ALGORITHMIC STRATEGIES IN INTELLIGENCE ANALYSIS

Article · March 2019

CITATIONS
0

READS
50

3 authors:

Gabriel Akibi Inyang
Cross River University of Technology
1 PUBLICATION **0** CITATIONS
SEE PROFILE

Anthony O Otiko
Cross River University of Technology
10 PUBLICATIONS **0** CITATIONS
SEE PROFILE

Prince Ana
Cross River University of Technology
20 PUBLICATIONS **23** CITATIONS
SEE PROFILE

ALGORITHMIC STRATEGIES IN INTELLIGENCE ANALYSIS

[1]G. A. Inyang, [1]A.O.Otiko, [1]P.Ana
08134141082, 08033183236, 08036178995
gain140270@gmail.com, anaprince@yahoo.com, otikotony@gmail.com

[1]Department of Computer Science, Cross River University of Technology, Calabar, Nigeria.

ABSTRACT
Knowledge discovery from data is an inherently iterative process. That is, what is known about the data greatly determines expectations, and therefore, what results would be found interesting and/or surprising. Given new knowledge about the data, expectations will change. Intelligence analysis is a process of collecting and generating intelligence from multiple sources such as data and information. In order for intelligence analysis to be accurate, there must be a process that ensures that the data produced is valid. In the physical and mathematical sciences, where the standards of evidence and proof are less vulnerable to subjective judgments, something much closer to a truly objective judgment results. This paper is to demonstrate how strategic algorithms are in intelligence analysis. The consideration in this paper is on identifying classes of algorithmic strategies relating to data structured process using an array of elements. The algorithmic strategies in the search, insertion, deletion, and updating operations support the iterative refinement of data exploration. This demonstrates how these methods guide on the investigation on intelligence analysis on data structures. The results reveal superiorities of the algorithmic strategies and lead to several design recommendation for creating data structures in computing.

Keywords: Algorithmic Strategies; Intelligence Analysis; Array; Data; Data Structures

1.0 INTRODUCTION

Intelligence analysis today is faced with many challenges, chief among them being the need to fuse disparate streams of data, and rapidly arrive at analytical decisions and quantitative predictions for use by policy makers.

Modern communication forms such as social media have increased the diversity of sources available but have also created further challenges, especially the need for algorithms and software tools to suitably guide in intelligence analysis and complement human analytical skills. Although intelligence analysis covers a wide range of specialties, such as Security, Treasury, Energy, Transportation, Commerce, Computing and Justice, one of the veritable problems in intelligence analysis is: given a collection of data to analyze, which entities should one look at first? What activities appear suspicious to warrant further investigation and follow up, and, what algorithmic strategy is to be applied?

This paper is to show how efficient algorithmic strategies are in intelligence analysis. To achieve this, we will examine and analyse the different algorithmic operations in an array of data. And also examine the algorithmic complexity in an array of data.

In examining and analysing the different algorithmic operations and examining the algorithmic complexity in an array of data, the divide and conquer approach will be considered.

2.0 RELATED WORK

Modern software tools that support intelligence analysis are motivated by different considerations. Specific categories of these systems include association rule mining based systems (Buczak and Gifford, 2010.), classification based tools e.g Skillicorn (2010), model-guided software e.g., Koltuksuz and Tekir (2006), collaborative systems e.g., Bier et

al (2010), and multi-agent systems e.g. Lindahl et al (2007). The analysis process in a typical intelligence analysis exercise can be viewed through the framework (Card Pirolli and Card 2007), namely; information search and sense-making. Intelligence analysis involves the development of recommended predictions of action, based on a wide range of available sources of information, both open and undercover. The analysis is developed in response to the requirements of the organization's or client's management to help make decisions. (Sfetcu 2016)

Some analytic systems, such as, (*Jigsa 2012)*, *NetLens (*Kang 2007) focus on supporting the information search loop. Other tools, such as *Analyst's Notebook i2group (2011)*, and *Entity Workspace (*Bier et al 2006) focus more on sensemaking. The aspect of correctness of intelligence analysis is dependent upon the algorithmic tool or strategic tool used. This paper is aimed at demonstrating the efficacy of algorithmic strategies in the task of information search and sensemaking processes in an array of elements using the Divide and Conquer method.

