Red-Black, Splay and Huffman Trees

Kuan-Yu Chen (陳冠宇)

2018/10/22 @ TR-212, NTUST

Review

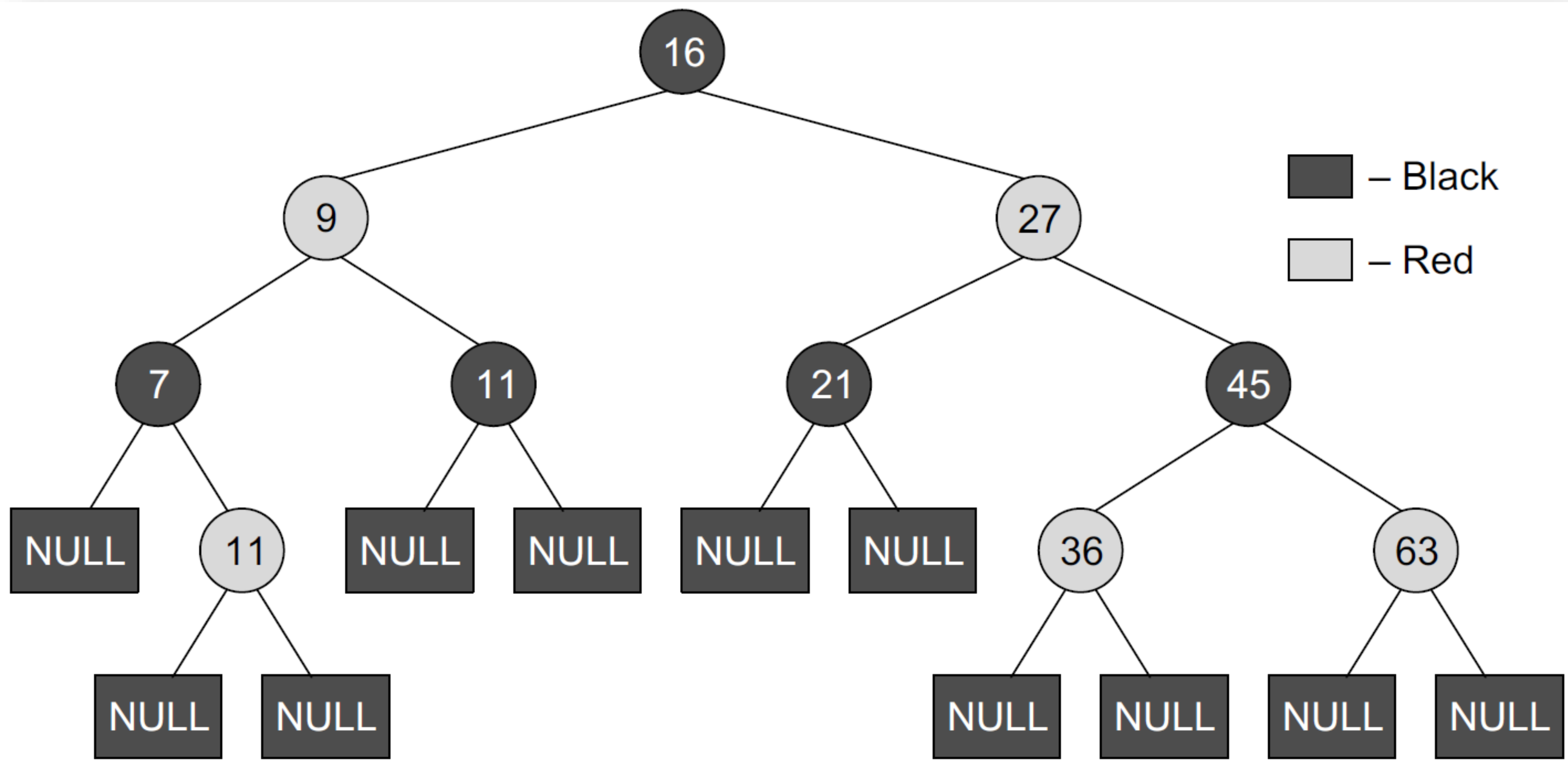
- AVL Trees
  - Self-balancing binary search tree
  - Balance Factor
    - Every node has a balance factor of −1, 0, or 1

Red-Black Trees.

- A red-black tree is a self-balancing binary search tree that was invented in 1972 by Rudolf Bayer
  - A special point to note about the red-black tree is that in this tree, no data is stored in the leaf nodes

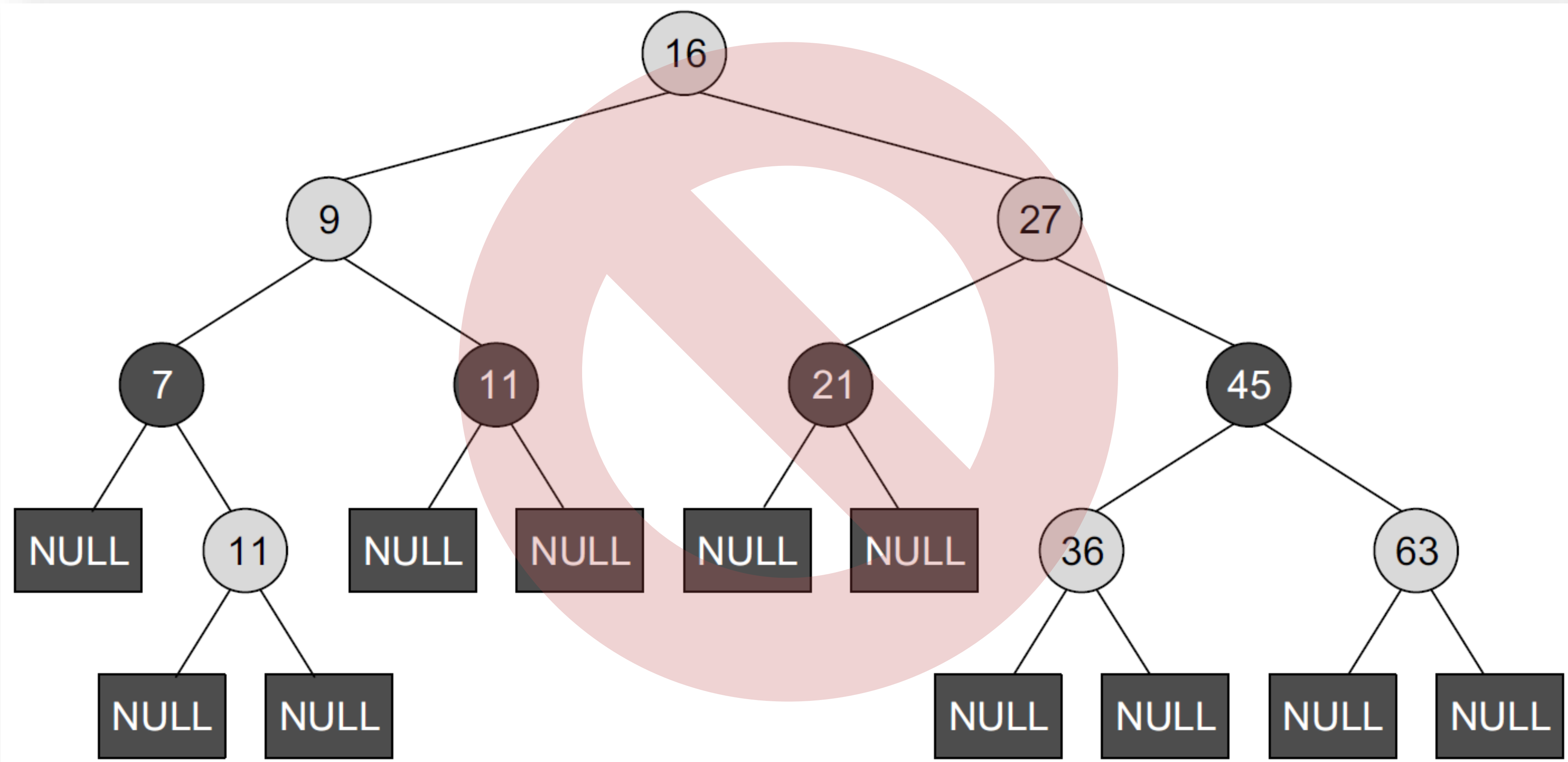
- A red-black tree is a binary search tree in which every node has a color which is either **red** or **black**
  1. The color of a node is either red or black
  2. The color of the root node is always black
  3. All leaf nodes are black
  4. Every red node has both the children colored in black
  5. Every simple path from a given node to any of its leaf nodes has an equal number of black nodes

Red-Black Trees..



