International Journal of Computer Applications (0975 – 8887)
Volume 182 – No. 18, September 2018

Speeding Up Search in Singly Linked List

Sarvesh Rakesh Bhatnagar
NMIMS University
India

ABSTRACT
Different Search techniques were developed over the period, each of them having certain advantages and disadvantages. Improving the search techniques result in better performance thus better efficiency. Data Structures can be improved by some minor tweaks leading to improving the quality of data structure. In Linked List, searching is slow due to sequential search requirement, which can be improved by properly indexing the list thus improving speed. There are various indexing methods such as uniform indexing, Tree based indexing, dense indexing, clustered indexing, etc. This paper focuses on the indexed based searching using an additional lane linked list and equips a method to incorporate different series such as squared series and cubic series which further leads to speed enhancement as compared with traditional indexing counterparts due to their nature of increasing gaps between sequential indexes, which reduces the dependency on the main linked list and increasing the dependency on the lane linked list thus using the nature of series more efficiently as the list size increases.

General Terms
Data Structures and Algorithms

Keywords
Speeding Up Linked List, Improvising Speed of Linked List, Search techniques, Indexing in Linked List, Indexing techniques.

1. INTRODUCTION
Data Structure is an efficient way of organizing and storing data so as to make it easy in accessing and retrieving the data or making the data well organized to perform different operations on it in a faster and efficient manner [1,2]. There are various data structures available with different advantages and disadvantages. A linked list is a linear node structure where each element is a separate object [3]. Various types of Linked List include Singly Linked List, Doubly Linked List and Circular Linked List. Each of the variation having common disadvantage of additional memory requirement due to usage of pointers which require additional storage while other disadvantage being sequential access to different nodes thus making them slower [4]. In Sequential Search we have to go to each node in the linked list wherein we check if the element on the node matches the element we are searching for, this process needs to be done exhaustively thus consuming a lot of time and computing resources[5], at the same time while in array if the index of a node is known, we can directly access the node without having to sequentially search of it at every step, thus saving a lot of time and resources, But along with the speed , Arrays come with a heavy disadvantage of having a static size, i.e. the size of the array cannot be changed after the execution of the program, thus making arrays inefficient in applications wherein a dynamic storage is required [6].

In Singly Linked List, suppose we have to search for an element N-1, to look for that element, we will have to look at all the elements from 1 to N-1 nodes taking up linear time [8,9]. To reduce this time, we can use the concept of lanes, by creating a linked list in sorted order of ID's (Indexes) and making another dynamically made linked list below it which will act as a table to get the linked list in less time than we require to search an element in a singly linked list. The linked list which will be below our main linked list, we will call it as a lane linked list. Although by addition of a new lane may end up increasing the storage space required, but the cost due to that is minor as compared to the speed gains.

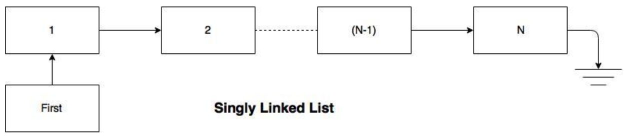

Fig.1: Singly Linked List

Note: - All the coding done for comparison purpose is done in CPP and with compiler – Apple's LLVM Compiler.

2. AIM
Our main goal for this paper is to reduce the time taken to search a particular index. We are trying to create a linked list in which searching is faster than traditional linked list and keeping the dynamic nature of the linked list intact. We want the resultant linked list to be as simple as possible thus making the implementation of it simpler. While doing so we will even try to compare the method found with the traditional approaches available.

3. CONCEPT OF LANES
We use lanes to visit and check minimum number of nodes, when we use lanes, we skip nodes which lies in between the current lane node and previous lane node as per our allotment. When we skip some nodes, we skip the time required to check the nodes which lie in between the current lane node and previous lane node as per our allotment. When we skip some nodes we skip the time required to check the nodes which lie between the two nodes in the lane list. We already have Uniform indexing method but in uniform indexing, the gap is uniform and as the gap is uniform, the number of nodes in k increase at a much higher rate as compared to the indexing using series such as Squared series and Cubic series [7]. When we use lanes, the main cost of time comes from the search in lane, then in the main linked list. Hence if we shift our dependency to the lanes, and try to gradually increase the gap between two consecutive lane nodes we will be able to search at much more efficient rate than normal method. For gradually increasing the gap in between the lane, we are using two series mainly Squared series and Cubic Series

3.1 Squared Series Lane Linked List
A Squared Series is series of i^2 where i varies from 1 to K [7]. A Squared Series Lane Linked List is a Linked List where in the Lane, there is a node for every id (index) which belongs to the squared series.

