Lab 5 –Tree

Problem 1

Implement the blank methods (//TODO).

```
#include <iostream>
#include <string>
using namespace std;
class TreeNode {
private:
       string character;
       int count;
       TreeNode* left = NULL;
       TreeNode* right = NULL;
public:
       TreeNode(string character); // TODO
       TreeNode(char character);   // TODO
       ~TreeNode();   // TODO
       void increaseCount();   // TODO: increase "count" by 1
       // get/set methods
       int getCount();   // TODO
       void setCount(int newCount);      // TODO
       string getChar();   // TODO
       void setChar(string newChar);     // TODO
       TreeNode* getLeft();   // TODO
       void setLeft(TreeNode* newLeft);   // TODO
       TreeNode* getRight();   // TODO
       void setRight(TreeNode* newRight); // TODO
};
```

Problem 2

Implement the blank methods (//TODO).

```
class BinarySearchTree {
public:
       TreeNode* root = NULL;
       void insert(TreeNode* node);   // TODO: insert a node, if the "character" of
"node" does not exist, then increase the "count"
       void remove(string character); // TODO: remove/delete a node having the same
"character"
       int search(string character);  // TODO: return "count"
       void print();   // TODO: print out the whole tree on the console
};
```

Problem 3

We can use the above Binary Search Tree to calculate how many times a character has appeared in a sentence. Write a function to build the BST from a given string.

```
BinarySearchTree* buildTreeFromString(string str)
```

For example:

```
int main() {
       string str = "A binary search tree is a binary tree with the following properties:
All items in the left subtree are less than the root.All items in the right subtree are
greater than or equal to the root.Each subtree is itself a binarysearch tree.";
       BinarySearchTree* bst = buildTreeFromString(str);
       bst->print();
       cout << endl;
       cout << "b = " << std::to_string(bst->search("b")) << endl; // 6   times
       cout << "s = " << std::to_string(bst->search("s")) << endl; // 13   times
       cout << "t = " << std::to_string(bst->search("t")) << endl; // 24 times
       system("pause");
       return 1;
}
```

Problem 4*

Given a tree built from the following tree node:

```
class IntNode {
private:
       int data;
       int count;
       TreeNode* left = NULL;
       TreeNode* right = NULL;
public:
       // You can implement support functions here to ease your work (Similar to class
TreeNode). Optional.
};
```

Write a function to check if a tree is a BST or not.

Problem 5*

Write a function to calculate the height of a tree. (Tree with only one root has a height of 0).