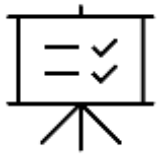
Advanced Algorithms & Data Structures

AIM OF THE SESSION

1. The students familiar with the basic concepts of root nodes, parents and Childrens nodes with real time examples and programming syntax.
2. These knowledge of basic concepts they can construction and demonstrate the RED-**BLACK** tree.

INSTRUCTIONAL OBJECTIVES



This Session is designed to:

1. Demonstrate RED-**BLACK** Tree Constructions with best, worst and average cases of time and space complexity
2. Describe RED-**BLACK** Tree basic terminology and why it is needed
3. List out the general ideas about RED-**BLACK** construction and why it is needed in data structure
4. Describe the structure of RED-**BLACK** tree constructions and its significance usages in computer science

LEARNING OUTCOMES

At the end of this session, you should be able to:

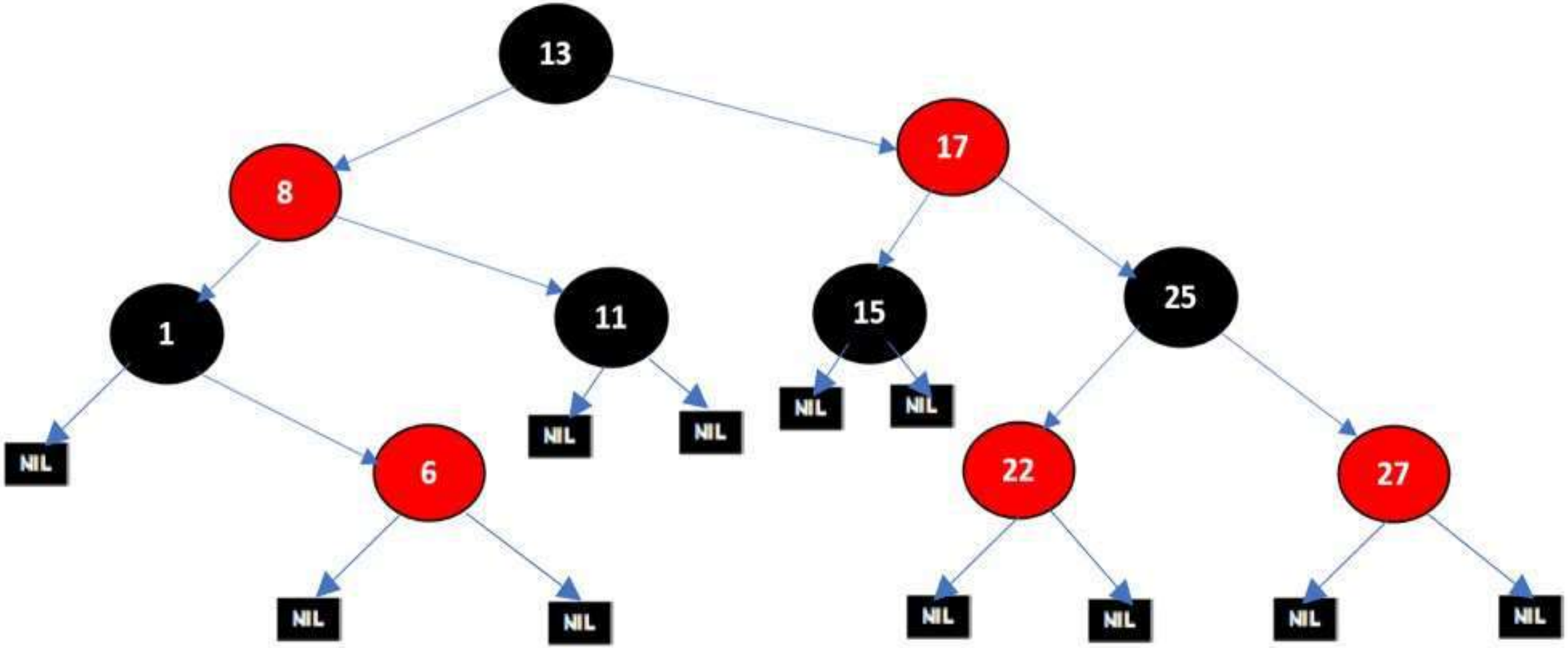
1. Define the concept of RED-**BLACK** Tree constructions phases, time and space complexity
2. Describe: RED-**BLACK** Tree constructions phases with all possible conditions
3. Summarize the RED-**BLACK** Tree importance in computer science in cases of storing and managing data

INTRODUCTION

- A **Red-Black** Trees are a type of self-balancing binary search tree.
- Why they are named as “**Red-Black**" because each node in the tree is marked with one of two colors, either **red** or **black**.
- **Red-Black** Trees were introduced by Rudolf Bayer and Volker Strassen in 1972.
- These colours are used to ensure that the tree remains balanced during insertions and deletions.

