JOURNAL OF CRITICAL REVIEWS

ISSN- 2394-5125 VOL 07, ISSUE 15, 2020

PERFORMANCE EVALUATION OF THE ENHANCED HASH ALGORITHM USING VECTOR AND HASH FUNCTION

Bobby Eclarin and Arnel Fajardo
[1]Mariano Marcos State University,
City of Batac,Ilocos Norte, 2906, Philippines.

[2]School of Engineering and Information Technology,
Manuel L. Quezon University, Manila, Philippines.

Corresponding Author's Email:[1]bob.eclarin@gmail.com

Article History: Received xxxxx; Revised xxxx; Accepted xxxx

ABSTRACT: Hashing is well-known due to its usefulness and rapid access to data in many Information Technology areas such as in networking, Natural Language Processing, Information Retrieval and Text Mining. The hash function distributes the items in the hash table. However, collision occurs when two or more items have the same hash values. The collision resolution will resolve such problem. The algorithm was enhanced by modifying the lookup procedure using Vector data structure and hash functionto improve the time spent during lookup while resolving the collision. The objectives are to determine the success rate of the modified hashing algorithm in resolving hash collisions; to evaluate and compare the performance of the modified hashing Algorithm with the original Linked List chained algorithm using the same string datasets in terms of Lookup time, Work Efficiency; and Number of memory accesses in the search procedure, as well as its Time Complexity. Results showed that the enhanced hashing algorithm has successfully resolved collision on any table and data sizes. The Big O notation analysis showed that the time complexity of the search operation is O(1). It also guarantees that every search operation can achieve 1 or 2 memory accesses on different data and table sizes.

KEYWORDS:*Hash Algorithm, Vector, Linked List, Collision Resolution, Hash Table*

1.0 INTRODUCTION

ThisHashing is one of the most essential procedures that offer means and ways for a program to rapidly gain access to data [1]. Hashing gained its popularity due to its usefulness in many Information Technology areas [2]. These are in the areas of networking such as route lookup, packet organization, per-flow state administration, load balancing and network checking for its constant access time [3-5]. It is also being utilized in cryptography development and string (data) indexing for fast searching [2, 6-11]. Currently, hashing is also essential in Text Mining, where vast unorganized

ISSN- 2394-5125 VOL 07, ISSUE 15, 2020

text from any sources, such as social networks, medical records and news, as well as in Data Mining [12]. Textual data can be analyzed that will result into valuable information [13-14]. It also supports in text processing[15] for Natural Language Processing [16], such as Bag-Of-Words representation of text information [14, 17-19]. Hashing also includes topics in vision data applications due to its quick search on similar objects and for its efficient and fast indexing[20-21].

Generally, the hashing algorithm has two parts: the Hash Function and its Collision Resolution [22-23]. Basically, a Hash Function converts string into a shorter fixed-length value or key and uses this search and find the original value from a hash table [2,8,24]. Hash table is a complex data structure, also known as an associative array that maps keys to hash values as the index to which data will be placed using a hash function [8,22,25]. A Hash function that provides uniform allocation of elements in the hash table is considered a good hash function [8], which in turn, by general assumption, delivers a time complexity O(1) and a near-single memory access [5]. However, as the data fills up the hash table, collision may occur [3] even with a well-defined hash function. Collision happens when two or more resulting hashed keys derived by the hash function are equal [11,23]. Collision resolution techniques are used to resolve such problems and guarantee future key look-up procedures [18] to return the correct data [2]. Open addressing collision resolution technique uses probing to search the next empty slot in the hash table that will accommodate the other data [5]. Open Addressing uses either or Linear probing, Quadratic probing and Double hashing methods to determine the number of steps in looking for an empty space [2][23]. On the other hand, Separate Chaining collision resolution uses Linked List data structure to dynamically allot space for information with similar hashed key [5].

