In an amortized analysis, the time of a sequence of operations is averaged.

- Average does not mean averaging over a distribution of inputs.
- No probability is involved.
- We do mean average cost in the worst case.

We’ll look at 3 methods:
- aggregate analysis
- accounting method
- potential method
Aggregate Analysis

• We show that
  – For all n, a sequence of n operations takes worst-case \( T(n) \) in total.

• Thus,
  – In the worst case, the \textit{amortized cost} (average cost) per operation is therefore \( T(n)/n \).

Stack with Multi-Pop Operations

• Operations:
  – \texttt{push} \((S, x)\).
  – \texttt{pop} \((S)\).
  – \texttt{multipop} \((S, k)\):
    
    \[
    \text{while } (S \neq \emptyset \text{ \& } k > 0) \text{ do} \\
    \hspace{1cm} \text{pop}(S), \ k = k - 1
    \]

• Worst case Complexity for a sequence of n operations.
  – At most \( n \) pushes.
  – Each object can be popped only once, hence, at most \( n \) pops, including those in multi-pop.
  – Therefore, total cost = \( O(n) \), and average = \( O(n)/n = O(1) \).
Binary Counters

- k-bit binary counter $A[0 \ldots k-1]$ of bits,
  - $A[0]$ is the least significant bit, and $A[k-1]$ is the most significant bit.
  - Counts upward from 0.
  - Value of counter is: $\sum_{i=0}^{k-1} A[i] 2^i$
  - To increment:
    - **increment** $(A, k)$
      
    while $(i < k \& A[i] = 1)$ do
      
    $A[i] = 0$
    $i = i + 1$
    
    if $(i < k)$ then
    $A[i] \leftarrow 1$

- Cost of increment = $\Theta$(# of flipped bits).

Tzachi (Isaac) Rosen

<table>
<thead>
<tr>
<th>Value</th>
<th>Counter Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>1</td>
<td>0 0 0 0 0 0 0 1</td>
</tr>
<tr>
<td>2</td>
<td>0 0 0 0 0 0 1 0</td>
</tr>
<tr>
<td>3</td>
<td>0 0 0 0 0 0 1 1</td>
</tr>
<tr>
<td>4</td>
<td>0 0 0 0 0 1 0 0</td>
</tr>
<tr>
<td>5</td>
<td>0 0 0 0 0 1 0 1</td>
</tr>
<tr>
<td>6</td>
<td>0 0 0 0 0 1 1 0</td>
</tr>
<tr>
<td>7</td>
<td>0 0 0 0 0 1 1 1</td>
</tr>
<tr>
<td>8</td>
<td>0 0 0 0 1 0 0 0</td>
</tr>
<tr>
<td>9</td>
<td>0 0 0 0 1 0 0 1</td>
</tr>
<tr>
<td>10</td>
<td>0 0 0 0 1 0 1 0</td>
</tr>
<tr>
<td>11</td>
<td>0 0 0 0 1 0 1 1</td>
</tr>
<tr>
<td>12</td>
<td>0 0 0 0 1 1 0 0</td>
</tr>
<tr>
<td>13</td>
<td>0 0 0 0 1 1 0 1</td>
</tr>
<tr>
<td>14</td>
<td>0 0 0 0 1 1 1 0</td>
</tr>
<tr>
<td>15</td>
<td>0 0 0 0 1 1 1 1</td>
</tr>
<tr>
<td>16</td>
<td>0 0 0 1 0 0 0 0</td>
</tr>
</tbody>
</table>

Binary Counters

<table>
<thead>
<tr>
<th>bit</th>
<th>flips how often</th>
<th>times in $n$ INCREMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>every time</td>
<td>$n$</td>
</tr>
<tr>
<td>1</td>
<td>1/2 the time</td>
<td>$[n/2]$</td>
</tr>
<tr>
<td>2</td>
<td>1/4 the time</td>
<td>$[n/4]$</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$i$</td>
<td>1/2^i the time</td>
<td>$[n/2^i]$</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$i \geq k$</td>
<td>never</td>
<td>0</td>
</tr>
</tbody>
</table>

Therefore, total # of flips = $\sum_{i=0}^{k-1} [n/2^i]$

< $\sum_{i=0}^{\infty} 1/2^i$

= $n \left( \frac{1}{1 - 1/2} \right)$

= $2n$.

