Journal of Theoretical and Applied Information Technology
20th November 2013. Vol. 57 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

HASHSORT - A NOVEL HASHING FUNCTION BASED ON SORTING TECHNIQUE TO RESOLUTE COLLISION

S. MUTHUSUNDARI [1], R.M. SURESH [2]

[1]Research Scholar, Sathyabama University,Chennai,India
[2]Principal, Jerusalem Engineering College, Chennai, India

E-mial: nellailath@yahoo.co.in , rmsuresh@hotmail.com

ABSTRACT

Hash tables are extensively used in data structures to implement tables that associate a set of keys to a set of values, as they provide *O(1),* to perform operations such as query, insert and delete operations. However, at moderate collisions are quite frequent which not only increases the access time, but also decrease the performance in the deterministic. Due to this deterministic performance, the hash table degrades. In some systems, it is very difficult to keep the hash operations more deterministic. In recent trends, more research papers have been proposed, which employs a new and fast hash functions to implement hash tables and to avoid collisions. In this paper, we propose a novel hash table implementation called Hashsort function, which reduces the Collisions occur in the hash table. *The basic idea of this paper is* to reduce the collisions, such that automatically it increases the access time. High performance can be obtained by reducing the collisions in the hash table. Hashsort, makes an easy choice for the implementation of hash table in data structure.

Keywords: *Collision, Hashsort, Hash table, Sorting, Data structures.*

1. INTRODUCTION

Hashing is a method for storing and retrieving data from a database. It is used to insert, delete, and search for records based on a search key value. To implement the hash table, these operations need to have constant time. In fact, a good hash system typically shows at only one or two records for each search, insert, or delete operation. This performance takes the O(log *n*) average cost required to implement a binary search on a sorted array of *n* records, or the O(log *n*) average cost required to perform an operation on a binary search tree.

1.1 Hashing Techniques

- Hashing provides very fast access to records on certain search conditions.
- The search condition key on a single field, called the hash field.
- The main aim behind hashing is to provide a function 'h' called a hash function (or) randomizing function, that is applied to the hash field value of a record and yields the address of the disk block in which the record is stored.

Hashing is used for an internal search within a program whenever a group of records is accessed or exclusively by using the value of one field.

Hashing, a ubiquitous information retrieval strategy for providing efficient access to information based on a key. Under many circumstances, hashing is very effective in both time and space. Information can usually be retrieved in constant time. Space reference use is not exactly, but is at least acceptable for most incidents.

Hashing does have some drawbacks; they can lead to large performance undulate. Relevant factors include some knowledge of the domain (English prose vs. technical text, for instance), regarding the keys that will be stored, and stability of data. If these factors can be predicted

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

with some reliability in the information retrieval system, they usually need hashing an advantage of retrieval algorithms.

Consider the problem first from the performance standpoint. The goal is to avoid or to reduce If no keys collide, then placing the information associated with a key is simply the process of determining the key's location. Whenever a collision occurs, we need to further determine a unique location for a key. A collision that leads to the performance degrades.

Assume the domain of keys has N possible values. Collisions occurs whenever $N > m$, that is, when the number of values exceeds the number of locations in which they can be stored. The performance is achieved by having $N = m$, and using a 1: 1 mapping between keys and locations. Defining such a mapping is easy; the representation of knowledge is required. For example, if keys are consecutive integers in the range (N_1, N_2), then $m = N_2 - N_1 + 1$ and the mapping on a key k is $k - N_1$. If keys are two-character strings of lowercase letters, then $m = 26 \times 26$, and the mapping (using C character manipulation for ASCII) is (k [0] - ' a ') * (k [1] - ' a ') . These two mappings can be performed in a constant time.

The mapping is involved in hashing, that has two facets of performance: number of collisions and amount of unused space in the hash table. Optimization of one leads the expense of the other. The main aim of this paper, in hashing is to optimize both; that is, to tune both the facets parallel, so as to achieve a low number of collisions together with a reasonably small amount of unused space.

