacarElePila()
conPila(): TipoEle
vacíaPila(): Lógico
numEle(): Entero+
despliegue()</td></tr>
</table>

12/3/98	PilaSec() {pos: tope = 0 ∧ max = 100}	
1	max, tope = 100, 0	**-max, tope.** Definidos en PilaSec.
1	PilaSec[Entero] pila1 ⇒ pila1.max = 100 y pila1.tope = 0	Crea la variable pila1 de enteros.

Constructor vacío, O(1).

12/3/98	vacíaPila(): Lógico {pre: tope ≥ 0 ∧ max = 100} {pos: tope ≥ 0 ∧ max = 100}	
1	regresa (tope = 0)	**-tope.** Definido en PilaSec
1	pila1.vacíaPila() ⇒ 1	La variable pila1 está vacía

Verifica si la pila está vacía, O(1).

12/3/98	numEle(): Entero+ {pre: tope ≥ 0 ∧ max = 100} {pos: tope ≥ 0 ∧ max = 100}	
1	regresa tope	**-tope.** Definido en PilaSec
1	pila1.numEle() ⇒ 0	La variable pila1 tiene 0 elementos

Devuelve el número de elementos actuales en la pila, O(1).

12/3/98	meterElePila(TipoEle: e)	
{pre: e ≠ {TipoEleNoDef} ∧ 0 ≤ tope < max }		{pos: 0 ≤ tope < max }
1	Si (tope = max) entonces Despliegue “Pila llena” sino p (tope) = e tope = tope + 1 fsi	-**e**: TipoEle. Nuevo elemento para ser insertado en la pila. - **tope, max, p**: Definidos en PilaSec.
1 2	pila1=(100, 2, 45, 34) y e = 8 ⇒ pila1=(100, 3, 45, 34, 8) pila1=(100,100, 45, ...) y e = 8 ⇒ pila1=(100,100, 45, ...) / Pila llena	Inserta 8 en pila1 No inserta 8 pues la pila está llena

Ingresa un nuevo elemento en la pila, O(1).

12/3/98	sacarElePila()	
{pre: 0 ≤ tope < max }		{pos: 0 ≤ tope < max }
1	Si (tope = 0) entonces Despliegue “Pila vacía” Sino tope = tope – 1 fsi	- **tope, max, p**: Definidos en PilaSec.
1 2	pila1=(100, 2, 45, 34) ⇒ pila1=(100, 3, 45) pila1=(100, 0) ⇒ Pila vacía	Elimina el elemento en el tope No elimina pues la pila está vacía

Elimina el elemento en el tope de la pila, O(1).

12/3/98	conPila(): TipoEle	
{pre: 0 ≤ tope < max }		{pos: 0 ≤ tope < max }

1	Si (tope = 0) entonces
	  regrese {TipoEleNoDef}
	sino
	  regrese p (tope)
	fsi
		-**tope, max, p**: Definidos en PilaSec.
		-**{TipoEleNoDef}**: Definido en meterElePila().

1	pila1=(100, 2, 45, 34) ⇒ 34	Se obtiene una copia del elemento en el tope
2	pila1=(100, 0) ⇒ {TipoEleNoDef}	Se obtiene el elemento especial

Regresa el valor del elemento en el tope de la pila, O(1).

12/3/98	despliegue ()	
{pre: tope ≥ 0 ∧ max = 100}		{pos: tope ≥ 0 ∧ max = 100}
1	Despliegue “Elementos de la pila en orden inverso”	-**i**: Entero+. Subíndice del lazo.
2	[Despliegue “Elemento(“, i , “)=” , este → p (i)] i = 0, tope –1	-**este**: Apuntador al objeto actual.
1	pila1=(100, 2, 45, 34) ⇒ Elementos de la pila en orden inverso / Elemento(0)=45 / Elemento(1)=34 /	Despliega el contenido de la pila1 con sus comentarios

Despliega los elementos que están en la pila, O(n).

