CS29003 ALGORITHMS LABORATORY
(PDS Brush Up)
Date: Sep 3 – September – 2020

Doubly linked list with one pointer per node!

Recall that, in a singly linked list, every node of a linked list stores some data value and a pointer to the next node in the list; the value of the next pointer of the last node is set to NULL. The idea of singly linked list has been extended to doubly linked list as follows. In a doubly linked list, we store a pointer to the next node and another pointer to its previous node. Please refer to Figure 1 for a pictorial overview.

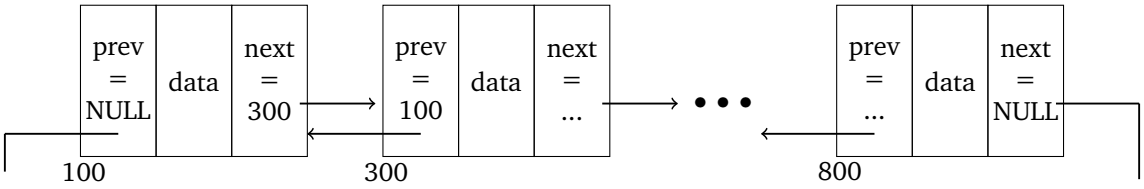

Figure 1: Usual structure of a doubly linked list with two pointers per node.

Let us tweak this basic structure of a doubly linked list as follows. Instead of storing two pointers per node, let us store only XOR of these two pointers in every node thereby saving (in every node) the space required to store one pointer. Please refer to Figure 2 for a pictorial overview.

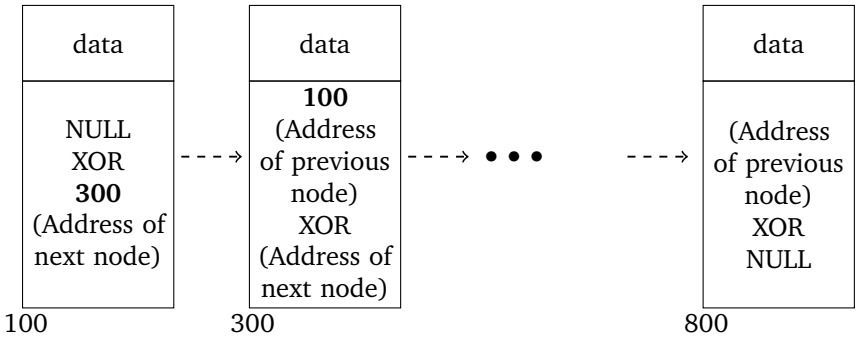

Figure 2: Modified structure of a doubly linked list with one pointer per node.

The XOR function is defined in the following manner. Let $X = (x_1, x_2, \ldots, x_n)$ and $Y = (y_1, y_2, \ldots, y_n)$ be two Boolean strings of length n bits each. Then X XOR $Y = (x_1$ XOR y_1, x_2 XOR $y_2, \ldots, x_n$ XOR $y_n)$ where 0 XOR $0 = 0, 1$ XOR $1 = 0, 0$ XOR $1 = 1, 1$ XOR $0 = 1$. One can easily verify the following properties of the XOR function.

- $\triangleright$ X XOR $Y = Y$ XOR X
- $\triangleright$ $(X$ XOR $Y)$ XOR $Z = X$ XOR $(Y$ XOR $Z)$
- $\triangleright$ X XOR $X = \bar{0}$

Convince yourself that all the usual operations on a doubly linked list can still be carried out correctly in our modified representation. In this exercise, you first take the length n of a doubly linked list from

the user. Then create a doubly linked list of length n with integers values as given by the user. Needless to say that you should only store the XOR of the addresses of the previous and the next node in every node as discussed above. Thus, the following data structure can be used:

```
struct node{
int data; // stores the value in the node.
struct node *add; // stores the XOR value of the next and prev pointer.
};
```

Also, here is how you can perform XOR operation on the addresses.

```
struct node *add, *prev *next;
//Assume that add should store XOR of prev and next
add = (unsigned long long)prev^(unsigned long long)next;
```

Note that for every doubly linked list, you maintain a head and a tail pointer pointing respectively to the first and the last node of the list. In this doubly linked list, you perform the following operations:

1. Traverse the doubly linked list both from the front to the end and from the end to the front and print data values in that order. The prototype of your functions should be as follows.

   void traverse_from_front_to_end(struct node *head);

   void traverse_from_end_to_front(struct node *tail);

2. Reverse your doubly linked list. **Once the doubly linked list is created in the start, you SHOULD NOT create any new node and change data field of any node.** The prototype of your function should be as follows.

   void reverse(struct node **head, struct node **tail);

   Observe that you need to pass double pointers here since the doubly linked list is going to change unlike traversal functions. Use function defined in part 1 to traverse the new list from front to end.

3. Transform the doubly linked list into having alternate values from the beginning and end. After transformations, the first element of the new linked list should be the first element of the original linked list, the second element of the new linked list should be the last element of the original linked list, the third element of the new linked list should be the second element of the original list, the fourth element of the new linked list should be the 2nd last element of the original linked list, and so on. **Once the doubly linked list is created in the start, you SHOULD NOT create any new node and change data field of any node.** The prototype of your function should be as follows:

   void alternate(struct node **head, struct node **tail);

   Here also you need to pass double pointers since the linked list is going to change. Note that this operation needs to be done on the reversed linked list formed in the previous step. Use function defined in part 1 to traverse this list from front to end.

Sample Output

$n = 10$

Enter the 10 integers between -100 to 100: $3, 10, -12, 34, -10, -50, 25, 1, -18, -71$

Doubly linked list traversed from front to end: $3, 10, -12, 34, -10, -50, 25, 1, -18, -71$

Doubly linked list traversed from end to front: $-71, -18, 1, 25, -50, -10, 34, -12, 10, 3$

Reversed doubly linked list traversed from front to end: $-71, -18, 1, 25, -50, -10, 34, -12, 10, 3$

Alternated doubly linked list traversed from front to end: $-71, 3, -18, 10, 1, -12, 25, 34, -50, -10$

File Naming Convention

Please note that your submissions will not be evaluated unless you follow the below specified file naming convention for the program file. <ROLLNO(IN CAPS)>_Assign<Assign_No>_G<Group_No>.c/cpp

Eg: 18CS30004_G03_Assign1.c / 18CS30004_G03_Assign1.cpp