12-0: Amortized Analysis

- Standard Stack
  - Push(S, elem)
  - Pop(S)
- How much time for each operation?
12-1: Amortized Analysis

- **Standard Stack**
  - Push(S, elem) $O(1)$
  - Pop(S) $O(1)$
  - Multipop(S, k)
    
    for $i \leftarrow 1$ to $k$ do
    
    Pop(S)

- How much time for multipop?
12-2: Amortized Analysis

- Standard Stack
  - Push(S, elem) $O(1)$
  - Pop(S) $O(1)$
  - Multipop(S, k) $O(k)$
12-3: Amortized Analysis

- Do $n$ operations, each of which could be either a Push, Pop, or Multipop
- How long will each operation take?
  - Push
  - Pop
  - Multipop
12-4: Amortized Analysis

- Do \( n \) operations, each of which could be either a Push, Pop, or Multipop
- How long will each operation take?
  - Push \( O(1) \)
  - Pop \( O(1) \)
  - Multipop \( O(n) \)
- What if we were to do \( n \) operations in a row, each of which is either a push/pop/multipop – how long would those \( n \) operations take?
12-5: Amortized Analysis

- $n$ operations in a row, each is either a push/pop/multipop. How long will it take?
  - Naive Method: $n$ operations, each takes time $O(n)$ – total time: $O(n^2)$

- Looking closer:
  - How many times can Pop be called (even Pop in Multipop)?
    - Once for each push!
    - Total number of Pushes $\in O(n)$
    - Total number of Pops (including pops in multipop) $\in O(n)$
    - Total time for $n$ operations: $O(n)$
Amortized Analysis

- $n$ operations in a row, each is either a push/pop/multipop.
- Total time for $n$ operations is $O(n)$
- Amortized cost for a Push, Pop, Multipop is $O(1)$
12-7: Aggregate Method

- Aggregate method
  - Total cost for $n$ operations is $g(n)$
  - Amortized cost for 1 operation is $\frac{g(n)}{n}$
- Previous analysis of push/pop/Multipop used aggregate method
Aggregate Method

- Ripple counter, width $k$
  - Examples on board
- How long does an increment take?
Aggregate Method

- Ripple counter, width $k$
- How long does an increment take?
  - $O(k)$
  - But ...
    - Least sig. bit flips every time
    - 2nd least sig. bit flips every other time
    - 3rd least sig. bit flips every 4th time
    - $k$th least sig. bit flips every $2^k$th time
- For $n$ increments (if no overflow):

$$\sum_{i=1}^{\log n} \left\lfloor \frac{n}{2^i} \right\rfloor < n \times \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$
12-10: **Aggregate Method**

- Ripple counter, width $k$
- Worst case time for a sequence of $n$ increment operations, if counter starts at 0:
  - $O(n)$
- Amortized cost for a single increment
  - $O(1)$
Accounting Method

- Assign a cost for each operation
  - Called “amortized cost”
- When amortized cost > actual cost, create a “credit” which can be used when actual cost > amortized cost
- Must design costs so that all sequences of operations always leave a “positive account”
12-12: Accounting Method

<table>
<thead>
<tr>
<th></th>
<th>actual cost</th>
<th>amortized cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Push</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Pop</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Multipop</td>
<td>min(k,s)</td>
<td></td>
</tr>
</tbody>
</table>

• What amortized costs should I give, so that any valid sequence of push/pop/multipop will never have a debt?
## 12-13: Accounting Method

<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>
12-14: Accounting Method

- Binary Counter
  - Actual Cost of setting a bit to 1 is 1
  - Actual Cost of setting a bit to 0 is 1
  - Actual Cost of an increment = # of bits flipped from 1 to 0 + 1
- What should our amortized costs be, and why?
12-15: Accounting Method

- Binary Counter
  - Amortized cost of setting a bit to 1 is: 2
  - Amortized cost of setting a bit to 0 is: ?
  - Amortized cost of an increment is: ?
Binary Counter

- Amortized cost of setting a bit to 1 is: 2
- Amortized cost of setting a bit to 0 is: 0 (!)
- Amortized cost of an increment is: 2

For $n$ increments, the total amortized cost is $O(n)$, which is also a bound on the actual cost.
12-17: Potential Method

- Define a “potential” for data structures that your algorithm uses
  - Kind of like potential energy
- When the amortized cost is greater than the actual cost, increase the potential of the data structure
- When the amortize cost is less than the actual cost, decrease the potential of the data structure
  - Potential can never be negative
Potential Method

- $\Phi(D) = \text{potential of the data structure}$
- Amortized cost of operation $c_i$ is $\text{am}(c_i)$
  - $\text{am}(c_i) = c_i + (\Phi(D_i) - \Phi(D_{i-1}))$
- Total amortized cost for a sequence of $n$ operations:

$$\sum_{i=1}^{n} \text{am}(c_i) = \sum_{i=1}^{n} (c_i \Phi(D_i) - \Phi(D_{i-1}))$$

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

- As long as the potential starts at 0 and never goes negative, the amortized cost will always be larger.
The potential function is on the *Data Structure*, not the operations.

