

Amortized Analysis

Contents

- **Introduction**
- **Aggregate analysis**
- **Accounting method**
- **Potential method**
- **Dynamic Table**

Introduction

• Amortized Analysis

- We average the time required to perform a sequence of data-structure operations over all the operations performed.
- Show that the average cost of an operation is small, even though a single operation within the sequence might be expensive.
 - average over a sequence of operations
- probability is not involved (contrary to average case analysis)
- Guarantees *the average performance of each operation* in the worst case

Introduction

- Three techniques for amortized analysis
 1. aggregate analysis
 2. accounting method
 3. potential method
- Examine these three methods using two examples
 - Stack with MULTIPOP operation
 - Incrementing binary counter

Contents

- *Introduction*
- Aggregate analysis
- Accounting method
- Potential method
- Dynamic Table

Aggregate analysis

Aggregate analysis

- A sequence of n operations takes *worst-case* time $T(n)$ in total.
- Average cost per operation is $T(n)/n$ in the worst case
= amortized cost
- Amortized cost is the same to each operation
 - Even when there are several types of operations in the sequence

Aggregate analysis

- ➊ Example of stack operation
 - Stack operations
 - $\text{PUSH}(S, x)$
 - $\text{POP}(S)$
 - $\text{MULTIPOP}(S, k)$
 - PUSH and POP run in $O(1)$ time.
 - Thus the cost of each is 1.
 - Actual running time for n operations is $\Theta(n)$.
 - Total cost of a sequence of these n operations is n .

Aggregate analysis

- Example of stack operation

- **MULTIPOP(S, k)**
 - Actual running time is linear in the number of POP operations actually executed.

```
MULTIPOP( $S, k$ )
```

```
1 while not STACK-EMPTY( $S$ ) and  $k > 0$ 
2   POP( $S$ )
3    $k = k - 1$ 
```

$T(1) * k$

- So, cost of $\text{MULTIPOP}(S, k)$ is $O(k)$.

Aggregate analysis

Example of stack operation

- $\text{MULTIPOP}(S, k)$
 - Remove the k top objects of stack

- If objects of stack are less than k , it removes the objects in the stack.
 - So, stack is empty

index	1	2	3	4	5	6
S						

$\text{MULTIPOP}(S, 2)$

↑
top

index	1	2	3	4	5	6
S						

$\text{MULTIPOP}(S, 6)$

↑
top

index	1	2	3	4	5	6
S						

Aggregate analysis

Example of stack operation

- Analysis of a sequence of n PUSH, POP and MULTIPOP operations
 - on an initially empty stack
- Intuitive analysis of time complexity (wrong way)
 - The worst-case cost of one MULTIPOP: $O(n)$
 - Stack size: at most n

→ Total cost : $O(n^2)$

- This cost isn't tight

Aggregate analysis

Example of stack operation

- Using Aggregate analysis
 - Can obtain a better upper bound from the entire sequence of n operations
- Any sequence of n PUSH, POP and MULTIPOP operations
 - on an initially empty stack
 - [Push, push, pop, push, push, push, multipop(2), ...]
= [Push, push, pop, push, push, push, {pop, pop}, ...]
$$n \geq \#(\text{push}) \geq \#(\text{pop})$$
$$2n \geq \#(\text{push}) + \#(\text{pop})$$

→ Total cost : $O(n)$
- Amortized cost is $O(n) / n = O(1)$

Aggregate analysis

- Example of incrementing a binary counter
 - Consider the problem of implementing a k -bit binary counter that counts upward from 0
 - Use an array $A[0..k-1]$ of bits

$A[k-1]$...	$A[2]$	$A[1]$	$A[0]$
----------	-----	--------	--------	--------

Aggregate analysis

- Example of incrementing a binary counter
 - Cost of INCREMENT operation is proportional to the number of bits flipped.

INCREMENT (A)

```
1   i = 0
2   while i < A.length and A[i] == 1
3       A[i] = 0
4       i = i + 1
5   if i < A.length
6       A[i] = 1
```

Bit flip, $T(1)$

Bit flip, $T(1)$

Aggregate analysis

- Example of incrementing a binary counter
 - Cost of INCREMENT operation is proportional to the number of bits flipped.

Ex. $\text{INCREMENT}(A)$

Counter value	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	cost	Total cost
0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1

Aggregate analysis

- Example of incrementing a binary counter
 - Cost of INCREMENT operation is proportional to the number of bits flipped.

