6.046	Lecture 5	Feb. 24. 2015

TODAY: Amortization

- aggregate method
- accounting method
- charging method
- potential method

different approaches/techniques for amortized analysis — all related, but one often easier than others

- table doubling
- binary counter
- 2-3 trees

examples of amortized analysis

Powerful technique for data structure analysis — often, what you really care about

Recall: table doubling [6.006]

- n elements in table of m slots
- want $m = \Omega(n)$ for $1 + \frac{m}{n} = O(1)$ expected performance (with hashing with chaining)

- idea: if n grows $\geq m$, double m
- cost: $\Theta(m+n) = \Theta(n)$ to build new table

$\Rightarrow$ pay $\Theta(2^0 + 2^1 + 2^2 + 2^3 + \cdots + 2^{\lceil \lg n \rceil}) = \Theta(n)$ total to resize table over n insertions

$\Rightarrow \Theta(1)$ amortized cost per insertion

Aggregate method: "just add it up"

$$\frac{\text{total cost of } k \text{ operations}}{k}$$

$=$ amortized cost per operation

- common only for simple analyses

Amortized bounds:

- assign an "amortized cost" to each operation such that "preserve total":

  Σ_i amortized costs $\geq$ Σ_i actual costs

  ↳ over all operations, for any operation sequence

  (average is just one option)

- e.g. can say 2-3 trees achieve

  $O(1)$ worst-case per create-empty

  $O(\lg n^*)$ amortized per insert

  $\emptyset$ amortized per delete (assuming exists)

  where n^* = maximum size of set at any time

  because c creations, i insertions, $d \leq i$ deletions

  cost $O(c + \underbrace{(i+d)}_{\leq 2i} \lg n^*) = O(c + i \lg n^* + \emptyset d)$

  - we'll tighten to $O(\lg n)$ where n = current set size, below

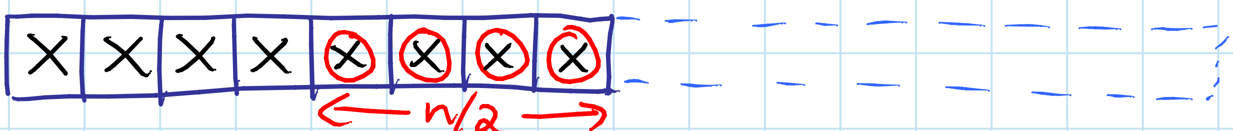
Accounting method: "planning ahead for rainy day"

- allow an operation to store credit (like bank)
  $\Rightarrow$ amortized cost > actual cost
- allow operations to pay using existing credit
  $\Rightarrow$ amortized cost < actual cost

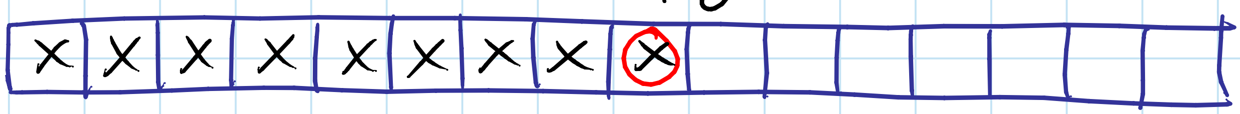
Example: table doubling

- when inserting an element, add a coin to it representing $c = \Theta(1)$ work
- when table needs to double $n \to 2n$, $n/2$ new elements still with coins

X element
O coin



- use up those coins to pay for $\Theta(n)$ rebuild



$\Rightarrow \Theta(n) - n/2 \cdot c$ amortized rebuild cost $= 0$ for large enough c

- $O(1) + c = \Theta(1)$ amortized cost per insert

Counterexample: free deletion in 2-3 trees

- claim: $O(\lg n)$ am. insert, $\emptyset$ am. delete
- attempt: put coin worth $\Theta(\lg n)$ on inserted element
- trouble: when deleting that element, n might be bigger $\Rightarrow$ coin worth too little

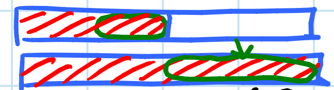
Charging method: (blaming the past) (not in CLRS)

- allow operations to charge cost retroactively to past operations (not future ops)
- amortized cost of op. = actual cost

usually one or other

  - total charge to past ops.
  + total charge by future ops. to this op.

