COMP171

Fall 2005

AVL-Trees

Balanced binary tree

- The disadvantage of a binary search tree is that its height can be as large as N-1
- This means that the time needed to perform insertion and deletion and many other operations can be O(N) in the worst case
- We want a tree with small height
- A binary tree with N node has height at least $\Theta(\log N)$
- Thus, our goal is to keep the height of a binary search tree O(log N)
- Such trees are called balanced binary search trees. Examples are AVL tree, red-black tree.

AVL tree

Height of a node

- The height of a leaf is 1. The height of a null pointer is zero.
- The height of an internal node is the maximum height of its children plus 1

Note that this definition of height is different from the one we defined previously (we defined the height of a leaf as zero previously).

AVL tree

- An AVL tree is a binary search tree in which
  - for *every* node in the tree, the height of the left and right subtrees differ by at most 1.

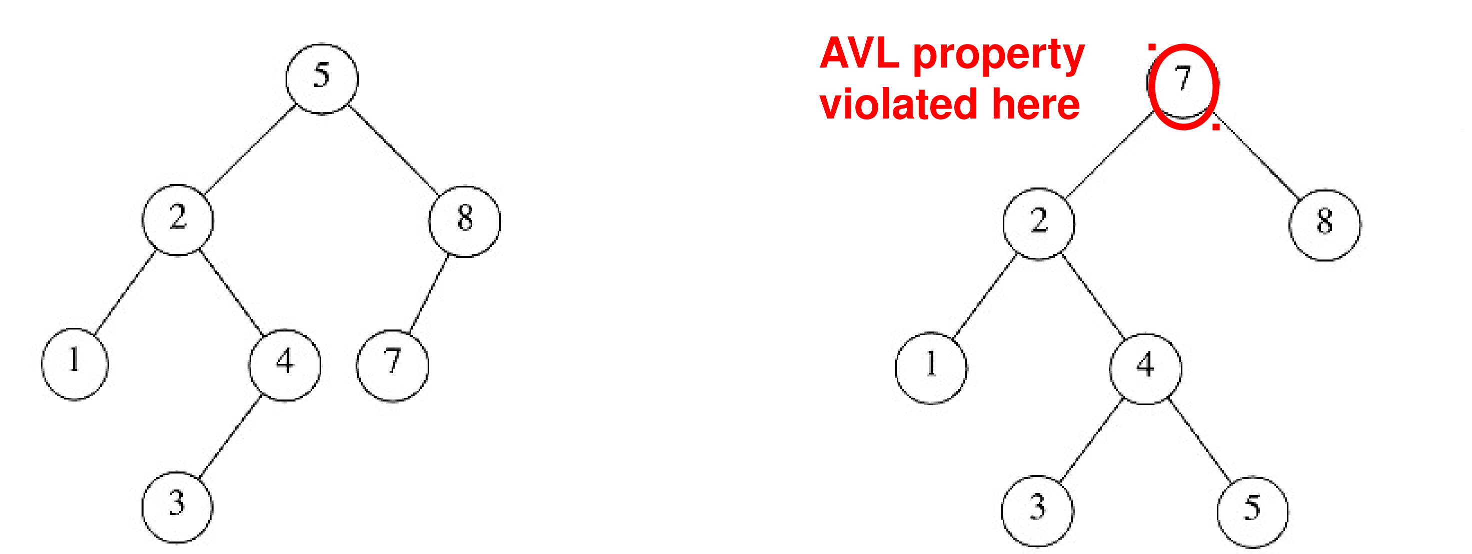


Figure 4.32 Two binary search trees. Only the left tree is AVL.

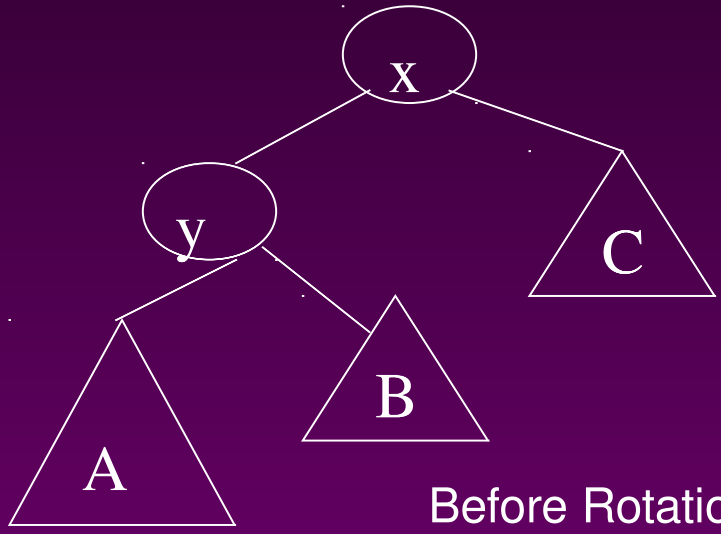
AVL tree

- Let x be the root of an AVL tree of height h
- Let N_h denote the minimum number of nodes in an AVL tree of height h
- Clearly, $N_i \geq N_{i-1}$ by definition
- We have
$$\begin{aligned} N_h &\geq N_{h-1} + N_{h-2} + 1 \\ &\geq 2N_{h-2} + 1 \\ &> 2N_{h-2} \end{aligned}$$
- By repeated substitution, we obtain the general form
$$N_h > 2^i N_{h-2}$$
- The boundary conditions are: $N_1=1$ and $N_2=2$. This implies that $h = O(\log N_h)$.
- Thus, many operations (searching, insertion, deletion) on an AVL tree will take O(log N) time.

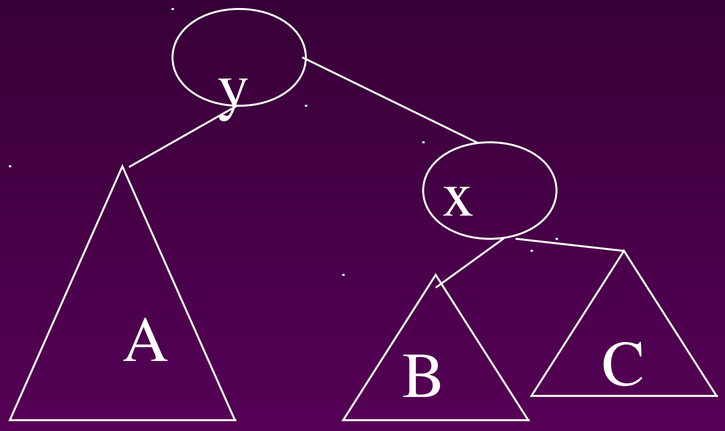
Rotations

- When the tree structure changes (e.g., insertion or deletion), we need to transform the tree to restore the AVL tree property.
- This is done using single rotations or double rotations.

e.g. Single Rotation



Before Rotation



After Rotation

Rotations

- Since an insertion/deletion involves adding/deleting a single node, this can only increase/decrease the height of some subtree by 1
- Thus, if the AVL tree property is violated at a node x, it means that the heights of left(x) ad right(x) differ by exactly 2.
- Rotations will be applied to x to restore the AVL tree property.

