

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

<b>1 Objectives</b>	<b>1</b>
<b>2 Problem Statement</b>	<b>1</b>
<b>3 Theory</b>	<b>1</b>
3.1 Binary Search Tree . . . . .	1
3.2 Breadth First Traversal . . . . .	1
3.3 Different operations on binary search tree.(copy ,mirror image and delete) . . . . .	1
<b>4 Platform</b>	<b>1</b>
<b>5 Input</b>	<b>1</b>
<b>6 Output</b>	<b>1</b>
<b>7 Test Conditions</b>	<b>2</b>
<b>8 Pseudo Code</b>	<b>2</b>
<b>9 Time Complexity</b>	<b>2</b>
<b>10 Code</b>	<b>2</b>
10.1 Program . . . . .	2
10.2 Input and Output . . . . .	8
<b>11 Conclusion</b>	<b>11</b>
<b>12 FAQ</b>	<b>12</b>

## 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

```
1 #include <iostream>
2 #include <queue>
3 #include <stack>
4 #include <string.h>
5 using namespace std;
6
7 class WordNode
8 {
9     string word;
10    string definition;
11    WordNode *left;
12    WordNode *right;
13    friend class BinarySearchTree;
14 };
15
16 class BinarySearchTree
17 {
18 public:
19     WordNode *root;
20     BinarySearchTree()
21     {
22         root = NULL;
23     }
24     void create_root()
25     {
26         root = new WordNode;
27         cout << "Enter the data: " << endl;
28         cin >> root->word;
29         cin >> root->definition;
30         root->left = NULL;
31         root->right = NULL;
32         create_recursive(root);
33     }
34     void create_recursive(WordNode *Node)
35     {
36         int choice = 0;
37         WordNode *new_node;
38         cout << "Enter if you want to enter a left node (1/0): "
39              << "for the node -- " << Node->word << ": " << Node->definition << "
40              -- ";
41         cin >> choice;
42         if (choice == 1)
```

```
42     {
43         new_node = new WordNode;
44         cout << "Enter the data: ";
45         cin >> new_node->word;
46         cin >> new_node->definition;
47         Node->left = new_node;
48         create_recursive(new_node);
49     }
50     cout << "Enter if you want to enter a right node (1/0): "
51         << "for the node -- " << Node->word << ": " << Node->definition << "
-- ";
52     cin >> choice;
53     if (choice == 1)
54     {
55         new_node = new WordNode;
56         cout << "Enter the data: ";
57         cin >> new_node->word;
58         cin >> new_node->definition;
59         Node->right = new_node;
60         create_recursive(new_node);
61     }
62 }
63 void create_root_and_tree_iteratively()
64 {
65     WordNode *temp, *current_word;
66     int choice = 0;
67     root = new WordNode;
68     cout << "Enter the Word: ";
69     cin >> root->word;
70     cout << "Enter the definition of the word: ";
71     cin >> root->definition;
72     bool flag = false;
73     cout << "Do you want to enter more Nodes? (0/1) " << endl;
74     cin >> choice;
75     while (choice == 1)
76     {
77         temp = root;
78         flag = false;
79         current_word = new WordNode;
80         cout << "Enter the Word: ";
81         cin >> current_word->word;
82         cout << "Enter the definition of the word: ";
83         cin >> current_word->definition;
84
85         while (!flag)
86         {
87             if (strcmp(current_word->word.c_str(), temp->word.c_str()) < 0)
88             {
89                 if (temp->left == NULL)
90                 {
91                     temp->left = current_word;
92                     flag = true;
93                 }
94                 else
95                 {
96                     temp = temp->left;
97                 }
98             }
99             else
```

```
100         {
101             if (temp->right == NULL)
102             {
103                 temp->right = current_word;
104                 flag = true;
105             }
106             else
107             {
108                 temp = temp->right;
109             }
110         }
111     }
112     cout << "Do you want to enter another word? (1/0): ";
113     cin >> choice;
114 }
115 }
116
117 // breadth First Search using queue.
118 void bfs()
119 {
120     WordNode *temp = root;
121     queue<WordNode *> q;
122     q.push(temp);
123     while (!q.empty())
124     {
125         temp = q.front();
126         q.pop();
127         cout << temp->word << " : " << temp->definition << endl;
128         if (temp->left != NULL)
129         {
130             q.push(temp->left);
131         }
132         if (temp->right != NULL)
133         {
134             q.push(temp->right);
135         }
136     }
137 }
138
139 WordNode *create_copy_recursive(WordNode *temp)
140 {
141     if (temp == NULL)
142     {
143         return NULL;
144     }
145     else
146     {
147         WordNode *new_node = new WordNode;
148         new_node->word = temp->word;
149         new_node->definition = temp->definition;
150         new_node->left = create_copy_recursive(temp->left);
151         new_node->right = create_copy_recursive(temp->right);
152         return new_node;
153     }
154 }
155
156 void mirror_recursive(WordNode *temp)
157 {
158     if (temp == NULL)
```

```
159     {
160         return;
161     }
162     else
163     {
164         WordNode *temp1;