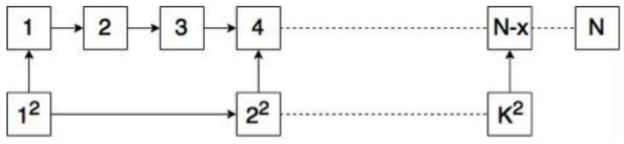

Fig.2: Squared Series Lane Linked List

3.2 CUBIC SERIES LANE LINKED LIST

A Cubic Series is series of i^3 where i varies from 1 to K [7]. A Cubic Series Lane Linked List is a Linked List where in the Lane, there is a node for every id (index) which belongs to the cubic series.

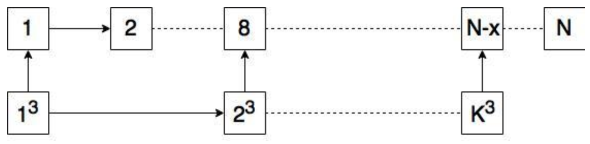

Fig.3: Cubic Series Lane Linked List

4. MATHEMATICAL PROOF

4.1 For Squared Series Lane Linked List

Let There be N nodes in main linked list.

Let There be $(K+1)$ nodes in Lane Linked List.

Let us assume, we have $(K+1)^2 = N$

Let the gap between K^{th} node and $(K+1)^{th}$ node be g.

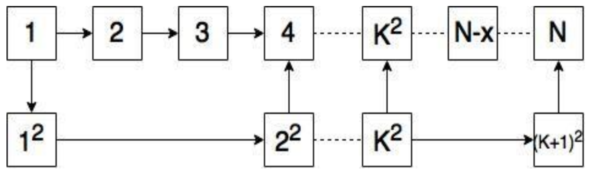

Fig.4: Squared Series Lane Linked List for Mathematical Proof.

We have,

$$g = (k+1)^2 - k^2$$

$$g = k^2 + 2k + 1 - k^2$$

$$g = 2k + 1$$

Hence , $(N-x)^{th}$ term will take at most $(k^2 + g)$ time for normal linked list, and $(k+g)$ time for linked list using squared series lane linked list, where $(N-x)^{th}$ term lies between k^2 &$(k+1)^2$.

$$k \_\_ k^2$$

Hence time taken to search by Squared series Lane Linked List ___ time taken by Normal Linked List.

4.2 For Cubic Series Lane Linked List

Let There be N nodes in Main Linked List.

Let There be $(k+1)$ nodes in Lane Linked List

Let us assume, we have $(k+1)^3 = N$.

Let the gap between k^{th} node and $(k+1)^{th}$ node be g.

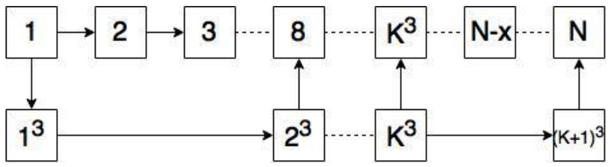

Fig.5: Cubic Series Lane Linked List For Mathematical Proof

We have,

$$g = (k+1)^3 - k^3$$

$$g = k^3 + 3k^2 + 3k + 1 - k^3.$$

$$g = 3k^2 + 3k + 1$$

Hence, $(N-x)^{th}$term will take at most $(k^3 + g)$ time for normal linked list, and $(k+g)$ time for linked list using cubic series lane linked list, where $(N-x)^{th}$ term lies between k^3 & $(k+1)^3$.

We know that,

$$k \_\_ k^3.$$

Hence time taken to search by the Cubic Series Lane Linked List ___ time taken by Normal Linked List.

