



COMP 482 / ELEC 420

Amortized Analysis

Reading

- Read [CLRS] 17.
- Read handout.

Outline

- Motivation with informal example – splay trees
- Amortization details
- Formal example 1 – stacks with multipop
- Formal example 2 – splay trees

Amortization Idea

Balanced trees →
 $O(\log n)$ per operation, $O(m \log n)$ for m operations.

What if we settle for this alone?

I.e., some individual operations might be slow, as long as total behavior is good.

Splay trees →
 $O(n)$ per operation worst case, $O(m \log n)$ for m operations.

Amortization Idea

Define:

All sequences of m operations starting with an empty data structure have $O(m f(n))$ cost

→

Each operation has *amortized* $O(f(n))$ cost

Splay Tree Idea

BST, not necessarily balanced

→

access may take $O(n)$ time.

During access (search, insert, or delete), adjust tree in a way that tends to improve future accesses.

Splay Tree Restructuring

Rotates nodes touched during access.

- Doesn't increase asymptotic cost of access.
- **Tends** to balance tree.
- Moves accessed node to root.

Restructuring rules are heuristics.

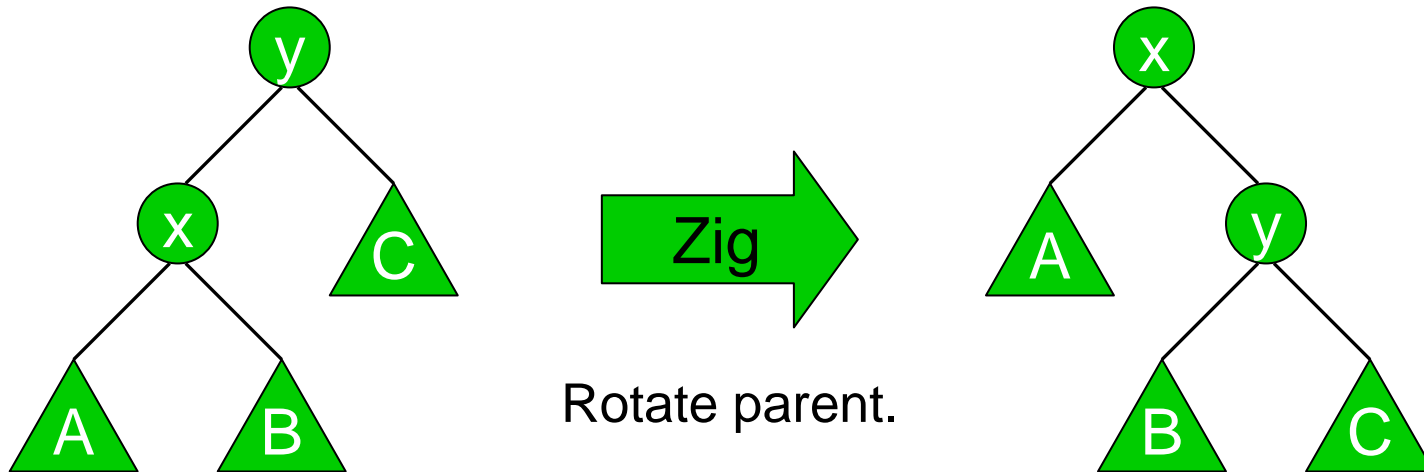
- Benefits **not** obvious.
- Many other heuristics tried until a sufficient set found.

Three cases...

Splay Tree Restructuring: Case 1

“Zig”

x is a child, & parent is not a child.

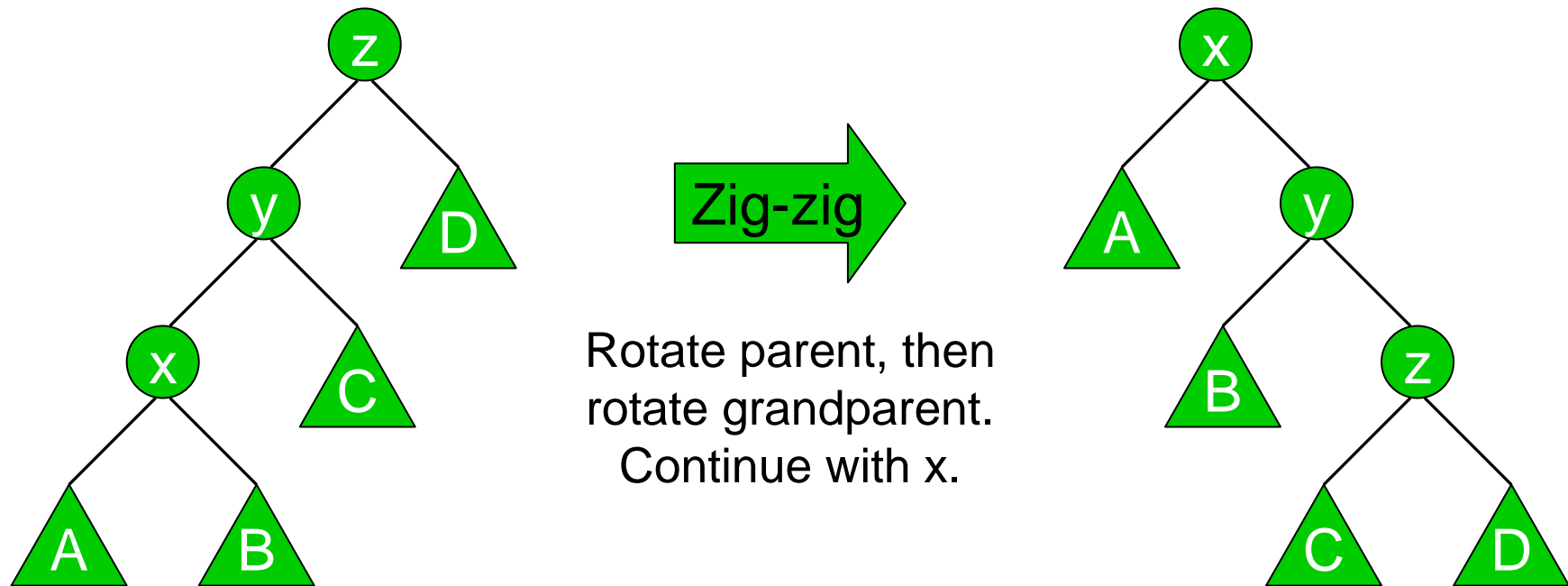


And similar if x is right child.

Splay Tree Restructuring: Case 2

“Zig-Zig”

x is a left (right) child, & parent is a left (right) child.

