Red-Black Trees

Prem Nair

Red-Black Tree

Wholeness of the Lesson

- Red-black trees provide a solution to the problem of unacceptably slow worst case performance of binary search trees. This is accomplished by introducing a new element: nodes of the tree are colored red or black, adhering to the balance condition for red-black trees. The balance condition is maintained

Red-Black Tree

during insertions and deletions and doing so introduces only slight overhead.

- **Science of Consciousness: Red-black trees, as an example of BSTs** with a balance condition, exhibit the Principle of the Second Element for solving the problem of skewed BSTs.

Binary Search Trees

- *Binary Search Trees* (BSTs) are an important data structure for dynamic sets

- In addition to satellite data, elements have:
    - *key*: an identifying field inducing a total ordering
    - *left*: pointer to a left child (may be NULL)
    - *right*: pointer to a right child (may be NULL)
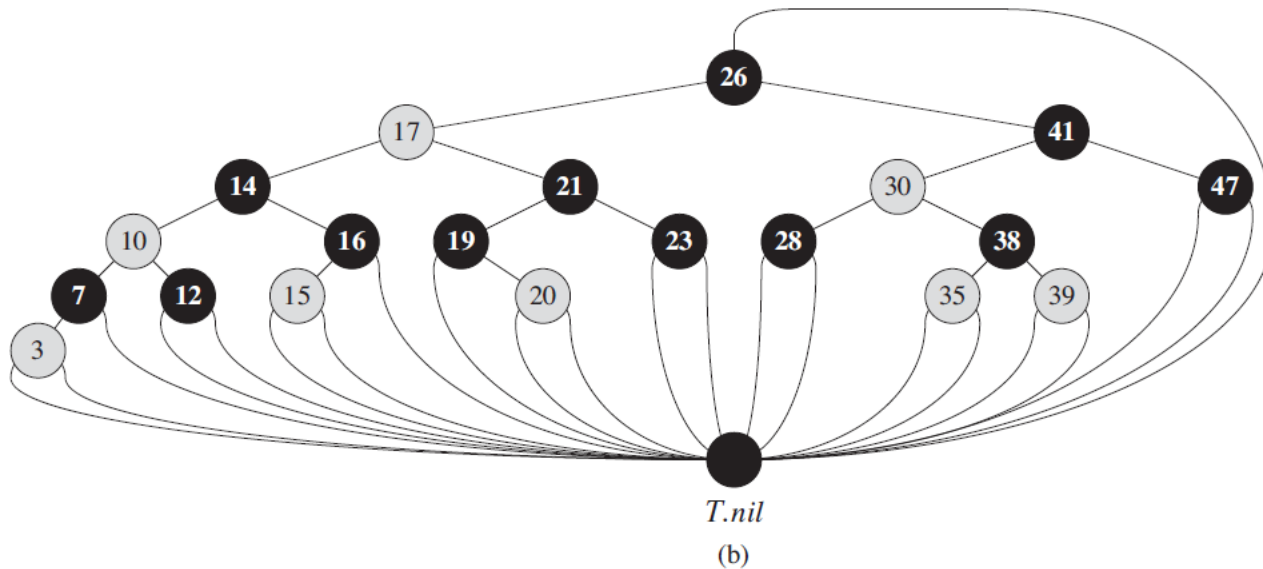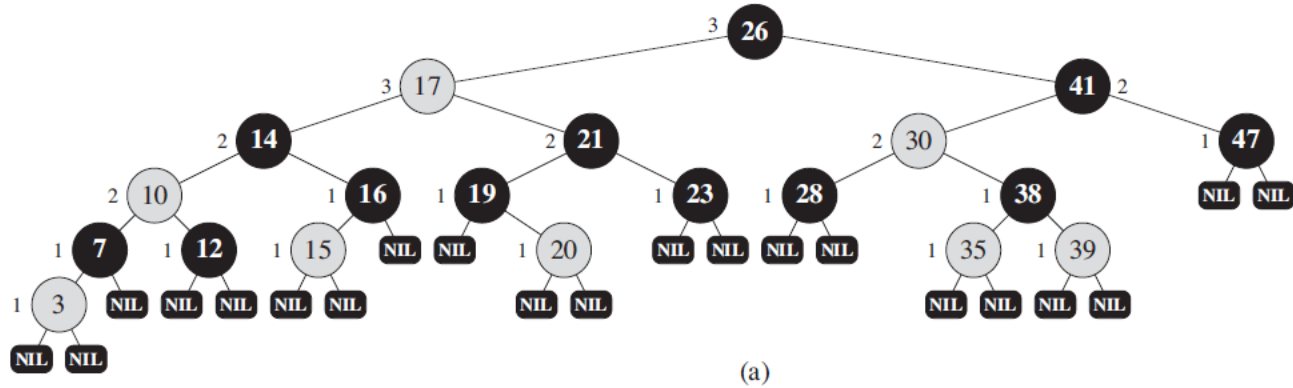   - *p*: pointer to a parent node (NULL for root)

