EVALUATING THE REVERSING DATA STRUCTURE ALGORITHM OF LINKED LIST

Rwan F. Al-Rashed, Atheer S. Al-Mutiri, Wadha M. Al-Marrai, Badriayh G. Al-Mutiri, Hanan F. Al-Qahtani, Jehan A. Al-Buainain, Hanaa F. Morse & Enas E. El-Sharawy*

Computer Department, College of Science and Humanities
Imam Abdulrahman Bin Faisal University
P.O.Box 31961, Jubail, Kingdom of Saudi Arabia
*eeelsharawy@iau.edu.sa

Abstract: Rapid development has enabled the production of advanced and important data structures in the programming world, and one of the most important structures of it is "single linked list", that have many features and advantages but it has some problems. One of these problems reversing data in it. Through our aspirations for previous studies, we found that this problem has two ways to solve, namely "Recursion and Iteration", but was our basic goal in this research comparing between these two ways, who is better? Who is way are more efficient?. One of the most common results we found in is: iteration is simpler and less complex than the recursion method, but recursive of the codes we created, we found that they give better results and fewer lines, though its main idea more complex than iteration, and by mathematical analysis of results: lines of codes in iteration 46 but in recursive, there is no big difference 44, Effective line of code in recursive 36 but in iteration was 38, and logical line of code in iteration 30 but in recursive 28, total cyclo complexity in all of them: 6, total function complex in recursive equals 16 but in iteration equals 14.

Keywords: *data structure, single linked list, recursion, iteration, reversing ways*

1. Introduction

Any computer software deals with data, so we need data structures to organize, store, process and use it. These data structures need algorithms to manipulate them. There are many types of data structures like arrays, linked lists, restricted data types, trees, graphs, and many others. Linked list is one of the common data structures [1]. It is probably the second most commonly used in general-purpose storage structure after arrays. it can be used in many cases in which you use an array, a linked list is a linear data structure where each element is a separate object, each element (node) consists of two items "data and reference to the next node", each data item is an object of a class called something like node, The linked list has many advantages like 1- dynamic data structure, 2- insertion and deletion nodes, 3- no memory wastage. The three types of linked list are single, double and circular linked list. But in this research, we will focus on the single linked list. The basic operations on the linked list are “create, insert and delete nodes”. After creating the node we use the insertion and deletion operations to modify the linked list. It’s difficult to reverse the linked list. There are a couple of algorithms that exists to reverse a singly linked list in Java “three-pointer approach using a Stack, or simply using Recursion without the external stack”. The two methods “Recursion and Iteration” are used to make reversing linked list easier [2].

1.1 Problem of research

Identifying the research problem that the study will be based on is important, so the researcher cannot do a study without a problem to solve it, so in our research, this problem of study is how do we reverse single linked list? And what ways can we do the opposite of single linked list? And how do we choose the best methods to reverse it? What are the methods and strategies that will lead us to choose the best methods to reverse single linked list? How do we verify that the method chosen to reverse the list is free of any code errors that may appear to us while applying the single linked list? How do we measure the quality of the method chosen to reverse the list? , So we should refer to previous studies that were previously take place and verify the results and problems encountered while reversing the list and how these problems were resolved? And to compare what we have reached with the results that have emerged for them.

1.2 Goals of research

The Goals of the research are to explain a single linked list, how it works, reversing it by using iterative and recursion methods, fill the holes and shortage in the previous researches, compare between the iterative method and recursion method, and choose the best method for reversing a single linked list from the two methods.

1.3 The importance of the study

The importance of the research is to find the best method to reverse a singly linked list using iterative and recursion methods and use RSM program, which works on analyzing the program codes and gives report about it.

2. Related work

2.1 Recursion vs. Iteration

First, in our search, we need to know the difference between the two used methods for reversing the linked list. As we deduced from this study that both recursion and iteration methods that perform the same tasks "solve a complex task one piece at a time, and combine/merge the results" [3]. The difference between recursion and iteration is that the iteration keeps repeating itself till a task is done (e.g. linked list reaches null pointer), programmers often prefer iterative solutions [4]. The iterative approach is linear and takes O(n) steps. but the recursion solve a major problem by dividing it into smaller and smaller pieces till you can solve it, combine/merge the results (e.g. recursive factorial function), mathematicians often prefer recursive approach because the solutions are shorter and closer. Good recursive solutions may be complicated to be designed and tested. The recursive approach takes O(2n) steps. We conclude that the complexity of the iterative algorithm is simpler than the recursive algorithm [5].