Example: table doubling



- when table doubles $n \to 2n$, charge $\Theta(n)$ cost to $n/2$ inserts since last doubling

$\Rightarrow$ each of these elements charged $\frac{\Theta(n)}{n/2} = \Theta(1)$ & won't be charged again

$\Rightarrow$ $\Theta(1)$ amortized per insert

Example: table doubling & halving

- motivation: want $\Theta(n)$ space even with deletes
- if table down to $1/4$ full ($n = m/4$): shrink to half size ($m \to m/2$) at $\Theta(m)$ cost

$\Rightarrow$ still half full after any resize

$\Rightarrow$ still $\geq m/2$ inserts to charge to on growth

- also $\geq m/4$ deletes to charge to on shrink
- each operation charged $\leq$ once, by $\Theta(1)$

$\Rightarrow$ $\Theta(1)$ amortized per insert & delete

- could do this argument with coins instead, but less intuitive (to me)

  ↳ 2 bank accts.

Example: free deletion in 2-3 trees

- claim: $O(\lg n)$ am. insert, $\emptyset$ am. delete (assuming exists)
- insert charges nothing
- delete charges one insert:
  - NOT the insertion of same element (same problem as accounting method)
  - insertion that brought n to its current value
- before n can reach this value again, must have another insert

$\Rightarrow$ each insert charged at most once

Potential method: (defining karma)

- define a potential function Φ mapping data-structure configuration $\rightarrow$ nonnegative integer
  - intuitively measuring "potential energy" = potential high costs in the future
  - equivalent to total unused credit (Σ unused coins) stored by all past ops.
    = bank account balance
  - nonnegative $\Rightarrow$ never owe the bank
- amortized cost = actual cost + $\underbrace{\Delta\Phi}_{= \Phi(\text{DS after op.}) - \Phi(\text{DS before op.})}$

$\Rightarrow$ sum of amortized costs telescopes
= sum of actual costs + $\underbrace{\Phi(\text{final DS})}_{\geq \emptyset}$ $-$ $\underbrace{\Phi(\text{initial DS})}_{\text{initial balance}}$

- so also need to pay Φ(initial DS) at start
  ~ ideally $\emptyset$ or $O(1)$ ~ else another amortization

- in accounting method, specify offset ($\Delta\Phi$) between actual cost & amortized cost, which determines total stored value (Φ)
- in potential method, specify total stored value Φ, which determines changes per op.: $\Delta\Phi$
- sometimes one is more intuitive than other
- potential method feels most powerful (to me) but also the hardest to come up with proof (Φ)

Example: binary counter

$$\begin{array}{l} 0011010111 \\ 0011011000 \end{array} \Big\} \text{ incr.}$$

- operation: increment
- increment costs $\Theta(1 + \#\text{ trailing 1 bits})$

  so intuition is that 1 bits are bad
- define $\Phi = c \cdot \#\text{ 1 bits in counter}$

$\Rightarrow \Delta\Phi$ from increment $= c(-\#\text{trailing 1 bits} + 1)$

$\Rightarrow$ amortized cost $=$ actual cost $+ \Delta\Phi$

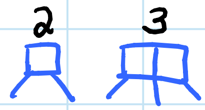
$= \Theta(1 + \#\text{trailing 1 bits}) + c(-\#\text{trailing 1 bits} + 1)$

$= O(1)$ for c large enough

- $\Phi(\text{initial DS}) = \emptyset$ assuming we start @ $000\cdots0$

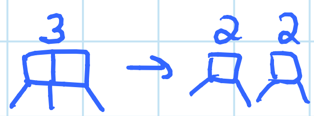
  (necessary for $O(1)$ amortized bound)

