Algorithms
Chapter 17
Amortized Analysis

Associate Professor: Ching-Chi Lin
林清池 副教授
chingchi.lin@gmail.com

Department of Computer Science and Engineering
National Taiwan Ocean University

Outline

- **Aggregate analysis** 總量分析
- The accounting method
- The potential method
- Dynamic tables

Amortized analysis 分析一序列的 operations，不是只有一個

- Analyze a **sequence** of operations on a data structure.
  目標：要說明單一 operation 可能很花時間，但平均而言，每一個花的時間不多
- **Goal:** Show that although some individual operations may be expensive, on **average** the cost per operation is small.
  - Average in this context does not mean that we're averaging over a distribution of inputs.
- No probability is involved. 沒有牽涉到机率分析
- We're talking about average cost in the worst case. 平均而言最壞情形
- We show that for all n, a sequence of n operations takes worst-case time $T(n)$ in total. 算的是總量

Example 1: Stack operations

- Stack operations: PUSH(S, x), POP(S), and MULTIPOP(S,k).
- PUSH(S, x): push object x onto stack S.
  - Each runs in $O(1)$ time.
  - A sequence of n PUSH operations takes $O(n)$ time.
- POP(S): pop the top of stack S and returns the popped object.
  - Each runs in $O(1)$ time.
  - A sequence of n POP operations takes $O(n)$ time.
- MULTIPOP(S,k)
  1. **while** not STACK-EMPTY(S) and $k > 0$
  2. POP(S)
  3. $k \leftarrow k-1$

Running time analysis$_{1/2}$

- Running time of MULTIPOP(S,*k*): 在 stack 中的個数：S 要 pop 的個数：k
  - Let each PUSH/POP cost $O(1)$.
  - The number of iterations of while loop is min(*s*, *k*), where *s* = number of objects on stack.
  - Therefore, total cost = min(*s*, *k*). 花费 = min(S, k)

- The running time of a sequence of *n* PUSH, POP, MULTIPOP operations?

- **Analysis(I):**
  - Worst-case cost of MULTIPOP is $O(n)$.
  - Have *n* operations.
  - Therefore, worst-case cost of sequence is $O(n^2)$. → $n \cdot O(n) = O(n^2)$

Running time analysis$_{2/2}$

- **Analysis(II):**
  - Each object can be popped only once per time that it's pushed.
  - At most n objects are pushed into S.
  - Have $\leq n$ PUSHes $\Rightarrow \leq n$ POPs, including those in MULTIPOP.
  - Therefore, total cost = $O(n)$.
  - Average cost of an operation = $O(1)$.
- Emphasize again, no probabilistic reasoning was involved.
  - Showed worst-case $O(n)$ cost for sequence.
  - Therefore, $O(1)$ per operation on average.

一个物件放到stack中，最多只会取出一次

最多放n个到stack中

multipop次数 + pop次数 ≤ n

push次数 ≤ n

⇒ multipop次数 + pop次数 + push次数 ≤ 2n

↳ 没有机率分析

進位由右至左 將"1"→"0"
"0"→"1"結束

Example 2: Incrementing a binary counter

- k-bit binary counter $A[0\,..\,k-1]$ of bits
  - $A[0]$ is the least significant bit.
  - $A[k-1]$ is the most significant bit.
  - Value of counter is $\sum_{i=0}^{k-1} A[i] \cdot 2^i$.
- Initially, counter value is 0.
- To add 1 (modulo 2^k), we use the following procedure.

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

進位

2進位表示法

Counter value	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	Total cost
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	0	1	0	0	7
5	0	0	0	0	0	1	0	1	8
6	0	0	0	0	0	1	1	0	10
7	0	0	0	0	0	1	1	1	11
8	0	0	0	0	1	0	0	0	15
9	0	0	0	0	1	0	0	1	16
10	0	0	0	0	1	0	1	0	18
11	0	0	0	0	1	0	1	1	19
12	0	0	0	0	1	1	0	0	22
13	0	0	0	0	1	1	0	1	23
14	0	0	0	0	1	1	1	0	25
15	0	0	0	0	1	1	1	1	26
16	0	0	0	1	0	0	0	0	31

Running time analysis

- The running time of a sequence of n INCREMENT operations ?

