CS 469: Special Topics in Computer Science

Assignment 4

Exercise 1: Binary trees

Let consider the following template of binary trees

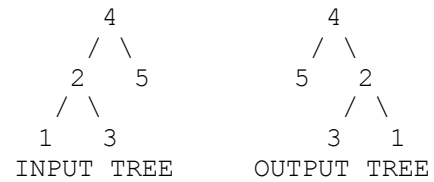
```
template <typename T>
class BinaryTree{
private:
    // Declare a structure for the list
    struct TreeNode
    {
        T value;
        struct TreeNode *left;
        struct TreeNode *right;
    };
    TreeNode *root;        // tree root pointer
    // A function to create a node
    TreeNode * createNode(T key){
        TreeNode *node = new TreeNode;
        node->value = key;
        node->left = NULL;
        node->right = NULL;
        return node;
    }
public:
    BinaryTree (void)      // Constructor
        { root = NULL; }
```

```
    BinaryTree (T);
    ~ BinaryTree (void); // Destructor
    T& top();
    T& pop_front();
    bool empty();
    void insertNode(T);
    void deleteNode(T, bool);
    void preOrderTraversal();
    void inOrderTraversal();
    void postOrderTraversal();
    int countLesserThan(T);
    int countGreaterThan(T);
    int length();
    int height();
    void clear();
    void mirror();
    bool isIdenticalTo(BinaryTree);
    bool isIsomorphicWith(BinaryTree);
    int countLeafNodes();
    int countSemiLeafNodes();
};
```

Suppose that this class permits the manipulation of binary search trees. Write an algorithm (**Pseudo-code please, no sentences!!**) and a C++ program for each of the following methods for this BinaryTree class:

1. **BinaryTree(T info)**: the constructor to create a binary tree which first node contains the value info and the next pointer is null.
   Hint: use the function TreeNode * createNode(T key);
2. **~ BinaryTree ()**: a destructor that remove all elements of binary tree.

3. **T& top()**: to get the first element of the binary tree.
4. **T& pop_front()**: to remove the first node of the binary tree and to return its value.
5. **bool empty()**: to test if the binary tree is empty or no.
6. **void insertNode(T info)**: to insert a node in the binary tree.
7. **void deleteNode(T info, bool removeAll)**: to delete a node the binary tree, if removeAll is true, all occurrence of info will be removed, otherwise, only the first occurrence of info will be removed.
8. **void preOrderTraversal()**: to display the binary tree using pre-order traversal.
9. **void inOrderTraversal()**: to display the binary tree using in-order traversal.
10. **void postOrderTraversal()**: to display the binary tree using post-order traversal.
11. **int countLesserThan(T val)**: to count the the nodes which **value** is lesser than **val** in the binary tree.
12. **int countGreaterThan(T val)**: to count the the nodes which **value** is greater than **val** in the binary tree.
13. **int length()**: to count the number of nodes in the binary tree.
14. **int height()**: to compute the height of the the binary tree.
15. **void clear()**: to remove all elements of the binary tree.
16. **void mirror()**:swaps the left and right pointers of the tree. Below trees provide an example input and output for mirror().



17. **bool isIdenticalTo(BinaryTree B):** determines if the current binary tree is identical to the binary tree B.
18. **bool isIsomorphicWith (BinaryTree *B*):** determines if the current binary tree is isomorphic with the binary tree B. A binary tree ***A*** is said to be isomorphic to ***B***, if its mirror is identical to the binary tree **B.**
19. **bool countLeafNodes ():** Counts the number of leaf nodes the binary tree**.**
20. **bool countSemiLeafNodes ():** Counts the number of semi-leaf nodes in the binary tree.

Hint: some method can reuse other methods, like for example you can reuse the method **clear()** in the destructor.

Exercise 2: Binary min-heap

We saw in class that a binary min-heap can be represented by an array as illustrated in the figure 1.

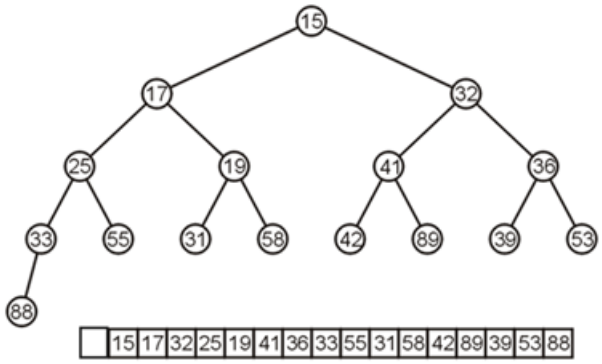


Figure 1

For a given entry k, it follows that the parent node is at $k/2$, the left child is at the index 2k and the right child it at the index $2k + 1$ as shown in the figure 2.

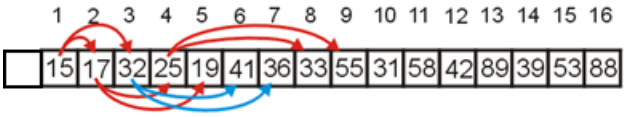


Figure 2

The aim is to implement a class in C++ that manipulate the binary min-heap. For that purpose, let the following template of the binary min-heap:

```
template <typename T>
class BinaryMinHeap{
private:
    T *array;       // An array to store the binary min-heap
    int capacity; // the capacity of this array
    int size;  // the current size of this array
public:
```

```
    BinaryMinHeap  (void)          // Constructor
            { array = NULL;
              capacity = 0;
              size = 0;
            }
    BinaryMinHeap  (int Capacity);
    ~ BinaryMinHeap  (void); // Destructor
    void precolate(int nodeI);
    T getMin();
    T extractMin();
    void deleteNode(int nodeI);
    void decreaseKey(int node, int new_val);
    void insertKey(T val);
    T getLeftChild(int  nodeI);
    T getRightChild(int  nodeI);
    bool empty():
};
    // A utility function  to swap two elements
    void swap(int *x, int *y) {
       int temp = *x;
       *x = *y;
       *y = temp;
    }
```

Write an algorithm (**Pseudo-code please, no sentences!!**) and a C++ program for each of the following methods for this BinaryMinHeap class:

1. **BinaryMinHeap (int Capacity)**: the constructor to create a binary min-heap with a capacity.

2. **~ BinaryMinHeap ()**: a destructor that remove all elements of the binary min-heap.
3. **void percolate(int nodeI)**: to heapify a subtree with the root at given index.
4. **T getMin()**: gets the minimum value of the binary min-heap.
5. **bool empty()**: tests if the binary min-heap is empty or no.
6. **T extractMin()**: removes minimum element (or root) from the binary min-heap.
7. **void deleteNode(int nodeI)**: deletes key at index i. (**Hint**: reduce the value associated to the key nodeI to minus infinite by using the function decreaseKey, then calls extractMin())
8. **void insertKey(T val)**: inserts a new node which key is **val**.
9. **T getLeftChild(int nodeI)**: gets the left child of nodeI.
10. **T getRightChild(int nodeI)**: gets the right child of nodeI.