[CSN-531] Lecture 3
Amortization Techniques

Scribe: Madhu

1 Introduction

Amortized analysis is an analysis of a sequence of operations to obtain a tighter bound on the total cost or average cost in comparison to its worst-case time. It is used when operations occasionally take more time and usually take less time. Thus, worst case analysis of operation time is not the best measure.

Definition 1 (Amortized time)**.** *For any k operation where total time is $f(k)$, then amortize time is $f(k)/k$*

We shall now look at some example problems where Amortized analysis proves significant improvement over worst-case analysis.

Example: Binary counter

Consider the binary representation of the numbers. Incrementing by one unit, several bits may change in the binary representation. The goal is to compute the *average number of bits flipped* per increment.
For example, consider a 4-bit counter counting up to 15 (in general counting up to k requires $\log k$ bits)

$$\begin{array}{cccc} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ \cdots & & & \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{array}$$

Worst case: Transition from 0111 to 1111 requires flipping all the bits. In general, worst case is $\log k$.
Best case: Transition from 0000 to 0001 requires to flip only LSB. In general, best case is 1.

In this particular example, if worst case time $\log k$ appears with frequency $1/2$ then total time for t operations would be at least $\Omega(\frac{t}{2}\log k)$, giving amortized time $\Omega(\log k)$. Hence if in the worst-case scenario the highest cost (expensive) operation is rare, we use amortization analysis.

Note: This can be generalized to any constant c, if the expensive operation taking $O(f)$ is not rare, say occurs say $1/c$ of total times, the total time for t operations is
$T(t) = f \times \frac{t}{c} +$ *Cost of cheap operation* $(\leq t) = \Omega(ft)$.
This gives an amortized time $\Omega(f)$, which is the same as the worst-case time.

2 Methods:

Following are the popular approaches to perform amortized analysis.

1. **Aggregate Method.** Looks at the cost from the perspective of overall work done.
   Computes a bound on the total cost for n operations, say $T(n)$, so average cost $\leq T(n)/n$.

2. **Accounting Method.** Looks at the cost from the perspective of an operation.
   Operations are assigned amortized cost, charging extra credit to pay for the future expensive operations.
   Amortized Cost = Actual Cost + Extra Credit

3. **Potential Method.** Looks from the perspective of the state of the system.
   Its stability is estimated for having potential of future expensive operations.
   Credit is computed in form of a potential which is used in heavy operations.
   Amortized Cost = Actual Cost + Change in Potential

   Note: If immediate operations are the least expensive, the state of the system is considered *stable*. And the potential is defined such that for expensive operations the negative change in potential can compensate the extra actual cost, and typically the potential in the stable state is *zero*. One can imagine a water tank storing potential in idle time and using it when required to compensate requirements.

3 Amortized Analysis of Binary Counter

Aggregate method:

Upper bounds total cost of n operations $T(n)$, so Amortized cost $\leq T(n)/n$
$T(n)$ = Number of bit flips after n increments.

	$3^{rd}Bit$	$2^{nd}Bit$	LSB
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
...			

We can see that LSB flips in every increment, $2^{nd}Bit$ flips after 2 increments, and $3^{rd}Bit$ flips after 4 increments, and so on. Hence, total flips required to for n increments can be written as (starting from LSB)
$T(n) = n + n/2 + n/4 + n/8 + \cdots \leq 2n$
Amortized cost $= T(n)/n \leq 2n/n = 2$.

Accounting method:

Each operation is assigned an extra credit to pay for future expensive operation.
Am. Cost = Actual Cost + Extra Credit
Looking at a single increment, we see LSB flips in every increment, $2^{nd}Bit$ flips after 2 increments, and $3^{rd}Bit$ flips after 4 increments, and so on. So for an increment, the operation is charged for each bit costing 1 for LSB, and storing 1/2 credit for future $2^{nd}Bit$ flip, 1/4 for future $3^{rd}Bit$ flip, and so on. Hence every operation only pays for flip of LSB, rest is payed for by the credits accumulated.
Amortized Cost $= 1 + 1/2 + 1/4 + 1/8 + \cdots \leq 2$.

