XOR Linked List

solved by	Senan
Platform	GeeksForGeeks
difficulty	Medium
tags	Linked List XOR
language	C++
solved on	@07/10/2024
link	https://www.geeksforgeeks.org/problems/xor-linked-list/1
Completion	☑

Intuition

The given code implements a XOR linked list. XOR linked lists are a memory-efficient version of doubly linked lists. Instead of storing two pointers (previous and next), each node stores a single XOR of the previous and next node's addresses. This allows us to traverse the list both forwards and backwards using XOR operations.

Approach

1. **Insertion:**
   - If the list is empty, we create a new node, set its `npx` to the XOR of `NULL` (both previous and next are `NULL`), and return it as the new head.
   - If the list is not empty, a new node is created. Its `npx` is set as the XOR of `NULL` and the current head, effectively making it the new head. The `npx` of the existing head is updated to link back to the new node.
2. **Traversal:**
   - To traverse, we start from the head and keep track of the previous and current nodes. At each step, we compute the next node using the XOR of the previous node's address and the current node's `npx` field. This way, we can move through the list until we reach the end.

Complexity

Time Complexity:

- **Insertion:**

  Each insertion takes constant time `O(1)` since it only involves creating a new node and updating the `npx` of the head.

- **Traversal (getList):**

  Traversing the list takes linear time `O(n)`, where `n` is the number of nodes in the list, as each node is visited once.

Space Complexity:

- **Insertion:**

  The space complexity of each node is `O(1)`. Overall, the space used for the entire list is `O(n)` where `n` is the number of nodes.

- **Traversal (getList):**

The space complexity for the traversal is `O(n)` for storing the result in the vector.

Code

```
struct Node* insert(struct Node* head, int data) {
    if (head == NULL) {
        struct Node* newNode = new Node(data);
        newNode->npx = XOR(NULL, NULL);
        head = newNode;
        return head;
    } else {
        struct Node* newNode = new Node(data);
        newNode->npx = XOR(NULL, head);
        head->npx = XOR(newNode, XOR(NULL, head->npx));
        head = newNode;
        return head;
    }
}


vector<int> getList(struct Node* head) {
    vector<int> ans;
    struct Node* prev = NULL;
    struct Node* curr = head;
    struct Node* next;

    while (curr != NULL) {
        ans.push_back(curr->data);
        next = XOR(prev, curr->npx);
        prev = curr;
        curr = next;
    }

    return ans;
}
```