Insertion

- First, insert the new key as a new leaf just as in ordinary binary search tree
- Then trace the path from the new leaf towards the root. For each node x encountered, check if heights of left(x) and right(x) differ by at most 1.
- If yes, proceed to parent(x). If not, restructure by doing either a single rotation or a double rotation [next slide].
- For insertion, once we perform a rotation at a node x, we won't need to perform any rotation at any ancestor of x.

Insertion

- Let x be the node at which left(x) and right(x) differ by more than 1
- Assume that the height of x is h+3
- There are 4 cases
  - Height of left(x) is h+2 (i.e. height of right(x) is h)
    - Height of left(left(x)) is h+1 ⇒ single rotate with left child
    - Height of right(left(x)) is h+1 ⇒ double rotate with left child
  - Height of right(x) is h+2 (i.e. height of left(x) is h)
    - Height of right(right(x)) is h+1 ⇒ single rotate with right child
    - Height of left(right(x)) is h+1 ⇒ double rotate with right child

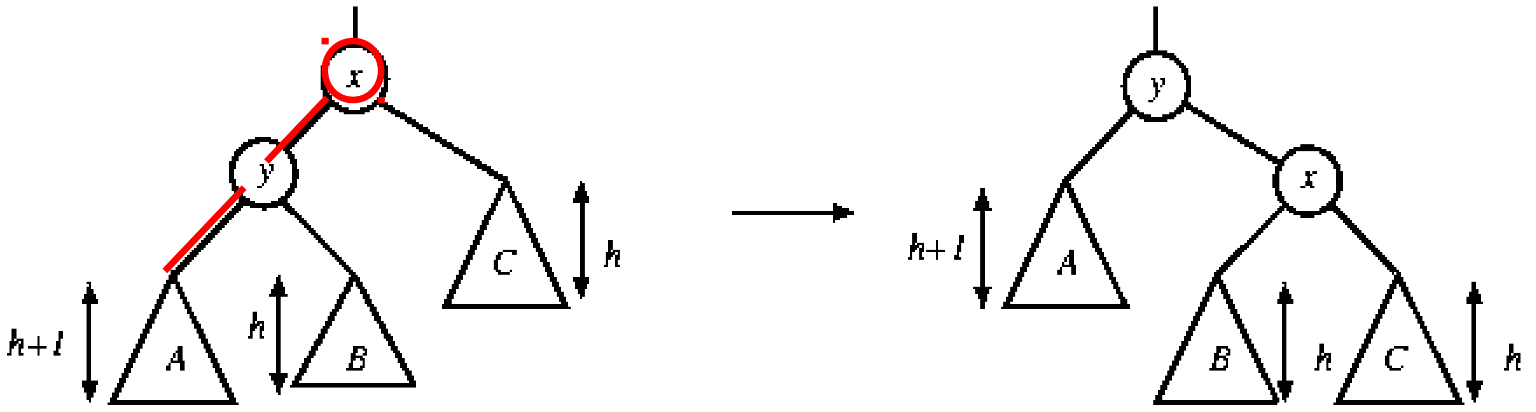
Note: Our test conditions for the 4 cases are different from the code shown in the textbook. These conditions allow a uniform treatment between insertion and deletion.

Single rotation

The new key is inserted in the subtree A.

The AVL-property is violated at x

- height of left(x) is h+2
- height of right(x) is h.

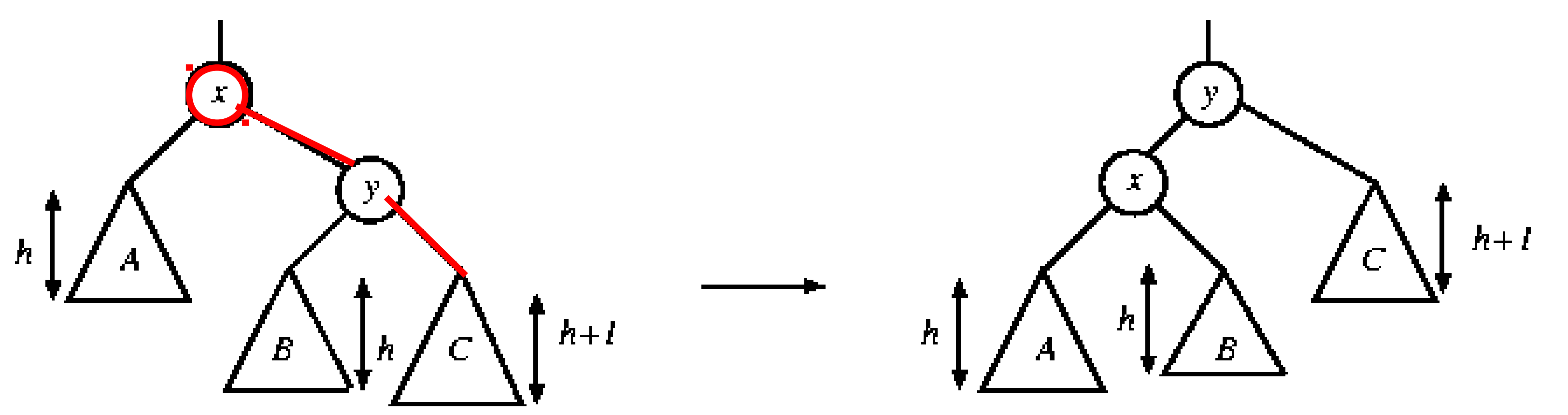


Rotate with left child

Single rotation

The new key is inserted in the subtree C.

The AVL-property is violated at x.

