**Q9: Red-Black Tree for Dynamic Leaderboard**: Implement a **red-black tree** to manage a dynamic gaming leaderboard. As players gain points, their rank in the tree adjusts in real time.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Dynamic Leaderboard</title>

<style>

body {

font-family: Arial, sans-serif;

margin: 0;

padding: 0;

display: flex;

flex-direction: column;

align-items: center;

background-color: #f4f4f9;

}

h1 {

color: #333;

}

.leaderboard {

margin-top: 20px;

border: 1px solid #ddd;

border-radius: 5px;

width: 80%;

max-width: 500px;

background: #fff;

}

.player {

display: flex;

justify-content: space-between;

padding: 10px 15px;

border-bottom: 1px solid #ddd;

}

.player:last-child {

border-bottom: none;

}

.player .rank {

font-weight: bold;

color: #ff5722;

}

.controls {

margin-top: 20px;

text-align: center;

}

.controls input, .controls button {

padding: 10px;

margin: 5px;

font-size: 16px;

}

</style>

</head>

<body>

<h1>Dynamic Leaderboard</h1>

<div class="leaderboard" id="leaderboard"></div>

<div class="controls">

<input type="text" id="playerName" placeholder="Player Name">

<input type="number" id="playerScore" placeholder="Score">

<button onclick="addPlayer()">Add/Update Player</button>

</div>

<script>

class Node {

constructor(player, score, color = 'RED') {

this.player = player;

this.score = score;

this.color = color;

this.left = null;

this.right = null;

this.parent = null;

}

}

class RedBlackTree {

constructor() {

this.TNULL = new Node(null, 0, 'BLACK');

this.root = this.TNULL;

}

// Perform in-order traversal to get sorted leaderboard

inOrder(node, result) {

if (node !== this.TNULL) {

this.inOrder(node.right, result); // Descending order by score

result.push({ player: node.player, score: node.score });

this.inOrder(node.left, result);

}

}

getLeaderboard() {

let result = [];

this.inOrder(this.root, result);

return result;

}

// Helper: Rotate left

rotateLeft(x) {

let y = x.right;

x.right = y.left;

if (y.left !== this.TNULL) y.left.parent = x;

y.parent = x.parent;

if (x.parent === null) {

this.root = y;

} else if (x === x.parent.left) {

x.parent.left = y;

} else {

x.parent.right = y;

}

y.left = x;

x.parent = y;

}

// Helper: Rotate right

rotateRight(y) {

let x = y.left;

y.left = x.right;

if (x.right !== this.TNULL) x.right.parent = y;

x.parent = y.parent;

if (y.parent === null) {

this.root = x;

} else if (y === y.parent.right) {

y.parent.right = x;

} else {

y.parent.left = x;

}

x.right = y;

y.parent = x;

}

// Insert a new node and fix violations

insert(player, score) {

let node = new Node(player, score);

node.left = this.TNULL;

node.right = this.TNULL;

let y = null;

let x = this.root;

while (x !== this.TNULL) {

y = x;

if (node.score < x.score) {

x = x.left;

} else if (node.score > x.score) {

x = x.right;

} else {

x.player = player; // Update player name

return;

}

}

node.parent = y;

if (y === null) {

this.root = node;

} else if (node.score < y.score) {

y.left = node;

} else {

y.right = node;

}

node.color = 'RED';

this.fixInsert(node);

}

fixInsert(k) {

while (k.parent && k.parent.color === 'RED') {

if (k.parent === k.parent.parent.right) {

let u = k.parent.parent.left;

if (u && u.color === 'RED') {

u.color = 'BLACK';

k.parent.color = 'BLACK';

k.parent.parent.color = 'RED';

k = k.parent.parent;

} else {

if (k === k.parent.left) {

k = k.parent;

this.rotateRight(k);

}

k.parent.color = 'BLACK';

k.parent.parent.color = 'RED';

this.rotateLeft(k.parent.parent);

}

} else {

let u = k.parent.parent.right;

if (u && u.color === 'RED') {

u.color = 'BLACK';

k.parent.color = 'BLACK';

k.parent.parent.color = 'RED';

k = k.parent.parent;

} else {

if (k === k.parent.right) {

k = k.parent;

this.rotateLeft(k);

}

k.parent.color = 'BLACK';

k.parent.parent.color = 'RED';

this.rotateRight(k.parent.parent);

}

}

}

this.root.color = 'BLACK';

}

}

// Initialize the Red-Black Tree

const tree = new RedBlackTree();

function addPlayer() {

const name = document.getElementById('playerName').value.trim();

const score = parseInt(document.getElementById('playerScore').value);

if (name && !isNaN(score)) {

tree.insert(name, score);

updateLeaderboard();

document.getElementById('playerName').value = '';

document.getElementById('playerScore').value = '';

} else {

alert('Please enter a valid name and score.');

}

}

function updateLeaderboard() {

const leaderboard = document.getElementById('leaderboard');

leaderboard.innerHTML = '';

const players = tree.getLeaderboard();

players.forEach((player, index) => {

const div = document.createElement('div');

div.className = 'player';

div.innerHTML = `

<span class="rank">#${index + 1}</span>

<span>${player.player}</span>

<span>${player.score} pts</span>

`;

leaderboard.appendChild(div);

});

}

</script>

</body>

</html>