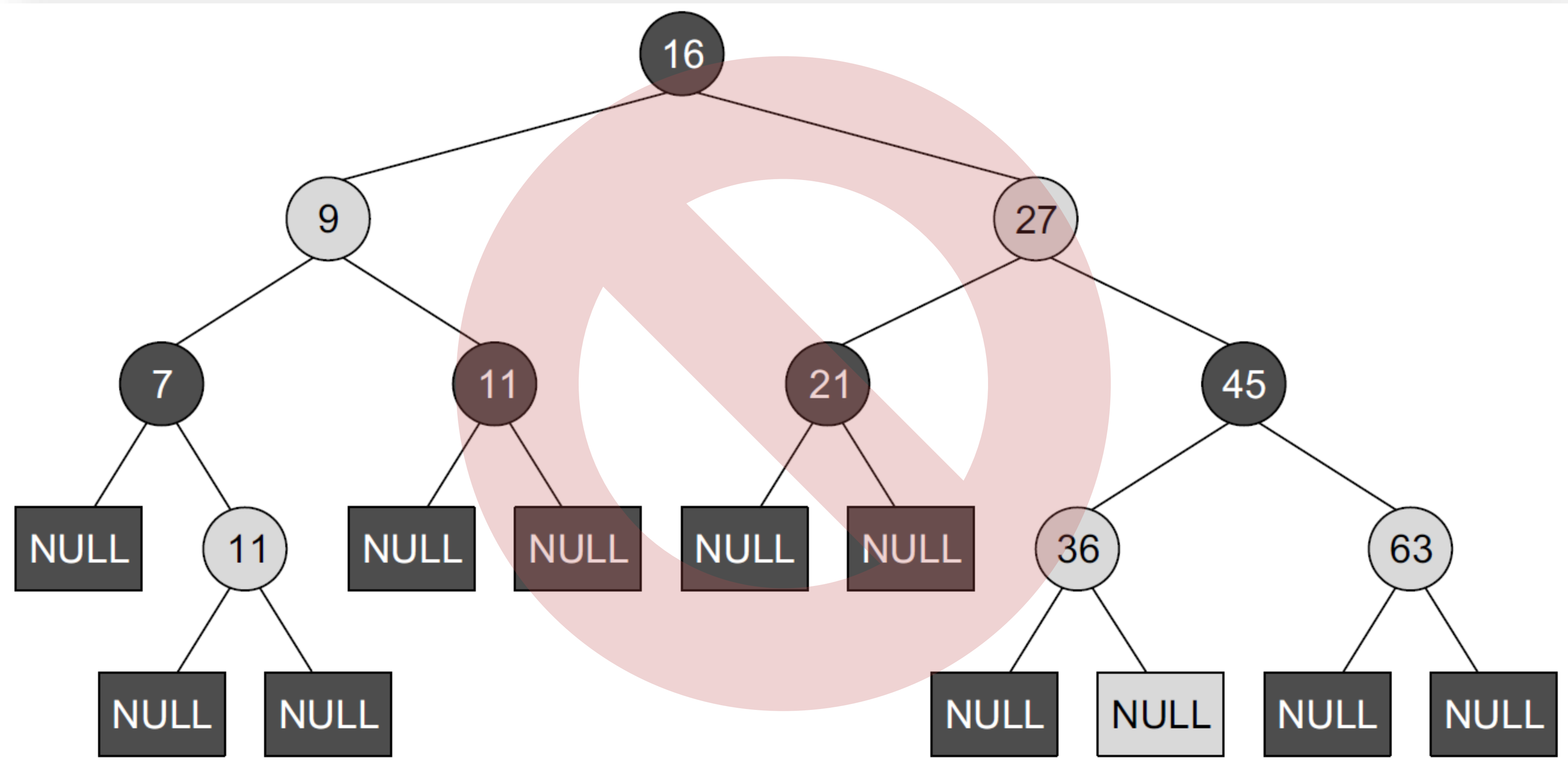
Red-Black Trees...

- Root is red



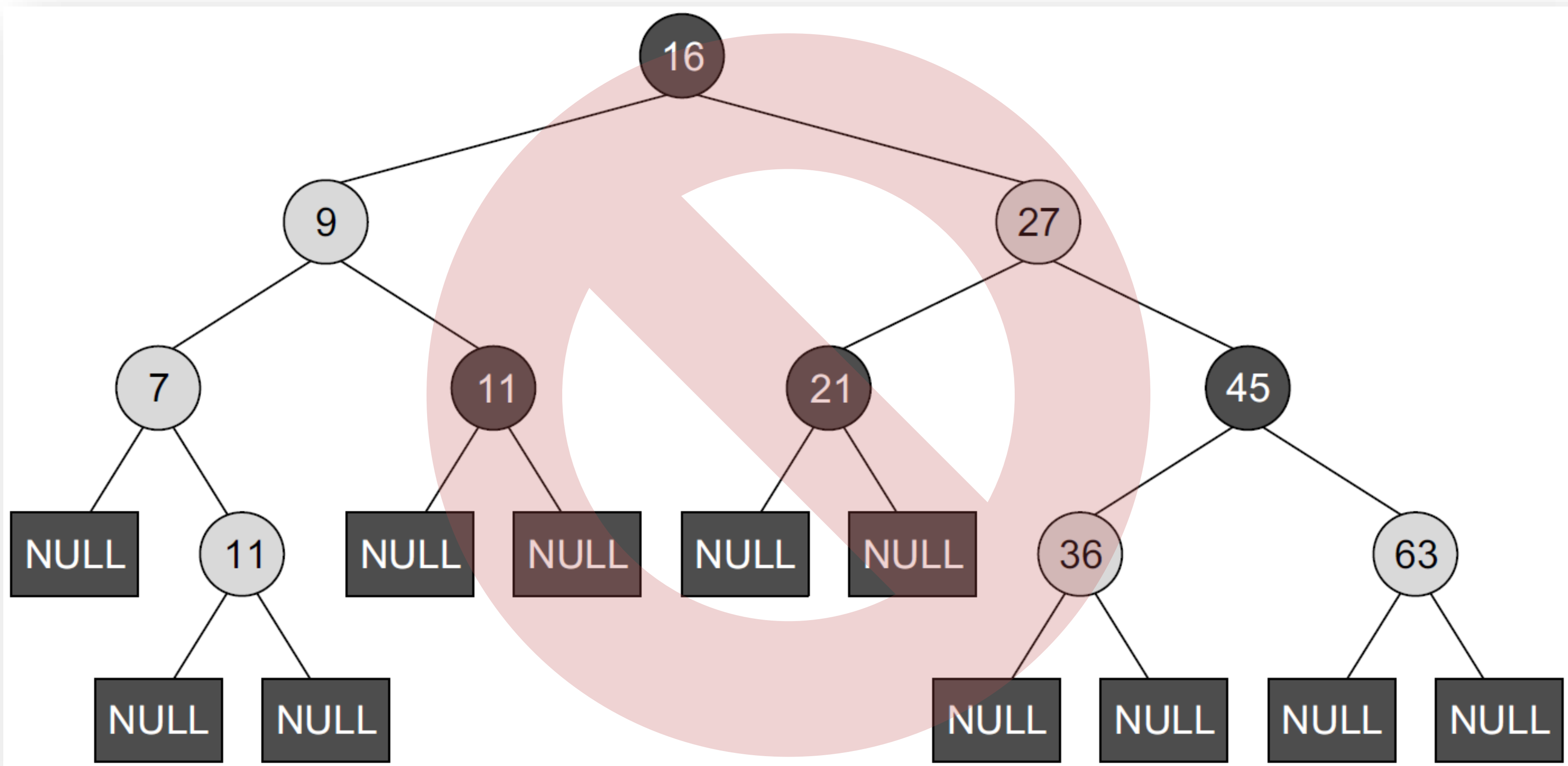
Red-Black Trees....

- A leaf node is red



Red-Black Trees…..

- Every red node does not have both the children colored in black
- Every simple path from a given node to any of its leaf nodes does not have equal number of black nodes



Searching in a Red-Black Tree

- Since red-black tree is a binary search tree, it can be searched using exactly the **same algorithm** as used to search an ordinary binary search tree!

