Global Journal of Computer Science and Technology: C Software & Data Engineering
Volume 23 Issue 2 Version 1.0 Year 2023
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

Literature Study on Analyzing and Designing of Algorithms

By Sneha Kumari & Aishwarya
Ajeenkya D.Y. Patil University

Abstract- The fundamental goal of problem solution under numerous limitations, such as those imposed by issue size, performance, and cost in terms of both space and time. Designing a quick, effective, and efficient solution to a problem domain is the objective. Certain problems are simple to resolve while others are challenging. To develop a quick and effective answer, much intelligence is needed. A new technology is required for system design, and the foundation of the new technology is the improvement of an already existing algorithm. The goal of algorithm research is to create effective algorithms that improve scalability, dependability, and availability in addition to cutting costs and turnaround times.

Keywords: *analysis, solution, time, algorithm, optimal, complexity, computing, application, space, design.*

GJCST-C Classification: *ACM: F.2.2, G.2.2*

Strictly as per the compliance and regulations of:

Literature Study on Analyzing and Designing of Algorithms

Sneha Kumari [α] & Aishwarya [σ]

Abstract- The fundamental goal of problem solution under numerous limitations, such as those imposed by issue size, performance, and cost in terms of both space and time. Designing a quick, effective, and efficient solution to a problem domain is the objective. Certain problems are simple to resolve while others are challenging. To develop a quick and effective answer, much intelligence is needed. A new technology is required for system design, and the foundation of the new technology is the improvement of an already existing algorithm. The goal of algorithm research is to create effective algorithms that improve scalability, dependability, and availability in addition to cutting costs and turnaround times.

Keywords: analysis, solution, time, algorithm, optimal, complexity, computing, application, space, design.

I. Introduction

Design and analysis of algorithms is referred to as DAA. It aids in the analysis of the answer prior to coding. Algorithms and documentation can be used to determine the space and time complexity. A clear image of the code you will write to address the problem is provided by algorithms and designs. It enables you to obtain the optimal time and spatial complexity for a shorter solution. The standards formeasuring algorithms before we can create effective ones. Algorithms are rated according to the amount of computing resources they need. The majority of these resources are running time and memory. Other factors may also be taken into consideration depending on the application, such as the volume of disc visits in a database programme or the amount of communication bandwidth in the networking application. The design of the algorithms must take into account a variety of challenges that arise in practice. Algorithms are instructions that you create in order to solve a complicated problem. You create these instructions by carrying out various computations, processing data, and scenario.

The methods are follows to solve a problem using descriptions of how to employ time and space resources are known as algorithms. Prior to implementing the actual code, you may use algorithms to learn more about the time and spatial complexity. Algorithms resemble technology in many ways. Although we all have the newest CPUs, we still need to run implementations of effective algorithms on that machine in order to get the full benefits of our investment in the most recent processor. When you develop the algorithms for the specific problem, you can determine the optimum solution. It is the most effective technique to illustrate any issue with the finest and most practical answers.

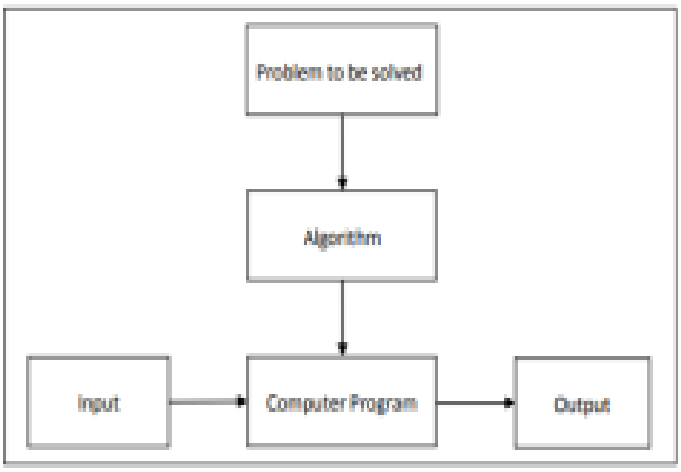

Fig. 1: The Notion of the Algorithm

Author α: MCA (Data Science) Ajeenkya D.Y. Patil University Pune, Maharashtra, India. e-mail: sneha.kumari@adypu.edu.in
Author σ: Ajeenkya D.Y. Patil University Pune, Maharashtra, India. e-mail: Aishwarya@inurture.co.in

II. Need for Algorithmic Problem

In computer science, algorithms are vitally significant.

