Slido: **#ADA2020**

CSIE 2136 Algorithm Design and Analysis, Fall 2020

Amortized Analysis

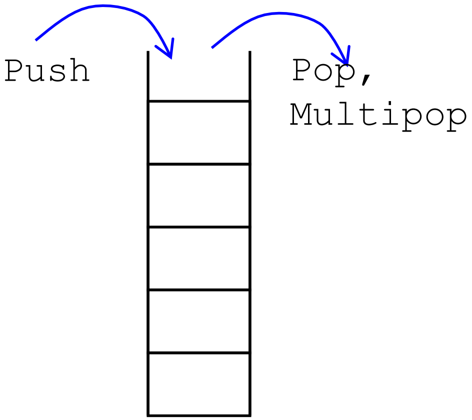
Hsu-Chun Hsiao

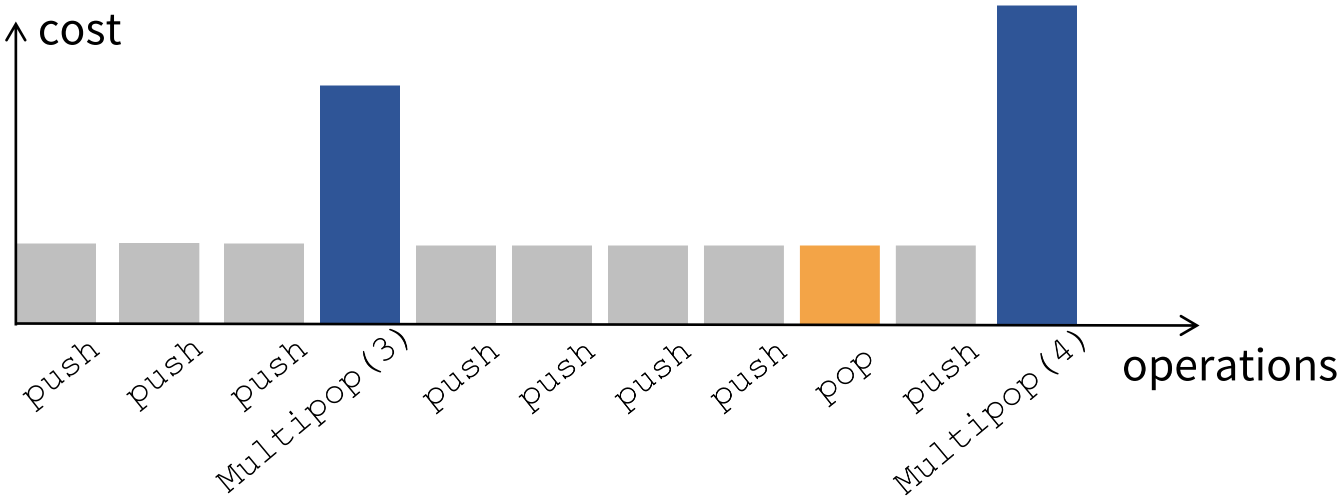
Agenda

- Why amortized analysis (均攤分析)
- Aggregate method (聚集方法)
- Accounting method (記帳方法) or banker's method
- Potential method (位能方法) or physicist's method