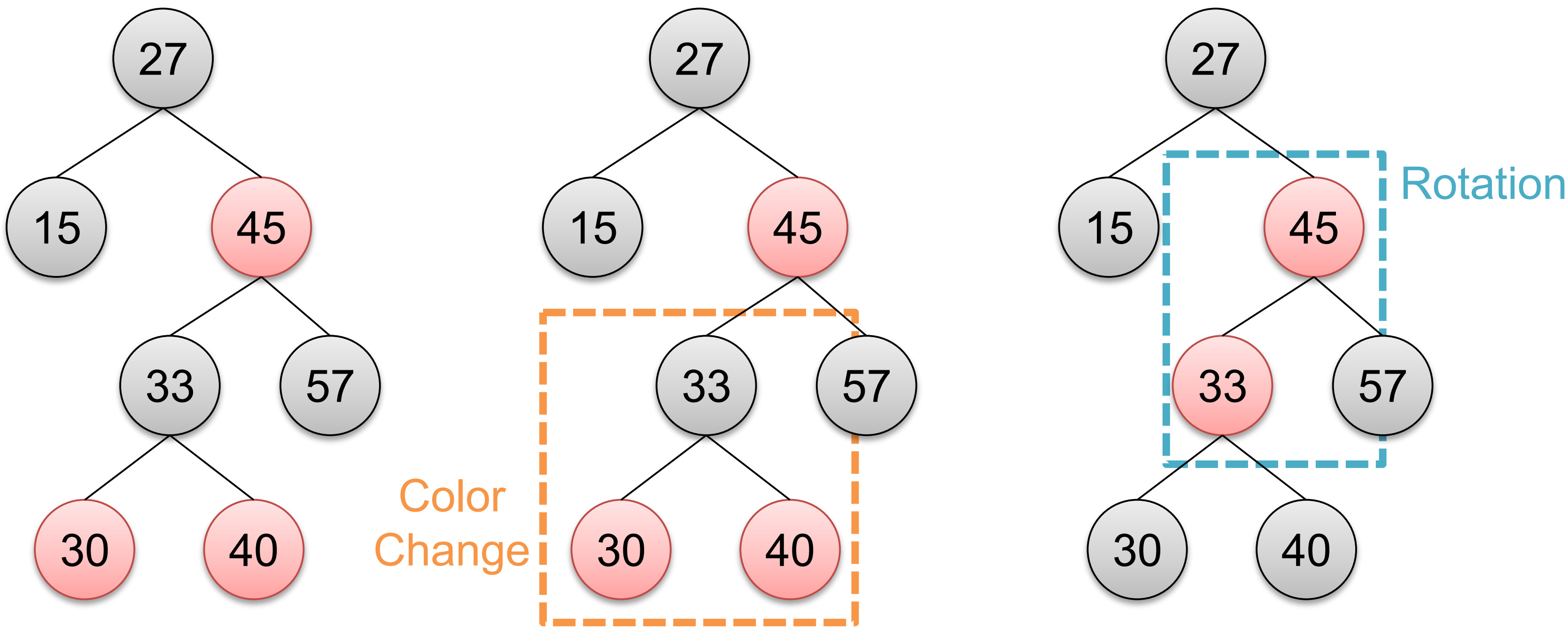
Insertion in a Red-Black Tree

- In a binary search tree, we always add the new node as a leaf, while in a red-black tree, leaf nodes contain no data
  - For a given data
    1. Searching the correct position for the data
    2. In the searching process, if there is a node with two red children
       a) Perform color change algorithm
       b) Check whether there are two consequent red nodes in the path
          ① If yes, do rotation!
    3. Insert the data and set to a red node
    4. Check whether there are two consequent red nodes in the path
       a) If yes, do rotation!
    5. Root should be black

Color Change → Rotation → Insert → Rotation → Check Root

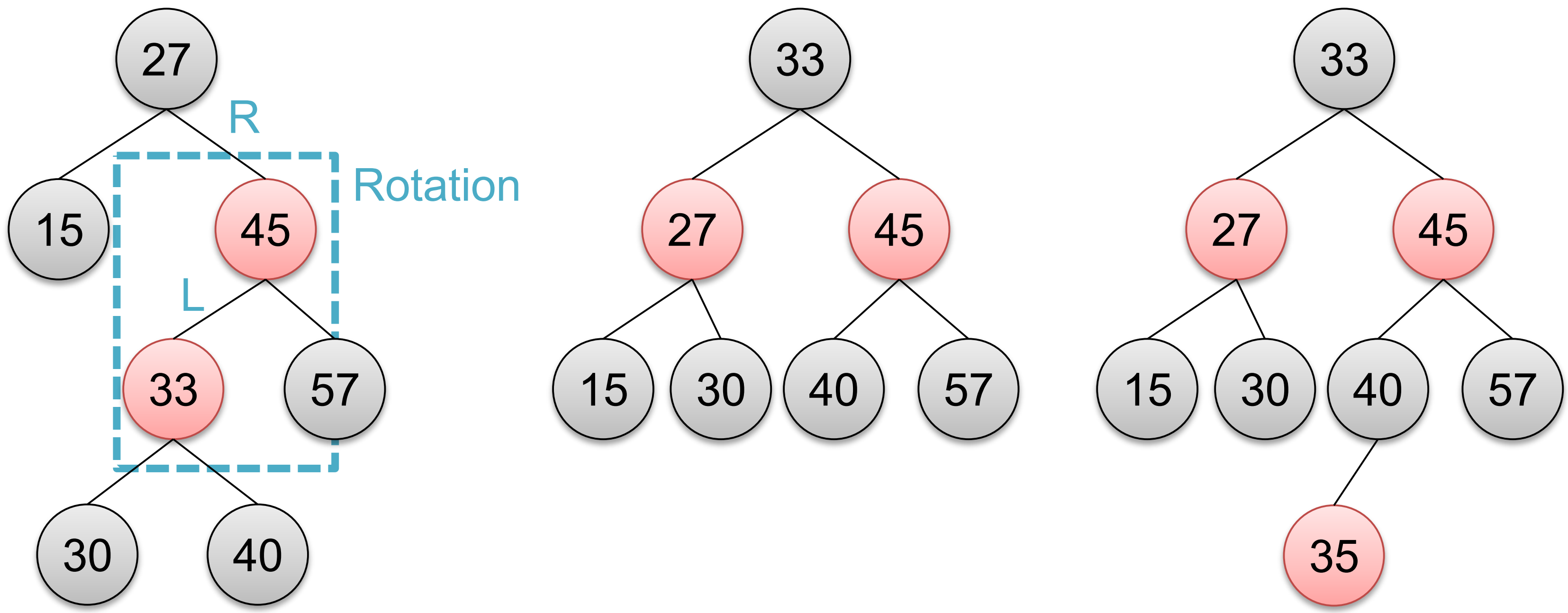
Examples – 1.

- For a given red-black tree, please insert element 35



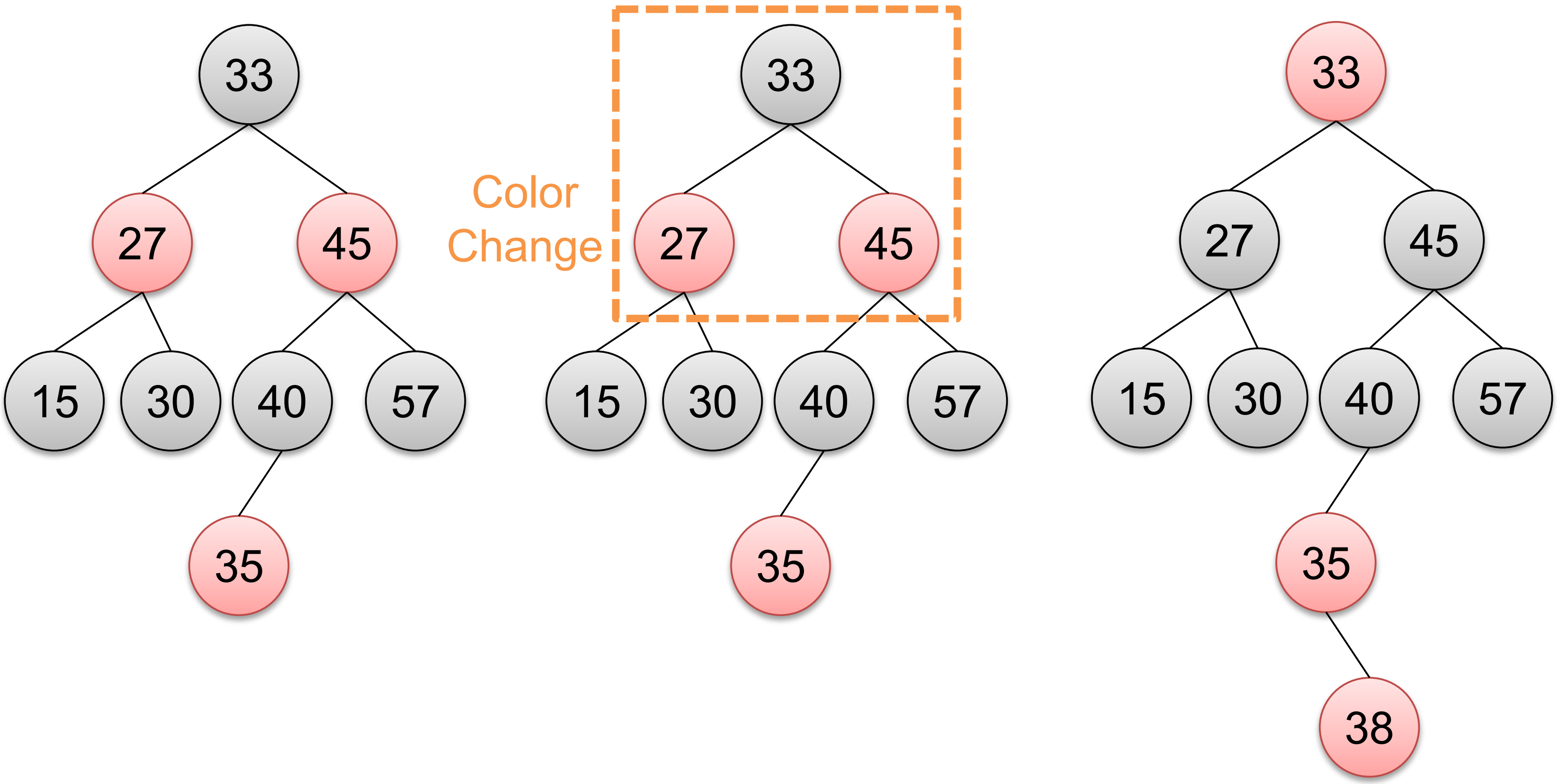
Examples – 1..

