International Journal For Technological Research In Engineering
Volume 1, Issue 7, March-2014 *ISSN (Online): 2347 - 4718*

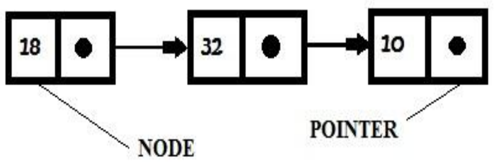
ALGORITHM ON CONCATENATING N NUMBER OF LINKED LISTS

Navrosh Singh Sehra[1], Ankit Malhotra[2]
Department of Information Technology
Amity School of Engineering and Technology
New Delhi, India.

***Abstract*: Computer science has numerous important concepts which are utilized in day to day applications. One such topic is linked list. Linked lists are considered as the most complex concept. In this paper, concatenation of n number of linked lists is explained. The concatenation of linked lists is explained through an algorithm which depicts a better method of concatenating n number of linked lists. Instead of traversing the elements of the linked lists one by one and then proceeding with the concatenation part, the algorithm maintains two arrays of pointers to ease the process of concatenating. The algorithm takes lesser time as the traversing part is skipped in this algorithm.**

I. INTRODUCTION

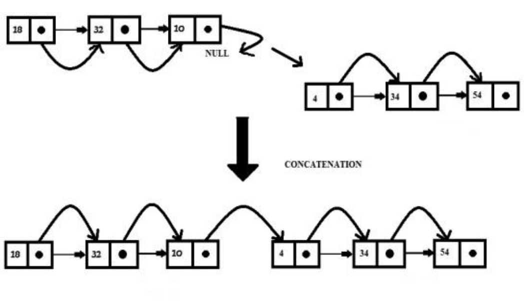
Linked lists can be defined as one of the fundamental data structures which can be used to implement other data structures like arrays, stacks etc. Linked list is characterized by a sequence of number of nodes. These nodes further consist of two parts: a value part and a linking part which is known as pointer.



The former which holds the record or the value to be stored is known as the element, and the latter that holds the address of the next node in the concerned linked list is known as the pointer; as depicted by the figure above. Several types of linked lists persist, each having its own use. Some of them are: singly linked list, doubly linked list, circular linked lists etc. A plethora of operations can be used to mould the linked lists according to the requirement of the scenario.

II. CONCATENATION OF LINKED LISTS

Concatenation of link lists can be explained as the operation in which two or more linked lists are merged in a way that the pointer of the last node of the former linked list, after concatenation, points to the first node of the latter linked list, that is, the pointer contains the address of the first node of the latter linked list.



The usual way of concatenating a set of linked lists is to traverse the whole linked list node by node and then make the pointer of the last node contain the address of the first node of the next link list, instead of a NULL that is an empty slot. And, this process continues for concatenating any number of linked lists. Traversing the linked list to reach the last node of the linked list consumes most of the time; hence traversing has been omitted from this algorithm.

III. ALGORITHM FOR CONCATENATION OF N NUMBER OF LINK LISTS

j: Denotes the current link list.
i: Denotes the current node under of the link list under consideration.
n: Number of link lists
ch: Denotes the choice that whether another element needs to be entered in the current link list.

Create a basic structure for the formation of the link lists.
Declare two instances of the structure in the form of array of pointers: beg [ ], end [ ].
Take three int variables as: n, j, i.
Take three pointers of the defined structure: *first, *tail, *x.
Enter the number of linked lists to be concatenated: n.
Repeat the following steps n (number of linked lists) times:
Allocate the memory to the first node of the linked list that is ‘first’.
Enter the first element in the link list.
Assign NULL to the pointer part of ‘first’.
Make ‘tail’ point to the ‘first’ that is the first node of the current link list.

LABEL: Enter the choice whether more nodes are to be entered: ch
If ch= ‘yes” perform the following steps:
Allocate the memory to the ith node of the link list that is ‘x’.
Assign NULL to the pointer part of ith node.
Enter the next element in the link list.
Assign address to the pointer part of ‘tail’.
Tail=x that is, now tail points to ith node
Increment the value of i.
Now, repeat from step LABEL.

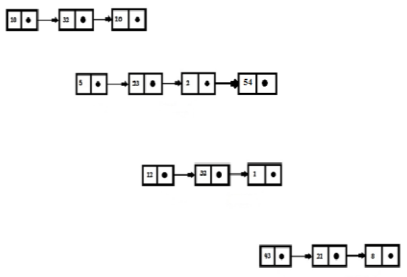
Else, beg[j] =first, that is pointer at jth position of *beg [ ], points to the first node of (j+1) link list.
And, end[j] =tail, that is pointer at the jth position of *end [ ], points to the last node of (j+1) link list.

