CSCI3130 Formal Languages and Automata Theory

Ryan Chan

September 1, 2025

Abstract

This is a note for **CSCI3130 Data Structures**.
Contents are adapted from the lecture notes of CSCI3130, prepared by Tsung-Yi Ho, as well as some online resources.
This note is intended solely as a study aid. While I have done my best to ensure the accuracy of the content, I do not take responsibility for any errors or inaccuracies that may be present. Please use the material thoughtfully and at your own discretion.
If you believe any part of this content infringes on copyright, feel free to contact me, and I will address it promptly.
Mistakes might be found. So please feel free to point out any mistakes.

Contents

Chapter 1

Introduction

1.1 Overview

A **data structure** is a way to organize and store data in a computer program, allowing for efficient access and manipulation.

An **algorithm** is different from a **program**. An algorithm is a process or set of rules used for calculation or problem-solving. It is a step-by-step outline or flowchart showing how to solve a problem. A program, on the other hand, is a series of coded instructions that control the operation of a computer or other machines. It is the implemented code of an algorithm.

For example, to solve the greatest common divisor (GCD) problem, we can use the following algorithm.

Algorithm 1.1: Euclid's Algorithm

Data: $m, n \in \mathbb{Z}^+$
Result: GCD(m, n)

```
1 while m > 0 do
2 |  if n > m then
3 |  |  swap m and n
4 |  subtract n from m
5 return n
```

By using a mathematical method to prove this algorithm, we can show that it is correct, provided that it terminates.

Having proved the correctness, we also need to use different test cases to check if there is anything wrong with the coding or the proof. We should consider special cases, including large values, swapped values, etc.

We are also interested in the time and space (computer memory) it uses, which we call **time complexity** and **space complexity**. Typically, complexity is a function of the values of the inputs, and we would like to know which function. We can also consider the best case, average case, and worst-case scenarios.

For example, in the above algorithm, the best case would be $m = n$, with just one iteration. If $n = 1$, there are m iterations, which is the worst case. However, for the average case, it is difficult to analyze.

Also, for space complexity, it is constant since we only use space for the three integers: m, n, and t.

To improve the above algorithm, we can use `mod`, so we don't need to keep doing subtraction.

1.2 Algorithm

An algorithm is a finite set of instructions which, if followed, accomplishes a particular task. Every algorithm must satisfy the following criteria:

- Input: There are zero or more quantities that are externally supplied.

- Output: At least one quantity is produced.

- Definiteness: Each instruction must be clear and unambiguous.

- Finiteness: If we trace out the instructions of an algorithm, then for all cases, the algorithm will terminate after a finite number of steps.

- Effectiveness: Every instruction must be sufficiently basic that it can, in principle, be carried out by a person using only pencil and paper.

We also define an algorithm as any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output. It is thus a sequence of computational steps that transform the input into output.

It can also be viewed as a tool for solving a well-specified computational problem. The problem statement specifies, in general terms, the desired input or output relationship, and the algorithm describes a specific computational procedure for achieving that input or output relationship.

An algorithm is said to be correct if, for every input instance, it halts with the correct output. It can solve the given computational problem. In contrast, an incorrect algorithm might not halt at all on some input instances, and sometimes it can even produce useful results.

1.3 Study of Data

A data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently.

A data structure is a set of domains D, a designated domain $d \in D$, a set of functions F, and a set of axioms A.

An implementation of a data structure d is a mapping from d to a set of other data structures e.