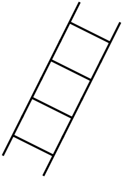
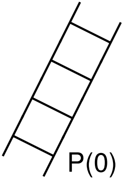
Climb an infinite ladder?

Climb an infinite ladder?



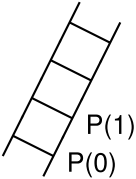
Climb an infinite ladder?

$P(0)$



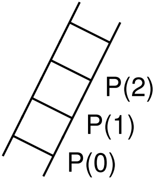
Climb an infinite ladder?

$$P(0)$$
$$\forall k, P(k) \implies P(k+1)$$



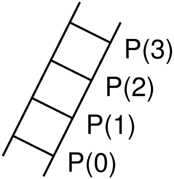
Climb an infinite ladder?

$$P(0)$$
$$\forall k, P(k) \implies P(k+1)$$
$$P(0) \implies P(1) \implies P(2)$$

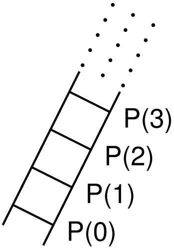


Climb an infinite ladder?

$$P(0)$$
$$\forall k, P(k) \implies P(k+1)$$
$$P(0) \implies P(1) \implies P(2) \implies P(3)$$

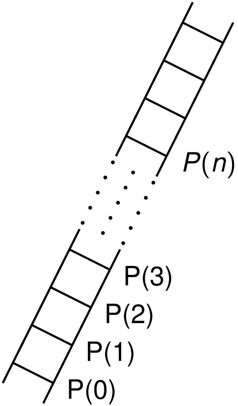


Climb an infinite ladder?



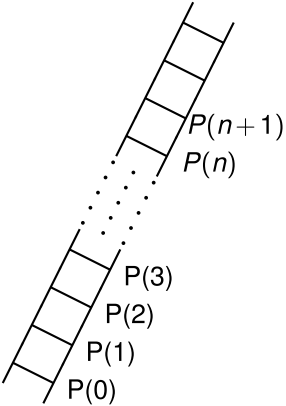
$$P(0)$$
$$\forall k, P(k) \implies P(k+1)$$
$$P(0) \implies P(1) \implies P(2) \implies P(3) \ldots$$

Climb an infinite ladder?



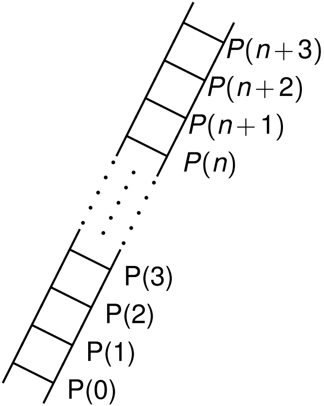
$$P(0)$$
$$\forall k, P(k) \implies P(k+1)$$
$$P(0) \implies P(1) \implies P(2) \implies P(3) \ldots$$

Climb an infinite ladder?



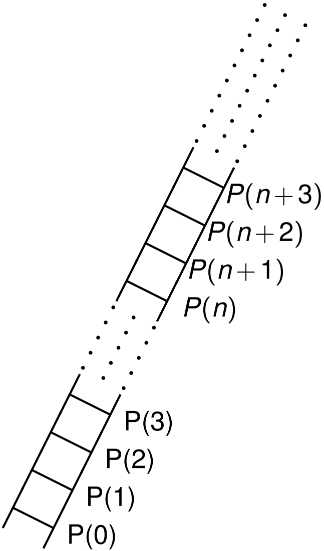
$$P(0)$$
$$\forall k, P(k) \implies P(k+1)$$
$$P(0) \implies P(1) \implies P(2) \implies P(3) \ldots$$

Climb an infinite ladder?



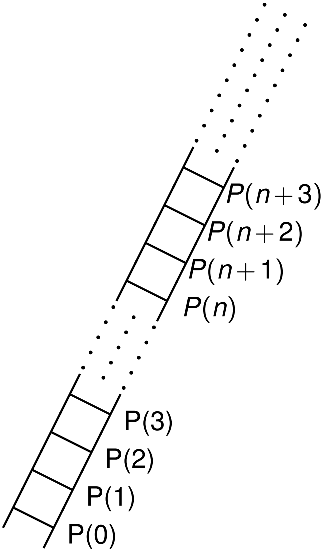
$$P(0)$$
$$\forall k, P(k) \implies P(k+1)$$
$$P(0) \implies P(1) \implies P(2) \implies P(3) \ldots$$

Climb an infinite ladder?



$$P(0)$$
$$\forall k, P(k) \implies P(k+1)$$
$$P(0) \implies P(1) \implies P(2) \implies P(3) \ldots$$
$$(\forall n \in N) P(n)$$

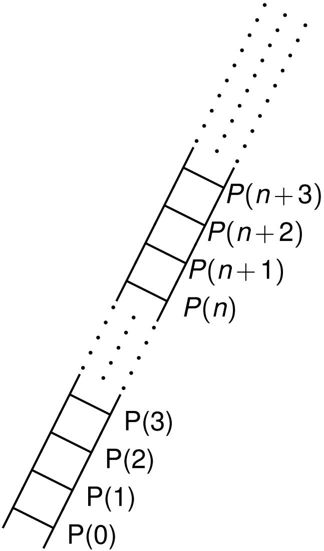
Climb an infinite ladder?



$$P(0)$$
$$\forall k, P(k) \implies P(k+1)$$
$$P(0) \implies P(1) \implies P(2) \implies P(3) \ldots$$
$$(\forall n \in N) P(n)$$

Your favorite example of forever..

Climb an infinite ladder?



$$P(0)$$
$$\forall k, P(k) \implies P(k+1)$$
$$P(0) \implies P(1) \implies P(2) \implies P(3) \ldots$$
$$(\forall n \in N) P(n)$$

Your favorite example of forever..or the natural numbers...

Tournaments have long paths.

Def: A **round robin tournament on** n **players**: all pairs p and q play, and either $p \to q$ (p beats q) or $q \to p$ (q beats q.)

Tournaments have long paths.

Def: A **round robin tournament on** n **players**: all pairs p and q play, and either $p \rightarrow q$ (p beats q) or $q \rightarrow p$ (q beats q.)

Def: A **Hamiltonian path**: a sequence

Tournaments have long paths.

Def: A **round robin tournament on** n **players**: all pairs p and q play, and either $p \rightarrow q$ (p beats q) or $q \rightarrow p$ (q beats q.)

Def: A **Hamiltonian path**: a sequence
$p_1, \ldots, p_n$, $(\forall i, 0 \leq i < n)$ $p_i \rightarrow p_{i+1}$.

Tournaments have long paths.

Def: A **round robin tournament on n players**: all pairs p and q play, and either $p \to q$ (p beats q) or $q \to p$ (q beats q.)

Def: A **Hamiltonian path**: a sequence
$p_1, \ldots, p_n$, $(\forall i, 0 \leq i < n)$ $p_i \to p_{i+1}$.

$$2 \to 1 \to \cdots \to 7$$

Tournaments have long paths.

Def: A **round robin tournament on** n **players**: all pairs p and q play, and either $p \to q$ (p beats q) or $q \to p$ (q beats q.)

Def: A **Hamiltonian path**: a sequence
$p_1, \ldots, p_n$, $(\forall i, 0 \leq i < n)\ p_i \to p_{i+1}$.

$2 \to 1 \to \cdots \to 7$

Base: True for two vertices.

Tournaments have long paths.

Def: A **round robin tournament on** n **players**: all pairs p and q play, and either $p \to q$ (p beats q) or $q \to p$ (q beats q.)

Def: A **Hamiltonian path**: a sequence

$p_1, \ldots, p_n$, $(\forall i, 0 \leq i < n)$ $p_i \to p_{i+1}$.

$$2 \to 1 \to \cdots \to 7$$

Base: True for two vertices. $1 \to 2$

Tournaments have long paths.

Def: A **round robin tournament on** n **players**: all pairs p and q play, and either $p \rightarrow q$ (p beats q) or $q \rightarrow p$ (q beats q.)

