East West University

Department of CSE

Project Report

Course Code: CSE207

Course Name: Data Structure

Submitted By:

Sifat Noor Siam

ID: 2022-1-60-368

Tasnim Saima Raita

ID: 2022-1-60-291

Md. Saeikh Abdullah

ID: 2022-1-60-304

Section: 02

Semester: Fall-2023

Submitted To:

Dr. Maheen Islam

Chairperson, Associate Professor

Department of Computer Science and Engineering

East West University

Date of Submission: 28 December ,2023

Introduction

XOR linked list is a data structure that uses the bitwise XOR operation to store the memory addresses of the next and previous nodes like a doubly linked list.

Algorithm

1.**X-OR Function:**

```
Algorithm XOR(p, q):
return (node*)((uintptr_t)(p) ^ (uintptr_t)(q))
```

2.**Create Node Function:**

```
Algorithm CreateNode(d):
   node = malloc(sizeof(node))
   if node is NULL:
      print "Memory allocation failed."
      exit(EXIT_FAILURE)
   node->data = d
   node->link = NULL
   return node
```

3.**Create List Function:**

```
Algorithm CreateList():
    Input: None
    Output: Head of XOR Linked List

    h = NULL
    t = NULL

    print "Enter the size of the list: "
    read x

    for i = 1 to x:
        print "Enter the element ", i, ": "
        read d

        m = CreateNode(d)

        if h is NULL:
            h = m
            t = m
        else:
            t->link = XOR(m, t->link)
            m->link = t
            t = m

    return h
```

4.**Display Function:**

```
Algorithm Display(h):
    Input: Head of XOR Linked List
    Output: None

    c = h
    prev = NULL
    next = NULL

    while c is not NULL:
        print c->data, "--> "
        next = XOR(prev, c->link)
        prev = c
        c = next

    print "NULL"
```

5.**Insert at Beginning Function:**

```
Algorithm InsertAtBeginning(h, d):
    Input: Head of XOR Linked List, Data to be inserted
    Output: Updated Head of XOR Linked List

    m = CreateNode(d)
    m->link = h

    if h is not NULL:
        h->link = XOR(m, h->link)
    h = m
    return h
```

6.Insert at position Function:

```
Algorithm InsertAtPosition(h, d, p):
    Input: Head of XOR Linked List, Data to be inserted, Position
    Output: None

    if p <= 0:
        call InsertAtBeginning(h, d)
        return

    m = CreateNode(d)

    if h is NULL:
        m->link = NULL
        h = m
        return

    c = h
    prev = NULL
    next = NULL
    a = 1

    while c is not NULL and a < p:
        next = XOR(prev, c->link)
        prev = c
        c = next
        a++

    if c is NULL:
        prev->link = XOR(m, prev->link)
        m->link = prev
    else:
        m->link = XOR(prev, c)
        prev->link = XOR(XOR(prev->link, c), m)

        if c->link is not NULL:
            c->link = XOR(m, XOR(prev, c->link))
```

7.Delete at Beginning Function:

```
Algorithm DeleteAtBeginning(h):
    Input: Head of XOR Linked List
    Output: None

    if h is NULL:
        print "List is empty. Cannot delete."
        return

    next = XOR(NULL, h->link)

    if next is not NULL:
        next->link = XOR(NULL, XOR(h, next->link))

    free(h)
    h = next
```

8.**Delete at position Function:**

```
Algorithm DeleteAtPosition(h, p):
    Input: Head of XOR Linked List, Position
    Output: None

    if p <= 0:
        call DeleteAtBeginning(h)
        return

    if h is NULL:
        print "List is empty. Cannot delete."
        return

    c = h
    prev = NULL
    next = NULL
    a = 1

    while c is not NULL and a < p:
        next = XOR(prev, c->link)
        prev = c
        c = next
        a++

    if c is NULL:
        print "Position not found. Cannot delete."
    else:
        next = XOR(prev, c->link)

        if next is not NULL:
            next->link = XOR(prev, XOR(c, next->link))

        if prev is not NULL:
            prev->link = XOR(XOR(prev->link, c), next)

        free(c)
```

9.**Search By Key Function:**

```
Algorithm SearchByKey(h, k):
    Input: Head of XOR Linked List, Key to search
    Output: Node with matching key or NULL

    c = h
    prev = NULL
    next = NULL

    while c is not NULL and c->data != k:
        next = XOR(prev, c->link)
        prev = c
        c = next

    if c is not NULL and c->data == k:
        return c
    else:
        return NULL
```

10. **Reversed list Function:**

```
Algorithm ReverseList(h):
    Input: Head of XOR Linked List
    Output: None

    c = h
    prev = NULL
    next = NULL

    while c is not NULL:
        next = XOR(prev, c->link)
        prev = c
        c = next

    h = prev
```

Conclusion

In conclusion, the XOR linked list approach offers a means of representing doubly linked list nodes using a single pointer instead of the traditional two, resulting in a notable reduction in memory usage. This method allows for optimization in various aspects, including memory efficiency, traversal capabilities, and addressing complexities and challenges associated with implementing XOR linked lists in code.