.UBAfiuba
FACULTAD DE INGENIERÍA

# Análisis amortizado: Teoría y Ejemplo

Teoría de Algoritmos I (75.29 / 95.06)

Ing. Víctor Daniel Podberezski
vpodberezski@fi.uba.ar

# Motivación

- Cuando analizamos un algoritmo utilizamos como medida su complejidad asintótica.
- Evaluamos para cada operación su “peor caso”
- Existen situaciones donde esto nos retornará un análisis incorrecto.

# ¿Qué es?

- Técnica para analizar el tiempo de ejecución de un algoritmo.
- Determina el rendimiento promedio de cada operación en el peor de los casos
- Apropiada cuando lo que interesa entender es el comportamiento asintótico de una secuencia de operaciones
- Puede mostrar que – aún existiendo algunas operaciones costosas – en promedio el costo es pequeño

# ¿Cuándo se originó?

- **Presentado en 1985 como una técnica de análisis de algoritmos.**

  En el paper “AMORTIZED COMPUTATIONAL COMPLEXITY” por Robert Endre Tarjan”.

- **Engloba varios métodos**

  Acredita a diferentes autores por su invención

# Métodos

- **Aggregate analysis**

  Aho, A. V., J. E. Hopcroft, and J. D. Ullman. 1974. The design and analysis of computer algorithms

- **Accounting method**

  Brown, M. R., and R. E. Tarjan. 1980. “Design and analysis of a data structure for representing sorted lists.” SIAM Journal on Computing

- **Potential method**

  D. Sleator (1983?)

# Aggregate analysis

**Demuestra que, para todo n, una secuencia de n operaciones requiere un tiempo total en el peor de los casos de T(n)**

Se debe acotar a aquellas secuencias posibles de operaciones por la naturaleza racional de su uso.

**Por lo tanto, el costo amortizado por operación es T(n) / n**

A todas las operaciones – no importa si son diferentes – le asigna el mismo costo amortizado.

# Aggregate analysis: PILA

**Tenemos la estructura “PILA”, que contiene las siguientes operaciones:**

PUSH (x): Agrega a la pila el elemento x → O(1)

POP(): Extrae un elemento de la pila → O(1)

MULTIPOP(k): Extrae k elementos → O(k), O(n) si k>=n

```
Mientras haya elementos en pila y K>0
    pop()
    k--
```

# Aggregate analysis: PILA (cont.)

## Cual es el peor costo posible de n operaciones?

El peor costo de una operación es de multipop O(n)

Puedo realizar n multipop? → n* O(n) = $O(n^2)$ **NO!!**

## Para realizar n pop, primero se deben realizar n push

#pop + #multipops ≤ #push

Por lo tanto en el peor de los casos puedo hacer "n-1" push y 1 multipop de "n-1" elementos.

(n-1)*O(1) + 1*O(n-1) → O(n)

## Por lo tanto T(n) = n

Y el costo amortizado de cada operación T(n)/n → O(1)

# Accounting method

**Este método es conocido también como “el método del banquero”**

**Se asignan diferentes costos a las diferentes operaciones**

Algunas con valor mayor y otras menor al costo real ($C_i$)

**El costo de la operación asignado se conoce como “costo amortizado” ($\hat{C}_i$)**

Si el costo amortizado es mayor al real → la diferencia es un crédito

Este crédito se puede utilizar para pagar futuras operaciones con costo real mayor a su costo amortizado.

# Accounting method (cont.)

## Usaremos al costo amortizado como la cota superior del costo real

Pediremos que para toda operación “n” cumpla con la cota:

$$\sum_{i=1}^{n} \hat{C}_i \geqslant \sum_{i=1}^{n} C_i$$

Por lo tanto el crédito para cualquier “n” no debe ser negativo

Y tenemos el costo amortizado como cota:

$$T(n) = \sum_{i=1}^{n} \hat{C}_i$$

# Accounting method: PILA

**Los costos reales de las operaciones son:**

PUSH → 1

POP → 1

MULTIPOP → min (k, #S)

