Amortized Analysis

- **Amortized analysis:**
  - Guarantees the avg. performance of each operation in the worst case.
  - Aggregate method
  - Accounting method
  - Potential method

- **Aggregate method**
  - A sequence of n operations takes worst-case time $T(n)$ in total. In the worst case, the amortized cost (average cost) per operation is $T(n)/n$

Eg.1 [Stack operation]

PUSH(S,x): pushes object x onto stack S

POP(S): pops the top of stack S and return the popped object

- Each runs in O(1) time

故 n 個 PUSH 和 POP operations 需 $\Theta(n)$ time

- MULTIPOP(S, k): pops the top k objects of stack S

MULTIPOP(S, k):

  while not STACK-EMPTY(S) and $k \neq 0$

    do { POP(S)

      $k \leftarrow k-1$ }

故執行一 MULTIPOP 需 min{s, k} 個步驟

問題：執行上述 PUSH, POP, MULTIPOP n次

假設 initial stack 為 empty

試問每一 operation 在 worst-case 之 amortized cost?

- Each object can be popped at most once for each time it is pushed.
- 上述問題只需 O(n) time

  O(n)/n = O(1) :每個 operation 平均所需的時間

- Eg.2 [Incrementing a binary counter]

A k-bit binary counter, counts upward from 0

A[k-1] A[k-2]…A[1] A[0]

highest order bit lowest order bit

```
Increment(A, k)
    i = 0
    while i< k and A[i]==1:
        A[i] = 0
        i = i+1
    if i< k:
        A[i] = 1
```

Ripple-carry counter

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

- 問題：執行 n 次(依序) Increment operations.

在 worst-case 每一operation 之 amortized cost 為何?

- A[0] 每 1 次改變一次

  A[1] 每 2 次改變一次

  A[i] 每 2^i 次改變一次

$$\sum_{i=0}^{\lfloor \lg n \rfloor} \left\lfloor \frac{n}{2^i} \right\rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$

Amortized cost of each operation:

O(n)/n = O(1)

Potential method

- D_0 : initial data structure
  c_i : actual cost of the i-th operation
  D_i : data structure after applying the i-th operation to D_{i-1}
  Φ : potential function, $\Phi(D_i)$ is a real number
  $\hat{c}_i$: the amortized cost of the i-th operation
  w.r.t potential function is $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1}))$$

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

If $\Phi(D_n) \geq \Phi(D_0)$, the total amortized cost is the upper bound of total actual cost.

- Eg. 1 [Stack operation]

  定義 potential function Φ: **stack size**

  $\Phi(D_0)=0$, 亦知 $\Phi(D_i) \geq 0$

  i-th op.

  PUSH:

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

  MULTIPOP(S, k): 令 k'=min{k,s}

  $$\Phi(D_i)-\Phi(D_{i-1})=-k'$$
  $$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$
  $$= k' - k' = 0$$

  POP: 同理 $\hat{c}_i = 0$

  所以一序列 n 個 op 之 total amortized cost 為 O(n)

- Eg. 2 [Incrementing a binary counter]

  對 counter 而言其 potential function 值為在 i-th op 後被設定為 1 的 bit 個數

  b_i = # of 1's in the counter after the i-th operation
- Suppose that the i-th Increment op. resets t_i bits to 0.

  actual cost $\leq t_i+1$
- # of 1's in the counter after the i-th op.:

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

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

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$
$$\leq (t_i + 1) + (1 - t_i) = 2$$

