Lab 7: AVL Tree

class AVLTree

The following functions are the minimum requirements for the AVL class. You can add any function from Assignment 2 to this class. You should modify the BSTree insert function so that the tree remains balanced after each insertion.

Required Public Member Functions

- `void insert(const string &)`: Insert an item to the binary search tree and perform rotation if necessary.
- `int balanceFactor(Node*)`: Return the balance factor of a given node.
- `void printBalanceFactors()`: Traverse and print the tree in inorder notation. Print the string followed by its balance factor in parentheses followed by a , and one space.
- `void visualizeTree(const string &)`: Generate dotty file and visualize the tree using dotty program. Call this function before and after rotation.

Recommended Private Helper Functions

These helper functions are recommended, but you can change them or add other helper functions if necessary.

- `findUnbalancedNode`: Find and return the closest unbalanced node (with balance factor of 2 or -2) to the inserted node.
- `rotate`: Implement four possible imbalance cases and update the parent of the given node.
- `rotateLeft`: Rotate the subtree to left at the given node and returns the new subroot.
- `rotateRight`: Rotate the subtree to right at the given node and returns the new subroot.
- `void printBalanceFactors(Node *)`
- `void visualizeTree(ofstream &, Node *)`

Implementation of visualizeTree function

```
void BSTree::visualizeTree(const string &outputFilename){
    ofstream outFS(outputFilename.c_str());
    if(!outFS.is_open()){
        cout<<"Error"<<endl;
        return;
    }
    outFS<<"digraph G {"<<endl;
    visualizeTree(outFS,root);
    outFS<<"}";
    outFS.close();
    string jpgFilename = outputFilename.substr(0,outputFilena
me.size()-4)+".jpg";
    string command = "dot -Tjpg " + outputFilename + " -o " +
jpgFilename;
    system(command.c_str());
}

void BSTree::visualizeTree(ofstream & outFS, Node *n){
    if(n){
        if(n->left){
            visualizeTree(outFS,n->left);
            outFS<<n->data <<" -> " <<n->left->data<<";"<<end
l;
        }

        if(n->right){
            visualizeTree(outFS,n->right);
            outFS<<n->data <<" -> " <<n->right->data<<";"<<en
dl;
        }
    }
}
```

Use the following main to test your program.

```
#include <iostream>
#include "AVLTree.h"

using namespace std;

int menu() {
  int choice = 0;
  cout << endl << "Enter menu choice: ";
  cout << endl;
  cout
    << "1. Insert" << endl
    << "2. Print" << endl
    << "3. Quit" << endl;
  cin >> choice;

  // fix buffer just in case non-numeric choice entered
  // also gets rid of newline character
  cin.clear();
  cin.ignore(256, '\n');
  return choice;
}

int main( ) {

  AVLTree tree;

  int choice = menu();

  string entry;

  while (choice != 3) {

    if (choice == 1) {
      cout << "Enter string to insert: ";
      getline(cin, entry);
```

```
      cout << endl;

      tree.insert(entry);

    } else if (choice == 2) {
      tree.printBalanceFactors();

    }
    //fix buffer just in case non-numeric choice entered
    choice = menu();
  }

  return 0;
}
```