Amortized Complexity

✓ Aggregate method.
  • Accounting method.
  • Potential function method.
Potential Function

- \( P(i) = \text{amortizedCost}(i) - \text{actualCost}(i) + P(i - 1) \)
- \( \sum(P(i) - P(i - 1)) = \sum(\text{amortizedCost}(i) - \text{actualCost}(i)) \)
- \( P(n) - P(0) = \sum(\text{amortizedCost}(i) - \text{actualCost}(i)) \)
- \( P(n) - P(0) \geq 0 \)
- When \( P(0) = 0 \), \( P(i) \) is the amount by which the first \( i \) operations have been over charged.
Potential Function Example

\[ a = x + ((a + b) \cdot c + d) + y; \]

| actual cost | 1 1 1 1 1 1 1 1 1 5 1 1 1 1 7 1 1 7 |
| amortized cost | 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 |
| potential | 1 2 3 4 5 6 7 8 9 6 7 8 9 10 5 6 7 2 |

Potential = stack size except at end.
Accounting Method

• Guess the amortized cost.
• Show that \( P(n) - P(0) \geq 0 \).
Accounting Method Example

create an empty stack;

for (int i = 1; i <= n; i++)

    // n is number of symbols in statement

processNextSymbol();

• Guess that amortized complexity of
  processNextSymbol is 2.
• Start with \( P(0) = 0 \).
• Can show that \( P(i) \geq \) number of elements on stack after \( i \)th symbol is processed.
Accounting Method Example

\[a = x + ( ( a + b ) * c + d ) + y;\]

| actual cost | 1 1 1 1 1 1 1 1 1 5 1 1 1 1 1 7 1 1 7 |
| amortized cost | 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 |
| potential | 1 2 3 4 5 6 7 8 9 6 7 8 9 10 5 6 7 2 |

- Potential \( \geq \) number of symbols on stack.
- Therefore, \( P(i) \geq 0 \) for all \( i \).
- In particular, \( P(n) \geq 0 \).
Potential Method

- Guess a suitable potential function for which $P(n) - P(0) \geq 0$ for all $n$.
- Derive amortized cost of $i$th operation using $\Delta P = P(i) - P(i-1)$
  $\quad = \text{amortized cost} - \text{actual cost}$
- amortized cost = actual cost + $\Delta P$
Potential Method Example

create an empty stack;

for (int i = 1; i <= n; i++)

    // n is number of symbols in statement

    processNextSymbol();

• Guess that the potential function is $P(i) =$ number of elements on stack after $i$th symbol is processed (exception is $P(n) = 2$).

• $P(0) = 0$ and $P(i) – P(0) \geq 0$ for all $i$. 
\( i^{th} \) Symbol Is Not \) or \;

- Actual cost of \texttt{processNextSymbol} is 1.
- Number of elements on stack increases by 1.
- \( \Delta P = P(i) - P(i-1) = 1 \).
- Amortized cost = actual cost + \( \Delta P \)

  \[ = 1 + 1 = 2 \]
The $i^{th}$ Symbol Is $
abla$

- Actual cost of `processNextSymbol` is $\#\text{unstacked} + 1$.
- Number of elements on stack decreases by $\#\text{unstacked} - 1$.
- $\Delta P = P(i) - P(i-1) = 1 - \#\text{unstacked}$.
- Amortized cost = actual cost + $\Delta P$
  
  \[
  = \#\text{unstacked} + 1 + (1 - \#\text{unstacked})
  
  = 2
  \]
i^{th} Symbol Is;

- Actual cost of `processNextSymbol` is 
  \#unstacked = P(n–1).
- Number of elements on stack decreases by 
P(n–1).
- \(\Delta P = P(n) - P(n–1) = 2 - P(n–1).\)
- amortized cost = actual cost + \(\Delta P\)
  
  \[= P(n–1) + (2 - P(n–1))\]
  
  \[= 2\]
• n-bit counter
• Cost of incrementing counter is number of bits that change.
• Cost of 001011 => 001100 is 3.
• Counter starts at 0.
• What is the cost of incrementing the counter m times?
Worst-Case Method

- Worst-case cost of an increment is $n$.
- Cost of $011111 \Rightarrow 100000$ is $6$.
- So, the cost of $m$ increments is at most $mn$. 
Aggregate Method

0 0 0 0 0

counter

• Each increment changes bit 0 (i.e., the right most bit).
• Exactly $\text{floor}(m/2)$ increments change bit 1 (i.e., second bit from right).
• Exactly $\text{floor}(m/4)$ increments change bit 2.
Aggregate Method

0 0 0 0 0

counter

- Exactly $\text{floor}(m/8)$ increments change bit 3.
- So, the cost of $m$ increments is $m + \text{floor}(m/2) + \text{floor}(m/4) + \ldots < 2m$
- Amortized cost of an increment is $2m/m = 2$. 
**Accounting Method**

- Guess that the amortized cost of an increment is 2.
- Now show that $P(m) - P(0) \geq 0$ for all $m$.