Therefore, $n$ INCREMENTS costs $O(n)$.

Tzachi (Isaac) Rosen
Accounting Method

- Assign different charges to different operations.
  - Some are charged more than actual cost, some are charged less.
  - **Amortized cost** = amount we charge.
    - When amortized cost > actual cost, store the difference on a credit object.
    - Use credit later to pay for operations whose actual cost > amortized cost.
    - Need credit to never go negative.

Let $c_i = \text{actual cost of } i\text{th operation}$.

\[
\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i \quad \text{for all sequences of } n \text{ operations.}
\]

Total credit stored = $\sum_{i=1}^{n} \hat{c}_i - \sum_{i=1}^{n} c_i \geq 0$.

Tzachi (Isaac) Rosen
Stack with Multi-Pop Operations

Stack

<table>
<thead>
<tr>
<th>operation</th>
<th>actual cost</th>
<th>amortized cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUSH</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>POP</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>MULTIPOP</td>
<td>min(k, s)</td>
<td>0</td>
</tr>
</tbody>
</table>

**Intuition:** When pushing an object, pay $2.

- $1 pays for the PUSH.
- $1 is prepayment for it being popped by either POP or MULTIPOP.
- Since each object has $1, which is credit, the credit can never go negative.
- Therefore, total amortized cost, = $O(n)$, is an upper bound on total actual cost.

Binary Counters

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 $\geq 0$.

Amortized cost of INCREMENT:

- Cost of resetting bits to $0$ is paid by credit.
- At most $1$ bit is set to $1$.
- Therefore, amortized cost $\leq$ $2$.
- For $n$ operations, amortized cost = $O(n)$.
Potential Method

• Like the accounting, but credit stored with the entire structure.
  – Accounting method stores credit with specific objects.
  – Potential method stores potential in the data structure as a whole.
  – Can release potential to pay for future operations.
  – Most flexible of the amortized analysis methods.

Tzachi (Isaac) Rosen

\[ D_i = \text{data structure after } i\text{th operation} \]
\[ D_0 = \text{initial data structure} \]

• Define a potential function \( \Phi: D \rightarrow \mathbb{R} \)

• Let
  – \( c_i = \text{actual cost of } i\text{th operation} \)

• Define
  – amortized cost of \( i\text{th operation} \)
    \[ \hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) \]
    \[ = c_i + \Delta \Phi(D_i) \]
    \( \Delta \Phi(D_i) \): increase in potential due to \( i\text{th operation} \)

Tzachi (Isaac) Rosen
Potential Method

Total amortized cost \[= \sum_{i=1}^{n} \tilde{c}_i \]
\[= \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1})) \]
(teleseoping sum: every term other than \(D_0\) and \(D_n\)
is added once and subtracted once)
\[= \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0). \]

If we require that \(\Phi(D_i) \geq \Phi(D_0)\) for all \(i\), then the amortized cost is always an upper bound on actual cost.
In practice: \(\Phi(D_0) = 0\), \(\Phi(D_i) \geq 0\) for all \(i\).

---

Stack with Multi-Pop Operations

\[\Phi = \# \text{ of objects in stack}\]
\[= \# \text{ of S1 bills in accounting method}\]

\(D_0 = \text{empty stack} \Rightarrow \Phi(D_0) = 0.\)

Since \# of objects in stack is always \(\geq 0\), \(\Phi(D_i) \geq 0 = \Phi(D_0)\) for all \(i\).

