CS 2124: DATA STRUCTURES
Spring 2024

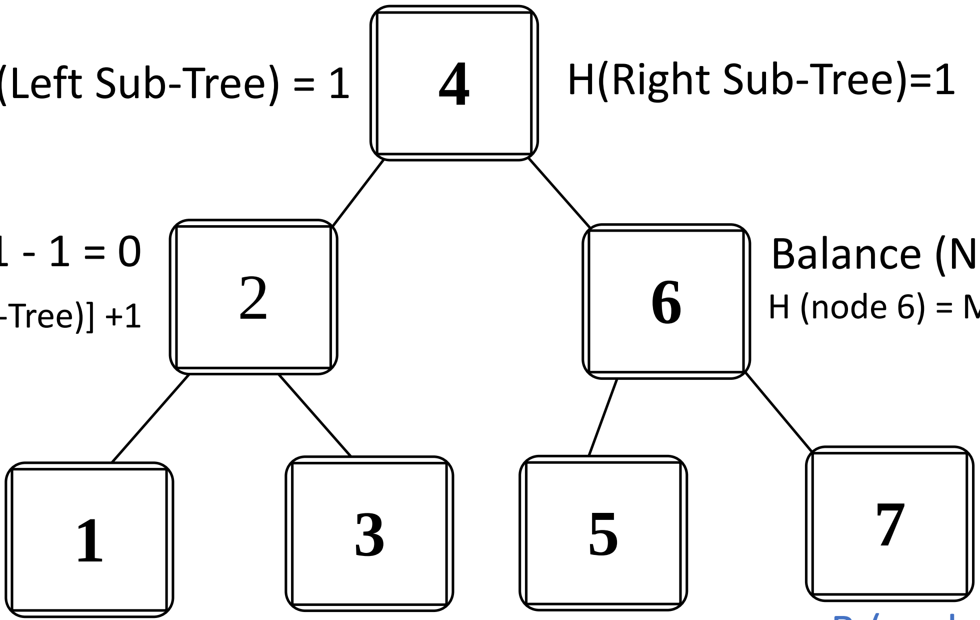
Topics: AVL Trees and Segment Trees

AVL Tree (Balance)

B (node 4) = H(Left Sub-Tree) - H(Right Sub-Tree) => 1-1 = 0

H(Left Sub-Tree) = 1

H(Right Sub-Tree)=1



Balance (Node 2) = 1 - 1 = 0

H (node 2) = Max [H(Left Sub-Tree), H(Right Sub-Tree)] +1
=> 0+1 = 1

Balance (Node 6) : 1 - 1 = 0

H (node 6) = Max [H(Left Sub-Tree), H(Right Sub-Tree)] +1
=> 0+1 = 1

Balance (Nodes 1, 3) = 0

Balance (Nodes 5, 7) = 0

B (node) = H(Left Sub-Tree) - H(Right Sub-Tree)
-1-(-1) = 0

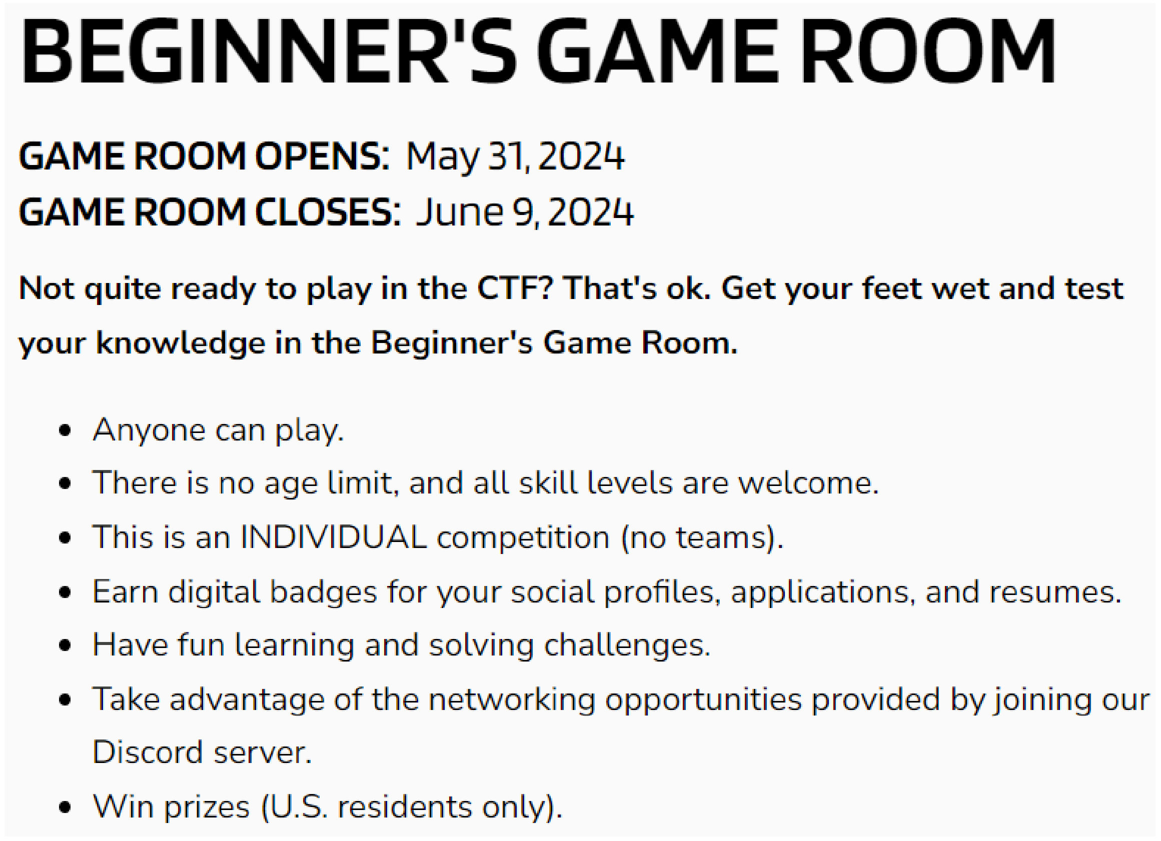
B (node) = H(Left Sub-Tree) - H(Right Sub-Tree)
-1-(-1) = 0

Height:
- H(null) = -1
- H(Single Node) = 0
- H (tree) = Max [H(Left Sub-Tree), H(Right Sub-Tree)] +1

- B (node) = H(Left Sub-Tree) - H(Right Sub-Tree)
- AVL Tree =$|B(node)| \leq 1$
- H (tree) = Max [H(Left Sub-Tree), H(Right Sub-Tree)] +1

SEASON IV: US Cyber Open

The US Cyber Games® Open is a Capture the Flag (CTF) competition that includes a Competitive CTF and Beginner's Game Room. (Link)

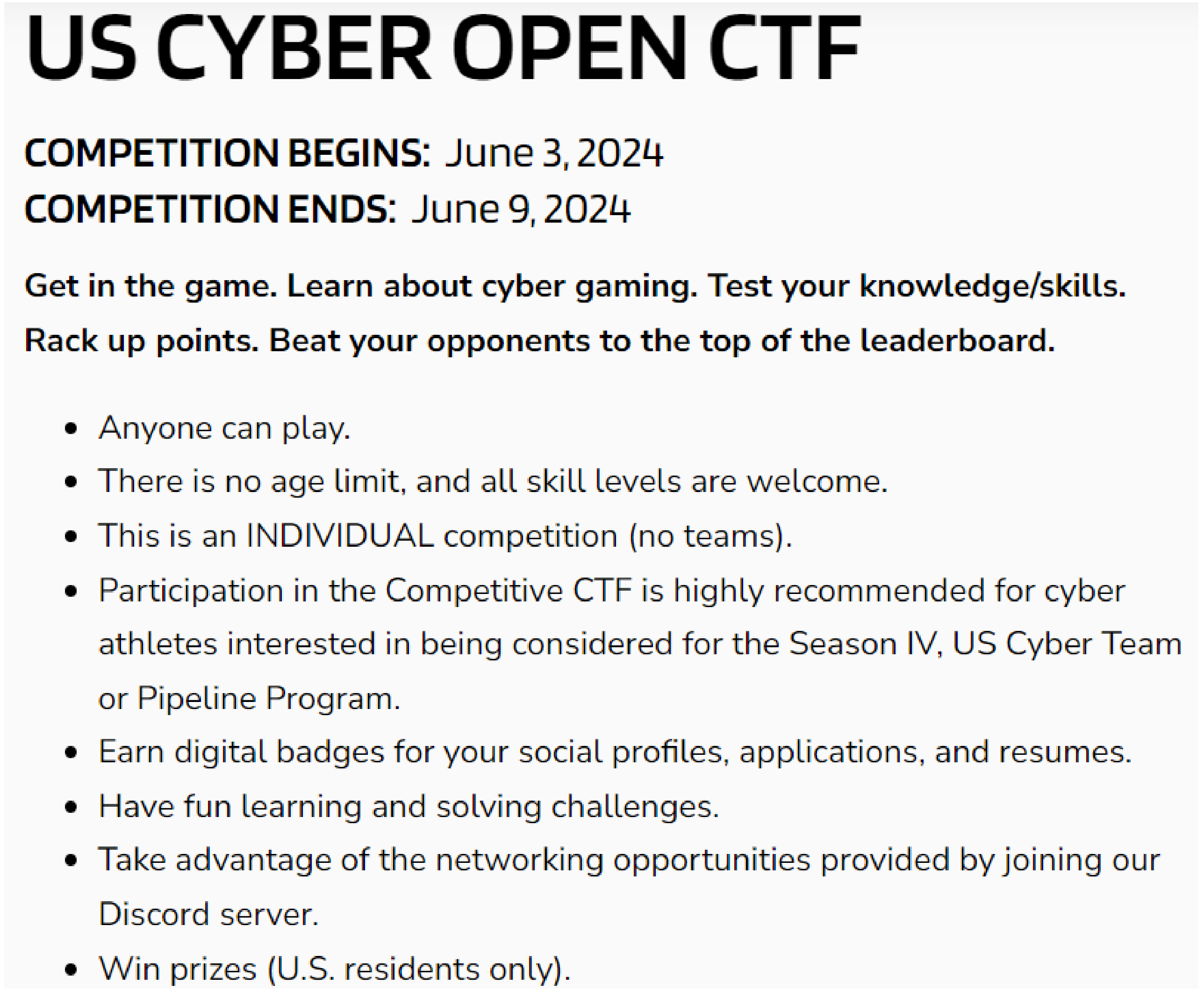


BEGINNER'S GAME ROOM

GAME ROOM OPENS: May 31, 2024
GAME ROOM CLOSES: June 9, 2024

Not quite ready to play in the CTF? That's ok. Get your feet wet and test your knowledge in the Beginner's Game Room.

- Anyone can play.
- There is no age limit, and all skill levels are welcome.
- This is an INDIVIDUAL competition (no teams).
- Earn digital badges for your social profiles, applications, and resumes.
- Have fun learning and solving challenges.
- Take advantage of the networking opportunities provided by joining our Discord server.
- Win prizes (U.S. residents only).



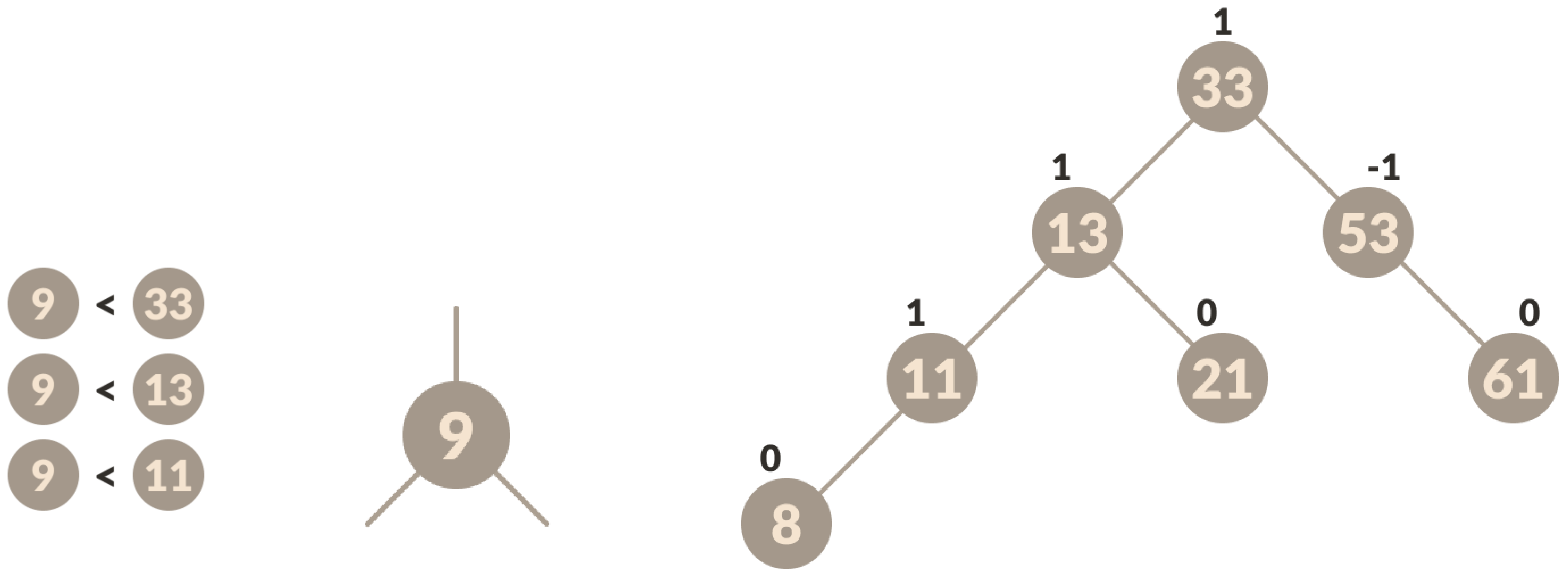
US CYBER OPEN CTF

COMPETITION BEGINS: June 3, 2024
COMPETITION ENDS: June 9, 2024

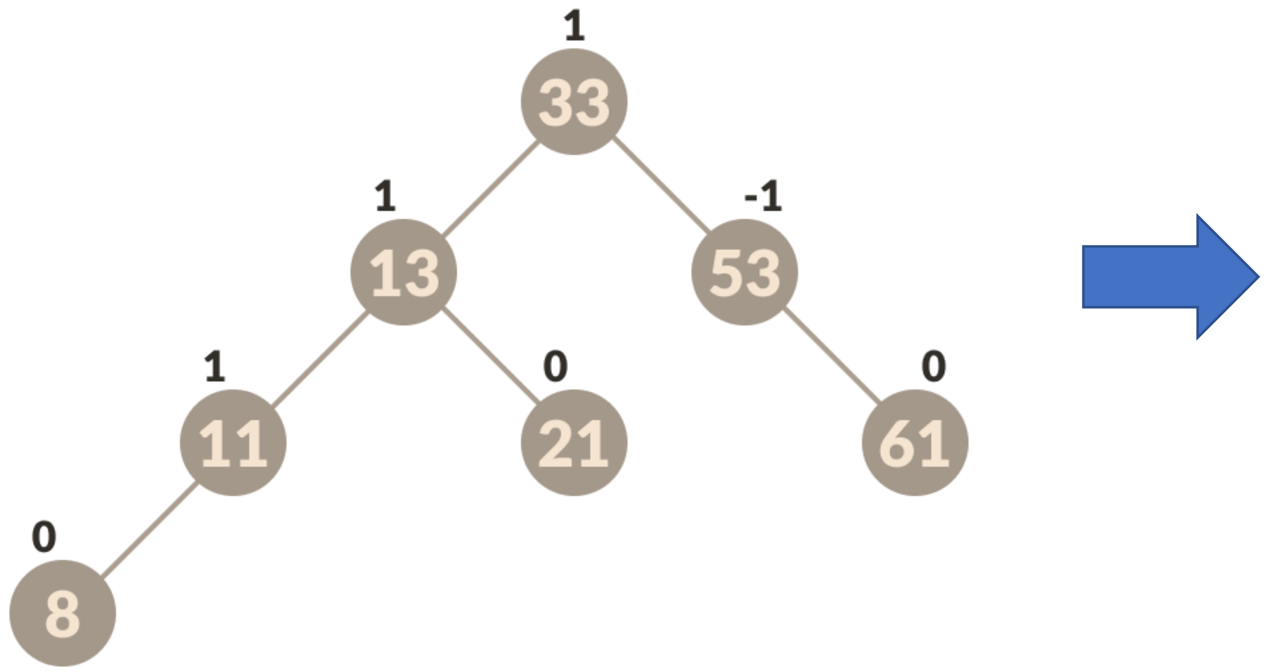
Get in the game. Learn about cyber gaming. Test your knowledge/skills. Rack up points. Beat your opponents to the top of the leaderboard.