- Acquire a thorough grasp of the problem.
- To identify the best answer to the problem.
- Allows for scalability. As it helps with comprehending, break the problem down into smaller steps.
- Being aware of the design guiding concepts and algorithms.
- Make use of the greatest technology to obtain the best and most effective solution.
- Getting thorough knowledge about the problem is impossible without implementation.

III. Procedural for Algorithmic Problem Solving

An initial input and a list of instructions are used by algorithms. The user's input, which may be expressed as words or numbers, is the first piece of information required to build judgements. The provided information is subjected to a series of computations, which may involve mathematical operations and moral assessments. The final step of an algorithm is called the output, and it is typically stated as more data. The picture below depicts the stages that go into creating and analyzing an algorithm.

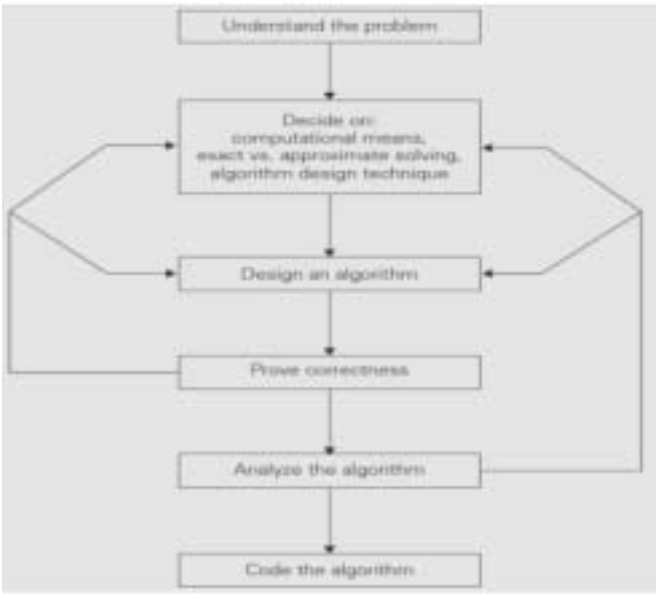

Fig. 2: Algorithm Design and Analysis Process [1]

a) Problem Recognition

Read the problem's description attentively to fully comprehend the problem statement; this is the 1st step in constructing an algorithm.

b) Making Decisions

Decisions are made based on the following:

a. Determining the Computational Device's Capabilities: In a RAM i; e random access machine, instructions are carried out one at a time (this is the underlying premise). As a result, algorithms created to run on these devices are known as sequential algorithms.
b. Selecting between exact and approximate problem-solving techniques. The choice between tackling the problem precisely or roughly is the next crucial option. An exact algorithm is one that solves a problem precisely and yields the desired outcome. When a problem is too complicated to have a precise solution, we must use a technique known as an approximation algorithm.
c. Techniques for designing algorithms: It is design methodology is a comprehensive approach for problem-solving that may be used to a variety of situations from different computing areas.

Programme = Algorithms + DS (Data Structures)

Although data structure & algorithm are separate concepts, programme is developed by combining them. Therefore, selecting the appropriate DS i; e data structures is necessary before constructing the algorithm. Algorithm implementation is only achievable with the aid of data structures and algorithms. Algorithmic strategy, methodology, and paradigm are a generic method for solving a variety of issues algorithmically. Examples include using "brute force," "divide and conquer," "dynamic programming," "greedy technique".

c) Methods for Specifying an Algorithm

An algorithm can be specified in three different ways. As follows: A flowchart, natural language &

pseudocode. The two methods that are most frequently used nowadays for describing algorithms are pseudocode and flowcharts.

Natural Language

Using plain language to describe an algorithm is really straightforward and simple. However, using normal language to describe an algorithm is not always straightforward, thus we only obtain a brief definition.

Pseudocode

It combines elements of normal language with those of programming languages. Natural English is frequently less exact than pseudocode.

A flowchart

Flowcharts were formerly the standard for expressing algorithms in the early days of computers, but this way of representation has since proven to be inconvenient. An algorithm is graphically represented by a flowchart. It is a way of representing an algorithm using a network of linked geometric forms that each include descriptions of an algorithm step.

d) Proving the Accuracy of an Algorithm

An algorithm's correctness must be established once it has been stated. An algorithm must provide a needed result in a finite period of time for each valid input.

e) Analysis of an Algorithm

The most crucial factor for an algorithm is efficiency. There are actually 2 types. They are efficiency in time, which measures how quickly the algorithm executes, and efficiency in space, which measures how much more memory it consumes. Therefore, the following criteria should be considered while analyzing an algorithm time efficiency, space efficiency, simplicity, and generality.

f) Code for Algorithm

An appropriate programming language is used to code or implement an algorithm. It is possible to make the conversion from an algorithm to a programme improperly or extremely inefficiently. An algorithm must be appropriately implemented. Writing efficient, optimized code is crucial if you want to lighten the load on the compiler.

