Amortized Analysis: Linear Lists & Search Trees

Binary search trees: worst case $\Theta(\lg n)$

Linear lists: worst case $\Theta(n)$

Stochastic model (probs p_i): Static optimal ($S_{Opt} = H$ and $\Sigma\, ip_i$ resp (expected cost)

Worst case for a sequence of requests: amortized cost = worst(seq cost/seq length)

We will consider self-adjusting data structures

Try to do as well as possible for the input sequence.

Self-adjusting Data Structures: Linear Linked Lists

Move to front (MTF)

Transpose

Move halfway…etc

Thm (Rivest): Under the stochastic model (fixed independent probs) expected cost of transpose better than move to front (except $n \leq 2$ or all p_i's the same)

Thm (Rivest): Under same model expected MTF cost $\leq 2 \Sigma\, i p_i = 2S_{Opt}$

(Actually this can be improved to $(\pi/2)\, S_{Opt}$)

Amortized cost of MTF

Model: Start with empty list

Scan to element requested

if not there insert/charge)at end

Apply heuristic (no charge)

Thm: Under "insert of first request" start-up model cost MTF $\leq 2S_{Opt}$ on any sequence of requests

Proof:…

(Note transpose can be a disaster … cost n)

Off-line vs On-line

Data structures usually on-line: respond to each operation before we see the next

Compare performance with off-line model: see entire sequence in advance

Competitive ratio = worst case ($\frac{\text{Online time of alg}}{\text{Optimal offline time}}$)

Method is "competitive" if ratio a constant

Model becomes extremely important

"Usual" Model (Sleator & Tarjan)

Basics: search element in pos i: scan to i at cost i

Unpaid exchange: move this element closer to front (other in same order)

Paid exchange: swap any pair of adjacent elements, charging to go further than i if you like, cost 1 per swap

Delete: Like search

Insert: n+1 if n elements already there (+ any "paid")

MTF Competitive, under a model

Thm(Sleator & Tarjan): Under model described, MTF is within a factor of 2 of offline optimal, i.e. 2-competitive

Approach: Run MF and any algorithm (such as off line opt) on same sequence.

Keep a potential function Φ= #inversions between MF and A

Count carefully

Be Careful About Model

Cost 1 per exchange of consecutive values

- So… reversing list $\Theta(n^2)$

- Swap as many pairs of consecutive elements as you like
  - (1 per exchange) so swapping front and back halves of list costs $\Theta(n^2)$

$\Rightarrow$ **(n/2+1) .. n 1 2 3 4 5 6 .. n/2**

A Realistic Change in the Model

- Scan as far as you like. Charge one per element inspected
- KEY POINT Can rearrange portion inspected arbitrarily at no charge (in fact we will perform very simple rearrangements) so
  - so swapping front and back halves of list costs $\Theta(n)$

$\Rightarrow$ (n/2+1) .. n 1 2 3 4 5 6 .. n/2

Order by Next Request

- Scan for element i, continue to the $2^{\lceil \lg i \rceil}$
- Reorder the element requested by NEXT REQUEST

Cost	New Ordering
16	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
2	2 1 ..
4	3 4 1 2 ..
2	4 3 ..
8	5 6 7 8 1 2 3 4 ..
2	6 5 ..
4	7 8 5 6 ..
2	8 7 ..
16	9 10 11 12 13 14 15 16 1 2 3 4 5 6 7 8

Cost of a "Cycle" for Order by Next Request

- Cost (under our model) is $n \lg n$, amortized $\lg n$
- Lower bound Theorem: Under our model, $n \ln n$ inspections are required to individually scan for each of n elements.

Proof idea: "Crossing sequence argument"

Access to elements in position i or later must occur at least n/i times … for a total cost of at least $\Sigma\ n/i \approx n \ln n$

More General Bound on Amortized Cost of OBNR

Main Theorem: Let r denote the number of distinct elements requested since the last for a given value, then the amortized cost for a search can be charged as at most $1 + 4 \lceil \lg r \rceil$

Idea of Proof: "Soak the Middle Class"

For any request, entire cost is borne by penultimate block of size $2^{\lceil \lg i \rceil - 2}$, so costs 4 for each of them.

