UIT2201 Programming and Data Structures
AVL Trees

Chandrabose Aravindan
<AravindanC@ssn.edu.in>

Professor of Information Technology
SSN College of Engineering

July 14, 2022

Order among the objects

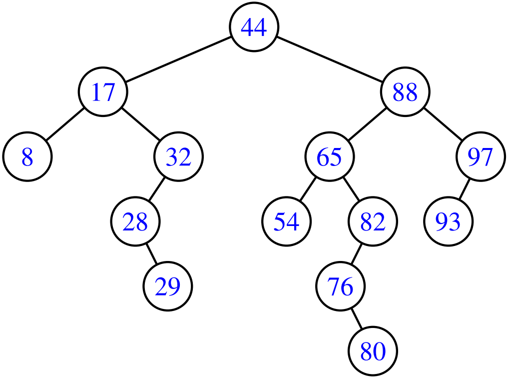
- Searching for an object in a tree collection is as complex as searching for an object in a linear collection
- There are several applications where we need operations such as
  - find()
  - findMin()
  - findMax()
  - deleteMin()
  - deleteMax()
  - insert()
  - delete()
- It is safe to assume that a partial order $<$ can be defined on set of objects and this order can be imposed on the tree as well!

Binary Search Tree

Binary Search Tree

Binary Search Tree (BST) is a binary tree that stores objects in its nodes in such a way that all the objects in the left subtree of any node with object x are $< x$ and all the objects in the right subtree are $x <$, for some partial order $<$ among the objects.

Binary Search Tree: Example



Analysis of BST Operations

- Complexities of operations on a BST are proportional to the height of the BST

Analysis of BST Operations

- Complexities of operations on a BST are proportional to the height of the BST
- If n objects are inserted in to an initially empty BST, what may be the height of the resulting BST?

Analysis of BST Operations

- Complexities of operations on a BST are proportional to the height of the BST
- If n objects are inserted in to an initially empty BST, what may be the height of the resulting BST?
- Worst Case: height of the BST is $n - 1$ ($O(n)$)

Analysis of BST Operations

- Complexities of operations on a BST are proportional to the height of the BST
- If n objects are inserted in to an initially empty BST, what may be the height of the resulting BST?
- Worst Case: height of the BST is $n - 1$ ($O(n)$)
- Best Case: resulting BST may be a complete (full) binary tree with height that grows like $\log n$ ($O(\log n)$)

Analysis of BST Operations

- Complexities of operations on a BST are proportional to the height of the BST
- If n objects are inserted in to an initially empty BST, what may be the height of the resulting BST?
- Worst Case: height of the BST is $n - 1$ ($O(n)$)
- Best Case: resulting BST may be a complete (full) binary tree with height that grows like $\log n$ ($O(\log n)$)
- Average Case?

Analysis of BST Operations

- Complexities of operations on a BST are proportional to the height of the BST
- If n objects are inserted in to an initially empty BST, what may be the height of the resulting BST?
- Worst Case: height of the BST is $n-1$ ($O(n)$)
- Best Case: resulting BST may be a complete (full) binary tree with height that grows like $\log n$ ($O(\log n)$)
- Average Case? $O(\log n)$

Height Balance

- Though the average case complexity is logarithmic on the number of items, the worst case performance may not be acceptable

Height Balance

- Though the average case complexity is logarithmic on the number of items, the worst case performance may not be acceptable
- Is it possible to keep the height under control (close to log N)?