一個 operation 最差要 O(k)

- **Analysis(I) :**
  - A single execution of INCREMENT takes time $O(k)$ in the worst case.
  - Have n operations. 有 n 個 operation → O(nk)
  - Therefore, worst-case cost of sequence is $O(nK)$.
  - Average cost of an operation = $O(k)$.
- **Analysis(II):**
  - $A[0]$ flips every time, $A[1]$ flips only every other time, $A[2]$ flips only every fourth time, and so on.
  - Total number of flips is $T(n) = n + \lfloor n/2 \rfloor + \lfloor n/4 \rfloor + \ldots \leq 2n$

第 0 個 每 1 次 要翻轉
1 , 2 次
2 , 4 次
3 , 8 次

  - Average cost of an operation = $O(1)$.

Outline

- Aggregate analysis
- **The accounting method** 每個物件有自己的個人帳戶
- The potential method
- Dynamic tables

未雨綢繆：先付錢後消費

The accounting method 1/2

讓不同的operation有不同的cost

- Assign different charges to different operations.
  - Some are charged more than actual cost. 有些比實際多
  - Some are charged less. 有些比實際少
- The amount we charge an operation is called its **amortized cost**. 付的cost叫amoritized cost
- When amortized cost > actual cost, store (amortized cost − actual cost) on specific objects in the data structure as **credit**. 多付的cost叫credit
- Use credit later to pay for operations whose actual cost > amortized cost. 不夠付時用credit付
- Differs from aggregate analysis:
  - In the accounting method, different operations can have different costs. 在 accounting metod，每一個 operation 有不同的 cost
  - In aggregate analysis, all operations have same cost. 在aggregate每一個cost都相同

保證 amorized cost ≥ actual cost
付的錢 ≥ 實際花費

The accounting method 2/2

- Need credit to **never go negative**.
  - Otherwise, have a sequence of operations for which the amortized cost is not an upper bound on actual cost. 不可以信用破產否則分析無效 (amoritzed cost < actual cost)
  - Amortized cost would tell us nothing.
- Let c_i = actual cost of ith operation, 第 i 個動作真正的花費
  $\hat{c}_i$ = amortized cost of ith operation. 第 i 個動作付的花費
- Then require $\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$ for all sequences of n operations.

前 i 個動作付的錢 ≥ 前 i 個動作實際的花費

- Total credit stored in the data structure = $\sum_{i=1}^{n} \hat{c}_i - \sum_{i=1}^{n} c_i$.

儲存的 credit = $\sum_{i=1}^{n} \hat{c}_i - \sum_{i=1}^{n} c_i$

Example 1: Stack operations

operation	actual cost	amortized cost
PUSH	1	2
POP	1	0
MULTIPOP	min(k, s)	0

push 付 $2：$1 為了 push
$1 為了 pop 或 multipop

每個在 stack 中的 object
有 $1，credit ≥ 0

- **Intuition**: When pushing an object, pay 2.
  - $1 pays for the PUSH.
  - $1 is prepayment for it being popped by either POP or MULTIPOP.
  - Since each object has $1, which is credit, the credit ≥ 0.
  - Therefore, total amortized cost ≤ $2n$, is an upper bound on total actual cost.
  - Average cost of an operation = $O(1)$.

total cost ≤ amortized cost ≤ 2n
↑
credit ≥ 0

Example 2: Incrementing a binary counter

- Charge \$2 to set a bit to 1.
  - \$1 pays for setting a bit to 1.
  - \$1 is prepayment for flipping it back to 0.
  - Have \$1 of credit for every 1 in the counter.
  - Therefore, credit ≥ 0.

當設定一個 bit 時，付 \$2
set 花 \$1
\$1 留著翻轉
所以 credit ≥ 0

- Amortized cost of INCREMENT:
  - Cost of resetting bits to 0 is paid by credit.
  - At most 1 bit is set to 1.
  - Therefore, amortized cost ≤ \$2.
  - For n operations, amortized cost = $O(n)$.
  - Average cost of an operation = $O(1)$.