**Proponemos los siguientes costos amortizados:**

PUSH → 2

POP → 0

MULTIPOP → 0

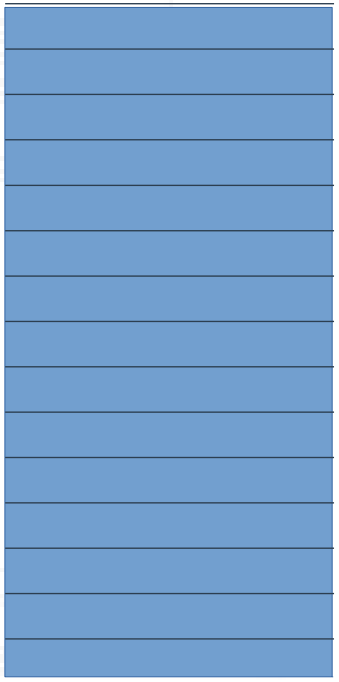
# Accounting method: PILA (cont.)

## El costo del PUSH “paga” su costo y el de un futuro POP (o multipop)

- PUSH → 1
- POP → 1
- MULTIPOP → min (k, #S)

- PUSH → 2
- POP → 0
- MULTIPOP → 0

| | $\Sigma\hat{C}_i$ | $\Sigma C_i$ |
|---|---|---|
| PUSH | 2 | 1 |
| PUSH | 4 | 2 |
| POP | 4 | 3 |
| PUSH | 6 | 4 |
| PUSH | 8 | 5 |
| MULTIPOP (3) | 8 | 8 |
| PUSH | 10 | 9 |
| PUSH | 12 | 10 |
| POP | 12 | 11 |
| | | |



PILA

# Accounting method: PILA (cont.)

**Finalmente:**

$$T(n) = \sum_{i=1}^{n} \hat{C}_i$$

**Y T(n)/n = O(1)**

# Potential method

## El trabajo prepagado se representa como “energia potencial”

Con eso se pagan operaciones futuras

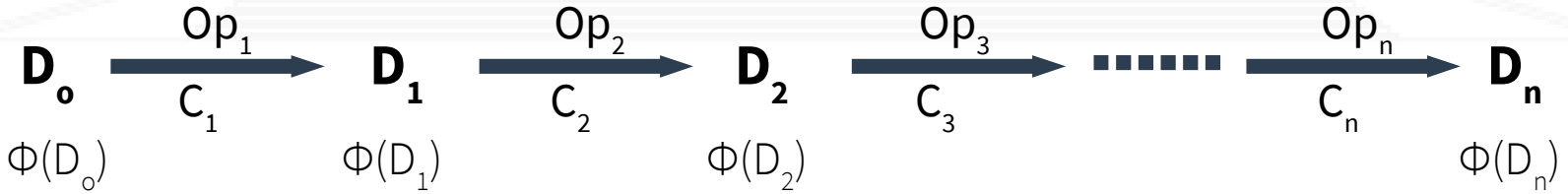
El potencial esta asociado a toda la estructura de datos (y no a objectos especificos dentro de ella)

## Llamaremos:

$C_i$ el costo real de la operación i-esima

$D_i$ la estructura de datos resultante de aplicar la operación i-esima a $D_{i-1}$

$\Phi(D_i)$ es la funcion de potencial que asigna un numero real a $D_i$



# Potential method (cont.)

**Definimos:**

El costo amortizado de la operación i-esima como:

$$\hat{C}_i = C_i + \Phi(D_i) - \Phi(D_{i-1})$$

**Podemos calcular el costo amortizado total de n operaciones:**

$$\sum_{i=1}^{n} \hat{C}_i = \sum_{i=1}^{n} \left(C_i + \Phi(D_i) - \Phi(D_{i-1})\right)$$

$$= \sum_{i=1}^{n} (C_i) + \Phi(D_n) - \Phi(D_0)$$

# Potential method (cont.)

**Si podemos definir Φ tal que $\Phi(D_n) \geq \Phi(D_o)$**

Entonces el costo amortizado $\sum_{i=1}^{n} \hat{C}_i$ nos sirve como cota superior del costo real $\sum_{i=1}^{n} C_i$

**Pedimos que $\Phi(D_i) \geq \Phi(D_o)$ para todo i.**

Entonces garantizamos el "pago en adelanto" (como en el metodo del banquero)

Usualmente se define $\Phi(D_0)=0$ y se prueba que $\Phi(D_i)\geq 0$ para toda operación i