Análisis amortizado

Oct.98 sacarElePila(Entero: k) {pre: $0 \leq k \leq \|Pila\|$}		{pos: $\|Pila\| = \|Pila\|-k$}
1	($\neg$ vaciaPila() $\wedge$ $k \neq 0$) [sacarElePila() $k = k - 1$]	**-vaciaPila(), sacarElePila().** Definidos en Pila.
2	regrese	
1	pila1=(100, 2, 45, 34), k=3 $\Rightarrow$ pila1=(100)	Elimina los tres elementos más cercanos al tope
2	pila1=(100, 2, 45, 34), k=4 $\Rightarrow$ pila1=()	Vacía la pila

Saca k elementos de la pila, si no está vacía, O(k).

Pila con n elementos $\Rightarrow$ costo total = min(n, k) $\Rightarrow$ W(n) = θ(n)
W(n) se mantiene en θ(n), pero en una secuencia de n operaciones donde sacarElePila(k) está incluida $\Rightarrow$ W(n) = $\theta(n^2)$.
El costo amortizado del TAD W(n) = $\theta(n^2)/n = \theta(n)$.

Análisis amortizado

- Método Contable:
- Se asignan costos amortizados a cada operación, colocándole una cantidad mayor o menor a lo que cada operación cuesta actualmente.
- Cuando se le asigna una cantidad mayor a lo que ella cuesta actualmente, la diferencia se asigna al TAD como su *crédito*, el cual puede ser utilizado después para *pagar* por aquellas operaciones con un costo menor o igual al crédito actual.
- Los costos de cada operación pueden ser diferentes.

Análisis amortizado

- El costo total amortizado de una secuencia de operaciones debe ser un límite superior del costo total actual de la secuencia y esta relación debe darse para cualquier secuencia.
- El crédito total del TAD **no** puede ser negativo para cualquier estado del TAD.
- Para cualquier secuencia de n operaciones, el costo total amortizado es un límite superior del costo total actual
- Los créditos no son negativos.

Análisis amortizado

	Operación	Costo actual	Costo amortizado
Ejemplo:	meterElePila(Pila, TipoEle)	1	2
	sacarElePila(Pila)	1	0
	sacarElePila(Pila, Entero)	min(n, k)	0

- Cada inserción paga 2, 1 por el costo actual y 1 más para completar el costo amortizado
- Cada eliminación no paga, pues cada elemento insertado tiene un crédito de 1 que es utilizado para pagar el costo actual de la eliminación.
- El costo total amortizado es O(n) que es el costo total actual.

Análisis amortizado

- **Método Potencial:**
  - Se asignan potenciales a cada operación.
  - Se comienza con un TAD vacío y sea a_i el costo actual de la i-ésima operación y D_i el estado del TAD luego de la i-ésima operación que estaba en el estado D_{i-1}.
  - La función potencial Φ va de D_i a un número real $\Phi(D_i)$, que es el potencial asociado a D_i.
  - El costo amortizado $\hat{a}_i = a_i + \Phi(D_i) - \Phi(D_{i-1})$.
- El costo total amortizado de n operaciones es:

$$\sum_{i=1}^{n} \hat{a}_i = \sum_{i=1}^{n} (a_i + \Phi(D_i) - \Phi(D_{i-1})) = \sum_{i=1}^{n} a_i + \Phi(D_n) - \Phi(D_0)$$

Análisis amortizado

- Definiendo una función potencial que haga $\Phi(D_n) \geq \Phi(D_0)$ entonces el costo total amortizado es un límite superior para el costo total actual.
- Normalmente se escoge $\Phi(D_0) = 0$ para que los potenciales $\Phi(D_i) \geq 0 \ \forall$ i.
- Ejemplo: Se escoge el tamaño de la pila como la función potencial

i	operación	D	$\Phi(D)$	a_i	$\hat{a}_i$
0	creaPila()		0		
1	meterElePila(a)	a	1	1	2
2	meterElePila(e)	a e	2	1	2
3	meterElePila(f)	a e f	3	1	2
4	sacarElePila(2)	a	1	2	0

Análisis amortizado

- Si cada secuencia comienza con la pila vacía, su potencial será cero en ese estado.
- La complejidad total de cualquier secuencia de **m** inserciones y **q** eliminaciones es O(2**m**+0**q**) = O(**m**).
- El costo en el peor de los casos sigue siendo O(n).