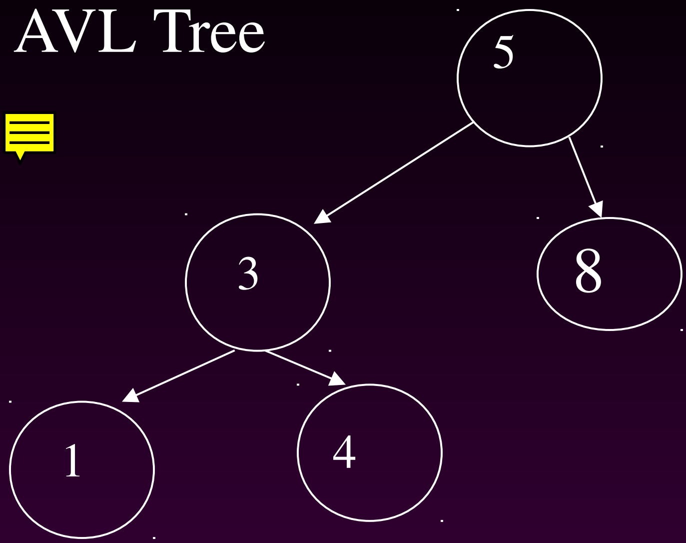


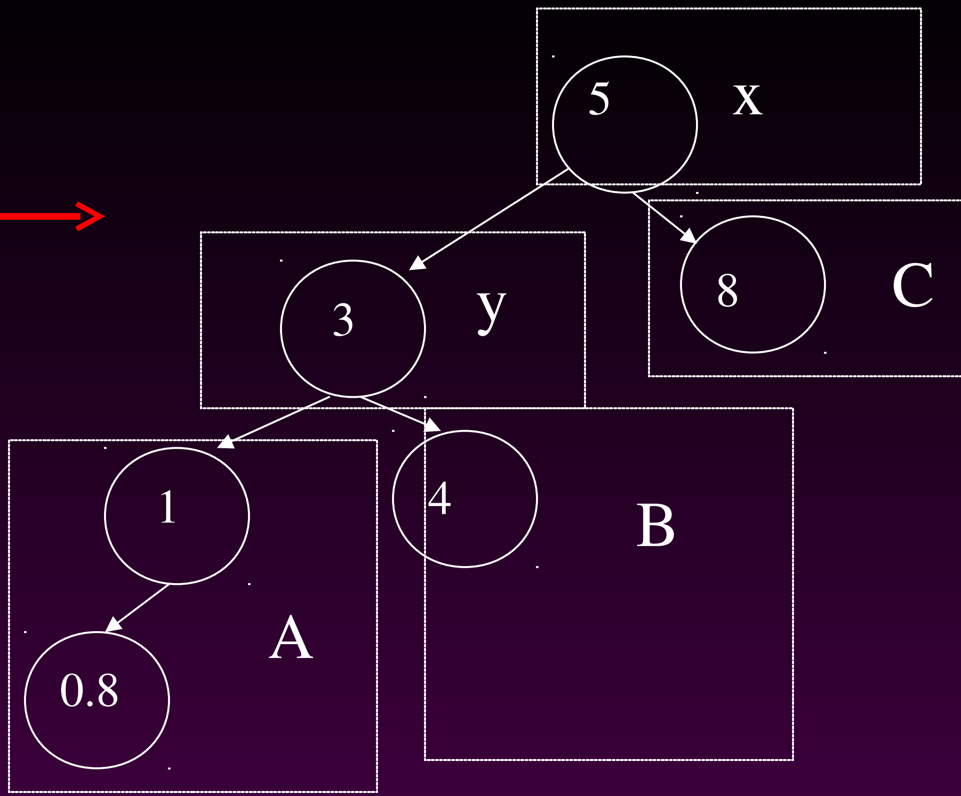
Rotate with right child

Single rotation takes O(1) time.

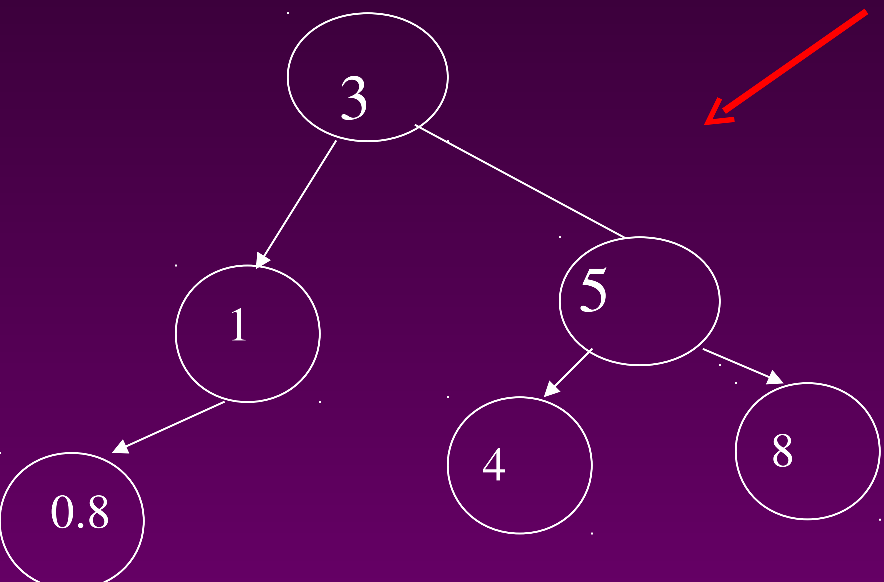
Insertion takes O(log N) time.

AVL Tree





Insert 0.8



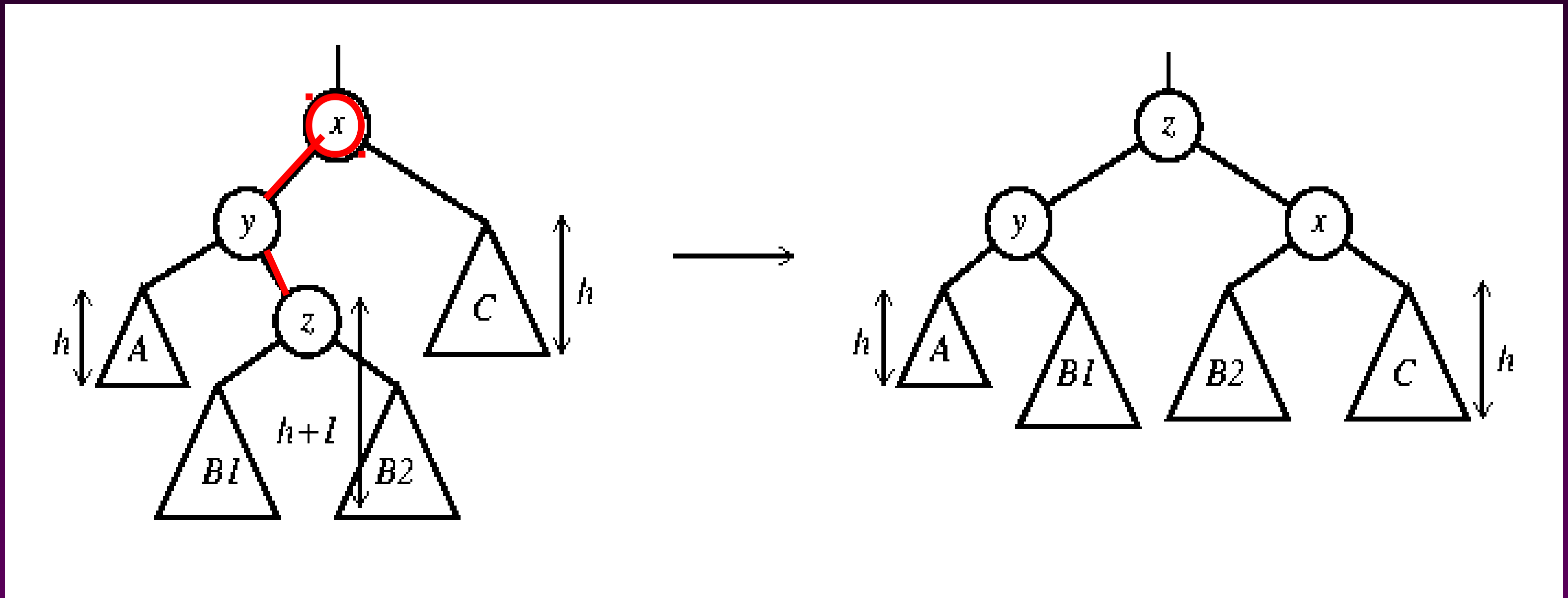
After rotation

Double rotation

The new key is inserted in the subtree B1 or B2.