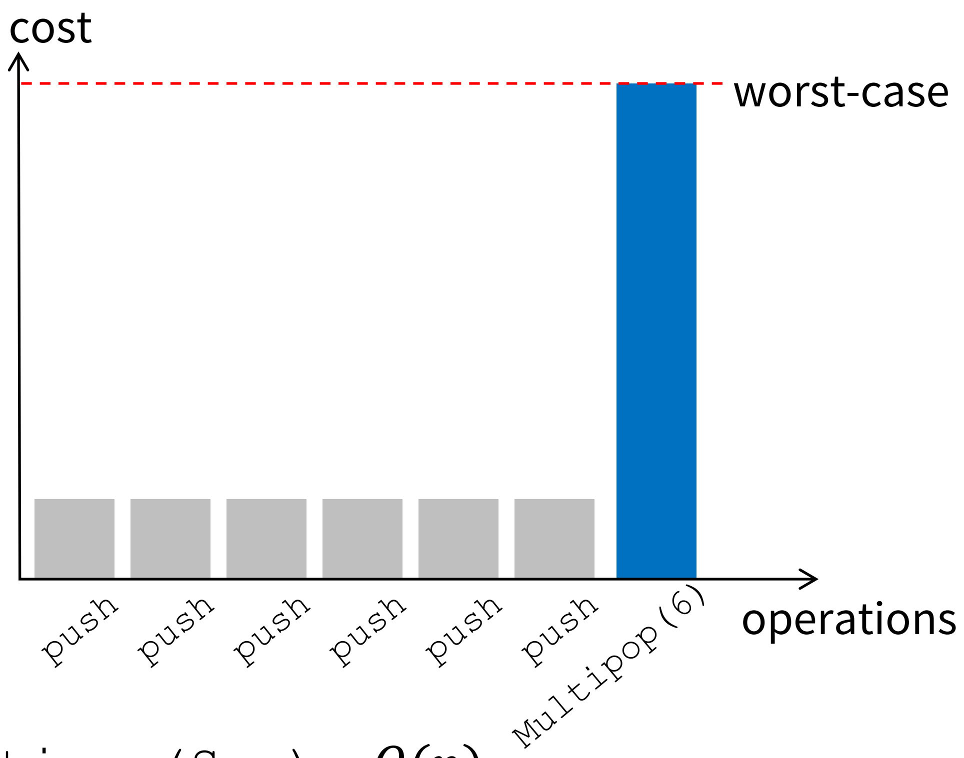
Operations on data structure

- A data structure comes with operations that organize the stored data
- Some operations may be slower than others
- The cost of the same operation may vary
- <u>Example</u>: stack supports `Push`, `Pop`, `Multipop`





Worst case analysis may be loose



<u>Cost of stack operations</u>
`push(S,x)` $= O(1)$
`pop(S)` $= O(1)$
`multipop(S,k)` $= O(\min\{|S|, k\})$

Worst-case time of the n^{th} operation = `multipop(S,n)` $= O(n)$
=> Worst-case time of a sequence of n operations $= O(n^2)$
However, this worst-case bound is not tight because this expensive `multipop` operation can't occur so frequently!

Goal of Amortized analysis

- Obtain an accurate worst-case bound in executing a sequence of operations on a given data structure

- An upper bound for ANY valid sequence of n operations

All of the valid operation sequences on stack when $n = 3$:

```
push, push, pop
push, push, multipop(1)
push, push, multipop(2)
push, pop, push
push, multipop(1), push
```

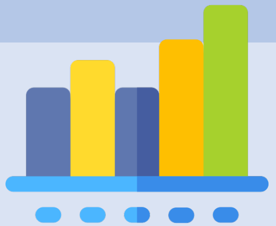
Types of running-time analysis

Worst case	Running time guarantee for any input of size n
Average case	Expected running time for a random input of size n
Probabilistic	Expected running time of a randomized algorithm
Amortized	Worst-case running time for a sequence of n operations

Amortized analysis: 3 common techniques

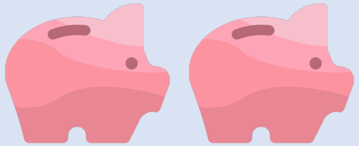
Aggregate method (聚集方法)

- Determine an upper bound on the cost over any sequence of n operations, $T(n)$
- The average cost per operation is then $T(n)/n$
- All operations have the same amortized cost



Accounting method (記帳方法)

- Each operation is assigned an amortized cost (may differ from the actual cost)
- Each object of the data structure is associated with a credit
- Need to ensure that every object has sufficient credit at any time