EXAMPLE OF RED-BLACK TREE

Constructing Red-**Black** tree with 10 nodes



REQUIREMENTS

- A **red-black** tree is a Binary tree where a particular node has colour as an extra attribute, either red or black.
- **Red-black** trees maintain a slightly looser height invariant than AVL trees.
- In AVL trees, balancing restricts the difference in heights to at most one.
- For **red-black** trees, we have a different set of rules related to the colors of the nodes

SYNTAX OF RED-BLACK TREE

To create a node in **Red-Black** Trees

Syntax:

```
struct Node
{
    int key;
    enum Color color;
    struct Node *left, *right, *parent;
};
```

PROPERTIES OF RED-BLACK TREE

1. Root property: The root is **black**.

2. External property: Every leaf (Leaf is a NULL child of a node) is **black** in **Red-Black** tree.

3. Internal property: The children of a **red** node are **black**. Hence possible parent of **red** node is a **black** node.

4. Depth property: All the leaves have the same **black** depth.

5. Path property: Every simple path from root to descendant leaf node contains same number of **black** nodes.

RULES FOR INSERTION OF NODE

- If the tree is empty, make newnode as root and color it with **black**.
- If tree is not empty, create newnode as leaf node and color it with **red**.
- If parent of newnode is **black** then exit.
- If parent of newnode is **red** then check the color of parent sibling

  a) If parent sibling color is **black** or null rotate and recolor.

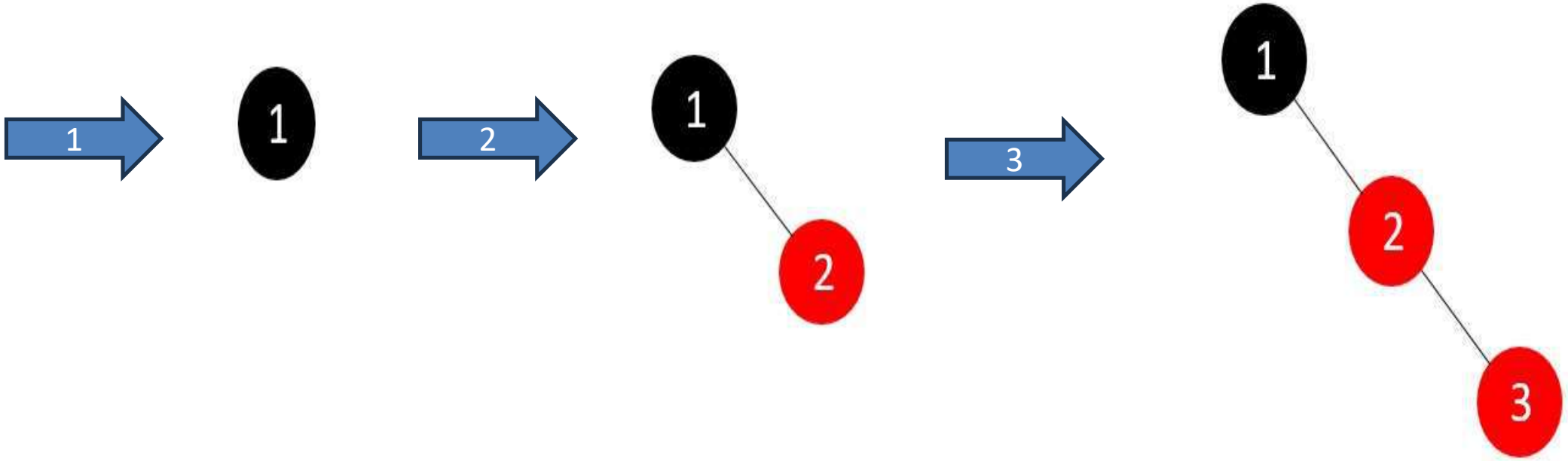
  b) If parent sibling color is **red**, then recolor parent and parent sibling

  c) And check parent's parent is not root node. Then recolor and recheck.

