ST. CHARLES COMMUNITY COLLEGE

CPT-281 - Introduction to Data Structures with C++

Module 14

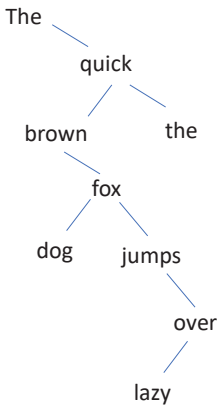
Self-Balanced Binary Search Trees

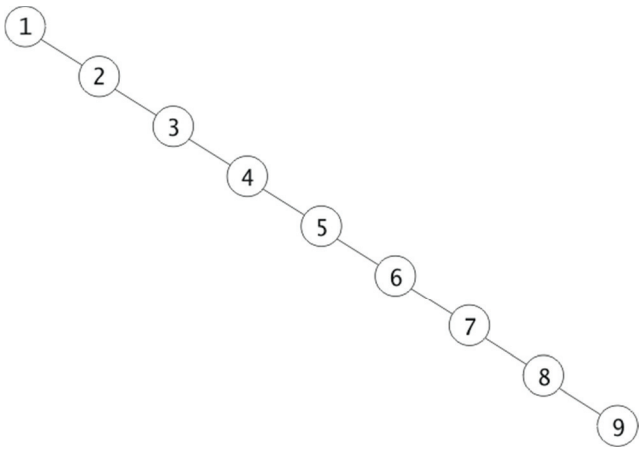
Dayu Wang

- **Unbalanced Binary Search Trees**

- The performance of binary search trees on average is **O (log n).**
- However, if the tree is heavily unbalanced, the performance could be **O(n)**

"The quick brown fox jumps over the lazy dog"



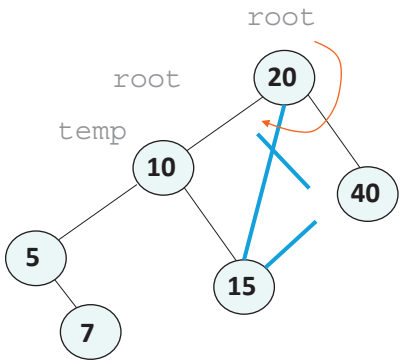


- **Rotation**

- **Rotation** helps us change the relative height of left & right subtrees while preserving the binary search tree property

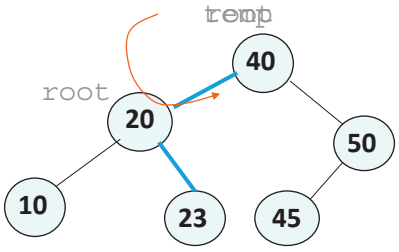
The Algorithm for Rotating Right

1. Save value of `root->left` (`temp = root->left`)
2. Set `root->left` to value of `root->left->right`
3. Set `temp->right` to `root`
4. Set `root` to `temp`



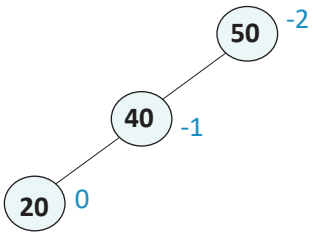
- **Rotation Left**

1. Save value of `root->right` (`temp = root->right`)
2. Set `root->right` to value of `root->right->left`
3. Set `temp->left` to `root`
4. Set `root` to `temp`



• **AVL Trees**

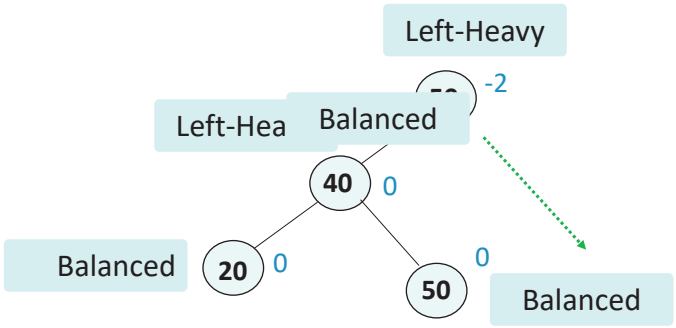
- Two Russian mathematicians, G.M. **A**del'son-**V**el'ski and E. M. **L**andis, published a paper in 1962 that describes an algorithm for maintaining overall balance of a binary search tree. Trees using this approach are known as **AVL trees** after the initials of the inventors.



Each node's balance = $h_R - h_L$

• **Balancing a Left-Left Tree**

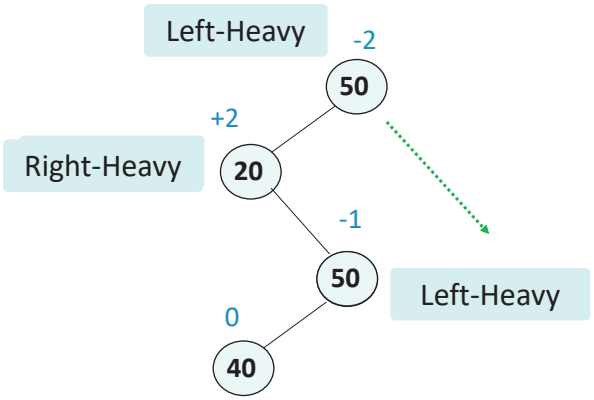
- To balance a left-left tree, we rotate right around parent.



• **Balancing a Left-Right Tree (I)**

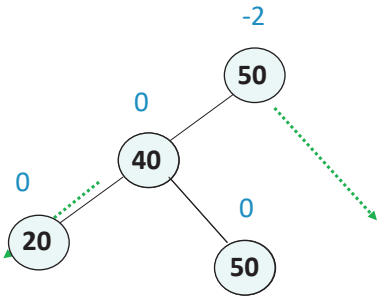
- Can we balance it by rotating the root to the right?

  No, we can't



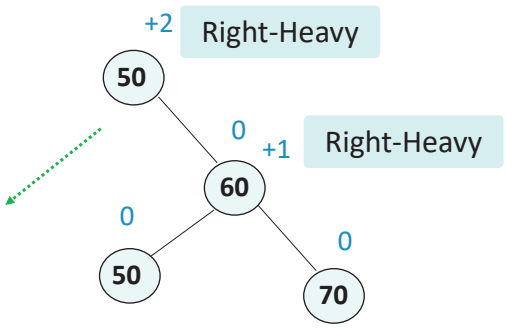
• **Balancing a Left-Right Tree (II)**

- Instead, we should rotate left around child.
- This will make it a left-left tree.
- Now we rotate right around parent.