- For a given red-black tree, please insert element 35



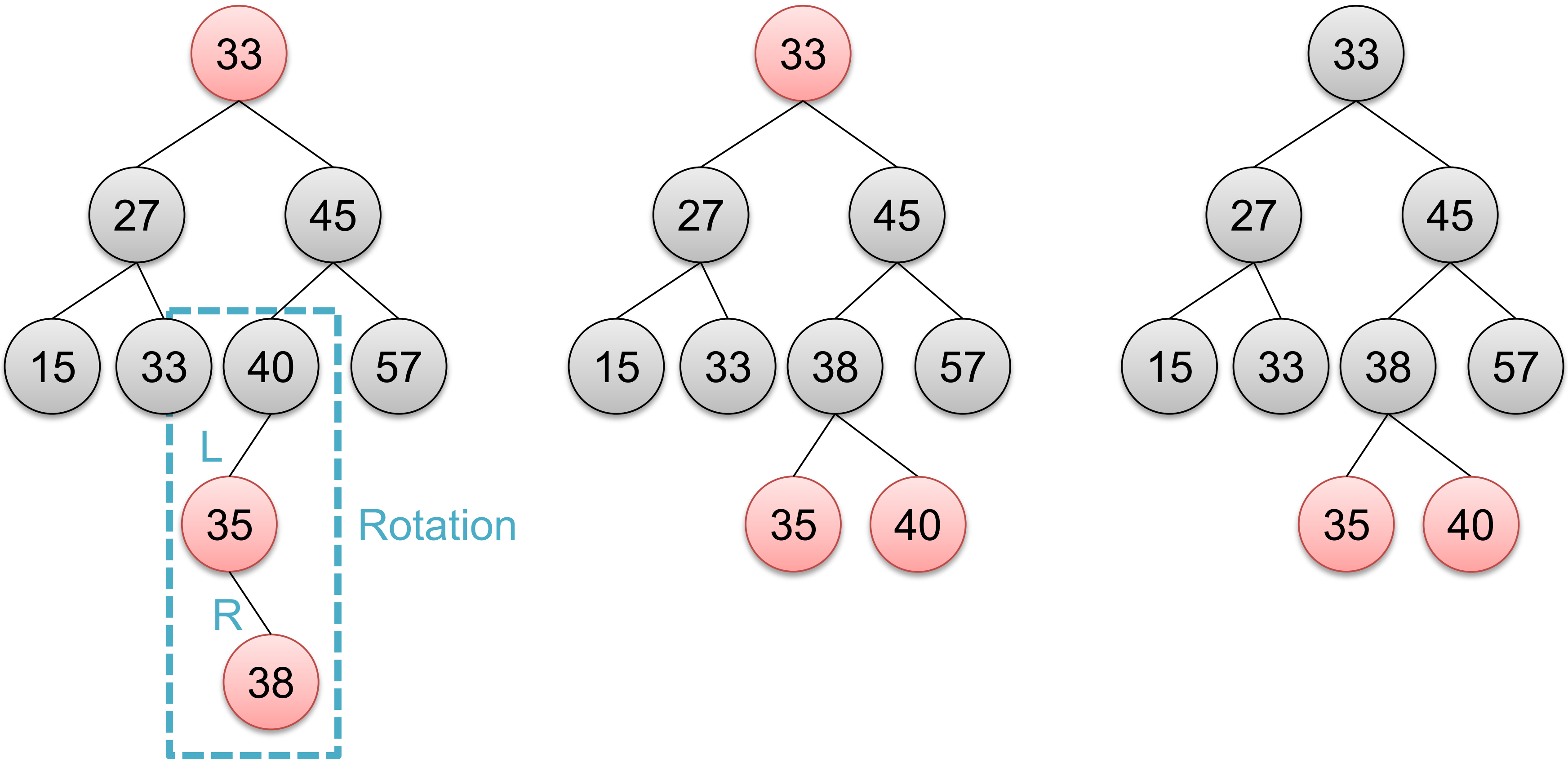
Examples – 2.

- For a given red-black tree, please insert element 38



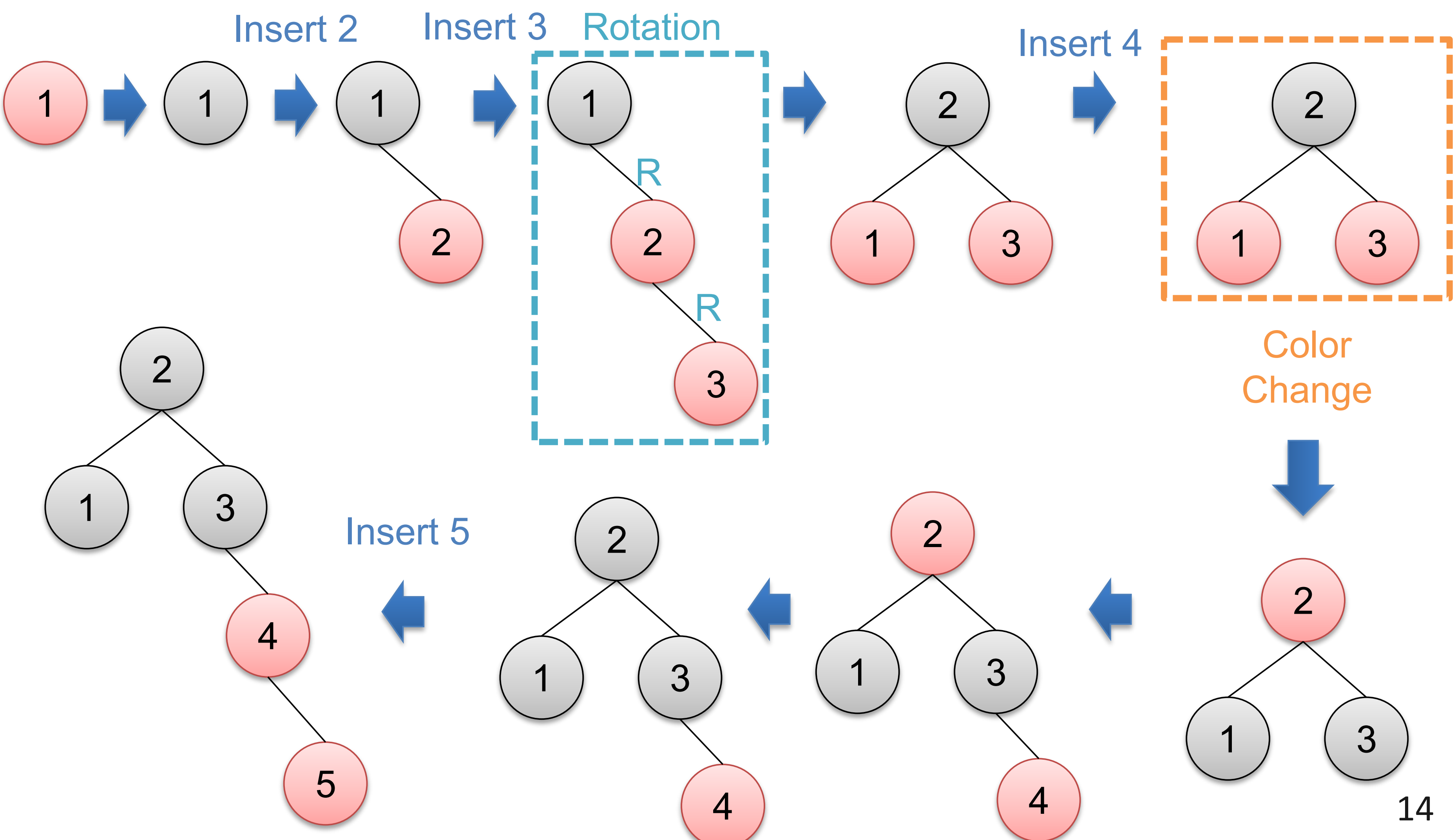
Examples – 2..

- For a given red-black tree, please insert element 38



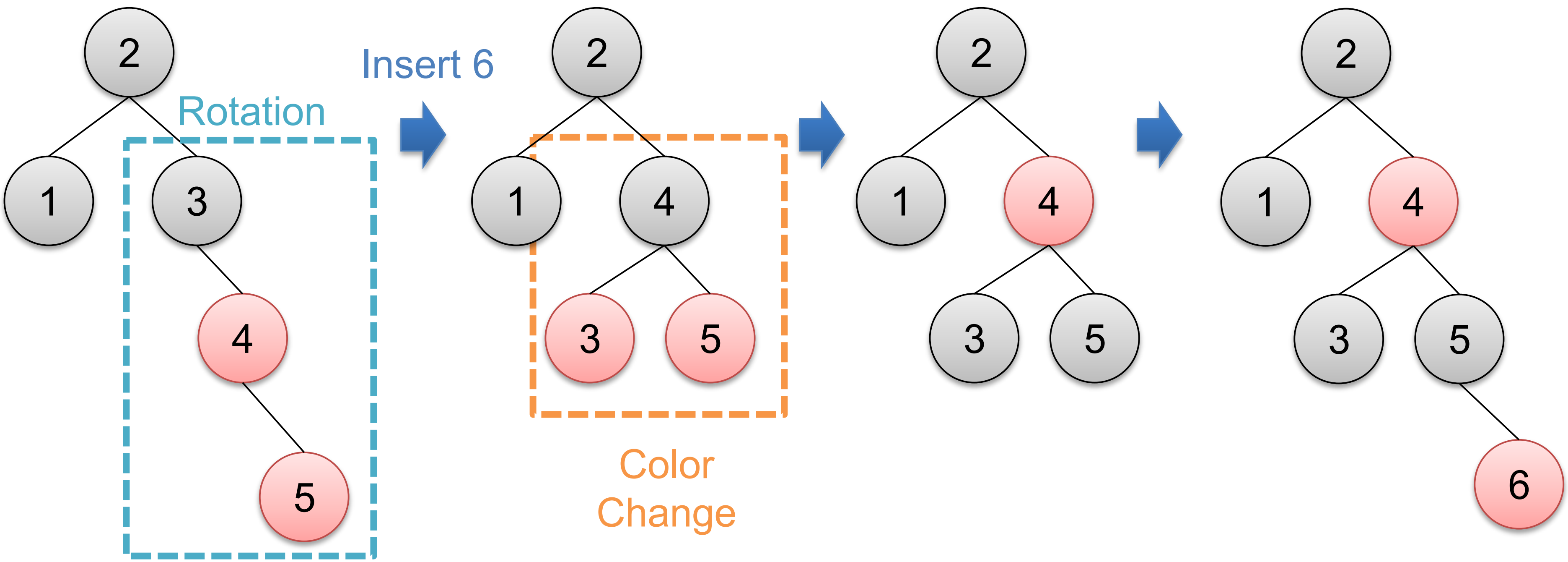
Examples – 3.