Def: A **Hamiltonian path**: a sequence

$p_1, \ldots, p_n$, $(\forall i, 0 \leq i < n)$ $p_i \rightarrow p_{i+1}$.

$$2 \rightarrow 1 \rightarrow \cdots \rightarrow 7$$

Base: True for two vertices. $1 \rightarrow 2$

(Also for one, but two is more fun as base case!)

Tournaments have long paths.

Def: A **round robin tournament on** n **players**: all pairs p and q play, and either $p \to q$ (p beats q) or $q \to p$ (q beats q.)

Def: A **Hamiltonian path**: a sequence
$p_1, \ldots, p_n$, $(\forall i, 0 \leq i < n)$ $p_i \to p_{i+1}$.

$2 \to 1 \to \cdots \to 7$

Base: True for two vertices. $1 \to 2$
(Also for one, but two is more fun as base case!)

Tournament on $n+1$ people,

Tournaments have long paths.

Def: A **round robin tournament on** n **players**: all pairs p and q play, and either $p \to q$ (p beats q) or $q \to p$ (q beats q.)

Def: A **Hamiltonian path**: a sequence
$p_1, \ldots, p_n$, $(\forall i, 0 \le i < n)$ $p_i \to p_{i+1}$.

$2 \to 1 \to \cdots \to 7$

Base: True for two vertices. $1 \to 2$
(Also for one, but two is more fun as base case!)

Tournament on $n+1$ people,
Remove arbitrary person

Tournaments have long paths.

Def: A **round robin tournament on** n **players**: all pairs p and q play, and either $p \to q$ (p beats q) or $q \to p$ (q beats q.)

Def: A **Hamiltonian path**: a sequence
$p_1, \ldots, p_n$, $(\forall i, 0 \le i < n)$ $p_i \to p_{i+1}$.

$2 \to 1 \to \cdots \to 7$

Base: True for two vertices. $1 \to 2$
(Also for one, but two is more fun as base case!)

Tournament on $n+1$ people,
Remove arbitrary person $\to$ yield tournament on $n-1$ people.

Tournaments have long paths.

Def: A **round robin tournament on *n* players**: all pairs p and q play, and either $p \to q$ (p beats q) or $q \to p$ (q beats q.)

Def: A **Hamiltonian path**: a sequence

$p_1, \ldots, p_n$, $(\forall i, 0 \leq i < n)\ p_i \to p_{i+1}$.

$$2 \to 1 \to \cdots \to 7$$

Base: True for two vertices. $1 \to 2$

(Also for one, but two is more fun as base case!)

Tournament on $n+1$ people,

Remove arbitrary person $\to$ yield tournament on $n-1$ people.

Tournaments have long paths.

Def: A **round robin tournament on** n **players**: all pairs p and q play, and either $p \to q$ (p beats q) or $q \to p$ (q beats q.)

Def: A **Hamiltonian path**: a sequence

$p_1, \ldots, p_n$, $(\forall i, 0 \leq i < n)$ $p_i \to p_{i+1}$.

$2 \to 1 \to \cdots \to 7$

Base: True for two vertices. $1 \to 2$

(Also for one, but two is more fun as base case!)

Tournament on $n+1$ people,

Remove arbitrary person $\to$ yield tournament on $n-1$ people.

By induction hypothesis: There is a sequence $p_1, \ldots, p_{n-1}$

Tournaments have long paths.

Def: A **round robin tournament on** n **players**: all pairs p and q play, and either $p \to q$ (p beats q) or $q \to p$ (q beats q.)

Def: A **Hamiltonian path**: a sequence
$p_1, \ldots, p_n$, $(\forall i, 0 \leq i < n)\ p_i \to p_{i+1}$.

$2 \to 1 \to \cdots \to 7$

Base: True for two vertices. $1 \to 2$
(Also for one, but two is more fun as base case!)

Tournament on $n+1$ people,
Remove arbitrary person $\to$ yield tournament on $n-1$ people.

By induction hypothesis: There is a sequence $p_1, \ldots, p_{n-1}$
contains all the people

Tournaments have long paths.

Def: A **round robin tournament on** n **players**: all pairs p and q play, and either $p \to q$ (p beats q) or $q \to p$ (q beats q.)

Def: A **Hamiltonian path**: a sequence
$p_1, \ldots, p_n$, $(\forall i, 0 \leq i < n)\ p_i \to p_{i+1}$.

$2 \to 1 \to \cdots \to 7$

Base: True for two vertices. $1 \to 2$
(Also for one, but two is more fun as base case!)

Tournament on $n+1$ people,
Remove arbitrary person $\to$ yield tournament on $n-1$ people.

By induction hypothesis: There is a sequence $p_1, \ldots, p_{n-1}$
contains all the people where $p_i \to p_{i+1}$

Tournaments have long paths.

Def: A **round robin tournament on** n **players**: all pairs p and q play, and either $p \to q$ (p beats q) or $q \to p$ (q beats q.)

Def: A **Hamiltonian path**: a sequence
$p_1, \ldots, p_n$, $(\forall i, 0 \leq i < n)\ p_i \to p_{i+1}$.

$$2 \to 1 \to \cdots \to 7$$

Base: True for two vertices. $1 \to 2$
(Also for one, but two is more fun as base case!)

Tournament on $n+1$ people,
Remove arbitrary person $\to$ yield tournament on $n-1$ people.

By induction hypothesis: There is a sequence $p_1, \ldots, p_{n-1}$
contains all the people where $p_i \to p_{i+1}$

$$a \to b \to \cdots \to c$$

Tournaments have long paths.

Def: A **round robin tournament on** n **players**: all pairs p and q play, and either $p \to q$ (p beats q) or $q \to p$ (q beats q.)

Def: A **Hamiltonian path**: a sequence
$p_1, \ldots, p_n$, $(\forall i, 0 \le i < n)$ $p_i \to p_{i+1}$.

$2 \to 1 \to \cdots \to 7$

Base: True for two vertices. $1 \to 2$
(Also for one, but two is more fun as base case!)

Tournament on $n+1$ people,
Remove arbitrary person $\to$ yield tournament on $n-1$ people.

By induction hypothesis: There is a sequence $p_1, \ldots, p_{n-1}$
contains all the people where $p_i \to p_{i+1}$

$a \to b \to \cdots \to c$

If p is big winner, put at beginning.

Tournaments have long paths.

Def: A **round robin tournament on** n **players**: all pairs p and q play, and either $p \to q$ (p beats q) or $q \to p$ (q beats q.)

Def: A **Hamiltonian path**: a sequence

$p_1, \ldots, p_n$, $(\forall i, 0 \leq i < n)$ $p_i \to p_{i+1}$.

$$2 \to 1 \to \cdots \to 7$$

Base: True for two vertices. $1 \to 2$

(Also for one, but two is more fun as base case!)

Tournament on $n+1$ people,

Remove arbitrary person $\to$ yield tournament on $n-1$ people.

By induction hypothesis: There is a sequence $p_1, \ldots, p_{n-1}$

contains all the people where $p_i \to p_{i+1}$

$$\text{W} \to \text{a} \to \text{b} \to \cdots \to \text{c}$$

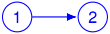
If p is big winner, put at beginning.

Tournaments have long paths.

Def: A **round robin tournament on** n **players**: all pairs p and q play, and either $p \to q$ (p beats q) or $q \to p$ (q beats q.)

Def: A **Hamiltonian path**: a sequence
$p_1, \ldots, p_n$, $(\forall i, 0 \leq i < n)$ $p_i \to p_{i+1}$.

Base: True for two vertices.



(Also for one, but two is more fun as base case!)

Tournament on $n+1$ people,
Remove arbitrary person $\to$ yield tournament on $n-1$ people.

By induction hypothesis: There is a sequence $p_1, \ldots, p_{n-1}$
contains all the people where $p_i \to p_{i+1}$

If p is big winner, put at beginning. Big loser at end.

Tournaments have long paths.

Def: A **round robin tournament on** n **players**: all pairs p and q play, and either $p \to q$ (p beats q) or $q \to p$ (q beats q.)

Def: A **Hamiltonian path**: a sequence
$p_1, \ldots, p_n$, $(\forall i, 0 \leq i < n)$ $p_i \to p_{i+1}$.

$$2 \to 1 \to \cdots \to 7$$

Base: True for two vertices. $1 \to 2$
(Also for one, but two is more fun as base case!)

Tournament on $n+1$ people,
Remove arbitrary person $\to$ yield tournament on $n-1$ people.

