CS1020E: Data Structures and Algorithms I

Tutorial 5 – Lists, list<T>::iterator
3 March 2017

1. Singly Linked List — With More Methods

In our previous tutorial, we implemented a linked list with some basic methods. You are now tasked to add more methods to this class.

(a) Union with another list

Implement the **unionWithList** method that performs a union of the current list and another list, resulting in a final single list that replaces the original current list. If both lists contain a **Node** with the same **number** value, the final list will only contain either **Node** but not both. For example, the union of **{1, 2, 4}** and **{1, 2, 3, 5}** results in **{1, 2, 3, 4, 5}**.

The other list should be empty after the union operation. You may assume that both lists are already sorted in ascending order, and that there are no duplicates within each list.

Restriction: You cannot create any new **Node**, and the number value in each **Node** must not be changed.

(b) Reverse list

Implement the **reverse** method that reverses the current list, and the last **Node** in the list becomes the new head.

Restriction: You cannot create any new **Node** or use another **LinkedList** object, and the number value in each **Node** must not be changed.

The specification for the methods described above can be found below. The methods that have already been implemented in the previous tutorial are omitted here in the interest of space, but you should use the same source code from Tutorial 4 Question 4 to continue working on this question and to test your code.

```
class LinkedList {
private:
    struct Node {
        int number;
        Node* next;
    };
    Node* head;
public:
    void unionWithList( LinkedList& other );
    void reverse ();
};
```

2. Doubly Linked List

Implement the **reverse** method that reverses a **circular doubly linked list**. An incomplete skeleton code of the class is provided here. You should implement the other methods in the class yourself in order to test your answer.

Restriction: You cannot create any new **Node** or use another **CircularDoublyLinkedList** object.

```
class CircularDoublyLinkedList {
private:
    struct Node {
        int number;
        Node* next;
        Node* prev;
    };
    Node* head;
public:
    void reverse ();
};
```

3. Using list<T>::iterator

We will now use the official STL **list** and **iterator** to implement some functions.

(a) Reverse list

Implement a **reverse** function that reverses an STL **list<int>.**

```
void reverse( list< int >& linkList );
```

(b) Remove duplicates from a list

Implement a **removeDuplicates** function that removes duplicate elements in an STL **list<int>**. You may assume the **list** is already sorted in ascending order.

```
void removeDuplicates( list< int >& linkList );
```