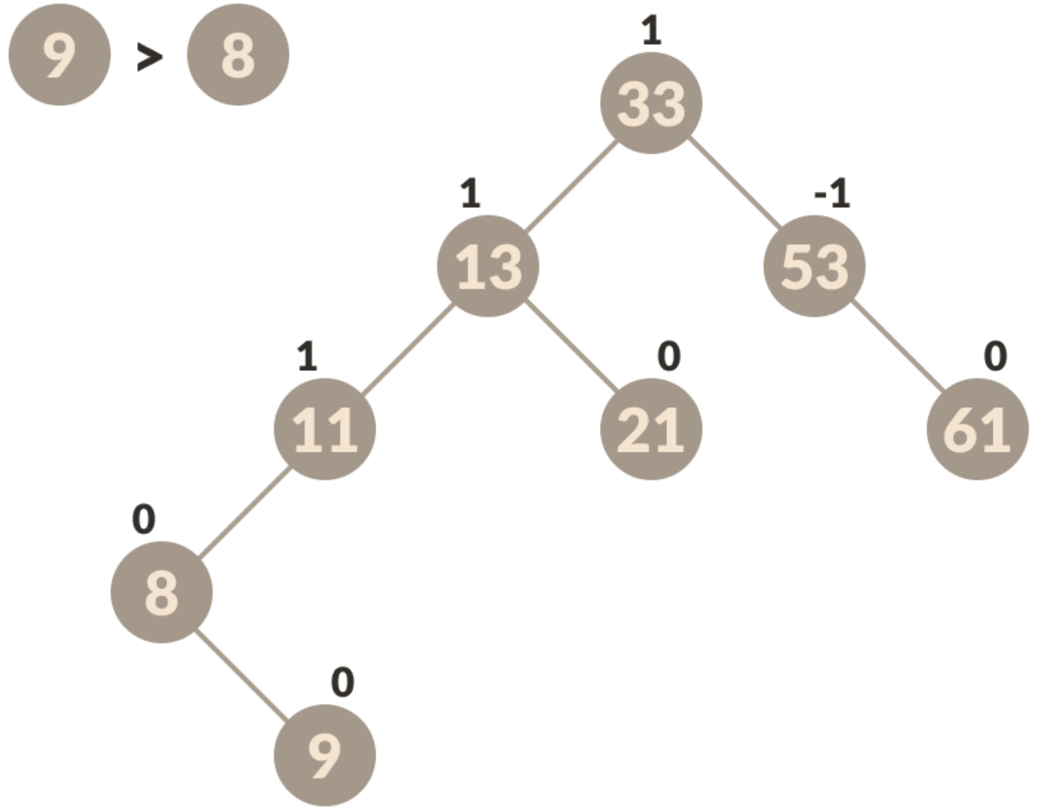
- Anyone can play.
- There is no age limit, and all skill levels are welcome.
- This is an INDIVIDUAL competition (no teams).
- Participation in the Competitive CTF is highly recommended for cyber athletes interested in being considered for the Season IV, US Cyber Team or Pipeline Program.
- Earn digital badges for your social profiles, applications, and resumes.
- Have fun learning and solving challenges.
- Take advantage of the networking opportunities provided by joining our Discord server.
- Win prizes (U.S. residents only).

AVL Tree (Insertion)



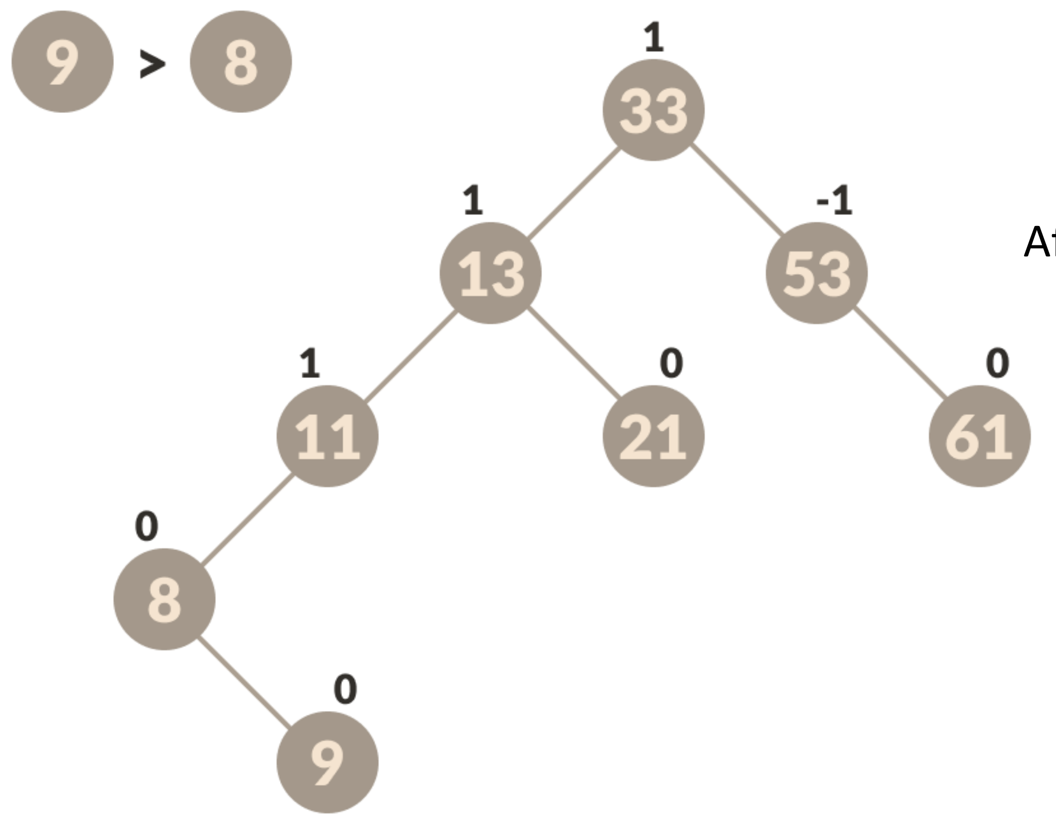
AVL Tree (Insertion)



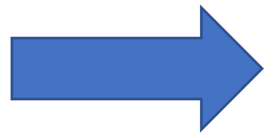


After attaching node 9, update tree balance

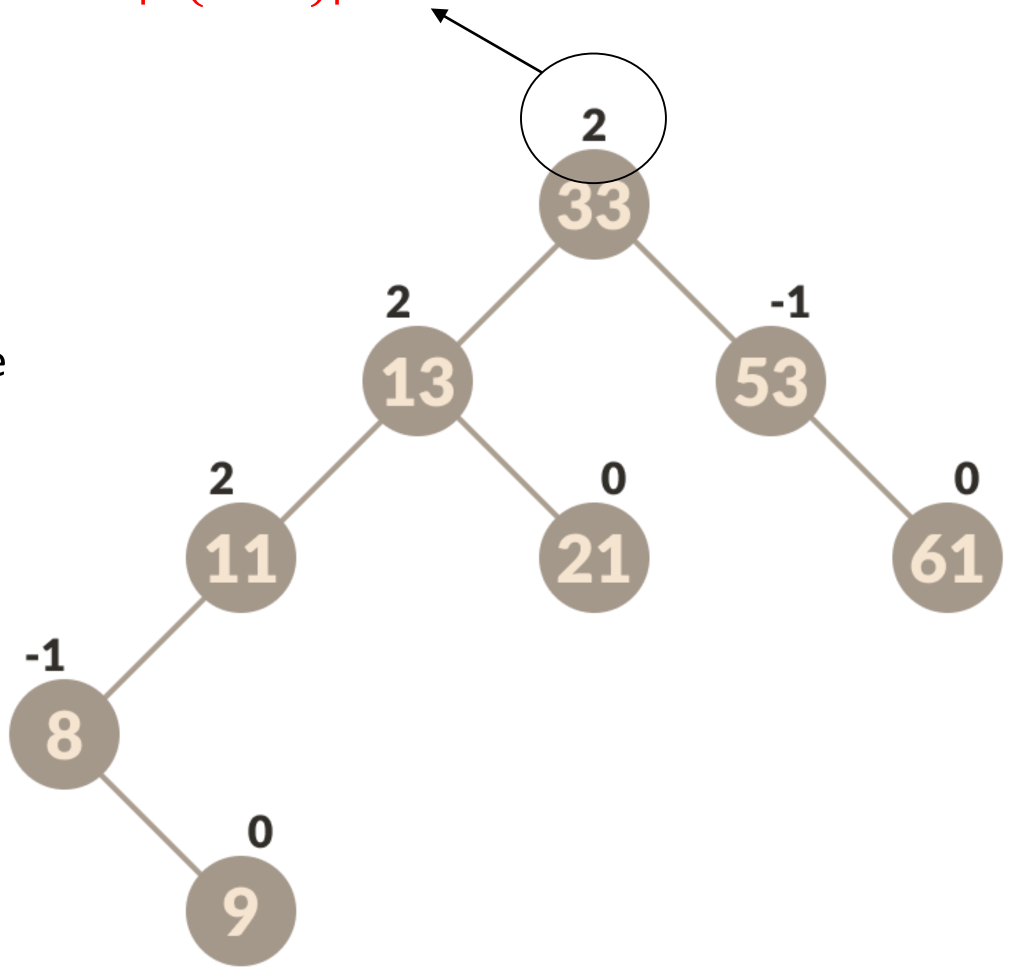
AVL Tree (Insertion)



After balance update

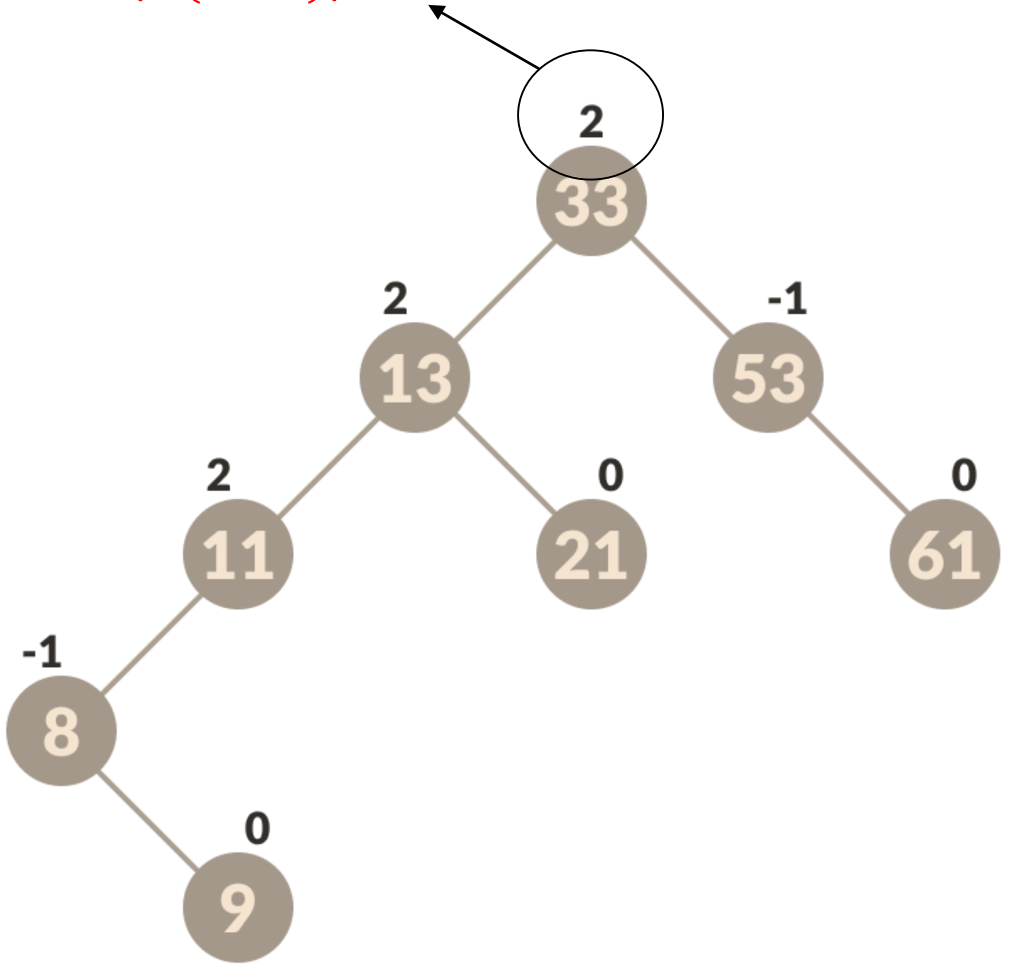


AVL Tree $= |B(node)| \leq 1$



AVL Tree (Insertion)

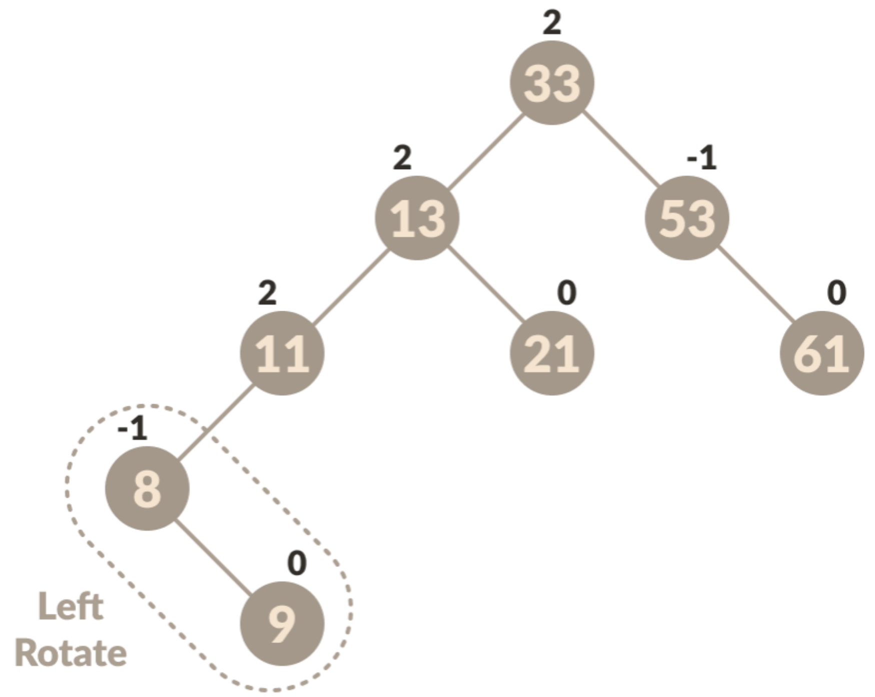
AVL Tree $=|B(node)| \leq 1$

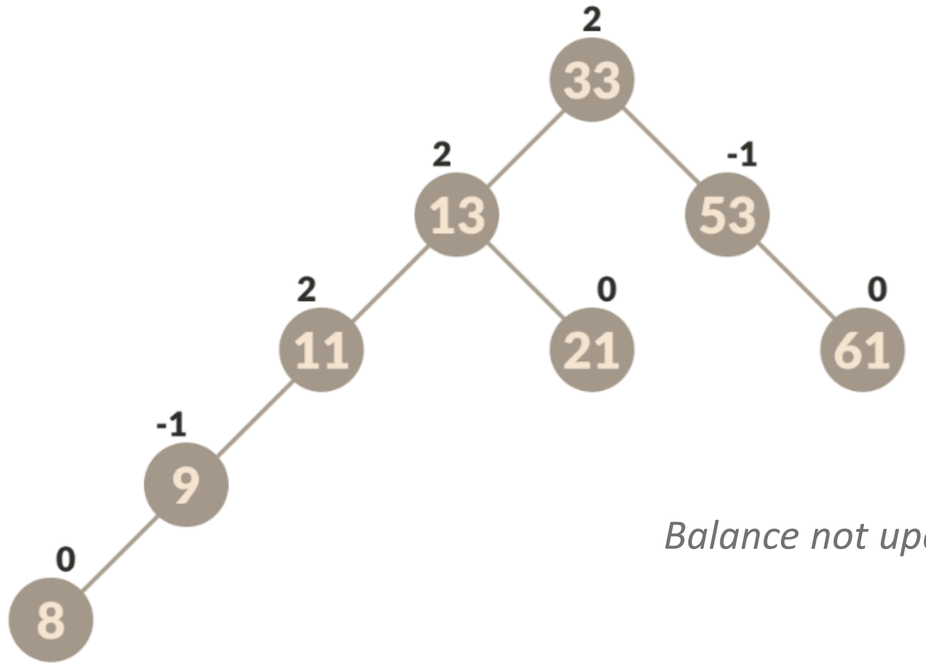


- If the nodes are unbalanced, then rebalance the node.
- If B(node) > 1, it means the height of the left subtree is greater than that of the right subtree.
- So, do a right rotation or left-right rotation
  - If newNodeKey < leftChildKey do right rotation.
    - Node 8 do not have a left node
  - Else, do left-right rotation

