Red-Black Trees

Letícia Rodrigues Bueno

Federal University of ABC (UFABC)

Red-Black Trees: Introduction

- **Purpose:** providing basic operations in time $O(\lg n)$ in the worst case;

Red-Black Trees: Introduction

- **Purpose:** providing basic operations in time $O(\lg n)$ in the worst case;
- **Red-Black Trees:** binary search trees with an extra bit in each node to the color (red or black);

Red-Black Trees: Introduction

- **Purpose:** providing basic operations in time $O(\lg n)$ in the worst case;
- **Red-Black Trees:** binary search trees with an extra bit in each node to the color (red or black);
- **Height:** at most $2\lg(n+1)$ where n is the number of nodes;

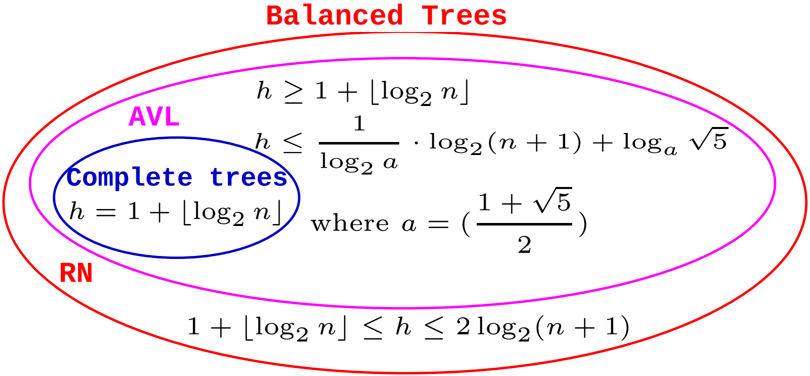
Red-Black Trees: Introduction

- **Purpose:** providing basic operations in time $O(\lg n)$ in the worst case;
- **Red-Black Trees:** binary search trees with an extra bit in each node to the color (red or black);
- **Height:** at most $2\lg(n+1)$ where n is the number of nodes;
- Insertion and deletion have time $O(\lg n)$;

Red-Black Trees: Introduction

- **Purpose:** providing basic operations in time $O(\lg n)$ in the worst case;
- **Red-Black Trees:** binary search trees with an extra bit in each node to the color (red or black);
- **Height:** at most $2\lg(n+1)$ where n is the number of nodes;
- Insertion and deletion have time $O(\lg n)$;
- The path from the root to the farthest leaf is no more than twice as long as the path from the root to the nearest leaf.

Comparison of Balanced Trees



Red-Black Trees: Definition

- A Red-Black tree is a binary search tree (BST) satisfying:

Red-Black Trees: Definition

- A Red-Black tree is a binary search tree (BST) satisfying:
  1. Every external node is black;

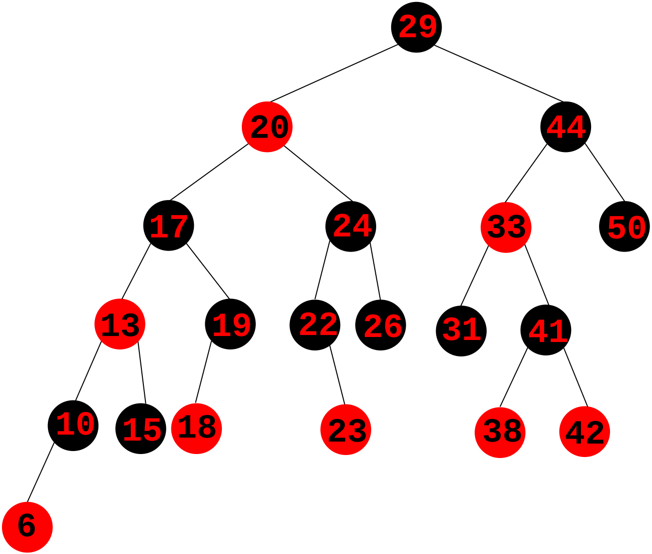
Red-Black Trees: Definition

- A Red-Black tree is a binary search tree (BST) satisfying:
  1. Every external node is black;
  2. For each node, all paths from a node to the leaves have the same number of black nodes;

Red-Black Trees: Definition

- A Red-Black tree is a binary search tree (BST) satisfying:
  1. Every external node is black;
  2. For each node, all paths from a node to the leaves have the same number of black nodes;
  3. If a node is red, then both child nodes are black.

