Amortized Analysis

for i = 1 to n do
op_i

Amortized Analysis:

Let $op_1, op_2, \ldots, op_n$ be a sequence of n operations.

In an *amortized analysis*, the average time of each op_i in the worst case is computed.

worst case average

That's $T(n)/n$ is computed, where $T(n)$ is the **worst case** time required for all of the n operations.

total

17.1 *The aggregate method*:

(First, compute $T(n)$. Then, compute $T(n)/n$.)

Example: n stack operations on a stack S

Initially, S is empty.

Each op_i is Push, Pop, or Multi-Pop(S,k).

$\min\{|S|, k\}$

Multi-Pop(S, k)

1. **while** not Stack-Empty(S) and $k > 0$
2. **do** Pop(S)
3. $k \leftarrow k-1$

$\le n-1$

- Push, Pop: $O(1)$; Multi-Pop: $O(\min\{|S|, k\})$.

Analysis (I):

1. The worst-case of an op_i is $O(n)$.
2. The worst-case of all op_i is $T(n)=O(n^2)$.
3. The average cost of each op_i is $T(n)/n=O(n)$.

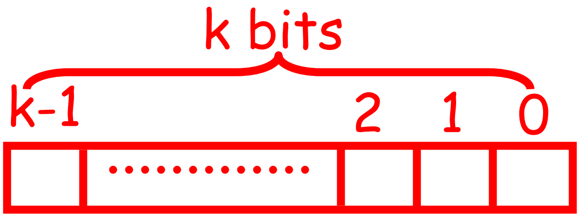
---> correct but not tight.

$\sum pop + \sum t(multipop) \le n$

Analysis (II):

17-2x

1. At most n objects are pushed into S.
2. Each object in S can be poped at most once.
3. The total number of Push's and Pop's is $O(n)$.
4. $T(n)=O(n)$ and the average cost of each op_i is $T(n)/n=O(1)$.



Example: Incrementing a binary counter of k-bit

Initially, all bits in $A[0..k-1]$ are 0.

Each op_i is to increase A by one.

Analysis (I):

1. The worst-case of an op_i takes $O(k)$ time.
2. The worst-case of all op_i is $T(n)=O(nk)$.
3. The average cost of an op_i is $T(n)/n=O(k)$.

17-3x

k = 8

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

17-3y

INCREMENT(A)

```
1s → 0s   i ← 0
          while i < length[A] and A[i] = 1
              do A[i] ← 0            reset to 0
                 i ← i + 1
0 → 1     if i < length[A]           (not overflow)
              then A[i] ← 1          set to 1
```

Each INCR sets at most one bit

17-3z

Analysis (II):

1. $A[0]$ flips each time.
2. $A[1]$ flips every other time.
3. $A[i]$ flips $\lfloor n/2^i \rfloor$ times in total.
4. $T(n) = n + \lfloor n/2 \rfloor + \lfloor n/4 \rfloor + \ldots \le 2n$

5. Average cost of each op_i is $T(n)/n=O(1)$

17-4a
17-4b

Emphasize again:

amortized analysis $\neq$ average-case analysis (機率)

average of each operation in worst case

(expected cost)

(probabilistic cost)

17.2 ***The accounting method***:

1. Assign different charges to different operations. The charge of an operation is called the ***amortized cost*** of the operation.

2. If (amortized cost > actual cost), the value (amortized cost – actual cost) is assigned to specific objects in the data structure as ***credit***.

3. Credit can be used later to pay for operations with (amortized cost < actual cost).

Operation	Amortized cost (付) =	Actual cost (花) +	Total credit (存款)
op_1	40	10	30
op_2	40	20	50
op_3	40	70	20 = C

$$\underbrace{\Sigma a_i}_{付} \underset{\geq}{=} \underbrace{\Sigma t_i}_{花} + C \geq 0$$

4. The credit after each operation should never become negative, which guarantees
   $T(n) \leq$ total amortized cost.

17-5a
17-5b

Example: *n* stack operations on an empty stack *S*

Analysis:

1. Actual costs:

Push:	1	t_i
Pop:	1	
Multi-Pop:	$\min\{\lvert S\rvert, k\}$	

17-5x

2. Assign amortized cost:

Push:	2	a_i ①
Pop:	0	
Multi-Pop:	0	

3. *Push an object* -- we pay 2 付 dollars, 1 花 for the actual cost and 1 存 for credit on the object.
   Pop an object -- we pay by the credit on the object. ②

③ *The total credit on S* -- $|S|$, which is never negative.

