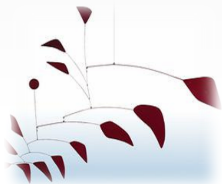


EECS 4101-5101
Advanced Data Structures

Shahin Kamali

Topic 2b Red-Black Trees

York University

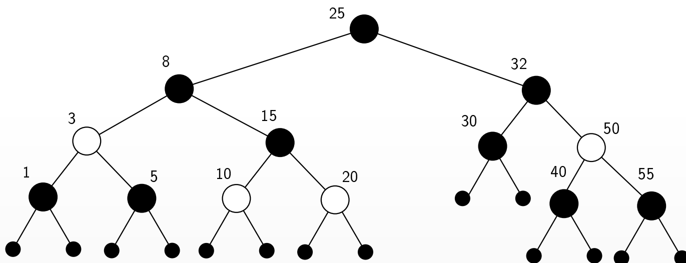
Picture is from the cover of the textbook CLRS.

Red-Black Trees

- Red-black trees were introduced in 1978 by Guibas and Sedgewick.
  - They were evolved from symmetric binary **B-trees** (more on them later).
  - The colours were selected because red and black pens were available to the authors to draw the trees!
  - Red-black trees offer a more relaxed structure than AVL-trees and are often faster!

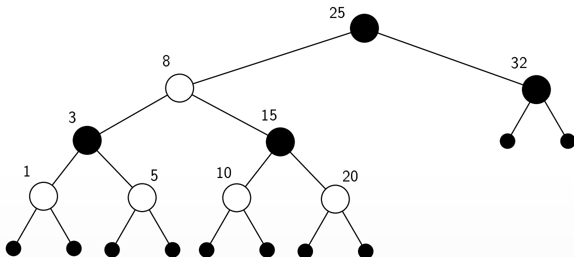
Red-Black Trees

- A **red-black tree** is a binary search tree in which:
  - Every node is colored either Red (pictured white on slides and board) or Black.
  - Each Null pointer is considered to be a Black "node".
  - If a node is Red, then both of its children must be Black.
  - Every path from a node X to a NULL (in the subtree rooted at X) contains the same number of Black nodes.
  - By convention, the root is Black



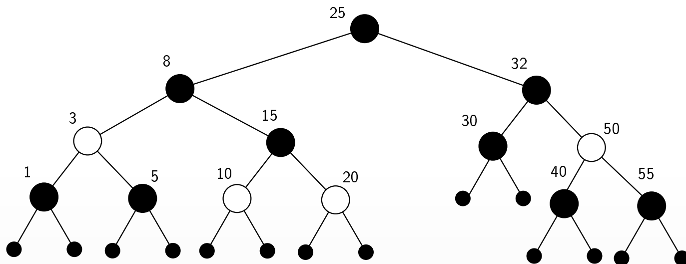
Red-Black Trees

- The left and right subtrees of a node can have heights that differ by a **factor** of 2.
  - Compared to the AVL tree, red-black trees have a much more relaxed structure.



Red-Black Trees

- The **black-height** of a node X in a red-black tree is the number of Black nodes on any path to a NULL, not counting X.
  - Black-Height of the tree (the root) = 3
  - Black-Height of the node with key 8 is 2.

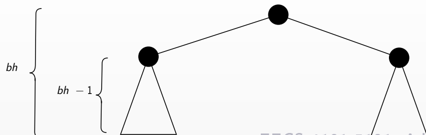


Red-Black Trees

Theorem

The black-height of any red-black tree with n nodes is $O(\log n)$.

- Proof: let $N(h)$ denote the maximum number of nodes in a red-black tree with black-height h.
  - We have $N(1) \geq 1$; For $h > 1$, the value of $N(h)$ is minimized if the tree has a black root and its two children are also black; we can write $N(h) \geq 1 + 2N(h-1)$, which gives $N(h) \geq 2^h$, or $h \in O(\log n)$.

