Contents

Section outline

1. **Red-Black trees**
   - Definition
   - Simple RBT properties
   - Maximally skewed RBT
   - RBT insertion with rotation based corrections
   - Correction with colour change and rotation
   - RBT deletion and colour correction
   - Practice problems

Definition

Definition (Red-Black Tree [RBT])

A red-black tree (RBT), developed by Guibas and Sedgewick, 1978, is a binary search tree that satisfies the following red-black properties:

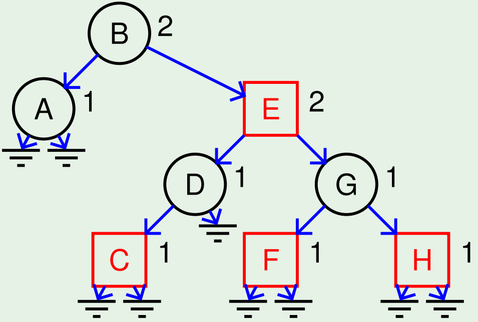
- Every node has a color that is either red or black.
- Every leaf (NULL pointer treated as a leaf node) is black.
- If a node is red, both children are black.
- Every path from a given node down to any descendant leaf contains the same number of black nodes.
- The root of the tree is black (this property is sometimes dropped).

Definition (Black height [bh]of a node)

The number of black nodes on any path to a leaf (not including the initial node but including the leaf).

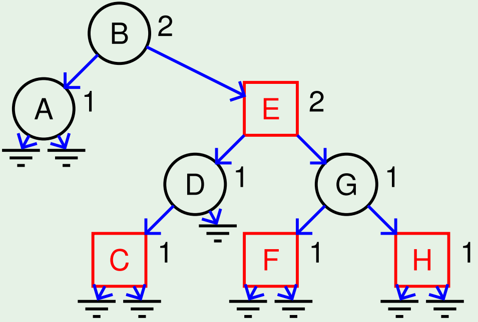
Simple RBT properties

Example (An RBT of bh 2)



Simple RBT properties

Example (An RBT of bh 2)

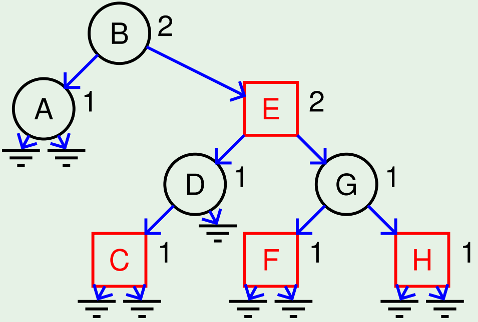


Search in RBT

Same as in BST

Simple RBT properties

Example (An RBT of bh 2)



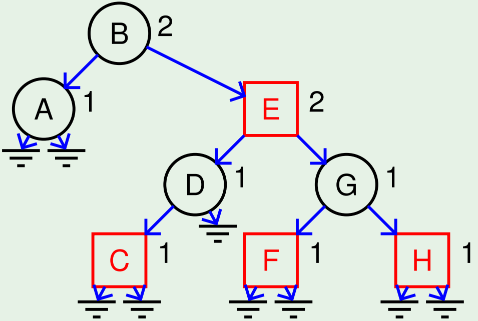
Search in RBT

Same as in BST

- Perfect binary tree of height h has $2^{h+1} - 1$ nodes

Simple RBT properties

Example (An RBT of bh 2)



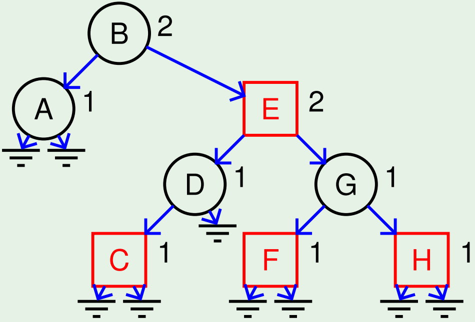
- Perfect binary tree of height h has $2^{h+1} - 1$ nodes
- Corresponds to the minimum number of nodes in a RBT of bh $(h + 1)$ – having only black nodes

Search in RBT

Same as in BST

Simple RBT properties

Example (An RBT of bh 2)

