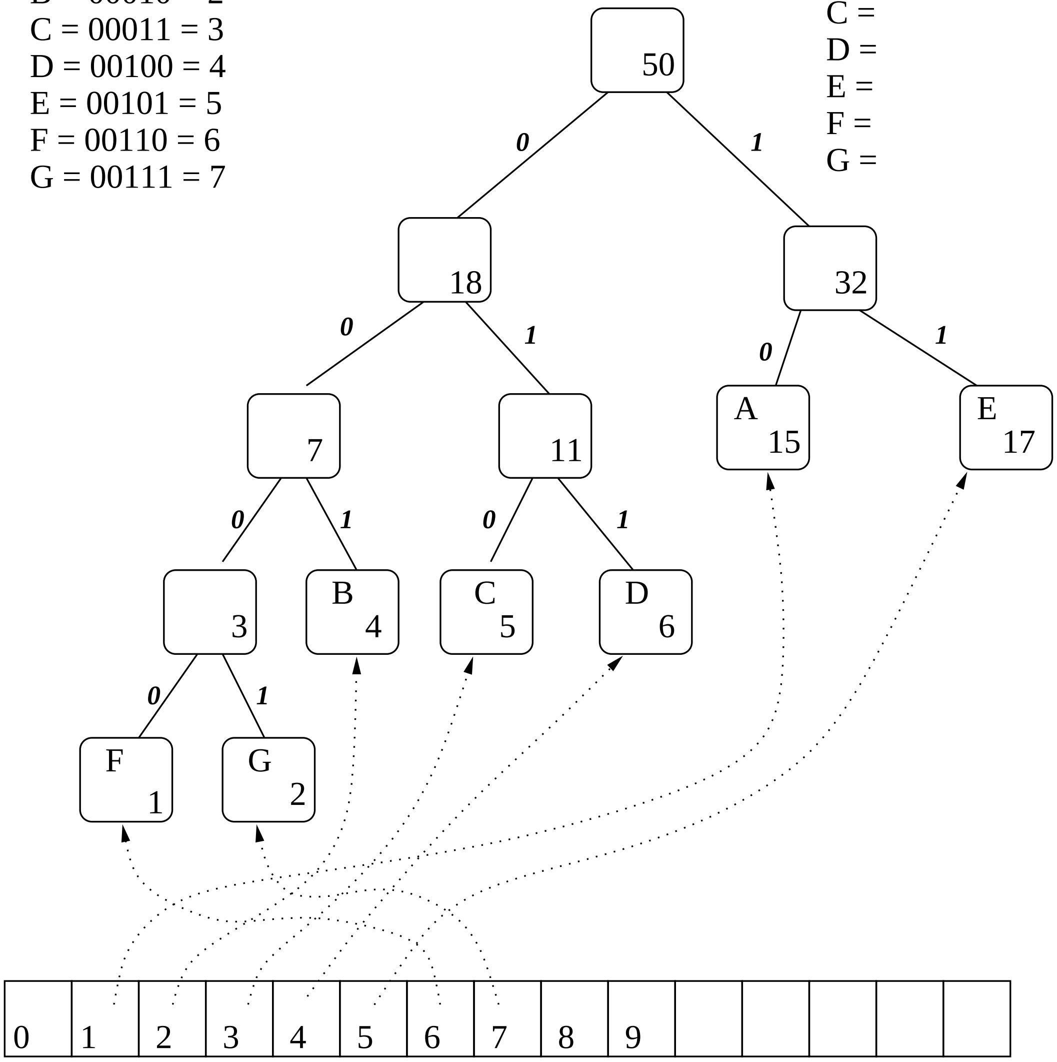
1 Trees and Huffman Coding Continued

ORIGINAL ENCODING

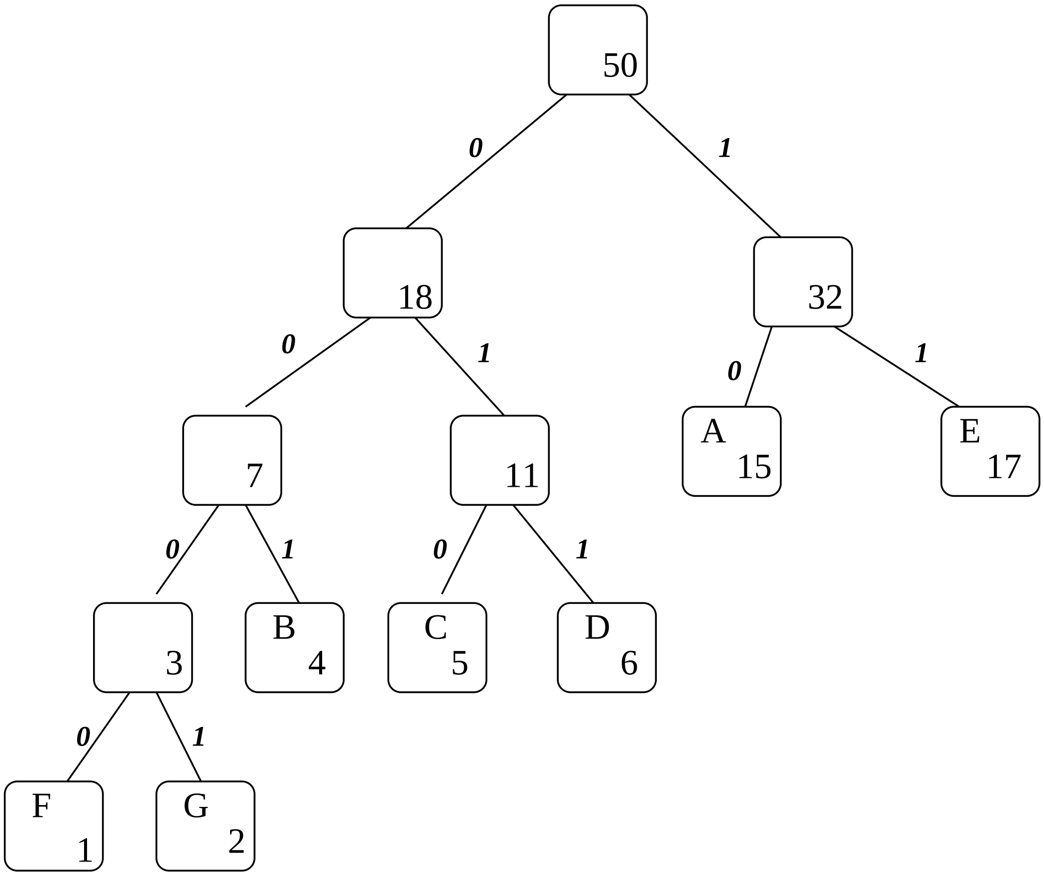
A = 00001 = 1
B = 00010 = 2
C = 00011 = 3
D = 00100 = 4
E = 00101 = 5
F = 00110 = 6
G = 00111 = 7

