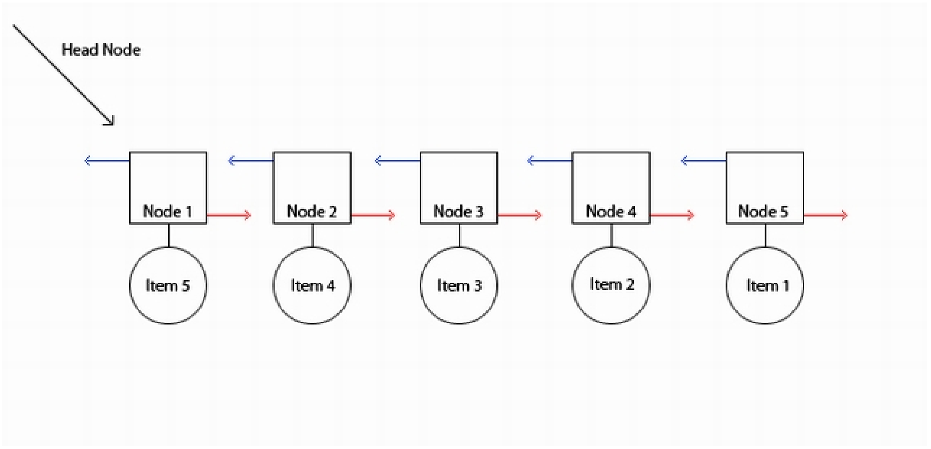
How To Reverse a Linked List (3 Different Ways)

Umut Tezduyar, 13 Jul 2008 GPL3

The purpose of this article is to show how to reverse a linked list

Introduction

There are a couple of ways to reverse a linked list. One of them requires knowledge of pointers and one of them is pretty straight forward. In this article, 3 different methods of reversing a linked list are demonstrated. All the linked list reversing algorithms assume that the given linked list is a double linked list.



Technique 1

In this way, a new linked list will be created and all the items of the first linked list will be added to the new linked list in reverse order.

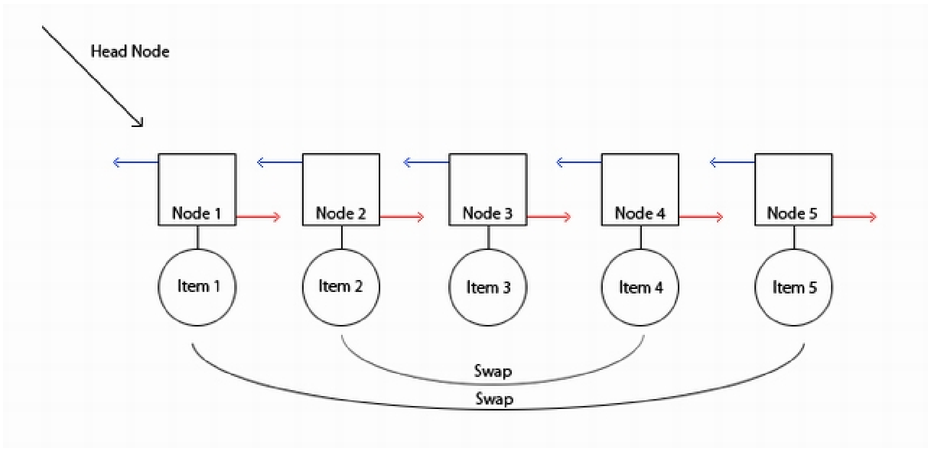
Collapse ▲ | Copy Code

```
public void ReverseLinkedList (LinkedList linkedList)
{
// ------------------------------------------------------------
// Create a new linked list and add all items of given
// linked list to the copy linked list in reverse order
// ------------------------------------------------------------

LinkedList copyList = new LinkedList();

// ------------------------------------------------------------
// Start from the latest node
// ------------------------------------------------------------

LinkedListNode start = linkedList.Tail;

// ------------------------------------------------------------
// Traverse until the first node is found
// ------------------------------------------------------------

while (start != null)
{
// ------------------------------------------------------------
// Add item to the new link list
// ------------------------------------------------------------
copyList.Add (start.Item);

start = start.Previous;
}

linkedList = copyList;

// ------------------------------------------------------------
// That's it!
// ------------------------------------------------------------
}
```

This way is probably the most inefficient among the three. Even though objects of each node are not copied to memory, a new link list node is created for each call to "`Add`" property. A new link list node means at least 3 pointers (`Next`, `Previous`, `Item`) will be created for each item. This method not only wastes memory, but also wastes processing power.