④

4. $T(n) \leq$ total amortized cost $\leq 2n$ (at most *n* push). Thus, the average cost of each op_i is $O(1)$. ⑤

Example: Incrementing a binary counter $A[0..k-1]$

Analysis:

1. Actual costs:
   Set a bit to 1: 1
   Reset a bit to 0: 1

2. Assign amortized cost:
   $\left.\begin{array}{ll}\text{Set a bit to 1:} & 2 \\ \text{Reset a bit to 0:} & 0\end{array}\right\} a_i$ ①

17-6x

3. *Set* a bit to 1 -- pay 2 dollars, 1 for the actual cost and 1 for credit on the bit.
   Reset a bit to 0 -- pay by the credit on the bit. ②

   The total credit on A -- the number of 1's in *A*, which is never negative. ③

④

4. $T(n) \leq$ total amortized cost $\leq 2n$ (INCREMENT(*A*) sets at most one bit at Line 6).

⑤ Thus, the average cost of each op_i is $O(1)$.

17-5ab
17-6a

17.3 ***The potential method***:

-- Similar to accounting method. But potential is associated with the *whole data structure* instead of with *specific objects*.

1. Let $D_0 \rightarrow_{op_1} D_1 \rightarrow_{op_2} D_2 \rightarrow \ldots \rightarrow_{op_n} D_n$.

2. Let t_i be the actual cost of op_i.

3. A ***potential function*** Φ (fai) maps each D_i to its potential $\Phi(D_i)$ (a real number).

4. The potential cost a_i of op_i is defined by

付＝花＋ ［存款變化］

$$a_i = t_i + \Phi(D_i) - \Phi(D_{i-1}).$$

$$(\text{or } \Phi(D_i)=\Phi(D_{i-1})+(a_i - t_i))$$

5. Total amortized cost of k operations is

$$\sum_{1\le i\le k} a_i = \sum_{1\le i\le k} \{t_i + \Phi(D_i) - \Phi(D_{i-1})\}$$

$$= \sum_{1\le i\le k} t_i + \Phi(D_k) - \Phi(D_0)$$

≥ 0

6. To guarantee $T(n)=O(\sum_{1\le i\le n} a_i)$, $\Phi(D_k)-\Phi(D_0)$ should never be negative.

   (Usually, we define $\Phi(D_0)=0$ and then prove $\Phi(D_i)\geq 0$ for all i.)

Example: n stack operations on a stack S

push. pop
multi-pop(S, k)

Analysis:

1. Let $\Phi(S)=|S|$.

2. Clearly, we have $\Phi(D_0)=0$ and $\Phi(D_i)\geq 0$ for all i.

Φ is valid!

3. The amortized cost of a Push operation is

付 花 存款變化

$$a_i = t_i + \Phi(D_i) - \Phi(D_{i-1})$$
$$= 1 + (|S|+1) - |S|$$
$$= 2.$$

Δcredit = + 1

The amortized cost of a Pop operation is

$$a_i = 1 + (|S|-1) - |S|$$
$$= 0.$$

Δcredit = - 1

The amortized cost of a Multi-Pop operation is

$$a_i = k' + (|S|-k') - |S| = 0. \quad (k'=\min\{|S|, k\})$$

Δcredit = - k

4. Each a_i is a constant. Thus,

$$T(n) = O\left(\sum_{1\leq i\leq n} a_i\right) = O(n).$$

Example: Incrementing a binary counter $A[0..k\text{-}1]$

Analysis:

1. Let $\Phi(A)$ be the number of 1's in A.

Φ is valid!