Ex. $\text{INCREMENT}(A)$

Counter value	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	cost	Total cost
0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1
2	0	0	0	1	0	2	3

Aggregate analysis

- Example of incrementing a binary counter
 - Cost of INCREMENT operation is proportional to the number of bits flipped.

Ex. $\text{INCREMENT}(A)$

Counter value	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	cost	Total cost
0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1
2	0	0	0	1	0	2	3
3	0	0	0	1	1	1	4

Aggregate analysis

- Example of incrementing a binary counter
 - Cost of INCREMENT operation is proportional to the number of bits flipped.

Ex. $\text{INCREMENT}(A)$

Counter value	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	cost	Total cost
0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1
2	0	0	0	1	0	2	3
3	0	0	0	1	1	1	4
4	0	0	1	0	0	3	7

Aggregate analysis

- Example of incrementing a binary counter
 - A single execution of INCREMENT takes time $\Theta(k)$ in the worst case
 - In which array A contains all 1's.

$A[k-1]$...	$A[2]$	$A[1]$	$A[0]$	cost
1	...	1	1	1	-
0	...	0	0	0	k

- Thus, a sequence of n INCREMENT operations on an initially zero counter takes time $O(nk)$ in the worst case.

Aggregate analysis

- Example of incrementing a binary counter
 - Aggregate Analysis
 - can tighten our analysis to yield a worst-case cost of $O(n)$ for a sequence of n INCREMENT operations by observing that not all bits flip each time INCREMENT is called

Aggregate analysis

- Example of incrementing a binary counter
 - Compute bit flip of Array A
 - flip of $A[0]$: n
 - flip of $A[1]$: $\left\lfloor \frac{n}{2} \right\rfloor$
 - flip of $A[2]$: $\left\lfloor \frac{n}{4} \right\rfloor$
 - The total number of flip in the sequence
 - $\sum_{i=0}^{k-1} \left\lfloor \frac{n}{2^i} \right\rfloor < \sum_{i=0}^{\infty} n/2^i = 2n$
 - Total cost $O(n)$
 - Amortized cost = $O(n)/n = O(1)$

	$A[3]$	$A[2]$	$A[1]$	$A[0]$
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0

\uparrow \uparrow \uparrow \uparrow \uparrow
 $n/8$ $n/4$ $n/2$ n

Aggregate analysis

- Running time
 - $O(n)$ time in total
- Amortized cost
 - $O(n) / n = O(1)$

Contents

- *Introduction*
- *Aggregate analysis*
- **Accounting method**
- **Potential method**
- **Dynamic Table**

Accounting method

• Accounting method

- Amortized cost of an operation: assign differing charges to different operations, with some operations charged more or less than they actually cost
- Credit
 - the difference between amortized cost and actual cost
 - can help pay for later operations whose amortized cost is less than their actual cost
- Different operations may have different amortized costs, unlike aggregate analysis.

Accounting method

• Accounting method

- choose amortized cost of operations, so that the total amortized cost provides an upper bound on the total actual cost.
 - c_i : actual cost of the i th operation
 - \hat{c}_i : amortized cost of the i th operation
 - $\sum_{i=0}^n \hat{c}_i \geq \sum_{i=0}^n c_i$, for all sequences of n operations
- The total credit stored in the data structure is $\sum_{i=0}^n \hat{c}_i - \sum_{i=0}^n c_i$, and must be non-negative at all times.

Accounting method

- Example of stack operation
 - The actual costs of the operations
 - PUSH 1
 - POP 1
 - MULTIPOP $\min(k, s)$
 - The amortized costs of the operations
 - PUSH 2
 - POP 0
 - MULTIPOP 0

Accounting method

- Example of stack operation

index	1	2	3	4
S				

S	1	2	3	4
credit				

Accounting method

- Example of stack operation

- PUSH

index	1	2	3	4
<i>S</i>				

↑
top

<i>S</i>	1	2	3	4
credit	1			

- PUSH : actual cost 1 + prepaid credit 1
- Amortized cost : actual cost + credit = 2

Accounting method

Example of stack operation

- PUSH

index	1	2	3	4
<i>S</i>				

↑
top

<i>S</i>	1	2	3	4
credit	1	1		

- PUSH : actual cost 1 + prepaid credit 1
- Amortized cost : actual cost + credit = 2