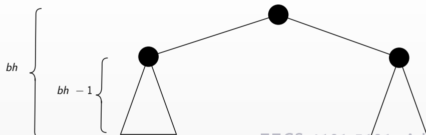


Red-Black Trees

Theorem

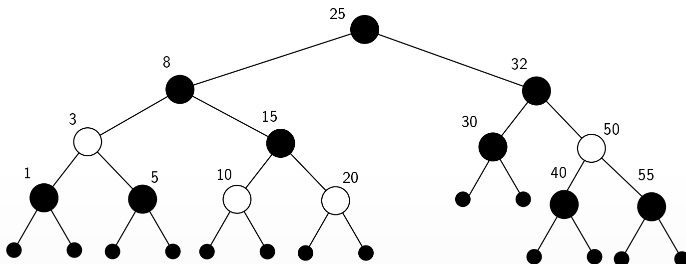
The black-height of any red-black tree with n nodes is $O(\log n)$.

- Proof: let $N(h)$ denote the maximum number of nodes in a red-black tree with black-height h.
  - We have $N(1) \geq 1$; For $h > 1$, the value of $N(h)$ is minimized if the tree has a black root and its two children are also black; we can write $N(h) \geq 1 + 2N(h-1)$, which gives $N(h) \geq 2^h$, or $h \in O(\log n)$.
- The actual height of a red-black tree is at most twice the black-height (why?) $\rightarrow$ the height of a red-black tree is $O(\log n)$.



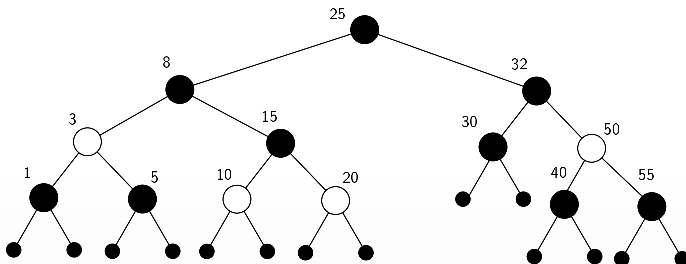
Red-Black Tree Insertion

- Insert node; Color it Red; X is pointer to it.
  1. X is the root – color it Black.



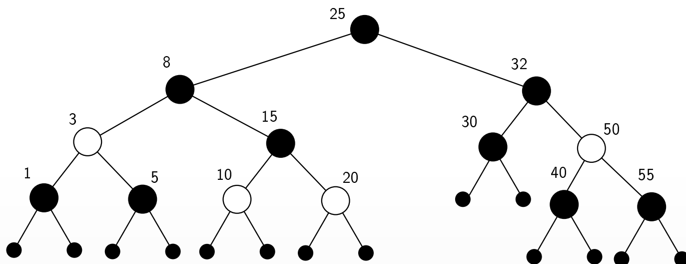
Red-Black Tree Insertion

- Insert node; Color it Red; X is pointer to it.
  1. X is the root – color it Black.
  2. Parent is black; nothing to do.



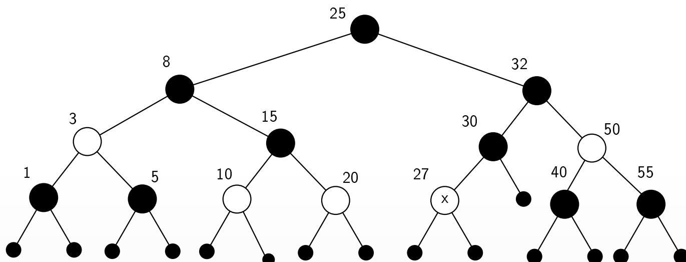
Red-Black Tree Insertion

- Insert node; Color it Red; X is pointer to it.
  1. X is the root – color it Black.
  2. Parent is black; nothing to do.
  3. E.g., insert(27)



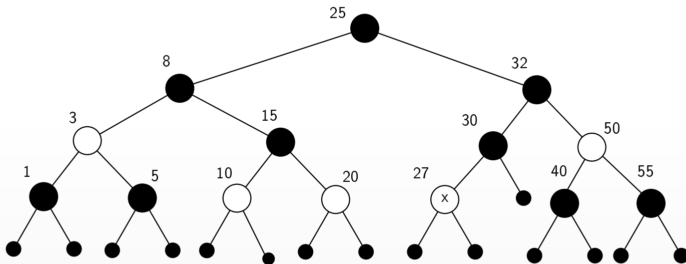
Red-Black Tree Insertion

- Insert node; Color it Red; X is pointer to it.
  1. X is the root – color it Black.
  2. Parent is black; nothing to do.
  3. E.g., insert(27)