Red-Black Trees

- *Red-black trees*:
  - Binary search trees augmented with node color
  - Operations designed to guarantee that the height $h = \mathrm{O}(\log n)$
- First: describe the properties of red-black trees
- Then: prove that these guarantee $h = \mathrm{O}(\log n)$
- Finally: describe operations on red-black trees

5 Properties of a Red-Black Tree

1. Every node is either red or black.
2. Every **leaf** (NULL pointer) is a black node.
    - Note: this means every **interior node** has 2 children.
3. If a node is red, both children are black.
    - Note: can't have 2 consecutive red nodes on a path.
4. Every path from a node to descendent leaf contains the same number of black nodes.
    - Note: The number of black nodes on the path **excluding the node** but **including the leaf** is the **black-height**. **Black-height of leaf is 0.**
5. The root is always black.

Red-Black Tree Example
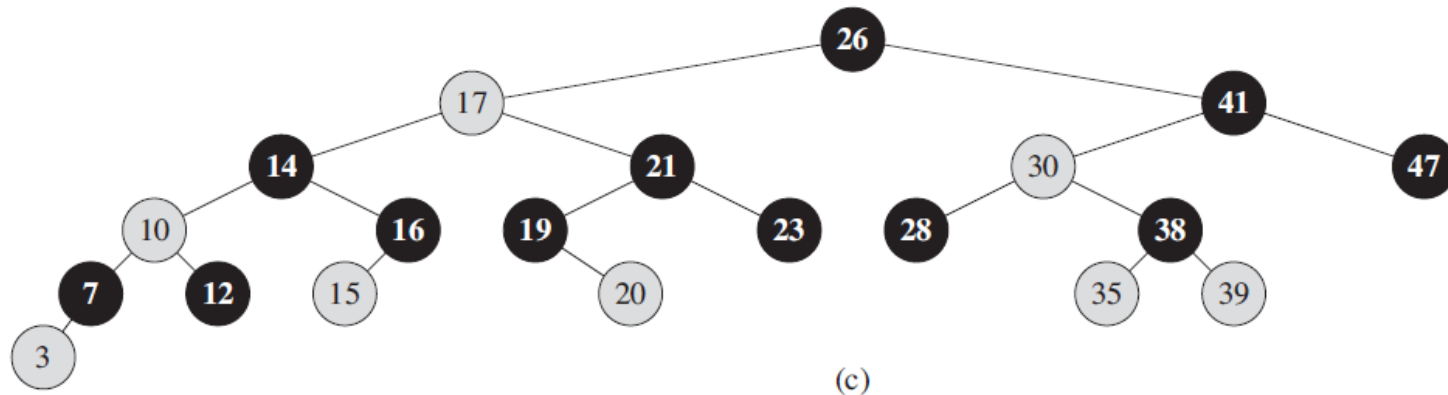

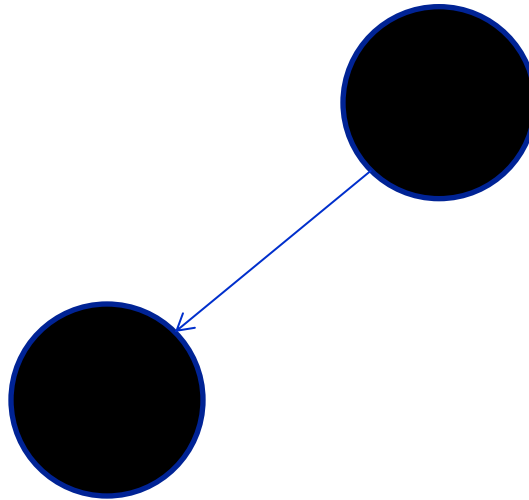(a)

(b)

Red-Black Tree Example

Figure 13.1 A red-black tree with black nodes darkened and red nodes shaded. Every node in a red-black tree is either red or black, the children of a red node are both black, and every simple path from a node to a descendant leaf contains the same number of black nodes. **(a)** Every leaf, shown as a NIL, is black. Each non-NIL node is marked with its black-height; NILs have black-height 0. **(b)** The same red-black tree but with each NIL replaced by the single sentinel $T.nil$, which is always black, and with black-heights omitted. The root's parent is also the sentinel. **(c)** The same red-black tree but with leaves and the root's parent omitted entirely. We shall use this drawing style in the remainder of this chapter.

