Design and Analysis of Algorithms

CS2800

Jayalal Sarma
jayalal@cse.iitm.ac.in
Office: SSB 306
Ph : 9840920902

Plan for today

1. Introduction
2. Administrative Details
3. JOSAA and SEAT

Introduction

Welcome to the course

Welcome to the course

An algorithm is an explicit, precise, unambiguous, mechanically-executable sequence of elementary instructions, usually intended to accomplish a specific purpose.

Welcome to the course

An algorithm is an explicit, precise, unambiguous, mechanically-executable sequence of elementary instructions, usually intended to accomplish a specific purpose. Of course !

Welcome to the course

An algorithm is an explicit, precise, unambiguous, mechanically-executable sequence of elementary instructions, usually intended to accomplish a specific purpose. Of course !

- What is this course about?

Welcome to the course

An algorithm is an explicit, precise, unambiguous, mechanically-executable sequence of elementary instructions, usually intended to accomplish a specific purpose. Of course !

- What is this course about?
  - Design algorithms -

Welcome to the course

An algorithm is an explicit, precise, unambiguous, mechanically-executable sequence of elementary instructions, usually intended to accomplish a specific purpose. Of course !

- What is this course about?
    - Design algorithms - Haven't I been doing this already?

Welcome to the course

An algorithm is an explicit, precise, unambiguous, mechanically-executable sequence of elementary instructions, usually intended to accomplish a specific purpose. Of course !

- What is this course about?
    - Design algorithms - Haven't I been doing this already?
    - Prove correctness -

Welcome to the course

An algorithm is an explicit, precise, unambiguous, mechanically-executable sequence of elementary instructions, usually intended to accomplish a specific purpose. Of course !

- What is this course about?
  - Design algorithms - Haven't I been doing this already?
  - Prove correctness - All the implementations that I write work for all the test cases!

Welcome to the course

An algorithm is an explicit, precise, unambiguous, mechanically-executable sequence of elementary instructions, usually intended to accomplish a specific purpose. Of course !

- What is this course about?
    - Design algorithms - Haven't I been doing this already?
    - Prove correctness - All the implementations that I write work for all the test cases!
    - Analyze complexity -

Welcome to the course

An algorithm is an explicit, precise, unambiguous, mechanically-executable sequence of elementary instructions, usually intended to accomplish a specific purpose. Of course !

- What is this course about?
  - Design algorithms - Haven't I been doing this already?
  - Prove correctness - All the implementations that I write work for all the test cases!
  - Analyze complexity - All the programs I write run very fast on my computer!

Integer multiplication

Integer multiplication

- Multiplication is repeated addition: $3 \times 5 = 3 + 3 + 3 + 3 + 3$

Integer multiplication

- Multiplication is repeated addition: $3 \times 5 = 3 + 3 + 3 + 3 + 3$
- What about the way you multiplied numbers in school?

Integer multiplication

- Multiplication is repeated addition: $3 \times 5 = 3 + 3 + 3 + 3 + 3$
- What about the way you multiplied numbers in school?
    - Learn to multiply single digits - multiplication tables!

Integer multiplication

- Multiplication is repeated addition: $3 \times 5 = 3 + 3 + 3 + 3 + 3$
- What about the way you multiplied numbers in school?
  - Learn to multiply single digits - multiplication tables!
  - Learn to multiply large numbers with single digits

Integer multiplication

- Multiplication is repeated addition: $3 \times 5 = 3 + 3 + 3 + 3 + 3$
- What about the way you multiplied numbers in school?
    - Learn to multiply single digits - multiplication tables!
    - Learn to multiply large numbers with single digits
    - Given $a = a_m a_{m-1} \ldots a_0$ and $b = b_n b_{n-1} \ldots b_0$ do the following:
        - Do $c_0 = a \times b_0$, $c_1 = a \times b_1$, . . ., $c_n = a \times b_n$

Integer multiplication

- Multiplication is repeated addition: $3 \times 5 = 3 + 3 + 3 + 3 + 3$
- What about the way you multiplied numbers in school?
  - Learn to multiply single digits - multiplication tables!
  - Learn to multiply large numbers with single digits
  - Given $a = a_m a_{m-1} \ldots a_0$ and $b = b_n b_{n-1} \ldots b_0$ do the following:
    - Do $c_0 = a \times b_0$, $c_1 = a \times b_1$, . . ., $c_n = a \times b_n$
    - Write down the number $\sum_{i=0}^{n} c_i \times 10^i$.

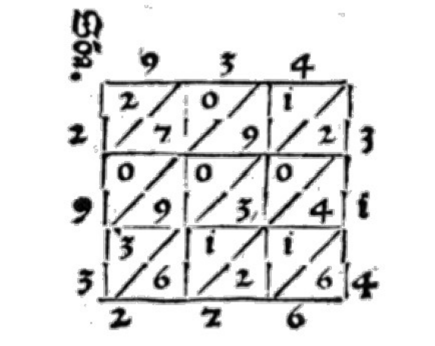


Figure 1: Anonymous 1458 textbook - Treviso Arithmetic

Integer multiplication

Peasant's multiplication (1650 B.C)
Given two numbers a, b
Start with $c = 0$, until $a = 0$

- Check parity of a
- Addition - If a is odd, do $c = c + b$
- Duplation - $b = 2 \times b$
- Mediation - $a = \lfloor \frac{a}{2} \rfloor$

a	b	c
		0
35	+46	46
17	+92	138
8	184	138
4	368	138
2	736	138
1	+1472	1610

Integer multiplication

Peasant's multiplication (1650 B.C)

Given two numbers a, b

Start with $c = 0$, until $a = 0$

- Check parity of a
- Addition - If a is odd, do $c = c + b$
- Duplation - $b = 2 \times b$
- Mediation - $a = \lfloor \frac{a}{2} \rfloor$

a	b	c
		0
35	+46	46
17	+92	138
8	184	138
4	368	138
2	736	138
1	+1472	1610

How do we know if this algorithm is correct?

Integer multiplication

Peasant's multiplication (1650 B.C)
Given two numbers a, b
Start with $c = 0$, until $a = 0$

- Check parity of a
- Addition - If a is odd, do $c = c + b$
- Duplation - $b = 2 \times b$
- Mediation - $a = \lfloor \frac{a}{2} \rfloor$

a	b	c
		0
35	+46	46
17	+92	138
8	184	138
4	368	138
2	736	138
1	+1472	1610

How do we know if this algorithm is correct?

What is the running time of this algorithm?

Integer multiplication

Peasant's multiplication (1650 B.C)

Given two numbers a, b

Start with $c = 0$, until $a = 0$

- Check parity of a
- Addition - If a is odd, do $c = c + b$
- Duplation - $b = 2 \times b$
- Mediation - $a = \lfloor \frac{a}{2} \rfloor$

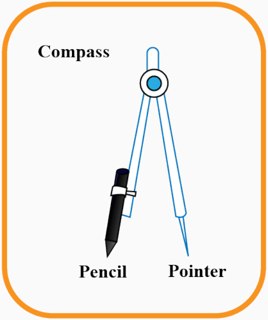
a	b	c
		0
35	+46	46
17	+92	138
8	184	138
4	368	138
2	736	138
1	+1472	1610

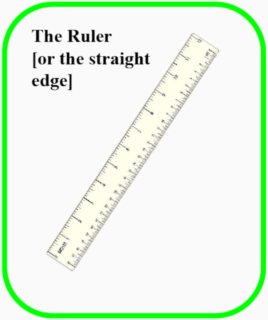
How do we know if this algorithm is correct?

What is the running time of this algorithm?

Measured in terms of what?

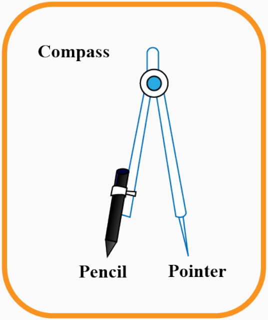
Detour : Ruler and Compass Constructions

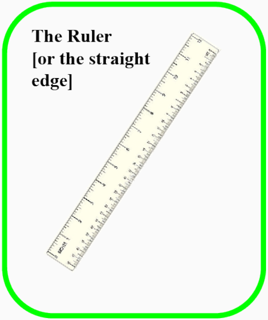




- You have a ruler and a compass and a pencil to draw lines.

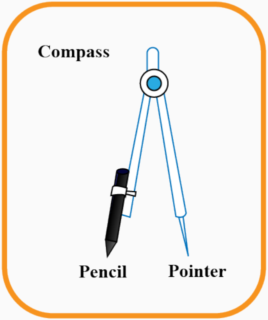
Detour : Ruler and Compass Constructions

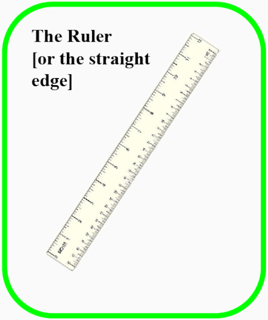




- You have a ruler and a compass and a pencil to draw lines.
- And you are given a line segment which is of length 2 units.