However, the use of any collision resolution will result into numerous key comparisons and memory reads, causing the time to increase linearly [5]. Thus, the lookup latency turns out to be undetermined, which poses unpredictable output to some time-critical systems [4]. When using a linked list chained collision resolution, pointers must be sustained in every bucket to provide associations with other links, which will make memory overhead in the form of added space for recording and getting the pointers [24]. The search time depends on the position of the data being searched for and the number of data present in the list. Finding a data will traverse the list from the beginning until such is found or the end of the list is reached; thus, a linear search [26-27], resulting in multiple memory access, which causes time delays.The work of [28] that uses a 2-dimensional vector that represents a hash table and the separate chain uses another hash function that will search for data in the chain.

The course of direction of this study is to evaluate the enhanced hashing algorithm using Vector data structure and a hash function on its Separate Chaining collision resolution to improve the time spent in the lookup process while resolving the collision problem[28]. Both the modified algorithm and the original linked list chained algorithm will be evaluated and compared using the same datasets across a different number of data and table sizes.

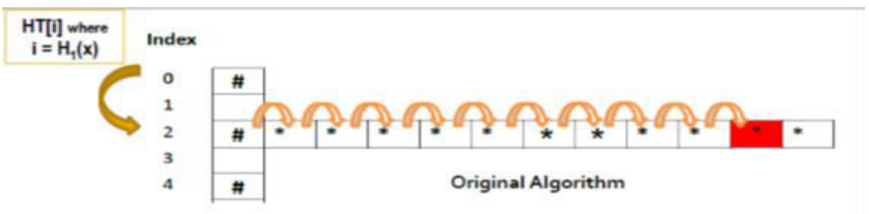
ISSN- 2394-5125 VOL 07, ISSUE 15, 2020

2.0 THE MODIFIED ALGORITHM

2.1 Separate Chaining and Linked List

The original algorithm linked list data structure to dynamically allocate space for data with the same hash key [5]. An array allocates space for all its elements as one chunk as they are in contiguous memory. On the other hand, a linked list allots space for each data separately using pointers to connect all its nodes together and each node has two fields; the one that contains the data and a pointer space to the next node [29]. So unlike arrays, a linked list doesn't have a declared fix size. The location of its elements can be anywhere in the memory and a Linked list can be dynamically shrink and grown to any extent as depending on the memory size and addressability of the system[28].

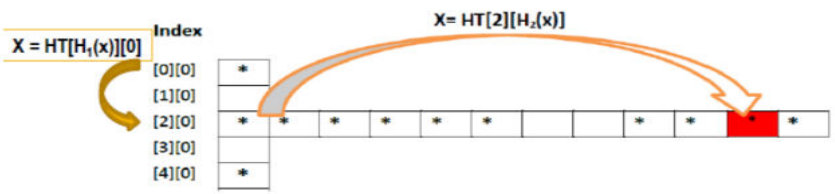
Figure1:Steps of the lookup procedure in the original algorithm [28].



The lookup procedure will traverse the linked lists starting from the index pointed to by the hash function H1, until the item looked for is found or the end of the list is reached.

2.2 The Modified Algorithm Using Vector and a Hash Function in the Chain

Vectors are dynamic array data structure which stays in contiguous memory [26]; therefore, providing a faster access time [30]. The contents of a vector can read/write into randomly through its index [30]; so a hash function can be utilized to get the key using the value of the information that will point to its memory location. Another distinguishing attribute of a vector is that it can be grown and shrink dynamically.



Reference [28] took advantage of these characteristics to enhance the hash algorithm.

Figure2:Steps in the lookup procedure in the modified algorithm [28].

ISSN- 2394-5125 VOL 07, ISSUE 15, 2020

Fig. 2 indicates how the enhanced hash algorithm searched for the item. It uses the first hash function to locate the item, first in the HT, by using the first function; item X = HT[h1(x)][0]. If the item being searched for was not in the HT, the next step was to look for it in the chain; item X = HT[h1(x)][h2(x)].

3.0 PROCEDURES AND EVALUATION

The two program modules which were used to simulate the enhanced and original algorithm were developed using the C++ STL (Standard Template Library). A computer with a Quad-Core Intel Celeron, 4G RAM and a 500G hard disk was used in the evaluation.