Group Activity

Can we have the following Red-Black tree?
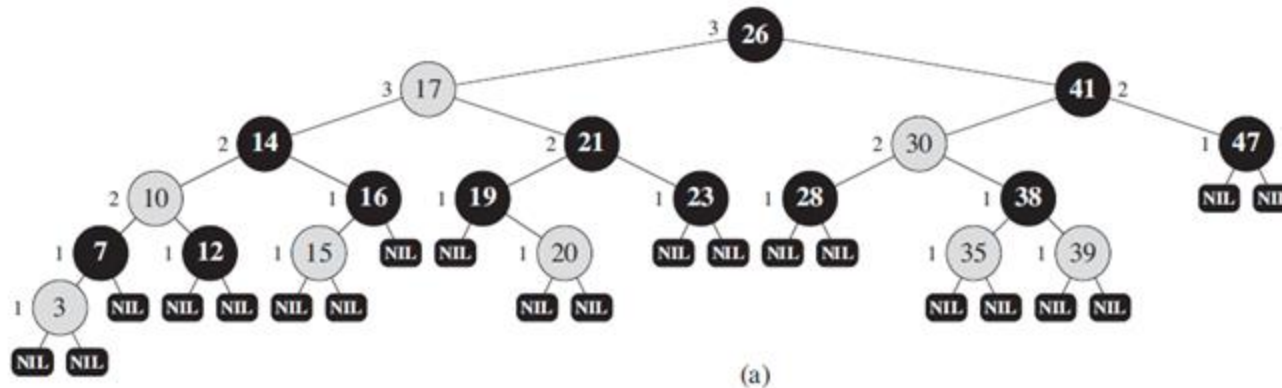
If not, which property is violated?

Leaf, interior node and height

Note: The definition of the leaf has changed.

Both interior node and height of a tree is defined using leaf.

(1) The node 3 is an interior node (for a Red-Black tree).

(2) The height of the tree is 6 (not 5 for this Red-Black tree).

(3) The longest path from root to a leaf (the height) is 6.

(4) The shortest path from root to leaf (black height) is 3.

(a)

h is O(log n)

Result 1. If x is a node with height h, bh(x) $\geq$ h/2.

