CSE 5311 Notes 3: Amortized Analysis

(Last updated 2/6/17 2:01 PM)

PROBLEM: Worst case for a *single* operation is too pessimistic for analyzing a *sequence* of operations.

ELEMENTARY EXAMPLES

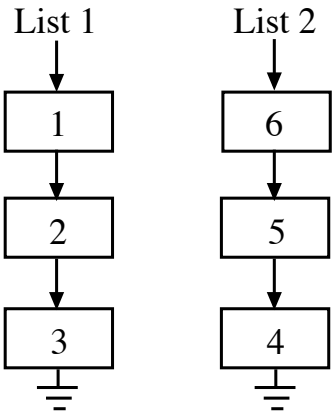
1. Stack operations with "multiple pop"

   Usual push for a single entry - $\theta(1)$

   Multi-pop for k entries - $\theta(k)$

   Sequence of n operations takes $\theta(n)$ time.

2. Queue implemented with two lists/stacks (in a functional language)



Enqueue: At head of list 2

Dequeue: if list 1 is empty
while list 2 not empty
Remove head of list 2
Insert as head of list 1
Remove head of list 1

Application to maximum message length (see end of CSE 2320 Notes 10)

3. Incrementing a counter repetitively by 1 (CLRS, p. 461)

```
0 0 0 1 1 1 1
          + 1
-------------
```

ANALYSIS

1. Aggregate Method

$$\frac{\sum \text{actual cost}}{\#\text{ of operations}} = \frac{\sum c_i}{n} = \hat{c}_i = \text{amortized cost}$$

2. Accounting Method - For any sequence $\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$

   Charge more for *early* operations in sequence to pay for *later* operations.

   Consider queue with 2 lists:

   Each item is touched 3 times

   Charge 2 for enqueue

   Charge 1 for dequeue

   Each item in list 2 has a *credit* of 1. Credit is consumed in dequeue with empty list 1.

3. Potential Method - Preferred method

   Concept:

   Generalizes accounting method.

   Tedious for initial designer, but hides details for others.

   Map entire state of data structure to a *potential*. Captures "difficulty" of future operations.

   Assuming a sequence of operations:

   c_i = actual cost of ith operation
   $\hat{c}_i$ = amortized cost of ith operation
   D_{i-1} = data structure state before ith operation
   D_i = data structure state after ith operation

   $$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

   Total amortized cost for a sequence is:

   $$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} \left(c_i + \Phi(D_i) - \Phi(D_{i-1})\right) = \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0)$$

   Book: Multipop stack (Φ = # of items on stack)

   Binary counter (Φ = # of ones)

   Defining Φ *is the hard part.*

BINARY TREE TRAVERSALS - Slightly more involved than previous examples

Observation: Tree traversal on tree with n nodes requires $2n$ - 2 edges "touches"

Operations: INIT: Finds first node in traversal

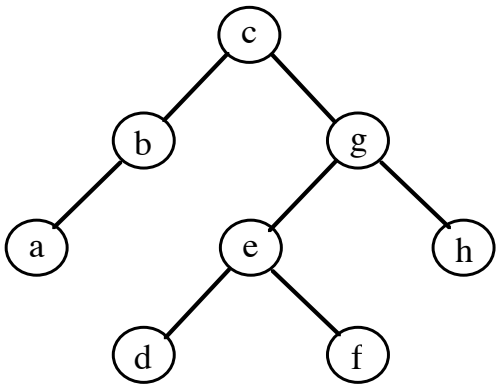
$\hat{c}_1 = 0$

SUCC(x): Finds successor of x

$\hat{c}_i = 2$ for $2 \le i \le n$ (n exits tree)

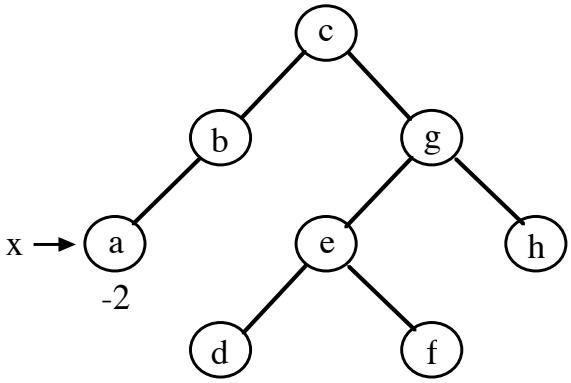
Need Φ for inorder, postorder, and preorder

Example:



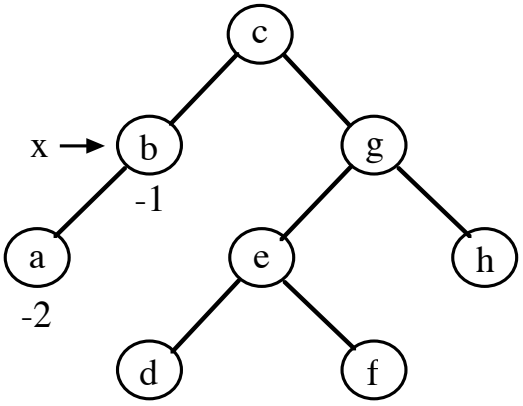
For INIT for *inorder*, must stop at a.

$$c_1 = 2, \hat{c}_1 = 0, \text{and } \hat{c}_1 = c_1 + \Phi(D_1) - \Phi(D_0)$$
$$0 = 2 + \Phi(D_1) - 0$$



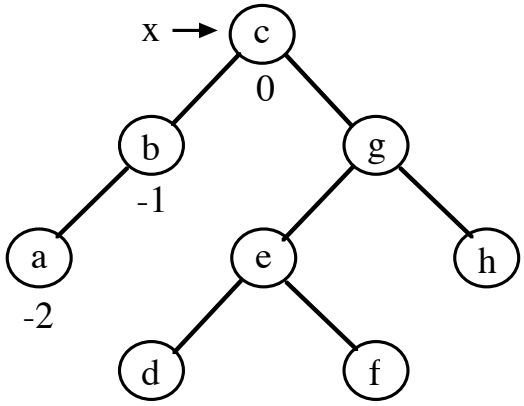
SUCC(a) = b

$$c_2 = 1, \hat{c}_2 = 2, \text{and } \hat{c}_2 = c_2 + \Phi(D_2) - \Phi(D_1)$$
$$2 = 1 + \Phi(D_2) - (-2)$$



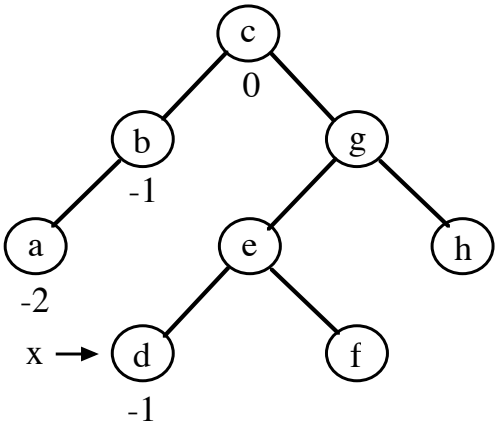
SUCC(b) = c

$$c_3 = 1, \hat{c}_3 = 2, \text{and } \hat{c}_3 = c_3 + \Phi(D_3) - \Phi(D_2)$$
$$2 = 1 + \Phi(D_3) - (-1)$$

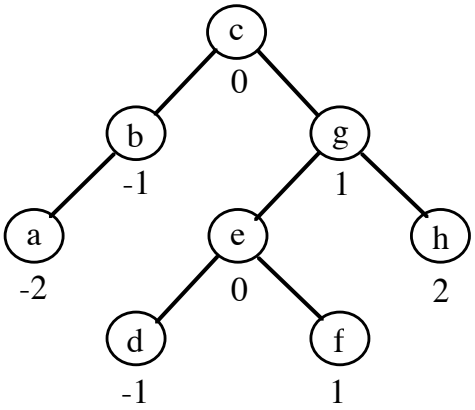


SUCC(c) = d

$$c_4 = 3, \hat{c}_4 = 2, \text{and } \hat{c}_4 = c_4 + \Phi(D_4) - \Phi(D_3)$$
$$2 = 3 + \Phi(D_4) - 0$$

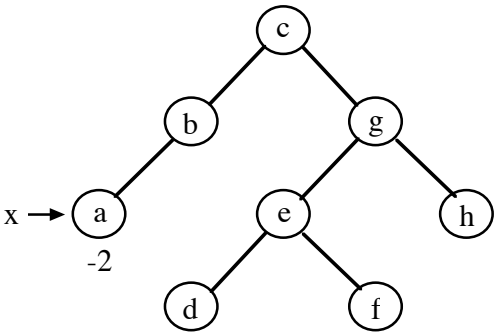


In general: rank, r(x), of node x is r(root) = 0, r(x→left) = r(x) - 1, r(x→right) = r(x) + 1



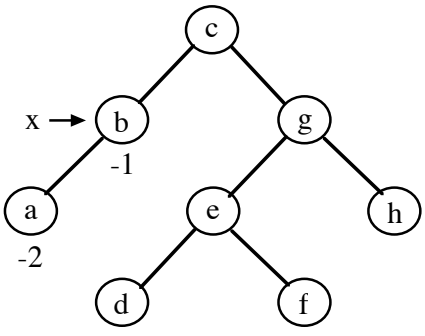
For INIT for *postorder*, must stop at a.