Accounting method

- Example of stack operation

- PUSH

index	1	2	3	4
S				

↑
top

S	1	2	3	4
credit	1	1	1	

- PUSH : actual cost 1 + prepaid credit 1
- Amortized cost : actual cost + credit = 2

Accounting method

- Example of stack operation

- POP

index	1	2	3	4
S				

↑
top

S	1	2	3	4
credit	1	1	0	

- POP and MULTIPOP : pay credit 1
- Amortized cost : actual cost - credit = 0

Accounting method

- Example of stack operation

- PUSH

index	1	2	3	4
S				

↑
top

S	1	2	3	4
credit	1	1	1	

- PUSH : actual cost 1 + prepaid of credit 1
- Amortized cost : actual cost + credit = 2

Accounting method

- Example of stack operation
 - POP and MULTIPOP must execute after PUSH operation
 - Charging the PUSH operation a little bit more (= credit)
So, credit pay actual cost of POP and MULTIPOP operation
 - The amount of credit is always nonnegative
 - Because the stack always has nonnegative objects.
 - Thus, the total amortized cost is an upper bound on the total actual cost
 - Total amortized cost : $O(n)$
 - Total actual cost : $O(n)$

Accounting method

- Example of incrementing a binary counter

- The actual costs
 - Bit set ($0 \rightarrow 1$) : 1
 - Bit reset ($1 \rightarrow 0$) : 1
 - The amortized costs
 - Bit set : 2
 - Bit reset : 0

Accounting method

- Example of incrementing binary counter

$A[3]$	$A[2]$	$A[1]$	$A[0]$
0	0	0	1

A	$A[0]$	$A[1]$	$A[2]$	$A[3]$	$A[4]$
credit	1				

Accounting method

- Example of incrementing binary counter

$A[3]$	$A[2]$	$A[1]$	$A[0]$
0	0	0	1
			0

A	$A[0]$	$A[1]$	$A[2]$	$A[3]$	$A[4]$
credit	0				

Accounting method

- Example of incrementing binary counter

$A[3]$	$A[2]$	$A[1]$	$A[0]$
0	0	0	1
0	0	1	0

A	$A[0]$	$A[1]$	$A[2]$	$A[3]$	$A[4]$
credit	0	1			

Accounting method

- Example of incrementing binary counter

$A[3]$	$A[2]$	$A[1]$	$A[0]$
0	0	0	1
0	0	1	0
0	0	1	1

A	$A[0]$	$A[1]$	$A[2]$	$A[3]$	$A[4]$
credit	1	1			

Accounting method

- Example of incrementing binary counter

$A[3]$	$A[2]$	$A[1]$	$A[0]$
0	0	0	1
0	0	1	0
0	0	1	1
			0

A	$A[0]$	$A[1]$	$A[2]$	$A[3]$	$A[4]$
credit	0	1			

Accounting method

- Example of incrementing binary counter

$A[3]$	$A[2]$	$A[1]$	$A[0]$
0	0	0	1
0	0	1	0
0	0	1	1
		0	0

A	$A[0]$	$A[1]$	$A[2]$	$A[3]$	$A[4]$
credit	0	0			

Accounting method

- Example of incrementing binary counter

$A[3]$	$A[2]$	$A[1]$	$A[0]$
0	0	0	1
0	0	1	0
0	0	1	1
	1	0	0

A	$A[0]$	$A[1]$	$A[2]$	$A[3]$	$A[4]$
credit	0	0	1		

Accounting method

- Example of incrementing binary counter

$A[3]$	$A[2]$	$A[1]$	$A[0]$
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1

A	$A[0]$	$A[1]$	$A[2]$	$A[3]$	$A[4]$
credit	1	0	1		

Accounting method

- Example of incrementing a binary counter
 - Bit reset must be executed after bit set
 - Charging the bit set in credit

So, the credit pays for actual cost of reset operation
 - The amount of credit is always nonnegative

Because the number of 1's in the counter never becomes negative
Thus, the total amortized cost is an upper bound on the total actual cost.
 - the total amortized cost : $O(n)$
 - the total actual cost : $O(n)$

Accounting method

- ➊ Amortized cost
 - $O(n)$ time in total
- ➋ Running time
 - $O(n)$ time in total

Contents

- *Introduction*
- *Aggregate analysis*
- *Accounting method*
- **Potential method**
- **Dynamic Table**

Potential method

• Potential method

- Similar to an accounting method
Credit → “potential energy” or just “potential”
- The potential with the data structure as a whole rather than with specific objects within the data structure.

