中国科学技术大学 计算机科学与技术系
University of Science and Technology of China
DEPARTMENT OF COMPUTER SCIENCE AND TECHNOLOGY

算法基础
Foundation of Algorithms

主讲人 徐 云

Fall 2018, USTC

Chapter 17 Amortized Analysis

The slides are from Prof. taoli@FIU, meng@binghamton and yu@PKU in part.

17.1 Background and Methods

- Background
- Amortized analysis
- Three methods

Incrementing a Binary Counter

- k-bit Binary Counter: $A[0..k\text{-}1]$

$$x = \sum_{i=0}^{k-1} A[i] \cdot 2^i$$

INCREMENT(A)
1. $i \leftarrow 0$
2. **while** $i <$ *length*$[A]$ **and** $A[i] = 1$
3. **do** $A[i] \leftarrow 0$ ⊳ *reset a bit*
4. $i \leftarrow i + 1$
5. **if** $i <$ *length*$[A]$
6. **then** $A[i] \leftarrow 1$ ⊳ *set a bit*

k-bit Binary Counter

Value	A[4]	A[3]	A[2]	A[1]	A[0]	*Cost*
0	0	0	0	0	0	***0***
1	0	0	0	0	1	***1***
2	0	0	0	1	0	***3***
3	0	0	0	1	1	***4***
4	0	0	1	0	0	***7***
5	0	0	1	0	1	***8***
6	0	0	1	1	0	***10***
7	0	0	1	1	1	***11***
8	0	1	0	0	0	***15***
9	0	1	0	0	1	***16***
10	0	1	0	1	0	***18***
11	0	1	0	1	1	***19***

Worst-case analysis

Consider a sequence of n insertions. The worst-case time to execute one insertion is $\Theta(k)$. Therefore, the worst-case time for n insertions is $n \cdot \Theta(k) = \Theta(n \cdot k)$.

WRONG! In fact, the worst-case cost for n insertions is only $\Theta(n) \ll \Theta(n \cdot k)$.

Let's see why.

Note: You'll be correct
If you'd said $O(n \cdot k)$.
But, it's an overestimate.

Tighter analysis

value	A[4]	A[3]	A[2]	A[1]	A[0]	*Cost*
0	0	0	0	0	**0**	***0***
1	0	0	0	**0**	**1**	***1***
2	0	0	0	1	**0**	***3***
3	0	0	**0**	**1**	**1**	***4***
4	0	0	1	0	**0**	***7***
5	0	0	1	**0**	**1**	***8***
6	0	0	1	1	**0**	***10***
7	0	**0**	**1**	**1**	**1**	***11***
8	0	1	0	0	**0**	***15***
9	0	1	0	**0**	**1**	***16***
10	0	1	0	1	**0**	***18***
11	0	1	**0**	**1**	**1**	***19***

Total cost of *n* operations

A[0] flipped every op n

A[1] flipped every 2 ops $n/2$

A[2] flipped every 4 ops $n/2^2$

A[3] flipped every 8 ops $n/2^3$

… … … … …

A[i] flipped every 2^i ops $n/2^i$

Tighter analysis (cont.)

Cost of n increments $= \sum_{i=1}^{\lfloor \lg n \rfloor} \left\lfloor \frac{n}{2^i} \right\rfloor$

$$< n \sum_{i=1}^{\infty} \frac{1}{2^i} = 2n$$

$$= \Theta(n).$$

Thus, the average cost of each increment operation is $\Theta(n)/n = \Theta(1)$.

Amortized Analysis (平摊分析)

- *Amortized analysis* is a cost analysis technique, which computes *the average cost* required to perform *a sequence of n operations on a data structure*.
- *Background:* Show that although some individual operations may be expensive, on average the cost per operation is small. Often *worst case analysis* is *not tight*.
- *Goal:* The amortized cost of an operation is less than its worst case, so that *average cost in the worst case* for a sequence of n operations is more *tighter*.
- *This average cost* is not based on averaging over a distribution of inputs. Here, *no probability is involved*.

Three Methods

- *Aggregate analysis (聚集分析)* – in worst case, the total amount of time needed for the n operations is computed and divided by n.
- *Accounting (记账方法)*– different operations are assigned different charges. Some operations charged more or less than their actual cost.
- *Potential (使能方法)*– the prepaid work is represented as "potential" energy that can be released to pay for future operations.

Chapter 17 Amortized Analysis

17.1 Background and Methods

17.2 Aggregate Analysis

17.3 Accounting Method

17.4 Potential Method

17.2 Aggregate Analysis (聚集分析)

- Basic idea
- Stack example
- Binary counter example

Basic Idea of Aggregate Analysis

- Assume that *n operations together* take worst-case time T(n).
- The *amortized cost* (or average cost) of an operation is T(n)/n.
- Remark
  - *Amortized cost is the same* for any operations, even for several types of operations.
  - Amortized cost *may be more or less than the actual cost* for an operations.