AVL Tree (Insertion)

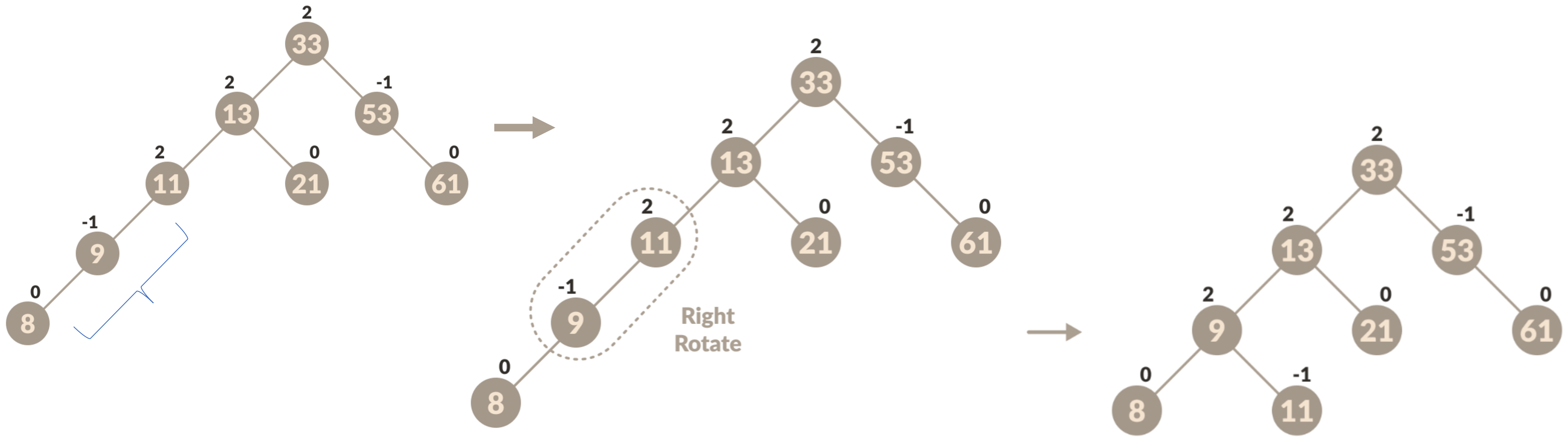
- If the nodes are unbalanced, then rebalance the node.
- If B(node) > 1, it means the height of the left subtree is greater than that of the right subtree.
- So, do a right rotation or left-right rotation
    - If newNodeKey < leftChildKey do right rotation.
    - Else, do left-right rotation





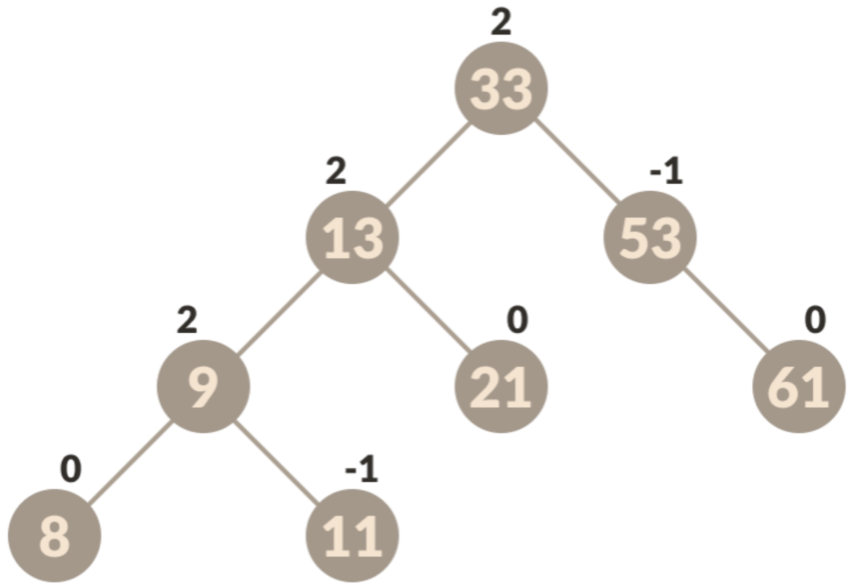
Balance not updated

AVL Tree (Insertion)



Balance not updated

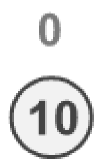
AVL Tree
(Insertion)



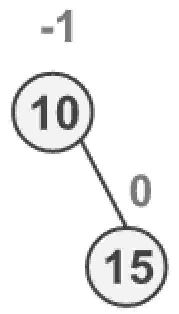
After balance update

AVL Tree (Insertion)

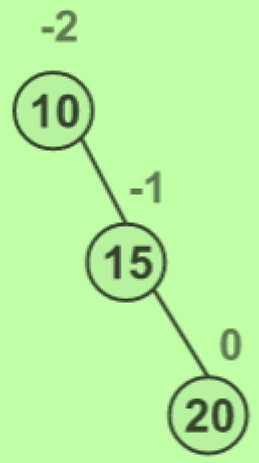
Array = {10, 15, 20, 9, 5, 16, 17, 8, 6}



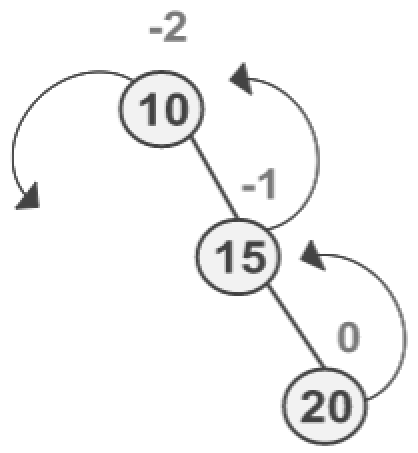
Tree is balanced



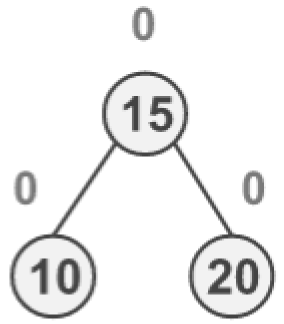
Tree is balanced



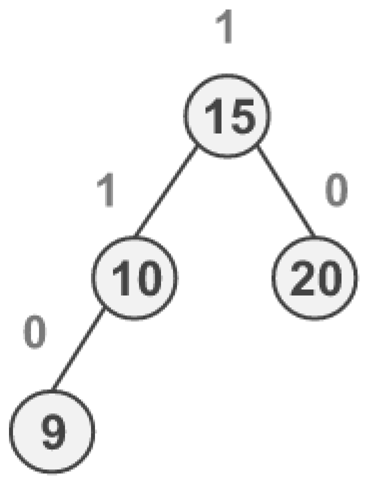
Tree is imbalanced



LL Rotation



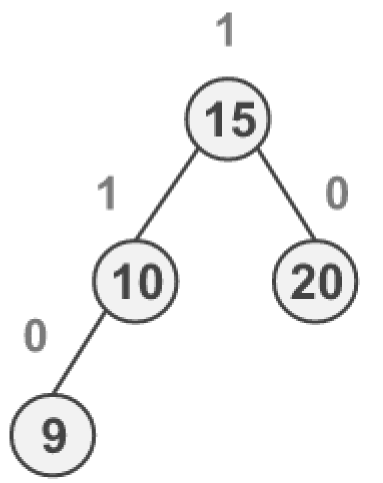
Tree is Balanced



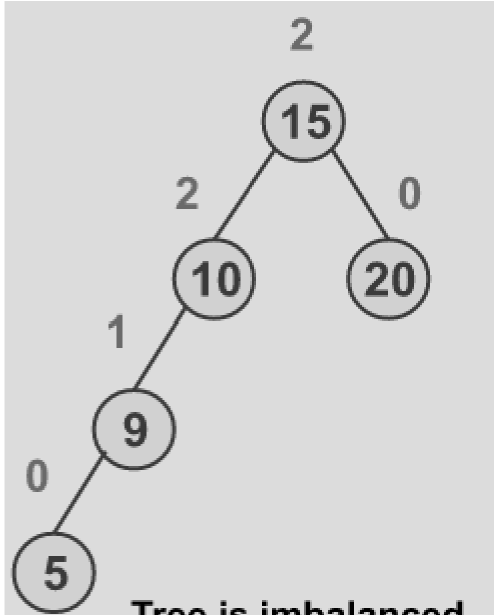
Tree is Balanced

AVL Tree (Insertion)

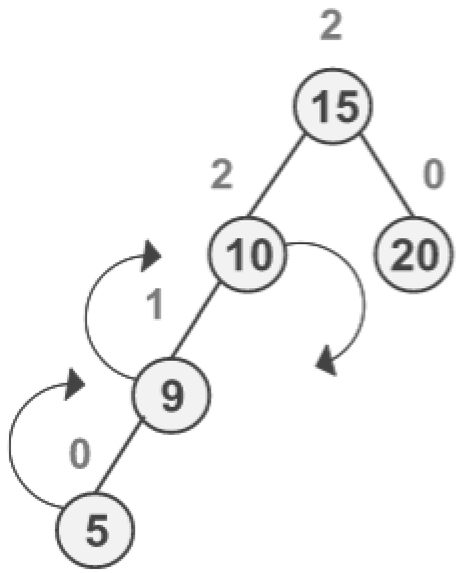
Array = {10, 15, 20, 9, 5, 16, 17, 8, 6}



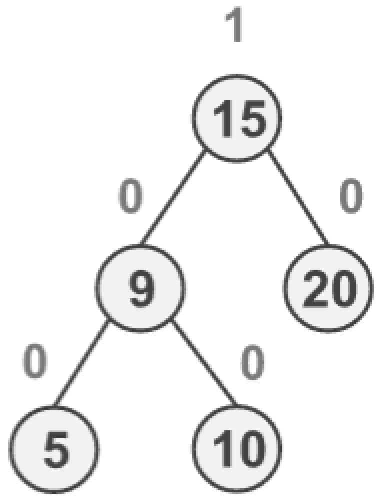
Tree is Balanced



Tree is imbalanced



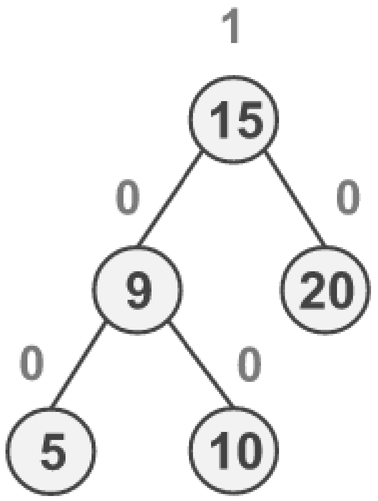
RR Rotation



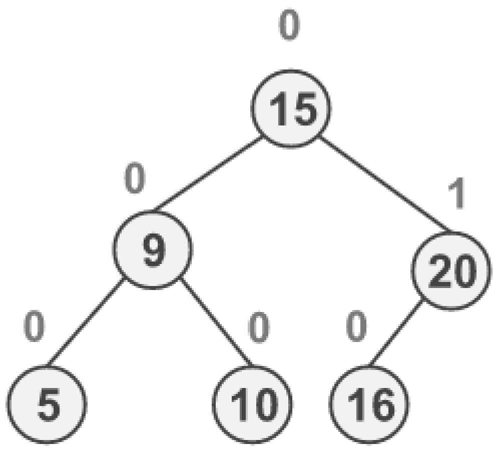
Tree is balanced

AVL Tree
(Insertion)

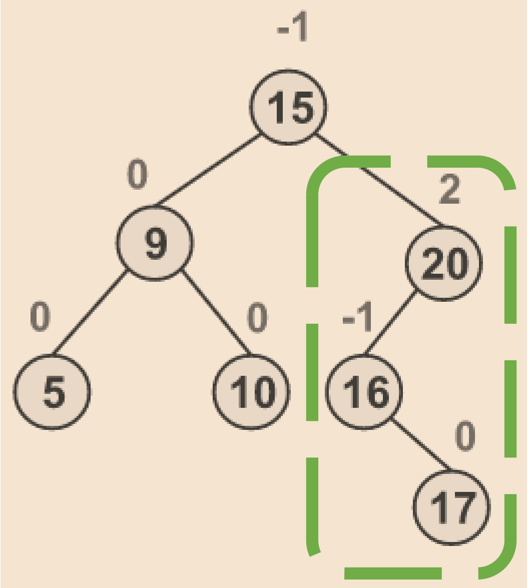
Array = {10, 15, 20, 9, 5, 16, 17, 8, 6}



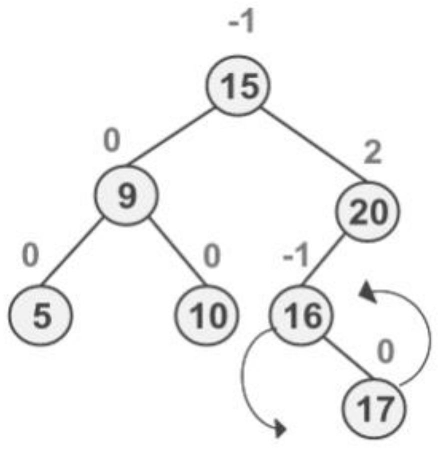
Tree is balanced



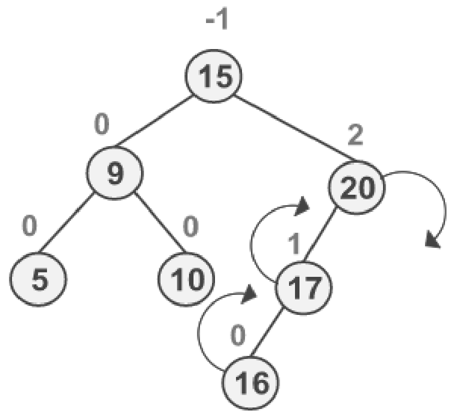
Tree is balanced



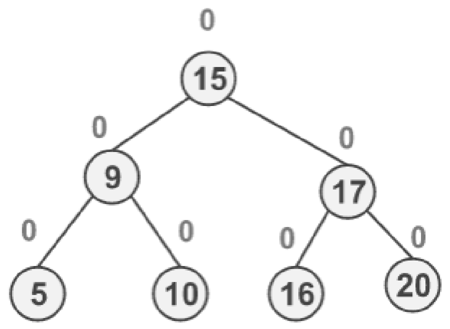
Tree is imbalanced



LL Rotation



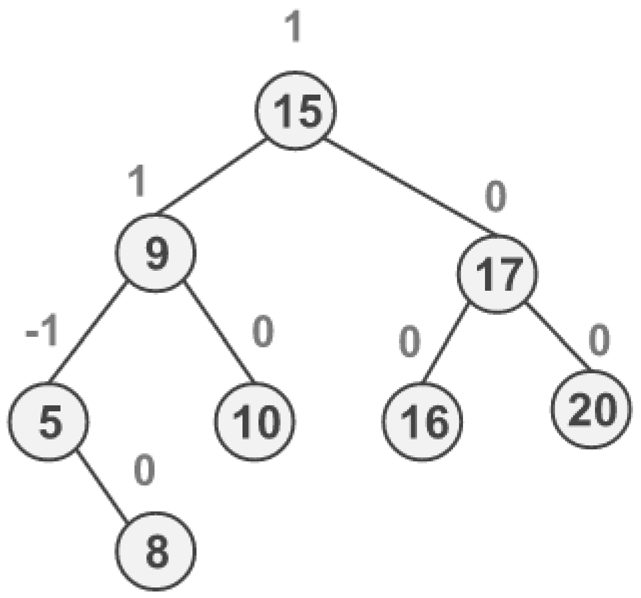
RR Rotation



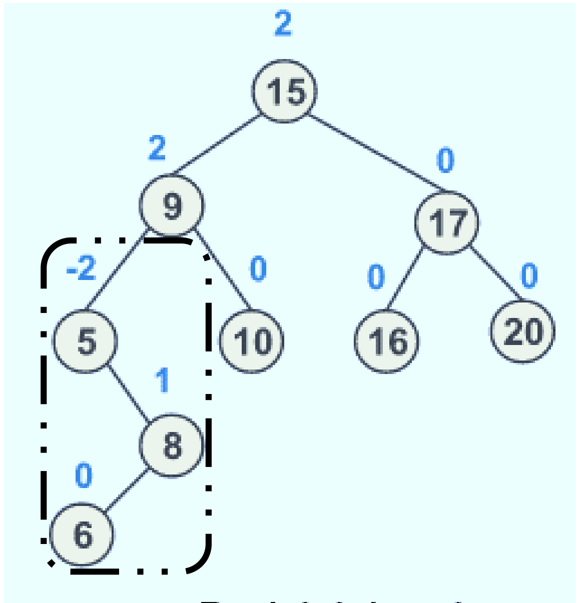
Tree is Balanced

AVL Tree
(Insertion)