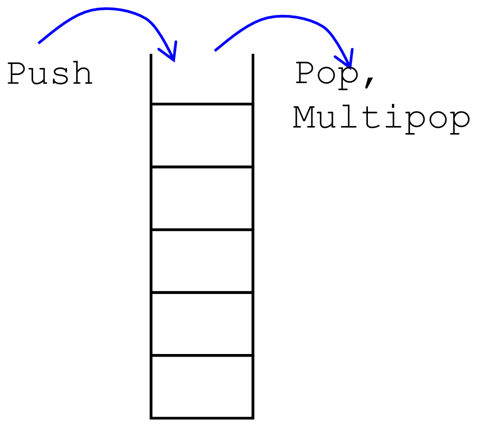
Potential method (位能方法)

- Similar to accounting method; each operation is assigned an amortized cost
- The data structure as a whole maintains a credit (i.e., potential)
- Need to ensure that the potential level is nonnegative at any time

Note: these are for analysis purpose only, not for implementation!

Example #1: stack

Implemented using an array or linked list:

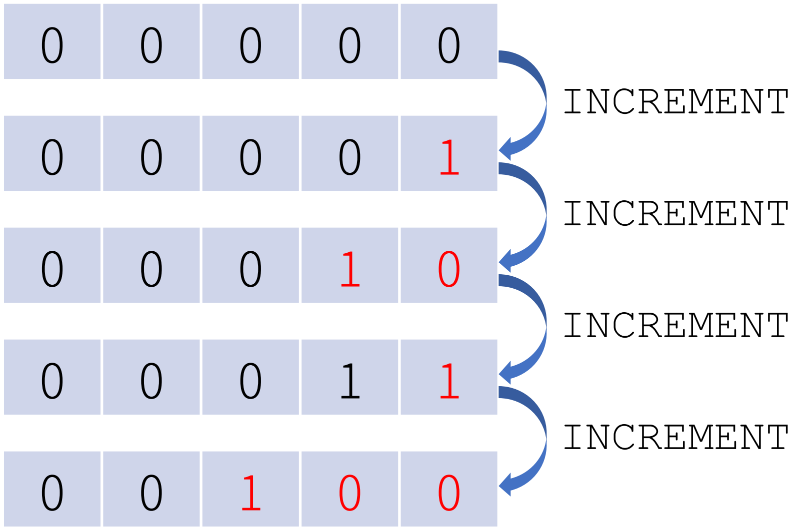



```
MULTIPOP(S,k):
    while not STACK-EMPTY(S) and k > 0
        POP(S)
        k = k - 1
```

Operation type	Cost
`Push(S,x)`	$O(1)$
`Pop(S)`	$O(1)$
`Multipop(S,k)`: pop top k elements at once	$O(\min\{\lvert S\rvert, k\})$

Example #2: k-bit counter

- Counts up from 0 by single operation, `INCREMENT`
- Implemented using a k-bit array
- Cost of `INCREMENT` is $O(k)$ in the worst case




```
INCREMENT(A):
    i = 0
    while i < A.length and A[i] == 1
        A[i] = 0
        i = i + 1
    if i < A.length
        A[i] = 1
```

More examples

- Redundant ternary counter
- Dynamic binary search
- Queue as two stacks

Aggregate Method (聚集方法)