Example of Red-Black Tree

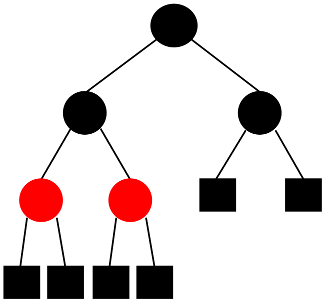


Red-Black Trees: Insertion

Node inserted q is red. Possibilities:

Case 1: v is black

Example:

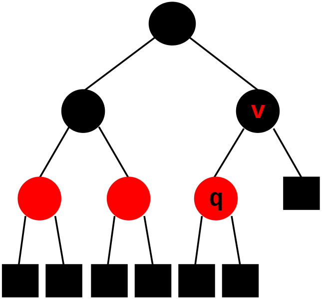


Red-Black Trees: Insertion

Node inserted *q* is red. Possibilities:

Case 1: *v* is black

Example:



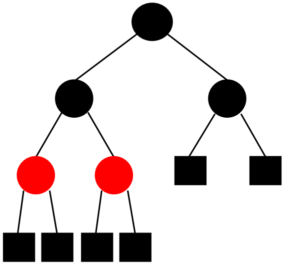
Red-Black Trees: Insertion

Node inserted q is red. Possibilities:

Case 2: v is red. Therefore, w (parent of v) is black.

Case 2.1: t is red;
We modify the color of v, t, w.

Example:



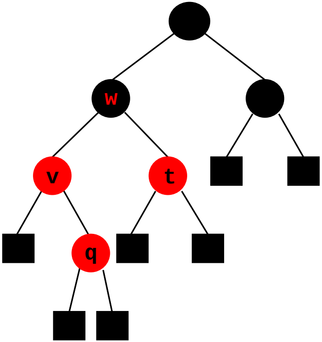
Red-Black Trees: Insertion

Node inserted q is red. Possibilities:

Case 2: v is red. Therefore, w (parent of v) is black.

Case 2.1: t is red;
We modify the color of v, t, w.

Example:



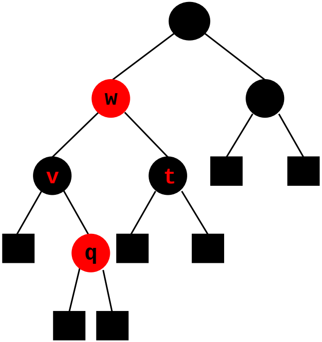
Red-Black Trees: Insertion

Node inserted q is red. Possibilities:

Case 2: v is red. Therefore, w (parent of v) is black.

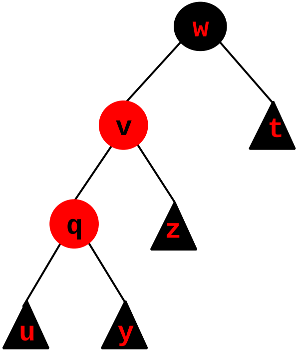
Case 2.1: t is red;
We modify the color of v, t, w.

Example:



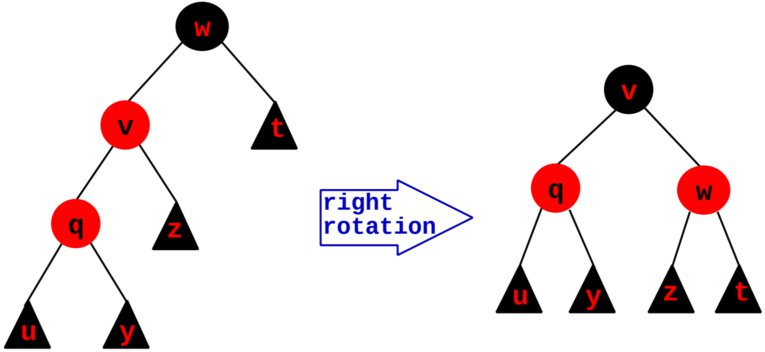
Rotations – Case 2.2: t is black

Case 2.2.1: q is the left child of v, and v is the left child of w.
Change the color of v and w.



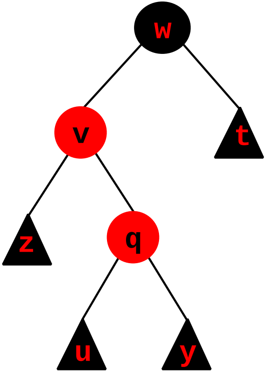
Rotations – Case 2.2: t is black

Case 2.2.1: q is the left child of v, and v is the left child of w. Change the color of v and w.