$$c_1 = 2, \hat{c}_1 = 0, \text{and } \hat{c}_1 = c_1 + \Phi(D_1) - \Phi(D_0)$$
$$0 = 2 + \Phi(D_1) - 0$$



SUCC(a) = b

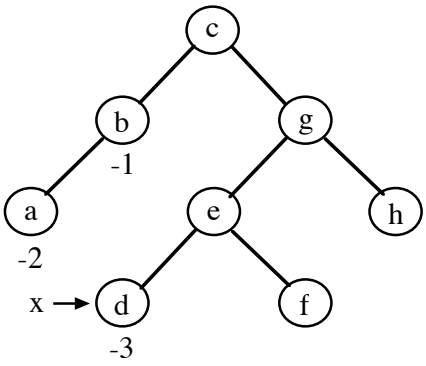
$$c_2 = 1, \hat{c}_2 = 2, \text{and } \hat{c}_2 = c_2 + \Phi(D_2) - \Phi(D_1)$$
$$2 = 1 + \Phi(D_2) - (-2)$$



SUCC(b) = d

$c_3 = 4, \hat{c}_3 = 2, \text{and } \hat{c}_3 = c_3 + \Phi(D_3) - \Phi(D_2)$

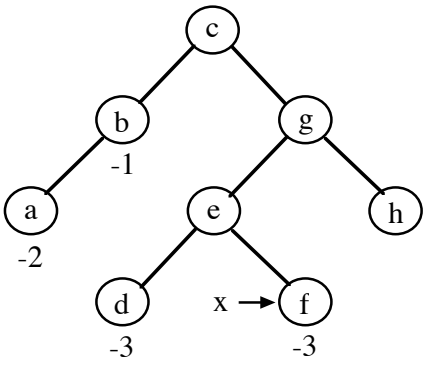
$2 = 4 + \Phi(D_3) - (-1)$



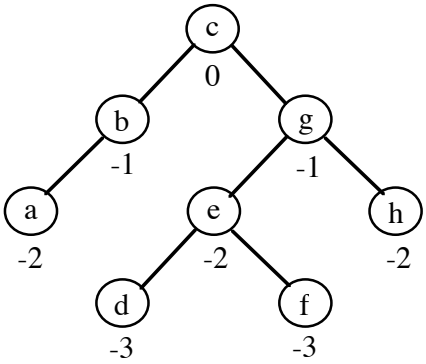
SUCC(d) = f

$c_4 = 2, \hat{c}_4 = 2, \text{and } \hat{c}_4 = c_4 + \Phi(D_4) - \Phi(D_3)$

$2 = 2 + \Phi(D_4) - (-3)$



In general: rank, r(x), of node x is r(root) = 0, r(x→left) = r(x) - 1, r(x→right) = r(x) - 1



For *preorder* (not shown): rank, r(x), of node x is r(root) = 0, r(x→left) = r(x) + 1, r(x→right) = r(x) + 1

Aside: If non-negative ranks/potential are desired (e.g. for inorder and postorder),
then make r(root) = D_0 = height of tree (or number of nodes if height is unknown).

DYNAMIC TABLE GROWTH – CLRS 17.4

Applies to tables with embedded free space.

Periodic reorganization takes $\Theta(n)$ time . . .

Fixed vs. fractional growth and amortizing reorganization cost over all inserts

Deletion issues

CLRS PROBLEM 17-2 – Making binary search dynamic

Related to binomial heaps in Notes 4.a

Representation of dictionary with n items

Binary searches to find item in $\Theta\left(\log^2 n\right)$ time

Inserting an item in $\Theta(\log n)$ amortized time using ordered merges

Deletion?

ASIDE – THE SKI RENTAL PROBLEM (Reference: S. Phillips and J. Westbrook, "On-Line Algorithms: Competitive Analysis and Beyond")

Cost of skiing:

Buy skis = $\$C$

Rent skis one time = \$1

You have never skied $\Rightarrow$ You don't know how many times (i) you will ski.

If you knew i . . .

$i \le C \Rightarrow$ Rent $\qquad C \le i \Rightarrow$ Buy

So, the optimal (offline) strategy yields a cost of $\min(i,c)$.

Deterministic (online) strategy: Rent $C - 1$ times, then buy.

Cost for different scenarios:

$i < C \qquad C < i \qquad i = C$

Maximum ratio of online to offline = *Competitive Ratio*

Can *CR* be improved?