2. Clearly, we have $\Phi(D_0)=0$ and $\Phi(D_i)\geq 0$ for all i.

(assume: not overflow)

3. The amortized cost of an increment operation is

$$
\begin{aligned}
a_i &= t_i + \Phi(D_i) - \Phi(D_{i-1}) \\
&= k' + (|A|-(k'-1)+1) - |A| \\
&= 2. \qquad \text{(reset } k'\text{-1 bits to 0)}
\end{aligned}
$$

\# of 1s

Δcredit = -(k'-1)+1 = -k'+2

4. Each a_i is a constant. Thus,

$$T(n) = O\Big(\sum_{1\leq i\leq n} a_i\Big) = O(n).$$

17.4 Dynamic table

- Initially, T is a table of size 0.
- Perform a sequence of n operations on T, each of which is either *Insert* or *Delete*.

17-10a

$$\text{Insert}\begin{cases} 1 \text{ step} & \text{if } T \text{ has a free slot or } |T| = 0 \\ |T| + 1 \text{ steps} & \text{Otherwise} \\ & (1.\ T \text{ is expanded to size } 2|T| \\ & \ 2.\ \text{Copy elements in } T \text{ to a new space} \\ & \ 3.\ \text{Perform Insert}) \end{cases}$$

move

Insert

$$\text{Delete}\begin{cases} 1 \text{ step} & \text{if } T \text{ is at least } 1/4 \text{ full} \\ & \text{after deleting one item} \\ |T|/4 \text{ steps} & \text{Otherwise} \\ & (1.\ \text{Perform Delete.} \\ & \ 2.\ T \text{ is contracted to } |T|/2 \\ & \ 3.\ \text{Copy elements in } T \text{ to a new space}) \end{cases}$$

$\frac{|T|}{4} - 1$ move

delete

Analysis based on potential method:

$= 2(\# - \frac{1}{2}|T|)$

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

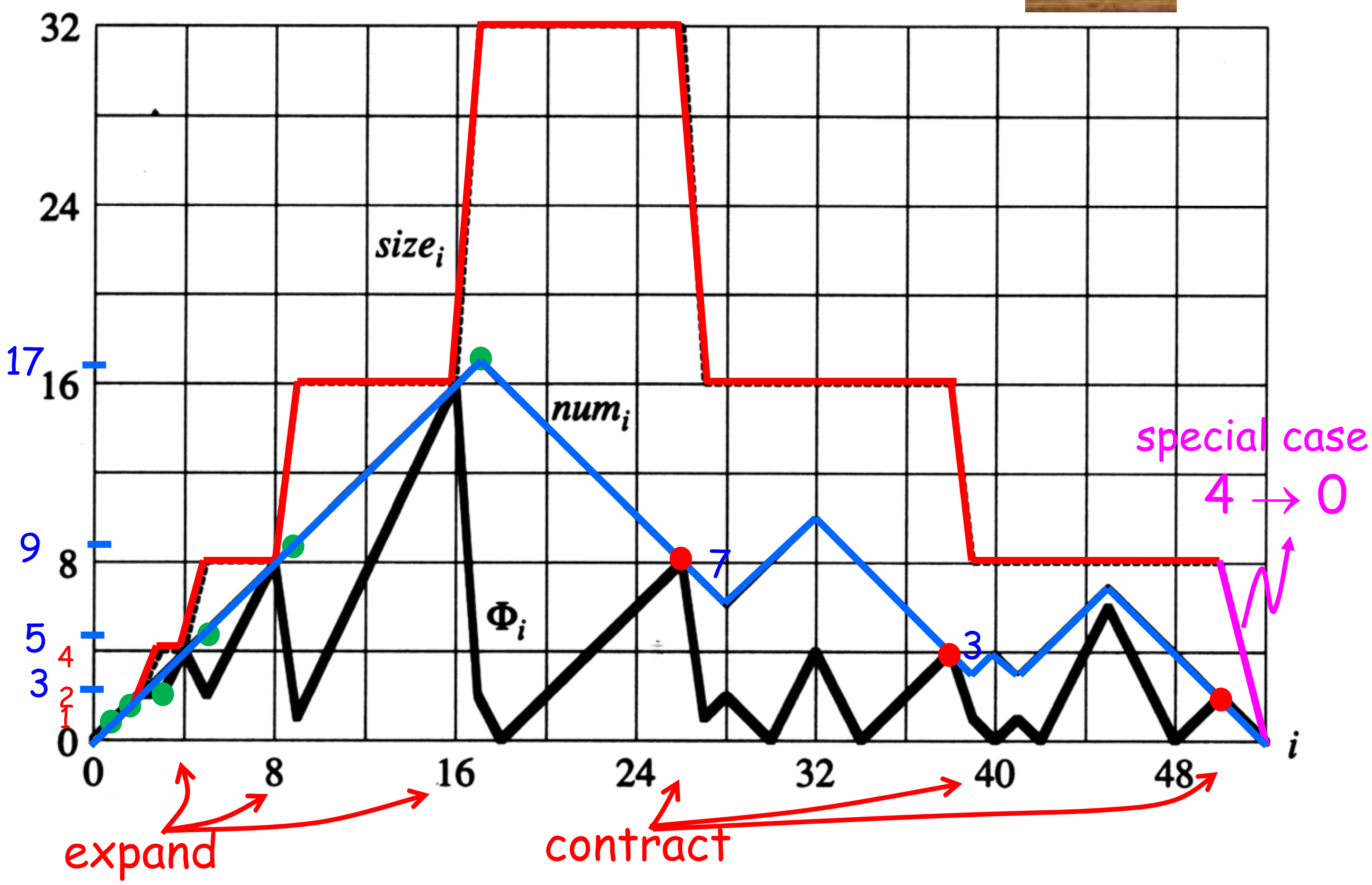
-- *num*[*T*]: the number of items in *T*