Potential method

• Potential method

- With n operations,

D_0 : an initial data structure

D_i : the data structure that results after applying the i th operation to data structure D_{i-1}

$\Phi(D_i)$: the potential associated with data structure D_i

- Potential difference ($\Phi(D_i) - \Phi(D_{i-1})$)

- positive

The potential of the data structure increases

- negative

The decrease in the potential pays for the actual cost of the operation

Potential method

• Potential method

- Amortized cost
 - $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$
- The total amortized cost of the n operations
 - $$\begin{aligned} \sum_{i=0}^n \hat{c}_i &= \sum_{i=0}^n (c_i + \Phi(D_i) - \Phi(D_{i-1})) \\ &= \sum_{i=0}^n c_i + \Phi(D_n) - \Phi(D_0) \end{aligned}$$
- We require $\Phi(D_i) \geq \Phi(D_0)$ for all i
 - so that $\sum_{i=0}^k \hat{c}_i \geq \sum_{i=0}^k c_i$ for all $1 \leq k \leq n$

Potential method

Example of stack operation

- Potential function Φ
 - the number of objects in the stack
 - $\Phi(D_0) = 0$
- The stack D_i that results after the i th operation has nonnegative potential
 - $\Phi(D_i) \geq 0 = \Phi(D_0)$

Potential method

Example of stack operation

- Amortized cost analysis of each operation
 - PUSH operation
 - If the i th operation on a stack containing s objects is a PUSH operation,
 - $\Phi(D_i) - \Phi(D_{i-1}) = (s + 1) - s = 1$
 - So, the amortized cost is $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$
$$= 1 + (s + 1) - s$$
$$= 2$$
 - POP operation
 - If the i th operation on a stack containing s objects is a POP operation,
 - $\Phi(D_i) - \Phi(D_{i-1}) = (s - 1) - s = -1$
 - So, the amortized cost is $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$
$$= 1 + (s - 1) - s$$
$$= 0$$

Potential method

Example of stack operation

- Amortized cost analysis of each operation
 - MULTIPOP(S, k) operation
 - If the i th operation on a stack containing s objects is a MULTIPOP operation,
 - $k' = \min(k, s)$: The number of objects to be popped off the stack
 - $\Phi(D_i) - \Phi(D_{i-1}) = -\min(k, s) = -k'$
The amortized cost is $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$
 $= k' - k'$
 $= 0$

Potential method

- Example of stack operation
 - Amortized cost : $O(1)$
 - Total amortized cost : $O(n)$
 - Total actual cost : $O(n)$

3. Potential method

- Example of incrementing binary counter
 - Potential function Φ
 - The number of 1's in the array
 - b_i : The number of 1's in the counter after the i th INCREMENT operation
 - t_i : The number of bits reset in the i th INCREMENT operation
 - Actual cost of the operation
 - c_i is at most $t_i + 1$

```
INCREMENT ( $A$ )
1    $i = 0$ 
2   while  $i < A.length$  and  $A[i] == 1$ 
3        $A[i] = 0$ 
4        $i = i + 1$ 
5   if  $i < A.length$ 
6        $A[i] = 1$ 
```

3. Potential method

- Example of incrementing a binary counter

- Case of $b_i = 0$
 - the i th operation resets all k bits
 - $b_{i-1} = t_i = k$
- Case of $b_i > 0$
 - $b_i = b_{i-1} - t_i + 1$
- In either case
 - $b_i \leq b_{i-1} - t_i + 1$

Ex) 1111 -> 0000

Counter value	$A[k]$...	$A[2]$	$A[1]$	$A[0]$	b_i
$i-1$	1	...	1	1	1	k
i	0	...	0	0	0	0

Ex) 0111 -> 1000

Counter value	$A[k]$...	$A[2]$	$A[1]$	$A[0]$	b_i
$i-1$	0	...	1	1	1	$k-1$
i	1	...	0	0	0	1