To test the effectiveness of the algorithm in resolving collision, the simulation program utilized datasets (English words and passwords) of different sizes, 50, 1000 and 3000. Three different table sizes were selected for each of the datasets. The simulation program counted the number of Collision (C) and the number of Resolved Collisions (RC) per table size per dataset. The number of items outside the hash table was considered the collided items. The success rate was computed using the following formula:

Success Rate (%) = (# of RC / # of C) x 100 (1)

The computed Success Rate indicated the effectiveness of the algorithm on different data and tables sizes. The items and their respective indices were also shown. If all collisions were successfully resolved, no items would be in the same index. The Runtime Speed for the Searching operation of the two algorithms was compared. Using the dataset with 3000 strings, the simulation program would search items while recording the time spent on each procedure. The average was also computed and recorded. In this research, the same procedure was executed 10X for each of the algorithms. TNEW is the time spent by the enhanced algorithm and TOLD is the time spent by the original algorithm during execution. The Percentage of Change (PC) was computed to determine if there was an improvement in the execution time for each procedure. The Percentage of Change is computed as follows:

PC (%) = ((TNEW – TOLD)/ TOLD) x 100 (2)

A negative result meant that the execution time was short and a positive result suggests that an increase in execution time was recorded. The number of memory accesses during Lookup was also recorded. In the algorithm using the 2D vector, the number of memory access is 1 if the item was found at the HT, and 2 if found in the SC.Using the Big-O notation, the Time Complexity of the algorithm for the Lookup procedure was determined.

4.0 RESULTS AND DISCUSSION

4.1 Collision Resolution Evaluation

The success rate in resolving collision was measured to be 100 percent on different

ISSN- 2394-5125 VOL 07, ISSUE 15, 2020

data and table sizes. For the n=50, all success rates were computed as 100%, where table size is equal to 71, 47 and 31 as shown in Table 1. Table 2 shows the same result with data size n=1000 and table sizes are equal to 1741, 719 and 33; it is recorded that all collisions were all resolved.Also, in Table 3 with n=3000, where table sizes are 5179, 3181 and 797; also, all collisions were resolved.

Table1. Data size N = 50.

Table Size	#Collision	#Resolved	Success Rate(%)
71	13	13	100
47	19	19	100
31	23	23	100

Table 2. Data sizeN = 1000

Table Size	#Collision	#Resolved	Success Rate(%)
1741	227	227	100
719	466	466	100
331	680	680	100

Table 3. Data sizeN = 3000.

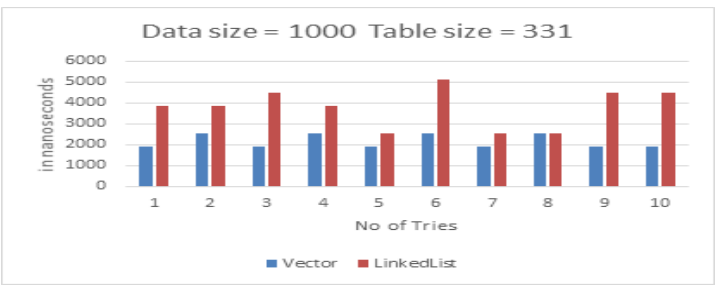
Table Size	#Collision	#Resolved	Success Rate(%)
5179	727	727	100
3181	1071	1071	100
797	2213	2213	100

Table 1 shows that there are 13, 19 and 23 collisions with table sizes 71, 47 and 31, respectively. On the other hand, Table 2 demonstrates that there are 227, 266 and 680 collisions with table sizes 1741, 719 and 331, respectively. Finally, in Table 3, it shows that there are 727, 1071 and 2213 collisions with table sizes 5179, 3181 and 791, respectively. All tables show a 100 percent success rate.

During the insertion procedure, rehashes occurred, which meant that there were collisions at a particular chain. Although the chain was much smaller than the hash table and much less expensive to do a rehash, it would still take additional time to insert the items. This might have attributed to the hash function and chain size.

