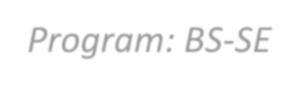
 COMSATS UNIVERSITY ISLAMABAD, ATTOCK CAMPUS

DEPARTMENT OF COMPUTER SCIENCE



*Program: BS*

*-*

*SE*

|  |  |  |
| --- | --- | --- |
| Course: |  | DS |
| Name: |  | EMAN SHAFIQUE |
| Registration Number: |  | SP23-BSE-058 |
| Date: |  | 10th December ,2024 |
| Assignment No: |  | 03 |
| Submitted to: |  | Muhammad Kamran |

**BST Implementation :**

#include <iostream>

#include <vector>

using namespace std;

struct Book {

int isbn;

double price;

Book\* left;

Book\* right;

Book(int i, double p) : isbn(i), price(p), left(nullptr), right(nullptr) {}

};

class BST {

private:

Book\* root;

Book\* insert(Book\* node, int isbn, double price) {

if (!node) return new Book(isbn, price);

if (isbn < node->isbn)

node->left = insert(node->left, isbn, price);

else

node->right = insert(node->right, isbn, price);

return node;

}

Book\* searchByISBN(Book\* node, int isbn) {

if (!node || node->isbn == isbn)

return node;

if (isbn < node->isbn)

return searchByISBN(node->left, isbn);

return searchByISBN(node->right, isbn);

}

void searchByPriceRange(Book\* node, double low, double high, vector<Book\*>& result) {

if (!node) return;

if (node->price >= low && node->price <= high)

result.push\_back(node);

if (low < node->price)

searchByPriceRange(node->left, low, high, result);

if (high > node->price)

searchByPriceRange(node->right, low, high, result);

}

Book\* findMin(Book\* node) {

while (node && node->left)

node = node->left;

return node;

}

Book\* deleteByISBN(Book\* node, int isbn) {

if (!node) return node;

if (isbn < node->isbn) {

node->left = deleteByISBN(node->left, isbn);

} else if (isbn > node->isbn) {

node->right = deleteByISBN(node->right, isbn);

} else {

if (!node->left) {

Book\* temp = node->right;

delete node;

return temp;

} else if (!node->right) {

Book\* temp = node->left;

delete node;

return temp;

}

Book\* temp = findMin(node->right);

node->isbn = temp->isbn;

node->price = temp->price;

node->right = deleteByISBN(node->right, temp->isbn);

}

return node;

}

void inOrder(Book\* node) {

if (!node) return;

inOrder(node->left);

cout << "ISBN: " << node->isbn << ", Price: $" << node->price << endl;

inOrder(node->right);

}

void preOrder(Book\* node) {

if (!node) return;

cout << "ISBN: " << node->isbn << ", Price: $" << node->price << endl;

preOrder(node->left);

preOrder(node->right);

}

void postOrder(Book\* node) {

if (!node) return;

postOrder(node->left);

postOrder(node->right);

cout << "ISBN: " << node->isbn << ", Price: $" << node->price << endl;

}

public:

BST() : root(nullptr) {}

void insert(int isbn, double price) {

root = insert(root, isbn, price);

}

void searchByISBN(int isbn) {

Book\* result = searchByISBN(root, isbn);

if (result)

cout << "Book Found - ISBN: " << result->isbn << ", Price: $" << result->price << endl;

else

cout << "Book not found!" << endl;

}

void searchByPriceRange(double low, double high) {

vector<Book\*> result;

searchByPriceRange(root, low, high, result);

if (result.empty())

cout << "No books found in the price range $" << low << " - $" << high << endl;

else {

cout << "Books in the price range $" << low << " - $" << high << ":" << endl;

for (auto book : result)

cout << "ISBN: " << book->isbn << ", Price: $" << book->price << endl;

}

}

void deleteByISBN(int isbn) {

root = deleteByISBN(root, isbn);

}

void inOrder() {

cout << "In-Order Traversal:" << endl;

inOrder(root);

}

void preOrder() {

cout << "Pre-Order Traversal:" << endl;

preOrder(root);

}

void postOrder() {

cout << "Post-Order Traversal:" << endl;

postOrder(root);

}

};

int main() {

BST tree;

tree.insert(1001, 20.5);

tree.insert(1002, 15.0);

tree.insert(1003, 25.0);

tree.insert(1004, 10.0);

tree.inOrder();

tree.preOrder();

tree.postOrder();

cout << "\nSearch by ISBN (1002):" << endl;

tree.searchByISBN(1002);

cout << "\nSearch books in price range $10 - $20:" << endl;

tree.searchByPriceRange(10, 20);

cout << "\nDeleting book with ISBN 1002..." << endl;

tree.deleteByISBN(1002);

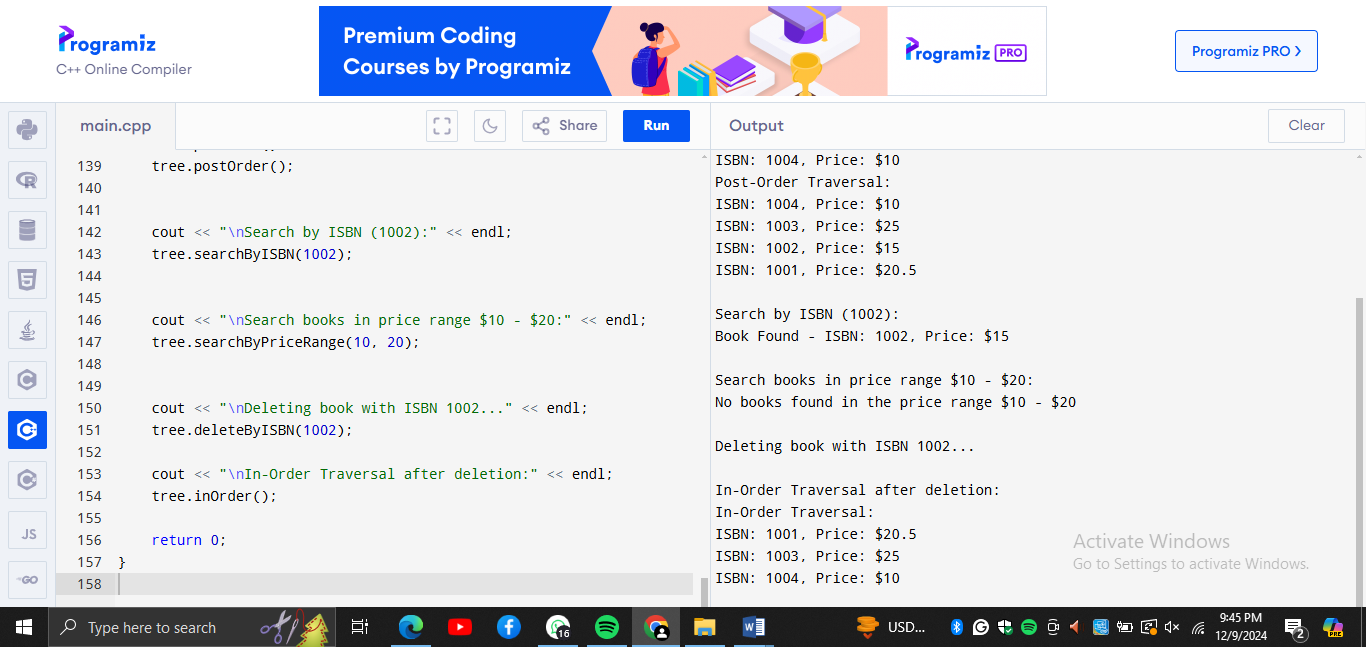
cout << "\nIn-Order Traversal after deletion:" << endl;

tree.inOrder();

return 0;

}





**AVL Tree Implementation :**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

struct Book {

int isbn;

double price;

int height;

Book\* left;

Book\* right;

Book(int i, double p) : isbn(i), price(p), height(1), left(nullptr), right(nullptr) {}

};

class AVLTree {

private:

Book\* root;

int height(Book\* node) {

return node ? node->height : 0;

}

int balanceFactor(Book\* node) {

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

}

Book\* rotateRight(Book\* y) {

Book\* x = y->left;

Book\* 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;

}

Book\* rotateLeft(Book\* x) {

Book\* y = x->right;

Book\* 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;

}

Book\* insert(Book\* node, int isbn, double price) {

if (!node) return new Book(isbn, price);

if (price < node->price)

node->left = insert(node->left, isbn, price);

else if (price > node->price)

node->right = insert(node->right, isbn, price);

else

return node; // Duplicate prices not allowed

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

int balance = balanceFactor(node);

if (balance > 1 && price < node->left->price)

return rotateRight(node); // LL Case

if (balance < -1 && price > node->right->price)

return rotateLeft(node); // RR Case

if (balance > 1 && price > node->left->price) {

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

return rotateRight(node); // LR Case

}

if (balance < -1 && price < node->right->price) {

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

return rotateLeft(node); // RL Case

}

return node;

}

void searchByPriceRange(Book\* node, double low, double high, vector<Book\*>& result) {

if (!node) return;

if (node->price >= low && node->price <= high)

result.push\_back(node);

if (low < node->price)

searchByPriceRange(node->left, low, high, result);

if (high > node->price)

searchByPriceRange(node->right, low, high, result);

}

void inOrder(Book\* node) {

if (!node) return;

inOrder(node->left);

cout << "ISBN: " << node->isbn << ", Price: $" << node->price << endl;

inOrder(node->right);

}

public:

AVLTree() : root(nullptr) {}

void insert(int isbn, double price) {

root = insert(root, isbn, price);

}

void searchByPriceRange(double low, double high) {

vector<Book\*> result;

searchByPriceRange(root, low, high, result);

if (result.empty())

cout << "No books found in the price range $" << low << " - $" << high << endl;

else {

cout << "Books in the price range $" << low << " - $" << high << ":" << endl;

for (auto book : result)

cout << "ISBN: " << book->isbn << ", Price: $" << book->price << endl;

}

}

void inOrder() {

cout << "In-Order Traversal:" << endl;

inOrder(root);

}

};

int main() {

AVLTree tree;

tree.insert(1001, 20.5);

tree.insert(1002, 15.0);

tree.insert(1003, 25.0);

tree.insert(1004, 10.0);

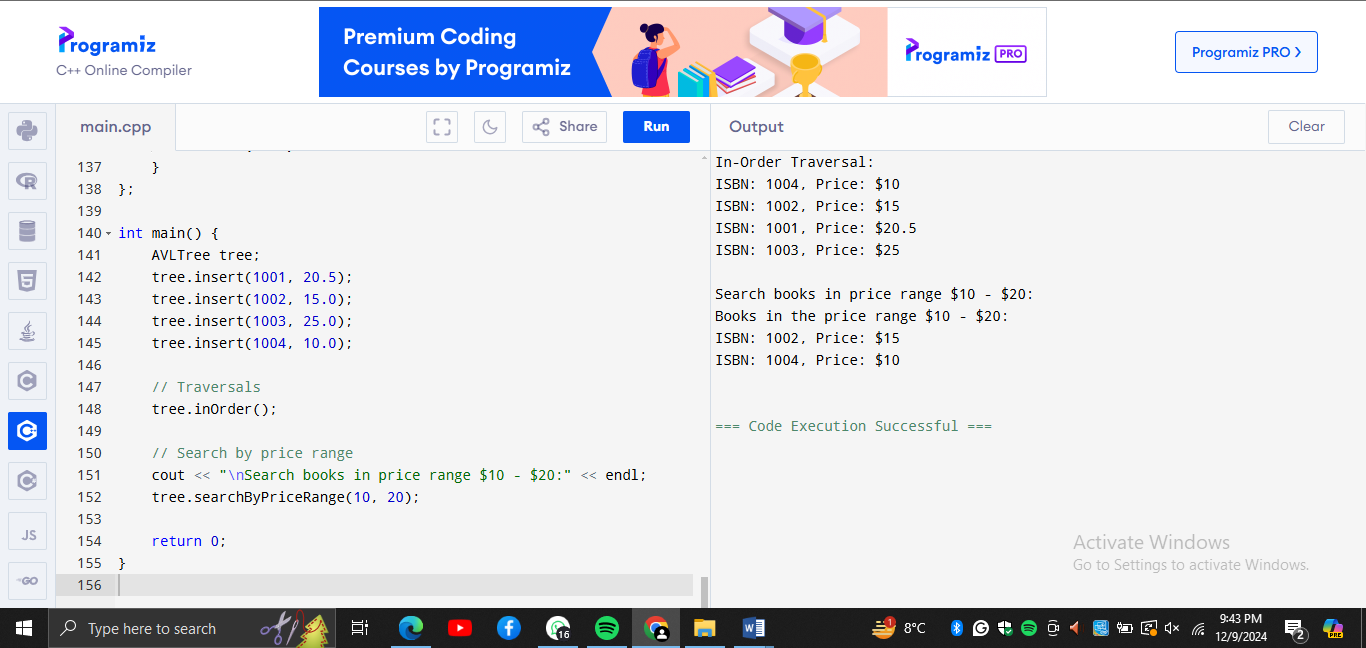
tree.inOrder();

