Linked list traversal time with Matrix Implementation

Sanjay Razdan
Computer Science Corporation
Noida, INDIA
sanjayrazdan@hotmail.com

Abstract: **This paper proposes the matrix structure of the linked lists to reduce the time required to access any given node . In order to access the Nth node in a linked list we need to traverse all (N-1) nodes. In our matrix structure we do not need to traverse all (N-1) nodes, hence time is reduced.**

Keywords: **Linked list, Matrix, O(N), Pointer.**

I INTRODUCTION

Linked list is a series of data structures which are not adjacent to each other. This data structure has two parts, first part contains the actual data and second part contains the address that points to the next data structure or the data structure that follows it . So the second part of the data structure is actually a pointer to the next data structure. We call this pointer as the "Next" Pointer. This means that we can traverse the linked list with the help of this "Next" pointer. We just need to point this "Next" pointer at the data structure that we want to access.

Linked list implementation is shown in Figure 1.

Linked list data structure can be defined with a series of C++ statements.

```
Struct node
  {
    item   int
    Next *  ptrtype
  }
```

In the above statements we define a structure "node". It contains the variable "item" which holds the actual data and "Next" which contains the address of next data structure. This is called as "Next" pointer.

In order to access the 1st node of the list , we also have a pointer called "Head" that points to the 1st data structure or 1st node.

It is also important to note that the "Next" pointer of the last node points to NULL which indicates the end of linked list.

```
node *   p                    (a)
```

statement (a) defines the variable p that can point to the variable of type "node".
p->data is the portion of the structure where actual data is placed and p->next is where the address which points to the next data structure is located.

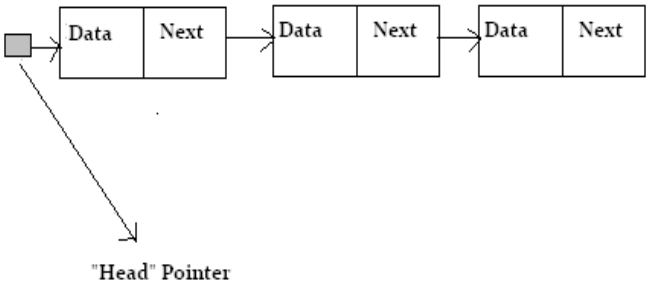

Figure 1. Implementation of linked list.

II LINKED LIST TRAVERSAL

Due to the non adjacent nature of the data structures or nodes, it becomes impossible to access the nodes directly. If we want to access 10th node, we will have to visit all the 9 nodes preceding it . In general if we want to go to Nth node , we will have to visit all (N-1) nodes before it. Thus in the worst case when we want to access last node , we have time complexity of O(N).
To traverse a linked list ,we start with the first node and using the "Next" pointer we visit all the nodes sequentially.

```
For  I = 1  to  N
     {
       p=p->Next;
     }
```

The above statements show how we can traverse nodes by using the "Next" pointer.

III PROPOSED LINKED LIST STRUCTURE

With the Matrix like structure of the linked lists, we have two cases. (A) Linked lists with even number of nodes (B) Linked list with odd number of nodes

(A) Linked list with even number of nodes.

With even number of nodes in a linked list, we arrange the nodes in the matrix structure as shown in Figure 2.
We place N/2 nodes in the 1st row and numder them with odd numbers using some "counter" variable. The remaining N/2 nodes are placed in second row and marked with even numbers. Each node will have three pointers "Next", "Down" and "Diag" as shown in Figure 2. Now if we want to visit the odd numbered node , we will traverse the upper row of the list and in case we have to visit the even numbered node , we will visit the lower row.

. This means we are actually dividing the list into two parts and at any point of time we will have to visit only N/2 nodes to reach Nth node.

(1))Algorithm to access xth node
{

If the node is last node , access using p->diag

if the node is odd numbered , access using p->next.
if the node is even numbered, access using p->down->next

}

(2)) Node Traversal

Following is the node traversal using the above algorithm.

Node 6 : p->diag
Node 3: p->Next
Node 5: p->Next->Next
Node 2: p->down
Node 4: p->down->next

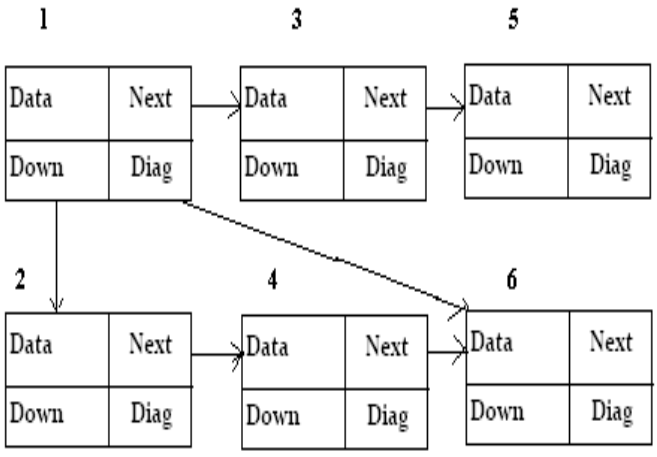

Figure 2. Linked list with even number of nodes.

(B) Linked list with odd number of nodes

With odd number of nodes, we arrange the nodes as shown in Figure 3. If we have N nodes (where N is any odd number), we place (N-1)/ 2 at the 1st row of the matrix and rest of the nodes at the 2nd row as shown.

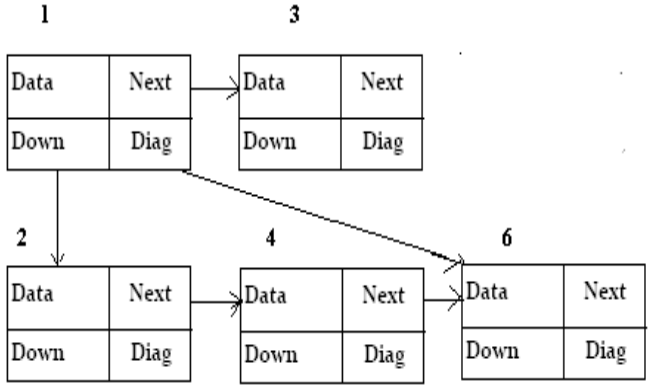

Figure 3. Linked list with odd number of node

The nodes are interconnected as shown in the Figure 3.

(1) Algorithm to access xth node

We can use the same algorithm as (A.1) to traverse this list also.

In this case also, the best case is O(1) when we want to access the last node or 1st in the list . We will achieve worst case O(N/2) when we want to access any other node.
In Figure 3, we can clearly see that node 4 will take the maximum time. To reach node 4 we have to traverse two nodes .
In general apart from node 6 and node 1 which will take O(1) time, all other nodes can be reached in O(N/2) time which is better than O(N) as in linear linked lists.

CONCLUSION

The matrix structure of the linked list divides the list into two branches or rows .While traversing the list we need identify whether the node is even or odd numbered. Based in this we will traverse only a single branch, i.e., (N/2) nodes in the worst case. This means we have better performance than linear linked lists shown in Fig1, where we have the time complexity of O(N).

REFERENCES

[1] Mark Allen Weiss, "Data Structure and Algorithm Analysis in C"
[2] Carrano-Helman-Veroff, "Data abstraction and Problem Solving with C++, pp.173–193.