5. PRACTICAL COMPARISIONS

Note that the practical comparisons will be done in XCODE, in C++ language with the compiler Apple's LLVM.

5.1 Normal Vs Squared Series Linked List

When an exhaustive search was done to measure the time taken for each node, the point where the time taken increased for the Square Series Lane Linked List was noted. The maximum time taken turns out to be 2.734 milliseconds, where as in normal linked list the time taken is 247.321 milliseconds corresponding to the same node.

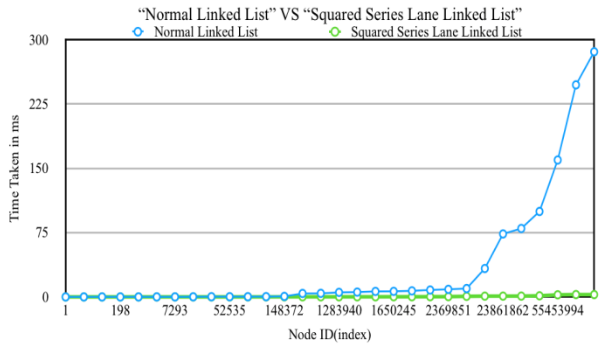

Fig.6: Normal Linked List vs Squared Series Lane Linked List

Table 1.1: Normal Vs Squared Series Linked List

NODE ID	TIME TAKEN FOR NORMAL LINKED LIST (in ms)	TIME TAKEN FOR SQUARED SERIES LINKED LIST (in ms)
1	0.02	0.019
198	0.023	0.025

7293	0.056	0.039
148372	0.647	0.122
1283940	5.315	0.136
1650245	6.519	0.198
2369851	8.865	0.346
23861862	73.491	0.953
55453994	159.679	2.459
79537233	247.321	2.734
100000000	285.953	2.72

5.2 Normal vs Cubic Series Linked List

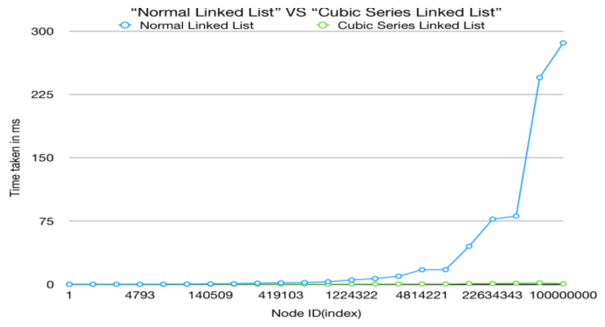

Fig.7: Normal Linked List vs Cubic series Linked List.

When an exhaustive search was done to measure the time taken for each node, the point where the time taken increased for the Cubic Series Lane Linked List was noted. The maximum time taken turns out to be 1.814 milliseconds, where as in normal linked list the time taken is 245.009 milliseconds at the same point.

Table 1.2: Normal vs Cubic Series

NODE ID	TIME TAKEN FOR NORMAL LINKED LIST (in ms)	TIME TAKEN FOR CUBIC SERIES LINKED LIST (in ms)
1	0.02	0.003
4793	0.023	0.025
140509	0.056	0.039
1224322	0.647	0.122
4814221	5.315	0.136
22634343	6.519	0.198
23861862	8.865	0.346
83848902	73.491	0.953
100000000	285.953	2.459

5.3 Normal vs Squared Series vs Mod 10 Indexed Linked List

Now that we saw how well our Squared Series performs in comparison to normal linked list, we will now try to check the same of mod 10 indexed linked list. Mod 10 indexed linked list is there is a referring node at every 10th location of the linked list it provides uniformity in accessing data.

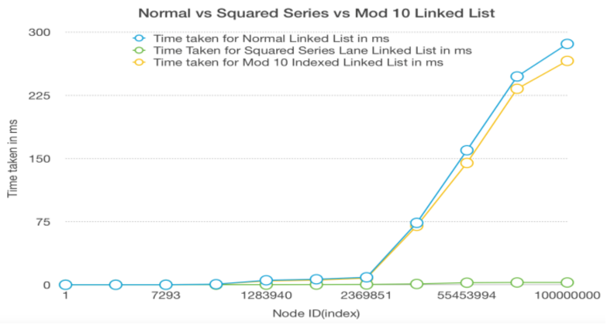

Fig.8: Normal vs Squared Series vs Mod 10 Indexed Linked List.