The AVL-property is violated at x.

x-y-z forms a zig-zag shape



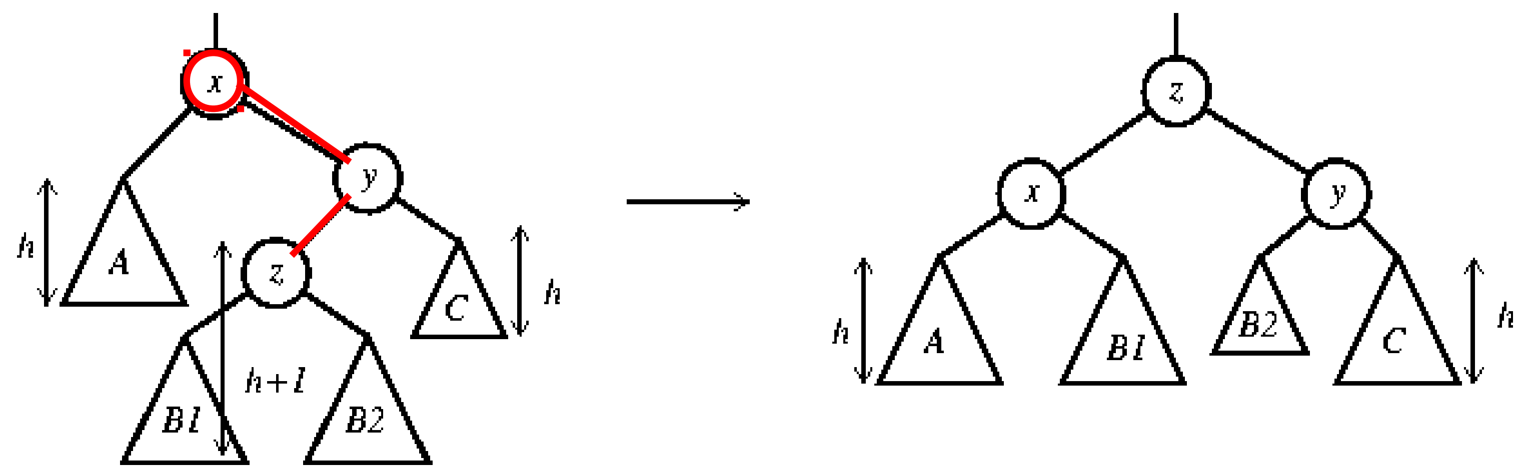
Double rotate with left child

also called left-right rotate

Double rotation

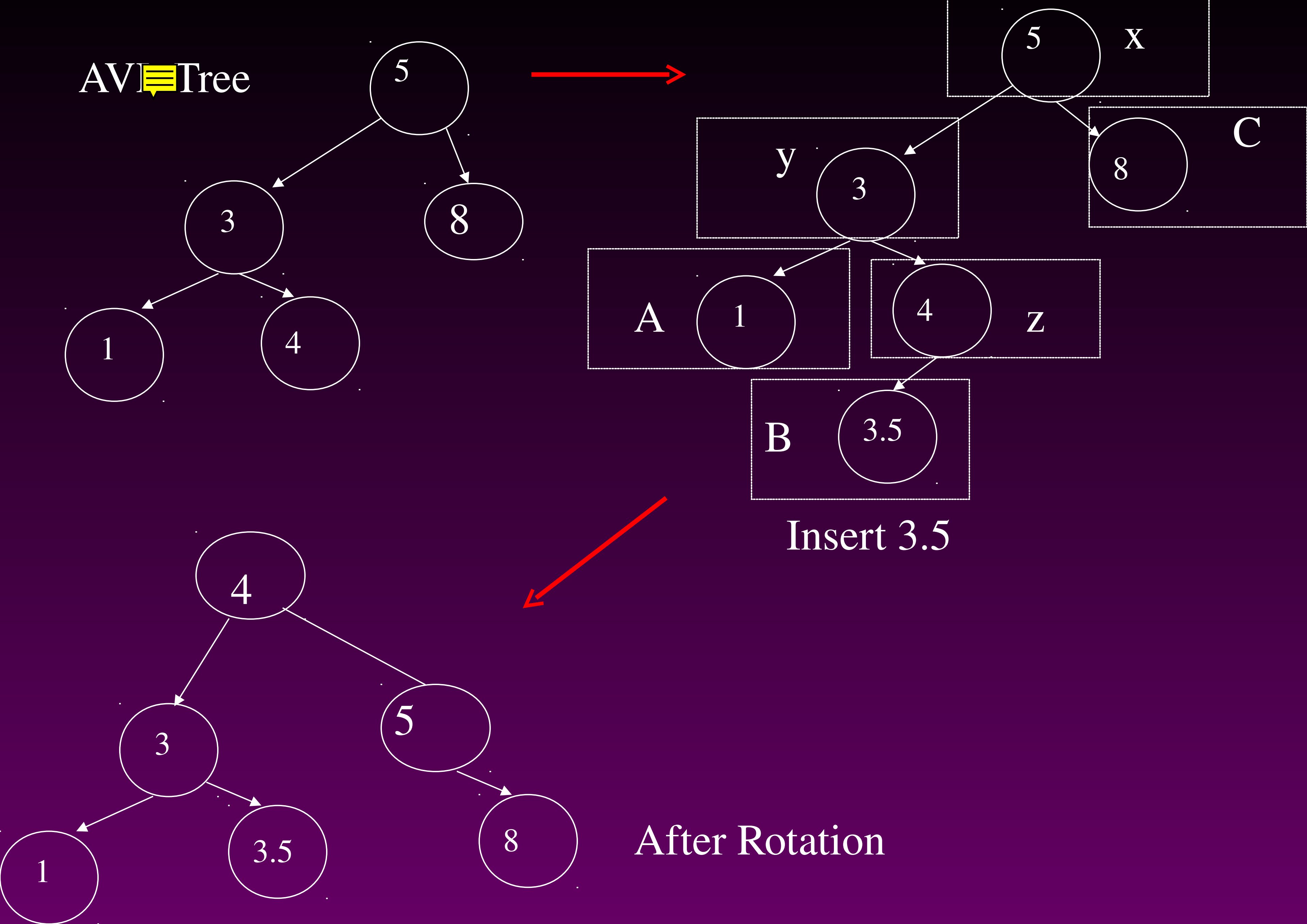
The new key is inserted in the subtree B1 or B2.

