CS 374: Algorithms & Models of Computation

Chandra Chekuri Manoj Prabhakaran

University of Illinois, Urbana-Champaign

Fall 2015

Administrivia, Introduction

Lecture 1
August 25, 2015

Part I

Administrivia

Instructional Staff

1. Instructors: Chandra Chekuri and Manoj Prabhakaran
2. 7 (or 8) Teaching Assistants
3. 16 Undergraduate Course Assistants
4. Office hours: See course webpage
5. Email: Use private notes on Piazza to reach course staff.

Online resources

1. Webpage: General information, announcements, homeworks, course policies `courses.engr.illinois.edu/cs374`
2. Moodle: HW submission, Quizzes, solutions to homeworks, grades
3. Piazza: Announcements, online questions and discussion, contacting course staff (via private notes)

See course webpage for links

Important: check Piazza/course web page at least once each day

Textbooks

1. Prerequisites: CS 173 (discrete math), CS 225 (data structures)
2. Recommended books: (not required)
   1. Introduction to Theory of Computation by Sipser
   2. Introduction to Automata, Languages and Computation by Hopcroft, Motwani, Ullman
   3. Algorithms by Dasgupta, Papadimitriou & Vazirani. Available online for free!
   4. Algorithm Design by Kleinberg & Tardos
3. Lecture notes/slides/pointers: available on course web-page
4. Additional References
   1. Lecture notes of Jeff Erickson, Sariel HarPeled, Mahesh Viswanathan and others
   2. Introduction to Algorithms: Cormen, Leiserson, Rivest, Stein.
   3. Computers and Intractability: Garey and Johnson.

Grading Policy: Overview

1. Quizzes: 4%
2. Homeworks: 24%
3. Midterms: 44% ($\mathbf{2 \times 22\%}$)
4. Finals: 28% (covers the full course content)

Midterms dates:

1. Midterm 1: Mon, Sept 28, 7–9pm
2. Midterm 2: Mon, Nov 9, 7–9pm

No conflict exam offered unless you have a valid excuse.

Homeworks

1. One quiz every week. Due by midnight on Thursday (except Quiz 0). Short questions to be answered on *Moodle*. Quizzes to be done individually.
2. One homework every week: Due on Tuesdays at 10am on *Moodle*. Assigned at least a week in advance.
3. Homeworks can be worked on in groups of up to 3 and each group submits *one* written solution (except Homework 0).
4. Important: academic integrity policies. See course web page.

More on Homeworks

1. No extensions or late homeworks accepted.
2. To compensate, four problems will be dropped. Homeworks typically have three problems each.
3. Important: Read homework faq/instructions on website.

Discussion Sessions

1. 50min problem solving session led by TAs
2. Two times a week
3. Go to your assigned discussion section
4. Bring pen and paper!

Advice

1. Attend lectures, please ask plenty of questions.
2. Attend discussion sessions.
3. Don't skip homework and don't copy homework solutions.
4. Study regularly and keep up with the course.
5. This is a course on problem solving. Solve as many as you can! Books/notes have plenty.
6. This is also a course on providing rigourous proofs of correctness. Refresh your 173 background on proofs.
7. Ask for help promptly. Make use of office hours/Piazza.

Homeworks

1. HW 0 is posted on the class website. Quiz 0 available on Moodle.
2. Quiz 0 due by Sunday August 30 at midnight
3. HW 0 due on Tuesday September 1 at 10am on Moodle
4. HW 0 to be done and submitted individually.

Miscellaneous

Please contact instructors if you need special accommodations.

Lectures are being taped. See course webpage.

Part II

Course Goals and Overview

High-Level Questions

1. Algorithms
   1. What is an algorithm?
   2. What is an *efficient* algorithm?
   3. Some fundamental algorithms for basic problems
   4. Broadly applicable techniques in algorithm design
2. What is the formal definition of a computer?
   1. Is there a formal definition?
   2. Is there a "universal" computer?
3. What can computers compute?
   1. Are there tasks that our computers cannot do?

Course Structure

Course divided into three parts:

1. Basic automata theory: finite state machines, regular languages, hint of context free languages/grammars
2. Algorithms and algorithm design techniques
3. Turing Machines, Undecidability, NP and NP-Completeness

Algorithm

Wikipedia:

An algorithm is an effective method expressed as a finite list of well-defined instructions for calculating a function. Starting from an initial state and initial input (perhaps empty), the instructions describe a computation that, when executed, proceeds through a finite number of well-defined successive states, eventually producing output and terminating at a final ending state.