In order to avoid and to reduce collisions and increases the fast performance, the proposed Hashsort hashing function scheme works well as long as there are no collisions and lead to small amount of unused memory space. The time needs to store and retrieve data is proportional to the time to compute the hash function. Typically, this function is very easy to be calculated in constant time. The space occupied to store the elements is that required for an array of m elements. if m is small, this is not to be a problem. The remainder of the paper is organized as follows. Section II discusses the exisiting hashing functions. Section III deals the Related work, Section IV describes proposed hashsort : a sorting based hashing collisions. A collision occurs when two or more keys refer to the same places in the hash table.

function in greater detail. Section V discusses the Illustration of the proposed method.. Section VI presents the comparison with existing method linear probing and reports of the simulation results. Section VII considers the results. The paper concludes with Section VIII.

2. EXISTING HASH FUNCTIONS

Most of the hash functions, which are already to perform to implement the hash tables are as follows.

1. Division Method (MODULO arithmetic):

It takes the modulo operation of the key. i.e it takes the remainder of the key value, which is associated in to the table. H: Key ----> Integer Index

E.g. - Table size of 10

76 % 10 = 6 location in the table the key element 76 to be placed.

2. Mid-Square Method - Concat, Square and Remove the Middle.

3. Folding Method:
   a) break key up into binary segments (ASCII)
   b) XOR these together
   c) Calculate the numeric integer equivalent

3. RELATED WORK

Ross Anderson et.al [10] have presented a new and fast tiger hash function, which believe to be secured and designed to run quickly on 16 bit processor. They used the compression function to achieve the fast hash function.

Sailesh Kumar et.al [9] have presented a new peacock hashing function, which reduces the on chip memory by more than 10 folds and keeping high degree performance.

Alexander Russell et.al [12] proposed a collision free hash function, to inherit the structural

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

properties from the underlying simple clew free functions.

Rasmus Pagh et.al [15] have introduced a new Cuckoo hashing, is found to be practical. It uses two hash functions like h1 and h2. Based on the two hash functions a collision free table is constructed.

Carter and Wegman [23] proposed Universal Hashing, a way of avoiding assumptions on the distribution of input values.

Kirsch et al [24] have proposed the use of a *stash*, a simple data structure independent of the cuckoo hash table that is used to store keys that cause irresolvable collisions. The use of a stash can gently improve the failure probability bounds of insertion, and given knowledge of the total number of keys inserted, only a constant amount of additional space is required (Kirsch et al. 2008). Additional variants of cuckoo hashing include one that is engineered for use in hardware (Kirsch&Mitzenmacher 2008) and history-independent hashing (Naor, Segev & Wieder 2008). Kutzelnigg (Kutzelnigg 2008) analyzed the performance of an asymmetric cuckoo hash table (Pagh & Rodler 2004) (where the two hash tables used contain a different number of slots). However, this variant was found to increase the probability of insertion failure.

Nikolas et al [25] proposed how to efficiently implement an array hash table for integers. They have demonstrated, through careful experimental results, which hash table, whether it be a bucketized cuckoo hash table, an array hash table, or alternative hash table schemes such as linear probing, provides the best performance—with respect to time and space— for maintaining a large dictionary of integers in-memory, on a current cache-oriented processor.

4. PROPOSED HASHSORT - A SORTING BASED HASHING FUNCTION

Sorting makes the problem much simpler and to optimize the use of other functions. The choice of viewing hash function is very important. The right choice function, termed a hashsort function, would distribute all the elements into the hash tables such that no collisions ever occurred. Mapping a key to a table is performed very fast. A hashsort hash function guarantees the best uniform performance.

A proposed hashsort hashing function is an alternative to resolving collisions which makes the hash function performance very fast. The hashsort hashing function consists of the following steps.

4.1 Algorithm

1. Key elements to be arranged in sorted manner.
2. The first bit of the key is taken, that shows the key element to be placed in that location in the table.
3. If that location is not free in the table then the last bit of the key is taken in its respective place the key element is placed.
4. If first and last bit position is not free in the table, then only collision is occurred.
5. If collision is occurred, then based on the linear probing method in the next free cell the key element is place

4.2 Advantages of Hashsort hashing function

- Easy and fast to identify the location in the table
- Two locations are considered for placing the key element.
- No calculation is required to perform compare with the other hash function like mod function.
- Naturally cost and time is reduced.
- The effort required to perform a search is constant time because of the key element is in sorted manner.
- Number of collisions is reduced or eliminated based on the key elements.

