EE 2331 Data Structures and Algorithms, Semester B, 2009/10

Tutorial 4: Maximal Sublist of a Circular Linked List

Week 4 (4th February, 2010)

The tasks of tutorial exercises are divided into three levels. Level 1 is the basic tasks. You should have enough knowledge to complete them after attending the lecture. Level 2 is the advanced tasks. You should able to tackle them after revision. Level 3 is the challenge tasks which may be out of the syllabus and is optional to answer. I expect you to complete at least task A in the tutorial.

Outcomes of this tutorial

1. Able to create circular linked lists
2. Able to traverse circular linked lists
3. Able to understand pass-by-reference and pass-by-value
4. Able to free the nodes of circular linked lists

Linked lists have different variations. In this tutorial, we will use a circular linked list with dummy header to store and operate on a number of integers.

A circular and linear linked list with dummy header can be visualized as below:

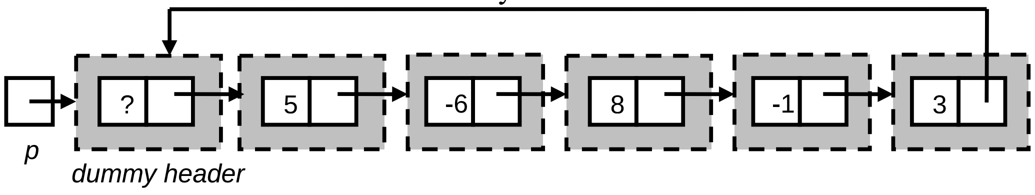


Fig. 1 An example circular linked list with dummy pointer pointed by *p*.

You are reminded that the last node of circular linked lists will point back to the dummy header (or first node if this list has no dummy header). So you should NOT compare the *next* pointers with ***NULL*** value to determine if it is the last node.

The integers are stored in a text file with the following format:

Row	Content	Remark
1st	n	The number of integers in this file
2nd	$a_1\ a_2\ a_3 \dots a_n$	The n integers

The routine of reading the integers from text file will be provided to you. You should concentrate on completing the following tasks only:

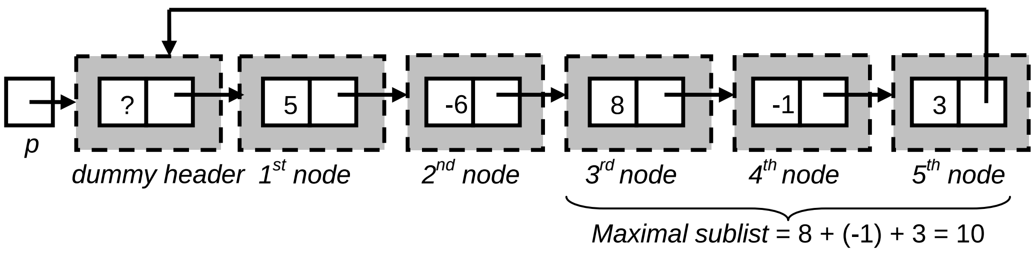
Task A (Level 2): Find the Maximal Sublist of a Circular Linked List

Given a list of n numbers $\{a_1, a_2, ..., a_n\}$, the value of a sublist a_i to $a_j = \sum_{x=i}^{j} a_x$

where $1 \le i \le j \le n$

Complete the function, **int find_max_sublist(Node *_head_)**, that accepts a pointer *head* to a list of integers which may contain negative values and returns the value of the largest sublist of integers.

For example, *p* is the linked list in figure 1:



Expected Output:

```
Enter the file name for testing: test1.txt
The linked list is: Dummy -> 5 -> -6 -> 8 -> -1 -> 3 -> Dummy
Please enter your action ( 1) task a, 2) task b ): 1


The maximal sublist value = 10
The final linked list is: Dummy -> 5 -> -6 -> 8 -> -1 -> 3 -> Dummy
```

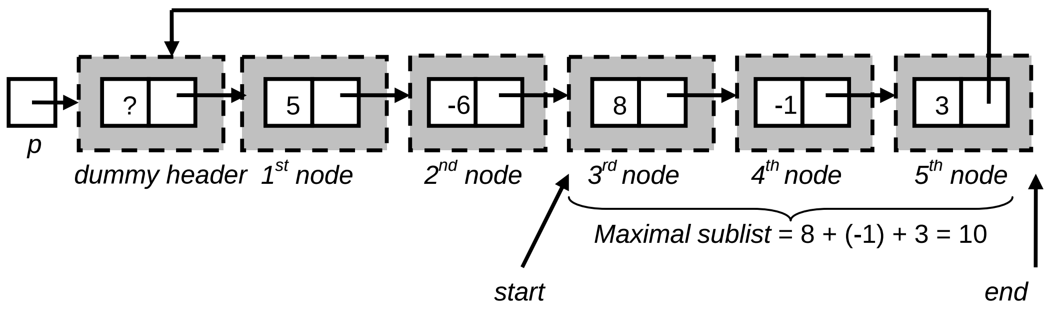
```
Enter the file name for testing: test2.txt
The linked list is: Dummy -> 4 -> -3 -> -2 -> 6 -> 7 -> -3 -> 8 -> -4 -> -2 -
> 2 -> 3 -> Dummy
Please enter your action ( 1) task a, 2) task b ): 1


The maximal sublist value = 18
The final linked list is : Dummy -> 4 -> -3 -> -2 -> 6 -> 7 -> -3 -> 8 -> -4 -
> -2 -> 2 -> 3 -> Dummy
```

Discussion: What's the time complexity of your algorithm in task a? Is there any difference between linear linked lists and circular linked lists? How to modify your code if the list is linear instead of circular?

Task B (Level 3): Find the Maximal Sublist of a Circular Linked List

Complete the function, **int find_max_sublist2(Node ****head***, Node ****start***, Node ****end***)**, that accepts a pointer *head* to a list of integers which may contain negative values, two pointers to pointer of *Node* structure and returns the value of the largest sublist of integers. The function will also determine the pointers to the start node and end node of the maximal sublist.

For example, *p* is the linked list in figure 1:



Expected Output:

```
Enter the file name for testing: test1.txt
The linked list is: Dummy -> 5 -> -6 -> 8 -> -1 -> 3 -> Dummy
Please enter your action ( 1) task a, 2) task b ): 2


The maximal sublist = 8 -> -1 -> 3
The maximal sublist value = 10
The final linked list is: Dummy -> 5 -> -6 -> 8 -> -1 -> 3 -> Dummy
```

```
Enter the file name for testing: test2.txt
The linked list is: Dummy -> 4 -> -3 -> -2 -> 6 -> 7 -> -3 -> 8 -> -4 -> -2 -
> 2 -> 3 -> Dummy
Please enter your action ( 1) task a, 2) task b ): 2


The maximal sublist = 6 -> 7 -> -3 -> 8
The maximal sublist value = 18
The final linked list is : Dummy -> 4 -> -3 -> -2 -> 6 -> 7 -> -3 -> 8 -> -4 -
> -2 -> 2 -> 3 -> Dummy
```

Discussion: Why do we use **Node *****start*** and **Node *****end*** as arguments in task b?

Task C (Level 2): Free Nodes in Circular Linked List

Complete the third function, **void free_nodes_circular(Node *p)**, that accepts a pointer *p* to a circular linked list and deletes the ALL nodes from *p*.

You are reminded that copying the original **free_nodes()** function defined in lecture notes will not work since that function is designed for linear linked list only.

Discussion: Why the program still works if the **free_nodes_circular()** function is empty? Are there any potential problems if we always leave the function blank?