Balanced Search Trees

Robert Tarjan, Princeton University & HP Labs

(Joint work with Bernhard Haeupler and Siddhartha Sen)

Searching: Dictionary Problem

Maintain a set of items, so that

- **Access**: find a given item
- **Insert**: add a new item
- **Delete**: remove an item

are efficient

Assumption: items are totally ordered, so that binary comparison is possible

Balanced Search Trees

AVL trees
red-black trees
weight balanced trees
binary B-trees

} binary

2,3 trees
B trees

} multiway

etc.

Topics

- Rank-balanced trees [WADS 2009]

  Example of exploring the design space

- Ravl trees [SODA 2010]

  Example of an idea from practice

- Splay trees [Sleator & Tarjan 1983]

Rank-Balanced Trees

Exploring the design space…

Joint work with B. Haeupler and S. Sen

Problem with BSTs: Imbalance

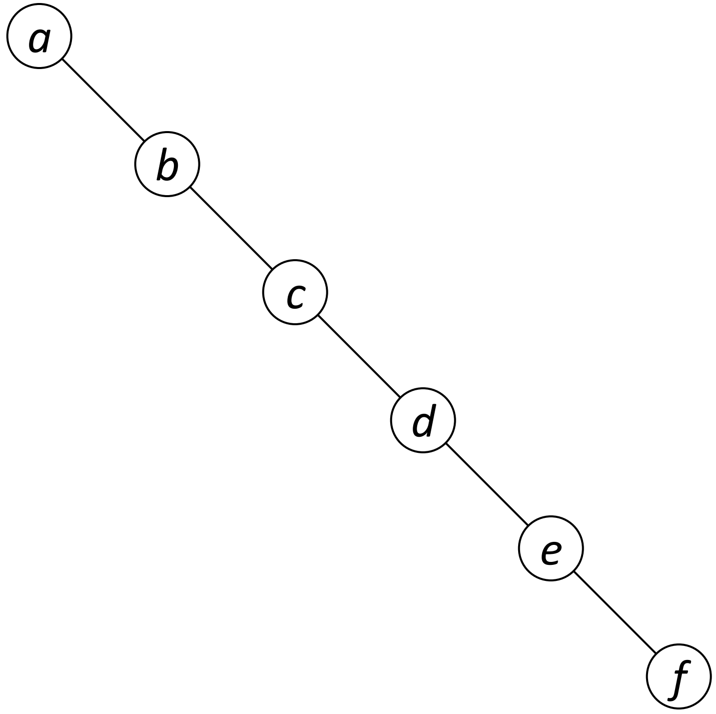
How to bound the height?

- Maintain local balance condition, rebalance after insert or delete
  balanced tree
- Restructure after each access
  self-adjusting tree

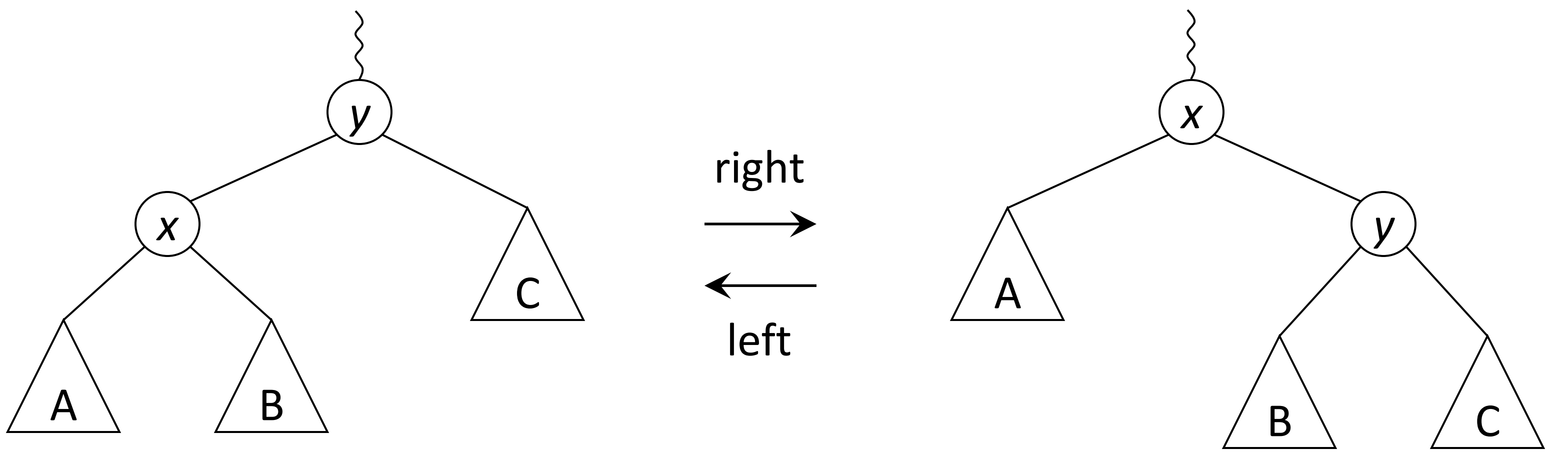
Store balance information in nodes, guarantee O(log *n*) height

After insert/delete, restore balance bottom-up (top-down):

- Update balance information
- Restructure along access path



Restructuring primitive: Rotation



Preserves symmetric order

Changes heights

Takes O(1) time

Known Balanced BSTs

AVL trees – small height

red-black trees – little rebalancing

weight balanced trees

binary B-trees

etc.

Goal: small height, little rebalancing, simple algorithms

Ranked Binary Trees

Each node has an integer rank

Convention: leaves have rank 0, missing nodes have rank -1

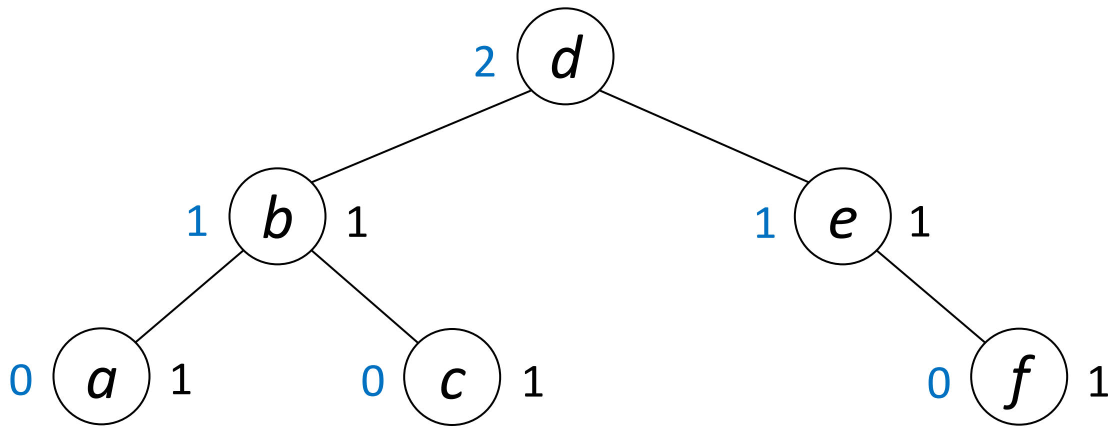
rank difference of a child =

rank of parent – rank of child

i-child: node of rank difference *i*

i,*j*-node: children have rank differences *i* and *j*

Example of a ranked binary tree



If all rank differences are positive, rank ≥ height

Rank-Balanced Trees

AVL trees: every node is a 1,1- or 1,2-node

Rank-balanced trees: every node is a 1,1-, 1,2-, or 2,2-node (rank differences are 1 or 2)

Red-black trees: all rank differences are 0 or 1, no 0-child is the parent of another

Each needs one balance bit per node.

