Time and Space Efficient Algorithm for Consumer's Priority Product Management

Mahdi Nasrullah Al-Ameen[1*]
Dept. of Computer Science and Engineering
University of Texas at Arlington
Arlington, USA
mahdi.al-ameen@mavs.uta.edu

Mehrab Shahriar[1*]
Dept. of Computer Science and Engineering
University of Texas at Arlington
Arlington, USA
mdmehrab.shahriar@mavs.uta.edu

Adip Billah
Institute of Business Administration (IBA)
Dhaka University
Dhaka, Bangladesh
adip_billah@yahoo.com

***Abstract*—**

In this highly competitive free market economy, the test or interest of a certain consumer plays a momentous role for the business organizations to select appropriate products to be advertised to him through the online store or their website. In other words, business organizations need to be proficient in advertising their products to attract the potential consumers. To achieve the goal, organizations have to keep track of the consumers' buying habit to figure out the priority products. Thus the consumers' priority product management is a candidate for high degree of attention. In this paper, we have developed and analyzed the algorithm for consumers' priority product management with minimum time and space complexity. We use the concept of balanced binary search tree in this case to attain efficient time complexity. For reducing space requirement, we have proposed merging algorithm that incorporates with the balanced binary search tree, to achieve the optimal performances. We have simulated for 1 million test cases and our results show that the algorithm achieves a high memory saving index with logarithmic time complexity in each working round.

***Keywords*-Information Management, Consumers' Priority Product Management, Digital Store, Binary Search Tree, Red Black Tree**

I. Introduction

Corporate Business has become highly competitive now-a-days mostly because of the ever flourishing free market economy around the globe.. The concept of digital store is getting popular day by day, as it not only saves the time and effort necessary for shopping in physical store but also makes the payment method easier and convenient. The organizations are smart enough these days to win the hardest corporate race and thus they are constantly working to find out the best strategy to manipulate consumers' test and interest and make them purchase their products [1-4]. They are highly attentive to keep track of consumers' buying behavior in digital store, analyze them in detail and consequently suggest additional products for them that are aligned more or less with their previously purchased products.

Each consumer is assumed to have his individual account (protected by user name and password) in the digital store (website) and every time he purchases an item, the buying request is issued from his account. Through the analysis of consumers' buying behavior, the organizations figure out the products of interest or in other words, consumers priority products [5,6]. The consumers can add an item to their digital cart and purchase those anytime later. In addition to purchased items, these selections help the organizations to figure out the consumers' priority products. The mechanism not only increases the revenue of the organization but also helps a consumer to find out his desired products.

Each organization has limited resource of digital storage but they have to store information of millions of consumers all over the world [6]. So, they must pay attention to store the consumers' priority products with minimum space complexity. In reality, for a consumer, the more priority product information they can store, more accurate will be the product suggestion for him. However, the organization has to pay for keeping and maintaining digital storage. So, managing the information with minimum space complexity is directly related to the financial facts of an organization.

In every second, a number of consumers visit the digital store of an organization. They expect a faster response from the server to get their desired products displayed. So, the time complexity of generating product suggestion for a consumer needs to be as minimum as possible.

In this paper, we have developed an algorithm for smart storage and retrieval of consumers' priority products with minimal space and time complexity. We discuss the algorithm in later sections.

II. Contribution

To realize the contribution of our work, we first analyze the existing mechanism for consumers' priority product managment and then show how our algorithm comes out with better performances.

A. Analysis of Existing Mechanism

To manage the consumers' priority product information, existing mechanism follows a straightforward approach. It maintains a multidimensional vector space. In one dimension, it maintains a list containing the name and ID of each

[1]These authors contributed equally to this work.
[*]To whom correspondence may be addressed.

978-1-4673-4836-2/12/$31.00 ©2012 IEEE

consumer and another dimension keeps their corresponding priority product lists. It stores the product name or ID as a string. So, if n is the number of priority products for a consumer, a long string that contains name or ID of these n products are stored for him and separator is used to identify each product independently in the string.

