Lab 4: Binary Search Tree - AVL Tree

1 Binary Tree - Binary Search Tree

Each Node of a Binary (Search) Tree is define as follow:

```
struct NODE{
    int key;
    NODE* left;
    NODE* right;
};
```

Students are required to implement the following functions:

1. Initialize a `NODE` from a given value:
    - `NODE* createNode(int data)`
2. Add a `NODE` with given value into a given Binary Search Tree:
    - `void Insert(NODE* &pRoot, int x)`
3. Pre-order Traversal:
    - `void NLR(NODE* pRoot)`
4. In-order Traversal:
    - `void LNR(NODE* pRoot)`
5. Post-order Traversal:
    - `void LRN(NODE* pRoot)`
6. Level-order Traversal:
    - `void LevelOrder(NODE* pRoot)`
7. Calculate the height of a given Binary Tree;
    - `int Height(NODE* pRoot)`
8. Count the number of `NODE` from a given Binary Tree:
    - `int countNode(NODE* pRoot)`
9. Calculate the total value of all `NODE`s from a given Binary Tree:
    - `int sumNode(NODE* pRoot)`
10. Find and return a `NODE` with given value from a given Binary Search Tree:
    - `NODE* Search(NODE* pRoot, int x)`
11. Remove a `NODE` with given value from a given Binary Search Tree:
    - `void Remove(NODE* &pRoot, int x)`
12. Initialize a Binary Search Tree from a given array:
    - `NODE* createTree(int a[])`
13. Completely remove a given Binary Search Tree:
    - `void removeTree(Node* &pRoot)`

14. Calculate the height of a `NODE` with given value: *(return -1 if value not exist)*
    - `heightNode(NODE* pRoot, int value)`
15. * Calculate the level of a given `NODE`:
    - `int Level(NODE* pRoot, NODE* p)`
16. * Count the number leaves from a given Binary Tree:
    - `int countLeaf(NODE* pRoot)`
17. * Count the number of `NODE` from a given Binary Search Tree which key value is less than a given value:
    - `int countLess(NODE* pRoot, int x)`
18. * Count the number of `NODE` from a given Binary Search Tree which key value is greater than a given value:
    - `int countGreater(NODE* pRoot, int x)`
19. * Determine if a given Binary Tree is Binary Search Tree:
    - `bool isBST(NODE* pRoot)`
20. * Determine if a given Binary Tree is a Full Binary Search Tree:
    - `bool isFullBST(NODE* pRoot)`

2 AVL Tree

Each Node of an AVL Tree is define as follow:

```
struct NODE{
    int key;
    NODE* left;
    NODE* right;
    int height;
};
```

Students are required to implement the following functions:

1. Initialize aNODEfrom a given value:
   - `NODE* createNode(int data)`
2. Add a `NODE` with given value into a given AVL tree (Notify if the given value existed):
   - `void Insert(NODE* &pRoot, int x)`
3. Remove a `NODE` with given value from a given AVL Tree(Notify if the given value not existed):
   - `void Remove(NODE* &pRoot, int x)`
4. Pre-order Traversal (`key` and `height` of `NODE` is required):
   - `void NLR(NODE* pRoot)`
5. In-order Traversal (`key` and `height` of `NODE` is required):
   - `void LNR(NODE* pRoot)`
6. Post-order Traversal (`key` and `height` of `NODE` is required):
   - `void LRN(NODE* pRoot)`
7. Level-order Traversal (`key` and `height` of `NODE` is required):
   - `void LevelOrder(NODE* pRoot)`
8. * Determine if a given Binary Tree is an AVL Tree:
   - `bool isAVL(NODE* pRoot)`