

**CS 361**  
**Data Structures & Algs**  
**Lecture 2**

**Prof. Tom Hayes**  
**University of New Mexico**  
**08-23-2012**

# Did you remember?

- All students: email me, hayes@unm.edu  
Tell me: your name, email (that you will check), recent CS and math courses, registered or not?, major/department, year/grad/undergrad
- Graduating seniors: please notify me

# First Assignments

Admin: Email me -- due yesterday.

Reading: Chapter 1 this weekend

Secs 2.1-2.4 by next weekend

Written Assignment (from K&T):

Chapter 1, problems 1,2,3,4,6

Programming Assignment: coming soon!

# New topic: Stable Matching

eHarmony town:

$n$  men,  $n$  women. Each provides a list giving the order that they rank the  $n$  members of the opposite sex.

Goal: pair them into  $n$  married couples.

Danger: don't want anyone to run off with someone else's spouse!

# Def: “Perfect” matchings

A **matching** is a pairing of men and women which avoids polygamy/polyandry. That is, it is at most 1-to-1.

A **perfect matching** is exactly 1-to-1. Nobody is left unmatched.

Perfect matchings can be very imperfect!

Why? Has nothing to do with people's preferences.

# Preference Lists

Input: Each man ranks every woman, from his favorite, to his least favorite.

Women do the same for men.

Ex: Men = {1, 2, 3}. Women = {A, B, C}

1 ranks: A, C, B      A ranks: 3, 2, 1

2 ranks: A, B, C      B ranks: 1, 3, 2

3 ranks: C, B, A      C ranks: 2, 3, 1

## Preference Lists example:

Ex: Men = {1, 2, 3}. Women = {A, B, C}

1 ranks: A, C, B      A ranks: 3, 2, 1

2 ranks: A, B, C      B ranks: 1, 3, 2

3 ranks: C, B, A      C ranks: 2, 3, 1

We are matchmakers. Suppose we assign: (1, A), (2, C), (3, B).

# Preference Lists example:

Ex: Men = {1, 2, 3}. Women = {A, B, C}

1 ~~rank: A, C, B~~ A ranks: 3, 2, 1

2 ~~rank: A, B, C~~ B ranks: 1, 3, 2

3 ~~rank: C, B, A~~ C ranks: 2, 3, 1

We are matchmakers. Suppose we assign: (1, A), (2, C), (3, B).



# Preference Lists example:

Ex: Men = {1, 2, 3}. Women = {A, B, C}

1 ranks: **A**, C, B      A ranks: 3, 2, **1**

2 ranks: A, B, **C**      B ranks: 1, **3**, 2

3 ranks: C, **B**, A      C ranks: **2**, 3, 1

We are matchmakers. Suppose we assign: (1, A), (2, C), (3, B).

# Preference Lists example:

Ex: Men = {1, 2, 3}. Women = {A, B, C}

1 ranks: **A**, C, B      A ranks: 3, **2**, **1**  
2 ranks: **A**, B, **C**      B ranks: 1, **3**, 2  
3 ranks: C, **B**, A      C ranks: **2**, 3, 1

We are matchmakers. Suppose we assign: (1, A), (2, C), (3, B).

Then 2 and A might run off together!  
(Each prefers the other over his spouse.)

# Instability

Given preference lists, and a matching, we say that  $(m,w)$  are an **instability** if

(1)  $m, w$  are not married to each other.

(2)  $m$  prefers  $w$  to his spouse and

(3)  $w$  prefers  $m$  to her spouse.

A matching is **stable** if it has no instabilities  $(m,w)$ .

# Stable Matching Problem

INPUT: preference lists for  $n$  men and  $n$  women.

OUTPUT: a perfect matching

Moreover, output is correct only if it is a stable matching.

Ex: Men = {1, 2, 3}. Women = {A, B, C}

1 ranks: A, C, B      A ranks: 3, 2, 1

2 ranks: A, B, C      B ranks: 1, 3, 2

3 ranks: C, B, A      C ranks: 2, 3, 1

6 possible matchings:

(1A) (2B) (3C)

or (1A) (2C) (3B)

or (1B) (2A) (3C)

or (1B) (2C) (3A)

or (1C) (2A) (3B)

or (1C) (2B) (3A)

Ex: Men = {1, 2, 3}. Women = {A, B, C}

1 ranks: A, C, B      A ranks: 3, 2, 1

2 ranks: A, B, C      B ranks: 1, 3, 2

3 ranks: C, B, A      C ranks: 2, 3, 1

6 possible matchings:      **Stable?**

(1A) (2B) (3C)

or (1A) (2C) (3B)

or (1B) (2A) (3C)

or (1B) (2C) (3A)

or (1C) (2A) (3B)

or (1C) (2B) (3A)