For m number of consumers, if the average number of priority products for each consumer is n then the space complexity of existing mechanism is $O(mn)$. Searching for the priority product list of a consumer takes $O(m)$ time as it is required to find the appropriate consumer and retrieve the whole string of his priority products. Insertion of a priority product into a list also takes $O(m)$ time as it is dominated by the time complexity of finding the appropriate consumer in the list. If the length of priority product string is limited then it may be required to delete the oldest priority product to insert the newest one. Each time a new product is inserted into the list it is concatenated to the end of string and thus the leftmost product becomes the oldest priority product. If a pointer is maintained for the oldest priority product, reassigning this pointer takes $O(1)$ time and thus the whole update operation takes $O(m)$ time.

B. Improvement Through our Algorithm

1) Logarithmic Time Complexity: We have developed a mechanism that takes logarithmic time to perform any operation like, insert, update, search or delete. So our algorithm attains a significant improvement over the existing mechanism. In this case, we propose to use Red-Black-Tree that asymptotically attains logarithmic behavior for the mentioned operations. We take the hash of priority products of a consumer that represents a numerical value and stores the information as a node in a Red-Black-Tree (RBT). A pointer is maintained to the corresponding consumer information (name, ID and so on), stored in another RBT.

2) Reduced Space Complexity: Our algorithm is space efficient in comparison to the existing mechanism. We use two RBTs where nodes of one RBT (say it, URBT) represent the lists of priority products that are unique (that is, the list of priority products that are stored in this RBT do not belong to more than one consumer). Nodes of another RBT (say it MRBT) contain those lists of priority products that belong to more than one consumer. So if more than one consumer have same lists of priority products, we merge the lists and a node in MRBT stores that merged list. For example, if Mr. X, Y and Z have same lists of priority products, in existing mechanism all these three lists are separately stored. But in our algorithm we store only one list in a node of MRBT and from this node we maintain pointers to the consumer information of Mr. X, Y and Z. So, we have reduced space complexity in our algorithm. In our simulation results, we have shown the memory savings index that corresponds to the significant improvement achieved through our algorithm over the existing mechanism.

III. Background

A binary search tree (BST) is also termed as an ordered or sorted binary tree, which is a node-based binary tree data structure. Generally, the information represented by each node is a record rather than a single data element [7]. However, for sequencing purposes, nodes are compared according to their keys rather than any part of their associated records. The major advantage of binary search trees over other data structures is that the related sorting algorithms and search algorithms such as in-order traversal can be very efficient [8, 9]. The basic operation of binary search tree are insert, delete and search which takes $O(h)$ time where h is the height of the tree. For a balanced binary tree we get $h = O(logn)$. So the basic operations in that case attain logarithmic time complexity.

A redblack tree is a type of self-balancing binary search tree, a data structure that is usually used to implement associative arrays. It is complex, but has good worst-case running time for its operations and is efficient in practice: it can search, insert, and delete in $O(logn)$ time, where n is the total number of elements in the tree. So, a redblack tree is a binary search tree that inserts and deletes in such a way that the tree is always reasonably balanced [10].

The properties of redblack trees enforce that the path from the root to the furthest leaf is no more than twice as long as the path from the root to the nearest leaf. The result is that the tree is roughly balanced. Since operations such as inserting, deleting, and finding values require worst-case time proportional to the height of the tree, this theoretical upper bound on the height allows redblack trees to be efficient in the worst-case, unlike ordinary binary search trees [10].

IV. Our Proposed Algorithm

A. Description

In our algorithm, if a list of priority products of a consumer do not match with the lists of any other consumers, it is stored in URBT. On the other hand, if the list of priority product of a consumer matches with the list of another consumer, the matched lists are merged and stored in MRBT.