Height Balance

- Though the average case complexity is logarithmic on the number of items, the worst case performance may not be acceptable
- Is it possible to keep the height under control (close to $\log N$)?
- Of course, the amount of additional work we do to keep the height under control should preferably be $O(1)$ or should not exceed $O(\log N)$

AVL Tree

- Idea proposed by Adelson-Velskii and Lendis (hence the name AVL)

AVL Tree

- Idea proposed by Adelson-Velskii and Lendis (hence the name AVL)
- Intention is to keep the height of the tree under control (close to $log\ N$)

AVL Tree

- Idea proposed by Adelson-Velskii and Lendis (hence the name AVL)
- Intention is to keep the height of the tree under control (close to $log\ N$)
- In addition to ordering condition of BST, a structural condition is also imposed

AVL Tree

- Idea proposed by Adelson-Velskii and Lendis (hence the name AVL)
- Intention is to keep the height of the tree under control (close to $log\ N$)
- In addition to ordering condition of BST, a structural condition is also imposed
- Like the ordering condition, the structural condition is also imposed on all the nodes (not just the root node)

AVL Tree

- Idea proposed by Adelson-Velskii and Lendis (hence the name AVL)
- Intention is to keep the height of the tree under control (close to $log\ N$)
- In addition to ordering condition of BST, a structural condition is also imposed
- Like the ordering condition, the structural condition is also imposed on all the nodes (not just the root node)
- Ideally we may expect both the left and right sub-trees to have the same height

AVL Tree

- Idea proposed by Adelson-Velskii and Lendis (hence the name AVL)
- Intention is to keep the height of the tree under control (close to $log\ N$)
- In addition to ordering condition of BST, a structural condition is also imposed
- Like the ordering condition, the structural condition is also imposed on all the nodes (not just the root node)
- Ideally we may expect both the left and right sub-trees to have the same height
- But, that may not be always possible!

Structural Condition

Structural Condition

For every node in the tree, the heights of left and right sub-trees can differ by at most 1

Structural Condition

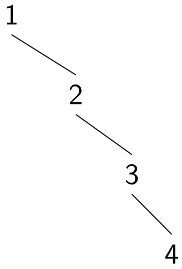
Structural Condition

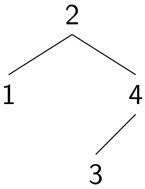
For every node in the tree, the heights of left and right sub-trees can differ by at most 1

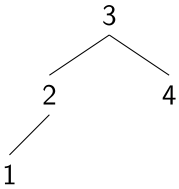
Note

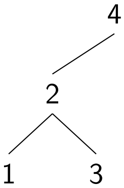
Note that the height of an empty tree is considered as -1

Which of these are AVL Trees?









AVL Analysis

- With this structural condition in place, are we sure that the height is maintained closer to *log N* ?

AVL Analysis

- With this structural condition in place, are we sure that the height is maintained closer to *log N* ?
- We can approach this in the reverse direction and ask what would be the minimum number of nodes in an AVL tree of height h?

AVL Analysis

- With this structural condition in place, are we sure that the height is maintained closer to *log N* ?
- We can approach this in the reverse direction and ask what would be the minimum number of nodes in an AVL tree of height h?
- It is easy to see that

$$T(0) = 1$$

$$T(1) = 2$$

$$T(h) = T(h-1) + T(h-2) + 1$$

AVL Analysis

- With this structural condition in place, are we sure that the height is maintained closer to $log\ N$?
- We can approach this in the reverse direction and ask what would be the minimum number of nodes in an AVL tree of height h?
- It is easy to see that

$$T(0) = 1$$

$$T(1) = 2$$

$$T(h) = T(h-1) + T(h-2) + 1$$

- This is closely related to Fibonacci numbers!

AVL Analysis

AVL Tree Property

Height of an AVL tree is approximately equal to

$$1.44 \log (N + 2) - 1.328$$

Preserving the structural condition

- Insertion and deletion (BST operations) may destroy the structural condition.

