Amortized Analysis

- Bound the total cost of a sequence of data structure operations.
- Amortize the cost of an expensive operation over the whole sequence of operations

Three methods:
- Aggregate
- Accounting
- Potential

Two examples
- A stack with multipop
- Incrementing a binary counter
Stack S supporting operations

- Push
- Pop
- Multipop(S,k) -- remove top k items from S, or empty S if |S| < k.

Worst case cost of a sequence of n operations?
Worst case per multipop -- O(n)

Aggregate analysis gives a better bound --- O(n)

Total cost = Total #push + Total #pop
Total #pop include those from multipop.

Total #pop ≤ Total #push
∴ An item that is popped must have been pushed at some earlier point.

Total #push ≤ n
∴ Total #push + Total #pop ≤ 2n.

O(1) per op on the average.
Increment an k-bit binary counter

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 0 & 1 \\
0 & 1 & 0 \\
0 & 1 & 1 \\
1 & 0 & 0 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
1 & 1 & 1 \\
\end{array}
\]

Worst case cost of an increment = \(O(k)\)
Worst case cost of \(n\) increments = \(O(nk)\)

Aggregate Analysis gives \(O(n)\) worst case total cost of \(n\) ops.

\[
\# \text{bit - flops} = \sum_{i=0}^{k} \# \text{bit - flops at } A[i]
\]

Along column \(A[i]\): bit flops once every \(2^i\)

\[
\# \text{bit - flops at } A[i] = \left\lfloor \frac{n}{2^i} \right\rfloor
\]

\[
\sum_{i=0}^{k} \left\lfloor \frac{n}{2^i} \right\rfloor < n \sum_{i=0}^{k} \frac{1}{2^i} < 2n.
\]
Accounting Method:

- Assign "amortized cost" to each operation.
- If amortized cost > actual cost, the difference is used as "credit" to pay for future cost.

Stack with multipop

- Charge $2 to every push -- $1 to pay for actual cost, $1 credit saved for the future
- Charge $0 for pop and multipop.

Total actual cost is no greater than the total amortized cost, which is bounded by $2^{#\text{push}} = O(n)$.

Binary Counter

Charging scheme:

When a bit is set, charge $2 amortized cost -- $1 to pay for the actual cost, $1 placed on the bit as credit; to pay for future reset.

Charge $0 amortized cost for resetting a bit.

Obs: In each increment, at most one bit is set.

$O(n)$ amortized cost upper-bounding total actual cost.
Potential Method

- Define a potential function on the data structure.

- Each configuration of the data structure has a potential value representing a "potential of cost".

Potential function $\Phi$ on a data structure.

$D$: a configuration of the data structure.

$\Phi(D)$: the potential of $D$.

Suppose a seq. of op. is applied on the data structure.

$D_0$: initial configuration.

$c_i$: cost of $i$th op.

$D_i$: configuration after $i$th op.

$c_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ (amortized cost)

$\Delta \Phi(i) = \Phi(D_i) - \Phi(D_{i-1})$: change in potential after the $i$th op.
\[
\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = c_i + \Delta \Phi(i)
\]

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

\[
\sum_{i=1}^{n} \Phi(D_i) - \Phi(D_{i-1}) = \Phi(D_n) - \Phi(D_0) = \Delta \Phi
\]

\(\hat{c}_i > c_i\): overcharged due to increase in potential

\(\hat{c}_i < c_i\): undercharge, potential drops, being released to pay for part of actual cost.

Total actual cost = \(\sum_{i=1}^{n} c_i\)

Total amortized cost = \(\sum_{i=1}^{n} \hat{c}_i\)

\[
\hat{c}_i = c_i + \Delta \Phi(i)
\]

\[
\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} c_i + \Delta \Phi
\]

\(\hat{c}_i > c_i\): overcharged due to increase in potential

\(\hat{c}_i < c_i\): undercharge, potential drops, being released to pay for part of actual cost.

Total actual cost = \(\sum_{i=1}^{n} c_i\)

Total amortized cost = \(\sum_{i=1}^{n} \hat{c}_i\)

\(\Delta \Phi = \text{net change in potential}\)

\(= \Phi(D_n) - \Phi(D_0)\)
Stack with multipop

\( \Phi(D) := \text{#items in the stack.} \)
\( \hat{c}_i = c_i + \Delta \Phi(i) \)
If \( i \)th op is push:
\( \hat{c}_i = c_i + 1 = 1 + 1 = 2. \)
If \( i \)th op is pop:
\( \hat{c}_i = c_i + (-1) = 1 + (-1) = 0. \)
If \( i \)th op is multipop and \( k \) items are popped:
\( \hat{c}_i = c_i + (-k) = k + (-k) = 0. \)
\[ \sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} c_i + \Delta \Phi = 2 \# \text{push} \]
\( \Delta \Phi = \text{net change in potential} \)
\( = \Phi(D_n) - \Phi(D_0) \geq 0 \)
if stack is initially empty.

Binary Counter

\( \Phi(D) := \text{#1s in the counter. (potential of reset cost)} \)
\( \hat{c}_i = c_i + \Delta \Phi(i) \)
If \( i \)th op resets \( k \) bits:
\( \Delta \Phi(i) \leq -(k-1) \)
\( \hat{c}_i \leq (k + 1) - (k - 1) = 2. \)
\[ \sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} c_i + \Delta \Phi \]
\[ \sum_{i=1}^{n} \hat{c}_i \leq 2n. \]
\( \Delta \Phi = \text{net change in potential} \)
\( = \Phi(D_n) - \Phi(D_0) \geq 0 \)
if the counter is 0 initially.