2.2 Linked list problems

Where we reached through this study to an important skill in programming and design is the skill of visualization where the coder can visualize the state of memory to enhance thinking through the solution, linked lists have a visual nature for practicing this kind of thinking. It's easy to draw the state of a linked list and use that drawing to implement the code. As well as the iterative and recursive functions of the singly linked list structure written in C. The iterative Rev () function that reverses a list by rearranging all the .next points and the head pointer. Ideally. The iterative solution is moderately complex. The efficient recursive solution is quite complex. This solution uses the "Push" strategy with the pointer re-arrangement hand-coded inside the loop. There's a slight trickiness in that it needs to save the value of the "cur-> next" pointer at the top of the loop since the body of the loop overwrites that pointer [6].

```
static void Rev (struct node** hRef)
{
        struct node* r=NULL;
    struct node* cur=*hRef;
    struct node* next;
    while (cur != NULL)
    {
            next = cur ->next;
            cur ->next=r;
            r=cur;
            cur=next;
    }
    *hRef=r;
}
```

There is a short and efficient recursive solution. Probably the hardest part is accepting the concept that the Rec_Rev (& rst) does, in fact, reverse the rest. Then there's a trick to getting the one front node all the way to the end of the list. The inefficient solution is to reverse the last n-1 elements of the list, and then iterate all the way down to the new tail and put the old head node there. That solution is very slow compared to the above which gets the head node in the right place without extra iteration [6].

```
Void Rec_Rev (struct node** hRef)
{
struct node* frst;
struct node* rst;
if(*hRef==NULL) return;
frst=*hRef;
rst=frst->next;
if(rst==NULL) return;
Rec_Rev (&rst);
frst->next->next=frst;
frst->next=NULL;
*hRef=rst;
```

```
}
```

2.3 Recursion

Data recursion happens whenever we are defining a new type of value, and we state that one of the fields (some state) is of the type we're defining. In this way, the structure of our values is defined in terms of what the structure of our values looks like. We will explore one very simplified version of a data structure in order to see data recursion. Now, if we create a new class of a list node, and another class of a list (which will just be a list node and a bunch of methods).

```
public class IntList
{
  public IntListNode head;
  public IntList()
  {
        head=null;
  }
}
public class IntListNode
{
  public int val;
  public IntListNode nxt;
  public IntListNode (int val , public IntListNode nxt )
  {
        this.val=val;
        this.nxt=nxt;
  }
}
```

Notice how we are writing a code that deals with recursive data (the addToNode method), and the definition was recursive. This is a common occurrence. We could have defined a non-recursive version of add.

```
public void add_iterative(int v)
{
 if(hed==null)
 {
 hed=new IntListNode(v);
 return;
 }
 IntListNode cr1=hed;
 while(cr1.nxt !=null)
 {
 cr1=cr1.nxt;
 }
 cr1.nxt=new IntListNode(v);
}
….
IntList ilist=new IntList();
Ilist. add_iterative(3);
Ilist. add_iterative(5);
```

Ilist. add_iterative(7);

In this study, we have come to know that there is always a tradeoff between iterative and recursive versions of a method. Iterative versions tend to be faster (whether it's imperceptible or severely apparent), but recursive definitions are often a more natural way to phrase a solution. The successive recursive calls are a natural place to 'store' temporary calculations. Note that there was no current node in the recursive version, it was implicitly tracked by calling the method add(..) on a different IntList object each time. The only overhead is in the effort spent calling a method versus the effort spent going to the next while loop iteration, and that difference is not huge [7].

3. Basic concepts

3.1 Data structure

Data structure is an part of data management. A data structure is a collection of temporary memory places grouped together contiguously like array or not contiguously like linked list, each data structure has a principle work to sort, organize and manipulate the data in a computer memory so that it can be used efficiently. Data structures are used in most programs or software systems. Some popular examples of data structures are arrays, linked lists, queues, stacks, binary trees, and hash tables. Data structures are commonly used in multi aspects [8].

3.2 Linked list

A linked list is a linear data structure that stores a set of data elements of the same or similar types. This data structure is dynamic in the sense that the number of data elements can change. A linked list can grow and shrink during the execution of the program through adding and removing elements, a linked list is a set of node objects. Each node has two fields (data element of some type and pointer to the next node in the list) [9].

