Skip List: Implementation, Optimization and Web Search

Godwin Adu-Boateng *gaduboateng@umc.edu*
Center of Biostatistics and Bioinformatics
Department of Medicine
University of Mississippi Medical Center
Jackson, MS 39216, USA

Matthew N. Anyanwu *matany58@yahoo.com*
Department of Computer Science
University of Memphis
Memphis, TN 38152, USA

Abstract

Even as computer processing speeds have become faster and the size of memory has also increased over the years, the need for elegant algorithms (programs that accomplish such tasks/operations as information retrieval, and manipulation as efficiently as possible) remain as important now as it did in the past. It is even more so as more complex problems come to the fore. Skip List is a probabilistic data structure with algorithms to efficiently accomplish such operations as search, insert and delete. In this paper, we present the results of implementing the Skip List data structure. The paper also addresses current Web search strategies and algorithms and how the application of Skip List implementation techniques and extensions can bring about optimal search query results.

Keywords: Skip List, Efficient Algorithms, Web Search.

1. BACKGROUND & INTRODUCTION

One of the fundamentals in the study of computer science is the representation of stored information. Even as computer hardware has become more sophisticated, and the size of memory has increased over the decades and sustains an upward trajectory, the resources required for computer programs (memory space and time) to execute implemented programs continue to be in the fore-front. Programs that accomplish such tasks/operations as information retrieval, and manipulation as efficiently as possible have garnered considerable attention from computer scientists [1]. In Galil and Italiano [2], the authors' survey data structures and algorithms proposed as a solution for a set union problem. Weiss [3] analyzes a disjoint set problem and provides a real world scenario of its application in a computer network. Both Galil and Italiano, and Weiss investigate the memory space and time complexity of proposed algorithmic solutions. Collins [4] also discusses it with the Java programming language as a back drop. Literature exists that approach this fundamental problem in other ways. Review [5, 6, 7] for further discussion on algorithm analysis and efficiency. To achieve optimal efficiency in data manipulation, the type of data structure used plays an important role. Shaffer [1] defines a data structure as "…any data representation and its associated operations." He goes further to state that "...a data structure is meant to be an organization or structuring for a collection of data items." A scheme of organizing related data items is how Levitin [7] defines a data structure. In essence, a data structure is any organization of related data items and its associated operations. The data is organized such that it can be used optimally. Hence, data structures are an integral part of algorithm execution, and it can determine how efficient or otherwise an algorithm is.

Today the World Wide Web or simply the Web or its contemporary name: Social web has become part of everyday life. Each day millions of people interact with the Web in several ways, for instance, sending and receiving e-mails, interacting with friends and colleagues, reviewing and

rating visited restaurants, searching for travel deals and merchandise. It is anticipated that this trend would continue [8]. Searching the internet via search engines has become quite poplar as it usually serves as an introductory point to internet usage. In fact reports show that *84%* of internet users have used search engines [9]. Many internet search engines exists today for example, Yahoo, Bing, Google and many other more targeted ones, like Amazon and EBay, among others. Each has its own structure in terms of query execution, information gathering/corpus construction, and display/presentation of the relevant pages/retrieved corpus. However, the baseline goal of retrieving relevant information remains consistent in all search engines.

As information on the Web is forecasts to increase and more especially as big data comes to the forefront, it is increasingly going to be essential for search engines to not only retrieve relevant documents and Web pages, but also, be optimal in ad hoc query execution. For decades now, Text REtrieval Conference (TREC) has remained the benchmark for Information Retrieval (IR). The main focus of this benchmark is to test for effectiveness (accuracy) in the resulting corpus of a search query, and not efficiency [10]. A review of the literature shows that to be the case [11]. Cao et al. [12] also examine the retrieval of handwritten document images.

As earlier stated the quest for optimality is an ongoing and important one. It is even more important when considered in the context of the ever increasing complexity of problems that require computer solutions. The emergence of big data: A contemporary problem/opportunity presents varying degrees of complexity [13]. It also presents challenges in the search and retrieval of data, and data analytics, among others. Reports suggest that Web content continues to increase as Web usage increases [13, 14] and more users across all age grades and nations become conversant with using it. Therefore, an efficient way for search and retrieval in the face of this continuous increase in Web data/information is in order.

Skip List (SL) is a data structure with properties that appropriately satisfies the computer science fundamental of stored information representation. An experiment conducted clearly shows the desirability of using SL as the preferred structure for data processing. It can also play an important role in Web search as it also provides a data organization scheme that yields better efficiency in operations such as search [15].