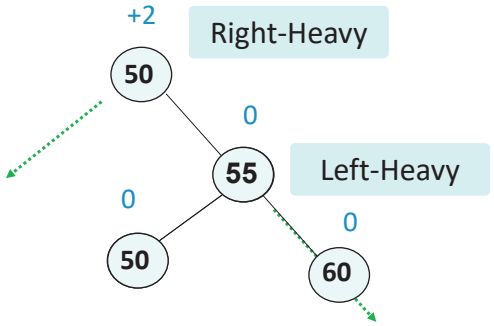
• **Balancing a Right-Right Tree**

- To balance a right-right tree, rotate left around parent.

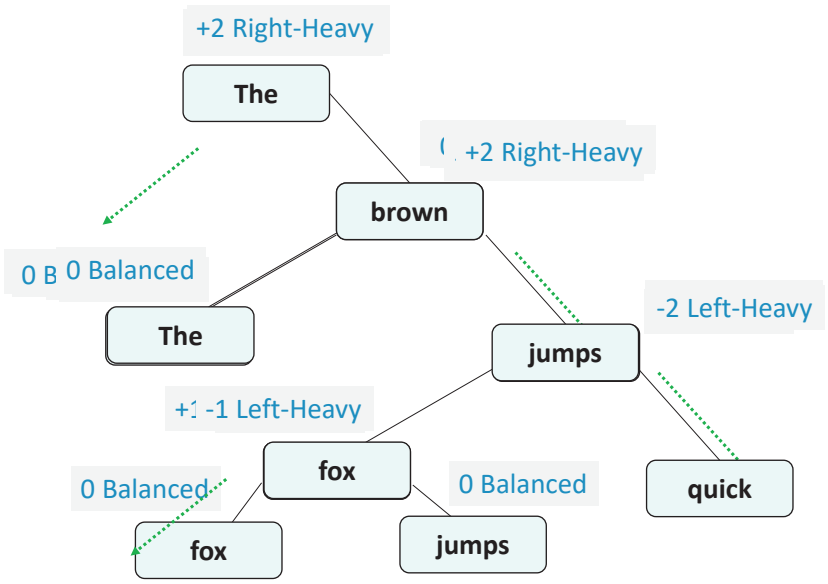


• **Balancing a Right-Left Tree**

- To balance a right-left tree, rotate right around child first. This will make it a right-right tree.
- Now rotate left around parent.

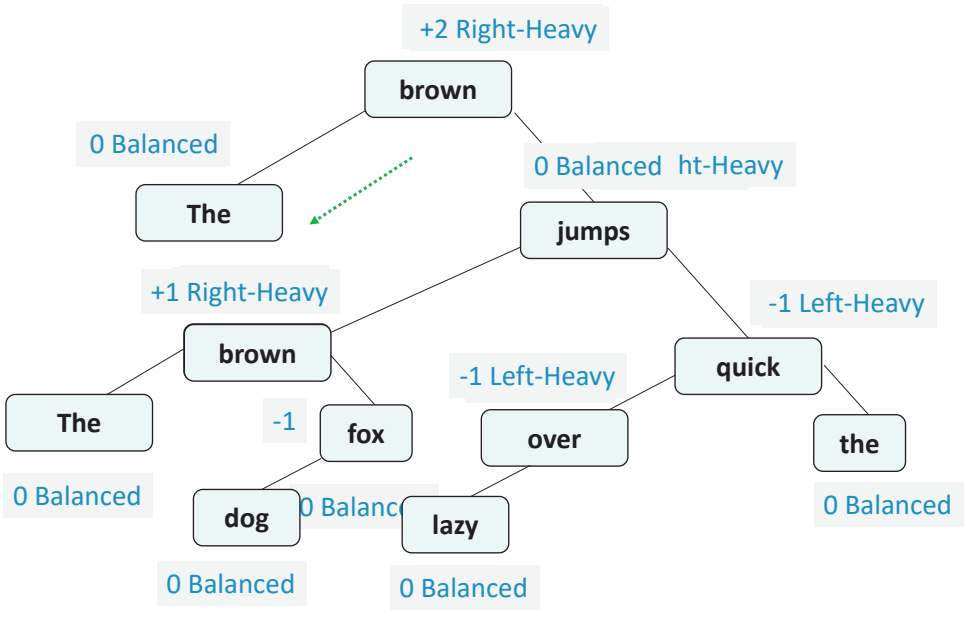


- **Building an AVL Tree**



"The quick brown fox jumps over the lazy dog"

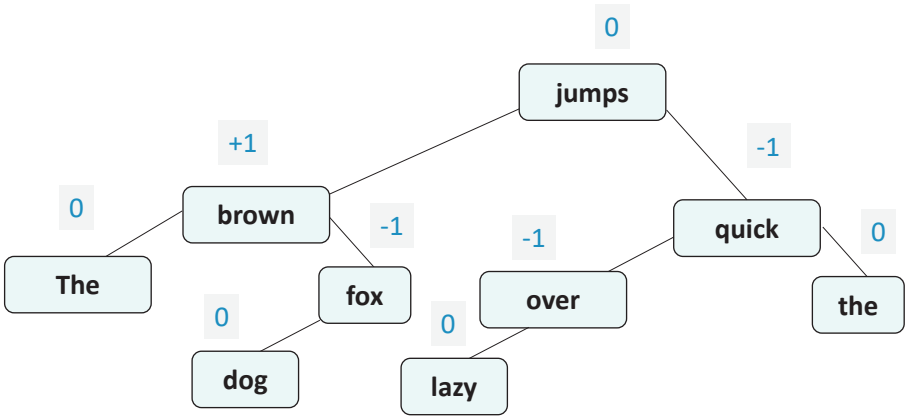
- **Building an AVL Tree**



"The quick brown fox jumps over the lazy dog"

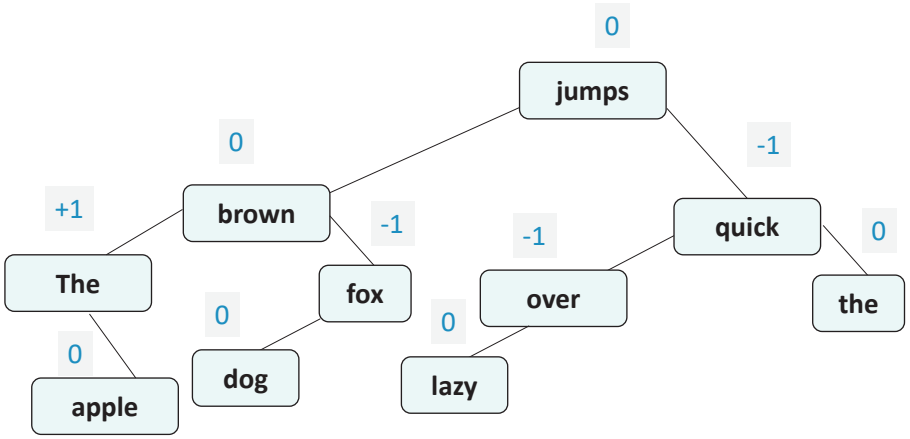
• **Exercise 1**

- Show the final AVL tree for the changes as you insert "apple", "cat", and "hat".



