IACSIT International Journal of Engineering and Technology, Vol.2, No.5, October 2010
ISSN: 1793-8236

Crux Search

Shrivatsan Rajagopalan and F.Sagayaraj Francis

***Abstract*— Search is a challenging concept in algorithms. It is challenging because striking the balance between performance and optimality is tough. Various numeric search techniques have been proposed, but achieving low time and space complexity is the main problem. A data structure and an algorithm should be thought of as a unit, neither one making sense without the other [7]. In this approach, a new data structure Binary Cube (BC) and an algorithm named Crux have been proposed for achieving the balance as stated. The proposed algorithm has a constant time complexity of one. The space complexity is minimal when compared to traditional approaches. The best, worst and average time complexity of the proposed search is O (1) for all the three. This performance is achieved using the proposed data structure, BC, which has been created specifically to render this high level of efficiency.**

***Index Terms*— Space complexity, Time complexity.**

I. Introduction

Search is a computing concept, used in every walk of life. The heart of simple ATM transaction or a Biometric security system inevitably is search. Optimal numeric search algorithms are certain tough challenges in the field of computing. Many search techniques have been proposed and are still being proposed. But achieving minimal space and time complexity is a tough constraint. Even if those constraints are met, new constrains creep into the problem. The proposed search aims at overcoming this constraint of striking the balance between minimal space and time complexity.

For instance, Binary search in arrays based storage reduces the number of comparisons in the search process. It has a best case time complexity of O(1), average case time complexity of O(log n) and worst case time complexity of O(log n) as in [1]. Despite all these benefits, it has a pre-condition that has to be satisfied. That is the sample set has to be pre-ordered. This means that apart from the time take for search, additional clock cycles are needed to render ease of search. Hence achieving this good a time complexity set is done by doing this single pre-condition taxing more time than the search itself, which is the actual aim. Hence the pre-condition that the input to the algorithm has to be sorted is a short coming of binary search.

The binary search trees, on the other hand, could be used to render search to get a time complexity of O (log n). However their time complexity depends on the sequence in which the data is feed to the tree. There are certain threaded traversal searches as in [2]. Many improvisations such as Morris search [2] have been proposed. Yet the increase in efficiency is just 5 – 10%. Thus binary trees are also in the same scale as that of simple binary search. Hence Binary search trees do not meet the constraints stated.

In tree traversal algorithms like depth first search, the time complexity is represented as T (v) = O (1 + dv), where dv is the depth of the vertex v, which is the search element [3]. Here the time complexity is dependent on the level in which the search element is present. Thus it is rather unconventional to use this search technique in search over large volumes of data since it has a polynomial time complexity. The other tree traversal search techniques like Pre order traversal and Euler Tour traversal also have their time complexity as O (n). Hence binary search is a better substitute to these algorithms and they don't overcome the constraints mentioned.

In Grover's algorithm the time complexity is O ($N^{1}/_{2}$) as in [4]. But the proposed algorithm has a time complexity of O (1). Although Grover's algorithm may render high degree efficiency in search, it can be implemented only in quantum computers which are yet to be commercialized, hence making the proposed one more preferable.

Hence is there a need to create one unique search that renders optimality without any pre-condition, taxing virtually small amount of space or and in minimal amount of time. The proposed search technique has a unique behavior of constant time complexity irrespective of sample set.

II. Scope of the Work

A search technique is now defined that renders a search mechanism that is relatively fast. This search when implemented in web can enhance the speed of searching. That is if a client searches for an item, which is not in the entire server farm, the proposed algorithm can instantaneously tell the client that an item of this sort doesn't exist in the Servers. This could be done without executing even a single query in any of the servers. Thus the proposed algorithm can directly reduce the CPU clock cycles and the energy needed to run the requests. Hence the scope of the proposed algorithm is relatively huge in the area of web search.

In distributed systems, the proposed algorithm can prove to be very effective in security. This is because the data structure can map a billion distinct entities in just five hundred and odd mega bytes of memory. The act of searching this huge a data set is done at just one check. Thus the proposed search technique can render breach-free security systems.

Parallel and highly massive computing systems need search mechanisms. The proposed algorithm shall prove to

Manuscript received August 2010. Shrivatsan Rajagopalan is with the SSN SASE, Chennai 603110, India.(Phone: +91-09789545260; e-mail: nshrivatsan@ gmail.com). Dr.F.Sagayaraj Francis, is currently an Associate Professor with the Department of Computer Science, Pondicherry Engineering College, Pondicherry, 605014, India (e-mail: fsfrancis@pec.edu).

be all the more valuable in terms of minimal latency in searching through the distributed system.

In networks, if the mechanism of identifying nodes in a subnet is flooding, the proposed search technique can reduce latency. If all the IP addresses of the nodes connected to the device are loaded on to the proposed data structure embedded to it, the proposed search can determine if a packet's destination exists in the device's network. This is done without sending even a single unnecessary packet.

Thus the proposed search technique has wide scope and variety of applications.

