Skip List Data Structure Based New Searching Algorithm and Its Applications: Priority Search

Mustafa Aksu
Department of Computer Technologies
Vocational School of Technical Sciences,
Sutcu Imam University
Kahramanmaras, Turkey

Ali Karcı
Department of Computer Engineering
Faculty of Engineering, Inonu University
Malatya, Turkey

***Abstract*—Our new algorithm, priority search, was created with the help of skip list data structure and algorithms. Skip list data structure consists of linked lists formed in layers, which were linked in a pyramidal way. The time complexity of searching algorithm is equal to O(lgN) in an N-element skip list data structure. The new developed searching algorithm was based on the hit search number for each searched data. If a datum has greater hit search number, then it was upgraded in the skip list data structure to the upper level. That is, the mostly searched data were located in the upper levels of the skip list data structure and rarely searched data were located in the lower levels of the skip list data structure. The pyramidal structure of data was constructed by using the hit search numbers, in another word, frequency of each data. Thus, the time complexity of searching was almost Θ(1) for N records data set. In this paper, the applications of searching algorithms like linear search, binary search, and priority search were realized, and the obtained results were compared. The results demonstrated that priority search algorithm was better than the binary search algorithm.**

***Keywords*—Algorithms; Priority search; Algorithm analysis; Data structures; Performance analysis**

I. Introduction

Various disciplines in computer sciences benefit from algorithms and data structures directly or indirectly, and different data structures were used as solutions to various problems. The limitations like processing, time complexity, required hardware or inefficiency of current algorithms conclude in defining new algorithms such as searching, sorting, and graph algorithms [1].

Sometimes, an algorithm was preferred on another one because of its processing, time complexity, etc. For example, binary search was preferred instead of sequential search to increase searching complexity. Skip list data structures based searching algorithm presented in this study is another option instead of binary search. Considering these factors, it is evident that new algorithms and data structures will continue to emerge as needed [2].

In computer science, the linked list is a data structure consisting of a group of nodes, which together represent a sequence (Fig. 1). The principal benefit of a linked list over a conventional array is that in the linked list elements can easily be inserted or removed without reallocation or reorganization of the entire structure, because the data items need not to be stored contiguously in memory or on disk. Linked lists allow insertion and removal of nodes at any point in the list, and can do so with a constant number of operations if the link previous to the link being added or removed was maintained during list traversal. Linked lists by themselves do not allow random access to the data, or any form of efficient indexing. Thus, many basic operations may require scanning most or all of the list elements [3], [20]. The time complexity of linked list is linear, so, the time complexity of searching in linked list of size N is O(N) [15], [19].

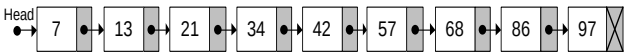

Fig. 1. Linked list

In this study, a new searching algorithm based on the skip list was developed and it was compared to other searching algorithms by doing some applications. The rest of paper was arranged as follows: Related works have been presented in Section II. The skip list data structure must be clarified for the sake of the understandability of developed searching algorithm. Due to this case, Section III explains the skip list data structure. The methodology of proposed algorithm has been explained in Section IV and Section V demonstrates the experimental results and significance of work. The conclusion has been given in Section VI.

II. Related Works

Skip list data structure, which was introduced by Pugh [8], is a data structure alternative to binary tree search structure. Search, insertion and deletion algorithms of nodes in skip list data structure is discussed in article written by Pugh [8]. The time complexity of searching in the skip list data structure is O(lgN). In addition, several studies have been conducted so far on the improvement and analysis of skip list data structure algorithms. In [2], how randomly creation of levels and different "P" thresholds (0.25, 0.5, 0.75) effect the performance was studied and solutions were proposed.

An optimized search algorithm for skip lists was analyzed in [6]. In [7], the probabilistic analysis of the search cost was considered in a slightly different way, namely, performing the asymptotic analysis of the total search cost or path length.

In [12], proposed exploring techniques based on the notion of a skip list to guarantee logarithmic search, insert and delete

costs. The basic idea is to insist on that between any pair of elements above a given height are a small number of elements of precisely that height.

Other studies are about level optimization in skip list data structure [1], formal verification of a lazy concurrent list-based set [4], a simple optimistic skip list algorithm [5], average search and update costs in skip lists [9], skip lists and probabilistic analysis of algorithms [10], the binomial transform and the analysis of skip lists [11], deterministic skip lists [12], a skip lists cookbook [13], and concurrent maintenance of skip lists [14].