5. ILLUSTRATION

Let us consider the key elements are inserted in to the hash table

141 28 34 41 58 69 129 85 65 127

Proposed Hashsort Algorithm steps:

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

1. Key elements to be arranged in sorted manner. By using Bubble sort technique the key elements are arranged in ascending order.

Given Key Elements: 141 28 34 41 58 69 129 85 65 127

Sorted Key Elements: 28 34 41 58 65 69 85 127 129 141

2. The first bit of the key is taken, that shows the key element to be placed in that location in the table. Consider the table size is 10.

Location	Key Element
0	141
1	127
2	28
3	34
4	41
5	58
6	65
7	129
8	85
9	69

Collision 1

Collision 2

The first key element is 28. The first bit of 28 is 2. Hence the first key element is to be placed in 2nd location in the table.

The second key element is 34. The first bit is 34 is 3. Hence the second key element is to be placed in 3rd location in the table.

The third key element is 41. The first bit is 41 is 4. Hence the third key element is to be placed in 4th location in the table.

The fourth key element is 58. The first bit is 58 is 5. Hence the fourth key element is to be placed in 5th location in the table.

The fifth key element is 65. The first bit is 65 is 6. Hence the fifth key element is to be placed in 6th location in the table.

The sixth key element is 69. The first bit is 69 is 6. Hence the first key element is to be placed in 6th location in the table. But the 6th location is not free. As per the step 3 the last bit of 69 is considered. The last bit of 69 is 9.

Hence the sixth key element is to be placed in 9th location in the table.

The seventh key element is 85. The first bit is 85 is 8. Hence the seventh key element is to be placed in 8th location in the table.

The eighth key element is 127. The first bit is 127 is 1. Hence the eighth key element is to be placed in 1st location in the table.

The ninth key element is 129. The first bit is 129 is 1. Hence the ninth key element is to be placed in 1st location in the table. But the 1st location is not free. As per the step 3 the last bit of 129 is considered. The last bit of 129 is 9. Hence the ninth key element is to be placed in 9th location in the table.

But ninth location is also not free. Hence collision is occurred.

As per the step 5 If collision is occurred, then based on the linear probing method in the next free cell the key element is placed.

The next free cell is 7th location. Hence it is placed in the 7th location.

The tenth key element is 141. The first bit of 141 is 1. The last bit of 141 is 1. Hence collision is occurred. By linear probing the next free cell is 0.
Hence it is placed in 0th location.

Only two collisions are occurred by this proposed Hashsort method.

6. COMPARISONS WITH EXISTING METHOD

6.1 Linear probing method

Let us consider the same key elements are inserted in to the hash table

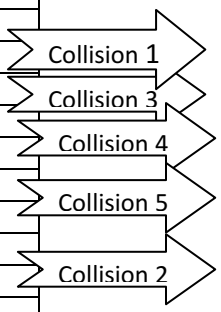
141 28 34 41 58 69 129 85 65 127

Journal of Theoretical and Applied Information Technology
20th November 2013. Vol. 57 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Consider the table size is 10.

Location	Key Element
0	69
1	141
2	41
3	129
4	34
5	85
6	65
7	127
8	28
9	58



First key element 141 is considered: 141 mod 10 = 1 : 1st location

Second key element 28 is considered: 28 mod 10 = 8 : 8th location

Third key element 34 is considered: 34 mod 10 = 4 : 4th location.

Fourth key element 41 is considered: 41 mod 10 = 1 : 1st location so collision (1).

The next free cell is 2nd location.

Fifth key element 58 is considered: 58 mod 10 = 8 : 8th location so collision(2)

The next free cell is 9th location.

Sixth key element 69 is considered: 69 mod 10 = 9 : 9th location so collision(3)

Hence the next free cell is 0th location.

Seventh key element 129 is considered: 129 mod 10 = 9 : 9th location so collision (4). Hence the next free cell is 3rd location.

Eighth key element 85 is considered: 85 mod 10 = 5 : 5th location.

Ninth key element 65 is considered: 65 mod 10 = 5 : 5th location so collision(5). Hence the next free cell is 6th location.

Tenth key element 127 is considered : 127 mod 10 = 7 : 7th location.