4.2 Lookup Time Performance Evaluation

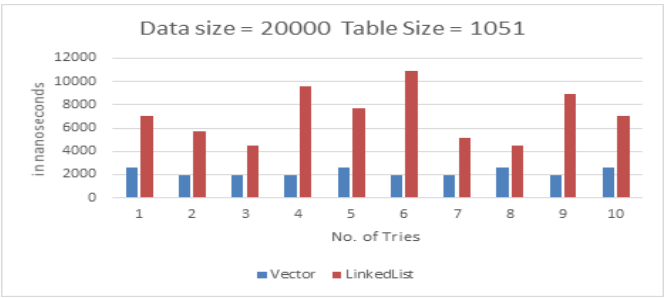
The comparison in terms of lookup speed was reflected and presented in Figure 3, Figure 4 and Figure 5. Three sets of tables and data sizes were used in the simulation;



ISSN- 2394-5125 VOL 07, ISSUE 15, 2020

specifically, n = 1000, n = 3000 and n = 20000 with table sizes 331, 797, 1051 respectively for each data size. As shown, in the figures, the enhanced algorithm using vector has an average speed of 2176 ns (nanoseconds) to search for an item in the first set where n = 1000, for the second set it has an average of 2112.10 ns where n = 3000 and 2176.1 where n = 20000.

Figure 3. Lookup time where table size =331 and n = 1000



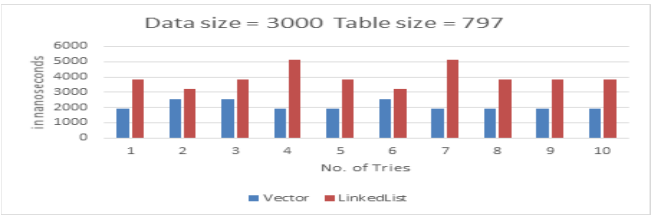


Figure 4. Lookup time where table size = 797 and n = 3000.

Figure 5. Lookup time where table size = 1051 and n = 20000.

The original algorithm using a linked list, on the other hand, takes an average of 3776.1 ns to process the same task for n = 1000 in the first set, 3968 ns where n = 3000 in the second set, and 7104.40 ns for n = 20000. All tables and data sizes were executed in 10 trials each.

TABLE 4. Summary of the Time Performance of the Lookup Procedure.

Table and Data Size	**Vector**	**Linked List**	**Difference (%)**
N = 1000 TS = 331	2176.0	3776.1	**-63.97**
N = 3000 TS = 797	2112.1	3968.0	**-46.77**
N = 20000 TS = 1051	2176.1	7104.4	**-42.37**

The summary Table 4 manifested that the enhanced algorithm using vector performs faster than the algorithm using a linked list in terms of lookup speed to which it

ISSN- 2394-5125 VOL 07, ISSUE 15, 2020

yielded a reduced time in the said operation. The modified algorithm was able to lessen the lookup time of the original hashing algorithm by –42.37% in the first set where the table size is 331 and n = 1000, -46.77% in the second set where table size is equal to 797 and n = 3000, and -69.37% in the third set where table size is 1051 and n = 20000. The negative values of the computed Percentage of Change (%) meant that it used lesser lookup time over the original algorithm. The summary also indicated that the lookup time barely changed on all tables and data sizes. The lookup time was not affected by the sizes of the table and data.

4.3 Number of Memory Accesses

Table 5 shows the number of memory accesses and time spent in searching for an item. The simulation made use of a data set with n = 20000 and table size is 1051. The ten (10) items listed in the table were the subject for searching and each item was searched one at a time. Index 0 (i = x, j = 0) in the memory locations indicates that the item is in the HT (first dimension of the vector). Indexes starting from 1 are the separate chains, where 1 is the memory location nearest to the HT. The recorded time for the enhanced algorithm using vector was almost the same, with most of the items searched at 1920 ns, while each memory access column shows either 1 or 2 values. When an item is found at the 0 index, it means that only the first hash function was used. If the j index is 1 or more, then both of the hash functions were used. On the other hand, the algorithm using a linked list has a greater number of memory accesses. This is due to the linear search starting from the head of the list.

Table 5. NUMBER OF MEMORY ACCESS AND LOOKUP TIME.