"casa"
"cosa"
"capa"
"copa"

"casa"
"cosa"
"capa"

"casa"

Tablas dinámicas

- Tamaño de la tabla: normalmente desconocido
- Operaciones de inserción y eliminación aumentan y reducen el tamaño
- Problema:
  - ¿Cómo aumentar y disminuir dinámicamente el tamaño de una tabla?
  - ¿Cómo garantizar que el espacio no utilizado nunca exceda una fracción constante del espacio total
- Solución: algoritmos con un análisis amortizado de O(1)

13/4/98	**Especificación Tabla[TipoEleTab]**	
1	**Sintáctica:** creaTabla()→Tabla, insEleTabla(Tabla,TipoEleTab)→Tabla, eliEleTabla(Tabla,TipoClave) →Tabla, conEleTabla(Tabla,TipoClave) →TipoEleTab, asignarTabla(Tabla) →Tabla, vacíaTabla(Tabla) →Lógico, destTabla(Tabla) →.	-**creaTabla():** Crea una tabla vacía. **-insEleTabla():** Ingresa un nuevo elemento en la tabla que ocupa el espacio de un item. **-eliEleTabla(): E**limina el elemento de la tabla, si existe, dejando disponible el espacio ocupado. **-conEleTabla():** Regresa el elemento si está en la tabla. **-asignarTabla():** Asigna la tabla. **-vacíaTabla**(): Regresa verdadero si la tabla está vacía. **-destTabla():** Destruye la tabla.
2	**Declaraciones** **TipoEleTab: e, {TipoEleTabNoDef}** TipoClave: cl	
3	**Semántica:** eliEleTabla(creaTabla(), cl) = creaTabla() conEleTabla(creaTabla(), cl) = {TipoEleTabNoDef} conEleTabla(insEleTabla(creaTabla(), e)) = e vacíaTabla(creaTabla()) = Verdadero vacíaTabla(insEleTabla(creaTabla(), e)) = Falso asignarTabla(creaTabla()) = creaTabla()	

Implementaciones posibles

- Pila
- Montículo (heap)
- Tabla hash
- Arreglo o colección de vectores
  - Factor de carga $\alpha(T)$ de una tabla no vacía, es el número de items almacenados actualmente entre el número máximo de ranuras disponibles para los items
  - Si $\alpha(T)$ está acotado por una constante, entonces el espacio no usado será una fracción constante del espacio total

14/4/98	Tabla[TipoEleTab] Clases: Entero, Lógico, TipoEleTab, Arreglo[n]De[Lista[TipoEleTab]]	
1	**Superclase:** Ninguna	-**n**: Entero+. Número actual de elementos en la tabla.
2	**Estructura**: privado: max: Entero+ = 100 n: Entero = 0 t: Arreglo[100] De[Lista[TipoEleTab]]	-**max**: Entero+.Número máximo de ranuras en la tabla. -**t**:Arreglo[100] De [Lista[TipoEleTab]]. Arreglo de listas para implementar la tabla -**Tabla()**. *Constructor*. Crea una tabla vacía
3	**Operaciones**: público: Tabla() ~Tabla() insEleTabla(TipoEleTab: e) eliEleTabla(TipoClave: cl) conEleTabla(TipoClave: cl): Lógico =(Tabla: t) vacíaTabla(): Lógico numEle(): Entero despliegue()	-**~Tabla()** . *Destructor.* Destruye la tabla. -**insEleTabla()**. *Transformador*. Inserta un nuevo elemento en la tabla. No acepta elementos repetidos. -**eliEleTabla()**. *Transformador*. Elimina un elemento de la tabla, si existe. -**conEleTabla()**. *Observador*. Regresa verdadero si el ele-mento se encuentra en la tabla., de lo contrario regresa falso. -**=()**. *Transformador*. Asigna una tabla a otra tabla. -**vacíaTabla()**. *Observador*. Regresa verdadero si la tabla está vacía., de lo contrario regresa falso. -**numEle().** *Observador*. Regresa el número actual de elementos. -**despliegue()**. *Observador*. Despliega el contenido actual de la tabla.