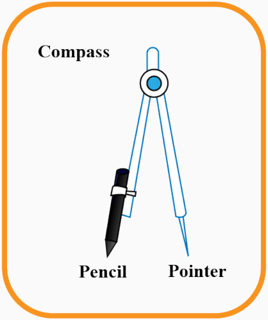
Detour : Ruler and Compass Constructions

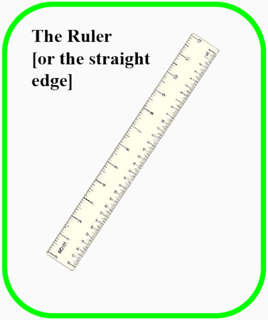




- You have a ruler and a compass and a pencil to draw lines.
- And you are given a line segment which is of length 2 units.
- Aim : to bisect this line

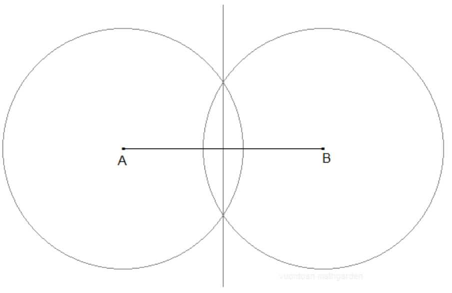
Detour : Ruler and Compass Constructions





- You have a ruler and a compass and a pencil to draw lines.
- And you are given a line segment which is of length 2 units.
- Aim : to bisect this line - that is, to produce line segments of 1 unit length each.

Bisection !



What else can be done?

- Can you construct an equivalenteral triangle?

What else can be done?

- Can you construct an equivalenteral triangle?
- Can you construct a square?

What else can be done?

- Can you construct an equivalenteral triangle?
- Can you construct a square?
- Can you construct a polygon with 10 sides?

What else can be done?

- Can you construct an equivalenteral triangle?
- Can you construct a square?
- Can you construct a polygon with 10 sides?

What else can be done?

- Can you construct an equivalenteral triangle?
- Can you construct a square?
- Can you construct a polygon with 10 sides?

Are there things that cannot be done?

What else can be done?

- Can you construct an equivalenteral triangle?
- Can you construct a square?
- Can you construct a polygon with 10 sides?

Are there things that cannot be done? (A question from ancient greeks !)

Gauss ....

Gauss .... Carl Frederich Gauss

1796 : He devised a method for consrtucting the regular 17-gon.

Gauss ....

1796 : He devised a method for consrtucting the regular 17-gon.

About 40 years later - he had a characterization of when it is possible and when is it not.

[**Gauss-Wantzel Theorem** - 1837] A regular n-gon can be constructed with compass and straightedge

$$\text{if and only if}$$

n is a power of 2 or the product of a power of 2 and any number of distinct Fermat primes.

Coming back : Integer multiplication

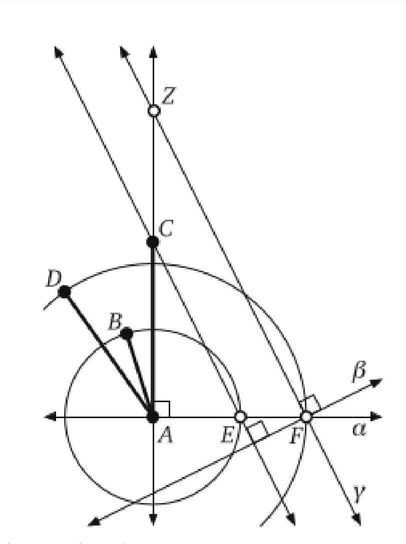
Question Can you perform multiplication using a compass and straight-edge?

Coming back : Integer multiplication

Question Can you perform multiplication using a compass and straight-edge?

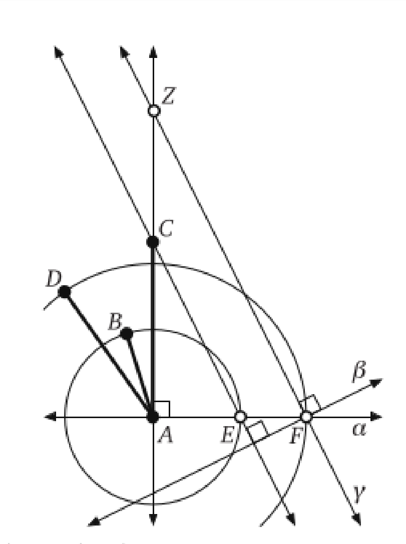
> Given two line segments of length a and b, with one endpoint as the origin and another line segment of unit length, construct a line segment of length $a \times b$

What if a and b are not integers



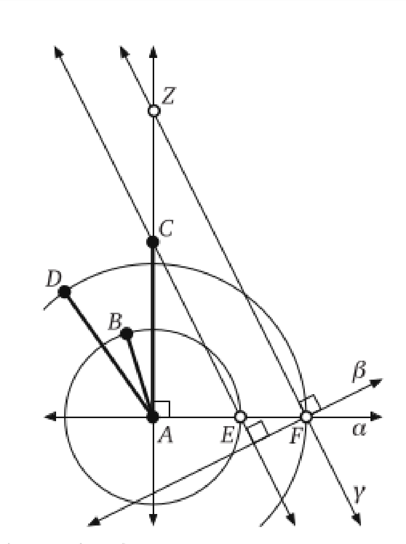
Why is this method correct?

Integer multiplication



Why is this method correct?

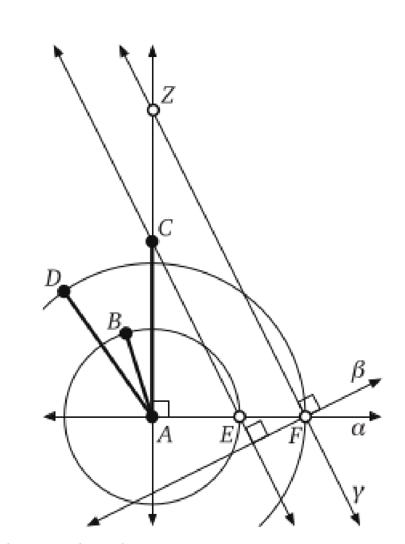
- Hint : Triangles AEC and AFZ are similar !!.



Why is this method correct?

- Hint : Triangles AEC and AFZ are similar !!.
- Prove: $|AZ| = \frac{|AD| \times |AC|}{|AB|}$

Integer multiplication



Why is this method correct?

- Hint : Triangles AEC and AFZ are similar !!.
- Prove: $|\mathrm{AZ}| = \frac{|\mathrm{AD}| \times |\mathrm{AC}|}{|\mathrm{AB}|}$
- Use the unit length given as the line AB.

Integer multiplication

- Which is a better algorithm or are they all equally good?

Integer multiplication

- Which is a better algorithm or are they all equally good?
- How do we measure the goodness of an algorithm?

Integer multiplication

- Which is a better algorithm or are they all equally good?
- How do we measure the goodness of an algorithm?
- Are there algorithms for integer multiplication that are better?
  - When do we decide to stop searching for better algorithms?

Administrative Details

Administrative details

- When: 'D' slot
  - 3 lectures (Mon, Tue, Wed) + 1 tutorial (Thu)
- Contact: By email or WA.
- Discussion Page : https://edstem.org
- Course page: `www.cse.iitm.ac.in/~jayalal/teaching/home.php?courseid=82`
- TAs: Ayman, Chahel, Dinesh, Harish, Nagashri, Parag, Raju, Rupankar, Sambhav, Shaun, Subramanian, Tejasvi.

Administrative details

Grading scheme:

Relative grading

- Best 3 out of 4 short exams: $3 \times 6 = 18\%$.
- Two Quizzes - 1 & 2: $2 \times 20\%$ = 30%
- End-sem Exam: 42%

Attendance requirements: as mandated by the institute.

Course Textbooks

We will not be following one textbook. Our main sources of reference will be the following textbooks.

- **E** - Algorithms by Jeff Erickson.
- **KT** - Algorithm Design by Jon Klienberg and Eva Tardos.
- **CLRS** - Introduction to Algorithms- Cormen, Leiserson, Rivest & Stein.
- **DPV** - Algorithms by Dasgupta, Papadimitriou and Vazirani.

What constitutes a good algorithm?

- Proposed procedure in unambiguous terms.
- Proof of termination.
- Proof correctness.
- Proof of running time or space used by the algorithm.

What constitutes a good algorithm?

- Proposed procedure in unambiguous terms.
- Proof of termination.
- Proof correctness.
- Proof of running time or space used by the algorithm.
  this may involve using the right data structures.
- Proof of tightness, if possible.

JOSAA and SEAT

SEAT algorithm at IITM

Student **E**lective **A**llocation **T**ool :

`http://www.cse.iitm.ac.in/~meghana/seat/`

SEAT algorithm at IITM

Student **E**lective **A**llocation **T**ool :

`http://www.cse.iitm.ac.in/~meghana/seat/`

- Set of students with preference order for courses offered in the institute
- Set of courses with some preference order for students (GPA based?)

