**Binary Search Tree**

Problem Statement:

Write Program to do the following:

1) Insert n integers in BST, where n integers are generated at random. With different values of n, ie, 50, 100, 1000, 10000, 25000, 50000 etc. create the BST. For each n value, run your program for 50 different permutations of random numbers. For each n value compute the total height to insert the n elements in each of the 50 cases and report the

average height. Verify if it follows O(n log n).

2) Search for an element.

3) Delete an element ( consider all possible cases)

4) Print the BST in inorder traversal (Iterative)

• Output example :

The average height for 50 is: 11.14

The average height for 100 is: 13.44

The average height for 1000 is: 21.84

The average height for 10000 is: 31.20

The average height for 25000 is: 35.22

The average height for 50000 is: 37.86

Press 1 to insert

Press 2 to search

Press 3 to delete

Press 4 to display in inorder

Press 5 to terminate

1

Enter Value: 12

1

Enter Value: 15

1

Enter Value: 23

4

12->15->23->

2

Enter Key: 12

Found 12

3

Enter Key: 15

Node Deleted

4

12->23->

5

Terminated

Proposed C Code:

/\* ------- main.c ------- \*/

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#include <time.h>

typedef struct N

{

int val;

struct N \*right;

struct N \*left;

} Node;

Node \*root = NULL;

// Creating Stack

Node \*\*stack;

int top = -1;

int s = 100000;

void init()

{

stack = (Node \*\*)malloc(s \* sizeof(Node \*));

}

void push(Node \*root)

{

stack[++top] = root;

}

Node \*pop()

{

if (top == -1)

{

printf("Stack is Empty");

return NULL;

}

return stack[top--];

}

bool isEmpty()

{

return top == -1;

}

Node \*createNode(int val)

{

Node \*branch = (Node \*)malloc(sizeof(Node));

branch->val = val;

branch->left = NULL;

branch->right = NULL;

return branch;

}

void insert(int val)

{

Node \*new = createNode(val);

Node \*x = root;

Node \*y = NULL;

while (x != NULL)

{

y = x;

if (x->val > val)

{

x = x->left;

}

else

{

x = x->right;

}

}

if (y == NULL)

{

root = new;

}

else if (val < y->val)

{

y->left = new;

}

else

{

y->right = new;

}

}

int search(Node \*root, int val)

{

if (root == NULL)

{

printf("Not Found ");

return -1;

}

if (root->val == val)

{

printf("Found ");

return root->val;

}

if (root->val > val)

{

return search(root->left, val);

}

else

{

return search(root->right, val);

}

}

int minValue(Node \*root)

{

int minv = root->val;

while (root->left != NULL)

{

minv = root->left->val;

root = root->left;

}

return minv;

}

Node \*deleteNode(Node \*root, int key)

{

if (root == NULL)

{

printf("Key not found\n");

return root;

}

if (key < root->val)

{

root->left = deleteNode(root->left, key);

}

else if (key > root->val)

{

root->right = deleteNode(root->right, key);

}

else

{

if (root->left == NULL)

{

return root->right;

}

else if (root->right == NULL)

{

return root->left;

}

root->val = minValue(root->right);

root->right = deleteNode(root->right, root->val);

}

return root;

}

void delete (int key)

{

root = deleteNode(root, key);

printf("Node Deleted\n");

}

int height(Node \*root)

{

if (root == NULL)

return 0;

else

{

/\* compute the depth of each subtree \*/

int lDepth = height(root->left);

int rDepth = height(root->right);

/\* use the larger one \*/

if (lDepth > rDepth)

return (lDepth + 1);

else

return (rDepth + 1);

}

}

void display(Node \*root)

{

if (root == NULL)

{

return;

}

init();

Node \*curr = root;

while (curr != NULL || !isEmpty())

{

while (curr != NULL)

{

push(curr);

curr = curr->left;

}

curr = pop();

printf("%d->", curr->val);

curr = curr->right;

}

printf("\n");

}

void swap(int \*a, int \*b)

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void randomPer(int arr1[], int n)

{

srand(time(NULL));

for (int i = n - 1; i > 0; i--)

{

int j = rand() % (i + 1);

swap(&arr1[i], &arr1[j]);

}

}

void ArrayDisplay(int arr1[], int n)

{

printf("The shuffled elements in the array are: \n");

for (int i = 0; i < n; i++)

printf("%d ", arr1[i]);

printf("\n");

}

int main()

{

int size[] = {50, 100, 1000, 10000, 25000, 50000};

srand(time(0));

for (int i = 0; i < 6; i++)

{

int \*arr = (int \*)malloc(size[i] \* sizeof(int));

for (int j = 0; j < size[i]; j++)

{

arr[j] = rand();

}

int x = 0;

for (int k = 0; k < 50; k++)

{

randomPer(arr, size[i]);

root = NULL;

for (int j = 0; j < size[i]; j++)

{

insert(arr[j]);

}

x += height(root);

}

printf("The average height for %d is: %.2f\n", size[i], (float)x / 50);

}

root = NULL;

printf("\nPress 1 to insert \nPress 2 to search \nPress 3 to delete \nPress 4 to display in inorder\nPress 5 to terminate\n");

int p, c;

do

{

scanf("%d", &p);

switch (p)

{

case 1:

printf("Enter Value: ");

scanf("%d", &c);

insert(c);

break;

case 2:

printf("Enter Key: ");

scanf("%d", &c);

printf("%d\n", search(root, c));

break;

case 3:

printf("Enter Key: ");

scanf("%d", &c);

delete (c);

break;

case 4:

display(root);

break;

case 5:

printf("Terminated\n");

break;

}

} while (p != 5);

return 0;

}

/\* ---------------------- \*/

Conclusion:

The proposed algorithm has a runtime of O(nlogn), where n is the number of nodes.

Limitations and assumptions for this algorithm include:

1.Here insertion and deletion are take place at O(nlogn) time in stead of constant time(O(1)).

2.Here as >50 nodes the inorder tree will be very large so I am not printing them.

3.I created a menu driven program by which you can check the search, display and the delete operation working currently.