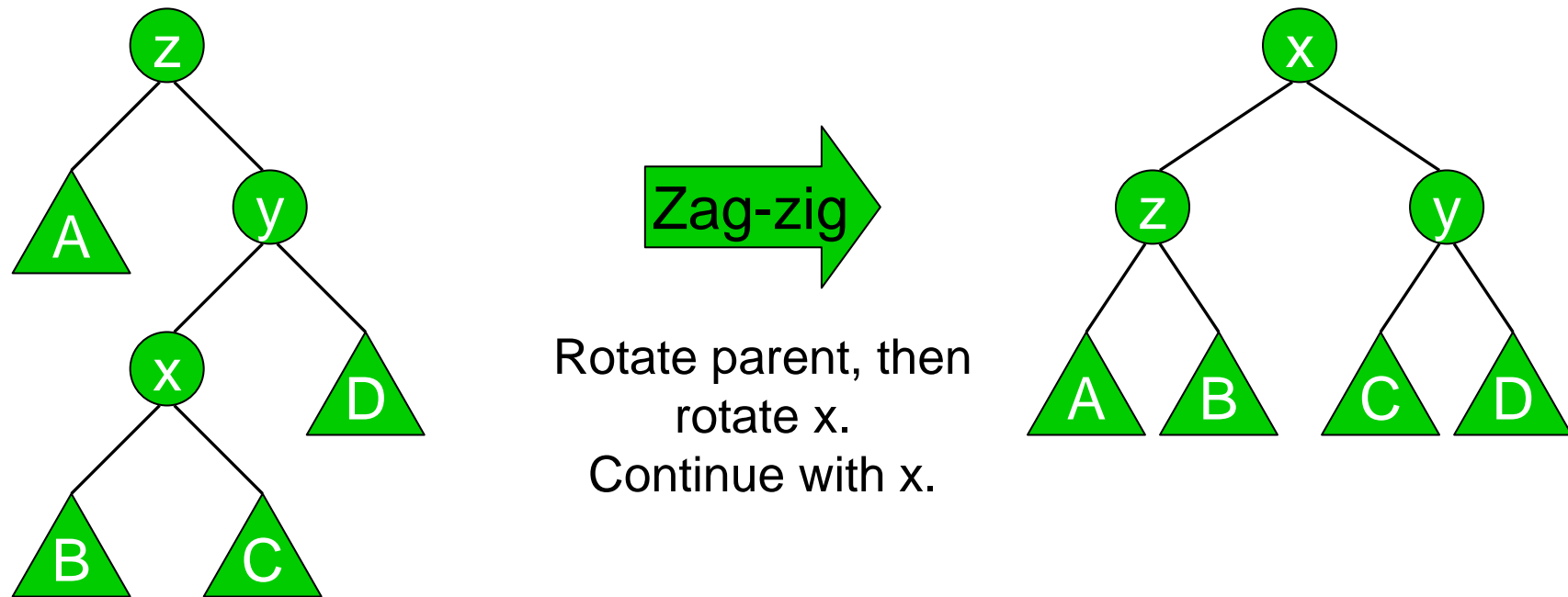


“Zag-zag”: if they are right children.

Splay Tree Restructuring: Case 3

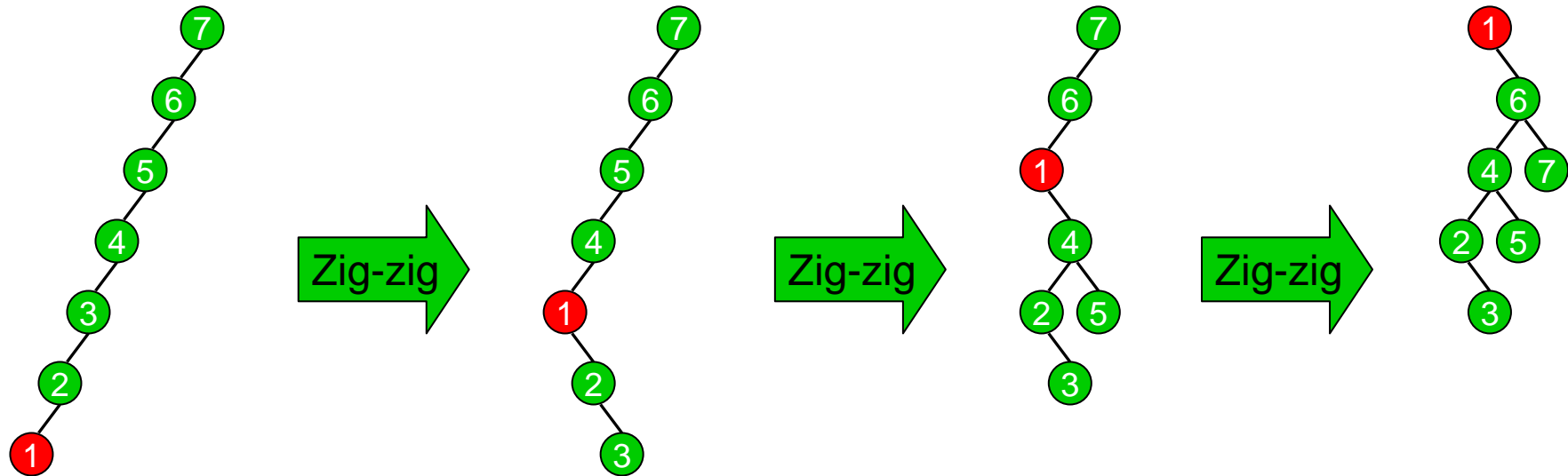
“Zag-Zig”

x is a left (right) child, & parent is a right (left) child.



