Amortized Analysis

DistributeMoney(n, k)

1 **Each of** n **people gets \$1.**
2 **for** $i = 1$**to** k
3 **do Give a dollar to a random person**

What is the maximum amount of money I can receive?

Amortized Analysis

DistributeMoney(n, k)

1 **Each of** n **people gets \$1.**
2 **for** $i = 1$**to** k
3 **do Give a dollar to a random person**

What is the maximum amount of money I can receive?

- **Worst case analysis.** **Each round, I might get** n **dollars, there are** k **rounds, so I receive at most** nk **dollars.**

Amortized Analysis

```
DistributeMoney(n, k)
Each of n people gets $1.
for i = 1 to k
        do Give a dollar to a random person
```

What is the maximum amount of money I can receive?

- **Worst case analysis.** Each round, I might get n dollars, there are k rounds, so I receive at most nk dollars.
- **Amortized lesson.** Sometimes a standard worst case analysis is too weak. It doesn't take into account (worst-case) dependencies between what happens at each step.

An example we have already seen

- Building a heap in heapsort.
  - Each insert takes $O(\lg n)$ time.
  - Insert n items
  - Total of $O(n \lg n)$ time.
- Buildheap – While any one insert may take $\lg n$ time, when you do a sequence of n of them, bottom up, you can argue that the whole sequence takes $O(n)$ time.

Amortized Analysis

```
Multipop(S, k)
  while not Stack-Empty(S) and k ≠ 0
        do Pop(S)
           k ← k − 1
```

Some Analysis

- **Push –** $O(1)$ **time**
- **Pop –** $O(1)$ **time.**
- **Multipop(k) –** $O(k)$ **time.**

Analysis

- **Each op takes** $O(k)$ **time.**
- $k \leq n$ **, so each op takes** $O(n)$ **time**
- n **operations take** $O(n^2)$ **time.**

Can you construct a sequence of n **operations that take** $\Omega(n^2)$ **time?**

The right approach

Claim Starting with an empty stack, any sequence of n Push, Pop, and Multipop operations take $O(n)$ time.

- We say that the **amortized** time per operation is $O(n)/n = O(1)$.
- 3 types of amortized analysis
  - Agggretate Analysis
  - Banker's (charging scheme) method
  - Physicist's (potential function) method

Aggregate Analysis

- **Call Pop - multipop(1)**
- **Let** $m(i)$ **be the number of pops done in the** i **th multipop**
- **Let** p **be the number of pushes done overall.**

Claim

$$\sum_i m(i) \leq p$$

Anlysis

$$\begin{aligned} \textbf{total time} &= \textbf{pushes} + \textbf{time for all multipops} \\ &= p + \sum_i m(i) \\ &\leq p + p \\ &= 2p \\ &\leq 2n \end{aligned}$$

Banker's Method

- **Each operation has a real cost c_i and an amortized cost $\hat{c}_i$.**
- **The amortized costs as valid if :**

$$\forall \ell \quad \sum_{i=1}^{\ell} \hat{c}_i \geq \sum_{i=1}^{\ell} c_i.$$

Methodology

- **Show that the amortized costs are valid**
- **Show that $\sum_{i=1}^{\ell} \hat{c}_i \leq X$, for some X .**
- **Conclude that the total cost is at most X .**

Why is the conclusion valid?

$$\sum_{i=1}^{\ell} c_i \leq \sum_{i=1}^{\ell} \hat{c}_i \leq X.$$

Important: **Your work is to come up with the amortized costs and to show that they are valid.**

Banker's Method for Multipop

	Real Cost c_i	Amortized cost $\hat{c}_i$
Push	**1**	**2**
Pop	**1**	**0**
Multipop(k)	**k**	**0**

Potential Function Method

- **Let D_i be the "state" of the system after the i th operation.**
- **Define a potential function $\Phi(D_i)$ to be the potential associated with state D_i .**
- **The i th operation has a real cost of c_i**
- **Define the amortized cost $\hat{c}_i$ of the i th operation by**

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

Why are we bothering?

