COMPSCI 311: Introduction to Algorithms

Marius Minea
marius@cs.umass.edu

University of Massachusetts Amherst

slides credit: Akshay Krishnamurthy, Andrew McGregor, Dan Sheldon

September 4, 2018

COMPSCI 311: Introduction to Algorithms

- **Instructor**: Marius Minea
- **Where**: Hasbrouck Lab Addition 124
- **When**: Tue/Thu 11:30-12:45
- **Discussion Sections**: Fri 10:10 - 11:00, Hasbrouck Add 124
  Fri 11:15-12:05 Ag. Engr. 119. (please observe your section)
- **TA**: Jesse Lingeman, Rik Sengupta, Amir Ghafari
- **Office hours**:
  - Marius: Wed 5-6pm, LGRC A261
  - Jesse: Thu 10-11, CS207
  - Rik: Tue 1-2pm, CS207
  - Amir: Wed 1-2pm, CS207

What is Algorithm Design?

How do you write a computer program to solve a complex problem?

- Computing similarity between DNA sequences
- Routing packets on the Internet
- Scheduling final exams at a college
- Assign medical residents to hospitals
- Find all occurrences of a phrase in a large collection of documents
- Finding the smallest number of gas stations that can be built in the US such that everyone is within 20 minutes of a gas station.

DNA sequence similarity

- **Input**: two strings s_1 and s_2 of length n
  - $s_1 =$ AGGCTACC
  - $s_2 =$ CAGGCTAC
- **Output**: minimum number of insertions/deletions to transform s_1 into s_2
- **Algorithm**: ????
- Even if the objective is precisely defined, we are often not ready to start coding right away!

What is Algorithm Design?

- **Step 1**: **Formulate** the problem precisely
- **Step 2**: **Design** an algorithm
- **Step 3**: **Prove** the algorithm is correct
- **Step 4**: **Analyze** its running time

Important: this is an iterative process
Sometimes we don't get the algorithm right on the first try
Sometimes we'll redesign the algorithm to prove correctness easier or to make it more efficient

Usually, two steps:

- getting to a (mathematical) clean core of the problem
- identify the appropriate algorithm design techniques

Course Goals

- Learn how to apply the algorithm design *process*... by practice!
- Learn specific algorithm design techniques
  - Greedy
  - Divide-and-conquer
  - Dynamic Programming
  - Network Flows
- Learn to communicate precisely about algorithms
  - Proofs, reading, writing, discussion
- Prove when no exact efficient algorithm is possible
  - Intractability and NP-completeness

Prerequisites: CS 187 and 250

- Algorithms use data structures
- Familiarity
  - at programming level (lists, stacks, queues, ...)
  - with mathematical objects (sets, lists, relations, partial orders) precise statement of algorithm is in terms of such objects
- Two key notions to revisit:
  - Recursion: many algorithm classes are recursive so are most relations for computing algorithmic complexity
  - Proofs: to establish correctness and complexity often by induction

Proofs Are Important!

- Need to make sure algorithm is correct
- Think of special / corner cases
- Case in point: Timsort sorting algorithm was broken!
  - developed in 2002 (Python), adopted as standard sort in Java
  - tries to find and extend segments that are already sorted
  - uses stack to track segments and their lengths
  - loop invariant was not correctly reestablished
  - thus computed worst case stack size was wrong!
  - crash for array $>$ 67M elements
  - bug found and fixed in 2015 by theorem proving

Grading Breakdown

- **Participation (10%):** Discussion section, in-class quizzes (iClicker)
- **Homework (25%):** Homework (every two weeks, usually due Thursday) and online quiz (every weekend due Monday).
- **Midterm 1 (20%):** Focus on first third of lectures. 7pm Wed Oct 3
- **Midterm 2 (20%):** Focus on second third of lectures. 7pm Wed Nov 14
- **Final (25%):** Covers all lectures. 1pm, Wed Dec 19

Course Information

Course websites:

people.cs.umass.edu/~marius/class/cs311/	Course information, slides, homework, **pointers to all other pages**
moodle.umass.edu	Quizzes, solutions, grades
piazza.com	Discussion forum, contacting instructors and TA's
gradescope.com	Submitting and returning homework

Announcements: Check your UMass email daily.
Log into Piazza regularly for course announcements.

Homeworks and Quizzes

- *Online Quizzes:* Quizzes must be submitted before 8pm Monday. No late quizzes allowed but we'll ignore your lowest scoring quiz.

