

# 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

The screenshot displays the LeetCode interface for the problem "Kth Smallest Element in a BST". The left sidebar shows the problem details, including the runtime (26 ms) and memory usage (24.5 MB). The main area shows the C++ code for an inorder traversal. The bottom section shows the submission results, indicating that the code was accepted with a runtime of 4 ms. The input is [3, 1, 4, null, 2] and the output is 1.

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

Time Submitted	Status	Runtime	Memory	Language
05/09/2022 12:15	Accepted	26 ms	24.5 MB	cpp

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:
    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

Success Details >

Runtime: 54 ms, faster than 54.43% of C++ online submissions for Two Sum IV - Input is a BST.

Memory Usage: 36.7 MB, less than 80.64% of C++ online submissions for Two Sum IV - Input is a BST.

Next challenges:

- Two Sum
- Two Sum II - Input Array is Sorted
- Two Sum III - Data structure design
- Two Sum BSTs

Show off your acceptance:

Time Submitted	Status	Runtime	Memory	Language
05/09/2022 12:40	Accepted	54 ms	36.7 MB	cpp

```

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;
            }
        }
        return ok;
    }
};

```

Accepted Runtime: 0 ms

Your input: [5,3,6,2,4,null,7]  
9

Output: true

Expected: true

Run Code ^ Submit

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 în 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  $f(i-1) \cdot f(n-i)$  arbori binari de căutare.

În total:  $f(n) = f(0) \cdot f(n-1) + f(1) \cdot f(n-2) + f(2) \cdot f(n-3) + \dots + f(i-1) \cdot f(n-i) + \dots + f(n-1) \cdot f(0)$