Algorithm

Wikipedia continued:

The concept of algorithm has existed for centuries, however a partial formalization of what would become the modern algorithm began with attempts to solve the Entscheidungsproblem (the "decision problem") posed by David Hilbert in 1928. Subsequent formalizations were framed as attempts to define "effective calculability" or "effective method"; those formalizations included the Gödel-Herbrand-Kleene recursive functions of 1930, 1934 and 1935, Alonzo Church's lambda calculus of 1936, Emil Post's "Formulation 1" of 1936, and Alan Turing's Turing machines of 1936-37 and 1939. Giving a formal definition of algorithms, corresponding to the intuitive notion, remains a challenging problem.

Algorithm

Key points:

1. Algorithm solves a specific problem.
2. Steps/instructions of an algorithm are simple/primitive and can be executed "mechanically"
3. Algorithm has a finite description; same description for all instances of the problem
4. Algorithm implicitly may have state

Algorithms and Computing

1. Algorithm solves a specific problem.
2. Steps/instructions of an algorithm are simple/primitive and can be executed "mechanically"
3. Algorithm has a finite description; same description for all instances of the problem
4. Algorithm implicitly may have state

A computer is a device that

1. "implements" the primitive instructions
2. allows for an "automated" implementation of the entire algorithm by keeping track of state

Algorithms and Computing

1. Algorithm solves a specific problem.
2. Steps/instructions of an algorithm are simple/primitive and can be executed "mechanically"
3. Algorithm has a finite description; same description for all instances of the problem
4. Algorithm implicitly may have state

A computer is a device that

1. "implements" the primitive instructions
2. allows for an "automated" implementation of the entire algorithm by keeping track of state

A human can be a computer! And they were in WW II!

Historical motivation for computing

1. Fast (and automated) *numerical calculations*
2. Automating mathematical theorem proving

Models of Computation vs Computers

1. Model of Computation: an "idealized mathematical construct" that describes the primitive instructions and other details
2. Computer: an actual "physical device" that implements a very specific model of computation

Models of Computation vs Computers

1. Model of Computation: an "idealized mathematical construct" that describes the primitive instructions and other details
2. Computer: an actual "physical device" that implements a very specific model of computation

Models and devices:

1. Algorithms: usually at a high level in a model
2. Device construction: usually at a low level
3. Intermediaries: compilers
4. How precise? Depends on the problem!
5. Physics helps implement a model of computer
6. Physics also inspires models of computation

Adding Numbers

Problem Given two **n**-digit numbers **x** and **y**, compute their sum.

Basic addition

$$\begin{array}{r} 3141 \\ +7798 \\ \hline 10939 \end{array}$$

Adding Numbers

```
c = 0
for i = 1 to n do
    z = x_i + y_i
    z = z + c
    If (z > 10)
        c = 1
        z = z − 10      (equivalently the last digit of z)
    Else c = 0
    print z
End For
If (c == 1) print c
```

Adding Numbers

```
c = 0
for i = 1 to n do
    z = x_i + y_i
    z = z + c
    If (z > 10)
        c = 1
        z = z − 10      (equivalently the last digit of z)
    Else c = 0
    print z
End For
If (c == 1) print c
```

1. Primitive instruction is addition of two digits
2. Algorithm requires $O(n)$ primitive instructions
3. Many details for actual implementation on a device
   1. How is input represented? How does device take input?
   2. How do we implement loops? Does devise have scratch space?

Multiplying Numbers

Problem Given two **n**-digit numbers **x** and **y**, compute their product.

Grade School Multiplication

Compute "partial product" by multiplying each digit of **y** with **x** and adding the partial products.

$$
\begin{array}{r}
3141 \\
\times 2718 \\
\hline
25128 \\
3141 \\
21987 \\
6282 \\
\hline
8537238
\end{array}
$$

Time analysis of grade school multiplication

1. Each partial product: $\Theta(n)$ time
2. Number of partial products: $\leq n$
3. Adding partial products: n additions each $\Theta(n)$ (Why?)
4. Total time: $\Theta(n^2)$
5. Is there a faster way?

Fast Multiplication

Best known algorithm: $\mathbf{O(n \log n \cdot 2^{O(\log^* n)})}$ time [Furer 2008]

Previous best time: $\mathbf{O(n \log n \log \log n)}$ [Schonhage-Strassen 1971]