- *Homework:* Submit via Gradescope, by 11:59 pm of due date. 50% penalty for homework that is late up to 24 hours. Homework that is late by more than 24 hours receives no credit. One homework may be up to 24 hours late without penalty.

Collaboration and Academic Honesty

- *Homework:* Collaboration OK (and encouraged) on homework. But: you should read and attempt on your own first. The writeup and code **must** be your own. **Looking** at written solutions that are not your own is considered cheating. There will be formal action if cheating is suspected. You must list your collaborators and any sources (printed or online) at the top of each assignment.

- *Online Quizzes:* Should be done entirely on your own. You may consult the book and slides as you do the quiz. Again, there will be formal action if cheating is suspected.

- *Discussions:* Groups for the discussion section exercises will be assigned at the start of each session. You must complete the exercises with your assigned group.

- *Exams:* Closed book and no electronics. Cheating will result in an F in the course.

Stable Matching

- Real-life scenario
  - matching student interns to companies
  - or medical residents to hospitals
- Both students and companies have preferences / ranking lists
- If not properly managed, can become chaotic
  (assume participants are selfish, act in their own self-interest)
  - student may get better offer and reject current one
  - student may actively call company, see if they are preferred over the current status

Stable Matching and College Admissions

- Suppose there are n colleges $c_1, c_2, \ldots, c_n$ and n students $s_1, s_2, \ldots, s_n$.
- Each college has a ranking of all the students that they could admit and each student has a ranking of all the colleges.
  To simplify, suppose each college can only admit one student.
- What other simplification(s) have we made?
- n students, n colleges – could potentially match one-to-one
- *Matching*: a set of pairs (c, s) such that every college and every student appears in at most one pair
- *Perfect* matching: every student and college is matched

Defining Stability

- Can we match students to colleges such that everyone is *happy*?
  - Not necessarily, e.g., if UMass was everyone's top choice.
- Can we match students to colleges such that matching is *stable*?
  - Need to precisely define stability
- *(In)stability*: Don't want to match (c, s) and (c', s') if c and s' would prefer to switch and be matched with each other.
- *Unstable pair*: A pair (c, s) is *unstable* if c prefers s to matched student and s prefers c to matched college
- Are the two wordings equivalent?
- We'll see that a stable matching always exists and there's an *efficient* algorithm to find that matching.

Stable matching: quiz 1

Which pair is unstable in the matching { A–X, B–Z, C–Y } ?

A. A–Y.

B. B–X.

C. B–Z.

D. None of the above.

	1st	2nd	3rd
Atlanta	Xavier	Yolanda	Zeus
Boston	Yolanda	Xavier	Zeus
Chicago	Xavier	Yolanda	Zeus

	1st	2nd	3rd
Xavier	Boston	Atlanta	Chicago
Yolanda	Atlanta	Boston	Chicago
Zeus	Atlanta	Boston	Chicago

Propose-and-Reject (Gale-Shapley) Algorithm

```
Initially all colleges and students are free
while some college is free and hasn't proposed to every student do
    Choose such a college c
    Let s be the highest ranked student to whom c has not proposed
    if s is free then
        c and s become matched
    else if s is matched to c' but prefers c to c' then
        c' becomes unmatched
        c and s become matched
    else                                        ▷ s prefers c'
        s rejects c and c remains free
    end if
end while
```

Analyzing the Algorithm

- Some natural questions:
  - Can we guarantee the algorithm terminates?
  - Can we guarantee the every college and student gets a match?
  - Can we guarantee the resulting allocation is stable?

Need Precise Problem Definition

- These questions are non-obvious
- Answer may differ if we slightly change problem
- Does the following setup differ, and if so, how?

Stable roommate problem

Q. Do stable matchings always exist?
A. Not obvious a priori.

Stable roommate problem.

- $2n$ people; each person ranks others from 1 to $2n-1$.
- Assign roommate pairs so that no unstable pairs.

	1st	2nd	3rd
A	B	C	D
B	C	A	D
C	A	B	D
D	A	B	C

no perfect matching is stable

A–B, C–D ⇒ B–C unstable
A–C, B–D ⇒ A–B unstable
A–D, B–C ⇒ A–C unstable

Observation. Stable matchings need not exist.

10

slide credit: Kevin Wayne / Pearson

Analyzing the Algorithm

- Some initial observations:
  - (F1) Once matched, students stay matched and only "upgrade" during the algorithm.
  - (F2) College propose to students in order of college's preferences.