2.1 ALGORITHMIC STRATEGIES

Algorithmic analysis deals with the execution or running time of various operations involved in any sequence of data. The running time of an operation can be defined as the number of computer instructions executed per operation. Based on the above, algorithm efficiency can be analyzed in two different stages, before implementation and after implementation. These stages are as follows −

- **A Priori Analysis:** This is a theoretical analysis of an algorithm. Efficiency of an algorithm here is measured by assuming that all other factors, for example, processor speed, are constant and have no effect on the implementation.
- **A Posterior Analysis:** This is an empirical analysis of an algorithm. The selected algorithm is implemented using programming language. This is then executed on target computer machine. In this analysis, actual statistics like running time and space required are collected.

2.2 ALGORITHM COMPLEXITY

Suppose **X** is an algorithm and **n** is the size of input data, the time and space used by the algorithm X are the two main factors, which decide the efficiency of X.

i. Time Factor − Time is measured by counting the number of key operations such as searching, insertion, deletion, updating and comparisons in the sorting algorithm.

ii. Space Factor − Space is measured by counting the maximum memory space required by the algorithm.

The complexity of an algorithm **f(n)** gives the running time and/or the storage space required by the algorithm in terms of **n** as the size of input data.

2.3 SPACE COMPLEXITY

Space complexity of an algorithm represents the amount of memory space required by the algorithm in its life cycle. The space required by an algorithm is equal to the sum of the following two components −

- A fixed part that is a space required to store certain data and variables that are independent of the size of the problem. For example, simple variables and constants used, program size, etc.
- A variable part is a space required by variables, whose size depends on the size of the problem. For example, dynamic memory allocation, recursion, stack space, etc.

Space complexity S(P) of any algorithm P is S(P) = C + SP(I), where C is the fixed part and S(I) is the variable part of the algorithm, which depends on instance characteristic I. Retrieved August 29, 2019,from https://www.studytonight.com/data-structures/space-complexity-of-algorithms.

2.4 TIME COMPLEXITY

Time complexity of an algorithm represents the amount of time required by the algorithm to run to completion. Time requirements can be

defined as a numerical function T(n), where T(n) can be measured as the number of steps, provided each step consumes constant time. For example, insertion operation involving n-bit of data takes **n** steps. Consequently, the total computational time is $T(n) = c * n$, where c is the time taken for the insertion of the bits. Here, it is observe that T(n) grows linearly as the input size increases. Retrieved August 28, 2019, from https://www.studytonight.com/data-structures/time-complexity-of-algorithms.

Usually, the time required by an algorithm falls under three types:

- **Best Case** − Minimum time required for program execution.
- **Average Case** − Average time required for program execution.
- **Worst Case** − Maximum time required for program execution.

2.5 INTELLIGENCE ANALYSIS

Intelligence analysis is a process of collecting and generating intelligence from multiple sources such as data and information. The process usually involves accumulating information about a variety of circumstances and individuals who have knowledge in areas that include strategy, operations, or tactical intelligence. According to the Central Intelligence Agency, "Intelligence analysis is the application of individual and collective cognitive methods to weigh data and test hypotheses within a secret socio-cultural context." Retrieved August 28, 2019, from https://www.securitydegreehub.com/what-is-intelligence-analysis/

2.5.1 WHY INTELLIGENCE ANALYSIS

In order for intelligence analysis to be accurate, there must be a process that ensures the data produced is valid. Accurate intelligence analysis is important because it provides correct judgment and decision making. Once the right information is produce, the possibility of making correct decision and conclusion remain evident, and the result relays the capacity to develop timely and well-formulated strategic intelligence. Retrieved August 28, 2019, from https://www.securitydegreehub.com/what-is-intelligence-analysis/

2.5.2 THE PROCESS OF INTELLIGENCE ANALYSIS

Intelligence analysis is successfully collected through an intelligence process, and this include.

- **Requirements**
  Requirements involve defining questions that identify what data or information is expected to be gathered, and it can also mean a detailed assembly of certain types of intelligence.
- **Collection**
  When the requirements are established, the process of collecting a variety of information takes place. There are some requirements that are specific and involve a number of different forms of data, which is determined by how each requirement should be met.
- **Processing and Exploitation**
  After the collection step is completed, the information must go through processing and exploitation before it can be considered intelligence information. Conversion is an important part of this step and can include translations, decryption, and interpretation.
- **Analysis and Production**
  Analysis and production is a crucial step in the intelligence analysis process. This step includes the evaluation, integration, and analysis of all the intelligence data, which can consist of detailed reports as well as single-source and all-source studies.
- **Dissemination and Consumption**
  Dissemination is transferring information from producers to consumers. Consumption refers to how consumers interpret the information.
- **Feedback**
  Feedback is the dialog that takes place between the intelligence producers and consumers, which starts and continues after the information is received. Retrieved August 29, 2019, from https://www.securitydegreehub.com/what-is-intelligence-analysis/

