Introduction to Algorithms

February 12, 2024

1 Introduction

Algorithms are fundamental to maximizing the benefits of advanced computer hardware. Just like having the latest and most powerful processors, the efficiency of algorithms determines the overall performance and effectiveness of computational tasks. In this lesson, we will explore the significance of implementing efficient algorithms, using a practical example to illustrate their impact on computing systems.

2 Definitions

2.1 Algorithms in General

An algorithm is a step-by-step procedure or set of rules designed to solve a specific problem or perform a particular task. It provides a systematic approach to problem-solving and is applicable across various fields. Here are two examples of algorithms in general:

1) **Algorithm:** Making a Cup of Tea
    a) Add water to the kettle
    b) Boil it
    c) Add tea leaves
    d) Add Milk
    e) Add sugar
2) **Algorithm:** Withdrawing Money from an ATM
    a) Insert the card
    b) Enter the PIN
    c) Enter the amount
    d) Withdraw amount
    e) Check balance
    f) Cancel/clear

2.2 Computer Algorithms

A computer algorithm is a step-by-step procedure or set of rules designed to solve a specific problem or perform a particular task, implemented for execution on a computer. It provides a systematic approach to writing computer programs and is crucial for software development. Here are five examples of computer algorithms:

1) **Quicksort:** An efficient sorting algorithm, commonly used in computer science.
2) **Depth-First Search (DFS):** An algorithm for traversing or searching tree or graph data structures.
3) **Merge Sort:** A divide-and-conquer algorithm for sorting lists.
4) **PageRank Algorithm:** Used by search engines to rank web pages in their search results.
5) **RSA Algorithm:** A public-key cryptosystem algorithm used for secure communication.

3 Characteristics of an Algorithm

Before delving into the importance of efficient algorithms, let's review the key characteristics that define an algorithm:

a) **Input and Output:** An algorithm must take input and produce some output, which could be in the form of true/false, a specific value, etc.

b) **Definiteness:** Each instruction in the algorithm must be clear and unambiguous.

c) **Finiteness:** The algorithm must terminate after a finite number of steps.

d) **Effectiveness:** Every instruction in the algorithm must be basic, constituting simple and executable steps.

4 Efficiency in Algorithm Design

Efficiency in algorithm design is crucial for utilizing the full potential of computer hardware. While powerful processors provide raw computing power, the choice of algorithm significantly influences the execution time and resource utilization. A well-designed algorithm can substantially enhance performance and optimize resource consumption.

4.1 Resource Usage

Resource usage is a critical aspect of algorithm design, primarily focusing on time and space considerations.

4.1.1 Aposteriori Analysis

Aposteriori analysis involves implementing the algorithm and then measuring the time taken by the system to successfully execute the program. Practical considerations, such as the speed of the computer, the programming language used, the compiler or interpreter, and the skill of the programmers, can impact the actual running time.

4.1.2 Apriori Analysis

Apriori analysis evaluates the algorithm's running time growth rate before implementation. This includes:

1) Determining how long the algorithm takes, represented as a function of the size of the input, denoted as $f(n)$ where n is the size of the input.

2) Analyzing the growth rate of the running time function with respect to the input size, known as the "running time growth rate."

Algorithms are compared based on their running time growth rates, with those exhibiting lower growth rates considered more favorable. This approach allows for a proactive assessment of an algorithm's efficiency before practical implementation, taking into account the algorithm's characteristics and its anticipated performance across varying input sizes.

5 Practical Example

Consider two computers: computer A, which is faster and executes 10 billion instructions per second, and computer B, which is slower, running at 10 million instructions per second. Both computers are tasked with sorting an array of 10 million numbers.

5.1 Algorithm Analysis

- **Computer A (Faster):**
  - Running time growth: n^2.
  - Craftiest programmer codes in machine language, resulting in code requiring $2n^2$ instructions to sort n numbers.
- **Computer B (Slower):**
  - Running time growth: $n \log n$.
  - Average programmer writes code in a high-level language with an inefficient compiler, resulting in code requiring $50n \log n$ instructions.

5.2 Running Time Comparison

$$\begin{aligned} &\text{Computer A:} \quad 2 \times (10^7)^2 = 20,000 \text{ seconds} \quad \text{(more than 5.5 hours)} \\ &\text{Computer B:} \quad 50 \times \log(10^7) \approx 1163 \text{ seconds} \quad \text{(under 20 minutes)} \end{aligned}$$

6 Conclusion

The example highlights the critical importance of choosing algorithms with slower rates of growth. Despite the significant discrepancy in raw computing power, the efficiency of the algorithm used by computer B significantly outperforms computer A. This underscores the profound impact that algorithm selection has on overall system performance and efficiency.