# Potential method: PILA

**Definimos la función potencial como la cantidad de elementos en la pila.**

$\Phi(D_0)=0$ (inicialmente hay 0 elementos en la pila)

$\Phi(D_i)\geq 0$ (no pueden existir cantidad negativo de elementos)

Calculamos el costo costo de las diferentes operaciones de la pila:

PUSH:

$$C_i = 1$$

$$\Phi(D_i) - \Phi(D_{i-1}) = (s+1) - s = 1$$

$$\hat{C}_i = C_i + \Phi(D_i) - \Phi(D_{i-1}) = 1 + 1 = 2$$

# Potential method: PILA (cont.)

POP:

$$C_i = 1$$

$$\Phi(D_i) - \Phi(D_{i-1}) = s - (s+1) = -1$$

$$\hat{C}_i = C_i + \Phi(D_i) - \Phi(D_{i-1}) = 1 - 1 = 0$$

MULTIPOP:

$$C_i = k$$

$$\Phi(D_i) - \Phi(D_{i-1}) = s - (s+k) = -k$$

$$\hat{C}_i = C_i + \Phi(D_i) - \Phi(D_{i-1}) = k - k = 0$$

.UBAfiuba
FACULTAD DE INGENIERÍA

# Análisis amortizado: Contador Binario

Teoría de Algoritmos I (75.29 / 95.06)

Ing. Víctor Daniel Podberezski
vpodberezski@fi.uba.ar

# Contador binario

## Tenemos un contador binario

Utilizamos un vector A de k bits

Comienza en cero


## Para incrementar en 1:

```
i = 0
Mientras i < k y A[i]=1
    A[i] = 0;
    i++
if i < k
    A[i] = 1;
```

## En el peor caso la complejidad del contador es O(k).

Realizar n operaciones es O(nk) **NO!!**

# Contador binario - Aggregate analysis

| Contador | A[3] | A[2] | A[1] | A[0] | Costo operación | Costo acumulado |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 2 | 0 | 0 | 1 | 0 | 2 | 3 |
| 3 | 0 | 0 | 1 | 1 | 1 | 4 |
| 4 | 0 | 1 | 0 | 0 | 3 | 7 |
| 5 | 0 | 1 | 0 | 1 | 1 | 8 |
| 6 | 0 | 1 | 1 | 0 | 2 | 10 |
| 7 | 0 | 1 | 1 | 1 | 1 | 11 |
| 8 | 1 | 0 | 0 | 0 | 4 | 15 |
| 9 | 1 | 0 | 0 | 1 | 1 | 16 |
| 10 | 1 | 0 | 1 | 0 | 2 | 18 |

"n/8"

"n/4"

Cambia siempre ("n" veces)

Cambia la mitad de las veces ("n/2")

# Contador binario - Aggregate analysis (cont.)

**En general, el bit A[i] cambio $\lfloor n/2^i \rfloor$ veces**

En una secuencia de n operaciones de incremento

**La cantidad total de cambios de bits en n operaciones**

En una secuencia de n operaciones de incremento

$$\sum_{i=0}^{k-1} \left\lfloor \frac{n}{2^i} \right\rfloor < n * \sum_{i=0}^{\infty} \left(\frac{1}{2^i}\right) = 2n$$

**Realizar n operaciones es – por lo tanto – O(n)**

El costo amortizado de cada operación es O(1)

# Contador binario - Accounting method

## Los costos de la operación “incrementar” son variables

Dependen de la cantidad de bits modificados

## Proponemos:

Costo amortizado de 2 por cada bit cambiado a 1.

Inicialmente el crédito es 0

No hay operaciones con crédito negativo (en el caso extremo no hay bits a 1).

Los costos de cambiar a 0 se “pagan” con lo ahorrado en el cambio a bits a 1.

