Ministry of Education and Science of Ukraine

National Technical University of Ukraine

«Kyiv Polytechnic Institute. Igor Sikorsky »

Faculty of Informatics and Computer Technologies

Department of Computer Engineering

LAB № 6

from the discipline "Theory of Algorithms"

on the topic «Binary search trees»

PERFORMED BY:

1st year student

group ІП-93

Zavalniuk M.E.

The credit - 9312

Variant – 12

CHECKED:

Associate Professor of OT

c.t.s.,s.r.

Antoniuk А.І.

Kiev - 2020

**TASK**

**Goal:**

transform an input binary tree into a binary search tree. Search for sums of consecutive nodes in a tree.

**Option task:**

1. Convert an input binary tree into a binary search tree.

A binary tree with a fixed structure (that is, links between nodes, their parent, and descendants) is fed into the input. You must rewrite the values ​​of the tree nodes so that:

a) their new values ​​were taken only from the set present in the input tree;

b) the internal structure of the tree (the links between the parent node and the descendant node) was maintained.

2. Search for sums of consecutive nodes in the tree

After the input tree has been converted to a binary search tree, the next task must be resolved. In addition, a certain number of S. is given. In the obtained binary search tree, it is necessary to find all such monotone paths (which do not necessarily come from the root, but all go from top to bottom) that the sum of the values ​​of the nodes belonging to the found paths is equal to S.

**CODE**

**using** System**;**

**using** System**.**Collections**.**Generic**;**

// A binary tree node

**public** class Node **{**

**public** int data**;**

**public** Node left**,** right**;**

**public** Node**(**int d**)**

**{**

data **=** d**;**

left **=** right **=** **null;**

**}**

**}**

**public** class BinarySearchTree **{**

// Root of BST

Node root**;**

// Constructor

BinarySearchTree**()**

**{**

root **=** **null;**

**}**

// Inorder traversal of the tree

void inorder**()**

**{**

inorderUtil**(this.**root**);**

**}**

// Utility function for inorder traversal of the tree

void inorderUtil**(**Node node**)**

**{**

**if** **(**node **==** **null)**

**return;**

inorderUtil**(**node**.**left**);**

Console**.**Write**(**node**.**data **+** " "**);**

inorderUtil**(**node**.**right**);**

**}**

// This method mainly calls insertRec()

void insert**(**int key**)**

**{**

root **=** insertRec**(**root**,** key**);**

**}**

/\* A recursive function to insert a new key in BST \*/

Node insertRec**(**Node root**,** int data**)**

**{**

/\* If the tree is empty, return a new node \*/

**if** **(**root **==** **null)** **{**

root **=** **new** Node**(**data**);**

**return** root**;**

**}**

/\* Otherwise, recur down the tree \*/

**if** **(**data **<** root**.**data**)**

root**.**left **=** insertRec**(**root**.**left**,** data**);**

**else** **if** **(**data **>** root**.**data**)**

root**.**right **=** insertRec**(**root**.**right**,** data**);**

**return** root**;**

**}**

// Method that adds values of given BST into ArrayList and hence returns the ArrayList

List**<**int**>** treeToList**(**Node node**,** List**<**int**>** list**)**

**{**

// Base Case

**if** **(**node **==** **null)**

**return** list**;**

treeToList**(**node**.**left**,** list**);**

list**.**Add**(**node**.**data**);**

treeToList**(**node**.**right**,** list**);**

**return** list**;**

**}**

// method that checks if there is a pair present

bool isPairPresent**(**Node node**,** int target**)**

**{**

// This list a1 is passed as an argument

// in treeToList method

// which is later on filled by the values of BST

List**<**int**>** a1 **=** **new** List**<**int**>();**

// a2 list contains all the values of BST

// returned by treeToList method

List**<**int**>** a2 **=** treeToList**(**node**,** a1**);**

int start **=** 0**;** // Starting index of a2

int end **=** a2**.**Count **-** 1**;** // Ending index of a2

**while** **(**start **<** end**)** **{**

**if** **(**a2**[**start**]** **+** a2**[**end**]** **==** target**)** // Target Found!

**{**

Console**.**WriteLine**(**"Pair Found: " **+** a2**[**start**]** **+** " + " **+** a2**[**end**]** **+** " "

**+** "= " **+** target**);**

**return** **true;**

**}**

**if** **(**a2**[**start**]** **+** a2**[**end**]** **>** target**)** // decrements end

**{**

end**--;**

**}**

**if** **(**a2**[**start**]** **+** a2**[**end**]** **<** target**)** // increments start

**{**

start**++;**

**}**

**}**

Console**.**WriteLine**(**"No such values are found!"**);**

**return** **false;**

**}**

// Driver code

**public** static void Main**(**String**[]** args**)**

**{**

BinarySearchTree tree **=** **new** BinarySearchTree**();**

int**[]** inputArray **=** **{**15**,** 10**,** 20**,** 8**,** 12**,** 16**,** 25**,** 2**,** 3**};**

**foreach** **(**int el **in** inputArray**)**

**{**

tree**.**insert**(**el**);**

**}**

// Input data

Console**.**WriteLine**(**"Enter search number:"**);**

int searchNumber **=** Convert**.**ToInt32**(**Console**.**ReadLine**());**

tree**.**isPairPresent**(**tree**.**root**,** searchNumber**);**

**}**

**}**

**RESULTS OF THE PROGRAM WORK**

The input:

Array = [15, 10, 20, 8, 12, 16, 25, 2, 3]

S = 13

Output array: Pair Found: 3 + 10 = 13.

**CONCLUSIONS**

I got acquainted with the topic of laboratory work.

Have acquired relevant work skills.

An appropriate test program has been developed.

Got the job of binary search trees. Considered several ways to set trees. I understood the benefits of implementing this algorithm