By induction hypothesis: There is a sequence $p_1, \ldots, p_{n-1}$
contains all the people where $p_i \to p_{i+1}$

If p is big winner, put at beginning. Big loser at end.
If neither, find first place i, where p beats p_i.

Tournaments have long paths.

Def: A **round robin tournament on** n **players**: all pairs p and q play, and either $p \rightarrow q$ (p beats q) or $q \rightarrow p$ (q beats q.)

Def: A **Hamiltonian path**: a sequence
$p_1, \ldots, p_n$, $(\forall i, 0 \leq i < n)$ $p_i \rightarrow p_{i+1}$.

$2 \rightarrow 1 \rightarrow \cdots \rightarrow 7$

Base: True for two vertices. $1 \rightarrow 2$
(Also for one, but two is more fun as base case!)

Tournament on $n+1$ people,
Remove arbitrary person $\rightarrow$ yield tournament on $n-1$ people.

By induction hypothesis: There is a sequence $p_1, \ldots, p_{n-1}$
contains all the people where $p_i \rightarrow p_{i+1}$

$a \rightarrow b \rightarrow \cdots \rightarrow m \rightarrow c$

If p is big winner, put at beginning. Big loser at end.
If neither, find first place i, where p beats p_i.

Tournaments have long paths.

Def: A **round robin tournament on n players**: all pairs p and q play, and either $p \to q$ (p beats q) or $q \to p$ (q beats q.)

Def: A **Hamiltonian path**: a sequence
$p_1,\ldots,p_n$, $(\forall i, 0 \le i < n)$ $p_i \to p_{i+1}$.

$2 \to 1 \to \cdots \to 7$

Base: True for two vertices. $1 \to 2$
(Also for one, but two is more fun as base case!)

Tournament on $n+1$ people,
Remove arbitrary person $\to$ yield tournament on $n-1$ people.

By induction hypothesis: There is a sequence $p_1,\ldots,p_{n-1}$
contains all the people where $p_i \to p_{i+1}$

$a \to b \to \cdots \to m \to c$

If p is big winner, put at beginning. Big loser at end.
If neither, find first place i, where p beats p_i.
$p_1,\ldots,p_{i-1},p,p_i,\ldots p_{n-1}$ is Hamiltonion path.

Tournaments have long paths.

Def: A **round robin tournament on** n **players**: all pairs p and q play, and either $p \to q$ (p beats q) or $q \to p$ (q beats q.)

Def: A **Hamiltonian path**: a sequence
$p_1, \ldots, p_n$, $(\forall i, 0 \leq i < n)$ $p_i \to p_{i+1}$.

$$2 \to 1 \to \cdots \to 7$$

Base: True for two vertices. $1 \to 2$
(Also for one, but two is more fun as base case!)

Tournament on $n+1$ people,
Remove arbitrary person $\to$ yield tournament on $n-1$ people.

By induction hypothesis: There is a sequence $p_1, \ldots, p_{n-1}$ contains all the people where $p_i \to p_{i+1}$

$$a \to b \to \cdots \to m \to c$$

If p is big winner, put at beginning. Big loser at end.
If neither, find first place i, where p beats p_i.
$p_1, \ldots, p_{i-1}, p, p_i, \ldots p_{n-1}$ is Hamiltonion path.

$\square$

Sad Islanders...

Island with 100 possibly blue-eyed and green-eyed inhabitants.

Sad Islanders...

Island with 100 possibly blue-eyed and green-eyed inhabitants.

Any islander who knows they have green eyes must “leave the island” that day.

Sad Islanders...

Island with 100 possibly blue-eyed and green-eyed inhabitants.

Any islander who knows they have green eyes must “leave the island” that day.

No islander knows there own eye color, but knows everyone elses.

Sad Islanders...

Island with 100 possibly blue-eyed and green-eyed inhabitants.

Any islander who knows they have green eyes must “leave the island” that day.

No islander knows there own eye color, but knows everyone elses.

All islanders have green eyes!

Sad Islanders...

Island with 100 possibly blue-eyed and green-eyed inhabitants.

Any islander who knows they have green eyes must “leave the island” that day.

No islander knows there own eye color, but knows everyone elses.

All islanders have green eyes!

First rule of island:

Sad Islanders...

Island with 100 possibly blue-eyed and green-eyed inhabitants.

Any islander who knows they have green eyes must “leave the island” that day.

No islander knows there own eye color, but knows everyone elses.

All islanders have green eyes!

First rule of island: Don’t talk about eye color!

Sad Islanders...

Island with 100 possibly blue-eyed and green-eyed inhabitants.

Any islander who knows they have green eyes must “leave the island” that day.

No islander knows there own eye color, but knows everyone elses.

All islanders have green eyes!

First rule of island: Don’t talk about eye color!

Visitor: “I see someone has green eyes.”

Sad Islanders...

Island with 100 possibly blue-eyed and green-eyed inhabitants.

Any islander who knows they have green eyes must “leave the island” that day.

No islander knows there own eye color, but knows everyone elses.

All islanders have green eyes!

First rule of island: Don’t talk about eye color!

Visitor: “I see someone has green eyes.”

Result: What happens?
(A) Nothing, no information was added.
(B) Information was added, maybe?
(C) They all leave the island.
(D) They all leave the island on day 100.

Sad Islanders...

Island with 100 possibly blue-eyed and green-eyed inhabitants.

Any islander who knows they have green eyes must "leave the island" that day.

No islander knows there own eye color, but knows everyone elses.

All islanders have green eyes!

First rule of island: Don't talk about eye color!

Visitor: "I see someone has green eyes."

Result: What happens?
(A) Nothing, no information was added.
(B) Information was added, maybe?
(C) They all leave the island.
(D) They all leave the island on day 100.

On day 100,

Sad Islanders...

Island with 100 possibly blue-eyed and green-eyed inhabitants.

Any islander who knows they have green eyes must “leave the island” that day.

No islander knows there own eye color, but knows everyone elses.

All islanders have green eyes!

First rule of island: Don’t talk about eye color!

Visitor: “I see someone has green eyes.”

Result: What happens?
(A) Nothing, no information was added.
(B) Information was added, maybe?
(C) They all leave the island.
(D) They all leave the island on day 100.

On day 100, they all leave.

Sad Islanders...

Island with 100 possibly blue-eyed and green-eyed inhabitants.

Any islander who knows they have green eyes must “leave the island” that day.

No islander knows there own eye color, but knows everyone elses.

All islanders have green eyes!

First rule of island: Don’t talk about eye color!

Visitor: “I see someone has green eyes.”

Result: What happens?
(A) Nothing, no information was added.
(B) Information was added, maybe?
(C) They all leave the island.
(D) They all leave the island on day 100.

On day 100, they all leave.

Why?

They know induction.

Thm: If there are n villagers with green eyes they leave on day n.

They know induction.

Thm: If there are n villagers with green eyes they leave on day n.

Proof:
Base: $n = 1$. Person with green eyes leaves on day 1.

They know induction.

Thm: If there are n villagers with green eyes they leave on day n.

Proof:
Base: $n = 1$. Person with green eyes leaves on day 1.

Induction hypothesis:

They know induction.

Thm: If there are n villagers with green eyes they leave on day n.

Proof:
Base: $n = 1$. Person with green eyes leaves on day 1.

Induction hypothesis:
If n people with green eyes, they would leave on day n.

They know induction.

Thm: If there are n villagers with green eyes they leave on day n.

Proof:
Base: $n = 1$. Person with green eyes leaves on day 1.

Induction hypothesis:
If n people with green eyes, they would leave on day n.

Induction step:
On day $n+1$, a green eyed person sees n people with green eyes.

They know induction.

Thm: If there are n villagers with green eyes they leave on day n.

Proof:
Base: $n = 1$. Person with green eyes leaves on day 1.

Induction hypothesis:
If n people with green eyes, they would leave on day n.

Induction step:
On day $n+1$, a green eyed person sees n people with green eyes.

But they didn't leave.

They know induction.

Thm: If there are n villagers with green eyes they leave on day n.

Proof:
Base: $n = 1$. Person with green eyes leaves on day 1.

Induction hypothesis:
If n people with green eyes, they would leave on day n.