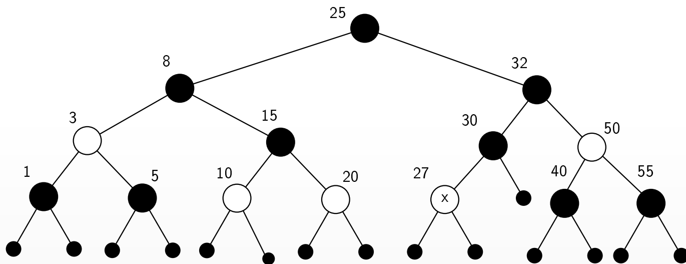
Red-Black Tree Insertion

- Insert node; Color it Red; X is pointer to it. Cases to consider:
  1. X is the root – color it Black.
  2. Parent is black; nothing to do.
  3. Both parent and uncle are Red – color parent and uncle Black, color grandparent Red. Point X to the grandparent and check the new situation.



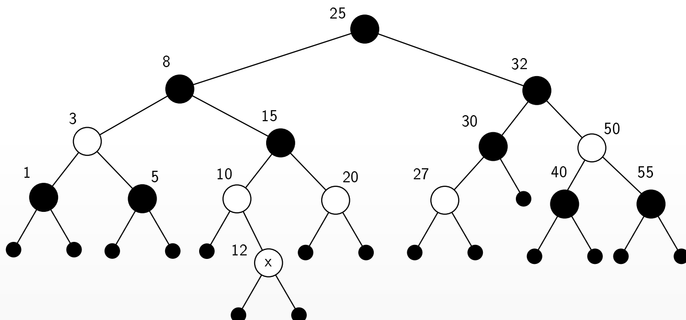
Red-Black Tree Insertion

- Insert node; Color it Red; X is pointer to it. Cases to consider:
  1. X is the root – color it Black.
  2. Parent is black; nothing to do.
  3. Both parent and uncle are Red – color parent and uncle Black, color grandparent Red. Point X to the grandparent and check the new situation.
  4. E.g., insert(12)



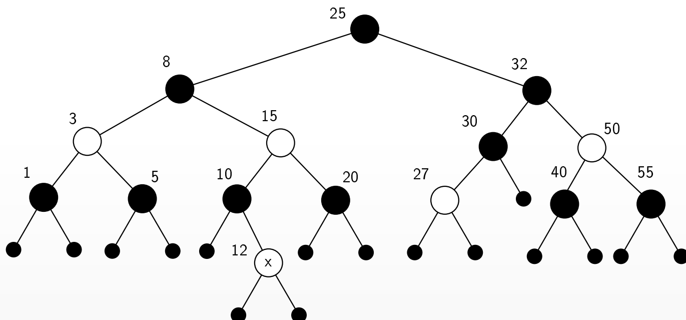
Red-Black Tree Insertion

- Insert node; Color it Red; X is pointer to it. Cases to consider:
  1. X is the root – color it Black.
  2. Parent is black; nothing to do.
  3. Both parent and uncle are Red – color parent and uncle Black, color grandparent Red. Point X to the grandparent and check the new situation.
  4. E.g., insert(12)



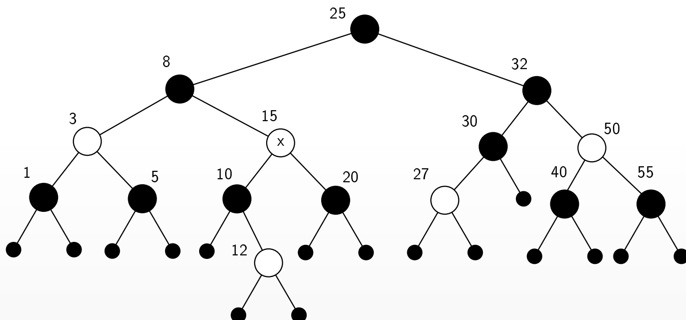
Red-Black Tree Insertion

- Insert node; Color it Red; X is pointer to it. Cases to consider:
  1. X is the root – color it Black.
  2. Parent is black; nothing to do.
  3. Both parent and uncle are Red – color parent and uncle Black, color grandparent Red. Point X to the grandparent and check the new situation.
  4. E.g., insert(12)