Table 1.3: Normal vs Squared Series vs Mod 10 Indexed Linked List

NODE ID	TIME TAKEN FOR NORMAL LINKED LIST (in ms)	TIME TAKEN FOR SQUARED SERIES LINKED LIST (in ms)	TIME TAKEN FOR MOD 10 INDEXED LINKED LIST(in ms)
1	0.02	0.019	0.019
198	0.023	0.025	0.019
7293	0.056	0.039	0.028
148372	0.647	0.122	0.543
1283940	5.315	0.136	4.549
1650245	6.519	0.198	5.543
2369851	8.865	0.346	7.881
23861862	73.491	0.953	69.917
55453994	159.679	2.459	144.611
79537233	247.321	2.734	232.697
100000000	285.953	2.72	275.857

5.4 Normal vs Cubic Series vs Mod 10 Indexed Linked List

Now that we saw how well our Cubic Series performs in comparison to normal linked list, we will now try to check the same of mod 10 indexed linked list in comparison to cubic series lane linked list

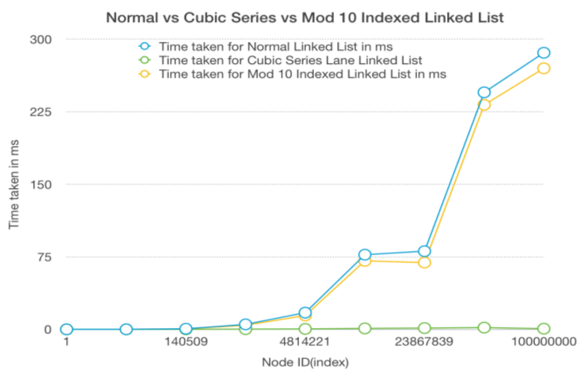

Fig.9: Normal vs Cubic Series Vs Mod 10 Indexed Linked List

Table 1.4: Normal vs Cubic Series Vs Mod 10 Indexed Linked List

NODE ID	**TIME TAKEN FOR NORMAL LINKED LIST (in ms)**	**TIME TAKEN FOR CUBIC SERIES LINKED LIST (in ms)**	**TIME TAKEN FOR MOD 10 INDEXED LINKED LIST(in ms)**
1	0.02	0.003	0.019
4793	0.023	0.025	0.022
140509	0.056	0.039	0.063
1224322	0.647	0.122	0.465
4814221	5.315	0.136	4.347
22634343	6.519	0.198	6.224
23861862	8.865	0.346	7.054
83848902	73.491	0.953	69.065
100000000	285.953	2.459	275.857

5.5 Fibonacci vs Squared Series

When an exhaustive search was done to measure the time taken at each node, the maximum time taken for Fibonacci series indexed linked list was found out to be 98.324ms at node 89876374. Now we will compare the Fibonacci indexed lane with squared series indexed linked list.

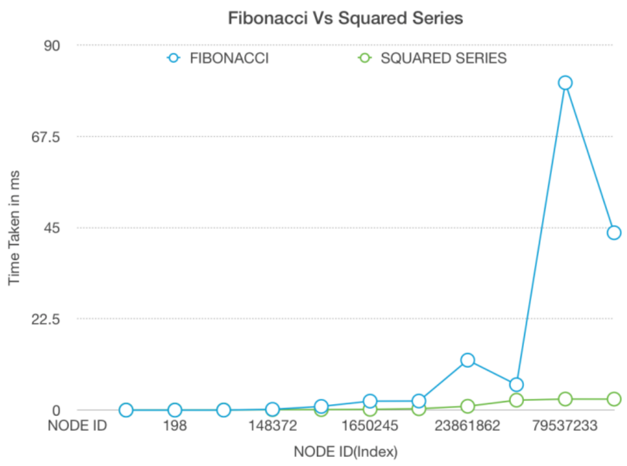

Fig.10: Fibonacci Vs Squared Series

Table 1.5: Fibonacci vs Squared series.