2. SKIP LIST OVERVIEW & OPERATIONS

Skip lists (SL) was invented by William Pugh in 1989 and proposed by him in 1990. SL was proposed as an alternative to the Binary Search Trees (BST) and other balanced trees. It is a probabilistic data structure based on parallel linked list where elements are kept by key. But, as opposed to linked list, a node can contain more than 1 pointer. The number of pointers in a node determines its level. Each pointer points to the next node in the list that has an equal or greater level. If there isn't any forward node having this property, the pointer points to NIL [16]. The level of a Skip list corresponds to the highest level between the nodes. When a node is inserted to the list, a node is given a random level [16]. Figure 1 provides a pictorial depiction of a SL.

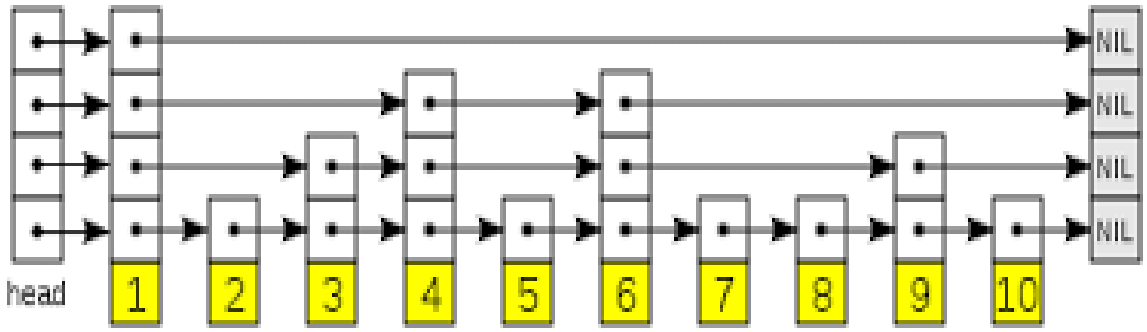

FIGURE 1: Skip List. Perfect Skip List.

The main motivation for using SL include among others is that it ensures a more natural representation than a tree, it is simple to implement, it has both practical and theoretical applications and its efficiency is comparable to a balanced tree where the expected time for an operation (such as searching for an element) is $O(\log n)$[1], where n is the node/random level [1, 16].

SL can be used as a replacement for other data structures. It also has other operational efficiencies as noted by Pugh [16];

- It is relatively easy to design and implement the algorithm
- It is efficient
- The randomization is not dependent on the input keys, thus it will be difficult for an advisory to access using the input keys
- It serves as an alternative to Binary search trees which become unbalanced after several insertion and deletion operations
- It requires a random number generator at the time of insertion
- The burden on memory management is due to varying size nodes

2.1 Skip List Operations

Like any data structure, SL supports such operations as *search*, *insert* and *delete*.
The search for an element starts at the header on the highest level, then traverses forward pointers that don't overshoot the element being searched for. If that happens, move down to the next level. A brief pseudo code description of the search algorithm is shown in Figure 2.

This algorithm is very simple to implement, it searches for the right place for an element to be inserted. Splice the list and insert the element with a random level.

Deleting/removing an element is also like inserting an element, it searches for the right place for an element to be deleted. Splice the list and delete the element with a given random level [17].

```
1. x <- list.header
2. for i <- list.level downto 1
3.     do    while x.forward[i].key < searchKey
4.               do    x <- x.forward[i]
5. x <- x.forward[1]
6. if x.key = searchKey
7.     then return x.value
8.     else return failure
```

FIGURE 2: Skip List Search Algorithm.

2.2 Practical Application of Skip List

A cursory search for practical application of the SL algorithm yielded varying levels of its application in the real world. Some of the practical applications are as follows:

- Parallel Computing: Skip List can be applied to parallel computation where insertions and searching of items can be accomplished in different parts of the list and in parallel without rebalancing of the data structure [18].

[1] This is a notation for identifying the order of growth of an algorithm's basic operation. In this particular case $O(\log n)$ represents the order of growth of SL search algorithm.

- Operating Systems: A common problem in computer science is the blocking of resources for processors to complete tasks. This can lead to race conditions. Skip list can be applied to this problem with a lock-free data structure which is more efficient and it is easy to implement [19].
- Forecasting/Prediction: Ge and Stan [20] used it for predicting queries by employing Pre-built Models (PM). The 'closest' PM is selected for use in the forecasting. Predicting can be applied to scheduling, resource acquisition, and resource requirement determination.
- Graphics and Visualization: Skip list-like data structure has been used to optimize the rendering of graphics and images [21].

It is important to state here that the list provided is by no means an exhaustive one as that would be beyond the scope of this paper.

3. EXPERIMENT

An experiment was conducted to determine the cost of searching for an element at a given random level.

The experiments were conducted on the algorithm using the java program on different values for p = 1/2, 1/4 and 1/8.

For each value of p, 1000 lists were randomly created for different sizes of n: 10, 100, 1000 and 10000.