COMPRESSED ENCODING

A =
B =
C =
D =
E =
F =
G =



0111001100001001111

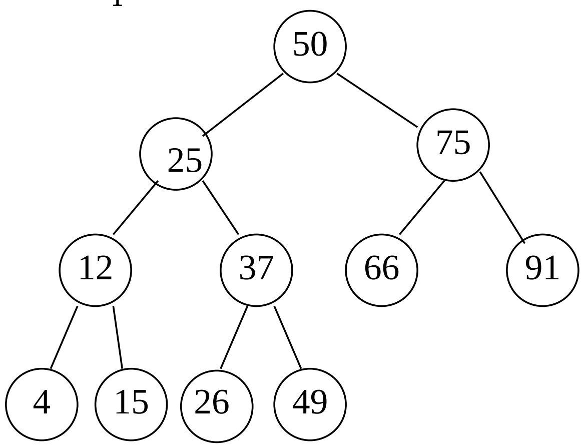


2 Binary Trees

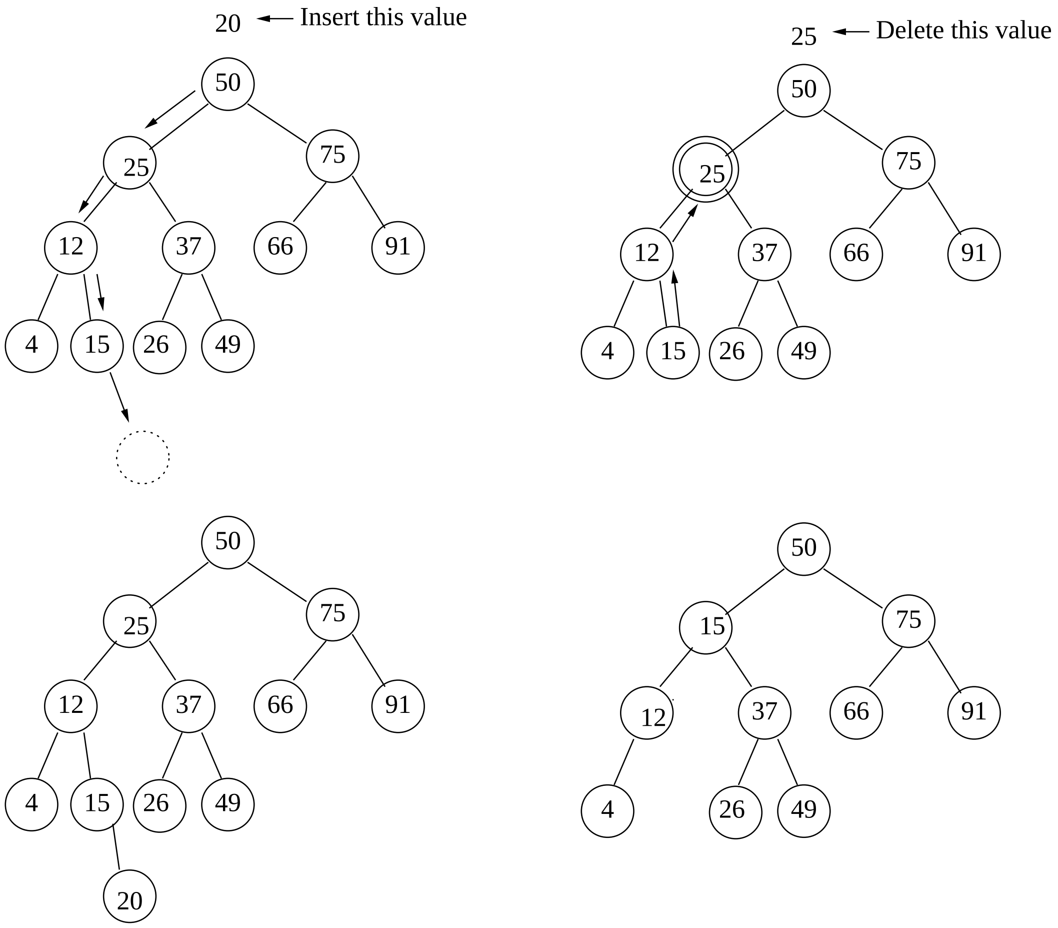
Binary search trees represent our first step toward a data structure allowing logarithmic time insertion, deletion and search.

Definition: a binary search tree is a binary tree such that at each internal node, v, the value stored at v is greater than (or equal) any value stored in v's left subtree and is less than (or equal) any value stored in v's right subtree.

Example:



Review of key operations: Search, Insert and Remove.



See the text for code. The only code from Chapter 4 that we will discuss in detail is the code for AVL trees.

3 AVL Trees

AVL stands for Adelson-Velskii and Landis.

