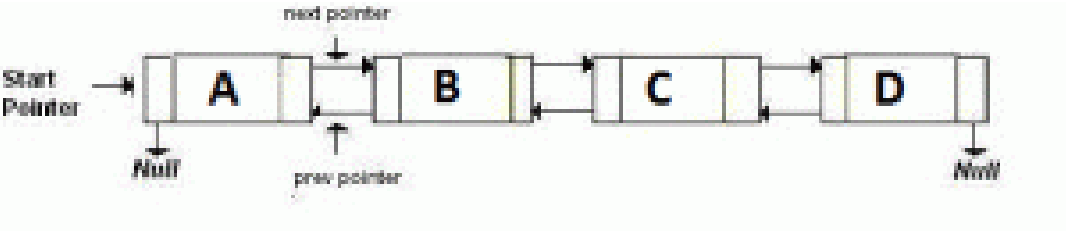
XOR Linked List – A Memory Efficient Doubly Linked List | Set 1

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 bitwise XOR operation to save space for one address. In the XOR linked list, instead of storing actual memory addresses, every node stores the XOR of addresses of previous and next nodes.



Consider the above Doubly Linked List. Following are the Ordinary and XOR (or Memory Effiecient) representations of the Doubly Linked List.

Ordinary Representation:

Node A:

prev = NULL, next = add(B) // previous is NULL and next is address of B

Node B:

prev = add(A), next = add(C) // previous is address of A and next is address of C

Node C:

prev = add(B), next = add(D) // previous is address of B and next is address of D

Node D:

prev = add(C), next = NULL // previous is address of C and next is NULL

XOR List Representation:

Let us call the address variable in XOR representation npx (XOR of next and previous)

Node A:

npx = 0 XOR add(B) // bitwise XOR of zero and address of B

Node B:
npx = add(A) XOR add(C) // bitwise XOR of address of A and address of C

Node C:
npx = add(B) XOR add(D) // bitwise XOR of address of B and address of D

Node D:
npx = add(C) XOR 0 // bitwise XOR of address of C and 0

Traversal of XOR Linked List:
We can traverse the XOR list in both forward and reverse direction. While traversing the list we need to remember the address of the previously accessed node in order to calculate the next node's address. For example when we are at node C, we must have address of B. XOR of add(B) and *npx* of C gives us the add(D). The reason is simple: npx(C) is "add(B) XOR add(D)". If we do xor of npx(C) with add(B), we get the result as "add(B) XOR add(D) XOR add(B)" which is "add(D) XOR 0″ which is "add(D)". So we have the address of next node. Similarly we can traverse the list in backward direction.

We have covered more on XOR Linked List in the following post.

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

References:
http://en.wikipedia.org/wiki/XOR_linked_list
http://www.linuxjournal.com/article/6828?page=0,0

29 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)

3 Average Difficulty : **3/5.0**
Based on **1** 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.

Conventional implementation has a overhead of 2n = O(n)
Xor implementation has a overhead of (n +1) = O(n)

The xor implementation saves half the space which is as good as a doubly linked list can get. Arrays are O(1), but they have their own problems.

3 • Reply • Share ›

coder • a year ago
Really awsome man !!!!!!!!!!

• Reply • Share ›

bale30 • a year ago
wow ! just wow :O

• Reply • Share ›

typing.. • 2 years ago
innovation is here!!!!!!

4 • Reply • Share ›

devil • 3 years ago
Good havens, does anyone ever code in Java here. !crying

1 • Reply • Share ›

subhin • 3 years ago
Can any one publish java code of above program

1 • Reply • Share ›

devil subhin • 3 years ago
http://cocoadev.com/wiki/Desig... Some explanation. Just incorporate this in your LL implementation.

• Reply • Share ›

Holden devil • 6 months ago
The link is broken :(

• Reply • Share ›

devil subhin • 3 years ago
Don't worry. I am going to try and come up with it here. Hold on..

1 • Reply • Share ›

Holden devil • 6 months ago
we are waiting still... after 2 years :I

1 • Reply • Share ›

JavaCoder ➚ Holden • 6 months ago

ha ha. You should not have waited so long just for a program.

"Honesty is of God and dishonesty of the devil; the devil was a liar from the beginning." (Just kidding...)

1 ▲ ▼ • Reply • Share ›

Holden ➚ JavaCoder • 6 months ago

agree with your "Just kidding"!!! :)

▲ ▼ • Reply • Share ›

Atul • 4 years ago

Somehow I didn't like the "node* next" in the given source code. Since it is the distance between the locations, why can't it be a simple number? Hence I implemented following.

```
 #include <stdio.h>
#include <stdlib.h>

typedef struct node
{
  int val;
  unsigned int pnx;  /* prev, next ptr XOR'ed value */
} NODE;
NODE *head, *tail;

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

see more

▲ ▼ • Reply • Share ›

Sudha • 4 years ago

// Insertion, Deletion and Both direction traversal.

#include
#include

struct node
{
int num;
struct node *ptrdiff;

```
};

void insert(struct node**,struct node**,struct node*,struct node*);
void displayForward(struct node*,struct node*);
void displayBackward(struct node*,struct node*);
struct node* newnode(int);
struct node *XOR(struct node *, struct node *);
void delete_node(struct node **,struct node **,int);

int main()
```

see more

• Reply • Share ›

code1234 ➚ Sudha • 3 years ago

Works very well! Great, thanks! :)

• Reply • Share ›

raj • 4 years ago

```
/* */
[#include
#include
typedef struct node
{
int data;
struct node *npx;
}node;
void createlist(node **p)
{
node *prev=NULL,*next,*current;
int i,j,n,x;
current=*p;
while(1)
{
```

see more

• Reply • Share ›

Yogesh • 4 years ago

But we always need the prev node address in order to traverse from a given node pointer. Its more like a single linked list where we rem the prev node of a current node ptr.

DS+Algo ➚ Yogesh • a year ago

yeah! I also didn't find it much better, Only advantage is that if we traverse the list we can go in both direction that is not possible in singly linked list.

1 • Reply • Share ›

kevindra • 5 years ago

I think there is something wrong with formatting of code. It's taking "" as >

• Reply • Share ›

GeeksforGeeks ➚ kevindra • 5 years ago

@kevindra: There seems to be some issue with formatting. We will look into this issue. As a temporary fix, we have updated the code with pre tags and the code is readable.

1 • Reply • Share ›

kevindra • 5 years ago

```
#include < iostream >

using namespace std;

struct node{
  int v;
  node *next;
};
node *start = NULL;
node *end = NULL;

node *newNode(int v){
  node *np = new node;
  np->v = v;
  np->next  = NULL;
  return np;
}
```

see more

• Reply • Share ›

kevindra • 5 years ago

Here is the working code for insertion and traversal (both directions) in XOR linked list:

```
#include <iostream>

using namespace std;



struct node{
  int v;
  node *next;
};
node *start = NULL;
node *end = NULL;

node *newNode(int v){
  node *np = new node;
  np->v = v;
```

see more

• Reply • Share ›

ktanay • 5 years ago

//minor typo
npx = add(A) XOR add(C) // bitwise XOR of address of A and address of B

// bitwise XOR of address of A and address of C

1 • Reply • Share ›

GeeksforGeeks ktanay • 5 years ago

@ktanay: Thanks for pointing this out. We have corrected the typo.

• Reply • Share ›

rajcools • 5 years ago

we are able to save memory but per node time of execution is increasing!!!! time - space tradeoff

• Reply • Share ›

kl rajcools • 5 years ago

struct b
{
int a;
int b;
};

• Reply • Share ›