IV. Algorithm Characteristics

Each algorithm should possess the following six essential characteristics:

A) Input-One or more inputs may be present in an algorithm. The inputs are taken from a predetermined group of participants. Any form of file can be entered, including text, pictures, and images.

B) Output- It can provide one or more results i; e. output. It is essentially number that has a predefined relationship with the input.

C) Finiteness-An algorithm should end after a finite number of steps, then only it considers as computational method.

D) Certainty- An algorithm must have every step well described. For each scenario, the action that has to be taken must be vaguely described. Because the step is difficult to grasp, one would assume that it lacks definiteness. As a result, mathematical expressions are expressed in these situations in a way that is similar to how instructions are written in a computer language.

E) Efficiency- An algorithm is typically assumed to be efficient. means that the processes should be sufficiently simple that a man might be able to solve them.

F) Language Independence - An algorithm type should be languages-independent, meaning that its instructions or commands must function consistently regardless of the language in which they are implemented.

V. Guidelines to Be followed for Development of Algorithm

The following guidelines must be adhered to while developing an algorithm:

- An algorithm will be surrounded by the symbols START (or BEGIN) and STOP (or END).
- The words OBTAIN, GET, READ, and INPUT are frequently employed to accept data from users.
- To display results or messages, statements like WRITE, PRINT, and DISPLAY are frequently used.
- Mathematical expressions are often denoted by the terms CALCULATE or COMPUTE and depending on the situation, appropriate operators may be used.

VI. Anaysis of an Algorithm & its Method

A method for evaluating an algorithm's performance is algorithm analysis. The time and spatial complexity are the main variables on which the algorithms rely. Two algorithms are examined using asymptotic analysis to see how well they perform when the input size is changed (increased or reduced).

i. Worst Case Analysis- It is the algorithm's worst case, or the circumstance that causes the majority of operations to be carried out must be understood. In the worst case, we are able to determine an algorithm's upper bound running time. When the sought-after element (x) is not present in the array, linear search experiences its worst-case scenario. The search () function checks each member of arr[] independently if x is absent. The worst-case temporal complexity of the linear search would thus be O(n). The worst-case situation is represented by the Big O notation. Only the upper bound of time is computed when the procedure is applied.

ii. Best Case Analysis- The scenario in which an algorithm is run with the fewest number of operations is known as its "best case." It establishes the algorithms lower bound for execution time in the best-case scene. It is necessary to understand the situation that just executes a few activities. When x appears at the first position, the best case for the linear searching problem happens. In the best situation, the number of processes is fixed and independent of n. Thus, for time complexity, (1) is the best-case situation. The best scenario is represented by Omega notation. When the method is used, just the lowest bound of time is calculated.

iii. Average Case Analysis-It is an algorithm, which is the scenario in which the algorithm becomes aroused after a few operations. For example, when we execute a linear search technique in any data structure and find an element at the midway place, that scenario is referred to as the average case. When studying typical instances, Theta Notation is used. It establishes the complexity of time with the aid of the upper and lower bounds.

VII. Advantages of an Algorithm

1. Since it shows a step-by-step approach to solving a particular problem, it is easy to understand.
2. An algorithm executes a predefined procedure.
3. Because each step has its own logical sequence, an algorithm is easy to debug.
4. An algorithm is used to divide the problem into smaller components or stages, which makes it simpler for programmers to turn the problem into usable software.
5. It is independent what programming language is used.

VIII. Disadvantages of an Algorithm

1. Algorithms take a lot of time.
2. It's challenging to demonstrate looping and branching in algorithms.
3. Big challenges are hard to describe and even more challenging to write algorithms for.