1\textsuperscript{st} increment:
- one unit of amortized cost is used to pay for the change in bit 0 from 0 to 1.
- the other unit remains as a credit on bit 0 and is used later to pay for the time when bit 0 changes from 1 to 0.

<table>
<thead>
<tr>
<th>bits</th>
<th>0 0 0 0 0</th>
<th>0 0 0 0 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>credits</td>
<td>0 0 0 0 0</td>
<td>0 0 0 0 1</td>
</tr>
</tbody>
</table>
2nd Increment.

- one unit of amortized cost is used to pay for the change in bit 1 from 0 to 1
- the other unit remains as a credit on bit 1 and is used later to pay for the time when bit 1 changes from 1 to 0
- the change in bit 0 from 1 to 0 is paid for by the credit on bit 0
3rd Increment.

- one unit of amortized cost is used to pay for the change in bit 0 from 0 to 1
- the other unit remains as a credit on bit 0 and is used later to pay for the time when bit 1 changes from 1 to 0
4th Increment.

- one unit of amortized cost is used to pay for the change in bit 2 from 0 to 1
- the other unit remains as a credit on bit 2 and is used later to pay for the time when bit 2 changes from 1 to 0
- the change in bits 0 and 1 from 1 to 0 is paid for by the credits on these bits
Accounting Method

- $P(m) - P(0) = \sum (\text{amortizedCost}(i) - \text{actualCost}(i))$
  
  = amount by which the first $m$ increments have been over charged

  = number of credits

  = number of 1s

  $\geq 0$
Potential Method

• Guess a suitable potential function for which $P(n) - P(0) \geq 0$ for all $n$.
• Derive amortized cost of $i$th operation using $\Delta P = P(i) - P(i-1)$
  $= \text{amortized cost} - \text{actual cost}$
• amortized cost $= \text{actual cost} + \Delta P$
Potential Method

- Guess $P(i) =$ number of 1s in counter after $i$th increment.
- $P(i) \geq 0$ and $P(0) = 0$.
- Let $q =$ # of 1s at right end of counter just before $i$th increment ($01001111 \Rightarrow q = 4$).
- Actual cost of $i$th increment is $1 + q$.
- $\Delta P = P(i) - P(i - 1) = 1 - q$ ($01001111 \Rightarrow 01010000$)
- amortized cost = actual cost + $\Delta P$
  
  $= 1 + q + (1 - q) = 2$
Amortized analyses: dynamic table

- A nice use of amortized analysis
- Operation
  - Table-insertion
  - table-deletion.
- Scenario:
  - A table – maybe a hash table
  - Do not know how large in advance
  - May **expand** with insertion
  - May **contract** with deletion
  - Detailed implementation is not important
Amortized analyses: dynamic table

• Goal:
  - $O(1)$ amortized cost.
  - Unused space always $\leq$ constant fraction of allocated space.
Dynamic table

• **Load factor**
  - $\alpha = \frac{\text{num}}{\text{size}}$
  - where $\text{num} = \# \text{ items stored}$, $\text{size} = \text{allocated size}$.
• If $\text{size} = 0$, then $\text{num} = 0$. Call $\alpha = 1$.
• Never allow $\alpha > 1$.
• Keep $\alpha$ a constant fraction $\rightarrow$ goal (2).
Dynamic table: expansion with insertion

- Table expansion
- Consider only insertion.
- When the table becomes full, double its size and reinsert all existing items.
- Guarantees that $\alpha \geq 1/2$.
- Each time we actually insert an item into the table, it’s an elementary insertion.
TABLE-INSERT \((T, x)\)