Let strategy A_j be:

Rent up to j - 1 times

Buy on jth trip

Worst-case *CR* for a particular A_j:

$j \le C$: Suppose $i = j$

$$CR = \frac{i-1+C}{i}$$

$$2 \le CR \le C$$

$j > C$: Suppose $i = C$

$$CR = \frac{j-1+C}{C} \ge 2$$

But, the expected *CR* can be improved by randomizing the choice of *j* to weaken the *adversary*:

Knows the algorithms (strategies)

Knows the distribution for *j*

Does not know *j*

(S. Ben-David et.al., "On the Power of Randomization in Online Algorithms")

Determining the optimal distribution (optimal mixed strategy from game theory):

Let $A_j(k)$ = cost of A_j for skiing *k* times

π_j = probability that A_j is chosen

$$\begin{bmatrix} A_1(1) & A_2(1) & \cdots & A_C(1) \\ A_1(2) & A_2(2) & \cdots & A_C(2) \\ \vdots & \vdots & \ddots & \vdots \\ A_1(C) & A_2(C) & \cdots & A_C(C) \end{bmatrix} \begin{bmatrix} \pi'_1 \\ \pi'_2 \\ \vdots \\ \pi'_C \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ \vdots \\ C \end{bmatrix}$$

Solve for $\pi'_1 \cdots \pi'_C$

$$CR = \alpha = \frac{1}{\sum_{j=1}^{C} \pi'_j} \qquad \left(\sum_{j=1}^{C} \pi'_j \ne 1 \right)$$

$$\pi_j = \alpha \pi'_j$$

Closed form:

$$\alpha = \frac{1}{\left(1+\frac{1}{C-1}\right)^{C-1}} + 1 \approx 1.582 = \frac{e}{e-1} \text{ as } C \to \infty \qquad \pi_i = \frac{\alpha-1}{C}\left(\frac{C}{C-1}\right)^{i}$$

MOVE-TO-FRONT (MTF) ONLINE STRATEGY VS. OPTIMAL OFFLINE STRATEGY

(Similar to CLRS problem 17-5. From D.D.Sleator and R.E. Tarjan, "Amortized Efficiency of List Update and Paging Rules", *CACM 28 (2)*, Feb. 1985, http://dl.acm.org.ezproxy.uta.edu/citation.cfm?doid=2786.2793)

Key Differences:

MTF is given requests *one-at-a-time* and thus applies an "obvious" online strategy

1) Go down the list to the requested item

2) Bring it to the front of the list

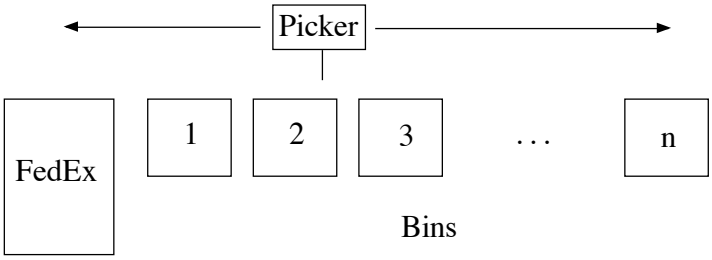
OPT is given the entire sequence of accesses (e.g. offline) *before* any searching or adjustments are applied. (There is no charge for the computation to determine the actions. This "robot scheduling" problem is NP-hard [1].)

For a request, the following "models" the cost of the processing that occurs

1) Go down the list to the requested item (charge 1 unit per item examined)

2) Optionally, bring requested item *closer* to the front of the list (no charge, "free")

3) Optionally, perform "paid" transpositions on adjacent elements (charge 1 unit each)



From earlier:

$$\hat{c}_i = c_i + \Phi(i) - \Phi(i-1)$$

$$\sum_{i=1}^{m} c_i + \Phi(m) - \Phi(0) = \sum_{i=1}^{m} \hat{c}_i$$

If $\Phi(m) - \Phi(0)$ is never negative, then an upper bound on $\sum_{i=1}^{m} \hat{c}_i$ is an upper bound on $\sum_{i=1}^{m} c_i$.

Potential Function Used to Compare Two Strategies:

To simplify the analysis, assume both strategies start with same list. The lists may differ while processing the sequence.

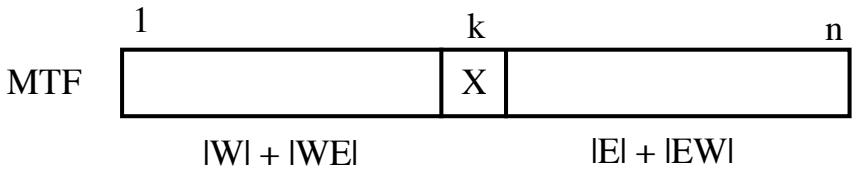
The potential (Φ) is the number of *inversions* in the MTF list with respect to the OPT list. (i.e. count the number of pairs whose order is different in the two lists.) Intuitively, this is the cost for MTF to correct its list to correspond to OPT.