Various data structures and algorithms were also created apart from skip list data structure such as Tiara: A self-stabilizing deterministic skip list and skip graph [22],Corona: A Stabilizing Deterministic Message-Passing Skip List [23] and Skip lift: A probabilistic alternative to red–black trees [24].

III. SKIP LIST DATA STRUCTURE

Linked lists were used in skip list data structure and it aimed to facilitate searching, insertion and deletion through placing elements in a pyramid-like order at different levels. In this data structure, elements were placed at different levels randomly.

First, all nodes were placed at level 0 and, starting from left row and skipping each 2^ith node (i=0,..,MaxLevel (15 or 31)), pointers representing each level are created towards the top. The list at level 0 is the linked list at the bottom in skip list data structure and encompasses all nodes. Each list from bottom to the top were arranged as an index of the previous list [1], [19] (Fig. 2).

When levels in skip list data structure were created (level 0, level 1,.., level k), it was done randomly (Pugh's random Level algorithm [8]; for P=1/4). Let us say that the number of ordered nodes in skip list data structure is N. Level 0 consists of these entire N ordered nodes (Fig. 4- Level 0).

Level 1 is created if every other element of the list at Level 0 has also an extra link to the element four ahead of it (Fig. 4 – Level 1). Since the maximum number of elements at Level 1 level equals to $\lceil \frac{N}{4} \rceil + 1$, so on, the data structure will be constructed.

The height of skip list depends on the probability P threshold value given in Pugh's "random Level algorithm". The effects of P threshold values were studied in a previous study [2] and skip list is more efficient when P threshold value is equal to 1/4. While if P=1/2, the height of skip list approaches to height of balanced tree (lgN). If P=1/4, one out of every four nodes in Level 0 copied to Level 1 (an upper level), and this process was continued in the same way until all data structure were constructed. This process resulted in the height of skip list will be half of the height of the balanced tree. These cases are seen in Fig. 2 and Fig. 4 respectively.

A group of data consisting of the elements {zinc, bee, fox, hill, dive, lift, null, total, vary, other, see} on a skip list shown in Fig. 2. The true skip list structure, which was constituted from these elements, is shown in Fig.3.

Time complexity is O(N) for search, insertion and deletion processes when linked and ordered lists are used. On the other hand, the time complexity is O(lg N) in skip list data structure [8], [15] when the same process were performed.

In a search algorithm, a node was searched from upper levels to lower levels. During insertion, first, the node to be inserted was searched. If not found, new value is inserted to the matching location starting from a random level and pointers and lists are updated. The process was repeated for other levels where a node is to be inserted. Search was performed from the top level to lower levels for removal operations. The node was deleted when found and pointers and lists were updated. The process was repeated on other levels where the node is available [2].

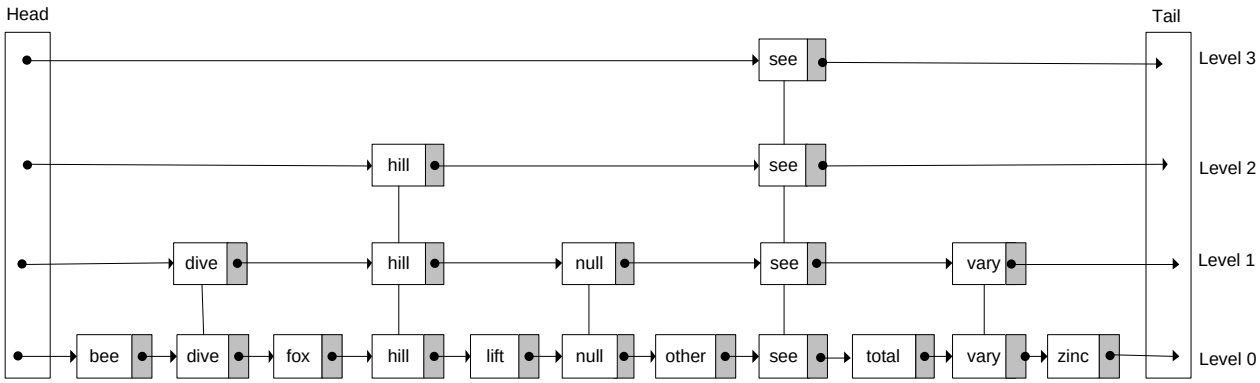

Fig. 2. Skip list (for P=1/2)