Ex: Men = {1, 2, 3}. Women = {A, B, C}

1 ranks: A, C, B

A ranks: 3, 2, 1

2 ranks: A, B, C

B ranks: 1, 3, 2

3 ranks: C, B, A

C ranks: 2, 3, 1

6 possible matchings: **Stable?**

(1A) (2B) (3C)

**No; (2,A)**

or (1A) (2C) (3B)

or (1B) (2A) (3C)

or (1B) (2C) (3A)

or (1C) (2A) (3B)

or (1C) (2B) (3A)

Ex: Men = {1, 2, 3}. Women = {A, B, C}

1 ranks: A, C, B

A ranks: 3, 2, 1

2 ranks: A, B, C

B ranks: 1, 3, 2

3 ranks: C, B, A

C ranks: 2, 3, 1

6 possible matchings:

Stable?

(1A) (2B) (3C)

No; (2,A)

or (1A) (2C) (3B)

No; (2,A)

or (1B) (2A) (3C)

or (1B) (2C) (3A)

or (1C) (2A) (3B)

or (1C) (2B) (3A)



Ex: Men = {1, 2, 3}. Women = {A, B, C}

1 ranks: A, C, B

A ranks: 3, 2, 1

2 ranks: A, B, C

B ranks: 1, 3, 2

3 ranks: C, B, A

C ranks: 2, 3, 1

6 possible matchings:

Stable?

(1A) (2B) (3C)

No; (2,A)

or (1A) (2C) (3B)

No; (2,A)

or (1B) (2A) (3C)

Yes!

or (1B) (2C) (3A)

or (1C) (2A) (3B)

or (1C) (2B) (3A)

Ex: Men = {1, 2, 3}. Women = {A, B, C}

1 ranks: A, C, B

A ranks: 3, 2, 1

2 ranks: A, B, C

B ranks: 1, 3, 2

3 ranks: C, B, A

C ranks: 2, 3, 1

6 possible matchings:

Stable?

(1A) (2B) (3C)

No; (2,A)

or (1A) (2C) (3B)

No; (2,A)

or (1B) (2A) (3C)

Yes!

or (1B) (2C) (3A)

Yes!

or (1C) (2A) (3B)

or (1C) (2B) (3A)

Ex: Men = {1, 2, 3}. Women = {A, B, C}

1 ranks: A, C, B

A ranks: 3, 2, 1

2 ranks: A, B, C

B ranks: 1, 3, 2

3 ranks: C, B, A

C ranks: 2, 3, 1

6 possible matchings:

Stable?

(1A) (2B) (3C)

No; (2,A)

or (1A) (2C) (3B)

No; (2,A)

or (1B) (2A) (3C)

Yes!

or (1B) (2C) (3A)

Yes!

or (1C) (2A) (3B)

No; (3,C)

or (1C) (2B) (3A)

Ex: Men = {1, 2, 3}. Women = {A, B, C}

1 ranks: A, C, B

A ranks: 3, 2, 1

2 ranks: A, B, C

B ranks: 1, 3, 2

3 ranks: C, B, A

C ranks: 2, 3, 1

6 possible matchings:

Stable?

(1A) (2B) (3C)

No; (2,A)

or (1A) (2C) (3B)

No; (2,A)

or (1B) (2A) (3C)

Yes!

or (1B) (2C) (3A)

Yes!

or (1C) (2A) (3B)

No; (3,C)

or (1C) (2B) (3A)

No; (3,C) or (3,B)

# Stable Matching Problem

INPUT: preference lists for  $n$  men and  $n$  women.

OUTPUT: a perfect matching

Moreover, output is correct only if it is a stable matching.

**Observation:** there may be more than one correct output possible!

Principle: *Fully understand the problem statement, before trying to solve it.*

1. Can you tell whether a proposed answer is correct or wrong?
2. Can you write code to do this?
3. What are the boundary cases/base cases?
4. Prepare some “test cases”, and use them to test out your later ideas.