<table>
<thead>
<tr>
<th>operation</th>
<th>actual cost</th>
<th>(\Delta \Phi)</th>
<th>amortized cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUSH</td>
<td>1</td>
<td>((s + 1) - s = 1)</td>
<td>1 + 1 = 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>where (s = # \text{ of objects initially})</td>
<td></td>
</tr>
<tr>
<td>POP</td>
<td>1</td>
<td>((s - 1) - s = -1)</td>
<td>1 - 1 = 0</td>
</tr>
<tr>
<td>MULTIPOP</td>
<td>(k' = \min(k, s))</td>
<td>((s - k') - s = -k')</td>
<td>(k' - k' = 0)</td>
</tr>
</tbody>
</table>

Therefore, amortized cost of a sequence of \(n\) operations = \(O(n)\).
Binary Counters

\( \Phi = b_i = \# \) of 1’s after \( i \)th INCREMENT

Suppose \( i \)th operation resets \( t \) bits to 0.
\( c_i \leq t_i + 1 \) (resets \( t_i \) bits, sets \( \leq 1 \) bit to 1)

- If \( b_i = 0 \), the \( i \)th 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 \( i \)th operation reset \( t_i \) bits, set one, so
  \( b_i = b_{i-1} - t_i + 1 \).

Either way, \( b_i \leq b_{i-1} - t_i + 1 \).

Therefore,
\[
\Delta \Phi(D_i) \leq (b_{i-1} - t_i + 1) - b_{i-1} = 1 - t_i .
\]
\[
\hat{c}_i = c_i + \Delta \Phi(D_i) \\
\leq (t_i + 1) + (1 - t_i) = 2 .
\]

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

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

Dynamic Tables

- Description
  - A table, maybe a hash table.
  - Don’t know in advance how many objects will be stored in it.
  - When it fills, must reallocate with a larger size, copying all objects into the new, larger table.
  - When it gets sufficiently small, might want to reallocate with a smaller size.
Dynamic Tables

• Goals
  – $O(1)$ amortized time per operation.
  – Unused space always $\leq$ constant fraction of allocated space.
  
  • Load factor $\alpha = \frac{\text{num}}{\text{size}}$, where
    
    – num = # items stored
    – size = allocated size.
  
  • If size = 0, then num = 0. Call $\alpha = 1$.
  
  • Never allow $\alpha > 1$.
  
  • Keep $\alpha >$ a constant fraction.

Tzachi (Isaac) Rosen

Dynamic Tables

• Consider only insertion.
  – When the table becomes full, double its size and reinsert all existing items.
  – Guarantees that $\alpha \geq 1/2$.
  – The actually insertion is always an elementary one.
**Dynamic Tables**

\[
\text{inset (} T, x) = \begin{cases} 
T = \text{allocate table with 1 slot} & \text{if (size}(T) = 0) \\
T = \text{allocate table with 2 \cdot size}(T) \text{ slots} & \text{if (num}(T) = \text{size}(T)) \\
\text{insert all items in } T \text{ into } NT \\
\text{free } T \\
T = NT \\
\text{insert } x \text{ into } T 
\end{cases}
\]

Initially, \( T \) is empty, so \( \text{num}(T) = \text{size}(T) = 0. \)

---

**Aggregate Analysis**

\( c_i = \text{actual cost of } i\text{th operation} \)

- If not full, \( c_i = 1. \)
- If full, have \( i - 1 \) items in the table at the start of the \( i\text{th operation}. \) Have to copy all \( i - 1 \) existing items, then insert \( i\text{th item} \Rightarrow c_i = i. \)

\[
c_i = \begin{cases} 
i & \text{if } i - 1 \text{ is exact power of 2} \\
1 & \text{otherwise} 
\end{cases}
\]

Total cost \[
\begin{align*}
\sum_{i=1}^{n} c_i \\
\leq n + \sum_{j=1}^{\lfloor \log_2 n \rfloor} 2^j \\
= n + \frac{2^\lfloor \log_2 n \rfloor + 1}{2 - 1} - 1 \\
< n + 2n \\
= 3n
\end{align*}
\]
Accounting Method

• Charge $3 per insertion of x.
  – $1 pays for x’s insertion.
  – $1 pays for x to be moved in the future.
  – $1 pays for some other item to be moved.