III. BC Data Structure and its Space Complexity

The data structure which has been proposed is a multi-dimensional cube bit array, hence the name. To store a number the proposed data structure uses only one bit. Be it one or nine crore ninety nine lakhs ninety nine thousand nine hundred and ninety nine, the memory used to store any of these numbers is just one bit. The data structure that has been proposed is named BC meaning Binary Cube, with each array representing a power of 10, say 'n', possessing each power of ten till n-1, which are nested internally. The following expression clearly explains the relation between the dimensions in x^{Th} power of ten.

No. of dimensions in BC_10x level = (x+1), where x is the no. of digits in the element. E.g. If x = 2, this is the 100's block and it possesses 2+1 =3 dimensions in BC.

To store each number in the data structure, the BC allocates a unique bit, with which a number's presence is denoted. An array of such cells constitutes a "block". The following expression explains the number of cells present in each block of the data structure.

No. of cells in BC_10x block = 9 x 10(x-1)

The following diagram explains how the sample set {1, 3, 7, 9} are stored in the proposed BC data structure.

	1		1				1		1
0	1	2	3	4	5	6	7	8	9

Fig 2.2 BC for storing all one digit numbers

Each of the ten squares cells represent the bits in the BC_10x the block. Here the value of x = 0. Hence the block is that of the "ones" block and each of the cells is used to represent all the ten one-digit numbers present, totaling to 9 x 100= 9 x 1 = 9 cells. This particular block of BC alone contains an extra element for representing zero and hence contains 9+1 = 10 bits to represent all one digit integers. All the numbers are stored using one bit each. Any traditional language like C, to store the number "1" it would consume 2 bytes that is sixteen bits. But the BC consumes only one bit per number. To store the same elements, int data type would consume 64 bits where-in, BC consumes only 4 bits.

If x = 2, this is the 10's block and it possesses 9 x 101 = 90 cells for BC of two digit numbers. This is explained by the following diagram.

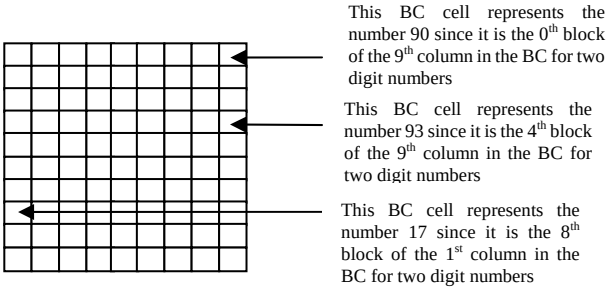

Fig 2.2 BC for storing all two digit numbers

This is the diagrammatic representation of BC for storing three digit numbers. Starting from 100 to 999, the BC stores all the 900 three digit numbers. The BC can represent all these 900 using 9 x 102 = 900 cells.

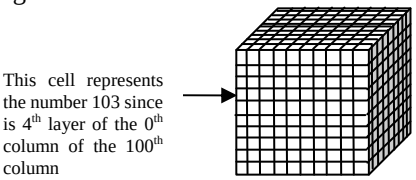

Fig 2.3 BC for storing all three digit numbers

To store 900 such cells, BC would consume only 900 bits which is nearly 113 bytes. Whereas int would need sixteen bits to store the same content just a one digit number "1". Hence to store 900 such entries C would consume 900 x 16 bits = 14400, which is 1800 bytes. Taking a ratio BC uses just 6.277 % of the space used by int. Thus ratio of space complexity of int in C to the proposed data structure is 16:1. That means the proposed data structure consumes just 6.27 % of space to store the same sample set as used by any traditional data structure. From one digit numbers to n digit numbers the percentage is the same. Thus the proposed algorithm has a unique space complexity and is near optimal compared to other data structure till date. The space occupied by a traditional data structures like int in C with the proposed data structure have been tabulated to elicit the proposed work's benefits.

TABLE 2.1. Space Complexity Analysis.

NO. OF DIGITS IN SAMPLE	TOTAL NUMBER OF ELEMENTS	BITS USED BY INT	BITS USED BY BC	MEMORY SAVED BY BC
1	10	160	10	150
2	90	1440	90	1350
3	900	14400	900	13500
4	9000	144000	9000	135000
5	90000	1440000	90000	1350000
6	900000	14400000	900000	13500000
7	9000000	144000000	9000000	135000000
8	90000000	1440000000	90000000	1340000000

The following graph and table elicit the benefits of BC's use in terms of memory utilization.

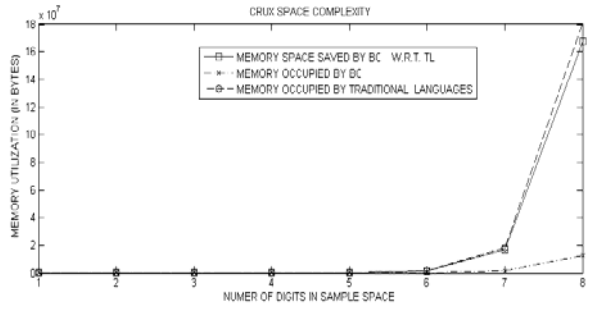

Fig. 2.1 Graphical comparison of memory used by BC and traditional languages

Thus from this table and the graph, it is evident that, for really large test spaces, the proposed BC can provide huge benefits in memory utilization. Hence the proposed data structure's space consumption for non sporadic test spaces is minimal.