3.3 Single linked list

A single linked list is the simplest type of the linked list in which every node contains some data and a pointer that points to the next node of the same data type, which means that it holds the address of the next node in order [8].

3.4 Iterative way

It is the repetition of sentences of code several times. The basic logic behind the repetition is that the code repeat itself until a certain condition is achieved. If this condition is not met, the repetition is stopped and completed the loop, there are various types of iteration statements available. The iteration statements are also known as loops [10].

3.5 Recursion way

Recursion is used to perform a repetitive problem and sometimes provides the solution in a simple way than iterative. Recursion is a programming technique by which a circular definition is employed, a structure is defined in terms of itself. Recursion can be used to describe complex algorithms. A function or method definition that contains a call to is recursive.

A recursive function calls itself with its own body. The definition of function consists of two parts:

1. One or more base cases that define the terminating conditions. In this part, the function returned value that determined by one or more values of the arguments.
2. One or more recursive cases. In this part, the value is returned by the function which depends on the value of the arguments and the previous value which returned by the function [9].

3.6 reverse a single linked list using iterative way

One approach is to use nested loops to repeat traversing the list, looking ahead to see if it is the right time to stop the traverse. The first time, the traversing stops at the node whose next link is null —that is, when we reach the end of the list. We then print the information of the node where the traversing stops. The next time through the list, the traversing stops at the node whose next node link is to the remembered node. This is the second from last node. Again we print the information and remember the node. Continuing in this fashion, each time through the list we stop one node earlier than before, printing the information of the node where we stop. Eventually, we print out the entire contents of the list, in reverse order [11].

3.7 reverse a singly linked list using recursion way

Thinking in terms of recursion. We cannot print the data of the first node before we print the remainder of the list (that is, the tail of the first node). Similarly, we cannot print the data of the second node before we print the tail of the second node, and so on. Every time we consider the tail of a node, we reduce the size of the list by 1. Eventually, the size of the list will be reduced to zero, as the result, the recursion will stop [12].

4. The proposed method

Our proposed method manipulates one of the linked list data structure problems that reversing data in it. We found that this problem has two ways to solve, namely "Recursion and Iteration", but was our basic goal in this research evaluating and comparing and between these two ways.

4.1 Comparing between Recursion and Iterative method

First phase in our proposal is comparing between recursion and iterative for implementing reverse the linked list.

4.1.1 Create a structure in recursion way

The following code is the creation of java code to build the structure in recursion way.

```
package recursion;
public class Recursion
 {
static N head;
      static class N
            {
String subject;
N nextsub;
N(String sub)
{
subject = sub;
nextsub = null;
}
}
N reverseUtil(N current, N previous)
{
if (current.nextsub == null)
{
head = current;
current.nextsub = previous;
return head;
}
N next1 = current.nextsub;
current.nextsub = previous;
reverseUtil(next1, current);
return head;
}
void printList(N node)
{
while (node != null)
{
System.out.print(node.subject;(" <-" + );
node = node.nextsub;
}
System.out.println("null" );
}
public static void main(String[] args)
{
Recursion list=new Recursion();
list.head = new N("Math" );
```

```
list.head.nextsub = new N("Physics" );
list.head.nextsub.nextsub = new N("Chemistry" );
list.head.nextsub.nextsub.nextsub = new N("Biology" );
System.out.println("Original Linked list" );
list.printList(head);
N res = list.reverseUtil(head, null);
System.out.println ("");
System.out.println("Reversed linked list"  );
list.printList(res);
System.out.println("") ;
}
}
```

4.1.2- Create astructureiniterative way.

The following code is the creation of java code to build the structure in iterative way.

```
package iterative;
public class Iterative
 {
static N head;
static class N
{
String subject;
N nextsub;
N(String sub)
{
subject = sub;
nextsub = null;
}
}
N reverse(N node)
{
N previous = null;
N current = node;
N next = null;
while (current != null)
{
next = current.nextsub;
current.nextsub = previous;
previous = current;
current = next;
}
node = previous;
return node;
}
void printList(N node)
{
while (node != null)
{
```

```
System.out.print(node.subject + " →" );
node = node.nextsub;
}
System.out.println("Null");
}
public static void main(String[] args)
{
Iterative list=new Iterative();
list.head = new N("Math" );
list.head.nextsub = new N("Physics" );
list.head.nextsub.nextsub = new N("Chemistry" );
list.head.nextsub.nextsub.nextsub = new N("Biology" );
System.out.println("Orginal Linked list" );
list.printList(head);
head = list.reverse(head);
System.out.println("");
System.out.println("Reversed linked list" );
list.printList(head);
System.out.println("");
}
}
```

4.2 Evaluating the recursion and iterative method

The following tables present the results of using the RSM platform that measure the quality of java code for each method.