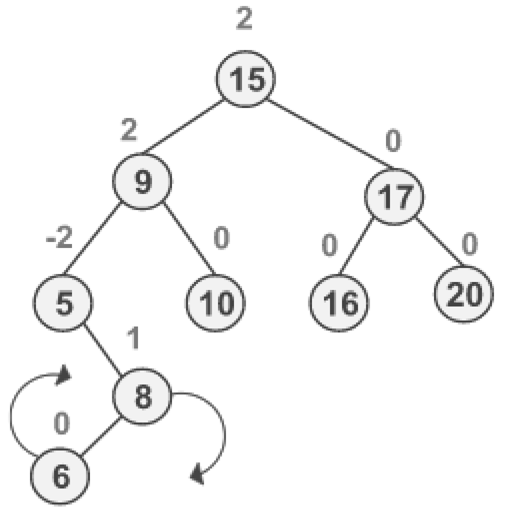
Array = {10, 15, 20, 9, 5, 16, 17, 8, 6}



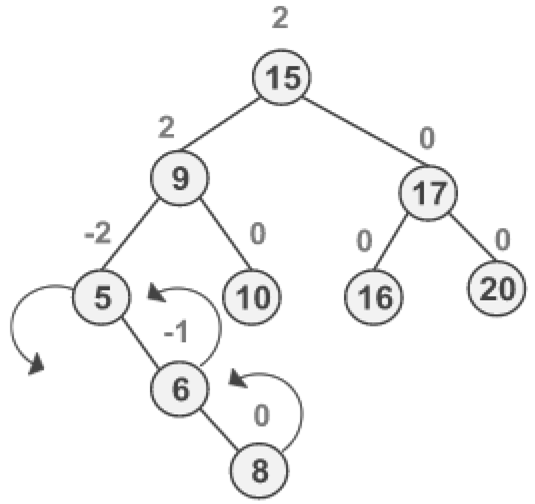
Tree is Balanced



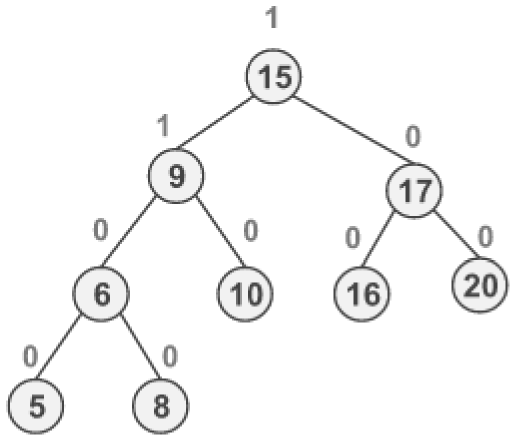
Tree is imbalanced



RR Rotation

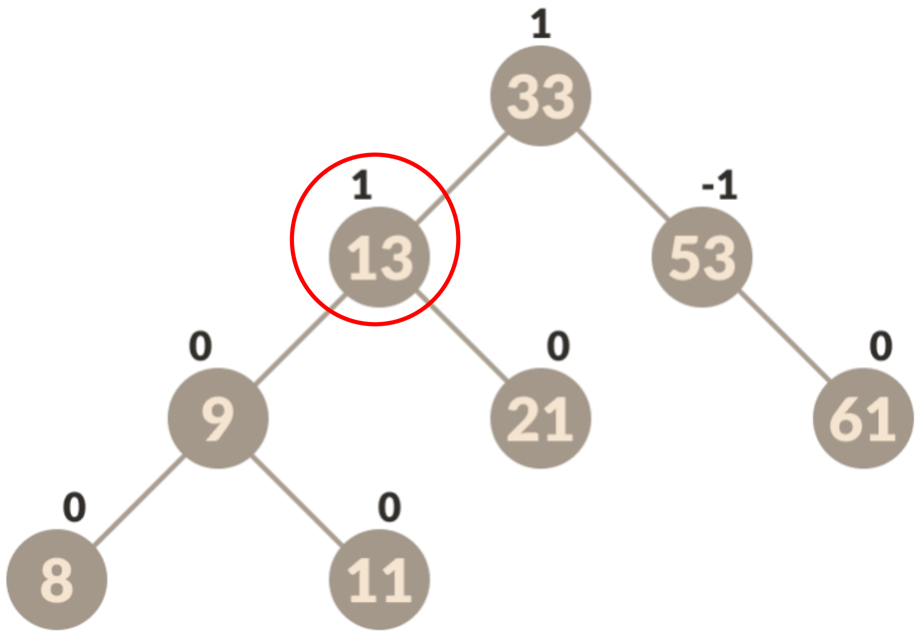


LL Rotation



Tree is balanced

AVL Tree (Deletion)



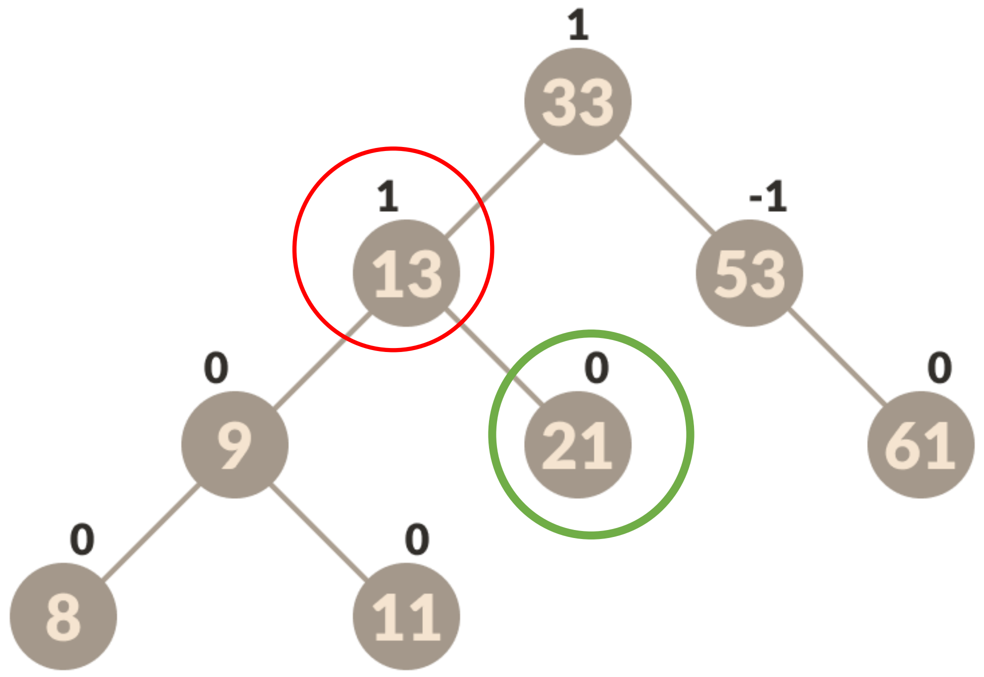
There are three cases for deleting a node:

Case 1
If nodeToBeDeleted is the leaf node (i.e. does not have any child), then remove nodeToBeDeleted.

Case 2
If nodeToBeDeleted has one child, then substitute the contents of nodeToBeDeleted with that of the child. Remove the child.

Case 3
If nodeToBeDeleted has two children, find the inorder successor of nodeToBeDeleted (i.e. node with a minimum value of key in the right subtree).

AVL Tree (Deletion)



1. If nodeToBeDeleted has two children, find the inorder successor of nodeToBeDeleted

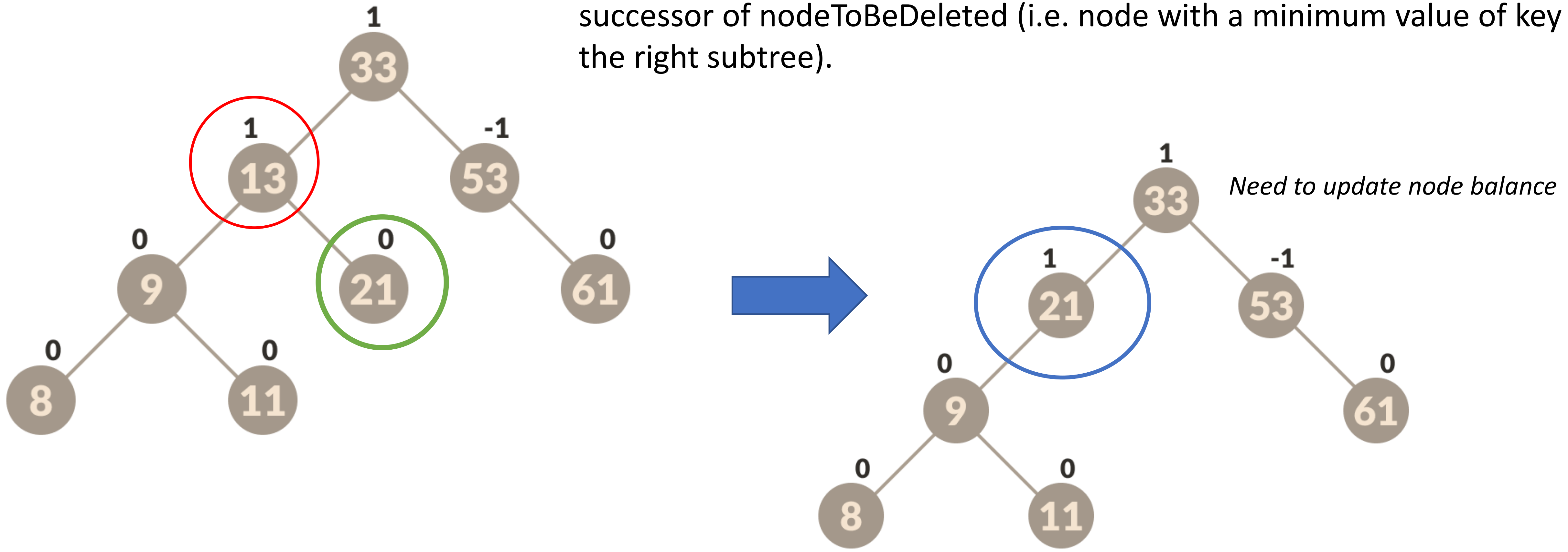
(i.e. node with a minimum value of key in the right subtree).

In-order

1. Left Subtree,
2. Root,
3. Right Subtree

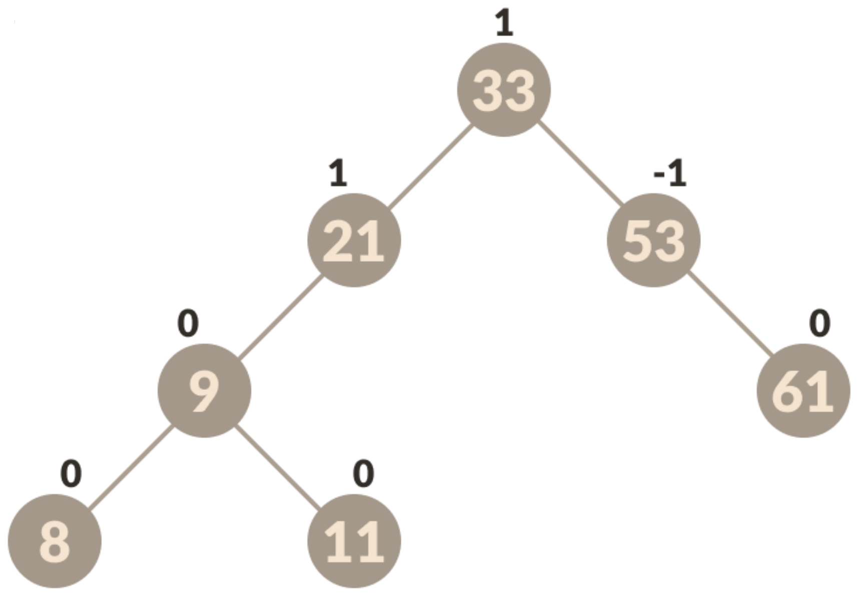
AVL Tree (Deletion)

1. If nodeToBeDeleted has two children, find the inorder successor of nodeToBeDeleted (i.e. node with a minimum value of key in the right subtree).

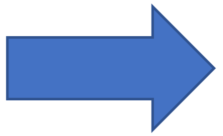


AVL Tree (Deletion)

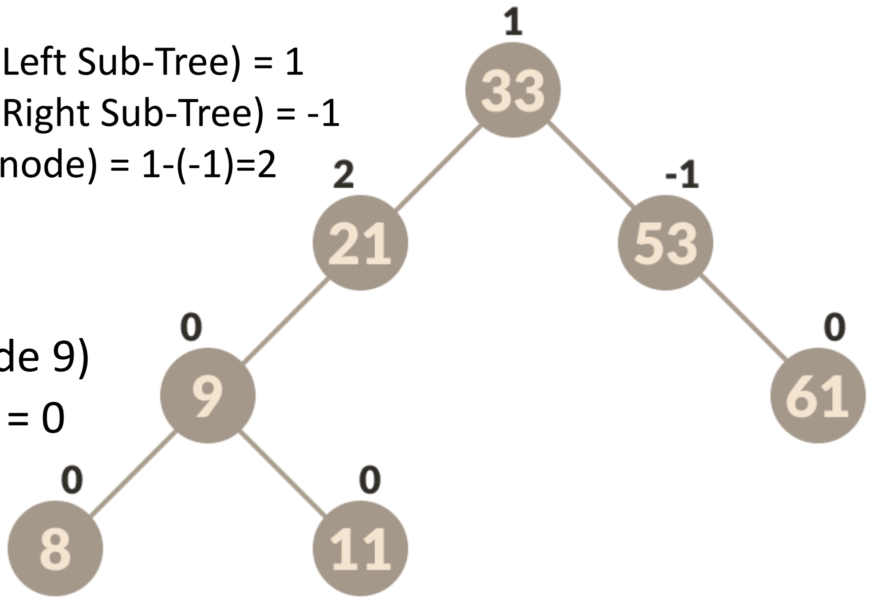
- B (node) = H(Left Sub-Tree) - H(Right Sub-Tree)
- AVL Tree =$|B(node)| \leq 1$
- H (tree) = Max [H(Left Sub-Tree), H(Right Sub-Tree)] +1



Height:
- H(null) = -1
- H(Single Node) = 0
- H (tree) = Max [H(Left Sub-Tree), H(Right Sub-Tree)] +1



- H(Left Sub-Tree) = 1
- H(Right Sub-Tree) = -1
- B(node) = 1-(-1)=2

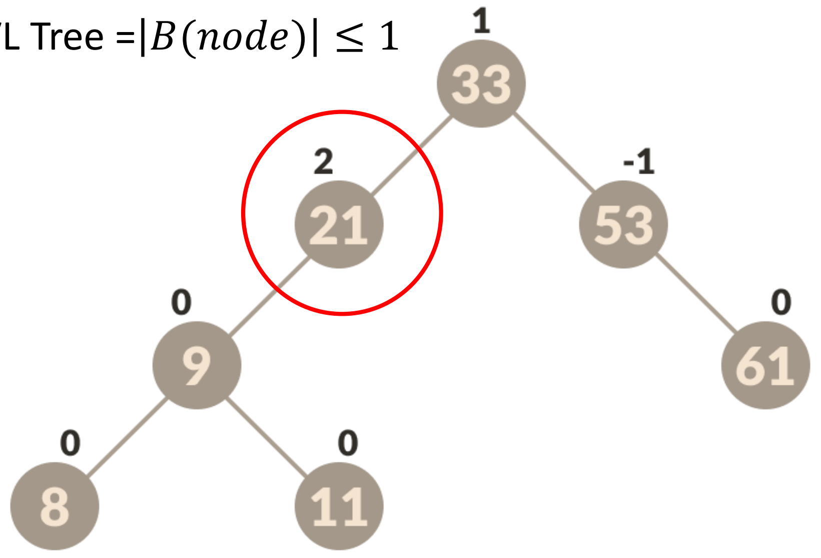


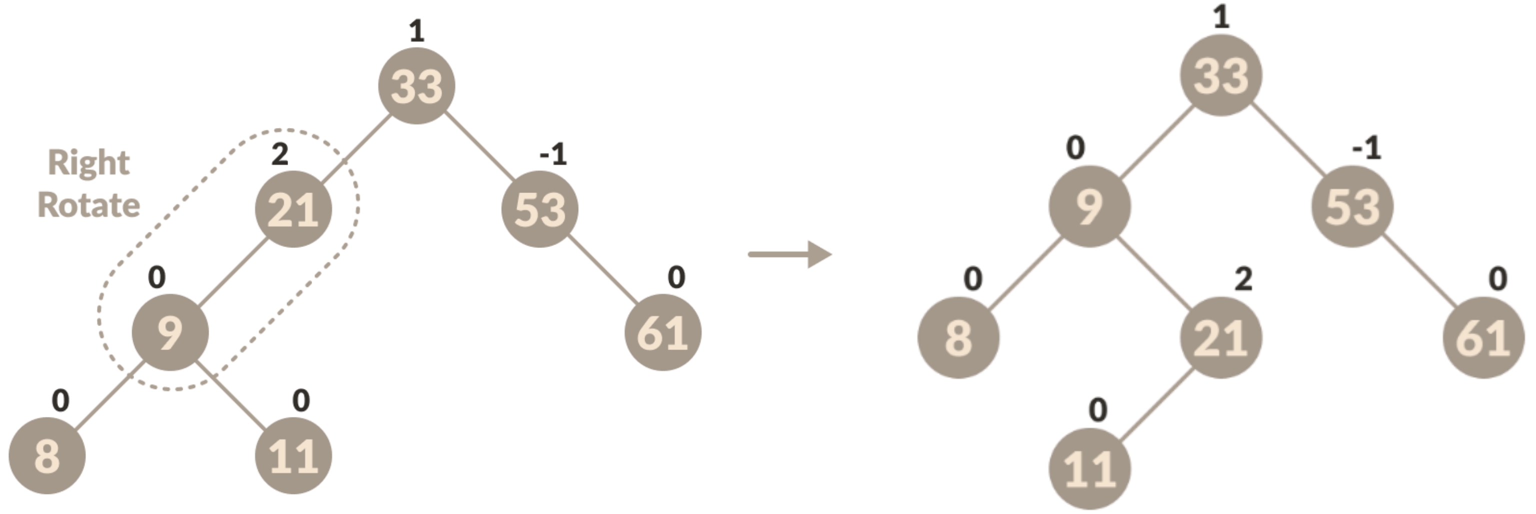
B (Node 9)
0 - 0 = 0

B (Nodes 8 & 11) = 0
-1-(-1) = 0

AVL Tree (Deletion)

