```cpp
#include <iostream>
#include <set>
using namespace std;

struct TreeNode
{
    int value;
    TreeNode* left;
    TreeNode* right;
    TreeNode() {
        left = nullptr;
        right = nullptr;
    }
    TreeNode(int val) {
        value = val;
        left = nullptr;
        right = nullptr;
    }
};
/*
// int countFavoriteNumbers(TreeNode* tree, const set<int> & favorites) {
//     if(tree == nullptr || favorites.empty())
//         return 0;
//     else
//         return favorites.count(tree->value)
//             + countFavoriteNumbers(tree->left, favorites)
//             + countFavoriteNumbers(tree->right, favorites);
//
// }
//
// bool hasPathSum(TreeNode* tree, int sum) {
//     if(tree == nullptr)
//         return sum == 0;
//     else {
//         return hasPathSum(tree->left, sum-tree->value)
//             || hasPathSum(tree->right, sum-tree->value);
//     }
// }
//
// int getLevelSumDifference(TreeNode* tree) {
//     if(tree == nullptr)
//         return 0;
//     else
//         return tree->value
//             - getLevelSumDifference(tree->left)
//             - getLevelSumDifference(tree->right);
//
// }

// bool contains(TreeNode* tree, int value) {
```

```
51 //     if(tree==nullptr)
52 //         return false;
53 //     else if(tree->value == value)
54 //         return true;
55 //     else
56 //         return contains(tree->left, value) || contains(tree->right,
   value);
57 // }
58
59 // int getLevel(TreeNode* tree, int value) {
60 //     if(!contains(tree, value))
61 //         return -1;
62 //     else if(tree->value == value)
63 //         return 0;
64 //     else if(tree->value > value)
65 //         return getLevel(tree->left, value) + 1;
66 //     else if(tree->value < value)
67 //         return getLevel(tree->right, value) + 1;
68 //
69 // }
70 //
71 // TreeNode* getParent(TreeNode* tree, int value) {
72 //     if(tree == nullptr || tree->value == value)
73 //         return nullptr;
74 //     else if(tree->left != nullptr && tree->left->value == value)
75 //         return tree;
76 //     else if(tree->right != nullptr && tree->right->value == value)
77 //         return tree;
78 //     else if(tree->value > value)
79 //         getParent(tree->left, value);
80 //     else
81 //         getParent(tree->right, value);
82 // }
83 //
84 // TreeNode* getNearestSharedRoot(TreeNode* tree, int a, int b) {
85 //     if(tree == nullptr)
86 //         return nullptr;
87 //     else if(a <= tree->value && b >= tree->value)
88 //         return tree;
89 //     else if(b <= tree->value)
90 //         return getNearestSharedRoot(tree->left, a, b);
91 //     else
92 //         return getNearestSharedRoot(tree->right, a, b);
93 // }
94
95 // bool isAdditionTree(TreeNode* tree) {
96 //     if(tree==nullptr || (tree->left == nullptr && tree->right ==
   nullptr))
97 //         return true;
98 //     else if(tree->left == nullptr)
```

```
99 //         return tree->right->value == tree->value
100 //             && isAdditionTree(tree->right);
101 //     else if(tree->right == nullptr)
102 //         return tree->left->value == tree->value
103 //         && isAdditionTree(tree->left);
104 //     return tree->left->value+tree->right->value == tree->value
105 //         && isAdditionTree(tree->left)
106 //         && isAdditionTree(tree->right);
107 // }
108 //
109 // int height(TreeNode* tree) {
110 //     if(tree == nullptr)
111 //         return 0;
112 //     else
113 //         return 1 + max(height(tree->left), height(tree->right));
114 // }
115 //
116 // int leafSpan(TreeNode* tree) {
117 //     if(tree == nullptr)
118 //         return 0;
119 //     if(tree->left == nullptr && tree->right == nullptr)
120 //         return 1;
121 //     int span = height(tree->left) + height(tree->right) + 1;
122 //
123 //     return max(span, max(leafSpan(tree->left), leafSpan(tree->right
    ));
124 // }
125 //
126 // bool containsPairSumHelper(TreeNode* one, TreeNode* two, int sum) {
127 //     if(one == nullptr || two == nullptr)
128 //         return false;
129 //     else if(contains(two, sum-one->value))
130 //         return true;
131 //     else