Don’t talk about the potential of a push or a pop.

Instead, talk about the potential of the stack.
- Define a potential function on the data structure.
- Use the potential function and actual cost to determine amortized cost.
12-20: **Potential Method**

- Potential Method Examples:
  - Stack, with push/pop/multipop
    - What should the potential be?
    - What are the resulting amortized costs?
12-21: **Potential Method**

- Potential Method Examples:
  - Stack, with push/pop/multipop
    - Potential = # of elements in the stack
    - amortized cost = actual cost + change in potential
    - amortized cost of push = 1 + 1 = 2
    - amortized cost for pop = 1 + (-1) = 0
    - amortized cost for multipop = k + (-k) = 0
Potential Method Examples:

- Binary Ripple-Carry Counter
  - What is the potential?
  - What are the resulting amortized costs
Potential Method Examples:

- Binary Ripple-Carry Counter
  - Potential = # of 1’s in the counter
  - amortized cost = actual cost + change in potential
  - actual cost = # of bits flipped
  - Change in potential = # of bits flipped from 1 to 0 - # of bits flipped from 1 to 0
    - = - # of bits flipped, if counter reset to 0
    - = 2 - # of bits flipped, otherwise
  - Amortized cost ≤ 2
Standard Hash Table
- Insert/find in time $O(1)$ (no delete for now)
- Need to know an upper bound on the table size beforehand

If we don’t know the table size beforehand?
- Pick a size to start with
- If table fills, double the table size, and add everything from old table to new table

What is the time for an insert if the table can grow?
12-25: Dynamic Hash Tables

- **Standard Hash Table**
  - Insert/find in time $O(1)$ (no delete for now)
  - Need to know an upper bound on the table size beforehand

- If we don’t know the table size beforehand?
  - Pick a size to start with
  - If table fills, double the table size, and add everything from old table to new table

- What is the time for an insert if the table can grow?
  - $O(n)$
• Any single insert into a Dynamic Hash Table can take time $O(n)$
• What is the amortized cost for an insert?
  • Aggregate method

\[ c_i = \begin{cases} 
  i & \text{if } i - 1 \text{ is a power of 2} \\
  1 & \text{otherwise}
\end{cases} \]
• Aggregate Method
  • Total cost for \( n \) inserts:

\[
\sum_{i=1}^{n} c_i \leq n + \sum_{i=1}^{\lg n} 2^i \\
\leq n + 2n \\
\leq 3n
\]

• Amortized cost per insert is thus \( O(1) \)
Dynamic Hash Tables

- Accounting Method
  - Amortized cost for insert = 3
    - Cost to insert the element
    - Cost to move element when the table is expanded next time
    - Cost to move one other element when the table is expanded next time (Examples)
**Potential Method**
- Potential starts at 0, grows as we insert elements
- When the table size increases, potential drops back to 0
  - Extra potential is used to grow the table
Potential Method
- Potential starts at 0, grows as we insert elements
- When the table size increases, potential drops back to 0
- $\Phi(T) = 2 \times num[T] - size[T]$
  - $num[T] = \text{number of elements in the table}$
  - $size[T] = \text{size of table}$
- Always positive (assuming we start with a table size of 0, when first element is added we go to a table size of 2 containing 1 element)
Potential Method

- Amortized cost for an insert = actual cost + change in potential
- If $i$th insert did not cause the table to grow:

$$am(c_i) = 1 + (2 \times num_i - size_i) - (2 \times num_{i-1} - size_{i-1})$$
$$= 1 + 2 \times i - size_i - 2 \times (i - 1) + size_i$$
$$= 3$$

- If $i$th insert did cause the table to grow:

$$am(c_i) = 1 + num_{i-1} + (2 \times num_i - size_i) - (2 \times num_{i-1} - size_{i-1})$$
$$= 1 + (i - 1) + 2 \times i - 2 \times (i - 1) - (2 \times (i - 1) - (i - 1))$$
$$= 3$$
Add in deletes
Want to keep the table from being too big
Shrink the table when it gets too large (freeing space)
First try:
  - When table gets full, double the size of the table, copying elements
  - When table gets less than half full, cut the size of the table in half, copying elements
Will this still give us $O(1)$ amortized cost for an insert/delete?
Consider a table that is full

What happens when we do the following operations:
- Insert, Delete, Delete, Insert, Insert Delete, Delete, ...
Consider a table that is full

What happens when we do the following operations:
  - Insert, Delete, Delete, Insert, Insert Delete, Delete, . . .
  - Every other operation takes time $O(n)$!
  - Amortized cost per operation is $O(n)$, not $O(1)$!