Preserving the structural condition

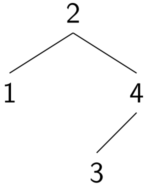
- Insertion and deletion (BST operations) may destroy the structural condition.
- AVL Tree approach: Insertion and deletion operations are carried out obeying first the ordering conditions — that is, perform BST insertion or deletion

Preserving the structural condition

- Insertion and deletion (BST operations) may destroy the structural condition.
- AVL Tree approach: Insertion and deletion operations are carried out obeying first the ordering conditions — that is, perform BST insertion or deletion
- If the structural property is violated, perform certain operations (in constant time?) to restore the balance

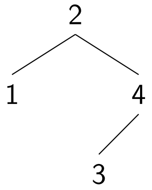
Example

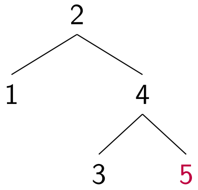
- Insert 5 into the following AVL Tree



Example

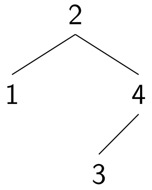
- Insert 5 into the following AVL Tree

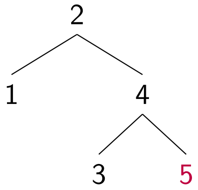




Example

- Insert 5 into the following AVL Tree

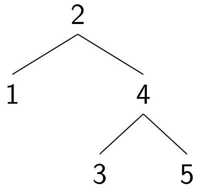




- Is the structural condition violated after the insertion?

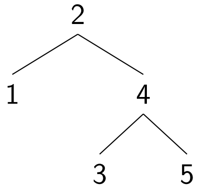
Example

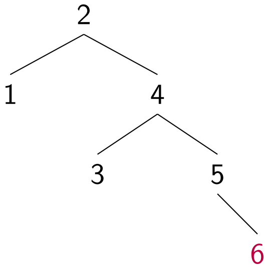
- Let us now continue to insert 6



Example

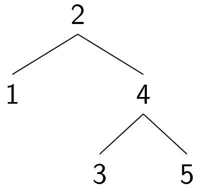
- Let us now continue to insert 6

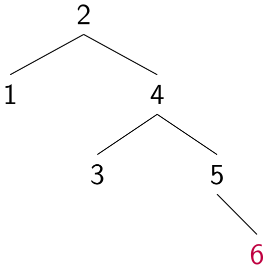




Example

- Let us now continue to insert 6





- Now, violation happens at the root node

Four cases of violation

- After inserting a new node, we should check the structural condition for every node from the place of insertion up towards the root

Four cases of violation

- After inserting a new node, we should check the structural condition for every node from the place of insertion up towards the root
- Suppose the violation is detected at some internal node α

Four cases of violation

- After inserting a new node, we should check the structural condition for every node from the place of insertion up towards the root
- Suppose the violation is detected at some internal node α
- Case 1: Insertion happened in the left subtree of left child of α

Four cases of violation

- After inserting a new node, we should check the structural condition for every node from the place of insertion up towards the root
- Suppose the violation is detected at some internal node α
- Case 1: Insertion happened in the left subtree of left child of α
- Case 2: Insertion happened in the right subtree of right child of α

Four cases of violation

- After inserting a new node, we should check the structural condition for every node from the place of insertion up towards the root
- Suppose the violation is detected at some internal node α
- Case 1: Insertion happened in the left subtree of left child of α
- Case 2: Insertion happened in the right subtree of right child of α
- Case 3: Insertion happened in the right subtree of left child of α

Four cases of violation

- After inserting a new node, we should check the structural condition for every node from the place of insertion up towards the root
- Suppose the violation is detected at some internal node α
- Case 1: Insertion happened in the left subtree of left child of α
- Case 2: Insertion happened in the right subtree of right child of α
- Case 3: Insertion happened in the right subtree of left child of α
- Case 4: Insertion happened in the left subtree of right child of α

Four cases of violation

- After inserting a new node, we should check the structural condition for every node from the place of insertion up towards the root
- Suppose the violation is detected at some internal node α
- Case 1: Insertion happened in the left subtree of left child of α
- Case 2: Insertion happened in the right subtree of right child of α
- Case 3: Insertion happened in the right subtree of left child of α
- Case 4: Insertion happened in the left subtree of right child of α
- We perform a special operation called 'rotation' at α to correct the violation