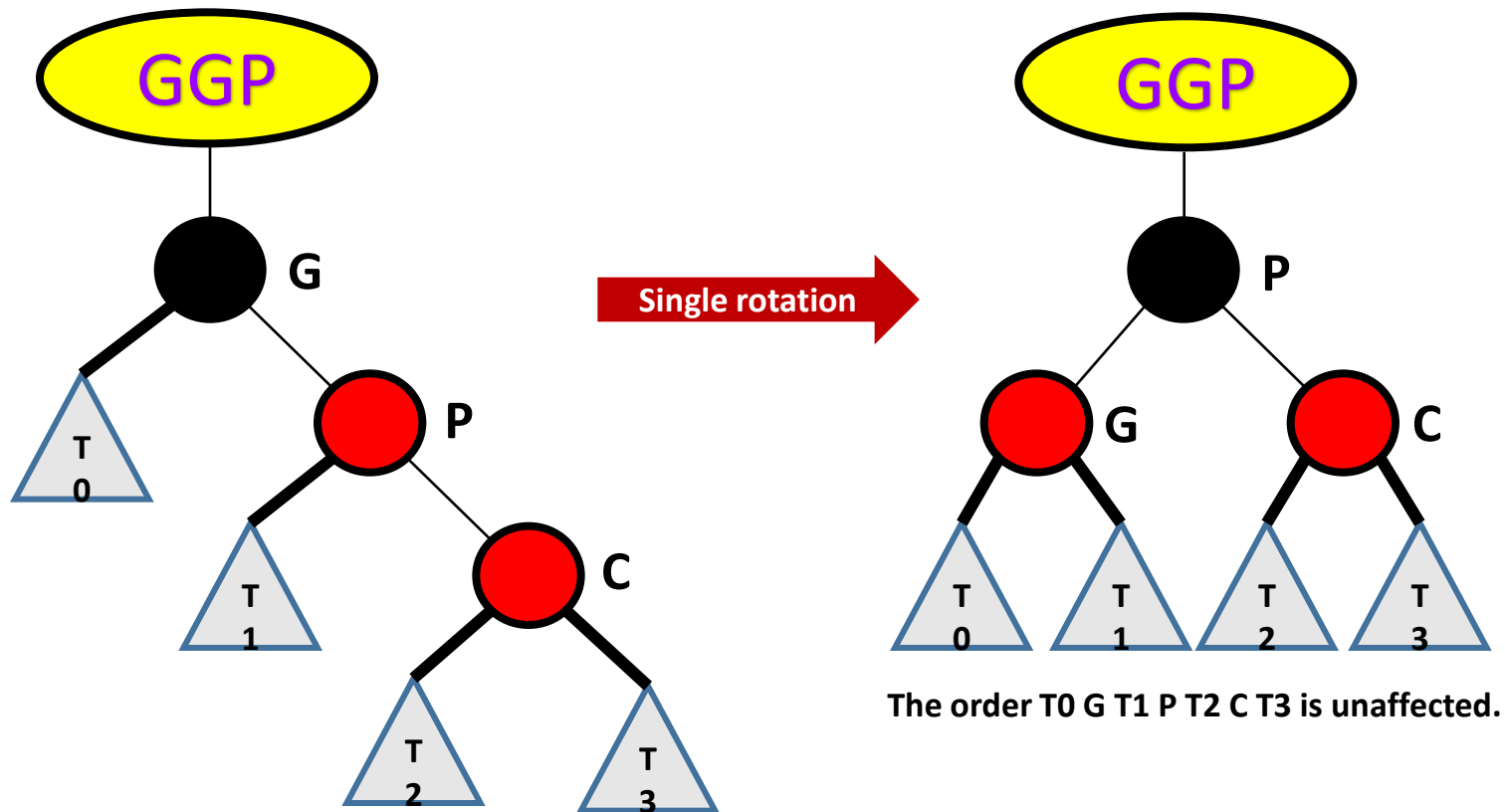*Result 2: A red-black tree with n internal nodes has height h $\leq$ 2 log(n + 1).*

h is O(log n)

The time complexity of search/insert/delete is O(log n)

Left Rotation   Left_RotateAt(P)