一次最多設定一個 bit
total cost ≤ amortized cost ≤ 2n
↑
credit ≥ 0

Outline

- Aggregate analysis
- The accounting method
- **The potential method**
- Dynamic tables

个人帐户

健保：① $\Phi(D_i) \geq 0$ （健保永远为正）

② $\sum_{i=1}^{n} \hat{c}_i = O(n)$：全部的健保费

全体物件有一个共同的帐户

全体物件有一個共同的帳戶

The Potential method$_{1/2}$

- Like the accounting method, but think of the credit as **potential** stored with the **entire data structure**.
  - Can release potential to pay for future operations.
  - Most flexible of the amortized analysis methods.
- Let D_i = data structure after ith operation, 在第 i 個 operation 後健保局的
  D_0 = initial data structure,
  c_i = actual cost of ith operation, 第 i 個 operation 的實際花費
  健保費 ← $\hat{c}_i$ = amortized cost of ith operation.
- **Potential function** $\Phi : D_i \rightarrow \mathrm{R}$
  - $\Phi(D_i)$ is the potential associated with data structure D_i.

$$\hat{c}_i = c_i + \underbrace{\Phi(D_i) - \Phi(D_{i-1})}_{\text{increase in potential due to } i\text{th operation}}$$

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

1000 = 300 + 700

健保費 感冒花費 健保局增加

The Potential method$_{2/2}$

- $\text{Total amortized cost} = \sum_{i=1}^{n} \hat{c}_i$

  全部付的錢

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

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

- If we require that $\Phi(D_i) \geq \Phi(D_0)$ for all i, then the amortized cost is always an upper bound on actual cost.

  如果 $\Phi(D_i) \geq \Phi(D_0)$，則前 i 個動作付的錢 ≥ 前 i 個實際的花費

- In practice: $\Phi(D_0) = 0$, $\Phi(D_i) \geq 0$ for all i.

  實作上，我們設定 $\Phi(D_0) = 0$，且讓 $\Phi(D_i) \geq 0$ ←健保不能倒

困難点：決定 potential function Φ

要繳的健保費是由 actual cost 和Φ算出

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

Example 1: Stack operations

- Φ = # of objects in stack.

存款 = 在stack中的杯目

- D_0 = empty stack $\Rightarrow \Phi(D_0) = 0$.

step1：確定 potential function

step2：確定存款比一開始多

⇒ $\Phi(D_i) \geq \Phi(D_0)$ for all i

- Since # of objects in stack ≥ 0, $\Phi(D_i) \geq 0 = \Phi(D_0)$ for all i. 存款為正

operation	actual cost	$\Phi(D_i) - \Phi(D_{i-1})$	amortized cost
PUSH	1	$(s + 1) - s = 1$	$1 + 1 = 2$
POP	1	$(s - 1) - s = -1$	$1 - 1 = 0$
MULTIPOP	$k' = \min(k, s)$	$(s - k') - s = -k'$	$k' - k' = 0$

s = # of objects initially.

每一個 operation

- Therefore, amortized cost of a sequence of n operations $= \sum_{i=1}^{n} \hat{c}_i = O(n)$. 全部的健保費

step3：將繳交的健保費相加

step4：全部健保費 = 全部花費

⇒可知平均而言的花費

Example 2: Incrementing a binary counter 1/2

- $\Phi = b_i$ = # of 1's after ith INCREMENT. 存款：1的個数
- D_0 = all bits are set to zero $\Rightarrow \Phi(D_0) = 0$.
- Suppose ith operation resets t_i bits to 0. 假設 ith operation 設 t_i bit 為 0
  - $c_i \le t_i + 1$. (resets t_i bits, sets at most one bit to 1) ith operation 的花費 $\le t_i + 1$ (設成1)
  - If $b_i = 0$, the ith operation reset all k bits and didn't set one, so $b_{i-1} = t_i = k \Rightarrow b_i = b_{i-1} - t_i$. 存款=0，設所有 bit 為 0，沒設任何 bit 為 1 $\Rightarrow b_i = b_{i-1} - t_i$
  - If $b_i > 0$, the ith operation reset t_i bits, set one, so $b_i = b_{i-1} - t_i + 1$. 存款 $> 0 \Rightarrow b_i = b_{i-1} - t_i + 1$
  - In either case, $b_i \le b_{i-1} - t_i + 1$. 不論存款為何，$b_i \le b_{i-1} - t_i + 1$