- Given 1, 2, 3, 4, 5 and 6, please construct a red-black tree



Examples – 3..

- Given 1, 2, 3, 4, 5 and 6, please construct a red-black tree



Compared with AVL Trees

- Red-black trees are efficient binary search trees, as they offer worst case time guarantee O(log n) for insertion, deletion, and search operations
  - It is roughly a balanced binary search tree

- AVL trees also support O(log n) search, insertion, and deletion operations, but they are more rigidly balanced than red-black trees
  - Thereby, AVL trees are slower insertion and removal but faster retrieval of data

SPLAY Trees

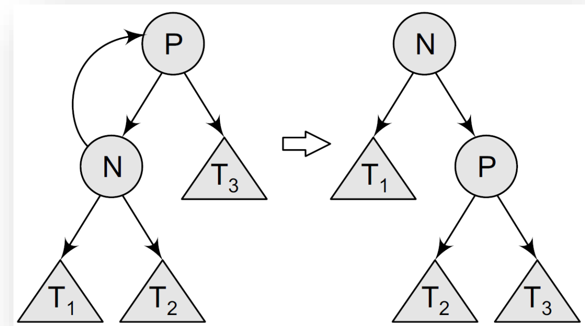
- Splay trees were invented by Daniel Sleator and Robert Tarjan

- A splay tree is a **self-balancing binary search tree** with an additional property that **recently accessed elements can be re-accessed fast**
  - A simple idea behind it is that if an element is accessed, it is likely that it will be accessed again

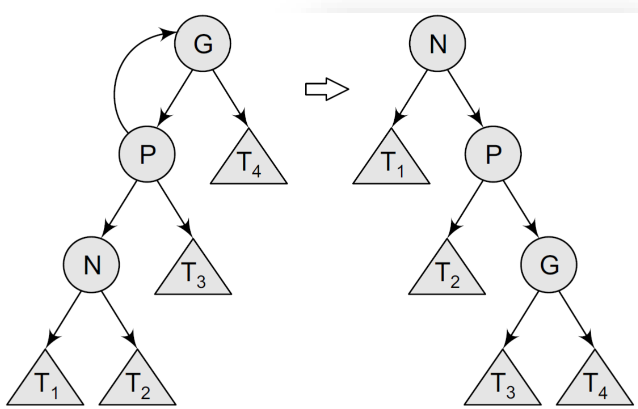
- For many non-uniform sequences of operations, splay trees perform better than other search trees

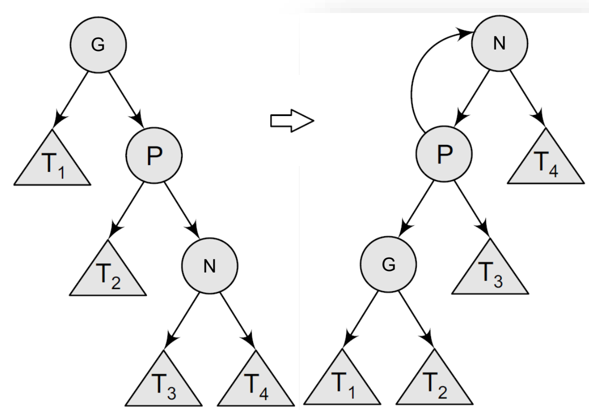
SPLAY Trees – Splaying.

- When we access a node *N*, splaying is performed on the node to move it to the root
  - Zig Step
    - The zig operation is done when *P* (the parent of *N*) is the root of the splay tree



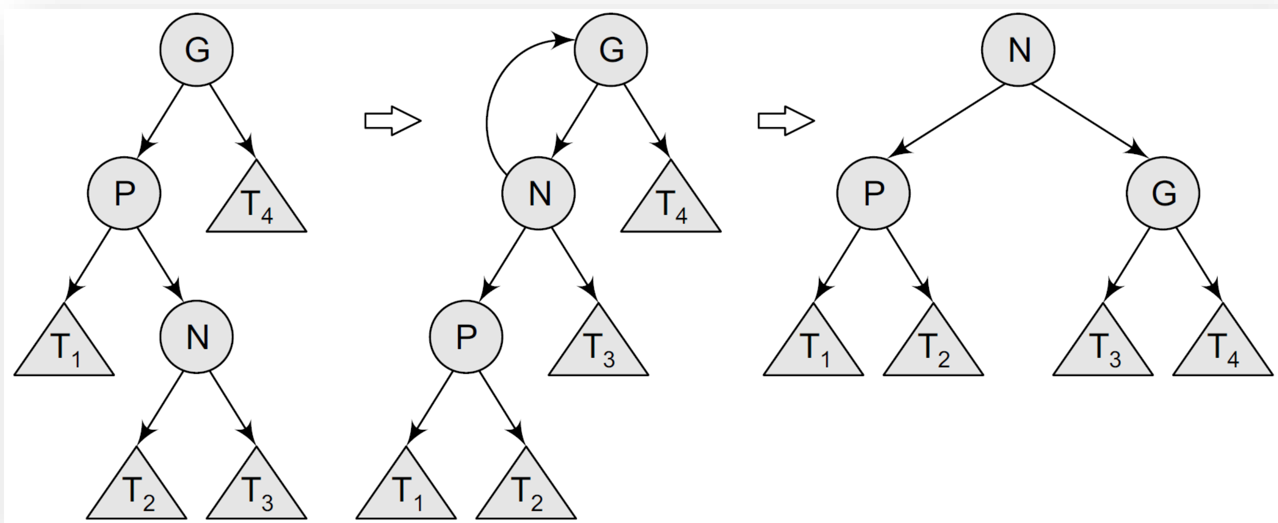
  - Zig-zig Step
    - The zig–zig operation is performed when *P* is not the root
    - Besides, *N* and *P* are either both right or left children of their parents





SPLAY Trees – Splaying..

- Zig-zag Step
  - The zig–zag operation is performed when P is not the root
  - In addition to this, N is the right child of P and P is the left child of G or vice versa

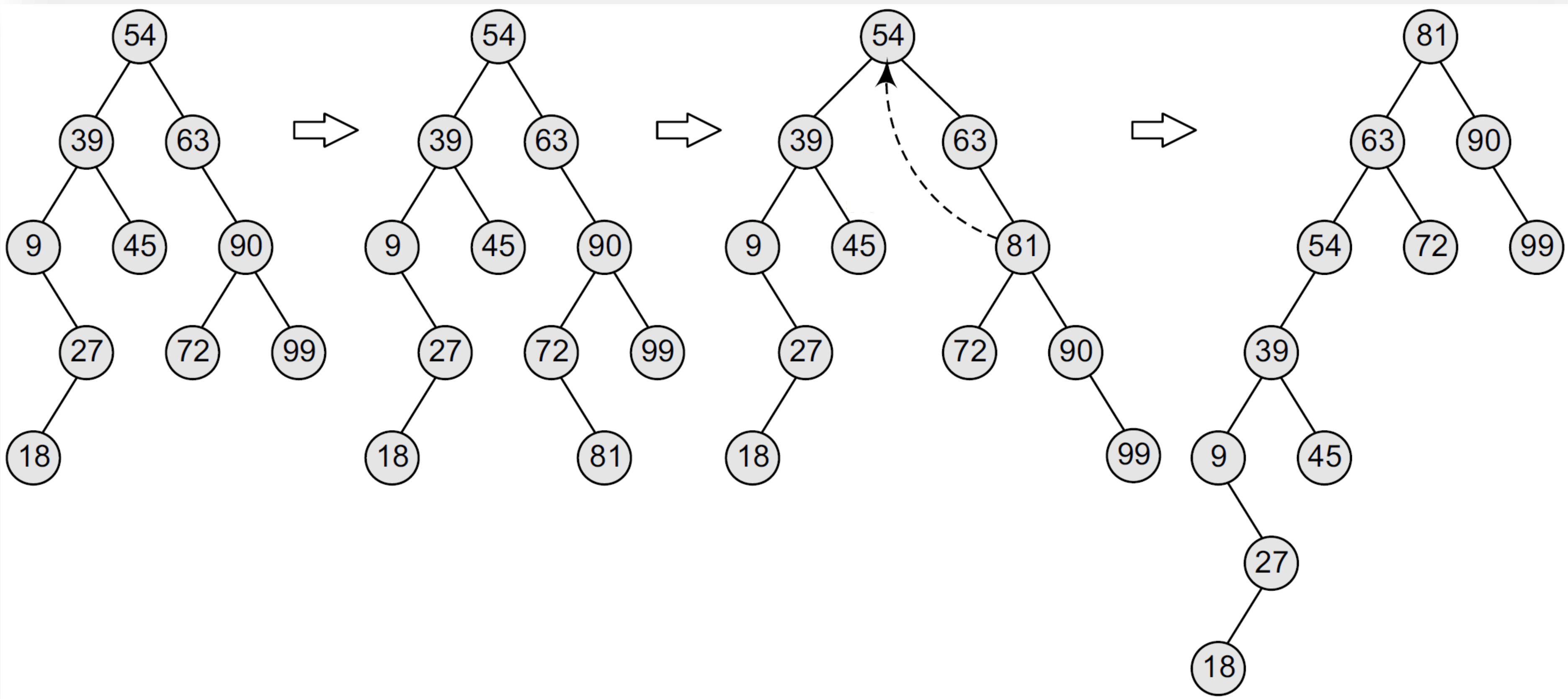


SPLAY Trees – Insertion

- The steps performed to insert a new node N in a splay tree can be given as follows
  - Search N in the splay tree
    - If the search is successful, splay at the node N
    - If the search is unsuccessful, add the new node N in such a way that it replaces the NULL pointer reached during the search by a pointer to a new node N
    - Splay the tree at N