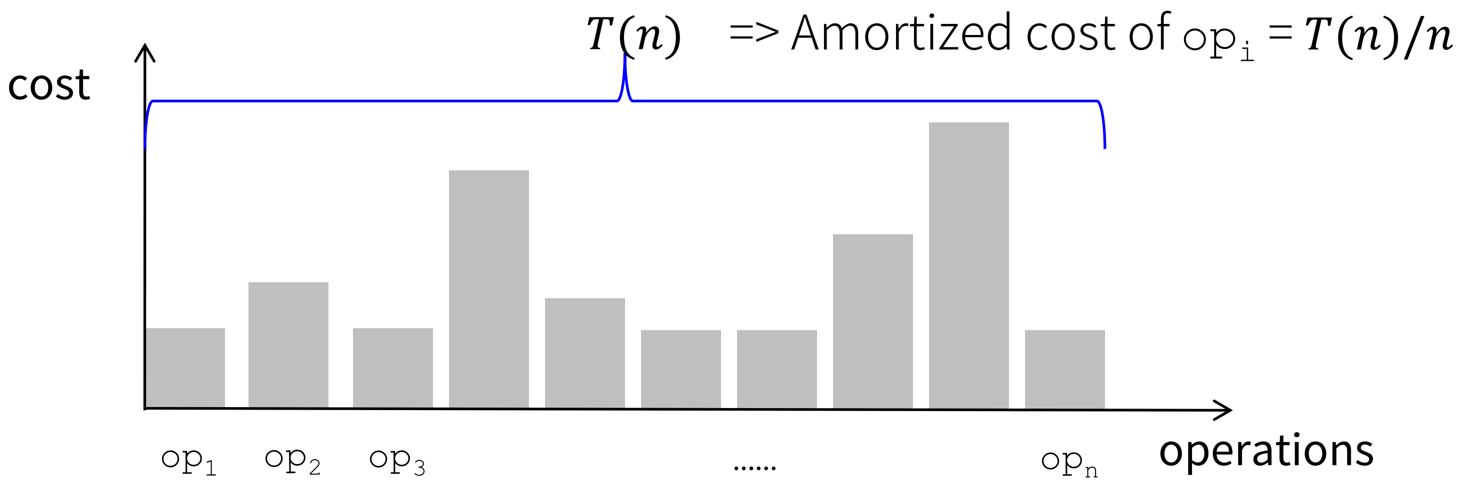
Chapter 17.1

Aggregate method

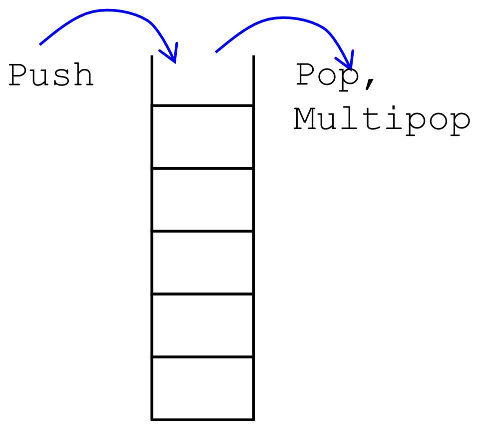
Idea: 直接觀察 n 次操作的總花費的上限

Approach:

1. Determine an upper bound $T(n)$ on the cost of any sequence of n operations
2. Calculate the amortized cost per operation as $T(n)/n$
   - All operations have the **same amortized cost**



Aggregate method for stack



Observation: # popped elements $\leq$ # of pushed elements

- 出來的不可能比進去的多

- For a sequence of n operations, maximum # of `push` is n
- => Total cost for entire sequence is $O(n)$
- => Amortized cost per operation is $O(n)/n = O(1)$

Aggregate method for k-bit counter

Counter value	A[3]	A[2]	A[1]	A[0]	Total cost of first n operations
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	3
3	0	0	1	1	4
4	0	1	0	0	7
5	0	1	0	1	8
6	0	1	1	0	10
7	0	1	1	1	11
8	1	0	0	0	15

Flip every increment

Flip every 2 increments

Flip every 4 increments

Flip every 8 increments

Aggregate method for k-bit counter

Observation: Total # of bit flips in n increment operations

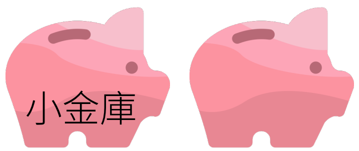
$$= n + n/2 + n/4 + \cdots + n/2^k$$
$$< 2n$$

- => total cost of the sequence is $O(n)$
- => Amortized cost per operation is $O(n)/n = O(1)$

Accounting Method (記帳方法)

