6. Amortized Analysis

What is amortized analysis?

- It is:
  - a collection of techniques we can use to analyze the time complexity of a **sequence of operations**.
    - Operations on a data structure.
    - Operations performed by an algorithm.
- It is not:
  - a way to prove a good upper bound on the worst case running time of a single operation.
  - a technique used to design an algorithm or data structure.

What is amortized analysis?

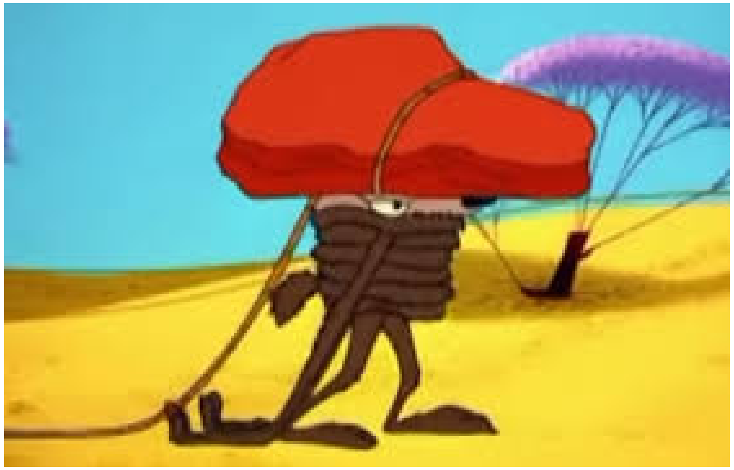
- Intuitively, we have a situation where:
  - most operations are cheap
  - but some can be very expensive.
- However we can only execute an expensive operation if we first did a lot of cheap operations.
- Then
  - we can prove a good upper bound on the total running time of the sequence of operations
  - because the many cheap operations “paid” for the one expensive operation.

What is amortized analysis?

- There are several methods that together are called amortized analysis
  - the aggregate method
  - the accounting method
  - the charging method
  - the potential method
- We will only look at the last one.
  - It's more general than the others.
  - It's sometimes a bit harder to use (maybe).

The potential method

- The idea for the potential method is derived from physics:
  - If I slowly lift a boulder then
    - I do some (a lot of) work, a little bit at a time.
    - I increase the boulder's potential energy each time.
  - That energy can then be used later (poor Wile E. Coyote).



The potential method

- You can think of potential as an account at a local store.
  - When you buy cheap items:
    - You pay a bit more than the items are worth and build “credit” (the potential increases).
  - When you buy an expensive item:
    - You use some of the accumulated credit (the potential decreases).
    - And so you still only pay a bit of money.
  - It’s easier to keep track of the sum of your payments than of the sum of the item costs.

The potential method

- We define a potential function $\Phi(D_i)$ where D_i is
  - the data structure
  - or the state of the algorithm

  after i operations.
- We define Φ such that
  - $\Phi(D_i) \geq 0$
  - $\Phi(D_0) = 0$

The potential method

- Examples:
  - $\Phi\left(D_i\right)$ is the number of elements of D_i.
  - $\Phi\left(D_i\right)$ is the number of 1 bits in a binary counter.
- When we perform the i^{th} operation op_i,
  - it takes $cost_{real}(op_i)$ steps.
  - it may change the potential of the data structure, that is, $\Phi\left(D_i\right)$ may be different from $\Phi\left(D_{i-1}\right)$.
  - the cost $cost_{real}(op_i)$ may vary unpredictably.
    - *That's the real price of the object you are buying.*

The potential method

- We define the *amortized cost* of operation op_i by
  - $cost_{am}(op_i) = cost_{real}(op_i) + \Phi\left(D_i\right) - \Phi\left(D_{i-1}\right)$
- With a well chosen potential function, the costs $cost_{am}(op_i)$ are
  - relatively consistent
  - easy to compute
  - *$cost_{am}(op_i)$ is the amount you're actually paying.*

The potential method

- Why is this useful?
  - We can prove that the cost of a sequence of n operations on the data structure is at most:

$$\sum_{i=1}^{n} cost_{am}(op_i)$$

  - That is,

$$\sum_{i=1}^{n} cost_{real}(op_i) \leq \sum_{i=1}^{n} cost_{am}(op_i)$$

  - The sum on the right is easy to compute.