# Contador binario - Accounting method (cont.)

| Contador | A[3] | A[2] | A[1] | A[0] | Costo operación | Costo amortizado | Crédito |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 2 | 1 |
| 2 | 0 | 0 | 1 | 0 | 2 | 2 | 1 |
| 3 | 0 | 0 | 1 | 1 | 1 | 2 | 2 |
| 4 | 0 | 1 | 0 | 0 | 3 | 2 | 1 |
| 5 | 0 | 1 | 0 | 1 | 1 | 2 | 2 |
| 6 | 0 | 1 | 1 | 0 | 2 | 2 | 2 |
| 7 | 0 | 1 | 1 | 1 | 1 | 2 | 3 |
| 8 | 1 | 0 | 0 | 0 | 4 | 2 | 1 |
| 9 | 1 | 0 | 0 | 1 | 1 | 2 | 2 |
| 10 | 1 | 0 | 1 | 0 | 2 | 2 | 2 |

# Contador binario - Accounting method (cont)

Se cumple – como se requiere - que $\sum_{i=1}^{n} \hat{C}_i \geqslant \sum_{i=1}^{n} C_i$

Ademas tenemos que $T(n) = \sum_{i=1}^{n} \hat{C}_i$ es O(n)

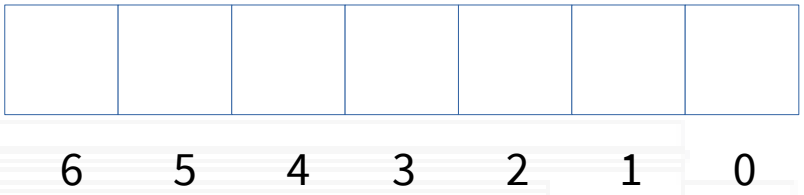
Y T(n)/n = O(1)

# Contador binario - Potential method

**Definimos:**

$\Phi(D_i) = b_i$ El numero de 1 en el contador luego de la operación i

$t_i$ = cantidad de bits reseteados a 0




**Vemos que:**

El costo real de la operación es $Ci \leq t_i + 1$
(cuando paso todos los bits a 0 es $t_i$)

Si $b_i = 0 \rightarrow$ la operación i resetea los k bits a 0.
Entonces $b_{i-1} = t_i = k$

Si $b_i > 0 \rightarrow b_i = b_{i-1} - t_i + 1$

Tenemos que: $b_i \leq b_{i-1} - t_i + 1$

| i | b | t | C | |
|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 1 |
| 2 | 1 | 1 | 2 | 10 |
| 3 | 2 | 0 | 1 | 11 |
| 4 | 1 | 2 | 3 | 100 |
| 5 | 2 | 0 | 1 | 101 |

# Contador binario - Potential method

**Definimos:**

$\Phi(D_i) = b_i$ El numero de 1 en el contador luego de la operación i

$t_i$ = cantidad de bits reseteados a 0

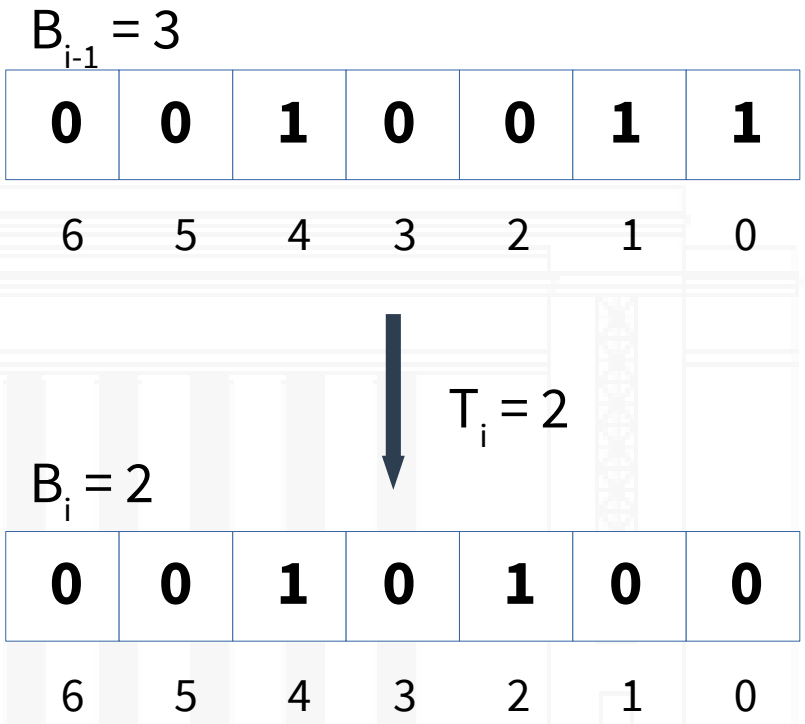
**Vemos que:**

El costo real de la operación es $Ci \leq t_i + 1$ (cuando paso todos los bits a 0 es $t_i$)

