**Assignment 2**

**Implement a multiplayer game system that uses an AVL tree data structure to organize and manage player data efficiently. The multiplayer game supports multiple players participating simultaneously, and the AVL tree is used to store player information such as player\_id and scores (key, value pair). The system should provide following operation:**

**1. Player Registration**

**2. Leaderboard Display**

**3. Remove player from game**

**CODE:**

#include <iostream>

using namespace std;

class AVLNode {

public:

int player\_id;

int score;

AVLNode \*left;

AVLNode \*right;

int height;

AVLNode(int player\_id, int score)

{

this->player\_id = player\_id;

this->score = score;

left = right = nullptr;

height = 1;

}

};

class AVLTree {

public:

AVLNode \*root;

AVLTree()

{

root = nullptr;

}

int height(AVLNode \*node)

{

return node ? node->height : 0;

}

int balanceFactor(AVLNode \*node)

{

return node ? height(node->left) - height(node->right) : 0;

}

AVLNode \*rotateRight(AVLNode \*y)

{

AVLNode \*x = y->left;

AVLNode \*T2 = x->right;

x->right = y;

y->left = T2;

y->height = max(height(y->left), height(y->right)) + 1;

x->height = max(height(x->left), height(x->right)) + 1;

return x;

}

AVLNode \*rotateLeft(AVLNode \*x)

{

AVLNode \*y = x->right;

AVLNode \*T2 = y->left;

y->left = x;

x->right = T2;

x->height = max(height(x->left), height(x->right)) + 1;

y->height = max(height(y->left), height(y->right)) + 1;

return y;

}

AVLNode \*insert(AVLNode \*node, int player\_id, int score)

{

if (!node)

{

return new AVLNode(player\_id, score);

}

if (player\_id < node->player\_id)

{

node->left = insert(node->left, player\_id, score);

}

else if (player\_id > node->player\_id)

{

node->right = insert(node->right, player\_id, score);

}

else

{

cout << "Player ID already exists!\n";

return node;

}

node->height = max(height(node->left), height(node->right)) + 1;

int balance = balanceFactor(node);

if (balance > 1 && player\_id < node->left->player\_id)

{

return rotateRight(node);

}

if (balance < -1 && player\_id > node->right->player\_id)

{

return rotateLeft(node);

}

if (balance > 1 && player\_id > node->left->player\_id)

{

node->left = rotateLeft(node->left);

return rotateRight(node);

}

if (balance < -1 && player\_id < node->right->player\_id)

{

node->right = rotateRight(node->right);

return rotateLeft(node);

}

return node;

}

AVLNode \*minValueNode(AVLNode \*node)

{

AVLNode \*current = node;

while (current->left)

{

current = current->left;

}

return current;

}

AVLNode \*remove(AVLNode \*root, int player\_id)

{

if (!root)

{

return root;

}

if (player\_id < root->player\_id)

{

root->left = remove(root->left, player\_id);

}

else if (player\_id > root->player\_id)

{

root->right = remove(root->right, player\_id);

}

else

{

if (!root->left || !root->right)

{

AVLNode \*temp = root->left ? root->left : root->right;

delete root;

return temp;

}

AVLNode \*temp = minValueNode(root->right);

root->player\_id = temp->player\_id;

root->score = temp->score;

root->right = remove(root->right, temp->player\_id);

}

if (!root)

{

return root;

}

root->height = max(height(root->left), height(root->right)) + 1;

int balance = balanceFactor(root);

if (balance > 1 && balanceFactor(root->left) >= 0)

{

return rotateRight(root);

}

if (balance > 1 && balanceFactor(root->left) < 0)

{

root->left = rotateLeft(root->left);

return rotateRight(root);

}

if (balance < -1 && balanceFactor(root->right) <= 0)

{

return rotateLeft(root);

}

if (balance < -1 && balanceFactor(root->right) > 0)

{

root->right = rotateRight(root->right);

return rotateLeft(root);

}

return root;

}

void inOrder(AVLNode \*root)

{

if (!root)

{

return;

}

inOrder(root->right);

cout << "Player ID: " << root->player\_id << ", Score: " << root->score << endl;

inOrder(root->left);

}

};

int main()

{

AVLTree game;

int choice, player\_id, score;

do

{

cout << "\n1. Player Registration\n2. Leaderboard Display\n3. Remove Player from Game\n4. Exit\n";

cout << "Enter choice: ";

cin >> choice;

switch (choice)

{

case 1:

{

cout << "Enter Player ID and Score: ";

cin >> player\_id >> score;

game.root = game.insert(game.root, player\_id, score);

break;

}

case 2:

{

cout << "Leaderboard:\n";

game.inOrder(game.root);

break;

}

case 3:

{

cout << "Enter Player ID to remove: ";

cin >> player\_id;

game.root = game.remove(game.root, player\_id);

break;

}

case 4:

{

cout << "Exiting...\n";

break;

}

default:

{

cout << "Invalid choice!\n";

}

}

}

while (choice != 4);

return 0;

}