3.0 METHODOLGY

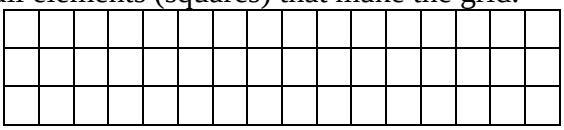
Many algorithmic strategies are recursive in nature. Solving a given problem requires recursively dealing with sub-problems using the minimum time and space.

In examining and analysing the different algorithmic operations and examining the algorithmic complexity in an array of data, the divide and conquer approach will be considered.

3.1 ARRAY

Arrays are classified as Homogeneous Data Structures because they store elements of the same type**.** They can store numbers, strings, boolean values (true and false), characters, objects, and so on. But once you define the type of values that your array will store, all its elements must be of that same type. It is not possible to "mix" different types of data.

To understand how array work, it is very helpful to visualize the computer memory as a grid, just like the grid in figure 1 below. Each piece of information is stored in one of the small elements (squares) that make the grid.



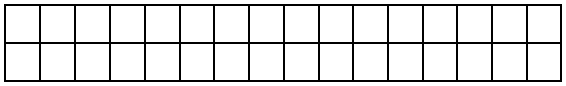


Figure 1: Grid

Arrays take advantage of grid structure to store lists of related information in adjacent memory locations to guarantee extreme efficiency for finding those values. Their elements are next to each other in memory**.** If there is a need to access more than one of element, the process is extremely optimized because the computer already knows where the value is located.

READING VALUES IN AN ARRAY

The amazing power of arrays comes from their efficiency to access values**.** When an array is created, the following actions takes place;

i. Assignment of values to a variable.
ii. Define the type of elements that it will store.
iii. Define its size (the maximum number of elements)

3.3 ARRAY REPRESENTATION

Array can be declared in various ways in different languages. For illustration, array can be declared as shown in figure 2 and 3 but figure 4 is not an acceptable elements representation in an array.

myArray =

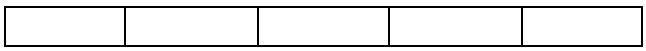


Figure 2: Array representation

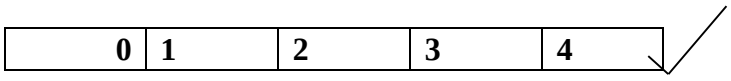


Figure 3: Array of elements

Not allowed!

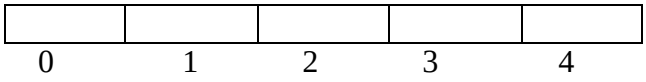
o	**"a"**	**"b"**	**5**	**true**

Figure 4: Array of mixed elements.

The name assigned to this variable is very important because it will be use later in the code to access values and to modify the array. But how can this access be possible by the computer for a particular value. This is where indices take a vital role.

3.3.1 INDEX

An "index" ("indices" in plural) is use to access a value in an array. This is a number that refers to the location where the value is stored. As shown in figure 5 below, the first element in the array is referred to using index 0. Moving further to the right, the index increases by one for each space in memory.



Indices

Figure 5: Array showing indices

3.3.2 ELEMENTS

Elements in an array are values or variables, each identified by at least one array index or key. Figure 6 shows the elements in the array. The array shown in figure 6 below is of length five (5) which means it can store five (5) elements. Each element can be accessed via its index.
There are three ways in which the elements of an array can be indexed:

i. **0** (zero-based indexing): The first element of the array is indexed by subscript of 0.
ii. **1** (one-based indexing): The first element of the array is indexed by subscript of 1.
iii. **n** (n-based indexing): The base index of an array can be freely chosen. Usually programming languages allowing n-based indexing also allow negative index values and other scalar data types like enumerations, or characters may be used as an array index.