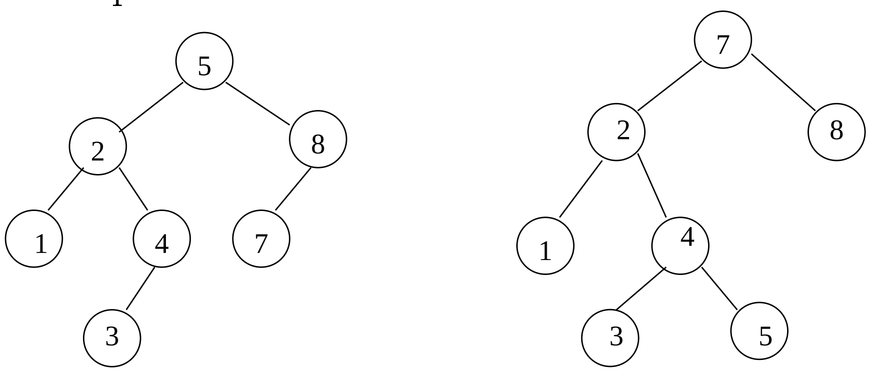
An AVL tree is a binary search tree with a balance condition.

The balance condition must:

- be easy to maintain
- ensure that the depth of the tree is $O(logN)$

AVL Balance Condition: The height of the left and right subtree can differ by at most 1.

Example:



What is the advantage the balance condition?

The condition guarantees each search, insert, and remove operation on a tree storing n values (containing n nodes), will require a worst-case of time.

Details

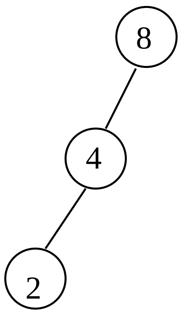
- At each node in an AVL tree the difference in height between the left subtree and the right subtree is guaranteed to be -1, 0 or 1.
- Note that leaves are considered height 0 and null subtrees (null pointers from a tree node) height -1.
- Maintaining this guarantee requires special "rotation" operations: single rotations and double rotations. These operations distinguish AVL trees from ordinary binary search trees.
- Insertion operations will require at most one rotation.
- Remove operations may require at most $O(\log n)$ rotations. However, this worst-case behavior is rare.
- The maximum height of an AVL tree containing n nodes is approximately $1.44 \times \log n$.
- A modified version of the code from the Weiss text will be is attached here andwill be discussed in detail in class. This includes code to remove nodes from the trees, which Weiss does not provide.

Maintaining AVL trees

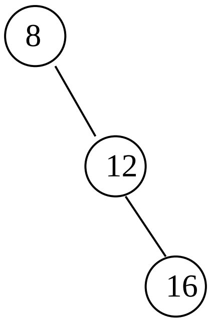
Maintaining AVL trees requires balancing the tree in the event that the height of left and right subtrees differ by more than 1.

First, lets consider the four cases where the height of the left and right subtrees differ by more than 1.