Four cases of violation

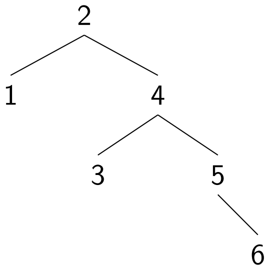
- After inserting a new node, we should check the structural condition for every node from the place of insertion up towards the root
- Suppose the violation is detected at some internal node α
- Case 1: Insertion happened in the left subtree of left child of α
- Case 2: Insertion happened in the right subtree of right child of α
- Case 3: Insertion happened in the right subtree of left child of α
- Case 4: Insertion happened in the left subtree of right child of α
- We perform a special operation called 'rotation' at α to correct the violation
- There is no need to look up further in the path!

Example: Rotation Operation

- After inserting 6, violation has occurred at the root node (α)

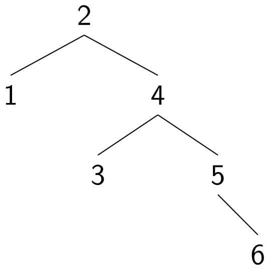
Example: Rotation Operation

- After inserting 6, violation has occurred at the root node (α)



Example: Rotation Operation

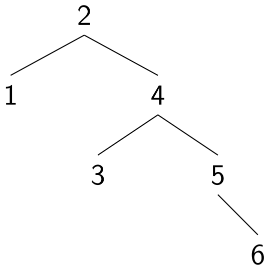
- After inserting 6, violation has occurred at the root node (α)



- This is Case 2 violation (insertion occurred in the right subtree of right child)

Example: Rotation Operation

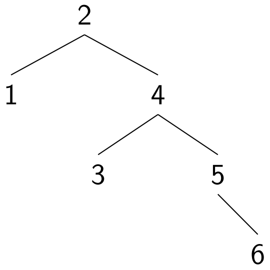
- After inserting 6, violation has occurred at the root node (α)



- This is Case 2 violation (insertion occurred in the right subtree of right child)
- This can be corrected by performing a 'rotation' operation at α

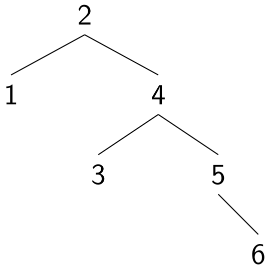
Example: Rotation Operation

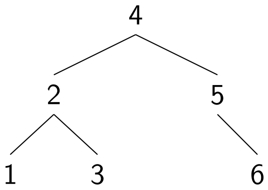
- Anti-clockwise rotation can be performed at α to restore the structural balance of the tree



Example: Rotation Operation

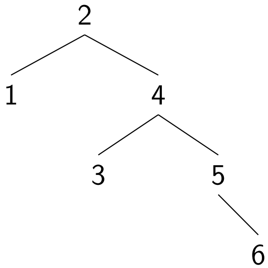
- Anti-clockwise rotation can be performed at α to restore the structural balance of the tree

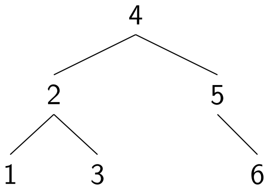




Example: Rotation Operation

- Anti-clockwise rotation can be performed at α to restore the structural balance of the tree

