CS294-1 On-line Computation & Network Algorithms **Spring 1997**

Lecture 2: January 23

Lecturer: Yair Bartal *Scribe: Keith Vanderveen*

This lecture introduces amortized analysis techniques and demonstrates their use in analyzing the cost of two simple data structures problems.

2.1 Amortized Anlysis

Amortized analysis [Tarjan85, CLR90] is orginated in the study of dynamic data structures. Each operation is regarded as starting with some initial amount of "money." If the operation costs less than this amount of money, the excess is stored as credit toward paying the costs of operations which cost more than the initial amount of money.

In Data Structures: Suppose you carry out a sequence of operations $o_1, o_2, ..., o_n$. Then

$$\text{worst-case cost} = \max_i c(o_i).$$

$$\text{total cost} = \sum_i c(o_i).$$

$$\text{amortized cost} = \frac{\text{total cost}}{n} \equiv \tilde{c}$$

For example, in *dynamic hashing* the worst-case search cost is $\Theta(\log n)$ whereas the amortized cost is $O(1)$.

2.2 Amortizatrion Techniques Through Examples

2.2.1 The Binary Counter

Initially, all the bits are set to 0. To increment the counter by 1, one goes right to left flipping bits until a bit is flipped from 0 to 1. The cost of the ith update, c_i, is the number of bits which are flipped.

$$\text{worst-case cost} \ = k.$$

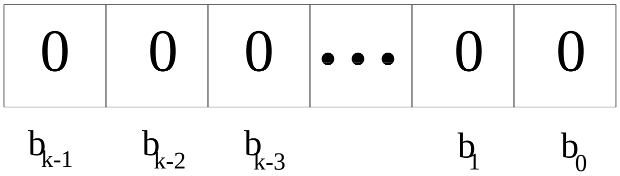

Figure 2.1: k-bit binary counter

$$\begin{aligned}\text{total cost} &= c(n) \\ &= \sum_{i}^{n} c_i = O(nk).\end{aligned}$$

$$\text{amortized cost} \tilde{c} = \frac{c(n)}{n}.$$

Claim 2.1 $\tilde{c} \leq 2$.

Proof: 3 techniques to prove this:

- The Engineer's view
- The Banker's view
- The Physicist's view

2.2.1.1 The Engineer's view

Idea: brute force calculation! Observe:

b_0 flips every update
b_1 flips every 2 updates
$\vdots$
b_i flips every 2^i updates

$$c(n) = \sum_{i=0}^{\lfloor \log n \rfloor} \lfloor \frac{n}{2^i} \rfloor < 2n.$$

$$\tilde{c} < 2.$$

2.2.1.2 The Banker's View: (The Credit-Debit Method)

Idea: Each operation comes in with $\tilde{c}$ \$. During execution, if the cost of an operation is less than $\tilde{c}$, store the excess ($\tilde{c}-$ cost) as credit. If the cost of an operation is greater than $\tilde{c}$, withdraw previously stored credit to complete payment of (cost $-(\tilde{c})$.

Theorem 2.2 *If system has non-negative credit at all times, then $\tilde{c}$ is a bound on the amotized cost.*

Back to the Binary Counter problem, we set $\tilde{c} = 2$ and associate credits with bit positions.

Case 1: We flip a bit into 1, pay 1 unit and store 1 unit at that bit's position.

Case 2: We flip a bit into 0, withdraw 1 unit from credit at that bit's position.

