Chapter 17: Amortized Analysis

Today we study amortized analysis.

What is that, master? Efficiency averaged over time?

That is right, boy.
We assume that a sequence of operations is executed on a data structure and calculate the cost per operation averaged over the sequence.

Does it mean that some operations are cheap and some are expensive depending on the situation?

That's right.
Our first example is Multipop, a new operation on a stack.

With this you are able to pop any number of elements from a stack.

However, it is implemented by repeated execution of Pop.

What are the other permissible operations?

Creation of an empty stack, Push, Pop, and Empty, which tests the emptiness.
Ostensibly Multipop is quite expensive because elimination of $k$ objects requires $O(k)$ steps.

However, for us to be able to eliminate $k$ objects Push has to be executed at least $k$ times prior to that...

Which means a bad thing does not happen so very often...
Our next example is a $k$-bit binary counter.

Suppose we will increment $n$ times a $k$-bit counter that is initially set to 0...

The number of bit operations required is high if there is a long run of 1’s at the lower bits of the counter, but that does not happen very often.

That is right.
Amortized Analysis

Suppose that $n$ operations chosen from Pop, Push, and Multipop are executed on an initially empty stack. The total cost for Multipop is the linear function of the total number of Push, which is at most $n$. So, the amortized cost of Multipop is $O(1)$.

Suppose that a $k$-bit counter initially set to 0 is incremented $n$ times. The total number of bit flips on the counter is

$$\sum_{i=0}^{\lfloor \log n \rfloor} \left\lfloor \frac{n}{2^i} \right\rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n.$$  

So the amortized cost is $O(1)$.

This calculation method is called aggregate method.
The potential method

**Policy:** For each \( i, 1 \leq i \leq n \), let \( c_i \) be the actual cost of the \( i \)-th operation and \( D_i \) be the data structure when the \( i \)-th operation has been done.

Pick a **potential function** \( \Phi \) that assigns a value to the data structure and define the amortized cost \( \hat{c}_i \) as \( c_i + \Phi(D_i) - \Phi(D_{i-1}) \). Let \( T(n) = \sum_{i=1}^{n} \hat{c}_i \) be the total amortized cost. Then

\[
T(n) = \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1}))
\]

\[
= \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0).
\]
We’ll pick $\Phi$ so that

- for all $i$ $\Phi(D_i) \geq \Phi(D_0)$ and
- $\sum_{i=1}^{n} \hat{c}_i$ is easy to compute.

Then $T(n)/n$ gives an upper-bound for the amortized cost. So, we will evaluate $\hat{c}_i$ instead of $c_i$. 
A. Stack: Define $\Phi(D_i)$ to be the stack size. Then $\Phi(D_0) = 0$, and so, for all $i \geq 1$, $\Phi(D_i) \geq \Phi(D_0)$.

The amortized cost $\hat{c}_i$ is $1 + 1 = 2$ for Push and 0 for both Pop and Multipop.
**B. Counter:** Define $\Phi(D_i)$ to be the number of bits 1 in the counter after the $i$-th incrementation. Then $\Phi(D_0) = 0$ and for all $i$, $i \geq 0$, $\Phi(D_i) \geq 0$.

Define $t_i$ to be the number of bits that are reset at the $i$-th operation. Then for all $i \geq 0$, $\Phi(D_{i+1}) \leq \Phi(D_i) - t_i + 1$. Then $c_i = t_i + 1$ and $\hat{c}_i \leq t_i + 1 + (1 - t_i) = 2$. So, the amortized cost is $O(1)$. 
Dynamic Tables

A dynamic table is a table of variable size, where an expansion (or a contraction is caused when the load factor has become larger (or smaller) than a fixed threshold.

Let the expansion threshold be 1 and the expansion rate be 2; i.e., the table size is doubled when an item is to be inserted when the table is full.

Let the contraction threshold be $\frac{1}{4}$ and the contraction rate be $\frac{1}{2}$; i.e., the table size is halved when an item is to be eliminated when the table is exactly one-fourth full.
Implementation of Expansion & Contraction

When these operations take place we create a new table and move all the elements from the old one to the new one.

Suppose that there are $n$ calls of insertion and deletion are made, what is the average cost of each operation?
If the size is kept the same the cost is $O(1)$.

If the size is doubled from $M$ to $2M$, the actual cost is $M + 1$. The time that it takes for the next table size change to occur is \textbf{at least $M$ steps for doubling} and \textbf{at least $M/2$ steps for halving}. So the actual cost can be spread over the next $M/2$ “normal” steps. This gives an amortized cost of $O(1)$.

