Lecture 13, Oct 23, 2023

Amortized Complexity

- Given a data structure with some operation, $T(n)$ is the worst case of doing *any* sequence of n operations, over all possible such sequences
- $\frac{T(n)}{n}$ is the "average" cost of an operation in the worst case
  - This is referred to as the *amortized cost* of an operation
- There are two ways to compute the amortized cost:
  - Aggregate analysis: computing $T(n)$ directly by summing the cost of n operations
    - Try to do a sequence of operations and see if we can find a periodic pattern
  - Accounting method: "charging" some amount per operation (which can be more or less than the actual cost of that operation), so that the overcharge is used as "credit" towards future more expensive operations that are undercharged
    - We require that the total charges on any sequence of n operations is greater than or equal to the actual cost of doing these operations; this is the *credit invariant*
- Example: incrementing a k-bit binary counter
  - The counter starts at $A = 0$
  - Cost is the number of bits that get flipped with each increment
  - $T(n)$ is then the total number of bits flipped by a sequence of n increments
  - Solve by aggregate analysis:
    - We notice a pattern: bit 0 is flipped on every increment, bit 1 is flipped every other increment, bit 2 is flipped every 4 increments and so on
    - Given n increments, bit i is flipped $\left\lfloor \frac{n}{2^i} \right\rfloor$ times
    - The total cost is $\sum_{i=0}^{k-1} \left\lfloor \frac{n}{2^i} \right\rfloor < \sum_{i=0}^{k-1} \frac{n}{2^i} = 2n$
    - Therefore the amortized cost is $\frac{T(n)}{n} \leq 2$, or $\Theta(1)$
  - Solve by the accounting method:
    - Notice that every increment will do some sequence of flipping 1s to 0s, and one single flip from 0 to 1
      - After the flip from 0 to 1 we no longer have a carry so the flipping stops
    - Whenever we flip from 0 to 1, we can charge for 2 flips, one for the actual operation and one more for the eventual flip from 1 back to 0
      - We attach the overcharge (credit) to the 1 that we just created
      - Next time when we need to flip the 1 back to a 0, we can use the credit that we attached to the 1 instead
      - This maintains the credit invariant, since we start the counter at 0
    - At any point the total credit we have is equal to the number of 1s in the counter; therefore whenever we do an increment, we can use the existing credit to flip all the bits from 1 to 0; this maintains the credit invariant
    - Since we charge 2 for each operation, the amortized cost is 2, or $\Theta(1)$