-- *size*[*T*]: the size of *T*

-- $\alpha = num[T]/size[T]$: the *load factor* of *T*

(If *num*[*T*]=*size*[*T*]= 0, define α=1.)

$\Phi(T) = 0$



2. Clearly, $\Phi(D_0)=0$ and $\Phi(D_i)\geq 0$ for all i.

Φ is valid!!

$\Phi(D_i) - \Phi(D_0) \geq 0$!

3. The amortized cost of an $\boxed{\text{Insert}}$ operation:

case 1a: $\alpha_{i-1}\geq 1/2$ and T is not expanded

$$a_i = t_i + \Phi(D_i) - \Phi(D_{i-1})$$
$$= 1 + 2$$
$$= 3.$$

$2\# - |T|$

$\# = |T|$

case 1b: $\alpha_{i-1}\geq 1/2$ and T is expanded

$$a_i = (size(D_{i-1})+1) + (2) - (size(D_{i-1}))$$

= 3. (assume size(D_{i-1}) $\neq$ 0; otherwise, 2)

case 2a: $\alpha_{i-1}<1/2$ and $\alpha_i<1/2$

(*T* is not expanded)

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

= 0.

|T|/2 - #

\# = |T|/2 -1

case 2b: $\alpha_{i-1}<1/2$ and $\alpha_i \geq 1/2$

(*T* is not expanded)

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

< 3.

4. The amortized cost of a $\boxed{\text{Delete}}$ operation:

case 1a: $\alpha_{i-1}<1/2$ and *T* is not contracted

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

= 2.

case 1b: $\alpha_{i-1}<1/2$ and *T* is contracted

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

= 1.

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

a_i is also bounded by a constant.

(See Exercise 17.4-2.)

5. Each a_i is bounded by a constant. Thus,

$$T(n) \quad \leq \sum_{1 \leq i \leq n} a_i \leq c \times n = O(n)$$

6. Amortized cost of each op_i is $T(n)/n = O(1)$

as good as
a static table !

17-10b

17-13a

Homework: Ex. 17.1-3, 17.2-2, 17.3-2, 17.4-2, Prob. 17-2. Read Ch18. B-tree

(a), (b)

- Single operation worst-case time t_i
  ⇨ you have right to complain after t_i
- Single operation amortized time a_i
  ⇨ don't complain after a_i
  (just bad luck!)