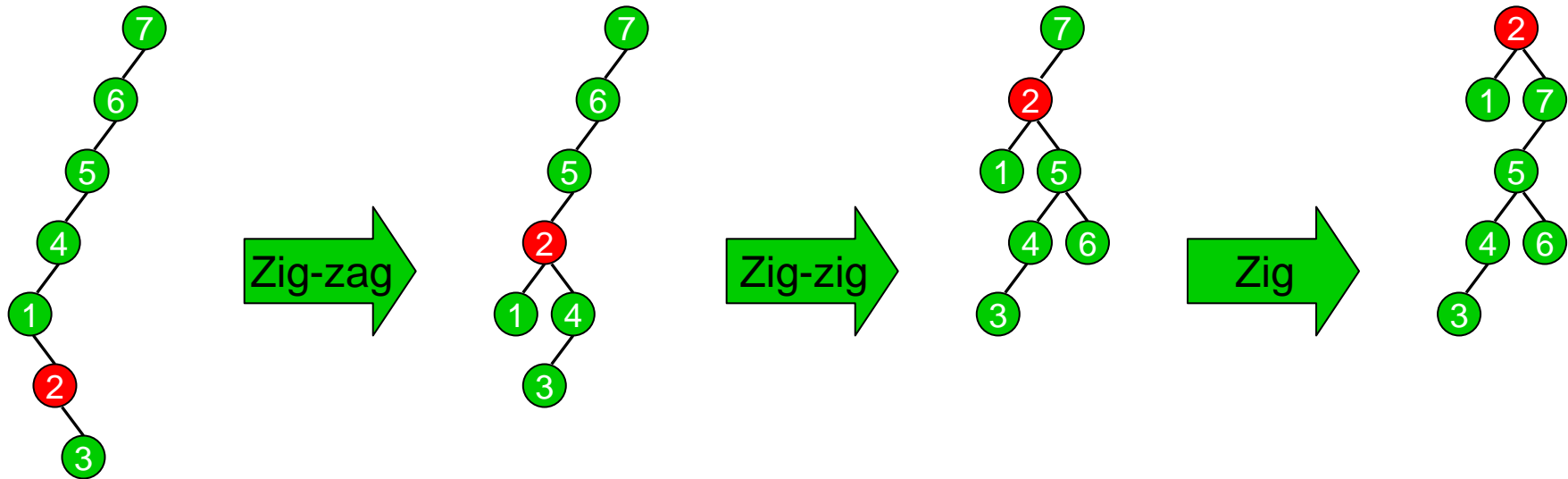
“Zig-zag”: if x is right child, & parent is left child.

Splay Tree Restructuring: Example 1



Splay 1.

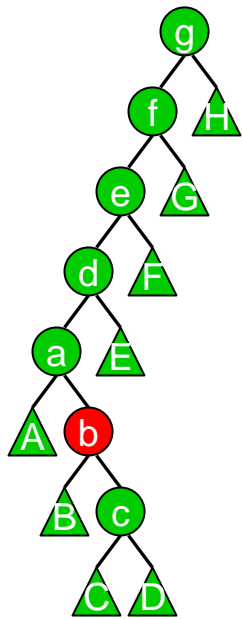
Splay Tree Restructuring: Example 2



Splay 2.

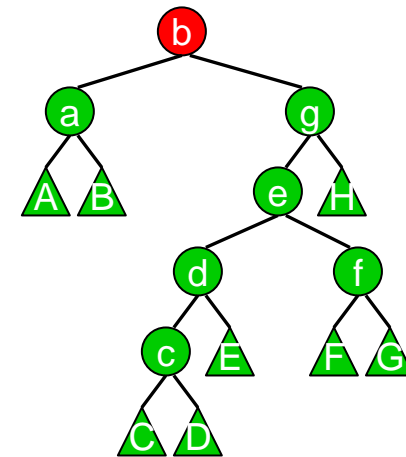
Splay Tree Restructuring: Example 3

Add subtrees to previous example.



Splay b.

Zig-zag, Zig-zig, & Zig



Splay Tree Operations

Operations:

- Search: Search for x , then splay x .
- Insert: Insert x , then splay x .
- Delete: Delete x , then splay its parent.

Costs:

Only increases constant costs of each access.

