Middlebury

CSCI 201: Data Structures

Spring 2025

Lecture 12W: Course wrap-up

Specifics we learned

Data Structures

- Arrays
- Lists: ArrayList and LinkedList
- Sets: HashSet and TreeSet
- Maps: HashMap and TreeMap
- Stacks, Queues, Priority Queues / Heaps
- Trees: Binary Search Trees
- Graph representations

Algorithms

- Iterative
- Hashing
- Big O Asymptotic Analysis
- Recursive
- Sorting
- Greedy
- Graph

Software

- Java API
- Objects, Classes
- Interfaces, implementations
- Testing, Debugging

Algorithms / code

In order to execute an algorithm on a real computer, we must write the algorithm in a formal language. An algorithm so written is a **program**.

In this class we explore both:

Theory

- Design an algorithm
- Analyze performance
- Data structure tradeoffs

Practice

- Write a Java program
- Debug/test
- Measure performance

Why efficiency matters

- You wrote the next big social media app:
  - Will it work if it has 1 billion users?
  - What about on a phone with limited memory?
- In the sciences, discovery depends on computing with big data:
  - Sequencing the human genome
  - Surveying millions of images in astronomy
  - Processing data logs from the CERN collider
- Pushing the limits of current technology:
  - Virtual / augmented reality?
  - Deep neural networks for large scale machine learning?

What can computers do?

What can’t computers do?

- Some problems ***cannot be solved at all***
  - One program detects all infinite loops
- Some problems ***cannot be solved efficiently***
  - Listing all N-bit sequences of 0's and 1's
- Some problems can be ***approximately solved***
  - AI, ML, close-to-optimal is good enough

Halting Problem

- Can we write doesHalt as specified? *Suppose so!*
  - Like the Java Compiler: reads a program

```
public class ProgramUtils
    /**
     * Returns true if progname halts on input,
     * otherwise returns false (infinite loop)
     */
    public static boolean doesHalt(String progname){
    }
}
```

Can we confuse doesHalt?

- What if **`doesHalt(confuse)`** returns true?
  - Then **`confuse()`** does not halt (see below)
- What if **`doesHalt(confuse)`** returns false?
  - Then **`confuse()`** does halt (see below)

```
public static boolean confuse(){
    if (ProgramUtils.doesHalt(confuse)) {
        while (true) {
            // do nothing forever
        }
    }
}
```

Formal proof by Alan Turing

- Alan Turing first showed this for programs: 1936
  - Had to formally specify what a program was
  - Needed to invent concept of Turing Machine
  - Also demonstrated by Alonzo Church
- Cantor showed # Real Numbers > # Rationals
  - So-called diagonalization, 1891
  - Ridiculed by establishment
  - Argument essential to above

Shortest / longest paths

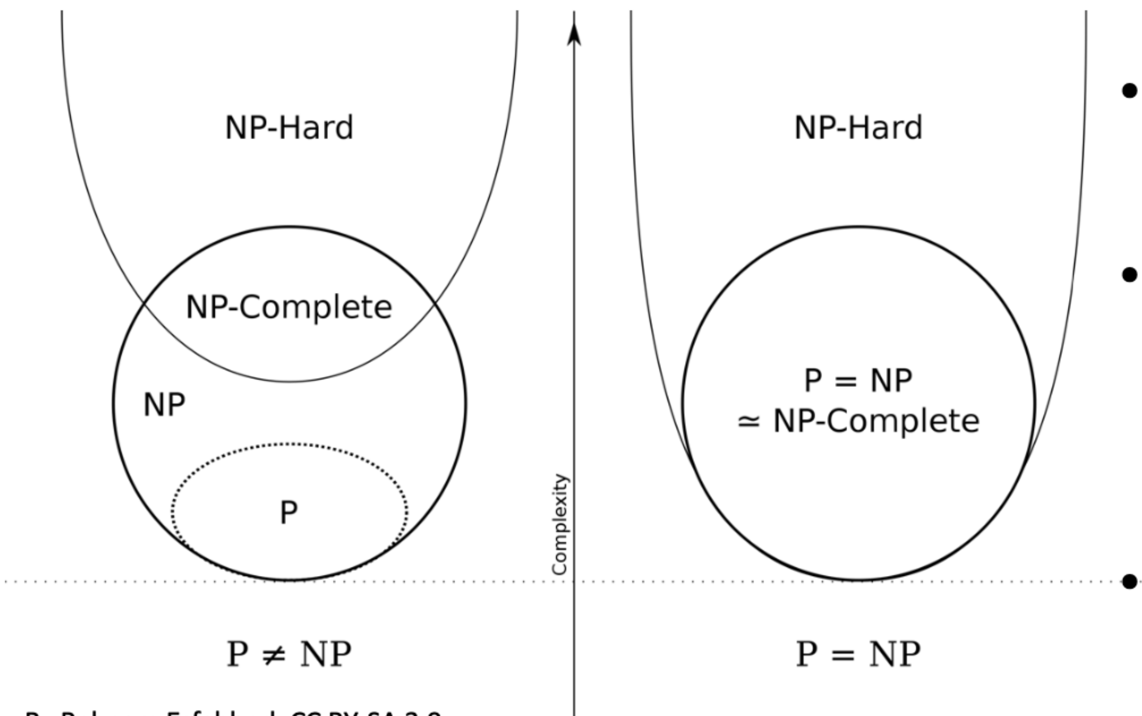
- Dijkstra's Algorithm one example
  - Others: Floyd-Warshall and more
  - Very efficient graph algorithms
- Longest Path? No efficient solution known
  - Easy to verify "is this path greater than length k"
  - Exponentially many paths