Example 2: Incrementing a binary counter 2/2

- Therefore, $\Phi(D_i) - \Phi(D_{i-1}) = b_i - b_{i-1}$

健保局增加

$$\leq (b_{i-1} - t_i + 1) - b_{i-1}$$
$$= 1 - t_i.$$

健保费 = 實際花费 + 健保局增加

- The amortized cost is therefore $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

$$\leq (t_i + 1) + (1 - t_i)$$
$$= 2.$$

- Thus, amortized cost of n operations = $\sum_{i=1}^{n} \hat{c}_i = O(n)$.

全部的健保费 = O(n)

Outline

- Aggregate analysis
- The accounting method
- The potential method
- **Dynamic tables**

Dynamic tables$_{1/2}$

- **Scenario**
  - Have a table – maybe a hash table. 不知有多少物件会存在 table 中
  - Don't know in advance how many objects will be stored in it.
  - When it fills, must reallocate with a larger size, copying all objects into the new, larger table. 滿→產生一個大的
  - When it gets sufficiently small, might want to reallocate with a smaller size. 夠小時，將 size 變小
  - Initially, T is a table of size 0. 一開始 size = 0
  - Perform a sequence of n operations on T, each of which is either Insert or Delete. n 個 operation，每一個可能 delete or insert
- **Goals**
  - $O(1)$ amortized time per operation. 平均而言，每一個 operation 為常數時間
  - Unused space always ≤ constant fraction of allocated space.

$$\frac{\text{沒有用到的space}}{\text{全部space}} = \text{constant}$$

Dynamic tables$_{2/2}$

- **Load factor $\alpha(T)$** 負載率 = $\alpha(T)$
  - $\alpha(T)$ = num[T]/size[T]
    - num[T] = # items stored, size[T] = allocated size.
  - If size[T] = 0, then num[T] = 0. Define α = 1.
  - Never allow $\alpha > 1$.
  - Keep $\alpha >$ a constant fraction.

當 size[T] = 0, 則 num[T] = 0
⇒ 定義 $\alpha = 1$

Table expansion

```
TABLE-INSERT (T, x)
1.   if size[T] = 0
2.        then  allocate table[T] with 1 slot
3.              size[T] ← 1
4.   if num[T] = size[T]
5.        then  allocate new-table with 2 · size[T] slots
6.              insert all items in table[T] into new-table
7.              free table[T]
8.              table[T] ← new-table
9.              size[T] ← 2 · size[T]
10.  insert x into table[T]
11.  num[T] ← num[T] + 1
```

size=0 ⇒ 產生一個 space

滿了 ⇒ 2倍大

Notice : If only insertions are performed, the load factor of a table is always at least 1/2.

如果只有 insertion ⇒ $\alpha(T) \geq \frac{1}{2}$

- When the table becomes full, double its size and reinsert all existing items. 滿了 ⇒ 找一個2倍大的教室，將學生移過去
- Each time we actually insert an item into the table, it's an **elementary insertion**.

Running time analysis

- The running time of a sequence of n TABLE-INSERT operations on an initially empty table ?
- **Analysis(I) :**
  - c_i = actual cost of ith operation.
  - If not full, $c_i = 1$.
  - If full, have $i - 1$ items in the table at the start of the ith operation. Have to copy all $i - 1$ existing items, then insert ith item $\Rightarrow c_i = i$.
  - n operations $\Rightarrow c_i = O(n) \Rightarrow O(n^2)$ time for n operations.