SEAT algorithm at IITM

Student **E**lective **A**llocation **T**ool :

`http://www.cse.iitm.ac.in/~meghana/seat/`

- Set of students with preference order for courses offered in the institute
- Set of courses with some preference order for students (GPA based?)
- Allocate students to courses so that the course instructors and students are ”happy”

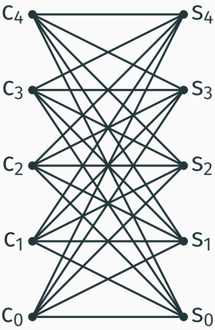
SEAT algorithm at IITM

Student **E**lective **A**llocation **T**ool :

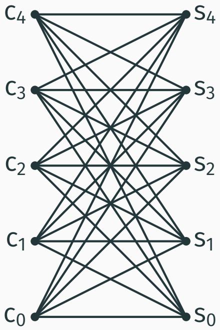
http://www.cse.iitm.ac.in/~meghana/seat/

- Set of students with preference order for courses offered in the institute
- Set of courses with some preference order for students (GPA based?)
- Allocate students to courses so that the course instructors and students are ”happy”
    - How should we define ”happy”?

The Stable Marriage problem



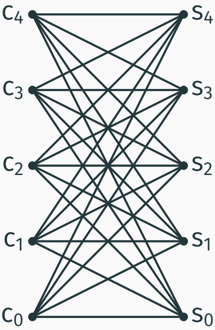
The Stable Marriage problem



Preference orders for students and courses

- $S_1 : (C_0, C_1, C_2, C_3, C_4)$,
  $S_2 : (C_0, C_3, C_4, C_1, C_2), \ldots$
- $C_0 : (S_1, S_3, S_4, S_0, S_2)$,
  $C_3 : (S_2, S_1, S_0, S_4, S_3), \ldots$

The Stable Marriage problem

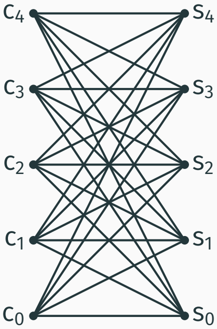


Preference orders for students and courses

- $S_1 : (C_0, C_1, C_2, C_3, C_4)$,
  $S_2 : (C_0, C_3, C_4, C_1, C_2), \ldots$
- $C_0 : (S_1, S_3, S_4, S_0, S_2)$,
  $C_3 : (S_2, S_1, S_0, S_4, S_3), \ldots$

- If S_1 is assigned the course C_0, then S_1 and C_0 are happy!

The Stable Marriage problem

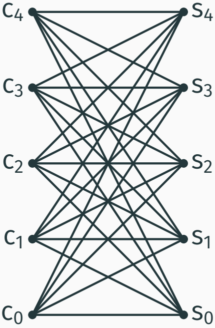


Preference orders for students and courses

- $S_1 : (C_0, C_1, C_2, C_3, C_4)$,
  $S_2 : (C_0, C_3, C_4, C_1, C_2), \ldots$
- $C_0 : (S_1, S_3, S_4, S_0, S_2)$,
  $C_3 : (S_2, S_1, S_0, S_4, S_3), \ldots$

- If S_1 is assigned the course C_0, then S_1 and C_0 are happy!
- If S_2 is assigned the course C_3, he/she may not be happy but cannot do anything about it! (why?)

The Stable Marriage problem

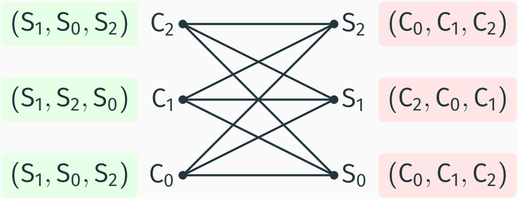


Preference orders for students and courses

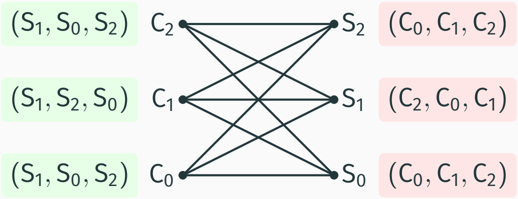
- $S_1 : (C_0, C_1, C_2, C_3, C_4)$,
  $S_2 : (C_0, C_3, C_4, C_1, C_2), \ldots$
- $C_0 : (S_1, S_3, S_4, S_0, S_2)$,
  $C_3 : (S_2, S_1, S_0, S_4, S_3), \ldots$

- If S_1 is assigned the course C_0, then S_1 and C_0 are happy!
- If S_2 is assigned the course C_3, he/she may not be happy but cannot do anything about it! (why?)
- The notion of happiness has to be modified to one of stability

Stable marriages

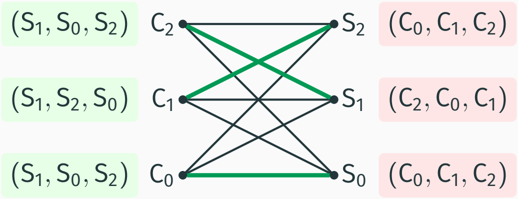


Stable marriages



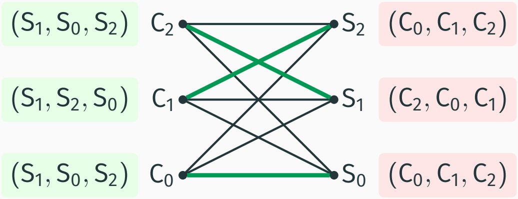
- How should the students be mapped to the courses?

Stable marriages



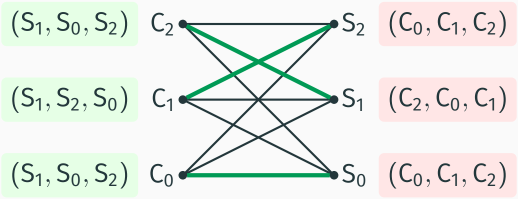
- How should the students be mapped to the courses?
  - $S_0 - C_0$, $S_1 - C_2$, $S_2 - C_1$

Stable marriages



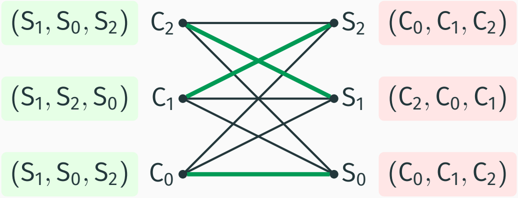
- How should the students be mapped to the courses?
  - $S_0 - C_0$, $S_1 - C_2$, $S_2 - C_1$
- Should such mappings always exist?

Stable marriages



- How should the students be mapped to the courses?
  - $S_0 - C_0$, $S_1 - C_2$, $S_2 - C_1$
- Should such mappings always exist?
- Can there be more than one such mapping? If so, which one is "better"?

Stable marriages



- How should the students be mapped to the courses?
  - $S_0 - C_0$, $S_1 - C_2$, $S_2 - C_1$
- Should such mappings always exist?
- Can there be more than one such mapping? If so, which one is "better"?
- If a mapping exists, how will we find it?

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.

Stable marriages

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.

We contend that the difficulties here described can be avoided. We shall describe a procedure for assigning applicants to colleges which should be satisfactory to both groups, which removes all uncertainties and which, assuming there are enough applicants, assigns to each college precisely its quota.

Stable marriages

How Game Theory Helped Improve New York City's High School Application Process

Stable marriages

How Game Theory Helped Improve New York City's High School Application Process

- Nobel prize in Economics (2012) for Shapley and Roth - ”for the theory of stable allocations and the practice of market design”

Matchings and unstable pairs

Input: Set of students, courses and the list of preferences

	1^{st}	2^{nd}	3^{rd}
C_0	S_0	S_1	S_2
C_1	S_1	S_0	S_2
C_2	S_0	S_1	S_2

	1^{st}	2^{nd}	3^{rd}
S_0	C_1	C_0	C_2
S_1	C_0	C_1	C_2
S_2	C_0	S_1	S_2

Matchings and unstable pairs

Input: Set of students, courses and the list of preferences

	1^{st}	2^{nd}	3^{rd}
C_0	S_0	S_1	S_2
C_1	S_1	S_0	S_2
C_2	S_0	S_1	S_2

	1^{st}	2^{nd}	3^{rd}
S_0	C_1	C_0	C_2
S_1	C_0	C_1	C_2
S_2	C_0	S_1	S_2

Output: A matching between the set of courses and the set of students.

Matchings and unstable pairs

Input: Set of students, courses and the list of preferences

	1^{st}	2^{nd}	3^{rd}
C_0	S_0	S_1	S_2
C_1	S_1	S_0	S_2
C_2	S_0	S_1	S_2

	1^{st}	2^{nd}	3^{rd}
S_0	C_1	C_0	C_2
S_1	C_0	C_1	C_2
S_2	C_0	S_1	S_2

Output: A matching between the set of courses and the set of students.

- Each course is assigned exactly one student.
- Each student is allotted exactly one course.

Matchings and unstable pairs

Input: Set of students, courses and the list of preferences

	1^{st}	2^{nd}	3^{rd}
C_0	S_0	S_1	S_2
C_1	S_1	S_0	S_2
C_2	S_0	S_1	S_2

	1^{st}	2^{nd}	3^{rd}
S_0	C_1	C_0	C_2
S_1	C_0	C_1	C_2
S_2	C_0	S_1	S_2

Output: A matching between the set of courses and the set of students.