cout << "\nSearch books in price range $10 - $20:" << endl;

tree.searchByPriceRange(10, 20);

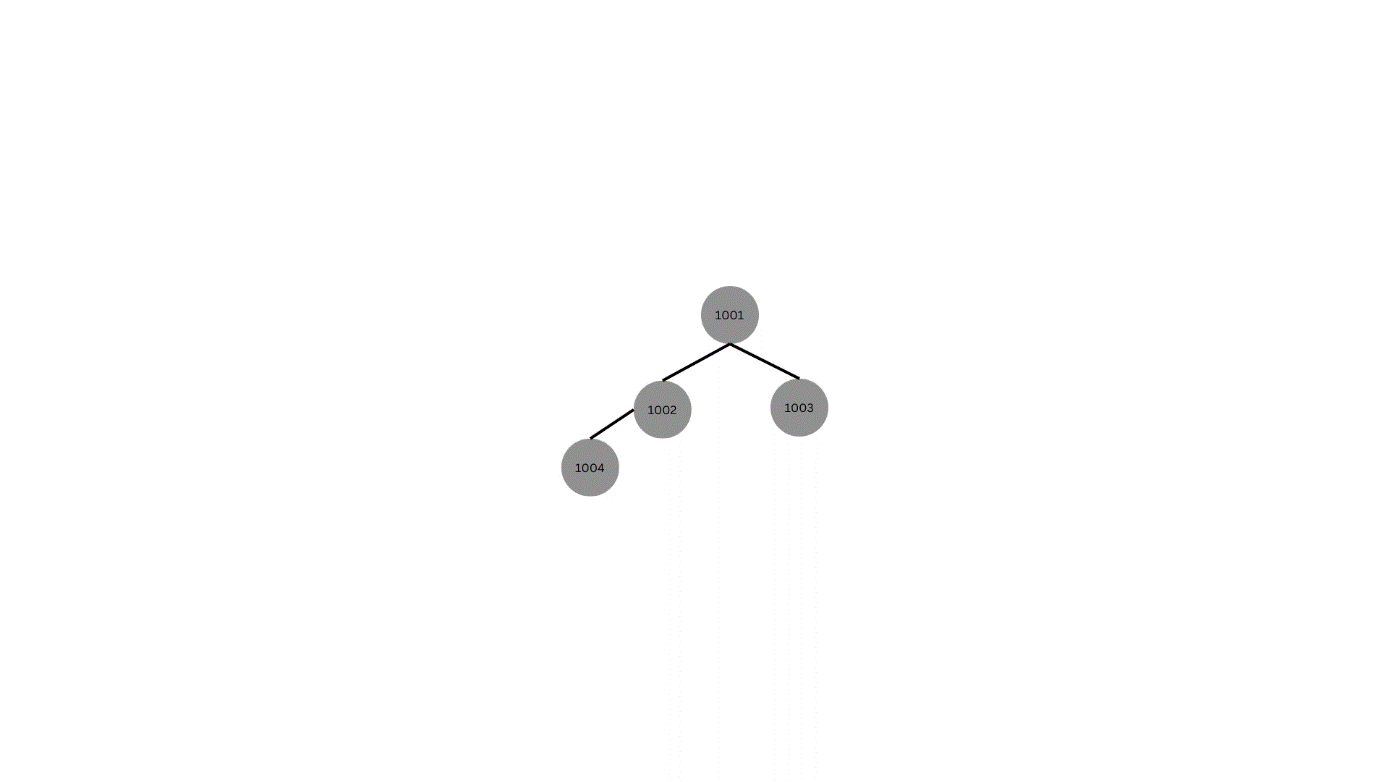
return 0;

