Introduction

CS 5633 Analysis of Algorithms

Computer Science
University of Texas at San Antonio

August 26, 2024

What are Algorithms and Why Learn Algorithms?

The Early History of Computer

- The original question about computers.
  - Can a machine solve a problem?

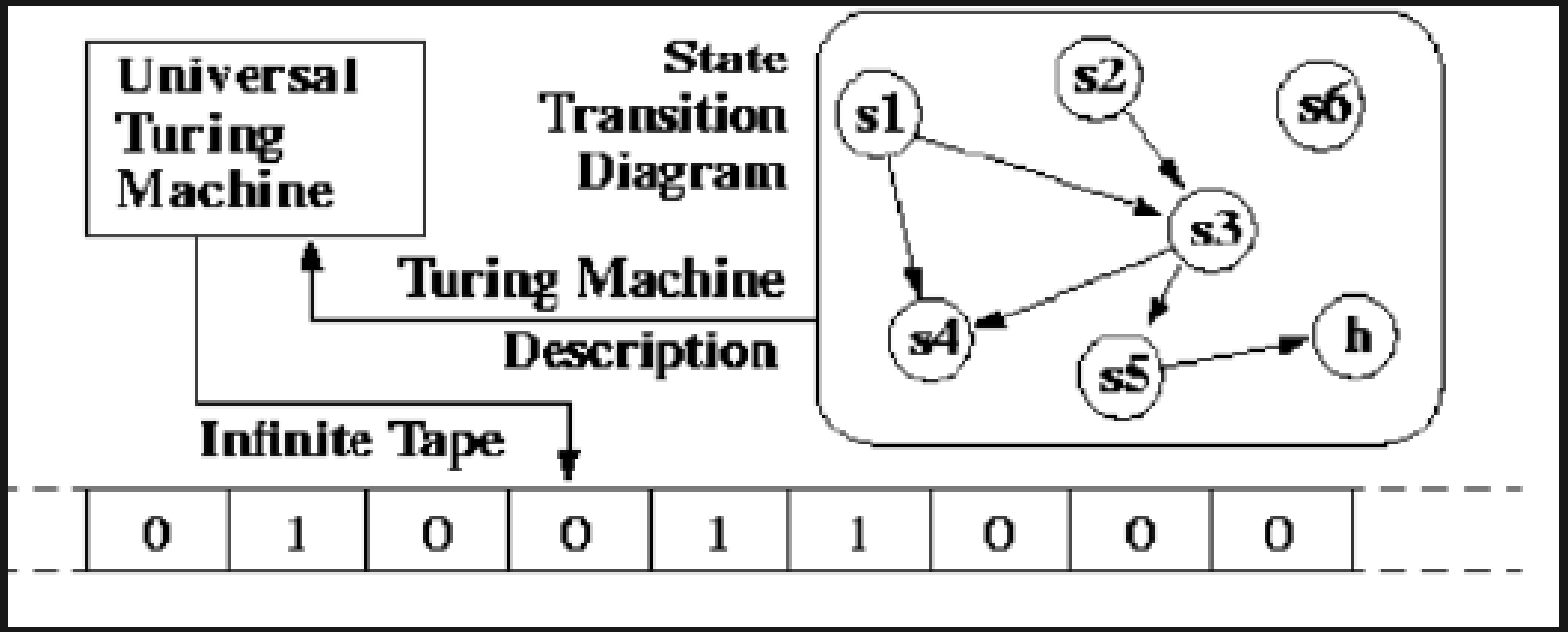
The Early History of Computer

- The original question about computers.
  - Can a machine solve a problem?
  - The answer is pretty clear: Yes
- A more advanced question.
  - Can a machine solve all problems?
  - This is a very hard question.
  - How do you want to answer this question?

The Early History of Computer

- The original question about computers.
  - Can a machine solve a problem?
  - The answer is pretty clear: Yes
- A more advanced question.
  - Can a machine solve all problems?
  - This is a very hard question.
  - How do you want to answer this question?
  - Alan Turing answered this question with a technique close to “Proof by Construction”
- The Turing machine
  - The basic mathematical model for modern computer.
  - It can solve some problems, but not all problems.
  - The problems that cannot be solved by Turing machine are called “Undecidable” problems.

The Early History of Computer



The Early History of Computer

- The Turing machine
  - The basic mathematical model for modern computer.
  - It can solve some problems, but not all problems.
  - The problem that cannot be solved by Turing machine are called “Undecidable” problems.
- The Halting Problem
  - Given a program *p* and an input *i*, will *p* ever stop when processing *i*?
  - Is there a way to answer this question with a computer?

The Early History of Computer

- The Turing machine
  - The basic mathematical model for modern computer.
  - It can solve some problems, but not all problems.
  - The problem that cannot be solved by Turing machine are called “Undecidable” problems.
- The Halting Problem
  - Given a program *p* and an input *i*, will *p* ever stop when processing *i*?
  - Is there a way to answer this question with a computer?
  - The answer is NO.
  - Intuitively, a problem may have an infinite solution space, and solving it requires exploring the whole solution space, which makes a computer to run forever to find the solution.