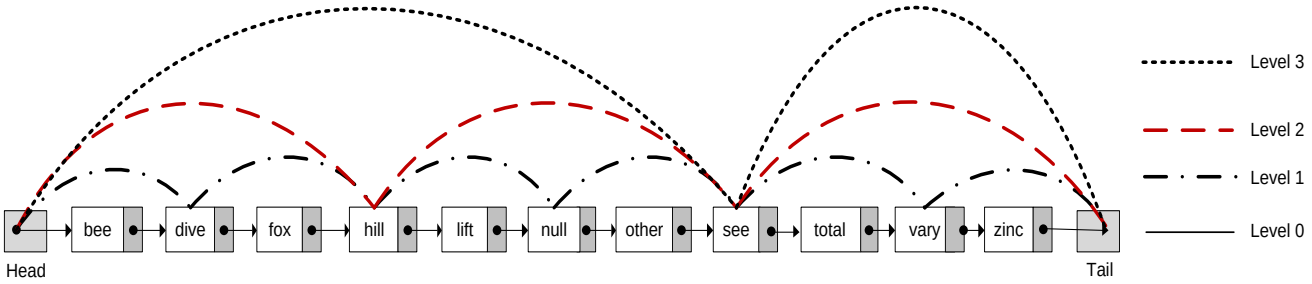

Fig. 3. Skip list (Real structure of skip list for Fig. 2)

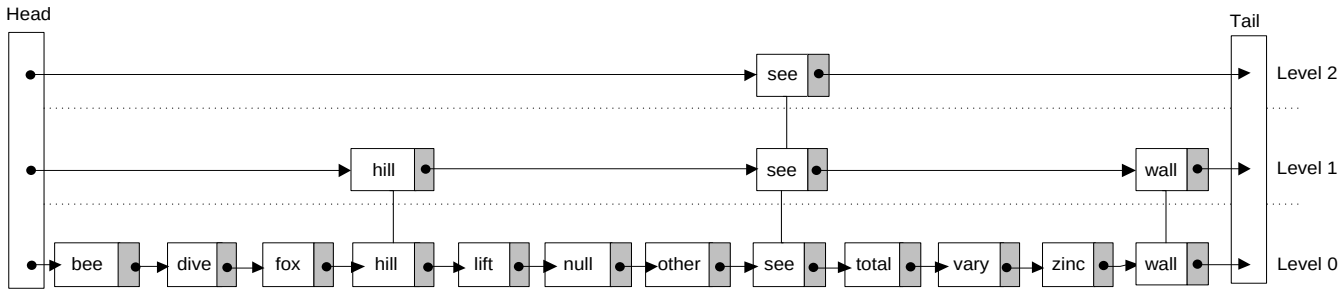

Fig. 4. Skip list (for P=1/4)

IV. Priority Search and Binary Search

The innovative search algorithm which was called priority search uses the skip list data structure. It was benefited from the pyramidal layered-structure of the skip list data structure. The standard searching algorithm (algorithm 1) in the skip list data structure starts at top-level to the lowest level until it finds the searching data or it ends up in the lowest level. The developed new searching algorithm (Algorithm 2) was based on the hit search number for each searched data. If a datum has greater hit search number, then it was upgraded in the skip list data structure to upper level. That is, the mostly searched data were located in the upper levels of the skip list data structure and rarely searched data were located in the lower levels of the skip list data structure. The time complexity of searching in the skip list data structure (Algorithm 1) is O(lgN), but the time complexity of searching algorithm in priority search (Algorithm 2) approximates to Θ(1). In another word, the mostly searched data were located in the top-level of the skip list data structure, thus, the searching for these data has time complexity as ~Θ(1). The rarely searched data were located in the lowest level and their searching time complexities approximate to O(lgN). The time complexity of searching by using priority search algorithm changes between Θ(1)-O(lgN).