For each case, the average number of steps to find 1000 random elements in the list was found. The average is then compared to $\log(1/p(n))$ and to $L(n)/p + 1/(1-p)$. Then the maximum number of steps recorded between 1000 searches was added. The result is shown in the following tables with the different p values and different n sizes.

3.1 Results

The results shown in the proceeding section illustrates well enough the analysis of the expected search cost.

In general, for the 3 different values of p, the average number of steps grows logarithmically to the number of elements n. In fact, for p = ½, the average is very close to log (1/2(n)). On average, the algorithm performs better for p = ½. But, as demonstrated in [17],
"…choosing p = ¼ slightly improves the constant of factors of the speed of the algorithm." This means that the algorithm is more consistent, and there is a higher probability to obtain a search cost close to the average when choosing ¼.

Thus, the maximum number of steps is relatively low for high instances of n. For p = ½ (see Table 1), it is only 38 steps for a list of size 10000. (38 is only the maximum number of steps in the experiment that we did, and it is possible to obtain a higher maximum). This implies that it is very unlikely to obtain a list largely unbalanced that will give us an extremely poor performance.

n	Average Number of Steps	Log (1/p(n))	L(n)/p + 1/(1-p)	Maximum Number of Steps
10	3.45	3.32	8.64	10
100	6.56	6.64	15.2	26
1000	9.19	9.96	21.9	35
10000	9.92	13.29	28.5	38

TABLE 1: Average number for steps for finding 1000 random list elements. P=1/2.

n	Average Number of Steps	Log (1/*p*(*n*))	L(*n*)/*p* + 1/(1-*p*)	Maximum Number of Steps
10	4.01	1.667	7.9	10
100	8.56	3.32	14.6	45
1000	12.57	4.98	21.9	59
10000	13.63	6.64	27.9	60

TABLE 2: Average number for steps for finding 1000 random list elements. *P*=1/4.

n	Average Number of Steps	Log (1/*p*(*n*))	L(*n*)/*p* + 1/(1-*p*)	Maximum Number of Steps
10	4.67	1.12	10	10
100	11.43	2.21	18.8	67
1000	17.86	3.32	27.7	112
10000	19.31	14.43	36.6	116

TABLE 3: Average number for steps for finding 1000 random list elements. *P*=1/8.

4. WEB SEARCH: KNOWN & UNKNOWN

Not all search engines are created equal and today an online search involves Web ‘crawlers’, ‘spiders’ or ‘robots’. These are algorithms that innocuously navigate web pages and links to gather information. They undertake the processes that make up the dimension of a web IR, and they can provide different results set [11].

Although algorithms for web searches are mostly proprietary, research shows they rely on inverted indexing in the processing of search requests and corpus construction. Many search algorithms and strategies use this approach as the basis of conducting a document search [22, 23]. Inverted indexing creates a list of index terms along with a logical linked list referred to by Grossman [10] as a *posting list*. Each linked list pointer references a unique term in the index, which is created before query execution.

Again, current web search results are represented in some form or fashion. The documents relevance to the query, the number of times the search term is found in the document, or the interval between search terms in the document are some of the representation criteria discussed by Li et al. [22].

As previously stated the benchmark for IR is on the effectiveness and not efficiency. Hence, there is little to no mention of the optimality of search engine algorithms. It is our belief that an efficient data structure for the inverted index can improve performance.

5. CONCLUSION & FUTURE WORK

SL has an expected time of $O(\log n)$ for an operation such as search or insertion. Based on the experiments that we did in this project, we can conclude that SL is a good alternative to balanced tree. Taking the same argument from the author, SL is very easy to implement which is the biggest advantage over balanced trees.

Also, SL can be applied to inverted indexing; which features prominently in web searches. Research has shown that SL, when applied to web search offers improved efficiency. Boldi and Vigna [24] discuss embedding SL in inverted index. Inverted index search improves the efficiency of SL by reducing the space requirement of web searches. The reduction in the space

requirement further improves the expected time of search of SL. Also inverted index ensures that all occurrences of a query is retrieved. Chierichetti et al. [25] report on using different ways of determining the precise locations of skips to be placed in an inverted index with an embedded SL. They report the time complexity on an optimal algorithm as being linearithmic. Also, Campinas et al. [26] extend the basic Skip list for a block-based inverted list and found a lower search cost.

There appears to be promise in the continued application of skip list techniques and extensions as it pertains to search engine optimization. This is a step in the right direction when considered in the context of current trends in web searches and document retrieval, where the amount of documents on the web is estimated to be in the billions. Google, one the most used search engines has over 2 billion indexed documents. The current estimate of online documents is over 500 billion today. Li et al. [22] report on the feasibility of a peer-to-peer web search. Future work will focus on determining SL performance for TREC as well for efficiency.

