Amortized Analysis

- Time required to perform a sequence of n operations to a data structure.
- In the sequence of operations, some cost a lot, some do not.
- Amortized cost is to calculate the total cost for a sequence of n operations, then take the average by dividing n.
- No probability involved, the average performace of each operation in the worst case.

Amortized Analysis Technique: Aggregate Method

- Aggregate method, determine an upper bound $T(n)$ for n operations. Amortized cost is $\frac{T(n)}{n}$.

Amortized Analysis Technique: Accounting Method

1. Determine an amrotized cost for each operation.
2. Some operations are over charged, “pre-paid cost”.
3. Prepaid credit should cover the cost for operations that are less charged.

Amortized Analysis: Potential Method

1. Similar to the accounting method, there are over charged operations.
2. The credit is maintained as “Potential Energy” associated with the data structure.
3. The data structure should always have positive potential.

Aggregate Method: Stack operation

- A stack, stores objects, operations are $\textsc{Push}(S, x)$, push x to the top of S, and $\textsc{Pop}(S)$, remove the top out of S.
- Any kind of implementation (linked list, array), $\textsc{Push}$ and $\textsc{Pop}$ take constant time. A sequence of $\textsc{Push}$es and $\textsc{Pop}$s take $O(n)$ time.
- What if there is $\textsc{Multipop}(S, k)$, remove the top k' objects, $k' = \min(k, s)$ where s is the number of objects S.

- A sequence of n PUSHes, POPs, MULTIPOPs take $O(n^2)$ time?
- Assume that we start with an empty stack, a better bound can be derived.
- Using aggregate method,
- there are at most $O(n)$ PUSHes, the number of POPs and MULTIPOPs can be only less than the number of PUSHes.
- So the total cost is $O(n)$, amortized cost is $O(n)/n = O(1)$.

Agregate Method: Incrementing a Binary Counter

- Implement a k-bit binary counter.
- Binary number stored in an array $A[0..k-1]$.
- length of $A[\ ]=k$, least significant bit $A[0]$, initially $x=0$, all 0s in the array.

- When incrementing, starting from the least significant bit, flip all consecutive 1's to 0's till a 0 is met. We then flip the 0 to 1.
- Cost for adding 1 = number of bits flipped. $\Theta(k)$ in the worst case since $A[0..k]$, total cost is $O(nk)$.

Accurate Analysis

There are n "add 1".
$A[0]$ flips each time,
$A[1]$ flips every other time $\lfloor \frac{n}{2} \rfloor$
$A[2]$ flips $\lfloor \frac{n}{4} \rfloor$
$A[3]$ flips $\lfloor \frac{n}{2^3} \rfloor$
$A[i]$ flips $\lfloor \frac{n}{2^i} \rfloor$

$$\sum \lfloor \frac{n}{2^i} \rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$

$2n/n = 2$ (The amortized cost).

The Accounting Method: Stack Operation

Give the amortized cost, PUSH=2, POP=0, MULTIPOP=0.
Example: Move plate to a stack and take a plate from the stack
Each operation will be paid for 1 dollar. So I prepay 2 dollars to the person move plates to the stack. He takes his one dollar and leaves one dollars on the plate. The person who moves the plate out of the stack takes that dollar.
Every one gets his one dollar, amortized cost is $O(1)$, so the total cost $< 2n$.

Accounting Method: Incrementing a Binary Counter

- Charge amortized cost 2 dollars to change from 0 to 1.
- One of the two dollars is for changing from 0 to 1 and the other is for changing back from 1 to 0 (to reset).
- The credit is always sufficient.
- So the amortized cost for incrementing operation takes at most 2 dollars.

The Potential Method

- In accounting method, the credit is associated with object.
- In the potential method, the credit is associated with the data structure as potential.
- Potential can be released to pay for future operations.
- We would like to have the potential to be non-negative all the way.

Potential Method

- Start with an initial data structure D_0
- n operations are performed
- for operations $i = 1, 2, \ldots, n$, c_i is the actual cost for operation i.
- D_i is the data structure after applying operation i to D_{i-1}.

Potential Method

- A potential function Φ maps each datastructure D_i to a real number $\Phi(D_i)$. $\Phi(D_i)$ the potential associated with the data structure D_i.
- Amortized cost $\hat{c}_i$ of the ith operation w.r.t. potential function Φ.

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

  actual cost (c_i) plus the increase in potential due to the operation.

Potential Method

- The total amortized cost of n operations is

$$\begin{aligned} \sum_{i=1}^{n} \hat{c}_i &= \sum_{i=1}^{n}(c_i + \Phi(D_i) - \Phi(D_{i-1})) \\ &= \sum_{n=1}^{n} c_i + \Phi(D_n) - \Phi(D_0) \end{aligned}$$

- if $\Phi(D_n) > \Phi(D_0)$ then total amortized cost $\sum_{i=1}^{n} \hat{c}_i$ is an upper bound.