The AVL-property is violated at x.



Double rotate with right child

also called right-left rotate


AVL Tree
5
3
8
1
4
5
x
y
3
C
8
A
1
4
z
B
3.5
Insert 3.5
4
3
5
1
3.5
8
After Rotation

An Extended Example

Insert 3,2,1,4,5,6,7, 16,15,14

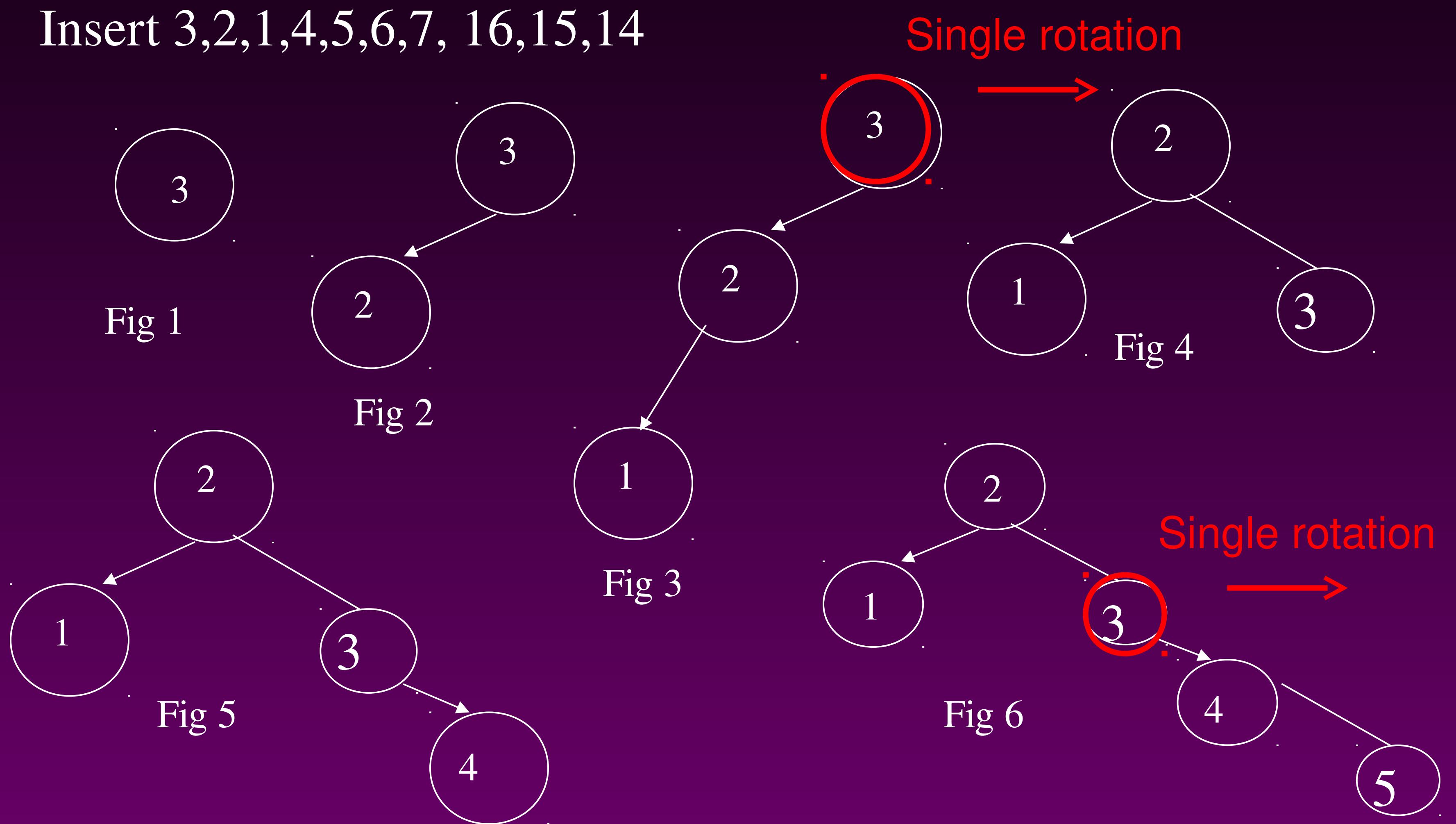


Fig 1

Fig 2

Fig 3

Fig 4

Fig 5

Fig 6

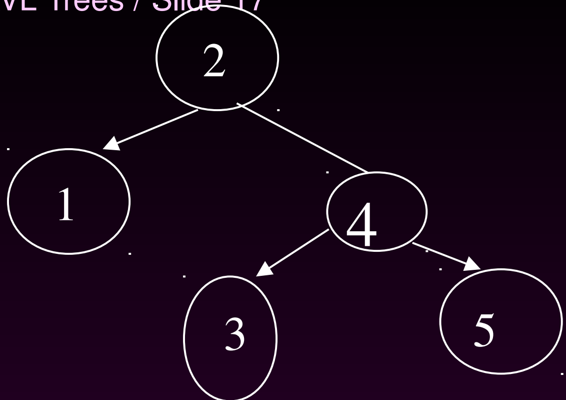


Fig 7

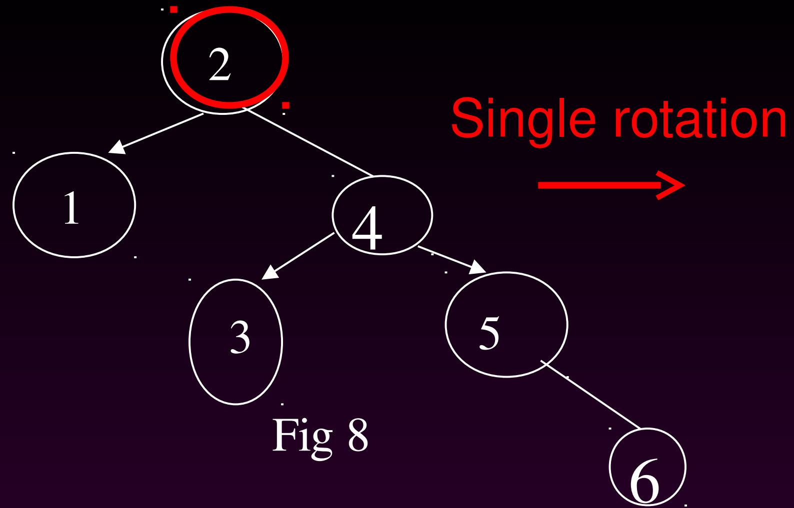


Fig 8

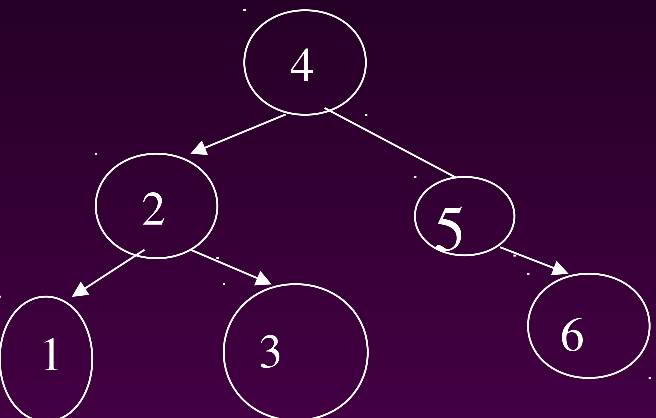


Fig 9

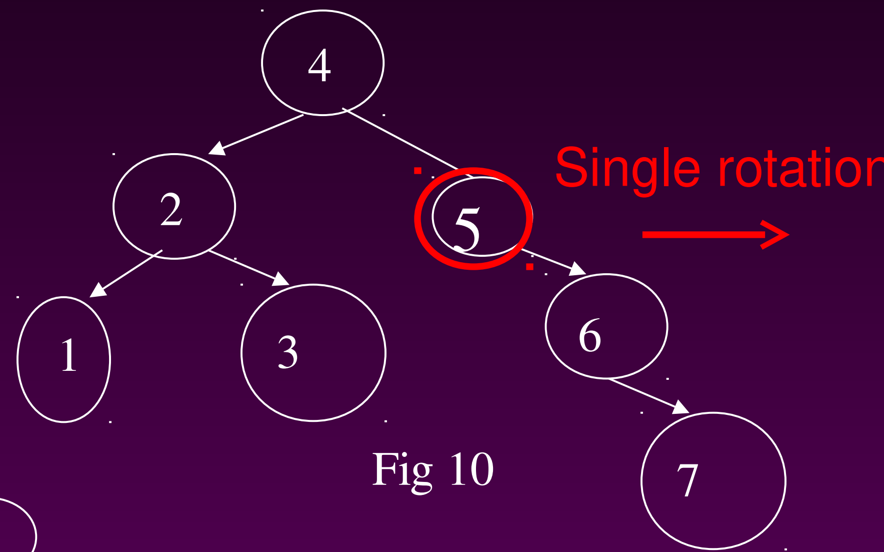


Fig 10

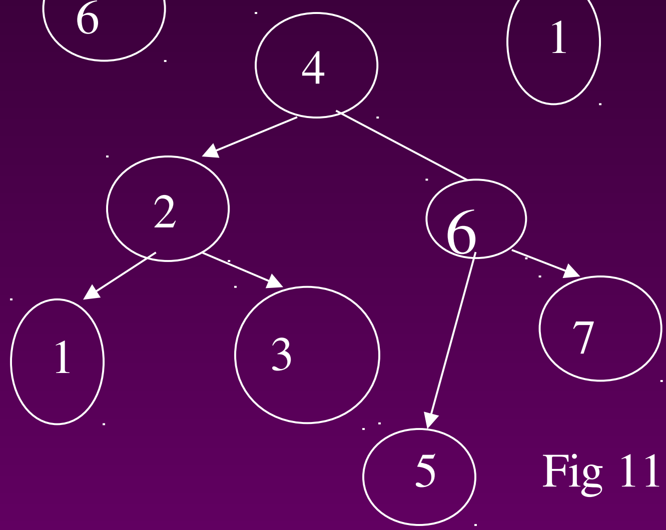


Fig 11

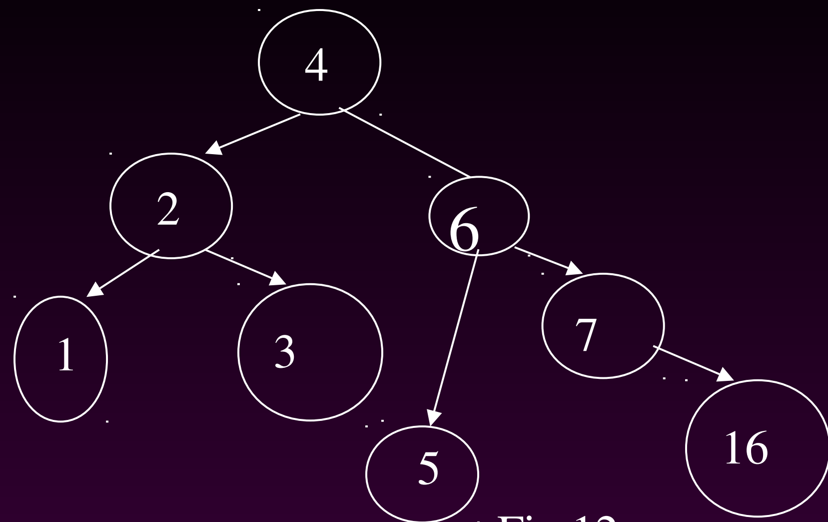


Fig 12

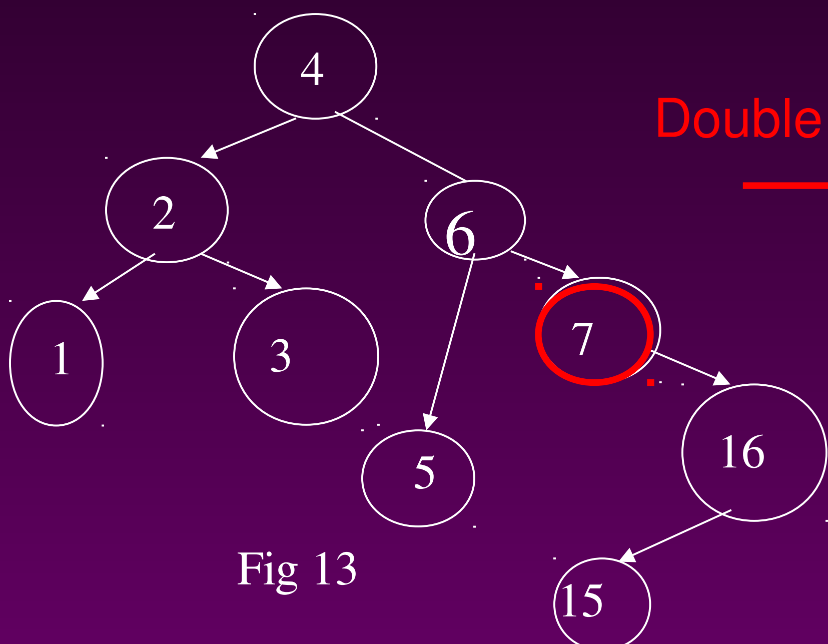