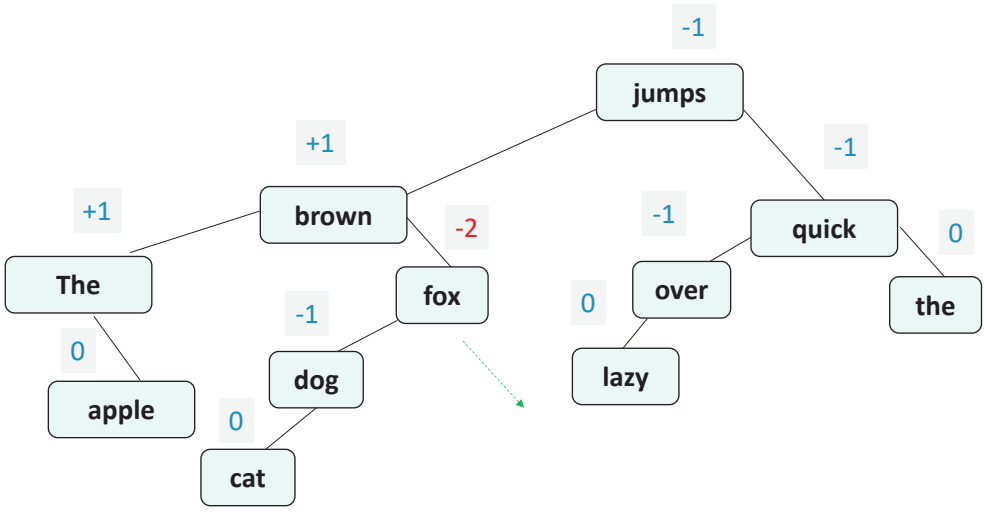
• **Answer of Exercise 1**

- **After inserting "apple"**



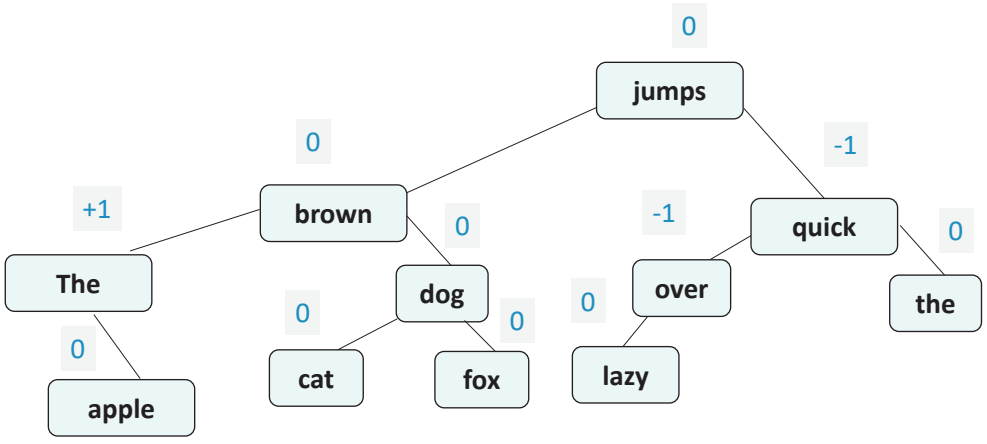
- **Answer of Exercise 1**

- Inserting "cat"



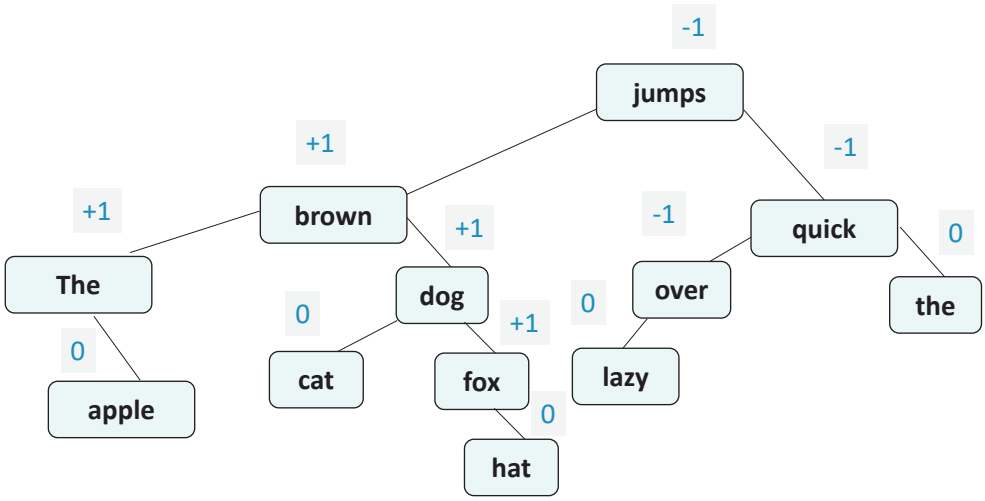
- **Answer of Exercise 1**

- After inserting "cat"



- **Answer of Exercise 1**

- **After inserting "hat"**

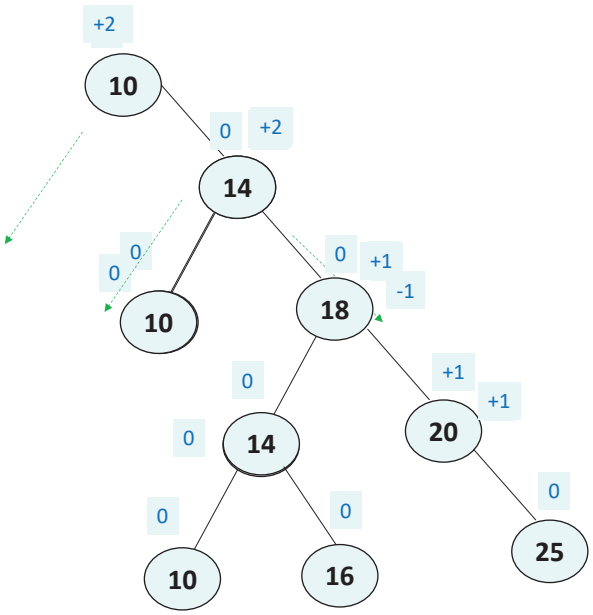


- **Exercise 2**

Build an AVL tree from the following integers: 10, 18, 14, 16, 20, 25.