Tries	**Vector**			**LinkedList**		
	Time	***#MA***	***ML***	***Time***	***#MA***	***ML***
1	1920	1	746,0	2561	1	746,0
2	1920	2	741,181	15361	32	741, 31
3	1921	2	818,72	12800	29	818,28
4	2560	2	582,150	7681	18	582,17
5	1920	2	357, 51	5760	11	357,10
6	2560	2	348,271	8321	19	348,18
7	2560	2	345,125	7041	11	345,10
8	1920	2	536,4	3840	2	536,1
9	1920	2	541,99	3200	2	541,1
10	1920	2	754,0	2560	3	754,0

ISSN- 2394-5125 VOL 07, ISSUE 15, 2020

As seen in Fig. 6 and Fig. 7, the number of memory access is directly proportional to the time spent in searching. The graph for the enhanced algorithm barely changed or at most constant. The search procedure of the enhanced algorithm is guaranteed to be either 1 or 2 memory accesses, no matter how long the chain is or where the item in the chain is.

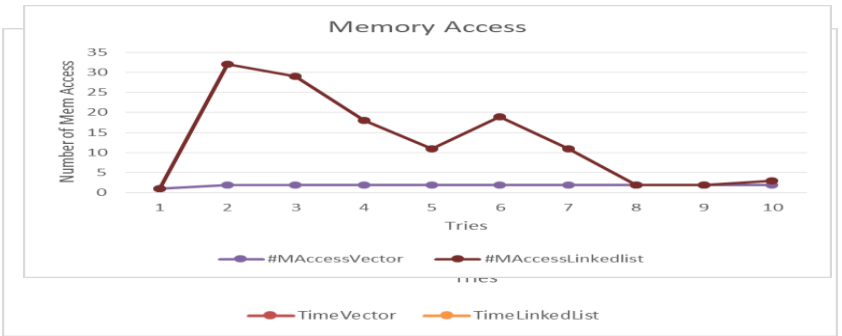


Figure 6. Time spent in the lookup procedure for each item.

Figure 7. Number of memory access in the lookup procedure for each item.

As seen in Fig. 6 and Fig. 7, the number of memory access is directly proportional to the time spent in searching. The graph for the enhanced algorithm barely changed or at most constant. The search procedure of the enhanced algorithm is guaranteed to be either 1 or 2 memory accesses, no matter how long the chain is or where the item in the chain is.

4.4 Time Complexity

The time complexity of the Lookup procedure provides a constant time or O(1) based on the pseudo-code analysis of the program, specifically the function used in the lookup procedure. The table shows the code snippet of the find_word() function. The result indicates that the algorithm used the same amount of time and was not influenced by the size of the input [32] as shown in Fig. 6 and Fig. 7 in the previous section.

Table 6. Time Complexity of the Lookup procedure.

Code	**Time Complexity**		**BigO Notation**
unsigned int key1 = Hash::func1(word);	C1	1 + 1	O(1)
if(word != WordTable[key1].second[0].word)	C2	1 + 1	

ISSN- 2394-5125 VOL 07, ISSUE 15, 2020

{return WordTable[key1].second[Hash::func2(word, Hash::chain_sizes[WordTable[key1].first])];	C3	1 + 1	
else	C4	1	
return WordTable[key1].second[0];	C5	1	

As shown in Table 6, the time complexity of the hash functions func1() and func2() were assumed to be O(1) since the size m(number of character) of the input key is much less than the number of items n in the whole hash table, so the calculation of the hash does not depend on n [31]. If the data looked for is at the HT, the computed Tsum = C1 + C4 + C5 = (1+1) + 1 + 1 = 4; therefore the Big O Notation is O(1). If the data is in the SC, Tsum = C1 + C2 + C3 = (1+1) + (1+1) + (1 + 1) = 6; therefore, the Big O Notation is also O(1). Either way, whether the item is on the HT or SC, the Time Complexity is O(1) on its average and worst case.

5.0 CONCLUSION

Using vector on its separate chaining yielded a 100% success rate in resolving collision across different data and table sizes. The enhanced algorithm using vector performs faster than the algorithm using a linked list in terms of lookup speed to which it yielded a reduced time in the said operation. The result demonstrate –42.37%, -46.77% and -69.37% Percentage Difference for each set. The search procedure of the enhanced algorithm is guaranteed to be either 1 or 2 memory accesses, no matter how long the chain is or where the item in the chain is. The number of memory access is directly proportional to the time spent in searching. The time complexity of the search operation provides a constant value or O(1), specifically the function used in the lookup procedure.

