**Tema seminar 6**

1) Calculcati cati arbori binari de cautare distincti se pot crea cu n noduri

Numarul total de arbori binari de cautare distincti care se pot crea cu n noduri este numarul catalan Cn = (2n)! / ((n+1)! \* n!).

Vom lua in considerare toti arborii de cautare binari posibili cu fiecare element ca radacina. Pentru n noduri pentru fiecare alegere de nod ca radacina exista n-1 noduri care nu sunt radacina. Aceste noduri vor si impartite in cele care sunt mai mici decat radacina aleasa si cele care sunt mai mari.

Sa presupunem ca alegem nodul i ca radacina. Atunci exista i -1 noduri mai mici decat i si n – i noduri mai mari decat i. Pentru fiecare din aceste doua seturi de noduri exista un anumit numar de subarbori posibili.

Fie total(n) numarul total de arbori binari de cautare cu n noduri. Numarul total de arbori binari de cautare cu radacina i este total(i – 1) \* total(n – i). Cei doi termeni sunt inmultiti pentru ca aranjamentele din subarborele din stanga si din dreapta sunt indepentente. Astfel, pentru fiecare aranjare din subarborele din stanga si pentru fiecare aranjament din arborele din dreapta se obtine un arbore binar de cautare cu radacina i.

  Insumand posibilitatile pentru fiecare i obtinem numarul total de arbori binari de cautare cu n noduri: .  
  Cazul de bază este total(0) = 1 și total(1) = 1, adică există un arbore binar de cautare gol și unul cu un nod.

2) **653. Two Sum IV - Input is a BST**

void inorder(TreeNode\* root, vector<int> &store\_inorder){

if(root == NULL)

return;

inorder(root->left, store\_inorder);

store\_inorder.push\_back(root->val);

inorder(root->right, store\_inorder);

}

bool findTarget(TreeNode\* root, int k) {

vector<int> store\_inorder;

inorder(root, store\_inorder);

int i = 0, j = store\_inorder.size() -1;

while(i < j){

int sum = store\_inorder[i] + store\_inorder[j];

if(sum == k)

return true;

else if(sum > k)

j--;

else

i++;

}

return false;

}

3) **230. Kth Smallest Element in a BST**

void inorder(TreeNode\* root, vector<int> &store\_inorder){

if(root == NULL)

return;

inorder(root->left, store\_inorder);

store\_inorder.push\_back(root->val);

inorder(root->right, store\_inorder);

}

int kthSmallest(TreeNode\* root, int k) {

vector<int> v;

inorder(root, v);

return v[k-1];

}