Readings Chapter 17.

Self test Exercises 17.1-2.

Lecture 13

Amortized Analysis

Often we perform *sequences* of operations on data structures and time complexity for processing the entire sequence is important. Define *worst-case sequence complexity* of sequence of m operations as the maximum runtime over *all* sequences of m operations.

$$\text{WCSC} \leq m \cdot \text{worst-case time of any one operation in any sequence of } m \text{ operations}$$

Ex. sorted linked list,
starting from empty,
`INS`, `DEL`, `SEARCH`

Upper Bound

Worst case for one operation is $\Theta(k)$ where k is the size of the list. Also max size after k operations $\leq k$. Worst-case runtime of operation $i \leq c(i-1)$.

$$\text{WCSC } \leq \sum_{i=1}^{m} c(i-1) = O\left(\frac{c(m)(m-1)}{2}\right)$$

Lower Bound

Consider the sequence `INS(1)`, `INS(2)`, ..., `INS(m)`

$$\sum_{i=1}^{m} d(i-1) = \frac{d(m-1)m}{2}$$

Combing upper and lower bounds we get that

$$\text{WCSC } \in \left(\frac{\Theta(m^2)}{m}\right) = \Theta(m) \text{ ammortized sequence complexity}$$

Binary Counter

Seqeunce of k bits (k fixed) with single operation:
`increment`: add 1 to integer represented by counter. The "cost" (i.e. running time) of one `increment` operation is equal to the number of bits that change during `increment`. For example, if $k = 5$:

initial counter	`00000`	(value = 0)	
after `increment`:	`00001`	(value = 1)	cost = 1
after `increment`:	`00010`	(value = 2)	cost = 2
after `increment`:	`00011`	(value = 3)	cost = 1
after `increment`:	`00100`	(value = 4)	cost = 3
after `increment`:	`00101`	(value = 5)	cost = 1
⋮			
after `increment`:	`11101`	(value = 29)	cost = 1
after `increment`:	`11110`	(value = 30)	cost = 2
after `increment`:	`11111`	(value = 31)	cost = 1
after `increment`:	`00000`	(value = 0)	cost = 5
⋮			

Aggregate Approach

Compare worst-case sequence complexity of a sequence of operations and divide by the number of operations in the sequence.

k bits in counter, `INCREMENT`
"cost" $\rightarrow$ runtime (# of bits flipped)

bit number	changes	total number of changes
0	every time	n
1	every other time	$\lfloor n/2 \rfloor$
2	every other^2 time	$\lfloor n/4 \rfloor$
i^{th}	every other^i time	$\lfloor n/2^i \rfloor$
$k-1$	every 2^{k-1} operations	$\lfloor n/2^{k-1} \rfloor$

$$\text{Total bits flipped } \leq \sum_{i=0}^{k-1} \left\lfloor \frac{n}{2^i} \right\rfloor \leq \sum_{i=0}^{\lfloor \lg n \rfloor} \left\lfloor \frac{n}{2^i} \right\rfloor \leq n \sum_{i=0}^{\lg n} \frac{1}{2^i} \leq n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$

Ammortized cost is $= 2n/n = 2$.

Accounting Method

Each operation is assigned a "cost" (representing running time) and a "charge" (representing ammortized worst-case running time, approximately). Goal: ensure total charge for sequence $\geq$ total cost for sequence.

Imagine individual elements in data structure can store "credit": when operation's charge $\geq$ cost, charge "pays" for cost and amount left over is assigned to specific elements in data structure; when operation's charge $<$ cost, use some stored credit to "pay" for cost. To ensure this works, argue credit never negative (equivalent to total charge for sequence $\geq$ total cost for sequence). Then ammortized complexity $\leq$ average charge.

Binary counter example:
During one `increment` operation many bits may change from 1 to 0 but exactly one bit will change from 0 to 1. (for example, `increment(00111)` $\rightarrow$ `01000`) Can we ensure enough credits to flip every 1 to 0? Then each operation only need be charged for flipping 0 to 1.

Idea: just charge each operaion \$2: \$1 to flip 0 to 1 and \$1 to store with the bit just changed to 1. Since counter starts at 0, possible to show "credit invariant":

> "At any step during the sequence, each bit of the counter that is equal to 1 will have \$1 credit."

Proof by induction:

Initially, counter is 0 and no credit: invariant trivially true.
Assume invariant true and `increment` is performed
  Cost of flipping 1's to 0's paid for by credits stored with each 1;
  Cost of flipping 0 to 1 paid for by \$1 of \$2 charge;
  Remaining \$1 stored with new 1.
No other bit changes so every 1 has \$1 credit at the end.

This shows total charge for sequence is upper bound on total cost. In this case, total charge $= 2n$, so ammortized cost per operation is $\leq 2n/n = 2$ (same as before).