Please insert the integers in the specified sequence, and show what happens after inserting every integer

- **Answer of Exercise 2**



10, 18, 14, 16, 20, 25

- **Outside Study Resource - Nice Illustration of AVL Trees**

  → **http://www.cs.usfca.edu/~galles/visualization/AVLtree.html**

• **Red-Black Trees**

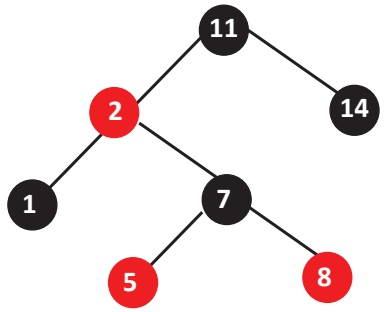
- A **Red-Black tree** maintains the following invariants:

  1. A node is either red or black.

  2. The root is always black.

  3. A red node always has black children (a NULL pointer is considered to refer to black node.)

  4. The number of black nodes in any path from the root to a leaf is the same (The root node's left and right subtrees must be the same black-height)



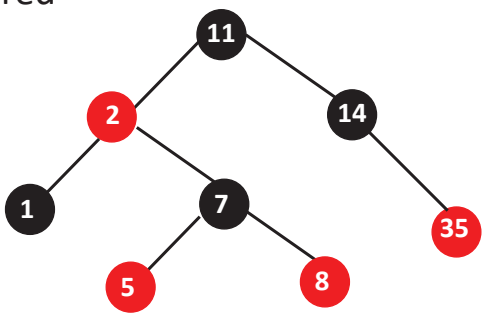
• **Insertion into a Red-Black Tree**

1. Search for where the item belongs based on the binary search algorithm.

2. When the item is found, it is initially given the color red (to maintain invariant 4)

For instance, we are inserting "35" into the shown binary tree.

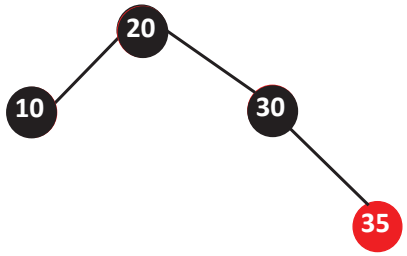
3. If the parent is black, we are done.



• **Insertion into a Red-Black Tree**

1. However, if the parent is red, we violated invariant 3.

2. In this case, if the parent's sibling is also red, then we can change the grandparent's color to red, and change the parent and the parent's sibling color to black.
(we maintained invariants 3 and 4)



3. To maintain invariant 2, we change the root color to black.

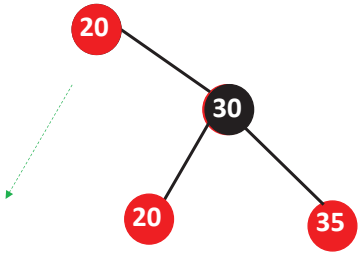
• **Insertion into a Red-Black Tree**

1. The parent is red, but doesn't have a sibling.

(we violated invariant 3)

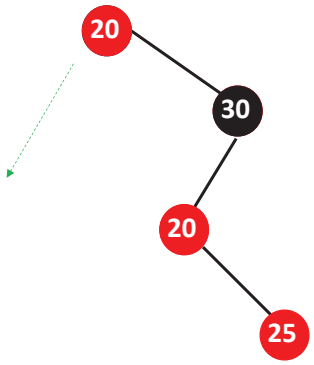
2. To maintain invariant 3, we change the parent and the grandparent colors.

(we violated invariant 4)



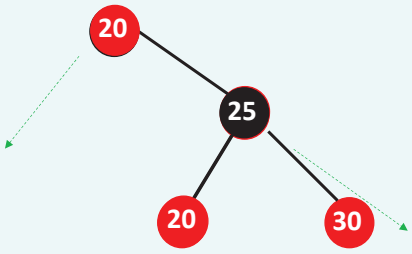
3. We correct this by rotating about the grandparent.