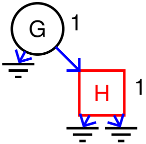


Search in RBT

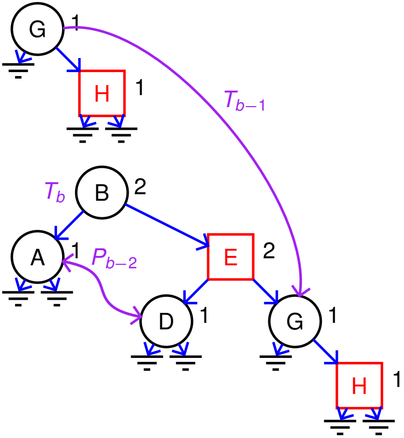
Same as in BST

- Perfect binary tree of height h has $2^{h+1} - 1$ nodes
- Corresponds to the minimum number of nodes in a RBT of bh $(h + 1)$ – having only black nodes
- Tree can be inflated without altering its bh, inserting red nodes between adjacent black nodes
- Height of inflated tree is $2h + 1$, maximum number of nodes in a RBT of bh h is $2^{2(h+1)} - 1$

Maximally skewed RBT

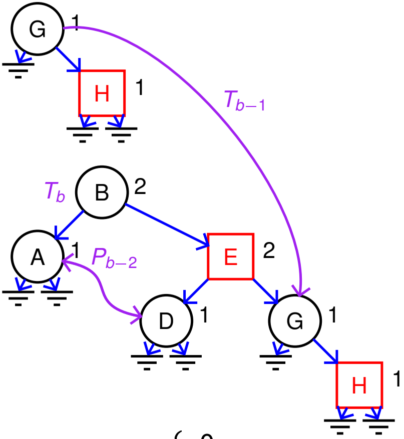


Maximally skewed RBT



- Let the RBT rooted at T have bh of b
- L-ST of T is a RBT of bh $b-1$ without any red nodes – a perfect binary tree of height $b-2$ having $2_{b-1}-1$ nodes
- R-Child of T is a red node with:
  - L-ST as a RBT of bh $b-1$ without any red nodes
  - R-ST as a maximally skewed RBT of bh $b-1$

Maximally skewed RBT

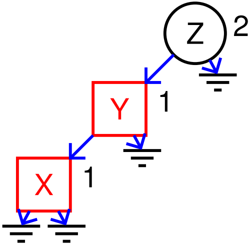


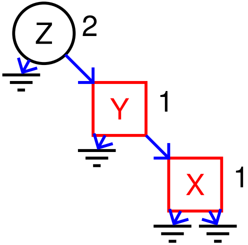
- Let the RBT rooted at T have bh of b
- L-ST of T is a RBT of bh $b-1$ without any red nodes – a perfect binary tree of height $b-2$ having $2_{b-1}-1$ nodes
- R-Child of T is a red node with:
  - L-ST as a RBT of bh $b-1$ without any red nodes
  - R-ST as a maximally skewed RBT of bh $b-1$

- $N(b) = \begin{cases} 0 & b = 0 \\ 2 + 2 \cdot (2^{b-1} - 1) + N(b-1) & b > 0 \end{cases}$
- By substitution, $N(b) = \sum_{j=1}^{b} 2^j = 2^{b+1} - 2$

RBT insertion with rotation based corrections

- First insert key as a red node in a leaf position as in a BST
- If insertion happens below a black node, no problem, otherwise red-red violation



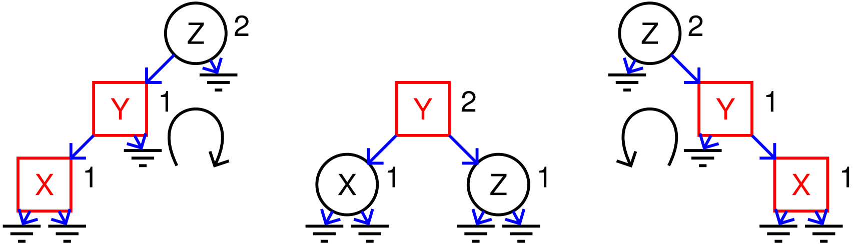


RBT insertion with rotation based corrections

- First insert key as a red node in a leaf position as in a BST
- If insertion happens below a black node, no problem, otherwise red-red violation
- Violation may be corrected via bh preserving rotations