Example: Lists 2, 3, 4, 5, 1 and 3, 5, 4, 1, 2 have 5 inversions.

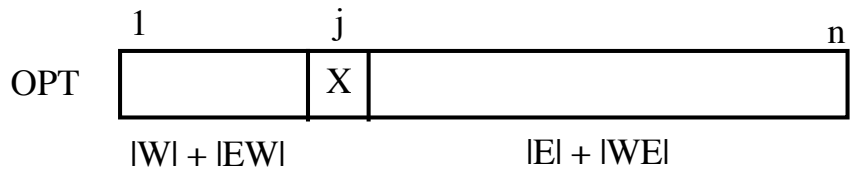
Defining Φ this way gives $\Phi(0) = 0$ and $\Phi(i) \geq 0$, so $\Phi(m) - \Phi(0)$ is never negative.

To compare MTF to OPT (without detailing OPT), the amortized cost ($\hat{c}_i$) of each MTF operation is bounded by the actual cost of the corresponding OPT operation.

Before processing ACCESS to some X



"West" "East"



W = Items "west" of X in both lists E = Items "east" of X in both lists

WE = Items west of X in MTF, but east of X in OPT

EW = Items east of X in MTF, but west of X in OPT

Observations:

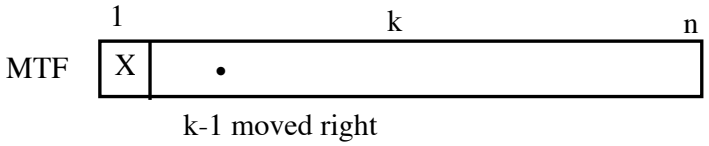
$\|W\| + \|EW\| + 1 = j$	(i)	From OPT diagram
$k = \|W\| + \|WE\| + 1$		From MTF diagram
$k - 1 - \|WE\| = \|W\|$	(ii)	Algebra on previous step
$k - \|WE\| + \|EW\| = j$		Add (i) and (ii), then simplify
$k - \|WE\| \leq j$		From previous step and fact that $\|EW\| \geq 0$

Amortized cost of ACCESS of X (by MTF)

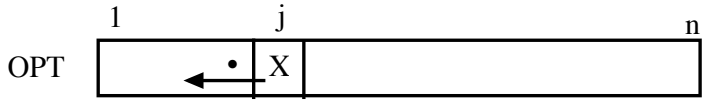
Search + |W| new inversions - |WE| lost inversions

$$k + (k - 1 - |WE|) - |WE| = 2(k - |WE|) - 1 \leq 2j - 1$$

BUT, OPT list also changes (but details are not needed)



k-1 moved right



f free transpositions

p paid transpositions

Actual cost of OPT = j + p

"Correction" to Δ in $\Phi \leq$ -f for lost inversions + p for new inversions

So, after accessing and updating both lists, the amortized cost of MTF ACCESS is

$$\hat{c}_i \leq k + k - 1 - |WE| - |WE| - f + p \leq 2j - 1 - f + p \leq 2j - 1 + 2p \leq 2OPT - 1$$

Based on the observation that an upper bound on $\sum_{i=1}^{m} \hat{c}_i$ is an upper bound on $\sum_{i=1}^{m} c_i$, MTF is dynamically optimal.

[1] C. Ambühl, "Offline List Update is NP-Hard", *Lecture Notes in Computer Science 1879*, Springer-Verlag, 2000.

Aside: Randomization may be used to obtain strategies with better competitive ratios.

BIT:

Every list element has a bit that is set randomly.
Each ACCESS complements bit, but only does MTF if bit is 1.

Achieves expected *CR* ≤ 1.75. Other methods achieve 1.6.

(Recent related paper: S. Angelopoulos and P. Schweitzer, "Paging and List Update under Bijective Analysis", *JACM* 60, *2*, 2013, http://dl.acm.org.ezproxy.uta.edu/citation.cfm?doid=2450142.2450143)

AMORTIZED ANALYSIS OF SPLAY TREE RETRIEVAL (ASIDE)

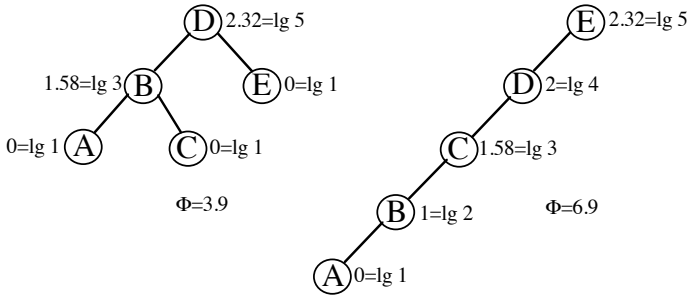
Actual cost (rotations) is 2 for zig-zig and zig-zag, but 1 for zig.

