

CSC 301: Analysis of Algorithms

Lecture 1: Introduction and Stable Matching

William Rebelsky
[rebelskyw]

Grinnell College

Monday 22nd January, 2024

Some slides modified from those used by Professors Akshay Krishnamurthy, Andrew McGregor, Marius Minea, Ghazaleh Parvini, and Dan Sheldon at the University of Massachusetts Amherst

Table of Contents

1 Course Logistics

2 Algorithmic Design

3 Stable Matching

Logistics

- Course website: tbd
- Syllabus available at: tbd
- Meeting Times: 10-10:50 or 3-3:50 MWF
- Mentors: Hyeyun Jung (Morning), Bradley Ramsey (Afternoon)
- Office Hours: tbd

Course Design

- Monday and Wednesday: mostly lecture, some practice problems
- Friday (other than this week): Discussion sections—practice applying the content from the week. Given a problem, work in small groups to figure out a solution.
- Weekly Problem Sets: released Wednesday, due following Wednesday at 10:30 PM
- Weekly Quizzes: released Friday, due following Monday at 10:30PM
 - 30 min time limit, expected to take 10–15 minutes
 - If a Monday due date becomes problematic, it can be moved to Tuesday
- Midterm (format to be determined): Week 7 (before Spring Break)
- Project (format to be determined): End of semester

Course Goals

Upon completion of this course, students will:

- 1 be familiar with and apply classic algorithms, ADTs and data structures
- 2 learn to read the literature of computer science, specifically within the domain of algorithms
- 3 be able to explain why comparison-based sorts take order $n \log n$ in the worst case
- 4 have a better understanding of the interplay between algorithms, technology, bias, and structural inequity
- 5 improve their ability to justify the runtime and correctness of their algorithms.

Grading

- Grading will be based on ideas of Specifications and Mastery Grading
- Tokens may be used to resubmit non-satisfactory work or make up for a missed class
 - Start with 3 tokens. More can be earned through attending various events and writing reflections

Grade	Attendance (out of 42)	Problem Sets (out of 10)	Quizzes (out of 11)	Essay	Midterm and Project
C	36	7	7	1	0
B	38	8	9	1	1
A	40	10	11	1	2

Disclaimers

- Grading scale may change during the semester if the number of problem sets, quizzes, etc. changes
- If you have any events you would like your classmates to go to, please let me know so that I can add them to the tokens list
- If you have feedback during the course of things that are going well or going poorly, please let me know
- I will do my best with names, but like Professor Samuel Rebelsky, evidence suggests that I am not just bad at remembering names, I also have some brain differences that regularly lead me to mix up peoples' names

Table of Contents

1 Course Logistics

2 Algorithmic Design

3 Stable Matching

What is Algorithmic Design?

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

What is Algorithmic 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

What is Algorithmic Design?

- This is an iterative process
 - May not be correct on the first try
 - May redesign the algorithm to be more efficient
 - May redesign the algorithm to make it easier to prove correctness
- Two main design steps:
 - Figure out the clean core of the problem using mathematical objects: sequences, sets, graphs, etc
 - Identify the appropriate algorithm design techniques

Proofs are Important

- An efficient, incorrect algorithm that is believed to be correct is worse than an inefficient, correct algorithm
- Consider edge cases
- Always **Prove** the algorithm is correct in all cases (I will be strict on proofs)
- Real world example: Timsort
 - Developed in 2002 (Python)
 - Adopted as the standard sort in Java
 - Bug found (and fixed) in 2015
- We ask for proofs so that you can detect if your design is incorrect (frequent errors: incorrect design technique, superficial proof that misses key parts)

Table of Contents

1 Course Logistics

2 Algorithmic Design

3 Stable Matching

Let's Practice

Today: Stable Matching theory and Gale-Shapely algorithm
(original paper here)

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

- Real-Life Scenarios:
 - Matching residents to hospitals
 - students to colleges
- Simplified Scenario for today:
 - n colleges, n students
 - each party has a complete preference list (no gaps, no ties, well ordered)
 - Goal: Match 1 college to 1 student
- Definitions:
 - **Matching:** a set of pairs (c,s) , where each college and student appears in at most one pair
 - **Perfect Matching:** every student and college is matched
 - **Instability or unstable pair** a pair that is **not** in the matching that prefers each other to their assigned matchings
 - **Stable Matching:** a matching with no instabilities

Questions to answer today (with examples)

- 1 Do stable matchings always exist?
- 2 If so, are they unique? (hint try a 2×2 set)
- 3 How would you design an algorithm for stable matching?

BLANK

This frame intentionally left blank

Answers

- ❶ Do stable matchings always exist? **YES**
- ❷ If so, are they unique? **NO**
 - College A: $1 > 2$
 - College B: $2 > 1$
 - Student 1: $B > A$
 - Student 2: $A > B$
 - $M_1 = \{(A, 1), (B, 2)\}$ stable
 - $M_2 = \{(A, 2), (B, 1)\}$ stable
- ❸ How would you design an algorithm for stable matching?
Propose and Reject

Gale-Shapely

- Idea: Build matching incrementally
- Group 1 (Colleges): active. Propose, going down preference lists
- Group 2 (students): passive. Accept (if better offer) or reject if not

Gale-Shapely

Algorithm 1: Gale-Shapely(C, S)

Input: Preferences $c_1, \dots, c_n$ (active) and $s_1, \dots, s_n$ (passive)

Output: Stable Matching

```
1 Initialization: no matches
2 while  $\exists$  unmatched  $c_i$  do
3   Choose a member  $c_i$ 
4   Let  $s_j$  be the highest ranked passive student to whom  $c_i$ 
     has not proposed
5   if  $s_j$  is free then
6      $c_i$  and  $s_j$  become matched
7   else if  $s_j$  is matched to some  $c_k$  but prefers  $c_i$  to  $c_k$  then
8      $c_k$  becomes unmatched
9      $c_i$  and  $s_j$  become matched
10  else
11     $s_j$  rejects  $c_i$  and  $c_i$  remains unmatched
12 return (matches)
```

Algorithm Analysis of Gale-Shapely

- Can we guarantee termination?
- Can we guarantee perfect matching?
- Can we guarantee stability?
- If multiple possible solutions, which is chosen?

Termination of Gale-Shapely

- In every round, some college proposes to some student that they haven't proposed to
- n colleges, n students $\implies$ at most n^2 proposals
- $\implies$ at most n^2 rounds of the algorithm
- $\implies$ algorithm terminates (and gives us a bound on time complexity of $O(n^2)$)

Termination of Gale-Shapely

- Is the result a perfect matching? Yes
- Is the resulting matching stable? Yes
- Proofs will be part of homework 0

Interesting Notes

- Making the best local choice at each step leads to a global optimum. We will see this more later with greedy algorithms.
- Nondeterministic: any free college can propose (and you get the same, unique result)
 - of the possible stable matchings, Gale-Shapely will produce the best matching for the proposing group (colleges), and the worst possible matching for the accepting group (students)
- It is more efficient than the brute force approach of generating all perfect matchings and checking them for stability $O(n!)$