• Total cost = $O(3n)$

Potential Method

$\Phi(T) = 2 \cdot num[T] - size[T]$

• Initially, $num = size = 0 \Rightarrow \Phi = 0$.
• Just after expansion, $size = 2 \cdot num \Rightarrow \Phi = 0$.
• Just before expansion, $size = num \Rightarrow \Phi = num \Rightarrow$ have enough potential to pay for moving all items.

• Need $\Phi \geq 0$, always.
  Always have
  
  $size \geq \frac{num}{2} \Rightarrow \frac{1}{2} \cdot size \Rightarrow \Phi \leq \frac{1}{2} \cdot size$, always.

Tzachi (Isaac) Rosen
Potential Method

\[ \text{num}_i = \text{num after } i\text{th operation}, \]
\[ \text{size}_i = \text{size after } i\text{th operation}, \]
\[ \Phi_i = \Phi \text{ after } i\text{th operation}. \]

- If no expansion:
  \[ \Phi(T) = 2 \cdot \text{num}[T] - \text{size}[T] \]
  \[ \text{size}_i = \text{size}_{i-1}, \]
  \[ \text{num}_i = \text{num}_{i-1} + 1, \]
  \[ c_i = 1. \]
  
  Then we have
  \[ \tilde{c}_i = c_i + \Phi_i - \Phi_{i-1} \]
  \[ = 1 + (2 \cdot \text{num}_i - \text{size}_i) - (2 \cdot \text{num}_{i-1} - \text{size}_{i-1}) \]
  \[ = 1 + (2 \cdot \text{num}_i - \text{size}_i) - (2(\text{num}_i - 1) - \text{size}_i) \]
  \[ = 1 + 2 \]
  \[ = 3. \]

Tzachi (Isaac) Rosen

Potential Method

- If expansion:
  \[ \text{size}_i = 2 \cdot \text{size}_{i-1}, \]
  \[ \text{size}_{i-1} = \text{num}_{i-1} = \text{num}_i - 1, \]
  \[ c_i = \text{num}_{i-1} + 1 = \text{num}_i. \]
  
  Then we have
  \[ \tilde{c}_i = c_i + \Phi_i + \Phi_{i-1} \]
  \[ = \text{num}_i + (2 \cdot \text{num}_i - \text{size}_i) - (2 \cdot \text{num}_{i-1} - \text{size}_{i-1}) \]
  \[ = \text{num}_i + (2 \cdot \text{num}_i - 2(\text{num}_i - 1)) - (2(\text{num}_i - 1) - (\text{num}_i - 1)) \]
  \[ = \text{num}_i + 2 - (\text{num}_i - 1) \]
  \[ = 3. \]

Tzachi (Isaac) Rosen
Expansion and Contraction

• When $\alpha$ drops too low, contract the table.
  – Allocate a new, smaller one.
  – Copy all items.
• Still want
  – $\alpha$ bounded from below by a constant,
  – amortized cost per operation $= O(1)$.

Expansion and Contraction

• Double as before:
  – When inserting with $\alpha = 1$
  – After doubling, $\alpha = 1/2$.
• Halve size
  – When deleting with $\alpha = 1/4$
  – After halving, $\alpha = 1/2$.
• Thus, immediately after either expansion or contraction have
  – $\alpha = 1/2$.
• Always have
  – $1/4 \leq \alpha \leq 1$.  

Expansion and Contraction

\[ \Phi(T) = \begin{cases} 
2 \cdot \text{num}(T) - \text{size}(T) & \text{if } \alpha \geq 1/2, \\
\text{size}(T)/2 - \text{num}(T) & \text{if } \alpha < 1/2. 
\end{cases} \]

- If \( T \) empty, then \( \Phi = 0. \)
- \( \alpha \geq 1/2 \Rightarrow \text{num} \geq \frac{1}{2} \cdot \text{size} \Rightarrow 2 \cdot \text{num} \geq \text{size} \Rightarrow \Phi \geq 0. \)
- \( \alpha < 1/2 \Rightarrow \text{num} < \frac{1}{2} \cdot \text{size} \Rightarrow \Phi \geq 0. \)