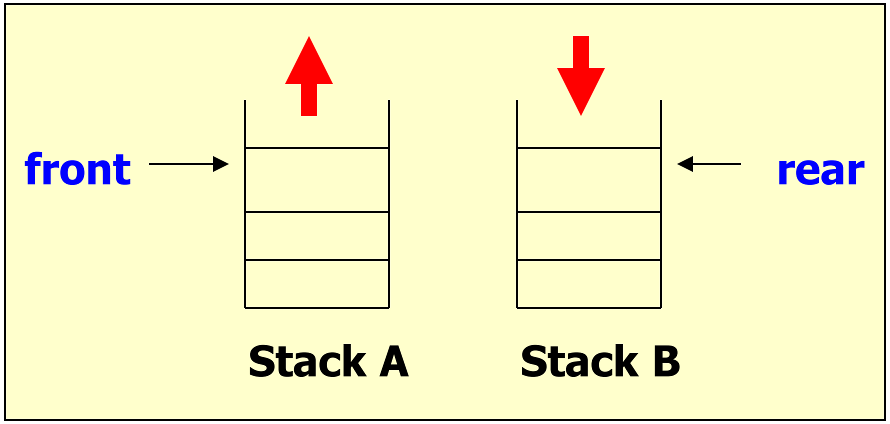
$$\sum_{i=1}^{n} c_i = \sum_{i=1}^{n} \hat{c}_i - \Phi(D_n) + \Phi(D_0)$$
$$= 2n - b_n + b_0$$

- Implement a Queue with **two** Stacks:

  Stack A : Dequeue

  Stack B : Enqueue

  S_B : stack size of B



- Enqueue(x)

  { PUSH(B, x) }

  Dequeue(Q)

  { if Empty(Q) then return "empty queue";

  else if Empty(A) then

  while not Empty(B)

  do PUSH(A, POP(B));

  POP(A);

  }

- $\Phi(D_i)$=2*[stack size of B after the i-th operation]

Enqueue:

$$c_i = 1$$
$$\Phi(D_i) - \Phi(D_{i-1}) = 2$$
$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$
$$= 3 = O(1)$$

Dequeue:

$$c_i = 2S_B + 1 + \varepsilon$$
$$\Phi(D_i) - \Phi(D_{i-1}) = 0 - 2S_B = -2S_B$$
$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$
$$= 2S_B + 1 + \varepsilon - 2S_B = O(1)$$

Dynamic tables:

Operations: Table-Delete, Table-Insert

Load factor: $\alpha(T)$ **=** (number of items)/(table size)

Define the load factor of an empty Table as 1.