$S(x)$ = number of nodes in subtree with x as root ("size")

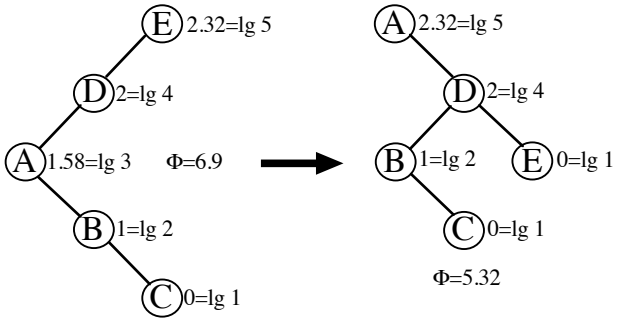
$r(x) = \lg S(x)$ ("rank")

$$\Phi(T) = \sum_{x \in T} r(x)$$

Examples:

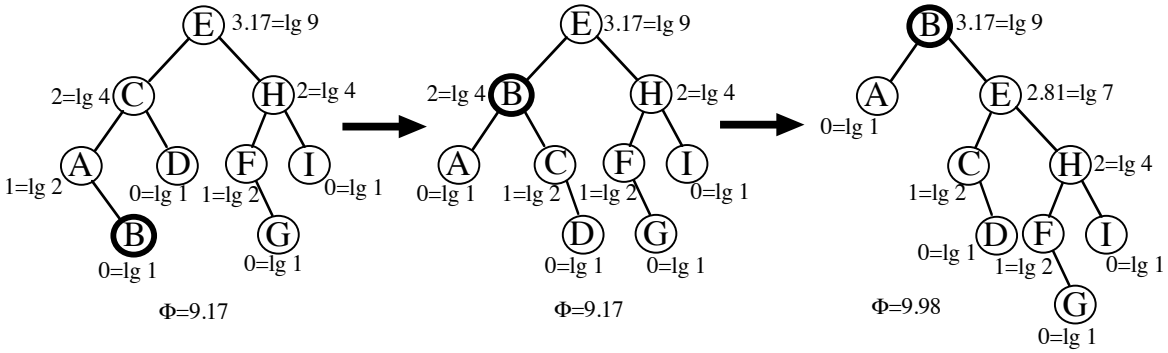


Now suppose that the leaf in the second example is retrieved. Two zig-zigs occur.



$$\sum \hat{C}_i = \sum C_i + \Phi(After) - \Phi(Before) = 4 + 5.32 - 6.9 = 2.42 \le 1 + 3\lg n = 7.96$$

Another example of splaying. There will be a zig-zag and a zig.



$$\sum \hat{C}_i = \sum C_i + \Phi(After) - \Phi(Before) = 3 + 9.98 - 9.17 = 3.81 \le 1 + 3\lg n = 10.51$$

Compute amortized complexity of individual steps and then the complete splaying sequence:

Lemma: If $\alpha > 0,\ \beta > 0,\ \alpha + \beta \le 1$, then $\lg\alpha + \lg\beta \le -2$.

Proof: $\lg\alpha + \lg\beta = \lg\alpha\beta$. $\alpha\beta$ is maximized when $\alpha = \beta = \frac{1}{2}$, so $\max(\lg\alpha + \lg\beta) = -2$.

Access Lemma:

Suppose 1) x is node being splayed

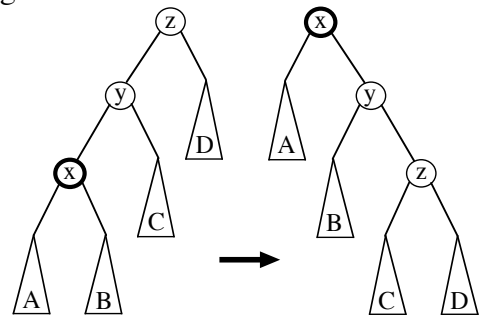
2) subtree rooted by x has

$S_{i-1}(x)$ and $r_{i-1}(x)$ before ith step
$S_i(x)$ and $r_i(x)$ after ith step

then $\hat{C}_i \le 3r_i(x) - 3r_{i-1}(x)$, except last step which has $\hat{C}_i \le 1 + 3r_i(x) - 3r_{i-1}(x)$.