When a consumer enters the system, first it is checked if the consumer already has an account, that is if he is a returning consumer or a new one. If he is a new consumer, his information are hashed and stored in a RBT (say it CRBT). The hashed value is a numerical figure that decides the position of a node in CRBT where the information are stored. Each node in CRBT is linked with the corresponding nodes in MRBT or URBT in cirucular fashion so that the consumer information in CRBT can be used to find out the corresponding list of priority products in MRBT or URBT. In the same way the list of priority products (in MRBT or URBT) can be used to find out the corresponding consumer (in CRBT).

If a returning consumer (say, Mr. A) enters the system, his priority product list is fetched from URBT or MRBT. A boolean index in each node of CRBT keeps the information about which of URBT or MRBT contains the priority product list of the corresponding consumer. At this point, based on his

```
Data Structrues :
pRB : Corresponds to MRBT
tRB : Corresponds to URBT
L: Corresponds to CRBT Input ::
(Hash of consumer information : CI, hash of consumers
priority product list : HL)
Operations :
check if CI exists in L;
if yes then
    p1 = position of CI in L;
    check if the given hash matches with the hash in
    pRB or tRB, pointed by entry : p1;
    if yes then
        do nothing;
    end
    else
        if p1 points to an entry in tRB then
            remove that entry from tRB;
        end
        if p1 points to an entry in pRB then
            i = number of CI in L that points to this
            entry in pRB;
            if i==2 then
                CI2 = the hash of the
                consumer-information that points to this
                entry in pRB other than the new
                consumer-information (input);
                find the entry p4 in L for CI2;
                insert (CI2, hash) into tRB;
                remove the entry from pRB;
                Update entry : p4 in L with the location
                of inserted hash in tRB;
            end
            else
                i = i -1 for this entry in pRB;
                Remove the pointer from this entry in
                pRB to the entry in L for CI and vice
                versa;
                call manage(CI, hash, p1);
            end
        end
    end
end
else
    insert the CI into L;
    p1 = position of CI in L;
    call manage (CI, hash, p1);
end
```

```
manage(CI, hash, p1) {
check if the hash matches with any entry in pRB;
if no then
    p2 = position in pRB where (CI, hash) can be
    inserted;
    check if the hash matches with an entry in tRB;
    if no then
        insert (CI, hash) into tRB;
        Update entry : p1 in L with the location of
        inserted hash in tRB;
    end
    else
        CI1 = Hash of the consumer information whose
        priority product list matches with the new
        priority product list (the input);
        insert (CI1, CI, hash) into the position : p2 in
        pRB;
        Delete (CI1, hash) from tRB; Update entry : p1
        in L with the location of inserted hash in pRB;
        find the entry p3 for CI1 in L;
        Update entry : p3 in L with the location of
        inserted hash in pRB;
    end
end
else
    Update entry : p1 in L with the location of matched
    hash in pRB;
    i = i + 1 for the location of the matched hash in
    pRB;
end
}
```

Algorithm 1: Pseudo-Code of the Algorithm

purchase behavior his priority product list may be changed. In this case, we assume that the purchase behavior of consumers is analyzed in correct manner using standard mechanism. We take the hash of priority product list before we compare and store in RBT. The hashed value represents a numerical figure that determines the position of the node (that stores the hash) in RBT. Now, the hash of new priority product list of the consumer is compared with the fetched one. If no difference is found, no additional action is taken from algorithmic perspective.

If the new priority product list is different from the fetched list, the actions are twofold. If the hash of the fetched list is stored in a node in URBT that node is deleted as it is not linked with any other consumer. But if the hash of the fetched list is stored in a node in MRBT and more than two consumers have the same list, then the node cannot be deleted but the circular link is deleted with the node in CRBT that stores corresponding consumer information (of Mr. A). So the consumer information of Mr. A no longer has any link with his old priority product list. If only two consumers (including Mr. A) have links to the node in MRBT that stores the fetched list then the node is deleted and the hash of this list is inserted as a new node into URBT. A circular link is created from this node with the node in CRBT that stores the information of

the consumer (other than Mr. A).

