Introduction to Algorithms

Topic 6-3 : Amortized Analysis

Xiang-Yang Li and Haisheng Tan

School of Computer Science and Technology
University of Science and Technology of China (USTC)

Fall Semester 2024

Outline

Aggregate Analysis

Accounting Method

Potential Method

Overview

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 the 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.

Here, this average cost is not based on averaging over a distribution of inputs. Here, no probability is involved.

Three Methods

- Aggregate analysis: in the worst case, the total amount of time needed for the n operations is computed and divided by n.
- Accounting: different operations are assigned different charges. Some operations charged more or less than their actual cost.
- Potential: the prepaid work is represented as "potential" energy that can be released to pay for future operations.

Contents

Aggregate Analysis
- Basic idea
- Stack example
- Binary counter example

Accounting Method

Potential Method

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 operation.

Stack example

We have learned two fundamental stack operations, each of which takes O(1) time.

- PUSH(S,x): pushes object x onto stack S.
- POP(S): pops the top of stack S and returns the popped object.

Considering the cost of each operation above to be 1, the total cost of a sequence of n PUSH and POP operations is therefore n,and the actual running time for n operations is therefore θ(n).

A new stack operation MULTIPOP(S,k),it removes the k top objects of stack S, or pops the entire stack if it contains fewer than k objects.

Stack example

The pseudocode of MULTIPOP(S, k) is as follows. In the pseudocode, the operation STACK -EMPTY returns TRUE if there are no objects currently on the stack, and FALSE otherwise.

MULTIPOP(S, k)

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

Stack example

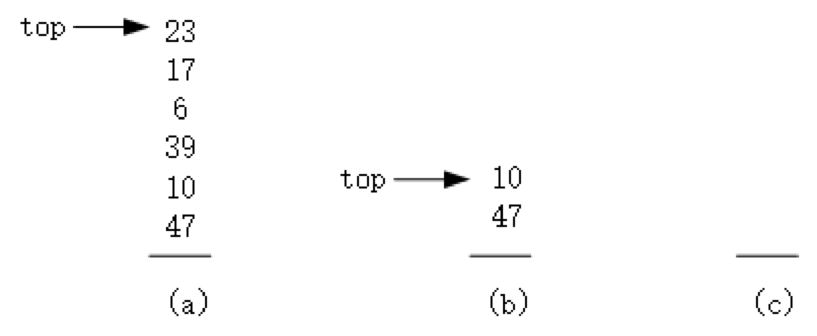

Figure: The action of MULTIPOP on a stack S, shown initially in (a). The top 4 objects are popped by MULTIPOP(S, 4), whose result is shown in (b). The next operation is MULTIPOP(S, 7),which empties the stack—shown in (c)—since there were fewer than 7 objects remaining.

Stack example

Time complexity

The actual running time is linear in the number of POP operations actually executed.

The number of iterations of the while loop is the number of objects popped off the stack (i.e. min(S, k)).

Thus, the total cost of MULTIPOP is min(S, k),and the actual running time is a linear function of this cost.

Stack example

Aggregate Analysis of Stack Operation

Given a sequence of n PUSH, POP, and MULTIPOP operations on an initially empty stack.

Each MULTIPOP operation costs O(n) and we may have O(n) such operations, hence a sequence of n operations costs $O(n^2)$.

Although this analysis is correct, the $O(n^2)$ result is not tight.

Using aggregate analysis, we can obtain a better upper bound that considers the entire sequence of n operations.

Stack example

Amortized Cost of Stack Operations

- Each object can be popped at most once for each time it is pushed.
- So the number of times that POP and MULTIPOP can be called is at most the number of PUSH operations, which is at most n.
- Thus, for any value of n, any sequence of n PUSH, POP, and MULTIPOP operations takes a total of O(n) time, then the average cost of an operation is $O(n)/n = O(1)$.
- In aggregate analysis, we assign the amortized cost of each operation to be the average cost. In this example, therefore, all three stack operations have an amortized cost of $O(1)$.

Binary counter example

- The problem of implementing a k-bit binary counter that counts upward from 0 is another example of aggregate analysis.
- The counter is an array $A[0,..,k-1]$ of bits, where $A.length = k$.
- A binary number x is stored in the counter. $A[0]$ is the lowest-order bit and $A[k-1]$ is its highest-order bit, so that $x = \sum_{i=0}^{k-1} A[i] \cdot 2^i$
- Initially, $x = 0$, and thus $A[i] = 0$ for $i = 0, 1, .., k-1$.