}

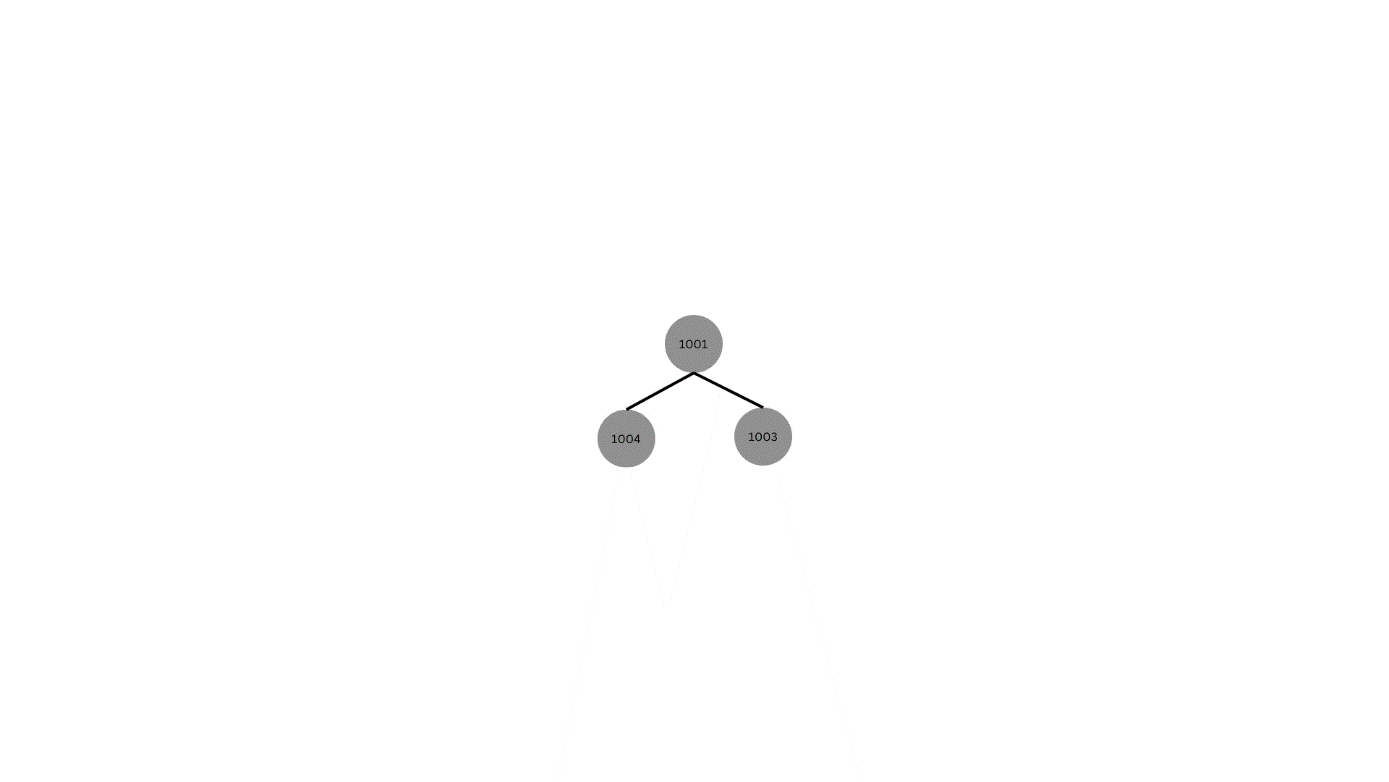


**Visualization :**

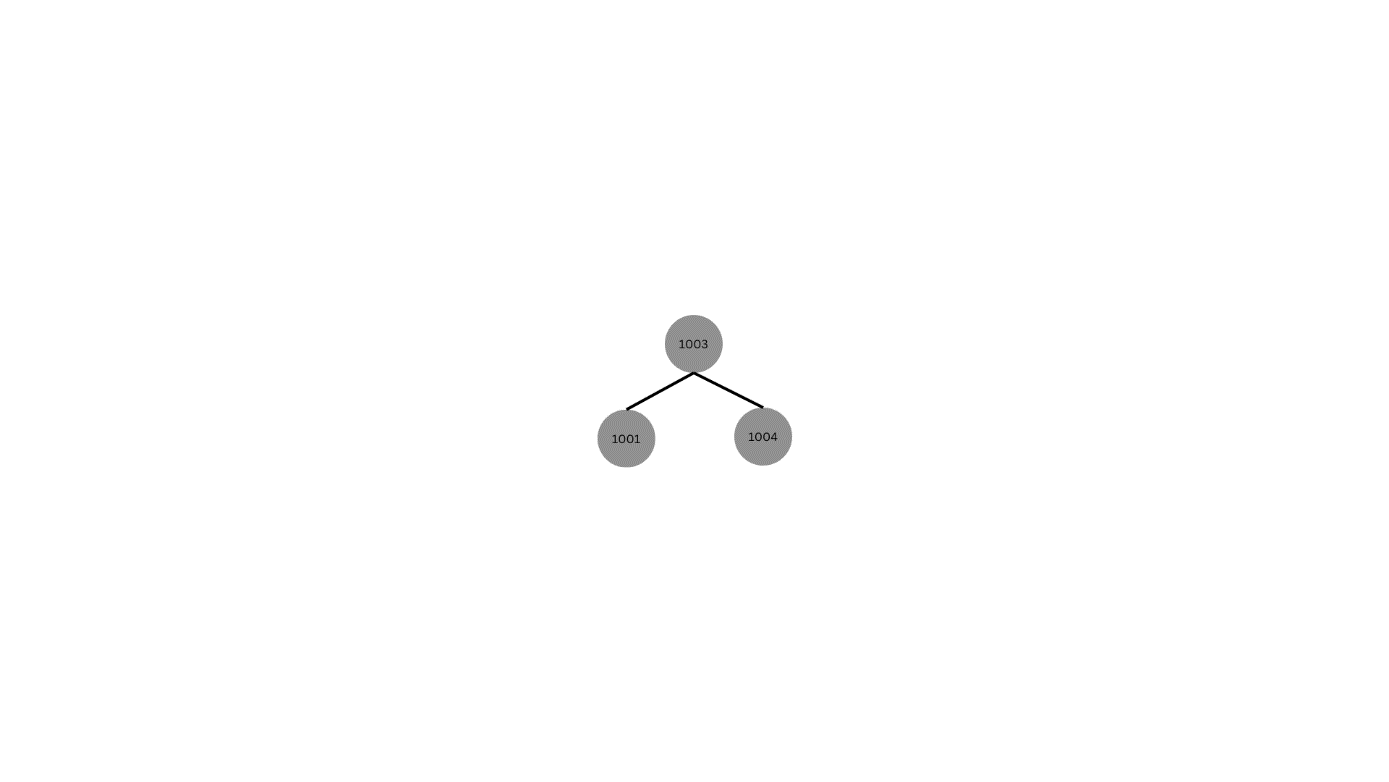
**Initial Insertion (Unbalanced BST)**

****

**After Deletion :**



**After Balancing (AVL Tree)**

****