Potential method:

Credit is computed in form of a potential which is used in heavy operations.
Amortized Cost = Actual Cost + Change in Potential

Clearly, the *expensive* operation requiring a lot of bit flips occurs when we have a lot of 1's in current, and the most *stable* state where expensive operations are most distant is the initial state of 0. Thus, a good choice of potential is as follows:

$$\Phi = \text{ Number of bits which are 1}$$

We get the following costs, potential (Φ) and change in potential ($\Delta\Phi$):

State	Φ	$\Delta\Phi$	Work (Flips)	Amortized (Work+$\Delta\Phi$)
00000	0	-	-	-
00001	1	+1	1	2
00010	1	0	2	2
00011	2	+1	1	2
00100	1	-1	3	2
...				
01111	4	-	-	-
10000	1	-3	5	2

Table 1: Potential Method for Amortized Analysis of Binary Counter

4 Amortized analysis of Multi Pop Stack

Multi Pop Stack. Consider a stack that supports the following operations:

1. Push(x): Push an element x on the top of Stack
2. Pop(k): Pop k elements from stack (if exists)

Clearly, *Push* is *cheap* operation taking $O(1)$ time, and *Pop* is *expensive* operation taking $O(k)$ time. Thus,

$$O(1) \leq \text{Amortized Time} \leq O(k)$$

Aggregate method:

Upper bounds total cost of n operations $T(n)$, so Amortized cost $\leq T(n)/n$
$T(n)$ = Cost of n push and pop operations.

The Stack can have some push operations and some pop operations.
Total cost = Cost of all Pushes + Cost of all pops.
Now, assume we have $i \leq n$ push operations among the total n operations. We get
Cost of all Pushes = i.
To bound the number of pops note that each element popped has to be pushed into the stack. Hence,
Cost of all Pops $\leq$ Cost of all Pushes $\leq i$.
Hence, Total cost $T(n)$ = Cost of all Pushes + Cost of all Pops $\leq i + i \leq 2i \leq 2n$.
Hence, Amortized cost $= T(n)/n \leq 2n/n = 2$.

Accounting method:

As mentioned previously , each cheap operation will be assigned an extra cost to compensate the expensive operation. In this particular example cost of push is considered to be cheap and pop is expensive. So we'll assign extra cost to push. i.e. for every element pushed into stack we'll pay for it's future pop operation so that whenever pop is called the cost will be zero as it is already taken care of.
To summarize,

$$\text{Cost assigned to push} = \text{actual cost to push} + \text{extra cost (for future pop)} = 1 + 1 = 2$$

$$\text{Cost assigned to pop} = 0$$

$$\text{Amortized Cost} = \max(\text{Cost of Push, Cost of Pop}) = 2$$

Potential method:

Here we can define potential function as

$$\Phi = \text{number of elements in stack}$$

Each time when we push the potential function will increase and when we pop the stored potential will be used. Here the change in potential is the change in the number of elements, if we have a stack with n elements, after push it'll be $n + 1$, i.e,

$$(\Delta\Phi) = (n + 1) - n = 1$$

Hence amortized cost will be,

$$\text{Amortized cost(PUSH)= Actual Cost (push)+ Change in potential}(\Delta\Phi) = 1 + 1 = 2$$

Similarly, for Pop(k) the initial number of elements will be n, after pop(k) it'll be $n - k$. change in potential $\Delta\Phi$ will be $-k$. Further, the actual cost of pop operation would be k. Hence,

$$\text{Amortized cost(POP)= Actual cost (pop)+ change in potential}(\Delta\Phi) = k + (-k) = 0$$

.

$$\text{Amortized Cost} = \max(\text{Cost of Push, Cost of Pop}) = 2$$

5 Discussion

Question 1. *Amortized Analysis is used when*

1. *Different Operations takes different times*
2. *Same operations take different times*
3. *Expensive operations are rare*
4. *All of the above*

Solution. In the case of the Binary counter same operation was taking different times, whereas in the case of the Multipop stack different operations were taking different times. As discussed in the Note in Section 1, the necessary condition for amortized analysis is the fact that expensive operation is rare. Hence, the correct choice is point 3.

Question 2. *Amortized Analysis can compute*

1. *Best Case Time complexity*
2. *Average Case Time complexity*
3. *Worst Case Time complexity*
4. *All of the above*

Solution. The best case, worst case etc. deal with the input for the problem. Amortized analysis can be used for all. Consider the best, and worst case for the Multi Pop Stack.

Binary Counter

The average number of flips in minimum just before the expensive operation. The best case is having exactly one flip for $n = 1$.

For a k bit number, the best case would be one having the least average flips, which is just before an expensive operation, and the worst case would be just after the expensive operation.