沒滿 ⇒ $C_i = 1$

滿 ⇒ C_i = Copy $i-1$ object + insert i

$= i$

n 個 operations ⇒ $O(n^2)$

Aggregate analysis 總量分析

- **Analysis(II):** 換教室時間表：2, 3, 5, 9 ...
  - Expand only when $i-1$ is an exact power of 2.

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

- Totol cost $= \sum_{i=1}^{n} c_i \le n + \sum_{j=0}^{\lfloor \lg n \rfloor} 2^j$ $\quad 2^0+2^1+2^2+\dots+2^{\lfloor \lg n \rfloor}$

$$= n + \frac{2^{\lfloor \lg n \rfloor + 1} - 1}{2-1}$$

$$< n + 2n$$

$$= 3n$$

1 2 3 4 5 6 7 8 9
真正 → 1 2 3 1 5 1 1 1 9
計算 → 2 3 1 5 1 1 1 9 1
1 1 1 1 1 1 1 1 1
1 2 4 8
在 2^k 時会多算，2^k+1 才換教室

- Therefore, **aggregate analysis** says amortized cost per operation = 3.

Accounting method 存款存在物件上

- Charge $3 per insertion of *x*.
  - $1 pays for x's insertion. 花$1
  - $1 pays for *x* to be moved in the future. 幫自己存$1
  - $1 pays for some other item to be moved. 幫別人存$1
- Suppose that the size of the table is *m* immediately after an expansion. 如page 40的範例 換教室後沒有存款，m個insertion後才会再換教室
  - Assume that the expansion used up all the credit, so that there's no credit stored after the expansion.
  - Will expand again after another *m* insertions.
  - Each insertion will put $1 on one of the *m* items that were in the table just after expansion and will put $1 on the item inserted.
  - Have $2*m* of credit by next expansion, when there are 2*m* items to move. Enough to pay for the expansion, with no credit left over!

每次存$2，m個insert後有$2m又可以換教室

Potential method$_{1/3}$

- $\Phi(T) = 2 \cdot \text{num}[T] - \text{size}[T]$ 健保局的存款
  - Initially, $\text{num}[T] = \text{size}[T] = 0 \Rightarrow \Phi(T) = 0$.
  - Just after expansion, $\text{size}[T] = 2 \cdot \text{num}[T] \Rightarrow \Phi(T) = 0$. 換教室後
  - Just before expansion, $\text{size}[T] = \text{num}[T] \Rightarrow \Phi(T) = \text{num}[T]$ 換教室前
    $\Rightarrow$ have enough potential to pay for moving all items.
  - Always have $\Phi(T) \geq 0$.
    $\text{num}[T] \geq 1/2 \cdot \text{size}[T]$
    $\Rightarrow 2 \cdot \text{num}[T] \geq \text{size}[T]$
    $\Rightarrow \Phi(T) \geq 0$.

存款永遠為正
健保不能倒

- **Amortized cost of ith operation:**

  num_i = num[T] after ith operation,

  $size_i$ = size[T] after ith operation ,

  $\Phi_i = \Phi$ after ith operation.

potential method:

step1: 確定 potential function

step2: 確定存款比一開始多
⇒ 重(D_i) ≥ 重(D_0) for all i

step3: 將繳交的健保費相加

step4: 全部健保費 ≥ 全部花費
⇒ 可知平均而言的花費

考慮2种情形的花费 ①有換教室 ②沒有換教室 ⇒2种都要繳 $3

Potential method 2/3

- If no expansion:

$$\begin{aligned} size_i &= size_{i-1}, \\ num_i &= num_{i-1}+1, \\ c_i &= 1. \end{aligned}$$

