Weekly Homework 7

Ngày 14 tháng 5 năm 2025

1 Binary Tree

Each node in a binary tree is defined with the following structure:

```
struct NODE{
    int key;
    NODE* p_left;
    NODE* p_right;
};
```

Students are required to implement the following functions:

1. Initialize a `NODE` from a given value:
   - `NODE* createNode(int data)`
2. Pre-order traversal:
   - `vector<int> NLR(NODE* pRoot)`
3. In-order traversal:
   - `vector<int> LNR(NODE* pRoot)`
4. Post-order traversal:
   - `vector<int> LRN(NODE* pRoot)`
5. Perform a level-order traversal:
   - `vector<vector<int>> LevelOrder(NODE* pRoot)`
6. Calculate the number of `NODE`s in a binary tree:
   - `int countNode(NODE* pRoot)`
7. Calculate the sum of all `NODE` values in a binary tree:

- `int sumNode(NODE* pRoot)`

8. Calculate the height of a `NODE` with a given value: *(return -1 if not found)*
   - `heightNode(NODE* pRoot, int value)`
9. * Calculate the level of a given `NODE`:
   - `int Level(NODE* pRoot, NODE* p)`
10. * Count the number of leaf nodes in a binary tree:
    - `int countLeaf(NODE* pRoot)`

2 Binary Tree - Binary Search Tree (BST)

Each node in a binary (search) tree is defined with the following structure:

```
struct NODE{
    int key;
    NODE* p_left;
    NODE* p_right;
};
```

Students are required to implement the following functions:

1. Find and return a `NODE` with a specified value from a binary search tree
   - `NODE* Search(NODE* pRoot, int x)`
2. Add a `NODE` with a specified value to a binary search tree:
   - `void Insert(NODE* &pRoot, int x)`
3. Delete a `NODE` with a given value from a binary search tree:
   - `void Remove(NODE* &pRoot, int x)`
4. Initialize a binary search tree from a given array:
   - `NODE* createTree(int a[], int n)`
5. Delete a binary search tree entirely:
   - `void removeTree(Node* &pRoot)`

6. Calculate the height of a binary search tree:
   - `int Height(NODE* pRoot)`
7. * Count the number of `NODE`s in a binary search tree with key values smaller than a given value:
   - `int countLess(NODE* pRoot, int x)`
8. * Count the number of `NODE`s in a given binary search tree whose key values are greater than a specified value:
   - `int countGreater(NODE* pRoot, int x)`
9. * Determine if a binary tree is a binary search tree:
   - `bool isBST(NODE* pRoot)`
10. * Check if a binary tree is a full binary search tree (BST):
    - `bool isFullBST(NODE* pRoot)`

3 AVL Tree

Each node of an AVL tree is defined as follows:

```
struct NODE{
    int key;
    NODE* p_left;
    NODE* p_right;
    int height;
};
```

Students are required to implement the following functions:

1. Initialize a NODE from a given value:
   - `NODE* createNode(int data)`
2. Insert a `NODE` with a given value into a given AVL tree (Note that the given value may already exist):
   - `void Insert(NODE* &pRoot, int x)`
3. Delete a `NODE` with a given value from a given AVL tree (Note that the value may not exist):
   - `void Remove(NODE* &pRoot, int x)`
4. * Determine if a binary tree is an AVL tree:
   - `bool isAVL(NODE* pRoot)`

4 Submission Rules

Students must adhere to the following submission guidelines:

1. The submission must be in a **compressed zip file** named **MSSV.zip**, containing:
   - The required C++ files: binary_tree.cpp, bst.cpp, avl.cpp (*Each file does not contain a `main` function*).
   - A `report.pdf` file describing the approach used in each solution. The image of GitHub home page to verify code is pushed on GitHub
   - Don't use $< bits/stdc++.h >$ library