Stable matching: quiz 2

Do all executions of Gale–Shapley lead to the same stable matching?

A. No, because the algorithm is nondeterministic.

B. No, because an instance can have several stable matchings.

C. Yes, because each instance has a unique stable matching.

D. Yes, even though an instance can have several stable matchings and the algorithm is nondeterministic.

slide credit: Kevin Wayne / Pearson

Can we guarantee the algorithm terminates?

- Yes! Proof...
  - In every round, some college proposes to some student that they haven't already proposed to.
  - n colleges and n students $\Longrightarrow$ at most n^2 proposals
  - $\Longrightarrow$ at most n^2 rounds of the algorithm

Can we guarantee all colleges and students get a match?

- Yes! Proof by contradiction...
  - Suppose not all colleges and students have matches. Then there exists unmatched college c and unmatched student s.
  - s was never matched during the algorithm (by F1)
  - But c proposed to every student (by termination condition)
  - When c proposed to s, she was unmatched and yet rejected c. Contradiction!

Can we guarantee the resulting allocation is stable?

- Yes! Proof by contradiction with a case analysis...
  - Suppose there is an instability (c, s)
    - c is matched to some s' but prefers s to s'
    - s is matched to some c' but prefers c to c'
  - Case 1: c has already offered to s
    - Since s isn't matched to c at the end of the algorithm, she must have rejected c's offer at some point and therefore be matched to a college she prefers to c (by F1). Contradiction.
  - Case 2: c did not offer to s
    - We know c proposed to and was matched to s'. Since s' is less preferred, c must have also proposed to s (by F2). Contradiction. (This case cannot happen.)

2012 Nobel Prize in Economics

Lloyd Shapley. Stable matching theory and Gale–Shapley algorithm.

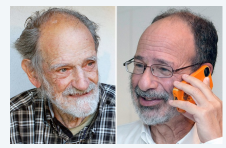
COLLEGE ADMISSIONS AND THE STABILITY OF MARRIAGE

D. GALE* AND L. S. SHAPLEY, Brown University and the RAND Corporation

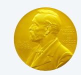
1. Introduction. The problem with which we shall be concerned relates to the following typical situation: A college is considering a set of n applicants of which it can admit a quota of only q. Having evaluated their qualifications, the admissions office must decide which ones to admit. The procedure of offering admission only to the q best-qualified applicants will not generally be satisfactory, for it cannot be assumed that all who are offered admission will accept.

original applications: college admissions and opposite-sex marriage

Alvin Roth. Applied Gale–Shapley to matching med-school students with hospitals, students with schools, and organ donors with patients.



Lloyd Shapley Alvin Roth



slide credit: Kevin Wayne / Pearson

A modern application

Content delivery networks. Distribute much of world's content on web.

User. Preferences based on latency and packet loss.
Web server. Preferences based on costs of bandwidth and co-location.
Goal. Assign billions of users to servers, every 10 seconds.

Algorithmic Nuggets in Content Delivery

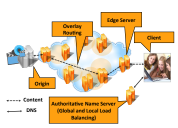
Bruce M. Maggs
Duke and Akamai
bmm@cs.duke.edu

Ramesh K. Sitaraman
UMass, Amherst and Akamai
ramesh@cs.umass.edu

This article is an editorial note submitted to CCR. It has NOT been peer reviewed.
The authors take full responsibility for this article's technical content. Comments can be posted through CCR Online.

ABSTRACT

This paper "peeks under the covers" at the subsystems that provide the basic functionality of a leading content delivery network. Based on our experiences in building one of the largest distributed systems in the world, we illustrate how sophisticated algorithmic research has been adapted to balance the load between and within server clusters, manage the caches on servers, select paths through an overlay routing network, and elect leaders in various contexts. In each instance, we first explain the theory underlying the algorithms, then introduce practical considerations not captured by the theoretical models, and finally describe what is implemented in practice. Through these examples, we highlight the role of algorithmic research in the design of complex networked systems. The paper also illustrates the close synergy that exists between research and industry where research ideas cross over into products and product requirements drive future research.



slide credit: Kevin Wayne / Pearson

For Thursday

- Think about:
  - Would it be better or worse for the students if we ran the algorithm with the students proposing?
  - Can a student get an advantage by lying about their preferences?
- Read: Chapter 1, course policies
- Enroll in Piazza, log into Moodle, and visit the course webpage.