Lemma: An element can be in the penultimate block at most once between its accesses.

So Where Are We?

Linear search:

MTF cost $< 2S_{opt}$

$<$ 2 Off Line Opt (under a particular model)

Problem: model allows only swapping adjacent elements

If told next access time on each request, & "merging" takes linear time

Can get: Amortized lg n to access each element in turn; as opposed to linear

And: amortized O(lg r_i) (r_i= #others since last time for i)

≈O(lg(1/pi) or O(H)

How about Binary Search Trees

Could "count & approx" optimal static

… but what if "likely" els change?

Simple exchange (rotate with parent):

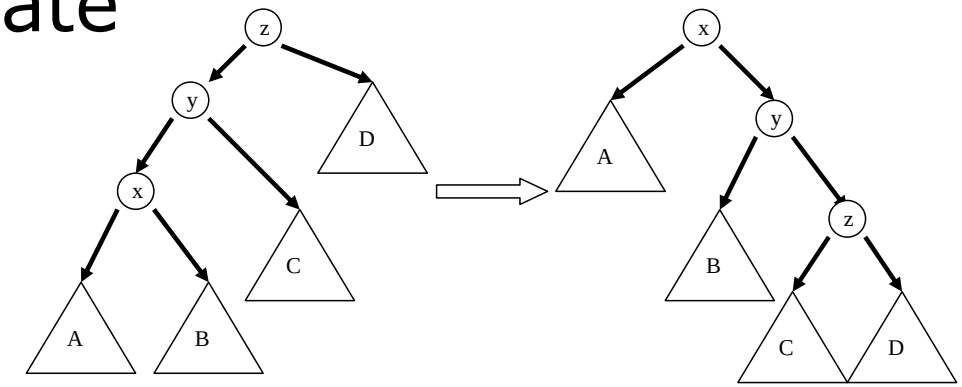
… bad, all probs fixed & equal $\Theta(n^{1/2})$

Rotate all the way to root: 1.38 S_{opt}; but poor on amortized basis

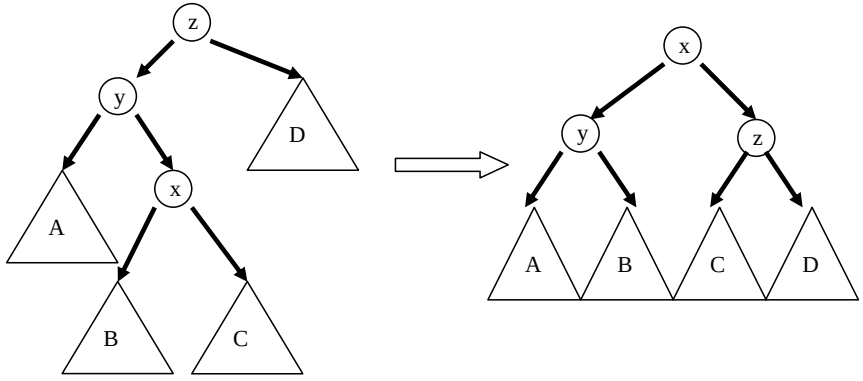
Splaying (Sleator & Tarjan)

A twist on move to root …

- If child of root, rotate
- Else zig-zig



- or zig-zag
- All the way to root



How Good is it?

Weighting nodes: We can give nodes arbitrary positive weights, w_i, $\Sigma w_i = W$.

Then n(x) = sum weights of nodes in subtree.
Scaling weights doesn't matter.

Thm: Change in r(T) in splaying x to the root $\equiv \Delta \leq 3(r(t)-r(x)) - d + 2$.

[n(t) size tree rooted at t; r(t)=lg(n(t)]

Splay Trees Have Good Amortized Behaviour

Boring Balance: Total access time for a sequence of length m is $O(m+(m+n) \lg n)$

Static Opt: Total cost $O(m+ \Sigma q_i \lg(m/q_i))$

$[q_i$ = # requests for i]

Working Set: $O(n\lg n +m+ \Sigma\lg(t_i+1))$

$[t_i$ = # different values since last i]

Splay Tree *Conjecture*: "competitive"