Binary counter example

- The problem of implementing a k-bit binary counter that counts upward from 0 is another example of aggregate analysis.
- The counter is an array $A[0,..,k-1]$ of bits, where $A.length = k$.
- A binary number x is stored in the counter. $A[0]$ is the lowest-order bit and $A[k-1]$ is its highest-order bit, so that $x = \sum_{i=0}^{k-1} A[i] \cdot 2^i$
- Initially, $x = 0$, and thus $A[i] = 0$ for $i = 0, 1, .., k-1$.

To add 1 (modulo 2^k) to the value in the counter, we use the following procedure.

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

Binary counter example

The following figure shows what happens to an $8-$bit binary counter as it is incremented 16 times,starting with the initial value 0 and ending with the value 16. Bits that flip to achieve the next value are shaded. (Note: Setting a bit is $0 \rightarrow 1$; Resetting a bit is $1 \rightarrow 0$.)

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 example

Time Complexity

- A single execution of INCREMENT takes time $\theta(\mathrm{k})$ in the worst case.
- Thus, a sequence of n INCREMENT operations on an initially zero counter takes time O(nk) in the worst case.
- This cursory analysis yields a bound that is correct but not tight.We can tighten it as follows:
  We observe that not all bits flip each time INCREMENT is called. In general, bit A[i] flips $\lfloor n/2^i \rfloor$ times in a sequence of n INCREMENT operations on an initially zero counter, for $i = 0, 1, .., \lceil \lg n \rceil$. For $i > \lceil \lg n \rceil$, bit A[i] never flips at all.

Binary counter example

Time Complexity

- Thus, the total number of flips in the sequence is:

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

- Therefore, the worst-case time for a sequence of n INCREMENT operations on an initially zero counter is $O(n)$.The average cost of each operation, i.e. the amortized cost per operation, is $O(n)/n = O(1)$.

Contents

Aggregate Analysis

Accounting Method
- Basic idea
- Stack example
- Binary counter example

Potential Method

Basic idea

The Accounting Method

- Accounting method: It is a method of amortized analysis and it assigns differing charges to different operations.The amount we charge an operation is called its amortized cost.
- When an operation's amortized cost exceeds its actual cost, the difference is called credit.
- Credit can be used later on to help pay for operations whose amortized cost is less than their actual cost.

Basic idea

The Accounting Method

- We must choose the amortized costs of operations carefully to make that:

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

for all sequences of n operations,wherein c_i is the actual cost of the i th operation,$\hat{c}_i$ is the amortized cost of the i th operation.

- By doing so, we guarantee that the total amortized cost of a sequence of operations must be an upper bound on the total actual cost of the sequence. Thus, we must take care that the total credit in the data structure never becomes negative.

Stack example

Let us assign the following amortized costs:

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.
- Due to at most n pushes, the total amortized cost in the worst case is: $2n \in O(n)$.
- It is an upper bound on total actual cost.

Binary counter example

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 each increment.
- For n operations, the total amortized cost in the worst case is $2n \in O(n)$. So, amortized cost for an op is $O(1)$.

Contents

Aggregate Analysis

Accounting Method

Potential Method
- Basic idea
- Stack example
- Binary counter example

Basic idea

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.
- Potential method can release potential to pay for future operations.

It is the most flexible among the amortized analysis methods.

Basic idea

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:$\{ D_i\} \to R$,such that $\Phi(D_0) = 0$ and $\Phi(D_i) \geq 0$ for all i.
- The amortized cost $\widehat{c_i}$ with respect to Φ is defined to be:

$$\widehat{c_i} = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

  and $\Phi(D_i) - \Phi(D_{i-1})$ is called potential difference $\Delta\Phi_i$

Basic idea

Framework(cont)

- Thus the total amortized cost of the n operations is:

$$\begin{aligned}\sum_{i=1}^{n} \hat{c_i} &= \sum_{i=1}^{n}(c_i + \Phi(D_i) - \Phi(D_{i-1})) \\ &= \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0)\end{aligned}$$

- If existing a potential function Φ so that $\Phi(D_n) \geq \Phi(D_0)$,then the total amortized cost $\sum_{i=1}^{n} \hat{c_i}$ is an upper bound on the total actual cost $\sum_{i=1}^{n} c_i$.
- We can define $\Phi(D_0)$ to be 0 and then to show that $\Phi(D_i) \geq 0$ for all i.