Proof: Proceeds by considering each of the three cases for splaying:

Zig-Zig:



$$\begin{aligned}
\hat{C}_i &= C_i + \Phi(T_i) - \Phi(T_{i-1}) \\
&= 2 + r_i(x) + r_i(y) + r_i(z) - r_{i-1}(x) - r_{i-1}(y) - r_{i-1}(z) && \text{Potential changes only in this subtree} \\
&= 2 + r_i(y) + r_i(z) - r_{i-1}(x) - r_{i-1}(y) && r_i(x) = r_{i-1}(z) \\
&\le 2 + r_i(y) + r_i(z) - 2r_{i-1}(x) && r_{i-1}(x) \le r_{i-1}(y) \\
(*) \quad &\le 2 + r_i(x) + r_i(z) - 2r_{i-1}(x) && r_i(y) \le r_i(x)
\end{aligned}$$

Let $\alpha = \frac{S_{i-1}(x)}{S_i(x)},\ \beta = \frac{S_i(z)}{S_i(x)},\ \alpha > 0,\ \beta > 0,\ \alpha + \beta = \frac{S_{i-1}(x) + S_i(z)}{S_i(x)} \le 1$. ($y$ is absent from numerator)

Lemma conditions are satisfied, so $\lg\alpha + \lg\beta \le -2$. Applying logs to α and β gives:

$$r_{i-1}(x) + r_i(z) - 2r_i(x) \le -2$$

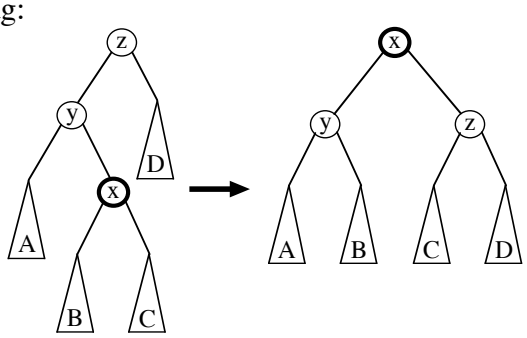
which may be rearranged as:

$$0 \le 2r_i(x) - r_{i-1}(x) - r_i(z) - 2$$

Add this to (*) to obtain:

$$\hat{C}_i \le 3r_i(x) - 3r_{i-1}(x)$$

Zig-Zag:



$$\hat{C}_i = C_i + \Phi(T_i) - \Phi(T_{i-1})$$

$= 2 + r_i(x) + r_i(y) + r_i(z) - r_{i-1}(x) - r_{i-1}(y) - r_{i-1}(z)$ Potential changes only in this subtree

$= 2 + r_i(y) + r_i(z) - r_{i-1}(x) - r_{i-1}(y)$ $r_i(x) = r_{i-1}(z)$

(**) $\le 2 + r_i(y) + r_i(z) - 2r_{i-1}(x)$ $r_{i-1}(x) \le r_{i-1}(y)$

Lemma may be applied by observing that $S_i(y) + S_i(z) \le S_i(x)$ and thus

$$\frac{S_i(y)}{S_i(x)} + \frac{S_i(z)}{S_i(x)} \le 1$$

By lemma, $\lg\left(\frac{S_i(y)}{S_i(x)}\right) + \lg\left(\frac{S_i(z)}{S_i(x)}\right) \le -2$

$r_i(y) + r_i(z) - 2r_i(x) \le -2$

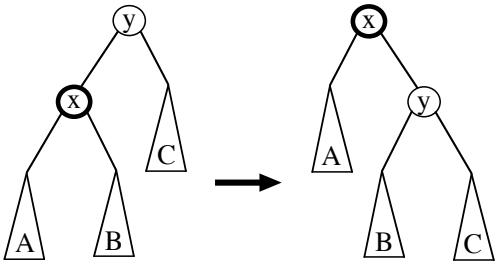
$r_i(y) + r_i(z) \le 2r_i(x) - 2$ which can substitute into (**)

$\hat{C}_i \le 2 + 2r_i(x) - 2 - 2r_{i-1}(x)$

$= 2r_i(x) - 2r_{i-1}(x)$

$\le 3r_i(x) - 3r_{i-1}(x)$ Since $r_{i-1}(x) \le r_i(x)$

Zig:



$$\begin{aligned}
\hat{C}_i &= C_i + \Phi(T_i) - \Phi(T_{i-1}) \\
&= 1 + r_i(x) + r_i(y) - r_{i-1}(x) - r_{i-1}(y) && \text{Potential changes only in this subtree} \\
&= 1 + r_i(y) - r_{i-1}(x) && r_{i-1}(y) = r_i(x) \\
&\le 1 + r_i(x) - r_{i-1}(x) && r_i(y) \le r_i(x) \\
&\le 1 + 3r_i(x) - 3r_{i-1}(x) && r_{i-1}(x) \le r_i(x)
\end{aligned}$$