165         // Swapping
166         temp1 = temp->left;
167         temp->left = temp->right;
168         temp->right = temp1;
169
170         // Recursively calling the function
171         mirror_recursive(temp->left);
172         mirror_recursive(temp->right);
173     }
174 }
175
176 WordNode *create_mirror_tree_recursive()
177 {
178     WordNode *mirror_tree = create_copy_recursive(root);
179     mirror_recursive(mirror_tree);
180     return mirror_tree;
181 }
182
183 void inorder_iterative(WordNode *temp)
184 {
185     if (!temp)
186     {
187         return;
188     }
189
190     stack<WordNode *> s;
191     WordNode *current = temp;
192
193     while (current != NULL || s.empty() == false)
194     {
195         while (current != NULL)
196         {
197             s.push(current);
198             current = current->left;
199         }
200         current = s.top();
201         s.pop();
202         cout << current->word << " : " << current->definition << endl;
203         current = current->right;
204     }
205 }
206 void preorder_iterative(WordNode *temp)
207 {
208     if (!temp)
209     {
210         return;
211     }
212
213     stack<WordNode *> s;
214     s.push(temp);
215
216     while (s.empty() == false)
217     {
```

```
218     WordNode *current = s.top();
219     cout << current->word << " : " << current->definition << endl;
220     s.pop();
221
222     if (current->right)
223     {
224         s.push(current->right);
225     }
226     if (current->left)
227     {
228         s.push(current->left);
229     }
230 }
231 }
232 void postorder_iterative(WordNode *temp)
233 {
234     if (!temp)
235     {
236         return;
237     }
238
239     stack<WordNode *> s1;
240     stack<WordNode *> s2;
241
242     s1.push(temp);
243
244     while (s1.empty() == false)
245     {
246         WordNode *current = s1.top();
247         s1.pop();
248         s2.push(current);
249
250         if (current->left)
251         {
252             s1.push(current->left);
253         }
254         if (current->right)
255         {
256             s1.push(current->right);
257         }
258     }
259     while (s2.empty() == false)
260     {
261         WordNode *current = s2.top();
262         cout << current->word << " : " << current->definition << endl;
263         s2.pop();
264     }
265 }
266 };
267
268 int main()
269 {
270     int choice = 0;
271     BinarySearchTree main_tree, mirror_tree, copy_tree;
272
273     while (choice != 8)
274     {
275         cout << "\nWhat would like to do? " << endl;
276         cout << "\n\nWelcome to ADS Assignment 2 - Binary Tree Traversals\n\nWhat
```



```
would you like to do? " << endl;
277     cout << "1. Create a Binary Search Tree"
278         << endl;
279     cout << "2. Traverse the Tree Inorder Iteratively"
280         << endl;
281     cout << "3. Traverse the Tree PreOrder Iteratively"
282         << endl;
283     cout << "4. Traverse the Tree PostOrder Iteratively"
284         << endl;
285     cout << "5. Traverse it using BFS"
286         << endl;
287     cout << "6. Create a Copy of the tree Recursively"
288         << endl;
289     cout << "7. Create a Mirror of the Tree Recursively"
290         << endl;
291     cout << "8. Exit" << endl
292         << endl;
293
294     cin >> choice;
295     switch (choice)
296     {
297     case 1:
298         main_tree.create_root_and_tree_iteratively();
299         break;
300     case 2:
301         cout << "Traversing through the Binary Tree Inorder Iteratively: " <<
endl;
302         main_tree.inorder_iterative(main_tree.root);
303         break;
304     case 3:
305         cout << "Traversing through the Binary Tree PreOrder Iteratively: " <<
endl;
306         main_tree.preorder_iterative(main_tree.root);
307         break;
308     case 4:
309         cout << "Traversing through the Binary Tree PostOrder Iteratively: "
<< endl;
310         main_tree.postorder_iterative(main_tree.root);
311         break;
312     case 5:
313         cout << "Traversing through the Binary Tree using BFS: " << endl;
314         main_tree.bfs();
315         break;
316     case 6:
317         cout << "Creating a copy of the tree" << endl;
318         copy_tree.root = copy_tree.create_copy_recursive(main_tree.root);
319         cout << "Traversing through the Binary Tree Inorder Iteratively: " <<
endl;
320         copy_tree.inorder_iterative(copy_tree.root);
321         cout << "Traversing via Breadth First Search: " << endl;
322         copy_tree.bfs();
323         break;
324     case 7:
325         cout << "Creating a mirror of the tree" << endl;
326         mirror_tree.root = main_tree.create_mirror_tree_recursive();
327         cout << "Traversing through the Binary Tree Inorder Iteratively: " <<
endl;
328         mirror_tree.inorder_iterative(mirror_tree.root);
329         cout << "Traversing via Breadth First Search: " << endl;
```

```
330         mirror_tree.bfs();
331         break;
332     case 8:
333         cout << "Exiting the program" << endl;
334         break;
335     default:
336         cout << "Invalid Choice" << endl;
337         break;
338     }
339 }
340 }
```