Basic height bounds

n_k = minimum n for rank k

AVL trees:

$n_0 = 1$, $n_1 = 2$, $n_k = n_{k-1} + n_{k-2} + 1$

$n_k = F_{k+3} - 1 \Rightarrow k \leq \log_\phi n \approx 1.44 \lg n$

Rank-balanced trees:

$n_0 = 1$, $n_1 = 2$, $n_k = 2n_{k-2}$,

$n_k = 2^{\lceil k/2 \rceil} \Rightarrow k \leq 2 \lg n$

Same height bound for red-black trees

F_k = k^{th} Fibonacci number

$\phi = (1 + \sqrt{5})/2$

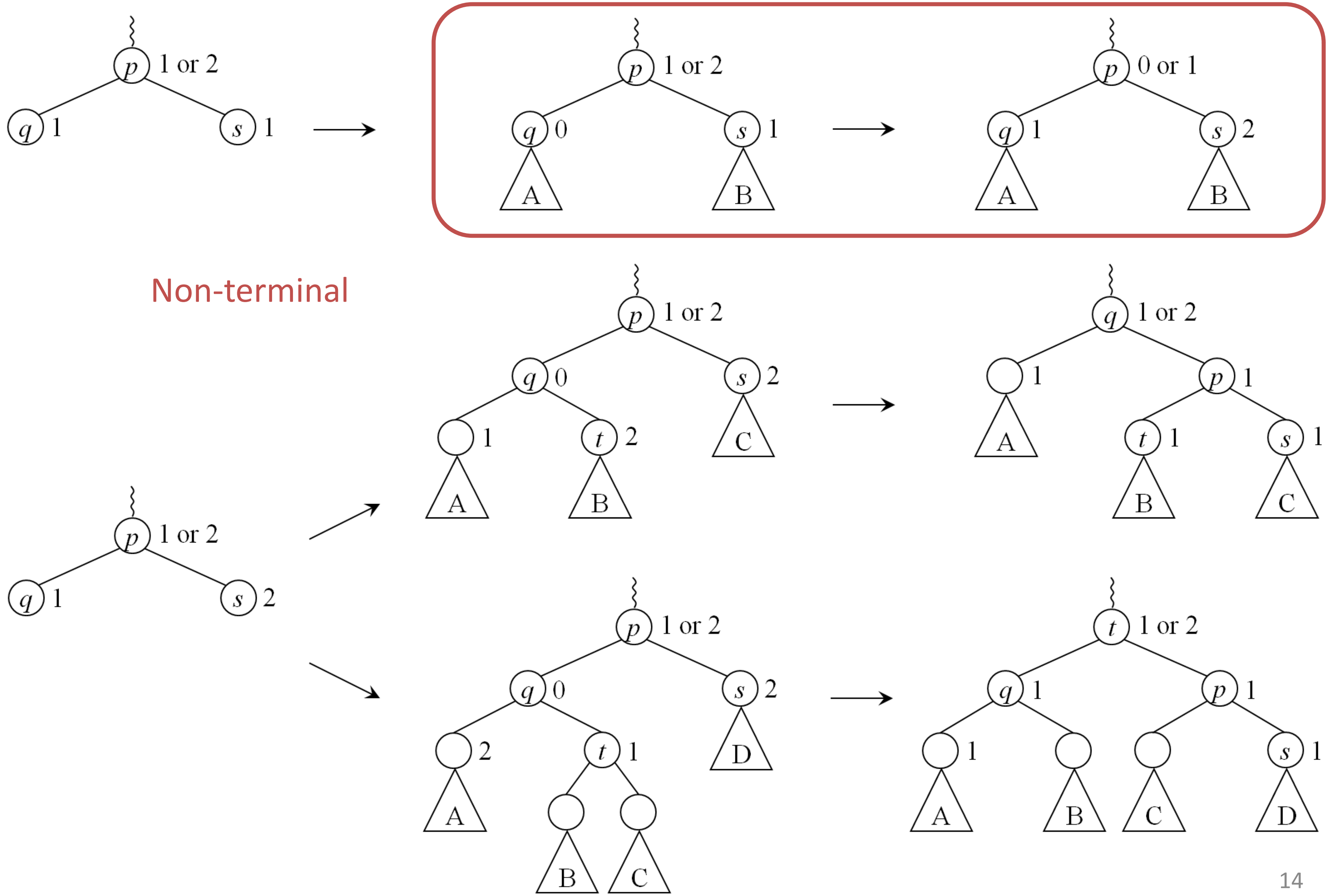
$F_{k+2} > \phi^k$

Rank-balanced trees: Insertion

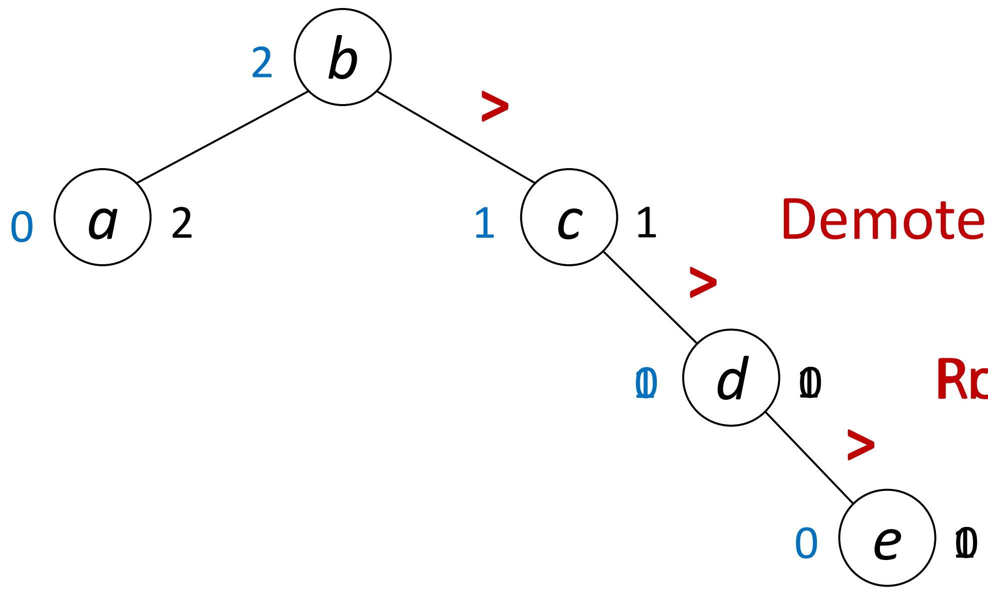
A new leaf *q* has a rank of zero

If the parent *p* of *q* was a leaf before, *q* is a 0-child and violates the rank rule