Table (1) Evaluating the recursion and iterative method

Compression aspect	Recursion	Iterative
Line of code (loc)	44	46
Effective line of code (eloc)	36	38
logical line of code (IIoc)	28	30
Total cyclo complexity	6	6
Comments lines	0	0
Total lines	44	46
Total function complex	16	14

4.2.1 Definition of Compression aspect:

Lines of Code: specific lines of code in a file that are not blank or sole comment lines.

Effective Lines of Code: specific lines of code in a file that are not blanks, sole comments or stand-alone parenthesis or braces. This metric best represents the quantity of work performed to create the source code document.

Logical Lines of Code: specific lines of code in a file that form a code statement by ending in a semicolon, where the for-loop construct counts as 1 code statement. Comment lines are those lines that contain a comment.

5. Results and discussion

After finishing writing chapters and paragraphs about reverse linked list, from all sources, whether in writing or samples, and based on the assumptions and questions that put in the study problem, the difference between iterative and recursive, the first one is repeated until well done the task, and the pointer going to empty node, most programmer likes iterative, that because iterative is close to them more. Iterative follows principle o(n) but recursive follows o(2n) which solves the large problem by divide to small pieces until we can solve it correctly and combine results and scientists likes these results because it is close to closing in spirit to abstract mathematical entity but it difficult in designing and testing.

Conclusion: complexity in iterative algorithm is simpler than recursive algorithm

When comparing between current result with the result of the previous studies:

Programmers often prefer iterative solutions, it takes O(n) steps, the complexity of the iterative algorithm is simpler than recursive algorithm and the solutions often longer in iterative so Iterative versions tend to be faster and the iterative solution is moderately complex.

Mathematicians often prefer recursive approach, it approach takes O(2n) steps.

complexity of the recursive algorithm is harder than the iterative algorithm and solutions often shorter in it, good solutions may be more difficult to design and test.

Recursive versions tend to be Slowest, and the efficient solution is quite complex

In the current results :complexity of the iterative algorithm is simpler than recursive algorithm and it takes O(n) steps, solutions and line of code often longer in it. iterative solutions tend to be Slowest.

complexity of the recursive algorithm is harder than the iterative algorithm, it approach takes O(2n) steps, solutions and line of code often shorter in it recursive solutions tend to be faster. After we analyzed the results of the previous studies and our own research, we extracted the difference between these results. And we found out: In the previous studies, the iterative version was faster than our research version. In the previous studies, the recursive solution is slower than the solution in our research.

Iterative vs Recursive:

1. Complexity of the iterative algorithm is simpler than recursive algorithm.
2. Solutions often longer in iterative.
3. iterative takes O(n) steps, while the recursive approach takes O(2n) steps.
4. Programmers often prefer iterative solutions, and Mathematicians often prefer recursive approach.

When we picked the Similarity between previous studies and current study, we found compatibility after analyzed the previous studies and our own search Compatibility in complexity of the iterative algorithm is simpler than recursive algorithm.

After comparing the iterative and recursion in reverse linked list. We have achieved by comparing that the length of the code in the iterative 46 line while in the recursion 44 line and the number of logical line of code in the program in the iterative is 30 lines while in the recursion 28 line and the total cyclo complexity of both is equal to 6 and total function complex of iterative 14 while recursion 16 . We can say that recursion is better than iterative because it is the least number of lines of software but the function complex of iterative simpler than recursion.