Bound on total amortized cost for an entire splay sequence:

$$\begin{aligned}
\sum_{i=1}^{m} \hat{C}_i &= \sum_{i=1}^{m-1} \hat{C}_i + \hat{C}_m \\
&\le \sum_{i=1}^{m-1} \left(3r_i(x) - 3r_{i-1}(x)\right) + 1 + 3r_m(x) - 3r_{m-1}(x) \\
&= 3r_{m-1}(x) - 3r_0(x) + 1 + 3r_m(x) - 3r_{m-1}(x) \\
&= 1 + 3r_m(x) - 3r_0(x) \\
&\le 1 + 3r_m(x) \\
&= 1 + 3\lg n && \text{since } x \text{ is the root after the final rotation}
\end{aligned}$$

Asides:

If each node is assigned a positive weight and the size of node x, $S(x)$, is the sum of the weights in the subtree, other results may be shown such as:

Static Optimality: Splay trees (online) perform within a constant factor of the optimal (static) binary search tree (offline) for a sequence of requests.

But the elusive goal remains:

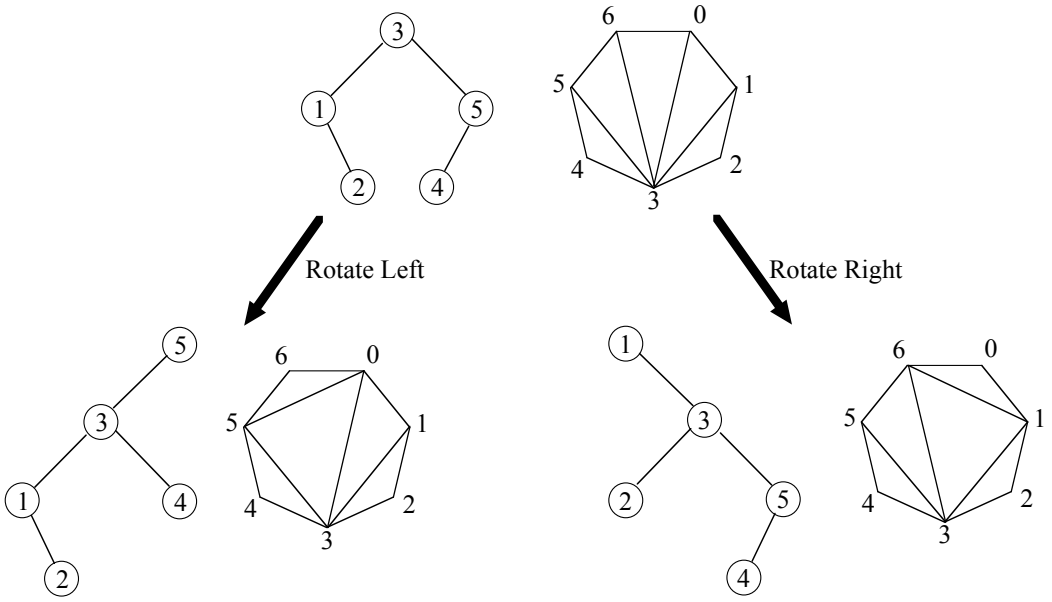
Dynamic Optimality *Conjecture*: Splay trees (online) perform within a constant factor of the optimal (dynamic) binary search tree (offline) for a sequence of requests.

In principle, this may be attacked like "MTF is dynamically optimal". Many difficulties arise . . .

Potential Function: Minimum number of rotations to transform a BST into another BST?

Transformation: Suppose a BST with n nodes has keys $1 \ldots n$. Construct the regular $(n+2)$-gon with vertices $0, 1, 2, \ldots, n+1$. If the BST has a subtree with root i, minimum key $\min(i)$, and maximum key $\max(i)$, then include edges $\{i, \min(i)-1\}$ and $\{i, \max(i)+1\}$. Also, include edge $\{0, n+1\}$. These edges give a triangulated polygon.

A BST rotation corresponds to "flipping" the diagonal of a quadrilateral:



(A bound of $O(\log \log n)$ on the competitive ratio has been shown:
`http://erikdemaine.org/papers/Tango_FOCS2004/`)

FURTHER APPLICATIONS OF POTENTIAL FUNCTION METHOD THIS SEMESTER . . .

Union-find trees (Notes 7) - not detailed

Push-relabel methods for maxflows (Notes 9) - not detailed

KMP string search (Notes 14) - easy