Induction step:
On day $n+1$, a green eyed person sees n people with green eyes.

But they didn't leave.

So there must be $n+1$ people with green eyes.

They know induction.

Thm: If there are n villagers with green eyes they leave on day n.

Proof:
Base: $n = 1$. Person with green eyes leaves on day 1.

Induction hypothesis:
If n people with green eyes, they would leave on day n.

Induction step:
On day $n+1$, a green eyed person sees n people with green eyes.

But they didn't leave.

So there must be $n+1$ people with green eyes.

One of them, is me.

They know induction.

Thm: If there are n villagers with green eyes they leave on day n.

Proof:
Base: $n = 1$. Person with green eyes leaves on day 1.

Induction hypothesis:
If n people with green eyes, they would leave on day n.

Induction step:
On day $n+1$, a green eyed person sees n people with green eyes.

But they didn't leave.

So there must be $n+1$ people with green eyes.

One of them, is me.

I have to leave the island.

They know induction.

Thm: If there are n villagers with green eyes they leave on day n.

Proof:
Base: $n = 1$. Person with green eyes leaves on day 1.

Induction hypothesis:
If n people with green eyes, they would leave on day n.

Induction step:
On day $n+1$, a green eyed person sees n people with green eyes.

But they didn't leave.

So there must be $n+1$ people with green eyes.

One of them, is me.

I have to leave the island. I like the island.

They know induction.

Thm: If there are n villagers with green eyes they leave on day n.

Proof:
Base: $n = 1$. Person with green eyes leaves on day 1.

Induction hypothesis:
If n people with green eyes, they would leave on day n.

Induction step:
On day $n+1$, a green eyed person sees n people with green eyes.

But they didn't leave.

So there must be $n+1$ people with green eyes.

One of them, is me.

I have to leave the island. I like the island. **SAD.**

They know induction.

Thm: If there are n villagers with green eyes they leave on day n.

Proof:
Base: $n = 1$. Person with green eyes leaves on day 1.

Induction hypothesis:
If n people with green eyes, they would leave on day n.

Induction step:
On day $n+1$, a green eyed person sees n people with green eyes.

But they didn't leave.

So there must be $n+1$ people with green eyes.

One of them, is me.

I have to leave the island. I like the island. **SAD.** $\square$

They know induction.

Thm: If there are n villagers with green eyes they leave on day n.

Proof:
Base: $n = 1$. Person with green eyes leaves on day 1.

Induction hypothesis:
If n people with green eyes, they would leave on day n.

Induction step:
On day $n+1$, a green eyed person sees n people with green eyes.

But they didn't leave.

So there must be $n+1$ people with green eyes.

One of them, is me.

I have to leave the island. I like the island. **SAD.** $\square$

Wait! Visitor added no information.

Common Knowledge.

Using knowledge about what other people's knowledge (your eye color) is.

Common Knowledge.

Using knowledge about what other people's knowledge (your eye color) is.

On day 1, everyone knows everyone sees more than zero.

Common Knowledge.

Using knowledge about what other people's knowledge (your eye color) is.

On day 1, everyone knows everyone sees more than zero.

Common Knowledge.

Using knowledge about what other people's knowledge (your eye color) is.

On day 1, everyone knows everyone sees more than zero.

On day 2, everyone knows everyone sees more than one.

Common Knowledge.

Using knowledge about what other people's knowledge (your eye color) is.

On day 1, everyone knows everyone sees more than zero.

On day 2, everyone knows everyone sees more than one.

Common Knowledge.

Using knowledge about what other people's knowledge (your eye color) is.

On day 1, everyone knows everyone sees more than zero.

On day 2, everyone knows everyone sees more than one.

...

Common Knowledge.

Using knowledge about what other people's knowledge (your eye color) is.

On day 1, everyone knows everyone sees more than zero.

On day 2, everyone knows everyone sees more than one.

...

On day 99, everyone knows no one sees 98

Common Knowledge.

Using knowledge about what other people's knowledge (your eye color) is.

On day 1, everyone knows everyone sees more than zero.

On day 2, everyone knows everyone sees more than one.

...

On day 99, everyone knows no one sees 98
since everyone knows everyone else does not see 97...

Common Knowledge.

Using knowledge about what other people's knowledge (your eye color) is.

On day 1, everyone knows everyone sees more than zero.

On day 2, everyone knows everyone sees more than one.

...

On day 99, everyone knows no one sees 98
since everyone knows everyone else does not see 97...

On day 100,

Common Knowledge.

Using knowledge about what other people's knowledge (your eye color) is.

On day 1, everyone knows everyone sees more than zero.

On day 2, everyone knows everyone sees more than one.

...

On day 99, everyone knows no one sees 98
since everyone knows everyone else does not see 97...

On day 100, ...uh oh!

Common Knowledge.

Using knowledge about what other people's knowledge (your eye color) is.

On day 1, everyone knows everyone sees more than zero.

On day 2, everyone knows everyone sees more than one.

...

On day 99, everyone knows no one sees 98
since everyone knows everyone else does not see 97...

On day 100, ...uh oh!

Another example:

Common Knowledge.

Using knowledge about what other people's knowledge (your eye color) is.

On day 1, everyone knows everyone sees more than zero.

On day 2, everyone knows everyone sees more than one.

...

On day 99, everyone knows no one sees 98
since everyone knows everyone else does not see 97...

On day 100, ...uh oh!

Another example:
Emperor's new clothes!

Common Knowledge.

Using knowledge about what other people's knowledge (your eye color) is.

On day 1, everyone knows everyone sees more than zero.

On day 2, everyone knows everyone sees more than one.

...

On day 99, everyone knows no one sees 98
since everyone knows everyone else does not see 97...

On day 100, ...uh oh!

Another example:
Emperor's new clothes!
No one knows other people see that he has no clothes.

Common Knowledge.

Using knowledge about what other people's knowledge (your eye color) is.

On day 1, everyone knows everyone sees more than zero.

On day 2, everyone knows everyone sees more than one.

...

On day 99, everyone knows no one sees 98
since everyone knows everyone else does not see 97...

On day 100, ...uh oh!

Another example:
Emperor's new clothes!
No one knows other people see that he has no clothes.
Until kid points it out.

Common Knowledge.

Using knowledge about what other people's knowledge (your eye color) is.

On day 1, everyone knows everyone sees more than zero.

On day 2, everyone knows everyone sees more than one.

...

On day 99, everyone knows no one sees 98
since everyone knows everyone else does not see 97...

On day 100, ...uh oh!

Another example:
Emperor's new clothes!
No one knows other people see that he has no clothes.
Until kid points it out.

Political arguments?

Common Knowledge.

Using knowledge about what other people's knowledge (your eye color) is.

On day 1, everyone knows everyone sees more than zero.

On day 2, everyone knows everyone sees more than one.

...

On day 99, everyone knows no one sees 98
since everyone knows everyone else does not see 97...

On day 100, ...uh oh!

Another example:
Emperor's new clothes!
No one knows other people see that he has no clothes.
Until kid points it out.

Political arguments?

Common Knowledge.

Using knowledge about what other people's knowledge (your eye color) is.

On day 1, everyone knows everyone sees more than zero.

On day 2, everyone knows everyone sees more than one.

...

On day 99, everyone knows no one sees 98
since everyone knows everyone else does not see 97...

On day 100, ...uh oh!

Another example:
Emperor's new clothes!
No one knows other people see that he has no clothes.
Until kid points it out.

Political arguments?

Everyone knows the arguments,

Common Knowledge.

Using knowledge about what other people's knowledge (your eye color) is.

On day 1, everyone knows everyone sees more than zero.

On day 2, everyone knows everyone sees more than one.

...

On day 99, everyone knows no one sees 98
since everyone knows everyone else does not see 97...

On day 100, ...uh oh!

Another example:
Emperor's new clothes!
No one knows other people see that he has no clothes.
Until kid points it out.

Political arguments?

Everyone knows the arguments,
everyone knows everyone knows the arguments.....

Common Knowledge.

Using knowledge about what other people's knowledge (your eye color) is.

On day 1, everyone knows everyone sees more than zero.

On day 2, everyone knows everyone sees more than one.

...

On day 99, everyone knows no one sees 98
since everyone knows everyone else does not see 97...

On day 100, ...uh oh!

