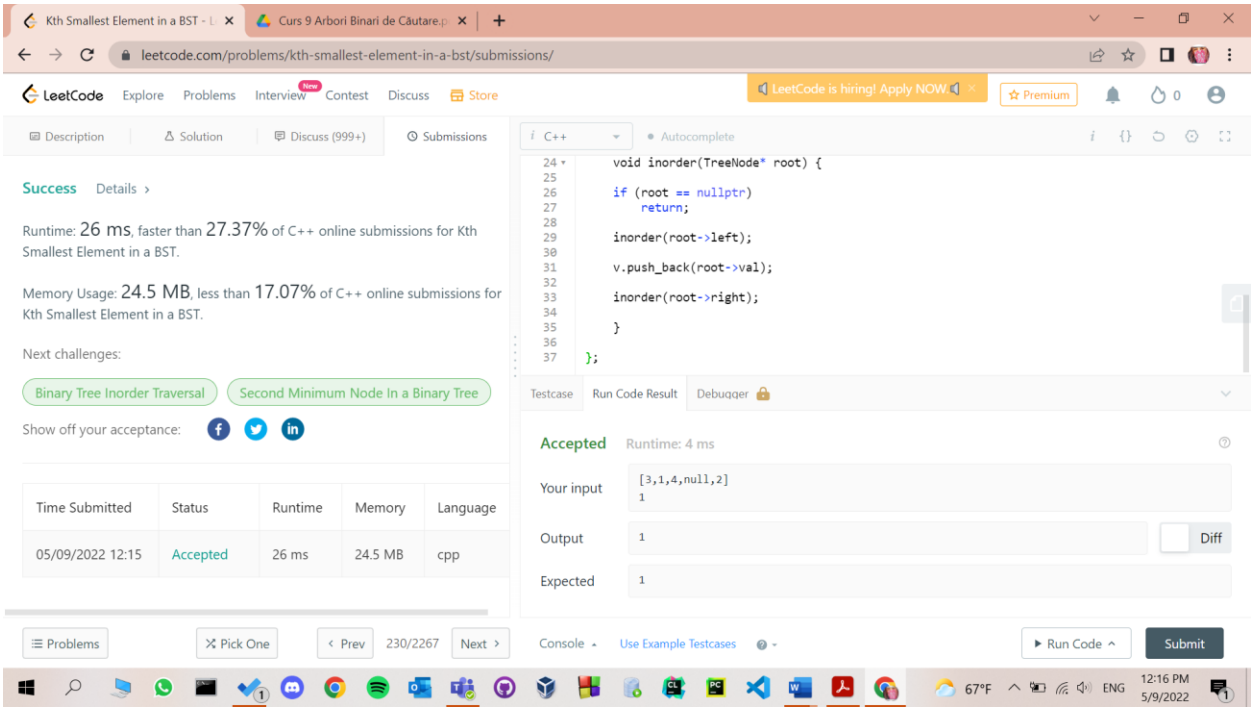
Tema Seminar 6

(Enescu Irina Stefania, Grupa 133)

1. Kth Smallest Element in a BST
   Given the root of a binary search tree, and an integer k, return the kth smallest value (1-indexed) of all the values of the nodes in the tree.
   Input: root = [3, 1, 4, null, 2], k = 1
   Output: 1



SURSA COD:

