COEN 79

Object-Oriented Programming and Advanced Data Structures

Assignment #6 – Practice for Tree Data Structure,
Submission date (if you choose to) Thursday June 5th

Each question is 3 pts.

---

1. Please complete the following implementation:

```
template < class Item >
binary_tree_node <Item>* tree_copy (const binary_tree_node <Item>* root_ptr)
```

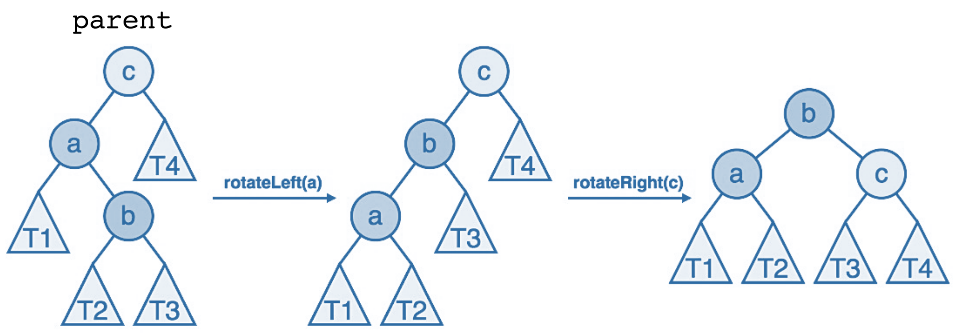
- Using the previous implementation, complete the following function for the bag class given in Appendix 1:

```
template < class Item >
void bag <Item>::operator = (const bag<Item>& source)
// Header file used: bintree.h
```

2. For the bag class defined in Appendix 1, please complete the following function:

```
template < class Item >
void bag<Item>::insert(const Item& entry)
// Postcondition: A new copy of entry has been inserted into the bag.
// Header file used: bintree.h
```

3. Write a function to perform *left-right* rotation on the following AVL tree. The figure shows the steps. (Note: Please implement the function in two steps: (1) left rotation, (2) right rotation.)



```
template < class Item >
binary_tree_node <Item>* left_right_rotation (binary_tree_node <Item>*& parent)
{
    binary_tree_node <Item>* temp;
```

4. Add the following numbers to an AVL tree. Please draw the final tree.
`2, 4, 6, 8, 10, 12, 20, 18, 16, 14`

5. The following functions are available:

```
template < class Item >
int height (const binary_tree_node <Item>* temp)
{
    int h = 0;
    if (temp != NULL) {
        int l_height = height(temp -> left());
        int r_height = height(temp -> right());
        int max_height = std::max (l_height, r_height);
        h = max_height + 1;
    }
    return h;
}
```

```
template < class Item >
int diff (const binary_tree_node <Item>* temp)
{
    int l_height = height(temp -> left());
    int r_height = height(temp -> right());
    int b_factor = l_height - r_height;

    return b_factor;
}
```

Also assume the following functions are available:

- `binary_tree_node<Item>* left_rotation (binary_tree_node<Item>*& parent)`
- `binary_tree_node<Item>* right_rotation (binary_tree_node<Item>*& parent)`
- `binary_tree_node<Item>* left_right_rotation (binary_tree_node<Item>*& parent)`
- `binary_tree_node<Item>* right_left_rotation (binary_tree_node<Item>*& parent)`

Complete the following function, which balances a tree rooted at `temp`.

```
template < class Item >
binary_tree_node<Item>* balance(binary_tree_node <Item>*& temp)
```

6. Please implement the following function (*recursively*).

```
template < class Item >
void flip(binary_tree_node < Item > * root_ptr)
// Precondition: root_ptr is the root pointer of a non-empty binary tree.
// Postcondition: The tree is now the mirror image of its original value.
```

Example:

```
//           1                                 1
//          / \                               / \
//         2   3                             3   2
//        / \                                   / \
//       4   5                                 5   4
```

```
template < class Item >
void flip (binary_tree_node <Item>* root_ptr)
```

Appendix 1: Bag class with binary search tree.

```cpp
template < class Item >
class bag {

public:
    // TYPEDEFS
    typedef std::size_t size_type;
    typedef Item value_type;

    // CONSTRUCTORS and DESTRUCTOR
    bag() {  root_ptr = NULL;  }
    bag(const bag& source);
    ~bag();

    // MODIFICATION functions
    size_type erase(const Item& target);
    bool erase_one(const Item& target);
    void insert(const Item& entry);
    void operator += (const bag& addend);
    void operator = (const bag& source);

    // CONSTANT functions
    size_type size() const;
    size_type count(const Item& target) const;
    void debug() const {  print(root_ptr, 0);  }

private:
    binary_tree_node<Item>* root_ptr; // Root pointer of binary search tree
    void insert_all (binary_tree_node<Item>* addroot_ptr);
};
```