CSE 30331
Red-Black Trees
B-Trees: Review

Good

- **Efficient** insert, erase, and search operations
- Good **data locality**

Bad

- **Complex** balancing mechanisms
- Potentially wasteful memory usage
Red-Black Tree
Red-Black Tree: Overview

A Red-Black Tree is a B-Tree overlayed on top of a BST

# Node where # is the number of possible children of B-Tree Node
Red-Black Tree: Invariants

1. Every **B-node** is a 2-node, a 3-node, or a balanced 4-node

2. Every path has the same number of **B-nodes**

3. The root is **black**

4. No path has two **red** nodes in a row

5. Every path has the same number of **black** nodes
Red-Black Tree: Visualization
Red-Black Tree: Search

Same as BST!
Red-Black Tree: Rotation

Rotation inverts the relationship of parent and child

Swap position and color

Must Also Move Middle Child (C)
Red-Black Tree: Insert

1. Beginning is identical to insertion in BST
   a. If tree is empty then new node is black
   b. Otherwise, new node is red (otherwise violate path length invariant)

2. Mark the new node with a red star
   a. If B-node is a 3-node or a balanced 4-node (ie. parent is black), remove red star and stop
   b. If B-node is an unbalanced 4-node, we need to balance by using one or two rotations
   c. If B-node is a 5-node, then it is overfull, so flip colors of top three nodes
      i. Move red star to new node in B-parent
      ii. If root, then paint it black
      iii. Otherwise, repeat process at (2)
Red-Black Tree: Insert(A, B, C, D)

Insert: A, B, C, D, E, F, G, H

A

B

A

B

3 Node

C

A

B

C

Unbalanced
4 Node

Rotate Left

D

B

A

C

D

Overfull
5 Node

Flip Colors of Top Three

Mark Root Black
Red-Black Tree: Insert(E, F)

Insert: A, B, C, D, E, F, G, H

Unbalanced 4 Node

Rotate Left

Overfull 5 Node

Flip Colors of Top Three
Red-Black Tree: \textbf{Insert}(G)

Insert: A, B, C, D, E, F, G, H

Unbalanced
4 Node

Rotate Left
Red-Black Tree: **Insert(H)**

Insert: A, B, C, D, E, F, G, H

- **Overfull 5 Node**
- **Unbalanced 4 Node**
- **Flip Colors of Top Three**
- **Rotate Left**
Red-Black Tree: Erase

1. Find the node to be deleted as in BST

2. If it’s in leaf B-node, then delete the element in it

3. If it’s not in a leaf B-node, swap it with the next biggest or next smallest element (BST)

4. Find the node within the B-node
   a. If it is not a leaf or unary node, swap it with the next biggest or next smallest element and delete it (B-Tree)
   b. If node to be deleted roots a 2-node, then we will be underfull
      i. Merge underfull B-node with its B-sister by flipping colors
      ii. Rebalance merged B-node if necessary
      iii. Split B-node again if possible by using color flip
Red-Black Tree: Properties

- **Best case**: $O(\log(n))$
- **Average case**: $O(\log(n))$
- **Worst case**: $O(\log(n))$

*Red-Black Trees* combine the guarantees of a *B-Tree* with the compact structure of a *BST*