P vs NP

- P is the set of (algorithmic) problems that can be solved by a deterministic Turing Machine (DTM) in time that is polynomial in the size of the input (polynomial time).
  - i.e., can solve with a program that is O(1), O(N), O(Nlog(N)), $O(N^2)$, $O(N^3)$, …, $O(N^{128})$, …

- NP is (roughly) the set of (algorithmic) problems for which a solution can be *verified* by a DTM in polynomial time.
  - Equivalently: problems that can be solved by a nondeterministic Turing Machine in polynomial time (Quantum computing???)

P ?= NP



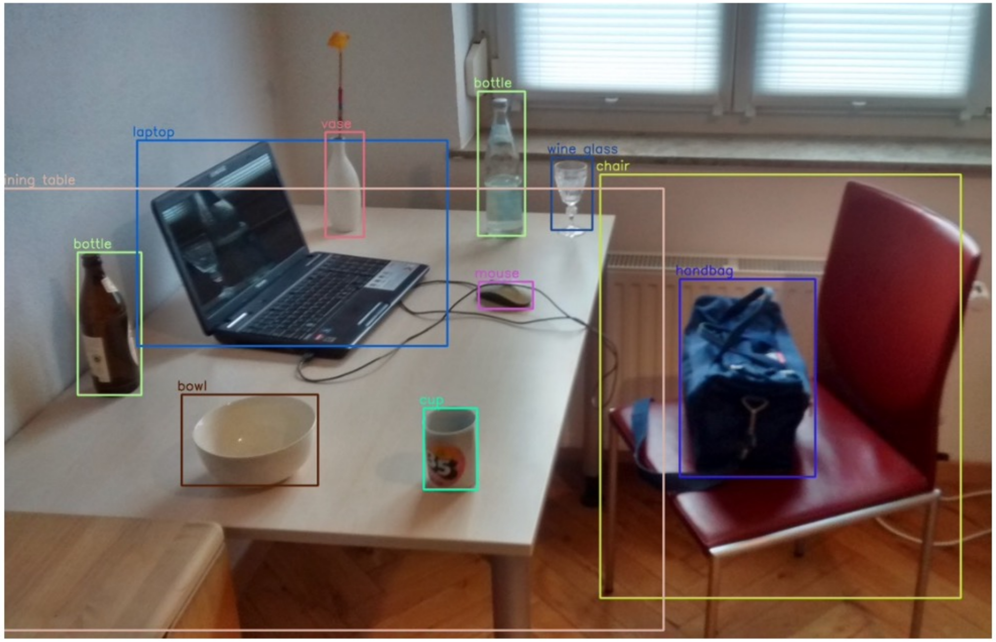
- Most think P != NP
- Greatest outstanding question in theoretical computer science
- Proof is worth a $1M prize from the Clay Mathematics Institute

“Easy” hard problems

- Some problems are hard to solve but easy to approximate:
  - Can’t write a program to give you the optimal solution efficiently but can find something within ϵ of optimal in polynomial time.
  - Greedy, randomized, etc.

- Some problems are hard to prove things in theory but easy to solve in practice
  - Can’t prove much but it works well in practice

AI / ML often work with experimental algorithms for hard problems



Common idea: Use a computer to learn a function/neural network that approximates a large dataset.

- Image segmengation / classification
- Face/speech recognition
- Machine translation
- Text generation
- Reinforcement learning
- Robotics
- ...

Practice coding!

- LeetCode
- CodingBat
- HackerRank
- CodeForce
- CodeJam
- Project Euler
- GitHub Student Developer Pack

- Write code for fun, to solve puzzle, game

Final exam format

Programming portion

- Monday 5/12 through Thursday 5/15
- Independent problems submitted to Gradescope
- Submit as often as you like, but Gradescope will only indicate whether the code compiled
- One hour grace period for lateness (not 1 day)

Written exam

- Thursday 5/15, 2:00 – 5:00, MBH 224 and 206
- Short answer, sketching
- No devices

Next:

- Course Response Forms
- Continue working on Homework 10, due Thursday 5/8
- Lab 10 on Friday 5/9: practice and surveys
- Monday 5/12: office hours, no lab meetings