Book: Longest common subsequence
Matrix chain order
Rod cutting

Amortized Analysis

Data structure with basic operations

Each basic op may have a worst-case run-time bound, but this may not be appropriate.

Amortized analysis gives a bound on the time taken by a sequence of basic ops (some cheap, some expensive), but average time per op may be modest.

3 methods:

- Aggregate method — trace work done on an item in the structure.
- Cost-accounting method — put a surcharge on cheap operations to "pay" for future expensive operations. [practically indistinguishable from the ...]
- Potential Method — we will use this one almost exclusively, define a potential function Φ which returns a real number given any particular state of the data structure.

amortized cost = (actual cost) + $\Delta\Phi$

Example: Stack with multipop:

Basic operations:

Push(x) — push an item x onto the stack

Pop(k) — pop k items off the stack (but stop if stack becomes empty)

Top() — return top item on stack (stack is unchanged)

Implement as simple linked list (front is the top)

Cost per operation:

Push — $\Theta(1)$

Pop(k) — $\Theta(k)$ [worst case]

Top — $\Theta(1)$

Sequence of n operations:

Non-amortized analysis is $\Theta(n^2)$ (worst case)

stack has $\le n$ items but Pop(n) takes time $\Theta(n)$ (worst-case)

Time = (# ops) · (time for most expensive of these: Pop(n))

$= \Theta(n^2)$

Actual upper bound (tight) is $\Theta(n)$ for any sequence of ops.

IE, $\Theta(1)$ time/op on average

Show using the potential method

Potential fcn

$\Phi(S) = $ # items in S (S = stack)

Let c_i be the actual cost of the i'th op $(1 \le i \le n)$

Let

$\hat{c}_i$ be the amortized cost of the i'th operation.

Let S_i be the stack after the i'th operation. Then

$$\hat{c}_i := c_i + \underbrace{\Phi(S_i) - \Phi(S_{i-1})}_{\Delta\Phi}$$

Assume Push, Top take time 1 & Pop(k) takes time min(k, stack size) [or assume $k \le$ stack size always]

Determine $\hat{c}_i$:

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

If i'th op is

- Push(x):
  $\hat{c}_i = 1 + 1 = 2$
- Top():
  $\hat{c}_i = 1 + 0 = 1$
- Pop(k): [$k \le$ stack size]
  $\hat{c}_i = k - k = 0$

In all cases: $\hat{c}_i = \Theta(1)$

Letting S_0 be the initial state of the structure (empty stack in static case)

The potential function must satisfy:

$\Phi(S_0) = 0$

$\forall i: \Phi(S_i) \ge 0$

Given such a potential,

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

Prop: $\sum_{i=1}^{n} c_i \le \sum_{i=1}^{n} \hat{c}_i$

Proof:

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} \left(c_i + \Phi(S_i) - \Phi(S_{i-1})\right)$$

$$= \sum_{i=1}^{n} c_i + \sum_{i=1}^{n} \left(\Phi(S_i) - \Phi(S_{i-1})\right)$$

$$= \sum_{i=1}^{n} c_i + \Phi(S_n) - \underbrace{\Phi(S_0)}_{0}$$

$$= \sum_{i=1}^{n} c_i + \underbrace{\Phi(S_n)}_{\ge 0} \ge \sum_{i=1}^{n} c_i$$

Cost of any sequence of n ops starting with an empty stack is

$$\sum_{i=1}^{n} c_i \le \sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} O(1)$$

$$= O(n)$$

Think of the potential function as a bank balance —
- cheap ops can be surcharged to build up the potential
- when an expensive op is needed, potential is high enough so that it can drop to cover the cost of the expensive op without going negative.

Ex: (Resizeable) array
Dynamic

Array of items supporting
InsertAtEnd(x) — insert at end $O(1)$
Return(k) — return k'th item in the array.

Let A be the array.
$A.size$ = the # of entries of the array
$A.num$ = the # of items in the collection
$A.num \le A.size$ always.

If $A.num = A.size$, then InsertAtEnd must trigger a resize: allocate an array twice the size; move everything into new array, then do insertion at end

InsertAtEnd(x) takes time
$\begin{cases} 1 & \text{if no resize} \\ A.size + c & \text{if resize} \end{cases}$
c constant

Show that amortized cost per op is $O(1)$.

Potential function? Let
A_i be the state of the structure after the ith op.
A_0 — initial state.
$A_0.size = 1$, $A_0.num = 0$

$$\Phi(A) := 2\,A.num - A.size + 1$$

Check:
$$\Phi(A_0) = 2\,A_0.num - A_0.size + 1$$
$$= 2 \cdot 0 - 1 + 1 = 0$$

$\forall i \ge 1$
$$\Phi(A_i) = 2A_i.num - A_i.size + 1$$
≥ 0 because (except at beginning), $A_i.num \ge \frac{1}{2} A_i.size$

$\therefore$ this is a legitimate potential fcn.

Computing $\hat{c}_i$:

If ith op is Return(k):
$$\hat{c}_i = c_i + \Delta\Phi$$
$$= O(1) + 0 = O(1)$$

If InsertAtEnd with no resize:
$$\hat{c}_i = c_i + \Phi(A_i) - \Phi(A_{i-1})$$
$$= 1 + \left(2A_i.num - A_i.size + 1\right) - \left(2A_{(i-1)}.num - A_{(i-1)}.size + 1\right)$$
$$= 1 + 2\left(A.num - A_{(i-1)}.num\right)$$
$$= 1 + 2 = 3 = O(1)$$

If InsertAtEnd with resize:
$$\hat{c}_i = c + A_{(i-1)}.size$$
$$\left(2A_i.num - \underbrace{A_i.size}_{=2 \cdot A_{(i-1)}.size} + 1\right)$$
$$- \left(2A_{(i-1)}.num - A_{(i-1)}.size + 1\right)$$
$$= c + A_{\ldots} + 2 - 2A_{\ldots}$$