Use it when G has P as Right child, Parent has C as right child
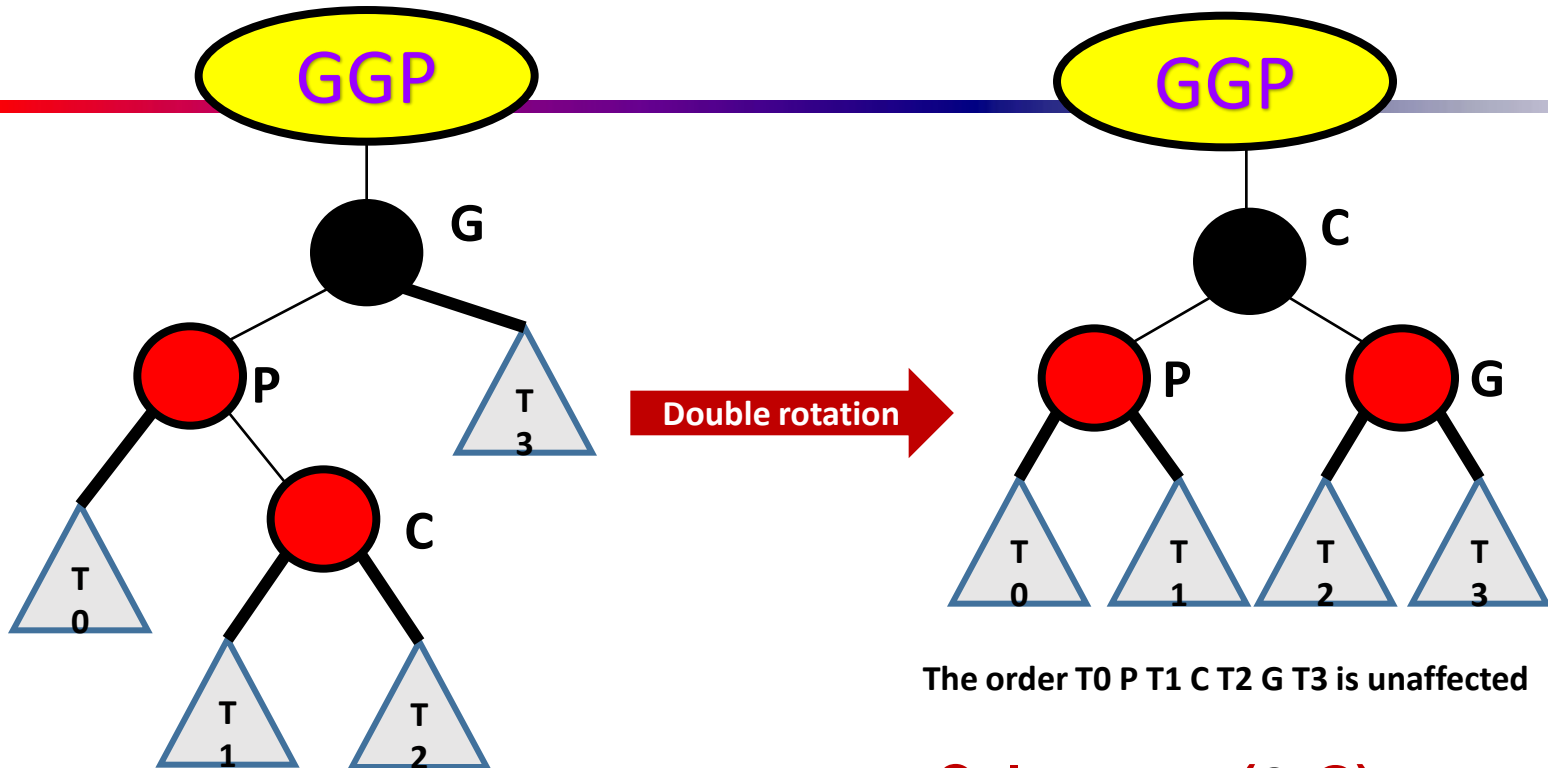

Single rotation

The order T0 G T1 P T2 C T3 is unaffected.

Color swap (P, G)

GGP has P as child

LR_RotateAt(P)