Insertion Rebalancing



Insertion example



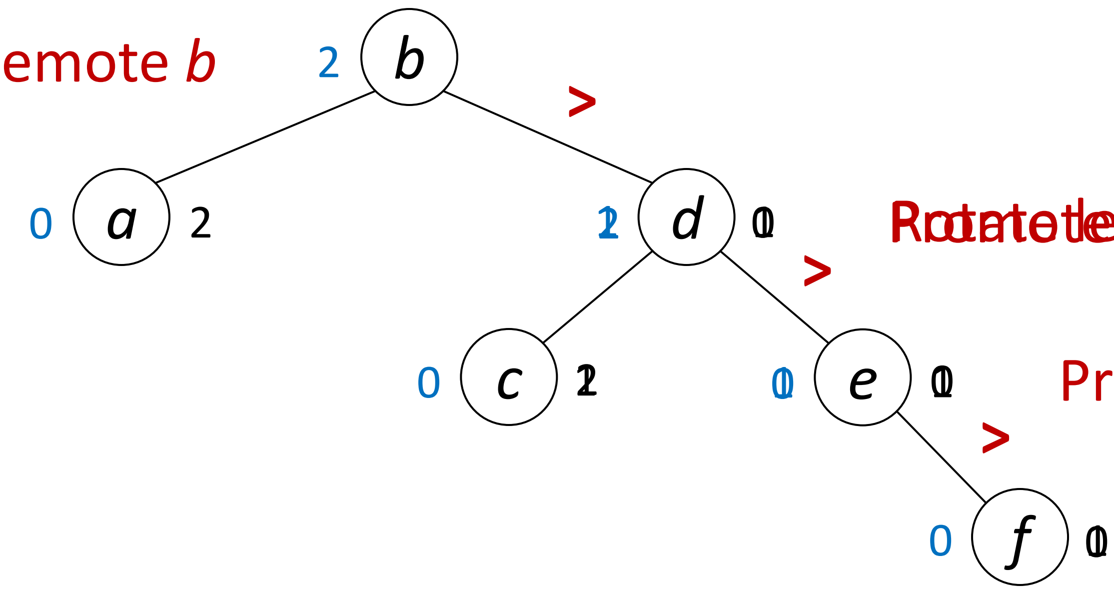
Demote *c*

Rotate left at *d*

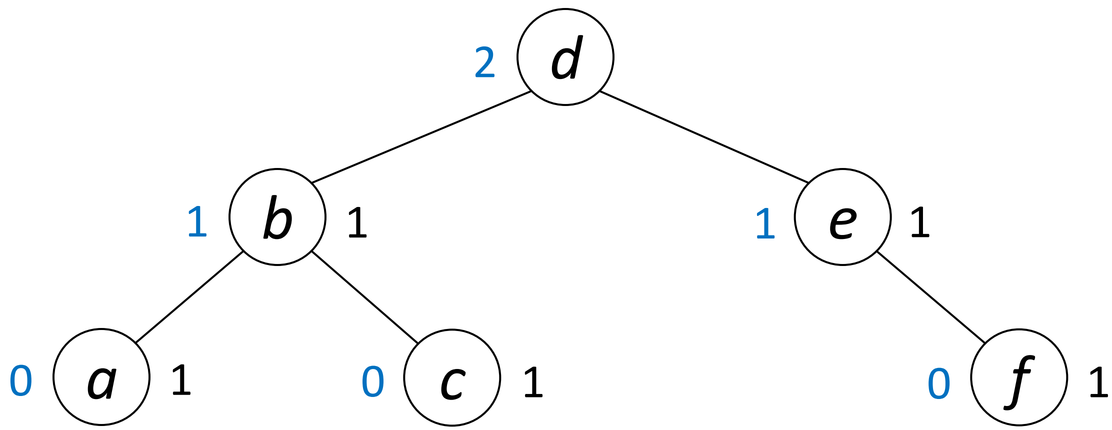
Promote *d*

Insert *e*

Insertion example



Insertion example



Insert *f*

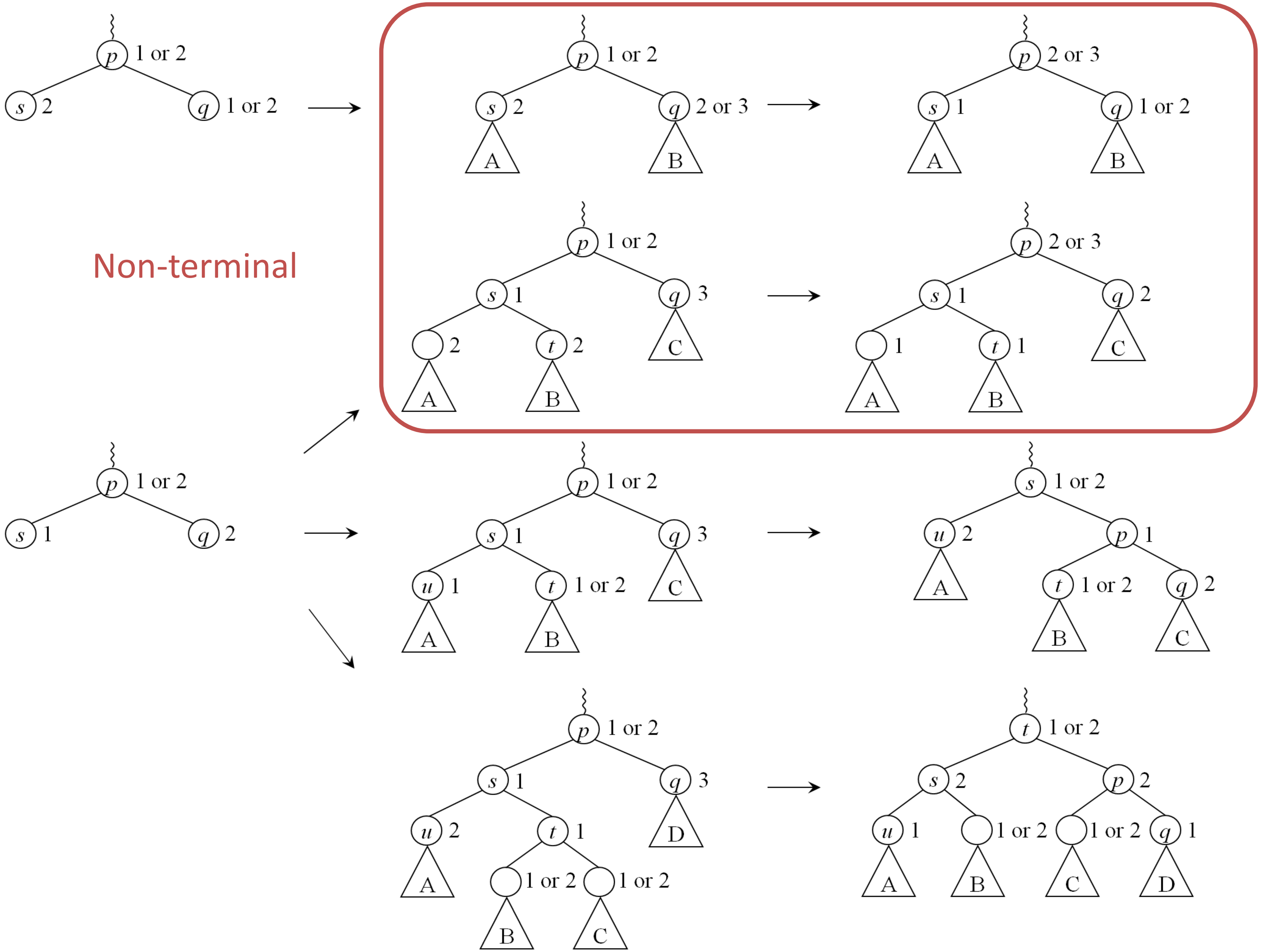
Rank-balanced trees: Deletion

If node has two children, swap with symmetric-order successor or predecessor

- Becomes a leaf (just delete) or node with one child (replace with child)

If node *q* replaces the deleted node and *p* is its parent, a violation occurs if *p* is a leaf of rank one or *q* is a 3-child