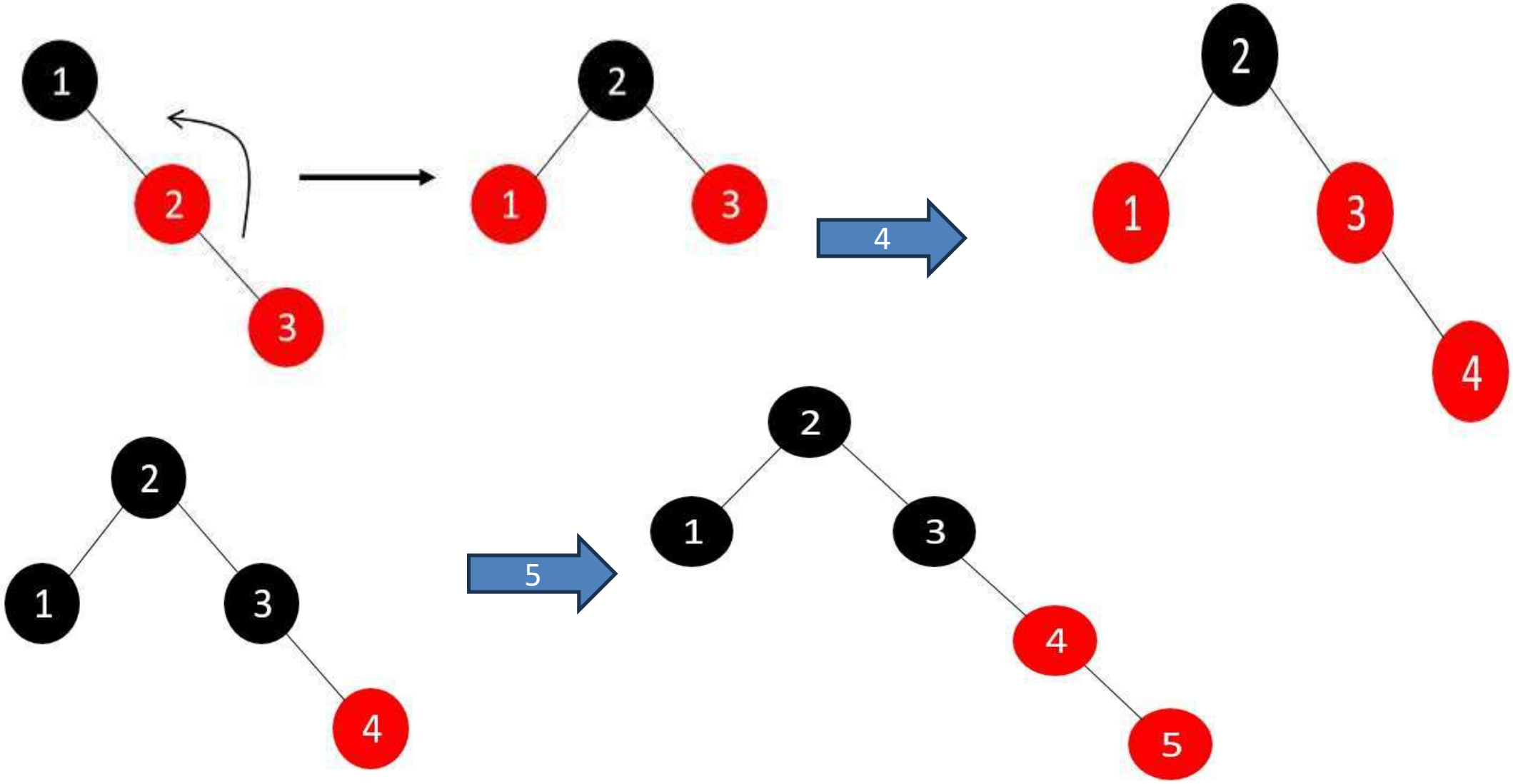
CONSTRUCTION OF RB TREES

- Let's understand the inserting elements with following the **red-black** tree properties and rules