IV. Crux Algorithm and Time Complexity

```
Step 1:  get k;                              // search element
Step 2:  set n = number_of_digits(k);        // no. of blocks in BC to be searched
Step 3:  for i = 1: n;
         x [i] = n % 10;
         n = n / 10;
     loop
Step 4: if BC [ x[n] ( x[n-1] ( x[n-2] (… x[0] ) ) ) ]     //The only comparison
         return "found";
else
         return "not found";
```

Fig 3.1 Crux - the search algorithm

The number of comparisons made inorder to find a search element is called the time complexiy of an algorithm. Various algorithms exist that have decent time complexity but the proposed has a minimal time complexity of "1". Crux makes "exactly one" check in the proposed data structure.

For instance if a search for the element "102" in the BC has to be done, the algorithm goes to the 2nd cell of the 0th column of the 100th block and check if it is "1". If it is present, it directly returns "found". If it is set to "0" then crux returns "not found". Thus this search technique has the least possible time complexity.

The unique characteristic of the proposed algorithm is that the best average and the worst case time complexities are the same. Hence in areas where a search has to be instantaneous with voluminous information, the proposed algorithm is the relatively better.

V. Implementation

The proposed algorithm has been implemented using java. There are certain issues in implementing like java heap size and frozen unused memory. This unused memory can be avoided. This is feasible since java supports jagging of arrays. Hence only those cells can be initialized which are currently under use and those which are not can be left undeclared saving space. Thus the only most obvious short coming of the BC-crux search could be handled easily.

VI. Real-Time Applications

In web searches, pages are scanned for presence of words and are mapped correspondingly. The English language, for instance has only 171,476 so many words as per [8]. If each word is given a number as per dictionary's chronological order and each web page is compressed into a simplified BC object, the crawler can be used to create a BC object for each of the web pages. And if crux search is performed, not only do we minimize space utilization but also shorten the search time.

As stated previously, in networks, if implemented the proposed algorithm can prevent network traffic totally, eliminating the need for flooding. This can totally reduce the round trip time in networks. This concept could be used in any area involving flooding as a necessary mode of detection of presence.

In distributed databases, if used, the proposed algorithms can stop irrelevant queries, reducing the disc seeks by the database.

In problems involving pattern search, a clustered crux search can render searches with a time complexity of n where n is the number of crux clusters. In social security identification systems, the proposed algorithm will play a major role in preventing imposters. This is because if a person's ID is already set in one location and if the same ID is found in some other location, it can detect it instantaneously. Thus intruders can be stopped immediately.

In Software security over internet, security can be enhanced if the proposed algorithm is used. The working is very similar to that of the unique ID concept. There are various other areas were the algorithm could be implemented.

VII. Conclusion And Future Work

Thus the proposed search technique has minimal space complexity and time complexity. Currently this algorithm has been implemented using java. This algorithm has been modified to render string search too. Since string searches are a bit different from numeric search, crux could be optimized for it and this is left as a future work.

Acknowledgment

The authors would like to acknowledge the contributions of various people, especially, Dr. S. Kanmani, Head of Department of IT, Pondicherry Engineering College.

References

[1] Elliot B. Koffman, Paul A. T. Wolfgang, *Data Structures: Abstraction and Design Using Java*, January 2010, Ch. 5 Pg – 262 to 264
[2] Adam Drozdek, *Data Structures and Algorithms in Java*, Second Edition - Cengage Learning.
[3] Michael T. Goodrich and I Roberto Tamassia, *Algorithm Design*, John Wiley & Sons, Inc, Pg. 79
[4] Grover L.K., " A fast quantum mechanical algorithm for database search", *Proceedings*, 28th Annual ACM Symposium on the Theory of Computing, (May 1996) Pg. 212
[5] Octree – Data Structure, http://en.wikipedia.org/wiki/Octree
[6] Clustered Index Structures, Microsoft SQL Server R2, http://msdn.microsoft.com/en-us/library/ms177443.aspx
[7] Ellis Horowitz and Sartaj Sahani, *Fundamentals of Data Structures*, Computer Science Press.
[8] http://www.askoxford.com/asktheexperts/faq/aboutenglish/numberwords.

Shrivatsan Rajagopalan completed his Bachelors in Information Technology, course at Pondicherry Engineering College in the year 2010. He is currently pursuing his Masters in Information Technology course, offered by Carnegie Mellon University in collaboration with SSN – SASE, Chennai, India. He was awarded the Best Research Student Paper award for his presentation at CCV 2010, at Singapore, where he had presented his paper on Cloud Computing.

Dr.F.Sagayaraj Francis, is a Doctorate in the field of Data Management. He is currently an Associate Professor with the Department of Computer Science, Pondicherry Engineering college.