

#include <iostream>

using namespace std;

#define nullptr NULL

// Data structure to store a Binary Search Tree node

struct Node

{

int data;

Node \*left, \*right;

};

// Function to create a new binary tree node having given key

Node\* newNode(int key)

{

Node\* node = new Node;

node->data = key;

node->left = node->right = nullptr;

return node;

}

// Function to perform inorder traversal of the tree

void inorder(Node\* root)

{

if (root == nullptr)

return;

inorder(root->left);

printf("%d,",root->data);

inorder(root->right);

}

// Recursive function to insert a key into BST

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

{

// if the root is null, create a new node and return it

if (root == nullptr)

return newNode(key);

// if given key is less than the root node, recur for left subtree

if (key < root->data)

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

// if given key is more than the root node, recur for right subtree

else

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

return root;

}

// This function finds predecessor and successor of key in BST.

// It sets pre and suc as predecessor and successor respectively

void findPreSuc(Node\* root, Node\*& pre, Node\*& suc, int key)

{

// Base case

if (root == NULL) return ;

// If key is present at root

if (root->data == key)

{

// the maximum value in left subtree is predecessor

if (root->left != NULL)

{

Node\* tmp = root->left;

while (tmp->right)

tmp = tmp->right;

pre = tmp ;

}

// the minimum value in right subtree is successor

if (root->right != NULL)

{

Node\* tmp = root->right ;

while (tmp->left)

tmp = tmp->left ;

suc = tmp ;

}

return ;

}

// If key is smaller than root's key, go to left subtree

if (root->data > key)

{

suc = root ;

findPreSuc(root->left, pre, suc, key) ;

}

else // go to right subtree

{

pre = root ;

findPreSuc(root->right, pre, suc, key) ;

}

}

// Recursive function to search in given BST

void search(Node\* root, int key, Node\* parent)

{

// if key is not present in the key

if (root == nullptr)

{

cout << "Key Not found";

return;

}

// if key is found

if (root->data == key)

{

if (parent == nullptr)

cout << "The node with key " << key << " is root node";

else if (key < parent->data)

cout << "Given key is left node of node with key "

<< parent->data;

else cout << "Given key is right node of node with key "

<< parent->data;

return;

}

// if given key is less than the root node, recur for left subtree

// else recur for right subtree

if (key < root->data)

return search(root->left, key, root);

return search(root->right, key, root);

}

int inOrderSuccessor(Node\* root) {

int minimum = root->data;

while (root->left != NULL) {

minimum = root->left->data;

root = root->left;

}

return minimum;

}

Node\* deleteRecursively(Node\* root, int value) {

if (root == NULL)

return root;

if (value < root->data) {

root->left = deleteRecursively(root->left, value);

} else if (value > root->data) {

root->right = deleteRecursively(root->right, value);

} else {

if (root->left == NULL) {

return root->right;

} else if (root->right == NULL)

return root->left;

root->data = inOrderSuccessor(root->right);

root->right = deleteRecursively(root->right, root->data);

}

return root;

}

/\* Compute the "height" of a tree -- the number of

nodes along the longest path from the root node

down to the farthest leaf node.\*/

int height(Node\* node)

{

if (node==NULL)

return 0;

else

{

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

int lheight = height(node->left);

int rheight = height(node->right);

/\* use the larger one \*/

if (lheight > rheight)

return(lheight+1);

else return(rheight+1);

}

}

void printGivenLevel(Node\* root, int level)

{

if (root == NULL)

return;

if (level == 1)

printf("%d ", root->data);

else if (level > 1)

{

printGivenLevel(root->left, level-1);

printGivenLevel(root->right, level-1);

}

}

/\* Print nodes at a given level \*/

/\* Function to print level order traversal a tree\*/

void printLevelOrder(Node\* root)

{

int h = height(root);

int i;

for (i=1; i<=h; i++){ printGivenLevel(root, i); printf("\n");

}

}

int main()

{

Node\* root = nullptr;

int keys[] = {11,66,6,9,40,28,5,88,125,90};

int key1,key2;

for (int i=0;i<10;i++)

root = insert(root, keys[i]);

printLevelOrder(root);

inorder(root);

printf("\n");

printf("Enter element to be searched in BST.\n");

scanf("%d",&key1);

search(root, key1, nullptr);