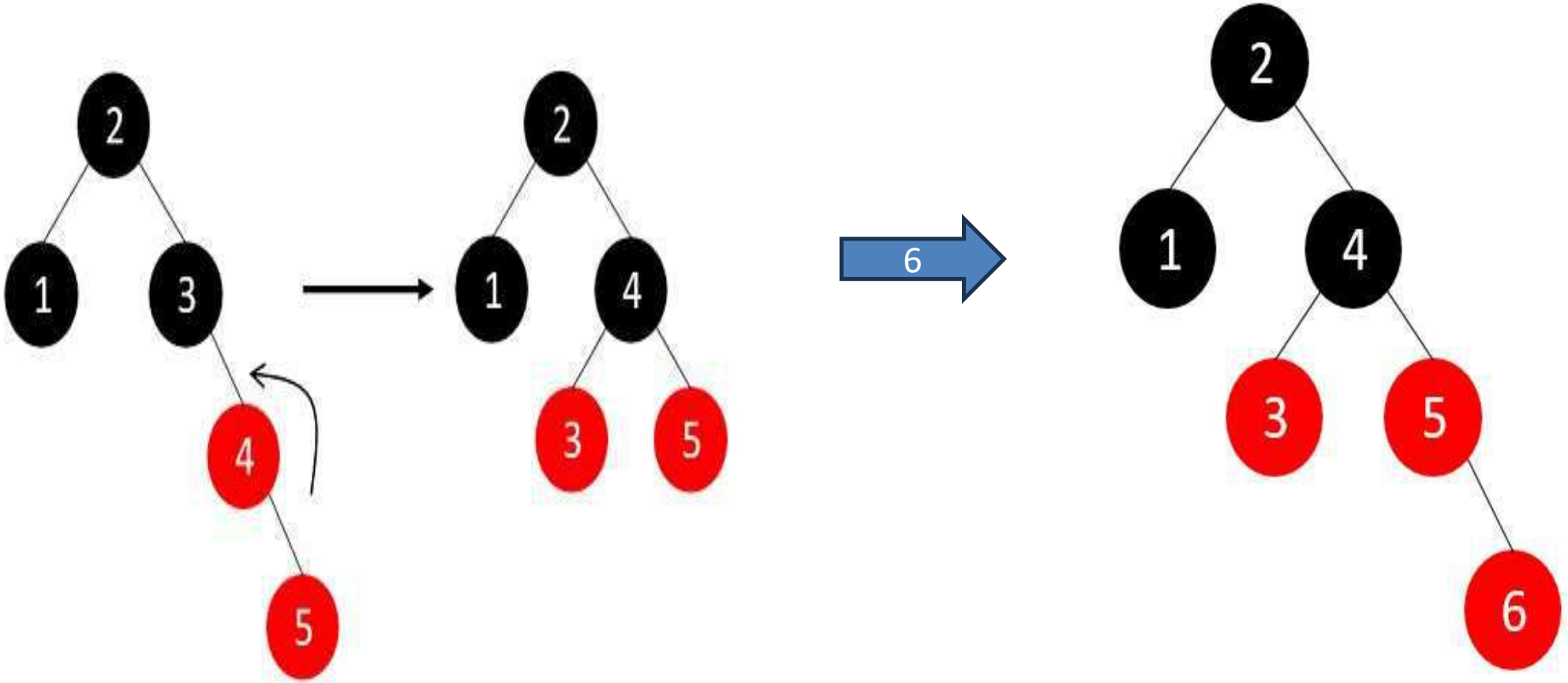
- Construct a red black tree with 1 to 7 numbers.