# Stable Matching Problem

Naive Approach: Guess & Check

for (M in {all matchings})

    if (M is stable)

        return M

    // else continue to next M

# Stable Matching Problem

Naive Approach: Guess & Check

for (M in {all matchings})

    if (M is stable)

        return M

    // else continue to next M

**How long will this take?**



# Stable Matching Problem

Naive Approach: Guess & Check

for (M in {all matchings})

    if (M is stable)

        return M

    // else continue to next M

**How long will this take?**

$\#\{\text{all matchings}\} * \text{time}(\text{"is M stable"})$

# Stable Matching Problem

Naive Approach: Guess & Check

for (M in {all matchings})

    if (M is stable)

        return M

    // else continue to next M

**How long will this take?**

$\#\{\text{all matchings}\} * \text{time}(\text{"is M stable"})$

# Checking stability

Given a matching, how can we check whether it is stable?

# Checking stability

Given a matching, how can we check whether it is stable?

```
for (i = 1 to n)
```

```
    for (j = 1 to n)
```

```
        if ( (i,j) is an instability )
```

```
            return “unstable”
```

```
return “stable”
```

# Checking stability

Given a matching, how can we check whether it is stable?

for (i = 1 to n)

    for (j = 1 to n)

        if ( (i,j) is an instability )

            return “unstable”

return “stable”

# Checking stability

Given a matching, how can we check whether it is stable?

```
for (i = 1 to n)
  for (j = 1 to n)
    if ( (i,j) is an instability )
      // does i prefer j to wife(i)?
      // does j prefer i to husband(j)
      return "unstable"
return "stable"
```

# Stable Matching Problem

Naive Approach: Guess & Check

for (M in {all matchings})

    if (M is stable)

        return M

    // else continue to next M

**How long will this take?**

$\#\{\text{all matchings}\} * \text{time}(\text{"is M stable"})$   
**poly(n)**

# Stable Matching Problem

Naive Approach: Guess & Check

for (M in {all matchings})

if (M is stable)

return M

// else continue to next M

How long will this take?

{all matchings} \* time("is M stable")

$n! = n * (n-1) * \dots * 2 * 1$       poly(n)



# Stable Matching Problem

Naive Approach: Guess & Check

for (M in {all matchings})

if (M is stable)

return M

// else continue to next M

**How long? super-exponential!**

$\#\{\text{all matchings}\} * \text{time}(\text{"is M stable"})$

$n! = n * (n-1) * \dots * 2 * 1$       **poly(n)**

# Stable Matching Problem

Better approach (but very vague):

Start pairing couples up.

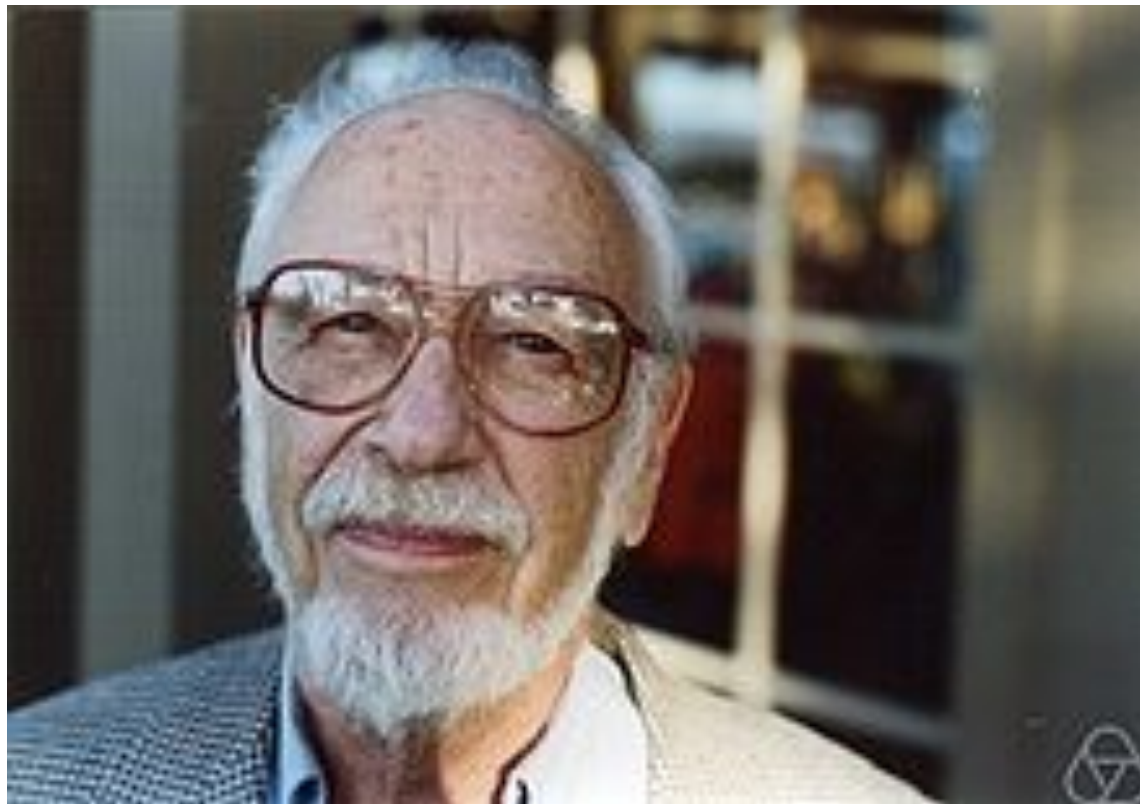
Check as you go.