1. if \(\text{size}[T] = 0\)
2. then allocate \(\text{table}[T]\) with 1 slot
3. \(\text{size}[T] \leftarrow 1\)
4. if \(\text{num}[T] = \text{size}[T]\)
5. then allocate \(\text{new-table}\) with \(2 \cdot \text{size}[T]\) slots
6. insert all items in \(\text{table}[T]\) into \(\text{new-table}\)
7. free \(\text{table}[T]\)
8. \(\text{table}[T] \leftarrow \text{new-table}\)
9. \(\text{size}[T] \leftarrow 2 \cdot \text{size}[T]\)
10. insert \(x\) into \(\text{table}[T]\)
11. \(\text{num}[T] \leftarrow \text{num}[T] + 1\)
Aggregate analysis

- **Running time:**
  - Charge 1 per elementary insertion.
- Count only elementary insertions,
  - all other costs together are constant per call.
- $c_i = \text{actual cost of } i\text{th operation}$
  - 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
    - $\Rightarrow c_i = i$
Aggregate analysis

- **Cursory analysis:**
  - $n$ operations $\Rightarrow$
  - $c_i = O(n) \Rightarrow$
  - $O(n^2)$ time for $n$ operations.

- Of course, we don’t always expand:
  - $c_i = i$
  - if $i - 1$ is exact power of 2,
    - 1 otherwise.
Aggregate analysis

• So total cost =
  - $\sum_{i=1}^{n} ci$
  - $\leq n+$
    - $\sum_{i=0}^{\log(n)} 2^i$
  - $\leq n + 2n = 3n$

• Therefore, aggregate analysis says
  - amortized cost per operation = 3.
Accounting analysis

• 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.

• Suppose we’ve just expanded
  - $size = m$ before next expansion
  - $size = 2m$ after next expansion.

• Assume that the expansion used up all the credit, so that there’s no credit stored after the expansion
Accounting analysis

- Will expand again after another $m$ insertions.
- Each insertion will
  - put $1$ on one of the $m$ items that were in the table just after expansion
  - put $1$ on the item inserted.
- Have $2m$ of credit by next expansion
- when there are $2m$ items to move.
- Just enough to pay for the expansion, with no credit left over!
Potential method

- $\Phi(T) = 2 \times \text{num}[T] - \text{size}[T]$

- Initially,
  - $\text{num} = \text{size} = 0$
  - $\Rightarrow \Phi = 0$.

- Just after expansion,
  - $\text{size} = 2 \times \text{num}$
  - $\Rightarrow \Phi = 0$.

- Just before expansion,
  - $\text{size} = \text{num}$
  - $\Rightarrow \Phi = \text{num}$
  - enough to pay for moving all items.
Potential method

• Need
  - $\Phi \geq 0$, always.

• Always have
  - $\text{size} \geq \text{num} \geq \frac{1}{2} \text{size} \Rightarrow$
  - $2 \cdot \text{num} \geq \text{size} \Rightarrow$
  - $\Phi \geq 0$.
Potential method

- **Amortized cost of \( i^{th} \) operation:**
  - \( \text{num}_i = \text{num} \) after \( i^{th} \) operation ,
  - \( \text{size}_i = \text{size} \) after \( i^{th} \) operation ,
  - \( \Phi_i = \Phi \) after \( i^{th} \) operation .

- **If no expansion:**
  - \( \text{size}_i = \)
  - \( \text{size}_{i-1} \),
  - \( \text{num}_i = \)
  - \( \text{num}_{i-1} + 1 \),
  - \( c_i = 1 \).

- **\( C_i' = c_i + \Phi_i - \Phi_{i-1} \)**
  - \( = 1 + (2\text{num}_i - \text{size}_i) - (2\text{num}_{i-1} - \text{size}_{i-1}) \)
  - \( = 3. \)
Potential method

• If expansion:
  
  - \(size_i = \)
  
  - \(2size_{i-1},\)
  
  - \(size_{i-1} = \)
  
  - \(num_{i-1} = num_i - 1,\)
  
  - \(c_i = num_{i-1} + 1 = num_i.\)

• \(C_i' = c_i + \Phi_i - \Phi_{i-1}\)
  
  - \(= num_i + (2num_i - size_i) - (2num_{i-1} - size_{i-1})\)
  
  - \(= num_i + (2num_i - 2(num_i - 1)) - (2(num_i - 1) - (num_i - 1))\)
  
  - \(= num_i + 2 - (num_i - 1) = 3\)
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 $= O(1)$.

• Measure cost in terms of elementary insertions and deletions.
Obvious strategy

• Double size when inserting into a full table (when \( \alpha = 1 \), so that after insertion \( \alpha \) would become <1).