6. ACKNOWLEDGEMENT

We are grateful to Serge Salan from University of Memphis TN for his contribution in performing the Skip-list experiment.

7. REFERENCES

[1] C.A. Shaffer. *A Practical Introduction to Data Structures and Algorithm Analysis – Java Edition.* Prentice-Hall, Inc., pp. 365-371, 1998.

[2] Z.Galil and G.F. Italiano. Data Structures and Algorithms for Disjoint Set Union Problems. ACM Computing Surveys vol. 23, pp. 319-344.

[3] M.A. Weiss, *Data Structures and Algorithm Analysis in C++,* Benjamin/Cummings, Redwood City, Calif., Chap 8, pp. 287, 1994.

[4] W.J. Collins. *Data Structures and the Java Collections Framework*. McGraw Hill. pp. 389 – 432, 2002.

[5] A.V. Aho, J.E. Hopcroft, J.Ullman. *Data Structures and Algorithms*. Addison-Wesley Longman Publishing Co., 1983.

[6] T. Kanungo, D.M. Mount, N.S. Netanyahu, C. Piatko, R. Silverman, A.Y. Wu. “An Efficient *k*-Means Clustering Algorithm: Analysis and Implementation.” IEEE Trans. Patt. Anal. Mach. Intell, v. 24, no.7, pp. 881-892, 2002.

[7] A. Levintin. *Introduction to the Design & Analysis of Algorithms*. Addison-Wesley, 2006.

[8] J. Porter. *Designing for the social web*. Peachpit Press, 2010.

[9] H.R. Varian. *The economics of Internet search*. University of California at Berkeley, 2006.

[10] D.A. Grossman. *Information retrieval: Algorithms and heuristics*. Vol. 15. Springer, 2004.

[11] M. Gordon and P. Pathak. “Finding information on the World Wide Web: The retrieval effectiveness of search engines.” Information Processing and Management*,* 35(2), pp. 141–180, 1999.

[12] H. Cao, A.Bhardwaj and V. Govindaraju. “A probabilistic method for keyword retrieval in handwritten document images.” Pattern Recognition, 42(12), pp. 3374-3382, 2009.

[13] S. Lohr. (2012). “The Age of Big Data.” The New York Times. Retrieved Jan 2013, available online: http://www.nytimes.com/2012/02/12/sunday-review/big-datas-impact-in-the-world.html?_r=2&scp=1&sq=Big%20Data&st=cse&.

[14] A. McAfee and E. Brynjolfsson. “Big data: the management revolution.” Harvard Business Review, 90(10), pp. 60-66, 2012.

[15] R. Delbru, S. Campinas and G. Tummarello. Searching web data: An entity retrieval and high-performance indexing model. *Web Semantics: Science, Services and Agents on the World Wide Web*, *10*, pp. 33-58.

[16] W. Pugh. “Skip Lists: A Probabilistic alternative to balanced trees.” Communications of the ACM, v. 33. pp. 668-676, 1990.

[17] W. Pugh. “A skip list cookbook.” University of Maryland, Department of Computer Science Report CS-TR-2286.1, 1989.

[18] J. Gabarro, C. Martinez and X. Messeguer. “A design of a parallel dictionary using skip lists.” Theoretical Computer Science 158, pp. 1-33, 1996.

[19] F. Keir. *Practical lock-freedom*. Diss. University of Cambridge, 2004.

[20] T. Ge and S. Zdonik. “A Skip-list approach for efficiently processing forecasting queries.” Proceedings of the VLDB Endowment 1.1, pp. 984-995, 2008.

[21] J. El-Sana, E. Azanli, and A. Varshney. Skip Strips: Maintaining Triangle Strips for View-Dependent Rendering. In IEEE Visualization ’99 Proceedings, pp. 131–138, 1999.

[22] J. Li, B.T. Loo, J. M. Hellerstein, M. F. Kaashoek, D. R. Karger, and R. Morris. “On the feasibility of peer-to-peer web indexing and search.” In Peer-to-Peer Systems II. Springer Berlin Heidelberg. pp. 207-215, 2003.

[23] L.A. Barroso, J. Dean, and U. Holzle. "Web search for a planet: The Google cluster architecture." Micro, IEEE 23.2, pp. 22-28, 2003.

[24] P. Boldi and V. Sebastiano. "Compressed perfect embedded skip lists for quick inverted-index lookups." String Processing and Information Retrieval. Springer Berlin Heidelberg, 2005.

[25] F. Chierichetti, R. Kumar and P. Raghavan. "Compressed web indexes." Proceedings of the 18th international conference on World Wide Web. ACM, 2009.

[26] S. Campinas, R. Delbru, and G. Tummarello. "SkipBlock: self-indexing for block-based inverted list." Advances in Information Retrieval. Springer Berlin Heidelberg, pp. 555-561, 2011.