What can we do?
Dynamic Hash Tables

- When table gets full, double the size of the table, copying elements
- When table gets less than 1/4 full, halve the size of the table, copying elements
**Dynamic Hash Tables**

- Potential Function $\Phi$:
  - 0 when list is exactly half full
  - Increase as # of elements in the list increases, so that the potential = # of elements in the list when the list is full
  - Increase as # of elements decreases (below 1/2 full) so that the potential = # of elements in the list when the list is 1/4 full
12-37: Dynamic Hash Tables

- Potential Function $\Phi$:
  - $\alpha = \text{load of the table: Size of table / # of elements}$

$$
\Phi(T) = \begin{cases} 
2 \cdot \text{num}[T] - \text{size}[T] & \text{if } \alpha(T) \geq 1/2 \\
\frac{\text{size}[T]}{2} - \text{num}[T] & \text{if } \alpha(T) < 1/2
\end{cases}
$$
Amortized cost for insert:

- Amortized cost = actual cost + growth in potential

\[ \text{am}(c_i) = c_i + \Phi(T_i) - \Phi(T_{i-1}) \]

Several cases:

- \( 1/4 < \alpha < 1/2 \)
- \( 1/2 \leq \alpha < 1 \)
- \( \alpha = 1 \)
• Amortized cost for insert, $1/2 \leq \alpha < 1$
  - $am(c_i) = c_i + \Phi(T_i) - \Phi(T_{i-1})$

\[
\begin{align*}
am(c_i) &= 1 + (2 \times \text{num}_i - \text{size}_i) - (2 \times \text{num}_{i-1} - \text{size}_{i-1}) \\
&= 1 + 2i - \text{size}_i - 2(i - 1) + \text{size}_i \\
&= 3
\end{align*}
\]
• Amortized cost for insert, $\alpha = 1$
  
• $am(c_i) = c_i + \Phi(T_i) - \Phi(T_{i-1})$

\[
am(c_i) = 1 + num_{i-1} + (2 \cdot num_i - size_i) - (2 \cdot num_{i-1} - size_{i-1})
\]

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

\[
= 3
\]
Amortized cost for insert, $1/4 < \alpha < 1/2$

- $am(c_i) = c_i + \Phi(T_i) - \Phi(T_{i-1})$

\[
am(c_i) = 1 + \frac{\text{size}_i}{2} - \text{num}_i - \left(\frac{\text{size}_{i-1}}{2} - \text{num}_{i-1}\right)
= 1 + \frac{\text{size}_i}{2} - \frac{\text{size}_i}{2} - \text{num}_i + (\text{num}_i) - 1
= 0
\]
Amortized cost for delete:

- Amortized cost = actual cost + growth in potential
- \( \text{am}(c_i) = c_i + \Phi(T_i) - \Phi(T_{i-1}) \)
- Several cases:
  - \( 1/4 < \alpha < 1/2 \)
  - \( 1/2 \leq \alpha \leq 1 \)
  - \( \alpha = 1/4 \)
• Amortized cost for delete, $\frac{1}{2} \leq \alpha \leq 1$

  $$\text{am}(c_i) = c_i + \Phi(T_i) - \Phi(T_{i-1})$$

\[
\begin{align*}
\text{am}(c_i) &= 1 + (2 \times \text{num}_i - \text{size}_i) - (2 \times \text{num}_{i-1} - \text{size}_{i-1}) \\
&= 1 + 2 \times \text{num}_i - 2 \times (\text{num}_i + 1) + \text{size}_i - \text{size}_i \\
&= -1
\end{align*}
\]
12-44: Dynamic Hash Tables

- Amortized cost for delete, $1/4 < \alpha < 1/2$
- $am(c_i) = c_i + \Phi(T_i) - \Phi(T_{i-1})$

\[
am(c_i) = 1 + \left(\frac{\text{size}_i}{2} - \text{num}_i\right) - \left(\frac{\text{size}_{i-1}}{2} - \text{num}_{i-1}\right)
= 1 + \left(\frac{\text{size}_i}{2} - \frac{\text{size}_i}{2}\right) + \text{num}_{i-1} - \text{num}_i
= 2
\]
Amortized cost for delete, $\alpha = 1/4$

- $am(c_i) = c_i + \Phi(T_i) - \Phi(T_{i-1})$

\[
\frac{size_i}{2} = \frac{size_{i-1}}{4} = num_{i-1} = num_i + 1
\]

\[
am(c_i) = 1 + num_i + \left(\frac{size_i}{2} - num_i\right) - \left(\frac{size_{i-1}}{2} - num_{i-1}\right)
= 1 + num_i + (num_i + 1 - num_i) - ((2 \times num_i + 2) - (num_i + 1))
= 1
\]