NB corrections propagate upward as root changes from black to red

- Finally, blacken root if it became red

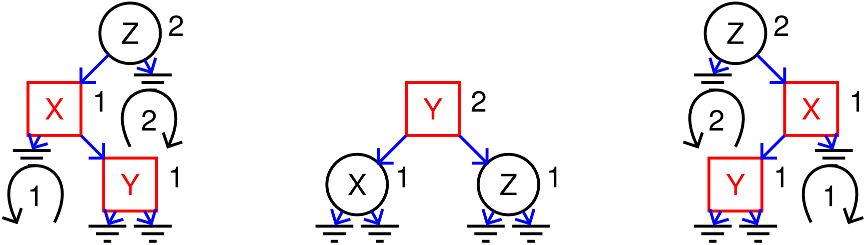


RBT insertion with rotation based corrections

- First insert key as a red node in a leaf position as in a BST
- If insertion happens below a black node, no problem, otherwise red-red violation
- Violation may be corrected via bh preserving rotations

NB corrections propagate upward as root changes from black to red

- Finally, blacken root if it became red

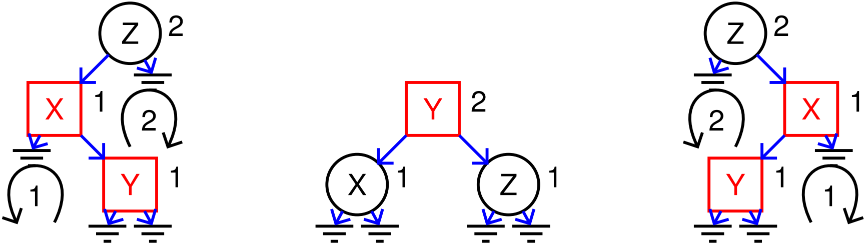


RBT insertion with rotation based corrections

- First insert key as a red node in a leaf position as in a BST
- If insertion happens below a black node, no problem, otherwise red-red violation
- Violation may be corrected via bh preserving rotations

NB corrections propagate upward as root changes from black to red

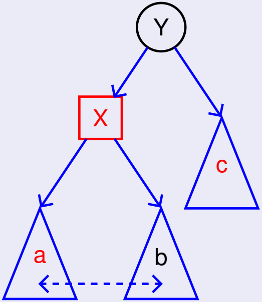
- Finally, blacken root if it became red
- Complexity: $O(\lg n)$



Correction with colour change and rotation

Superior node in red-red violation has red sibling

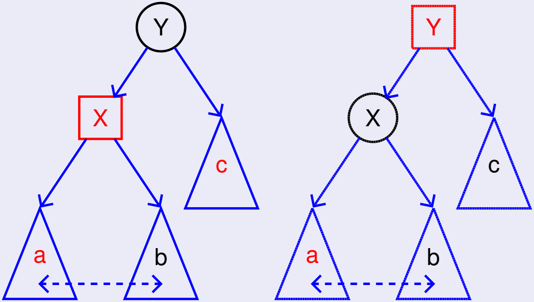
- These siblings must have a black common parent



Correction with colour change and rotation

Superior node in red-red violation has red sibling

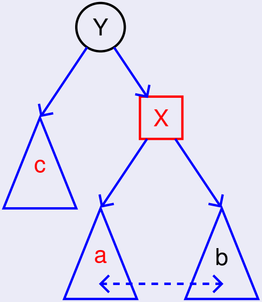
- These siblings must have a black common parent
- Transmit the black colour of the root to both children (bh is preserved) and colour the root red
- Root has changed from black to read, so new violations possible, changes will propagate up



Correction with colour change and rotation

Superior node in red-red violation has red sibling

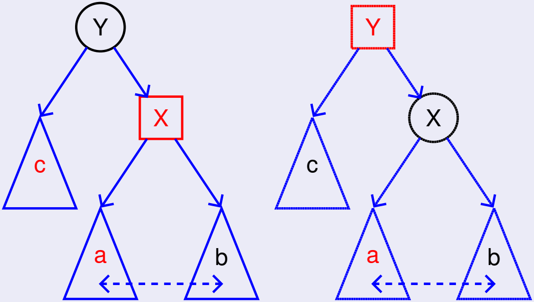
- These siblings must have a black common parent
- Transmit the black colour of the root to both children (bh is preserved) and colour the root red
- Root has changed from black to read, so new violations possible, changes will propagate up