NODE ID	**TIME TAKEN FOR FIBONACCI SERIES LINKED LIST (in ms)**	**TIME TAKEN FOR SQUARED SERIES LINKED LIST (in ms)**
1	0.004	0.019
198	0.004	0.025
7293	0.01	0.039
148372	0.196	0.122
1283940	0.912	0.136
1650245	2.217	0.198
2369851	2.245	0.346
23861862	12.321	0.953
55453994	6.277	2.459
79537233	80.703	2.734
100000000	43.726	2.72

5.6 Fibonacci vs Cubic Series

Now, we will compare the Fibonacci series indexed linked list with the Cubic Series linked list.

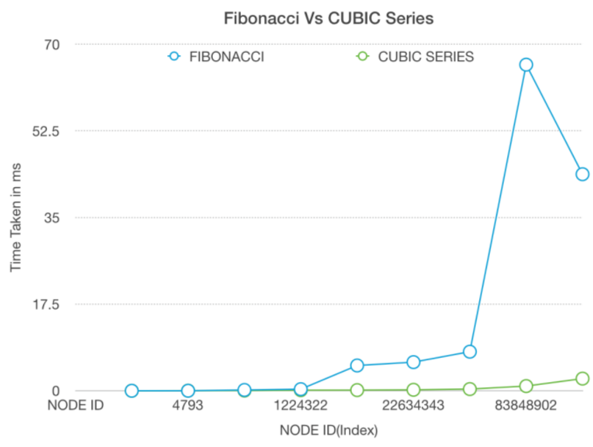

Fig.11: Fibonacci Vs Cubic Series

Table 1.6: Fibonacci Vs Cubic Series

NODE ID	**TIME TAKEN FOR FIBONACCI SERIES LINKED LIST (in ms)**	**TIME TAKEN FOR CUBIC SERIES LINKED LIST (in ms)**
1	0.004	0.003
4793	0.01	0.025
140509	0.162	0.039
1224322	0.325	0.122
4814221	5.105	0.136
22634343	5.784	0.198
23861862	7.893	0.346
83848902	65.843	0.953
100000000	43.726	2.459

6. ANALYSIS OF THE STRUCTURE

In Squared Series and Cubic Series Lane linked list, we are using Squared series and cubic series, in both the series the number of nodes at lane are less, but the actual time taken to search changes dependency as we move farther from the initial point. At first the time taken is dependent upon the lane nodes we are travelling in, but as the number of nodes increase, the gap between them increases, at some point this gap becomes high and the dependency of the time taken to search a particular node shifts towards the gap between the consecutive lane nodes. This point is too far away from the initial point and by then we have skipped a lot of nodes. Hence overall time taken is much less than in normal linked list.

One can argue that it is possible to find the optimal uniform distance wherein indexing should be done, but it should also be noted that the process for finding the optimal uniform distance is time consuming in itself, and on addition to that since we are expecting the dynamic nature to the list, the re-calculation of uniform distance is needed. Thereby adding additional overhead.

Let us look at the best case and worst case time complexity for the Squared Series Linked List and Cubic Series Linked List.

Best Case for Squared series linked list: -

O(1)

Best Case for Cubic series linked list: -

O(1)

Worst Case for Squared series linked list: -

O(K + g)

Where K is number of nodes in lane linked list and g is the gap between N and $\boldsymbol{K^2}$.

Worst Case for Cubic series linked list: -

O(K + g)

Where K is number of nodes in lane linked list and g is the gap between N and $\boldsymbol{K^3}$.