3. Potential method

- Example of incrementing a binary counter

- Potential difference

- $$\begin{aligned}\Phi(D_i) - \Phi(D_{i-1}) &= b_i - b_{i-1} \\ &\leq (b_{i-1} - t_i + 1) - b_{i-1} \\ &= 1 - t_i\end{aligned}$$

- Amortized cost

- $$\begin{aligned}\widehat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &\leq (t_i + 1) + (1 - t_i) = 2\end{aligned}$$

 $\rightarrow O(1)$

3. Potential method

- Example of incrementing a binary counter
 - If the counter starts at zero, $\Phi(D_0) = 0$ and since $\Phi(D_i) \geq 0$ for all i
 - The total amortized cost of a sequence of n INCREMENT operations is an upper bound on the total actual cost
 - The worst-case cost of n INCREMENT operations is $O(n)$

3. Potential method

- Example of incrementing a binary counter
 - If it does not start at zero
 - $b_0 \geq 0, b_n \leq k$ (k : the number of bits in the counter)
 - $\sum_{i=0}^n \hat{c}_i = \sum_{i=0}^n c_i + \Phi(D_n) - \Phi(D_0)$
 - $\sum_{i=0}^n c_i = \sum_{i=0}^n \hat{c}_i - \Phi(D_n) + \Phi(D_0)$ ($\hat{c}_i \leq 2$ for all $1 \leq i \leq n$)
 $\leq \sum_{i=0}^n 2 - b_n + b_0$ ($\Phi(D_n) = b_n, \Phi(D_0) = b_0$)
 $= 2n - b_n + b_0$
 - The total actual cost is $O(n)$ ($b_0 \leq k, k = O(n)$)

Contents

- *Introduction*
- *Aggregate analysis*
- *Accounting method*
- *Potential method*
- **Dynamic Table**

Dynamic tables

- We do not always know in advance how many objects some applications will store in a table
 - insertion
 - So allocate space for a table and reallocate the table when new item is added.
 - deletion
 - Similarly, if many objects have been deleted from the table, it may be worthwhile to reallocate the table with a smaller size
 - Using amortized analysis, we shall show that the amortized cost of insertion and deletion is only $O(1)$

Aggregate analysis

• INSERT

- When inserting an item into a full table, we can expand the table by allocating a new table with more slots than the old table had.
- A common heuristic allocates a new table with **twice** as many slots as the old one.

Aggregate analysis

• INSERT

- $T.table$: a pointer to the block of storage representing the table.
- $T.num$: the number of items in the table
- $T.size$: the total number of slots in the table.

Aggregate analysis

```
TABLE-INSERT( $T, x$ )
```

```
1   if  $T.size == 0$ 
2       allocate  $T.table$  with 1 slot
3        $T.size = 1$ 
4   if  $T.num == T.size$ 
5       allocate new-table with  $2 * T.size$  slots
6       insert all items in  $T.table$  into new-table
7       free  $T.table$ 
8        $T.table = new-table$ 
9        $T.size = 2 * T.size$ 
10      insert  $x$  into  $T.table$ 
11       $T.num = T.num + 1$ 
```

elementary insertion

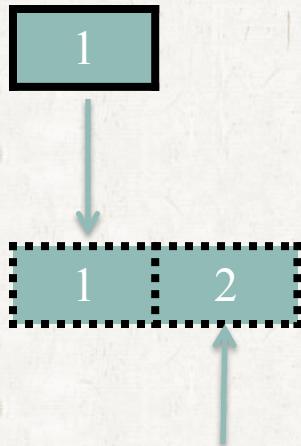
expansion

Aggregate analysis

- Let us analyze a sequence of n TABLE-INSERT operations on an initially empty table.
 - If the current table has room for the new item, then cost $c_i = 1$.
 - If the current table is full, an expansion occurs, then $c_i = i$.
 - 1 for insert new item, $i-1$ for copies from the old table to the new.