Conjecture: there exists an $\mathbf{O(n \log n)}$ time algorithm

Fast Multiplication

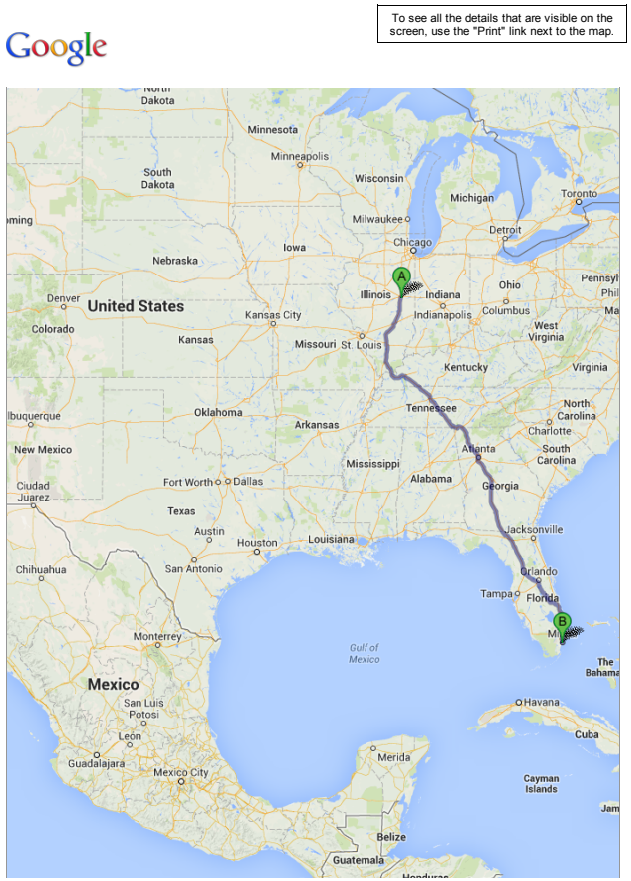
Best known algorithm: $\mathbf{O(n \log n \cdot 2^{O(\log^* n)})}$ time [Furer 2008]

Previous best time: $\mathbf{O(n \log n \log \log n)}$ [Schonhage-Strassen 1971]

Conjecture: there exists an $\mathbf{O(n \log n)}$ time algorithm

We don't fully understand multiplication!
Computation and algorithm design is non-trivial!

Shortest Paths



Shortest Path Problem

Shortest Path Problem

1. Input: A (undirected or directed) graph $\mathbf{G = (V, E)}$ with non-negative edge lengths. For edge $\mathbf{e = (u, v)}$, $\boldsymbol{\ell(e) = \ell(u, v)}$ is its length.
2. $\mathbf{n}$ number of nodes and $\mathbf{m}$ number of edges
3. Given nodes $\mathbf{s, t}$ find shortest path from $\mathbf{s}$ to $\mathbf{t}$.

Non-obvioius but efficient algorithm: Dijkstra's algorithm can be implemented in $\mathbf{O(n \log n + m)}$ time

Longest Path Problem

Longest Path Problem

1. Input: A (undirected or directed) graph $\mathbf{G = (V, E)}$ with non-negative edge lengths. For edge $\mathbf{e = (u, v)}$, $\boldsymbol{\ell(e) = \ell(u, v)}$ is its length.
2. Given nodes $\mathbf{s, t}$ find *longest* path from $\mathbf{s}$ to $\mathbf{t}$.

Longest Path Problem

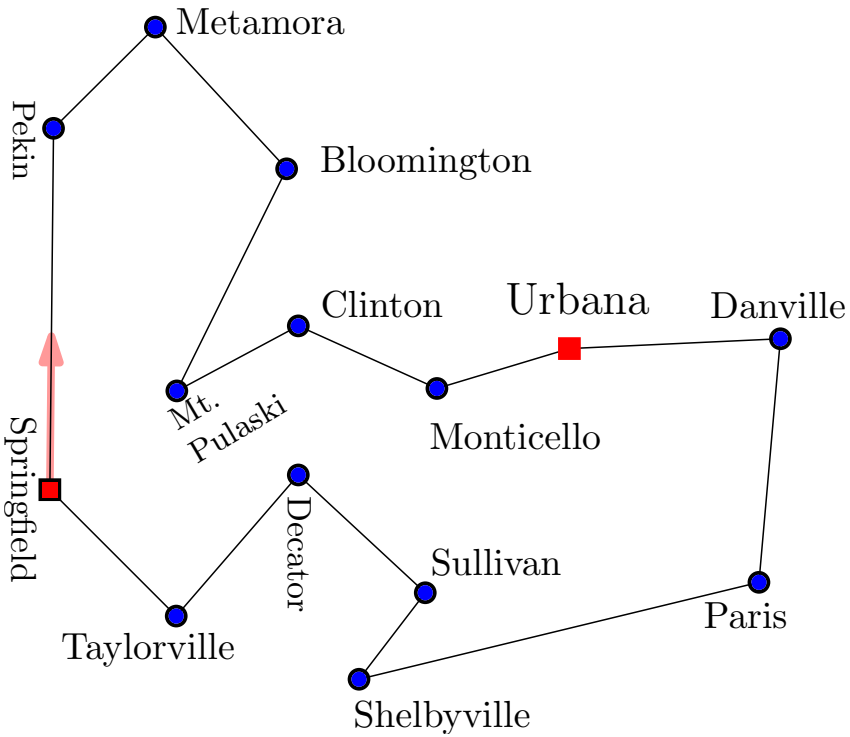
Longest Path Problem

1. Input: A (undirected or directed) graph $\mathbf{G = (V, E)}$ with non-negative edge lengths. For edge $\mathbf{e = (u, v)}$, $\mathbf{\ell(e) = \ell(u, v)}$ is its length.
2. Given nodes $\mathbf{s, t}$ find *longest* path from $\mathbf{s}$ to $\mathbf{t}$.