Correction with colour change and rotation

Superior node in red-red violation has red sibling

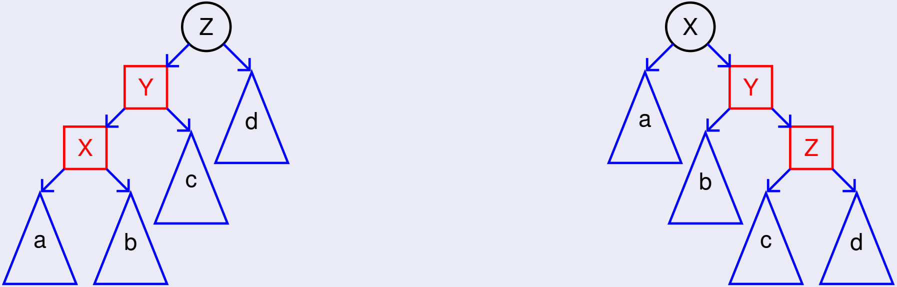
- These siblings must have a black common parent
- Transmit the black colour of the root to both children (bh is preserved) and colour the root red
- Root has changed from black to read, so new violations possible, changes will propagate up



Efficient red-red correction

Superior node in red-red violation has black sibling

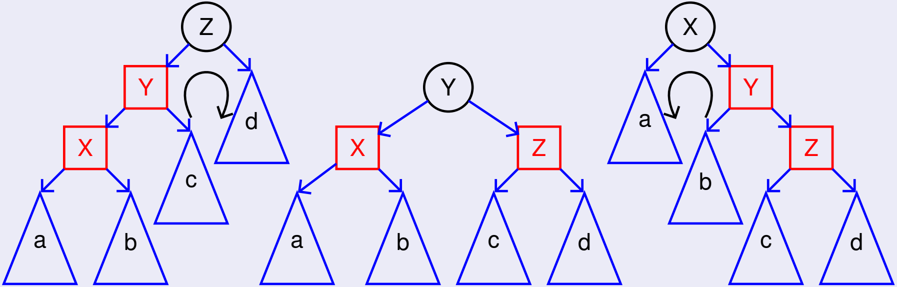
- Now the siblings have different colours, so no black transmission
- Use single or double rotations to push extra red to sibling



Efficient red-red correction

Superior node in red-red violation has black sibling

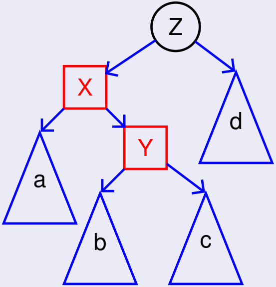
- Now the siblings have different colours, so no black transmission
- Use single or double rotations to push extra red to sibling

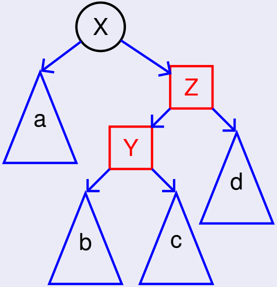


Efficient red-red correction

Superior node in red-red violation has black sibling

- Now the siblings have different colours, so no black transmission
- Use single or double rotations to push extra red to sibling

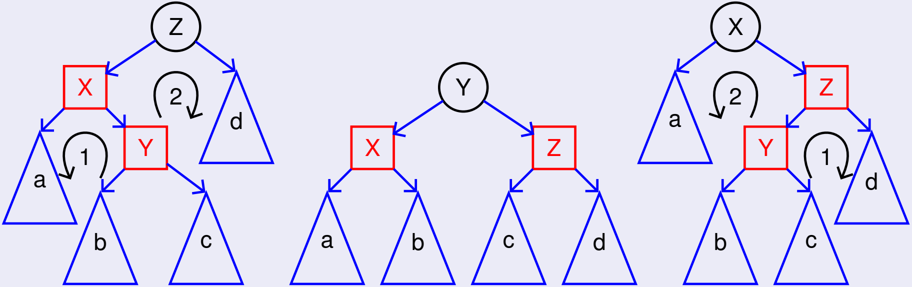




Efficient red-red correction

Superior node in red-red violation has black sibling

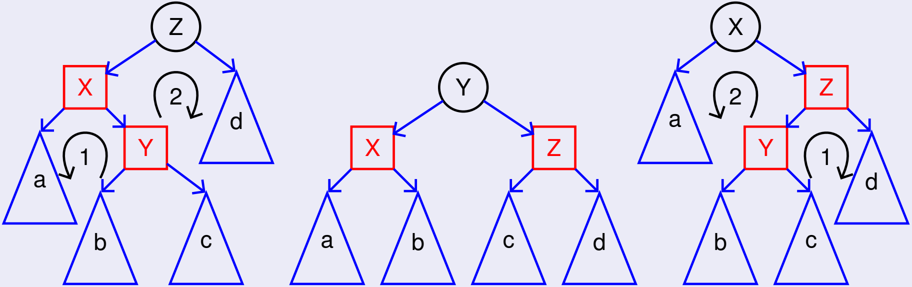
- Now the siblings have different colours, so no black transmission
- Use single or double rotations to push extra red to sibling