Implementación como tabla hash

14/4/98	TipoEleTab Clases: Lógico, TipoClave, TipoResto	
1	**Superclases:** Ninguna	-**cl**: Elemento que sirve de clave.
2	**Estructura**: privado: cl: TipoClave resto: TipoResto	-**resto:** Resto de la información a almacenar en la tabla. -**TipoEleTab()**. *Constructor*. Crea un objeto de TipoEleTab
3	**Operaciones**: público: TipoEleTab() TipoEleTab(TipoEleTab: e) Clave(): TipoClave Clave(TipoClave: c) Resto():TipoResto Resto(TipoResto: r) =(TipoEleTab: e) =(TipoEleTab e) :Lógico lee():TipoEleTab modif(TipoEleTab: e) despliegue()	- **Clave().** *Observador y transformador*. Permite manipular el atributo clave -**Resto().** *Observador y transformador*. Permite manipular el atributo resto -**=()**. *Transformador*. Asigna a un elemento a otro elemento. -**=()**. *Observador*. Verifica si dos TipoEleTab son iguales. -**despliegue()**. *Observador*. Despliega el contenido de la tabla. -**lee().** *Observador.* Da el valor de una variable TipoEleTab. -**modif().** *Transformador.* Modifica el contenido de la clave y del resto.

Tablas dinámicas

- Valores iniciales y suposiciones:
  - Para T vacía (n = 0), $\alpha(T) = 1$
  - Implementación como un vector de ranuras
  - T está llena cuando no hayan más ranuras o $\alpha(T) = 1$
  - Para insertar un nuevo item en una T llena, se producirá un crecimiento de T, mediante la copia del vector de ranuras en otro vector de ranuras normalmente dos veces más grande
    - $\alpha(T)$ será siempre cercano a ½ si sólo se hacen inserciones $\Rightarrow$ el espacio no utilizado nunca excederá 1/2 del espacio total

Especificación de una tabla dinámica

14/9/09 TablaDina[TipoEle] Clases: Entero, Lógico, TipoEleTab, Arreglo[n]De[Lista[TipoEleTab]]		
1	**Superclase:** Ninguna	-**n**: Entero+. Número actual de elementos en la tabla.
2	**Estructura**: privado: max: Entero+ = 0 n: Entero+ = 0 t: Arreglo[max] De[TipoEle]	-**max**: Entero+.Número máximo de ranuras en la tabla. -**t**:Arreglo[max] De [TipoEle]. Arreglo de ranuras para implementar la tabla dinámica.
3	**Operaciones**: público: TablaDina() insEleTabla(TipoEle: e) eliEleTabla(TipoClave: cl) conEleTabla(TipoClave: cl): Lógico vacíaTabla(): Lógico numEle(): Entero	-**TablaDina()**. *Constructor*. Crea una tabla dinámica vacía -**insEleTabla()**. *Transformador*. Inserta un nuevo elemento en la tabla. No acepta elementos repetidos. -**eliEleTabla()**. *Transformador*. Elimina un elemento de la tabla, si existe. -**conEleTabla()**. *Observador*. Regresa verdadero si el ele-mento se encuentra en la tabla., de lo contrario regresa falso. -**vacíaTabla()**. *Observador*. Regresa verdadero si la tabla está vacía., de lo contrario regresa falso. -**numEle().** *Observador*. Regresa el número actual de elementos.

Expansión

Inserción-especificación

14/9/09	insEleTabla(TipoEle: e)	
{pre: max ≥ 0 ∧ n ≥ 0 ∧ e ≠ {TipoEleNoDef}}		{pos: c´= c + e}
1	Si (max = 0) entonces crear nueva tabla dinámica con 1 ranura max = 1 fsi	**-n:** Entero. Número de elementos actuales en la tabla dinámica. **-max:** Entero. Número de ranuras actuales en la tabla dinámica. **-e:** TipoEle. Elemento nuevo. **-t:** Arreglo de ranuras [TipoEle].
2	Si (n = max) entonces crear nueva tabla dinámica con 2*max ranuras insertar los elementos en la nueva tabla liberar las ranuras de la tabla actual t max = 2*max fsi	
3	t(n), n = e, n + 1	
1	tab1= (0,0), e=5 ⇒ (1,1,(5))	Primera inserción
2	tab1=(1, 1, (5)), e=4 ⇒ (2,2,(5, 4))	Segunda inserción
3	tab1=(2, 2, (5, 4)), e=7⇒ (4,3,(5, 4, 7,))	Tercera inserción
4	tab1=(4, 3, (5, 4, 7)), e=8⇒ (4,4,(5, 4, 7, 8))	Cuarta inserción
5	tab1=(4, 4, (5, 4, 7, 8)), e=2⇒ (8,5,(5, 4, 7, 8, 2, , , ,))	Quinta inserción