Fig 13

Double rotation

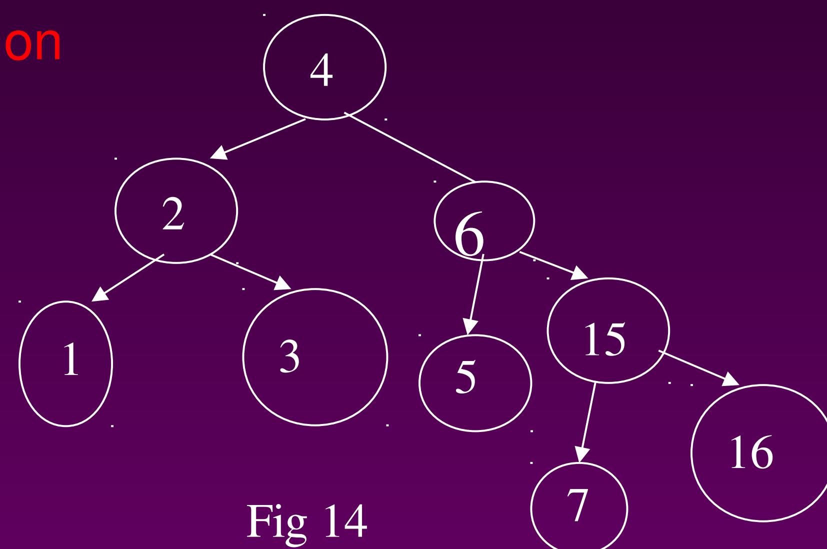


Fig 14

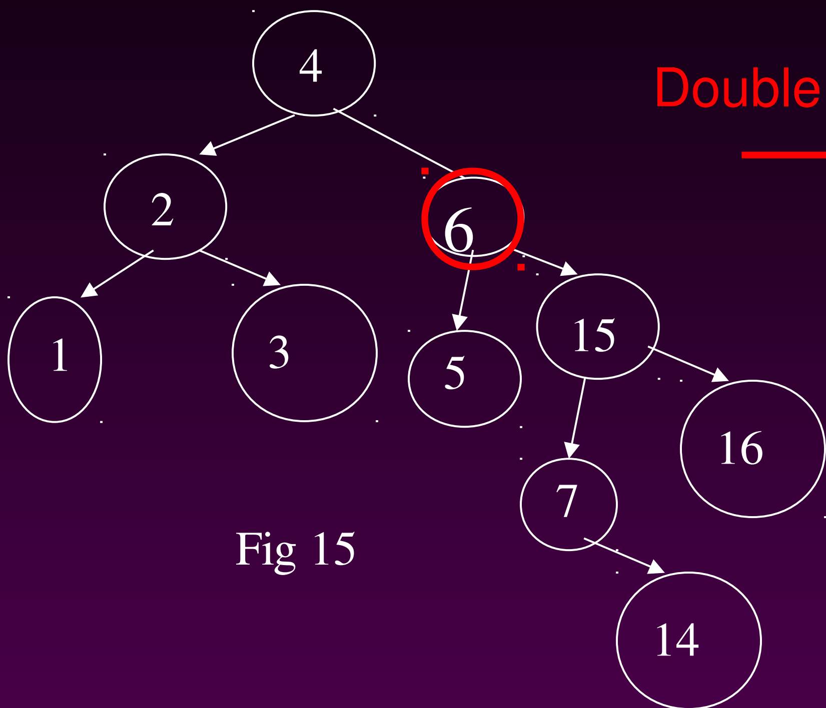


Fig 15

Double rotation

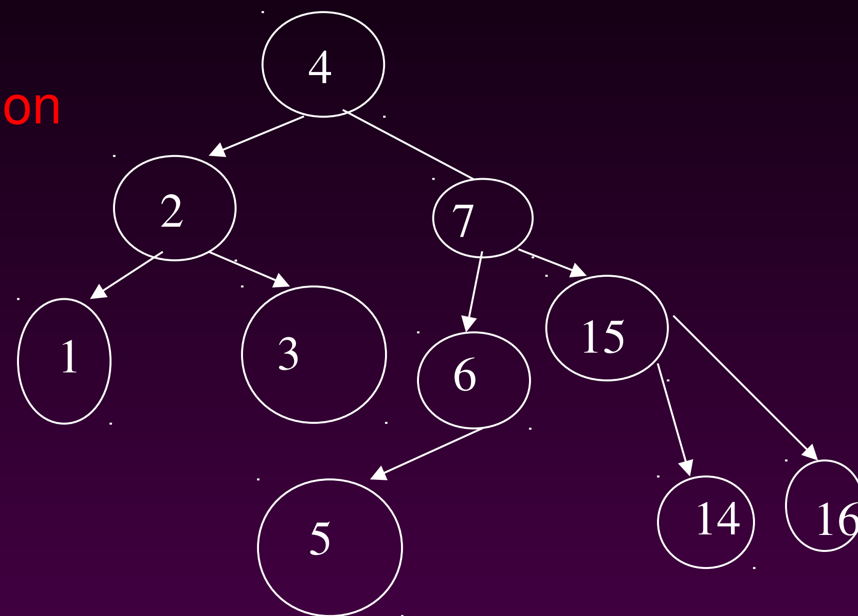


Fig 16

Deletion

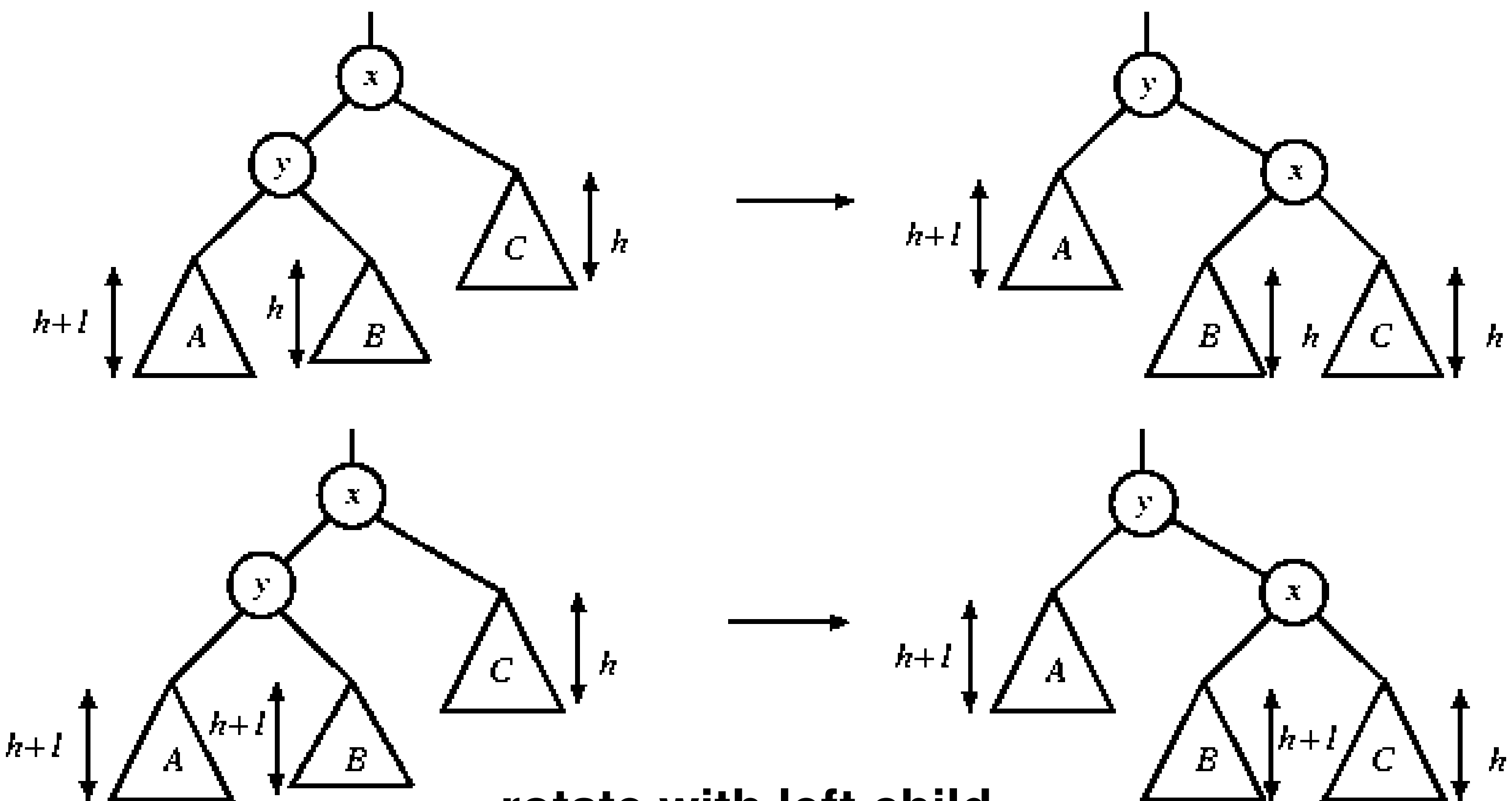
- Delete a node x as in ordinary binary search tree. Note that the last node deleted is a leaf.
- Then trace the path from the new leaf towards the root.
- For each node x encountered, check if heights of left(x) and right(x) differ by at most 1. If yes, proceed to parent(x). If not, perform an appropriate rotation at x. There are 4 cases as in the case of insertion.