When 'dive' two times, 'null' four times and 'vary' three times were searched as in Table I, the results in Table II will be obtained. The skip list data structure for data in Table II is seen in Fig. 5, in which priority search algorithm (Algorithm 2) was used. It was performed by using frequencies (hit search numbers). That is, the searched data is upgraded once for each search process. Therefore, the mostly searched data were located at the top of skip list data structure (pyramidal structure) and rarely searched data were located at the bottom of skip list data structure.

```
Algorithm 1 {Search in skip list }
SearchNode(slist, key)
 HEAD ←slist→head
 LEVEL ←slist→level
 if (HEAD→next[0] = NULL) or (LEVEL<0)
     return false
 for i ← LEVEL downto 0 do
  while(HEAD→next[i]≠NULL
         and HEAD→next[i]→value < key)
  HEAD ←HEAD→next[i]
  HEAD ←HEAD→next[0]
  if (HEAD ≠ NULL and HEAD→value = key)
      return true;
 return false;
```

TABLE I. Frequency-Wise Level Distribution Of Nodes On Fig. 4

nodes	bee	dive	fox	hill	lift	null
frequency	0	0	0	1	0	0
level	0	0	0	1	0	0
nodes	**other**	**see**	**total**	**vary**	**zinc**	**wall**
frequency	0	2	0	0	0	1
level	0	2	0	0	0	1

The priority search algorithm was used in the skip list data structure due to its pyramidal structure. Additionally, the standard search algorithm (Algorithm 1) for the skip list of size N has time complexity as O(lgN). Data were sorted in ascending order in the skip list data structure when skip list data structure were constructed (Fig. 5 Level0, Level1, Level2, Level 3, and Level 4). The most important property of skip list data structure is its pyramidal structure and ordered data in it.

The searching process started at the first element in the list and carried on till the end of list, when data were unordered.

So, the searching algorithm is a linear algorithm in term of the number of data in the list. The time complexity of linear search is O(N). The searching process considered the data as unordered whether data were ordered or not. But it is not a suitable search process for ordered data [16], [17], [21].

TABLE II. FREQUENCY-WISE LEVEL DISTRIBUTION OF NODES ON FIG. 5

nodes	**bee**	**dive**	**fox**	**hill**	**lift**	**null**
frequency	0	2	0	1	0	4
Level	0	2	0	1	0	4
nodes	**other**	**see**	**total**	**vary**	**zinc**	**wall**
frequency	0	2	0	3	0	1
Level	0	2	0	3	0	1

