CME 2201 Assignment 3

BALANCED RED BLACK TREE

Due date: 15.12.2017 Friday 23:59. Late submissions are not allowed.

You must upload your all '.java' files as an archive file (.zip or .rar). Your archived file should be named as 'studentnumber_name_surname.rar/zip', e.g., 2016510031_Mehmet_Cengiz.rar.

Control date: Control of the homework will be on 27th December week. Research assistants will control your assignments in their office. Therefore, you should write your name to the schedule list on the door of R.A. (door no: 124). You will have 10 minutes to show your assignment.

Plagiarism Control: The submissions will be checked for code similarity. Copy assignments will be graded as zero, and they will be announced in the Classroom.

Please do not forget to bring your laptops while coming to the assignment control!

In this assignment, you are expected to implement a modified red black tree named Balanced Red Black Tree in Java.

RULES

- **Input Type:**
  - V = Value of node. (Mehmet)
  - K = Key of node. (29)
- **Balancing Conditions:**
  - If *root* is not **BLACK**.
  - If two nodes in a row are **RED**.
  - If the difference of depth between any two subtrees is more than 1.
- **Balancing Transactions:**
  - If *root* = ***RED*** then *Recolor(root).*
  - If *node.uncle* = ***RED*** then *Recolor(node.parent, node.grandparent, node.uncle).*
  - If *node.uncle* = ***BLACK*** and node_positions = ***TRIANGLE*** then *Rotate(node.parent).* (Figure 1.A)
  - If *node.uncle* = ***BLACK*** and node_positions = ***LINE*** then *Rotate(node.grandparent)* and *Recolor(node.parent, node.grandparent).* (Figure 1.B)
  - If the height of the left and right subtrees of each node is more than 1 then *Rotate(node.grandgrandparent)* and *Recolor(node.grandparent, node.grandgrandparent).*
- Usage of Abstract Data Types (ADT) is required.
- Object Oriented Principles (OOP) and try-catch exception handling must be used when it is needed.
- After balancing process, difference between depth of the right and left tree at any part of the tree should be 0 or 1.
- All Red Black Tree and balance codes must be implemented by yourself.

- **Main Functionalities:**
    - **INSERT:** Insert the given inputs to the tree.
    - **SEARCH:** Search a user input (word) in the tree and return word, key, color, parent, grandparent, and uncle information of the node.