- For deletion, after we perform a rotation at x, we may have to perform a rotation at some ancestor of x. Thus, we must continue to trace the path until we reach the root.

Deletion

- On closer examination: the single rotations for deletion can be divided into 4 cases (instead of 2 cases)
  - Two cases for rotate with left child
  - Two cases for rotate with right child

Single rotations in deletion

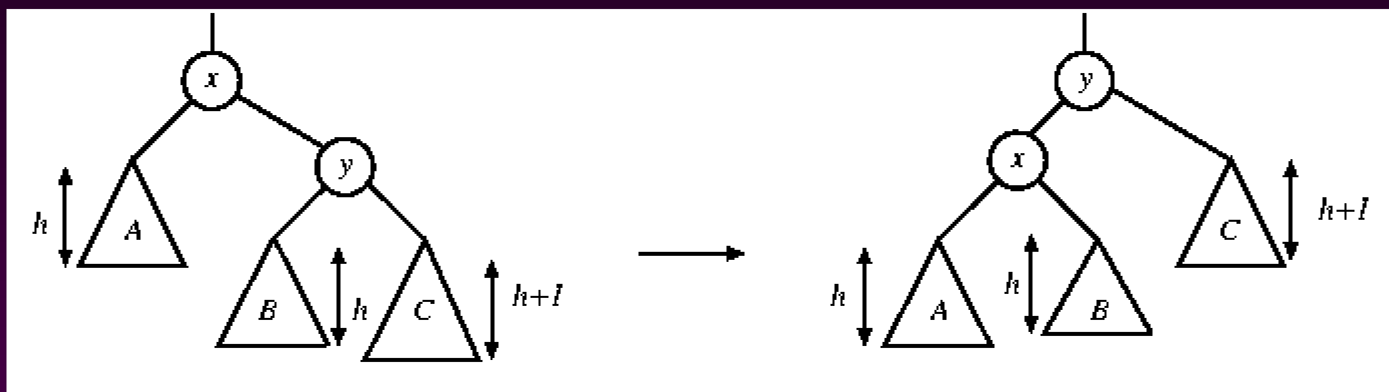
In both figures, a node is deleted in subtree C, causing the height to drop to h. The height of y is h+2. When the height of subtree A is h+1, the height of B can be h or h+1. Fortunately, the same single rotation can correct both cases.

