Proof (completed)

Q. How many $h$'s cause $x$ and $y$ to collide?

A. There are $m$ choices for each of $a_1, a_2, \ldots, a_r$, but once these are chosen, exactly one choice for $a_0$ causes $x$ and $y$ to collide, namely

$$\left( x_0 - y_0 - 1 \right) \sum_{i=1}^r a_i (x_i - y_i) \mod m.$$

Thus, the number of $h$'s that cause $x$ and $y$ to collide is $m^r \cdot 1 = m^r = |H|/m$.

October 5, 2005

Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson
How large should a hash table be?

**Goal:** Make the table as small as possible, but large enough so that it won’t overflow (or otherwise become inefficient).

**Problem:** What if we don’t know the proper size in advance?

**Solution:** *Dynamic tables.*

**Idea:** Whenever the table overflows, “grow” it by allocating (via `malloc` or `new`) a new, larger table. Move all items from the old table into the new one, and free the storage for the old table.
Example of a dynamic table

1. INSERT
2. INSERT

overflow
Example of a dynamic table

1. INSERT
2. INSERT

overflow
Example of a dynamic table

1. INSERT
2. INSERT
Example of a dynamic table

1. **INSERT**
2. **INSERT**
3. **INSERT**

overflow
Example of a dynamic table

1. INSERT
2. INSERT
3. INSERT

overflow
Example of a dynamic table

1. INSERT
2. INSERT
3. INSERT
Example of a dynamic table

1. INSERT
2. INSERT
3. INSERT
4. INSERT
Example of a dynamic table

1. INSERT
2. INSERT
3. INSERT
4. INSERT
5. INSERT

overflow
Example of a dynamic table

1. INSERT
2. INSERT
3. INSERT
4. INSERT
5. INSERT

overflow
Example of a dynamic table

1. **INSERT**
2. **INSERT**
3. **INSERT**
4. **INSERT**
5. **INSERT**
Example of a dynamic table

1. INSERT
2. INSERT
3. INSERT
4. INSERT
5. INSERT
6. INSERT
7. INSERT
Worst-case analysis

Consider a sequence of \( n \) insertions. The worst-case time to execute one insertion is \( \Theta(n) \). Therefore, the worst-case time for \( n \) insertions is \( n \cdot \Theta(n) = \Theta(n^2) \).

WRONG! In fact, the worst-case cost for \( n \) insertions is only \( \Theta(n) \ll \Theta(n^2) \).

Let’s see why.
Tighter analysis

Let \( c_i \) = the cost of the \( i \)th insertion

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

<table>
<thead>
<tr>
<th>( i )</th>
<th>( 1 )</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>( size_i )</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>( c_i )</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>9</td>
</tr>
</tbody>
</table>
Tighter analysis

Let $c_i =$ the cost of the $i$th insertion

$$= \begin{cases} 
  i & \text{if } i - 1 \text{ is an exact power of } 2, \\
  1 & \text{otherwise.}
\end{cases}$$

<table>
<thead>
<tr>
<th></th>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>size$_i$</strong></td>
<td></td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td><strong>$c_i$</strong></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Tighter analysis (continued)

Cost of $n$ insertions $= \sum_{i=1}^{n} c_i$

$\leq n + \sum_{j=0}^{\lfloor \lg(n-1) \rfloor} 2^j$

$\leq 3n$

$= \Theta(n)$.

Thus, the average cost of each dynamic-table operation is $\Theta(n)/n = \Theta(1)$. 
Amortized analysis

An *amortized analysis* is any strategy for analyzing a sequence of operations to show that the average cost per operation is small, even though a single operation within the sequence might be expensive.

Even though we’re taking averages, however, probability is not involved!

- An amortized analysis guarantees the average performance of each operation in the *worst case*. 
Types of amortized analyses

Three common amortization arguments:

- the *aggregate* method,
- the *accounting* method,
- the *potential* method.

We’ve just seen an aggregate analysis.

The aggregate method, though simple, lacks the precision of the other two methods. In particular, the accounting and potential methods allow a specific *amortized cost* to be allocated to each operation.
Accounting method

- Charge $i$th operation a fictitious amortized cost $\hat{c}_i$, where $1$ pays for $1$ unit of work (i.e., time).
- This fee is consumed to perform the operation.
- Any amount not immediately consumed is stored in the bank for use by subsequent operations.
- The bank balance must not go negative! We must ensure that
  \[ \sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{c}_i \]
  for all $n$.
- Thus, the total amortized costs provide an upper bound on the total true costs.
Accounting analysis of dynamic tables

Charge an amortized cost of $\hat{c}_i = $3 for the $i$th insertion.
• $1$ pays for the immediate insertion.
• $2$ is stored for later table doubling.