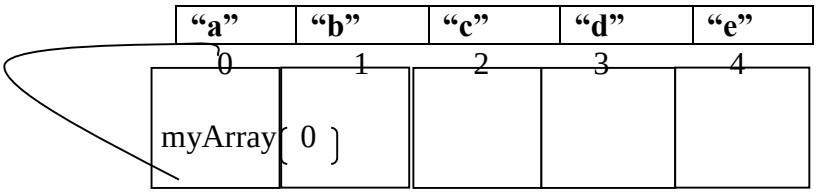


Figure 6: Array showing elements of alphabets

3.3.3 SYNTAX TO ACCESS AN ELEMENT IN ARRAY

The general syntax to access an element is: <ArrayVariable>[<index>]
Suppose an array is stored in the variable myArray**,** to access the first element (at index 0), the syntax to use would be**:** myArray[0]

4.0 IMPLEMENTATION

To solve a problem, different approaches can be followed. Some of them can be efficient with respect to time consumption, whereas other approaches may be memory efficient. However, one has to keep in mind that both time consumption and memory usage cannot be optimized simultaneously. If an algorithm must run in lesser time, more memory is required and vice versa.

4.1 THE ALGORITHMS

The proposed algorithms are surveyed in terms of these four operations:

(1) Search Operation.
(2) Deletion Operation.
(3) Insertion Operation.
(4) Update operation.

4.2 SEARCH OPERATION:

4.2.1 THE SEARCH ALGORITHM

It is the algorithmic process of finding a particular item in a collection of items. The search operation can be performed for an array of element based on its value or its index. To search an element in a given array, it can be done in the following ways: Sequential Search or Binary Search.

Consider **A** as an array with **N** elements and **K** is an element such that **K<=N**. the following is the algorithm to find an element with a value of ITEM using sequential search.

1. Start
2. Set J = 0
3. Repeat steps 4 and 5 while J < N
4. IF A[J] is equal ITEM THEN GOTO STEP 6
5. Set J = J +1
6. PRINT J, ITEM
7. Stop

The algorithm when compiled and executed, the following result is produced−

4.2.2. RESULT
The original array elements are:
A[0] = 1
A[1] = 3
A[2] = 5
A[3] = 7
A[4] = 8
Found element 5 at position 3

4.3. DELETION OPERATION
Deletion refers to removing an existing element from the array and re-organizing all elements of the array. In order to delete an element from an array, first the element from specified position must be eliminated and then shift remaining elements upwards to take vacant space of the deleted element.

4.3.1 DELETION ALGORITHM
Consider **A** as an array with **N** elements and **K** is an element such that **K<=N**. the following is the algorithm to delete an element available at the K^{th} position of A.
1. Start
2. Set J = K
3. Repeat steps 4 and 5 while J < N
4. Set A[J] = A[J + 1]
5. Set J = J+1
6. Set N = N-1
7. Stop
The algorithm when compiled and executed, it produces the following result −

4.3.2. RESULT
The original array elements are:
A[0] = 1
A[1] = 3
A[2] = 5
A[3] = 7
A[4] = 8
The array elements after deletion:
A[0] = 1
A[1] = 3
A[2] = 7
A[3] = 8

4.4. INSERTION OPERATION
Insert operation is to insert one or more data elements into an array. Based on the requirement, a new element can be added at the beginning, end, or any given index of array.
Insertion operation can take place in the following situations with an array−

- At the beginning of an array
- At the given index of an array
- After the given index of an array
- Before the given index of an array

4.4.1. INSERTION ALGORITHM
Let **A** be an Array (unordered) with **N** elements and **K** is an element such that **K<=N**. The following is the algorithm where ITEM is inserted into the K^{th} position of A −
1. Start
2. Set J = N
3. Set N = N+1
4. Repeat steps 5 and 6 while J >= K
5. Set A[J+1] = A[J]
6. Set J = J-1
7. Set A[K] = ITEM
8. Stop
When compiled and execute the above algorithm, it produces the following result −

4.4.2. RESULT
The original array elements are:
A[0] = 1
A[1] = 3
A[2] = 5
A[3] = 7
A[4] = 8

The array elements after insertion:
A[0] = 1
A[1] = 3
A[2] = 5
A[3] = 10
A[4] = 7
A[5] = 8

4.5. UPDATE OPERATION
Update operation refers to updating an existing element from the array at a given index.

4.5.1. UPDATE ALGORITHM
Consider **A** as an array with **N** elements and **K** is an element such that **K<=N**. The following is the algorithm to update an element available at the K^{th} position of A.