### 10.2 Input and Output

```
1 What would like to do?
2
3
4 Welcome to ADS Assignment 2 - Binary Tree Traversals
5
6 What would you like to do?
7 1. Create a Binary Search Tree
8 2. Traverse the Tree Inorder Iteratively
9 3. Traverse the Tree PreOrder Iteratively
10 4. Traverse the Tree PostOrder Iteratively
11 5. Traverse it using BFS
12 6. Create a Copy of the tree Recursively
13 7. Create a Mirror of the Tree Recursively
14 8. Exit
15
16 1
17 Enter the Word: apple
18 Enter the definition of the word: fruit
19 Do you want to enter more Nodes? (0/1)
20
21 1
22 Enter the Word: banana
23 Enter the definition of the word: fruit
24 Do you want to enter another word? (1/0): 1
25 Enter the Word: keyboard
26 Enter the definition of the word: input
27 Do you want to enter another word? (1/0): 1
28 Enter the Word: pears
29 Enter the definition of the word: fruit
30 Do you want to enter another word? (1/0): 1
31 Enter the Word: bottle
32 Enter the definition of the word: water
33 Do you want to enter another word? (1/0): 1
34 Enter the Word: charger
35 Enter the definition of the word: charging
36 Do you want to enter another word? (1/0): 1
37 Enter the Word: monitor
38 Enter the definition of the word: see
39 Do you want to enter another word? (1/0): 1
40 Enter the Word: paper
41 Enter the definition of the word: 1
42 Do you want to enter another word? (1/0): 1
43 Enter the Word: pen
44 Enter the definition of the word: writing
45 Do you want to enter another word? (1/0): 1
```