Now it is checked if the new hash matches with any hash in MRBT. If such a match is found, a circular link is created from that matched node in MRBT with the node in CRBT that stores the corresponding consumer information (of Mr. A). But, if no such match is found the new hash is checked with the hash in URBT. If no match is found, a new node is inserted into URBT that stores the new hash and the circular link is created with the corresponding node in CRBT. But if a match is found in CRBT, now we find two consumers having same list. The node that stores the hash of the matched list in URBT, is deleted as the matched hash is now merged with the new hash and is inserted into MRBT as a new node. The circular links are updated accordingly. When a priority product list of a new consumer is found, it is handled in the same way as described in last paragraph.

B. Theoretical Analysis

For analysis we assume, An Operation = Load : Consumers' Information + Update : The list of consumers' priority product + Merge : When more than one consumer has same priority product list.

In our algorithm, while executing an operation, the basic operations include insert, delete and find operations on RBT that takes $O(logn)$, where n = number of consumers whose information are stored in the system. So, performing n number of such operations represents the scenario where n number of consumers subsequently enter the system and based on their purchase behavior their priority product lists are updated.

In the worst case, the value of n is increased by 1 in each operations that represents the scenario where each time a new consumer enters the system. So our algorithm takes $O(log1 + log2+...+logn)$ time that is asymptotically represented by $O(nlogn)$.

V. Simulation and Results

We have used Java simulator for implementing our algorithm and randomly generated test cases using existing Java libraries.

A. Performance Analysis

We have simulated over different number of operations based on different statistical distributions. Each of the test cases in our simulation accomodates different number of requests containing consumer information and his priority product list. For a specific number of operations, we have conducted the test a significant number of times and taken their average output result.

We have simulated for upto 1 million test cases and the execution time (ET) is shown in Table I. A close observation on the ratio of $T(i)$ and $T(i-1)$, $n(i)logn(i)$ and $n(i-1)logn(i-1)$ reveals the fact that each operation in our algorithm takes $O(logn)$ time. So for n operations the time complexity is $O(nlogn)$. It worths mentioning that using different statistical distributions doesn't affect the execution time, rather it solely depends on the number of requests. So the simulation results match with our theoretical analysis.

TABLE I
Performance Analysis [$n(i)$ = $Number\ of\ Operations\ at\ i$]

i	n(i)	ET,T(ms)	$\frac{T(i)}{T(i-1)}$	$\frac{n(i)log(n(i))}{n(i-1)log(n(i-1))}$
1	1000	102		
2	2000	221	2.16	2.20
3	4000	475	2.15	2.18
4	8000	1007	2.12	2.16
5	16000	2120	2.10	2.15
6	32000	4449	2.09	2.14
7	64000	9164	2.05	2.13
8	128000	18696	2.04	2.12

B. Effectiveness in Memory Savings

The effectiveness of our algorithm in reducing space complexity is represented by memory savings index that is a numerical indication of space-reduction efficiency, gained through merging the same hash contents instead of having individual storage for each same hash value. Here memory savings index is represented in the following way,

Memory Savings Index (MSI) = X- Y

where, X = Number of Consumers who have the hash of their priority product lists in MRBT

Y = Number of nodes in MRBT.

TABLE II
Effectiveness in Memory Savings for Uniformly Distributed Dataset [n = $Number\ of\ Operations$, c = $Number\ of\ Diff.\ Consumers$]

n	c	X	Y	MSI
1000	431	394	140	254
10000	4332	3855	1292	2563
50000	21559	19125	6393	12732
100000	43219	38258	12713	25815
1000000	432077	382462	127063	255399

For the first case, we use uniform distribution to generate the test cases, considering all the priority lists to be equally likely. Our results in table II shows that for 1 million operations memory savings index is 255399 that represents the effectiveness of our algorithm in achieving the significant gain in reducing space complexity in comparison to the existing mechanism for consumers' priority product management.