1. Start
2. Set A[K-1] = ITEM
3. Stop

The algorithm above when compiled and executed, produces the following result –

4.5.2. RESULT

The original array elements are:
A[0] = 1
A[1] = 3
A[2] = 5
A[3] = 7
A[4] = 8

The array elements after updating:
A[0] = 1
A[1] = 3
A[2] = 10
A[3] = 7
A[4] = 8

4.6. DISCUSION OF RESULT

Arrays are extremely powerful data structures that store elements of the same type. The type of elements and the size of the array are fixed and defined when created.

Once an array is created, memory is allocated immediately for elements to be located so that they can be accessed very efficiently using indices.

The experimental results on five intelligence analysis datasets indicate that all of the algorithmic strategies presented in this paper have the ability to generate some meaningful results for intelligence analysis.

i. The Search Algorithm:

The search algorithm seeks to find a particular element in a collection of items. The search operation is performed for the array of five elements based on index. After the execution of the algorithm, the search of element is done in a sequential order where element 5 in is found in index 3.

ii. The Deletion Algorithm:

The algorithm is to delete an element available at the K^{th} position of the array.(1,3,5,7,8) The result shows that the third element (5) after executing the deletion algorithm was deleted producing an array of four elements (1,3,7,8).

iii. The Insertion Algorithm

In this algorithm, the array of five elements (1,3,5,7,8) was increased by the insertion process. After the execution of the algorithm, the new array is (1,3,5,10,7,8) producing six (6) elements against the original five(5).

iv. The Update algorithm

The update algorithm is executed such that there is a replacement of an existing element in the array. The result after updating produced the array of (1,3,10,7,8) against the original array of (1,3,5,7,8). Here, the element 10 in inserted in the K^{th} position of the original array to produce (1,3,10,7,8).

5.0 CONCLUSION

A common source of difficulty in computing is the overload that occurs when the demand on working memory exceeds its capacity. One solution to this overload is to build strategies that decrease a task's demand on working memory. Arrays are extremely efficient in accessing values because all the elements are stored in contiguous spaces in memory. Array supports Random Access, which means elements can be accessed directly using their index, like arr[0] for 1st element, arr[4] for 5th element etc.

Hence, accessing elements in an array is **fast** with a constant time complexity of O(1).

Algorithmic strategies are essential component that automate individual steps in intelligence analysis in an array of element, most especially as regards data structures.

REFERENCES

Bier, E., Card S. & Bodnar J. (2010). Principles and Tools for Collaborative Entity-Based Intelligence Analysis. *TVCG*, 16(2):178–191.

Bier, E., Ishak E., & Chi E. (2006). Entity Workspace: An Evidence File That Aids Memory, Inference, and Reading. In *ISI '06*, pages 466–472.

Buczak, A. L., & Gifford, C. M Fuzzy (2010). Association Rule Mining for

Community Crime Pattern Discovery. In *ACM ISI-KDD '10*, pages 2:1–2:10.

Kang, H., Plaisant, C., Lee, B., & Bederson, B. B. NetLens (2007). Iterative Exploration of Content-actor Network Data. *Info.Vis.*, 6(1):18–31.

Lindahl, E., O'Hara, S., & Zhu, Q., A. (2007). Multi-agent System of Evidential Reasoning for Intelligence Analyses. In *AAMAS'07*, pages 279:1–279:6.

Pirolli, P., Card, S. (2005). The Sensemaking Process and Leverage Points for Analyst Technology as Identified through Cognitive Task Analysis. In *ICIA '05*.

Sfetcu, Nicolae. (2016). Cunoaștere și Informații. Nicolae Sfetcu

Skillicorn, D., B. (2010) Applying Interestingness Measures to Ansar forum Texts. In *ACM ISI-KDD '10*, pages 7:1–7:9.

Security Degree Hub. (2019). Retrieved August 28, 2019 from https://www.securitydegreehub.com/what-is-intelligence-analysis/

Study tonight (2019). Retrieved August 28, 2019 from https://www.studytonight.com/data-structures/space-complexity-of-algorithms.

Study tonight (2019b). Retrieved August 28, 2019 https://www.studytonight.com/data structures/time-complexity-of-algorithms.

Jigsaw: (2012). Visual Analytics for Exploring and Understanding Document Collections, http://www.cc.gatech.edu/gvu/ii/jigsaw/

i2group. The Analyst's Notebook. (2011). http://www.i2group.com/us.

PNNL. Pacific Northwest National Laboratory, INSPIRE Visual Document Analysis, 2011, http://in-spire.pnl.gov.