Technique 2

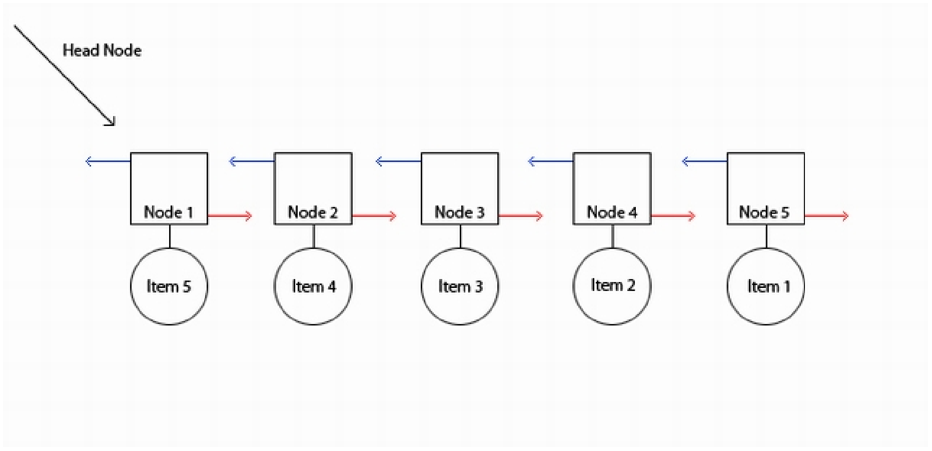
In this method, we will swap linked list node objects (references to the data). Swapping starts from the first node's object and the first node's object is swapped with the last node's object. Then, the second node's object is swapped with the one before the last node's object.



Assuming we have N nodes in the link list:

- Swap: 1st node's object with Nth node's object
- Swap: 2nd node's object with (N-1)th node's object
- Swap: 3rd node's object with (N-2)th node's object

After swapping:



Swapping goes on until the middle of the linked list is found.

Collapse ▲ | Copy Code

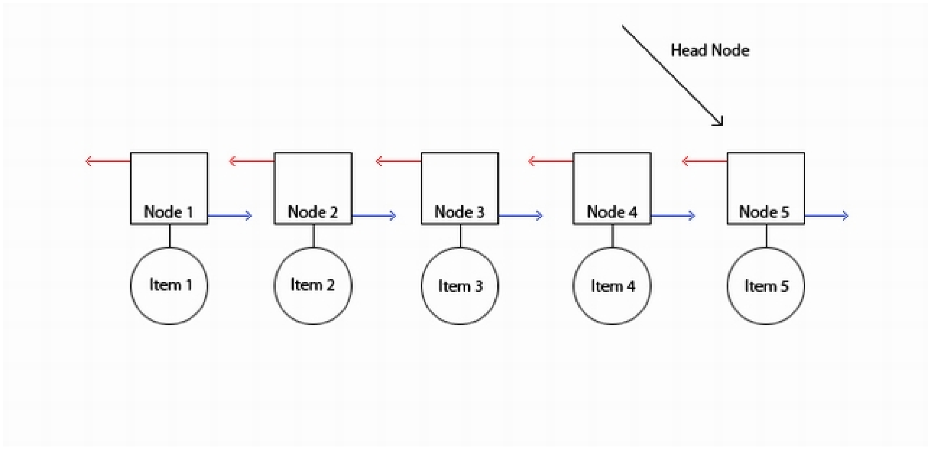
```
public void ReverseLinkedList (LinkedList linkedList)
{
// ------------------------------------------------------------
// Create variables used in the swapping algorithm
// ------------------------------------------------------------
LinkedListNode firstNode; // This node will be the first node in the swapping
LinkedListNode secondNode; // This node will be the second node in the swapping
int numberOfRun; // This variable will be used to determine the number of swapping runs

// ------------------------------------------------------------
// Find the tail of the linked list
// ------------------------------------------------------------

// Assume that our linked list doesn't have a property to find the tail of it.
// In this case, to find the tail, we need to traverse through each node.
// This variable is used to find the tail of the linked list
LinkedListNode tail = linkedList.Head;  // Start from first link list node
                    // and go all the way to the end
while (tail.Next != null)
tail = tail.Next;

// ------------------------------------------------------------
// Assign variables
// ------------------------------------------------------------

firstNode = linkedList.Head;
secondNode = tail;
numberOfRun = linkedList.Length / 2;
// Division will always be integer since the numberOfRun variable is an integer

// ------------------------------------------------------------
// Swap node's objects
// ------------------------------------------------------------

object tempObject; // This will be used in the swapping 2 objects
for (int i=0; i < numberOfRun; i++)
{
// Swap the objects by using a 3rd temporary variable
tempObject = firstNode.Item;
firstNode.Item = secondNode.Item;
secondNode.Item = tempObject;

// Hop to the next node from the beginning and previous node from the end
firstNode = firstNode.Next;
secondNode = secondNode.Previous;
}

// ------------------------------------------------------------
// That's it!
// ------------------------------------------------------------
}
```