Dealing with Decidable Problems

- It is all about how to efficiently solve these problems.
  - "Brute-force" solutions are usually slow.
  - Efficiency includes both time efficiency and memory (space) efficiency.
- Not all decidable problems can be solved quickly.
  - "P" problems: problems that has polynomial time solutions (fast).
  - "NP-complete" problems: problem that may not have polynomial time solutions (slow).
  - We don't know for sure if "NP-complete" problems have polynomial time solutions or not. Most people believe they don't.
  - A not-very-accurate-intuition: "NP-complete" problems may only have brute-force solutions; we have to check every solution in the solution space to find the solution we want.

Dealing with Decidable Problems cont.

- For decidable problems,
  - We need to be able to tell if a problem is “NP-complete.”
  - If it is not “NP-complete,” we need to be able to find a good, if not best, polynomial time solution.
- Solving “decidable” problems
  - We can solve them with a computer (Turing machine).
  - The procedure to solve a problem on a computer is called an “Algorithm.”

What are Algorithms?

- Algorithm
  - 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.
  - Example: the *sorting problem.*
    - Input: Array $A[1 \ldots n]$ of numbers.
    - Output: Sorted array A. That is, the elements should be ordered such that

$$A[1] \leq A[2] \leq \cdots \leq A[n].$$

Steps of Algorithm Design

- Define the computational problem to be solved.
  - What is the input and output of the problem?
- Describe the algorithm.
  - Description of algorithm in English followed by pseudocode.
- Proof of correctness.
  - Convince the reader of correctness.
  - Algorithm must return the desired output for **any input**. Showing that the algorithm works for some specific example is **not** a proof of correctness of the algorithm.

Steps of Algorithm Design

- Proof of runtime.
  - "Stopwatch" runtime depends heavily on things independent of the algorithm (e.g. hardware). We instead measure runtime in terms of the number of steps the algorithm takes.
  - Often interested in the **worst-case** runtime and space used (i.e. memory).
- Proof of space (sometimes).
  - Prove bounds on the amount of memory the algorithm will need to produce the output.

Other Related Algorithm Concepts

- We call a fixed input for a computational problem an **instance** of the problem.
  - For example, the input $[6, 2, 0, 1, 5, 2]$ is an instance of the sorting problem.
- We say that an algorithm is **correct** if the algorithm terminates with the correct output for every instance of the problem.
- We say that a correct algorithm **solves** the given computational problem.

Why Learn Algorithms

- Be able to write programs to solve basic problems.
  - Many common problems are well-researched.
  - These problems usually have time-efficiency solutions.
- Extend known solutions or employ known methodology to solve new problems.
- Be able to optimize your programs.
- A key to find good jobs.

Goals of this Course

- Understand the complexity analysis of algorithms.
- Learn basic data structures, such as trees, heap, graphs and stack.
- Learn basic algorithm constructing methods, such as divide-and-conquer, dynamic programming and greedy algorithms.
- Learn classic sorting, graphic and optmization algorithms.
- Understand the limitations of the modern computers.

Course Information

Course Information

- Prerequisites: CS 3343 Undergraduate Algorithms.
- Three in class midterm exams (40%); low exam grade is dropped
- One final exam (25%)
- 9-11 Assignments (30%)
- Class participation, in class quizzes, special and extra credit events (5%)
- The percentages may be adjusted depending on the number of assignments and quizzes

Course Information cont.

- Text book: Introduction to Algorithms, 3rd Edition – Cormen, Leiserson, Rivest, and Stein.
  - This book will be referred as “CLRS” (initials of the authors) in this class.
- Late homework is docked 10%
  - If it is more than one week late, the assignment will not be accepted.
- Office Hours
  - NPB 3.310
  - Mon and Wed 10:00am – 11:00am
  - and by appointment
- For e-mail contact, always include ”CS5633” in the subject line.

Course Information cont.

- Class site
  - Syllabus
  - Blackboard for slides, handouts and assignments
  - Prepare a notebook for taking notes.
- Read emails!
- TA: Rojan Hosseini
  - rojan.hosseini@utsa.edu

Topics and Schedule

- Topics
  - Algorithm analysis techniques: approximating functions asymptotically, bounding sums, and solving recurrences.
  - Basic algorithm design techniques: divide-and-conquer, dynamic programming, amortization and greedy algorithms.
  - Basic data structures: various trees, graphs and heap.
  - Classic algorithms: sorting, graph and optimization.
  - P and NP.
- Schedule
  - Tentative schedule can be found on Canvas
  - This schedule will definitely change based on our pace.
  - Exam dates are fixed, so plan your travel ahead.

About Me

- Research areas: Software engineering, Software Security, Augmented Reality
- Research interests: Software build and analysis, AR software testing, AR security

Student Participation

- Attendance.
- Missing many lectures will result in an F grade.
- In-class discussion.
- Ask questions.
- Let me know what do you think about this course.
- Any feedback is welcome.

Acknowledgment

- This lecture is based on Dr. Matt Gibson's graduate Algorithms from past years.