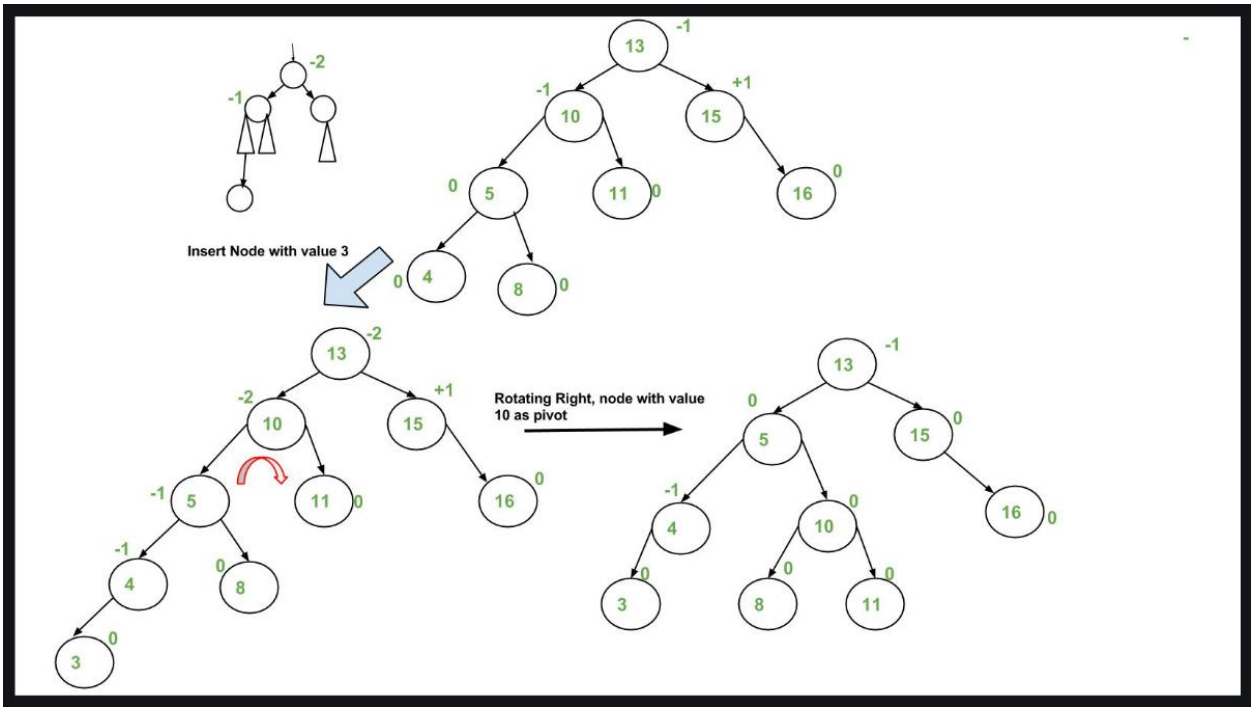
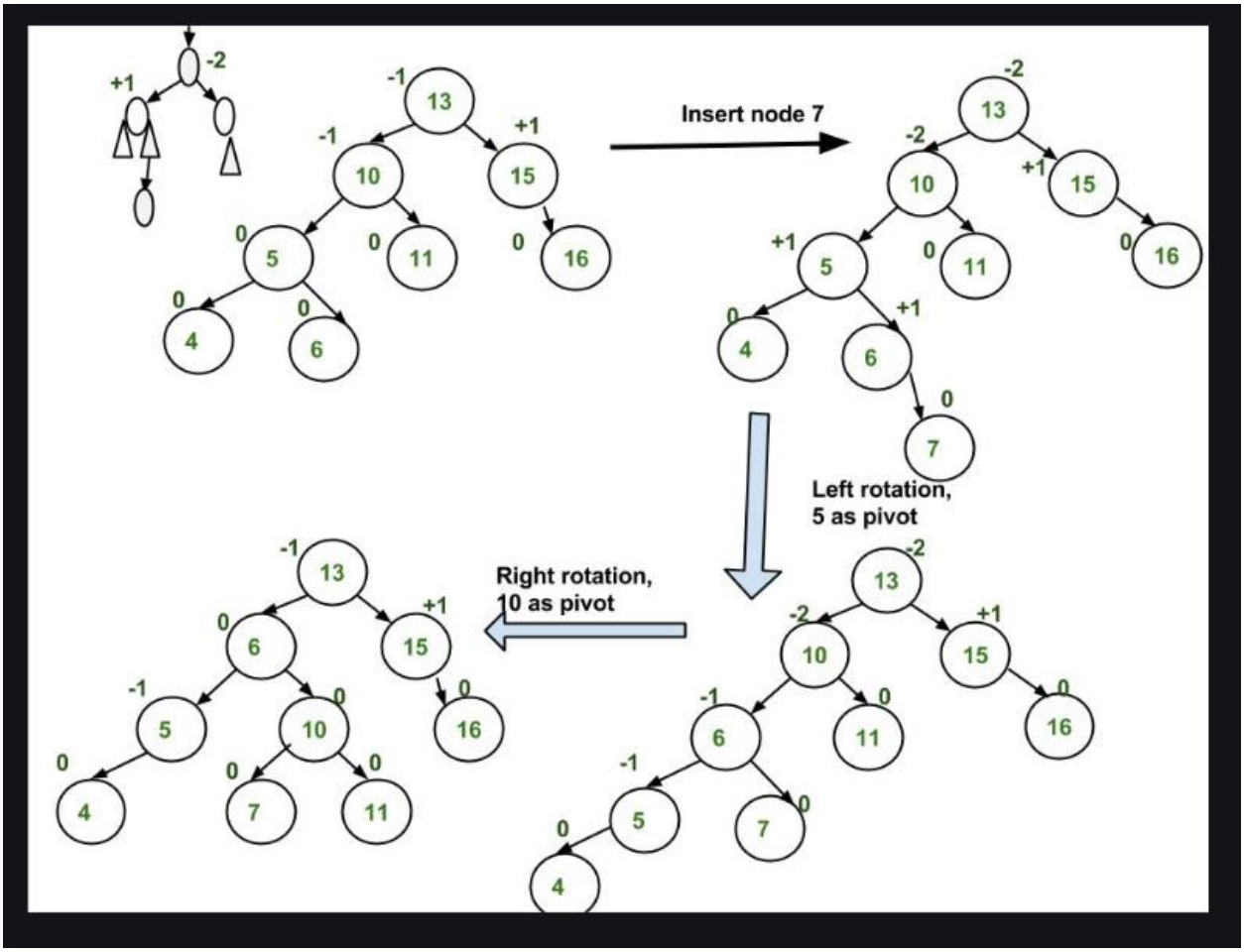
- **Dijkstra's algorithm pseudocode**

```
function dijkstra(G, S)
    for each vertex V in G
        distance[V] <- infinite
        previous[V] <- NULL
        If V != S, add V to Priority Queue Q
    distance[S] <- 0

    while Q IS NOT EMPTY
        U <- Extract MIN from Q
        for each unvisited neighbour V of U
            tempDistance <- distance[U] + edge_weight(U, V)
            if tempDistance < distance[V]
                distance[V] <- tempDistance
                previous[V] <- U
    return distance[], previous[]
```