Average bit flips: $\frac{n+\lfloor\frac{n}{2}\rfloor+\lfloor\frac{n}{4}\rfloor+\cdots+\lfloor\frac{n}{2^{k-1}}+\lfloor\frac{n}{2^k}\rfloor}{n}$

Worst Case (flips upto $(10000\cdots0)_2$): $\frac{n+\frac{n}{2}+\frac{n}{4}+\cdots+\frac{n}{2^{k-1}}+\frac{n}{2^k}}{n} = 1+\frac{1}{2}+\frac{1}{4}+\cdots+\frac{1}{2^k} = 2-\frac{1}{2^k} \approx 2.$

Best Case (flips upto $(111111\cdots1)_2$): $\frac{n+\frac{n-1}{2}+\frac{n-3}{4}+\cdots+\frac{n+1-2^{k-1}}{2^{k-1}}}{n} = 1+\frac{1}{2}+\frac{1}{4}+\cdots+\frac{1}{2^{k-1}}-\frac{\frac{2-1}{2}+\frac{4-1}{4}+\cdots+\frac{2^{k-1}-1}{2^{k-1}}}{n}$

$$= 2-\frac{1}{2^{k-1}}-\frac{(k-1)-(1-\frac{1}{2^{k-1}})}{n} \approx 2-\frac{1}{2^{k-1}}-\frac{k-2}{n}$$

For example for a 4 bit number, the best case and worst case are:

Worst Case: $(1000)_2 = 8$: $\frac{8+\lfloor\frac{8}{2}\rfloor+\lfloor\frac{8}{4}\rfloor+\lfloor\frac{8}{8}\rfloor}{8} = \frac{8+4+2+1}{8} = \frac{15}{8}$

Best Case: $(1111)_2 = 15$: $\frac{15+\lfloor\frac{15}{2}\rfloor+\lfloor\frac{15}{4}\rfloor+\lfloor\frac{15}{8}\rfloor}{15} = \frac{15+7+3+1}{15} = \frac{26}{15}$

Multi Pop Stack

Best-case: Alternate push and pop operations.

$$Push|Pop|Push|Pop-------$$

For every push operation, there will be an immediate pop (which can pop only one element). Hence, work done per operation is exactly 1. To summarize,

$$\frac{2\times i}{i+i} = 1$$

Worst-case: All pushes are followed by a single pop operation.

$$Push|Push|----------Push|Pop$$

If we consider k push followed by only one pop, expensive operation is only appearing once taking k time. The over-all work done will increase as follows. To summarize,

$$\frac{2\times k}{k+1} \approx 2$$

Homework Problems

Question 3. *Compute the number of literal changed per incremental in a*

1. *Digit Counter (having literals* $0, \cdots, 9$*)*
2. $b-$*ary Counter (having literals* $0, 1, \cdots, b-1$*)*

Question 4. *Describe an implementation of the operations of a Queue (pop and push by FIFO) using the operations of a Stack (pop and push by LIFO). You can use two stacks to implement the queue, such that each element occurs in exactly one stack and each queue operation takes constant amortized time.*

6 Classical problems

Connectivity:

Connectivity is defined in an undirected Graph. If a vertex is said to be connected to any other vertex it simply implied there exists a path connecting those two vertices. Connectivity is a symmetric relation hence, if x is connected to y then y is also connected to x. All vertices that are connected to each other form the *connected components* of the graph.

Query(x, y): Are x and y connected? or Does there exist a path between x and y in graph G?

Reachability:

Reachability is defined in a directed Graph. A vertex x is said to be reachable from a vertex y if there exists a path from y to x. This is not a symmetric relationship, i.e., x being reachable from y, does not imply y is reachable from x. All there vertices that are reachable from each other (both directions) form a strongly connected component of the directed graph.

Query(x, y): Is y reachable from x? or Does there exist a path from x to y?

Question 5. *Is computing reachability harder than connectivity?*

Solution. The answer is yes, because suppose there exist an algorithm to compute reachability in a directed graph. We can always use this algorithm to compute connectivity of an undirected graph, by making it a directed graph adding edges in both directions. However, the reverse is not true, an algorithm to compute connectivity may not work for computing reachability.

Reference

Amortized Analysis, Chapter 17 of the course text book *Introduction to Algorithms* by
Cormen T, Leiserson C, Rivest R, and Stein C, Fourth Edition, MIT Press.