- **Insertion into a Red-Black Tree**

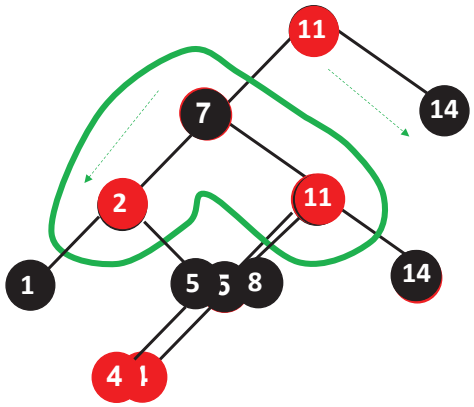


Doesn't work

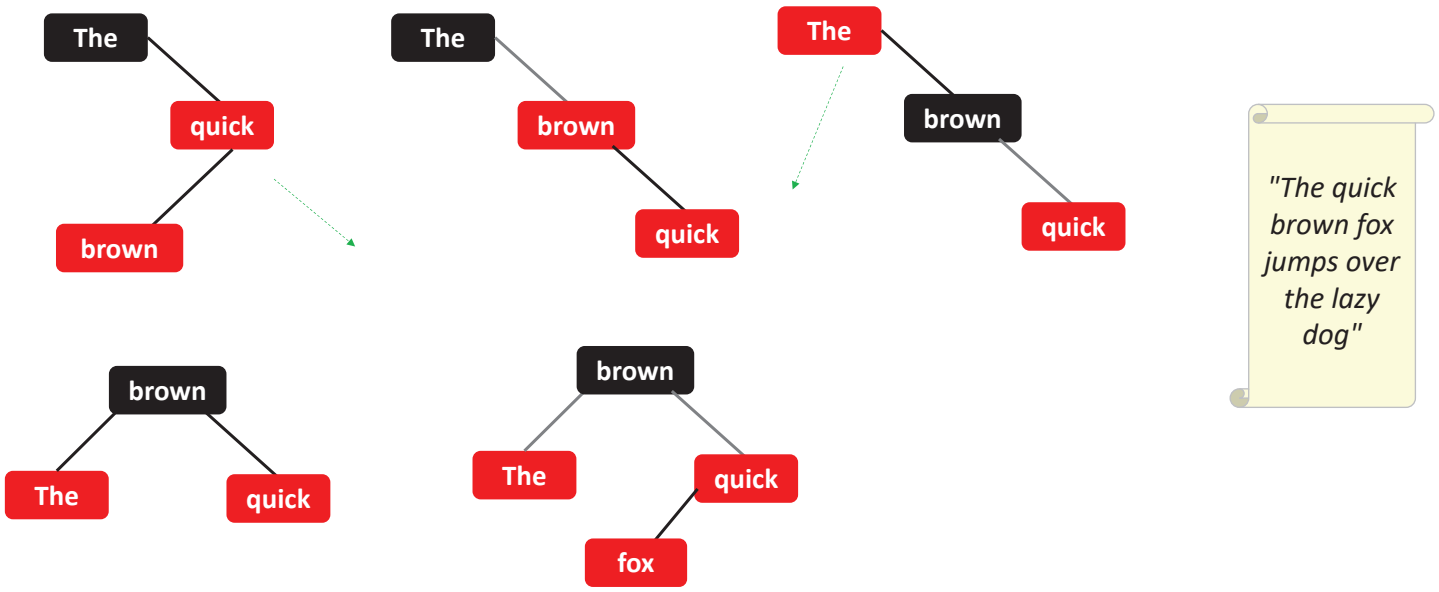
Still a red parent-red a child relationship