Example 1: A Stack

- Three operations:
  - push(S, x)
  - pop(S)
  - multipop(S, k) : Pop the stack k times
- Multipop operation

```
MULTIPOP(S, k)
while not STACK-EMPTY(S) and k ≠ 0
        do POP(S)
           k ← k − 1
```

The total cost of MULTIPOP(S, k) is $\min(s, k)$.
The worst-case cost of a MULTIPOP is $O(n)$.

Stack: Regular Cost Analysis

- Consider a sequence of n push(S, x), pop(S) and multipop(S, k) operations on a stack having as many as n items (元素).

- Regualr anlysis:
  - Note that worst-case cost of multipop() is O(n).
  - So, the worst-case cost for n-ops is $O(n^2)$.
  - This is *not tight*.

Stack: Aggregate Analysis

- For a stack is initially empty, Consider a n-sequence of push(), pop() and multipop().

- Aggregate analysis:
  - Each item (元素) can be popped only once for each time it is pushed.
  - So the total number of times pop() can be called, either directly or from multipop, is bounded by the number of pushes.
  - The number of pushes in a sequence of n ops is $\le n$, then the number of all pops (including those from multipop) is $O(n)$.
  - So the total cost of the sequence of n ops is $O(n)$. Therefore, we have $O(1)$ per op on average.

Example 2: A Binary Counter

- A k-bit binary counter A[0..k-1] of bits, where A[0] is the least bit and A[k-1] is the most bit.
  - Value of the counter is $\sum_{i=0}^{k-1} A[i]\cdot 2^i$
  - Initially, counter value is 0. Then, Counts upward from 0.
- Increment operation, add 1:

- Flip all 1's from right to 0 until encountering the first 0.
- Change this 0 to 1 and stop.

$\textsc{Increment}(A, k)$

$i = 0$
while $i < k$ and $A[i] == 1$
    $A[i] = 0$
    $i = i + 1$
if $i < k$
    $A[i] = 1$

Actual Cost and Regular analysis

- It shows a 8-bit binary counter as its value goes from 0 to 16 by a sequence of 16 Increment operations.
- The average cost per operation is 31/16 < 2.
- However, regular analysis gets O(nk) in the worst case (see 17.1)

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

Binary Counter: Aggregate Analysis

- Observations about Increment():
  - No all bits are flipped for each call.
  - In general, A[i] flips only every 2^ith time.
- Thus, A[i] flips only $\lfloor n/2^i \rfloor$ times in a sequence of n Increment ops on an initially 0 counter.
- So the total number of flips is:

$$T(n) = \sum_{i=0}^{k-1} \left\lfloor \frac{n}{2^i} \right\rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n.$$

- We have T(n) = O(n). And the amortized cost per operation is O(n)/n = O(1).

Chapter 17 Amortized Analysis

17.3 Accounting Method

- Basic idea
- Stack example
- Binary counter example

Basic Idea of Accounting Method

- *Assign different charges to different operations.*
  - Some are charged more than actual cost.
  - Some are charged less than actual cost.
- *Amortized cost = amount we charge.*

- Remark:
  - When amortized cost > actual cost, store the difference on *specific items* in the data structure as *credit* (存款).
  - *Use credit later to pay* for operations whose actual cost > amortized cost.

Credit Rules

- Need credit to *never go negative*.
- Let c_i = actual cost of i-th operation,
  $\hat{c}_i$ = amortized cost of i-th operation.
- For all sequences of n operations, **require**:

$$\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$$

- *Total credit stored* = $\sum_{i=1}^{n} \hat{c}_i - \sum_{i=1}^{n} c_i$

Example 1: A Stack

Operation	Actual Cost	Amortized Cost
push	1	2
pop	1	0
multipop	min{n, k}	0

- When pushing an item, pay $2:
  - $1 pays for the push.
  - $1 is prepayment for it being popped by either pop or multipop.
  - Since each item on the stack has $1 credit, the credit can never go negative.
  - The total amortized cost in the worst case is: $2n \in O(n)$
  - It is an upper bound on total actual cost.

Example 2: 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.

- Amortized cost of Increment:
  - Cost of resetting bits to 0 is paid by credit.
  - At most 1 bit is set to 1 in each increment operation.
  - Therefore, amortized cost $\leq$ \$2.
  - For n operations, the total amortized cost in the worst case is $2n \in O(n)$. So, amortized cost for an op is $O(1)$.