IX. Types of Algorithms

a) Brute Force Algorithm

The most fundamental algorithm that may be developed to address a problem is of this type. We must first identify at least one answer before attempting to enhance it in order to create the optimal one. The most simple and fundamental algorithm is one of them. Any problem can be solved using the brute force approach; however, it often doesn't add much time or space complexity.

b) Recursive Algorithm

It is the easiest algorithms to create since it doesn't need to individually consider each sub problem. When the problem scale is substantially decreased, the recursive algorithm procedure converts the problem into a smaller scale but similar form problem, which is then solved. Among these, the basic issue-solving process is characterized by self-reference at the level of recursive description, where the scenario and approach that may solve the problem directly are defined. Similar to mathematical induction, the fundamental concept of recursive process description involves quoting oneself in order to minimize the complexity of the problem. Recursion is a very efficient approach, but since it calls a recursive stack each time the recursion function is invoked, memory management must always be kept in mind. When the complexity is reduced to a given extent, the problem is then directly solved.

i. *Divide and Conquer Algorithm*

This is one of the techniques that programmers utilize the most. With this approach, the problems are divided into smaller ones, each of which is solved independently, before the combined solutions are used to determine the solution to the initial challenges. As it is relatively stable and ideal for the majority of the challenges posed, this algorithm is widely employed in a variety of problems. When deciding how to address a problem, the divide-and-conquer tactic is widely used. Strassen's Matrix Multiplication, Merge Sorting, Binary Search, Quick Sorting, etc. are a few typical issues that are resolved utilizing Divide and Conquer algorithms.

Fig. 1: Divide and Conquer Algorithm [15]

ii. *Dynamic Programming Algorithms*

This type of algorithm is most efficient ways of problem solutions, this algorithm is the most popular. This technique is very efficient in terms of time complexity since it just requires recalling previous results and applying it to future results that correspond. Since this type of procedure maintains the previously computed answer in order to avoid having to compute it repeatedly, it is also known as the recalled technique.

There are two versions of this algorithm:

Bottom-Up Approach: This method begins by resolving the smallest feasible subproblems first, building on the answers obtained from those subproblems to solve the larger problem.

Top-Down Approach: This method begins by resolving all of the problems until it reaches the necessary subproblem, which is then addressed utilizing subproblems that have already been resolved.

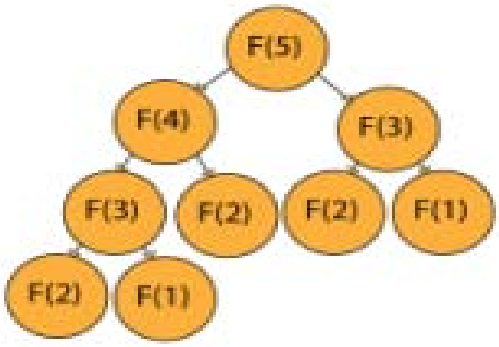

Fig. 2: Fibonacci Series in Dynamic Algorithm [12]

iii. *Greedy Algorithms*

This algorithm does not consider the future while making decisions; instead, it considers the situation at hand. It doesn't matter if the best outcome at the moment leads to the best result altogether. A greedy algorithm gradually assembles an approach, always choosing as the next step the element that offers the most apparent and immediate advantage. Greedy so works well with problems when choosing locally optimal also leads to a global solution. Although the algorithm that is greedy is not consistently successful, when it exists, it is fantastic! This method is typically the simplest since it is easy to develop. It's probable that this method won't work for all problems. But, if the problem has any of the following characteristics, we can decide if this approach can be applied to any of the problem cases.

A Greedy Property Choice

A greedy method can be used to tackle a problem if it is possible to make the best or most advantageous option at each stage without going back and changing the decision made at the prior stage. The "greedy choice property" is the name given to this trait.

The Runner-Up Substructure

If the most effective solution to the challenge is also the most effective solution to each of its subproblems, the problem can be solved using a greedy approach.

This trait is known as "optimal substructure". It achieves this because it is continuously working to get the best result possible locally. Examples of common cases or problems that the Greedy Algorithm solves includes the Kruskal's Algorithm, Prim's Algorithm, Dijkstra Shortest Path Algorithm, Huffman Coding, and others.

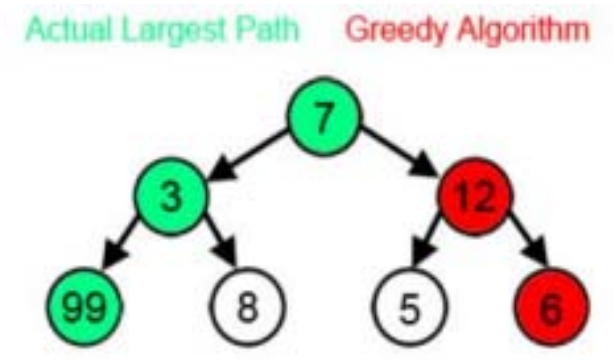

Fig. 3: Greedy Algorithm [15]

Backtracking Algorithms

It is based on a depth-first recursive search. It is an improvement over using raw force. Here, we choose one choice from the many that are available and try to solve the problem. The Brute force approach, which evaluates each potential answer, is used to choose the desired/best solutions. It is an algorithmic method for recursively addressing problems. The Backtracking Algorithm may be used to solve problems like the Hamiltonian Cycle, Rat in Maze Problem, the N Queen Problem, the M-Coloring Problem, etc.