Pair instabilities together.

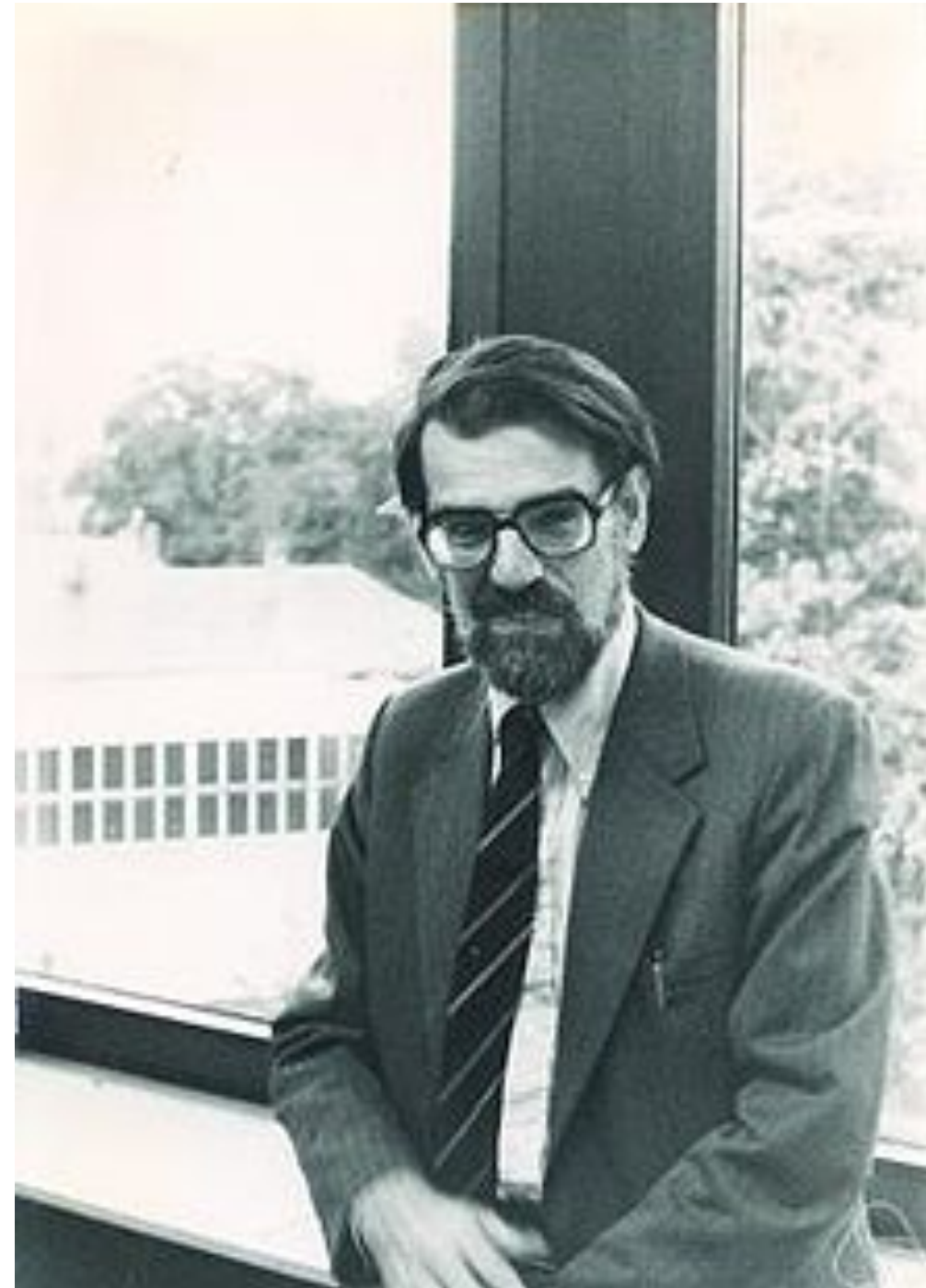
How to make precise?