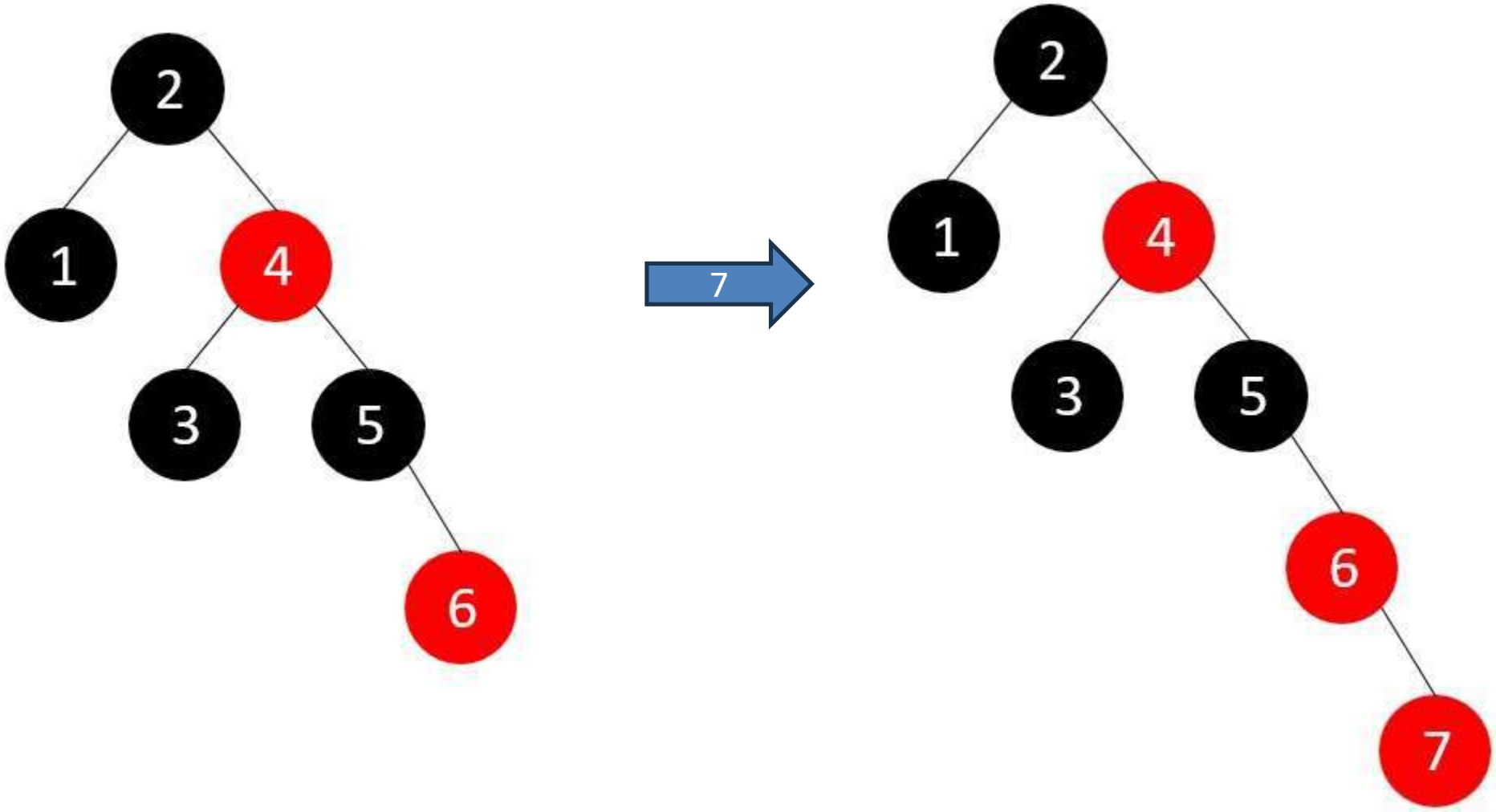
Example of RED-BLACK Tree



Example of RED-BLACK Tree

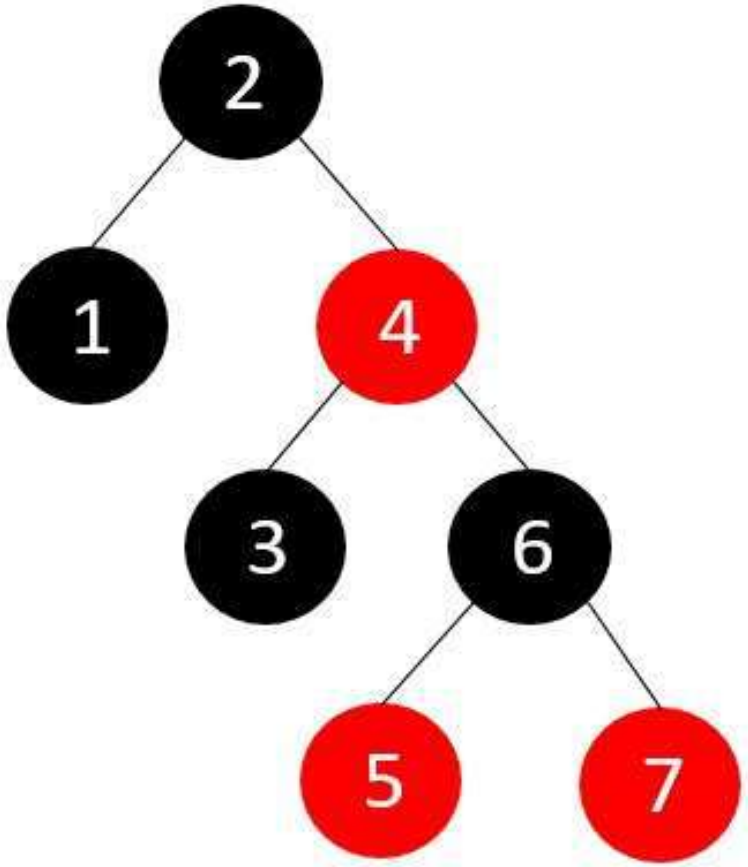


Example of RED-BLACK Tree



Example of RED-BLACK Tree

The final **Red** **Black** Tree with 1 to 7 Numbers



ADVANTAGES

- **Red Black** Trees have a guaranteed time complexity of O(log n) for basic operations like insertion, deletion, and searching.
- **Red Black** Trees are self-balancing.
- **Red Black** Trees can be used in a wide range of applications due to their efficient performance and versatility.
- The process which is used to maintain balance in **Red Black** Trees is relatively simple and easy to understand.

DISADVANTAGES

- A **red black** trees in a lot of cases is that it is a binary tree and thus lookups are O(log(n)) where as hash tables have a lookup of O(1).
- **Red Black** Trees require one extra bit of storage for each node to store the color of the node (red or black).
- Complexity of Implementation.
- Although **Red Black** Trees provide efficient performance for basic operations, they may not be the best choice for certain types of data or specific use cases.

EXAMPLE

1. Describe **RED –BLACK** tree with proper syntax and example
2. Construct the **RED-BLACK** tree with these elements
    a) 10,85,15,70,20,60,30,50,65,80,90,40,5,55.
    b) 10,18,7,15,16,30,25,40,60,2,1,70.
3. List out the time complexities of **RED – BLACK** tree based on different cases
4. Analyze the different cases of **RED-BLACK** tree operations and example

SUMMARY

- The **Red-Black** tree properties ensure a balanced structure, limiting the longest path from the root to any leaf to at most twice the length of the shortest path.