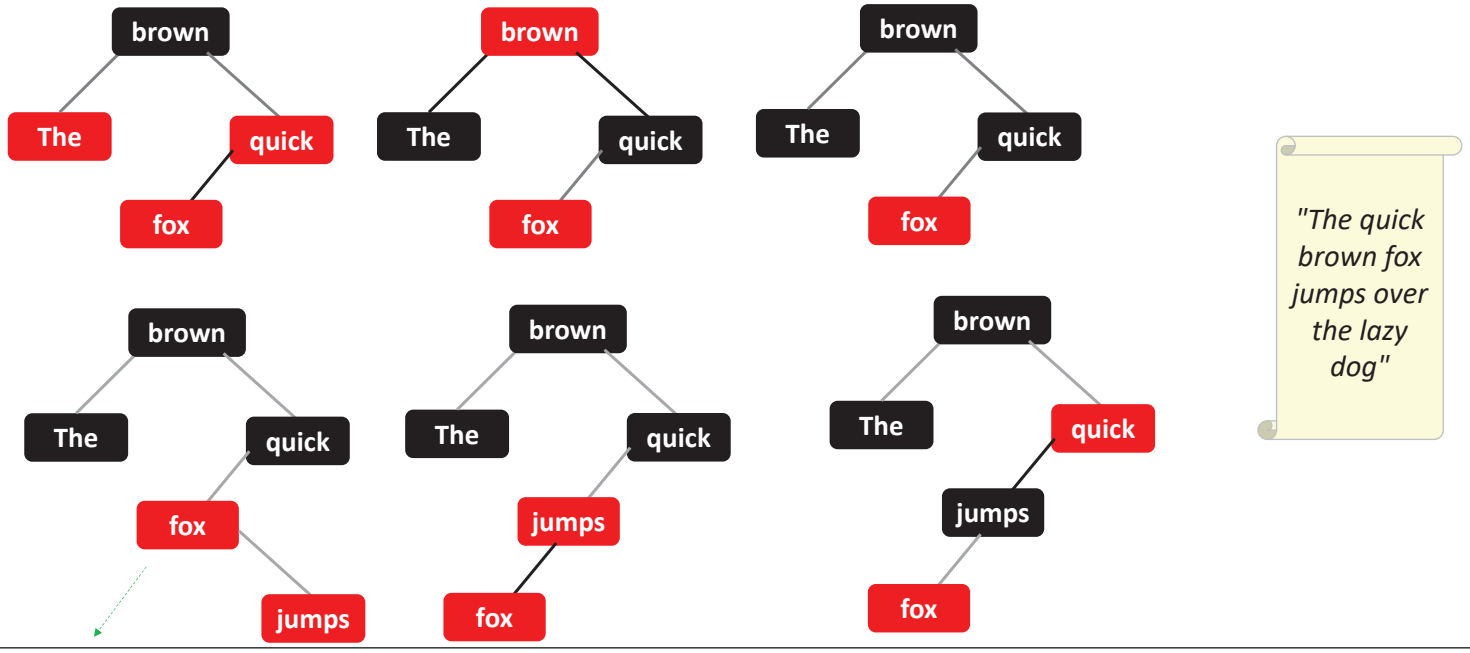
- **Insertion into a Red-Black Tree**



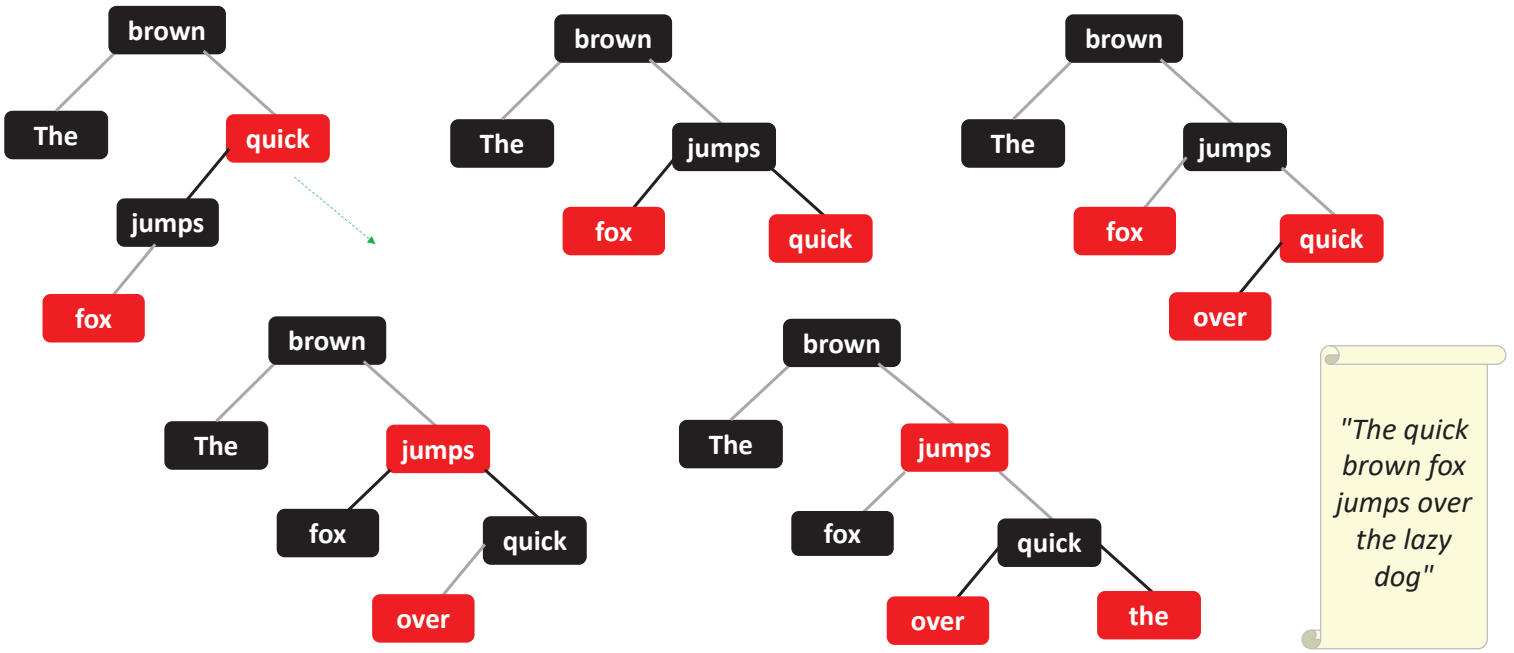
- **Building a Red-Black Tree**



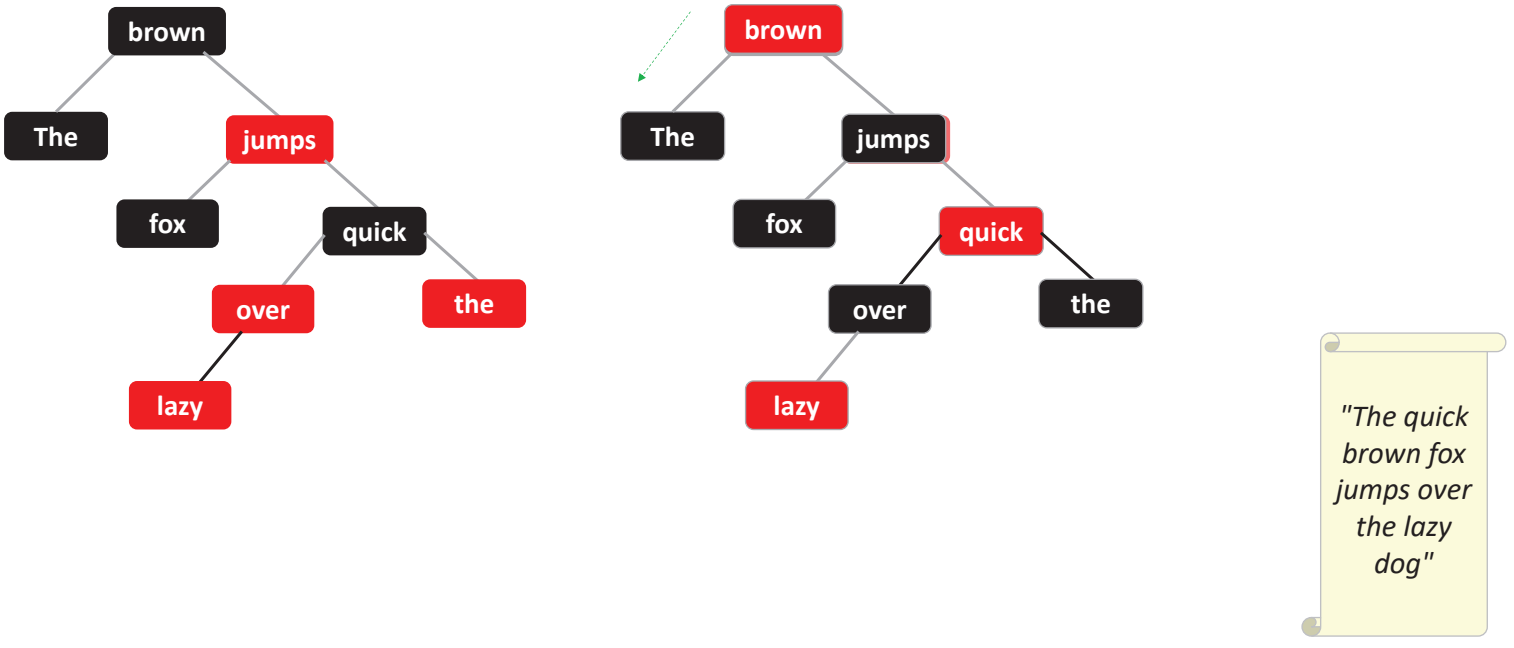
- **Building a Red-Black Tree**



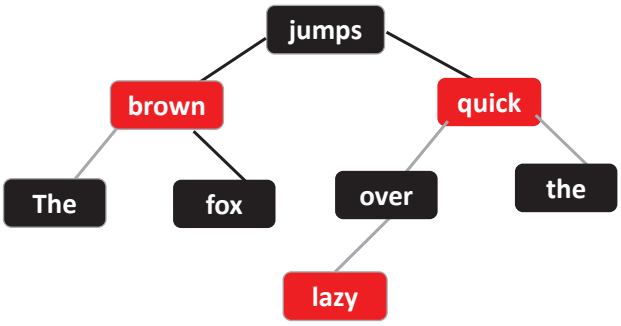
- **Building a Red-Black Tree**

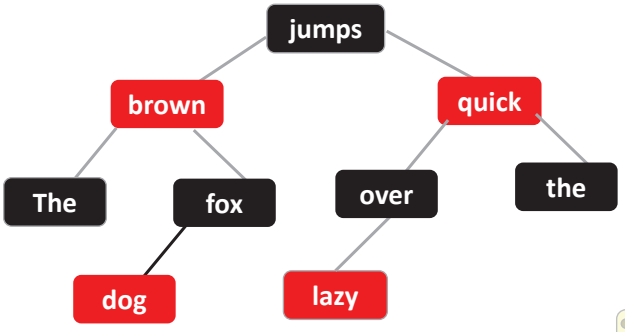


- **Building a Red-Black Tree**



- **Building a Red-Black Tree**

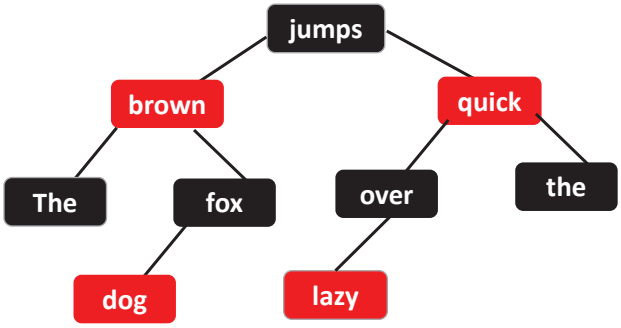