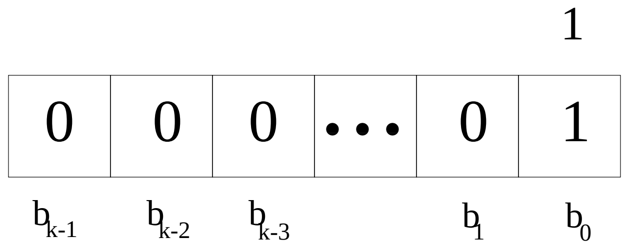

Figure 2.2: after first update: 1 credit on bit 0

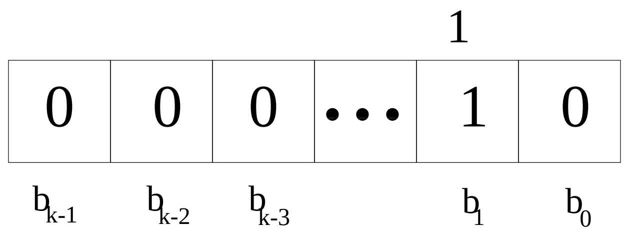

Figure 2.3: after 2nd update: 1 credit on bit 1 (paid to flip bit 0 with credit for that bit)

It can be shown by an induction argument that every time a bit i is flipped to 1, it will have a credit of 1 deposited at its position. This is because all of the bits to its right must have been 1 after the previous update, hence they had credit to pay for their flips, and we got two units of credit for the update, one to flip the bit i and the other to store as credit at i's position.

2.2.1.3 The Physicist's View: (The Potential Function Method)

Idea: Store pre-paid credit as a "potential energy" of the entire system. Use potential to pay for operations which cost more energy than they bring in.

Formally, let Φ_i denote the potential at time i. Clearly,

$$\tilde{c} = c_i + \Delta\Phi,$$

where $\Delta\Phi = \Phi_i - \Phi_{i-1}$, thus assume $\tilde{c}_i = b$, the total amortized cost is

$$\begin{aligned} b * n &= \sum_{i=0}^{n} \tilde{c}_i = \sum_{i=1}^{n} (c_i + \Phi_i - \Phi_{i-1}) \\ &= \sum_{i=1}^{n} c_i + \Phi_n - \Phi_0 \end{aligned}$$

Theorem 2.3 *If $\Phi_n \geq \Phi_0$ (usually $\Phi_0 = 0$), then $\tilde{c}(n) = \frac{c(n)}{n} \leq b$.*

Important: Want $\Phi_i \geq \Phi_0$ for all $i \leq n$.

To prove $\tilde{c}_i$ bounded, we can show that $c_i \leq \tilde{c}_i - \Delta\Phi$.

The potential method is applied to the Binary Counter problem as follows. Let Φ_i = number of bits set to 1 at time i. Let $\Phi_0 = 0$. Then $\forall i, \Phi_i \geq 0$. Consider the ith update. There exists a j such that bit j is flipped into a 1 and all bits less than j are flipped to 0. Therefore,

$$\Phi_i - \Phi_{i-1} = 1 - j.$$

We want to show $\tilde{c} = 2$.

$$c_i = j + 1 = 2 - (1 - j) = \tilde{c}_i - \Delta\Phi$$

■

2.2.2 A Queue From Two Stacks

We may use two stacks A and B and need to implement a queue. The Queue operations include: insert item x ($I(x)$) and delete item x ($D(x)$). An insert places a new item at the tail of the queue, and a delete removes an item from the head of the queue. We are given a sequence of insert and delete operations.

Consider the following algorithm. Newly inserted items are pushed into stack A. When a delete occurs we first check if there are items in stack B. If so we pop the item at the top of stack B. Otherwise all items in stack A are popped off the stack and all except the last one are pushed into stack B.

It is easy to check the correctness of this algorithm. We prove the following:

Claim 2.4 *The 2-Stacks algorithm has amortized cost at most 3.*

Proof: We use the Credit-Debit method. Let each insert operation come with 3 units and each delete operation come with 1 unit. Then inserting an item into stack A takes 1 unit, and popping the item and inserting it into stack 2 takes 2 units. Deleting the item from stack 2 then takes 1 unit. Then the cost (which is the total number of pushes and pops) obeys

$$\text{total cost} \leq 3 * \# \text{ of Inserts} + \# \text{ of Deletes} \leq 3n,$$

where n is the overall number of operations. ■

References

[CLR90] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms.* The MIT Press, 1990.

[Tarjan85] R.E. Tarjan. Amortized Computational Complexity. *SIAM Journal on Algebraic and Discrete Methods*, 6(2):306-318, 1985.