- This balance guarantees efficient average-case performance for search, insertion, and deletion operations, with a worst-case time complexity of O(log n).

SELF-ASSESSMENT QUESTIONS

1. What is the color of the root node in a Red-Black Tree?

A. Red
B. Black
C. Alternates between red and black
D. It has no color

2. What is the worst-case time complexity for searching a key in a Red-Black Tree?

A. O(1)
B. O(log n)
C. O(n)
D. O(n log n)

3. Which self-balancing tree structure is a common alternative to Red-Black Trees?

A. AVL Trees
B. B-Trees
C. Quad Trees
D. Binary Heaps

TERMINAL QUESTIONS

1. What is the color of root in **red** **black** tree?
2. What is the color property of **red** **black** tree?
3. What is the color of null node?

REFERENCES FOR FURTHER LEARNING OF THE SESSION

TEXTBOOKS:

1)Mark Allen Weiss, Data Structures and Algorithm Analysis in C, 2010 , Second Edition, Pearson Education.
2)Ellis Horowitz, Fundamentals of Data Structures in C: Second Edition, 2015

REFERENCE BOOKS:

•A.V.Aho, J. E. Hopcroft, and J. D. Ullman, “Data Structures And Algorithms”, Pearson Education, First Edition Reprint2003.
•Horowitz, Sahni, Anderson Freed, “Fundamentals of data structures in C” , Second Edition-2007.
•R. F. Gilberg, B. A. Forouzan, “Data Structures”, Second Edition, Thomson India Edition, 2005
•Robert Kruse, C.L. Tondo, Bruce Leung, Shashi Mogalla, “Data Structures & Program Design in C”, FourthEdition-2007.

WEB REFERNCES/MOOCS:

- https://nptel.ac.in/courses/106102064
- https://nptel.ac.in/courses/106101060/4
- https://www.edx.org/course/algorithms-and-data-structures-1
- https://in.udacity.com/course/intro-to-algorithms--cs215
- https://www.coursera.org/learn/data-structures?action=enroll

THANK YOU