Deletion Rebalancing



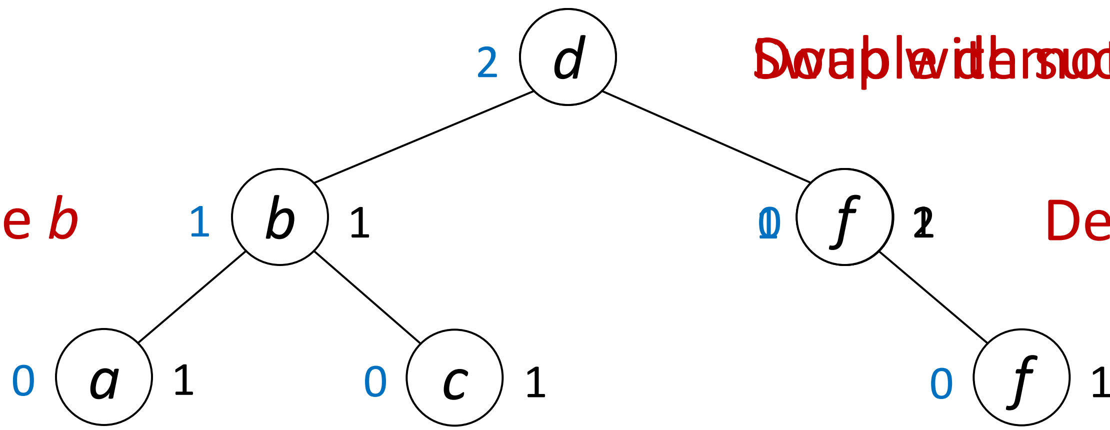
Deletion example

Swap with successor

Double demote

Demote *b*

Delete



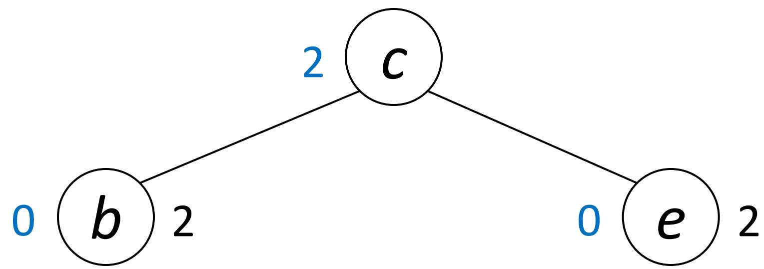
Delete *d*

Delete *f*

Double rotate at *c*

Double promote *c*

Deletion example

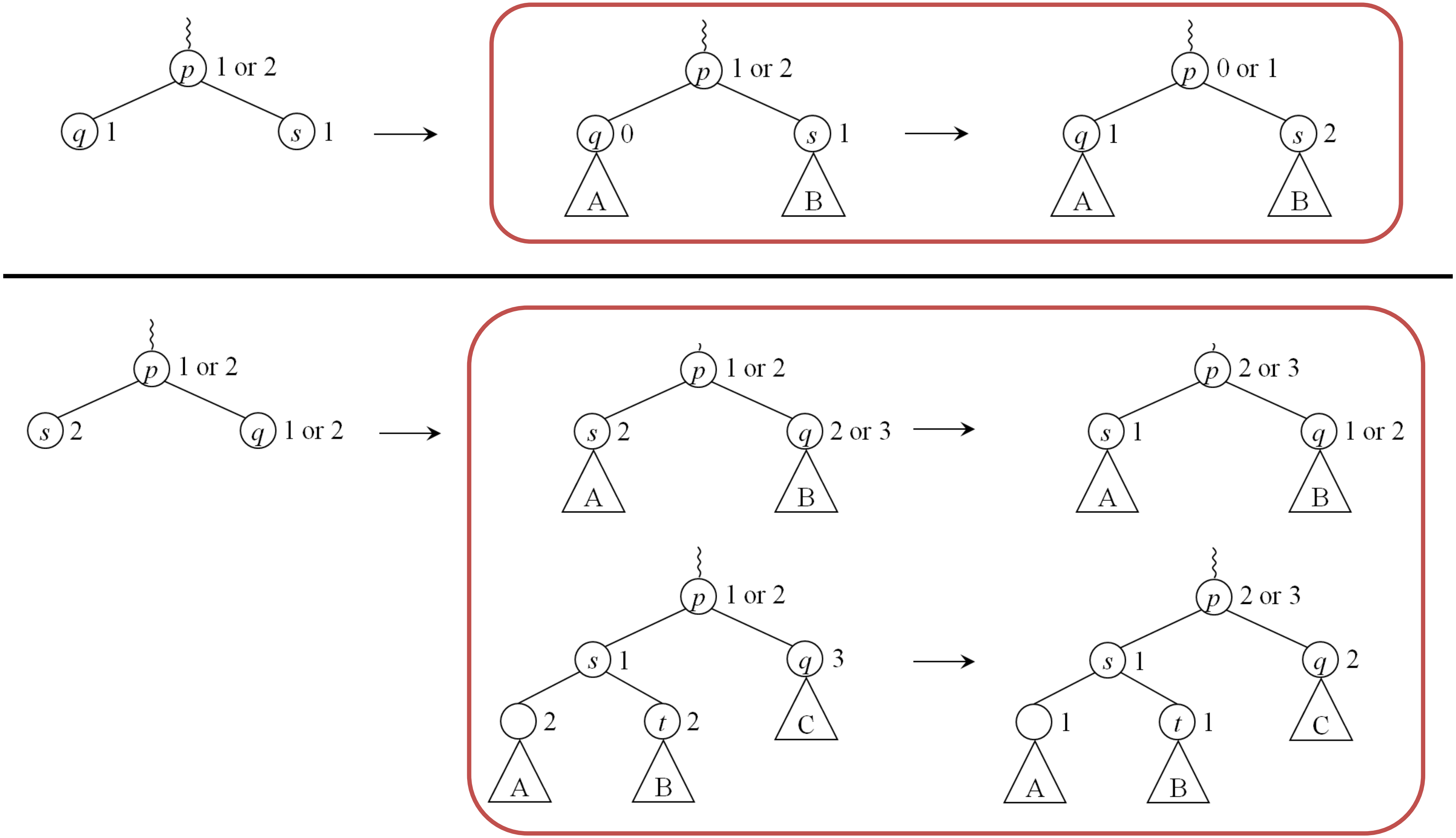


Delete *f*

Rebalancing Time

Theorem. *A rank-balanced tree built by m insertions and d deletions does at most* 3m + 6d *rebalancing steps.*

Proof idea: Make non-terminating cases release potential



Proof. Define the potential of a node:

1	if it is a 1,1-node
2	if it is a 2,2-node
Zero	otherwise

Potential of tree = sum of potentials of nodes

Non-terminating steps are free

Terminating steps increase potential by O(1)

Rank-Balanced Trees

height $\leq 2\lg n$

≤ 2 rotations per rebalancing

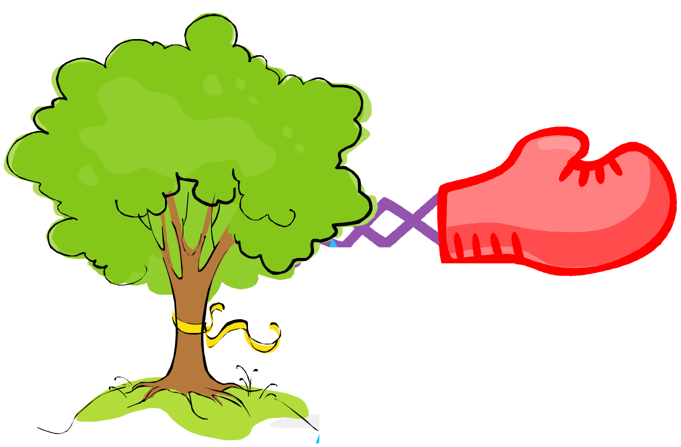
O(1) amortized rebalancing time

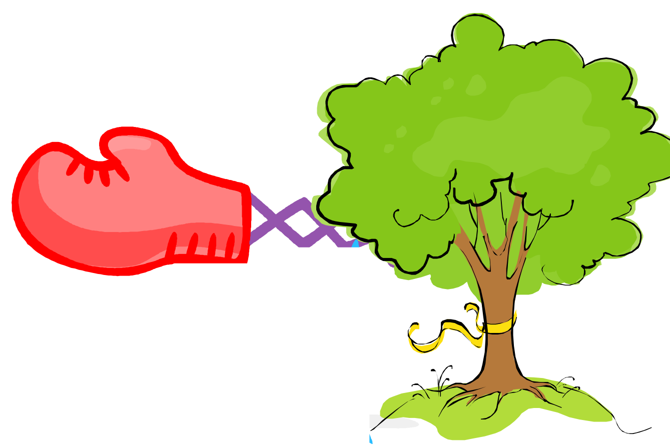
Red-Black Trees

height $\leq 2\lg n$

≤ 3 rotations per rebalancing

O(1) amortized rebalancing time





Tree Height

Sequential Insertions:

rank-balanced	red-black
height = lg n (best)	height = 2lg n (worst)

Tree Height

Theorem 1. *A rank-balanced tree built by m insertions intermixed with arbitrary deletions has height at most* $\log_\phi m$.

If $m = n$, same height as AVL trees

Overall height is $\min\{2\lg n, \log_\phi m\}$

Proof idea: Exponential potential function

Exploit the exponential structure of the tree

Proof. Give a node a count of 1 when inserted. Define the potential of a node:

Total count in its subtree

When a node is deleted, add its count to parent

Φ_k = minimum potential of a node of rank k

Claim:

$\Phi_0 = 1$, $\Phi_1 = 2$, $\Phi_k = 1 + \Phi_{k-1} + \Phi_{k-2}$ for $k > 1$

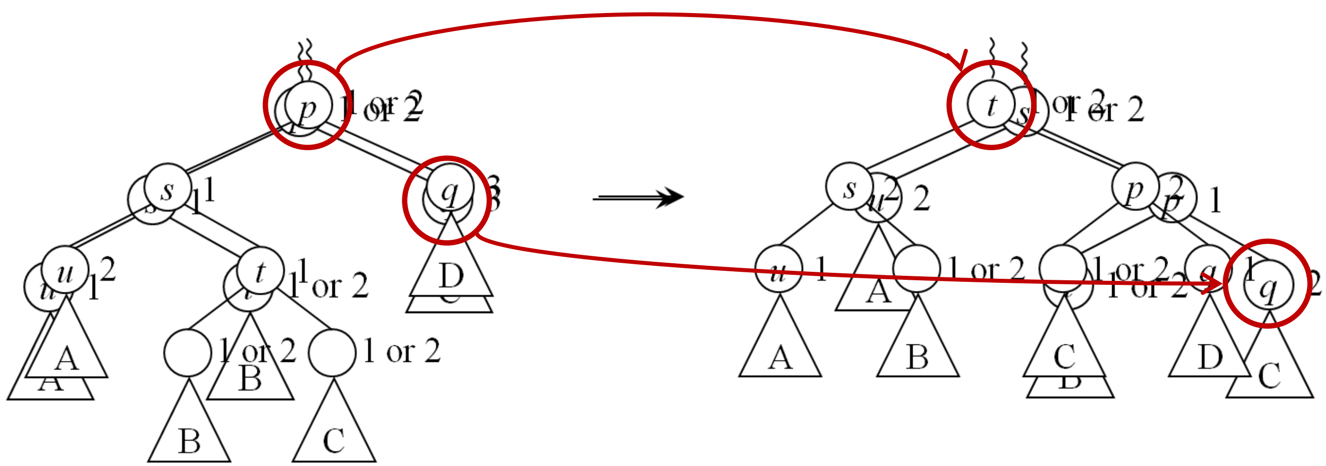
$\Rightarrow m \geq F_{k+3} - 1 \geq \phi^k$

Show that $\Phi_k = 1 + \Phi_{k-1} + \Phi_{k-2}$ for k > 1

Easy to show for 1,1- and 1,2-nodes

Harder for 2,2-nodes (created by deletions)

But counts are inherited



Rebalancing Frequency

How high does rebalancing propagate?