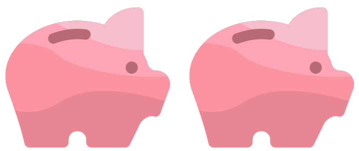
Chapter 17.2

Accounting method: Idea



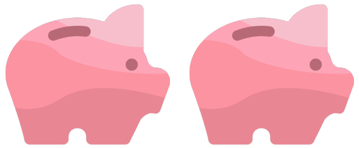
- 每個操作指定一個均攤費用 (amortized cost)
- 若個別操作的均攤費用合理，則算出 n 個操作的均攤費用
  - Q: 要怎麼知道個別操作的均攤費用是合理的？
- 檢查個別操作的均攤費用是否合理：
  - 假設每個 object 有個小金庫，確認小金庫在任何情況下都不會透支。
  - 針對此 object 做操作時：
    - 若實際費用比均攤費用低，就存錢到小金庫，供未來實際費用較均攤費用高的操作使用。
    - 若實際費用比均攤費用高，從小金庫裡拿錢補貼。

Accounting method: Approach



1. Each operation is assigned an amortized cost
   - Let c_i and $\widehat{c_i}$ be the actual and amortized costs of the i^{th} op, respectively
2. Validity check: Check if the per-op amortized costs are valid
   - Assume every object initially has credit $= 0$
   - Check if it has sufficient credit (≥ 0) for any sequence of n ops
     - If amortized cost > actual cost ($\widehat{c_i} > c_i$), the difference becomes credit
     - If amortized cost < actual cost ($\widehat{c_i} < c_i$), then withdraw stored credits
   - If the check fails, go back to Step 1
3. Calculate total amortized cost $T(n)$ using individual ones

Accounting method



Validity check: Check if the per-op amortized costs are valid

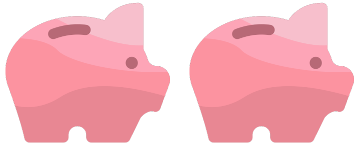
- Assume every object initially has credit $= 0$
- Check if it has sufficient credit (≥ 0) for any sequence of n ops
  - If amortized cost > actual cost ($\hat{c}_i > c_i$), the difference becomes credit
  - If amortized cost < actual cost ($\hat{c}_i < c_i$), then withdraw stored credits

$\Rightarrow$ For an object j, the overall amortized cost is an upper bound of the actual cost:

$$\sum_{op(i,j)=1}^{n} \hat{c}_i \geq \sum_{op(i,j)=1}^{n} c_i$$

- $op(i,j)$ denotes whether the i-th op has effect on the j-th object

Accounting method



- 跟 aggregate method 的主要差別
  - Each type of operations can have a different amortized cost
  - Assign valid per-op amortized costs first and then compute $T(n)$

Accounting method for stack

1. Assign (guess) per-op amortized costs:

Operation	Actual cost	Amortized cost	
`push(S,x)`	1	2 	存 1 元在 x 裡
`pop(S)`	1	0	從 popped 的領 1 元
`multipop(S,k)`	$min\{\lvert S\rvert, k\}$	0	從每個 popped 的領 1 元

Accounting method for stack

1. Assign (guess) per-op amortized costs:

Operation	Actual cost	Amortized cost	
`push(S,x)`	1	2 	存 1 元在 x 裡
`pop(S)`	1	0	從 popped 的領 1 元
`multipop(S,k)`	$min\{\lvert S\rvert, k\}$	0	從每個 popped 的領 1 元

2. Show that for an object j, $\sum_{op(i,j)=1}^{n} \hat{c}_i \geq \sum_{op(i,j)=1}^{n} c_i$
   - **push**: the pushed object is deposited $1 credit
   - **pop and multipop**: use the credit stored with the popped element
   - There is always enough credit to pay for each operation

Accounting method for stack

1. Assign (guess) per-op amortized costs:

Operation	Actual cost	Amortized cost	
`push(S,x)`	1	2 	存 1 元在 x 裡
`pop(S)`	1	0	從 popped 的領 1 元
`multipop(S,k)`	$min\{\|S\|, k\}$	0	從每個 popped 的領 1 元