Basic idea

Framework(cont)

- If $\Phi(D_i) - \Phi(D_{i-1})$ is positive, the amortized cost $\widehat{c_i}$ represents an overcharge to the i th operation; otherwise, it represents an undercharge to the i th operation and the actual cost of the operation is paid by the decrease in the potential.
- Different potential functions may yield different amortized costs yet still be upper bounds on the actual costs.
- There are often trade-offs that can be made in choosing a potential function; the best potential function to use depends on the desired time bounds.

Stack example

Stack operations

- The potential function Φ on a stack is defined to be the number of objects in the stack.
- For the initial empty stack D_0 , we have $\Phi(D_0) = 0$.
- Then we have $\Phi(D_i) \geq 0 = \Phi(D_0)$.
- Therefore the total amortized cost of n operations with respect to Φ represents an upper bound on the actual cost.

Stack example

Now compute the amortized costs of the various stack operations

- If the i th operation on a stack containing s objects is a PUSH operation,then the potential difference is:

  $$\Phi(D_i) - \Phi(D_{i-1}) = (s+1) - s = 1$$

  So the amortized cost of this PUSH operation is:

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

- Suppose that the i th operation on the stack is MULTIPOP(S,k) and that $k' = \min(k,s)$ objects are popped off the stack. The actual cost of the operation is k', and the potential difference is:

  $$\Phi(D_i) - \Phi(D_{i-1}) = -k'$$

  Thus the amortized cost of the MULTIPOP operation is:

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

Stack example

- Similarly, the amortized cost of an ordinary POP operation is 0.
- The amortized cost of each of the three operations is O(1), and thus the total amortized cost of a sequence of n operations is O(n).
- Since $\Phi(D_i) \geq \Phi(D_0)$, the total amortized cost of n operations is an upper bound on the total actual cost.
- Therefore the worst-case cost of n operations is O(n).

Binary counter example

Incrementing a binary counter

Let b_i denote the potential of the counter after the i th INCREMENT operation, it is the number of 1's in the counter after the i th operation.

Now compute the amortized cost of an INCREMENT operation.

- Suppose that the i th INCREMENT operation resets t_i bits,the actual cost of the operation is therefore at most $t_i + 1$ (besides resetting t_i bits, we might set one more bit to 1)
- If $b_i = 0$, that is the i th operation resets all k bits (k is the number of bits in the counter), then $b_{i-1} = t_i = k$; If $b_i > 0$, then $b_i = b_{i-1} - t_i + 1$.
- In either case, $b_i \leq b_{i-1} - t_i + 1$.

Binary counter example

- So the potential difference is:

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

- Therefore the amortized cost is:

$$\begin{aligned} \widehat{c_i} &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &\leq (t_i + 1) + (1 - t_i) = 2 \end{aligned}$$

- If the counter starts at zero, then $\Phi(D_0) = 0$.
- Since $\Phi(D_i) \geq 0$ for all i, the total amortized cost of a sequence of n INCREMENT operations is an upper bound on the total actual cost.
- So the worst-case cost of n INCREMENT operations is $O(n)$.

Binary counter example

- When it does not start at zero, there are initially b_0 1's. After n INCREMENT operations there are b_n 1's, where $0 \le b_0, b_n \le k$ (k is the number of bits in the counter).
- So we have:
$$\sum_{i=1}^{n} c_i = \sum_{i=1}^{n} \widehat{c_i} - \Phi(D_n) + \Phi(D_0)$$
- Because we have $\widehat{c_i} \le 2$ for all $1 \le i \le n$; $\Phi(D_0) = b_0$ and $\Phi(D_n) = b_n$, the total actual cost of n INCREMENT operations is:
$$\begin{aligned}\sum_{i=1}^{n} c_i &\le \sum_{i=1}^{n} 2 - b_n + b_0 \\ &= 2n - b_n + b_0\end{aligned}$$
- Since $b_0 \le k$, as long as $k = O(n)$, the total actual cost is $O(n)$.
- Thus, if we execute at least $n = \Omega(k)$ INCREMENT operations,the total actual cost is $O(n)$, no matter what initial value the counter contains.