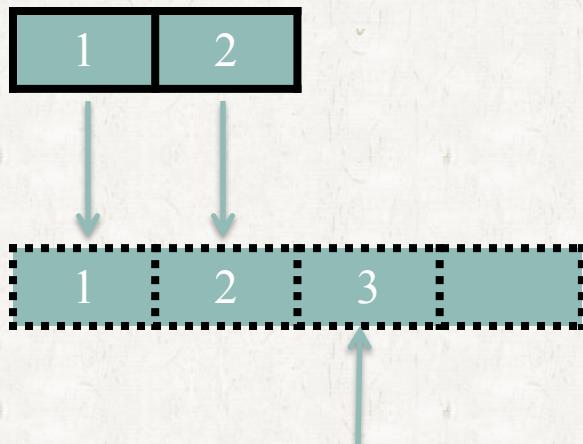
Aggregate analysis

- Let us analyze a sequence of n TABLE-INSERT operations on an initially empty table.
 - If the current table has room for the new item, then cost $c_i = 1$.
 - If the current table is full, an expansion occurs, then $c_i = i$.
 - 1 for insert new item, $i-1$ for copies from the old table to the new.



Aggregate analysis

- Let us analyze a sequence of n TABLE-INSERT operations on an initially empty table.
 - If the current table has room for the new item, then cost $c_i = 1$.
 - If the current table is full, an expansion occurs, then $c_i = i$.
 - 1 for insert new item, $i-1$ for copies from the old table to the new.



Aggregate analysis

- Let us analyze a sequence of n TABLE-INSERT operations on an initially empty table.
 - If the current table has room for the new item, then cost $c_i = 1$.
 - If the current table is full, an expansion occurs, then $c_i = i$.
 - 1 for insert new item, $i-1$ for copies from the old table to the new.

1	2	3	4
---	---	---	---



Aggregate analysis

- Let us analyze a sequence of n TABLE-INSERT operations on an initially empty table.

$$c_i = \begin{cases} i & \text{if } i-1 \text{ is an exact power of 2} \\ 1 & \text{otherwise} \end{cases}$$

Aggregate analysis

- The total cost of n TABLE-INSERT operations Is therefore

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

- Since the total cost of n TABLE-INSERT operations is bounded by $3n$, the amortized cost of single operation is at most 3 ($3n / n$)

Accounting method

- By using the accounting method, we can gain some feeling for why the amortized cost of a TABLE-INSERT operation should be 3.

Accounting method

TABLE-INSERT(T, x)

```
1      if  $T.size == 0$ 
2          allocate  $T.table$  with 1 slot
3           $T.size = 1$ 
4      if  $T.num == T.size$ 
5          allocate new-table with  $2 * T.size$  slots
6          insert all items in  $T.table$  into new-table
7          free  $T.table$ 
8           $T.table = new-table$                                 elementary insertion
9           $T.size = 2 * T.size$ 
10         insert  $x$  into  $T.table$ 
11          $T.num = T.num + 1$ 
```

Accounting method

- By using the accounting method, we can gain some feeling for why the amortized cost of a TABLE-INSERT operation should be 3.
 - There are two types of elementary insertion:
 - 6 insert all items in $T.table$ into new-table
 - 10 insert x into $T.table$

Accounting method

- By using the accounting method, we can gain some feeling for why the amortized cost of a TABLE-INSERT operation should be 3.
 - each item pays for 3 elementary insertions:
 - 1 **cost** for line 10,
 - 2 **credit** for line 6.
 - Credit is used to move items when expansion occurs.

Accounting method

- By using the accounting method, we can gain some feeling for why the amortized cost of a TABLE-INSERT operation should be 3.
 - each item pays for 3 elementary insertions:
 - inserting itself into the current table
 - moving itself when the table expands
 - moving another item that has already been moved once when the table expands

1	2		
0	0		

Credit for move

Accounting method

- By using the accounting method, we can gain some feeling for why the amortized cost of a TABLE-INSERT operation should be 3.
 - each item pays for 3 elementary insertions:
 - inserting itself into the current table
 - moving itself when the table expands
 - moving another item that has already been moved once when the table expands

1	2	3	
0	0	0	

Credit for move

Accounting method

- By using the accounting method, we can gain some feeling for why the amortized cost of a TABLE-INSERT operation should be 3.
 - each item pays for 3 elementary insertions:
 - inserting itself into the current table
 - moving itself when the table expands
 - moving another item that has already been moved once when the table expands

1	2	3	
0	0	1	

Credit for move

Accounting method

- By using the accounting method, we can gain some feeling for why the amortized cost of a TABLE-INSERT operation should be 3.
 - each item pays for 3 elementary insertions:
 - inserting itself into the current table
 - moving itself when the table expands
 - moving another item that has already been moved once when the table expands

