MIT World Peace University

Advanced Data Structures
Second Year B. Tech, Semester 4

Implementation of Dictionary using Binary Search Tree

Assignment No. 2

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

February 14, 2023

Contents

1 Objectives

1. To study data structure : Binary Search Tree
2. To study breadth first traversal.
3. To study different operations on Binary search Tree.

2 Problem Statement

Implement dictionary using binary search tree where dictionary stores keywords and its meanings. Perform following operations:

1. Insert a keyword
2. Delete a keyword
3. Create mirror image and display level wise
4. Copy

3 Theory

3.1 Binary Search Tree

3.2 Breadth First Traversal

3.3 Different operations on binary search tree.(copy ,mirror image and delete)

4 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Visual Studio Code
Compilers : g++ and gcc on linux for C++

5 Input

1. Input at least 10 nodes.
2. Display binary search tree levelwise traversals of binary search tree with 10 nodes
3. Display mirror image and copy operations on BST

6 Output

1. The traversal of the binary tree in different ways.

7 Test Conditions

1. Input at least 10 nodes.
2. Display all traversals of binary tree with 10 nodes.(recursive and nonrecursive)

8 Pseudo Code

9 Time Complexity

10 Code

10.1 Program

```
#include <iostream>
#include <queue>
#include <stack>
#include <string.h>
using namespace std;

class WordNode
{
    string word;
    string definition;
    WordNode *left;
    WordNode *right;
    friend class BinarySearchTree;
};

class BinarySearchTree
{
public:
    WordNode *root;
    BinarySearchTree()
    {
        root = NULL;
    }
    void create_root()
    {
        root = new WordNode;
        cout << "Enter the data: " << endl;
        cin >> root->word;
        cin >> root->definition;
        root->left = NULL;
        root->right = NULL;
        create_recursive(root);
    }
    void create_recursive(WordNode *Node)
    {
        int choice = 0;
        WordNode *new_node;
        cout << "Enter if you want to enter a left node (1/0): "
             << "for the node -- " << Node->word << ":  " << Node->definition << "
 -- ";
        cin >> choice;
        if (choice == 1)
```

```
        {
            new_node = new WordNode;
            cout << "Enter the data: ";
            cin >> new_node->word;
            cin >> new_node->definition;
            Node->left = new_node;
            create_recursive(new_node);
        }
        cout << "Enter if you want to enter a right node (1/0): "
             << "for the node -- " << Node->word << ":  " << Node->definition << "
    -- ";
        cin >> choice;
        if (choice == 1)
        {
            new_node = new WordNode;
            cout << "Enter the data: ";
            cin >> new_node->word;
            cin >> new_node->definition;
            Node->right = new_node;
            create_recursive(new_node);
        }
    }
    void create_root_and_tree_iteratively()
    {
        WordNode *temp, *current_word;
        int choice = 0;
        root = new WordNode;
        cout << "Enter the Word: ";
        cin >> root->word;
        cout << "Enter the definition of the word: ";
        cin >> root->definition;
        bool flag = false;
        cout << "Do you want to enter more Nodes? (0/1) " << endl;
        cin >> choice;
        while (choice == 1)
        {
            temp = root;
            flag = false;
            current_word = new WordNode;
            cout << "Enter the Word: ";
            cin >> current_word->word;
            cout << "Enter the definition of the word: ";
            cin >> current_word->definition;

            while (!flag)
            {
                if (strcmp(current_word->word.c_str(), temp->word.c_str()) < 0)
                {
                    if (temp->left == NULL)
                    {
                        temp->left = current_word;
                        flag = true;
                    }
                    else
                    {
                        temp = temp->left;
                    }
                }
                else
```

```
                {
                    if (temp->right == NULL)
                    {
                        temp->right = current_word;
                        flag = true;
                    }
                    else
                    {
                        temp = temp->right;
                    }
                }
            }
            cout << "Do you want to enter another word? (1/0): ";
            cin >> choice;
        }
    }

    // breadth First Search using queue.
    void bfs()
    {
        WordNode *temp = root;
        queue<WordNode *> q;
        q.push(temp);
        while (!q.empty())
        {
            temp = q.front();
            q.pop();
            cout << temp->word << " : " << temp->definition << endl;
            if (temp->left != NULL)
            {
                q.push(temp->left);
            }
            if (temp->right != NULL)
            {
                q.push(temp->right);
            }
        }
    }

    WordNode *create_copy_recursive(WordNode *temp)
    {
        if (temp == NULL)
        {
            return NULL;
        }
        else
        {
            WordNode *new_node = new WordNode;
            new_node->word = temp->word;
            new_node->definition = temp->definition;
            new_node->left = create_copy_recursive(temp->left);