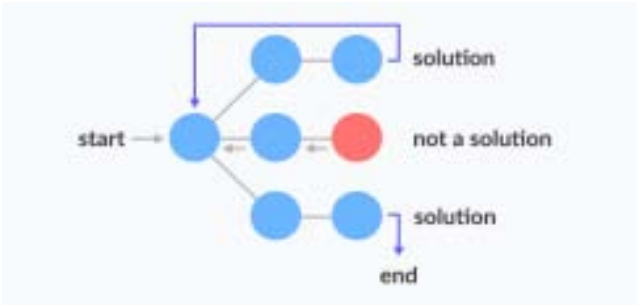

Fig. 4: Backtracking Algorithm [13]

c) Randomized Algorithm

This sort of algorithm bases its conclusions on random numbers, i.e., it incorporates random numbers into its reasoning. Selecting the desired result is helpful. the process of choosing a number at random that offers an instant benefit. One of the problems that the randomized Algorithm could fix is quicksort. The pivot in Quicksort is selected at random.

d) Searching Algorithm

A searching algorithm is a method for finding a certain key among a group of sorted or unordered data. A number of problems may be solved using the searching algorithm, such as the following: Binary search, sometimes referred to as linear search, is one form of search technique.

X. Conclusion

Algorithms may be used by both individuals and machines to carry out routine activities. The primary distinction is that computers employ algorithms far more quickly and effectively than we can. A series of actions used to solve a problem is called an algorithm. In the field of information technology and computer science, building algorithms to tackle various sorts of problems requires careful planning and analysis. A problem that needs to be solved initiates the process of designing an algorithm, which is then followed by the classification of the problem's type into the categories listed above, the implementation of the algorithm, and finally an evaluation of the finished algorithm's efficiency (both in terms of time and space). The computer theory of complexity, which offers a theoretical estimate of the resources needed for an algorithm to effectively address a certain computer issue, includes algorithm analysis as a key component. Analysis is used to calculate how much space and time are needed to run a programme. Applying various algorithmic design techniques, such as divide-and-conquer, greedy, and others, to real-world issues. The capacity to comprehend and calculate the algorithm's performance. Algorithms are frequently simple to design, simple to implement, and quick to execute. Insidiously difficult mathematical proofs may be needed to demonstrate their correctness.

References Références Referencias

1. https://www.brainkart.com/article/Algorithmic-problem solving_35898/.
2. Thomas H Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. Introduction to algo-rithms, third edition. pp 360- 395.
3. Brassard, G. and Bratley, P. Fundamental of Algorithm- mics, Prentice-Hall, 1996.
4. Kashale Chimmanga, Josephat Kalezhi and Phillimon Mumba, “Application of best first search algorithm to demand control”, 2016 IEEE PES Power Africa Conference,pp. 51-55,IEEE 2016.
5. Sankar Peddapati and K.K. Phanisri Kruthiventi, ”A New Random Search Algorithm: Multiple Solution Vector Approach”, 2016 6th International Advanced Computing Conference,pp.187-190, IEEE 2016..
6. D. E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, New York: Addison-Wesley, 1989.
7. J. Cioffi, "The block-processing TF adaptive algorithm," IEEE Trans. Acoust., Speech, Signal Processing, vol. A SSP-34, no. 1, pp. 77-90, 1986. .
8. Ziewitz M (2015) Governing algorithms: Myth, mess, and methods. Science, Technology & Human Values 41(4): 3– 16.
9. M. Young, The Technical Writers Handbook, Mill Valley, CA: University Science, 1989.
10. Jiang Na, Yang Haiyan, Gu Qingchuan, Huang Jiya. Machine learning and its algorithm and development analysis [J]. Information and Computer Science (Theoretical Edition), 2019 (01): 83-84 + 87.

11. Montazeri and P. Duhamel, "A set of algorithms linking NLM S and RLS algorithms," in Proc. EU SP ICO-94, 1994, vol. 2, pp. 744-747.
12. https://stackabuse.com/dynamic-programming-in-java/.
13. https://www.programiz.com/dsa/backtra-cking-algo ithm.
14. https://favtutor.com/blogs/divide-and-conquer-algo rithm.
15. https://vikram-bajaj.gitbook.io/cs-gy-6033-i-design-and analysis-of-algorithms-1/chapter1.