"The quick brown fox jumps over the lazy dog"

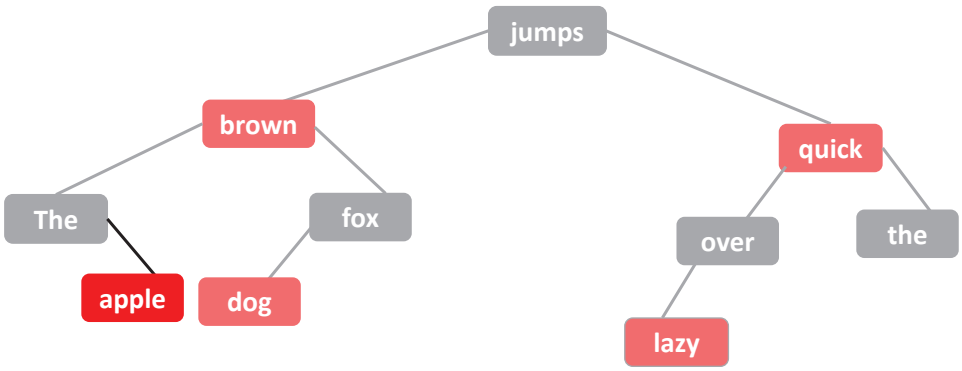
- **Exercise 3**

- Show how the Red-Black tree below as you insert "apple", "cat", and "hat" in that order.



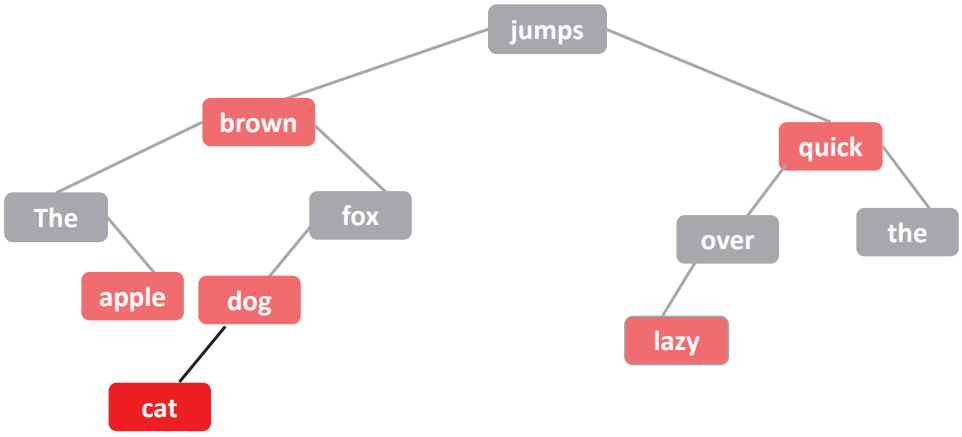
- **Answer of Exercise 3**

  - After inserting "apple"



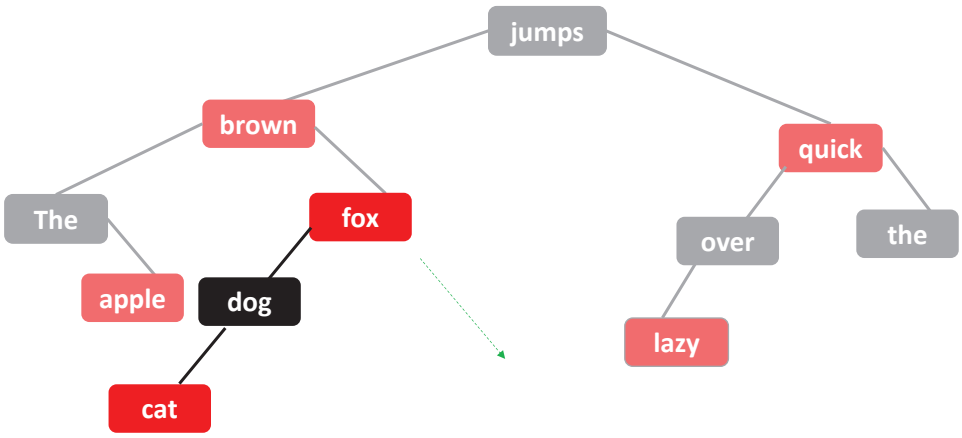
- **Answer of Exercise 3**

  - After inserting "cat" (A)



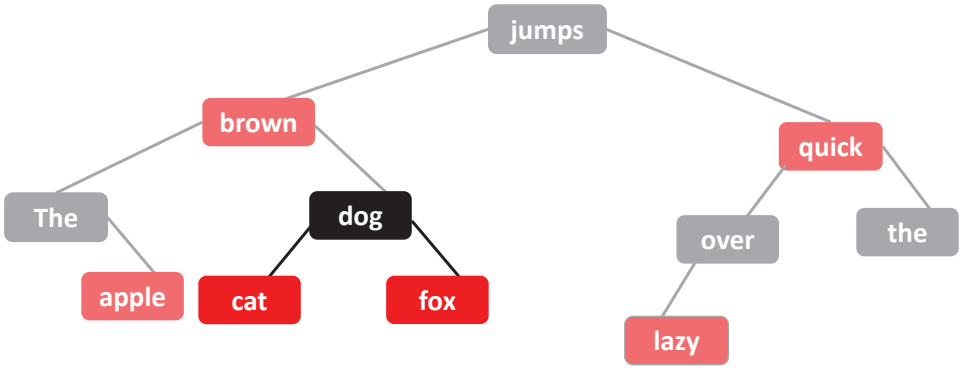
- **Answer of Exercise 3**

  - After inserting "cat" (B)