AVL Tree =$|B(node)| \leq 1$



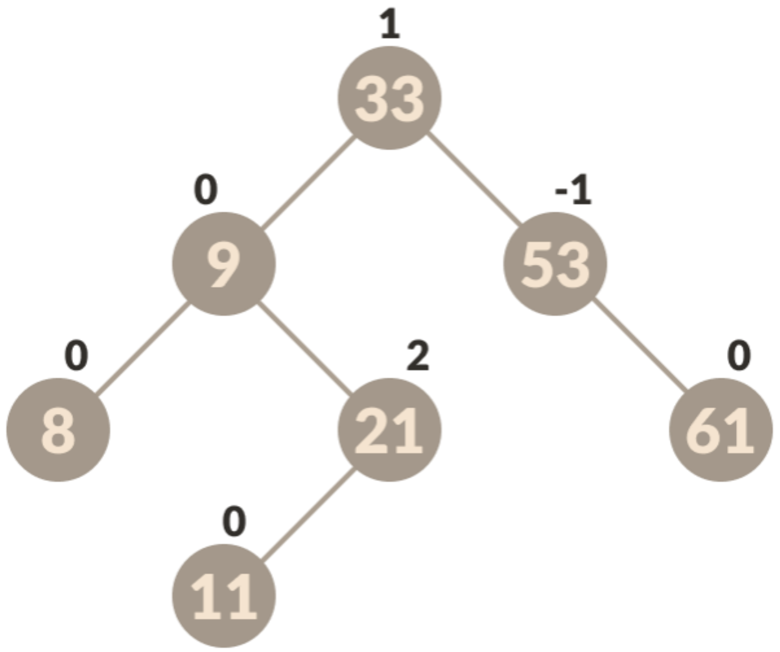


Rebalance the tree if the balance factor of any of the nodes is not equal to -1, 0 or 1.

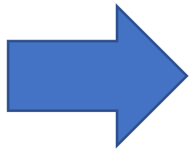
- If balanceFactor of currentNode > 1,
- If balanceFactor of leftChild >= 0, do right rotation
- Else do left-right rotation.

AVL Tree

(Deletion - This slide will not be on canvas)



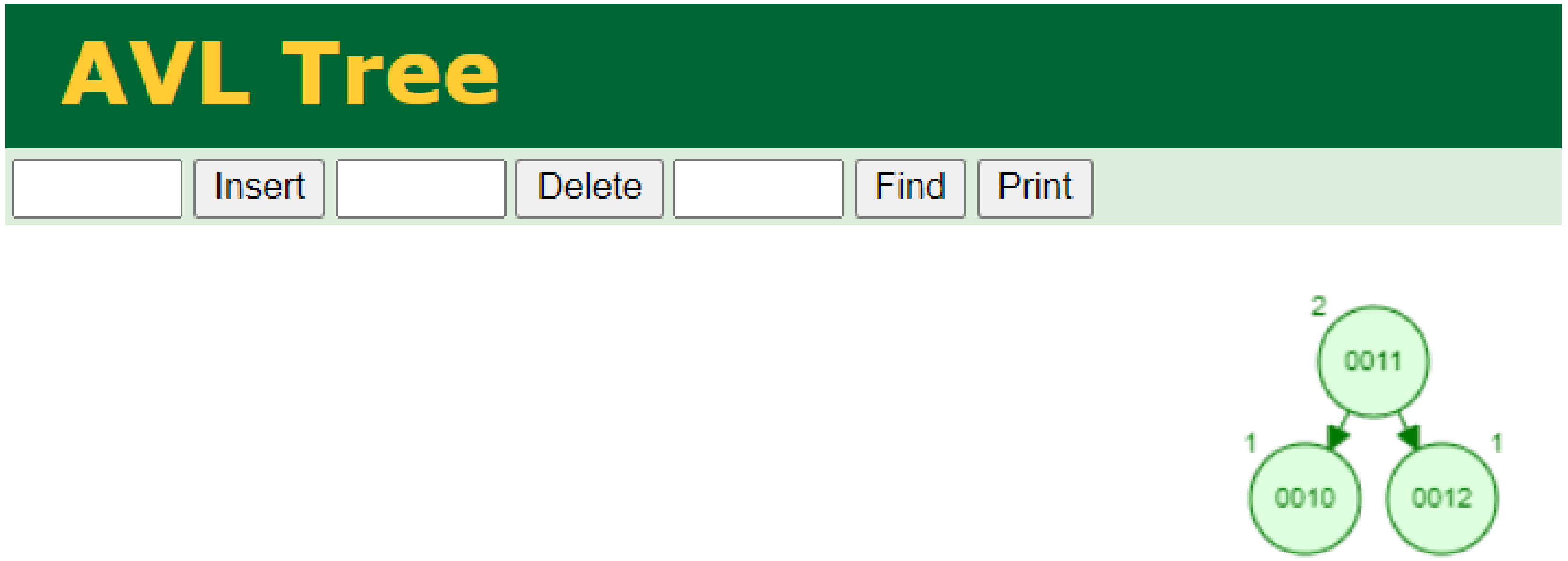
Need to update node balance



- If balanceFactor of currentNode < -1,
    - If balanceFactor of rightChild <= 0, do left rotation.
    - Else do right-left rotation.

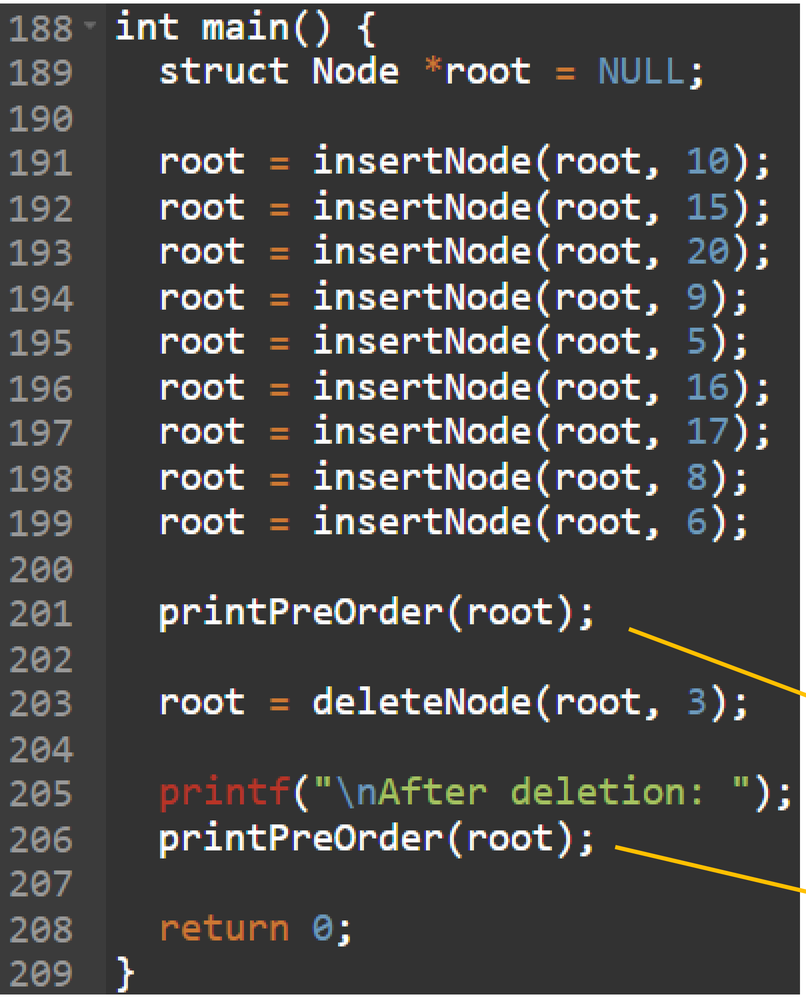
AVL Tree

- Input = 10, 11, 12

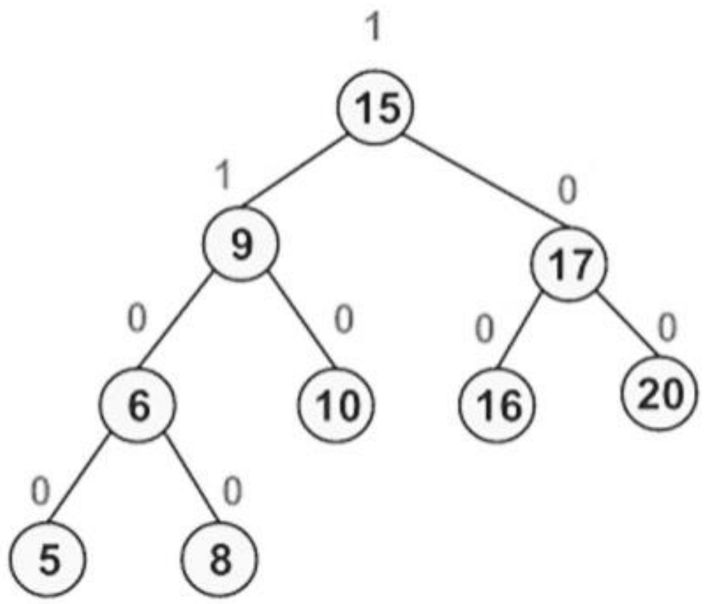


Source

AVL Tree

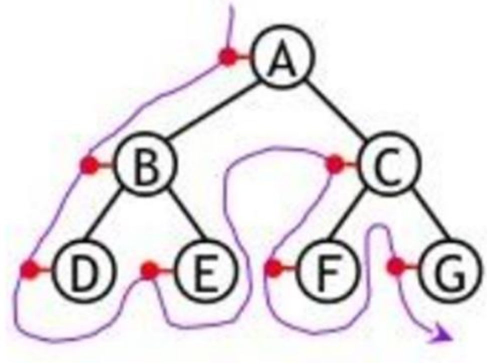


```c
int main() {
    struct Node *root = NULL;

    root = insertNode(root, 10);
    root = insertNode(root, 15);
    root = insertNode(root, 20);
    root = insertNode(root, 9);
    root = insertNode(root, 5);
    root = insertNode(root, 16);
    root = insertNode(root, 17);
    root = insertNode(root, 8);
    root = insertNode(root, 6);

    printPreOrder(root);

    root = deleteNode(root, 3);

    printf("\nAfter deletion: ");
    printPreOrder(root);

    return 0;
}
```

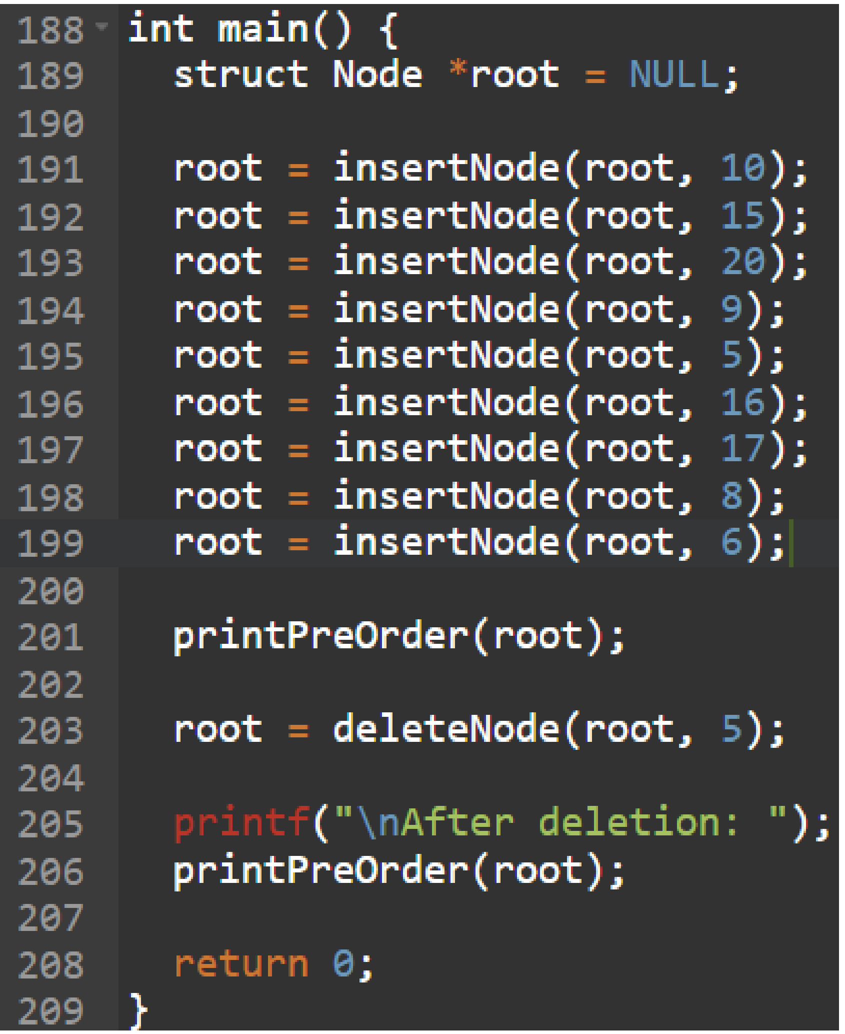


Pre-order

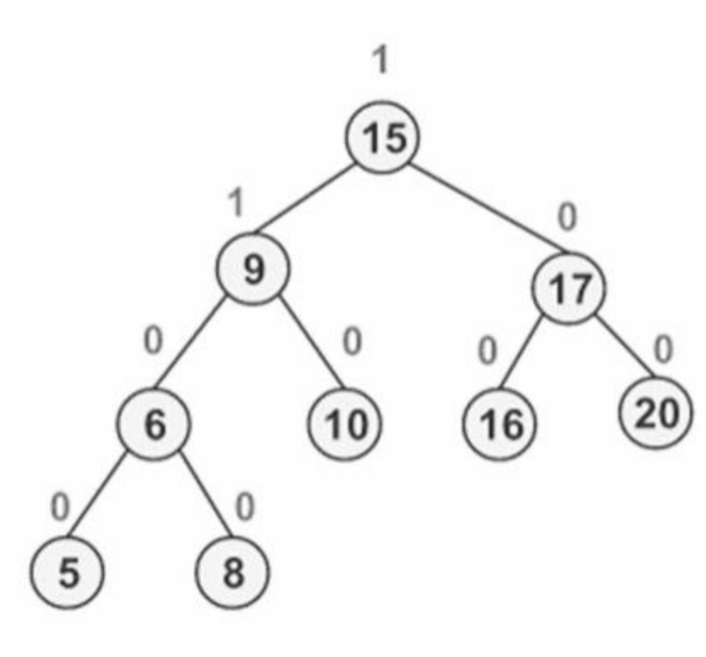
1. Root,
2. Left Subtree,
3. Right Subtree



