CPSC 2150 – Algorithms and Data Structure II

Assignment 3: Binary Trees

Total - 100 Marks

"Good design adds value faster than it adds cost."
- Thomas C. Gale

Learning Outcomes

- Design, implement and analyze the tree-based algorithms in terms of time and space complexity.
- Design and implement the operations of binary search tree and expression trees.
- Write recursive solutions to non-trivial problems, such as binary search tree traversals.
- Develop C++ code based on the existing constraints.

Resources

- Chapter 7 of the text book

Description

This is a practice on design, implement and analyzing problems using binary search trees. The main function of your test program (testBST.cpp) should be compatible with the following main function. By compatible it means the number of parameters of each function should match. The type of parameters is determined by your design and implementation. This makes some constraints in your design, which is likely to occur in the real world.

```
int main() {
  // declaration of your variables ...
  n1 = getInput(); // either generates a random non-negative
                   // integer or reads it from input
  list1 = genData(n1); //generates a list of n1 random numbers [-n1, n1]
  cout << "The List1: ";
  printList(list1); //prints elements of the given list

  n2 = getInput();
  list2 = genData(n2); //generates a list of n2 random numbers [-n2, n2]
  printList(list2);

  bst1 = makeBST(list1);
  cout << "In-order traversal of bst1 is: ";
  printBT(bst1);
```

```
  remove(list1[n1/2], bst1); // removes list1[n1/2] from corresponding tree (bst1)

  cout << "In-order traversal of bst1 after deleting " <<list1[n1/2] <<" is: ";
  printBT(bst1);

  bst2 = makeBST(list2);
  cout << "In-order traversal of bst2 is: ";
  printBT(bst2);

  bst3 = mergeBST(bst1, bst2);
  cout << "In_order traversal of bst3 is: ";
  printBT(bst3);

  cout << "The height of bst1 is " << height(bst1) << endl;
  cout << "The height of bst2 is " << height(bst2) << endl;
  cout << "The height of merged tree is " << height(bst3) << endl;

  string infix = getExpression(); // read infix expression from input

  bt4 = infixExprTree(infix);
  cout << "In-order traversal of bt4 is: ";
  printBT(bt4);

  cout << "The postfix expression is " << InfixPostfixExpr(infix) << endl;

  return 0;
}
```

Exercise 1: BST class (BST.h)

[25 marks] Provide an appropriate data structure and necessary methods to build a binary search tree which enables you to complete the rest of this assignment. Define your class with the generic types.

Exercise 2: Binary Search Tree (testBST.cpp)

a. **[5 marks]** Write a function named **genData** that given an integer **n**, generates a list of **n** random integers in the interval **[-n, n]** (square bracket means inclusive).

b. **[5 marks]** Write a function named **makeBST** that given a list of data, generates a binary search tree.

c. **[5 marks]** Write a function named **printBT()** that given a binary tree, prints the tree's elements using an in-order traversal.

d. **[5 marks]** Write a function named **height()** that finds the height of the binary tree which has been passed as its parameter.

e. **[5 marks]** Write a function named **remove()** to delete a given key from a given BST. It removes at most one node.

f. **[17 marks]** Write an efficient function (time and space efficient) named **mergeBST()** which, given two binary search trees, merges them into one binary search tree. The original BSTs must remain unchanged.

[3 bonus marks] if your algorithm avoids ending up with an unbalanced BST.

Exercise 3: Expression Trees

Only binary operations +, -, * and / with priority and parentheses are allowed in an infix expression. Also, to keep it simple, use 1-digit numbers in your expressions. Implement your expression trees as explained in the lecture.

a. **[7 marks]** Write a function named **infixExprTree()** that given an infix expression, generates its corresponding expression tree.

b. **[7 marks]** Write a function named **InfixPostfixExpr()** that receives an infix expression and transforms it into a postfix expression.

Hint- Make the expression tree of the infix expression and then apply the post-order traversal on the tree.

Exercise 4: Time and Space Complexity (**answers.pdf**)

a. **[16 marks]** Calculate the time complexity of your function in two previous exercises; i.e. genData(), makeBST(), printBT(), height(), remove(), mergeBST(), InfixPostfixExpr() and infixExprTree().

b. **[3 marks]** Calculate the space complexity of **mergeBST**.

SUBMIT to D2L

Submit a zip file named **StudentNumber-Asgn3.zip** including all related files such as **answers.pdf** and **testBST.cpp, BST.h** and **make file** by the due date. For example, if your student number is 10023449, the submitted file must be named as **10023449-Asgn3.zip**.