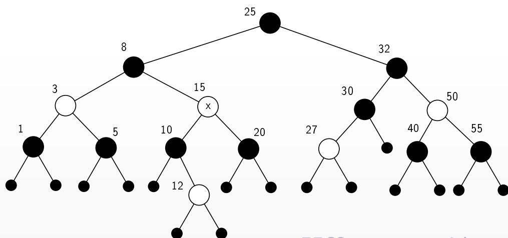
Red-Black Tree Insertion

- Insert node; Color it Red; X is pointer to it. Cases to consider:
  1. X is the root – color it Black.
  2. Parent is black; nothing to do.
  3. Both parent and uncle are Red – color parent and uncle Black, color grandparent Red. Point X to the grandparent and check the new situation.
  4. E.g., insert(12)



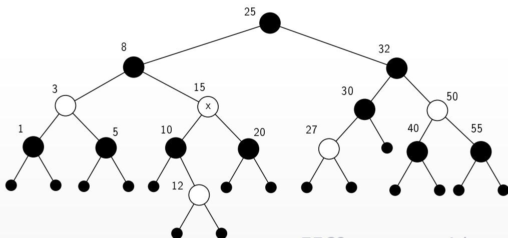
Red-Black Tree Insertion

- Insert node; Color it Red; X is pointer to it. Cases to consider:
  1. X is the root – color it Black.
  2. Parent is black; nothing to do.
  3. Both parent and uncle are Red – color parent and uncle Black, color grandparent Red. Point X to the grandparent and check the new situation.
  4. Parent is Red and uncle is Black: rotate on the three nodes x, parent, and its grandparent (s.t. the middle key becomes the parent of the other two). Children of these three nodes are all black (why?) $\rightarrow$ recolor to fix the structure. E.g., insert (28):



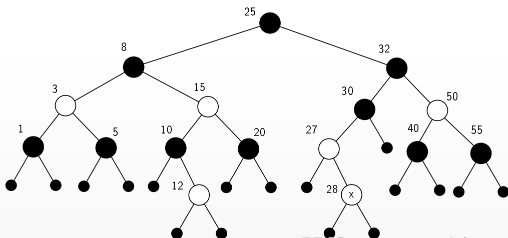
Red-Black Tree Insertion

- Insert node; Color it Red; X is pointer to it. Cases to consider:
  1. X is the root – color it Black.
  2. Parent is black; nothing to do.
  3. Both parent and uncle are Red – color parent and uncle Black, color grandparent Red. Point X to the grandparent and check the new situation.
  4. Parent is Red and uncle is Black: rotate on the three nodes x, parent, and its grandparent (s.t. the middle key becomes the parent of the other two). Children of these three nodes are all black (why?) $\rightarrow$ recolor to fix the structure. E.g., insert (28):



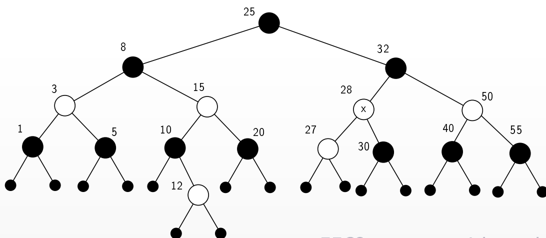
Red-Black Tree Insertion

- Insert node; Color it Red; X is pointer to it. Cases to consider:
  1. X is the root – color it Black.
  2. Parent is black; nothing to do.
  3. Both parent and uncle are Red – color parent and uncle Black, color grandparent Red. Point X to the grandparent and check the new situation.
  4. Parent is Red and uncle is Black: rotate on the three nodes x, parent, and its grandparent (s.t. the middle key becomes the parent of the other two). Children of these three nodes are all black (why?) $\rightarrow$ recolor to fix the structure. E.g., insert (28):



