Amortized Analysis 09.25

Sometimes we use an algorithm (and its accompanying DS) just once to solve a problem

eg "find gcd of 1096 and 128"

Often, however, a DS has a "lifetime" and queries and modifications are executed on the DS.

Eg database
network

We have studied worst-case (and perhaps average-case) running times of a single application of an algorithm, but it makes sense sometimes to widen our lens and look at running times over the lifetime of the DS.

Eg Stack

push(x) — pushes x onto top of stack

pop() — removes top element from stack

top() — returns value at top of stack.

If we use the array implementation of the stack, the running times are:

Now suppose you need a new operation

```
multipop (k)
    while !stackempty() and k > 0
        pop()
        k--
```

The running time for multipop is clearly $\Theta(k)$

if $0 \leq k \leq n$, where n is # elements ever pushed.

But let us consider the problem thusly:

"given a sequence of n push, pop, multipop ops, what is the worst case running time of the sequence?"

The result is divided by the number of ops to yield the AMORTIZED ANALYSIS

Facile analysis:

- each op is $\in O(n)$ ← max # pushes is n so $\exists \leq n$ elements
- $\exists$ n operations

$\Rightarrow$ running time is $O(n^2)$

$\Rightarrow$ amortized analysis is $O(n)$ per op.

Aggregate Analysis

Total # of pushes is $\leq n$

Total # of effective pops including the pops in the multipops $\leq$ total # pushes $\leq n$

Observe: "work done" in the n operations is + constant amount per opn + total amount of effective pops $\leq n \leq$ # pushes $\leq n$

$\leq 2n$

$\therefore$ amortized running per operation is $\Theta(1)$

Accounting Method of Amortized Analysis

$\forall$ operation is "paid for" in the currency of the analysis $\equiv$ time

But you can run up a deficit or a credit, like a bank

... you don't have to pay for the work exactly when it is done ... but payment must reflect actual running time (work done).

Eg multipop stack:

∀ push, "pay for" the push and the eventual pop

push(x) — pay 1 credit for the actual work of the push

leave one credit on element (payment in advance)

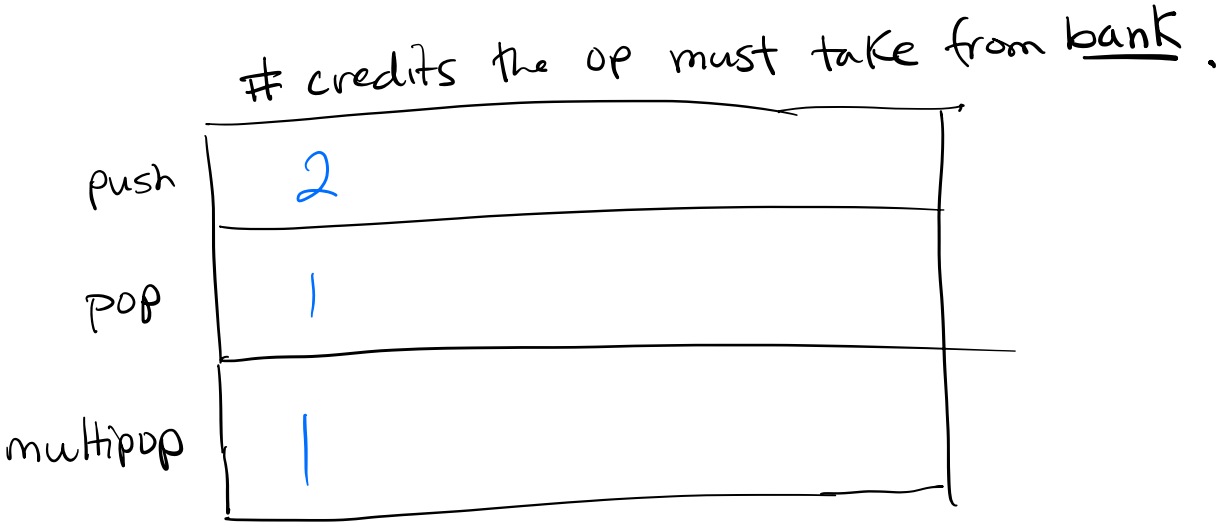
} 2 credits

multi pop() — pay 1 credit for the actual work entering and leaving the code.

If not empty, use credit on top of each element to pay for the work of popping the element (if necessary)

} 1 credit

principle: 1 credit only ever pays for a constant amount of work.



$\Rightarrow$ Running time is 1 or 2 credits $= \Theta(1)$ work per operation, amortized over the run of the sequence of operations.

i.e. - n ops takes $O(n)$ time

- each op takes $O(1)$ amortized time.

Potential Function Method of Amortized Running-time Analysis.

- represents "prepaid work" as "potential energy" (or just "potential") that can be released later to pay for future work.

Φ – maps state of the data structure to a real number

Each operation i does an amount of work c_i and may also result in a change in the state of the DS; $\Phi_i - \Phi_{i-1} = \Delta\Phi$