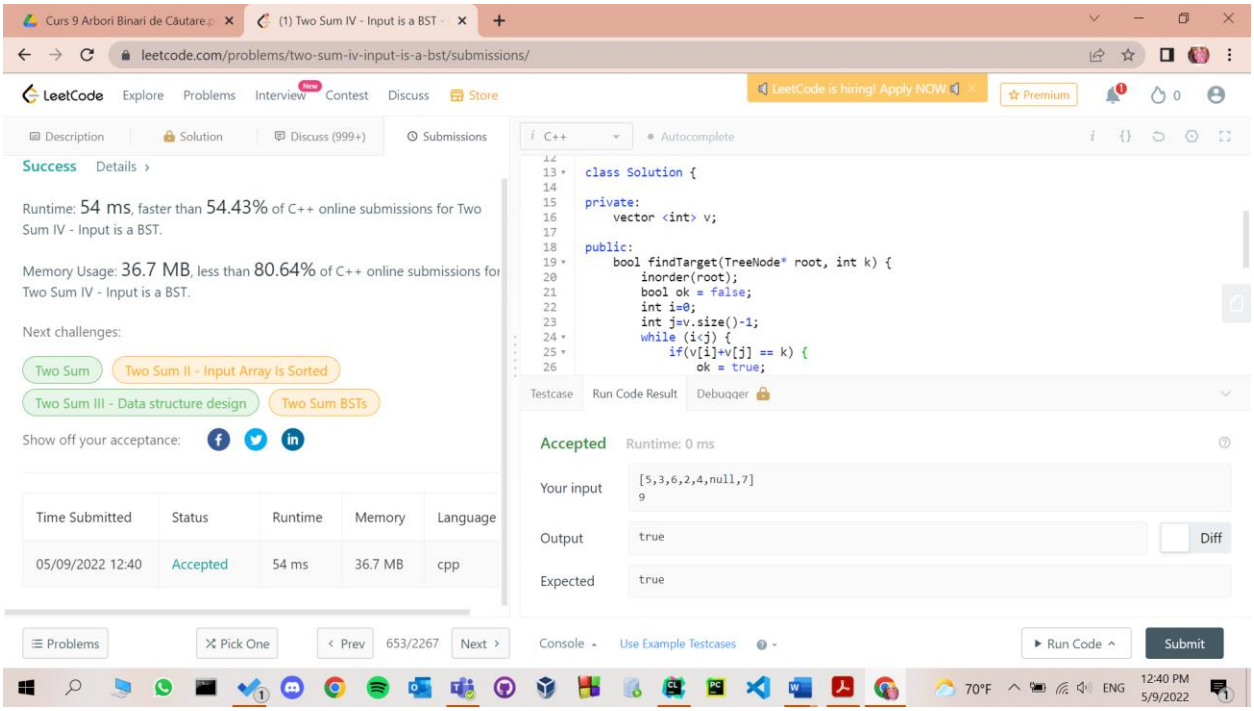
```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */
```

```cpp
class Solution {

private:
    vector <int> v;

public:
    int kthSmallest(TreeNode* root, int k) {
        inorder(root);
        return v[k-1];
    }
    void inorder(TreeNode* root) {
        if (root == nullptr)
            return;
        inorder(root->left);
        v.push_back(root->val);
        inorder(root->right);
    }
};
```

2. Two Sum IV - Input is a BST
   Given the root of a Binary Search Tree and a target number k, return true if there exist two elements in the BST such that their sum is equal to the given target.
   Input: root = [5, 3, 6, 2, 4, null, 7], k = 9
   Output: true



SURSA COD:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
 * };
 */

class Solution {

private:
    vector <int> v;

public:
    bool findTarget(TreeNode* root, int k) {
        inorder(root);
        bool ok = false;
        int i=0;
        int j=v.size()-1;
        while (i<j) {
            if(v[i]+v[j] == k) {
                ok = true;
                break;
            }
            else if(v[i]+v[j] > k) {
                j--;
            }
            else if (v[i]+v[j] < k){
                i++;
            }
        }
        return ok;
    }

    void inorder(TreeNode* root) {
        if (root == nullptr)
            return;
        inorder(root->left);
        v.push_back(root->val);
        inorder(root->right);
    }
};
```

3. Calculati cati arbori binari de cautare distincti se pot crea. 2 1 3 si 2 3 1 sunt acelasi arbore, dar 1 2 3, 1 3 2, 2 1 3, 3 1 2, 3 2 1 sunt toti distincti.

Fie 1, 2, 3, 4, ... n noduri.

Arbore binar de căutare:

- Stânga – elemente mai mici decât rădăcina
- Dreapta – elemente mai mari decât rădăcina

Pentru fiecare nod ales ca rădăcină, rămân n-1 noduri ce nu sunt rădăcini. Aceste noduri trebuie împărțite în noduri care sunt mai mari decât rădăcina și noduri care sunt mai mici decât rădăcina.

- Dacă alegem rădăcina arborelui 1:
  - nici un element nu mai poate fi inserat în stânga
  - n-1 elemente vor fi inserate în dreapta
- Dacă alegem rădăcina arborelui 2:
  - un element va fi inserat in stânga
  - n-2 elemente vor fi inserate în dreapta
- Dacă alegem rădăcina arborelui 3:
  - două elemente vor fi inserate în stânga
  - n-3 elemente vor fi inserate în dreapta

  Așadar, pentru rădăcina i:

  - i-1 elemente vor fi inserate în stânga
  - n-i elemente vor fi inserate în dreapta

Pentru fiecare aranjament în partea stângă și pentru fiecare aranjament în partea dreaptă vom obține un arbore binar de căutare corect cu rădăcina i.

Așadar, pentru rădăcina i vom avea f(i-1)*f(n-i) arbori binari de căutare.

În total: f(n) = f(0)*f(n-1) + f(1)*f(n-2) + f(2)*f(n-3) + ... + f(i-1)*f(n-i) + ... + f(n-1)*f(0)