6.0 FUTURE WORKS

1.] Since the enhanced hashing algorithm uses another hash function in its chain, further study is recommended to find out a better combination of methodologies for the hash functions to lessen collision in the chain during insertion. 2.] The chain grows only when there is a collision in the HT, and it grows following a sequence of primary numbers, further study is recommended to derive the sequence of prime numbers for better optimization.

ACKNOWLEDGMENT

The author would like to express his heartfelt gratitude to the Mariano Marcos State University for the continuing support to faculty development and research endeavors, in the quest for academic excellence and global competitiveness. MMSU ACHIEVE!

ISSN- 2394-5125 VOL 07, ISSUE 15, 2020

REFERENCES

[1] F. Khorasani, M. E. Belviranli, R. Gupta, and L. N. Bhuyan, "Stadium Hashing: Scalable and Flexible Hashing on GPUs," in Parallel Architectures and Compilation Techniques - Conference Proceedings, PACT, 2016, vol. 2016-March, pp. 63–74.

[2] M. Singh and D. Garg, "Choosing Best Hashing Strategies and Hash Functions," in 2009 IEEE International Advance Computing Conference, IACC 2009, 2009, pp. 50–55.

[3] G. He, Y. Du, and D. Yu, "Efficient Hashing technique based on bloom filter for High-Speed Network," in Proceedings - 2016 8th International Conference on Intelligent Human-Machine Systems and Cybernetics, IHMSC 2016, 2016, vol. 1, pp. 58–63.

[4] D. Li, J. Li, and Z. Du, "Deterministic and efficient hash table lookup using discriminated vectors," in 2016 IEEE Global Communications Conference, GLOBECOM 2016 - Proceedings, 2016, pp. 0–5.

[5] T. Song, Y. Yang, and P. Crowley, "RwHash: Rewritable Hash Table for Fast Network Processing with Dynamic Membership Updates," 2017 ACM/IEEE Symp. Archit. Netw. Commun. Syst., pp. 142–152, 2017.

[6] M. Cantu, J. Kim, and X. Zhang, "Finding hash collisions using MPI on HPC clusters," in 2017 IEEE Long Island Systems, Applications and Technology Conference, LISAT 2017, 2017, pp. 1–6.

[7] M. Kidon and R. Dobai, "Evolutionary design of hash functions for IP address hashing using genetic programming," in 2017 IEEE Congress on Evolutionary Computation, CEC 2017 - Proceedings, 2017, pp. 1720–1727.

[8] D. Liu, S. Xu, H. Liu, and Z. Cui, "An Empirical Study on the Performance of Hash Table," in ICIS 2014, 2014.

[9] N. Sarantinos, C. Benzaïd, O. Arabiat, and A. Al-Nemrat, "Forensic malware analysis: The value of fuzzy hashing algorithms in identifying similarities," Proc. - 15th IEEE Int. Conf. Trust. Secur. Priv. Comput. Commun. 10th IEEE Int. Conf. Big Data Sci. Eng. 14th IEEE Int. Symp. Parallel Distrib. Proce, pp. 1782–1787, 2016.

[10] M. Steinebach, P. Klöckner, N. Reimers, D. Wienand, and P. Wolf, "Robust hash algorithms for text," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2013, vol. 8099 LNCS, pp. 135–144.

[11] D. Wang, Y. Jiang, H. Song, F. He, M. Gu, and J. Sun, "Verification of Implementations of Cryptographic Hash Functions," IEEE Access, vol. 5, no. c, pp. 7816–7825, 2017.

[12] H. S. Dandvate and S. Pandya, "New approach for frequent item set generation based on mirabit hashing algorithm," Proc. Int. Conf. Inven. Comput. Technol.

ISSN- 2394-5125 VOL 07, ISSUE 15, 2020

ICICT 2016, vol. 2016, 2016.