If the size is halved from $M$ to $M/2$, the actual cost is $M/4$. The time that it takes for the next table size change to occur is \textbf{at least $3M/4$ steps for doubling} and \textbf{at least $M/8$ steps for halving}. So the actual cost can be spread over the next $M/8$ steps to yield an amortized cost of $O(1)$.
Amortized Cost Analysis Using the Potential Method

For each $i, 1 \leq i \leq n$, define $c_i$ to be the number of insertions and deletions that are executed at the $i$-th operation, and define

$$\Phi_i \; \overset{\text{def}}{=} \begin{cases} 
2num_i - size_i & \text{if } \alpha_i \geq \frac{1}{2}, \\
\frac{size_i}{2} - num_i & \text{if } \alpha_i < \frac{1}{2}, 
\end{cases}$$

Here $size_i$ is the table size, $num_i$ is the number of elements in the table, and $\alpha_i$ is the ratio $\frac{num_i}{size_i}$ after the $i$-th operation. Note that

- at time 0, the table is empty, so $\Phi_0 = 0$,
- for all $i$, $\Phi_i \geq 0$, and thus, $\Phi_n \geq \Phi_0$, and
- $\Phi_n \leq 2n - n = n$, so the contribution of the potential function to the amortized cost is at most 1.
The Amortized Cost $\hat{c}_i$ for Insertion

Here

$m = num_{i-1}$ and $s = size_{i-1}$

(a) $\alpha_{i-1} = 1$: Here $m = s$.

\[
\begin{array}{c|c|c|c|c}
  c_i & \Phi_i & \Phi_{i-1} & \hat{c}_i \\
  m+1 & 2(m+1) - 2s & 2m - s & 3 \\
\end{array}
\]

(b) $\frac{1}{2} \leq \alpha_{i-1} < 1$:

\[
\begin{array}{c|c|c|c|c}
  c_i & \Phi_i & \Phi_{i-1} & \hat{c}_i \\
  1 & 2(m+1) - s & 2m - s & 3 \\
\end{array}
\]

(c) $\alpha_i = \frac{1}{2}$: Here $m + 1 = \frac{s}{2}$.

\[
\begin{array}{c|c|c|c|c}
  c_i & \Phi_i & \Phi_{i-1} & \hat{c}_i \\
  1 & 2(m+1) - s & s/2 - m & 0 \\
\end{array}
\]

(d) $\alpha_i < \frac{1}{2}$:

\[
\begin{array}{c|c|c|c|c}
  c_i & \Phi_i & \Phi_{i-1} & \hat{c}_i \\
  1 & s/2 - m - 1 & s/2 - m & 0 \\
\end{array}
\]

So the amortized cost of insertion is $O(1)$.  
15
The Amortized Cost $\hat{c}_i$ for Deletion

(a) $\alpha_i \geq \frac{1}{2}$:

<table>
<thead>
<tr>
<th>$c_i$</th>
<th>$\Phi_i$</th>
<th>$\Phi_{i-1}$</th>
<th>$\hat{c}_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$2(m - 1) - s$</td>
<td>$2m - s$</td>
<td>$-1$</td>
</tr>
</tbody>
</table>

(b) $\alpha_{i-1} = \frac{1}{2}$: Here $2m = s$.

<table>
<thead>
<tr>
<th>$c_i$</th>
<th>$\Phi_i$</th>
<th>$\Phi_{i-1}$</th>
<th>$\hat{c}_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\frac{s}{2} - (m - 1)$</td>
<td>$2m - s$</td>
<td>$2$</td>
</tr>
</tbody>
</table>

(c) $\frac{1}{4} < \alpha_{i-1} \leq \frac{1}{2}$:

<table>
<thead>
<tr>
<th>$c_i$</th>
<th>$\Phi_i$</th>
<th>$\Phi_{i-1}$</th>
<th>$\hat{c}_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$s/2 - (m - 1)$</td>
<td>$s/2 - m$</td>
<td>$2$</td>
</tr>
</tbody>
</table>

(d) $\alpha_{i-1} = \frac{1}{4}$: $m = \frac{s}{4}$ and $\alpha_i < \frac{1}{2}$.

<table>
<thead>
<tr>
<th>$c_i$</th>
<th>$\Phi_i$</th>
<th>$\Phi_{i-1}$</th>
<th>$\hat{c}_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m$</td>
<td>$s/4 - (m - 1)$</td>
<td>$s/2 - m$</td>
<td>$1$</td>
</tr>
</tbody>
</table>

So the amortized cost of deletion is $O(1)$. 

16