Node\* pre = NULL, \*suc = NULL;

printf("Enter element to find predecessor and successor in BST.\n");

scanf("%d",&key2);

findPreSuc(root, pre, suc, key2);

if (pre != NULL)

printf("Predecessor is %d\n",pre->data);

else

printf("No Predecessor");

if (suc != NULL)

printf("Successor is %d\n",suc->data);

else

printf("No Successor\n");

printf("After deleting 40\n");

deleteRecursively(root, 40);

printLevelOrder(root);

printf("After deleting 88\n");

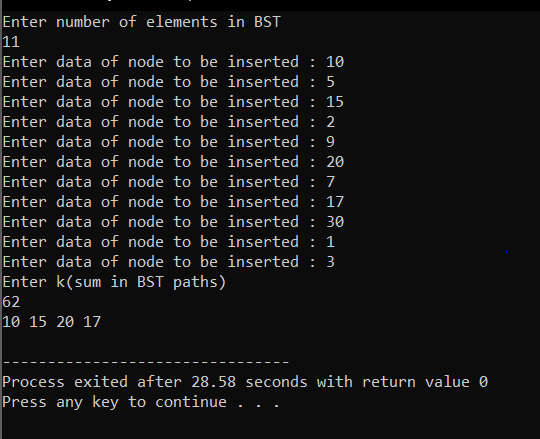
deleteRecursively(root, 88);

printLevelOrder(root);

printf("\n");

return 0;

}



2) #include <stdio.h>

#include <stdlib.h>

#include <bits/stdc++.h>

using namespace std;

#include <vector>

struct btnode

{

int value;

struct btnode \*l;

struct btnode \*r;

}\*root = NULL, \*temp = NULL, \*t2, \*t1;

//utility function to print contents of

//a vector from index i to it's end

void printVector(const vector<int>& v, int i)

{

for (int j=i; j<v.size(); j++)

printf("%d ",v[j]);

printf("\n");

}

/\* To create a node \*/

void create()

{

int data;

printf("Enter data of node to be inserted : ");

scanf("%d", &data);

temp = (struct btnode \*)malloc(1\*sizeof(struct btnode));

temp->value = data;

temp->l = temp->r = NULL;

}

/\* Function to search the appropriate position to insert the new node \*/

void search(struct btnode \*t)

{

if ((temp->value > t->value) && (t->r != NULL)) /\* value more than root node value insert at right \*/

search(t->r);

else if ((temp->value > t->value) && (t->r == NULL))

t->r = temp;

else if ((temp->value < t->value) && (t->l != NULL)) /\* value less than root node value insert at left \*/

search(t->l);

else if ((temp->value < t->value) && (t->l == NULL))

t->l = temp;

}

/\* To insert a node in the tree \*/

void insert()

{

create();

if (root == NULL)

root = temp;

else

search(root);

}

// This function prints all paths that have sum k

void printKPathUtil(struct btnode \*root, vector<int>& path,

int k)

{

int min=9;

// empty node

if (!root)

return;

// add current node to the path

path.push\_back(root->value);

// check if there's any k sum path

// in the left sub-tree.

printKPathUtil(root->l, path, k);

// check if there's any k sum path

// in the right sub-tree.

printKPathUtil(root->r, path, k);

// check if there's any k sum path that

// terminates at this node

// Traverse the entire path as

// there can be negative elements too

int f = 0,count=0;

for (int j=path.size()-1; j>=0; j--)

{

f += path[j];

if (f == k) {

printVector(path, j);

min=path.size();

}

// If path sum is k, print the path

if (f == k && path.size()<min&&count==0) {

printVector(path, j);

min=path.size();

}

count++;}

// Remove the current element from the path

path.pop\_back();

}

// A wrapper over printKPathUtil()

void printKPath(struct btnode \*root, int k)

{

vector<int> path;

printKPathUtil(root, path, k);

}

// Driver code

int main()

{

int n;

printf("Enter number of elements in BST\n");

scanf("%d",&n);

for(int i=1;i<=n;i++)

insert();

printf("Enter k(sum in BST paths)\n");

int k;

scanf("%d",&k);

printKPath(root, k);

return 0;

}