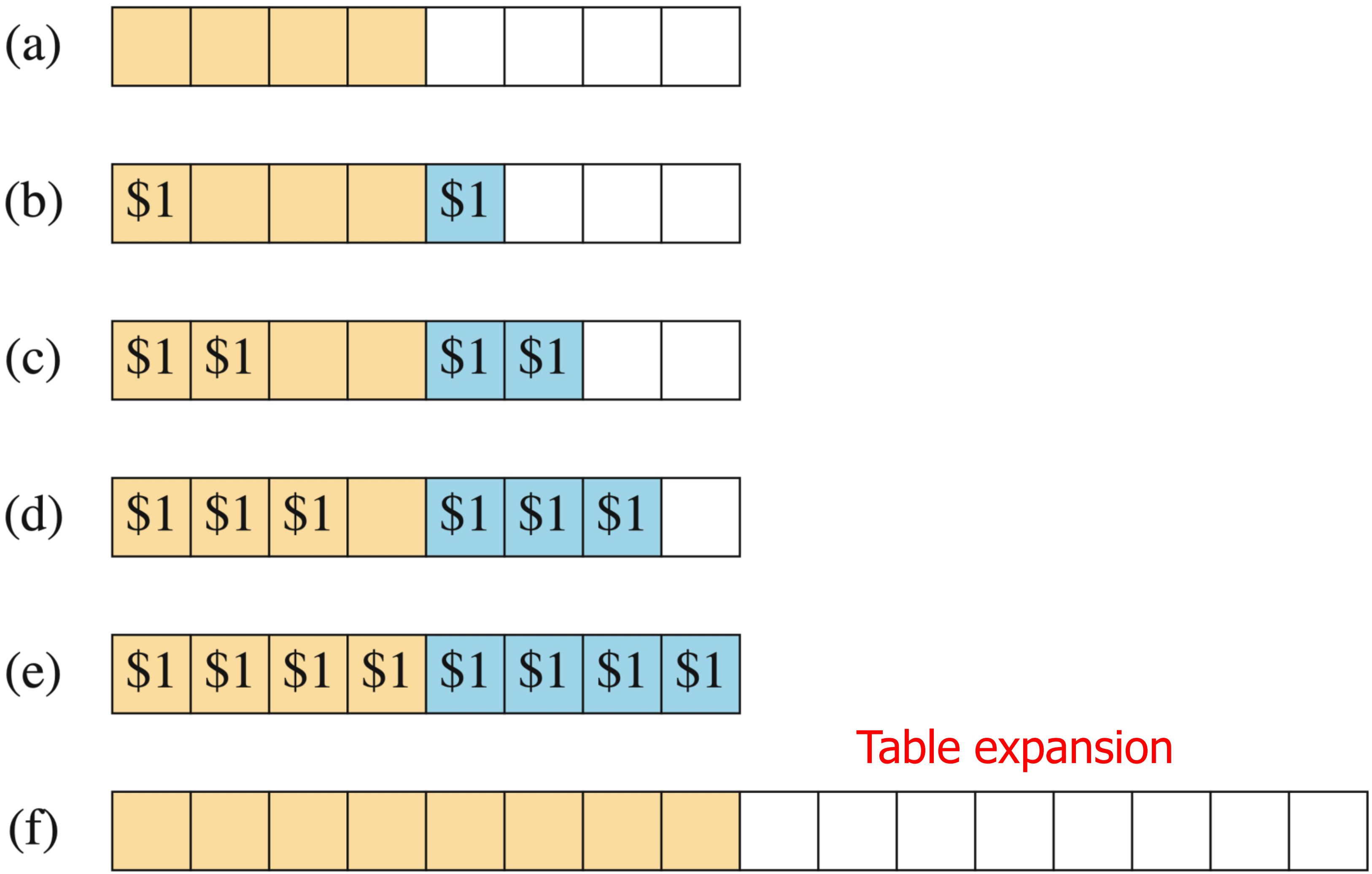
Table T:

i	**100**	**11**	**7**			

$$\alpha(T) = ?$$

TABLE-INSERT(T, x)

```
if T.size == 0
    allocate T.table with 1 slot
    T.size = 1
if T.num == T.size
    allocate new-table with 2 · T.size slots
    insert all items in T.table into new-table
    free T.table
    T.table = new-table
    T.size = 2 · T.size
insert x into T.table
T.num = T.num + 1
```


(a)
(b) $1 $1
(c) $1 $1 $1 $1
(d) $1 $1 $1 $1 $1 $1
(e) $1 $1 $1 $1 $1 $1 $1 $1
Table expansion
(f)

Define potential function: $\Phi(T) = 2T.num - T.size$

No expansion after the ith op: $(\mathrm{size}_i = \mathrm{size}_{i-1})$

$$\hat{c}_i = c_i + \Phi_i - \Phi_{i-1}$$

$$= 1 + (2num_i - size_i) - (2num_{i-1} - size_{i-1})$$

$$= 1 + (2num_i - size_i) - (2(num_i - 1) - size_{i-1})$$

$$= 3$$

With expansion after the ith op: $(\mathrm{size}_i/2 = \mathrm{size}_{i-1} = \mathrm{num}_i - 1)$

$$\hat{c}_i = c_i + \Phi_i - \Phi_{i-1}$$

$$= num_i + (2num_i - size_i) - (2num_{i-1} - size_{i-1})$$

$$= num_i + (2num_i - (2num_i - 2)) - (2(num_i - 1) - (num_i - 1))$$

$$= num_i + 2 - (num_i - 1)$$

$$= 3$$

Table expansion and contraction:

I, D, I, I, D, D, I, I, I……… : a sequence of n operations.