2. Show that for an object j, $\sum_{op(i,j)=1}^{n} \hat{c_i} \geq \sum_{op(i,j)=1}^{n} c_i$
   - **push**: the pushed object is deposited \$1 credit
   - **pop and multipop**: use the credit stored with the popped element
   - There is always enough credit to pay for each operation
3. Per-op amortized costs are all $O(1)$, total amortized cost is $T(n) = O(n)$

Accounting method for k-bit counter

1. Assign (guess) per-op amortized costs:

Operation	Actual cost	Amortized cost
`INCRENET`	# of bits flipped	\$2 for setting a bit to 1
`bit 0 -> 1`	1	\$2 存1元在bit 1裡
`bit 1 -> 0`	1	\$0 用掉存在bit 1的1元

Accounting method for k-bit counter

1. Assign (guess) per-op amortized costs:

Operation	Actual cost	Amortized cost
`INCRENET`	# of bits flipped	$2 for setting a bit to 1
`bit 0 -> 1`	1	$2 存1元在bit 1裡
`bit 1 -> 0`	1	$0 用掉存在bit 1的1元

2. Validity check:
   - 每次 `INCRENET` 都會把一個 0 設成 1，可能把很多 1 設成 0
   - 可在第一個為 1 的 bit 存一元
   - 把 1 設成 0 時，花掉存在這個 bit 的一元即可

Accounting method for k-bit counter

1. Assign (guess) per-op amortized costs:

Operation	Actual cost	Amortized cost	
`INCRENET`	# of bits flipped	\$2 for setting a bit to 1	
`bit 0 -> 1`	1	\$2	存1元在bit 1裡
`bit 1 -> 0`	1	\$0	用掉存在bit 1的1元

2. Validity check:
   - 每次 `INCRENET` 都會把一個 0 設成 1，可能把很多 1 設成 0
   - 可在第一個為 1 的 bit 存一元
   - 把 1 設成 0 時，花掉存在這個 bit 的一元即可
3. Per-op amortized cost is $O(1)$ => total amortized cost $T(n) = O(n)$

Accounting method for k-bit counter

Counter value	A[3]	A[2]	A[1]	A[0]	Total cost	Total amortized cost
0	0	0	0	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		

Per-op amortized cost is $O(1)$ => total amortized cost $T(n) = O(n)$

Accounting method for k-bit counter

Counter value	A[3]	A[2]	A[1]	A[0]	Total cost	Total amortized cost
0	0	0	0	0	0	0
1	0	0	0	1	1	2
2	0	0	1	0	3	4
3	0	0	1	1	4	6
4	0	1	0	0	7	8
5	0	1	0	1	8	10
6	0	1	1	0	10	12
7	0	1	1	1	11	14
8	1	0	0	0	15	16

Per-op amortized cost is $O(1)$ => total amortized cost $T(n) = O(n)$

Potential Method (位能方法)

Chapter 17.3

Potential method: idea

- 選一個位能函數，將資料結構的狀態對應到一個位能值
  - 對資料結構的操作會改變其狀態，進而改變位能值
- 若位能函數合理，則算出個別操作的均攤費用，進而算出總均攤費用
  - Q: 要怎麼知道位能函數是合理的？
- 檢查位能函數是否合理：
  - 假設資料結構本身有位能值，此位能值在任何情況都需 ≥ 0
  - 使用花費較低的操作時先儲存位能，供未來花費較高的操作使用
- 跟 accounting method 的差異：資料結構本身有 credit，而不是每個 object 都有 credit

Potential method: Approach

1. Select a potential function Φ that takes the current data structure state as input and outputs a potential level
    1. Let D_i be the state of data structure after i^{th} operation
2. Validity check: check if the potential level is never lower than the initial value after any sequence of n operations
    - WLOG, check if $\Phi(D_0) = 0; \forall\, i = 1 \ldots n, \Phi(D_i) \geq 0$
    - If the check fails, go back to Step 1
3. Calculate the per-op amortized cost based on the potential function
4. Calculate total amortized cost based on individual ones