1	2	3	
1	0	1	

Credit for move

Accounting method

- By using the accounting method, we can gain some feeling for why the amortized cost of a TABLE-INSERT operation should be 3.
 - each item pays for 3 elementary insertions:
 - inserting itself into the current table
 - moving itself when the table expands
 - moving another item that has already been moved once when the table expands

1	2	3	4
1	1	1	1

Credit for move

Accounting method

- By using the accounting method, we can gain some feeling for why the amortized cost of a TABLE-INSERT operation should be 3.
 - each item pays for 3 elementary insertions:
 - inserting itself into the current table
 - moving itself when the table expands
 - moving another item that has already been moved once when the table expands

1	2	3	4
1	1	1	1

Credit for move



1							
0							

Accounting method

- By using the accounting method, we can gain some feeling for why the amortized cost of a TABLE-INSERT operation should be 3.
 - each item pays for 3 elementary insertions:
 - inserting itself into the current table
 - moving itself when the table expands
 - moving another item that has already been moved once when the table expands

1	2	3	4
1	1	1	1

Credit for move

1	2		
0	0		

Accounting method

- By using the accounting method, we can gain some feeling for why the amortized cost of a TABLE-INSERT operation should be 3.
 - each item pays for 3 elementary insertions:
 - inserting itself into the current table
 - moving itself when the table expands
 - moving another item that has already been moved once when the table expands

1	2	3	4				
1	1	1	1				
				Credit for move			
1	2	3	4				
0	0	0	0				

Potential method

- We can use the potential method to analyze a sequence of n TABLE-INSERT operations.
 - and we can use it in Section 17.4.2 to design a TABLE-DELETE operation that has an $O(1)$ amortized cost as well

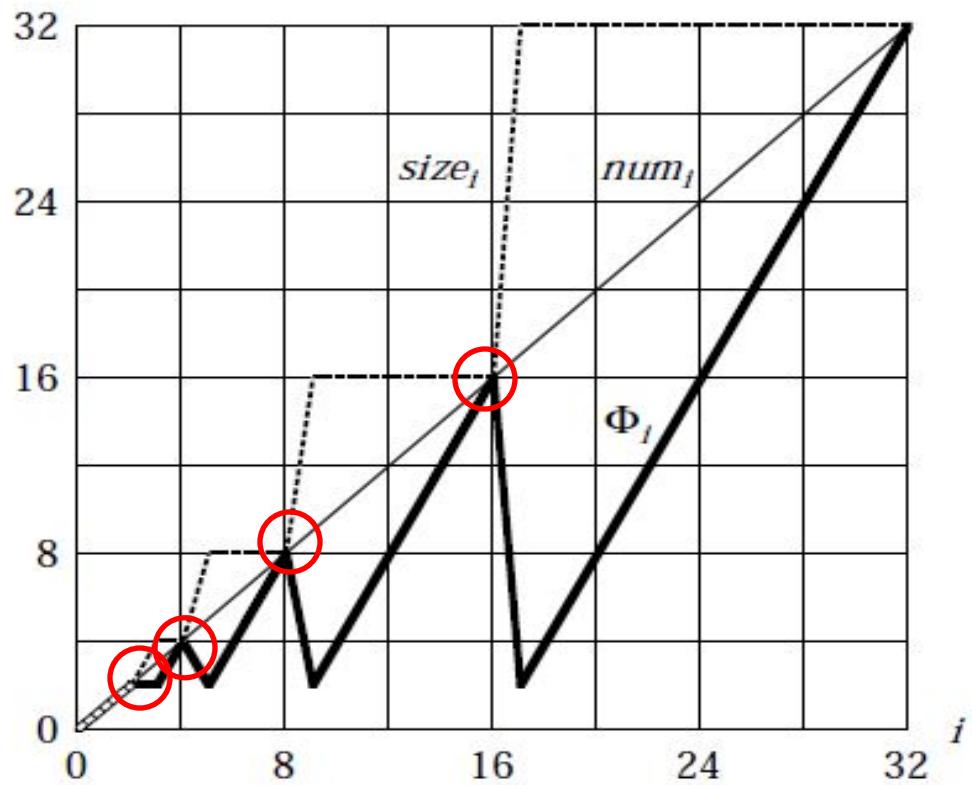
Potential method

- We can use the potential method to analyze a sequence of n TABLE-INSERT operations.
 - and we can use it in Section 17.4.2 to design a TABLE-DELETE operation that has an $O(1)$ amortized cost as well
- We start by defining a potential function Φ
 - 0 immediately after an expansion
 - table size by the time the table is full