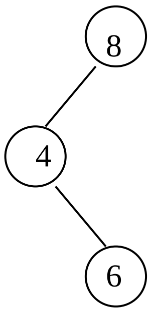
CASE 1



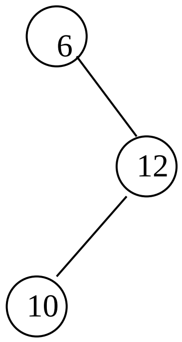
CASE 2



CASE 3



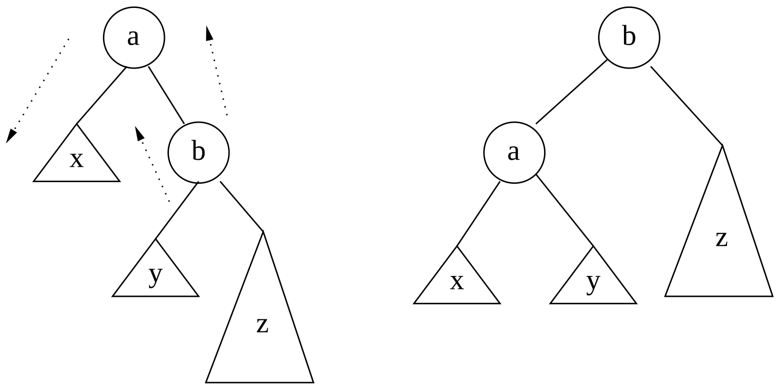
CASE 4



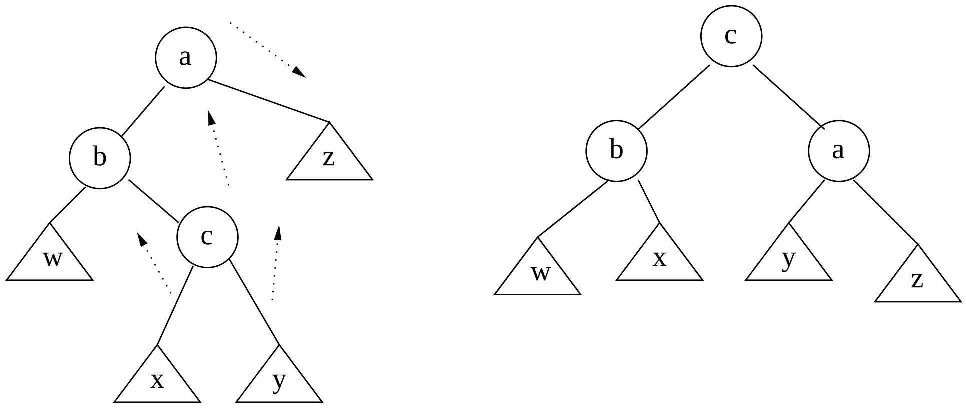
Case 1 and 2 require a single rotation to balance the tree.

Case 3 and 4 require a double rotation to balance the tree.

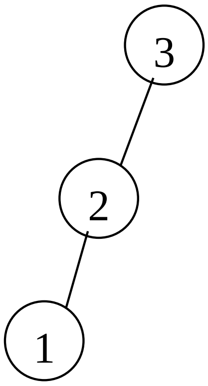
Here is what the single and double rotations look like from a conceptual point of view.