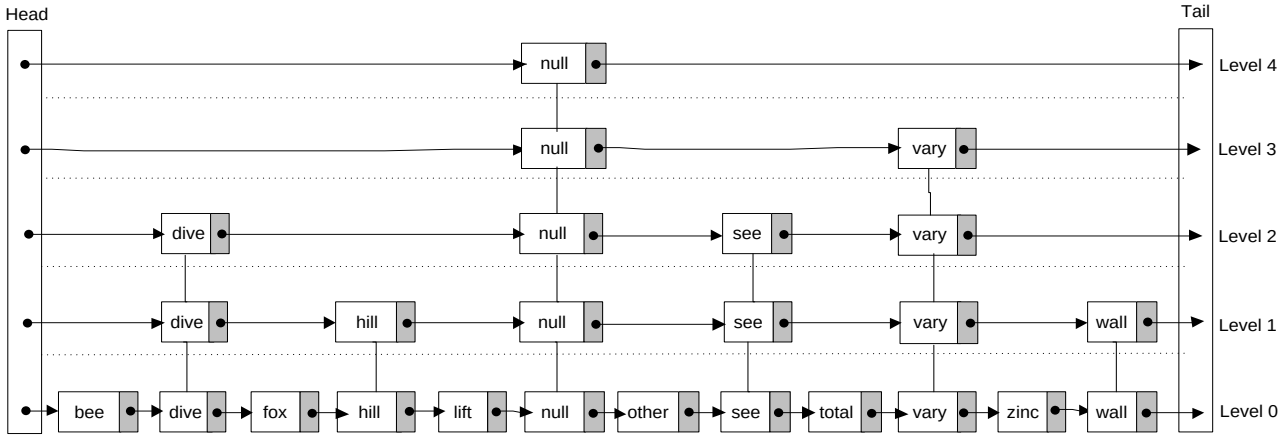

Fig. 5. Obtained Priority Search schemes for searching 'dive' two times, 'null' four times and 'vary' three times on Fig. 4)

```
Algorithm 2 {Priority search}
PrioritySearch(slist, search_value)
 HEAD←slist→head ,
 LEVEL←slist→level
 update[MaxLevel +1]
 while (LEVEL>=0)
 if(HEAD->next[LEVEL]->value=search_value)
  for i ← LEVEL downto 0 do
    while (HEAD→next[i] ≠ NULL and
      HEAD→next[i]→value <search_value)
    HEAD←HEAD→next[i]
    update[i] ← HEAD
   end for
  HEAD←HEAD→next[0]
  intlvl = LEVEL+1;
   if(lvl>slist->level)
     update[lvl] = slist->head
     slist->level = lvl
   end if
  HEAD->next[lvl] = update[lvl]->next[lvl]
  update[lvl]->next[lvl] = HEAD;
 return true
 end if
 if(HEAD->next[LEVEL]->value<search_value)
    HEAD= HEAD->next[LEVEL]
 if(HEAD->next[LEVEL]->value>search_value)
    LEVEL=LEVEL-1
end while
return false;
```

Another searching algorithm is binary search algorithm for ordered data. In order to use this algorithm, data have to be ordered on the list. If data were unordered, initially they must be ordered by using any sorting algorithm.

The mechanism of binary searching algorithm is as follows [15], [16], [18], [20]:

- If list or array is not sorted, it is firstly sorted.
- Sorted array is divided into two equal sub-arrays or approximately equal sub-arrays.
- The searched data is compared with the middle element of array. If it is equal then, it is found. If searched data is less than the middle element of array, then right sub-array is discarded and data will be searched in the left sub-array. If searched data are greater than the middle element of array, then searched data will be searched in the right sub-array.
- The searched data will be scanned on the left or right sub-array in the same manner.
- The process goes on in the same manner until searched data is found or search is terminated.

The time complexity of binary search algorithm is O(lgN) for an array of size N elements. Moreover, the time complexity of binary search for balanced binary trees is also O(lgN).

V. Experimental Results: Priority Search and Others

The proposed algorithm was implemented by using C++ and tested successfully on distinct arrays. In order to compare Priority Search (PS), Linear Search (LS), and Binary Search (BS), random arrays and sorted arrays were used. The searching times of PS, LS and BS for sizes from 1000 to 100000 of arrays were illustrated in the Table III and Table IV. Moreover, each algorithm was applied to same size arrays 100 times and all times for all executions was added up and then their average was computed. This means that the effect of data permutation will be minimized and the comparison will be more equitable. If there is one search for algorithm, the comparisons may be non-equitable. For example, searched data for PS may be on the top level of skip list data structure, and then its time will be Θ(1). If the searched data for BS is not found in the binary search tree, then its time will be longer. This case may be available for each search algorithm. Due to this case, there were 100 executions for equitable comparisons of search algorithms.

All results were obtained on the same computer and the results in Table III and Table IV demonstrated that when size of array is small, BS shows normal performance; when the size of array increases, the performance of PS increases and PS is better than LS and BS. The results were illustrated in Fig. 6.

TABLE III. Searching Timesfor LS, BS and PS for Sorted Arrays (if the Searched Data are Near to the Begining of Array) (Ms=Millisecond)

# of nodes	1000	5000	10000	30000	50000	100000
LS	0.0032 ms	0.0103 ms	0.0167 ms	0.0374 ms	0.0671 ms	0.1382 ms
BS	0.00015 ms	0.00018 ms	0.00020 ms	0.00022 ms	0.00023 ms	0.00025 ms
PS	0.00009 ms	0.00011 ms	0.00013 ms	0.00016 ms	0.00017 ms	0.00019 ms

Table III, Table IV and Fig. 6, Fig. 7 depict that PS is better than LS and BS with respect to searching time. The time complexities for searching PS, and BS on sorted arrays are O(lgN). The time complexities for searching LS on sorted array is O(N). While computing time complexity for any algorithm, the dominant (term with the greatest degree) term is regarded as time complexity. The asymptotic behaviors of PS and BS are similar; however, the constant coefficients are different and this case makes PS be the best algorithm.

It is noticeable in Table III and Table IV; PS algorithm has better performance than LS and BS. Moreover, PS algorithm is better than BS algorithm as seen in Fig. 7. Searched data in PS algorithm were located to the top of Skip List, hence time complexity will be Θ(1) for these data.

