Non-linear Data Structures & problem solving

Assignemnt-3

Topic: XoR List

Normally, a doubly linked list requires two fields for previous and next pointers other than data field(s).

An XOR list requires only one field with each node other than data field(s).

The idea is to store XOR of previous and next addresses in this field.

Example: If A XOR B = C then, A XOR C = B as well as, B XOR C = A

By using this property we can create an XOR list which requires only one field for addressing.

Consider a list as follows:

A -> B -> C -> D

Here, address variable (let's call it npx (XOR of next and previous)) of each node will contain following information:

Node A: npx = NULL XOR address(B)

Node B: npx = address(A) XOR address(C)

Node C: npx = address(B) XOR address(D)

Node D: npx = address(C) XOR NULL

We can traverse this linked list in both forward and reverse directions. To get the next node we will require the address of previous node.

Example: Operations to traverse from A to D (We have Node A) are as follows :

B = NULL XOR A->npx

C = A XOR B->npx

D = B XOR C->npx

Similarly, List can be traversed in backward direction

1. An ordinary Doubly Linked List requires space for two address fields to store the addresses of previous and next nodes. A memory efficient version of Doubly Linked List can be created using only one space for address field with every node. This memory efficient Doubly Linked List is called XOR Linked List or Memory Efficient as the list uses bit-wise XOR operation to save space for one address.
2. Given stream of data of size N for the linked list, your task is to complete the function **insert()** and **printList().** The **insert()** function pushes (or inserts at beginning) the given data in the linked list and the **printList()**function prints the linked list first in forward direction and then in backward direction.

Input:

The insert function takes 2 arguments as input, first the reference pointer to the head of the linked list and an integer data to be inserted in the linked list.

The **printList()** function takes a single argument as input the reference pointer to the head

of the linked list.

Output:

For each test function **printList()** first should print the linked list in the forward direction

and then in the backward direction.

User Task:

1. The task is to complete the functions **insert()** and **printList()** as required.

2. Write a function to Convert doubly Linked List to XOR Linked List