Preserve two properties:

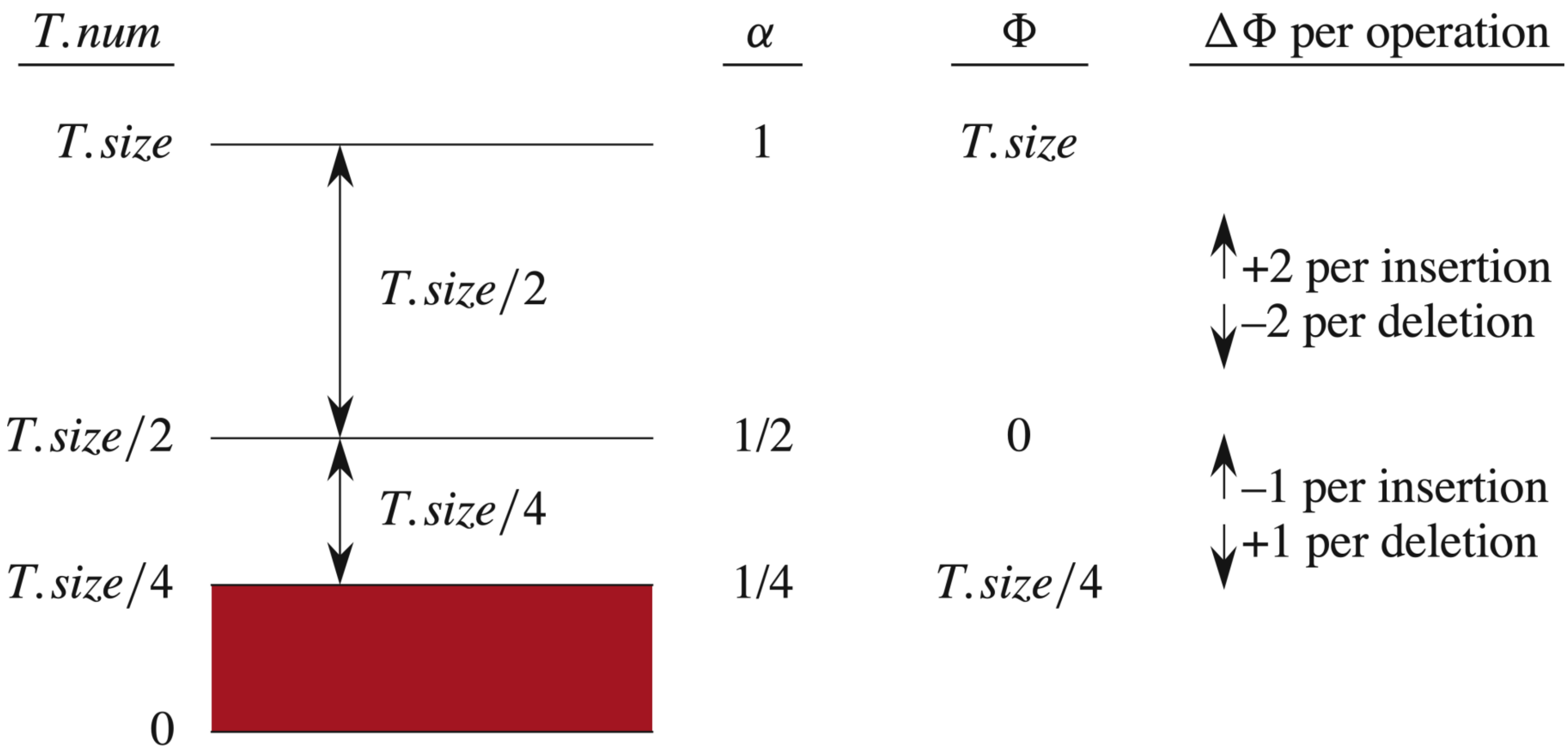
. Load factor is bounded below by a constant.

. The amortized cost of a table op is bounded above by a const.

Strategy:

.Double the table size when an item is inserted into a full table.