- Each course is assigned exactly one student.
- Each student is allotted exactly one course.

Matchings and unstable pairs

	1^{st}	2^{nd}	3^{rd}
C_0	S_0	S_1	S_2
C_1	S_1	S_0	S_2
C_2	S_0	S_1	S_2

	1^{st}	2^{nd}	3^{rd}
S_0	C_1	C_0	C_2
S_1	C_0	C_1	C_2
S_2	C_0	C_1	C_2

Matching: $S_0 \leftrightarrow C_2$, $S_1 \leftrightarrow C_1$, $S_2 \leftrightarrow C_0$

Matchings and unstable pairs

	1^{st}	2^{nd}	3^{rd}
C_0	S_0	S_1	S_2
C_1	S_1	S_0	S_2
C_2	S_0	S_1	S_2

	1^{st}	2^{nd}	3^{rd}
S_0	C_1	C_0	C_2
S_1	C_0	C_1	C_2
S_2	C_0	C_1	C_2

Matching: $S_0 \leftrightarrow C_2$, $S_1 \leftrightarrow C_1$, $S_2 \leftrightarrow C_0$

Unstable pair A pair (S_i, C_j) is said to be an unstable pair if

- S_i prefers C_j to the currently allotted course, and
- C_j prefers S_i to the currently assigned student

Matchings and unstable pairs

	1st	2nd	3rd
C_0	S_0	S_1	S_2
C_1	S_1	S_0	S_2
C_2	S_0	S_1	S_2

	1st	2nd	3rd
S_0	C_1	C_0	C_2
S_1	C_0	C_1	C_2
S_2	C_0	C_1	C_2

Matching: $S_0 \leftrightarrow C_2$, $S_1 \leftrightarrow C_1$, $S_2 \leftrightarrow C_0$

Unstable pair A pair (S_i, C_j) is said to be an unstable pair if

- S_i prefers C_j to the currently allotted course, and
- C_j prefers S_i to the currently assigned student

Question Are there any unstable pairs in the matching above?

Matchings and unstable pairs

	1^{st}	2^{nd}	3^{rd}
C_0	S_0	S_1	S_2
C_1	S_1	S_0	S_2
C_2	S_0	S_1	S_2

	1^{st}	2^{nd}	3^{rd}
S_0	C_1	C_0	C_2
S_1	C_0	C_1	C_2
S_2	C_0	C_1	C_2

Matching: $S_0 \leftrightarrow C_2$, $S_1 \leftrightarrow C_1$, $S_2 \leftrightarrow C_0$

Unstable pair A pair (S_i, C_j) is said to be an unstable pair if

- S_i prefers C_j to the currently allotted course, and
- C_j prefers S_i to the currently assigned student

Question Are there any unstable pairs in the matching above?

An unstable pair can improve their mutual happiness by breaking the current matching!

Stable matchings

A **stable matching** is a matching where every course is allotted a student such that there are no unstable pairs

Stable matchings

A **stable matching** is a matching where every course is allotted a student such that there are no unstable pairs

Stable Marriage problem Given a set of courses, students and preference lists, find a stable matching (if one exists)

Stable matchings

A **stable matching** is a matching where every course is allotted a student such that there are no unstable pairs

Stable Marriage problem Given a set of courses, students and preference lists, find a stable matching (if one exists)

	1^{st}	2^{nd}	3^{rd}
C_0	S_0	S_1	S_2
C_1	S_1	S_0	S_2
C_2	S_0	S_1	S_2

	1^{st}	2^{nd}	3^{rd}
S_0	C_1	C_0	C_2
S_1	C_0	C_1	C_2
S_2	C_0	C_1	C_2

Stable matchings

A **stable matching** is a matching where every course is allotted a student such that there are no unstable pairs

Stable Marriage problem Given a set of courses, students and preference lists, find a stable matching (if one exists)

	1^{st}	2^{nd}	3^{rd}
C_0	S_0	S_1	S_2
C_1	S_1	S_0	S_2
C_2	S_0	S_1	S_2

	1^{st}	2^{nd}	3^{rd}
S_0	C_1	C_0	C_2
S_1	C_0	C_1	C_2
S_2	C_0	C_1	C_2

Stable matchings

A **stable matching** is a matching where every course is allotted a student such that there are no unstable pairs

Stable Marriage problem Given a set of courses, students and preference lists, find a stable matching (if one exists)

	1^{st}	2^{nd}	3^{rd}
C_0	S_0	S_1	S_2
C_1	S_1	S_0	S_2
C_2	S_0	S_1	S_2

	1^{st}	2^{nd}	3^{rd}
S_0	C_1	C_0	C_2
S_1	C_0	C_1	C_2
S_2	C_0	C_1	C_2

$$\left.\begin{array}{l} S_0 \leftrightarrow C_0,\ S_1 \leftrightarrow C_1,\ S_2 \leftrightarrow C_2 \\ S_0 \leftrightarrow C_1,\ S_1 \leftrightarrow C_0,\ S_2 \leftrightarrow C_2 \end{array}\right\} \text{– two stable matchings}$$

Stable roommates

- $2n$ people - each person ranks the remaining $2n - 1$
- Construct a matching with no unstable pairs

Stable roommates

- $2n$ people - each person ranks the remaining $2n - 1$
- Construct a matching with no unstable pairs

	1^{st}	2^{nd}	3^{rd}
S_1	S_2	S_3	S_4
S_2	S_3	S_1	S_4
S_3	S_1	S_2	S_4
S_4	S_1	S_2	S_3

Stable roommates

- 2n people - each person ranks the remaining $2n - 1$
- Construct a matching with no unstable pairs

	1^{st}	2^{nd}	3^{rd}
S_1	S_2	S_3	S_4
S_2	S_3	S_1	S_4
S_3	S_1	S_2	S_4
S_4	S_1	S_2	S_3

$S_1 \leftrightarrow S_2, S_3 \leftrightarrow S_4$:

Stable roommates

- 2n people - each person ranks the remaining $2n - 1$
- Construct a matching with no unstable pairs

	1^{st}	2^{nd}	3^{rd}
S_1	S_2	S_3	S_4
S_2	S_3	S_1	S_4
S_3	S_1	S_2	S_4
S_4	S_1	S_2	S_3

$S_1 \leftrightarrow S_2$, $S_3 \leftrightarrow S_4$: (S_2, S_3) unstable

Stable roommates

- 2n people - each person ranks the remaining $2n - 1$
- Construct a matching with no unstable pairs

	1^{st}	2^{nd}	3^{rd}
S_1	S_2	S_3	S_4
S_2	S_3	S_1	S_4
S_3	S_1	S_2	S_4
S_4	S_1	S_2	S_3

$S_1 \leftrightarrow S_2$, $S_3 \leftrightarrow S_4$: (S_2, S_3) unstable

$S_1 \leftrightarrow S_3$, $S_2 \leftrightarrow S_4$:

Stable roommates

- 2n people - each person ranks the remaining $2n - 1$
- Construct a matching with no unstable pairs

	1^{st}	2^{nd}	3^{rd}
S_1	S_2	S_3	S_4
S_2	S_3	S_1	S_4
S_3	S_1	S_2	S_4
S_4	S_1	S_2	S_3

$S_1 \leftrightarrow S_2$, $S_3 \leftrightarrow S_4$: (S_2, S_3) unstable

$S_1 \leftrightarrow S_3$, $S_2 \leftrightarrow S_4$: (S_1, S_2) unstable

Stable roommates

- $2n$ people - each person ranks the remaining $2n - 1$
- Construct a matching with no unstable pairs

	1^{st}	2^{nd}	3^{rd}
S_1	S_2	S_3	S_4
S_2	S_3	S_1	S_4
S_3	S_1	S_2	S_4
S_4	S_1	S_2	S_3

$S_1 \leftrightarrow S_2, S_3 \leftrightarrow S_4 : (S_2, S_3)$ unstable

$S_1 \leftrightarrow S_3, S_2 \leftrightarrow S_4 : (S_1, S_2)$ unstable

$S_1 \leftrightarrow S_4, S_2 \leftrightarrow S_3$:

Stable roommates

- $2n$ people - each person ranks the remaining $2n - 1$
- Construct a matching with no unstable pairs

	1^{st}	2^{nd}	3^{rd}
S_1	S_2	S_3	S_4
S_2	S_3	S_1	S_4
S_3	S_1	S_2	S_4
S_4	S_1	S_2	S_3

$S_1 \leftrightarrow S_2$, $S_3 \leftrightarrow S_4$: (S_2, S_3) unstable

$S_1 \leftrightarrow S_3$, $S_2 \leftrightarrow S_4$: (S_1, S_2) unstable

$S_1 \leftrightarrow S_4$, $S_2 \leftrightarrow S_3$: (S_1, S_3) unstable

Gale-Shapley algorithm: A pseudocode

Gale-Shapley algorithm: A pseudocode

Set $M \leftarrow \emptyset$

Gale-Shapley algorithm: A pseudocode

Set M $\leftarrow \emptyset$
while $\exists$ unmatched student S and course to which (s)he has not applied **do**

Gale-Shapley algorithm: A pseudocode

Set $M \leftarrow \emptyset$
while $\exists$ unmatched student S and course to which (s)he has not applied **do**
  C $\leftarrow$ first course in list of S to which (s)he has not applied