When the table doubles, $1$ pays to move a recent item, and $1$ pays to move an old item.

Example:

\[
\begin{array}{cccccccc}
$0$ & $0$ & $0$ & $0$ & $2$ & $2$ & $2$ & $2$
\end{array}
\]

overflow
Accounting analysis of dynamic tables

Charge an amortized cost of $\hat{c}_i = \$3$ for the $i$ th insertion.
- $\$1$ pays for the immediate insertion.
- $\$2$ is stored for later table doubling.

When the table doubles, $\$1$ pays to move a recent item, and $\$1$ pays to move an old item.

Example:
Accounting analysis of dynamic tables

Charge an amortized cost of $\hat{c}_i = 3$ for the $i$ th insertion.

- $1$ pays for the immediate insertion.
- $2$ is stored for later table doubling.

When the table doubles, $1$ pays to move a recent item, and $1$ pays to move an old item.

Example:
Accounting analysis (continued)

**Key invariant:** Bank balance never drops below 0. Thus, the sum of the amortized costs provides an upper bound on the sum of the true costs.

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$size_i$</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>$c_i$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>$\hat{c}_i$</td>
<td>2*</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>bank$_i$</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

*Okay, so I lied. The first operation costs only $2, not $3.*
Potential method

**Idea:** View the bank account as the potential energy (à la physics) of the dynamic set.

**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 $\Phi : \{D_i\} \rightarrow \mathbb{R}$, such that $\Phi(D_0) = 0$ and $\Phi(D_i) \geq 0$ for all $i$.
- The amortized cost $\hat{c}_i$ with respect to $\Phi$ is defined to be $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$. 
Understanding potentials

\[ \hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) \]

**potential difference** \( \Delta \Phi_i \)

- If \( \Delta \Phi_i > 0 \), then \( \hat{c}_i > c_i \). Operation \( i \) stores work in the data structure for later use.

- If \( \Delta \Phi_i < 0 \), then \( \hat{c}_i < c_i \). The data structure delivers up stored work to help pay for operation \( i \).
The amortized costs bound the true costs

The total amortized cost of $n$ operations is

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1}))$$

Summing both sides.
The amortized costs bound the true costs

The total amortized cost of $n$ operations is

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1}))$$

$$= \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0)$$

The series telescopes.
The amortized costs bound the true costs

The total amortized cost of $n$ operations is

$$
\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} \left( c_i + \Phi(D_i) - \Phi(D_{i-1}) \right)
$$

$$
= \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0)
$$

$$
\geq \sum_{i=1}^{n} c_i \quad \text{since} \quad \Phi(D_n) \geq 0 \quad \text{and} \quad \Phi(D_0) = 0.
$$
Potential analysis of table doubling

Define the potential of the table after the $i$th insertion by $\Phi(D_i) = 2i - 2^{\lfloor \log i \rfloor}$. (Assume that $2^{\lfloor \log 0 \rfloor} = 0$.)

Note:
- $\Phi(D_0) = 0$,
- $\Phi(D_i) \geq 0$ for all $i$.

Example:

$\Phi = 2 \cdot 6 - 2^3 = 4$

(accounting method)
Calculation of amortized costs

The amortized cost of the $i$th insertion is

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$
Calculation of amortized costs

The amortized cost of the $i$th insertion is

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

$$= \begin{cases} i & \text{if } i - 1 \text{ is an exact power of } 2, \\ 1 & \text{otherwise}; \\ + (2i - 2^{\lfloor \log i \rfloor}) - (2(i - 1) - 2^{\lfloor \log (i-1) \rfloor}) \end{cases}$$
Calculation of amortized costs

The amortized cost of the $i$th insertion is

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

$$= \begin{cases} 
 i & \text{if } i - 1 \text{ is an exact power of } 2, \\
 1 & \text{otherwise}; 
\end{cases}$$

$$+ \left(2i - 2^{\lfloor \log i \rfloor}\right) - \left(2(i-1) - 2^{\lfloor \log (i-1) \rfloor}\right)$$

$$= \begin{cases} 
 i & \text{if } i - 1 \text{ is an exact power of } 2, \\
 1 & \text{otherwise}; 
\end{cases}$$

$$+ 2 - 2^{\lfloor \log i \rfloor} + 2^{\lfloor \log (i-1) \rfloor}.$$
Calculation