Use it when G has P as Left child, Parent has C as Right child

Double rotation

The order T0 P T1 C T2 G T3 is unaffected
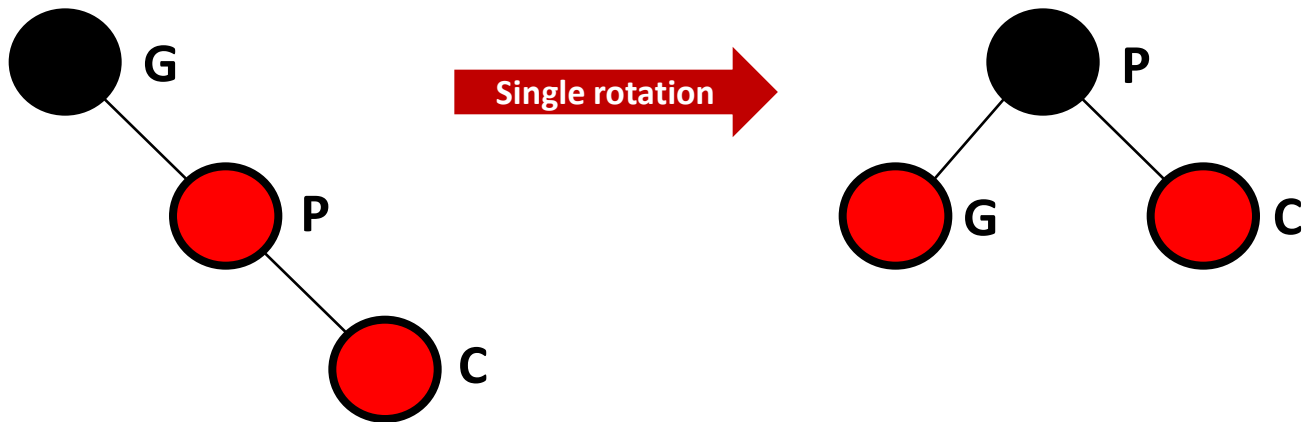
Color swap (C, G)
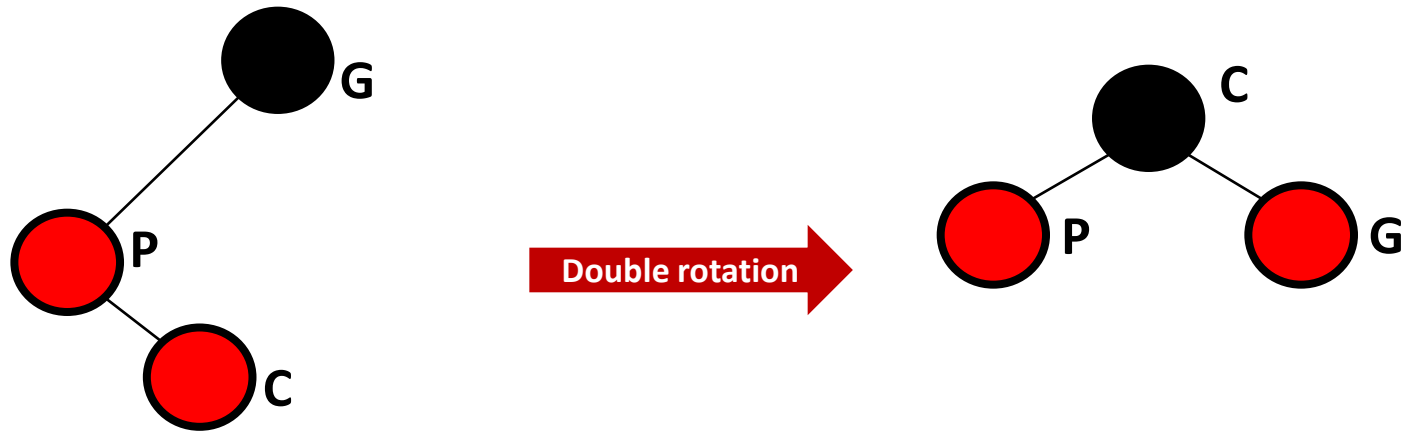
GGP has C as child

Left Rotation   Left_RotateAt(P)

Use it when G has P as Right child, Parent has C as right child

LR_RotateAt(P)
Use it when G has P as Left child, Parent has C as Right child

Inserting a new node

- Create a node C and insert as in any BST

- Color C Red

- **Restore the Red-Black tree properties**
  - **This involves many different steps.**
  - **That is illustrated next.**

Five cases

C is the node that is RED, C's Parent P is also RED.

Case 1: C has a RED uncle U

Change Parent P, Uncle U to BLACK

Change Grand Parent G to RED

(Recursively do it until no two consecutive reds).

REMAINING FOUR CASES ARE DONE ONLY IF THERE IS NO RED UNCLE

NO RED UNCLE

Case 2. LL (P is a LEFT child of G, C is a LEFT child of P). Right Rotate at P.

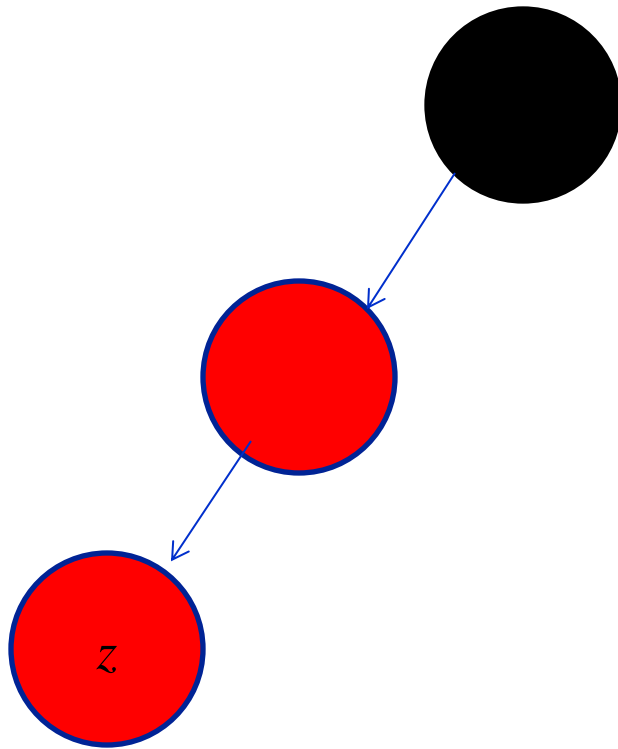Case 3. RR (P is a RIGHT child of G, C is a RIGHT child of P). Left Rotate at P.

