

Swinburne University of Technology

Faculty of Science, Engineering and Technology

ASSIGNMENT COVER SHEET

Subject Code:

COS30008

Subject Title:

Data Structures and Patterns

Assignment number and title:

4, Binary Search Trees & In-Order Traversal

Due date:

November 18<sup>th</sup>, 2024

Lecturer:

Dr. Ky Trung Pham

StudentName:

Lau Ngoc Quyen

StudentID:

104198996

Check Tutorial	Mon 10:30	Mon 14:30	Tues 08:30	Tues 10:30	Tues 12:30	Tues 14:30	Tues 16:30	Wed 08:30	Wed 10:30	Wed 12:30	Wed 14:30
	X										

Marker's comments:

Problem	Marks	Obtained
1	94	
2	42	
3	8+86=94	
Total	230	

Extension certification:

This assignment has been given an extension and is now due on November 18<sup>th</sup> 2024

Signature of Convener: \_\_\_\_\_

## BinaryTreeNode.h

```
1
2 #include <stdexcept>
3 #include <algorithm>
4 using namespace std;
5 template<typename I>
6 struct BinaryTreeNode
7 {
8     using BNode = BinaryTreeNode<I>;
9     using BTreeNode = BNode*;
10
11     I key;
12     BTreeNode left;
13     BTreeNode right;
14
15     static BNode NIL;
16     const I& findMax() const
17     {
18         if (empty())
19         {
20             throw domain_error("Empty tree encountered");
21         }
22         if (right->empty())
23         {
24             return key;
25         }
26         return right->findMax();
27     }
28     const I& findMin() const
29     {
30         if (empty())
31         {
32             throw domain_error("Empty tree encountered");
33         }
34         if (left->empty())
35         {
36             return key;
37         }
38         return left->findMin();
39     }
40     bool remove(const I& aKey, BTreeNode aParent)
41     {
42         BTreeNode x = this;
43         BTreeNode y = aParent;
44         while (!x->empty())
45         {
46             if (aKey == x->key)
47             {
48                 break;
49             }
50             y = x;
51             x = aKey < x->key ? x->left : x->right;
52         }
53         if (x->empty())
54         {
55             return false;
56         }
57         if (!x->left->empty())
58         {
59             const I& lKey = x->left->findMax();
60             x->key = lKey;
61             x->left->remove(lKey, x);
62         }
63         else
64         {
65             if (!x->right->empty())
66             {
67                 const I& lKey = x->right->findMin();
68                 x->key = lKey;
69                 x->right->remove(lKey, x);
70             }
71             else
72             {
73                 if (y != &NIL)
74                 {
75                     if (y->left == x)
76                     {
77                         y->left = &NIL;
78                     }
79                     else
80                     {
81                         y->right = &NIL;
82                     }
83                 }
84                 delete x;
85             }
86         }
87     }
88     return true;
89 }
90
```

```

1
2 BinaryTreeNode() : key(I()), left(&NIL), right(&NIL) {}
3 BinaryTreeNode(const I& aKey) : key(aKey), left(&NIL), right(&NIL) {}
4 BinaryTreeNode(I&& aKey) : key(move(aKey)), left(&NIL), right(&NIL) {}
5 ~BinaryTreeNode()
6 {
7     if (!left->empty())
8     {
9         delete left;
10    }
11    if (!right->empty())
12    {
13        delete right;
14    }
15 }
16
17 bool empty() const
18 {
19     return this == &NIL;
20 }
21 bool leaf() const
22 {
23     return left->empty() && right->empty();
24 }
25 size_t height() const
26 {
27     if (empty())
28     {
29         throw domain_error("Empty tree encountered");
30     }
31     if (leaf())
32     {
33         return 0;
34     }
35     const int left_height = left->empty() ? 1 : left->height() + 1;
36     const int right_height = right->empty() ? 1 : right->height() + 1;
37     return max(left_height, right_height);
38 }
39 bool insert(const I& aKey)
40 {
41     if (empty())
42     {
43         return false;
44     }
45     if (aKey > key)
46     {
47         if (right->empty())
48         {
49             right = new BNode(aKey);
50         }
51         else
52         {
53             return right->insert(aKey);
54         }
55         return true;
56     }
57     if (aKey < key)
58     {
59         if (left->empty())
60         {
61             left = new BNode(aKey);
62         }
63         else
64         {
65             return left->insert(aKey);
66         }
67         return true;
68     }
69     return false;
70 }
71 };
72 template<typename I>
73 BinaryTreeNode<I> BinaryTreeNode<I>::NIL;

