Amortized Analysis

Data Structure S
basic operations.

Sequence of operations

Total time for a sequence of operations

Stack with multi-pop
stack
- push(S, x)
- multipop(S, k)
  while $k > 0$ do
    pop(S)
    $k := k - 1$
  end while

Start with empty stack,
n operations in sequence
How much total time in worst case for the sequence?

Straightforward way:
stack has $\leq n$ items at any time
push takes $O(1)$ time
multipop(S, k) takes $\Theta(k)$ i.e. $O(n)$ time

Upper bound on w.c. time for a sequence is $n \cdot O(n) = O(n^2)$

Better analysis
- Aggregate analysis
- Accounting method
- Potential method

Each item is
pushed at most once
then (maybe) popped or part of a multipop.

items $\leq$ # pushes $\leq n$
when pushed — 1 unit of time
when popped — 1 unit of time
$\therefore$ total time to process item x is 2 units
$2 \times n$ items $= O(n)$ total time.
Worst-case time for sequence of n ops on an initially empty stack is $O(n)$, or
$O(1)$ per operation, averaged over the sequence.

Binary Counter with increment.
$A[0..(k-1)]$ of bits
$A[0]$ has l.s.b.
stores a number i from 0 to $2^k - 1$.
Assume counter i initially 0
$A[0] = \cdots = A[k-1] = 0$

basic op: increment

increment


```
j := 0
while A[j] == 1 do
    A[j] := 0
    j++
end while
A[j] := 1
```

increment takes k units of time worst case.

Sequence of n increments, starting at 0:
time $O(nk)$ (unamortized worst-case analysis)

Aggregate analysis

$A[0]$ is flipped n times
$A[1]$ is flipped $\doteq \frac{n}{2}$ times
$A[2]$ is flipped $\doteq \frac{n}{4}$ times
$A[j]$ " n $\geq \frac{n}{2^j}$ times

Total time

$$= \sum_{i=0}^{k-1} 1 \cdot \left(\frac{n}{2^i}\right) = n \sum_{j=0}^{k-1} 2^{-j}$$

(time for flip; # times flipped)

$$\leq n \sum_{j=0}^{\infty} 2^{-j} = 2n$$

$\therefore$ total time for n increments is $O(n)$, or $O(1)$ per increment, on average

Accounting / Potential method

Nov 1-12:29 PM

Bank account (no negative balance)
balance at any given time is also called the potential

Assume a sequence $S_1, S_2, \ldots, S_n$ of operations on some data structure D
Let Φ_i be the potential after operation S_i
(Φ_i depends on the application at hand). Such that:
1. Φ_0 = initial potential before S_1 = 0.
2. For all $1 \le i \le n$, $\Phi_i \ge 0$

Let c_i be the actual cost (in time) of operation S_i.
Define amortized cost of S_i as:

$$\hat{c}_i := c_i + (\Phi_i - \Phi_{i-1})$$

(c_i: actual cost, >0; $\Phi_i - \Phi_{i-1} = \Delta\Phi$: change of potential, >0, =0, <0)

Idea: define Φ_i in such a way that $\hat{c}_i$ is small, for all i

Note:
Total time (actual)

$$= \sum_{i=1}^{n} c_i$$

Total amortized time

$$= \sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \Phi_i - \Phi_{i-1})$$

$$= \sum_{i=1}^{n} c_i + \sum_{i=1}^{n} (\Phi_i - \Phi_{i-1})$$

$$= \sum_{i=1}^{n} c_i + \Phi_n - \underbrace{\Phi_0}_{0}$$

$$= \sum_{i=1}^{n} c_i + \Phi_n$$

$$\ge \sum_{i=1}^{n} c_i$$

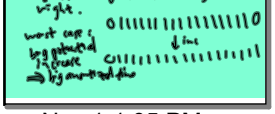
$\therefore$ Total amortized time $\ge$ Total actual time.
(shall for each i)

Stack with multipop
Define Φ_i = size of the stack after the ith operation

$\Phi_0 = 0$
$\Phi_i \ge 0$ for all i
Now compute $\hat{c}_i$:
S_i is a push $\Rightarrow \hat{c}_i = c_i + 1 = 1+1 = 2$
$\hat{c}_i = 2$
S_i is pop(S,k)
$\hat{c}_i = c_i - k = k - k = 0$
So: $\hat{c}_i \le 2$
so total time $\le$ total amortized time $\le 2n$

Potential for binary counter with increment:
Define Φ_i = # of 1's in A[0..(i-1)]
$\Phi_1 = 0$
$\Phi_i \ge 0$
increment with j many carries
actual time = j+1
$\Phi_{after} - \Phi_{before} = 1 - j$
$\hat{c}_i = (j+1) + (1-j) = 2$
Total time for n increments is $\le 2n$

Proposed potential function:
# of contiguous 1's counting from right.



worst case: big potential to begin with ⇒ big amortized time

Nov 1-1:05 PM

Resizable arrays

$A[1..k]$ k = capacity

if overflow:

- allocate a bigger array (assume constant)
- copy items from old array into new array
- deallocate old array (negligible time)

Want size of current array to be $O(\#\text{items stored})$

Assume unit cost for insertion

$k = 1$ initially

What is a good strategy for resizing?

Bad strategy: increase size from k to $k+1$ on overflow

Good strategy: size $k \to 2k$ on overflow.

assume this takes k units of time (resize)

cost 1 per insertion

want $\geq k$ units in bank at resize time

decrease potential by k at resize

charge 2 extra units per insertion so

$\hat{c}_{insert} = 1 + 2 = 3$

$\hat{c}_{resize} = k - k = 0$

} constant amortized time

Nov 1-1:33 PM