Small constant, since simple.

Will show $O(\log n)$ amortized...

Amortized Analysis

Basic idea:

- Can “overpay” cost of an op.
- Credits can be used later to compensate for expensive ops.
- Can't go into debt.

?

Why important?

?

Credit balance = cost estimate - actual cost.

Ultimate result: Cost estimate = $O(\dots)$

Debt \rightarrow estimate too low \rightarrow result useless.

Amortized Analysis

Can “overpay” cost of an op.

Credits can be used later to compensate for expensive ops.

Can't go into debt.

3 styles of analysis:

Aggregate:

- Show all sequences of m ops require $O(g(m))$ time.
- Thus, each op $O(g(m) / m)$ amortized time.

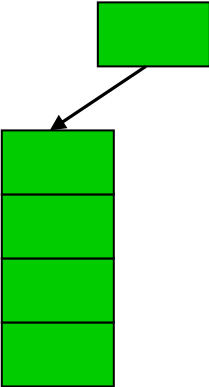
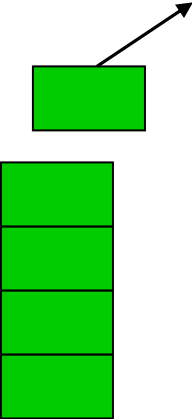
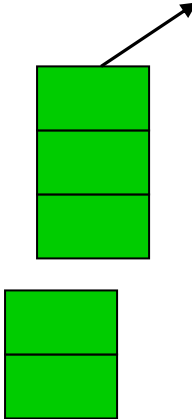
Accounting:

- Credits associated with items in the data structure.

Potential:

- Credits associated with overall data structure.

A Simple Example

3 ops:			
	Push(S,x)	Pop(S)	Multi-pop(S,k)
Worst-case cost:	$O(1)$	$O(1)$	$O(\min(S ,k))$ $= O(n)$

Amortized cost: $O(1)$ per op

A Simple Example: Aggregate

Need to show:

All sequences of m ops take $O(m)$ time.

? Fairly obvious – why? ?

Each pushed item can be popped only once.

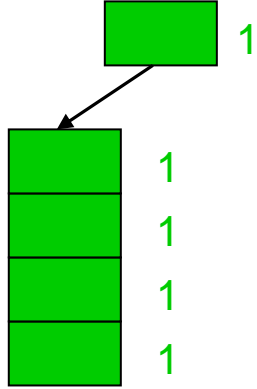
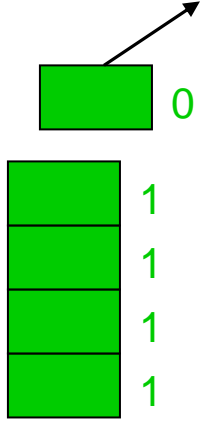
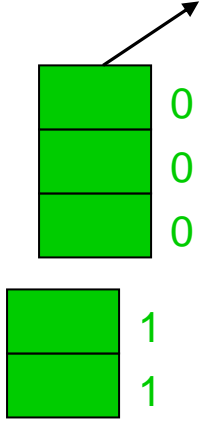
Total cost = $O(\text{items pushed} + \text{items popped}) = O(m)$

Aggregate

Potential difficulties in use:

- No clear strategy how to use in all cases.
- Hard to use when costs are dependent on data, not just structure.
 - Here, only used stack structure – data values didn't matter.

A Simple Example: Accounting

3 ops:			
	Push(S,x)	Pop(S)	Multi-pop(S,k)
Assigned cost:	2	0	0
Actual cost:	1	1	$\min(S ,k)$

Push(S,x) pays for possible later pop of x.

Accounting

Potential difficulties in use:

- Determining what costs to assign.
 - Here, followed easily from problem intuition.
 - Often obtain by trial-and-error.
- Associating credits with specific items in data structure.
 - Esp. if items get moved around, like in splay tree.