Case 1: \( i - 1 \) is an exact power of 2.

\[
\hat{c}_i = i + 2 - 2^{\lfloor \log i \rfloor} + 2^{\lfloor \log (i-1) \rfloor}
\]
Calculation

Case 1: \( i - 1 \) is an exact power of 2.

\[
\hat{c}_i = i + 2 - 2^{\lfloor \lg i \rfloor} + 2^{\lfloor \lg (i-1) \rfloor} \\
= i + 2 - 2(i - 1) + (i - 1)
\]
**Calculation**

**Case 1:** \( i - 1 \) is an exact power of 2.

\[
\hat{c}_i = i + 2 - 2^{\lceil \lg i \rceil} + 2^{\lceil \lg (i-1) \rceil}
\]

\[
= i + 2 - 2(i - 1) + (i - 1)
\]

\[
= i + 2 - 2i + 2 + i - 1
\]
Calculation

**Case 1:** $i - 1$ is an exact power of $2$.

\[
\hat{c}_i = i + 2 - 2^{\lfloor \lg i \rfloor} + 2^{\lfloor \lg (i-1) \rfloor}
= i + 2 - 2(i - 1) + (i - 1)
= i + 2 - 2i + 2 + i - 1
= 3
\]
Calculation

Case 1: $i - 1$ is an exact power of 2.

\[ \hat{c}_i = i + 2 - 2^{\lfloor \lg i \rfloor} + 2^{\lfloor \lg (i-1) \rfloor} \]
\[ = i + 2 - 2(i - 1) + (i - 1) \]
\[ = i + 2 - 2i + 2 + i - 1 \]
\[ = 3 \]

Case 2: $i - 1$ is not an exact power of 2.

\[ \hat{c}_i = 1 + 2 - 2^{\lfloor \lg i \rfloor} + 2^{\lfloor \lg (i-1) \rfloor} \]
Calculation

Case 1: \( i - 1 \) is an exact power of 2.
\[
\hat{c}_i = i + 2 - 2^{\left\lfloor \log_2 i \right\rfloor} + 2^{\left\lfloor \log_2 (i-1) \right\rfloor} \\
= i + 2 - 2(i - 1) + (i - 1) \\
= i + 2 - 2i + 2 + i - 1 \\
= 3
\]

Case 2: \( i - 1 \) is not an exact power of 2.
\[
\hat{c}_i = 1 + 2 - 2^{\left\lfloor \log_2 i \right\rfloor} + 2^{\left\lfloor \log_2 (i-1) \right\rfloor} \\
= 3 \quad \text{(since } 2^{\left\lfloor \log_2 i \right\rfloor} = 2^{\left\lfloor \log_2 (i-1) \right\rfloor} \text{)}
\]
Calculation

Case 1: $i - 1$ is an exact power of 2.

\[
\hat{c}_i = i + 2 - 2^{\lceil \log_2 i \rceil} + 2^{\lceil \log_2 (i-1) \rceil}
\]
\[
= i + 2 - 2(i - 1) + (i - 1)
\]
\[
= i + 2 - 2i + 2 + i - 1
\]
\[
= 3
\]

Case 2: $i - 1$ is not an exact power of 2.

\[
\hat{c}_i = 1 + 2 - 2^{\lceil \log_2 i \rceil} + 2^{\lceil \log_2 (i-1) \rceil}
\]
\[
= 3
\]

Therefore, $n$ insertions cost $\Theta(n)$ in the worst case.
Calculation

Case 1: $i - 1$ is an exact power of 2.

\[
\hat{c}_i = i + 2 - 2^{\lfloor \lg i \rfloor} + 2^{\lfloor \lg (i-1) \rfloor} \\
= i + 2 - 2(i - 1) + (i - 1) \\
= i + 2 - 2i + 2 + i - 1 \\
= 3
\]

Case 2: $i - 1$ is not an exact power of 2.

\[
\hat{c}_i = 1 + 2 - 2^{\lfloor \lg i \rfloor} + 2^{\lfloor \lg (i-1) \rfloor} \\
= 3
\]

Therefore, $n$ insertions cost $\Theta(n)$ in the worst case.

Exercise: Fix the bug in this analysis to show that the amortized cost of the first insertion is only $2$. 
Conclusions

• Amortized costs can provide a clean abstraction of data-structure performance.

• Any of the analysis methods can be used when an amortized analysis is called for, but each method has some situations where it is arguably the simplest or most precise.

• Different schemes may work for assigning amortized costs in the accounting method, or potentials in the potential method, sometimes yielding radically different bounds.