Gale-Shapley algorithm: A pseudocode

Set $\mathsf{M} \leftarrow \emptyset$

while $\exists$ unmatched student S and course to which (s)he has not applied **do**

  C $\leftarrow$ first course in list of S to which (s)he has not applied

  if C is unmatched **then**

    $\mathsf{M} \leftarrow \mathsf{M} + \{(\mathsf{C}, \mathsf{S})\}$

  else

Gale-Shapley algorithm: A pseudocode

Set $M \leftarrow \emptyset$
while $\exists$ unmatched student S and course to which (s)he has not applied **do**
  $C \leftarrow$ first course in list of S to which (s)he has not applied
  if C is unmatched **then**
    $M \leftarrow M + \{(C, S)\}$
  else
    if C prefers S to its current student S′ **then**
      $M \leftarrow M - \{(C, S')\} + \{(C, S)\}$
    else

Gale-Shapley algorithm: A pseudocode

Set $M \leftarrow \emptyset$
while $\exists$ unmatched student S and course to which (s)he has not applied **do**
    $C \leftarrow$ first course in list of S to which (s)he has not applied
    if C is unmatched **then**
        $M \leftarrow M + \{(C, S)\}$
    else
        if C prefers S to its current student S' **then**
            $M \leftarrow M - \{(C, S')\} + \{(C, S)\}$
        else
            C rejects S

Gale-Shapley algorithm: an example

	1^{st}	2^{nd}	3^{rd}	4^{th}
S_0	C_1	C_0	C_3	C_2
S_1	C_3	C_1	C_0	C_2
S_2	C_1	C_2	C_3	C_0
S_3	C_0	C_3	C_2	C_1

	1^{st}	2^{nd}	3^{rd}	4^{th}
C_0	S_0	S_1	S_3	S_2
C_1	S_2	S_1	S_3	S_0
C_2	S_1	S_2	S_3	S_0
C_3	S_0	S_3	S_2	S_1

Gale-Shapley algorithm: an example

	1^{st}	2^{nd}	3^{rd}	4^{th}
S_0	C_1	C_0	C_3	C_2
S_1	C_3	C_1	C_0	C_2
S_2	C_1	C_2	C_3	C_0
S_3	C_0	C_3	C_2	C_1

	1^{st}	2^{nd}	3^{rd}	4^{th}
C_0	S_0	S_1	S_3	S_2
C_1	S_2	S_1	S_3	S_0
C_2	S_1	S_2	S_3	S_0
C_3	S_0	S_3	S_2	S_1

S_0 applies to C_1:

Gale-Shapley algorithm: an example

	1^{st}	2^{nd}	3^{rd}	4^{th}
S_0	C_1	C_0	C_3	C_2
S_1	C_3	C_1	C_0	C_2
S_2	C_1	C_2	C_3	C_0
S_3	C_0	C_3	C_2	C_1

	1^{st}	2^{nd}	3^{rd}	4^{th}
C_0	S_0	S_1	S_3	S_2
C_1	S_2	S_1	S_3	S_0
C_2	S_1	S_2	S_3	S_0
C_3	S_0	S_3	S_2	S_1

S_0 applies to C_1: C_1 accepts

Gale-Shapley algorithm: an example

	1^{st}	2^{nd}	3^{rd}	4^{th}
S_0	C_1	C_0	C_3	C_2
S_1	C_3	C_1	C_0	C_2
S_2	C_1	C_2	C_3	C_0
S_3	C_0	C_3	C_2	C_1

	1^{st}	2^{nd}	3^{rd}	4^{th}
C_0	S_0	S_1	S_3	S_2
C_1	S_2	S_1	S_3	S_0
C_2	S_1	S_2	S_3	S_0
C_3	S_0	S_3	S_2	S_1

S_1 applies to C_3:

Gale-Shapley algorithm: an example

	1^{st}	2^{nd}	3^{rd}	4^{th}
S_0	C_1	C_0	C_3	C_2
S_1	C_3	C_1	C_0	C_2
S_2	C_1	C_2	C_3	C_0
S_3	C_0	C_3	C_2	C_1

	1^{st}	2^{nd}	3^{rd}	4^{th}
C_0	S_0	S_1	S_3	S_2
C_1	S_2	S_1	S_3	S_0
C_2	S_1	S_2	S_3	S_0
C_3	S_0	S_3	S_2	S_1

S_1 applies to C_3: C_3 accepts

Gale-Shapley algorithm: an example

	1st	2nd	3rd	4th
S_0	C_1	C_0	C_3	C_2
S_1	C_3	C_1	C_0	C_2
S_2	C_1	C_2	C_3	C_0
S_3	C_0	C_3	C_2	C_1

	1st	2nd	3rd	4th
C_0	S_0	S_1	S_3	S_2
C_1	S_2	S_1	S_3	S_0
C_2	S_1	S_2	S_3	S_0
C_3	S_0	S_3	S_2	S_1

S_2 applies to C_1:

Gale-Shapley algorithm: an example

	1st	2nd	3rd	4th
S_0	C_1	C_0	C_3	C_2
S_1	C_3	C_1	C_0	C_2
S_2	C_1	C_2	C_3	C_0
S_3	C_0	C_3	C_2	C_1

	1st	2nd	3rd	4th
C_0	S_0	S_1	S_3	S_2
C_1	S_2	S_1	S_3	S_0
C_2	S_1	S_2	S_3	S_0
C_3	S_0	S_3	S_2	S_1

S_2 applies to C_1: C_1 accepts

Gale-Shapley algorithm: an example

	1st	2nd	3rd	4th
S_0	C_1	C_0	C_3	C_2
S_1	C_3	C_1	C_0	C_2
S_2	C_1	C_2	C_3	C_0
S_3	C_0	C_3	C_2	C_1

	1st	2nd	3rd	4th
C_0	S_0	S_1	S_3	S_2
C_1	S_2	S_1	S_3	S_0
C_2	S_1	S_2	S_3	S_0
C_3	S_0	S_3	S_2	S_1

S_0 applies to C_0:

Gale-Shapley algorithm: an example

	1^{st}	2^{nd}	3^{rd}	4^{th}
S_0	C_1	C_0	C_3	C_2
S_1	C_3	C_1	C_0	C_2
S_2	C_1	C_2	C_3	C_0
S_3	C_0	C_3	C_2	C_1

	1^{st}	2^{nd}	3^{rd}	4^{th}
C_0	S_0	S_1	S_3	S_2
C_1	S_2	S_1	S_3	S_0
C_2	S_1	S_2	S_3	S_0
C_3	S_0	S_3	S_2	S_1

S_0 applies to C_0: C_0 accepts

Gale-Shapley algorithm: an example

	1^{st}	2^{nd}	3^{rd}	4^{th}
S_0	C_1	C_0	C_3	C_2
S_1	C_3	C_1	C_0	C_2
S_2	C_1	C_2	C_3	C_0
S_3	C_0	C_3	C_2	C_1

	1^{st}	2^{nd}	3^{rd}	4^{th}
C_0	S_0	S_1	S_3	S_2
C_1	S_2	S_1	S_3	S_0
C_2	S_1	S_2	S_3	S_0
C_3	S_0	S_3	S_2	S_1

S_3 applies to C_0:

Gale-Shapley algorithm: an example

	1^{st}	2^{nd}	3^{rd}	4^{th}
S_0	C_1	C_0	C_3	C_2
S_1	C_3	C_1	C_0	C_2
S_2	C_1	C_2	C_3	C_0
S_3	C_0	C_3	C_2	C_1

	1^{st}	2^{nd}	3^{rd}	4^{th}
C_0	S_0	S_1	S_3	S_2
C_1	S_2	S_1	S_3	S_0
C_2	S_1	S_2	S_3	S_0
C_3	S_0	S_3	S_2	S_1

S_3 applies to C_0: C_0 rejects

Gale-Shapley algorithm: an example

	1st	2nd	3rd	4th
S_0	C_1	C_0	C_3	C_2
S_1	C_3	C_1	C_0	C_2
S_2	C_1	C_2	C_3	C_0
S_3	C_0	C_3	C_2	C_1

	1st	2nd	3rd	4th
C_0	S_0	S_1	S_3	S_2
C_1	S_2	S_1	S_3	S_0
C_2	S_1	S_2	S_3	S_0
C_3	S_0	S_3	S_2	S_1

S_3 applies to C_3:

Gale-Shapley algorithm: an example

	1st	2nd	3rd	4th
S_0	C_1	C_0	C_3	C_2
S_1	C_3	C_1	C_0	C_2
S_2	C_1	C_2	C_3	C_0
S_3	C_0	C_3	C_2	C_1

	1st	2nd	3rd	4th
C_0	S_0	S_1	S_3	S_2
C_1	S_2	S_1	S_3	S_0
C_2	S_1	S_2	S_3	S_0
C_3	S_0	S_3	S_2	S_1

S_3 applies to C_3: C_3 accepts

Gale-Shapley algorithm: an example

	1^{st}	2^{nd}	3^{rd}	4^{th}
S_0	C_1	C_0	C_3	C_2
S_1	C_3	C_1	C_0	C_2
S_2	C_1	C_2	C_3	C_0
S_3	C_0	C_3	C_2	C_1