Chapter 17 Amortized Analysis

17.1 Background and Methods

17.2 Aggregate Analysis

17.3 Accounting Method

17.4 <u>Potential Method</u>

17.4 Potential Method

- Basic idea
- Stack example
- Binary counter example

Basic Idea of Potential Method

- Idea: like the accounting method, but think of the credit as *potential stored with the entire data structure*.
  - *Accounting method* stores credit with specific items.
  - Can release potential to pay for future operations.
- It is the most flexible among the amortized analysis methods.

Understanding Potential (1)

- Framework:
  - Start with an initial data structure D_0.
  - Operation i transforms D_{i-1} to D_i.
  - The cost of operation i is c_i.
  - Define a *potential function* Φ: $\{D_i\} \to \mathrm{R}$, such that $\Phi(D_0) = 0$ and $\Phi(D_i) \geq 0$ for all i
  - The *amortized cost* $\hat{c}_i$ with respect to Φ is defined to be $\hat{c}_i = c_i + \underbrace{\Phi(D_i) - \Phi(D_{i-1})}_{\textit{potential difference } \Delta\Phi_i}$
- In practice, $\Phi(\mathrm{D}_0) = 0$, $\Phi(\mathrm{D}_i) \geq 0$ for all i. So,
  - the *amortized cost* is always *an upper bound on actual cost*.
  - work is stored *in the entire data structure* for later use.

Understanding Potential（2）

- The total amortized cost of n operations is

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

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

$$\geq \sum_{i=1}^{n} c_i$$

since $\Phi(D_n) \geq 0$ and $\Phi(D_0) = 0$.

Example 1: A Stack

- Define potential function Φ on a stack = number of items on the stack.
- Let D_0 = empty, then $\Phi(D_0) = 0$.
- Since the number of items on a stack is always ≥ 0, $\Phi(D_i) \geq \Phi(D_0) = 0$.

operation	actual cost	$\Delta\Phi$	amortized cost
PUSH	1	$(s+1) - s = 1$ where $s = \#$ of objects initially	$1 + 1 = 2$
POP	1	$(s-1) - s = -1$	$1 - 1 = 0$
MULTIPOP	$k' = \min(k, s)$	$(s - k') - s = -k'$	$k' - k' = 0$

- So, the total amortized cost of a sequence of n operations in the worst case is $2n = O(n)$.

Example 1: A Stack (cont.)

operation	actual cost	$\Delta\Phi$	amortized cost
PUSH	1	$(s+1)-s=1$ where s = # of objects initially	$1+1=2$
POP	1	$(s-1)-s=-1$	$1-1=0$
MULTIPOP	$k'=\min(k,s)$	$(s-k')-s=-k'$	$k'-k'=0$

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

$= 1 + j - (j-1)$

$= 2$

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

$= 1 + (j-1) - j$

$= 0$

Multi-pop: $\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1})$

$= k' + (j-k') - j$ $\quad k'=\min(|S|,k)$

$= 0$

Example 2: A Binary Counter

$\Phi = b_i = \#$ of 1's after ith INCREMENT

Suppose ith operation resets t_i bits to 0.

$c_i \le t_i + 1$ (resets t_i bits, sets ≤ 1 bit to 1)

- If $b_i = 0$, the ith operation reset all k bits and didn't set one, so $b_{i-1} = t_i = k \Rightarrow b_i = b_{i-1} - t_i$.
- If $b_i > 0$, the ith operation reset t_i bits, set one, so $b_i = b_{i-1} - t_i + 1$.
- Either way, $b_i \le b_{i-1} - t_i + 1$.
- Therefore,

$$\begin{aligned}\Delta\Phi(D_i) &\le (b_{i-1} - t_i + 1) - b_{i-1} \\ &= 1 - t_i\,.\end{aligned}$$

$$\begin{aligned}\hat{c}_i &= c_i + \Delta\Phi(D_i) \\ &\le (t_i + 1) + (1 - t_i) \\ &= 2\,.\end{aligned}$$

If counter starts at 0, $\Phi(D_0) = 0$.

Therefore, amortized cost of n operations $= O(n)$.

Example 2: A Binary Counter (cont.)

General Case

The potential method gives us an easy way to analyze the counter even when it does not start at 0. There are initially b_0 1's and after n INCREMENT operations there are b_n 1's.

$$\begin{aligned}\sum_{i=1}^{n} c_i &\le \sum_{i=1}^{n} 2 - b_n + b_0 \\ &= 2n - b_n + b_0\end{aligned}$$

No matter what initial value the counter contains, the actual cost has an upper bound of $O(n)$.

End of Ch17