Si $b_i = 0 \rightarrow$ la operación i resetea los k bits a 0. Entonces $b_{i-1} = t_i = k$

Si $b_i > 0 \rightarrow b_i = b_{i-1} - t_i + 1$

Tenemos que: $b_i \leq b_{i-1} - t_i + 1$




$$B_i \leq B_{i-1} - t_i + 1$$

$$2 \leq 3 - 2 + 1$$

# Contador binario - Potential method (cont.)

**Finalmente vemos que:**

$$\Phi(D_i) - \Phi(D_{i-1}) \leqslant (b_{i-1} - t_i + 1) - b_{i-1} = 1 - t_i$$

$$\begin{aligned}\hat{C}_i &= C_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &\leq (t_i + 1) + (1 - t_i) \\ &= 2\end{aligned}$$

**Y tenemos que**

$\Phi(D_0)=0$ y $\Phi(D_i)\geq 0$ para todo $i>0$

Por lo tanto $\sum_{i=1}^{n} \hat{C}_i$ es O(n) y el costo de una operación amortizadas es O(1)

.UBAfiuba
FACULTAD DE INGENIERÍA

# Análisis amortizado: Expansión de tablas

Teoría de Algoritmos I (75.29 / 95.06)

Ing. Víctor Daniel Podberezski
vpodberezski@fi.uba.ar

# Tablas (Lista, Hash...)

Diversas estructuras de datos utilizan vectores para almacenar datos.

La cantidad de datos a almacenar varia (aumentando o disminuyendo)

En ocasiones se requiere expandir (o contraer) el vector.

Este redimensión implica “mover” todos los datos a un nuevo vector.

Se debe definir en base al factor de carga

- Cuando expandirse
- Cuando contraerse

# Primera aproximación (simplificada)

## Se permitirá únicamente expandir la tabla

TABLE_INSERT(x)

## Se expandirá:

Cuando el vector se llene

Al doble del tamaño actual

## Definimos factor de carga a:

#T / size(T) donde size(T) capacidad de la tabla

y #T cantidad de elementos en la tabla.

# TABLE_INSERT(x)

```
Si size(T)==0
    Crear T con size(T)=1

Si #T == size(T)
    Crear T' con size(T') = 2*size(T)
    Copiar T a T'
    Liberar T
    T = T'

insertar x en T
Incrementar #T
```

Inicialización

Expansión

Inserción

# Costo de la inserción

## El costo de la inserción i-esima es

1 si la tabla tiene lugar

i si la tabla esta llena (copiar i-1 elementos + insertar elemento i)

## La tabla se expande cuando i-1 es potencia de 2

# Expansión de tablas - Aggregate analysis

**Si insertamos n elementos tendremos:**

⌊ $\log_2 n$ ⌋ expansiones

n inserciones

**El costo de las n operaciones:**

$$\sum_{i=1}^{n} C_i \leqslant n + \sum_{i=1}^{\lfloor \log(n) \rfloor} 2^i < n + 2n = 3n$$

**El costo amortizado de cada operación:**

$3 \rightarrow O(1)$

# Expansión de tablas - Accounting method

## Los costos de la operación “insertar” son variables

Dependen de si se requiere expansión o no

## Proponemos:

Costo amortizado de 3 por cada inserción (1 de inserción + 2 “a cuenta”).

## Inicialmente el crédito es 0

## Los costos de expansión se pagan con lo ahorrado por las inserciones

1 para moverse a si mismo

1 para mover otro elemento previo a la expansión anterior.