Case 4. LR (P is a LEFT child of G, C is a RIGHT child of P). LR Rotate at P.

Case 5. RL (P is a RIGHT child of G, C is a LEFT child of P). RL Rotate at P.

At the end, assign BLACK color to Root.

Red-Black Insertion

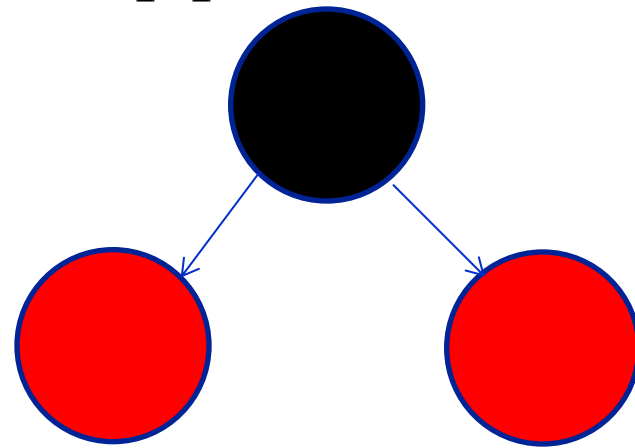Code – Case 3

[Right Rotate]

z

Red-Black Insertion
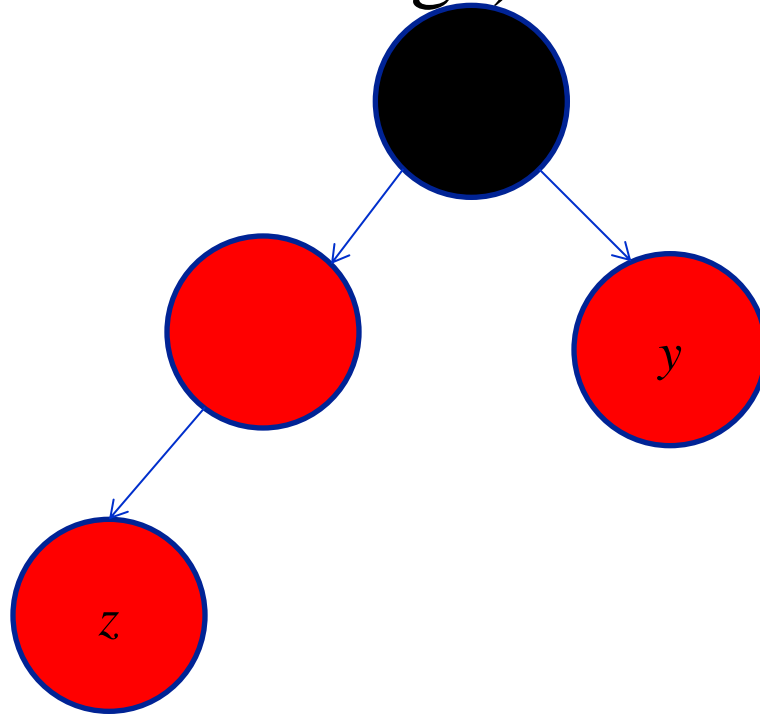
Code – Case 3

Change color:

Patent, GreandParent

Right-Rotate(T, z.p.p)

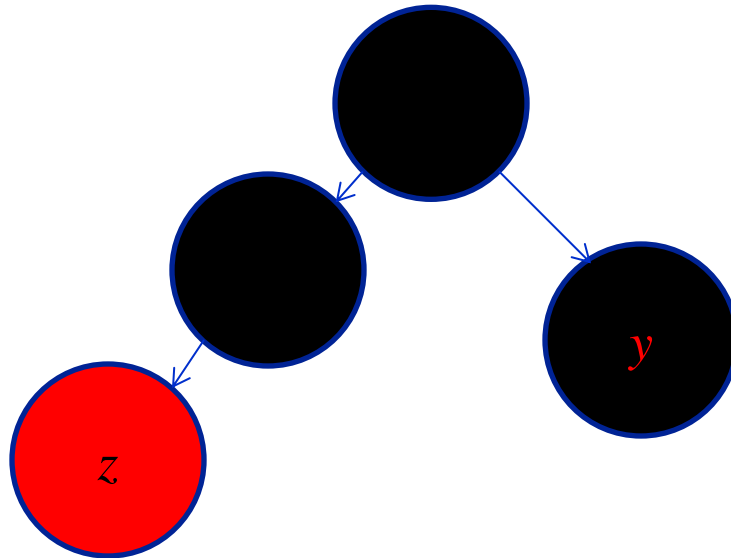# Red-Black Insertion

Code – Case 1 : Has a red uncle

(Just color change)