Análisis de n inserciones

- Cuál es el costo c_i de la i-ésima operación?
- Si hay espacio disponible, $c_i = 1$
- De lo contrario, $c_i = i$
- En el peor de los casos, luego de n inserciones se tiene O(n) con un límite superior no estrecho de $O(n^2)$
- La i-ésima operación causa la expansión sólo cuando i-1 es una potencia de 2

  c_i ={ i si i-1 es una potencia de 2 y 1 en otro caso}

Método agregado

$$\begin{aligned}\sum_{i=1}^{n} c_i &\le n + \sum_{j=0}^{\lfloor \lg n \rfloor} 2^j \\ &< n + 2n \\ &= 3n \ ,\end{aligned}$$

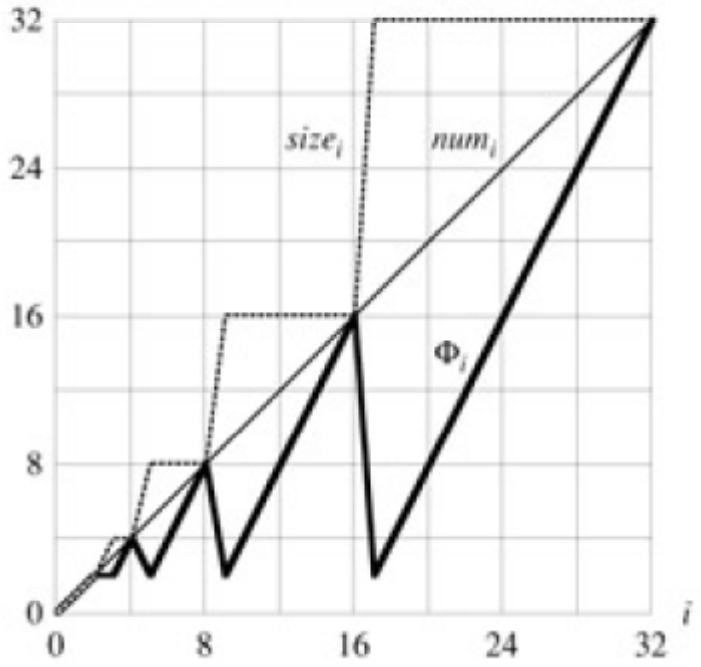
- A lo sumo n operaciones con costo 1
- El resto forma una serie geométrica
- El costo amortizado para una inserción es 3

Método contable

- Cada inserción tiene $c_i = 3$
- Intuitivamente:
  - Por su inserción 1
  - Por su copia al expandir la tabla 1
  - Por el movimiento de otro item 1
- Si max = m inmediatamente después de una expansión, entonces n = m/2
  - Cada nueva inserción paga 3
  - Luego de las m/2 – 1 inserciones, cada item tiene con que pagar su movimiento

Método del potencial

- Función potencial $\Phi(T) = 0$ inmediatamente después de una expansión
- Sea num[T]=n y size[T]=max, $\Phi(T) = 2num[T] - size[T]$
- num_i, n luego de la operación i, igual para $size_i$ y Φ_i , c_i costo amortizado



$$\begin{aligned}\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\ &= num_i + (2 \cdot num_i - size_i) - (2 \cdot num_{i-1} - size_{i-1}) \\ &= num_i + (2 \cdot num_i - 2 \cdot (num_i - 1)) - (2(num_i - 1) - (num_i - 1)) \\ &= num_i + 2 - (num_i - 1) \\ &= 3.\end{aligned}$$