AVL Tree

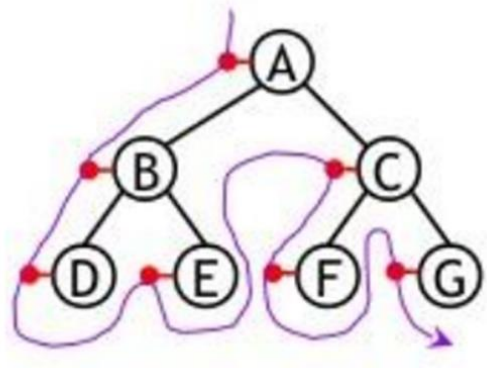


```
int main() {
   struct Node *root = NULL;

   root = insertNode(root, 10);
   root = insertNode(root, 15);
   root = insertNode(root, 20);
   root = insertNode(root, 9);
   root = insertNode(root, 5);
   root = insertNode(root, 16);
   root = insertNode(root, 17);
   root = insertNode(root, 8);
   root = insertNode(root, 6);

   printPreOrder(root);

   root = deleteNode(root, 5);

   printf("\nAfter deletion: ");
   printPreOrder(root);

   return 0;
}
```

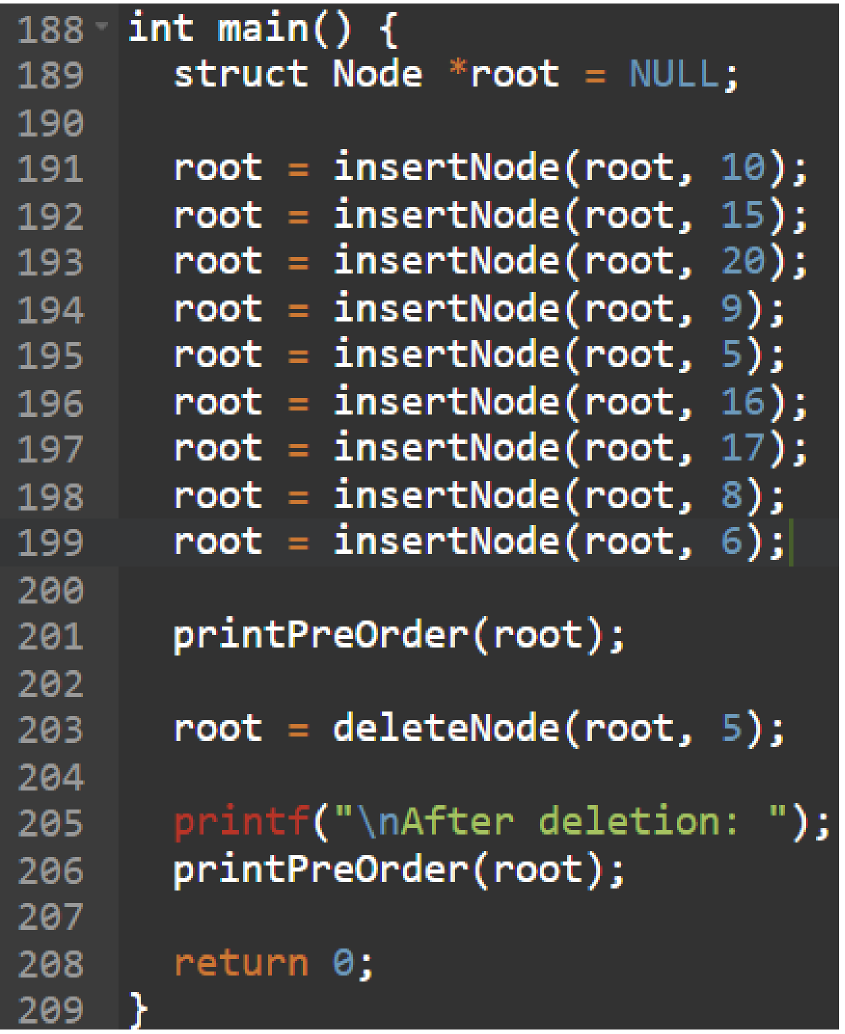


Pre-order

1. Root,
2. Left Subtree,
3. Right Subtree



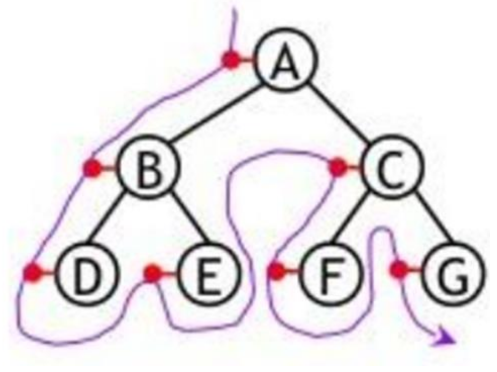
AVL Tree



```
int main() {
    struct Node *root = NULL;

    root = insertNode(root, 10);
    root = insertNode(root, 15);
    root = insertNode(root, 20);
    root = insertNode(root, 9);
    root = insertNode(root, 5);
    root = insertNode(root, 16);
    root = insertNode(root, 17);
    root = insertNode(root, 8);
    root = insertNode(root, 6);

    printPreOrder(root);

    root = deleteNode(root, 5);

    printf("\nAfter deletion: ");
    printPreOrder(root);

    return 0;
}
```

Pre-order

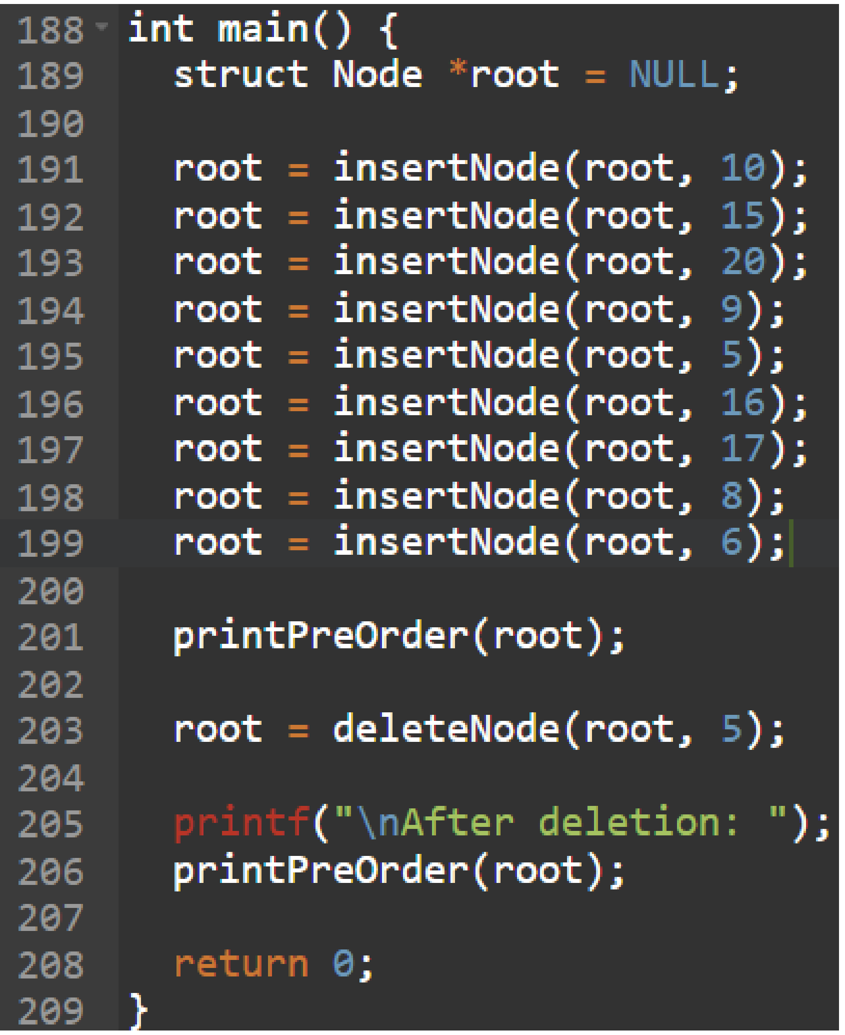
1. Root,
2. Left Subtree,
3. Right Subtree



- B (node) = H(Left Sub-Tree) - H(Right Sub-Tree)
- AVL Tree =$|B(node)| \leq 1$
- H (tree) = Max [H(Left Sub-Tree), H(Right Sub-Tree)] +1

- H(null) = -1
- H(Single Node) = 0

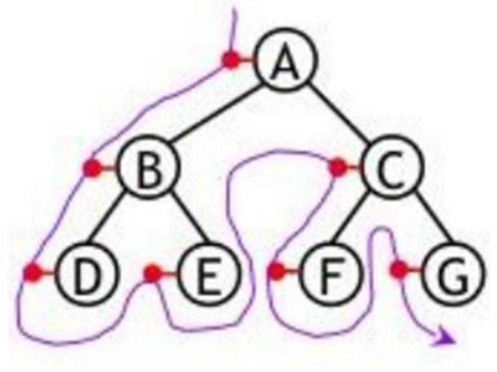
AVL Tree



```c
int main() {
    struct Node *root = NULL;

    root = insertNode(root, 10);
    root = insertNode(root, 15);
    root = insertNode(root, 20);
    root = insertNode(root, 9);
    root = insertNode(root, 5);
    root = insertNode(root, 16);
    root = insertNode(root, 17);
    root = insertNode(root, 8);
    root = insertNode(root, 6);

    printPreOrder(root);

    root = deleteNode(root, 5);

    printf("\nAfter deletion: ");
    printPreOrder(root);

    return 0;
}
```

Pre-order

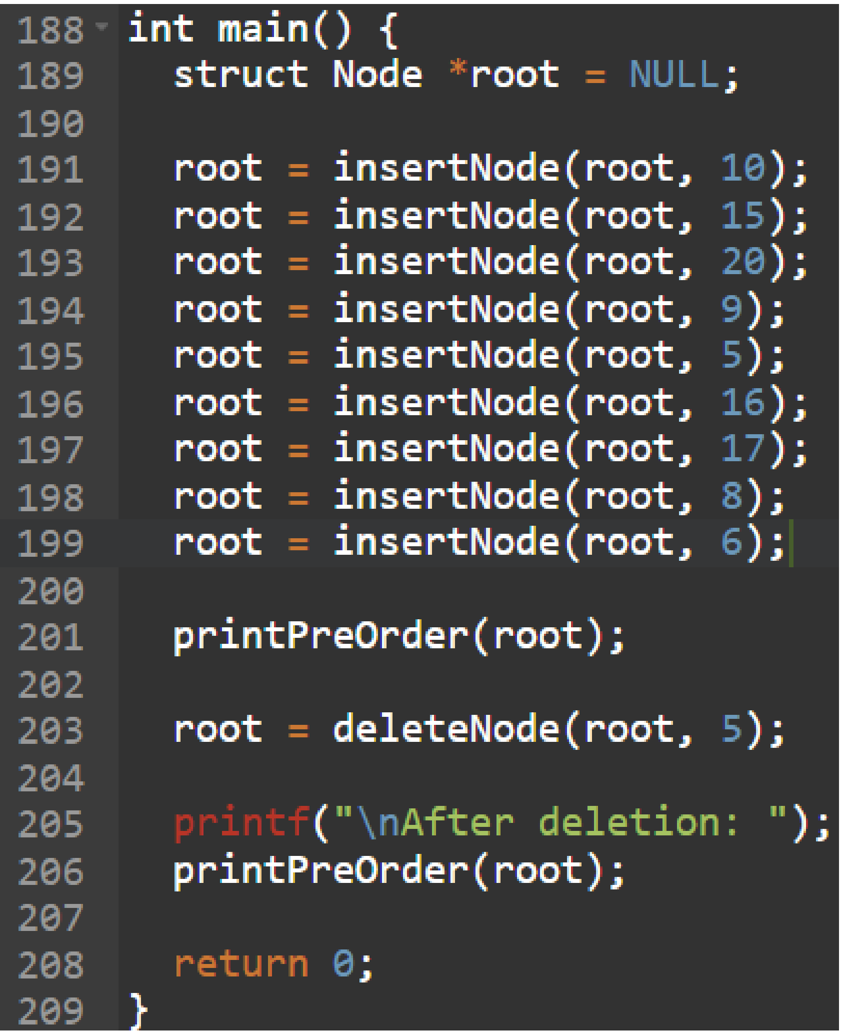
1. Root,
2. Left Subtree,
3. Right Subtree



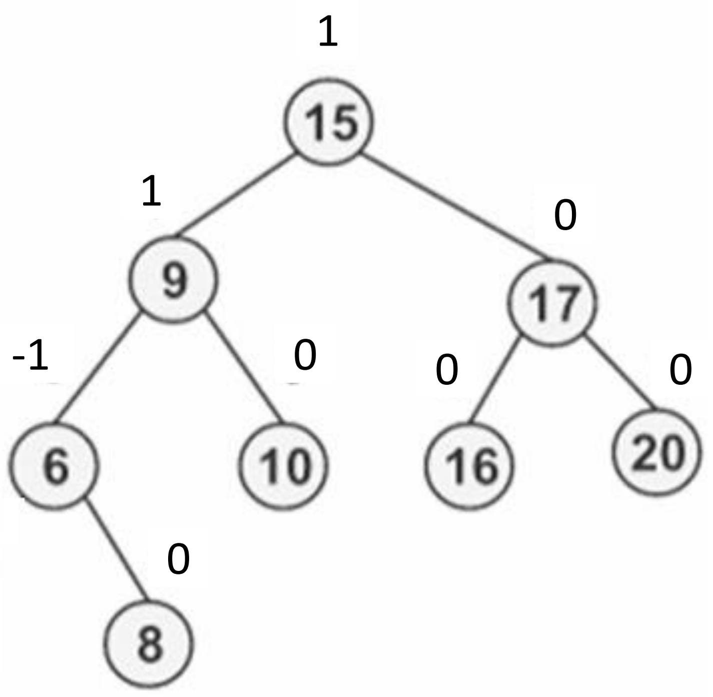
- B (node) = H(Left Sub-Tree) - H(Right Sub-Tree)
- AVL Tree =$|B(node)| \leq 1$
- H (tree) = Max [H(Left Sub-Tree), H(Right Sub-Tree)] +1

- H(null) = -1
- H(Single Node) = 0

AVL Tree

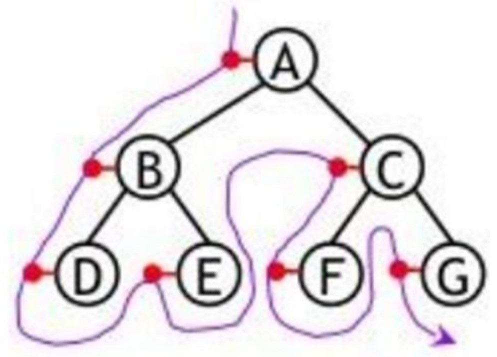


```c
int main() {
    struct Node *root = NULL;

    root = insertNode(root, 10);
    root = insertNode(root, 15);
    root = insertNode(root, 20);
    root = insertNode(root, 9);
    root = insertNode(root, 5);
    root = insertNode(root, 16);
    root = insertNode(root, 17);
    root = insertNode(root, 8);
    root = insertNode(root, 6);

    printPreOrder(root);

    root = deleteNode(root, 5);

    printf("\nAfter deletion: ");
    printPreOrder(root);

    return 0;
}
```



Pre-order

1. Root,
2. Left Subtree,
3. Right Subtree

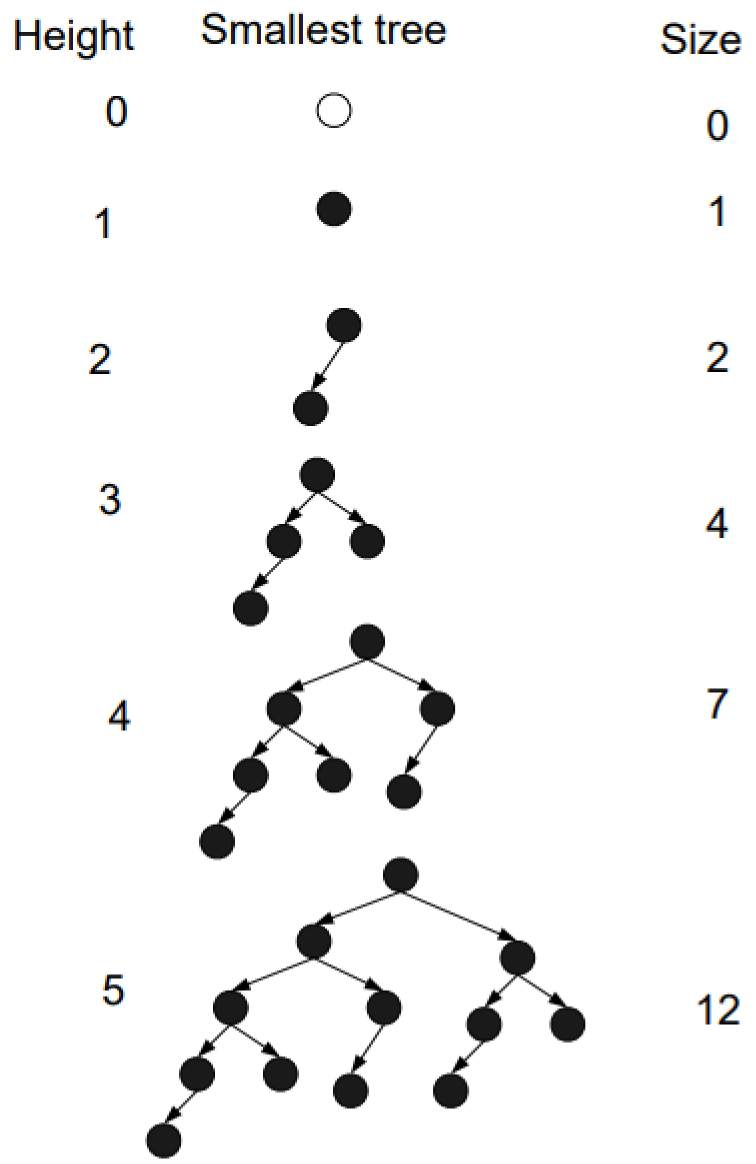


- B (node) = H(Left Sub-Tree) - H(Right Sub-Tree)
- AVL Tree =$|B(node)| \leq 1$
- H (tree) = Max [H(Left Sub-Tree), H(Right Sub-Tree)] +1