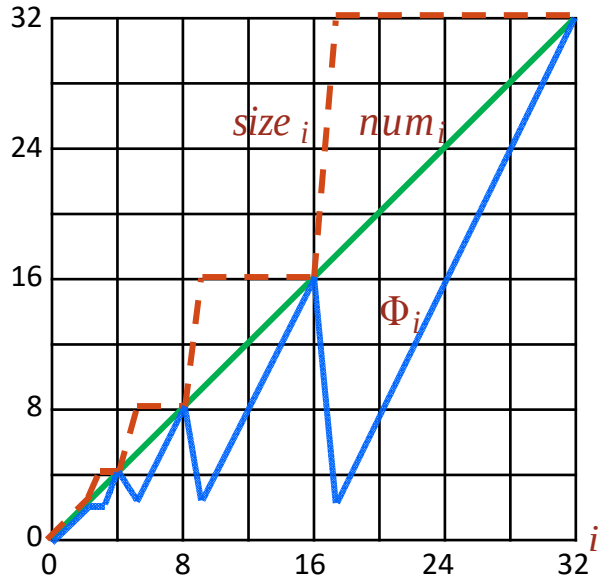
要繳的健保費

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

- If expansion:

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

Potential method$_{3/3}$



- The above Figure plots the values of num_i, $size_i$, and Φ_i against i.
- Notice how the potential builds to pay for the expansion of the table.

Expansion and contraction$_{1/2}$ 擴張和縮減

- When α drops too low, contract the table.
  - Allocate a new, smaller one.
  - Copy all items.

維持2個性質 ① $\alpha \geq$ constant ⇒ 使用量要和size成常数比
② 健保不能太貴 ⇒ amoritized cost ≤ constant

- Preserve two properties
  - load factor α bounded below by a positive constant, and
  - amortized cost per operation bounded above by a constant.

- **Obvious strategy**
  - Double size when inserting into a full table. 滿→2倍
  - Halve size when deletion would make table less than half full.
  - Then always have $1/2 \leq \alpha \leq 1$. 少於一半→減半

Expansion and contraction$_{2/2}$

- **Consider the following scenario**
  - The first $n/2$ operations are insertions, 前 $\frac{1}{2}$ 的 operation 是 insert
  - For the second $n/2$ operations, we perform the following sequence:
    insert, delete, delete, insert, insert, delete, delete, insert, insert,…
    double halve double halve double 接下來
  - The cost of each expansion and contraction is $\Theta(n)$. 不斷換教室
  - The total cost of the n operations is $\Theta(n^2)$. → $n \cdot O(n) = O(n^2)$ 最差的情形
- **Simple solution:**
  - Double size when inserting into a full table. 滿→2倍
  - Halve size when deleting from a 1/4 full table. 剩 $\frac{1}{4}$ 時減半
  - After either expansion or contraction, $\alpha = 1/2$.
  - Always have $1/4 \le \alpha \le 1$. 可以使 $\frac{1}{4} \le \alpha \le 1$

