COL 702 Lecture 11 Aug 30

Consider a sequence of operations $O_1, O_2, \ldots O_m$ on a given data structure D where we are interested to bound the total cost of the m operations. This can be done by bounding the worst case cost of any operation, say t_{max}

$\Rightarrow \quad m \cdot t_{max}$

But is possible that we do not encounter the cost t_{max} very frequently i.e., we may be able to get a superior bound.

<u>Example</u> 1 : Suppose D is a stack and $O_i \in \{$push, pop, empty stack$\}$

Cost of push, pop is $O(1)$ and cost of empty stack = # elements in stack

Worst case cost of Empty Stack is

$$O(m)$$

$\Rightarrow$ Total cost for a sequence of m operations is $O(m \cdot m) = O(m^2)$

Note: For special cases like the stack not growing beyond say constant size, total cost $\leq O(c \cdot m) = O(m)$

Observation: The worst case bound $\Omega(m)$ cannot happen very often

In particular if we encounter a cost K for empty stack $\Rightarrow$ there were at least K push operation after the previous empty-stack opn

$$\text{Push Pop push Pop} \quad \text{E.S.} \underbrace{\text{ push pop } \cdots}_{\geq K \text{ push opns}} \text{ E.S}$$

i.e the average cost $= \dfrac{O(K) \times O(1) + O(K)}{K}$

$\sim O(1)$

pops $\leq$ # push ops

Empty stack can be written as a sequence of pop ops

Push pop push . . $\underbrace{\text{ES}}_{\text{pop pop pop}}$. $\overbrace{\text{pop po}}^{\text{ES}}$

averaging arguments is known as

amortized analysis

A general technique for amortized analysis

We define a potential function $\phi : D \to \mathbb{R}$

The amortized cost of a single operation

= actual cost + the change in potential

: w + $\Delta\phi$: potential after − potential before

actual cost

$\Rightarrow$ For a sequence of operation starting from an initial potential ϕ_0

Total amortized cost : $w_1 + [\phi_1 - \phi_0] + w_2 + (\phi_2 - \phi_1)$

$+ \ldots \ldots w_m + \phi_f - \phi_{f-1}$

Total actual cost

$= \sum w_i + \phi_f - \phi_0$

$$\boxed{\text{Total amortized cost} = \text{Total actual cost} + \phi_f - \phi_0}$$

In particular, if $\phi_f - \phi_0 \geq 0$

$\Rightarrow$ Amortized cost $>$ Actual cost

Example Stacks

$\phi(\text{Stack}) = \#$ elements in stack

amortized cost of push : $\underbrace{1}_{\text{actual cost}} + \underbrace{1}_{\text{change in potential}} = 2$

amortized cost of pop : $1 + (-1) = 0$

amortize cost of Empty Stack $= 0$

Total amortize cost of m opns on stack

$=$ worst case amortized cost of a single opn $\times\, m$

$= 2 \times m = O(m)$

Note: $\phi_f - \phi_0 \geq 0$ $\quad \left(\begin{array}{l} \phi_0 = 0 \\ \phi_f - \phi_0 \geq 0 \end{array} \right)$

Suppose we did not begin with an empty stack

Ex 2

Counters : A counter starting from 0 that can count till 2^n can be represented by n bits

3 bit counter

The cost of incrementing a counter : # bit flips

			~~0~~
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	2
1	0	0	1
1	0	1	2
1	1	0	2
1	1	1	3

Total cost for the counter to go from 0 to 2^n $\leq n \cdot 2^n$

__Observation__ : The i^{th} bit flips $\frac{2^n}{2^i}$ times

$0 \leq i \leq n-1$

$\Rightarrow$ Total # bits flipped $= \sum_{i=0}^{n-1} \frac{2^n}{2^i} \leq \boxed{2} \cdot 2^n$

ϕ (counter) = # bits with value 1

amortized cost of incrementing the counter

$=$ # bits flipped $+ \begin{pmatrix} \#1\text{s in count}(i+1) \\ - \#1\text{s in count } i \end{pmatrix}$

when going from $i \to i+1$

$\overleftarrow{\quad K \quad}\!\!\rightarrow$

0 1 0 0 0 1 1 0 1 1 1 1 1   +1

0 1 0 0 0 1 1 $\cancel{0}$ 0 0 0 0 0

$\quad\quad\quad\quad\quad\quad\quad\;\; 1$

For a block of K 1's actual cost : $k+1$

change in potential $\quad = -(k-1)$

$\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad 2$

Amortised cost of increment = 2

<u>Alternative for BST</u> (without deletion)

Operation supported : search, insert

<u>Semi-dynamic</u> dictionary

Arrays :

Given n elements we store it in a <u>set</u> of <u>arrays</u> that can accommodate 2^i elements and we have no more than one for each i

$n = 6$

$\rightarrow A_2 : 2^2$ | 0 | 3 | 7 | 9 |

$\rightarrow A_1 : 2^1$ | 1 | 8 |

$A_0 : 0$

Within each array we keep the elements sorted, but there is no relation between the elements of two distinct arrays

<u>Search</u> ? $\log n$ binary searches one for each of the $\log n$ arrays

$\Rightarrow O(\log^2 n)$

<u>Insert</u> ?