Another example:
Emperor's new clothes!
No one knows other people see that he has no clothes.
Until kid points it out.

Political arguments?

Everyone knows the arguments,
everyone knows everyone knows the arguments.....
The islanders didn't talk.

Common Knowledge.

Using knowledge about what other people's knowledge (your eye color) is.

On day 1, everyone knows everyone sees more than zero.

On day 2, everyone knows everyone sees more than one.

...

On day 99, everyone knows no one sees 98
since everyone knows everyone else does not see 97...

On day 100, ...uh oh!

Another example:
Emperor's new clothes!
No one knows other people see that he has no clothes.
Until kid points it out.

Political arguments?

Everyone knows the arguments,
everyone knows everyone knows the arguments.....
The islanders didn't talk. Induction is quieter.

Common Knowledge.

Using knowledge about what other people's knowledge (your eye color) is.

On day 1, everyone knows everyone sees more than zero.

On day 2, everyone knows everyone sees more than one.

...

On day 99, everyone knows no one sees 98
since everyone knows everyone else does not see 97...

On day 100, ...uh oh!

Another example:
Emperor's new clothes!
No one knows other people see that he has no clothes.
Until kid points it out.

Political arguments?

Everyone knows the arguments,
everyone knows everyone knows the arguments.....
The islanders didn't talk. Induction is quieter.

Strong Induction and Recursion.

Thm: For every natural number $n \geq 12$, $n = 4x + 5y$.

Strong Induction and Recursion.

Thm: For every natural number $n \geq 12$, $n = 4x + 5y$.

Instead of proof, let's write some code!

Strong Induction and Recursion.

Thm: For every natural number $n \geq 12$, $n = 4x + 5y$.

Instead of proof, let's write some code!

```
def find-x-y(n):
   if (n==12) return (3,0)
   elif (n==13): return(2,1)
   elif (n==14): return(1,2)
   elif (n==15): return(0,3)
   else:
     (x',y') = find-x-y(n-4)
     return(x'+1,y')
```

Strong Induction and Recursion.

Thm: For every natural number $n \geq 12$, $n = 4x + 5y$.

Instead of proof, let's write some code!

```
def find-x-y(n):
   if (n==12) return (3,0)
   elif (n==13): return(2,1)
   elif (n==14): return(1,2)
   elif (n==15): return(0,3)
   else:
     (x',y') = find-x-y(n-4)
     return(x'+1,y')
```

Prove: Given n, returns (x, y) where $n = 4x + 5y$, for $n \geq 12$.

Strong Induction and Recursion.

Thm: For every natural number $n \geq 12$, $n = 4x + 5y$.

Instead of proof, let's write some code!

```
def find-x-y(n):
   if (n==12) return (3,0)
   elif (n==13): return(2,1)
   elif (n==14): return(1,2)
   elif (n==15): return(0,3)
   else:
     (x',y') = find-x-y(n-4)
     return(x'+1,y')
```

Prove: Given n, returns (x,y) where $n = 4x + 5y$, for $n \geq 12$.

Base cases:

Strong Induction and Recursion.

Thm: For every natural number $n \geq 12$, $n = 4x + 5y$.

Instead of proof, let's write some code!

```
def find-x-y(n):
   if (n==12) return (3,0)
   elif (n==13): return(2,1)
   elif (n==14): return(1,2)
   elif (n==15): return(0,3)
   else:
     (x',y') = find-x-y(n-4)
     return(x'+1,y')
```

Prove: Given n, returns (x, y) where $n = 4x + 5y$, for $n \geq 12$.

Base cases: P(12)

Strong Induction and Recursion.

Thm: For every natural number $n \geq 12$, $n = 4x + 5y$.

Instead of proof, let's write some code!

```
def find-x-y(n):
   if (n==12) return (3,0)
   elif (n==13): return(2,1)
   elif (n==14): return(1,2)
   elif (n==15): return(0,3)
   else:
     (x',y') = find-x-y(n-4)
     return(x'+1,y')
```

Prove: Given n, returns (x, y) where $n = 4x + 5y$, for $n \geq 12$.

Base cases: P(12) , P(13)

Strong Induction and Recursion.

Thm: For every natural number $n \geq 12$, $n = 4x + 5y$.

Instead of proof, let's write some code!

```
def find-x-y(n):
   if (n==12) return (3,0)
   elif (n==13): return(2,1)
   elif (n==14): return(1,2)
   elif (n==15): return(0,3)
   else:
     (x',y') = find-x-y(n-4)
     return(x'+1,y')
```

Prove: Given n, returns (x, y) where $n = 4x + 5y$, for $n \geq 12$.

Base cases: P(12) , P(13) , P(14)

Strong Induction and Recursion.

Thm: For every natural number $n \geq 12$, $n = 4x + 5y$.

Instead of proof, let's write some code!

```
def find-x-y(n):
   if (n==12) return (3,0)
   elif (n==13): return(2,1)
   elif (n==14): return(1,2)
   elif (n==15): return(0,3)
   else:
     (x',y') = find-x-y(n-4)
     return(x'+1,y')
```

Prove: Given n, returns (x, y) where $n = 4x + 5y$, for $n \geq 12$.

Base cases: P(12) , P(13) , P(14) , P(15).

Strong Induction and Recursion.

Thm: For every natural number $n \geq 12$, $n = 4x + 5y$.

Instead of proof, let's write some code!

```
def find-x-y(n):
   if (n==12) return (3,0)
   elif (n==13): return(2,1)
   elif (n==14): return(1,2)
   elif (n==15): return(0,3)
   else:
     (x',y') = find-x-y(n-4)
     return(x'+1,y')
```

Prove: Given n, returns (x,y) where $n = 4x + 5y$, for $n \geq 12$.

Base cases: P(12) , P(13) , P(14) , P(15). Yes.

Strong Induction and Recursion.

Thm: For every natural number $n \geq 12$, $n = 4x + 5y$.

Instead of proof, let's write some code!

```
def find-x-y(n):
   if (n==12) return (3,0)
   elif (n==13): return(2,1)
   elif (n==14): return(1,2)
   elif (n==15): return(0,3)
   else:
     (x',y') = find-x-y(n-4)
     return(x'+1,y')
```

Prove: Given n, returns (x,y) where $n = 4x + 5y$, for $n \geq 12$.

Base cases: P(12) , P(13) , P(14) , P(15). Yes.

Strong Induction step:

Strong Induction and Recursion.

Thm: For every natural number $n \geq 12$, $n = 4x + 5y$.

Instead of proof, let's write some code!

```
def find-x-y(n):
   if (n==12) return (3,0)
   elif (n==13): return(2,1)
   elif (n==14): return(1,2)
   elif (n==15): return(0,3)
   else:
     (x',y') = find-x-y(n-4)
     return(x'+1,y')
```

Prove: Given n, returns (x, y) where $n = 4x + 5y$, for $n \geq 12$.

Base cases: P(12) , P(13) , P(14) , P(15). Yes.

Strong Induction step:
Recursive call is correct: $P(n-4)$

Strong Induction and Recursion.

Thm: For every natural number $n \geq 12$, $n = 4x + 5y$.

Instead of proof, let's write some code!

```
def find-x-y(n):
   if (n==12) return (3,0)
   elif (n==13): return(2,1)
   elif (n==14): return(1,2)
   elif (n==15): return(0,3)
   else:
     (x',y') = find-x-y(n-4)
     return(x'+1,y')
```

Prove: Given n, returns (x, y) where $n = 4x + 5y$, for $n \geq 12$.

Base cases: P(12) , P(13) , P(14) , P(15). Yes.

Strong Induction step:
Recursive call is correct: $P(n-4) \implies P(n)$.

Strong Induction and Recursion.

Thm: For every natural number $n \geq 12$, $n = 4x + 5y$.

Instead of proof, let's write some code!

```
def find-x-y(n):
   if (n==12) return (3,0)
   elif (n==13): return(2,1)
   elif (n==14): return(1,2)
   elif (n==15): return(0,3)
   else:
     (x',y') = find-x-y(n-4)
     return(x'+1,y')
```

Prove: Given n, returns (x, y) where $n = 4x + 5y$, for $n \geq 12$.

Base cases: P(12) , P(13) , P(14) , P(15). Yes.

