IJIRSET
International Journal of Innovative Research in
SCIENCE | ENGINEERING | TECHNOLOGY

e-ISSN: 2319-8753 | p-ISSN: 2347-6710

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN SCIENCE | ENGINEERING | TECHNOLOGY

Volume 11, Issue 6, June 2022

ISSN INTERNATIONAL STANDARD SERIAL NUMBER INDIA

Impact Factor: 8.118

9940 572 462 | 6381 907 438 | ijirset@gmail.com | www.ijirset.com

International Journal of Innovative Research in Science, Engineering and Technology (IJIRSET)

| e-ISSN: 2319-8753, p-ISSN: 2320-6710| www.ijirset.com | Impact Factor: 8.118|

||Volume 11, Issue 6, June 2022||

| DOI:10.15680/IJIRSET.2022.1106176 |

Application of Algorithms

Aryaman Yaduvanshi [1]

Student, Delhi Technological University, New Delhi, India[1]

ABSTRACT: Computer programs can be used to develop computer algorithms. They are then run on an acceptable set of inputs to determine each program's requirement of each resource. Unfortunately, for four reasons, this technique is frequently unsatisfactory:

1. Selecting one algorithm and assessing the time and effort required to build and test two algorithms.
2. When contrasting two algorithms empirically, there's a probability of one being better written than the other. Resultantly, the implementations do not accurately reflect the relative strengths of the underlying algorithms. This frequently happens when the programmer has a preconceived notion about the algorithms.
3. Empirical test scenarios may favor one algorithm over another.
4. The two good algorithms do not fit the budget.

On the other hand, computer algorithms will be an essential aspect of computer science and become increasingly vital as computers solve increasingly challenging issues.
This literature review covers how algorithms work, the development and demonstration of algorithms, and their use in real life. Algorithms incredibly manage the daily tasks we perform on our gadgets.

KEYWORDS: Computer programs, Computer algorithms, Set of inputs, Resource, Technique, Programmer, Empirical test scenarios, Computer science, Sorting algorithms, Optimizing approximation algorithms

I. INTRODUCTION

All computer operations are algorithms, for example, completing a report after writing a paragraph, conducting a spell check, and opening a spreadsheet to perform financial forecasts to see if a new automobile loan is within the budget. Then, researching to know which car to purchase.
An algorithm (Katoch, Chauhan, & Kumar, 2021) is a set of rules used to find solutions or perform tasks. A series of clear rules is another approach to defining an algorithm. The word 'clear' implies some space for personal interpretations. A computer performs the same algorithm and provides the same output upon providing the same input. A methodology or a numerical process for problem-solving and engineering algorithms is another definition of the algorithm. Many operation research approaches like divide-and-conquer and dynamic programming include algorithms. Algorithm design patterns are techniques for developing and implementing algorithms, with examples, including the decorator and the template method patterns.
Resource (runtime, memory use) efficacy is an essential element of an algorithm. The big O notation indicates, for example, an algorithm's runtime growth as the input size grows.
As per the previous instance, algorithms check spelling and financial calculations. The uses don't end here. Search engines use algorithms. It's impossible to imagine a computer task that doesn't involve the employment of algorithms.

II. WORKING OF ALGORITHMS

Algorithms' working help identify the greatest number out of five million numbers that aren't ordered. We will need a computer, which needs an algorithm to do this.
Here's an example of what the algorithms can look like.
Let's imagine the input is a list of numbers, referred to as L. The first number on the list would be L1. The second number would be L2, and so on. We also know the list isn't sorted; otherwise, the solution would be obvious. As a result, the algorithm's input is a set of numbers, and the algorithm's output is the greatest number in the set.

This is how the algorithm would look-alike:

Step 1: Define the largest number as L1
Begin by thinking the first number is the largest.

| e-ISSN: 2319-8753, p-ISSN: 2320-6710| www.ijirset.com | Impact Factor: 8.118|

||Volume 11, Issue 6, June 2022||

| DOI:10.15680/IJIRSET.2022.1106176 |

Step 2: For each item that is there in the list
Withstand the number list one after the other.

Step 3: If the item is > greatest, go to the next step
Go to step four to find the new biggest number. If this is the case, return to step two and move on to the following number on the list.

Step 4: After that, the largest item
This changes the entire greatest number with the newly discovered largest number. Return to step two until the list no longer contains any numbers.

Step 5: Return the largest item
This yields the intended outcome.
It's worth noting that the algorithm (Khishe, & Mosavi, 2020) is explained as a set of logical stages in plain English. These instructions must be written in a programming language that a computer can understand and use.

III. DEVELOPING ALGORITHMS

Computer algorithms are necessary to process data. For example, most computer programs (Sinaga, & Yang, 2020) include algorithms, which specify the instructions that a computer should follow in a specific order to complete a task, such as calculating employee salaries or printing kids' report cards. Hence, any action that a Turing-complete system mimics is an algorithm (in computability theory, Turing reduction is derived from a decision problem).
Turing machines (a mathematical formalism of computing that describes an abstract machine) can define non-terminating computational tasks. However, an informal definition of algorithms, such as "a series of rules that accurately describes a list of steps," often demands that they always end. Due to the halting problem, a fundamental theorem of computability theory, the process of determining if an algorithm is possible in the general situation.
Data is read from an input, written to an output device, and stored for further processing when an algorithm is connected with information processing. The entity conducting the algorithm considers the data stored as a part of its internal state. In practice, the storage is done in one or more data structures.
The algorithm must be precisely established for some computational operations: it must explain how it works in all potential scenarios. Moreover, all conditional procedures must be handled in a systematic, particular instance manner, with defined parameters for each case (and computable).
The computation order is always essential to the algorithm's operation since an algorithm is a precise set of precise steps. Therefore, instructions are typically supposed to be listed openly and defined as beginning "from the above to the bottom"—a more technically articulated concept by the flow of control.
The premises of imperative programming have been taken to formalize an algorithm. The most widely held belief is that a task can be described in discrete, "mechanical" terms. The allocation procedure, which sets the value of a variable, is unique to this paradigm of formalized algorithms. It's based on the idea of "memory" as a scratchpad.

IV. DEMONSTRATING ALGORITHMS

Natural language, flow charts, pseudocode, and programming languages (Khaire, & Dhanalakshmi, 2019) can illustrate an algorithm.

- **Natural language:** A common choice to communicate algorithm steps to a mass audience. When we're building algorithms, we frequently collaborate with programmers and nonprogrammers—but they all understand natural language.
  Natural language has several disadvantages. Because of no imposed structure, it tends to be unclear and poorly defined. Others will consequently struggle to understand the algorithm and be confident in its accuracy. Computer scientists and programmers prefer flow charts and pseudocode because they are more structured representations that can better express an algorithm.
- **Flow charts:** Represents an algorithm more formally and are depicted as arrows connecting the boxes. The arrows flow from one step to the next, and each rectangle represents a phase in the sequence.
  Flow charts help visualize it at a high level and push us to think carefully about sequencing and selection. For example, what arrow leads to which node? Are there any arrows missing? These problems arise when

| e-ISSN: 2319-8753, p-ISSN: 2320-6710| www.ijirset.com | Impact Factor: 8.118|

||Volume 11, Issue 6, June 2022||

| DOI:10.15680/IJIRSET.2022.1106176 |

converting an algorithm toa flow chart.

- **Pseudocode:** Most algorithms are turned into code that can be executed on a computer. Before then, programmers frequently choose to express algorithms via pseudocode.
  Because there is no official standard for pseudocode, each programmer writes it individually. Therefore, encounteringcompletely different pseudocode.
  Pseudocode allows a coder to think about familiar concepts without worrying about syntax or intricacies. It also provides a language-independent means for computer scientists to explain an algorithm, allowing programmers fromany language to join, read the pseudocode, and convert it into their preferred language.
- **Programming languages:** The algorithm should be put into running code upon planning an algorithm inoneor a combination of the above methods.
  Any programming language can translate an algorithm. The software is written in several programming languages depending on what it does and who does it. Interestingly, we can apply the same algorithms everywhere!

V. ANALYSIS OF ALGORITHMS

Theoretically, understanding the requirement of given resources (like time and storage) for a given algorithm (Kearns, & Roth, 2019) can help achieve quantitative responses (estimates), methods for analyzing algorithms have been created.For example, a big O notation probably amounting to the elements in a list of 'n' numbers would require some O(n) time. However, the algorithm only has to retain two values at a time: the sum of all previous elements and its current position in the input list. As a result, it has an O(1) space demand if the space required to store the input integers is not counted and an O(n) space requirement if it is counted.

- ***Formal versus empirical:*** Research and algorithm analysis falls under computer science topic that is frequently done abstractly without a specific programming language or implementation. In this way, algorithmanalysis is similar to other mathematical disciplines that concentrate on the algorithm's basic features rather than the intricacies of any given implementation. It is the most straightforward and most general representation. Pseudocode is frequently used for analysis. However, most algorithms are finally implemented on specific hardware/software, and their algorithmic efficiency is tested using actual code.

The efficiency of a specific algorithm may not have substantial ramifications for the solution of a "one-off" problem (unless n is exceedingly large). Still, it may be crucial for algorithms built for fast, interactive, commercial, or long-term research use. When expanding from a small n to a high n, inefficient methods that are otherwise acceptable are frequently exposed.

Empirical testing is beneficial because it can reveal previously unknown correlations that influence performance. In addition, benchmarks can help compare proposed algorithm improvements before and after software optimization. On the other hand, empirical testing cannot substitute formal analysis and are not easy to conduct fairly.

- ***Execution efficiency:*** To demons(widely used in image processing) can reduce processing time for applications such as medical imaging by up to 1,000 times. Speed increases depending on the problem's unique features, frequent in real-world applications. Computing devices requiring a lot of image processing, such as digital cameras and medical equipment, can save a lot of electricity with these speedups.

VI. IMPORTANCE OF ANALYSING ALGORITHMS EFFICIENCY

Algorithmic efficiency (Bambauer, & Zarsky, 2018) is the algorithm's feature connected to many computational resources that the algorithm requires in computer science. Examination of algorithm efficiency depends on the usage of various resources and the analysis of resource use. Algorithmic efficiency is analogous to engineering performance for a recurrent or continuous process. To increase efficiency, use as few resources as feasible. Because different resources like space and time efficiency cannot be directly compared, determining which proposed algorithm is more efficient often depends on which efficiency indicator is regarded more important. Two factors determine the efficiency of an algorithm:

- **Memory efficiency:** The quantity of memory/space consumed in particular circumstances must be checked. For example, memory consumption must be addressed when working with large volumes of data or programming embedded systems. The following are the components of memory/space usage:

1. **Instruction space:** Target machine (CPU), the compiler, the compiler settings affect instruction space.
2. **Dataspace:** The size of the data and allocated memory and the number of static program variables affects data

| e-ISSN: 2319-8753, p-ISSN: 2320-6710| www.ijirset.com | Impact Factor: 8.118|

||Volume 11, Issue 6, June 2022||

| DOI:10.15680/IJIRSET.2022.1106176 |

space.

3. **Stack space at runtime:** Stack space at runtime is affected by the runtime function calls and recursion, local variables, compiler, and arguments.

- **Time efficiency:** The sooner a program/function achieves its goal, the better is the algorithm. Various factors influence the running time:
  1. **Processor speed** (rather than just clock speed), I/O, etc.
  2. **Compiler** refers to both the compiler and the compiler settings.
  3. **Data size**, such as whether to search a long or shortlist.
  4. **Actual data** to find whether a name is first or last in a search.

VII. REAL-LIFE USES OF ALGORITHM

- ***Applications of array***
  1. Arrange data sequentially (Yuan, & Yang, 2019), such as on our phones for storing contacts, speech signals in speech processing, etc.
  2. Stack and Queue implementation, hashtags and heaps implementation, and adjacency matrix representation of graphs.

- ***Applications of linked list***
  1. Keeping a directory of names, computing polynomials, establishing Stack and Queue, adjacency list representation of graphs, and so on are all examples of **singly-linked lists.**
  2. Using a **doubly-linked list** to illustrate a deck of cards, implement (Faraj, Pachidi, & Sayegh, 2018) undo and redo capability in apps, navigate to the previous and next pages in a web browser, etc.
  3. A **circular linked list** to schedule processes in an operating system or monitor turns in a multi-player game.

- ***Applications of stack***
  1. UNDO/REDO (Washington, 2018) options in a text editor, saving browser history
  2. Scheduling processes
  3. Allocate static memory
  4. In recursion, storing function calls
  5. To detect missing braces in a compiler or IDE
  6. Evaluating Syntax Parsing Expression

- ***Applications of queue***
  1. Process scheduling in operating systems (CPU and IO scheduling)
  2. Call management for customer service
  3. Communication between processes
  4. Applications with a waiting list

- ***Applications of hashing***
  1. Using terms in a search engine to find webpages
  2. Phone numbers in Android phones, iPhones, and Employee Information System
  3. Encrypting vital information (Cryptography)
  4. In word processing software, there are spell checkers.
  5. A compiler's symbol table

- ***Applications of tree data structure***
  1. Linux uses a **binary tree** to support hierarchical data such as folder structure, organizational structure, and filesystems in the Document Object Model.
  2. **Binary search trees** are used in applications (Rahkar Farshi, 2021) where data is constantly inserted and deleted and in language libraries to construct maps and set objects.
  3. Autocompletion in Google search, contact search in a phone book, swipe features in mobile keypads, spell checking in word processors, etc., are all **trie data structures.**
  4. In operating systems, **heap data structures** are used to construct efficient priority queues, order statistics, scheduling processes, and dynamic memory allocation.

| e-ISSN: 2319-8753, p-ISSN: 2320-6710| www.ijirset.com | Impact Factor: 8.118|

||Volume 11, Issue 6, June 2022||

| DOI:10.15680/IJIRSET.2022.1106176 |

5. **Advanced data structure** (B-tree), 3D computer graphics (BSP tree), and quick full-text search in word processors are all examples of advanced data structures (Suffix tree)

- ***Application of graph data structure***
1. **Shortest path algorithms** like robot path planning and google maps.
2. **Breadth-first search (BFS)** is used in social networking sites to find people within a certain distance k from aperson and GPS tracking to find nearby places.
3. **Depth-first search (DFS)** to discover cycles in a graph and solve maze puzzles.
4. **Topological sorting** is used to identify the order in which complicated database tables with interdependenciesshould be created.
5. Google Knowledge Graph and Facebook Graph Search **are two other significant applications.**

- ***Applications of dynamic programming***
1. Sequence Alignment and Plagiarism detection
2. Duckworth Lewis Method in Cricket
3. Flight Control
4. Speech Recognition
5. Image processing
6. Machine learning algorithms
7. Economics
8. Financial Trading
9. Operations Research
10. Bioinformatics

- ***Applications of greedy algorithms***
1. Compression of.png and.mp3 files without loss of quality
2. Kruskal and Prim's algorithms for minimum spanning tree
3. Dijkstra Algorithms for Finding the Shortest Path
4. Approximation algorithms for NP-hard problems
5. Cracking activity choice and other optimization problems

- ***Applications of backtracking***
1. Verbal arithmetic, crosswords, verbal arithmetic, Sudoku, and N-queens are some of the most well-known puzzles.
2. Solving the restraint fulfillment problem and various optimization techniques.

- ***Applications of number theory***
1. Computer graphics
2. Cryptography
3. Designing hash functions
4. Fast arithmetic operations
5. Random number generators

- *Other applications of algorithms and data structures*
1. Load Balancing (Randomized Algorithms)
2. Image editing software like Photoshop (Convex-hull algorithms)
3. Exact String Matching (KMP Algorithms)
4. Filter out stories that people have never seen before (Quora uses bloom factor for this)
5. Fast search in a sorted dataset (Binary search)
6. Block-sorting compression (Suffix Array)

Breaking down signals into frequencies (Fast Fourier Transform)

VIII. CONCLUSION

Every task performed on a computer deploys an algorithm. Usage of algorithms is vast and for a good reason, such as

| e-ISSN: 2319-8753, p-ISSN: 2320-6710| www.ijirset.com | Impact Factor: 8.118|

||Volume 11, Issue 6, June 2022||

| DOI:10.15680/IJIRSET.2022.1106176 |

completing days and months of efforts in a few minutes. It is fantastic how to compute algorithms, process and manage a vast amount of data. Fortunately then, there are numerous data structures to store that data. Natural language, flow charts, pseudocode, and programming languages are methods to depict algorithms. Each method has its reasons for use. Analyzing algorithms are helpful from the memory and time efficiencies point of view, and resources contribute to this matter.

REFERENCES

1. Bambauer, J., & Zarsky, T. (2018). The algorithm game. *Notre Dame L. Rev.*, *94*, 1.
2. Faraj, S., Pachidi, S., & Sayegh, K. (2018). Working and organizing in the age of the learning algorithm. *Information and Organization*, *28*(1), 62-70.
3. Katoch, S., Chauhan, S. S., & Kumar, V. (2021). A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications*, *80*(5), 8091-8126.
4. Kearns, M., & Roth, A. (2019). *The ethical algorithm: The science of socially aware algorithm design*. Oxford University Press.
5. Khaire, U. M., & Dhanalakshmi, R. (2019). Stability of feature selection algorithm: A review. *Journal of King Saud University-Computer and Information Sciences*.
6. Khishe, M., & Mosavi, M. R. (2020). Chimp optimization algorithm. *Expert systems with applications*, *149*, 113338.
7. Rahkar Farshi, T. (2021). Battle royale optimization algorithm. *Neural Computing and Applications*, *33*(4), 1139-1157.
8. Sinaga, K. P., & Yang, M. S. (2020). Unsupervised K-means clustering algorithm. *IEEE access*, *8*, 80716-80727.
9. Washington, A. L. (2018). How to argue with an algorithm: Lessons from the COMPAS-ProPublica debate. *Colo. Tech. LJ*, *17*, 131.
10. Yuan, C., & Yang, H. (2019). Research on K-value selection method of K-means clustering algorithm. *J*, *2*(2),226-235.

INNO SPACE
SJIF Scientific Journal Impact Factor
Impact Factor: 8.118

doi cross ref

ISSN INTERNATIONAL STANDARD SERIAL NUMBER INDIA

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN SCIENCE | ENGINEERING | TECHNOLOGY

9940 572 462 6381 907 438 ijirset@gmail.com

Scan to save the contact details