## *Advanced Data Structures - Assignment 3*

---

```
46 Enter the Word: phone
47 Enter the definition of the word: scrolling
48 Do you want to enter another word? (1/0): 0
49
50 What would like to do?
51
52
53 Welcome to ADS Assignment 2 - Binary Tree Traversals
54
55 What would you like to do?
56 1. Create a Binary Search Tree
57 2. Traverse the Tree Inorder Iteratively
58 3. Traverse the Tree PreOrder Iteratively
59 4. Traverse the Tree PostOrder Iteratively
60 5. Traverse it using BFS
61 6. Create a Copy of the tree Recursively
62 7. Create a Mirror of the Tree Recursively
63 8. Exit
64
65 2
66 Traversing through the Binary Tree Inorder Iteratively:
67 apple : fruit
68 banana : fruit
69 bottle : water
70 charger : charging
71 keyboard : input
72 monitor : see
73 paper : 1
74 pears : fruit
75 pen : writing
76 phone : scrolling
77
78 What would like to do?
79
80
81 Welcome to ADS Assignment 2 - Binary Tree Traversals
82
83 What would you like to do?
84 1. Create a Binary Search Tree
85 2. Traverse the Tree Inorder Iteratively
86 3. Traverse the Tree PreOrder Iteratively
87 4. Traverse the Tree PostOrder Iteratively
88 5. Traverse it using BFS
89 6. Create a Copy of the tree Recursively
90 7. Create a Mirror of the Tree Recursively
91 8. Exit
92
93 5
94 Traversing through the Binary Tree using BFS:
95 apple : fruit
96 banana : fruit
97 keyboard : input
98 bottle : water
99 pears : fruit
100 charger : charging
101 monitor : see
102 pen : writing
103 paper : 1
104 phone : scrolling
```

## Advanced Data Structures - Assignment 3

---

```
105
106 What would like to do?
107
108
109 Welcome to ADS Assignment 2 - Binary Tree Traversals
110
111 What would you like to do?
112 1. Create a Binary Search Tree
113 2. Traverse the Tree Inorder Iteratively
114 3. Traverse the Tree PreOrder Iteratively
115 4. Traverse the Tree PostOrder Iteratively
116 5. Traverse it using BFS
117 6. Create a Copy of the tree Recursively
118 7. Create a Mirror of the Tree Recursively
119 8. Exit
120
121 6
122 Creating a copy of the tree
123 Traversing through the Binary Tree Inorder Iteratively:
124 apple : fruit
125 banana : fruit
126 bottle : water
127 charger : charging
128 keyboard : input
129 monitor : see
130 paper : 1
131 pears : fruit
132 pen : writing
133 phone : scrolling
134 Traversing via Breadth First Search:
135 apple : fruit
136 banana : fruit
137 keyboard : input
138 bottle : water
139 pears : fruit
140 charger : charging
141 monitor : see
142 pen : writing
143 paper : 1
144 phone : scrolling
145
146 What would like to do?
147
148
149 Welcome to ADS Assignment 2 - Binary Tree Traversals
150
151 What would you like to do?
152 1. Create a Binary Search Tree
153 2. Traverse the Tree Inorder Iteratively
154 3. Traverse the Tree PreOrder Iteratively
155 4. Traverse the Tree PostOrder Iteratively
156 5. Traverse it using BFS
157 6. Create a Copy of the tree Recursively
158 7. Create a Mirror of the Tree Recursively
159 8. Exit
160
161 7
162 Creating a mirror of the tree
163 Traversing through the Binary Tree Inorder Iteratively:
```

```
164 phone : scrolling
165 pen : writing
166 pears : fruit
167 paper : 1
168 monitor : see
169 keyboard : input
170 charger : charging
171 bottle : water
172 banana : fruit
173 apple : fruit
174 Traversing via Breadth First Search:
175 apple : fruit
176 banana : fruit
177 keyboard : input
178 pears : fruit
179 bottle : water
180 pen : writing
181 monitor : see
182 charger : charging
183 phone : scrolling
184 paper : 1
185
186 What would like to do?
187
188
189 Welcome to ADS Assignment 2 - Binary Tree Traversals
190
191 What would you like to do?
192 1. Create a Binary Search Tree
193 2. Traverse the Tree Inorder Iteratively
194 3. Traverse the Tree PreOrder Iteratively
195 4. Traverse the Tree PostOrder Iteratively
196 5. Traverse it using BFS
197 6. Create a Copy of the tree Recursively
198 7. Create a Mirror of the Tree Recursively
199 8. Exit
200
201 8
202 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.