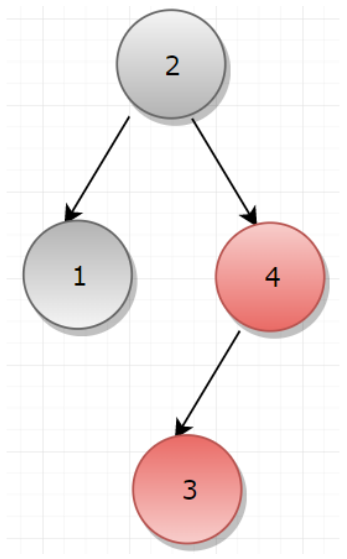


Figure 1.A – node_position = **TRIANGLE**

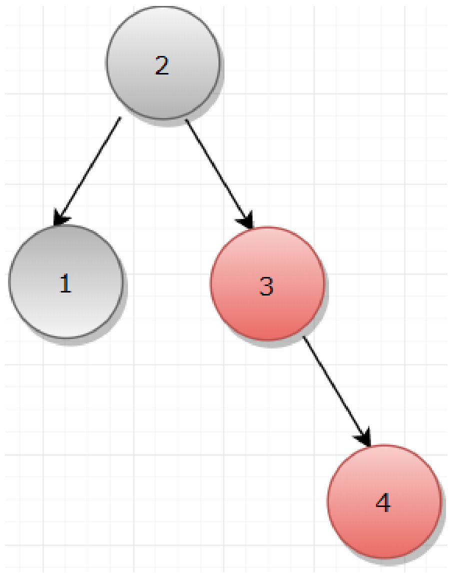


Figure 1.B – node_position = **LINE**

SEARCHING EXAMPLE

Search: Mehmet

Key: 6

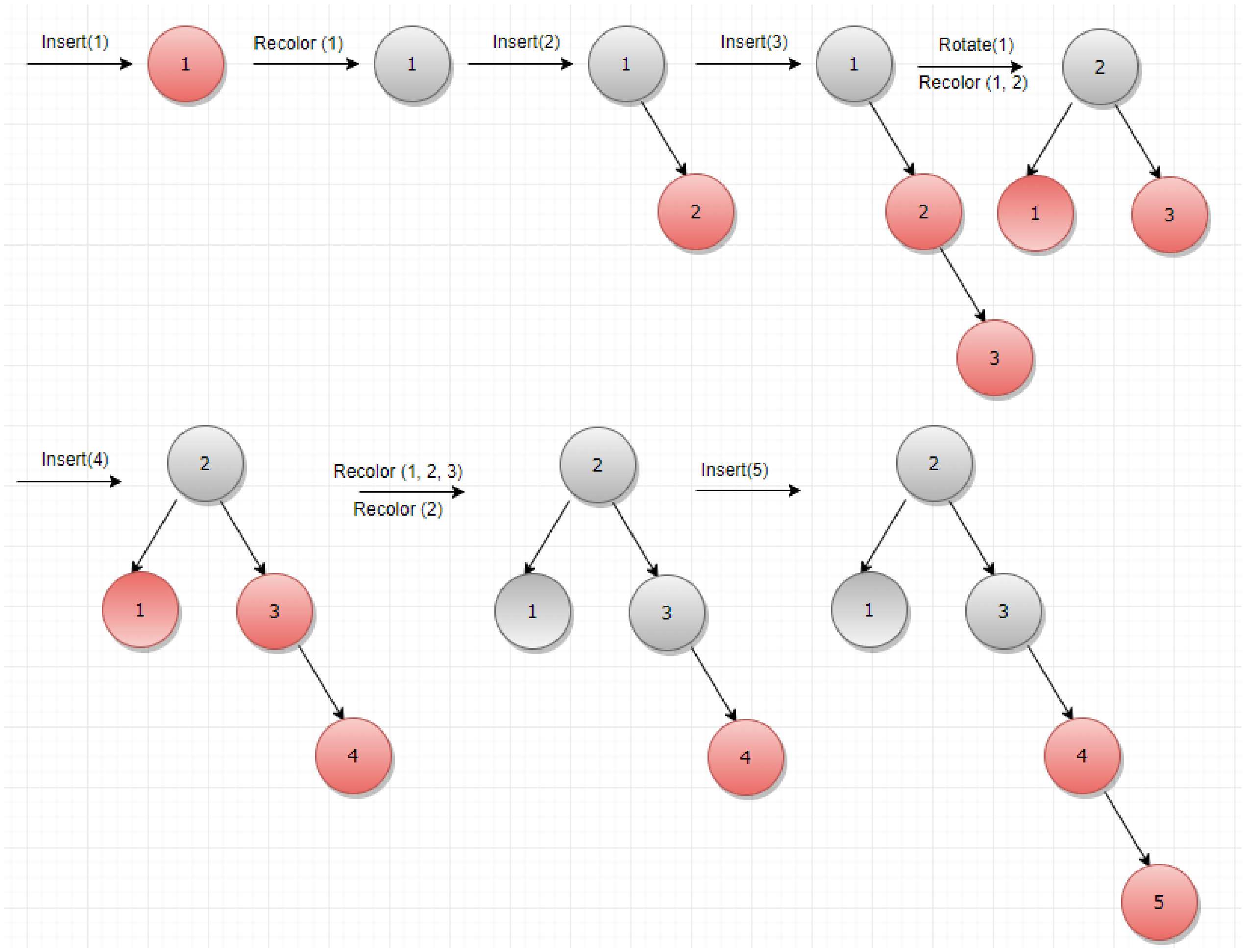
Color: RED

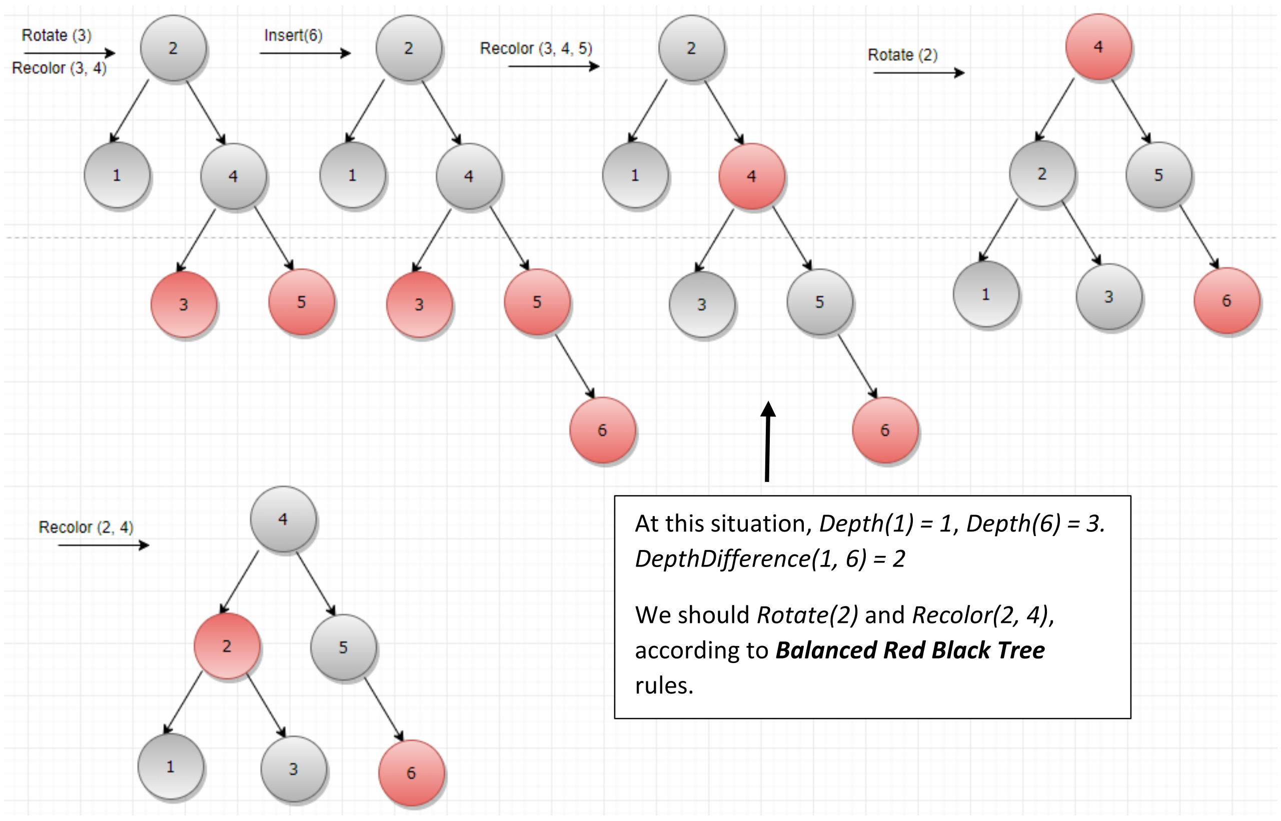
Parent.Key: 5 -> Parent.Color: BLACK

Grandparent.Key: 4 -> Parent.Color: BLACK

Uncle.Key: 2 -> Parent.RED

INSERTION EXAMPLES





At this situation, *Depth(1) = 1*, *Depth(6) = 3.*
DepthDifference(1, 6) = 2

We should *Rotate(2)* and *Recolor(2, 4)*, according to ***Balanced Red Black Tree*** rules.

Grading Policy

Items	**Percentage**
1- Usage of ADT, OOP and Try-Catch	%30
2- Implementation of Implementation of Balanced Red Black Tree	%50
3- Search over the tree	%20