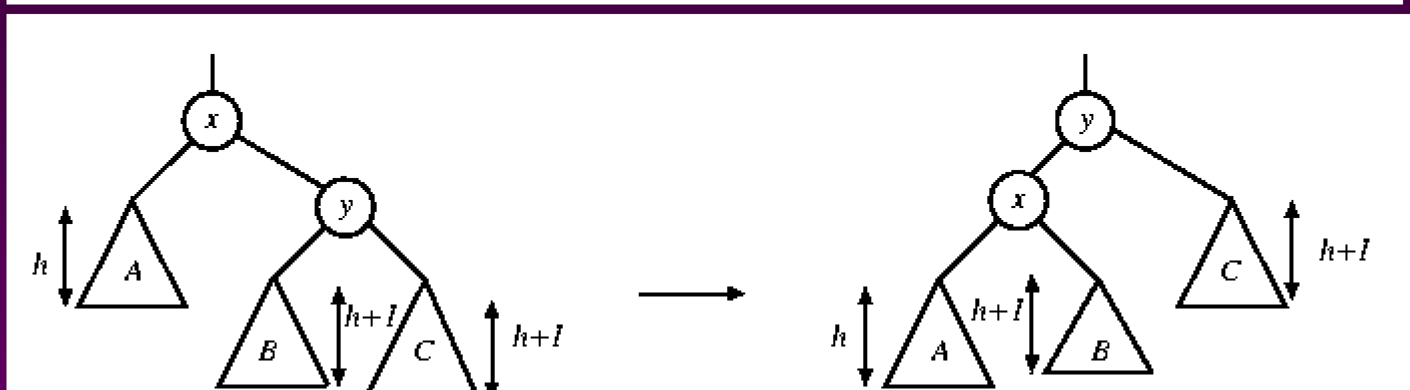


rotate with left child

Single rotations in deletion

In both figures, a node is deleted in subtree A, causing the height to drop to h. The height of y is h+2. When the height of subtree C is h+1, the height of B can be h or h+1. A single rotation can correct both cases.





rotate with right child

Rotations in deletion

- There are 4 cases for single rotations, but we do not need to distinguish among them.
- There are exactly two cases for double rotations (as in the case of insertion)
- Therefore, we can reuse exactly the same procedure for insertion to determine which rotation to perform