	1^{st}	2^{nd}	3^{rd}	4^{th}
C_0	S_0	S_1	S_3	S_2
C_1	S_2	S_1	S_3	S_0
C_2	S_1	S_2	S_3	S_0
C_3	S_0	S_3	S_2	S_1

S_1 applies to C_1:

Gale-Shapley algorithm: an example

	1^{st}	2^{nd}	3^{rd}	4^{th}
S_0	C_1	C_0	C_3	C_2
S_1	C_3	C_1	C_0	C_2
S_2	C_1	C_2	C_3	C_0
S_3	C_0	C_3	C_2	C_1

	1^{st}	2^{nd}	3^{rd}	4^{th}
C_0	S_0	S_1	S_3	S_2
C_1	S_2	S_1	S_3	S_0
C_2	S_1	S_2	S_3	S_0
C_3	S_0	S_3	S_2	S_1

S_1 applies to C_1: C_1 rejects

Gale-Shapley algorithm: an example

	1^{st}	2^{nd}	3^{rd}	4^{th}
S_0	C_1	C_0	C_3	C_2
S_1	C_3	C_1	C_0	C_2
S_2	C_1	C_2	C_3	C_0
S_3	C_0	C_3	C_2	C_1

	1^{st}	2^{nd}	3^{rd}	4^{th}
C_0	S_0	S_1	S_3	S_2
C_1	S_2	S_1	S_3	S_0
C_2	S_1	S_2	S_3	S_0
C_3	S_0	S_3	S_2	S_1

S_1 applies to C_0:

Gale-Shapley algorithm: an example

	1^{st}	2^{nd}	3^{rd}	4^{th}
S_0	C_1	C_0	C_3	C_2
S_1	C_3	C_1	C_0	C_2
S_2	C_1	C_2	C_3	C_0
S_3	C_0	C_3	C_2	C_1

	1^{st}	2^{nd}	3^{rd}	4^{th}
C_0	S_0	S_1	S_3	S_2
C_1	S_2	S_1	S_3	S_0
C_2	S_1	S_2	S_3	S_0
C_3	S_0	S_3	S_2	S_1

S_1 applies to C_0: C_0 rejects

Gale-Shapley algorithm: an example

	1^{st}	2^{nd}	3^{rd}	4^{th}
S_0	C_1	C_0	C_3	C_2
S_1	C_3	C_1	C_0	C_2
S_2	C_1	C_2	C_3	C_0
S_3	C_0	C_3	C_2	C_1

	1^{st}	2^{nd}	3^{rd}	4^{th}
C_0	S_0	S_1	S_3	S_2
C_1	S_2	S_1	S_3	S_0
C_2	S_1	S_2	S_3	S_0
C_3	S_0	S_3	S_2	S_1

S_1 applies to C_2:

Gale-Shapley algorithm: an example

	1^{st}	2^{nd}	3^{rd}	4^{th}
S_0	C_1	C_0	C_3	C_2
S_1	C_3	C_1	C_0	C_2
S_2	C_1	C_2	C_3	C_0
S_3	C_0	C_3	C_2	C_1

	1^{st}	2^{nd}	3^{rd}	4^{th}
C_0	S_0	S_1	S_3	S_2
C_1	S_2	S_1	S_3	S_0
C_2	S_1	S_2	S_3	S_0
C_3	S_0	S_3	S_2	S_1

S_1 applies to C_2: C_2 accepts

Gale-Shapley algorithm: pseudocode

```
Set M ← ∅
while ∃ unmatched student S and course to which (s)he
  has not applied do
    C ← first course in list of S to which (s)he has not
      applied
    if C is unmatched then
        M ← M + {(C, S)}
    else
        if C prefers S to its current student S′ then
            M ← M − {(C, S′)} + {(C, S)}
        else
            C rejects S
```

Gale-Shapley algorithm: termination

Gale-Shapley algorithm: termination

Lemma: Once a course is matched to a student, the following are true

- The course is never unmatched
- If the course is assigned a new student, (s)he will be higher in preference order for the course

Gale-Shapley algorithm: termination

Lemma: Once a course is matched to a student, the following are true

- The course is never unmatched
- If the course is assigned a new student, (s)he will be higher in preference order for the course

- Only reason for a course to change the assigned student is if a student who is higher in its preference order applies

Gale-Shapley algorithm: termination

Lemma: The Gale-Shapley algorithm terminates after n^2 iterations of the while-loop with a matching

Gale-Shapley algorithm: termination

Lemma: The Gale-Shapley algorithm terminates after n^2 iterations of the while-loop with a matching

- Each student applies to a course at most once

Gale-Shapley algorithm: termination

Lemma: The Gale-Shapley algorithm terminates after n^2 iterations of the while-loop with a matching

- Each student applies to a course at most once
- After n^2 steps, every course has been applied to at least once

Gale-Shapley algorithm: termination

Lemma: The Gale-Shapley algorithm terminates after n^2 iterations of the while-loop with a matching

- Each student applies to a course at most once
- After n^2 steps, every course has been applied to at least once
- Once a course is applied to, it remains matched

Gale-Shapley algorithm: termination

Lemma: The Gale-Shapley algorithm terminates after n^2 iterations of the while-loop with a matching

- Each student applies to a course at most once
- After n^2 steps, every course has been applied to at least once
- Once a course is applied to, it remains matched
- If all courses are matched, then all students are also matched

Gale-Shapley algorithm: stability

Lemma: If M is the matching returned by the Gale-Shapley algorithm, then M has no unstable pairs

Gale-Shapley algorithm: stability

Lemma: If $\mathtt{M}$ is the matching returned by the Gale-Shapley algorithm, then $\mathtt{M}$ has no unstable pairs

Consider a pair $(\mathtt{C}, \mathtt{S}) \notin \mathtt{M}$ - Can $(\mathtt{C}, \mathtt{S})$ form an unstable pair?

Lemma: If M is the matching returned by the Gale-Shapley algorithm, then M has no unstable pairs

Consider a pair $(C, S) \notin M$ - Can (C, S) form an unstable pair?

- S never applied to C

Gale-Shapley algorithm: stability

Lemma: If M is the matching returned by the Gale-Shapley algorithm, then M has no unstable pairs

Consider a pair $(C, S) \notin M$ - Can (C, S) form an unstable pair?

- S never applied to C
  - S applies to courses in the decreasing order of preference

Gale-Shapley algorithm: stability

Lemma: If M is the matching returned by the Gale-Shapley algorithm, then M has no unstable pairs

Consider a pair $(C, S) \notin M$ - Can (C, S) form an unstable pair?

- S never applied to C
  - S applies to courses in the decreasing order of preference
  - S is matched to a course C' higher in its preference order

Gale-Shapley algorithm: stability

Lemma: If M is the matching returned by the Gale-Shapley algorithm, then M has no unstable pairs

Consider a pair $(C, S) \notin M$ - Can (C, S) form an unstable pair?

- S never applied to C
    - S applies to courses in the decreasing order of preference
    - S is matched to a course C' higher in its preference order
- S applied to C at some iteration

Gale-Shapley algorithm: stability

Lemma: If M is the matching returned by the Gale-Shapley algorithm, then M has no unstable pairs

Consider a pair $(C, S) \notin M$ - Can (C, S) form an unstable pair?

- S never applied to C
  - S applies to courses in the decreasing order of preference
  - S is matched to a course C' higher in its preference order
- S applied to C at some iteration
  - S was rejected by C because C was assigned a more preferred student

Gale-Shapley algorithm: stability

Lemma: If M is the matching returned by the Gale-Shapley algorithm, then M has no unstable pairs

Consider a pair $(C, S) \notin M$ - Can (C, S) form an unstable pair?

- S never applied to C
  - S applies to courses in the decreasing order of preference
  - S is matched to a course C' higher in its preference order
- S applied to C at some iteration
  - S was rejected by C because C was assigned a more preferred student
  - S was ditched by C when it received request from a student higher in the preference order

Gale-Shapley algorithm: stability

Lemma: If M is the matching returned by the Gale-Shapley algorithm, then M has no unstable pairs

Consider a pair $(C, S) \notin M$ - Can (C, S) form an unstable pair?

- S never applied to C
  - S applies to courses in the decreasing order of preference
  - S is matched to a course C' higher in its preference order
- S applied to C at some iteration
  - S was rejected by C because C was assigned a more preferred student
  - S was ditched by C when it received request from a student higher in the preference order

Question Do all executions of the Gale-Shapley algorithm lead to the same stable matching?

Gale-Shapley algorithm: running time

Gale-Shapley algorithm: running time

The while-loop executes at most n^2 times

- How long does it take to find an unmatched student and a course that (s)he has not applied to?

Gale-Shapley algorithm: running time

The while-loop executes at most n^2 times

- How long does it take to find an unmatched student and a course that (s)he has not applied to?
    - Queue of students who are not matched
    - Queue of courses not applied to (for every student)

Gale-Shapley algorithm: running time

The while-loop executes at most n^2 times

- How long does it take to find an unmatched student and a course that (s)he has not applied to?
  - Queue of students who are not matched
  - Queue of courses not applied to (for every student)
- How should the matching be stored so that you can check if C is matched in O(1) time?