Potential method

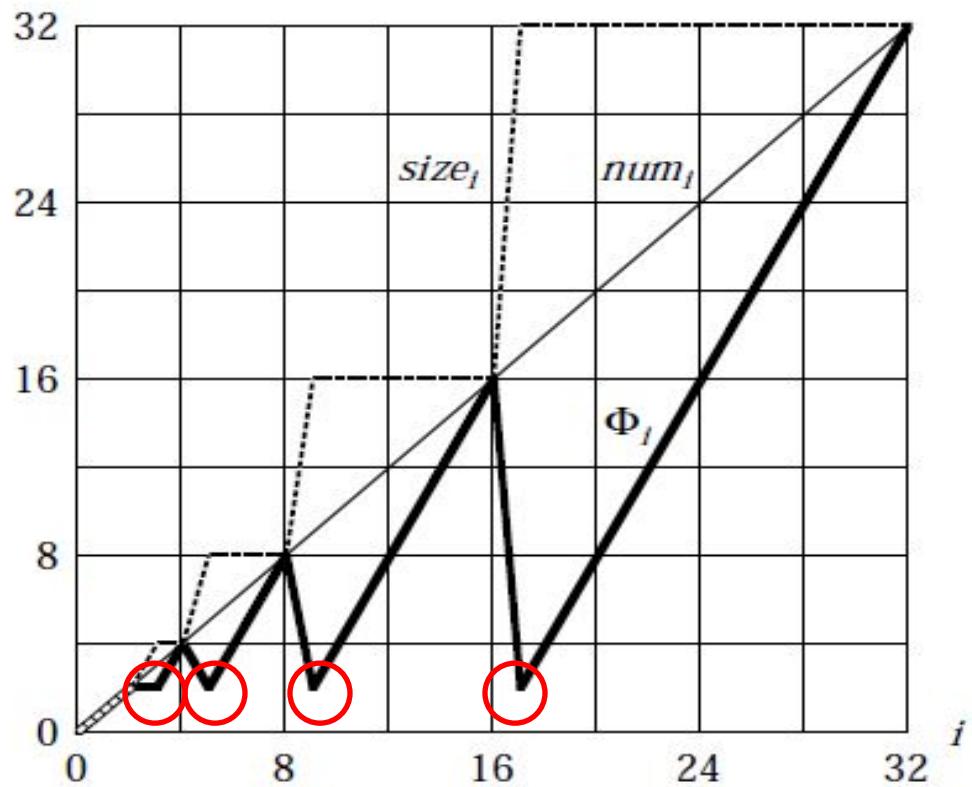
- $\Phi(T) = 2*T.\text{num} - T.\text{size}$ (17.5)
- Immediately before an expansion, we have $T.\text{num} = T.\text{size}$ and thus $\Phi(T) = T.\text{num}$
- $\Phi(T)$ is always nonnegative
 - The initial value of the potential is 0
 - and since the table is always at least half full, $T.\text{num} \geq T.\text{size}/2$

Potential method



- Before expansion, $\Phi_i = num_i$

Potential method



- After expansion, $\Phi_i = 0$ but immediately increased by 2

Potential method

- The amortized cost of the i th TABLE-INSERT operation
 - num_i : the number of items stored in the table after the i th operation
 - $size_i$: the total size of the table after the i th operation
 - Φ_i : the potential after the i th operation
 - \hat{c}_i : its amortized cost with respect to Φ
- Initially, we have $num_0 = 0$, $size_0 = 0$, and $\Phi_0 = 0$.

Potential method

- The amortized cost of the i th TABLE-INSERT operation
 - If the i th TABLE-INSERT operation does not trigger an expansion, then we have $size_i = size_{i-1}$ and the amortized cost of the operation is
 - $\Phi(T) = 2 * T.num - T.size$

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

Potential method

- The amortized cost of the i th TABLE-INSERT operation
 - If the i th operation does trigger an expansion, then we have
$$\begin{aligned}size_i &= 2 * size_{i-1} \\size_{i-1} &= num_{i-1} = num_i - 1 \\size_i &= 2 * (num_i - 1).\end{aligned}$$
Thus, the amortized cost of the operation is

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