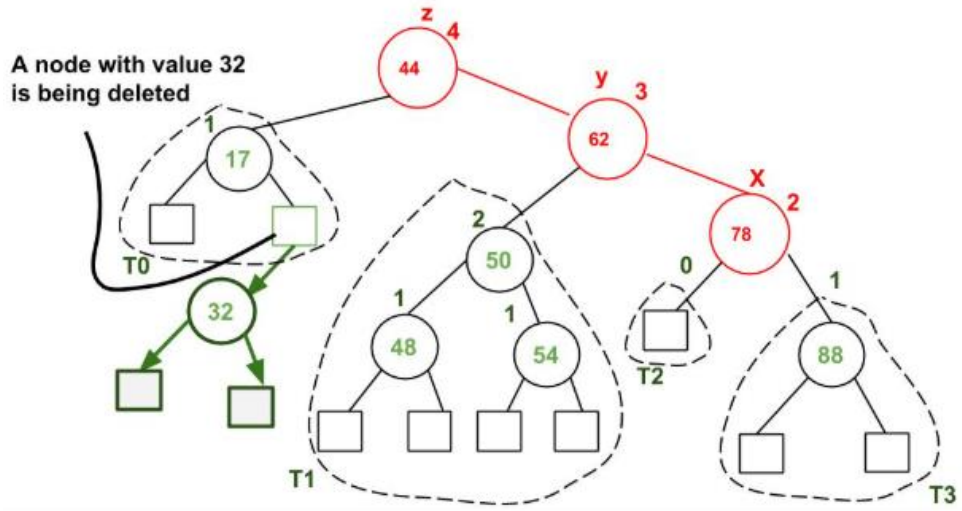
```cpp
// Function to add an edge to the graph
void Graph::addEdge(int u, int v, int w) {
    adj[u].push_back(make_pair(v, w));
    adj[v].push_back(make_pair(u, w)); // Since the graph is undirected
}

// Function to print shortest paths from source
void Graph::shortestPath(int src) {
    // Create a priority queue to store vertices being processed
    // Priority queue sorted by the first element of the pair (distance)
    priority_queue<iPair, vector<iPair>, greater<iPair>> pq;

    // Create a vector to store distances and initialize all distances as INF
    vector<int> dist(V, INF);

    // Insert source into priority queue and initialize its distance as 0
    pq.push(make_pair(0, src));
    dist[src] = 0;

    // Process the priority queue
    while (!pq.empty()) {
        // Get the vertex with the minimum distance
        int u = pq.top().second;
        pq.pop();

        // Iterate through all adjacent vertices of the current vertex
        for (auto &neighbor : adj[u]) {
            int v = neighbor.first;
            int weight = neighbor.second;

            // If a shorter path to v is found
            if (dist[v] > dist[u] + weight) {
                // Update distance and push new distance to the priority queue
                dist[v] = dist[u] + weight;
                pq.push(make_pair(dist[v], v));
            }
        }
    }
```