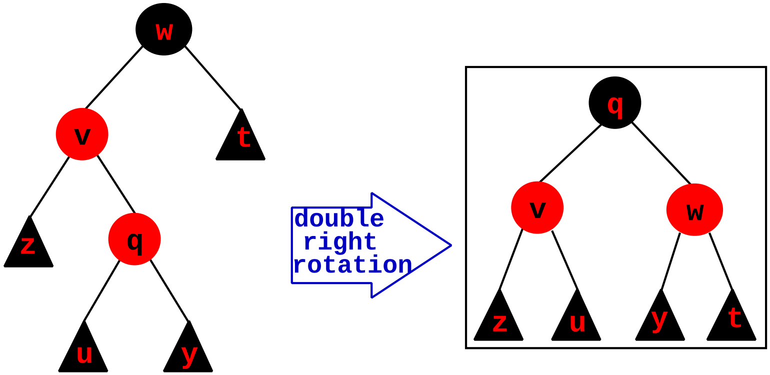
Rotations – Case 2.2: t is black

Case 2.2.2: q is the right child of v, and v is the left child of w. Change color of q and w.



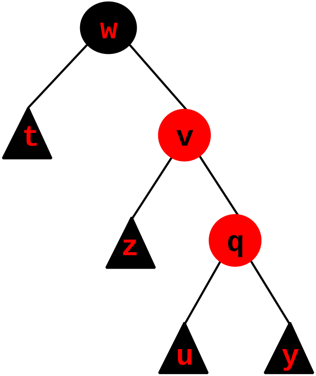
Rotations – Case 2.2: *t* is black

Case 2.2.2: *q* is the right child of *v*, and *v* is the left child of *w*.
Change color of *q* and *w*.



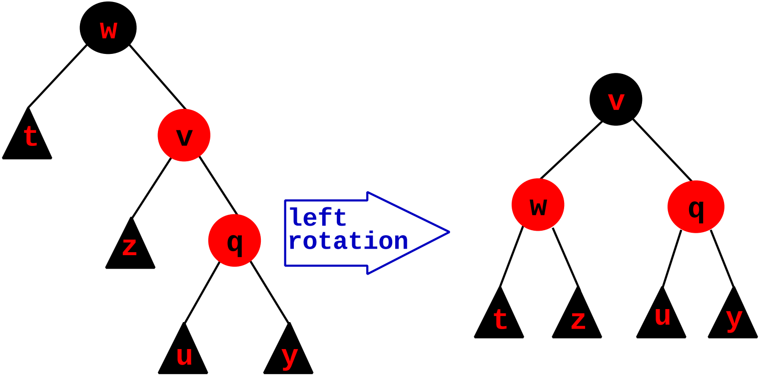
Rotations – Case 2.2: *t* is black

Case 2.2.3: *q* is the right child of *v*, and *v* is the right child of *w*. Change color of *v* and *w*.



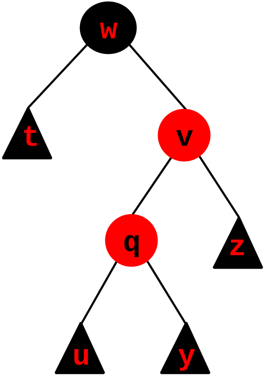
Rotations – Case 2.2: t is black

Case 2.2.3: q is the right child of v, and v is the right child of w. Change color of v and w.



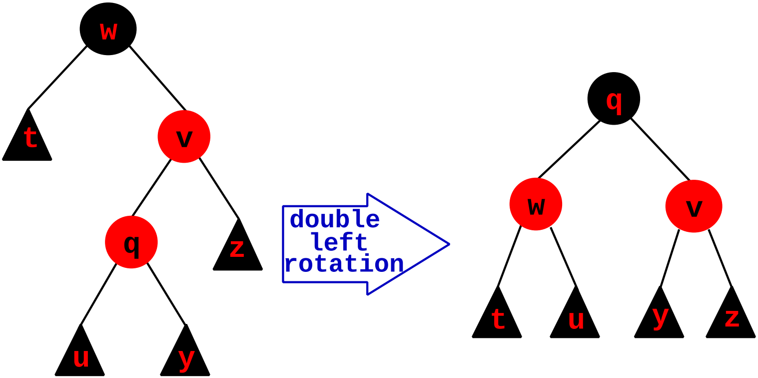
Rotations – Case 2.2: t is black

Case 2.2.4: q is the left child of v, and v is the right child of w. Change color of q and w.



Rotations – Case 2.2: t is black

Case 2.2.4: q is the left child of v, and v is the right child of w. Change color of q and w.