Gale-Shapley algorithm: running time

The while-loop executes at most n^2 times

- How long does it take to find an unmatched student and a course that (s)he has not applied to?
  - Queue of students who are not matched
  - Queue of courses not applied to (for every student)
- How should the matching be stored so that you can check if C is matched in O(1) time?

  One bit per course/student to indicate whether the course is matched, and id of the matched course/student

Gale-Shapley algorithm: running time

The while-loop executes at most n^2 times

- How long does it take to find an unmatched student and a course that (s)he has not applied to?
  - Queue of students who are not matched
  - Queue of courses not applied to (for every student)
- How should the matching be stored so that you can check if C is matched in O(1) time?

  One bit per course/student to indicate whether the course is matched, and id of the matched course/student
- The data structure storing the matching should allow an update of the matching in O(1) time

Gale-Shapley algorithm: running time

The while-loop executes at most n^2 times

- How long does it take to find an unmatched student and a course that (s)he has not applied to?
  - Queue of students who are not matched
  - Queue of courses not applied to (for every student)
- How should the matching be stored so that you can check if C is matched in O(1) time?

  One bit per course/student to indicate whether the course is matched, and id of the matched course/student
- The data structure storing the matching should allow an update of the matching in O(1) time

  Update the corresponding bits, and change the course/student information

Gale-Shapley algorithm: running time

The while-loop executes at most n^2 times

- How long does it take to find an unmatched student and a course that (s)he has not applied to? O(1)-time
- Checking if C is a matched course O(1)-time
- Update the matching O(1)-time

Gale-Shapley algorithm: running time

The while-loop executes at most n^2 times

- How long does it take to find an unmatched student and a course that (s)he has not applied to? $O(1)$-time
- Checking if C is a matched course $O(1)$-time
- Update the matching $O(1)$-time

Gale-Shapley algorithm runs in time $O(n^2)$

Gale-Shapley algorithm: properties of the solution

	1st	2nd	3rd
C_0	S_0	S_1	S_2
C_1	S_1	S_0	S_2
C_2	S_0	S_1	S_2

	1st	2nd	3rd
S_0	C_1	C_0	C_2
S_1	C_0	C_1	C_2
S_2	C_0	C_1	C_2

$$\left.\begin{array}{l} S_0 \leftrightarrow C_0,\ S_1 \leftrightarrow C_1,\ S_2 \leftrightarrow C_2 \\ S_0 \leftrightarrow C_1,\ S_1 \leftrightarrow C_0,\ S_2 \leftrightarrow C_2 \end{array}\right\} \text{– two stable matchings}$$

Gale-Shapley algorithm: properties of the solution

	1st	2nd	3rd
C_0	S_0	S_1	S_2
C_1	S_1	S_0	S_2
C_2	S_0	S_1	S_2

	1st	2nd	3rd
S_0	C_1	C_0	C_2
S_1	C_0	C_1	C_2
S_2	C_0	C_1	C_2

$$\left.\begin{array}{l} S_0 \leftrightarrow C_0,\ S_1 \leftrightarrow C_1,\ S_2 \leftrightarrow C_2 \\ S_0 \leftrightarrow C_1,\ S_1 \leftrightarrow C_0,\ S_2 \leftrightarrow C_2 \end{array}\right\} \text{– two stable matchings}$$

- There is no stable matching that matches S_2 to C_0 or C_1

Gale-Shapley algorithm: properties of the solution

	1st	2nd	3rd
C_0	S_0	S_1	S_2
C_1	S_1	S_0	S_2
C_2	S_0	S_1	S_2

	1st	2nd	3rd
S_0	C_1	C_0	C_2
S_1	C_0	C_1	C_2
S_2	C_0	C_1	C_2

$$\left.\begin{array}{l} S_0 \leftrightarrow C_0,\ S_1 \leftrightarrow C_1,\ S_2 \leftrightarrow C_2 \\ S_0 \leftrightarrow C_1,\ S_1 \leftrightarrow C_0,\ S_2 \leftrightarrow C_2 \end{array}\right\} \text{ – two stable matchings}$$

- There is no stable matching that matches S_2 to C_0 or C_1
- There is no stable matching that matches C_2 to S_0 or S_1

Gale-Shapley algorithm: properties of the solution

	1^{st}	2^{nd}	3^{rd}
C_0	S_0	S_1	S_2
C_1	S_1	S_0	S_2
C_2	S_0	S_1	S_2

	1^{st}	2^{nd}	3^{rd}
S_0	C_1	C_0	C_2
S_1	C_0	C_1	C_2
S_2	C_0	C_1	C_2

$$\left.\begin{array}{l} S_0 \leftrightarrow C_0,\ S_1 \leftrightarrow C_1,\ S_2 \leftrightarrow C_2 \\ S_0 \leftrightarrow C_1,\ S_1 \leftrightarrow C_0,\ S_2 \leftrightarrow C_2 \end{array}\right\} \text{– two stable matchings}$$

- There is no stable matching that matches S_2 to C_0 or C_1
- There is no stable matching that matches C_2 to S_0 or S_1
- M is the best matching for the students, and the worst for the courses

Gale-Shapley algorithm: properties of the solution

	1^{st}	2^{nd}	3^{rd}
C_0	S_0	S_1	S_2
C_1	S_1	S_0	S_2
C_2	S_0	S_1	S_2

	1^{st}	2^{nd}	3^{rd}
S_0	C_1	C_0	C_2
S_1	C_0	C_1	C_2
S_2	C_0	C_1	C_2

$$\left.\begin{array}{l} S_0 \leftrightarrow C_0,\ S_1 \leftrightarrow C_1,\ S_2 \leftrightarrow C_2 \\ S_0 \leftrightarrow C_1,\ S_1 \leftrightarrow C_0,\ S_2 \leftrightarrow C_2 \end{array}\right\} \text{– two stable matchings}$$

- There is no stable matching that matches S_2 to C_0 or C_1
- There is no stable matching that matches C_2 to S_0 or S_1
- M is the best matching for the students, and the worst for the courses
- M' is the best matching for the courses, and the worst for the students

Gale-Shapley algorithm: properties of the solution

	1^{st}	2^{nd}	3^{rd}
C_0	S_0	S_1	S_2
C_1	S_1	S_0	S_2
C_2	S_0	S_1	S_2

	1^{st}	2^{nd}	3^{rd}
S_0	C_1	C_0	C_2
S_1	C_0	C_1	C_2
S_2	C_0	C_1	C_2

$$\left.\begin{array}{l} S_0 \leftrightarrow C_0,\ S_1 \leftrightarrow C_1,\ S_2 \leftrightarrow C_2 \\ S_0 \leftrightarrow C_1,\ S_1 \leftrightarrow C_0,\ S_2 \leftrightarrow C_2 \end{array}\right\} \text{– two stable matchings}$$

Theorem:

- If the students apply to the courses, then the output obtained will be the best matching for the students, and the worst for the courses.
- If the courses propose to the students, then the output obtained will be the best matching for the courses, and the worst for the students.

Gale-Shapley algorithm: Optimality

	1st	2nd	3rd
C_0	S_0	S_1	S_2
C_1	S_1	S_0	S_2
C_2	S_0	S_1	S_2

	1st	2nd	3rd
S_0	C_1	C_0	C_2
S_1	C_0	C_1	C_2
S_2	C_0	C_1	C_2

- A course C_j is a valid match for S_i if there exists a stable matching that matches C_j to S_i

Gale-Shapley algorithm: Optimality

	1st	2nd	3rd
C_0	S_0	S_1	S_2
C_1	S_1	S_0	S_2
C_2	S_0	S_1	S_2

	1st	2nd	3rd
S_0	C_1	C_0	C_2
S_1	C_0	C_1	C_2
S_2	C_0	C_1	C_2

- A course C_j is a valid match for S_i if there exists a stable matching that matches C_j to S_i

  C_0 is a valid match for S_0 and S_1, but not for S_2

Gale-Shapley algorithm: Optimality

	1st	2nd	3rd
C_0	S_0	S_1	S_2
C_1	S_1	S_0	S_2
C_2	S_0	S_1	S_2

	1st	2nd	3rd
S_0	C_1	C_0	C_2
S_1	C_0	C_1	C_2
S_2	C_0	C_1	C_2

- A course C_j is a valid match for S_i if there exists a stable matching that matches C_j to S_i

  C_0 is a valid match for S_0 and S_1, but not for S_2
- C_j is the best valid match for S_i if in every stable matching M', either C_j is matched to S_i, or if C' is matched to S_i, then C' is lower in the preference order than C_j

Gale-Shapley algorithm: Optimality

	1^{st}	2^{nd}	3^{rd}
C_0	S_0	S_1	S_2
C_1	S_1	S_0	S_2
C_2	S_0	S_1	S_2

	1^{st}	2^{nd}	3^{rd}
S_0	C_1	C_0	C_2
S_1	C_0	C_1	C_2
S_2	C_0	C_1	C_2

- A course C_j is a valid match for S_i if there exists a stable matching that matches C_j to S_i

  C_0 is a valid match for S_0 and S_1, but not for S_2
- C_j is the best valid match for S_i if in every stable matching M', either C_j is matched to S_i, or if C' is matched to S_i, then C' is lower in the preference order than C_j

Theorem: Let $M^* = \{(S_i, \text{best}(S_i)) | 1 \leq i \leq n\}$. If Gale-Shapley algorithm is run with students making requests, then it always returns M^* irrespective of the order in which the students make the requests.