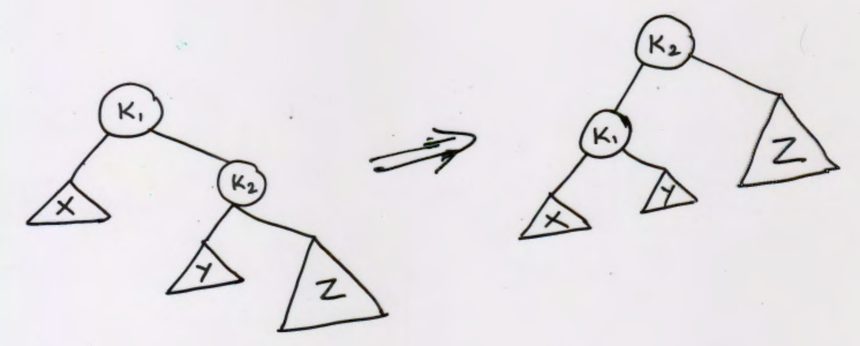




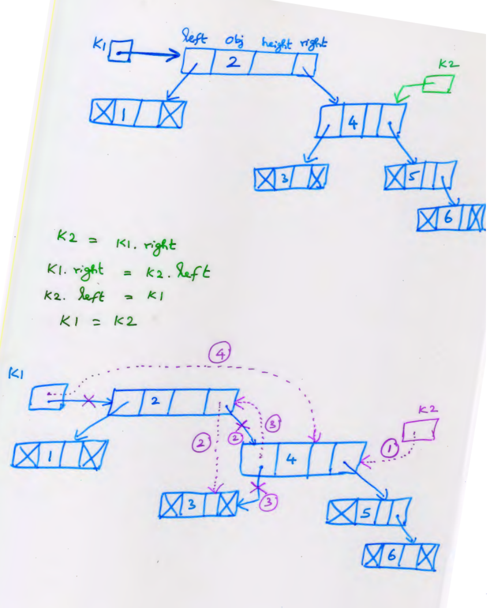
- Since only one path is involved, the complexity remains as $O(\log N)$

Rotation Operation

- As we have seen, Case 2 violation is corrected by performing an anti-clockwise rotation at α



Data Strucure view of rotation



Rotation Operation

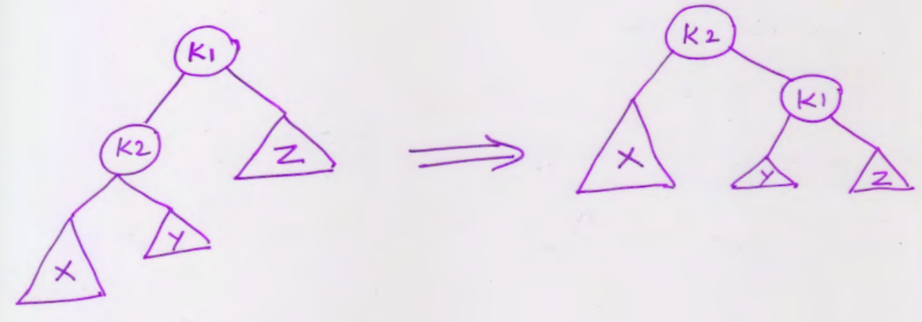
- Case 1 violation is very similar to that of Case 2

Rotation Operation

- Case 1 violation is very similar to that of Case 2
- Case 1 violation is corrected by performing a clockwise rotation at α

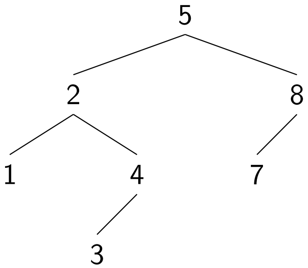
Rotation Operation

- Case 1 violation is very similar to that of Case 2
- Case 1 violation is corrected by performing a clockwise rotation at α