Gale-Shapley: one side proposes, the other side enforces stability.

# Gale-Shapley Algorithm



David Gale



Lloyd Shapley

# Gale-Shapley Algorithm

Informal, high-level idea:

One side, say the men, propose to their favorite choices.

Each woman will accept her favorite among those who propose to her. Then they are “engaged.”

Unengaged men propose to their next favorite. Engagements are broken when a woman receives a proposal she prefers.

Stop once everyone is engaged. Marry!

# G-S on an example

Ex: Men = {1, 2, 3}. Women = {A, B, C}

1 ranks: A, C, B	A ranks: 3, 2, 1
2 ranks: A, B, C	B ranks: 1, 3, 2
3 ranks: C, B, A	C ranks: 2, 3, 1

Men propose: Round 1

# G-S on an example

Ex: Men = {1, 2, 3}. Women = {A, B, C}

1 ranks: A, C, B      A ranks: 3, 2, 1

2 ranks: A, B, C      B ranks: 1, 3, 2

3 ranks: C, B, A      C ranks: 2, 3, 1

Men propose: Round 1

# G-S on an example

Ex: Men = {1, 2, 3}. Women = {A, B, C}

1 ranks: A, C, B      A ranks: 3, 2, 1

2 ranks: A, B, C      B ranks: 1, 3, 2

3 ranks: C, B, A      C ranks: 2, 3, 1

Men propose: Round 1

# G-S on an example

Ex: Men = {1, 2, 3}. Women = {A, B, C}

1 ranks:  C, B

A ranks: 3, , 

2 ranks:  A, B, C

B ranks: 1, 3, 2

3 ranks: C, B, A

C ranks: 2, 3, 1

Men propose: Round 1



# G-S on an example

Ex: Men = {1, 2, 3}. Women = {A, B, C}

1 ranks:  C, B

A ranks: 3, , 

2 ranks:  A, B, C

B ranks: 1, 3, 2

3 ranks:  C, B, A

C ranks: 2, , 1

Men propose: Round 1

# G-S on an example

Ex: Men = {1, 2, 3}. Women = {A, B, C}

1 ranks:  **C**, B      A ranks: 3, **2**, 


2 ranks: **A**, B, C      B ranks: 1, 3, 2

3 ranks: **C**, B, A      C ranks: 2, **3**, **1**

Men propose: Round 2

# G-S on an example

Ex: Men = {1, 2, 3}. Women = {A, B, C}

1 ranks:   B

A ranks: 3, , 

2 ranks: , B, C

B ranks: 1, 3, 2

3 ranks: , B, A

C ranks: 2, , 1

Men propose: Round 2

# G-S on an example

Ex: Men = {1, 2, 3}. Women = {A, B, C}

1 ranks:   **B**      A ranks: 3, **2**, 

2 ranks: **A**, B, C      B ranks: **1**, 3, 2

3 ranks: **C**, B, A      C ranks: 2, **3**, 1

Men propose: Round 3

# G-S on an example

Ex: Men = {1, 2, 3}. Women = {A, B, C}

1 ranks: A, C, **B**      A ranks: 3, **2**, 1

2 ranks: **A**, B, C      B ranks: **1**, 3, 2

3 ranks: **C**, B, A      C ranks: 2, **3**, 1

Men propose: Round 3

Output: (1, B) (2, A) (3, C)

# Gale-Shapley, pseudocode

Initially, all men and women are free

While (exists  $m$  in {free men})

    Let  $w$  = “top” woman in  $m$ 's pref list

    If ( $w$  is free): ( $m, w$ ) become engaged

    else if ( $w$  engaged to  $m'$  but prefers  $m$ ) {  
         $m'$  becomes free again.

$m'$  removes  $w$  from his pref list.

        ( $m, w$ ) become engaged.

    } else { //  $w$  engaged to  $m'$ , prefers  $m'$

$m$  stays free, removes  $w$  from pref list

    } // (see page 6 in text).

# Gale-Shapley, analysis

We need to show:

- (1) Gale-Shapley always terminates and produces correct output.
  - (a) perfect matching
  - (b) stable
- (2) Running time doesn't blow up too fast, as a function of  $n$ .

But first--some basic tools!

# First Assignments

Admin: Email me -- due yesterday.

Reading: Chapter 1 this weekend  
Secs 2.1-2.4 by next weekend

Written Assignment (from K&T):

Chapter 1, problems 1,2,3,4,6

Programming Assignment: coming soon!