- **The amortized costs give us a nicer way of analyzing operations of varying real cost (like multipop)**
- **We use the potential function to "smooth" out the difference**

First, the math

Potential function

- **Let** D_i **be the "state" of the system after the** i **th operation.**
- **Define a potential function** $\Phi(D_i)$ **to be the potential associated with state** D_i **.**
- **The** i **th operation has a real cost of** c_i
- **Define the amortized cost** $\hat{c}_i$ **of the** i **th operation by**

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

$$\begin{aligned}
\sum_{i=1}^{n} \hat{c}_i &= \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1})) \\
&= \left(\sum_{i=1}^{n} c_i\right) \\
&\quad + (\Phi(D_1) - \Phi(D_0)) + (\Phi(D_2) - \Phi(D_1)) + \ldots + (\Phi(D_{n-1}) - \Phi(D_{n-2})) + (\Phi(D_n) - \Phi(D_{n-1} \\
&= \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0)
\end{aligned}$$

Potential function

- Let D_i be the "state" of the system after the i th operation.
- Define a potential function $\Phi(D_i)$ to be the potential associated with state D_i .
- The i th operation has a real cost of c_i
- Define the amortized cost $\hat{c}_i$ of the i th operation by $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$
- Summing, we have $\Sigma_{i=1}^{n} \hat{c}_i = \Sigma_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0)$.

Using this

- Suppose that $\Phi(D_n) \geq \Phi(D_0)$.
- Then $\Sigma_{i=1}^{n} \hat{c}_i \geq \Sigma_{i=1}^{n} c_i$
- Next suppose that we have an upper bound X on $\Sigma_{i=1}^{n} \hat{c}_i$.
- Putting it all together we have

$$X \geq \sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$$

Conclusion: X is an upper bound on the real cost.

Using this method

- **Choose an appropriate potential function** Φ
- **Show that** $\Phi(D_0) = 0$
- **Show that** $\Phi(D_n) \geq 0$
- **Given an upper bound of** X **on** $\sum_{i=1}^{n} \hat{c}_i$ **.**
- **Declare victory and celebrate, secure in the knowledge that your real cost for any** n **operations is upper bounded by** X

Applying the Method to Multipop

- **Choose $\Phi(D_i)$ to be the number of items on the stack after the i th operation.**
- **Clearly,**
  - $\Phi(D_0) = 0$ **because initial stack is empty**
  - $\Phi(D_n) \geq 0$ **because Φ is always non-negative.**
- **Now let's compute amortized cost of each operation.**

Applying the Method to Multipop

- **Choose $\Phi(D_i)$ to be the number of items on the stack after the i th operation.**

Push: $\Phi(D_i) - \Phi(D_{i-1}) = 1$
So

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1 + 1 = 2$$

Pop: $\Phi(D_i) - \Phi(D_{i-1}) = -1$
So

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1 - 1 = 0$$

MultiPop of k items: $\Phi(D_i) - \Phi(D_{i-1}) = -k$
So

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = k - k = 0$$

Concluding

- **For any operation** $\hat{c}_i \leq 2$ **.**
- **So for any** n **operations,** $\sum_{i=1}^{n} \hat{c}_i \leq 2n$ **.**
- **Concluding, this means that for any** n **operations,** $\sum_{i=1}^{n} c_i \leq 2n$ **.**

Binary Counter

Increment(A)

```
i ← 0
while i < length[A] and A[i] = 1
      do A[i] ← 0
         i ← i + 1
if i < length[A]
   then A[i] ← 1
```

Question: **How many times is a bit flipped, while doing n increments on a k bit counter?**

Example of a 4 bit counter

Bits	# of bits flipped
0000	
0001	1
0010	2
0011	1
0100	3
0101	1
0110	2
0111	1
1000	4
1001	1
1010	2
1011	1
1100	3
1101	1
1110	2
1111	1
0000	4

Is there some structure here?

Example of a 4 bit counter

Bits	# of bits flipped	number of new 1's
0000		
0001	1	1
0010	2	1
0011	1	1
0100	3	1
0101	1	1
0110	2	1
0111	1	1
1000	4	1
1001	1	1
1010	2	1
1011	1	1
1100	3	1
1101	1	1
1110	2	1
1111	1	1
0000	4	0

Is there some structure here? The number of new 1's is at most 1. Can we charge new 0's to new 1's?