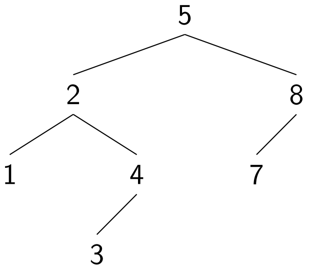
Example

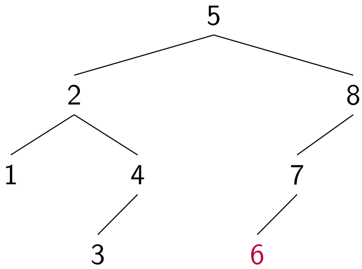
- Insert 6 into the following AVL Tree



Example

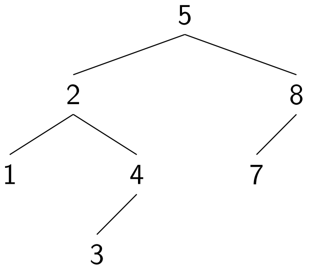
- Insert 6 into the following AVL Tree

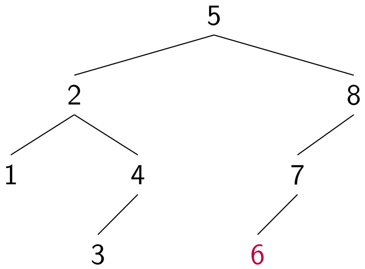




Example

- Insert 6 into the following AVL Tree

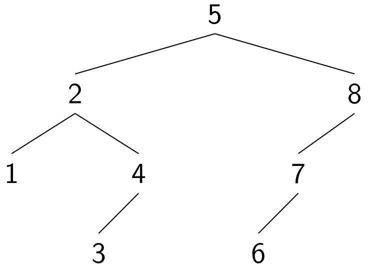




- Is there any violation after the insertion? If so, which node α is the first violating node?

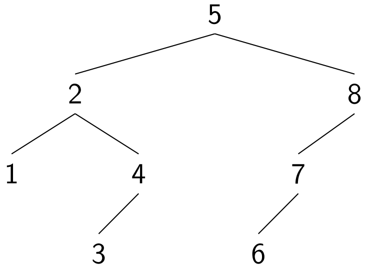
Example

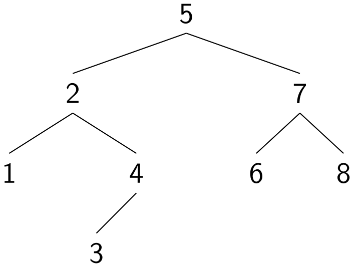
- Violation occurs at the node containing the key 8
- This is Case 1 violation and can be corrected by a clockwise rotation



Example

- Violation occurs at the node containing the key 8
- This is Case 1 violation and can be corrected by a clockwise rotation





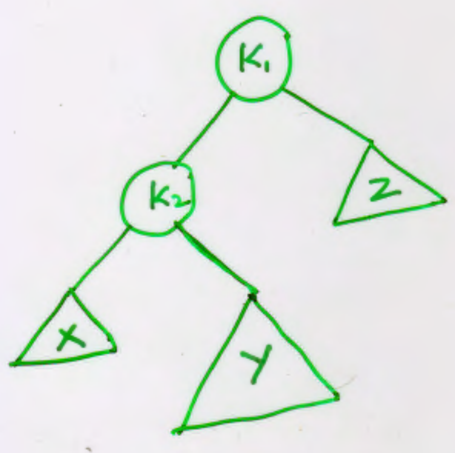
Inside Insertions

- Solution similar to Case 1 and Case 2 does not work for Case 3 and Case 4 inside insertions (left-right and right-left insertions)

Case 3 Violation

- Let us consider Case 3 violation, where insertion has occurred in the right subtree of left child of α

- Let us consider Case 3 violation, where insertion has occurred in the right subtree of left child of α

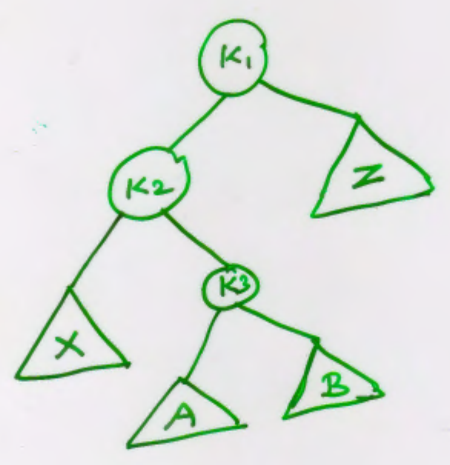


Case 3 Violation

- Note that the subtree Y is not empty, since insertion has occurred there
- In other words, Y must have at least one node