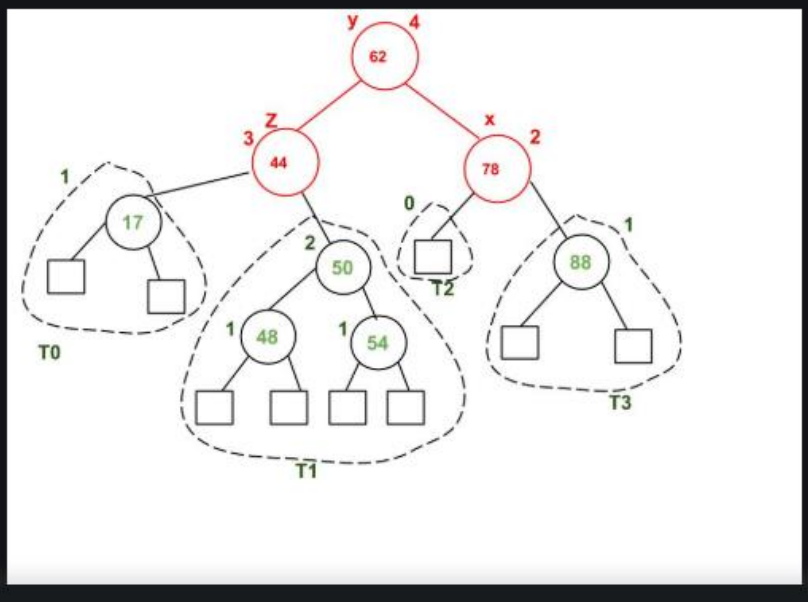
- **AVL**


-2
-1
13
-1
10
+1
15
0
5
11
0
16
0
Insert Node with value 3
0
4
8
0
13
-2
-2
10
+1
15
-1
5
11
0
16
0
-1
4
0
8
3
0
Rotating Right, node with value
10 as pivot
13
-1
0
5
15
0
-1
4
0
10
0
16
0
0
3
0
8
0
11


Insert node 7
Left rotation,
5 as pivot
Right rotation,
10 as pivot

Example of deletion from an AVL Tree:





```
// Calculate height
int height(Node* N) {
    if (N == nullptr)
        return 0;
    return N->getHeight();
}

// Rotate right
Node* rightRotate(Node* N) {

    Node* x = N->getLeft();

    x->setRight(N);

    return x;
}

// Rotate left
Node* leftRotate(Node* N) {
    Node* y = N->getLeft();

    y->setRight(N);

    return y;
}

// Get the balance factor of a node
int getBalanceFactor(Node* N) {
    if (N == nullptr)
        return 0;
    return height(N->getLeft()) - height(N->getRight());
}
```

```
66 Node* insertNode(Node* node, int key) {
67     if (node == nullptr)
68         return new Node(key);
69 
70     if (key < node->getKey())
71         node->setLeft(insertNode(node->getLeft(), key));
72     else if (key > node->getKey())
73         node->setRight(insertNode(node->getRight(), key));
74     else
75         return node;
76 
77     node->setHeight(1 + max(height(node->getLeft()), height(node->getRight())));
78 
79     int balanceFactor = getBalanceFactor(node);
80     // Left Left Case
81     if (balanceFactor > 1 && key < node->getLeft()->getKey())
82         return rightRotate(node);
83 
84     // Left Right Case
85     if (balanceFactor > 1 && key > node->getLeft()->getKey()) {
86         node->setLeft(leftRotate(node->getLeft()));
87         return rightRotate(node);
88     }
89 
90     // Right Right Case
91     if (balanceFactor < -1 && key > node->getRight()->getKey())
92         return leftRotate(node);
93 
94     // Right Left Case
95     if (balanceFactor < -1 && key < node->getRight()->getKey()) {
96         node->setRight(rightRotate(node->getRight()));
97         return leftRotate(node);
98     }
99     return node; }
```