Nothing better than an $\mathbf{O(2^n)}$ time algorithm known!
Essentially the same as the famous TSP problem.

TSP problem

Lincoln's tour



1. Circuit court - ride through counties staying a few days in each town.
2. Lincoln was a lawyer traveling with the Eighth Judicial Circuit.
3. Picture: travel during 1850.
   1. Very close to optimal tour.
   2. Might have been optimal at the time..

Solving TSP by a Computer

Is it hard?

1. $\mathbf{n}$ = number of cities.
2. $\mathbf{n^2}$: size of input.
3. Number of possible solutions is

$$\mathbf{n * (n-1) * (n-2) * \ldots * 2 * 1 = n!.}$$

4. $\mathbf{n!}$ grows very quickly as $\mathbf{n}$ grows.
   $\mathbf{n = 10}$: $\mathbf{n! \approx 3628800}$
   $\mathbf{n = 50}$: $\mathbf{n! \approx 3 * 10^{64}}$
   $\mathbf{n = 100}$: $\mathbf{n! \approx 9 * 10^{157}}$

Solving TSP by a Computer

Fastest computer...

1. A good super computer can do (some what out dated)

$$\mathbf{2.5 * 10^{15}}$$

   operations a second.
2. Assume: computer checks $\mathbf{2.5 * 10^{15}}$ solutions every second, then...
   1. $\mathbf{n = 20} \implies$ 2 hours.
   2. $\mathbf{n = 25} \implies$ 200 years.
   3. $\mathbf{n = 37} \implies \mathbf{2 * 10^{20}}$ years!!!

What is a good algorithm?

Running time...

Input size	n^2 ops	n^3 ops	n^4 ops	$n!$ ops
5	0 secs	0 secs	0 secs	0 secs
20	0 secs	0 secs	0 secs	16 mins
30	0 secs	0 secs	0 secs	$3 \cdot 10^9$ years
100	0 secs	0 secs	0 secs	never
8000	0 secs	0 secs	1 secs	never
16000	0 secs	0 secs	26 secs	never
32000	0 secs	0 secs	6 mins	never
64000	0 secs	0 secs	111 mins	never
200,000	0 secs	3 secs	7 days	never
2,000,000	0 secs	53 mins	202.943 years	never
10^8	4 secs	12.6839 years	10^9 years	never
10^9	6 mins	12683.9 years	10^{13} years	never

What is a good algorithm?

Running time...

Efficient algorithms

Question:

What is an efficient algorithm?

Efficient algorithms

Question:

What is an efficient algorithm?

In this class *efficiency* is broadly equated to *polynomial time*. $\mathbf{O(n)}, \mathbf{O(n \log n)}, \mathbf{O(n^2)}, \mathbf{O(n^3)}, \mathbf{O(n^{100})}, \ldots$ where $\mathbf{n}$ is size of the input.

Efficient algorithms

Question:

What is an efficient algorithm?

In this class *efficiency* is broadly equated to *polynomial time*. $\mathbf{O(n)}, \mathbf{O(n \log n)}, \mathbf{O(n^2)}, \mathbf{O(n^3)}, \mathbf{O(n^{100})}, \ldots$ where $\mathbf{n}$ is size of the input.

Why? Is $\mathbf{n^{100}}$ really efficient/practical? Etc.

Efficient algorithms

Question:

What is an efficient algorithm?

In this class *efficiency* is broadly equated to *polynomial time*. $\mathbf{O(n)}, \mathbf{O(n \log n)}, \mathbf{O(n^2)}, \mathbf{O(n^3)}, \mathbf{O(n^{100})}, \ldots$ where $\mathbf{n}$ is size of the input.

Why? Is $\mathbf{n^{100}}$ really efficient/practical? Etc.

Short answer: polynomial time is a **robust**, mathematically sound way to define efficiency. Has been useful for several decades.

Problems and Algorithms

Many many different problems.

1. Add two numbers: obvious algorithm essentially optimal.
2. Multiplying two numbers: simple and obvious algorithm efficient. Much faster non-trivial algorithms exist.
3. Shortest Path: an efficient algorithm known but not obvious.
4. TSP: No efficient alogorithm known, conjectured to not exist (the famous **P** $\neq$ **NP** conjecture)
5. Some problems: no algorithm exists!