Data structures and Algorithms

Note01-Introduction notes

Edited By

Arshia Gharooni

The University of central Tehran branch
Tehran

2023

0.1 Introduction

An algorithm is a set of well-defined steps or procedures that is used to solve a particular problem. The efficiency of an algorithm is an important factor that determines its usefulness. The efficiency of an algorithm is typically measured in terms of its time complexity and space complexity. The time complexity of an algorithm is the amount of time it takes to run as a function of the size of the input data. The space complexity of an algorithm is the amount of memory it requires to run as a function of the size of the input data.

In this lecture note, we will explore various techniques for analyzing the time complexity and space complexity of algorithms. We will begin by discussing the basics of algorithm analysis, including the notation used to describe the time and space complexity of algorithms. We will then introduce several common algorithm design techniques, including brute-force algorithms, divide-and-conquer algorithms, and dynamic programming algorithms. Finally, we will explore several advanced topics in algorithm analysis, including randomized algorithms, parallel algorithms, and approximation algorithms.

0.2 Basics of Algorithm Analysis

Algorithm analysis is the process of evaluating the efficiency of an algorithm. There are several factors that can affect the efficiency of an algorithm, including the size of the input data, the data structure used to represent the input data, and the specific implementation of the algorithm. The two most common measures of algorithm efficiency are time complexity and space complexity.

0.2.1 Time complexity

The time complexity of an algorithm is the amount of time it takes to run as a function of the size of the input data. Typically, the size of the input data is denoted by n. The time complexity of an algorithm is typically expressed using big O notation, which is a mathematical notation used to describe the upper bound of the running time of an algorithm as a function of the size of the input data. The big O notation is denoted by $O(f(n))$, where $f(n)$ is a function that describes the upper bound of the running time of the algorithm.

0.2.2 Space complexity

The space complexity of an algorithm is the amount of memory it requires to run as a function of the size of the input data. Typically, the size of the input data is denoted by n. The space complexity of an algorithm is typically

expressed using big O notation, which is a mathematical notation used to describe the upper bound of the amount of memory required by an algorithm as a function of the size of the input data. The big O notation is denoted by O(f(n)), where f(n) is a function that describes the upper bound of the amount of memory required by the algorithm.

For example, if an algorithm requires a constant amount of memory to run regardless of the size of the input data, its space complexity would be O(1). If an algorithm requires linear space to run as a function of the size of the input data, its space complexity would be O(n). If an algorithm requires quadratic space to run as a function of the size of the input data, its space complexity would be O(n2). The space complexity of an algorithm is important because it determines the amount of memory required by the algorithm and can have a significant impact on the performance of the algorithm.

0.3 Computational Model

Computational models are abstract representations of how computations can be performed. They are used to study the properties of algorithms and to analyze their performance. Computational models provide a framework for reasoning about the limitations and capabilities of computers and algorithms.

There are many different computational models, each with its own strengths and weaknesses. Some models are designed to be simple and easy to analyze, while others are more complex and better able to capture the details of specific computing environments.

In this lecture, we will discuss some of the most commonly used computational models, including Turing machines, register machines, and the RAM model.

0.3.1 Turing Machines

The Turing machine is a theoretical model of computation invented by Alan Turing in 1936. It is a simple abstract machine that consists of an infinite tape divided into cells, a read/write head that can move left or right along the tape, and a finite control that can read and write symbols on the tape. The tape is initially blank, and the machine starts in a predefined state. The machine moves along the tape, reads the symbols on the tape, and performs some computation based on its current state and the symbol it is reading. It can then write a new symbol on the tape, move the read/write head, and change its state. Turing machines can perform any computation that can be done by a physical computer, and they are widely used in the study of algorithms and complexity theory.

0.3.2 Register Machines

Register machines are another type of computational model that are used to study the performance of algorithms. A register machine consists of a finite set of registers, each of which can hold an integer value. The machine has a finite set of instructions, each of which can perform basic arithmetic and logical operations on the values in the registers.

Register machines are more powerful than Turing machines in some respects, because they can perform arithmetic operations more efficiently. However, they are also more complex and harder to analyze.

0.3.3 RAM Model

The RAM model is a modern computational model that is widely used in the study of algorithms. It is based on the architecture of modern computers, and is designed to capture the essential features of their performance. A RAM machine consists of a finite set of registers, each of which can hold an integer value. The machine has a finite set of instructions, each of which can perform basic arithmetic and logical operations on the values in the registers.

The RAM model is useful because it provides a more realistic representation of the performance of algorithms on modern computers. It is also relatively simple and easy to analyze, which makes it a popular choice for theoretical analysis.

0.4 Algorithm Design Techniques

There are several algorithm design techniques that can be used to create efficient algorithms.

0.4.1 Brute-force algorithms

Brute-force algorithms are algorithms that solve a problem by systematically enumerating all possible solutions and selecting the one that satisfies the problem requirements. Brute-force algorithms are typically simple to implement, but can be very inefficient for problems with large input sizes. The time complexity of a brute-force algorithm is typically very high, often exponential or factorial in terms of the size of the input data.

0.4.2 Divide-and-conquer algorithms

Divide-and-conquer algorithms are algorithms that solve a problem by breaking it down into smaller subproblems, solving the subproblems independently, and then combining the solutions to the subproblems to solve the original problem. Divide-and-conquer algorithms are typically more efficient than

brute-force algorithms, but can still have high time complexity if the subproblems are not well-designed. The time complexity of a divide-and-conquer algorithm is typically expressed using recurrence relations, which describe the running time of the algorithm in terms of the running time of its subproblems.

0.4.3 Dynamic programming algorithms

Dynamic programming algorithms are algorithms that solve a problem by breaking it down into smaller subproblems and storing the solutions to the subproblems in a table. The solutions to the subproblems are then used to compute the solution to the original problem. Dynamic programming algorithms are typically more efficient than brute-force algorithms and divide-and-conquer algorithms, but can require a significant amount of memory to store the table of solutions. The time complexity of a dynamic programming algorithm is typically expressed using recurrence relations, which describe the running time of the algorithm in terms of the running time of its subproblems.

0.5 Advanced Topics in Algorithm Analysis

0.5.1 Randomized algorithms

Randomized algorithms are algorithms that use randomization to solve a problem. Randomized algorithms can be very efficient for certain types of problems, but their efficiency is typically measured in terms of their expected running time rather than their worst-case running time. The expected running time of a randomized algorithm is typically expressed using big O notation, which describes the upper bound of the expected running time of the algorithm as a function of the size of the input data.

0.5.2 Parallel algorithms

Parallel algorithms are algorithms that solve a problem by dividing the computation among multiple processors or cores. Parallel algorithms can be very efficient for problems that can be decomposed into independent subproblems, but can be more complex to implement than serial algorithms. The time complexity of a parallel algorithm is typically expressed using parallel complexity measures, which describe the amount of computation that can be performed in parallel as a function of the number of processors or cores.

0.5.3 Approximation algorithms

Approximation algorithms are algorithms that provide an approximate solution to a problem that is guaranteed to be within a certain bound of the

optimal solution. Approximation algorithms can be very efficient for problems that are NP-hard or NP-complete, but their approximation guarantees can be difficult to analyze. The quality of the approximation provided by an approximation algorithm is typically expressed using approximation ratios, which describe the ratio between the approximate solution and the optimal solution.