132 //         return containsPairSumHelper(one->left, two, sum)
133 //             || containsPairSumHelper(one->right, two, sum);
134 // }
135 //
136 // bool hasPairSum(TreeNode *tree, int sum) {
137 //     return containsPairSumHelper(tree, tree, sum);
138 //
139 // }
140 */
141 TreeNode* construct() {
142     TreeNode* tree = new TreeNode;
143     tree->value = 14;
144     tree->left = new TreeNode;
145     tree->left->value = 10;
146     tree->left->left = new TreeNode;
147     tree->left->left->value = 2;
```

```
148     tree->left->right = new TreeNode;
149     tree->left->right->value = 12;
150     tree->left->left->left = new TreeNode;
151     tree->left->left->left->value = 1;
152     tree->right = new TreeNode;
153     tree->right->value = 18;
154     tree->right->left = new TreeNode;
155     tree->right->left->value = 15;
156     tree->right->right = new TreeNode;
157     tree->right->right->value = 20;
158     return tree;
159 }
160
161 TreeNode* getDuplicateCopy(TreeNode* tree) {
162     if(tree == nullptr)
163         return nullptr;
164
165     TreeNode* newTree = new TreeNode(tree->value);
166     newTree->left = getDuplicateCopy(tree->left);
167     newTree->right = getDuplicateCopy(tree->right);
168
169     return newTree;
170 }
171
172 int getMinValue(TreeNode* tree) {
173     if(tree == nullptr)
174         return INT_MAX;
175     return min(tree->value, min(getMinValue(tree->left),getMinValue(tree
    ->right)));
176 }
177 int getMaxValue(TreeNode* tree) {
178     if(tree == nullptr)
179         return INT_MIN;
180     return max(tree->value, max(getMaxValue(tree->left),getMaxValue(tree
    ->right)));
181 }
182
183 bool isBST(TreeNode* tree) {
184     if(tree == nullptr)
185         return true;
186
187     int leftMax = getMaxValue(tree->left);
188     int rightMin = getMinValue(tree->right);
189
190     //if req a && req b && req c1 && req c2, return true, else return
    false
191     return (leftMax < tree->value) && (rightMin > tree->value) && isBST(
    tree->left) && isBST(tree->right);
192 }
193
```

```
194 int getShortestPathLength(TreeNode* tree) {
195     if(tree == nullptr)
196         return 0;
197     if(tree->left == nullptr && tree->right != nullptr)
198         return 1 + getShortestPathLength(tree->right);
199     else if(tree->left != nullptr && tree->right == nullptr)
200         return 1 + getShortestPathLength(tree->left);
201     else
202         return 1 + min(getShortestPathLength(tree->left),
    getShortestPathLength(tree->right));
203 }
204
205 int main()
206 {
207     cout << "Unit 3 Test!" << endl;
208     TreeNode* tree = construct();
209
210     cout << "Problem 2024.0" << endl;
211     TreeNode* dup = getDuplicateCopy(tree);
212
213     cout << "Problem 2024.1" << endl;
214     if(isBST(tree))
215         cout << "It is a BST" << endl;
216     else
217         cout << "It is not a BST" << endl;
218
219     cout << "Problem 2024.2" << endl;
220     cout << getShortestPathLength(tree) << endl;
221
222     /* cout << "Problem 2022.0" << endl;
223     //
224     // set<int> s;
225     // int num = countFavoriteNumbers(tree, s);
226     //
227     // cout << "Problem 2022.1" << endl;
228     // hasPathSum(tree, 3);
229     //
230     // cout << "Problem 2022.2" << endl;
231     // getLevelSumDifference(tree);
232
233     // cout << "Problem 2021.0" << endl;
234     // cout << "Level: " << getLevel(tree, 6) << endl;
235     //
236     // cout << "Problem 2021.1" << endl;
237     // cout << "Parent Num: " << getParent(tree, 4)->value << endl;
238     //
239     // cout << "Problem 2021.2" << endl;
240     // getNearestSharedRoot(tree, 5);
241
242     // cout << "Problem 2023.0" << endl;
```

```
243     // if(isAdditionTree(tree))
244     //     cout << "It is an addition tree" << endl;
245     // else
246     //     cout << "It is not an addition tree" << endl;
247     //
248     // cout << "Problem 2023.1" << endl;
249     // cout << leafSpan(tree) << endl;
250     //
251     // cout << "Problem 2023.2" << endl;
252     // if(hasPairSum(tree, 5))
253     //     cout << "Has Pair Sum" << endl;
254 */
255
256     return 0;
257 }
258
```