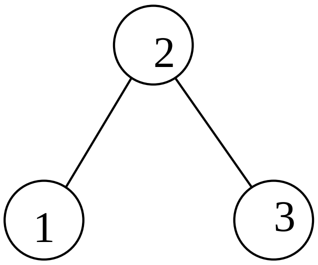
SINGLE ROTATION



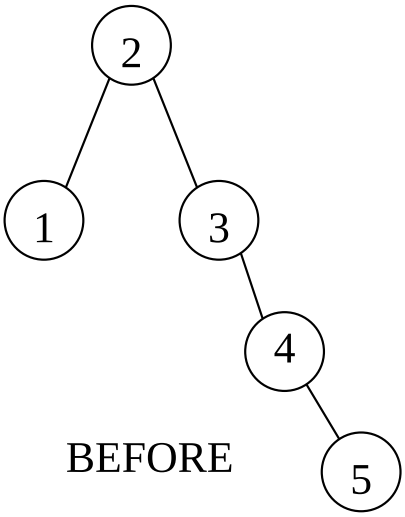
DOUBLE ROTATION



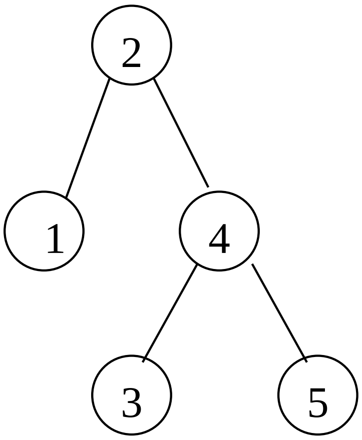
BEFORE



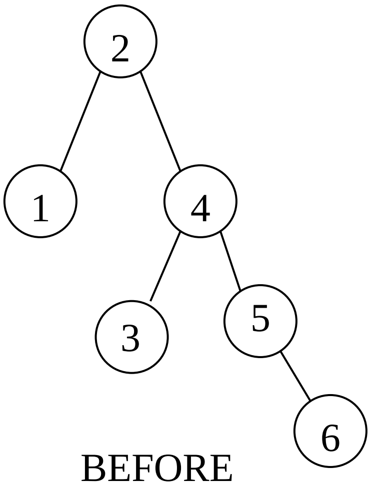
AFTER



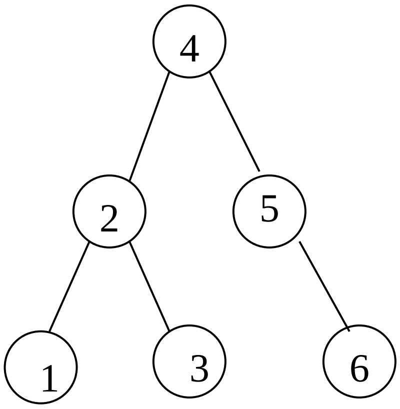