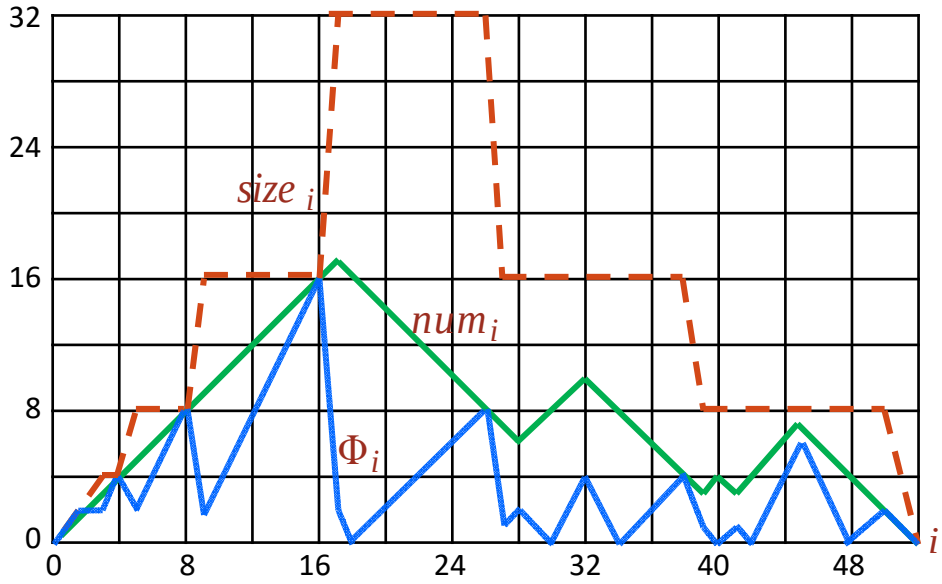
Some properties$_{1/3}$

小觀察1：在減半之前要delete一半
在expansion前要增成2倍人权

- **Observation 1:**
  - Need to delete half the items before contraction.
  - Need to double number of items before expansion.
- Let $\Phi(T) = \begin{cases} 2 \cdot num[T] - size[T] & \text{if } \alpha \geq 1/2, \\ size[T]/2 - num[T] & \text{if } \alpha < 1/2. \end{cases}$
  - T empty $\Rightarrow \Phi = 0$.
  - $\alpha \geq 1/2 \Rightarrow num \geq 1/2 \cdot size \Rightarrow 2 \cdot num \geq size \Rightarrow \Phi \geq 0$.
  - $\alpha < 1/2 \Rightarrow num < 1/2 \cdot size \Rightarrow \Phi \geq 0$.

確定存款比一開始多

Some properties$_{2/3}$



- The potential is never negative. Thus, the total amortized cost of a sequence of operations with respect to Φ is an upper bound on the actual cost of the sequence.

Some properties 3/3

小觀察2：

$\alpha = \frac{1}{2}$ 時，沒 credit，沒存款

當 $\alpha = 1$ 要 double 或 $\alpha = \frac{1}{4}$ 要減半時，

都有足夠的 credit 可以搬遷

- **Observation 2:**
  - $\alpha = 1/2 \Rightarrow \Phi = 2 \cdot num - 2 \cdot num = 0$.
  - $\alpha = 1 \Rightarrow \Phi = 2 \cdot num - num = num$. 滿
  - $\alpha = 1/4 \Rightarrow \Phi = size/2 - num = 4 \cdot num/2 - num = num$. 減半
  - Therefore, when we double or halve, have enough potential to pay for moving all num items.

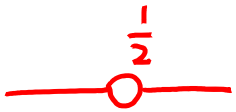
- Let c_i = actual cost of ith operation,
  $\hat{c}_i$ = amortized cost of ith operation,
  num_i = the number of items after the ith operation,
  $size_i$ = the size of the table after the ith operation,
  α_i = the load factor of the table after the ith operation,
  Φ_i = the potential after the ith operation.

Analysis: insert operation$_{1/2}$

- Case 1: $\alpha_{i-1} \geq 1/2$

  利用 α_{i-1} 的值，將 insertion 情況分 3 種

  - The same analysis as before.
  - The amortized cost $\hat{c}_i = 3$.



- Case 2: $\alpha_{i-1} < 1/2$ and $\alpha_i < 1/2$ (no expansion)

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i + \Phi_{i-1} \\
&= 1 + \left(size_i / 2 - num_i\right) - \left(size_{i-1} / 2 - num_{i-1}\right) \\
&= 1 + \left(size_i / 2 - num_i\right) - \left(size_i / 2 - \left(num_i - 1\right)\right) \\
&= 0 .
\end{aligned}$$

insert 之前小於 $\frac{1}{2}$，insert 後不需換教室

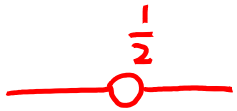
Analysis: insert operation$_{2/2}$

- Case 3: $\alpha_{i-1} < 1/2$ and $\alpha_i \geq 1/2$ (no expansion)

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

- Therefore, amortized cost of insert is at most 3.

利用 α_{i-1} 的值将 delete 情形分为 2 种



Analysis: delete operation$_{1/3}$

- Case 1: $\alpha_{i-1} < 1/2$
  - This implies $\alpha_i < 1/2$. delete 之後 $\alpha_i < \frac{1}{2}$
  - If no contraction: 沒有發生縮減

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

  - If contraction: 發生縮減

$$\begin{aligned}\hat{c}_i &= \underbrace{\left(num_i+1\right)}_{\text{move}+\text{delete}}+\left(size_i/2-num_i\right)-\left(size_{i-1}/2-num_{i-1}\right)\\ &\quad \left[size_i/2 = size_{i-1}/4 = num_{i-1} = num_i+1\right]\\ &= \left(num_i+1\right)+\left(\left(num_i+1\right)-num_i\right)-\left(\left(2\cdot num_i+2\right)-\left(num_i+1\right)\right)\\ &= 1\,.\end{aligned}$$