```

## BinarySearchTree.h

```
1  #include "BinaryTreeNode.h"
2  #include <stdexcept>
3  // Problem 3 requirement
4  template<typename I>
5  class BinarySearchTreeIterator;
6  template<typename I>
7  class BinarySearchTree
8  {
9  private:
10     using BNode = BinaryTreeNode<I>;
11     using BTreeNode = BNode*;
12     BTreeNode fRoot;
13
14 public:
15     BinarySearchTree() : fRoot(&BNode::NIL) {}
16     ~BinarySearchTree()
17     {
18         if (!fRoot->empty())
19         {
20             delete fRoot;
21         }
22     }
23     bool empty() const
24     {
25         return fRoot->empty();
26     }
27     size_t height() const
28     {
29         if (empty())
30         {
31             throw domain_error("Empty tree has no height.");
32         }
33         return fRoot->height();
34     }
35
36     bool insert(const I& aKey)
37     {
38         if (empty())
39         {
40             fRoot = new BNode(aKey);
41             return true;
42         }
43         return fRoot->insert(aKey);
44     }
45     bool remove(const I& aKey)
46     {
47         if (empty())
48         {
49             throw domain_error("Cannot remove in empty tree.");
50         }
51         if (fRoot->leaf())
52         {
53             if (fRoot->key != aKey)
54             {
55                 return false;
56             }
57             fRoot = &BNode::NIL;
58             return true;
59         }
60         return fRoot->remove(aKey, &BNode::NIL);
61     }
62     using Iterator = BinarySearchTreeIterator<I>;
63     friend class BinarySearchTreeIterator<I>;
64     Iterator begin() const
65     {
66         return Iterator(*this).begin();
67     }
68     Iterator end() const
69     {
70         return Iterator(*this).end();
71     }
72 };
```

## BinarySearchTreeIterator.h

```
1 #include "BinarySearchTree.h"
2 #include <stack>
3 template<typename I>
4 class BinarySearchTreeIterator
5 {
6 private:
7
8     using BSTree = BinarySearchTree<I>;
9     using BNode = BinaryTreeNode<I>;
10    using BTreeNode = BNode*;
11    using BTNStack = std::stack<BTreeNode>;
12    const BSTree& fBSTree; // binary search tree
13    BTNStack fStack; // DFS traversal stack
14
15    void pushLeft(BTreeNode aNode)
16    {
17        if (!aNode->empty())
18        {
19            fStack.push(aNode);
20            pushLeft(aNode->left);
21        }
22    }
23
24 public:
25
26     using Iterator = BinarySearchTreeIterator<I>;
27
28     BinarySearchTreeIterator(const BSTree& aBSTree) : fBSTree(aBSTree), fStack()
29     {
30         pushLeft(aBSTree.fRoot);
31     }
32     const I& operator*() const
33     {
34         return fStack.top()->key;
35     }
36     Iterator& operator++()
37     {
38         BTreeNode lPopped = fStack.top();
39         fStack.pop();
40         pushLeft(lPopped->right);
41         return *this;
42     }
43     Iterator operator++(int)
44     {
45         Iterator temp = *this;
46         ++(*this);
47         return temp;
48     }
49     bool operator==(const Iterator& aOtherIter) const
50     {
51         return &fBSTree == &aOtherIter.fBSTree && fStack == aOtherIter.fStack;
52     }
53     bool operator!=(const Iterator& aOtherIter) const
54     {
55         return !(*this == aOtherIter);
56     }
57
58     Iterator begin() const
59     {
60         Iterator temp = *this;
61         temp.fStack = BTNStack();
62         temp.pushLeft(temp.fBSTree.fRoot);
63         return temp;
64     }
65     Iterator end() const
66     {
67         Iterator temp = *this;
68         temp.fStack = BTNStack();
69         return temp;
70     }
71 };
```