Here 5 collisions are occurred. For the same given key elements.

Table 1. Comparison Of A Proposed Hashsort Function With Existing Linear Probing Method

Given Data	Sorted Data	First bit	Last bit	Performance of proposed Hashsort		Performance of Exisitng Linear probing	
				Remarks	Location	Remarks	Location
141	28	2	8	No Collision	2	No Collision	1
28	34	3	4	No Collision	3	No Collision	8
34	41	4	1	No Collision	4	No Collision	4
41	58	5	8	No Collision	5	Collision	2
58	65	6	5	No Collision	6	Collision	9
69	69	6	9	No Collision	9	Collision	0
129	85	8	5	No Collision	8	Collision	3
85	127	1	7	No Collision	1	No Collision	5
65	129	1	9	Collision	7	Collision	6
127	141	1	1	Collision	0	No Collision	7
Total Number of Collisions				2		5	

7. SAMPLE TEST RESULT COMPARISON

Journal of Theoretical and Applied Information Technology
20th November 2013. Vol. 57 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Based on the comparisons were made from the above table 1, our proposed hashsort method is occurred with 2 collisions and the existing linear probing method is occurred with 5 collisions. Hence it is proven, our proposed method is 2.5 times much efficient than the existing method in terms of collisions. For any random generated set of data, the proposed hashsort method has a better choice to distribute the key elements in to the Hash Table.

8. RESULTS

We have measured the performance of the proposed hashsort method with existing collision avoidance technique linear probing method. The algorithm is implemented in C language.

The sample test data results are given below.

8.1 Case 1

Test Data : 5 elements are given as 23 45 43 12 78

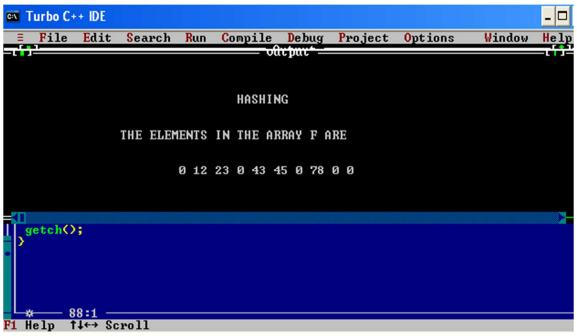


Fig 1. Case 1 Test Data Result For The Proposed Hashsort Function

The above fig 1 shows the test data for the proposed Hashsort function, where as no collisions are occurred. If it has been placed by the existing linear probing method, it takes one collision.

8.2 Case 2

Test Data: 8 elements are given as 12 56 65 78 98 34 23 34

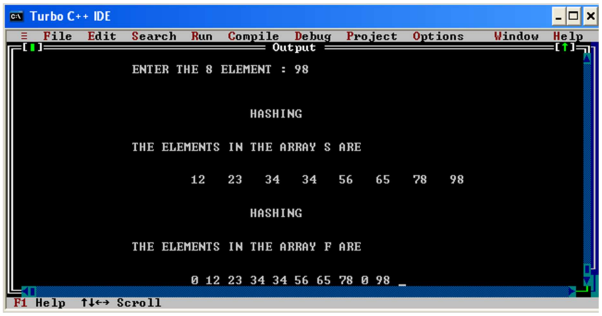


Fig 2. Case 2 Test Data Result For The Proposed Hashsort Function

In the above Fig 2 shows, the Hashsort function works with no collisions, while it is compared with linear probing, it takes 3 collisions. Hence we conclude, our proposed Hashsort method is performing minimum of 2.5 times greater than the existing linear probing method for any random number of n numbers. Hence the efficiency and High determinism is achieved by reducing the collisions in the hash table construction. Hashsort, makes an easy choice for the implementation of hash table in data structure.

9. CONCLUSION

In this paper we presented a new proposed Hashsort function is being implemented in C language. By analyzing the results above, we concluded that our proposed Hashsort method is better than the existing linear probing method. Hence the search operation is obtained in linear time and due to the less number of collisions the accessing time is increased and obtained the deterministic hash function. In future work, our proposed Hashsort method can be implemented by using D-Shuffle Sorting Technique instead of Bubble sort to arrange the key elements in ascending order.