Example of a 4 bit counter

Bits	# of bits flipped	number of new 1's
0000		
0001	1	1
0010	2	1
0011	1	1
0100	3	1
0101	1	1
0110	2	1
0111	1	1
1000	4	1
1001	1	1
1010	2	1
1011	1	1
1100	3	1
1101	1	1
1110	2	1
1111	1	1
0000	4	0

Is there some structure here? The number of new 1's is at most 1. Can we charge new 0's to new 1's?

Example of a 4 bit counter

Bits	# of bits flipped	number of new 1's
0000		
0001	1	1
0010	2	1
0011	1	1
0100	3	1
0101	1	1
0110	2	1
0111	1	1
1000	4	1
1001	1	1
1010	2	1
1011	1	1
1100	3	1
1101	1	1
1110	2	1
1111	1	1
0000	4	0
TOTAL	30	15

Is there some structure here? The number of new 1's is at most 1. Can we charge new 0's to new 1's? Seem to be twice as many flips as switches from 0 to 1.

Banker's Analysis

- For each increment, pay \$1, and leave \$1 to pay for the flip back to 0. amortized cost of 2.
- Number of flips to 0 $\leq$ number of flips to 1.
- Always sufficient money in the bank.
- Amortized cost is therefor valid.
- Total of n cost for n operations.
- Independent of k !!

Potential Function

Definitions

- f_{01} is the number of bits flipped from 0 to 1 .
- f_{10} is the number of bits flipped from 1 to 0 .
- Potential function $\Phi(D_k)$ is the number of 1 's in the current counter state.

First check that potential function is valid

- $\Phi(D_0) = 0$, since the initial state is 0
- $\Phi(D_i \geq 0)$ always.

Now compute amortized cost

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\
&= (f_{01} + f_{10}) + (f_{01} - f_{10}) \\
&= 2f_{01} \\
&\leq 2 \cdot 1 \\
&= 2
\end{aligned}$$

So the amortized cost is 2.

Note that when there is wraparound the cost is actually 0, every other time it is 2.

Aggregate Analysis

- **Look at the columns of the example and count how many times there is a flip in each column.**
- **Last column –** n
- **Penultimate column –** $n/2$
- **. . .**
- **First column –** $n/2^k$

Total flips

$$n + n/2 + n/4 + \cdots + n/2^k \leq n + n/2 + n/4 + \cdots \leq 2n$$

Table Insert

Table-Insert(T, x)

if $size[T] = 0$
    then allocate $table[T]$ **with** 1 **slot**
        $size[T] \leftarrow 1$
if $num[T] = size[T]$
    then allocate $new\text{-}table$ **with** $2 \cdot size[T]$ **slots**
        insert all items in $table[T]$ **into** $new\text{-}table$
        free $table[T]$
        $table[T] \leftarrow new\text{-}table$
        $size[T] \leftarrow 2 \cdot size[T]$
insert x **into** $table[T]$
$num[T] \leftarrow num[T] + 1$

A potential function for table insert

Real cost

$$c_i = \begin{cases} i & \textbf{if } i-1 \textbf{ is a power of 2} \\ 1 & \textbf{otherwise} \end{cases}$$

Potential function

- $\Delta\Phi$ **should be constant for a normal insert**
- $\Delta\Phi$ **should drop by about** i **for an expensive insert.**

$$\Phi(T_i) = 2\,num(T_i) - size(T_i)$$

Analysis

$$\Phi(T_i) = 2\,num(T_i) - size(T_i)$$

Analysis **Case 1: No table doubling (** $num_i = num_{i-1} + 1$ **,** $size_i = size_{i-1}$ **)**

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + 2\,num_i - size_i - (2\,num_{i-1} - size_{i-1}) \\
&= 1 + 2(num_i - num_{i-1}) - (size_i - size_{i-1}) \\
&= 1 + 2(1) - 0 \\
&= 3
\end{aligned}$$

Case 2: Table doubling ($num_i = num_{i-1} + 1$ **,** $size_i = 2 * size_{i-1}$ **)**

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

So any n **operations take at most** $3n$ **time.**