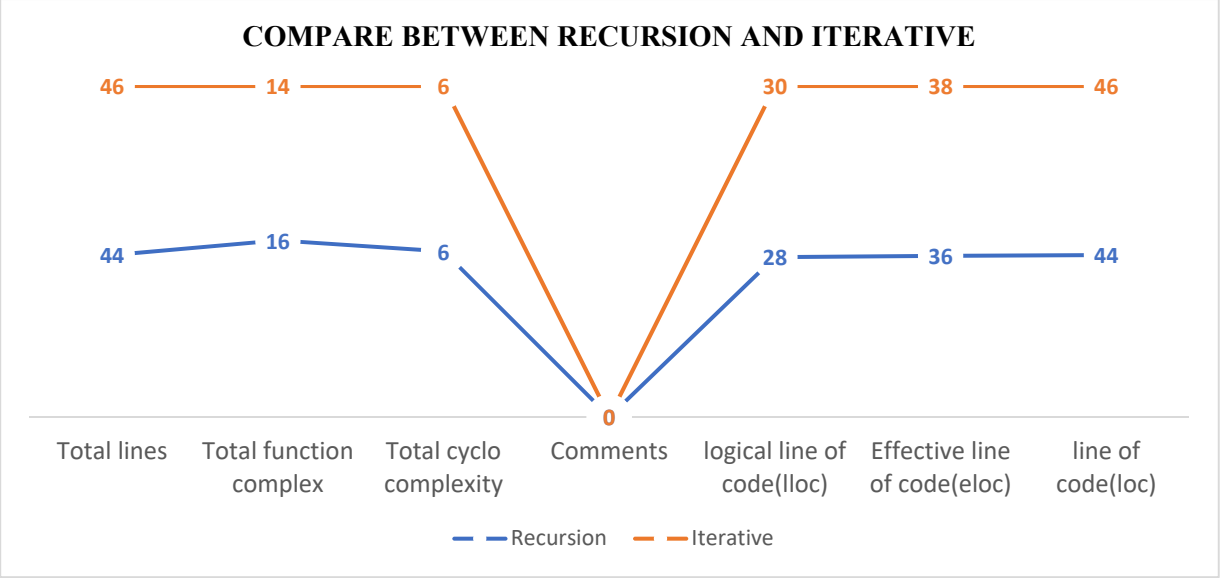

Figure 1. The diagram of comparing the iterative and recursion in reverse linked list.

6. Conclusion and future work

At the end of the research, we would like to mention the most important results from our research when evaluating and comparing between the two reversing ways of linked list that included the following: the complexity of the iterative algorithm is simpler than recursive algorithm although in the previous studies the iterative version was faster than our research version, and recursive solution is slower than the solution in our research .The iterative solutions and line of code often longer than recursive .We suggest continuing to search in this area and propose software code that solve the problem of reverse a single linked list, using the iterative and recursion. We suggest search a solution to a reverse the linked list by using the three-pointers approach or using stack and making a comparison between these two methods

7. References

[1] Michael, T. G.; Roberto, T.; Michael, H. G. (2014). Data structures and algorithms in Java. John Wiley & Sons.
[2] How to Reverse a linked list in Java Using Recursion and Iteration (Loop) - Example. (n.d.). Retrieved from https://javarevisited.blogspot.com/2017/03/how-to-reverse-linked-list-in-java-using-iteration-and-recursion.html.
[3] Vladimir S. (2016). Iteration vs. Recursion in Intro-duction to Programming Classes. An Empirical Study. In Cybernetics and Information Tech-nologies16.4, pp. 63–72.
[4] Raji, R. Nair. (2020). Performance Anatomization of Computer Algorithms Based on Iterative and Recursive Method-ologies. International Journal of Re-cent Technology and Engineering.
[5] S. K. (2016). Advantages-of-recursion-vs-iteration-in-linked-lists. Retrieved from https://www.quora.com/What-are-the-advantages-of-recursion-vs-iteration-in-linked-lists.
[6] [Cornell CS211 Lecture Notes on [Recursion] [online] available from: http://www.cs.cornell.edu/info/courses/spring-98/cs211/lecturenotes/07-recursion.pdf].
[7] Parlante, N. (2001). Linked list basics. Stanford CS Education Library, 1, 25.
[8] Thareja; Reema. (2014) .Data structures using C. 2nd edn. Oxford University Press. India.
[9] Jose. M. G. (2009). Object Oriented Simulation: A Modeling and Programming Perspective .Springer.
[10] Inc. K. L. S. (2009). Java 6 in simple steps. Dreamtech Press. New Delhi.
[11] Dale, N.; Daniel, T. J.; Chip, W. (2016). Object Oriented data structure using java. 4th edn.
[12] Davender, S. M. (2015). C++ Programming: From Problem Analysis to Program Design. Cengage Learning.