Potential: Definitions

D_i	Data structure – after i^{th} op
$\phi(D_i)$	“Potential” (i.e., credits) of D_i
c_i	Actual cost of i^{th} op
\hat{c}_i	Amortized cost of i^{th} op
	$= c_i + \phi(D_i) - \phi(D_{i-1})$

$$\sum_{i=1}^m \hat{c}_i = \sum_{i=1}^m (c_i + \phi(D_i) - \phi(D_{i-1})) = \left(\sum_{i=1}^m c_i \right) + \phi(D_m) - \phi(D_0)$$

$$\geq \sum_{i=1}^m c_i \quad \text{if } \phi(D_m) > \phi(D_0) \quad \text{– typically } \phi(D_0) = 0$$

A Simple Example: Potential

Define: $\phi(D_i) = \# \text{items in stack}$ Thus, $\phi(D_0) = 0$.

Plug in for operations:

Push: $\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1})$
 $= 1 + j - (j-1)$
 $= 2$

Pop: $\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1})$
 $= 1 + (j-1) - j$
 $= 0$

Multi-pop: $\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1})$
 $= k' + (j-k') - j$ $k' = \min(|S|, k)$
 $= 0$

Thus $O(1)$ amortized time per op.

Potential

Potential difficulty in use:

- Choosing $\phi()$.
 - Here, followed easily from problem intuition.
 - Often obtain by trial-and-error.

Splay Tree Analysis

Aggregate?

- Actual costs of ops highly dependent on data.

Accounting?

- Maybe credits on nodes pay for rotations?
 - But some nodes get rotated much more than others.
- Data/nodes get moved around fairly unpredictably.
 - Hard to understand what credits are where.

Potential?

- Can ignore details of current tree structure.
- What $\phi()$?

Splay Tree Analysis: Potential Function

Choice of ϕ not at all obvious.

? Ideas? ?

Most ideas involve some notion of the tree size – heights, #items, #leaves, ... – but which?

? Will these work? ?

Splay Tree Analysis: Potential Function

$$\phi(T) = \sum_{i \in T} \lg S(i)$$

$S(i)$ = #descendants of i , including i .

Alternate notation:

$$\phi(T) = \sum_{i \in T} R(i)$$

$R(i) = \lg S(i)$ = “rank” of i .

Advantage: During rotations, at most 3 nodes change rank:
 x , its parent, & its grandparent.

Before operation: $R_i(x), S_i(x)$

After operation: $R_f(x), S_f(x)$

Splay Tree Analysis: Potential Function

Amortized time claims to prove:

Individual steps:

$$x=\text{root}: \quad \hat{c}_i = 1$$

$$\text{Zig}: \quad \hat{c}_i \leq 3(R_f(x) - R_i(x)) + 1$$

$$\text{Zig-zig}: \quad \hat{c}_i \leq 3(R_f(x) - R_i(x))$$

$$\text{Zig-zag}: \quad \hat{c}_i \leq 3(R_f(x) - R_i(x))$$

Overall:

$$\text{Sum telescopes to } \leq 3(R(\text{root}) - R(x)) + 2 = O(\log n)$$

Splay Tree Analysis: Individual Op Costs

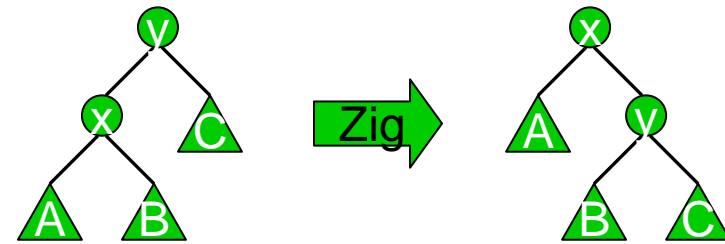
Case: $x=\text{root}$

No rotation \rightarrow $c_i = 1$
no change in potential function.

$$\begin{aligned}\hat{c}_i &= c_i + \phi(D_i) - \phi(D_{i-1}) \\ &= 1 + 0 \\ &= 1\end{aligned}$$

Splay Tree Analysis: Individual Op Costs

Case: Zig



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

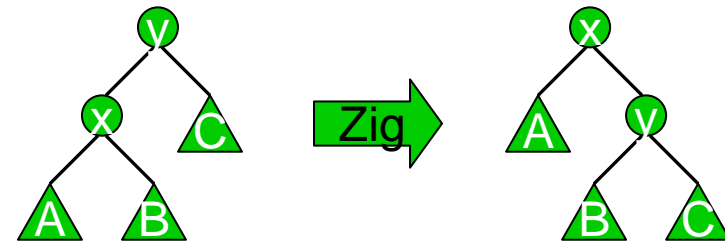
$$= 1 + R_f(x) + R_f(y) - R_i(x) - R_i(y)$$

Rotate

Only x,y can change ranks.