            new_node->right = create_copy_recursive(temp->right);
            return new_node;
        }
    }

    void mirror_recursive(WordNode *temp)
    {
        if (temp == NULL)
```

```
        {
            return;
        }
        else
        {
            WordNode *temp1;
            // Swapping
            temp1 = temp->left;
            temp->left = temp->right;
            temp->right = temp1;

            // Recursively calling the function
            mirror_recursive(temp->left);
            mirror_recursive(temp->right);
        }
    }

    WordNode *create_mirror_tree_recursive()
    {
        WordNode *mirror_tree = create_copy_recursive(root);
        mirror_recursive(mirror_tree);
        return mirror_tree;
    }

    void inorder_iterative(WordNode *temp)
    {
        if (!temp)
        {
            return;
        }

        stack<WordNode *> s;
        WordNode *current = temp;

        while (current != NULL || s.empty() == false)
        {
            while (current != NULL)
            {
                s.push(current);
                current = current->left;
            }
            current = s.top();
            s.pop();
            cout << current->word << " : " << current->definition << endl;
            current = current->right;
        }
    }
    void preorder_iterative(WordNode *temp)
    {
        if (!temp)
        {
            return;
        }

        stack<WordNode *> s;
        s.push(temp);

        while (s.empty() == false)
        {
```

```
            WordNode *current = s.top();
            cout << current->word << " : " << current->definition << endl;
            s.pop();

            if (current->right)
            {
                s.push(current->right);
            }
            if (current->left)
            {
                s.push(current->left);
            }
        }
    }
    void postorder_iterative(WordNode *temp)
    {
        if (!temp)
        {
            return;
        }

        stack<WordNode *> s1;
        stack<WordNode *> s2;

        s1.push(temp);

        while (s1.empty() == false)
        {
            WordNode *current = s1.top();
            s1.pop();
            s2.push(current);

            if (current->left)
            {
                s1.push(current->left);
            }
            if (current->right)
            {
                s1.push(current->right);
            }
        }
        while (s2.empty() == false)
        {
            WordNode *current = s2.top();
            cout << current->word << " : " << current->definition << endl;
            s2.pop();
        }
    }
};

int main()
{
    int choice = 0;
    BinarySearchTree main_tree, mirror_tree, copy_tree;

    while (choice != 8)
    {
        cout << "\nWhat would like to do? " << endl;
        cout << "\n\nWelcome to ADS Assignment 2 - Binary Tree Traversals\n\nWhat
```

```
would you like to do? " << endl;
        cout << "1. Create a Binary Search Tree"
             << endl;
        cout << "2. Traverse the Tree Inorder Iteratively"
             << endl;
        cout << "3. Traverse the Tree PreOrder Iteratively"
             << endl;
        cout << "4. Traverse the Tree PostOrder Iteratively"
             << endl;
        cout << "5. Traverse it using BFS"
             << endl;
        cout << "6. Create a Copy of the tree Recursively"
             << endl;
        cout << "7. Create a Mirror of the Tree Recursively"
             << endl;
        cout << "8. Exit" << endl
             << endl;

        cin >> choice;
        switch (choice)
        {
        case 1:
            main_tree.create_root_and_tree_iteratively();
            break;
        case 2:
            cout << "Traversing through the Binary Tree Inorder Iteratively: " <<
 endl;
            main_tree.inorder_iterative(main_tree.root);
            break;
        case 3:
            cout << "Traversing through the Binary Tree PreOrder Iteratively: " <<
 endl;
            main_tree.preorder_iterative(main_tree.root);
            break;
        case 4:
            cout << "Traversing through the Binary Tree PostOrder Iteratively: "
 << endl;
            main_tree.postorder_iterative(main_tree.root);
            break;
        case 5:
            cout << "Traversing through the Binary Tree using BFS: " << endl;
            main_tree.bfs();
            break;
        case 6:
            cout << "Creating a copy of the tree" << endl;
            copy_tree.root = copy_tree.create_copy_recursive(main_tree.root);
            cout << "Traversing through the Binary Tree Inorder Iteratively: " <<
 endl;
            copy_tree.inorder_iterative(copy_tree.root);
            cout << "Traversing via Breadth First Search: " << endl;
            copy_tree.bfs();
            break;
        case 7:
            cout << "Creating a mirror of the tree" << endl;
            mirror_tree.root = main_tree.create_mirror_tree_recursive();
            cout << "Traversing through the Binary Tree Inorder Iteratively: " <<
 endl;
            mirror_tree.inorder_iterative(mirror_tree.root);
            cout << "Traversing via Breadth First Search: " << endl;
```

```
                mirror_tree.bfs();
                break;
            case 8:
                cout << "Exiting the program" << endl;
                break;
            default:
                cout << "Invalid Choice" << endl;
                break;
            }
        }
}
```

