

K.K.Wagh Institute of Engineering Education & Research, Nashik

Department of Computer Engineering

A

Mini-Project

**“Inventory Management for Computer Parts”**

**Submitted by : (SE-B)**

Janhavi Chavan [Roll No.14]

Shraddha Chavhan [Roll No.15]

Divya Chavan [Roll No.16]

Sakshi Danej [Roll No.17]

**Project Report**

**Title:** Inventory Management for Computer Parts using Binary Search Tree

**Problem Statement:** The aim of this mini project is to develop an inventory management system for computer parts using the Binary Search Tree (BST) data structure. The system should allow users to insert, update, and delete computer parts, as well as perform inorder, postorder, and preorder traversals of the inventory.

**Prerequisites:**

* Knowledge of Java programming language.
* Familiarity with the Binary Search Tree data structure and its basic operations.
* Understanding of object-oriented programming concepts.

**Objective:** The objective of this mini project is to implement a simple inventory management system for computer parts using the Binary Search Tree data structure. The project aims to provide the following functionalities:

* Insert new computer parts into the inventory.
* Update existing computer parts' information (such as name, brand, type, quantity, etc.).
* Delete computer parts from the inventory.
* Perform inorder, postorder, and preorder traversals of the inventory to display the parts in different orders.

**Theory:**

To achieve the required functionalities, the following non-recursive algorithms are implemented:

a) Insertion Algorithm:

1. Start at the root of the BST.
2. If the tree is empty, create a new node with the given data and make it the root.
3. Otherwise, iterate through the tree by comparing the data with each node.
4. If the data is less than the current node, move to the left child.
5. If the data is greater than the current node, move to the right child.
6. Repeat the above steps until a suitable empty position is found, then insert the new node.

b) Search Algorithm:

1. Start at the root of the BST.
2. Initialize a pointer current to point to the root.
3. While current is not null:

* If the value to be searched is equal to the value of current, return current as the node containing the desired value.
* If the value to be searched is less than the value of current, update current to point to its left child.
* If the value to be searched is greater than the value of current, update current to point to its right child.
* If the value is not found and current becomes null, return null to indicate that the value is not present in the BST

.c) Height of tree Algorithm:

1. If the root of the BST is null, return 0 as the height of the tree.
2. Initialize a variable height to 0.
3. Create an empty stack and push the root node onto the stack.
4. Initialize a variable maxHeight to 0.
5. While the stack is not empty:
6. Increment height by 1.
7. Pop a node from the stack and assign it to a variable current.
8. If current has a left child, push the left child onto the stack.
9. If current has a right child, push the right child onto the stack.
10. If current is a leaf node (both left and right children are null):
11. If height is greater than maxHeight, update maxHeight with the value of height.
12. Decrement height by 1.
13. Return maxHeight as the height of the BST.

d) Inorder Traversal Algorithm:

1. Start at the root of the BST.
2. Initialize an empty stack.
3. Traverse the tree iteratively as follows:
4. Push the current node and move to its left child until reaching a null node.
5. Pop the top node from the stack, process it, and move to its right child.

e) Preorder Traversal Algorithm:

1. Start at the root of the BST.
2. Initialize an empty stack and push the root node.
3. Traverse the tree iteratively as follows:
4. Pop the top node from the stack, process it, and push its right child (if exists).
5. Push the left child (if exists) of the popped node.
6. Repeat the above steps until the stack is empty.

f) Postorder Traversal Algorithm:

1. Start at the root of the BST.
2. Initialize two empty stacks: stack1 and stack2.
3. Push the root node to stack1.
4. Traverse the tree iteratively as follows:
5. Pop a node from stack1, push it to stack2, and push its left and right children to stack1.
6. Repeat the above step until stack1 is empty

**Conclusion**: In conclusion, the mini project "Inventory Management for Computer Parts using Binary Search Tree" has been successfully implemented. The project provides a user-friendly system for managing computer parts inventory efficiently. It allows users to insert new parts, update existing parts, delete parts, and perform different traversals to display the inventory in various orders.

The implementation of the Binary Search Tree data structure enables fast search, insertion, and deletion operations, making it suitable for managing inventory data. The non-recursive algorithms for insertion, update, deletion, and tree traversals ensure efficient execution of the operations without relying on recursive function calls.

**Output of the project:**