Strong Induction step:
Recursive call is correct: $P(n-4) \implies P(n)$.
$n - 4 = 4x' + 5y' \implies n = 4(x'+1) + 5(y')$

Strong Induction and Recursion.

Thm: For every natural number $n \geq 12$, $n = 4x + 5y$.

Instead of proof, let's write some code!

```
def find-x-y(n):
   if (n==12) return (3,0)
   elif (n==13): return(2,1)
   elif (n==14): return(1,2)
   elif (n==15): return(0,3)
   else:
      (x',y') = find-x-y(n-4)
      return(x'+1,y')
```

Prove: Given n, returns (x, y) where $n = 4x + 5y$, for $n \geq 12$.

Base cases: P(12) , P(13) , P(14) , P(15). Yes.

Strong Induction step:
Recursive call is correct: $P(n-4) \implies P(n)$.
$n - 4 = 4x' + 5y' \implies n = 4(x'+1) + 5(y')$

Strong Induction and Recursion.

Thm: For every natural number $n \geq 12$, $n = 4x + 5y$.

Instead of proof, let's write some code!

```
def find-x-y(n):
   if (n==12) return (3,0)
   elif (n==13): return(2,1)
   elif (n==14): return(1,2)
   elif (n==15): return(0,3)
   else:
     (x',y') = find-x-y(n-4)
     return(x'+1,y')
```

Prove: Given n, returns (x,y) where $n = 4x + 5y$, for $n \geq 12$.

Base cases: P(12) , P(13) , P(14) , P(15). Yes.

Strong Induction step:
Recursive call is correct: $P(n-4) \implies P(n)$.
$n-4 = 4x' + 5y' \implies n = 4(x'+1) + 5(y')$

Slight differences: showed for all $n \geq 16$ that $\wedge_{i=4}^{n-1} P(i) \implies P(n)$.

Tidying up induction.

The induction principle works on the natural numbers.

Tidying up induction.

The induction principle works on the natural numbers.

Proves statements of form: $\forall n \in \mathbb{N}, P(n)$.

Tidying up induction.

The induction principle works on the natural numbers.

Proves statements of form: $\forall n \in \mathbb{N}, P(n)$.

Yes.

Tidying up induction.

The induction principle works on the natural numbers.

Proves statements of form: $\forall n \in \mathbb{N}, P(n)$.

Yes.

What if the statement is only for $n \geq 3$?

Tidying up induction.

The induction principle works on the natural numbers.

Proves statements of form: $\forall n \in \mathbb{N}, P(n)$.

Yes.

What if the statement is only for $n \geq 3$?

$\forall n \in \mathbb{N}, (n \geq 3) \implies P(n)$

Tidying up induction.

The induction principle works on the natural numbers.

Proves statements of form: $\forall n \in \mathbb{N}, P(n)$.

Yes.

What if the statement is only for $n \geq 3$?

$\forall n \in \mathbb{N}, (n \geq 3) \implies P(n)$

Restate as:

Tidying up induction.

The induction principle works on the natural numbers.

Proves statements of form: $\forall n \in \mathbb{N}, P(n)$.

Yes.

What if the statement is only for $n \geq 3$?

$\forall n \in \mathbb{N}, (n \geq 3) \implies P(n)$

Restate as:
$\forall n \in \mathbb{N}, Q(n)$ where $Q(n) =$ "$(n \geq 3) \implies P(n)$".

Tidying up induction.

The induction principle works on the natural numbers.

Proves statements of form: $\forall n \in \mathbb{N}, P(n)$.

Yes.

What if the statement is only for $n \geq 3$?

$\forall n \in \mathbb{N}, (n \geq 3) \implies P(n)$

Restate as:
$\forall n \in \mathbb{N}, Q(n)$ where $Q(n) =$ "$(n \geq 3) \implies P(n)$".

Base Case: typically start at 3.

Tidying up induction.

The induction principle works on the natural numbers.

Proves statements of form: $\forall n \in \mathbb{N}, P(n)$.

Yes.

What if the statement is only for $n \geq 3$?

$\forall n \in \mathbb{N}, (n \geq 3) \implies P(n)$

Restate as:
$\forall n \in \mathbb{N}, Q(n)$ where $Q(n) =$ "$(n \geq 3) \implies P(n)$".

Base Case: typically start at 3.
Since $\forall n \in \mathbb{N}, Q(n) \implies Q(n+1)$ is trivially true before 3.

Tidying up induction.

The induction principle works on the natural numbers.

Proves statements of form: $\forall n \in \mathbb{N}, P(n)$.

Yes.

What if the statement is only for $n \geq 3$?

$\forall n \in \mathbb{N}, (n \geq 3) \implies P(n)$

Restate as:
$\forall n \in \mathbb{N}, Q(n)$ where $Q(n) =$ "$(n \geq 3) \implies P(n)$".

Base Case: typically start at 3.
Since $\forall n \in \mathbb{N}, Q(n) \implies Q(n+1)$ is trivially true before 3.

Can you do induction over other things? Yes.

Tidying up induction.

The induction principle works on the natural numbers.

Proves statements of form: $\forall n \in \mathbb{N}, P(n)$.

Yes.

What if the statement is only for $n \geq 3$?

$\forall n \in \mathbb{N}, (n \geq 3) \implies P(n)$

Restate as:
$\forall n \in \mathbb{N}, Q(n)$ where $Q(n) =$ "$(n \geq 3) \implies P(n)$".

Base Case: typically start at 3.
Since $\forall n \in \mathbb{N}, Q(n) \implies Q(n+1)$ is trivially true before 3.

Can you do induction over other things? Yes.

Any set where any subset of the set has a smallest element.

Tidying up induction.

The induction principle works on the natural numbers.

Proves statements of form: $\forall n \in \mathbb{N}, P(n)$.

Yes.

What if the statement is only for $n \geq 3$?

$\forall n \in \mathbb{N}, (n \geq 3) \implies P(n)$

Restate as:
$\forall n \in \mathbb{N}, Q(n)$ where $Q(n) =$ "$(n \geq 3) \implies P(n)$".

Base Case: typically start at 3.
Since $\forall n \in \mathbb{N}, Q(n) \implies Q(n+1)$ is trivially true before 3.

Can you do induction over other things? Yes.

Any set where any subset of the set has a smallest element.

In some sense, the natural numbers.

Stable Matching Problem

Stable Matching Problem

- n candidates and n jobs.

Stable Matching Problem

- n candidates and n jobs.
- Each job has a ranked preference list of candidates.

Stable Matching Problem

- n candidates and n jobs.
- Each job has a ranked preference list of candidates.
- Each candidate has a ranked preference list of jobs.

Stable Matching Problem

- n candidates and n jobs.
- Each job has a ranked preference list of candidates.
- Each candidate has a ranked preference list of jobs.

How should they be matched?

- Maximize total satisfaction.

Stable Matching Problem

- n candidates and n jobs.
- Each job has a ranked preference list of candidates.
- Each candidate has a ranked preference list of jobs.

How should they be matched?

- Maximize total satisfaction.
- Maximize number of first choices.

Stable Matching Problem

- n candidates and n jobs.
- Each job has a ranked preference list of candidates.
- Each candidate has a ranked preference list of jobs.

How should they be matched?

- Maximize total satisfaction.
- Maximize number of first choices.
- Maximize worse off.

Stable Matching Problem

- n candidates and n jobs.
- Each job has a ranked preference list of candidates.
- Each candidate has a ranked preference list of jobs.

How should they be matched?

- Maximize total satisfaction.
- Maximize number of first choices.
- Maximize worse off.
- Minimize difference between preference ranks.

The best laid plans..

Consider the pairs..

- (Anthony) Davis and Pelicans
- (Lonzo) Ball and Lakers

The best laid plans..

Consider the pairs..

- (Anthony) Davis and Pelicans
- (Lonzo) Ball and Lakers

Davis prefers the Lakers.

The best laid plans..

Consider the pairs..

- (Anthony) Davis and Pelicans
- (Lonzo) Ball and Lakers

Davis prefers the Lakers.

Lakers prefer Davis.

The best laid plans..

Consider the pairs..

- (Anthony) Davis and Pelicans
- (Lonzo) Ball and Lakers

Davis prefers the Lakers.

Lakers prefer Davis.

Uh..oh.

The best laid plans..

Consider the pairs..

- (Anthony) Davis and Pelicans
- (Lonzo) Ball and Lakers

Davis prefers the Lakers.

Lakers prefer Davis.

Uh..oh. Sad Lonzo and Pelicans.

So..

Produce a matching where there are no crazy moves!

So..

Produce a matching where there are no crazy moves!

Definition: A **matching** is disjoint set of n job-candidate pairs.

So..

Produce a matching where there are no crazy moves!

Definition: A **matching** is disjoint set of n job-candidate pairs.

Example: A matching $S = \{(\textit{Lakers}, \textit{Ball}); (\textit{Pelicans}, \textit{Davis})\}$.

So..

Produce a matching where there are no crazy moves!

Definition: A **matching** is disjoint set of n job-candidate pairs.

Example: A matching $S = \{(Lakers, Ball); (Pelicans, Davis)\}$.

Definition: A **rogue couple** b, g^* for a pairing S:
b and g^* prefer each other to their partners in S

So..

Produce a matching where there are no crazy moves!

Definition: A **matching** is disjoint set of n job-candidate pairs.

Example: A matching $S = \{(Lakers, Ball); (Pelicans, Davis)\}$.

Definition: A **rogue couple** b, g^* for a pairing S:
b and g^* prefer each other to their partners in S

Example: Davis and Lakers are a rogue couple in S.

A stable matching??

Given a set of preferences.

A stable matching??

Given a set of preferences.

Is there a stable matching?

A stable matching??

Given a set of preferences.

Is there a stable matching?

How does one find it?

A stable matching??

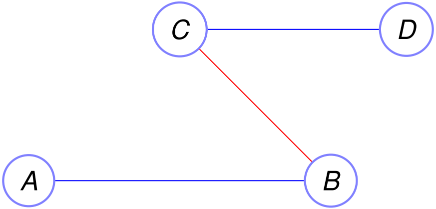
Given a set of preferences.

Is there a stable matching?

How does one find it?

Consider a single type version: stable roommates.

A	B	C	D
B	C	A	D
C	A	B	D
D	A	B	C



A stable matching??

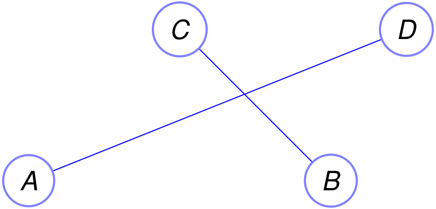
Given a set of preferences.

Is there a stable matching?

How does one find it?

Consider a single type version: stable roommates.

A	B	C	D
B	C	A	D
C	A	B	D
D	A	B	C



A stable matching??

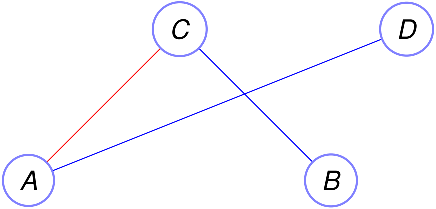
Given a set of preferences.

Is there a stable matching?

How does one find it?

Consider a single type version: stable roommates.

A	B	C	D
B	C	A	D
C	A	B	D
D	A	B	C



A stable matching??

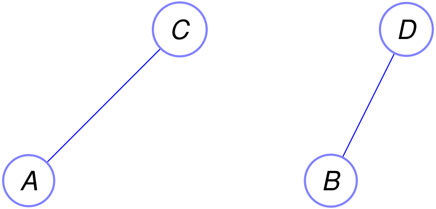
Given a set of preferences.

Is there a stable matching?

How does one find it?

Consider a single type version: stable roommates.

A	B	C	D
B	C	A	D
C	A	B	D
D	A	B	C



A stable matching??

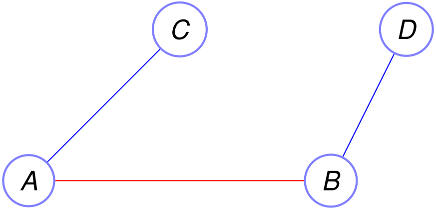
Given a set of preferences.

Is there a stable matching?

How does one find it?

Consider a single type version: stable roommates.

A	B	C	D
B	C	A	D
C	A	B	D
D	A	B	C



A stable matching??

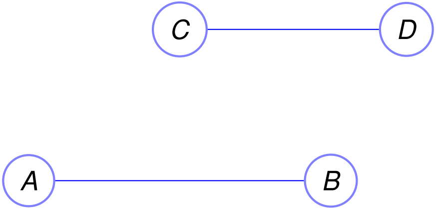
Given a set of preferences.

Is there a stable matching?

How does one find it?

Consider a single type version: stable roommates.

A	B	C	D
B	C	A	D
C	A	B	D
D	A	B	C



A stable matching??

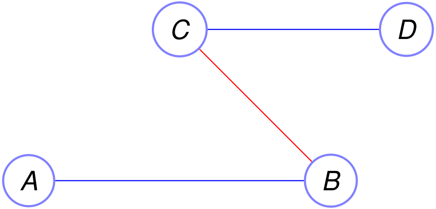
Given a set of preferences.

Is there a stable matching?

How does one find it?

Consider a single type version: stable roommates.

A	B	C	D
B	C	A	D
C	A	B	D
D	A	B	C



A stable matching??

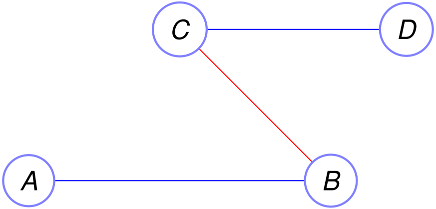
Given a set of preferences.

Is there a stable matching?

How does one find it?

Consider a single type version: stable roommates.

A	B	C	D
B	C	A	D
C	A	B	D
D	A	B	C



A stable matching??

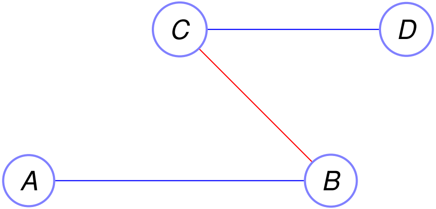
Given a set of preferences.

Is there a stable matching?

How does one find it?

Consider a single type version: stable roommates.

A	B	C	D
B	C	A	D
C	A	B	D
D	A	B	C



Example.

Jobs			
A	1	2	3
B	1	2	3
C	2	1	3

Candidates			
1	C	A	B
2	A	B	C
3	A	C	B

Example.

	Jobs		
A	1	2	3
B	1	2	3
C	2	1	3

	Candidates		
1	C	A	B
2	A	B	C
3	A	C	B

	Day 1	Day 2	Day 3	Day 4	Day 5
1					
2					
3					

Example.

	Jobs		
A	1	2	3
B	1	2	3
C	2	1	3

	Candidates		
1	C	A	B
2	A	B	C
3	A	C	B

	Day 1	Day 2	Day 3	Day 4	Day 5
1	A, B				
2	C				
3					

Example.

Jobs			
A	1	2	3
B	~~1~~	2	3
C	2	1	3

Candidates			
1	C	A	B
2	A	B	C
3	A	C	B

	Day 1	Day 2	Day 3	Day 4	Day 5
1	A, ~~B~~				
2	C				
3					

Example.

Jobs			
A	1	2	3
B	~~1~~	2	3
C	2	1	3

Candidates			
1	C	A	B
2	A	B	C
3	A	C	B

	Day 1	Day 2	Day 3	Day 4	Day 5
1	A, ~~B~~	A			
2	C	B, C			
3					

Example.

	Jobs		
A	1	2	3
B	~~1~~	2	3
C	~~2~~	1	3

	Candidates		
1	C	A	B
2	A	B	C
3	A	C	B

	Day 1	Day 2	Day 3	Day 4	Day 5
1	A, ~~B~~	A			
2	C	B, ~~C~~			
3					

Example.

Jobs			
A	1	2	3
B	~~1~~	2	3
C	~~2~~	1	3

Candidates			
1	C	A	B
2	A	B	C
3	A	C	B

	Day 1	Day 2	Day 3	Day 4	Day 5
1	A, ~~B~~	A	A , C		
2	C	B, ~~C~~	B		
3					

Example.

