Welcome to
Theory of Computing (CSE-4309)

Tanjila Alam Sathi
Lecturer, CSE Department
Islamic University of Technology (IUT)

Text Book

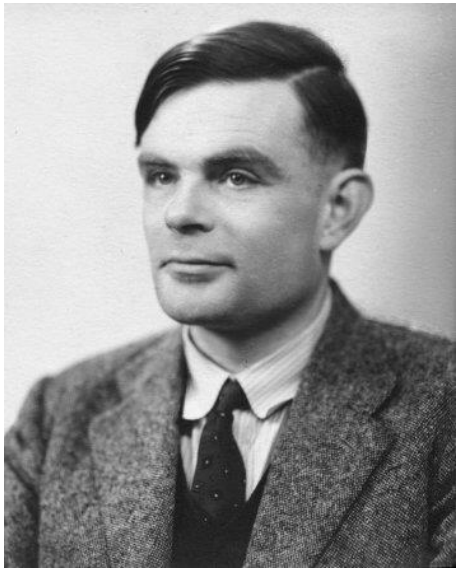
- Introduction to Automata Theory, Languages, and Computation: John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman.
- Introduction to the Theory of Computation- Michael Sipser

What is Theory of Computing?

- One of the most fundamental courses of Computer Science
- It will help you understand how people have thought about computer Science as a science in the past 50 years.
- It is mainly about what kind of things can you really compute mechanically, how fast and how much space does it take to do so.

What is Theory of Computing?

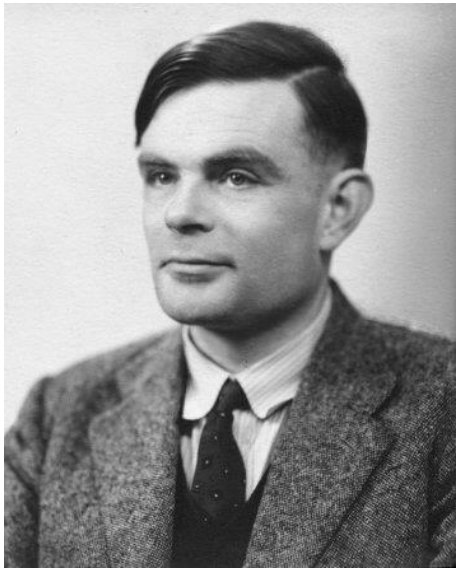
#The science in computer science.
#The formal study of what a computer can, and cannot do *even in principle*.
#Independent of technology: Limitations of -any- physical device: e.g. your laptop, your brain, future technology
--The physics of computation: Computation as a fundamental phenomenon



What is Theory of Computing?

Once a problem is formalized as a "computational" problem with clearly specified inputs and outputs, before you write code, let's understand:

#Can the problem be solved at all by a computer?
#How efficiently can it be solved?
#How can we be convinced that the solution is correct?



Course objectives:

- What is an algorithm?
- What can and what cannot be computed?
- When should an algorithm be considered practically feasible?
- Search answer more than 70 years?
- To introduce fundamental ideas, models & results that permeate computer science

Key Points

- What the course is about
- Why study Automata?

Theory of Computing divided into:

- Automata Theory
- Computability Theory
- Computational Complexity Theory

Introduction to Automata theory

#Automata = “Self Acting”

#Automation: an abstract self-propelled computing device which follows a predetermined sequence of operations automatically

#Allows scientists to understand how machine solves problems

#What we can compute and what we cannot

Introduction to Automata theory

- Automata theory is the study of abstract devices/machines.
- Before there were computers, in the 1930's, Alan Turing studied an abstract machine that had all the capabilities of today's computers, at least as far as in what they could compute.
- What a computing machine could do? what it could not do?

Why studying?

- Much of modern computer science is based more/less on them.
- These ideas/models are powerful, beautiful, excellent examples of mathematical modeling that is elegant, productive, & of lasting value
- Besides, they are so much a part of the history & the collective subconscious, that it is hard to understand computer science without first being exposed to them

Levels of Automata:

Finite State Machine

Context Free Language

Turing Machine

NP-Hard problems

Why Study Automata?

- Finite automata are a useful model for many important kinds of hardware & software:

1. Software for designing & checking the behavior of the digital circuits.
2. The lexical analyzer of a typical compiler, that is, the compiler component that breaks the input text into logical units, such as identifiers, keyword, and punctuation.
3. Software for scanning large bodies of text, such as collections of web pages, to find occurrences of words, phrases, or other patterns

Why Study Automata?

4. Software for verifying systems of all types that have a finite number of distinct states, such as communications protocols or protocols for secure exchange of information

-finite number of states

-implement in hardware as a circuit/simple form of program

-use limited amount of data/using position in the code itself to make the decision

How Could That Be?

- Regular expressions are used in many systems.
- Finite automata model protocols, electronic circuits.

How? – (2)

- Context-free grammars are used to describe the syntax of essentially every programming language.
  - Not to forget their important role in describing natural languages.

How? – (3)

- When developing solutions to real problems, we often confront the limitations of what software can do.
  - *Undecidable* things – no program whatever can do it.
  - *Intractable* things – there are programs, but no fast programs.

Problem 1: Character Coverage in Documents

Given a text document, check if each of the vowels a, e, i, o, u appears at least once in the document

As the algorithm scans the document reading one character at a time, what information should it track?

For each vowel, need to know whether or not it has appeared in the document read so far

- Maintain one bit, initialized to 0, for each of the vowels
- If the read character is a vowel, set the corresponding bit to 1
- In the end, check if all bits are 1

Problem 2: Character Count in Documents

Given a text document, check if the number of occurrences of the characters a and e are the same

As the algorithm scans the document reading one character at a time, what information should it track?

Track the difference in number of times a and e have occurred so far

- Count is initially 0
- Increment it if the read character is a
- Decrement it if the read character is e
- In the end, check if the count is 0

Finite-State Computation

Problem 1: All five vowels appear at least once

- Need to maintain 5 bits of memory (constant)
- Computing device to solve this problem needs only 32 states

Problem 2: Count of a's = count of e's

- The value of the variable tracking the difference in the two counts is potentially unbounded
- Memory needed for the computation depends on input length
- If a computing device has only finitely many states, it cannot solve this problem! (How do we prove such a statement?)