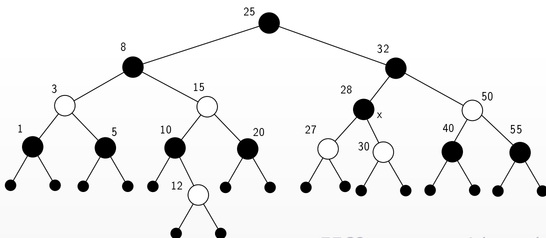
Red-Black Tree Insertion

- Insert node; Color it Red; X is pointer to it. Cases to consider:
  1. X is the root – color it Black.
  2. Parent is black; nothing to do.
  3. Both parent and uncle are Red – color parent and uncle Black, color grandparent Red. Point X to the grandparent and check the new situation.
  4. Parent is Red and uncle is Black: rotate on the three nodes x, parent, and its grandparent (s.t. the middle key becomes the parent of the other two). Children of these three nodes are all black (why?) $\rightarrow$ recolor to fix the structure. E.g., insert (28):



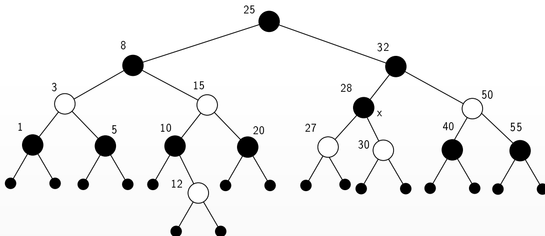
Red-Black Tree Insertion

- Insert node; Color it Red; X is pointer to it. Cases to consider:
  1. X is the root – color it Black.
  2. Parent is black; nothing to do.
  3. Both parent and uncle are Red – color parent and uncle Black, color grandparent Red. Point X to the grandparent and check the new situation.
  4. Parent is Red and uncle is Black: rotate on the three nodes x, parent, and its grandparent (s.t. the middle key becomes the parent of the other two). Children of these three nodes are all black (why?) $\rightarrow$ recolor to fix the structure. E.g., insert (28):



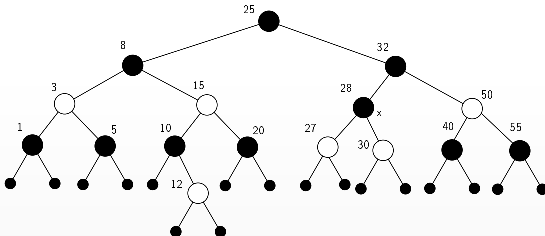
Red-Black Tree Insertion

- How long does insertion take?
  1. Step 3 (when both parents and uncle are Red) only recolors and repeats at a higher level.



Red-Black Tree Insertion

- How long does insertion take?
  1. Step 3 (when both parents and uncle are Red) only recolors and repeats at a higher level.
  2. Rotations occur at least once at Step 4 (after rotation, we do not need to re-structure the tree).



Red-Black Tree Insertion

- How long does insertion take?
  1. Step 3 (when both parents and uncle are Red) only recolors and repeats at a higher level.
  2. Rotations occur at least once at Step 4 (after rotation, we do not need to re-structure the tree).
  3. Insertion in a Red-Black Tree takes at most **1 rotation and $\Theta(\log n)$ time**.

Red-Black Tree Insertion

- How long does insertion take?
  1. Step 3 (when both parents and uncle are Red) only recolors and repeats at a higher level.
  2. Rotations occur at least once at Step 4 (after rotation, we do not need to re-structure the tree).
  3. Insertion in a Red-Black Tree takes at most **1 rotation and** $\Theta(\log n)$ **time**.
  4. Deletion has the same spirit; you will practice them in your next assignment.

Red-Black Tree Summary

- Red-Black trees support insertion, deletion, and search in $O(\log n)$ time.
- Although the time-complexities of operations are the same as AVL trees, in practice, Red-Black trees are faster and require fewer rotations.
- Red-Black trees can be augmented in a similar way that AVL trees can!