Potential Method

- if $\Phi(D_i) > \Phi(D_0)$ $\forall i$ then we guarantee that the prepaid can cover the later.
- if $\Phi(D_0) = 0$, we only need to argue $\Phi(D_i) \geq 0$

Stack Operation

- Define Φ on a stack, “the number of objects in the stack”, $\Phi(D_i) \geq 0$.
- Start with an empty stack D_0, $\Phi(D_0) = 0$, the total amortized cost of n operations w.r.t. Φ is an upper bound on the actual cost.
- Now the amortized cost for various stack operations, assume that the i operation on a stack containing s objects,

PUSH :

- Potential Difference $\Phi(D_i) - \Phi(D_{i-1}) = (s+1) - s = 1$
- The amortized cost $\hat{c} = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1 + 1 = 2$

MULTIPOP:

- Multipop$(s.k)$ $k' = \min(s.k)$, k' objects are popped off the stack, the thte actual cost is k'.
- $\Phi(D_i) - \Phi(D_{i-1}) = -k'$
- $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = k' - k' = 0$
- Total amortized cost is no more than $2n$ that bounds the actual cost.

Incrementing a binary counter

The potential of the counter after the ith increment operation to be b_i, the number of 1's in the counter after the ith operation.
The ith increment reset t_i bits, actual cost is $t_i + 1$.
Number of 1's in the counter after the ith operation
$b_i = b_{i-1} - t_i + 1$
The Potential difference is
$\Phi(D_i) - \Phi(D_{i-1}) = (b_{i-1} - t_i + 1) - b_{i-1} = 1 - t_i$

Amortized cost

$$\begin{aligned}\hat{c} &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &\leq (t_i + 1) + (1 - t_i) \\ &= 2\end{aligned}$$

$\Phi(D_0) = 0$, why?? $\Phi(D_i) \geq 0$, $\forall i$, why??
Total amortized cost $2n$, bounds the total cost.

Dynamic Table

- Store objects into a table, but we don't know the number of objects will be inserted.
- When the table is full, we need to expand the table.
- When there are many free slots, we need to contract the table.
- The loading factor $\alpha(T)$ of the table, number of items in the table divided by the table size.
- the table can be a heap, an array or a hashing table.

Table Expansion

- When there is an insertion into a full table $\alpha(T) = 1$, we expand the table by allocating a new and larger table, then copying all slots into the new table.
- Assume allocating the new table doesn't take any time, thus expansion takes time of copying slots.

Table Expansion

Insert *i*th item

- Definition $size[T]$: size of the table, $num[T]$: number of items in the table.
- if $num[T] = size[T]$, we allocate a new table of size $= 2size[T]$, copy every item to the new table, then insert the *i*th item into the new table, otherwise we just insert the *i*th item.
- cost, if expansion, $size[T] + 1$; if no expansion, 1.
- Worst case, $O(n^2)$ for insertion n items??

Aggregate Method

Cost for insertion ith item denoted c_i,

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

Total cost

$$\begin{aligned} \sum_{i=1}^{n} c_i &\leq n + \sum_{j=0}^{\lfloor \lg n \rfloor} 2^j \\ &\leq n + 2n \\ &= 3n \end{aligned}$$

This amortized cost is 3.

Accounting Method

- Each insertion has pay 3 dollars.
- 1 dollar for the insertion, 1 prepaid dollar for moving itself in the next expansion, the last prepaid dollars is to help the other one (one of the "seniors") in the next expansion.
- supose the table size is m immediately after expansion, and the table has no credit.
- Pay for any insertion 3 dollars, one for the insertion, one for moving itself next time, and one for helping one of the item already in the table.
- In the next expansion, every one has prepaid money for moving.
- after moving, there are no more credit.

Potential Method

- $\Phi(T) = 2 \cdot num[T] - size[T]$.
- $\Phi(T)$=0 after expansion.
- Table is always at least half-full, so $\Phi(T)$ always greater than 0. Thus sum of amortizeed cost is the upper bound for the actual cost.
- num_i = number of items after ith insertion. $size_i$ = size of the table after insertion. Φ_i the potential after i insertion.

If an insertion does not trigger expansion, the $size_i = size_{i+1}$

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (2 \cdot num_i - size_i) - (2 \cdot num_{i-1} - size_{i-1}) \\
&= 1 + (2 \cdot num_i - size_i) - (2(num_i - 1) - size_i) \\
&= 3
\end{aligned}$$

If the ith operation trigger expansion, the $size_i/2 = size_{i-1} = num_i - 1$.