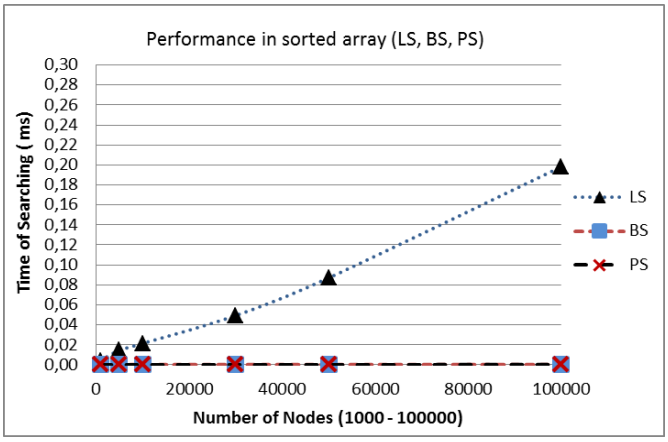

Fig. 6. Performance comparison for LS, BS, PS (If the searched data are in middle of array)

TABLE IV. Searching Times for LS, BS and PS for Sorted Arrays (if the Searched Data are Near to the End of Array) (Ms=Millisecond)

# of nodes	1000	5000	10000	30000	50000	100000
LS	0.0047 ms	0.0171 ms	0.0327 ms	0.0858 ms	0.1471 ms	0.2876 ms
BS	0.00012 ms	0.00014 ms	0.00017 ms	0.00020 ms	0.00022 ms	0.00025 ms
PS	0.00008 ms	0.00010 ms	0.00012 ms	0.00015 ms	0.00017 ms	0.00020 ms

The results in Table III were obtained when the searched data were located near to the beginning of array. Whereas, Table IV shows the situation where the searched data were located near to the end of the array. Comparing the results of LS algorithm in both tables, it was seen that the search time increases if the data were located at the end of array. However, the results were the same for BS and PS algorithms no matter where the searched data was located.

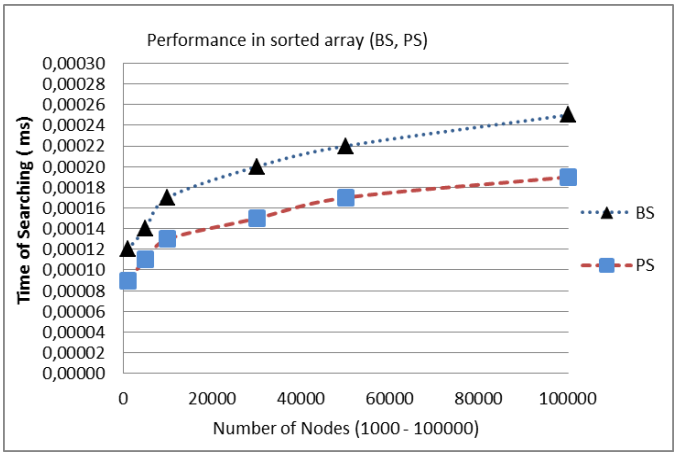

Fig. 7. Performance comparison for BS and PS (Sorted arrays)

When arrays are unsorted, the performance of linear search algorithm is better than the other algorithms, since remaining algorithms require the sorted arrays to show better performances.

A. Significance of work

Priority search algorithm locates the most searched data to the top of the pyramid-shaped skip list data structure. For these reason, enabling time complexity Θ(1) of frequent searched data were important.

The priority search algorithm may be used in the search engine like Google, Yandex, etc. The greater frequency (search hit number) the upper level for searched data; the smaller frequency the lower level for searched data. The mostly searched data were located in the top level of skip list data structure, so, searching this data will take less time. The rarely searched data were located in the lowest level of the skip list data structure, so, its searching time will take longer. If searching process was grouped with respect to frequencies of data, the searching would be easier. There many data (may be billion data, etc.) in the internet. If data were located in a large skip list data structure for search engine, it would be more advantageous.

This data structure is also advantageous for dictionary operations, since the most hit data will be on the top level of skip list data structure and its searching will take shorter time; the least hit data will be on the lowest level of the skip list data structure and its searching time will take longer time.

VI. Conclusion

Skip list data structure was created with the help of linked list data structures. Thanks to its layered structure, skip list data structure presented in this study reduces the time complexity of search, insertion and deletion processes in linked list data structure to O(lgN), which was O(N).

The applications of linear search, binary search and priority search were realized, and obtained results were compared. The obtained results verified that priority search was better than the linear search and binary search considering the applications. Priority search superior than binary searching and linear searching due to its application results. The time complexity of priority search algorithm was between Θ(1)-O(lgN); the most searched data has time complexity as Θ(1), the least searched data has time complexity as O(lgN).