Case 3 Violation

- Note that the subtree Y is not empty, since insertion has occurred there
- In other words, Y must have at least one node

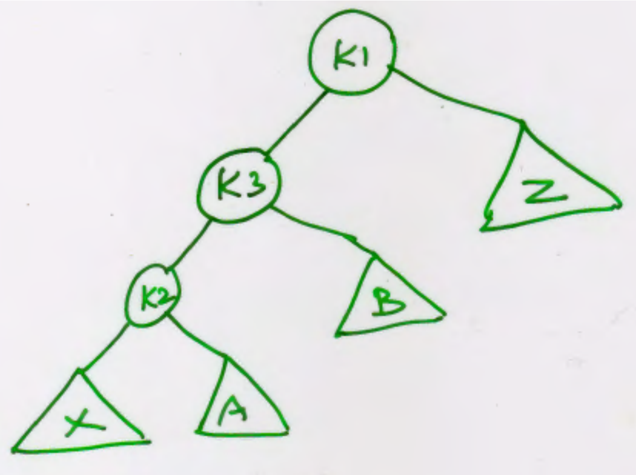


Case 3 Violation

- Anti-clockwise rotation between $k3$ and $k2$ converts the violation to "outside" violation!

Case 3 Violation

- Anti-clockwise rotation between $k3$ and $k2$ converts the violation to "outside" violation!

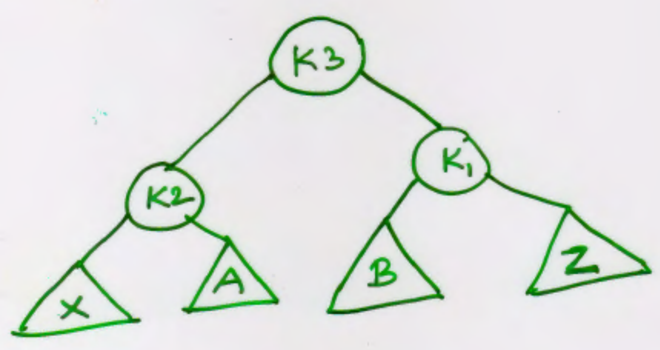


Case 3 Violation

- Now we can perform one more clockwise rotation between $k3$ and $k1$ to restore the balance!

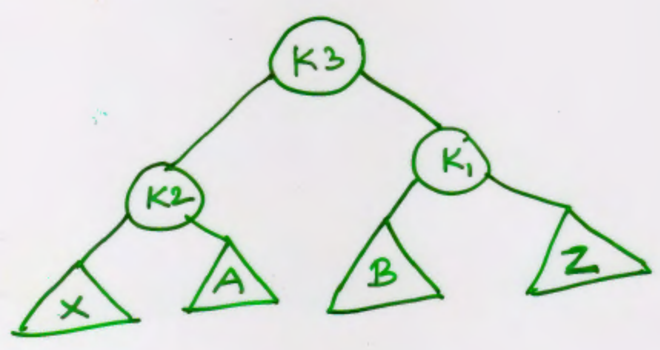
Case 3 Violation

- Now we can perform one more clockwise rotation between $k3$ and $k1$ to restore the balance!



Case 3 Violation

- Now we can perform one more clockwise rotation between $k3$ and $k1$ to restore the balance!



- Thus, a "double rotation" operation restores balance in Case 3 violation

Case 4 Violation

- A similar "first clockwise then anti-clockwise" "double rotation" can be performed to restore the balance in Case 4 violation

Exercise

- Insert the following keys in the given order into an initially empty AVL tree: $2, 1, 4, 5, 9, 3, 6, 7, 8$