6.006 Recitation 7 25 February 2008

Note Title 2/25/2009

* Pset1 discussion
* Lecture review
* Amortized analysis
  - Binary Counter Problem
* Accounting method

AMORTIZED ANALYSIS

The time required to perform a sequence of data structure operations is averaged over all the operations performed.

Gaurantees the average performance of each operation in worst case.

Three common techniques

- Aggregate Analysis (✓)
- The Accounting method (✓)
- The potential method (not discussed today)

AGGREGATE ANALYSIS

→ Show sequence of n operations take T(n) time in worst case, amortize cost per operation is T(n)/n.

Stack Operations

PUSH(S,x) : pushes x on stack S
POP(S) : pops the top of stack S and returns the popped object.
MULTIPOP(S,k)

```
MULTIPOP(S,k)
    while not STACK-EMPTY(S) and k≠0
        POP(S)
        k--
```

abstract costs of PUSH → 1 , POP → 1, MULTIPOP(S,k) → min(s,k) |S| = s

Consider a sequence of n PUSH, POP and MULTIPOP operations
Stack initially empty
worst case running time of any stack operation → O(n)
sequence of n operations → $O(n^2)$ (We can achieve tighter bounds !!)

Sequence of n PUSH, POP, MULTIPOP can atmost cost $O(n)$?

Each object can be popped at most once for each time it is pushed.

Number of push operations $\leq n$

Number of times POP can be called on non-empty stack $\leq n$

Average cost of operation is $O(n)/n = O(1)$.

Incrementing a Binary Counter

k-bit binary counter

array $A[0..k-1]$, $\text{length}(A) = k$

binary numbers x stored in A, lowest order bit in $A[0]$ and highest order in $A[k-1]$

$$x = \sum_{i=0}^{k-1} A[i] \cdot 2^i$$

```
INCREMENT(A)
  i=0
  while i<length(A) and A[i]==1
        A[i]=0
        i=i+1
  if i<length(A)
        A[i]=1
```

$O(k)$ time one INCREMENT operation worst-case

n INCREMENT $\rightarrow O(nk)$ (Go for tighter! :))

Consider a sequence of n INCREMENT operations

$A[0]$ flips n times

$A[1]$ flips $\lfloor n/2 \rfloor$ times

$A[2] \rightarrow \lfloor n/4 \rfloor$

⋮

$A[i] \rightarrow \lfloor n/2^i \rfloor \quad \forall\, i,\ 0 \le i \le k-1$

	A	Total Cost
0	00000000	0
1	00000001	1
2	00000010	3
3	00000011	4
4	00000100	7
5	00000101	8
6	00000110	10
7	00000111	11
8	00001000	15
9	00001001	16
10	00001010	18
11	00001011	19
12	00001100	22
13	00001101	23
14	00001110	25
15	00001111	26
16	00010000	31

Total number of flips

$$\sum_{i=0}^{k-1} \text{flips}(A[i]) = \sum_{i=0}^{k-1} \left\lfloor \frac{n}{2^i} \right\rfloor \leq n \sum_{i=0}^{k-1} \left\lfloor \frac{1}{2^i} \right\rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = n \cdot \frac{1}{1-1/2} = 2n = O(n)$$

Amortized cost per operation → $O(n)/n = O(1)$

THE ACCOUNTING METHOD

Amortized cost of an operation?

Assign different charges ($\hat{c}_i$) to different operations (c_i) (some charged more, some less than actual cost)

Operation's amortized cost > actual cost, difference (CREDIT) used to pay for other operations.

Total amortized cost of sequence of operations must ≥ Total actual cost of sequence.

$$\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i \quad \forall \text{ operations } i.$$

Total credit stored in the data structure $\sum_{i=1}^{n} \hat{c}_i - \sum_{i=1}^{n} c_i \geq 0$ at all times.

Stack operations

	Actual costs	Amortized costs	
PUSH	1	2	O(1)
POP	1	0	O(1)
MULTIPOP	min(s,k)	0	O(1)

stack of plates

A dollar bill → a unit of cost

Push a plate, 1 dollar to push and 1 dollar as credit on top of it.
At any point, every plate has $1 credit on the stack.

POP → no charge for operation, pay using the credit

MULTIPOP → no charge, again use the credit
Amount of credit is always <u>non-negative</u> as number of plates are non-negative

Incrementing a binary counter

Amortized cost
\$2 to set a bit to 1.
When a bit is set, use \$1 to pay for actual setting of bit, place \$1 as credit on the bit to be used later for flipping back to 0.

INCREMENT
while loop cost of resetting bits → 0 (paid by the credit on the bits)
At most one bit is set

Amortized cost of 1 INCREMENT → \$2 $O(1)$.

Number of 1's in the counter ≥ 0

Amount of credit ≥ 0

n INCREMENT operations → $O(n)$.

DYNAMIC HASH TABLES

Amortized complexity