BEFORE



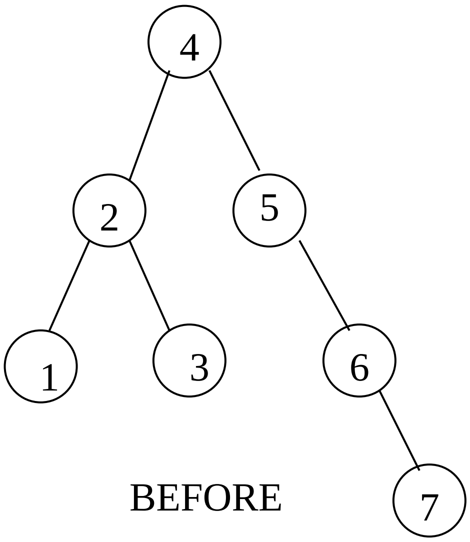
AFTER



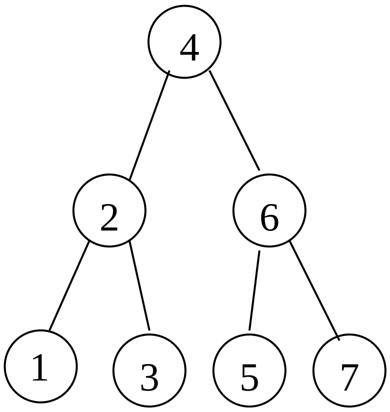
BEFORE



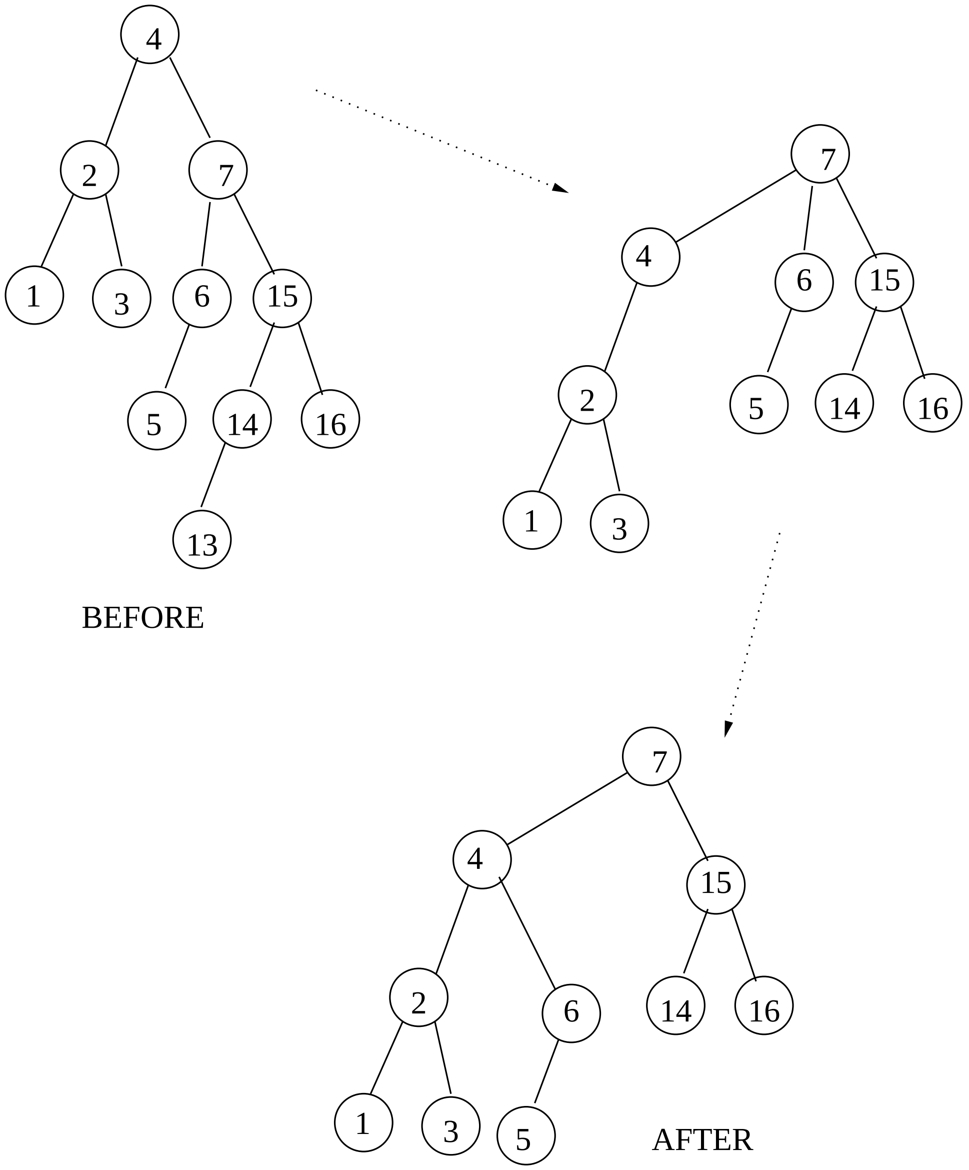
AFTER

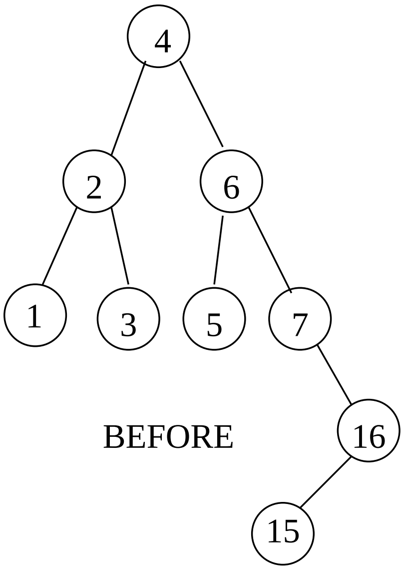


BEFORE

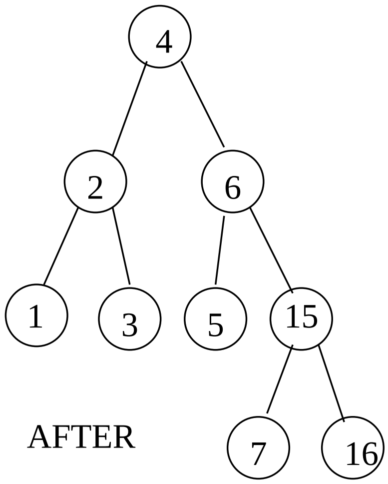