.Halve the table size when a deletion causes the table to become less than **¼** full.



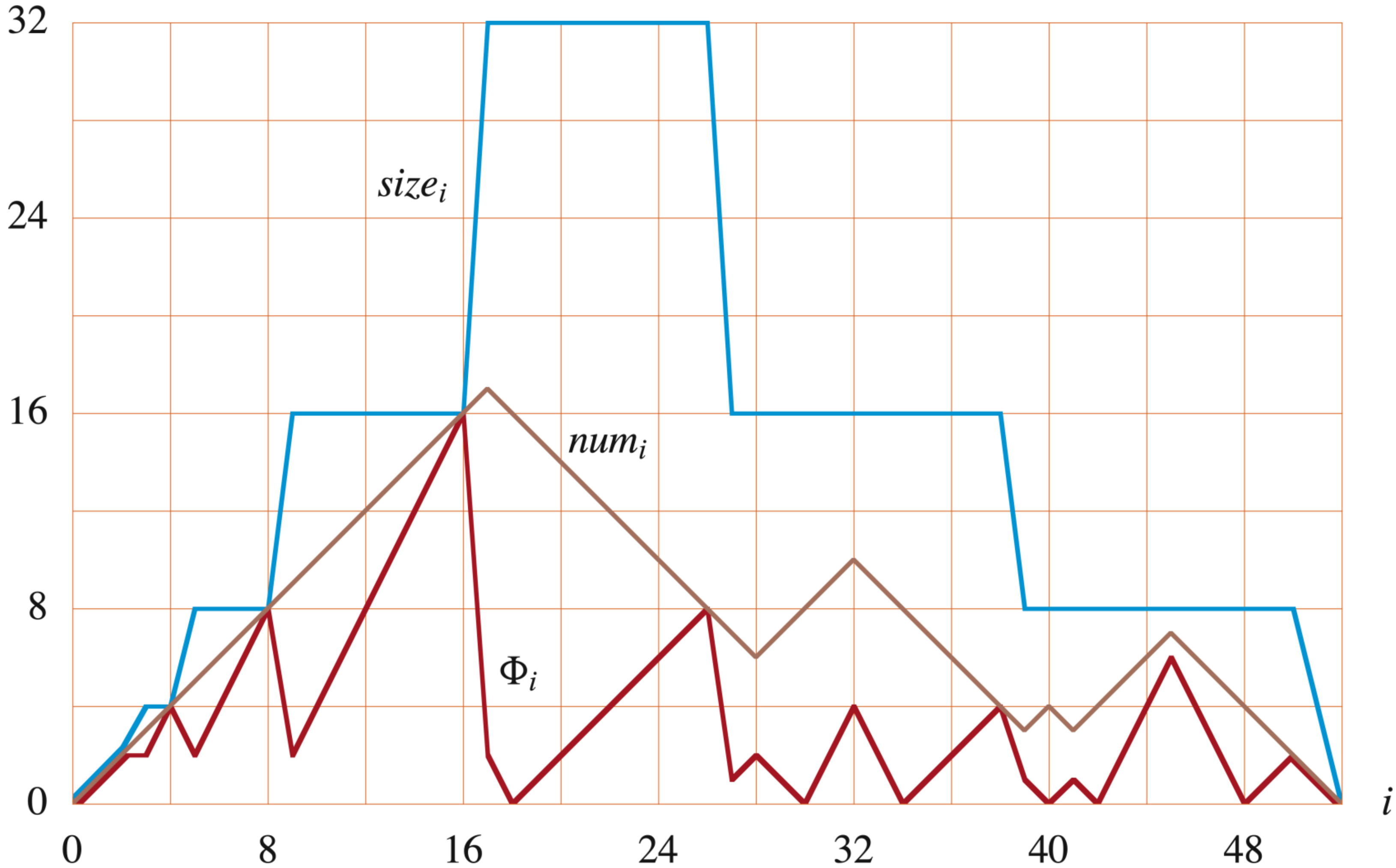
T.num
α
Φ
ΔΦ per operation
T.size
1
T.size
T.size/2
+2 per insertion
−2 per deletion
T.size/2
1/2
0
T.size/4
−1 per insertion
+1 per deletion
T.size/4
1/4
T.size/4
0

