**Tema seminar V**

1. ***Dati un algoritm ce transforma un BST intr-un alt BST avand aceeasi multime de chei.***

***( se vor da 2 arbori – T1 = arborele ce trebuie tranformat in T2)***

*Complexitate? O(n) (demonstrat la seminar)*

*Transform T1 intr-un BST degenerat pe stanga;*

*Pe arborele T2 pot fi aplicate rotatii stanga SI dreapta, deci pentru a obtine aceeasi structura si pentru T1 (degenerat ) trebuie sa aplic ambele tipuri ( T1.k > T2.k => leftRotate, altfel rightRotate)*

void TransformToAnotherBST(BType\*\* T1, BType\* T2)

{

if (T2 != NULL)

{

// merg pana cand T1->key coincide cu T2->key

while ((\*T1)->key < T2->key)

{

\*T1 = LeftRotate(\*T1);

}

while ((\*T1)->key > T2->key)

{

\*T1 = RightRotate(\*T1);

}

//

TransformToAnotherBST(&((\*T1)->left), T2->left);

TransformToAnotherBST(&((\*T1)->right), T2->right);

}

}

BType\* TransformToAnotherBSTInit(BType\* T1, BType\* T2)

{

T1 = Transform(T1); // O(n)

// verificam daca T2 are aceleasi chei ca si T1 ~~ n \* h pasi (Aparent!) --> dar pentru unele chei ne vom opri mai devreme => in total n pasi

int valid = HasAllKeys(T1, T2); //O(n)

if (!valid)

{

printf("Date Invalide");

exit(1234);

}

TransformToAnotherBST(&T1, T2);

return T1;

}

1. ***Se da un element x intr-un arbode de statistici de ordine de dimensiune n; determinati al k-lea succesor in O(log n) ;***

***Complexitate:*** OsSel – O(h) ; OsRank – O(h) 🡺 total : O(h) – asumam ca arborele e balansat => h = log n => O(log n)

***Implementare:***

BType\* KSuccessor(BType\* T, BType\* x, int k)

{

return OS\_Select(T, OS\_Rank(T, x) + k);

}

BType\* OS\_Select(BType\* T, int i)

{

int r = 1;

if (T == NULL)

{

return NULL;

}

if (T->left != NULL)

{

r += T->left->dim;

}

if (i == r)

{

return T;

}

else if (i < r)

{

return OS\_Select(T->left, i);

}

else

{

return OS\_Select(T->right, i - r);

}

}

int OS\_Rank(BType\* T, BType\* x)

{

int r = 1;

if (x->left != NULL)

{

r += x->left->dim; // cate elemente sunt in stanga

}

BType\* y = x;

while (y->parent != NULL)

{

if (y == y->parent->right)

{

r += 1;

if (y->parent->left != NULL)

{

r += y->parent->left->dim;

}

}

y = y->parent;

}

return r;

}