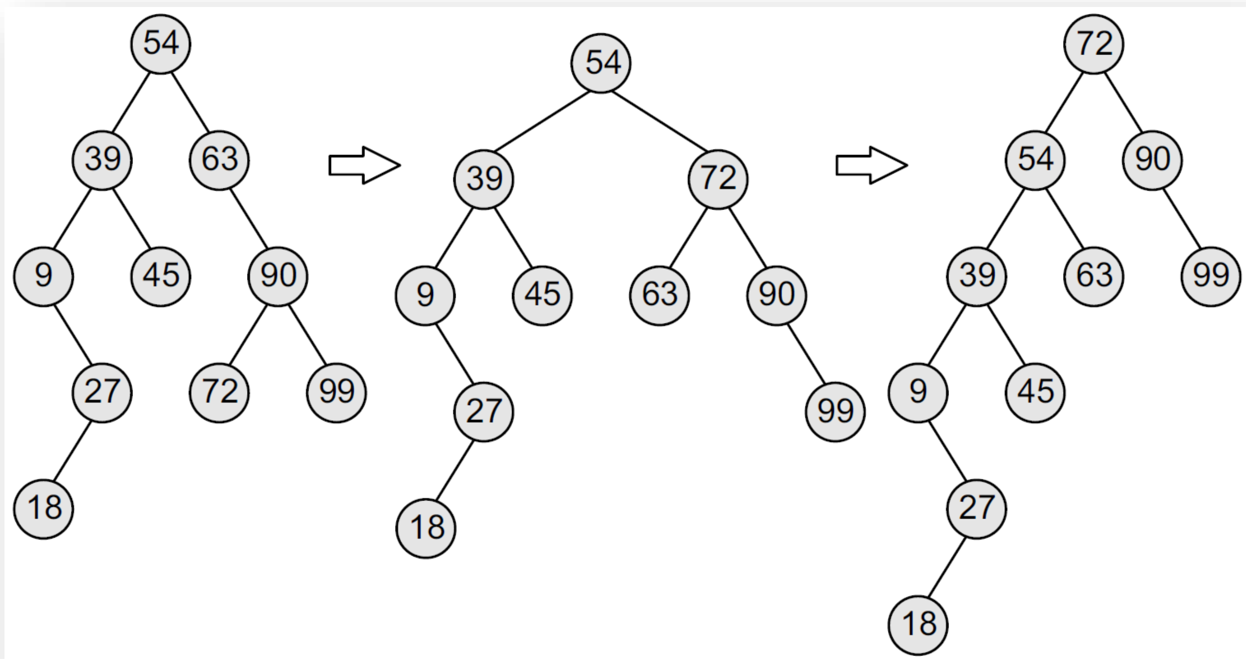
Example

- Insert 81 into a given Splay tree



SPLAY Trees – Search

- If a particular node N is present in the splay tree, then a **pointer to N is returned**; otherwise a **pointer to the null node is returned**
  - If the search is unsuccessful (the splay tree does not contain), Splay the tree at the last non-null node reached during the search
    - Searching 81



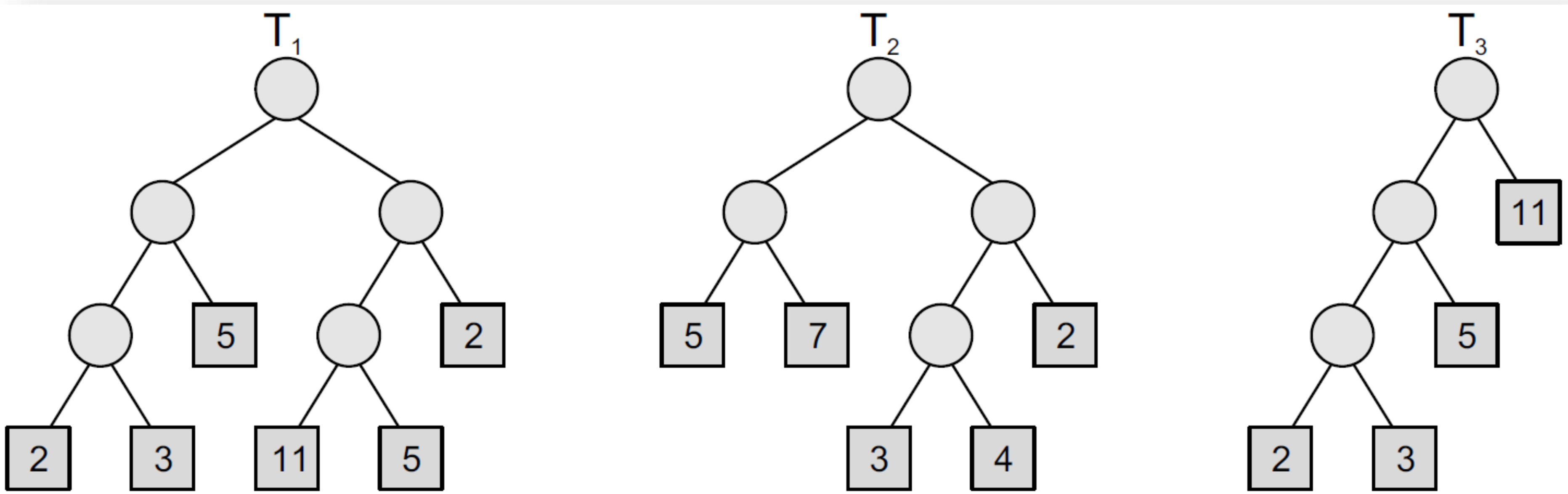
Huffman Trees

- Huffman coding is an entropy encoding algorithm developed by David A. Huffman that is widely used as a lossless data compression technique

- The key idea behind Huffman algorithm is that it encodes the most common characters using shorter strings of bits than those used for less common source characters
  - The internal node is used to link to its child nodes
  - The external node contains the actual character and weight

Weighted External Path Length

- The weighted external path length
  - For T_1
    - $2 \times 3 + 3 \times 3 + 5 \times 2 + 11 \times 3 + 5 \times 3 + 2 \times 2 = 77$
  - For T_2
    - $5 \times 2 + 7 \times 2 + 3 \times 3 + 4 \times 3 + 2 \times 2 = 49$
  - For T_3
    - $2 \times 3 + 3 \times 3 + 5 \times 2 + 11 \times 1 = 36$

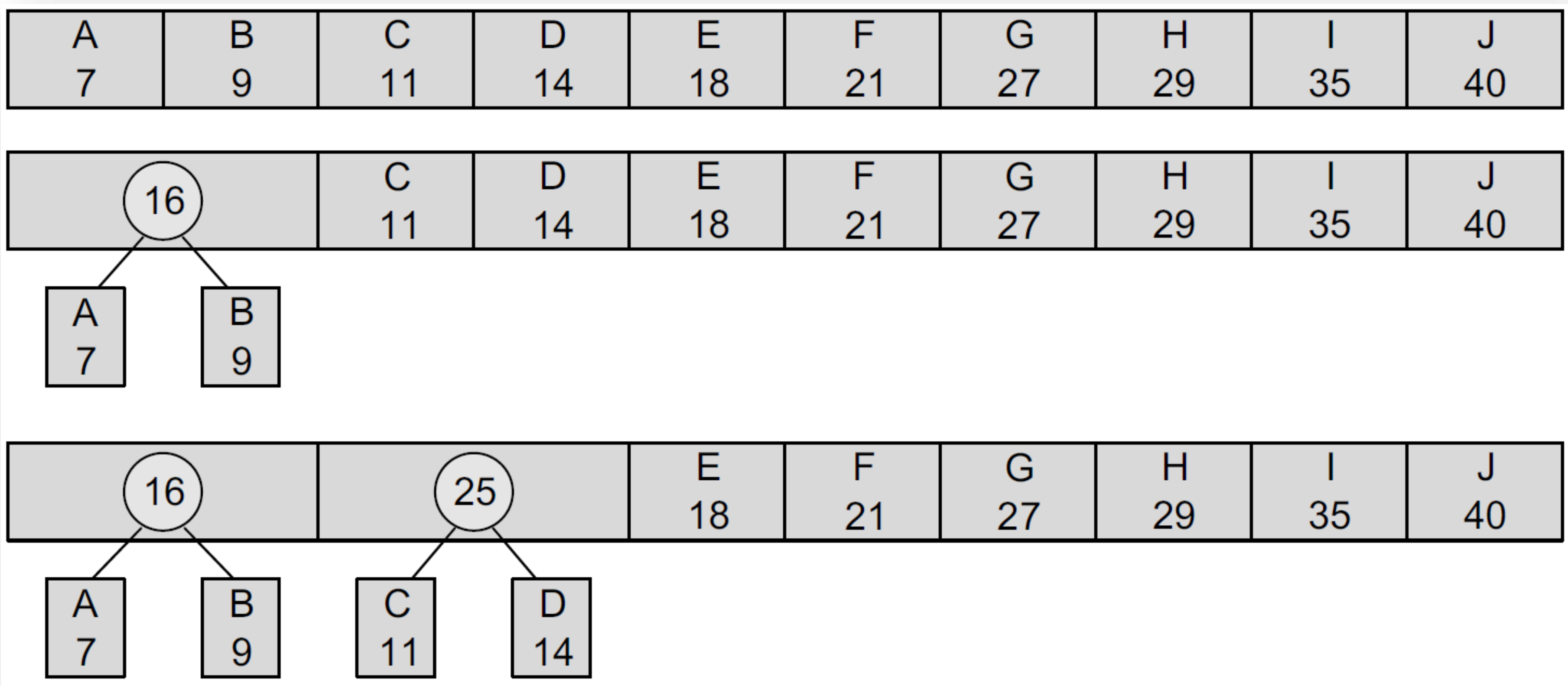


Creating a Huffman Tree

- Given n nodes and their weights, the Huffman algorithm is used to find a tree with a **minimum** weighted path length
  - Creating a new node whose children are the two nodes with the smallest weight
  - The two nodes are merged into one node
  - Repeat the process until the tree has only one node