Splay Tree Analysis: Individual Op Costs

Case: Zig



$$\begin{aligned}\hat{c}_i &= c_i + \phi(D_i) - \phi(D_{i-1}) \\ &= 1 + R_f(x) + R_f(y) - R_i(x) - R_i(y)\end{aligned}$$

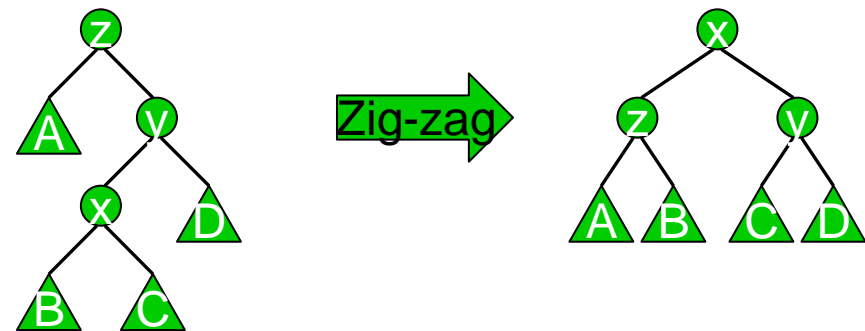
$$S_i(y) > S_f(y) \rightarrow R_i(y) \geq R_f(y) \rightarrow R_i(y) - R_f(y) \geq 0$$

$$S_f(x) > S_i(x) \rightarrow R_f(x) \geq R_i(x) \rightarrow R_f(x) - R_i(x) \geq 0$$

$$\begin{aligned}\hat{c}_i &\leq 1 + R_f(x) - R_i(x) \\ &\leq 1 + 3(R_f(x) - R_i(x))\end{aligned}$$

Splay Tree Analysis: Individual Op Costs

Case: Zig-Zag



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

$$= 1 + R_f(x) + R_f(y) + R_f(z) - R_i(x) - R_i(y) - R_i(z)$$

? What relations do we know? ?

$$S_i(z) = S_f(x) \rightarrow R_i(z) = R_f(x) \rightarrow R_f(x) - R_i(z) = 0$$

$$S_i(y) \geq S_i(x) \rightarrow R_i(y) \geq R_i(x)$$

$$S_f(x) \geq S_f(y) + S_f(z) \rightarrow ???$$

A Useful Lemma

$$a+b \leq c, \quad a>0, \quad b>0$$

→

$$\lg a + \lg b \leq 2 \lg c - 2$$

Proof – just algebra:

$$0 \leq (a-b)^2 = a^2 - 2ab + b^2$$

$$4ab \leq a^2 + 2ab + b^2 = (a+b)^2$$

$$ab \leq (a+b)^2 \div 4$$

$$ab \leq c^2 \div 4$$

$$\lg ab \leq \lg (c^2 \div 4)$$

$$\lg a + \lg b \leq 2 \lg c - 2$$

Square is always positive.

Adding 4ab.

Dividing by 4.

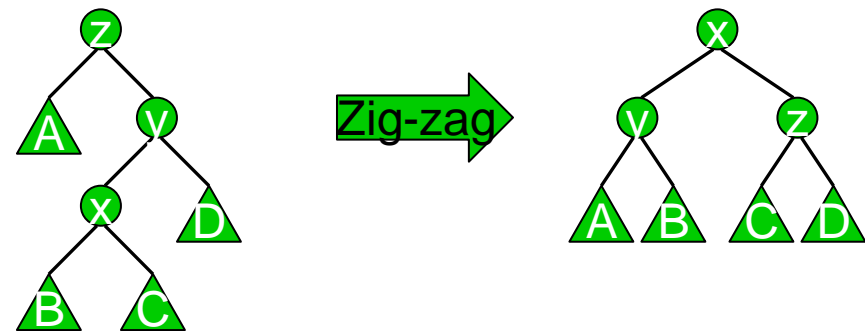
Transitivity with assumption.