O(m + d) rebalancing steps total, which implies

$\Rightarrow$ O((m + d)/k) insertions/deletions at rank k

Actually, we can show something much stronger

Rebalancing Frequency

Theorem. *In a rank-balanced tree built by m insertions and d deletions, the number of rebalancing steps of rank k is at most* $O((m + d)/2^{k/3})$.

Good for concurrent workloads

Proof. Define the potential of a node of rank k:

b^k if it is a 1,1- or 2,2-node

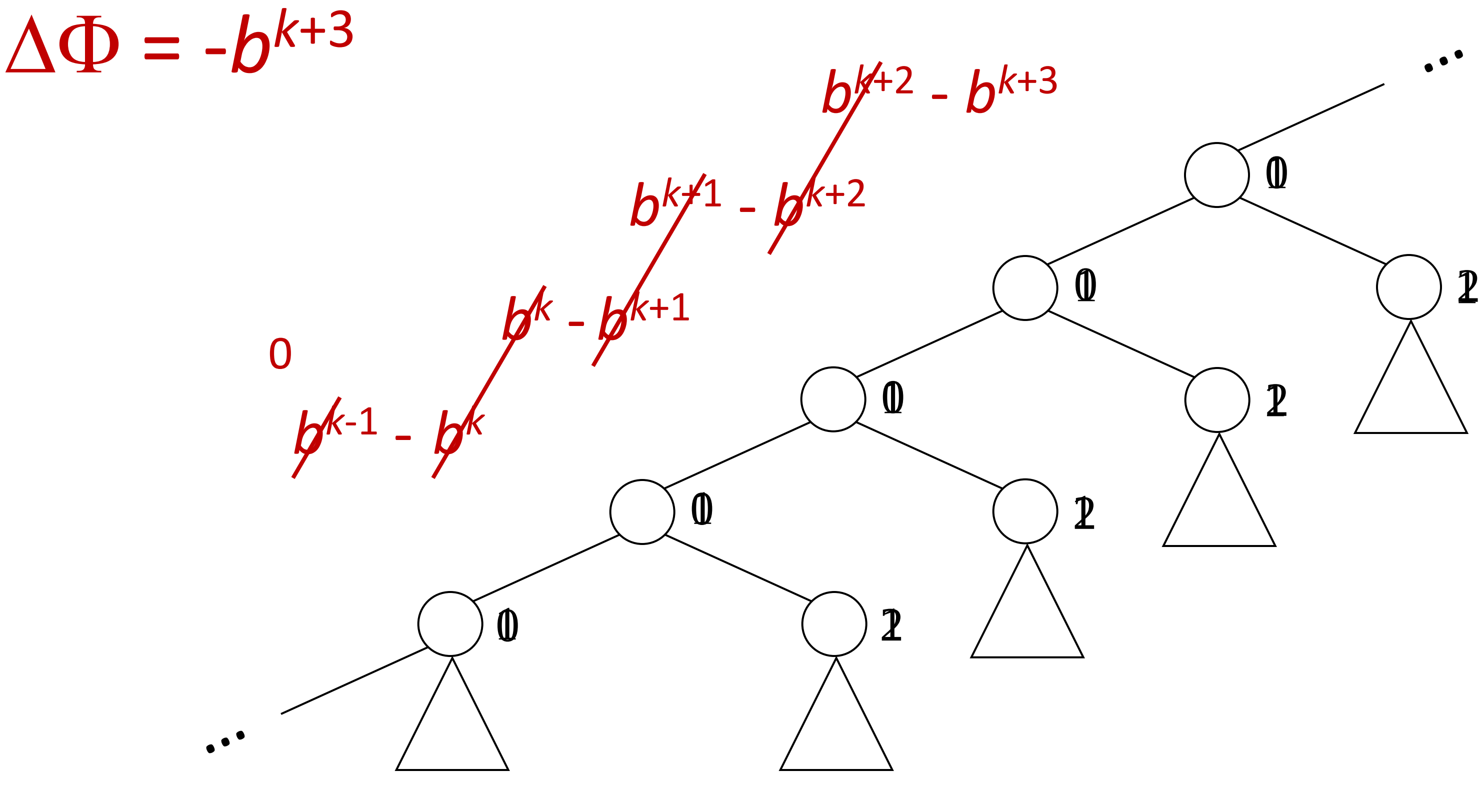
b^{k-2} if it is a 1,2-node

where $b = 2^{1/3}$

Potential change in non-terminal steps telescopes

Combine this effect with initialization and terminal step

Telescoping potential:



Truncate growth of potential at rank $k - 3$:

Nodes of rank < k–3 have same potential

Nodes of rank ≥ k–3 have potential as if rank k–3

Rebalancing step of rank k reduces the potential by b^{k-3}

Same idea should work for red-black trees (we think)

Summary

Rank-balanced trees are a relaxation of AVL trees with behavior theoretically as good as red-black trees and better in important ways.

Especially height bound of min{2lg n, $\log_\phi$ m}

Exponential potential functions yield new insights into the efficiency of rebalancing

Ravl Trees

An idea from practice…

Joint work with S. Sen

Balanced Search Trees

AVL trees
rank-balanced trees
red-black trees
weight balanced trees
Binary B-trees
} binary

2,3 trees
B trees
} multiway

etc.

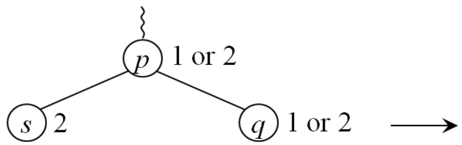
Common problem: Deletion is a pain!

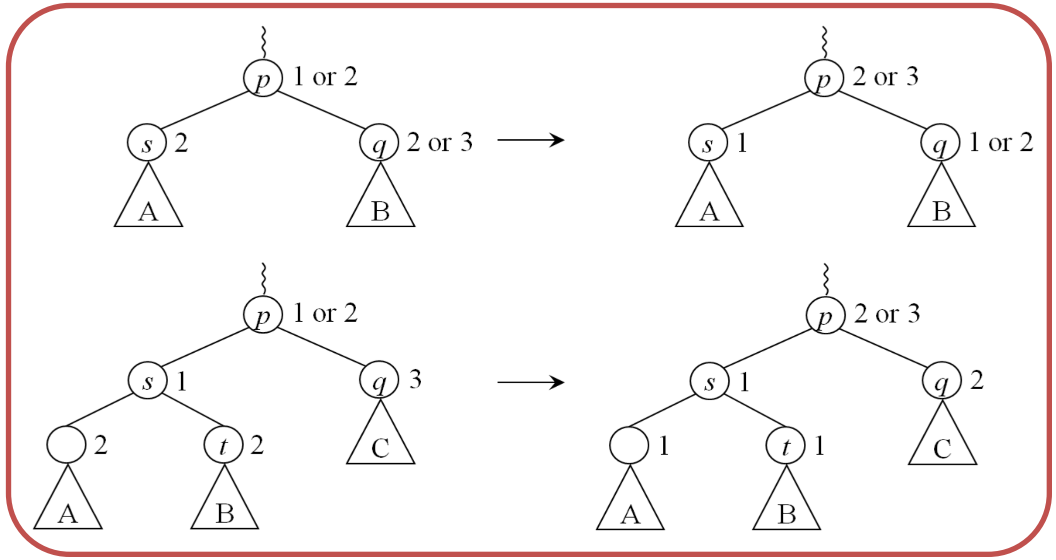
Deletion in balanced search trees

Deletion is problematic

- May need to swap item with its successor/ predecessor
- Rebalancing is more complicated than during insertion
- Synchronization reduces available parallelism [Gray and Reuter]