To summary priority search algorithm could be used in searching processes more efficiently. It enables saving remarkable time when larger sets of data were handled.

References

[1] M. Aksu, A. Karcı, and Ş. Yılmaz, "Level optimization in Skip List data structure," in Proc. 1ST International Symposium on Innovative Technologies in Engineering and Science (ISITIES2013), 2013, pp. 389-396.

[2] M. Aksu, A. Karcı, and Ş. Yılmaz, "Effects of P Threshold Values in Creation of Random Level and to the Performance of Skip List Data Structure," BitlisEren University Journal of Science, Vol. 2, No. 2, 2013, pp. 148-153.

[3] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, Introduction to Algorithms, MIT Press, 2009.

[4] R. Colvin, L. Groves, V. Luchangco, and M. Moir, "Formal verification of a lazy concurrent list-based set," in Proc. Computer Aided Verification, Lecture Notes in Computer Science, Vol. 4144, 2006, pp. 475-488.

[5] M. Herlihy, Y. Lev, V. Luchangco, and N. Shavit, "A Simple Optimistic Skiplist Algorithm," in Proc. Structural Information and Communication Complexity, Lecture Notes in Computer Science, Vol. 4474, 2007, pp. 124-138.

[6] P. Kirschenhofer, C. Martinez, and H. Prodinger, "Analysis of an optimized search algorithm for skip lists," Theoretical Computer Science, Vol. 144, 1995, pp. 199-220.

[7] P. Kirschenhofer, and H. Prodinger,"The path length of random skip lists," ActaInformatica, Vol. 31, No. 8, 1994, pp. 775-792.

[8] W. Pugh, "Skip Lists: A Probabilistic Alternative to Balanced Trees," Communications of the ACM, Vol. 33, No. 6, 1990, pp. 668-676.

[9] T. Papadakis, J. I. Munro, and P. V. Poblete, "Average search and update costs in skip lists," BIT, Vol. 32, 1992, pp. 316-332.

[10] T. Papadakis, "Skip lists and probabilistic analysis of algorithms," PhD Thesis, University of Waterloo, Tech. Report CS-93-28, 1993.

[11] P. V. Poblete, J. I. Munro, and T. Papadakis,"The binomial transform and the analysis of skip lists," Theoretical Computer Science, Vol. 352, 2006, pp. 136-158.

[12] J. I. Munro, T. Papadakis, and P. V. Poblete, "Deterministic Skip Lists," in Proc. SODA '92 Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms, 1992, pp.367-375.

[13] W. Pugh, "A Skip List Cookbook," Dept. of Computer Science, University of Maryland, College Park, Technical report, CS–TR–2286.1, 1990.

[14] W. Pugh, "Concurrent Maintenance of Skip Lists," Dept. of Computer Science, University of Maryland, College Park, Technical report,TR–2222.1, 1989.

[15] M. T. Goodrich, and R. Tamassia, Algorithm Design and Applications, Wiley, America, 2014.

[16] M. J. Dinnen, G. Gimel'farb, and M. C. Wilson, Introduction to Algorithms, Data Structures and Formal Languages, Pearson Education, Second edition, 2009.

[17] R. Sedgewick, and K. Wayne, Algorithms, Addison Wesley, Fourth Edition, America, 2011.

[18] K. Mehlhorn, and P. Sanders, Algorithms and Data Structures; The Basic Toolbox, Springer, 2007.

[19] M. McMillan, Data Structures and Algorithms Using C#, Cambridge University Press, 2007.

[20] C. A. Shaffer, Data Structures & Algorithm Analysis in C++, Dover Publications, 2011.

[21] D. E. Knuth, The Art of Computer Programming—Sorting and Searching, Volume 3, Addison Wesley, Second edition, 1998.

[22] T. Clouser, M. Nesterenko, and C. Scheideler, "Tiara: A self-stabilizing deterministic skip list and skip graph," Theoretical Computer Science, Vol. 428, 2012, pp. 18-35

[23] R. M. Nor, M. Nesterenko, and C. Scheideler, "Corona: A Stabilizing Deterministic Message-Passing Skip List," Theoretical Computer Science, Vol. 512, 2013, pp. 119-129.

[24] P. Bose, K. Douïeb, and P. Morin, "Skip lift: A probabilistic alternative to red–black trees," Journal of Discrete Algorithms, Vol. 14, 2012, pp. 13–20.