- H(null) = -1
- H(Single Node) = 0

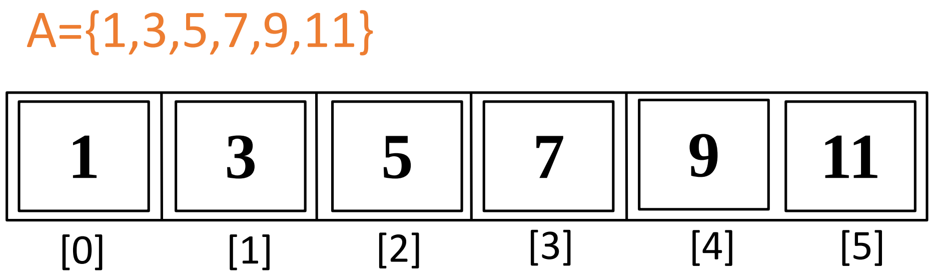
AVL Tree (Application)

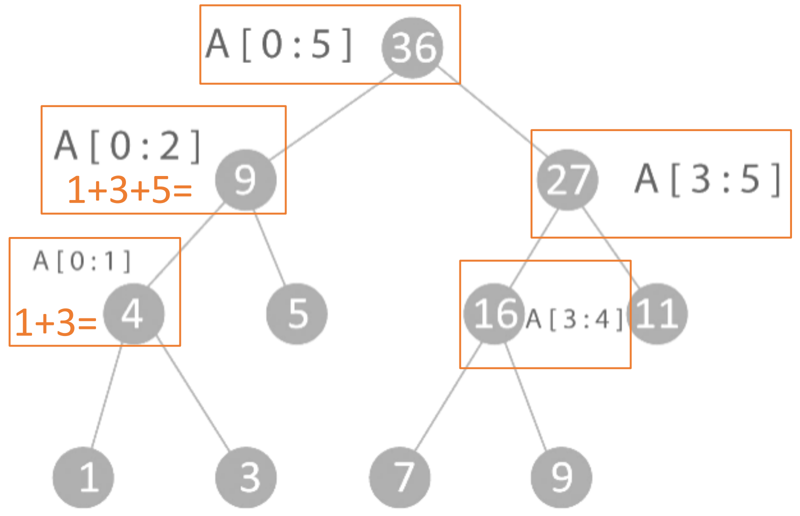
- Access large amounts of data quickly.
  - Amount of information is proportional to size of tree and access time is proportional to the height of the tree.
- From Discrete Math. for Engineering:
  - When working with Fibonacci sequence AVL tress can optimize the time to access data.
- AVL trees are particularly used for look up intensive applications i.e. used for indexing large records in database to improve search
- AVL Trees are used for all sorts of in-memory collections such as sets and dictionaries



Segment Tree

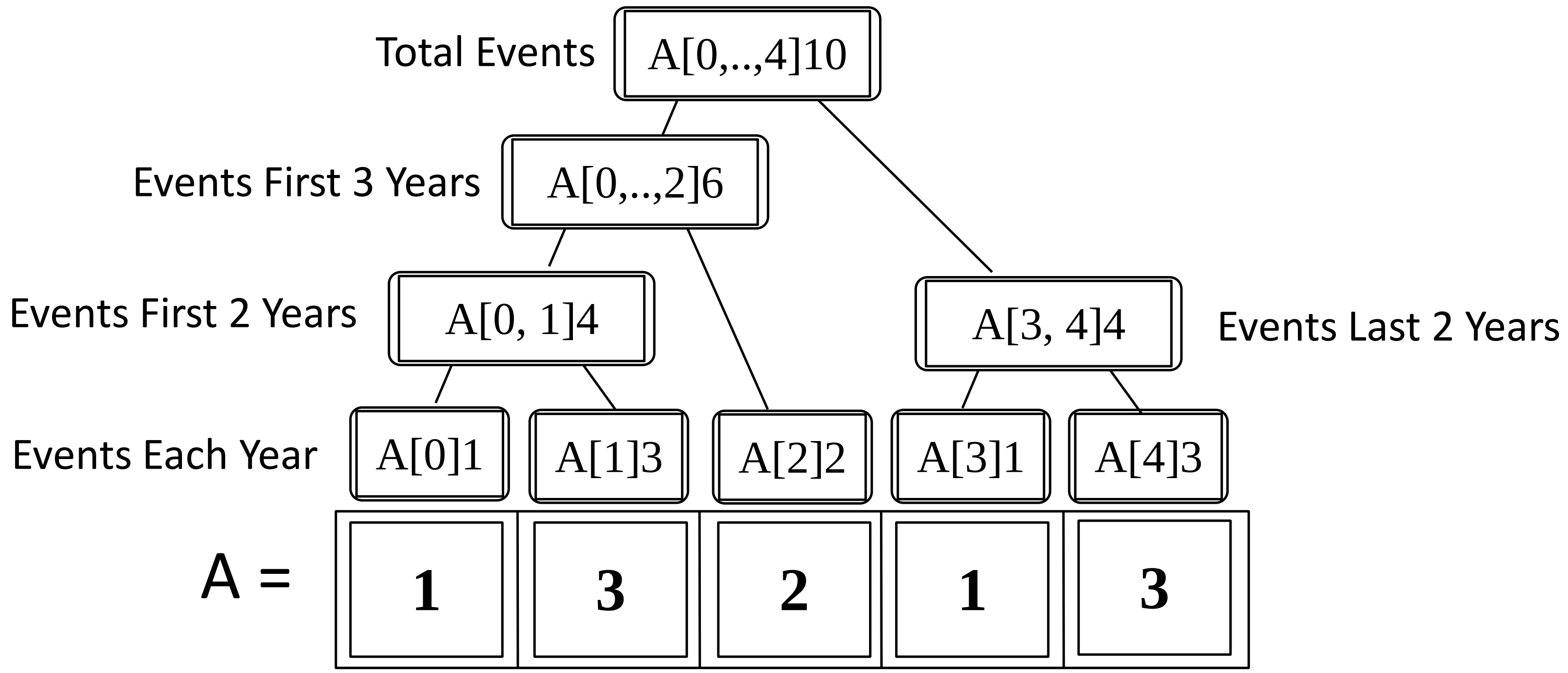
- A Segment Tree is a data structure that stores information about array intervals/segments as a tree.
- This allows answering range queries over an array efficiently, while still being flexible enough to allow quick modification of the array.
- This includes finding the sum of consecutive array elements or finding the minimum element in a such a range in O(log n) time.
- Between answering such queries, the Segment Tree allows modifying the array by replacing one element, or even changing the elements of a whole subsegment (e.g. assigning all elements to any value, or adding a value to all element in the subsegment).





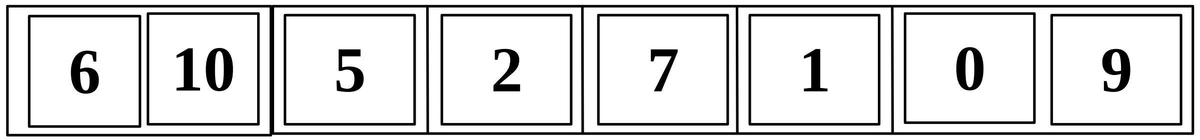
Segment Tree

- We want to compute sum of events over a range to time
- Each Array index represent 1 year
- Index [0] = Year 2000 and data/value represent events organized by students.



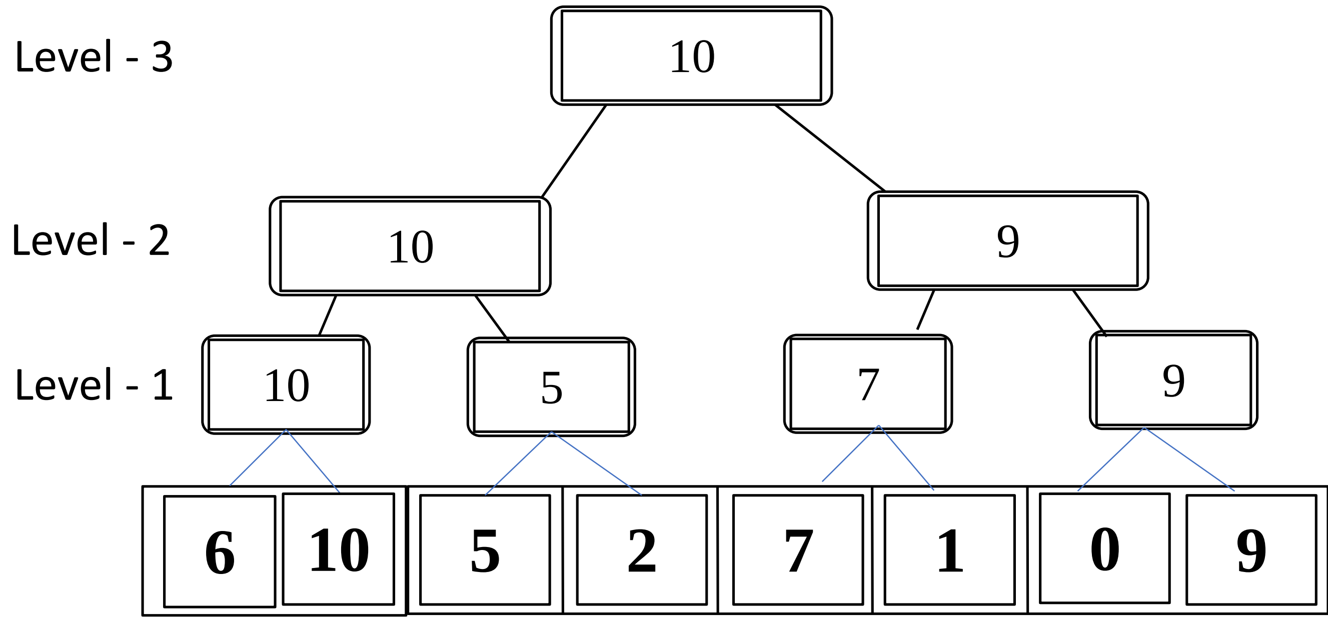
Segment Tree

- We want to Find the Max value in this Array
  - Issue: Array is not sorted
  - We might need an iterative approach to search the Max value
  - Try using Segment Tree



Segment Tree
(This slide will not be on canvas)

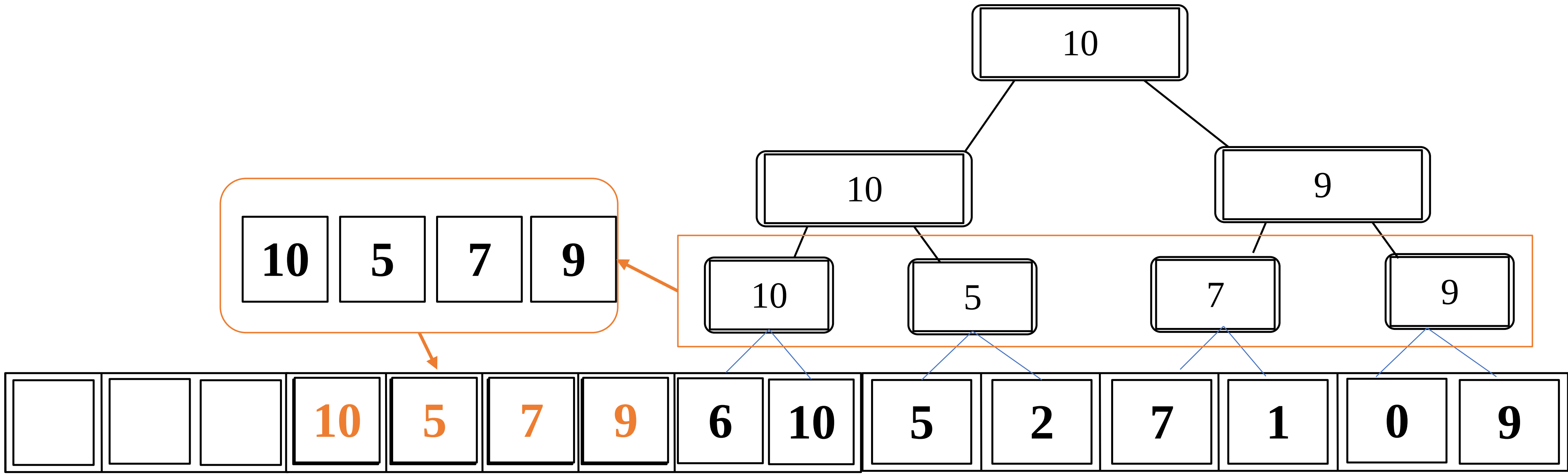
- We want to Find the Max value using Segment Tree
- Start by comparing neighbors



Segment Tree

(Storing The Max Value Tree In Array) - (This slide will not be on canvas)

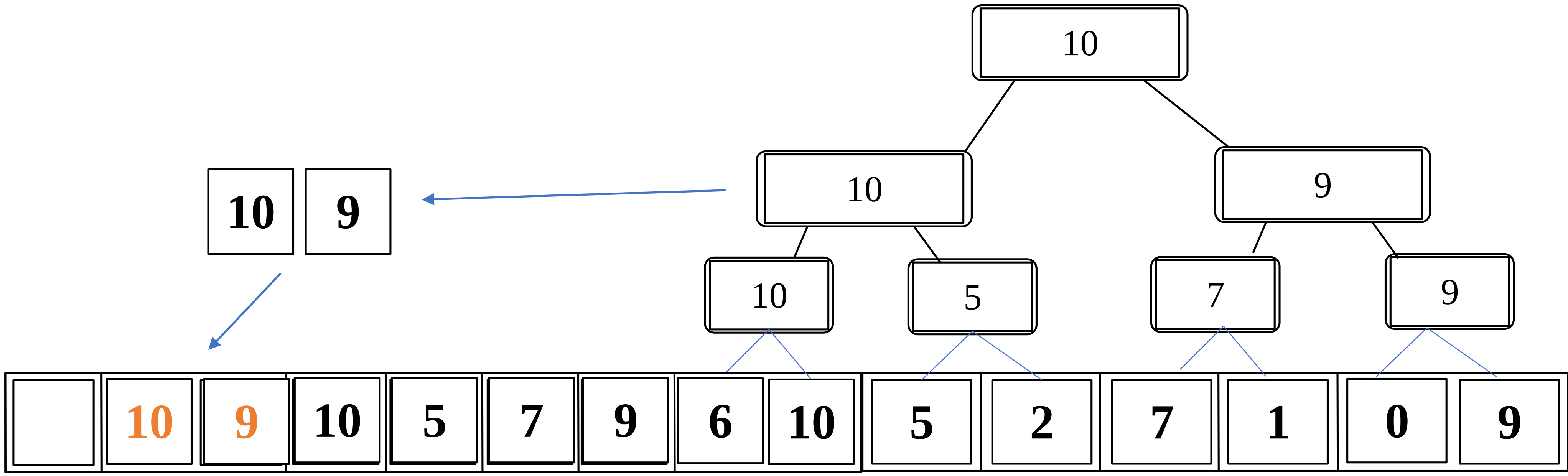
- We want to Find the Max value using Segment Tree
- Start from the last level of the tree and move toward the root



Segment Tree

(Storing The Max Value Tree In Array) - (This slide will not be on canvas)

