Assignment 3 Technical Report

2171039 Chaewon Lee

Code 2

This code merges two list, a and b in ascending order, and return this result by the list c.

First, to insert the elements inside the list, there are insert_node, get_node_at, and add function. These functions are same as it is in the lecture file.

After adding elements inside list a and b, to show that all the elements are put well in the list, first this code displays what is inside the list. You can check that all elements are inserted properly.

Next, there is merge function to merge these two lists in ascending order, and return the result of this operation, new list c. First, we set three indexes, idx : index of c, I : index of a, j : index of b. And we find the length of a and b and declare them as a variable.

Then, while i or j reaches the length of their list, this code keep compares data inside a and b then inserts the proper data into the list. When data inside b is bigger, the code put data of b and then move the idx, and j right. On the other hand, when data inside a is bigger, the code put data of a. Then the node is reset to the data inside that node.

Even I or j reached the end, there still could be elements that are not merged inside c. To check this, we use two loops that guarantee that both I and j reached the end of their length of the list and all elements are inside the new list c.

Finally, the function returns the c and with display function, we can see that all elements of a and b are merged well.

During this merge function, the insertion operated is identical with the sum of the list a and list b's length. Therefore, we could say that the time complexity of this function is O(n+m). (n : length of a, m : length of b).

- **Result**

```
( 1 2 5 10 15 20 25 )
( 3 7 8 15 18 30 )
( 1 2 3 5 7 8 10 15 15 18 20 25 30 )
```

Code 3

This code is an implementation of list ADT with list which contains both head pointer and tail pointer.

Most of the functions in this code are same as the functions that are in the lecture material, but others are changed to use tail pointer of the list.

First, in add function, add_first and add_last are added. They are operated when you try to put the element at the beginning or end of the list.

Add_first function first checks that the list is empty or not. If the list is empty, link of the node becomes NULL, since there is no upcoming element, and both head pointer and tail pointer points the node.

Add_last function also checks the list is empty or not, and if it is, then the functions does the same this as add_first function did in this condition. If it is not the case, then the function set the link of the node NULL, and the node that tail pointer points is changed to node.

Second, in delete function, delete_first and delete_last are added. They are operated when you try to delete the element from the beginning or end of the list.

Delete_first function set the head pointer of the list to the link of the head pointer. And then, if the length of the list is smaller than 2, it means that there is only one element or no element left in the list. Therefore, tail pointer points same element as head pointer does.

Delete_last function first finds the node before the last node. After finding this element, then function changes tail pointer to point that node. This function also checks the length of the list, and when it comes out that there is only one or no node left inside the list, then head pointer is set to point the same element as tail pointer does.

- **Result**

```
( 10 20 70 30 40 )
( 20 30 )
TRUE
20
```

Explanation : 20 is added in the first of the list.

Then, 30 is added in the last.

Next, 10 is added in the first of the list.

Then, 40 is added in the last.

Finally, 70 is added in the 2 position of the list.

Then, the result would be same as you can see from the picture.

Next, third node of the list, which is 70, is deleted from the list.

Then, the first node of the list, which is 10, is deleted from the list.

Lastly, the last node of the list, which is 40, is deleted from the list.

Therefore, only 20 and 30 are left in the list as you can see from the picture.

Since there is 20 in the list, the code prints true. And, since the first node of the list is 20, the code prints 20.