Gale-Shapley algorithm: Optimality

Theorem: Let $M^* = \{(S_i, \text{best}(S_i)) | 1 \leq i \leq n\}$. If Gale-Shapley algorithm is run with students making requests, then it always returns M^* irrespective of the order in which the students make the requests.

Proof: Suppose the GS algorithm returns matching $M \neq M^*$

Gale-Shapley algorithm: Optimality

Theorem: Let $M^* = \{(S_i, best(S_i)) | 1 \leq i \leq n\}$. If Gale-Shapley algorithm is run with students making requests, then it always returns M^* irrespective of the order in which the students make the requests.

Proof: Suppose the GS algorithm returns matching $M \neq M^*$

Consider first instance where a student is rejected by a course that is a valid match

- S_i is rejected by a valid match C_j

Gale-Shapley algorithm: Optimality

Theorem: Let $M^* = \{(S_i, \text{best}(S_i)) | 1 \leq i \leq n\}$. If Gale-Shapley algorithm is run with students making requests, then it always returns M^* irrespective of the order in which the students make the requests.

Proof: Suppose the GS algorithm returns matching $M \neq M^*$

Consider first instance where a student is rejected by a course that is a valid match

- S_i is rejected by a valid match C_j
- C_j matched to S', higher in preference order

Gale-Shapley algorithm: Optimality

Theorem: Let $M^* = \{(S_i, \text{best}(S_i)) | 1 \leq i \leq n\}$. If Gale-Shapley algorithm is run with students making requests, then it always returns M^* irrespective of the order in which the students make the requests.

Proof: Suppose the GS algorithm returns matching $M \neq M^*$

Consider first instance where a student is rejected by a course that is a valid match

- S_i is rejected by a valid match C_j
- C_j matched to S', higher in preference order

Consider a stable matching M' where $(S_i, C_j) \in M'$, and $(S', C') \in M'$

Gale-Shapley algorithm: Optimality

Theorem: Let $M^* = \{(S_i, \text{best}(S_i)) | 1 \leq i \leq n\}$. If Gale-Shapley algorithm is run with students making requests, then it always returns M^* irrespective of the order in which the students make the requests.

Proof: Suppose the GS algorithm returns matching $M \neq M^*$

Consider first instance where a student is rejected by a course that is a valid match

- S_i is rejected by a valid match C_j
- C_j matched to S', higher in preference order

Consider a stable matching M' where $(S_i, C_j) \in M'$, and $(S', C') \in M'$

Question Which course does S' prefer more: C_j or C'?

Gale-Shapley algorithm: Optimality

Theorem: Let $M^* = \{(S_i, \text{best}(S_i)) | 1 \leq i \leq n\}$. If Gale-Shapley algorithm is run with students making requests, then it always returns M^* irrespective of the order in which the students make the requests.

Proof: Suppose the GS algorithm returns matching $M \neq M^*$

Consider first instance where a student is rejected by a course that is a valid match

- S_i is rejected by a valid match C_j
- C_j matched to S', higher in preference order

Consider a stable matching M' where $(S_i, C_j) \in M'$, and $(S', C') \in M'$

Question Which course does S' prefer more: C_j or C'?

- S' was not rejected by a valid match when it is matched to C_j

Gale-Shapley algorithm: Optimality

Theorem: Let $M^* = \{(S_i, \text{best}(S_i)) | 1 \leq i \leq n\}$. If Gale-Shapley algorithm is run with students making requests, then it always returns M^* irrespective of the order in which the students make the requests.

Proof: Suppose the GS algorithm returns matching $M \neq M^*$

Consider first instance where a student is rejected by a course that is a valid match

- S_i is rejected by a valid match C_j
- C_j matched to S', higher in preference order

Consider a stable matching M' where $(S_i, C_j) \in M'$, and $(S', C') \in M'$

Question Which course does S' prefer more: C_j or C'?

- S' was not rejected by a valid match when it is matched to C_j
- S' prefers C_j to $C' \Rightarrow (S', C_j)$ is unstable in M'

Gale-Shapley algorithm: Optimality

Theorem: Let $M^* = \{(S_i, \text{best}(S_i)) | 1 \leq i \leq n\}$. If Gale-Shapley algorithm is run with students making requests, then it always returns M^* irrespective of the order in which the students make the requests.

Proof: Suppose the GS algorithm returns matching $M \neq M^*$

Consider first instance where a student is rejected by a course that is a valid match

- S_i is rejected by a valid match C_j
- C_j matched to S', higher in preference order

Consider a stable matching M' where $(S_i, C_j) \in M'$, and $(S', C') \in M'$

Question Which course does S' prefer more: C_j or C'?

- S' was not rejected by a valid match when it is matched to C'
- S' prefers C_j to $C' \Rightarrow (S', C_j)$ is unstable in M' → contradicts the fact that M' is a stable matching

Gale-Shapley algorithm: Optimality

Theorem: Let $\widehat{M} = \{(\mathrm{worst}(C_j), C_j) | 1 \leq j \leq n\}$. Then $\widehat{M} = M^*$

Gale-Shapley algorithm: Optimality

Theorem: Let $\widehat{M} = \{(\text{worst}(C_j), C_j) | 1 \leq j \leq n\}$. Then $\widehat{M} = M^*$

Proof: Consider $(S_i, C_j) \in M^*$ such that $S_i \neq S = \text{worst}(C_j)$

Gale-Shapley algorithm: Optimality

Theorem: Let $\widehat{M} = \{(\text{worst}(C_j), C_j) | 1 \leq j \leq n\}$. Then $\widehat{M} = M^*$

Proof: Consider $(S_i, C_j) \in M^*$ such that $S_i \neq S = \text{worst}(C_j)$

- $\exists$ a stable matching M' such that $(S, C_j) \in M'$, and
- C_j prefers S_i to S

Gale-Shapley algorithm: Optimality

Theorem: Let $\widehat{M} = \{(\text{worst}(C_j), C_j) | 1 \leq j \leq n\}$. Then $\widehat{M} = M^*$

Proof: Consider $(S_i, C_j) \in M^*$ such that $S_i \neq S = \text{worst}(C_j)$

- $\exists$ a stable matching M' such that $(S, C_j) \in M'$, and
- C_j prefers S_i to S

Suppose $(S_i, C') \in M'$

Gale-Shapley algorithm: Optimality

Theorem: Let $\widehat{M} = \{(\text{worst}(C_j), C_j) | 1 \leq j \leq n\}$. Then $\widehat{M} = M^*$

Proof: Consider $(S_i, C_j) \in M^*$ such that $S_i \neq S = \text{worst}(C_j)$

- $\exists$ a stable matching M' such that $(S, C_j) \in M'$, and
- C_j prefers S_i to S

Suppose $(S_i, C') \in M'$

- S_i prefers C_j to C'

Gale-Shapley algorithm: Optimality

Theorem: Let $\widehat{M} = \{(\text{worst}(C_j), C_j) | 1 \le j \le n\}$. Then $\widehat{M} = M^*$

Proof: Consider $(S_i, C_j) \in M^*$ such that $S_i \neq S = \text{worst}(C_j)$

- $\exists$ a stable matching M' such that $(S, C_j) \in M'$, and
- C_j prefers S_i to S

Suppose $(S_i, C') \in M'$

$C_j = \text{best}(S_i)$

- S_i prefers C_j to C'

Gale-Shapley algorithm: Optimality

Theorem: Let $\widehat{M} = \{(\text{worst}(C_j), C_j) | 1 \leq j \leq n\}$. Then $\widehat{M} = M^*$

Proof: Consider $(S_i, C_j) \in M^*$ such that $S_i \neq S = \text{worst}(C_j)$

- $\exists$ a stable matching M' such that $(S, C_j) \in M'$, and
- C_j prefers S_i to S

Suppose $(S_i, C') \in M'$

$C_j = \text{best}(S_i)$

- S_i prefers C_j to C'
- (S_i, C_j) is an unstable pair in M'

What constitutes a good algorithm?

- Proposed procedure in unambiguous terms.
- Proof of termination.
- Proof correctness.
- Proof of running time or space used by the algorithm.

What constitutes a good algorithm?

- Proposed procedure in unambiguous terms.
- Proof of termination.
- Proof correctness.
- Proof of running time or space used by the algorithm.
  this may involve using the right data structures.
- Proof of tightness, if possible.

What do we intend to do?

- Incremental and Decremental Design
- Recursion & Divide-and-conquer Strategy
- Greedy Algorithm Strategy
- Dynamic Programming Strategy
- Incremental Strategy
- Limitations and Intractability

As a part of the analysis techniques, we will see data structures, amortization.

Acknolwedgements

- The material is based on book on Algorithms by Jeff Erickson. Please see the course homepage for details.
- Slides for the first lecture is derived from Yadu Vasudev's slides from 2023 edition of the course.