CCC 204 Data Structures and Algorithms LABORATORY
REPORT :
LAB 9# - BST

Samano, Rajel Johann
Information Technology Program, CABECS
Colegio San Agustin – Bacolod

I. INTRODUCTION

For Laboratory Activity Number Nine is to follow the objectives of the exercises and answer questions. The Three main objectives as follows:

- **Define Binary Search Trees (BST)**
- **Recognize characteristics and applications of BST**
- **Observe, analyze and run BST implementations in code**

II. IMPLEMENTATION / APPROACH

Figure 1-7. LA9_codeTasks.c with output

```c
#include <stdio.h>
#include <stdlib.h>

struct node {
  int key;
  struct node *left, *right;
};

// Create a node
struct node *newNode(int item) {
  struct node *temp = (struct node *)malloc(sizeof(struct node));
  temp->key = item;
  temp->left = temp->right = NULL;
  return temp;
}

// Function Prototype
void inorder(struct node *root);
struct node *insert(struct node *node, int key);
struct node *insert(struct node *node, int key);
struct node *minValueNode(struct node *node);
struct node *deleteNode(struct node *root, int key);
struct node* search(struct node *root, int key);
void BSTmenu();
// Driver code
int main() {
  int key;
  int generate = 1;
  char input[2];
  struct node *root = NULL;

  while (generate) {
    char BSTselection;
    do {
      BSTmenu();
      printf("Input: ");
      scanf(" %c", &BSTselection);

      switch (BSTselection) {
        case '1':
          printf("\nInsert Value to BST: ");
          scanf("%d", &key);
          root = insert(root, key);
          break;

        case '2':
          printf("\nSearch Value in BST: ");
          scanf("%d", &key);
          struct node *result = search(root, key);
          printf("\n");
          if (result != NULL){
            printf("\n[]==============================[]");
            printf("\n Element with Key %d: Found", result->key);
            printf("\n[]==============================[]\n");
          }else{
            printf("\n[]==============================[]");
            printf("\n Element with Key %d: Not Found", result->key);
            printf("\n[]==============================[]\n");
          }
          break;

        case '3':
          printf("\nDelete Value in BST: ");
          scanf("%d", &key);
          root = deleteNode(root, key);
          printf("\n");
          printf("\n[]==============================[]");
          printf("\nAfter Deleting: ");
          inorder(root);
           printf("\n[]==============================[]\n");
          break;

        case '4':
          printf("\n[]==============================[]");
          printf("\n Inorder traversal: ");
          inorder(root);
          printf("\n[]==============================[]\n");
          break;

        case '5':
          break;

        default:
          printf("Invalid choice. Please enter a valid option.\n");
          break;
      }
    } while (BSTselection != '5');

    printf("\nPress any key to regenerate again. Press 'x' to quit: ");
    scanf(" %1s", input);

    if (input[0] == 'x' || input[0] == 'X') {
      generate = 0;
      printf("\n");
    }
  }

  return 0;
}
```

```c
// Inorder Traversal
void inorder(struct node *root) {
  if (root != NULL) {
    // Traverse left
    inorder(root->left);

    // Traverse root
    printf("-> %d ", root->key);

    // Traverse right
    inorder(root->right);
  }
}

// Insert a node
struct node *insert(struct node *node, int key) {
  // Return a new node if the tree is empty
  if (node == NULL) return newNode(key);

  // Traverse to the right place and insert the node
  if (key < node->key)
    node->left = insert(node->left, key);
  else
    node->right = insert(node->right, key);

  return node;
}

// Find the inorder successor
struct node *minValueNode(struct node *node) {
  struct node *current = node;

  // Find the leftmost leaf
  while (current && current->left != NULL)
    current = current->left;

  return current;
}

// Deleting a node
struct node *deleteNode(struct node *root, int key) {
  // Return if the tree is empty
  if (root == NULL) return root;

  // Find the node to be deleted
  if (key < root->key)
    root->left = deleteNode(root->left, key);
  else if (key > root->key)
    root->right = deleteNode(root->right, key);
  else {
    // If the node is with only one child or no child
    if (root->left == NULL) {
      struct node *temp = root->right;
      free(root);
      return temp;
    } else if (root->right == NULL) {
      struct node *temp = root->left;
      free(root);
      return temp;
    }

    // If the node has two children
    struct node *temp = minValueNode(root->right);