REFERENCES

[1] A. Broder, M. Mitzenmacher, "Using Multiple Hash Functions to Improve IP Lookups", IEEE INFOCOM, 2001, pp. 1454-1463.
[2] W. Cunto, P. V. Poblete,"Two Hybrid Methods for Collision Resolution in Open Addressing Hashing", SWAT 1988, pp. 113-119
[3] P. Larson, Dynamic Hash Tables, CACM, 1988, 31 (4).

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[4] R. Pagh, F. F. Rodler, Cuckoo Hashing, Proc. 9th Annual European Symposium on Algorithms, August 28-31, 2001, pp.121-133.

[5] D. E. Knuth, The Art of Computer Programming, volume 3, Addison-Wesley Publishing Co, second Edition, 1998.

[6] L. C. K. Hui, C. Martel, On efficient unsuccessful search, Proc. 3rd ACM-SIAM Symposium on Discrete Algorithms, 1992, pp. 217-227.

[7] H. Song, S. Dharmapurikar, J. Turner, J. Lockwood, "Fast Hash Table Lookup Using Extended Bloom Filter: An Aid to Network Processing," SIGCOMM, Philadelphia PA, August 20-26, 2005.

[8] S. Kumar, and P. Crowley, "Segmented Hash: An Efficient Hash Table Implementation for High Performance Networking Subsystems", IEEE/ACM Symposium on Architectures for Networking and Communications Systems (ANCS), Princeton, October, 2005.

[9] Sailesh Kumar, Jonathan Turner, Patrick Crowley Peacock Hashing: Deterministic and Updatable Hashing for High Performance Networking *IEEE* Communications Society subject matter experts for publication *in* the IEEE INFOCOM 2008 proceedings pp 556-564

[10] Ross Anderson and El*i Biham*, " A new fast tiger hash function",

[11] Rasmus Pagh, Flemming Friche Rodler " Cuckoo Hashing" Elsevier Science December 2003

[12] Alexander Russell," Necessary and sufficient conditions for Collision free hashing", November 1995

[13]. Andrei Z. Broder and Michael Mitzenmacher. Using multiple hash functions to improve IP lookups. In Proceedings of the Twentieth Annual Joint Conferenceof the IEEE Computer and Communications Societies (INFOCOM 2001), volume 3, pages 1454–1463. IEEE Comput. Soc. Press, 2001.

[14]. Martin Dietzfelbinger and Philipp Woelfel. Almost random graphs with simple hash functions. In Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC '03), pages 629–638, 2003.

[15] Rasmus Pagh. On the Cell Probe Complexity of Membership and Perfect Hashing. In Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC '01), pages 425–432. ACM Press, 2001.

[16] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. In Proceedings of the 9th European Symposium on Algorithms (ESA '01), volume 2161 of Lecture Notes in Computer Science, pages 121–133. Springer-Verlag, 2001.

[17] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. Research Series RS-01-32, BRICS, Department of Computer Science, University of Aarhus, August 2001. 21 pp.

[18] Patricio V. Poblete and J. Ian Munro. Last-come-first-served hashing. J. Algorithms, 10(2):228–248, 1989.

[19] Ronald L. Rivest. Optimal arrangement of keys in a hash table. J. Assoc. Comput. Mach., 25(2):200–209, 1978.

[20] Mikkel Thorup. Even strongly universal hashing is pretty fast. In Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '00), pages 496–497. ACM Press, 2000.

[21] John Tromp, 2003. Personal communication.

[22] Petra Berenbrink, Artur Czumaj, Angelika Steger, and Berthold V¨ocking. Balanced allocations: the heavily loaded case. In Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC '00), pages 745–754. ACM Press, 2000.

[23] Carter J L and Wegmam M.N," Universal classes of Hash function", Journal of Computer Science 18(1979),pp 143 – 154

[24] Kirsch, A., Mitzenmacher, M., & Wieder, U. (2008), More robust hashing: Cuckoo hashing with a stash, *in* 'To appear in Proceedings of the 16th Annual European Symposium on Algorithms (ESA)'

[25] Nikolas Askitis, Fast and Compact Hash Tables for Integer Keys, appeared at the 32nd Australasian Computer Science Conference (ACSC 2009), Wellington, New Zealand, January 2009. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 91