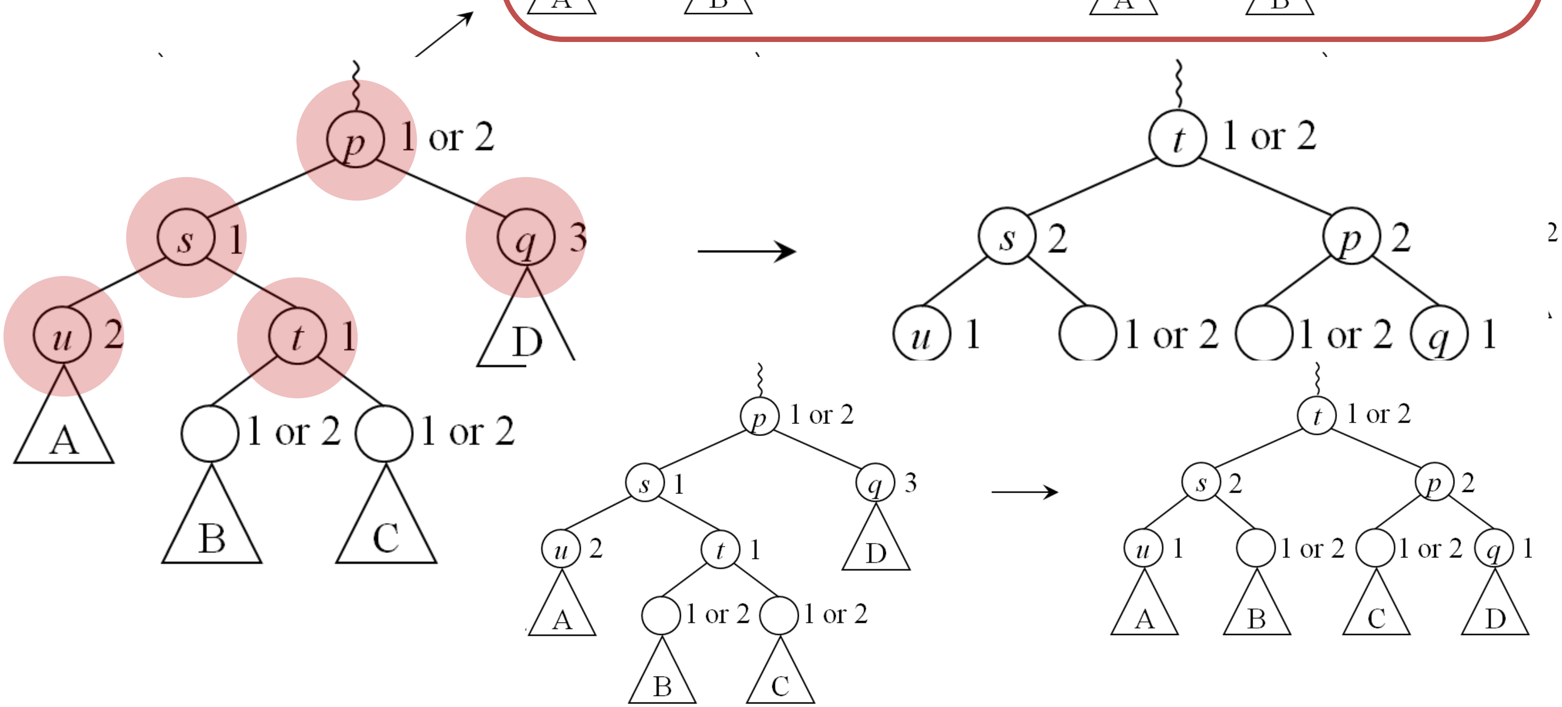
Example: Rank-balanced trees





Non-terminal

Synchronization ☹



Deletion rebalancing: solutions?

Don’t discuss it!

- Textbooks

Don’t do it!

- Berkeley DB and other database systems
- Unnamed database provider…

Storytime…

Deletion Without Rebalancing

Is this a good idea?

Empirical and average-case analysis suggests yes for B+ trees (database systems)

How about binary trees?

Failed miserably in real application with red-black trees

No worst-case analysis, probably because of assumption that it is very bad

Deletion Without Rebalancing

We present such balanced search trees, where:

- Height remains logarithmic in *m*, the number of insertions
- Amortized time per insertion or deletion is O(1)
- Rebalancing affects nodes exponentially infrequently in their heights

Binary trees: use Ω(loglog *m*) bits of balance information per node

Red-black, AVL, rank-balanced trees use only one bit!

Similar results hold for B^+ trees, easier [ISAAC 2009]

Ravl(relaxed AVL) Trees

AVL trees: every node is a 1,1- or 1,2-node

Rank-balanced trees: every node is a 1,1-, 1,2-, or 2,2-node (rank differences are 1 or 2)

Red-black trees: all rank differences are 0 or 1, no 0-child is the parent of another

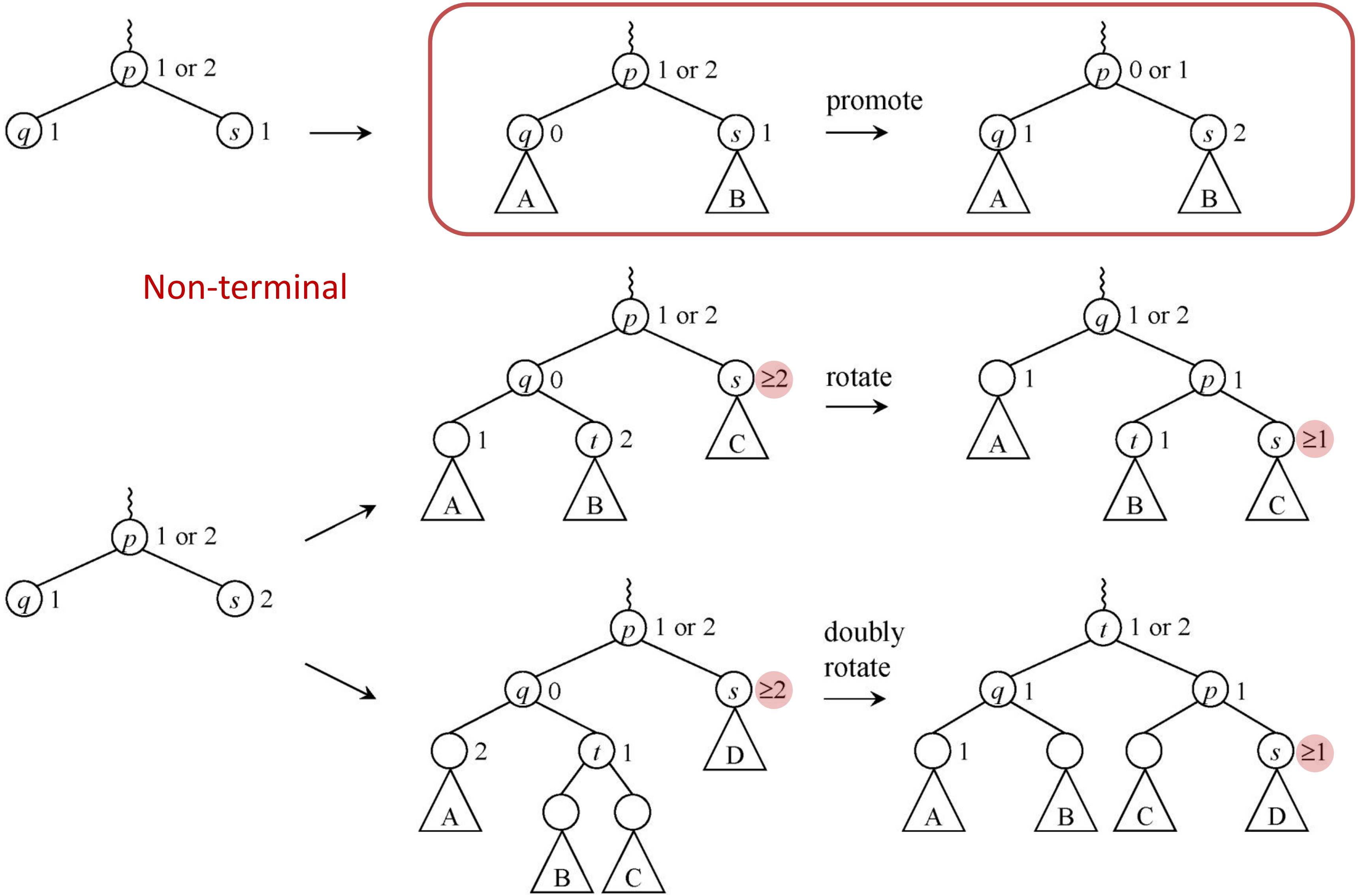
Ravl trees: every rank difference is positive

- Any tree is a ravl tree; efficiency comes from design of operations

Ravl trees: Insertion

Same as rank-balanced trees (AVL trees)!

