Amortized Analysis

- Time required to perform a sequence of data-structure operations is averaged over all operations performed.
- Can be used to show that average cost is small.
- It is different from average case analysis since it does not use probabilities.

Aggregate Analysis: Incrementing a binary counter

- We show that for all n, a sequence of n operations takes *worst-case* time $T(n)$.
- The amortized cost is $T(n)/n$ in the worst case.
- We use an array $A[0, \ldots k-1]$ to store the counter.
- The lowest order bit is stored in $A[0]$.
- We use the counter to count upward from 0.
- $x = \sum_{i=0}^{k-1} A[i] \cdot 2^i$.
- $l[A] =$ length of A.

Incrementing a binary counter II

INCREMENT(A)
  $i \leftarrow 0$
  while $i < l[A]$ and $A[i] = 1$ **do**
    $A[i] \leftarrow 0$
    $i \leftarrow i + 1$
  if $i < l[A]$ **then** $A[i] \longleftarrow 1$

Analysis

- A single exec. of INCREMENT takes $O(k)$ worst case time.
- A sequence of n increments would take time $O(nk)$.
- We can tighten our analysis to yield a worst case of $O(n)$ op.
- $A[0]$ is flipped every time we call INCREMENT.
- $A[1]$ is only flipped every other time.
- $A[2]$ is flipped $\lfloor n/4 \rfloor$ times.
- In general, for $i = 0, 1, \ldots, \lfloor \lg n \rfloor$, bit $A[i]$ flips $\lfloor n/2^i \rfloor$ times in a sequence of n increments.
- For $i > \lfloor \lg n \rfloor$, bit $A[i]$ never flips at all.
- The total number of flips is $\sum_{i=0}^{\lfloor \lg n \rfloor} \left\lfloor \frac{n}{2^i} \right\rfloor < n \cdot \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$.

Accounting method

- We assign different charges to different operations.
- Some operations are charged more than and others less than they actually cost.
- The amount we charge an operation is called **amortised** cost.
- When the amortised cost of an operation exceeds the real cost, the difference is assigned to specific objects in the data structure, *credit*
- The credit can be used later for another operation.
- The amortized cost function has to be chosen very carefully!
- The amortised cost of a sequence of operations must be an upper bound on the total cost of the sequence of operations, also in the worst case.

Accounting method: incrementing binary counter

- Let c_i be the actual cost of the i-th operation, and $\hat{c}_i$ the amortised cost.
- $\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$.
- The total credit stored in the data structure is the difference between the total amortised cost and the total actual cost:

$$\sum_{i=1}^{n} \hat{c}_i - \sum_{i=1}^{n} c_i.$$

- The total credit is never allowed to become negative!

Analysis

- We charge an amortised cost of \$2 to set a bit to one.
- When a bit is set, we use \$1 out of the \$2 to pay for the setting.
- We place the other dollar on the bit as credit to be used later when we flip it back to zero.
- At any point, every one in the counter has a dollar on it. Thus, we need not charge anything to reset a bit to zero.
- The cost of resetting a bit in the while loop is paid for by the saved dollars paid for by the bits that are reset.
- At most one bit is set in every round of the loop. Hence, the amortised cost of an INCREMENT operation is \$2.
- n INCREMENT operations cost $O(n)$.

Potential Method

- The "prepaid" work is represented by a potential that can be used to pay for future operations.
- Very important method that is used in lots of different contexts.
- We start with an initial data structure D_0 on which n operations are performed.
- For $1 \leq i \leq n$ let c_i be the cost of the i-th operation.
- D_i is the data structure that results after applying the i-th operation to the data structure D_{i-1}.
- A *potential function* ϕ maps each data structure D_i to a real number $\phi(D_i)$.

Potential Method II

- The amortised cost $\hat{c}_i$ of the i-th operation is defined by

$$\hat{c}_i + \phi(D_i) - \phi(D_{i-1}).$$

- The amortised cost of each operation is therefore its actual cost plus the increase in the potential due to the operation.
- The total amortised cost of the n operations is

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \phi(D_i) - \phi(D_{i-1})) = \sum_{i=1}^{n} c_i + \phi(D_n) - \phi(D_0).$$

- If we can define a potential function ϕ such that $\phi(D_n) \geq \phi(D_0)$, then the total amortized cost $\sum_{i=1}^{n} \hat{c}_i$ is an upper bound on the actual cost $\sum_{i=1}^{n} c_i$

Potential Method III

- In practice, we often do not know in advance how many operations we will make. Then we need $\phi(D_i) \geq \phi(D_0)$ for all i.
- In most cases $\phi(D_0) = 0$.
- If the potential is positive, then we overcharged for some operations. If it is negative we undercharged.
- Of course, different potential functions will result if different amortised costs!

Potential Method: Binary Counter

- We define the potential of the counter after the i-th INCREMENT operation to be b_i, the number of 1's in the counter after the i-th operation.
- Suppose the i-th INCREMENT operation resets t_i bits. The actual cost of the operation is $(t_i + 1)$ (resetting t_i bits and set one bit to one).
- $b_i \leq b_{i-1} - t_i + 1$:
  - If $b_i = 0$ then the i-th operation resets all k bits, and so $b_{i-1} = t_i = k$.
  - If $b_i > 0$, then $b_i = b_{i-1} - t_i + 1$.

Potential Method: Binary Counter

- The potential difference is

$$\phi(D_i) - \phi(D_{i-1}) \leq (b_{i-1} - t_i + 1) = 1 - t_i.$$

- The amortised cost is therefore

$$\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1}) \leq (t_i + 1) + (1 - t_i) = 2.$$

- If the counter starts at zero, then $\phi(D_0) \geq 0$ for all i. The amortised cost is an upper bound on the actual cost and the worst case cost on n INCREMENT operations is $O(n)$.
- But this time we also get results if the counter starts at another value:

$$\sum_{i=1}^{n} c_i = \sum_{i=1}^{n} \hat{c}_i - \phi(D_n) = \phi(D_0) \leq \sum_{i=1}^{n} 2 - b_n + b_0 = 2n - b_n + b_0$$