This way of reversing a linked list is pretty optimized and pretty fast. The only overhead of this algorithm is finding the end of the linked list.

Technique 3

In this way, the head of the linked list will point to the last node of the linked list. Also, each node's "`Next`" and "`Previous`" properties need to be swapped too.

`Next` (red arrow) and `Previous` (blue arrow) are swapped:



Collapse ▲ | Copy Code

```
public void ReverseLinkedList (LinkedList linkedList)
{
LinkedListNode start = linkedList.Head;
LinkedListNode temp = null;

// ------------------------------------------------------------
// Loop through until null node (next node of the latest node) is found
// ------------------------------------------------------------
```

```
while (start != null)
{
// ------------------------------------------------------------
// Swap the “Next” and “Previous” node properties
// ------------------------------------------------------------

temp = start.Next;
start.Next = start.Previous;
start.Previous = temp;

// ------------------------------------------------------------
// Head property needs to point to the latest node
// ------------------------------------------------------------

if (start.Previous == null)
{
linkedList.Head = start;
}

// ------------------------------------------------------------
// Move on to the next node (since we just swapped
// “Next” and “Previous”
// “Next” is actually the “Previous”
// ------------------------------------------------------------

start = start.Previous;
}

// ------------------------------------------------------------
// That's it!
// ------------------------------------------------------------
}
```

This method of reversing the linked list is performed in a single traverse through the link list. This way is more optimized than the second method. If the reversing requires to keep the link list nodes at their same position, technique 2 will fail.

Conclusion

Method 1 is the most straight forward and it can be implemented pretty quickly. However it uses lots of system resources. Every time a new node is added, a new memory in the heap will be reserved. Method 2 is pretty professional but if link list doesn’t keep track of its tail, link list will be traversed twice. In this case, it will be slower than method 3. Method 3 is the most professional and the fastest one.

Original article can be found here.

License

Share

EMAIL

About the Author

Umut Tezduyar
Software Developer
United States

Interests:
Operating Systems
Embedded Systems
.Net Framework
C#
Asp.net

Eduation:
MS Computer Science

Comments and Discussions

You must Sign In to use this message board.

Search Comments Go

Profile popups Spacing Relaxed Noise Medium Layout Normal Per page 10 Update

First Prev Next

recursive function to reverse a linked list	**Member 9407737**	**5-Sep-12 4:31**
start.Next = start.Previous; does not compile	JayDacca	**28-Sep-08 4:07**
Effect on existing references to list items	**supercat9**	**13-Jul-08 20:28**

Last Visit: 31-Dec-99 19:00 Last Update: 3-Mar-15 2:58 Refresh 1

General News Suggestion Question Bug Answer Joke Rant Admin

Use Ctrl+Left/Right to switch messages, Ctrl+Up/Down to switch threads, Ctrl+Shift+Left/Right to switch pages.