10.2 Input and Output

```
What would like to do?


Welcome to ADS Assignment 2 - Binary Tree Traversals

What would you like to do?
1. Create a Binary Search Tree
2. Traverse the Tree Inorder Iteratively
3. Traverse the Tree PreOrder Iteratively
4. Traverse the Tree PostOrder Iteratively
5. Traverse it using BFS
6. Create a Copy of the tree Recursively
7. Create a Mirror of the Tree Recursively
8. Exit

1
Enter the Word: apple
Enter the definition of the word: fruit
Do you want to enter more Nodes? (0/1)

1
Enter the Word: banana
Enter the definition of the word: fruit
Do you want to enter another word? (1/0): 1
Enter the Word: keyboard
Enter the definition of the word: input
Do you want to enter another word? (1/0): 1
Enter the Word: pears
Enter the definition of the word: fruit
Do you want to enter another word? (1/0): 1
Enter the Word: bottle
Enter the definition of the word: water
Do you want to enter another word? (1/0): 1
Enter the Word: charger
Enter the definition of the word: charging
Do you want to enter another word? (1/0): 1
Enter the Word: monitor
Enter the definition of the word: see
Do you want to enter another word? (1/0): 1
Enter the Word: paper
Enter the definition of the word: 1
Do you want to enter another word? (1/0): 1
Enter the Word: pen
Enter the definition of the word: writing
Do you want to enter another word? (1/0): 1
```

```
Enter the Word: phone
Enter the definition of the word: scrolling
Do you want to enter another word? (1/0): 0

What would like to do?


Welcome to ADS Assignment 2 - Binary Tree Traversals

What would you like to do?
1. Create a Binary Search Tree
2. Traverse the Tree Inorder Iteratively
3. Traverse the Tree PreOrder Iteratively
4. Traverse the Tree PostOrder Iteratively
5. Traverse it using BFS
6. Create a Copy of the tree Recursively
7. Create a Mirror of the Tree Recursively
8. Exit

2
Traversing through the Binary Tree Inorder Iteratively:
apple : fruit
banana : fruit
bottle : water
charger : charging
keyboard : input
monitor : see
paper : 1
pears : fruit
pen : writing
phone : scrolling

What would like to do?


Welcome to ADS Assignment 2 - Binary Tree Traversals

What would you like to do?
1. Create a Binary Search Tree
2. Traverse the Tree Inorder Iteratively
3. Traverse the Tree PreOrder Iteratively
4. Traverse the Tree PostOrder Iteratively
5. Traverse it using BFS
6. Create a Copy of the tree Recursively
7. Create a Mirror of the Tree Recursively
8. Exit

5
Traversing through the Binary Tree using BFS:
apple : fruit
banana : fruit
keyboard : input
bottle : water
pears : fruit
charger : charging
monitor : see
pen : writing
paper : 1
phone : scrolling
```

```

What would like to do?


Welcome to ADS Assignment 2 - Binary Tree Traversals

What would you like to do?
1. Create a Binary Search Tree
2. Traverse the Tree Inorder Iteratively
3. Traverse the Tree PreOrder Iteratively
4. Traverse the Tree PostOrder Iteratively
5. Traverse it using BFS
6. Create a Copy of the tree Recursively
7. Create a Mirror of the Tree Recursively
8. Exit

6
Creating a copy of the tree
Traversing through the Binary Tree Inorder Iteratively:
apple : fruit
banana : fruit
bottle : water
charger : charging
keyboard : input
monitor : see
paper : 1
pears : fruit
pen : writing
phone : scrolling
Traversing via Breadth First Search:
apple : fruit
banana : fruit
keyboard : input
bottle : water
pears : fruit
charger : charging
monitor : see
pen : writing
paper : 1
phone : scrolling

What would like to do?


Welcome to ADS Assignment 2 - Binary Tree Traversals

What would you like to do?
1. Create a Binary Search Tree
2. Traverse the Tree Inorder Iteratively
3. Traverse the Tree PreOrder Iteratively
4. Traverse the Tree PostOrder Iteratively
5. Traverse it using BFS
6. Create a Copy of the tree Recursively
7. Create a Mirror of the Tree Recursively
8. Exit

7
Creating a mirror of the tree
Traversing through the Binary Tree Inorder Iteratively:
```

```
phone : scrolling
pen : writing
pears : fruit
paper : 1
monitor : see
keyboard : input
charger : charging
bottle : water
banana : fruit
apple : fruit
Traversing via Breadth First Search:
apple : fruit
banana : fruit
keyboard : input
pears : fruit
bottle : water
pen : writing
monitor : see
charger : charging
phone : scrolling
paper : 1

What would like to do?


Welcome to ADS Assignment 2 - Binary Tree Traversals

What would you like to do?
1. Create a Binary Search Tree
2. Traverse the Tree Inorder Iteratively
3. Traverse the Tree PreOrder Iteratively
4. Traverse the Tree PostOrder Iteratively
5. Traverse it using BFS
6. Create a Copy of the tree Recursively
7. Create a Mirror of the Tree Recursively
8. Exit

8
Exiting the program
```