Efficient red-red correction

Superior node in red-red violation has black sibling

- Now the siblings have different colours, so no black transmission
- Use single or double rotations to push extra red to sibling
- No upward propagation of changes
- Complexity: $O(\lg n)$



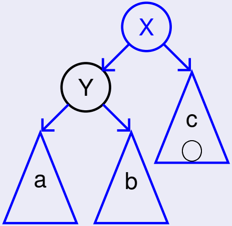
RBT deletion and colour correction

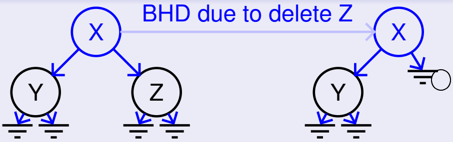
- Basic deletion is as in BST
- If the deleted node is black, *black height deficiency* (BHD) results

RBT deletion and colour correction

- Basic deletion is as in BST
- If the deleted node is black, *black height deficiency* (BHD) results

Case A: Black root of black deficient sub-tree has black sibling and black nephews

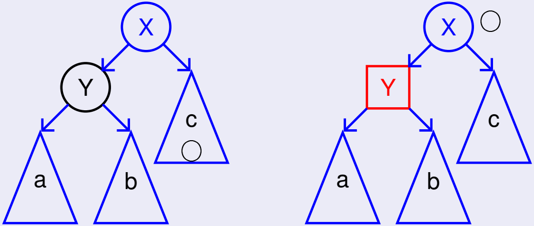




RBT deletion and colour correction

- Basic deletion is as in BST
- If the deleted node is black, *black height deficiency* (BHD) results

Case A: Black root of black deficient sub-tree has black sibling and black nephews

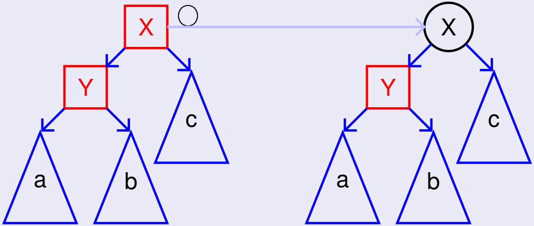


- Black ST root (Y) coloured red, no red-red violation with children
- BHD of both sibling sub-trees moved to common parent

RBT deletion and colour correction

- Basic deletion is as in BST
- If the deleted node is black, *black height deficiency* (BHD) results

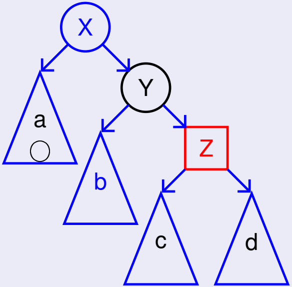
Case A: Black root of black deficient sub-tree has black sibling and black nephews

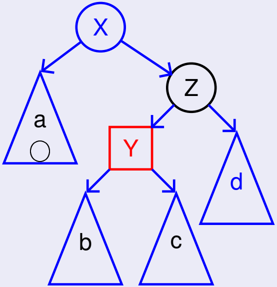


- Black ST root (Y) coloured red, no red-red violation with children
- BHD of both sibling sub-trees moved to common parent
- If root of new sub-tree is red, colour it black to remove BHD and correct red-red violation

Correction with colour change and rotation

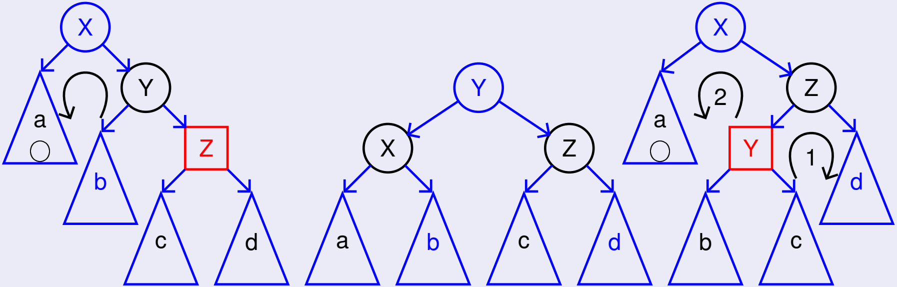
Case B: Black root of black deficient sub-tree has black sibling and at least one red nephew





Correction with colour change and rotation

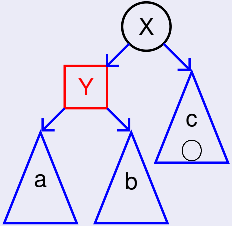
Case B: Black root of black deficient sub-tree has black sibling and at least one red nephew

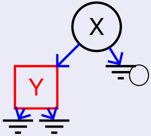


- Transfer a node from the sibling sub-tree to the BHD sub-tree at the expense of the red node
- Colour the transferred node to black to resolve the BHD

Another case of correction

Case C: Black root of black deficient sub-tree has red sibling

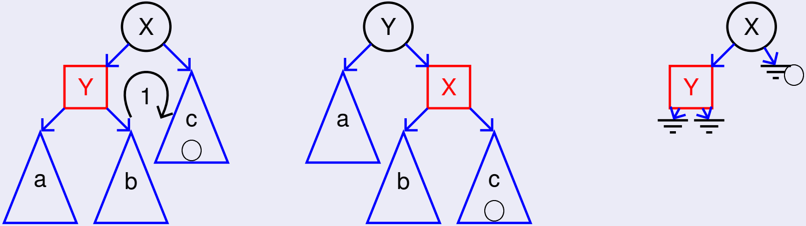




NB Parent must be black; sub-trees of the red sibling must be black

Another case of correction

Case C: Black root of black deficient sub-tree has red sibling

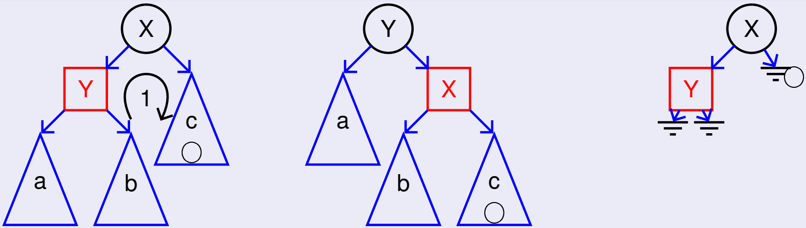


NB Parent must be black; sub-trees of the red sibling must be black

- Apply a single rotation so that the sub-tree with BDH has a black rooted sibling
- Now apply rules of the earlier cases – does not revisit this case

Another case of correction

Case C: Black root of black deficient sub-tree has red sibling



NB Parent must be black; sub-trees of the red sibling must be black

- Apply a single rotation so that the sub-tree with BDH has a black rooted sibling
- Now apply rules of the earlier cases – does not revisit this case

Deletion in RBT can be done in $\Theta(\lg n)$ time

Practice problems

Rank of a node in a RBT

- Store the size of every red-black sub-tree in the local root

```
size[leaf] = 0
size[node] = 1 + size[left[node]] +
  size[right[node]]
```

Practice problems

Rank of a node in a RBT

- Store the size of every red-black sub-tree in the local root

```
size[leaf] = 0
size[node] = 1 + size[left[node]] +
  size[right[node]]
```

- Node of order r can be found in $\Theta(\lg n)$ time
- Rank of a given key can be found in $\Theta(\lg n)$ time
- Number of nodes in tree can be found in $\Theta(1)$ time

Practice problems

Rank of a node in a RBT

- Store the size of every red-black sub-tree in the local root

```
size[leaf] = 0
size[node] = 1 + size[left[node]] +
  size[right[node]]
```

- Node of order r can be found in $\Theta(\lg n)$ time
- Rank of a given key can be found in $\Theta(\lg n)$ time
- Number of nodes in tree can be found in $\Theta(1)$ time
- Size can be updated during insert and delete operations on path back to root from point of insertion/deletion

Practice problems (contd.)

- Insert into an RBT in the given sequence: 2, 1, 4, 5, 9, 3, 6, 7
- Delete from the RBT in the given sequence: 5, 3, 7
- Indicate, with justification, whether the following statements are true or false
  - The subtree of the root of a Red-Black tree is always itself a red-black tree.
  - The sibling of a null child reference in a red-black tree is either another null child reference or a red node.
  - The maximum height of a RBT of n nodes is $2\lg(n+1)$