    // Place the inorder successor in position of the node to be deleted
    root->key = temp->key;

    // Delete the inorder successor
    root->right = deleteNode(root->right, temp->key);
  }
  return root;
}

//search
struct node* search(struct node *root, int key){
  printf("\n[]==============================[]");
  printf("\n Visiting elements: ");
  while(root->key != key) {
    if(root != NULL) {
      printf("-> %d ",root->key);

      //go to left tree
      if(root->key > key) {
        root = root->left;
      }//else go to right tree
      else {
        root = root->right;
      }

      //not found
      if(root == NULL) {
        return NULL;
      }
    }
  }
  printf("\n[]==============================[]");
  return root;
}
```

```c
void BSTmenu() {
  printf("\n====================================\n");
  printf("||<<<<        LA9 Task Code        >>>>||\n");
  printf("====================================\n");
  printf("||     [1] Insert    ||    [2] Search  ||\n");
  printf("====================================\n");
  printf("||     [3] Delete    ||    [4] Display ||\n");
  printf("====================================\n");
  printf("||              [5] Exit              ||\n");
  printf("====================================\n");
  printf("||<<<<        Select Command       >>>>||\n");
  printf("====================================\n\n");
}
```

```
====================================
||<<<<        LA9 Task Code        >>>>||
====================================
||     [1] Insert    ||    [2] Search  ||
====================================
||     [3] Delete    ||    [4] Display ||
====================================
||              [5] Exit              ||
====================================
||<<<<        Select Command       >>>>||
====================================

Input: 1

Insert Value to BST: 8

====================================
||<<<<        LA9 Task Code        >>>>||
====================================
||     [1] Insert    ||    [2] Search  ||
====================================
||     [3] Delete    ||    [4] Display ||
====================================
||              [5] Exit              ||
====================================
||<<<<        Select Command       >>>>||
====================================

Input: 1

Insert Value to BST: 9

====================================
||<<<<        LA9 Task Code        >>>>||
====================================
||     [1] Insert    ||    [2] Search  ||
====================================
||     [3] Delete    ||    [4] Display ||
====================================
||              [5] Exit              ||
====================================
||<<<<        Select Command       >>>>||
====================================

Input: 4

[]==============================[]
 Inorder traversal: -> 8 -> 9
[]==============================[]

====================================
||<<<<        LA9 Task Code        >>>>||
====================================
||     [1] Insert    ||    [2] Search  ||
====================================
||     [3] Delete    ||    [4] Display ||
====================================
||              [5] Exit              ||
====================================
||<<<<        Select Command       >>>>||
====================================

Input: 2

Search Value in BST: 9

[]==============================[]
 Visiting elements: -> 8
[]==============================[]

[]==============================[]
 Element with Key 9: Found
[]==============================[]
```

My approach towards the problem was first copy that code that our teacher has provided to us and modify it by copying the code that I used for stacks and queues menu from the past activities while changing some functions output and moving the functions in the code to the bottom and creating function prototypes so that it would run properly.

III. EXPERIMENTAL FINDINGS / DISCUSSIONS

What I found through various scanning from websites was that BST is very efficient as it does not use additional memory for pointers or other data structures. While compared to others in terms of searching, inserting, and deleting. (Depends on the specific characteristics of the data and the way the tree is balanced.)

IV. CONCLUSIONS

To conclude this laboratory activity, I have a somewhat a grasped on how to use the searching, inserting, and deleting of BST or Binary Search Tree. While also kind of knowing how to make a BST from scratch.

V. References:

https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-binary-search-tree/

https://www.javatpoint.com/binary-search-tree

https://www.tutorialspoint.com/data_structures_algorithms/binary_search_tree.htm