Insertion Rebalancing



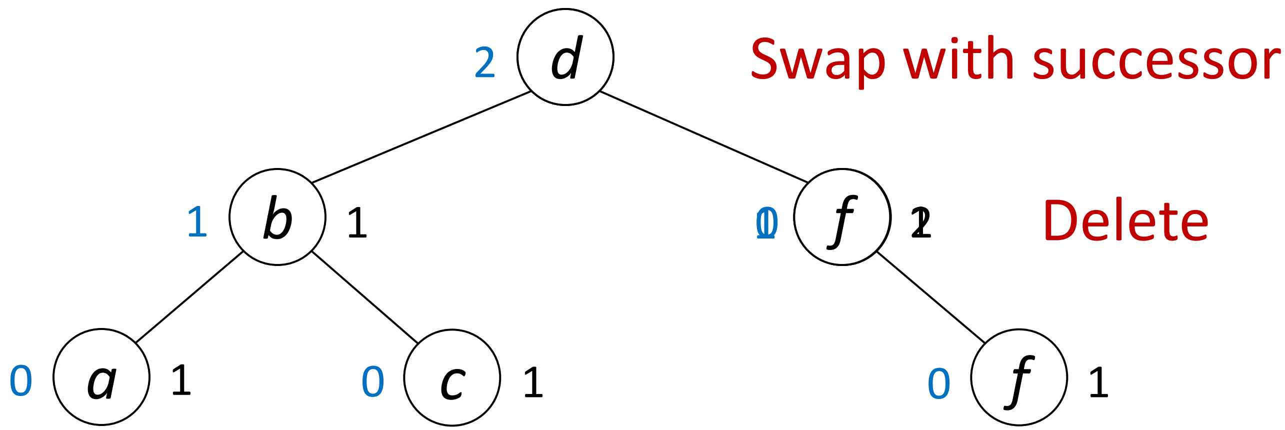
Ravl trees: Deletion

☺

If node has two children, swap with symmetric-order successor or predecessor. Delete. Replace by child.

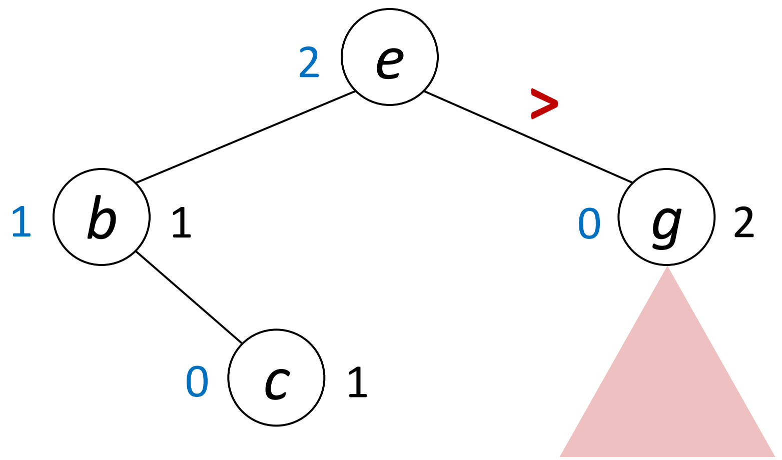
Swapping not needed if all data in leaves (external representation).

Deletion example



Delete *f* *d*

Deletion example



Insert *g*

Tree Height

Theorem 1. *A ravl tree built by m insertions intermixed with arbitrary deletions has height at most* $\log_\phi m$.

$$\phi = (1 + \sqrt{5})/2$$

Compared to standard AVL trees:

If *m* = n, height is the same

If *m* = O(*n*), height within an additive constant

If *m* = *poly*(*n*), height within a constant factor

Proof idea: exponential potential function

Exploit the exponential structure of the tree

Proof. Let F_k be the k^{th} Fibonacci number.
Define the potential of a node of rank k:

F_{k+2}	if it is a 0,1-node
F_{k+1}	if it has a 0-child but is not a 0,1-node
F_k	if it is a 1,1 node
Zero	otherwise

Potential of tree = sum of potentials of nodes

Recall: $F_0 = 1$, $F_1 = 1$, $F_k = F_{k-1} + F_{k-2}$ for $k > 1$

$$F_{k+2} > \phi^k$$

Proof. Let F_k be the k^{th} Fibonacci number. Define the potential of a node of rank k:

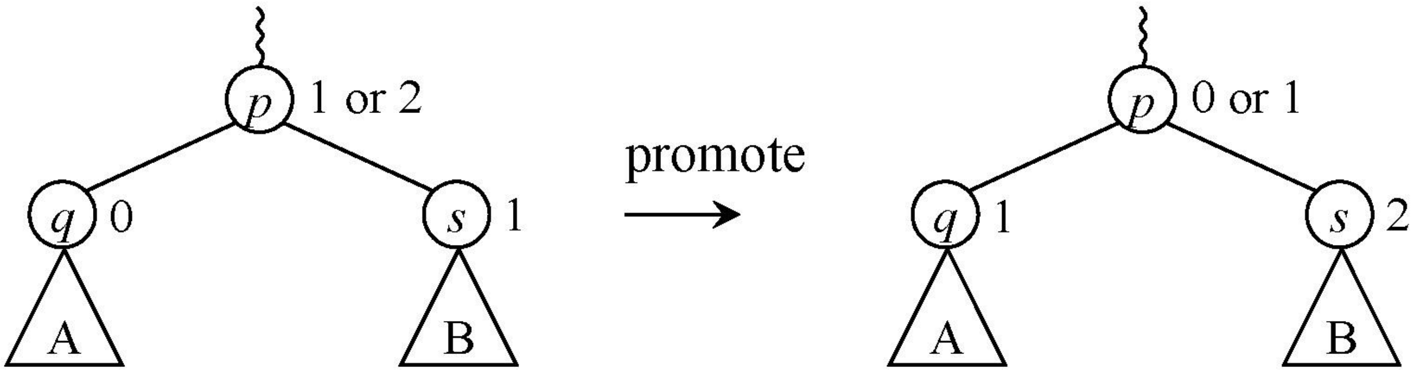
F_{k+2}	if it is a 0,1-node
F_{k+1}	if it has a 0-child but is not a 0,1-node
F_k	if it is a 1,1 node
Zero	otherwise

Deletion does not increase potential

Insertion increases potential by ≤ 1, so total potential is $\leq m - 1$

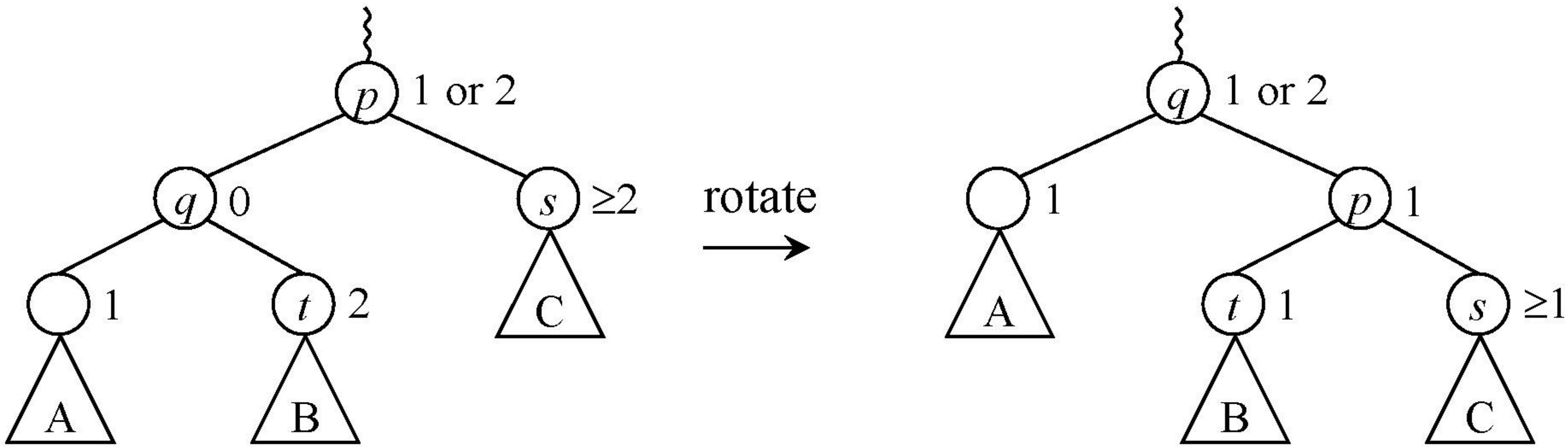
Rebalancing steps don't increase the potential

Consider a rebalancing step of rank k:



$F_{k+1} + F_{k+2}$	$F_{k+3} + 0$
$0 + F_{k+2}$	$F_{k+2} + 0$
$F_{k+2} + 0$	$0 + 0$