Logarithms of both sides.

Simplifying logarithms.

Splay Tree Analysis: Individual Op Costs

Case: Zig-Zag



$$\begin{aligned}\hat{c}_i &= c_i + \phi(D_i) - \phi(D_{i-1}) \\ &= 1 + R_f(x) + R_f(y) + R_f(z) - R_i(x) - R_i(y) - R_i(z)\end{aligned}$$

$$S_i(z) = S_f(x) \rightarrow R_i(z) = R_f(x) \rightarrow R_f(x) - R_i(z) = 0$$

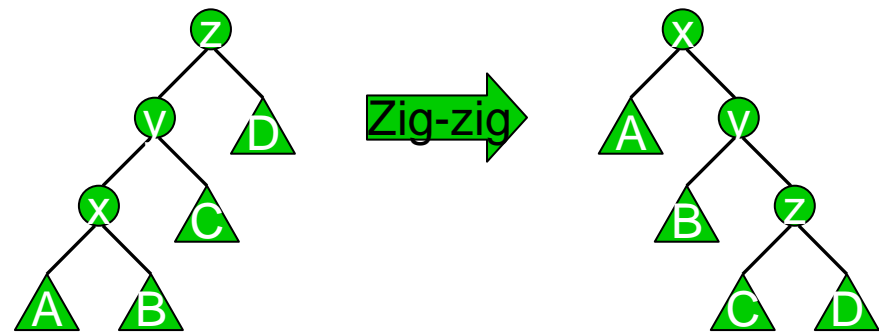
$$S_i(y) \geq S_i(x) \rightarrow R_i(y) \geq R_i(x)$$

$$\begin{aligned}S_f(x) &\geq S_f(y) + S_f(z) \rightarrow 2 \lg S_f(x) - 2 \geq \lg S_f(y) + \lg S_f(z) \\ &\rightarrow 2 R_f(x) - 2 \geq R_f(y) + R_f(z)\end{aligned}$$

$$\hat{c}_i \leq 2R_f(x) - 2R_i(x) \leq 3(R_f(x) - R_i(x))$$

Splay Tree Analysis: Individual Op Costs

Case: Zig-Zig



Similar to zig-zag. Details omitted.

$$\hat{c}_i \leq 3(R_f(x) - R_i(x))$$

Splay Tree Analysis: Total Costs

$$\begin{array}{ll} x=\text{root}: & \hat{c}_i = 1 \\ \text{Zig}: & \hat{c}_i \leq 3(R_f(x) - R_i(x)) + 1 \\ \text{Zig-zig}: & \hat{c}_i \leq 3(R_f(x) - R_i(x)) \\ \text{Zig-zag}: & \hat{c}_i \leq 3(R_f(x) - R_i(x)) \end{array}$$

Zigs happen one level below root. \rightarrow At most one Zig.

Sum of operations telescopes.

$$\begin{aligned} \hat{c}_i &\leq 3(R_f(x) - R_i(x)) + 2 \\ &\leq 3R_f(x) + 2 \\ &= O(\log n) \end{aligned}$$

Search, insert, delete – each a splay & only constant factor other work.

Method Summaries

Aggregate:

1. Sum actual costs of any m ops.
Potentially difficult to know actual costs or bound well.
2. Result is sum/m .

Accounting:

1. Derive actual costs c_i of each kind of op.
2. Define accounting costs \hat{c}_i of each kind of op, such that can assign credits $\hat{c}_i - c_i$ consistently to data elts.
Poor definitions lead to loose bounds.
3. Result is max acct. cost.

Potential:

1. Define potential function $\Phi(D)$, such that $\Phi(D_i) \geq \Phi(D_0)$.
Poor definition leads to loose bounds.
2. Calculate accounting costs $\hat{c}_i = c_i + \Delta\Phi$ of each kind of op.
3. Result is max acct. cost.