[13] Mihuandayani, E. Utami, and E. T. Luthfi, "Text mining based on tax comments as big data analysis using SVM and feature selection," 2018 Int. Conf. Inf. Commun. Technol., pp. 537–542, 2018.

[14] M. Ong, "Bag of Words (BoW) - Natural Language Processing," 2014. [Online]. Available: https://ongspxm.github.io/blog/2014/12/bag-of-words-natural-language-processing/. [Accessed: 10-May-2018].

[15] G. Fuentes-pineda and I. V. Meza-ruiz, "Topic Discovery in Massive Text Corpora Based on Min-Hashing arXiv : 1807 . 00938v1 [cs . CL] 3 Jul 2018," pp. 1–23, 2018.

[16] S. Vallejo-figueroa, M. Rodríguez-artacho, M. Castro-gil, and E. San, "Using Text Mining and Linked Open Data to assist," pp. 1612–1617, 2018.

[17] J. Brownlee, "A Gentle Introduction to the Bag-of-Words Model," 2017. [Online]. Available: https://machinelearningmastery.com/gentle-introduction-bag-words-model/. [Accessed: 12-May-2018].

[18] B. A. Eclarin, A. C. Fajardo, and R. P. Medina, "A Novel Feature Hashing With Efficient Collision Resolution for Bag-of-Words Representation of Text Data," in Proceedings of the 2nd International Conference on Natural Language Processing and Information Retrieval - NLPIR 2018, 2018, pp. 12–16.

[19] S. George and S. Joseph, "Text Classification by Augmenting Bag of Words (BOW) Representation with Co-occurrence Feature," IOSR J. Comput. Eng., vol. 16, no. 1, pp. 2278–8727, 2014.

[20] Z. Li, J. F. Yang, L. Chen, and J. Zha, "Robust vehicle tracking using perceptual hashing algorithm," Proc. - 2015 IEEE 14th Int. Conf. Mach. Learn. Appl. ICMLA 2015, pp. 1111–1116, 2016.

[21] H. Yu and P. Moulin, "MULTI-FEATURE HASHING BASED ON SNR MAXIMIZATION ECE Dept ., University of Illinois at Urbana-Champaign , USA Advanced Digital Sciences Center , Singapore," pp. 1815–1819, 2015.

[22] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Hash Tables," Introduction to Algorithms. 2001.

[23] P. Nimbe, S. Ofori Frimpong, and M. Opoku, "An Efficient Strategy for Collision Resolution in Hash Tables," Int. J. Comput. Appl., vol. 99, no. 10, pp. 35–41, 2014.

[24] S. Kumar and P. Crowley, "Segmented hash: An efficient hash table implementation for high performance networking subsystems," in 2005 Symposium on Architectures for Networking and Communications Systems, ANCS 2005, 2005, pp. 91–103.

[25] D. Liu and S. Xu, "Comparison of Hash Table Performance with Open Addressing and Closed Addressing : An Empirical Study," vol. 3, no. 1, pp. 60–68, 2015.

[26] F. Franek, "Linked Data Structures," pp. 132–158, 2003.

ISSN- 2394-5125 VOL 07, ISSUE 15, 2020

[27] C. Shaffer, Data Structures and Algorithm Analysis, vol. 2. 2012.
[28] B. A. Eclarin, A. C. Fajardo, and R. P. Medina, "Enhanced hash algorithm using a two-dimensional vector to improve data search performance," Lect. Notes Networks Syst., vol. 67, pp. 59–69, 2019.
[29] D. G. Sullivan, "Fall 2012 Alternative Representation : A Linked List." pp. 1–19, 2012.
[30] M. R. Limon, R. Sharker, S. Biswas, and M. S. Rahman, "Efficient de Bruijn graph construction for genome assembly using a hash table and auxiliary vector data structures," in 2014 17th International Conference on Computer and Information Technology, ICCIT 2014, 2014, pp. 121–126.
[31] "algorithm - Can hash tables really be O(1)_ - Stack Overflow." [Online]. Available: https://stackoverflow.com/questions/2771368/can-hash-tables-really-be-o1. [Accessed: 11-Jun-2018].