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= num_i + (2 \cdot num_i - size_i) - (2 \cdot num_{i-1} - size_{i-1}) \\
&= num_i + (2 \cdot num_i - (2num_i - 2)) - (2(numn_i) - (num_i - 1)) \\
&= num_i + 2 - (num_i - 1) \\
&= 3.
\end{aligned}$$

Table Expansion and Contraction

- What if there are deletion?
- Contraction when the loading factor is low.
- If we contract the table when the number of objects is less than half of the table size, loading factor is no less than $1/2$.
- There exists a sequence of operations that total cost is $\Theta(n^2)$.

Table Expansion and Contraction

- $n/2$ insertions, then perform the following operations
- I, D, D, I, I, D, D, I, I, D, D, ...
- After expansion, we don't have enough deletions to pay for the contraction.
- After contraction, we don't have enough insertions to pay for the expansion.

Table Expansion and Contraction

- Expansion: double the size of the table when insertion to a full table.
- Contraction: halve the size of the table when a deletion causes the loading factor below $1/4$.
- Use potential method to show the amortized cost is constant.
- $\alpha[T] = \frac{num[T]}{size[T]}$.

Table Expansion and Contraction

The Poteitial function

$$\Phi(T) = \begin{cases} 2 \cdot \mathit{num}[T] - \mathit{size}[T] & \text{if } \alpha(T) \geq 1/2, \\ \mathit{size}[T]/2 - \mathit{num}[T] & \text{if } \alpha(T) < 1/2. \end{cases}$$

- When the loading factor is $1/2$, potential $= 0$.
- loading factor $= 1$, $\mathit{size}[T] = \mathit{num}[T]$, poteintial $= \mathit{num}[T]$, poteintial can pay for an expansion.
- loading factor $= 1/4$, $\mathit{size}[T] = 4 \cdot \mathit{num}[T]$, potential $= \mathit{num}[T]$, can pay for a contraction.

Consider the cases

1. *i*th operation is insertion
   - loding factors α_{i-1}, α_i, are greater than, equal to, less than $1/4$, $1/2$, does *i*th operation triger an expansion?
2. *i*th operation is deletion
   - loding factors α_{i-1}, α_i, are greater than, equal to, less than $1/4$, $1/2$, does *i*th operation triger a contraction?
3. Consider the relationships between $size_i$, num_i, $size_{i-1}$, and num_{i-1}.

ith op is Insertion

- if $\alpha_{i-1} \geq \frac{1}{2}$, analysis is identical to the table expansion. $\hat{c}_i = 3$.
- if $\alpha_{i-1} < \frac{1}{2}$ and $\alpha_i < \frac{1}{2}$

$$
\begin{aligned}
\hat{c}_i &= c_i + \Phi_i + \Phi_{i-1} \\
&= 1 + (\frac{size_i}{2} - num_i) - (\frac{size_{i-1}}{2} - num_{i-1}) \\
&= 1 + (\frac{size_i}{2} - num_i) - (\frac{size_i}{2} - num_i - 1) \\
&= 0.
\end{aligned}
$$

ith op is Insertion

if $\alpha_{i-1} < \frac{1}{2}$ but $\alpha_i \geq \frac{1}{2}$

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (2 \cdot num_i - size_i) - (\frac{size_{i-1}}{2} - num_{i-1}) \\
&= 1 + (2(num_{i-1} + 1) - size_{i-1}) - (\frac{size_{i-1}}{2} - num_{i-1}) \\
&= 3 \cdot num_{i-1} - \frac{3}{2} size_{i-1} + 3 \\
&= 3\alpha_{i-1} size_{i-1} - \frac{3}{2} size_{i-1} + 3 \\
&< \frac{3}{2} size_{i-1} - \frac{3}{2} size_{i-1} + 3 \\
&= 3.
\end{aligned}$$

ith op is Deletion

If $\alpha_{i-1} < \frac{1}{2}$, and if it does not trigger a contraction

$$\begin{aligned}
\hat{c}_i &= c_i - \Phi_i - \Phi_{i-1} \\
&= 1 + (\frac{size_i}{2}) - (num_i) - (\frac{size_i}{2} - num_{i-1}) \\
&= 1 + (\frac{size_i}{2} - num_i) - (\frac{size_i}{2} - (num_i + 1)) \\
&= 2.
\end{aligned}$$

*i*th op is Deletion

If $\alpha_{i-1} < \frac{1}{2}$, and if it does trigger a contraction
The we have $\frac{size_i}{2} = \frac{size_{i-1}}{4} = num_i + 1$.

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= (num_i + 1) + (\frac{size_i}{2} - num_i) - (\frac{size_{i-1}}{2} - num_{i-1}) \\
&= (num_i + 1) + ((num_i + 1) - num_i) - \\
&\quad ((2 \cdot num_i + 2) - (num_i + 1)) \\
&= 1.
\end{aligned}$$