amortized analysis

CIS 313, 315, 413/513, 621
stronger than average case

- with average case there is probability involved
- amortized analysis looks at a sequence of operations
- bound is on total worst case time
- which provides a guarantee on the average time per operation
- total-time/number-of-operations
simple example – array expansion

• start with an array of (say) $k=16$ locations
• when it fills, allocate array of size $2k$
• copy everything from array of size $k$ into that of size $2k$ (and maybe initialize the unused portion)
• this does not have to be done very often
• a total of $n$ array insertions takes $O(n)$ total time, even for large $n$
• we say $O(1)$ amortized time per insertion
types of amortized analysis

• aggregate
  count the total time, divide by the number of operations

• accounting
  assign a cost per operation, money goes into bank to pay for other operations, bank account must stay positive

• potential
  assign a money function to the data structure, cost of an operation is the actual time plus change in potential (which can increase or decrease)
array example

• consider the $n$ insertions into array, count write steps and copy steps

• **aggregate**: total time is $3n$

• **accounting**: insert $3$, expand $0$

• **potential function**: $(\text{# filled spaces}) - (\text{# empty spaces})$
accounting case

$3 to insert? where does the money go?

in array of size $2^k$

- $1 to insert in location $i$  \( (2^{k-1}+1 \leq i \leq 2^k) \)
- $1 to copy location $i$ to location $i$ in array of size $2^{k+1}$
- $1 to copy location $i-2^{k-1}$ to location $i-2^{k-1}$ in array of size $2^{k+1}$

that location never has to pay again – locations in future expansions will pay for the copy
notation

- $c$ is the actual cost of a step
- $\hat{c}$ is the amortized cost of a step
- $S$ is a data structure, $\varphi(S)$ is the potential
- if $S$ is the data structure before a step, and $S'$ is the data structure after a step, the change in potential is $\varphi(S') - \varphi(S)$
- amortized cost $\hat{c}$ is defined to be the actual cost plus change in potential

**definition**: $\hat{c} = c + \varphi(S') - \varphi(S)$
potential applied to array expansion

let the array $A$ have $f$ filled locations and $e$ empty locations
potential prior to an operation: $\phi(A) = f - e$

**insert:**
in $A'$: $f+1$ filled locations and $e-1$ empty locations
new potential: $\phi(A') = (f+1) - (e-1)$
actual cost: $c=1$ (for the write)

amortized cost: $\hat{c} = c + \phi(A') - \phi(A) = 1 + [(f+1) - (e-1)] - [f-e] = 3$

**expand:**
initially in $A$, $e=0$
in $A'$: size is doubled, $f$ full locations, $f$ empty locations
new potential: $\phi(A') = f-f = 0$
actual cost: $c=f$ (copy steps)

amortized cost: $\hat{c} = c + \phi(A') - \phi(A) = f + [f - f] - [f-0] = 0$
(potential has decreased to pay for the copying)
**general idea for potential**

**notation:**
- $c_i$ is actual cost of step $i$
- $\hat{c}_i$ is amortized cost of step $i$
- $S_i$ is data structure at step $i$
- $\hat{c}_i = c_i + \varphi(S_i) - \varphi(S_{i-1})$

**add up steps:**
- $\hat{c}_1 = c_1 + \varphi(S_1) - \varphi(S_0)$
- $\hat{c}_2 = c_2 + \varphi(S_2) - \varphi(S_1)$
- $\hat{c}_3 = c_3 + \varphi(S_3) - \varphi(S_2)$
- ...
- $\hat{c}_n = c_n + \varphi(S_n) - \varphi(S_{n-1})$

$\sum \hat{c}_i = \sum c_i + \varphi(S_n) - \varphi(S_0)$

**moral of story:** sum of amortized costs is upper bound on sum of actual costs, so long as $\varphi(S_n) \geq \varphi(S_0)$
stimulate queue with 2 stacks

Q = stack S1, S1

enqueue(x):
   S1.push(x)

dequeue:
   if S2 empty
      while S1 not empty
         S2.push(S1.pop)
   return S2.pop

aggregate method
look at the life cycle of any x,
at most 4 push/pops each

accounting method
enqueue $4
dequeue $0

potential: $\varphi(Q) = 3 \cdot \text{size}(S1) + \text{size}(S2)$
CLAIM: any series of enqueue and dequeue operations, n of which are enqueues, takes Θ(n) time (push or pop operations).

**accounting method:** the $4 charge for an enqueue pays for the initial push, then perhaps at some later point, the pop from S1, the push onto S2, and a pop from S2.

**potential method:**
(enqueue) Suppose initially S1 has i items and S2 has j items. After the enqueue, S1 has i+1 and S2 has j items. The actual cost was 1 (push). The actual cost plus potential change is

\[ 1 + [(3(i+1)+j)] - [(3i+j)] = 1 + 3i + 3 + j - 3i - j = 4 \]
**potential method:**
(dequeue)

**case j>0:** this causes a pop from S2, which now has size j-1
new Q’: potential $\phi(Q’) = 3i + (j-1)$
actual cost: $c = 1$ (for the pop)
amortized cost: $\hat{c} = c + \phi(Q’) - \phi(Q) = 1 + [3i + (j-1)] - [3i + j] = 0$

**case j=0:** i items popped from S1, then pushed onto S2, then one item popped from S2
new Q’: potential $\phi(Q’) = 3 \cdot 0 + (i-1)$
actual cost: $c = i + i + 1 = 2i + 1$
amortized cost: $\hat{c} = c + \phi(Q’) - \phi(Q) = 2i + 1 + [3 \cdot 0 + (i-1)] - [3i + 0] = 0$

as before, S1 has i items, S2 has j items, potential of Q is $\phi(Q) = 3i + j$
a compendium of potentials

**array expansion:**
\[ \varphi(A) = f - e \]
f=# filled locations
e=# empty locations

**splay trees:**
\[ \varphi(T) = \sum_{v \in T} \lg \text{size}(v) \]
where \( \text{size}(v) \) is the number of nodes
in \( T \) of the subtree rooted at \( v \)

**binary counter (p 454):**
\[ \varphi(A) = \# \text{of 1's in A} \]

**queue with 2 stacks:**
\[ \varphi(Q) = 2 \cdot \text{size}(S1) + \text{size}(S2) \]

**union find by rank with path compression:**
something horrible involving inverse Ackermann
(see p 577)

**Fibonacci heap:**
\[ \varphi(H) = e(2m+t) \]
m=# marked nodes
t=# trees in root list
e=constant