Analysis: delete operation$_{1/2}$

- Case 2: $\alpha_{i-1} \geq 1/2$ (No contraction happens)
  - Case 2a: $\alpha_i \geq 1/2$:

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

  - Case 2b: $\alpha_i < 1/2$:

Since $\alpha_{i-1} \geq 1/2$, we have

$num_i = num_{i-1} - 1 \geq \frac{1}{2} \cdot size_{i-1} - 1 = \frac{1}{2} \cdot size_i - 1.$

Thus,

$$\begin{aligned}\hat{c}_i &= 1+(size_i/2 - num_i) - (2\cdot num_{i-1} - size_{i-1}) \\ &= 1+(size_i/2 - num_i) - (2\cdot num_{i-1} - size_i) \\ &= -1 + \frac{3}{2}\cdot size_{i-1} - 3\cdot num_i \\ &\leq -1 + \frac{3}{2}\cdot size_{i-1} - 3\left(\frac{1}{2}\cdot size_{i-1} - 1\right) \\ &= 2.\end{aligned}$$

Summary

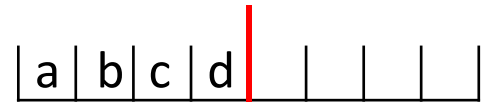
- Therefore, amortized cost of delete is at most 2.
- The amortized cost of each operation is bounded above by a constant .
- The actual time for any sequence of n operations on a dynamic table is $O(n)$.

delete 健保最多付$2

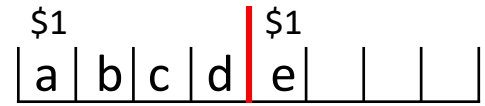
n個operations最多付O(n)元

Insertion Only ($\alpha \geq 1/2$)

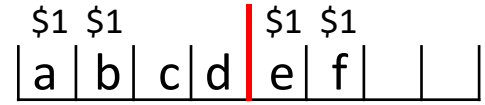
C = 0



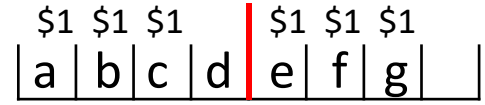
C = 2 Pay $3(case 1a)



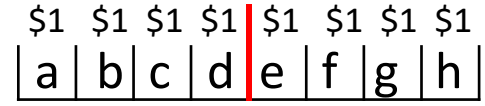
C = 4 Pay $3



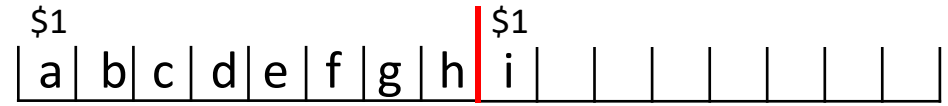
C = 6 Pay $3



C = 8 Pay $3

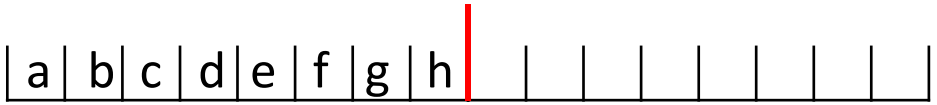


C = 2 Pay $3(case 1b)

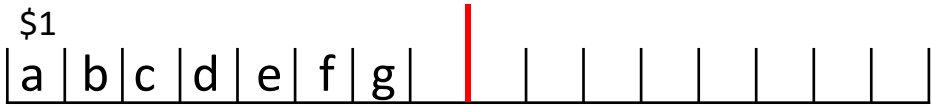


Delection Only ($\alpha \leq 1/2$)

C = 0



C = 1 Pay $2(case 1a)



C = 2 Pay $2

C = 3 Pay $2

C = 4 Pay $2

C = 1 Pay $1(case 1b)