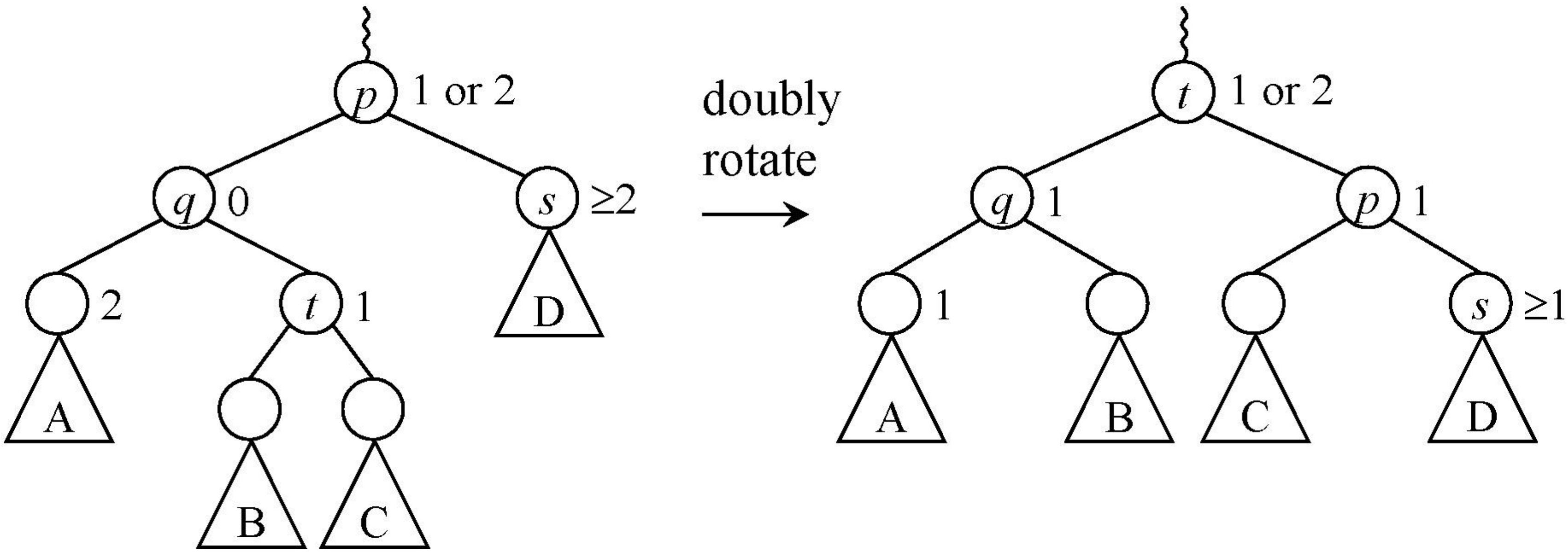
Consider a rebalancing step of rank *k*:



$F_{k+1} + 0$

$F_k + F_{k-1}$

Consider a rebalancing step of rank k:



$F_{k+1} + 0 + 0$

$F_k + F_{k-1} + 0$

If rank of root is *r*, there was a promotion of rank *k* that did not create a 1,1-node, for $0 < k < r-1$

Total decrease in potential:

$$\sum_{k=2}^{r+1} F_k = F_{r+3} - 2$$

Since potential is always non-negative:

$$m - 1 \geq F_{r+3} - 2$$

$$m \geq F_{r+3} - 1 \geq F_{r+2} \geq \phi^r$$

Rebalancing Frequency

Theorem 2. *In a ravl tree built by m insertions intermixed with arbitrary deletions, the number of rebalancing steps of rank k is at most* $(m-1)/F_k \leq (m-1)/\phi^{k-2}$.

$\Rightarrow$ O(1) amortized rebalancing steps

Proof. Truncate the potential function:

Nodes of rank < k have same potential

Nodes of rank $\geq k$ have zero potential (with one exception for rank = k)

Deletion does not increase potential

Insertion increases potential by ≤ 1, so total potential is $\leq m - 1$

Rebalancing steps don’t increase the potential

Proof. Truncate the potential function:

Nodes of rank < k have same potential

Nodes of rank $\geq k$ have zero potential (with one exception for rank = k)

Step of rank k preceded by promotion of rank $k - 1$, which reduces potential by:

F_{k+1} if stop or promotion at rank k

$F_{k+1} - F_{k-1} = F_k$ if (double) rotation at rank k

Potential can decrease by at most $(m-1)/F_k$

Disadvantage of Ravl Trees?

Tree height may be $\omega(\log n)$

Only happens when ratio of deletions to insertions approaches 1, but may be a concern for some applications

Address by periodically rebuilding the tree

Periodic Rebuilding

Rebuild the tree (all at once or incrementally) when rank *r* of root ($\geq$ tree height) is too high

Rebuild when $r > \log_\phi n + c$ for fixed $c > 0$:

- Rebuilding time is $O(1/(\phi^c - 1))$ per deletion

Then tree height is always $\log_\phi n + O(1)$

Constant bits?

Ravl tree stores Ω(loglog n) balance bits per node

Various methods that use O(1) bits fail (see counterexamples in paper)

Main problem: deletion can increase the ranks of nodes; if we force all deletions to occur at leaves, then an O(1)-bit scheme exists

- But now a deletion may require multiple swaps

Summary

Deletion without rebalancing in binary trees has good worst-case properties, including:

- Logarithmic height bound
- Exponentially infrequent node updates

With periodic rebuilding, can maintain height logarithmic in *n*

Open problem: Requires $\Omega(\log\log n)$ balance bits per node?

Experiments

Preliminary Experiments

Compared three trees that achieve O(1) amortized rebalancing time

- Red-black trees
- Rank-balanced trees
- Ravl trees

Performance in practice depends on the workload!

Preliminary Experiments

Test	Red-black trees				Rank-balanced trees				Ravl trees			
	# rots $\times 10^6$	# bals $\times 10^6$	avg. pLen	max. pLen	# rots $\times 10^6$	# bals $\times 10^6$	avg. pLen	max. pLen	# rots $\times 10^6$	# bals $\times 10^6$	avg. pLen	max. pLen
Random	26.44	116.07	10.47	15.63	29.55	133.74	10.39	15.09	14.32	80.61	11.11	16.75
Queue	50.32	285.13	11.38	22.50	50.33	184.53	11.20	14.00	33.55	134.22	11.38	14.00
Working set	41.71	185.35	10.51	16.18	43.69	159.69	10.45	15.35	28.00	119.92	11.20	16.64
Static Zipf	25.24	112.86	10.41	15.46	28.27	130.93	10.34	15.05	13.48	78.03	11.12	17.68
Dynamic Zipf	23.18	103.48	10.48	15.66	26.04	125.99	10.40	15.16	12.66	74.28	11.11	16.84

2^{13} nodes, 2^{26} operations

No periodic rebuilding in ravl trees

Preliminary Experiments

Test	Red-black trees				Rank-balanced trees				Ravl trees			
	# rots $\times 10^6$	# bals $\times 10^6$	avg. pLen	max. pLen	# rots $\times 10^6$	# bals $\times 10^6$	avg. pLen	max. pLen	# rots $\times 10^6$	# bals $\times 10^6$	avg. pLen	max. pLen
Random	26.44	116.07	10.47	15.63	29.55	133.74	10.39	15.09	14.32	80.61	11.11	16.75
Queue	50.32	285.13	11.38	22.50	50.33	184.53	11.20	14.00	33.55	134.22	11.38	14.00
Working set	41.71	185.35	10.51	16.18	43.69	159.69	10.45	15.35	28.00	119.92	11.20	16.64
Static Zipf	25.24	112.86	10.41	15.46	28.27	130.93	10.34	15.05	13.48	78.03	11.12	17.68
Dynamic Zipf	23.18	103.48	10.48	15.66	26.04	125.99	10.40	15.16	12.66	74.28	11.11	16.84

rank-balanced: 8.2% more rots, 0.77% more bals

ravl: 42% fewer rots, 35% fewer bals

Preliminary Experiments

Test	Red-black trees				Rank-balanced trees				Ravl trees			
	# rots $\times 10^6$	# bals $\times 10^6$	avg. pLen	max. pLen	# rots $\times 10^6$	# bals $\times 10^6$	avg. pLen	max. pLen	# rots $\times 10^6$	# bals $\times 10^6$	avg. pLen	max. pLen
Random	26.44	116.07	10.47	15.63	29.55	133.74	10.39	15.09	14.32	80.61	11.11	16.75
Queue	50.32	285.13	11.38	22.50	50.33	184.53	11.20	14.00	33.55	134.22	11.38	14.00
Working set	41.71	185.35	10.51	16.18	43.69	159.69	10.45	15.35	28.00	119.92	11.20	16.64
Static Zipf	25.24	112.86	10.41	15.46	28.27	130.93	10.34	15.05	13.48	78.03	11.12	17.68
Dynamic Zipf	23.18	103.48	10.48	15.66	26.04	125.99	10.40	15.16	12.66	74.28	11.11	16.84

rank-balanced: 0.87% shorter apl, 10% shorter mpl

ravl: 5.6% longer apl, 4.3% longer mpl

Ongoing/future experiments

Trees:

- AVL trees
- Binary B-trees (Sedgewick's implementation)

Deletion schemes:

- Lazy deletion (avoids swapping, uses extra space)

Tests:

- Real workloads!
- Degradation over time

The End