7. ALGORITHMS

SEARCH ALGORITHM PSEUDOCODE FOR SQUARED SERIES LINKED LIST AND CUBIC SERIES LINKED LIST:

```
holdLane = tempLane;

WHILE ( tempLane != NULL) {
    tempID = tempLane.ID
    IF( tempID >= idToBeFound)
    {
        BREAK
    }
    ELSE
    {
        holdLane = tempLane;
        tempLane = tempLane.NEXT
    }
}
temp = holdLane.TOP
WHILE (temp != NULL)
{
    newTempId = temp.ID
    IF( newTempId == idToBeFound)
    {
        idFound = true
        BREAK
    }
    temp = temp.NEXT
}
```

SEARCH ALGORITHM PSEUDOCODE USED FOR SINGLY LINKED LIST :

```
WHILE( temp != NULL) {
    IF( temp.ID == idToBeFound )
    {
        idFound = true
        BREAK
    }
    temp = temp.NEXT
}
```

EXPLAINATION:
SQUARED AND CUBIC SERIES LINKED LIST:
We look in the lane for the id which is to be searched, and keep trailing the previous lane node if we encounter a lane with higher id , we start searching the main list until we encounter the ID. In simple terms, we perform two searches, one in the lane linked list

and other in the main linked list. In lane linked list we only try to find the point from which we will need to search ahead in the main linked list and then we search normally from that point in the main linked list.

SINGLY LINKED LIST:
In Singly Linked list, we search and compare every id with the id to be found in a single list till we encounter the id to be found or the last node, whichever comes first.

8. ADVANTAGES AND DISADVANTAGES

8.1 Advantages of using Squared Series and Cubic Series Lane Linked List

1) Fast Access to the Index to be searched.
2) Less Memory Consumption than other Indexing methods such as Uniform indexing.
3) Easy Implementation, can be used in a lot of applications which uses linked list as their base for better performance.
4) Significantly Faster than Normal Linked List.

8.2 Disadvantages of using Squared Series and Cubic Series Lane Linked List

1) We lose identity when searching using indexed node. i.e. we enumerate the nodes by giving ids.
2) Extra Space Required than Normal Linked List.
3) Data Redundancy, Same id is to be stored in lane linked list and main linked list.

9. APPLICATIONS

1) Can be used in colleges to store student details along with unique identification number.
2) Can be used for storing employee details in large organization, for quick access and retrieval.
3) Can be used for storing network data by enumerating nodes as ids.
4) Can be used in Library Management Systems.
5) Can be used as a new data structure in itself where in we have fast access with dynamic storage**.**

10. CONCLUSION AND FUTURE WORK

We have presented a new data structure for reducing the time cost to search a node in linked list by employing series as an indexing method thus helping to reduce the dependency on main linked list thereby reducing the effective time taken to search a node. We give empirical evidence that run-time performance is significantly improved as compared to traditional search methods and indexing methods as well.

Future work can be done by testing various series available, and even trying to use series which is in Arithmetic sequence or Geometric sequence.

11. REFERENCES

[1] Cormen, Thomas H.; Charles E. Leiserson; Ronald L. Rivest; Clifford Stein(2003). *Introduction to Algorithms.* MIT Press. pp. 205-213 & 501-505. ISBN 0-262-03293-7.

[2] H. Sahni and A. Freed, Fundamentals of Data Structures in C 2nd edition, ch. 4, pp 190-195.

[3] “Defination of a linked list”. National Institute of Standards and Technology.

[4] Antonakos, James L.; Mansfield, Kenneth C., Jr. (1999). *Practical Data Structures Using C/C++*. Prentice-Hall. pp. 165–190. ISBN 0-13-280843-9.

[5] Knuth, Donald (1977). “Section 6.1: Sequential Searching,". *Sorting and Searching*. The Art of Computer Programming. **3** (3rd ed.). Addison-Wesley. pp. 396–408. ISBN 0-201-89685-0.

[6] David R. Richardson (2002), The Book on Data Structures. iUniverse, 112 pages. ISBN 0-595-24039-9, ISBN 978-0-595-24039-5.

[7] Conway, J. H. and Guy, R. K. The Book of Numbers. New York: Springer-Verlag, pp. 30-32, 1996. ISBN 0-387-97993-X

[8] Sipser, Michael (2006). Introduction to the Theory of Computation. Course Technology Inc. ISBN 0-619-21764-2.

[9] Sedgewick, R. and Wayne K(2011). Algorithms 4th ed. p.186. Pearson Education, Inc.