**Insert:**
- If \( \alpha_{i-1} \geq 1/2 \), same analysis as before \( \Rightarrow \tilde{c}_i = 3. \)
- If \( \alpha_{i-1} < 1/2 \Rightarrow \text{no expansion (only occurs when } \alpha_{i-1} = 1). \)
- If \( \alpha_{i-1} < 1/2 \) and \( \alpha_i < 1/2: \)
  \[ \tilde{c}_i = c_i + \Phi_i + \Phi_{i-1} \]
  \[ = 1 + (\text{size}_i / 2 - \text{num}_i) - (\text{size}_{i-1} / 2 - \text{num}_{i-1}) \]
  \[ = 1 + (\text{size}_i / 2 - \text{num}_i) - (\text{size}_{i-1} / 2 - (\text{num}_{i-1} - 1)) \]
  \[ = 0. \]
- If \( \alpha_{i-1} < 1/2 \) and \( \alpha_i \geq 1/2: \)
  \[ \tilde{c}_i = 1 + (2 \cdot \text{num}_i - \text{size}_i) - (\text{size}_{i-1} / 2 - \text{num}_{i-1}) \]
  \[ = 1 + (2(\text{num}_{i-1} + 1) - \text{size}_{i-1}) - (\text{size}_{i-1} / 2 - \text{num}_{i-1}) \]
  \[ = 3 \cdot \text{num}_{i-1} - \frac{3}{2} \cdot \text{size}_{i-1} + 3 \]
  \[ = 3 \cdot \alpha_{i-1} \text{size}_{i-1} - \frac{3}{2} \cdot \text{size}_{i-1} + 3 \]
  \[ < \frac{3}{2} \cdot \text{size}_{i-1} - \frac{3}{2} \cdot \text{size}_{i-1} + 3 \]
  \[ = 3. \]
Expansion and Contraction

Delete:

• If $\alpha_{i-1} < 1/2$, then $\alpha_i < 1/2$.
  • If no contraction:
    \[
    \hat{c}_i = 1 + \frac{\text{size}_i}{2} - \text{num}_i - \frac{\text{size}_{i-1}}{2} - \text{num}_{i-1}
    \]
    \[
    = 1 + \frac{\text{size}_i}{2} - \text{num}_i - \frac{\text{size}_{i-1}}{2} - (\text{num}_i + 1)
    \]
    \[
    = 2.
    \]
  • If contraction:
    \[
    \hat{c}_i = \frac{\text{num}_i + 1}{2} + \frac{\text{size}_i}{2} - \text{num}_i - \frac{\text{size}_{i-1}}{2} - \text{num}_{i-1}
    \]
    move + delete
    \[
    = \frac{\text{size}_i}{2} = \frac{\text{size}_{i-1}}{4} = \frac{\text{num}_{i-1}}{2} = \text{num}_i + 1
    \]
    \[
    = \frac{\text{num}_i + 1}{2} + \frac{(\text{num}_i + 1) - \text{num}_i}{2} - ((2 \cdot \text{num}_i + 2) - (\text{num}_i + 1))
    \]
    \[
    = 1.
    \]

Tzachi (Isaac) Rosen

Expansion and Contraction

• If $\alpha_{i-1} \geq 1/2$, then no contraction.
  • If $\alpha_i \geq 1/2$:
    \[
    \hat{c}_i = 1 + (2 \cdot \text{num}_i - \text{size}_i) - (2 \cdot \text{num}_{i-1} - \text{size}_{i-1})
    \]
    \[
    = 1 + (2 \cdot \text{num}_i - \text{size}_i) - (2 \cdot \text{num}_i + 2 - \text{size}_i)
    \]
    \[
    = -1.
    \]
  • If $\alpha_i < 1/2$, since $\alpha_{i-1} \geq 1/2$, have
    \[
    \text{num}_i = \text{num}_{i-1} - 1 > \frac{1}{2} \cdot \text{size}_{i-1} - 1 = \frac{1}{2} \cdot \text{size}_i - 1.
    \]

Tzachi (Isaac) Rosen