• Halve size when deletion would make table less than half full (when \( \alpha = 1/2 \), so that after deletion \( \alpha \) would become \( \geq 1/2 \)).

• Then always have \( 1/2 \leq \alpha \leq 1 \).

• Something BAD happened…
Obvious strategy

- Suppose we fill table.
  - insert →
    - double
  - 2 deletes →
    - halve
  - 2 inserts →
    - double
  - 2 deletes →
    - halve
  - ...

- Cost of each expansion or contraction is $\Theta(n)$, so total n operation will be $\Theta(n^2)$. 
Obvious strategy

• Problem is that:
  ▪ Not performing enough operations after expansion or contraction to pay for the next one.

• Want to make sure that we perform enough operations between consecutive expansions/contractions to pay for the change in table size.
Simple solution

• Double as before: when inserting with $\alpha = 1$
  - $\Rightarrow$ after doubling, $\alpha = 1/2$.

• Halve size
  - when deleting with $\alpha = 1/4$
  - $\Rightarrow$ after halving, $\alpha = 1/2$.

• Thus, immediately after either expansion or contraction
  - $\alpha = 1/2$.

• Always have $1/4 \leq \alpha \leq 1$. 
Simple solution

- Suppose we’ve just expanded/contracted
- Need to delete half the items before contraction.
- Need to double number of items before expansion.
- Either way, number of operations between expansions/contractions is at least a constant fraction of number of items copied.
Potential function

- $\Phi(T) = 2\text{num}(T) - \text{size}(T)$ if $\alpha \geq \frac{1}{2}$
  
- $\text{size}(T)/2 - \text{num}(T)$ if $\alpha < \frac{1}{2}$.

- $T$ empty $\Rightarrow \Phi = 0$.

- $\alpha \geq 1/2$ $\Rightarrow$
  - $\text{num} \geq 1/2\text{size}$ $\Rightarrow$
  - $2\text{num} \geq \text{size}$ $\Rightarrow$
  - $\Phi \geq 0$.

- $\alpha < 1/2$ $\Rightarrow$
  - $\text{num} < 1/2\text{size}$ $\Rightarrow$
  - $\Phi \geq 0$. 
intuition

- measures how far from $\alpha = 1/2$ we are.
  - $\alpha = 1/2 \Rightarrow$
    - $\Phi = 2\text{num} - 2\text{num} = 0.$
  - $\alpha = 1 \Rightarrow$
    - $\Phi = 2\text{num} - \text{num}$
    - $= \text{num}.$
  - $\alpha = 1/4 \Rightarrow$
    - $\Phi = \text{size}/2 - \text{num} =$
    - $= 4\text{num}/2 - \text{num} = \text{num}.$
intuition

• Therefore, when we double or halve, have enough potential to pay for moving all num items.

• Potential increases linearly between $\alpha = 1/2$ and $\alpha = 1$, and it also increases linearly between $\alpha = 1/2$ and $\alpha = 1/4$.

• Since $\alpha$ has different distances to go to get to 1 or 1/4, starting from 1/2, rate of increase differs.
intuition

• $\Phi(T) = 2\text{num}[T] - \text{size}[T]$ if $\alpha \geq \frac{1}{2}$

• For $\alpha$ to go from $1/2$ to $1$,
  - $\text{num}$ increases from $\text{size}/2$ to $\text{size}$, for a total increase of $\text{size}/2$.
  - $\Phi$ increases from $0$ to $\text{size}$.
  - $\Phi$ needs to increase by $2$ for each item inserted.

• That’s why there’s a coefficient of $2$ on the $\text{num}[T]$ term in the formula for when $\alpha \geq 1/2$. 
intuition

- \( \Phi(T) = size[T]/2 - num[T] \) if \( \alpha < \frac{1}{2} \).

- For \( \alpha \) to go from \( \frac{1}{2} \) to \( \frac{1}{4} \)
  - \( num \) decreases from \( size/2 \) to \( size /4 \), for a total decrease of \( size/4 \).
  - \( \Phi \) increases from 0 to \( size/4 \).
  - \( \Phi \) needs to increase by 1 for each item deleted.

- That’s why there’s a coefficient of \(-1\) on the \( num[T] \) term in the formula for when \( \alpha < 1/2 \).
Amortized cost for each operation

- Amortized costs: more cases
  - insert, delete
  - \( \alpha \geq 1/2, \: \alpha < 1/2 \) (use \( \alpha_i \), since \( \alpha \) can vary a lot)
  - size does/doesn’t change
- Exercise