AFTER

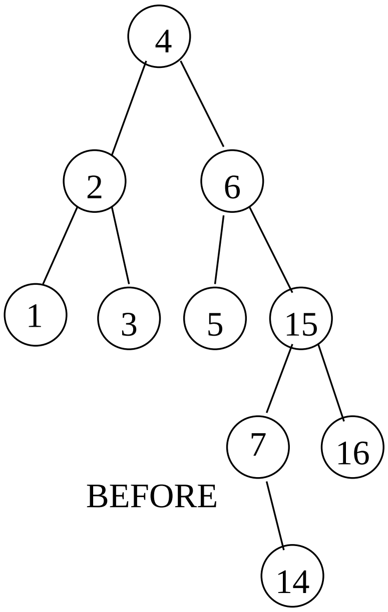

BEFORE
AFTER



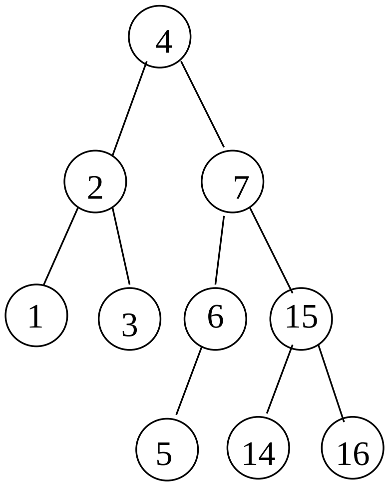
BEFORE



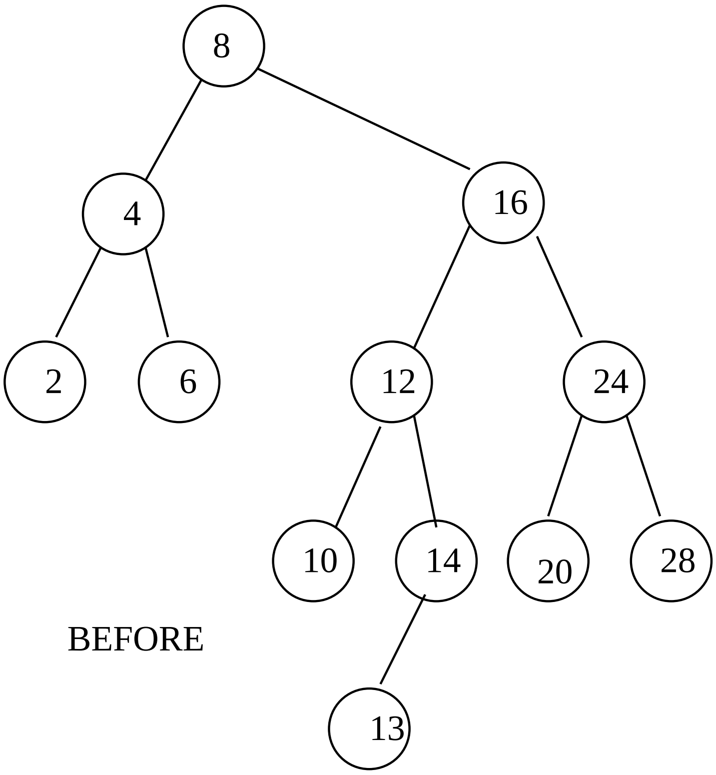
AFTER



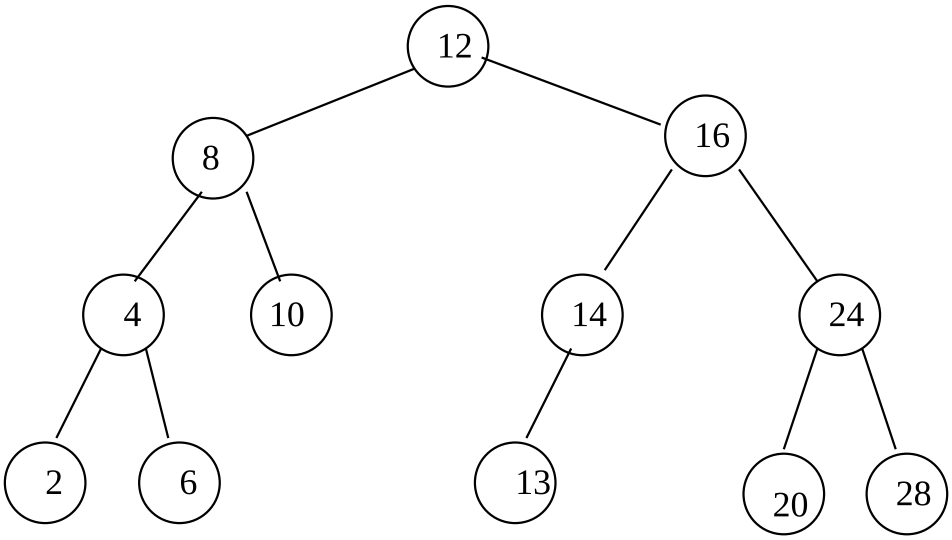
BEFORE



AFTER



BEFORE



AFTER