Example: insert in 2-3 trees



- $O(\lg n)$ splits in worst case
- but claim only $O(1)$ amortized splits
- what causes splits? nodes overflowing
- $\Phi = \#$ nodes with 3 children

$\Rightarrow \Delta\Phi \le \underbrace{1}_{\text{add child @ top}} - \underbrace{\#\text{ splits}}_{\text{each split turns}}$



$\Rightarrow$ amortized $\#$ splits $=$ actual $\#$splits $+ \Delta\Phi$

$\le \#\text{ splits} + (1 - \#\text{ splits}) = 1.$

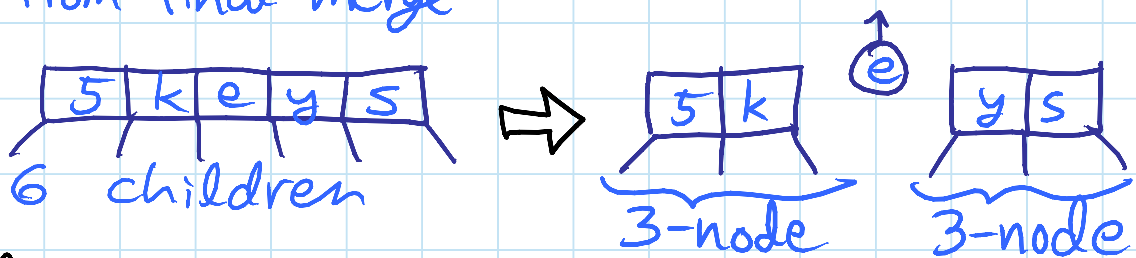
- $\Phi(\text{initial DS}) = \emptyset$ if we start empty

In B-trees: $\Phi = \#$ nodes with B children

In (a,b)-trees: $\Phi = \#$ nodes with b children

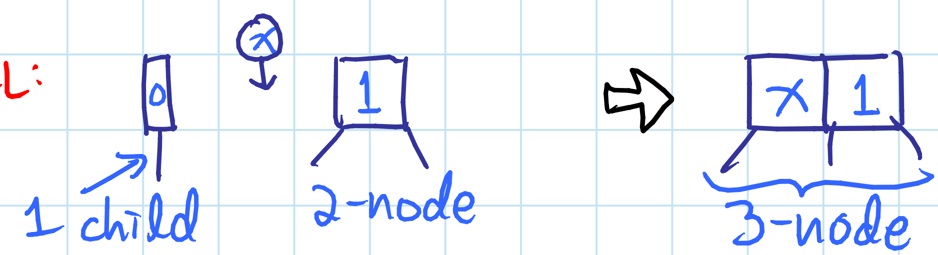
Example: insert & delete in $\underbrace{(2,5)}^{\text{2-5 children}}$-trees
- claim $O(1)$ amortized splits & merges
- overflows cause splits → 5-nodes
- underflows cause merges → 2-nodes
- Φ = # 5-nodes + # 2-nodes
- insert: $\Delta\Phi \le \underbrace{1}_{\text{make a 5-node from final merge}} - \underbrace{\#\text{ splits}}_{\text{destroy 5-nodes (\& no new 2-nodes)}}$

OVERFULL:



- delete: $\Delta\Phi \le \underbrace{1}_{\text{make a 2-node from final steal}} - \underbrace{\#\text{ merges}}_{\text{destroy 2-nodes (\& no new 5-nodes)}}$

UNDERFULL:



⇒ amortized costs = $O(1)$
- Φ(initial DS) = $\emptyset$ if we start empty

In (a,b)-trees: need $b > 2a$

Potential examples could also be done with accounting method: coins on 1s or 2/5-nodes.

6.046J / 18.410J Design and Analysis of Algorithms
Spring 2015