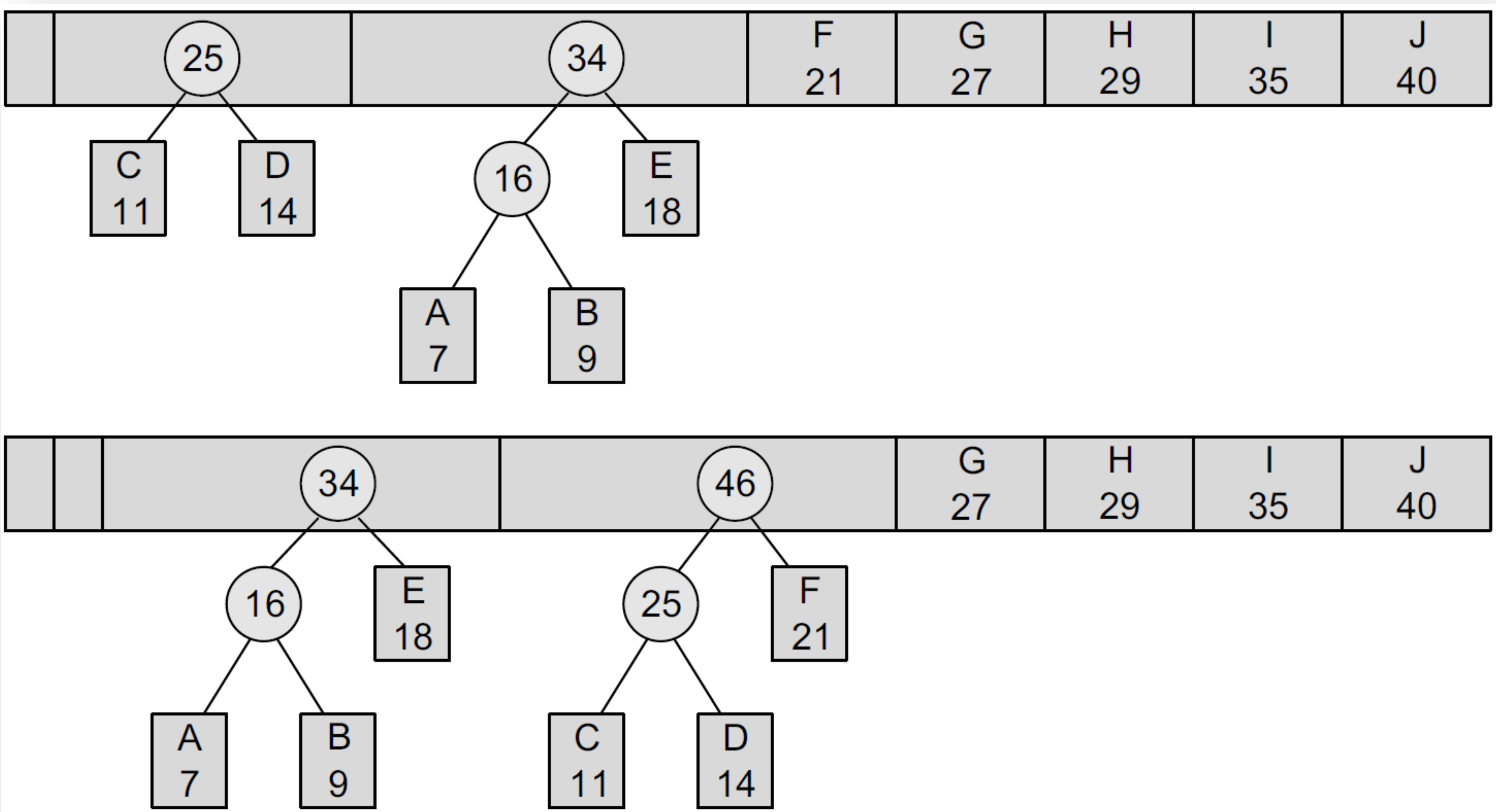
Example.

- Create a Huffman tree with the following sorted nodes



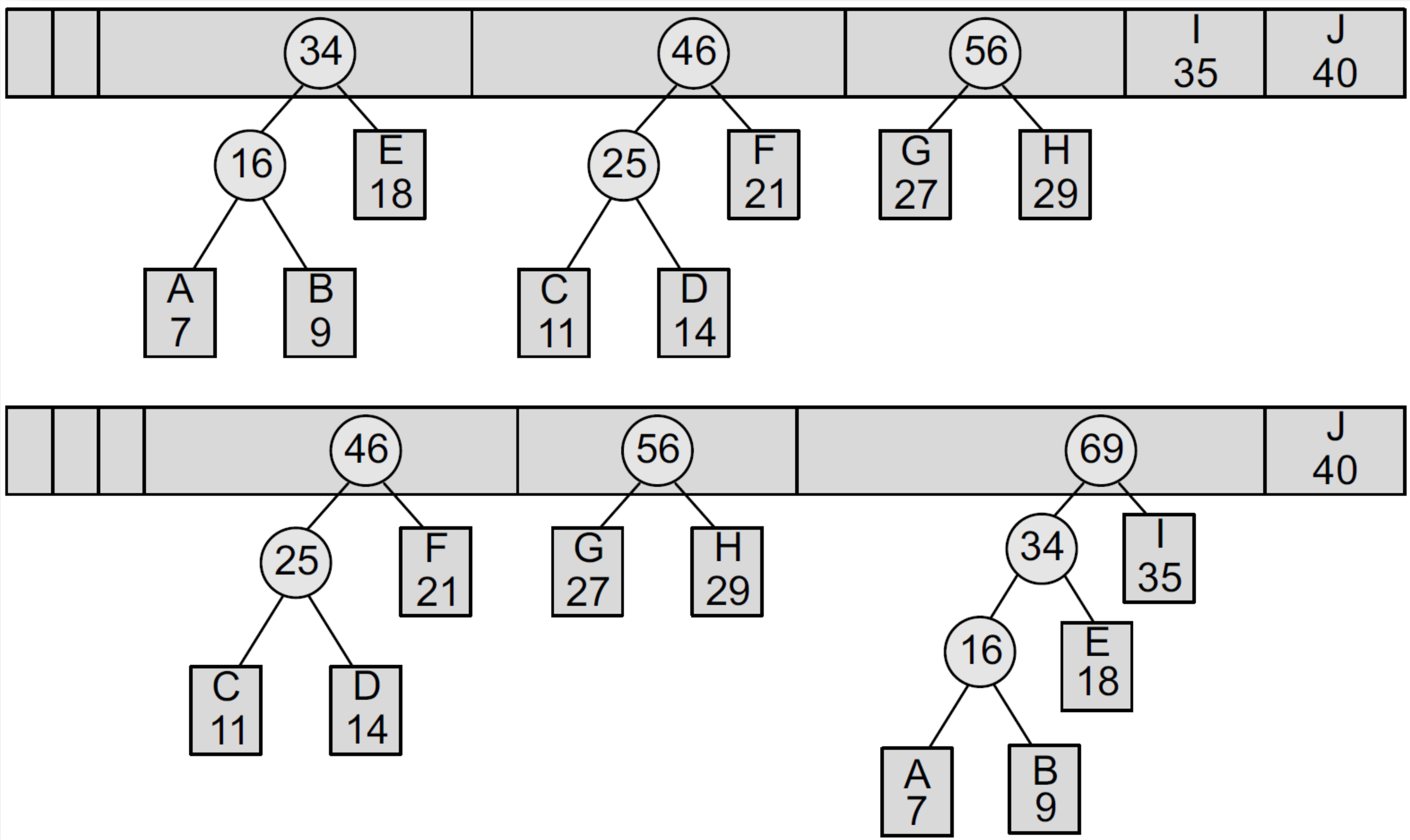
Example..