# Expansión de tablas - Potential method

## Definimos:

$\Phi(D_i) = 2*num(T) - size(T)$

Inicialmente $\Phi(D_0) = 0$

Al estar siempre la mitad o mas de la tabla ocupada → $\Phi(D_i) \geq 0$

## Debemos analizar el costo amortizado para:

Inserción si no hay expansión

Inserción si hay expansión

# Expansión de tablas - Potential method (cont.)

**Si inserto el elemento i y no hay expansión:**

$$\hat{C}_i = C_i + \Phi(D_i) - \Phi(D_{i-1}) \qquad C_i = 1$$

$$\Phi(D_i) = 2\,num(T_i) - \text{size}(T_i) \qquad \Phi(D_{i-1}) = 2\,num(T_{i-1}) - \text{size}(T_{i-1})$$

$$\text{size}(T_{i-1}) = \text{size}(T_i) \qquad num(T_i) = num(T_{i-1}) + 1$$

$$\hat{C}_i = 1 + [2\,num(T_i) - \text{size}(T_i)] - [2\,num(T_{i-1}) - \text{size}(T_{i-1})]$$

$$\hat{C}_i = 1 + 2[num(T_{i-1}) + 1] - 2\,num(T_{i-1})$$

$$\hat{C}_i = 3$$

# Expansión de tablas - Potential method (cont.)

**Si inserto el elemento i y hay expansión:**

$$\hat{C}_i = C_i + \Phi(D_i) - \Phi(D_{i-1}) \qquad C_i = num(T_i)$$

$$\Phi(D_i) = 2\,num(T_i) - \text{size}(T_i) \qquad \Phi(D_{i-1}) = 2\,num(T_{i-1}) - \text{size}(T_{i-1})$$

$$2\,\text{size}(T_{i-1}) = \text{size}(T_i) \qquad num(T_i) = num(T_{i-1}) + 1$$

$$\text{size}(T_{i-1}) = num(T_{i-1})$$

$$\hat{C}_i = num(T_i) + [2\,num(T_i) - \text{size}(T_i)] - [2\,num(T_{i-1}) - \text{size}(T_{i-1})]$$

$$= num(T_i) + [2\,num(T_i) - 2\,\text{size}(T_{i-1})] - [2\,num(T_{i-1}) - (num(T_i - 1))]$$

$$= num(T_{i-1}) + 1 + [2(num(T_{i-1} + 1)) - 2\,\text{num}(T_{i-1})] - [2\,num(T_{i-1}) - (num(T_i - 1))]$$

$$\hat{C}_i = 3 \quad \Rightarrow \quad O(1)$$

# Agregando la eliminación

## Si permitimos la eliminación de elementos

Se requiere contraer la tabla cuando el factor de carga sea menor a un valor

Elegir incorrectamente el factor puede generar problemas (ej: ½)

## Llamaremos al método

TABLE_DELETE(x)

## Proponemos

Mantener el criterio de expansión

Contraer la tabla a la mitad cuando el factor de carga sea  menor a 1/4

# Redimensión de tablas - Potential method

## Definimos:

$$\Phi(D_i) = \begin{cases} 2\,num(T) - size(T) & \text{si factor carga } \geq 1/2 \\ size(T)/2 - num(T) & \text{si factor carga } < 1/2 \end{cases}$$

Inicialmente $\Phi(D_0) = 0$

La energía potencial siempre es mayor a cero.

## Debemos analizar el costo amortizado para:

i-esa operación es Inserción

Si factor de carga ≥ ½ y no hay expansión

Si factor de carga ≥ ½ y hay expansión

Es valido el análisis del modelo simplificado

Si factor de carga es < ½ y luego de insertar el factor sigue por debajo de ½

Si factor de carga es < ½ y luego de insertar el factor es ,mayor o igual a ½

# Redimensión de tablas - Potential method (cont.)

## Debemos analizar el costo amortizado para (cont):

I-esima operación es eliminación

- Si factor de carga < ½ y luego de eliminar el factor es mayor a ¼ (sin contracción)
- Si factor de carga < ½ y luego de eliminar el factor es menor o igual a ¼ (contracción)
- Si factor de carga es ≥ ½ y luego de eliminar el factor sigue por arriba del ½
- Si factor de carga es ≥ ½ y luego de eliminar el factor es menor a ½

## Para cada uno de estos casos

Determinar costo real, ecuación de potencial i-1 y ecuación de potencial i, equivalencias entre size y num de la tabla

Calcular el costo amortizado (llegaremos a valores de 0,1,2,3)

Por lo tanto podemos determinar que cada operación es general es O(1)

.UBAfiuba

FACULTAD DE INGENIERÍA

Presentación realizada en Abril de 2020