|  |  |
| --- | --- |
| **Assignment No: 8** | |
| **Aim:** | Develop a menu-driven C++ program to implement a Binary Search Tree (BST) using linked nodes. The program should include basic binary search tree operations: create, inorder traversal, preorder traversal, postorder traversal, and search. |
| **Objective:** | The objective of this assignment is to understand how to implement a Binary Search Tree (BST) using linked nodes. By the end of this assignment, students will gain a thorough understanding of dynamic memory allocation, binary search tree properties, and tree traversal techniques. |
| **Theory:** | A Binary Search Tree (BST) is a type of binary tree in which all the nodes are organized based on the following properties:  - For each node, the value of all the nodes in its left subtree is smaller than its own value.  - The value of all the nodes in its right subtree is greater than its own value.  The operations typically performed on a BST include insertion (create), traversals, and searching.  Operations in a Binary Search Tree (BST):  - Create (Insert): A new node is inserted in the binary search tree such that the properties of the BST are maintained.  - Inorder Traversal: Traverse the tree in left-root-right order, which results in the nodes being displayed in increasing order.  - Preorder Traversal: Traverse the tree in root-left-right order.  - Postorder Traversal: Traverse the tree in left-right-root order.  - Search: Find whether a specific value exists in the tree.  Advantages of Binary Search Tree (BST):  1. Efficient Searching: The search operation in a balanced BST has an average time complexity of O(log n).  2. Dynamic Structure: Nodes can be dynamically added or deleted as needed.  3. In-order Traversal: The in-order traversal of a BST results in a sorted list of elements.  Applications of Binary Search Tree (BST):  - Storing data in a sorted manner for quick lookup.  - Implementing sets and maps.  - Memory-efficient data structure for hierarchical data storage.  - Searching, insertion, and deletion operations in applications like databases and file systems. |
| **Algorithm:** | **1. Create (Insert) Operation:**  1. Start.  2. Allocate memory for a new node.  3. Set the left and right pointers of the new node to NULL.  4. Compare the new node's value with the current node (starting from the root):  - If the new value is less than the current node, move to the left subtree.  - If the new value is greater, move to the right subtree.  5. Recursively insert the node in the appropriate position.  6. End.  **2. Inorder Traversal (Left-Root-Right):**  1. Start.  2. Traverse the left subtree.  3. Visit the root node and display its value.  4. Traverse the right subtree.  5. End.  **3. Preorder Traversal (Root-Left-Right):**  1. Start.  2. Visit the root node and display its value.  3. Traverse the left subtree.  4. Traverse the right subtree.  5. End.  **4. Postorder Traversal (Left-Right-Root):**  1. Start.  2. Traverse the left subtree.  3. Traverse the right subtree.  4. Visit the root node and display its value.  5. End.  **5. Search Operation:**  1. Start.  2. Compare the value to be searched with the current node (starting from the root).  3. If the value matches the current node, return the node.  4. If the value is smaller, search the left subtree.  5. If the value is greater, search the right subtree.  6. Repeat the steps until the value is found or the node is NULL.  7. End. |
| **Program:** | #include <iostream>  using namespace std;  struct Node {  int data;  Node\* left;  Node\* right;  };  Node\* createNode(int value) {  Node\* newNode = new Node();  newNode->data = value;  newNode->left = nullptr;  newNode->right = nullptr;  return newNode;  }  Node\* insert(Node\* root, int value) {  if (root == nullptr) {  return createNode(value);  }  if (value < root->data) {  root->left = insert(root->left, value);  } else {  root->right = insert(root->right, value);  }  return root;  }  void inorder(Node\* root) {  if (root != nullptr) {  inorder(root->left);  cout << root->data << " ";  inorder(root->right);  }  }  void preorder(Node\* root) {  if (root != nullptr) {  cout << root->data << " ";  preorder(root->left);  preorder(root->right);  }  }  void postorder(Node\* root) {  if (root != nullptr) {  postorder(root->left);  postorder(root->right);  cout << root->data << " ";  }  }  Node\* search(Node\* root, int value) {  if (root == nullptr || root->data == value) {  return root;  }  if (value < root->data) {  return search(root->left, value);  }  return search(root->right, value);  }  int main() {  Node\* root = nullptr;  int choice, value;  do {  cout << "\nMenu:\n";  cout << "1. Insert\n";  cout << "2. Inorder Traversal\n";  cout << "3. Preorder Traversal\n";  cout << "4. Postorder Traversal\n";  cout << "5. Search\n";  cout << "6. Exit\n";  cout << "Enter your choice: ";  cin >> choice;  switch (choice) {  case 1:  cout << "Enter value to insert: ";  cin >> value;  root = insert(root, value);  break;  case 2:  cout << "Inorder Traversal: ";  inorder(root);  cout << endl;  break;  case 3:  cout << "Preorder Traversal: ";  preorder(root);  cout << endl;  break;  case 4:  cout << "Postorder Traversal: ";  postorder(root);  cout << endl;  break;  case 5:  cout << "Enter value to search: ";  cin >> value;  if (search(root, value)) {  cout << value << " found in the BST." << endl;  } else {  cout << value << " not found in the BST." << endl;  }  break;  case 6:  cout << "Exiting program." << endl;  break;  default:  cout << "Invalid choice. Please try again." << endl;  }  } while (choice != 6);  return 0;  } |
| **Output:** | Menu:  1. Insert  2. Inorder Traversal  3. Preorder Traversal  4. Postorder Traversal  5. Search  6. Exit  Enter your choice: 1  Enter value to insert: 10  Menu:  1. Insert  2. Inorder Traversal  3. Preorder Traversal  4. Postorder Traversal  5. Search  6. Exit  Enter your choice: 1  Enter value to insert: 5  Menu:  1. Insert  2. Inorder Traversal  3. Preorder Traversal  4. Postorder Traversal  5. Search  6. Exit  Enter your choice: 1  Enter value to insert: 15  Menu:  1. Insert  2. Inorder Traversal  3. Preorder Traversal  4. Postorder Traversal  5. Search  6. Exit  Enter your choice: 2  Inorder Traversal: 5 10 15  Menu:  1. Insert  2. Inorder Traversal  3. Preorder Traversal  4. Postorder Traversal  5. Search  6. Exit  Enter your choice: 3  Preorder Traversal: 10 5 15  Menu:  1. Insert  2. Inorder Traversal  3. Preorder Traversal  4. Postorder Traversal  5. Search  6. Exit  Enter your choice: 4  Postorder Traversal: 5 15 10  Menu:  1. Insert  2. Inorder Traversal  3. Preorder Traversal  4. Postorder Traversal  5. Search  6. Exit  Enter your choice: 5  Enter value to search: 15  15 found in the BST.  Menu:  1. Insert  2. Inorder Traversal  3. Preorder Traversal  4. Postorder Traversal  5. Search  6. Exit  Enter your choice: 5  Enter value to search: 20  20 not found in the BST.  Menu:  1. Insert  2. Inorder Traversal  3. Preorder Traversal  4. Postorder Traversal  5. Search  6. Exit  Enter your choice: 6  Exiting program. |
| **Conclusion:** | |
| **Date:** | |
| **Staff Sign:** | |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***END**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | |