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

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 $n$
  - $s_1 = \text{AGGCTACC}$
  - $s_2 = \text{CAGGCTAC}$
- 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!

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.
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

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.

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 rejects c and c remains free
    ▷ s prefers c’
  end if
end while

Stable Matching and College Admissions

- Suppose there are n colleges c₁, c₂, . . . , cₙ and n students
  s₁, s₂, . . . , sₙ.
- 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

Stable Matching: quiz 1

Which pair is unstable in the matching { A–X, B–Z, C–Y }?
A.  q–YZ
B.  r–XZ
C.  r–ZZ
D. LisyoneLofLtheLaboveZ

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.

<table>
<thead>
<tr>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>C</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>D</td>
<td>A</td>
<td>C</td>
</tr>
</tbody>
</table>

A–DXLB–C L⇒LLLLA–CLunstable

A–DXLB–C L⇒LLLLA–CLunstable

A–DXLB–C L⇒LLLLA–CLunstable

Observation. Stable matchings need not exist.

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.

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 \(\Rightarrow\) at most \(n^2\) proposals
  - \(\Rightarrow\) 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 instablility \((c, s)\)
  - \(c\) 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.)

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**
bruce@cs.duke.edu

**Ramesh K. Sitaraman**
ramesh@cs.umass.edu

**Authoritative Name Server**

- **Origin**
- **Round**
- **Overlay**
- **Balancing**

**Volume**

**Country**

**Number**

**July**

**2012 Nobel Prize in Economics**

**Lloyd Shapley.** Stable matching theory and Gale-Shapley algorithm.

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

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.