XOR Linked List – A Memory Efficient Doubly Linked List | Set 2

In the previous post, we discussed how a Doubly Linked can be created using only one space for address field with every node. In this post, we will discuss implementation of memory efficient doubly linked list. We will mainly discuss following two simple functions.

1) A function to insert a new node at the beginning.
2) A function to traverse the list in forward direction.

In the following code, *insert()* function inserts a new node at the beginning. We need to change the head pointer of Linked List, that is why a double pointer is used (See this). Let use first discuss few things again that have been discussed in the previous post. We store XOR of next and previous nodes with every node and we call it npx, which is the only address member we have with every node. When we insert a new node at the beginning, npx of new node will always be XOR of NULL and current head. And npx of current head must be changed to XOR of new node and node next to current head.

printList() traverses the list in forward direction. It prints data values from every node. To traverse the list, we need to get pointer to the next node at every point. We can get the address of next node by keeping track of current node and previous node. If we do XOR of curr->npx and prev, we get the address of next node.

```
/* C/C++ Implementation of Memory efficient Doubly Linked List */
#include <stdio.h>
#include <stdlib.h>

// Node structure of a memory efficient doubly linked list
struct node
{
    int data;
    struct node* npx;  /* XOR of next and previous node */
};

/* returns XORed value of the node addresses */
struct node* XOR (struct node *a, struct node *b)
{
    return (struct node*) ((unsigned int) (a) ^ (unsigned int) (b));
}

/* Insert a node at the begining of the XORed linked list and makes the
   newly inserted node as head */
void insert(struct node **head_ref, int data)
```

```
{
    // Allocate memory for new node
    struct node *new_node  = (struct node *) malloc (sizeof (struct node) );
    new_node->data = data;

    /* Since new node is being inserted at the begining, npx of new node
       will always be XOR of current head and NULL */
    new_node->npx = XOR(*head_ref, NULL);

    /* If linked list is not empty, then npx of current head node will be XOR
       of new node and node next to current head */
    if (*head_ref != NULL)
    {
        // *(head_ref)->npx is XOR of NULL and next. So if we do XOR of
        // it with NULL, we get next
        struct node* next = XOR((*head_ref)->npx,  NULL);
        (*head_ref)->npx = XOR(new_node, next);
    }

    // Change head
    *head_ref = new_node;
}

// prints contents of doubly linked list in forward direction
void printList (struct node *head)
{
    struct node *curr = head;
    struct node *prev = NULL;
    struct node *next;

    printf ("Following are the nodes of Linked List: \n");

    while (curr != NULL)
    {
        // print current node
        printf ("%d ", curr->data);

        // get address of next node: curr->npx is next^prev, so curr->npx^prev
        // will be next^prev^prev which is next
        next = XOR (prev, curr->npx);

        // update prev and curr for next iteration
        prev = curr;
        curr = next;
    }
}

// Driver program to test above functions
int main ()
{
    /* Create following Doubly Linked List
       head-->40<-->30<-->20<-->10   */
    struct node *head = NULL;
    insert(&head, 10);
    insert(&head, 20);
    insert(&head, 30);
    insert(&head, 40);

    // print the created list
    printList (head);

    return (0);
}
```

Run on IDE

Output:

```
Following are the nodes of Linked List:
40 30 20 10
```

Note that XOR of pointers is not defined by C/C++ standard. So the above implementation may not work on all platforms.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

33 Comments Category: Advanced Data Structure Linked Lists Tags: Advanced Data Structures

Related Posts:

- Count Inversions of size three in a give array
- Count inversions in an array | Set 3 (Using BIT)
- Find LCA in Binary Tree using RMQ
- Range Minimum Query (Square Root Decomposition and Sparse Table)
- Find the maximum subarray XOR in a given array
- Treap | Set 2 (Implementation of Search, Insert and Delete)
- Treap (A Randomized Binary Search Tree)
- Find shortest unique prefix for every word in a given list

(Login to Rate and Mark)

4 Average Difficulty : **4/5.0**
Based on **3** vote(s)

Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.