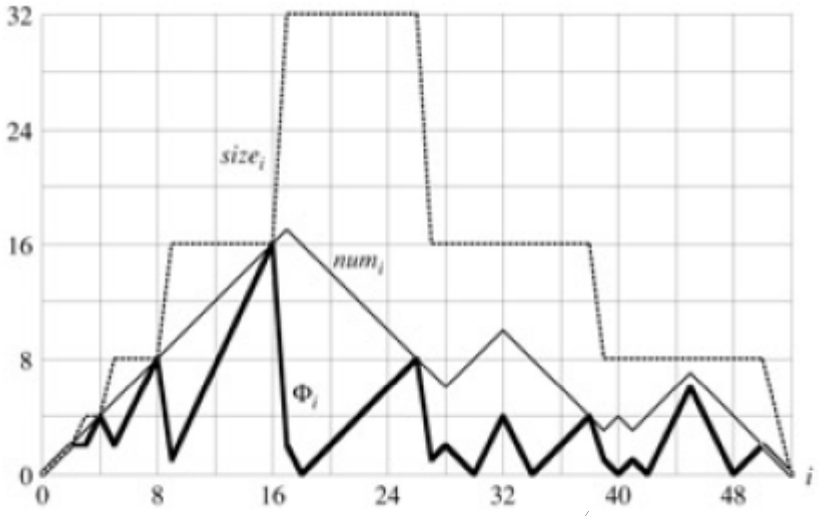
Inserciones y eliminaciones

- Método del potencial
  - Propiedades de la tabla dinámica que se desean mantener:
    - Factor de carga (α) acotado por debajo de una constante
    - Costo amortizado de cada operación acotado por arriba de una constante
  - Estrategia 1
    - Doblar max cuando n=max
    - Encoger la mitad cuando n<max/2
    - No conserva el costo amortizado acotado por arriba

Inserciones y eliminaciones

- Método del potencial
  - Estrategia 2
    - Doblar max cuando n=max
    - Encoger la mitad cuando n<α/2
    - Eliminación del último debe reiniciar n=max=0 y α=1
    - α=n/max y siempre se tiene que n= α*max
    - Para num[T]=n y size[T]=max, la función potencial Φ(T)

$$\Phi(T) = \begin{cases} 2 \cdot num[T] - size[T] & \text{if } \alpha(T) \geq 1/2 \\ size[T]/2 - num[T] & \text{if } \alpha(T) < 1/2 \end{cases}$$



Método del potencial

- Secuencia de n inserciones y eliminaciones combinadas
- Costo de la operación i es c_i
- Costo amortizado con respecto de Φ es $\hat{c}$
- Sea num_i, n luego de la operación i, igual para $size_i$, Φ_i y α_i
- Inicio: num_0, $size_0$, $\Phi_0 = 0$, $\alpha_0 = 1$
- Para inserciones, si $\alpha_{i-1} < ½$ pero $\alpha_i \geq ½$ entonces

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (2 \cdot num_i - size_i) - (size_{i-1}/2 - num_{i-1}) \\
&= 1 + (2(num_{i-1} + 1) - size_{i-1}) - (size_{i-1}/2 - num_{i-1}) \\
&= 3 \cdot num_{i-1} - 3/2;\ size_{i-1} + 3 \\
&= 3\alpha_{i-1}\ size_{i-1} - 3/2\ size_{i-1} + 3 \\
&< 3/2\ size_{i-1} - 3/2\ size_{i-1} + 3 \\
&= 3.
\end{aligned}$$

Método del potencial

- Para eliminaciones, si $\alpha_{i-1} < ½$ y no hay contracción, entonces $size_i = size_{i-1}$

$$\begin{aligned}\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\ &= 1 + (size_i/2 - num_i) - (size_{i-1}/2 - num_{i-1}) \\ &= 1 + (size_i/2 - num_i) - (size_i/2 - (num_i + 1)) \\ &= 2.\end{aligned}$$

- Si $\alpha_{i-1} < ½$ y hay contracción, entonces $size_i/2 = size_{i-1}/4 = num_{i-1} = num_i + 1$

$$\begin{aligned}\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\ &= (num_i + 1) + (size_i/2 - num_i)\ (size_{i-1}/2 - num_{i-1}) = (num_i + 1) + ((num_i + 1) - num_i) - ((2 \cdot num_i + 2) - (num_i + 1)) \\ &= 1.\end{aligned}$$

- Para la i-ésima eliminación y $\alpha_{i-1} \geq ½$, ĉ está acotado
- Las operaciones de la tabla dinámica tienen O(n)