11 Conclusion

Thus, implemented Dictionary using Binary search tree.

12 FAQ

1. **1.Explain application of BST** The Applications of Binary Search Tree are:
   (a) Binary Search Tree is used to implement dictionaries.
   (b) Binary Search Tree is used to implement priority queues.
   (c) Binary Search Tree is used to implement disjoint sets.
   (d) Binary Search Tree is used to implement sorting algorithms.
   (e) Binary Search Tree is used to implement expression trees.
   (f) Binary Search Tree is used to implement Huffman coding.
   (g) Binary Search Tree is used to implement B-trees.
   (h) Binary Search Tree is used to implement red-black trees.

2. **Explain with example deletion of a node having two child.** If a node has two children, then we need to find the inorder successor of the node. The inorder successor is the smallest in the right subtree or the largest in the left subtree. After finding the inorder successor, we copy the contents of the inorder successor to the node and delete the inorder successor. Note that the inorder predecessor can also be used.

   An Example would be:

```
Let us consider the following BST as an example.
         50
        / \
        30 70
    / \ / \
    20 40 60 80
/ \ / \
10 25 45 65
Deleting 30 will be done in following steps.
1. Find inorder successor of 30.
2. Copy contents of the inorder successor to 30.
3. Delete the inorder successor.
4. Since inorder successor is 40 which has no left child, we simply make right child of


         50
        / \
    40 70
    / \ / \
20 25 60 80
```

3. **Define skewed binary tree.** A binary tree is said to be skewed if all of its nodes have only one child. A skewed binary tree can be either left or right skewed. A left skewed binary tree is a binary tree in which all the nodes have only left child. A right skewed binary tree is a binary tree in which all the nodes have only right child.