Amortized Analysis

Though the worst case performance of an algorithm may be f(n), the average worst case over n runs may be asymptotically less than n*f(n).

We will study three methods for average worst case analysis:

1. **Aggregate Method.** Show all n runs take a total worst case time T, thus the average worst case is T/n.
2. **Accounting Method.** Performance costs assigned to some of the n runs are overestimates and are used as credit for underestimates of other runs.
3. **Potential Method.** The overestimates add to the "potential energy" of the method.

Example: Binary Counter A[0,...,k-1]

A[0] is the low-order bit.

A[k-1] is the high-order bit.

```
Increment(A)
   i = 0
   while i < length(A) and A[i] = 1
      A[i] = 0
      i = i+1
   if i < length(A)
   then A[i] = 1
```

3	2	1	O
O	O	O	O
O	O	O	1
O	O	1	O
O	O	1	1
O	1	O	O
O	1	O	1
O	1	1	O
O	1	1	1
1	O	O	O

- The worst case run time is $\Theta(k)$,
- Over n calls, worst case is ________
- But actual worst-case run time for n calls is _______

Aggregate Method

Note: Not all k bits flip for each call.
Bit A[0] flips _ times for n calls.

Bit A[1] flips $\lfloor n/2 \rfloor$ times for n calls.
Bit A[2] flips $\lfloor n/4 \rfloor$ times for n calls.
Bit A[i] flips $\lfloor n/2^i \rfloor$ times for n calls, where i = 0, 1, ..., $\lfloor \lg n \rfloor$.
For i > $\lfloor lgn \rfloor$, A[i] does not flip.

Aggregate Method

$$
\begin{gathered}
\text{T(n) is the worst case time for n calls} \\
= \Sigma_{i=0}^{\lfloor lgn \rfloor} n/2^i \\
= n\, \Sigma_{i=0}^{\lfloor lgn \rfloor} 1/2^i \;<\; n\, \Sigma_{i=0}^{\infty} 1/2^i \\
= n(\tfrac{1}{1-1/2}) \\
= \text{2n} \\
\text{T(n) = O(n)}
\end{gathered}
$$

The amortized cost of each call is thus $O(n)/n = \text{O}(1)$.

Accounting Method

Different operations have different costs.

Cost overestimates fund cost underestimates.

Example

Operation	Cost	
bit to 1	2*	overestimate
bit to 0	0**	underestimate

Need to set some bits to 1 before set to 0.

* one for actual bit flip, one for credit

* use credit from the time when set to 1

Because resetting bits in the while loop are "paid for", each call to Increment incurs a cost of 2 for the bit set to 1.

Each call is ______ amortized cost

n calls is ______ amortized cost

Constraints

1. The total amortized cost must be an upper bound on the actual cost.

2. The total credit in data structures must always be nonnegative.

Potential Method

Credit adds to "potential" of whole data structure instead of to individual objects.

Definitions

- D_0 = initial data structure
- c_i = actual cost of ith operation resulting in data structure D_i after operating on D_{i-1} (i = 1, ..., n).
- $\Phi(D_i)$ = real number potential associated with data structure D_i. Φ represents the potential function.
- $\hat{c}_i$ = amortized cost of ith operation with respect to Φ
  $\hat{c}_i$ = actual cost + potential increase
  $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$.
- The total amortized cost is

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

This is a ______________________________ $\Sigma_{i=1}^{n}(a_i - a_{i-1}) = a_n - a_0$

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

If $\Phi(D_n) \geq \Phi(D_0)$, then the amortized cost is an upper bound on the actual cost.

However, we do not know n.

If $\Phi(D_i) \geq \Phi(D_0)$ for all i, then we always pay in advance.

Let $\Phi(D_0) = 0$, thus we want $\Phi(D_i) \geq 0$.

This is similar to the accounting method since we credit potential when $\Phi(D_i) - \Phi(D_{i-1})$ is positive and debit potential when $\Phi(D_i) - \Phi(D_{i-1})$ is negative.

Example

In our example, $\Phi(D_i) = b_i$, the number of 1s in D_i (counter).

Let t_i = number of bits reset to 0 on the ith call to Increment.

Thus, the actual cost c_i is at most (cost of reset) + (cost to set one) = $t_i + 1$.

We know that $b_i \leq b_{i-1} - t_i + 1$

$$\text{Thus, } \Phi(D_i) - \Phi(D_{i-1}) \leq (b_{i-1} - t_i + 1) - b_{i-1}$$

$$= 1 - t_i$$

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

$$\leq t_i + 1 + (1 - t_i)$$

$$= 2$$

If $\Phi(D_0) = 0$, then $\Phi(D_i) \geq 0$ for all i, and the total amortized cost is an upper bound to the actual cost.

The counter starts at zero, $\Phi(D_0) = 0$.

$$\hat{c_i} = O(2)$$

n calls is ______

Dynamic Tables

```
TableInsert(T, x)
    if size(T) = 0
    then new table[T] with 1 slot
        size[T] = 1
    if num(T) = size(T)
    then create new table with 2*size[T] slots          ; α ≥ 1/2
        copy items from table[T] to new table
        free table[T]
        table[T] = new table
        size[T] = 2*size[T]
    insert x into table[T]
    num[T] = num[T] + 1
```

Aggregate Method

Let n = number of items in table.

The worst-case running time of this algorithm is O(n).

For n calls the worst-case running time is $O(n^2)$.

Double the table size when full. This expansion is performed once every power of 2 steps in 1...n.

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

$$\Sigma_{i=1}^{n} c_i \leq n + \Sigma_{j=0}^{\lfloor lgn \rfloor} 2^j$$

This is a geometric series $\Sigma_{k=0}^{n} x^k = \frac{x^{n+1}-1}{x-1}$

$$< n + 2n$$

$$= 3n$$

The total amortized cost of a single call to TableInsert is 3.

Accounting Method

TableInsert cost should be 3. This cost pays for:

- Insert in existing table
- Copy to new table
- Copy one item already in table

If m = size[T] after expanding, then num[T] = ____ and ______________
Charge $3 per Insert.
Insert costs $1 ($2 left).
For m/2 items, m/2 credit for new items + m/2 credit for existing items.

Potential Method

Define potential function Φ.

- $\Phi(T)$ = 2 * num[T] - size[T]
- $\Phi(T) = 0$ immediately after expansion
- $\Phi(T)$ approaches size[T] when T gets full
- num[T] $\geq$ size[T]/2, so $\Phi(T) \geq 0$

Thus, the sum of the amortized costs of n TableInsert operations is an upper bound on the sum of the actual costs.

Analysis

To analyze the amortized cost of the ith TableInsert operation, let

num_i denote the number of items stored in the table after the ith operation

$size_i$ denote the total size of the table after the ith operation

Φ_i denote the potential after the ith operation

Initially, $num_0 = 0$, $size_0 = 0$, and $\Phi_0 = 0$.

Consider two cases based on whether a table expansion is done.

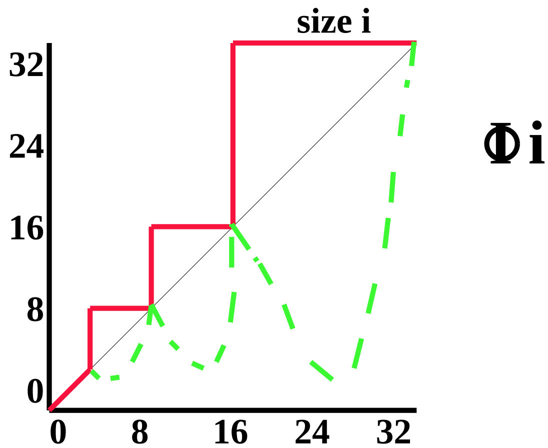
No expansion:

$$\hat{c}_i = c_i + \Phi_i - \Phi_{i-1}$$
$$= 1 + (2^*num_i\text{-}size_i) \text{ - } (2^*num_{i-1} \text{ - } size_{i-1})$$
$$= 1 + (2^*num_i\text{-}size_i) \text{ - } (2^*(num_i - 1) \text{ - } size_i)$$
$$= 1 \text{ - } (\text{-}2) = 3$$

Analysis

Expansion: $(size_i/2 = size_{i-1} = num_i - 1)$

$$\hat{c}_i = c_i + \Phi_i + \Phi_{i-1}$$
$$= num_i + (2^*num_i\text{-}size_i) \text{ - } (2^*num_{i-1} \text{ - } size_{i-1})$$
$$= num_i + (2^*num_i\text{-}(2^*num_i \text{ - } 2)) \text{ - } (2^*(num_i - 1) \text{ - } (num_i \text{ - } 1))$$
$$= num_i + 2 \text{ - } (num_i \text{ -}1)$$
$$= 3$$



Note how potential builds up to number of elements just before expansion.

Applications