Potential function

- Potential function Φ maps a data structure state to a real number
  - D_0 is the initial state of data structure
  - D_i is the state of data structure after i^{th} operation
  - c_i is the actual cost of i^{th} operation
  - $\widehat{c_i}$ is the amortized cost of i^{th} operation, defined as $\widehat{c_i} = c_i + \Phi(D_i) - \Phi(D_{i-1})$
- Based on the definition, we have

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

Potential function

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

- $\sum_{i=1}^{n} \widehat{c_i}$ is the amortized cost
- $\sum_{i=1}^{n} c_i$ is the actual cost
- If $\Phi(D_n) \geq \Phi(D_0)$, we ensure that $\sum_{i=1}^{n} \widehat{c_i} \geq \sum_{i=1}^{n} c_i$
  - That is, the amortized cost is an upper bound on the actual cost
- Hence, the validity check needs to verify the potential function satisfies
  - $\Phi(D_n) \geq \Phi(D_0)$
  - WLOG, we check $\Phi(D_0) = 0; \forall\, i = 1 \ldots n, \Phi(D_i) \geq 0$

Potential method for stack

1. Define $\Phi(D_i)$ to be # of objects in the stack after the i-th op
2. Validity check:
   - $\Phi(D_0) = 0$, because stack is initially empty
   - $\Phi(D_i) \geq 0$, because # of objects in stack is always ≥ 0

$\Phi(D\_i)$: the # of objects in the stack after the $\mathbf{i}$-th op
c_i: the actual cost of the i-th op
$\hat{c}_i$: the amortized cost of the i-th op

3. Compute per-op amortized cost:
   - For `push(S,x)`: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1 + (|S| + 1) - |S| = 2$
   - For `pop(S)`: $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1 + (|S| - 1) - |S| = 0$
   - For `multipop(S,k)`: $\hat{c}_i = 0$

4. All operations have $O(1)$ amortized cost, so total cost of n operations is $O(n)$

Q: justify why $\hat{c}_i = 0$ for `multipop(S,k)`

Potential method for k-bit counter

1. Define $\Phi(D_i)$ to be # of 1's in the counter after the i-th op
2. Validity check:
   - $\Phi(D_0) = \mathbf{0}$, because counter is initially all 0's
   - $\Phi(D_i) \geq \mathbf{0}$, because # of 1's cannot be negative

$\Phi(D\_i)$: the # of 1's in the counter after the i-th op
c_i: the actual cost of the i-th op
$\widehat{c_i}$: the amortized cost of the i-th op

3. Compute the amortized cost of `INCREMENT`:
   - Let $LSB_0(i)$ be the index of the least significant 0 bit of i
   - For example, $LSB_0(01011011) = 2$, and $LSB_0(01011111) = 5$
   - $\widehat{c_i} = c_i + \Phi(D_i) - \Phi(D_{i-1})$
     $= (LSB_0(i-1) + 1) + (\Phi(D_{i-1}) - LSB_0(i-1) + 1) - \Phi(D_{i-1}) = 2$

4. All operations have $O(1)$ amortized cost, so the total cost of n operations is $O(n)$

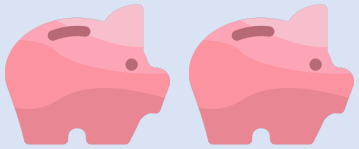
Amortized analysis: 3 common techniques

Aggregate method (聚集方法)

- Determine an upper bound on the cost over any sequence of n operations, $T(n)$
- The average cost per operation is then $T(n)/n$
- All operations have the same amortized cost

Accounting method (記帳方法)

- Each operation is assigned an amortized cost (may differ from the actual cost)
- Each object of the data structure is associated with a credit
- Need to ensure that every object has sufficient credit at any time



Potential method (位能方法)

- Similar to accounting method; each operation is assigned an amortized cost
- The data structure as a whole maintains a credit (i.e., potential)
- Need to ensure that the potential level is nonnegative at any time

* 三種方法一般都能獲得相同的分析結果，可依個人偏好採用