Red-Black Insertion

Case 1 P : Black, Uncle: Black,
Grand Parent: **Red**)

Make **root Black**

Red-Black Trees

Red-Black Trees: Key Comments

- *So, the trick is to use Red and Back labels to nodes and use appropriate rules to manipulate nodes in such a way that makes the tree balanced after inserting a new value.*

RB Trees: Worst-Case Time

- We've proved that a red-black tree has O(lg n) height

- Corollary: These operations take O(log n) time:

  - Minimum(), Maximum(), Successor(), Predecessor(), Search()

- Insert() and Delete():

  - Will also take O(log n) time

  - But will need special care since they modify tree

Using a Red-Black Tree as a Dictionary

1. A dictionary is a collection of pairs (k, e), which we consider to be key/value pairs. The purpose of a dictionary is to make it possible to look up values by key.

2. The Dictionary ADT supports these operations:

   1. get(k) returns value (or e).
   2. put(k, e) insert the pair into dictionary.
   3. remove(k) remove any key/value pair whose key is k.
   4. containsKey(k) returns true if k occurs as a key in the dictionary.

Dictionary Implementations

1. List. Key/value pairs can be inserted into any kind of list. Insertion can be done by adding to the end of the list in O(1) time. However, lookups and deletes require O(n) time.

2. Hashtable. O(1) average case time for insert, lookup and delete operations. However, hashtables do not store keys in any particular order.

Dictionary Implementations

3. Red-Black Trees. When keys have a natural ordering (like integers or strings), key/value pairs can be stored in the nodes of a red-black tree and ordered by keys. Red-black tree has O(log n) worst case time for insert, lookup and delete.

Ordered Dictionary

- An ordered dictionary is a dictionary that maintains its data in sorted order.

- Red-black tree is an example of ordered dictionary

- The price paid for maintaining the order is
  - $O(\log n)$ vs. $O(1)$ (Hashtable).

- Java's implementation of an ordered dictionary is TreeMap.

Main Point 3

The integrity of red-black trees is preserved after tree operations (insertions and deletions) are performed by maintaining the balance condition after execution of each operation. This maintenance does not increase the cost of operations because it requires only constant time, involving local color changes, color flips, and rotations.

Main Point 3

Science of Consciousness: The ability to maintain its fundamental character in the face of change is the expression of the *invincible* quality of pure consciousness. Pure consciousness, in giving rise to diversity, maintains its unbounded and immortal status. In society, this invincible quality is seen when a small percentage of a population engages in group practice of the TM and TM-Sidhi Programs – the inherent harmony of the society is enlivened to the extent that it "averts the birth of an enemy."

Connecting the Parts of Knowledge With the Wholeness of Knowledge

RED-BLACK TREES

1. In order to keep the benefits of a BST but eliminate the possibility of a worst case running time of $\Omega(n)$, different types of balance conditions have emerged, resulting in AVL trees and others.

2. Among enhanced BSTs with a balance condition, red-black trees are the most efficient. They outperform AVL trees by eliminating the need, in the worst case, to traverse the full height of the tree more than once.

Connecting the Parts of Knowledge With the Wholeness of Knowledge

3. *Transcendental Consciousness* is the home of all the laws of nature. Enlivenment of this field brings support of nature's almighty power to the field of action.

4. *Impulses Within The Transcendental Field*. Within its unmanifest nature, pure consciousness, as a field of all possibilities, is capable of accomplishing anything, and sets forth a blueprint for manifesting the entire universe.

Connecting the Parts of Knowledge With the Wholeness of Knowledge

5. *Wholeness Moving Within Itself.* In Unity Consciousness, the transformational dynamics of consciousness, at the basis of all of creation, are appreciated as the lively impulses of one's own consciousness, one's own being.

Visualization

https://www.cs.usfca.edu/~galles/visualization/RedBlack.html