Red-Black Trees: insertion algorithm

```
InsertionRB(x, ptv, ptw, ptr, a):
    if ptv = external then
        new(ptv)
        ptv ↑ .left ← ptv ↑ .right ← external
        ptv ↑ .key ← x; ptv ↑ .color ← R
        if ptroot = external then
            ptv ↑ .color ← B; ptroot ← ptv
        else if x < ptw ↑ .key then
            ptw ↑ .left ← ptv
        else ptw ↑ .right ← ptv
    else if x ≠ ptv ↑ .key then
        if x < ptv ↑ .key then ptq ← ptv ↑ .left
        else ptq ← ptv ↑ .right
            InsertionRB(x, ptq, ptv, ptw, a)
            if a = 1 then route(ptq, ptv, ptw, ptr, a)
            else if a = 0 then a = 1
            else "Invalid Insertion"
```

Comparison: AVL and Red-Black Trees

Comparison: AVL and Red-Black Trees

1. **AVL Trees**:

Comparison: AVL and Red-Black Trees

1. **AVL Trees**:
   1.1 first self-balancing binary search tree, proposed by Adel'son-Vel'skii and Landis in 1962;

Comparison: AVL and Red-Black Trees

1. **AVL Trees**:
   - 1.1 first self-balancing binary search tree, proposed by Adel'son-Vel'skii and Landis in 1962;
   - 1.2 **height:** between $\log_2(n+1)$ and $1.4404\log_2(n+2) - 0.328$, therefore, $O(\log n)$;

Comparison: AVL and Red-Black Trees

1. **AVL Trees**:
   1.1 first self-balancing binary search tree, proposed by Adel'son-Vel'skii and Landis in 1962;
   1.2 **height:** between $\log_2(n+1)$ and $1.4404 \log_2(n+2) - 0.328$, therefore, $O(\log n)$;
2. **Red-Black Trees**:

Comparison: AVL and Red-Black Trees

1. **AVL Trees**:
   1.1 first self-balancing binary search tree, proposed by Adel'son-Vel'skii and Landis in 1962;
   1.2 **height:** between $\log_2(n+1)$ and $1.4404\log_2(n+2) - 0.328$, therefore, $O(\log n)$;
2. **Red-Black Trees**:
   2.1 proposed by Guibas and Sedgewick in 1978;

Comparison: AVL and Red-Black Trees

1. **AVL Trees**:
   1.1 first self-balancing binary search tree, proposed by Adel'son-Vel'skii and Landis in 1962;
   1.2 **height:** between $\log_2(n+1)$ and $1.4404\log_2(n+2) - 0.328$, therefore, $O(\log n)$;
2. **Red-Black Trees**:
   2.1 proposed by Guibas and Sedgewick in 1978;
   2.2 **height:** $2\log_2(n+1)$, therefore, $O(\log n)$;

Comparison: AVL and Red-Black Trees

1. **AVL Trees**:
   1.1 first self-balancing binary search tree, proposed by Adel'son-Vel'skii and Landis in 1962;
   1.2 **height:** between $\log_2(n+1)$ and $1.4404\log_2(n+2) - 0.328$, therefore, $O(\log n)$;
2. **Red-Black Trees**:
   2.1 proposed by Guibas and Sedgewick in 1978;
   2.2 **height:** $2\log_2(n+1)$, therefore, $O(\log n)$;

Comparison: AVL trees are more strictly balanced than Red-Black trees, making insertion and deletion slower but retrieval (search) faster;

Exercises

1. Prove or give a counterexample:
    1.1 Every complete tree is AVL.
    1.2 Every AVL tree is complete.
    1.3 Every AVL tree is red-black.
    1.4 Every red-black tree is AVL.
    1.5 Every complete tree is red-black.
    1.6 Every red-black tree is complete.

Exercises

2. Show that, in a Red-Black tree, the longest path from a node x to a leaf has length at most twice the length of the shortest path from x to a leaf.

Exercises

2. Show that, in a Red-Black tree, the longest path from a node x to a leaf has length at most twice the length of the shortest path from x to a leaf.
3. Give an example of insertion in a Red-Black tree whose recoloring of nodes propagates up to the root.

Exercises

2. Show that, in a Red-Black tree, the longest path from a node x to a leaf has length at most twice the length of the shortest path from x to a leaf.
3. Give an example of insertion in a Red-Black tree whose recoloring of nodes propagates up to the root.
4. Write the procedure of deletion for Red-Black trees.

Exercises

2. Show that, in a Red-Black tree, the longest path from a node x to a leaf has length at most twice the length of the shortest path from x to a leaf.
3. Give an example of insertion in a Red-Black tree whose recoloring of nodes propagates up to the root.
4. Write the procedure of deletion for Red-Black trees.
5. Prove or give a counterexample: given a Red-Black tree with a red root, if we change its color for black, the resulting tree is still a Red-Black tree.

Bibliography

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L. and STEIN, C. *Introduction to Algorithms*, 3rd edition, MIT Press, 2009.

SZWARCFITER, J. L. and MARKENZON, L. Estruturas de Dados e seus Algoritmos, LTC, 1994. (in Portuguese)