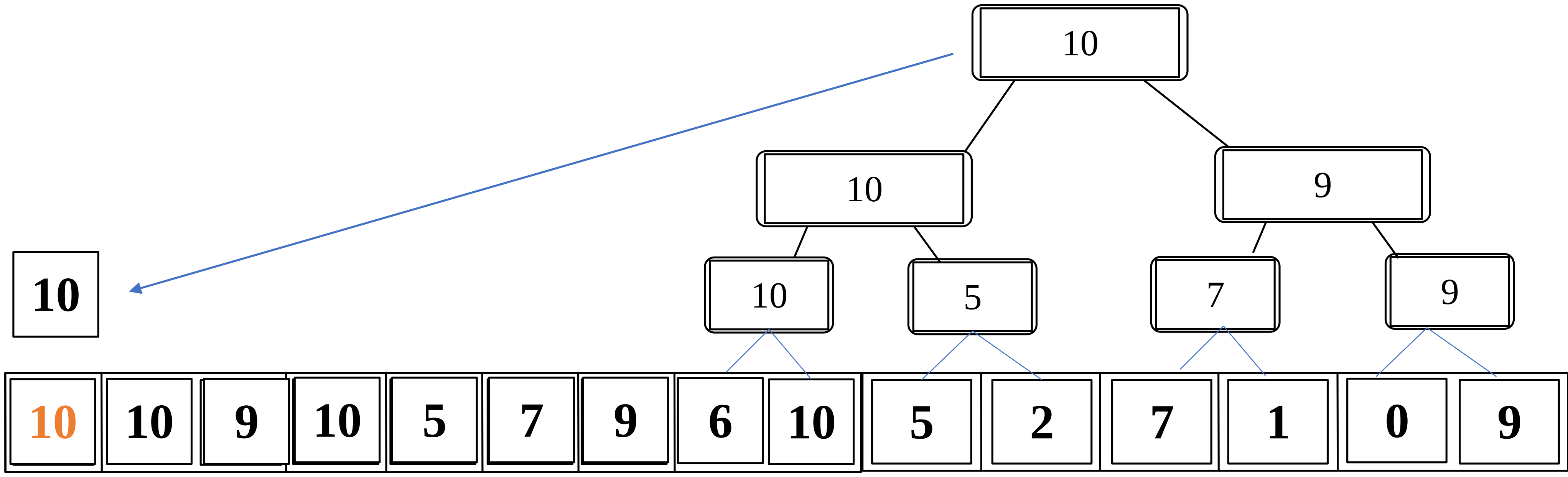
- We want to Find the Max value using Segment Tree
- Start from the last level of the tree and move toward the root



Segment Tree

(Storing The Max Value Tree In Array) - (This slide will not be on canvas)

- We want to Find the Max value using Segment Tree
- Start from the last level of the tree and move toward the root

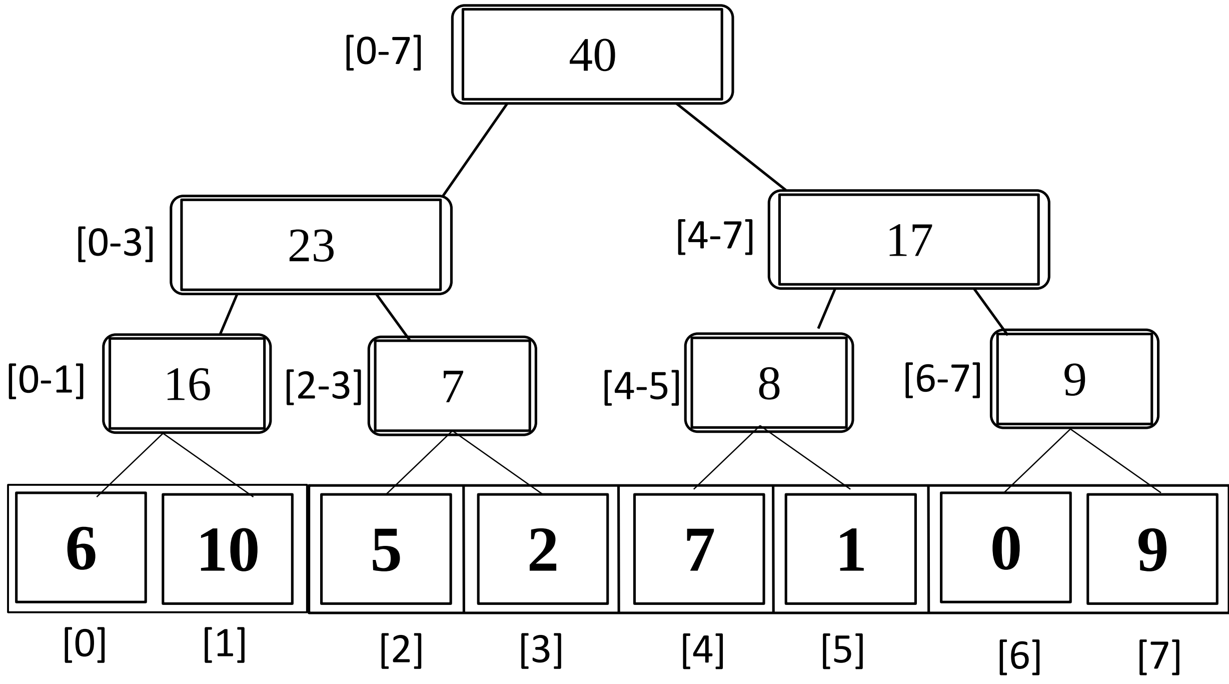


Segment Tree

- Since a Segment Tree is a binary tree, a simple linear array can be used to represent the Segment Tree.
- Before building the **Segment Tree**, one must figure what needs to be stored in the Segment Tree's node?
- **For example,** if the question is to find the sum of all the elements in an array from indices L to R, then at each node (except leaf nodes) the sum of its children nodes is stored.

null	**40**	**23**	**17**	**16**	**7**	**8**	**9**
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

null	[0-7]	[0-3]	[4-7]	[0-1]	[2-3]	[4-5]	[6-7]
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]



Segment Tree

- The main idea behind segment trees is this:
  1. Calculate the sum of the entire array and write it down somewhere;
  2. Split the array into two halves, calculate the sum on both halves, and also write them down somewhere;
  3. Split these halves into halves, calculate the total of four sums on them, and also write them down;
  4. …And so on, until we recursively reach segments of length one.
- These computed subsegment sums can be logically represented as a binary tree.

Advantages of Segment tree

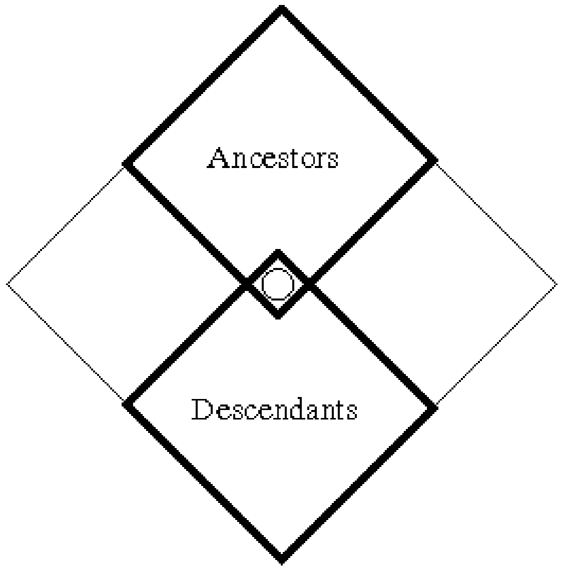
1. **Efficient Range Queries**
2. **Dynamic Updates**: Segment trees can efficiently handle updates to the elements of an array or sequence. Whether it's modifying a single element or updating a range of values, segment trees can perform these updates in logarithmic time complexity. This ability to handle dynamic updates makes them suitable for scenarios where the array elements change frequently.
3. **Versatility:** This versatility allows segment trees to be applied in a wide range of problems, including computational geometry, statistics, and data analysis.
4. **Space Efficiency:** This efficiency arises from the recursive nature of segment trees, where memory is shared among overlapping segments.
5. **Easy to Implement**
6. **Range Decomposition:** Segment trees naturally decompose an array or sequence into smaller segments, allowing for efficient recursive processing. This property makes them useful in divide-and-conquer algorithms, where problems are divided into smaller subproblems, solved recursively, and then combined to obtain the final result.

Disadvantages of Segment tree

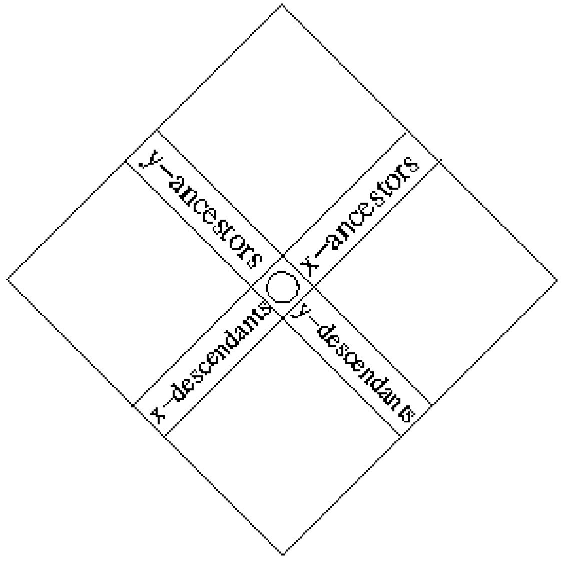
1. **Space Complexity:** Where memory usage is a concern. The space complexity of a segment tree is O(n), where n is the number of elements in the input array.
2. **Construction Time:** Building a segment tree can take a significant amount of time, especially when the input array is large. The construction process involves recursively dividing the array into smaller segments and performing operations to build the tree. The time complexity of constructing a segment tree is O(n), where n is the number of elements in the input array.
3. **Updates and Modifications:** When an element in the input array changes, the corresponding segment tree needs to be updated to reflect the modification, which can require traversing multiple tree nodes. The time complexity for updating an element in a segment tree is O(log n), where n is the number of elements in the input array.
4. **Static Structure:** Segment trees are designed to work with static or semi-static arrays. If the input array frequently changes its size or elements, rebuilding the segment tree from scratch can be inefficient.
5. **Complex Implementation:** Implementing a segment tree can be challenging compared to simpler data structures. The recursive nature of segment trees and the need for efficient indexing can make the code more intricate and error-prone. Understanding and implementing various operations, such as range queries and updates, require a good grasp of tree algorithms

Segment Tree (Applications)

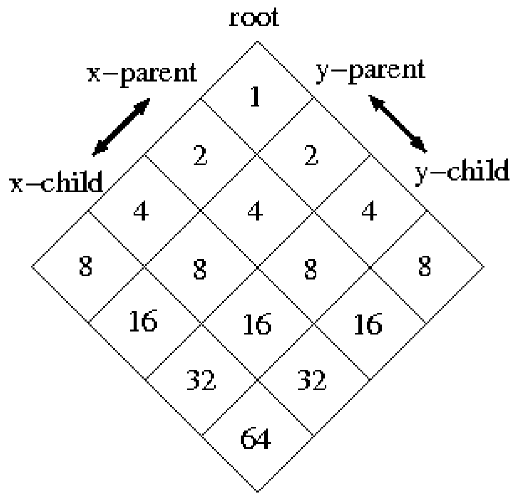
- The Segment Tree was used to efficiently list all pairs of intersecting rectangles from a list of rectangles in the plane.
- We can use this method to report the list of all rectilinear line segments in the plane which intersect a query line segment.
- We use this technique to report the perimeter of a set of rectangles in the plane.



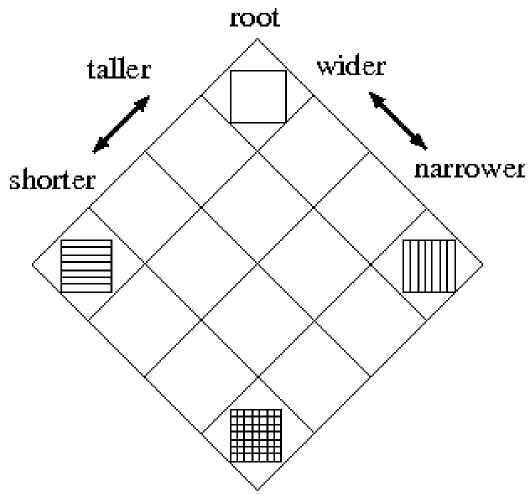
(a)



(b)



(a)

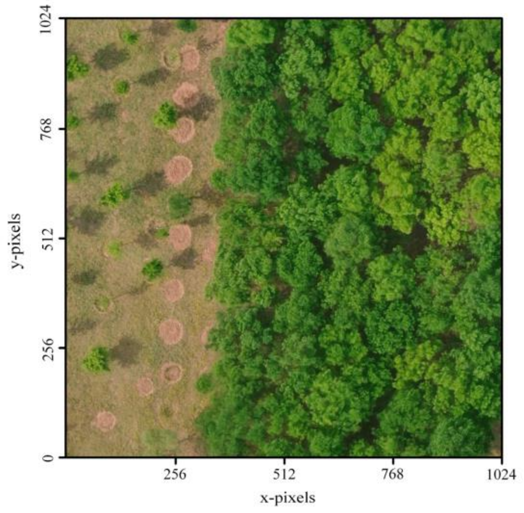


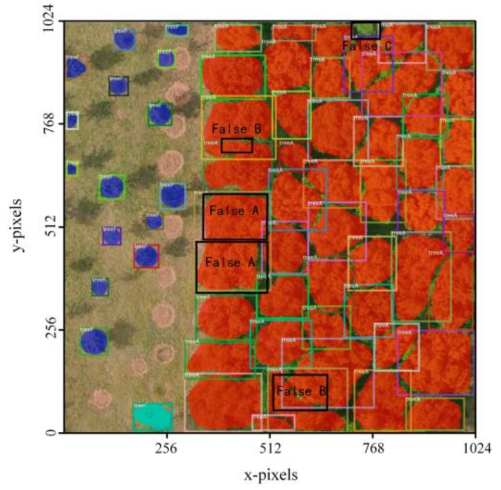
(b)

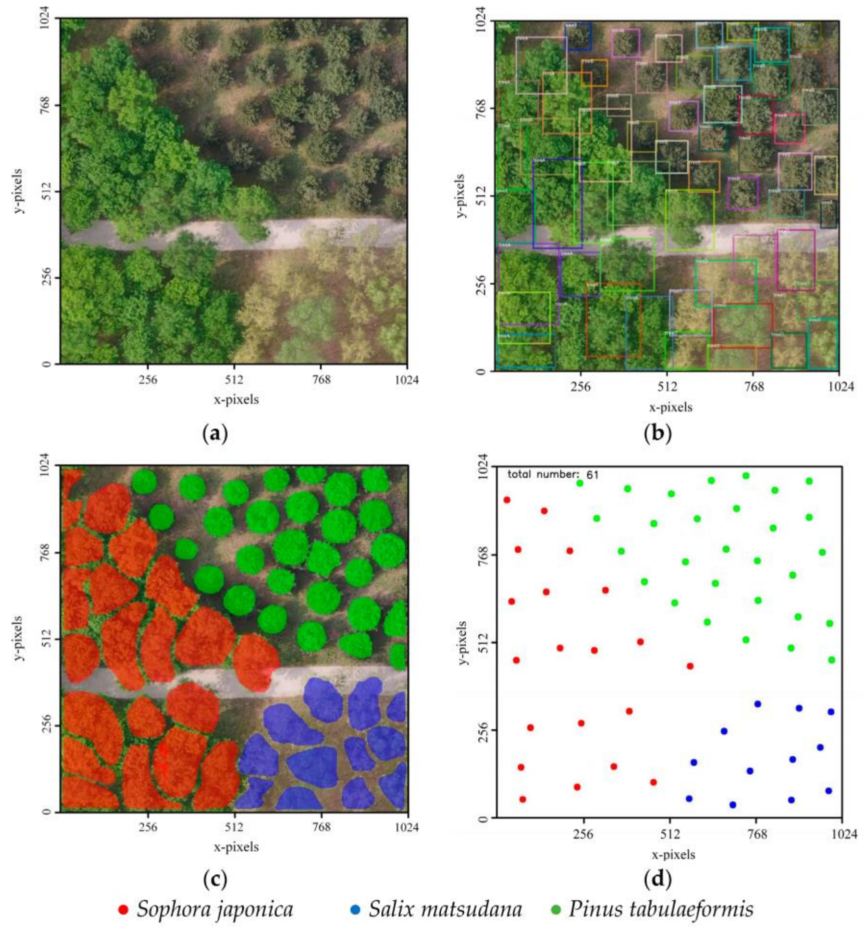
Source: Wagner, David P.. “The Unified Segment Tree and its Application to the Rectangle Intersection Problem.” ArXiv abs/1302.6653 (2013)

Segment Tree (Applications)

- More recently, the segment tree has become popular for use in pattern recognition and image processing.
- We can also specify other applications that are very well known:
  - Finding range sum/product, range max/min, prefix sum/product, etc
  - Computational geometry
  - Geographic information systems
  - Static and Dynamic RMQ (Range Minimum Query)
  - Storing segments in an arbitrary manner







Source: Multi-Species Individual Tree Segmentation and Identification Based on Improved Mask R-CNN and UAV Imagery in Mixed Forests (Link)