- Create a Huffman tree with the following sorted nodes



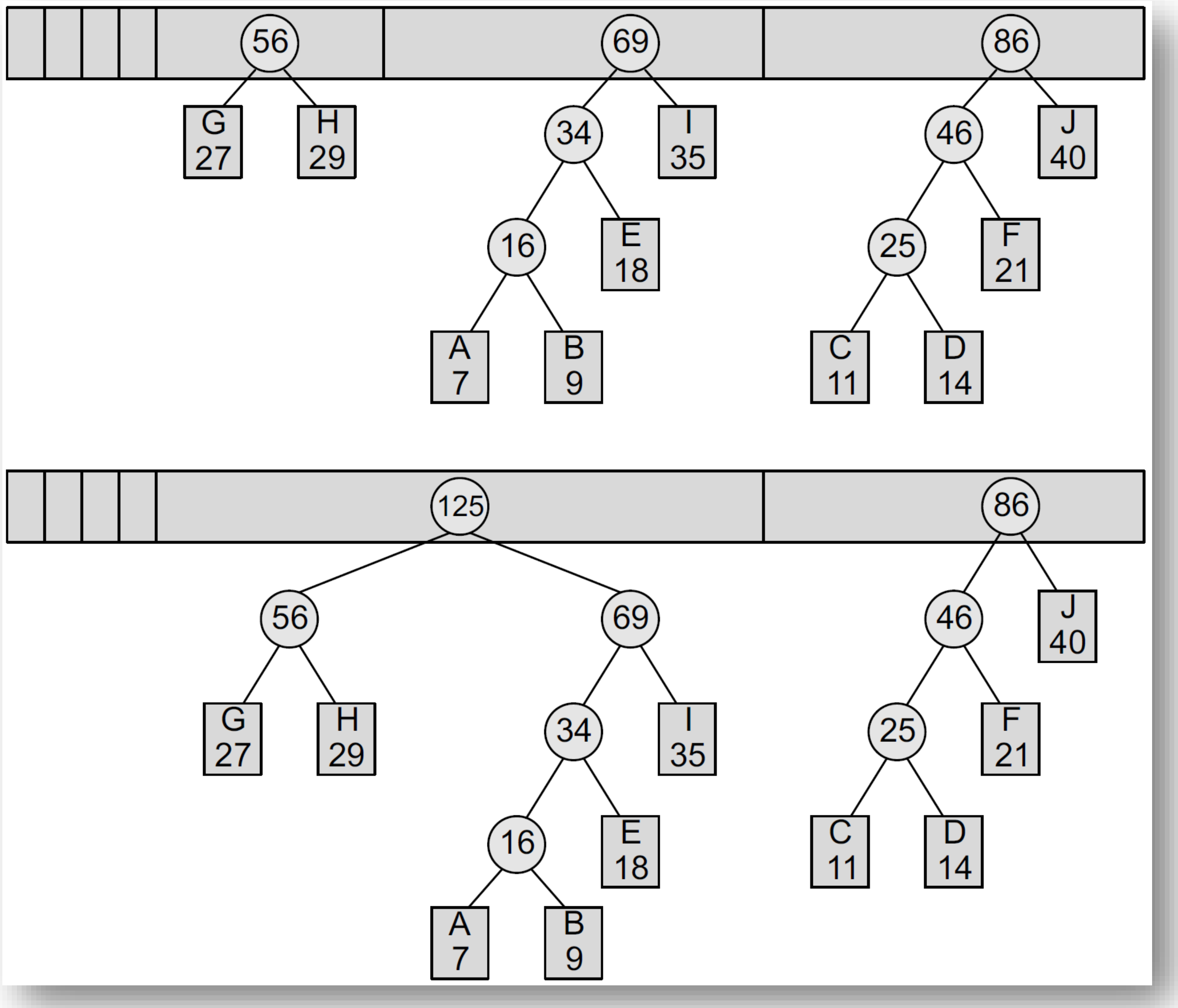
Example…

- Create a Huffman tree with the following sorted nodes



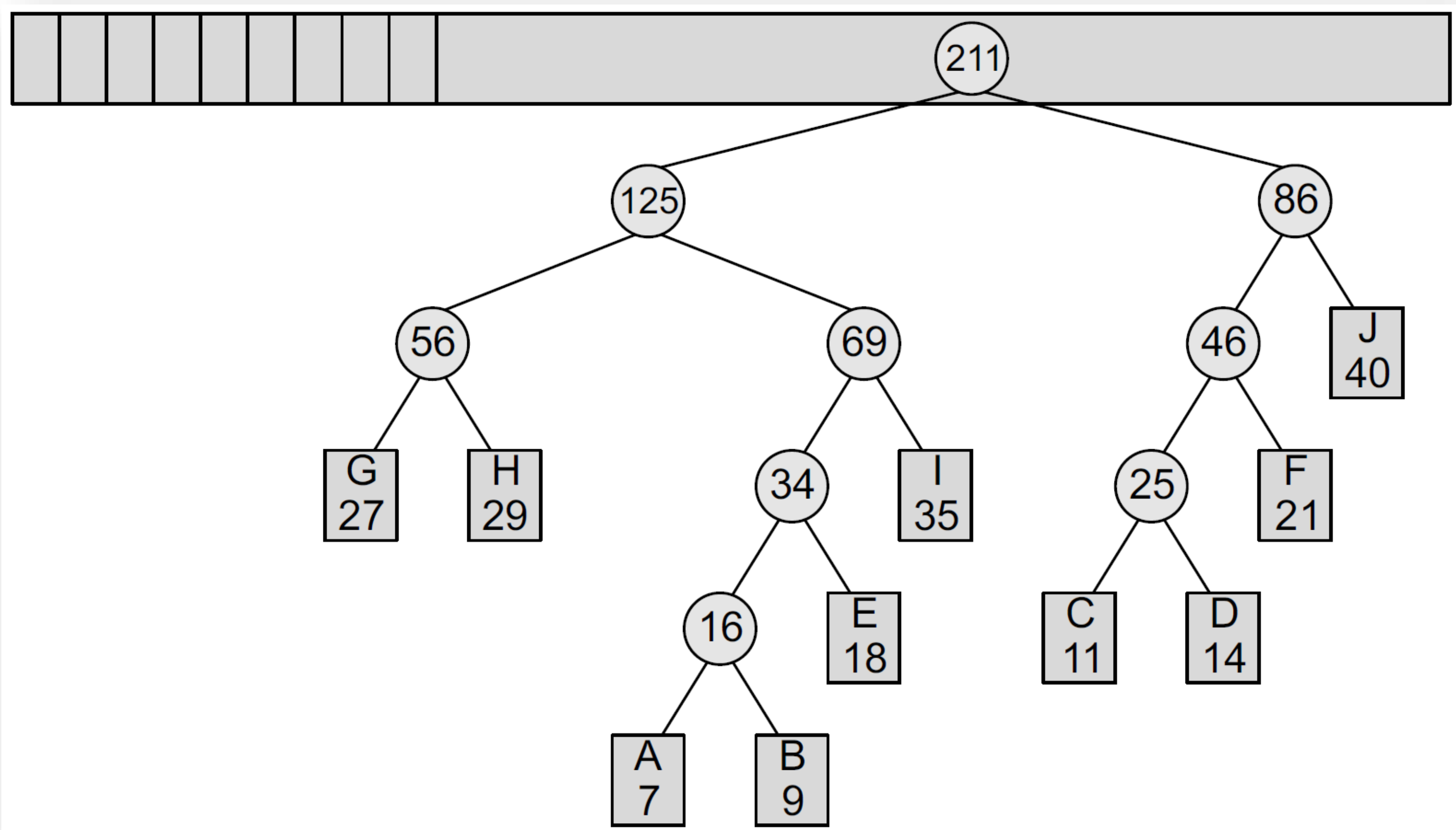
Example....

- Create a Huffman tree with the following sorted nodes



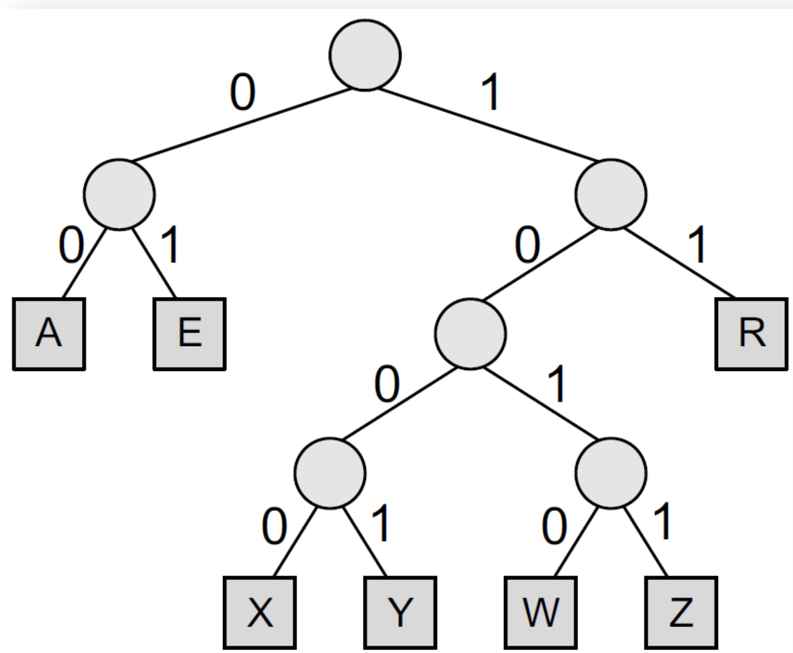
Example.....

- Create a Huffman tree with the following sorted nodes



Data Coding

- For a Huffman tree, every left branch is coded with 0 and every right branch is coded with 1
  - For a character sequence: AAERZ
    - By Huffman Coding Scheme: 000001111011
    - By Original Coding Scheme: 000000001010110



Character	Code	Original Coding
A	00	000
E	01	001
R	11	010
W	1010	011
X	1000	100
Y	1001	101
Z	1011	110

Questions?

kychen@mail.ntust.edu.tw