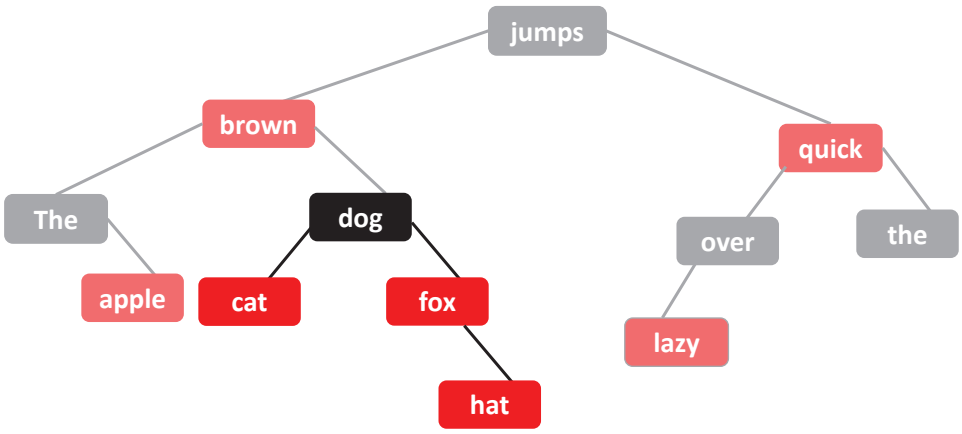
- **Answer of Exercise 3**

  - After inserting "cat" (B)



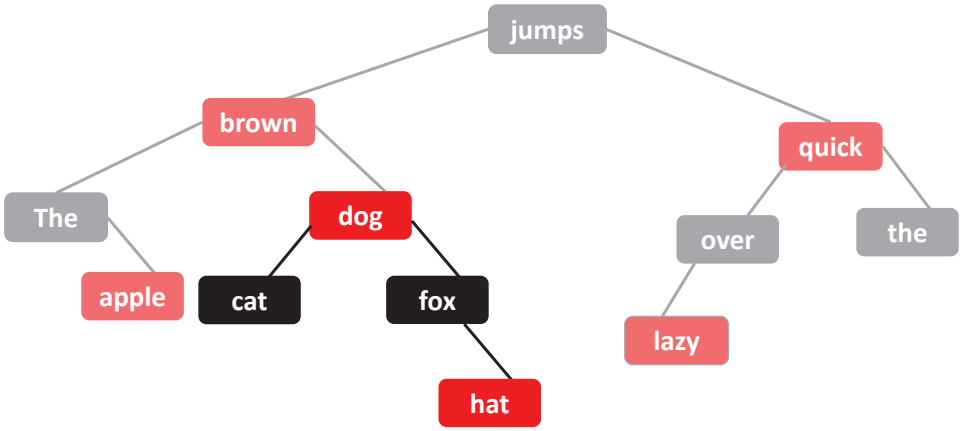
- **Answer of Exercise 3**

  - After inserting "hat"



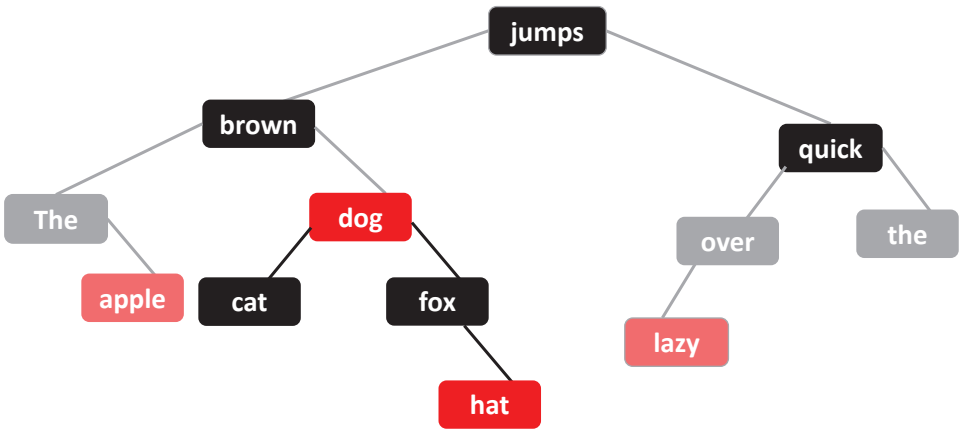
- **Answer of Exercise 3**

  - After inserting "hat"



• **Answer of Exercise 3**

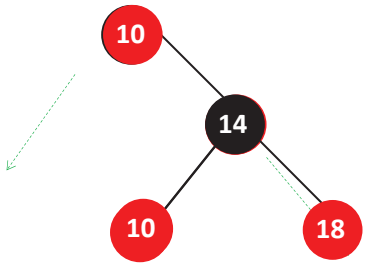
- After inserting "hat"



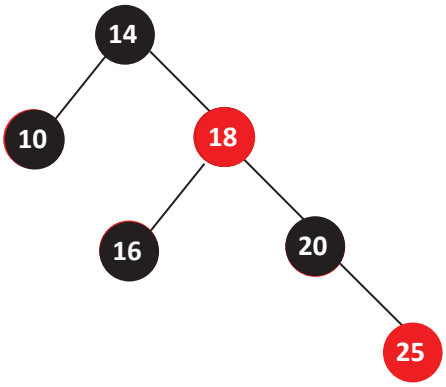
• **Exercise 4**

- Build a red-black tree from the following integers: 10, 18, 14, 16, 20, 25.
- Please insert the integers in the specified sequence, and show what happens after inserting every integer

- **Answer of Exercise 4**



- **Answer of Exercise 4**



- **Performance of AVL Trees and Red-Black Trees**

- The worst case of search in a red-black tree is 2log n + 2, which is still O (log n).
- For AVL trees, the worst performance for search is 1.44log n which is also O(log n).
- The average performance of a red-black tree is significantly better than the worst case.
- Both AVL and red-black trees give performance that is close to that of a complete binary search tree.
- Red-black trees are favorable to AVL trees because they require fewer rotations. AVL trees are better if the tree is to be mainly used for searching.