Define potential function:

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

- if $\alpha(T) \geq 1/2$, then the potential
  - ✓ increases by 2 for an insertion and
  - ✓ decreases by 2 for a deletion

- if $\alpha(T) < 1/2$, then the potential
  - ✓ increases by 1 for a deletion and
  - ✓ decreases by 1 for an insertion.



32
24
16
8
0
$size_i$
num_i
Φ_i
0
8
16
24
32
40
48
i

Define potential function:

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

- if the ith op is an insertion, its amortized cost $\widehat{c_i} = c_i + \Delta\Phi_i$ is:
  - 1+2=3 if $\alpha(T) \geq 1/2$, and
  - 1+(-1)=0 if $\alpha(T) < 1/2$,
- if the ith op is a deletion, its amortized cost $\widehat{c_i} = c_i + \Delta\Phi_i$ is:
  - 1 +(-2)=-1 if $\alpha(T) \geq 1/2$, and
  - 1 + 1 = 2 if $\alpha(T) < 1/2$

Define potential function:

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

Four cases remain:

1. an insertion that takes the load factor from below 1/2 to 1/2,
2. a deletion that takes the load factor from 1/2 to below 1/2,
3. a deletion that causes the table to contract, and
4. an insertion that causes the table to expand.

The last case has been shown that its amortized cost is 3.

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

When the ith operation is deletion that causes the table to contract:

- Before contraction:

$$num_{i-1} = size_{i-1}/4$$

- After deleting the item and contraction:

$$num_i = \frac{size_i}{2} - 1$$

- $\Phi_{i-1} = \frac{size_{i-1}}{2} - num_{i-1} = \frac{size_{i-1}}{4}$

- $\Phi_i = \frac{size_i}{2} - num_i = 1$
- $\hat{c}_i = c_i + \Phi_i - \Phi_{i-1} = \frac{size_{i-1}}{4} + \left(1 - \frac{size_{i-1}}{4}\right) = 1$

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

Handle the cases where $\alpha(T)$ fits one case of the above equation before the operation and the other case afterward:

- If the ith op is deletion without contraction:

  $num_{i-1} = \frac{size_{i-1}}{2}, \alpha_{i-1} = 1/2$ before deletion.

- After deleting the item:

  $num_i = \frac{size_i}{2} - 1$ and $\alpha_i < 1/2$

- $\Phi_{i-1} = 2num_{i-1} - size_{i-1} = 0$

- $\Phi_i = \frac{size_i}{2} - num_i = 1$

- $\hat{c}_i = c_i + \Phi_i - \Phi_{i-1} = 1 + (1 - 0) = 2$

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

Handle the cases where $\alpha(T)$ fits one case of the above equation before the operation and the other case afterward:

- If the ith op is insertion and takes the load factor from below ½ to equaling ½:

$$num_{i-1} = \frac{size_{i-1}}{2} - 1, \alpha_{i-1} < 1/2$$ before insertion.

- After inserting the item:

$$num_i = \frac{size_i}{2} \text{ and } \alpha_i = 1/2$$

- $\Phi_{i-1} = \frac{size_{i-1}}{2} - num_{i-1} = 1$
- $\Phi_i = 2(num_i - \frac{size_i}{2}) = 0$
- $\hat{c}_i = c_i + \Phi_i - \Phi_{i-1} = 0$

In summary, since $\hat{c}_i = O(1)$, the actual time for any sequence of n operations on a dynamic table is $O(n)$.

Example: Use aggregate analysis to determine the amortized cost per operation for a sequence of n operations on a data structure in which the ith operation costs i if i is an exact power of 2, and 1 otherwise.