However, analysis shows that in reality, the buying trend of the consumers doesn't follow any uniform direction. It is evident that all sort of products are not sold at the same frequency. So, instead of all the priority lists having the same probability, it is more likely that some lists will appear more than the other lists [11, 12]. In order to introduce some bias in the consumer's choice , we use weibull distribution with paramters k=0.4 and keep the mean value same as the uniform distribution. To note, weibull distribution is a representative distribution in some other biased data-sets such as file sizes [13].

TABLE III
EFFECTIVENESS IN MEMORY SAVINGS FOR BIASED DATASET [n = $Number\ of\ Operations$, c = $Number\ of\ Diff.\ Consumers$]

n	c	X	Y	MSI
1000	439	359	69	290
10000	4339	3804	895	2909
50000	21639	18816	4265	14551
100000	43247	37732	8697	29035
1000000	432464	375945	85282	290663

Results shown in table III ensure that the memory savings index increases with a considerable amount when we use biased dataset that stands for the effectiveness of our algorithm.

VI. CONCLUSION AND FUTURE WORK

The existing mechanism for consumers' priority product management takes $O(n^2)$ time for performing n operations where our algorithm takes $O(nlogn)$ time. In case of space complexity our algorithm gains a significant improvement that is reflected by high memory savings index. So, considering both time and space complexity our proposed algorithm performs much better than the existing mechanism. Thus our algorithm highly fits in large scale digital information management of the consumers priority products with a faster response and reduced space complexity. The future work involves simulating our algorithm in a real world environment.

VII. REFERENCE

[1] Johnson, G, Scholes, K, Whittington, R *Exploring Corporate Strategy*, 8th Edition, FT Prentice Hall, Essex, 2008, ISBN 978-0-273-71192-6.

[2] Blaxill, Mark Eckardt, Ralph, *The Invisible Edge: Taking your Strategy to the Next Level Using Intellectual property*(Portfolio, March 2009).

[3] Buzzell, R. and Gale, B. *The PIMS Principles: Linking Strategy to Performance*, Free Press, New York, 1987.

[4] Mulcaster, W.R. *"Three Strategic Frameworks"*, Business Strategy Series, Vol 10, No1, pp68 75, 2009.

[5] Nag, R.; Hambrick, D. C.; Chen, M.-J, *What is strategic management, really? Inductive derivation of a consensus definition of the field*. Strategic Management Journal. Volume 28, Issue 9, pages 935955, September 2007.

[6] de Wit and Meyer, *Strategy Process, Content and Context*, Thomson Learning 2008.

[7] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition.* MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Chapter 12: Binary search trees, pp. 253272. Section 15.5: Optimal binary search trees, pp. 356363.

[8] Heger, Dominique A. (2004), *"A Disquisition on The Performance Behavior of Binary Search Tree Data Structures"*, European Journal for the Informatics Professional 5 (5): 6775.

[9] Donald Knuth. *The Art of Computer Programming*, Volume 3: Sorting and Searching, Third Edition. Addison-Wesley, 1997. ISBN 0-201-89685-0. Section 6.2.2: Binary Tree Searching, pp. 426458.

[10] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition.* MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7 . Chapter 13: RedBlack Trees, pp. 273301.

[11] Erik Brynjolfsson, Michael D. Smith, *The Great Equalizer? Consumer Choice Behavior at Internet Shopbots*, published by [Cambridge, Mass.] : MIT Sloan School of Management.

[12] Michael D. Smith, Erik Brynjolfsson, *Consumer Decision-Making at an Internet Shopbot : Brand Still Matters*, published in The Journal of Industrial Economics (Vol. 49, Issue 4 (December 2001), Pages : 541-58).

[13] Allen B. Downey, *The Structural Cause of File Size Distributions*, published in the proceedings of international conference on measurement and modeling of computer systems (ACM SIGMETRICS 2001).