Tail=beg [0], that is tail points to the first node of first link list.
Repeat the following steps (n-1) times for concatenation of created ‘n’ link lists:
End[i]→pointer= beg [i+1] // pointer at the ith position of the *end [ ], points to the last node of the (i+1) linked list, further the pointer of that last node points to the first node of the next link list.

Tail=beg [0]
While (tail! = NULL)
Display the final concatenated linked lists.

IV. EXPLANATION

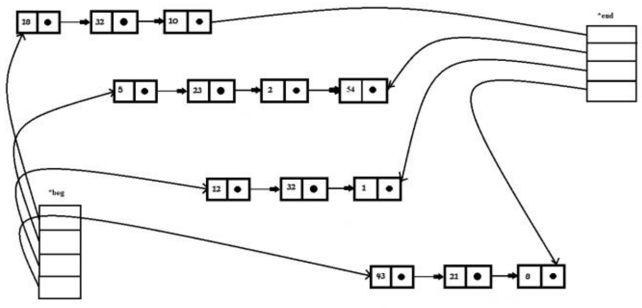
Algorithm will now be explained through an example. As depicted through the figure below, four linked list are considered, which are to be concatenated using the algorithm described in this paper.



*beg [ ] and *end [ ] which are arrays of pointers, are declared, which points to the first and the last node of all the linked lists respectively.
The basic notion of the algorithm is explained in the following steps:

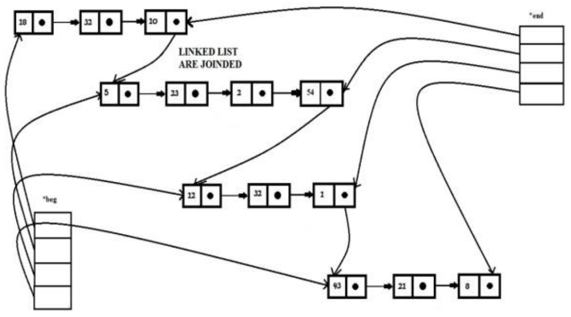
1. Entering the linked lists.
2. Maintaining two arrays of pointers that is * beg [ ] and *end [ ], whose functions have already been explained.
3. Lastly, to concatenate them, the essence of this algorithm is used. All the linked lists under consideration are concatenated at once by making the pointer at the ith position of the end[i], to point to the first node of the (i+1) linked list, which is in turn pointed to by the pointer at (i+1) position of the *beg[ ]. (End[i]→pointer= beg [i+1]).
4. Concatenated linked lists are displayed.

The example in which four linked list was considered are concatenated in a similar fashion. The pointer at the 1st position of the *end [ ] points to the last node of the first linked list. And, the pointer at the first position of the *beg [ ] points to the first node of the first link list, similarly second pointer in *beg [ ] will point to the first node of the second linked list.



CONCATENATION

The algorithm then eases the process of concatenating these four linked list as, the first pointer of *end [ ] is made to point to the last node of the first linked list and this node is made to point to the first node of the second linked list, as this first node of the second linked list is pointed to by the pointer at the second position the *beg [ ]. Thus the first two linked lists are concatenated, and by following the same principle all the other linked lists are concatenated to form a single linked list.



V. CONCLUSION

A unique methodology of concatenating n number of linked list has been presented an algorithm through this paper. The algorithm used to concatenate n number of linked list has been tried and tested. The results have proved that the algorithm is effective and efficient than the earlier method of concatenating the linked list.

The concatenation process do not in involves traversing till the last node, as in the conventional method. The newer method simply makes use of two arrays of pointer to perform the task of concatenation.

VI. FUTURE SCOPE

We significantly plan to extend our research on this algorithm. By comparison with other available methods and estimating the time complexities for both the conventional methods and the newer algorithm, this research will reach a whole new level.

By drawing a parallelism, and explaining through time complexities, the scope of research in this path has just begun. Concatenation of linked list is an open problem, thus newer algorithms which are developed for solving this problem can be compared with the algorithm listed in this paper.

REFERENCES

[1] www.studymode.com.
[2] citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.167.8667&rep=rep1&type=pdf.
[3] cse.iitkgp.ac.in/~pds/semester/2009a/slides/l9-linkedlist.pdf.
[4] C how to program, Deitel and Deitel, 3rd edition.