CS 361
Data Structures & Algs
Lecture 3

Prof. Tom Hayes
Guest Lecturer: Michael Janes
University of New Mexico
08-28-2012

Last Time: Stable Matching

Defined: n x n stable matching problem, with complete preference lists. Matching, perfect matching, instability, stable matching.

Gale-Shapley alg claims to find one of the stable matchings. Went over pseudocode.

Today: Analyzing the algorithm (does it really work, and how fast?)

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

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:	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
```

How long? TOO LONG!

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

n! = n*(n-1)*...*2*1   poly(n)

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: 💔 💔 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

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.

Let's do (2) first!

Gale-Shapley, runtime

How many proposals will be made?

Gale-Shapley, runtime

How many proposals will be made?

Claim: *Each man proposes to each woman at most once.*

Gale-Shapley, runtime

How many proposals will be made?

Claim: *Each man proposes to each woman at most once.*

Proof: look at pseudocode. After m proposes to w, 2 possibilities:
(a) accepted. Then m is no longer free.

(b) rejected. Then w is removed from m's preference list (permanently).

How many proposals will be made?

Claim: *Each man proposes to each woman at most once.*

Proof: look at pseudocode. After m proposes to w, 2 possibilities:
(a) accepted. Then m is no longer free.

(b) rejected. Then w is removed from m's preference list (permanently).

m won't propose at all until free.

when m becomes free, he removes w from his preference list (permanently).

Gale-Shapley, runtime

How many proposals will be made?

Claim: *Each man proposes to each woman at most once.*

So, at most n^2 proposals in all.

O(1) work per proposal. (Details! Data structures! Later. O(n) obvious. Why?)

Thus $O(n^2)$ work overall. ($O(n^3)$ obvious)

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. *Checked: poly(n)*

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. *Checked: poly(n)*

Next: Show correctness. (1)

Gale-Shapley, correctness

(1) Check: Program doesn’t crash.

By run-time analysis, we know that the program terminates in time poly(n). Now we will also know it produces output.

(2) Check: The output must be a perfect matching.

(3) Check: there cannot be any instabilities in the output.

Invariants

Properties that are true at the beginning (or end) of every loop iteration are called “loop invariants.”

Often these are *key to correctness* of the program.

Prove these by induction on the number of loop iterations. Need to show, if P holds at the beginning of the loop body, then P still holds at the end of the loop body.

A Loop Invariant for G-S

At the end of each round, every woman is either:

(1) engaged, or

(2) has never been proposed to.

PROOF: Initially, all women satisfy (2).

Only 1 woman has her status altered in each round. And she always ends the round engaged (see code).

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).
```

A Second Loop Invariant

Inv 1: At the end of each round, every woman is either:

(1) engaged, or

(2) has never been proposed to.

Inv 2: At the end of each round, the set of engaged pairs forms a matching. That is, no woman or man is doubly-engaged.

PROOF: again, see the code. Clear initially.

Correctness of G-S

Inv 1: At the end of each round, every woman is either engaged or has never been proposed to.

Inv 2: At the end of each round, the set of engaged pairs forms a matching.

Now, from this, we deduce: no single man ever has an empty preference list (i.e. runs out of women to propose to). Why?

Correctness of G-S

Inv 1: At the end of each round, every woman is either engaged or has never been proposed to.

Inv 2: At the end of each round, the set of engaged pairs forms a matching.

From this, we deduce: no single man ever has an empty preference list (i.e. runs out of women to propose to). Why? Then all N women would be engaged by Inv 1. But by Inv 2, that means all N men are engaged, a contradiction.

Correctness of G-S

Inv 1: At the end of each round, every woman is either engaged or has never been proposed to.

Inv 2: At the end of each round, the set of engaged pairs forms a matching.

Lemma: No single man ever has an empty preference list (i.e. runs out of women to propose to).

Thus, the algorithm eventually terminates and outputs a perfect matching.

Correctness of G-S

Inv 1: At the end of each round, every woman is either engaged or has never been proposed to.

Inv 2: At the end of each round, the set of engaged pairs forms a matching.

Lemma: No single man ever has an empty preference list (i.e. runs out of women to propose to).

Thus, the algorithm eventually terminates and outputs a perfect matching.

But, is it stable?

Correctness of G-S

The algorithm eventually terminates and outputs a perfect matching.

But, is it stable?

Inv 3: Any engaged woman prefers her current fiance over any past proposer.

Inv 4: Any engaged man prefers w to his current fiancee if and only if he previously proposed to w.

Proof of invariant 3

Inv 3: Any engaged woman prefers her current fiancé over any past proposer.

PROOF: Only the woman w proposed to in round i can change her fiance. And she only accepts a new fiance if she prefers him to her old fiance, whom she already preferred to all previous proposers (inductive hypothesis)

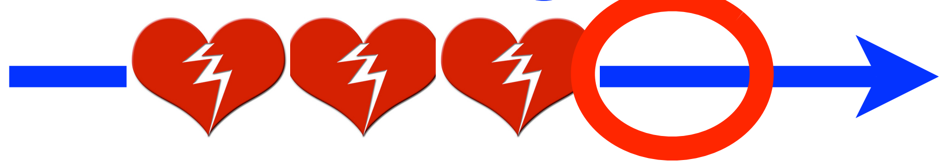
Proof of invariant 4

Inv 4: Any engaged man prefers w to his current fiancee if and only if he previously proposed to w.

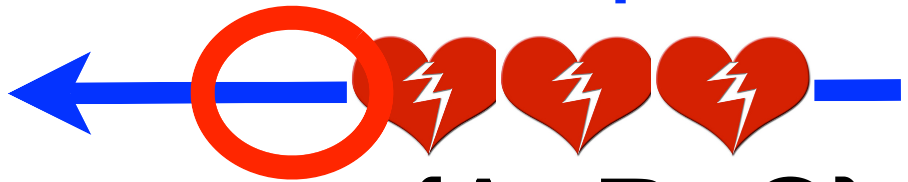
PROOF: Each man always proposes to the highest ranked woman on his list. Since women are only removed from the list after they have been proposed to, this means every higher-ranked woman than the most recent proposee (i.e. the fiancee) has already been proposed to.

G-S on an example

Inv 4: fiancees get worse



Inv 3: fiances improve



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

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

Inv 3: Any engaged woman prefers her current fiance over any past proposer.

Inv 4: Any engaged man prefers w to his current fiancee if and only if he previously proposed to w.

Theorem: The matching output by G-S must be a stable matching.

PROOF: Suppose (m,w) is an instability. By Inv 4, m must have proposed to w. But by Inv 3, w prefers her current fiance. So it’s not an instability after all.

General Lessons

To show correctness, find key invariants.
Prove them by induction.

Finding these gives new insight.

Issues: Where could the code crash?
(assumptions violated)

Must the code terminate (and how fast)?

Why can't the output be wrong?