Jobs			
A	~~1~~	2	3
B	~~1~~	2	3
C	~~2~~	1	3

Candidates			
1	C	A	B
2	A	B	C
3	A	C	B

	Day 1	Day 2	Day 3	Day 4	Day 5
1	A, ~~B~~	A	~~A~~, C		
2	C	B, ~~C~~	B		
3					

Example.

Jobs			
A	~~1~~	2	3
B	~~1~~	2	3
C	~~2~~	1	3

Candidates			
1	C	A	B
2	A	B	C
3	A	C	B

	Day 1	Day 2	Day 3	Day 4	Day 5
1	A, ~~B~~	A	~~A~~ , C	C	
2	C	B, ~~C~~	B	A,B	
3					

Example.

Jobs			
A	~~1~~	2	3
B	~~1~~	~~2~~	3
C	~~2~~	1	3

Candidates			
1	C	A	B
2	A	B	C
3	A	C	B

	Day 1	Day 2	Day 3	Day 4	Day 5
1	A, ~~B~~	A	~~A~~, C	C	
2	C	B, ~~C~~	B	A, ~~B~~	
3					

Example.

Jobs			
A	~~1~~	2	3
B	~~1~~	~~2~~	3
C	~~2~~	1	3

Candidates			
1	C	A	B
2	A	B	C
3	A	C	B

	Day 1	Day 2	Day 3	Day 4	Day 5
1	A, ~~B~~	A	~~A~~, C	C	C
2	C	B, ~~C~~	B	A, ~~B~~	A
3					B

Example.

Jobs			
A	~~1~~	2	3
B	~~1~~	~~2~~	3
C	~~2~~	1	3

Candidates			
1	C	A	B
2	A	B	C
3	A	C	B

	Day 1	Day 2	Day 3	Day 4	Day 5
1	A, ~~B~~	A	~~A~~, C	C	C
2	C	B, ~~C~~	B	A, ~~B~~	A
3					B

The Propose and Reject Algorithm.

The Propose and Reject Algorithm.

Each Day:

The Propose and Reject Algorithm.

Each Day:

1. Each job **proposes** to its favorite candidate on its list.

The Propose and Reject Algorithm.

Each Day:

1. Each job **proposes** to its favorite candidate on its list.
2. Each candidate rejects all but their favorite proposer (whom they put on a string.)

The Propose and Reject Algorithm.

Each Day:

1. Each job **proposes** to its favorite candidate on its list.
2. Each candidate rejects all but their favorite proposer (whom they put on a string.)
3. Rejected job crosses rejecting candidate off its list.

The Propose and Reject Algorithm.

Each Day:

1. Each job **proposes** to its favorite candidate on its list.
2. Each candidate rejects all but their favorite proposer (whom they put on a string.)
3. Rejected job crosses rejecting candidate off its list.

Stop when each job gets exactly one proposal.

The Propose and Reject Algorithm.

Each Day:

1. Each job **proposes** to its favorite candidate on its list.
2. Each candidate rejects all but their favorite proposer (whom they put on a string.)
3. Rejected job crosses rejecting candidate off its list.

Stop when each job gets exactly one proposal.
Does this terminate?

The Propose and Reject Algorithm.

Each Day:

1. Each job **proposes** to its favorite candidate on its list.
2. Each candidate rejects all but their favorite proposer
   (whom they put on a string.)
3. Rejected job crosses rejecting candidate off its list.

Stop when each job gets exactly one proposal.
Does this terminate?

...produce a matching?

The Propose and Reject Algorithm.

Each Day:

1. Each job **proposes** to its favorite candidate on its list.
2. Each candidate rejects all but their favorite proposer (whom they put on a string.)
3. Rejected job crosses rejecting candidate off its list.

Stop when each job gets exactly one proposal.
Does this terminate?

...produce a matching?

....a stable matching?

The Propose and Reject Algorithm.

Each Day:

1. Each job **proposes** to its favorite candidate on its list.
2. Each candidate rejects all but their favorite proposer (whom they put on a string.)
3. Rejected job crosses rejecting candidate off its list.

Stop when each job gets exactly one proposal.
Does this terminate?

...produce a matching?

....a stable matching?

Do jobs or candidates do “better”?

The Propose and Reject Algorithm.

Each Day:

1. Each job **proposes** to its favorite candidate on its list.
2. Each candidate rejects all but their favorite proposer (whom they put on a string.)
3. Rejected job crosses rejecting candidate off its list.

Stop when each job gets exactly one proposal.
Does this terminate?

...produce a matching?

....a stable matching?

Do jobs or candidates do “better”?

Termination.

Termination.

Every non-terminated day a job **crossed** an item off the list.

Termination.

Every non-terminated day a job **crossed** an item off the list.

Total size of lists?

Termination.

Every non-terminated day a job **crossed** an item off the list.

Total size of lists? n jobs, n length list.

Termination.

Every non-terminated day a job **crossed** an item off the list.

Total size of lists? n jobs, n length list. n^2

Termination.

Every non-terminated day a job **crossed** an item off the list.

Total size of lists? n jobs, n length list. n^2

Terminates in $\leq n^2$ steps!

It gets better every day for candidates.

It gets better every day for candidates.

Improvement Lemma: It just gets better for candidates

It gets better every day for candidates.

Improvement Lemma: It just gets better for candidates
If on day t a candidate g has a job b on a string,

It gets better every day for candidates.

Improvement Lemma: It just gets better for candidates
If on day t a candidate g has a job b on a string,
any job, b', on candidate g's string for any day $t' > t$

It gets better every day for candidates.

Improvement Lemma: It just gets better for candidates
If on day t a candidate g has a job b on a string,
any job, b', on candidate g's string for any day $t' > t$
is at least as good as b.

It gets better every day for candidates.

Improvement Lemma: It just gets better for candidates
If on day t a candidate g has a job b on a string,
any job, b', on candidate g's string for any day $t' > t$
is at least as good as b.

Example: Candidate “Alice” has job “Amalmagated Concrete” on string on day 5.

It gets better every day for candidates.

Improvement Lemma: It just gets better for candidates
If on day t a candidate g has a job b on a string,
any job, b', on candidate g's string for any day $t' > t$
is at least as good as b.

Example: Candidate "Alice" has job "Amalmagated Concrete" on string on day 5.

She has job "Amalmagated Asphalt" on string on day 7.

It gets better every day for candidates.

Improvement Lemma: It just gets better for candidates
If on day t a candidate g has a job b on a string,
any job, b', on candidate g's string for any day $t' > t$
is at least as good as b.

Example: Candidate "Alice" has job "Amalmagated Concrete" on string on day 5.

She has job "Amalmagated Asphalt" on string on day 7.

It gets better every day for candidates.

Improvement Lemma: It just gets better for candidates
If on day t a candidate g has a job b on a string,
any job, b', on candidate g's string for any day $t' > t$
is at least as good as b.

Example: Candidate "Alice" has job "Amalmagated Concrete" on string on day 5.

She has job "Amalmagated Asphalt" on string on day 7.

Does Alice prefer "Almalmagated Asphalt" or "Amalmagated Concrete"?

It gets better every day for candidates.

Improvement Lemma: It just gets better for candidates
If on day t a candidate g has a job b on a string,
any job, b', on candidate g's string for any day $t' > t$
is at least as good as b.

Example: Candidate "Alice" has job "Amalmagated Concrete" on string on day 5.

She has job "Amalmagated Asphalt" on string on day 7.

Does Alice prefer "Almalmagated Asphalt" or "Amalmagated Concrete"?

g - 'Alice', b - 'Am. Con.', b' - 'Am. Asph.', $t = 5$, $t' = 7$.

It gets better every day for candidates.

Improvement Lemma: It just gets better for candidates
If on day t a candidate g has a job b on a string,
any job, b', on candidate g's string for any day $t' > t$
is at least as good as b.

Example: Candidate “Alice” has job “Amalmagated Concrete” on string on day 5.

She has job “Amalmagated Asphalt” on string on day 7.

Does Alice prefer “Almalmagated Asphalt” or “Amalmagated Concrete”?

g - 'Alice', b - 'Am. Con.', b' - 'Am. Asph.', $t = 5$, $t' = 7$.

Improvement Lemma says she prefers 'Amalmagated Asphalt'.

It gets better every day for candidates.

Improvement Lemma: It just gets better for candidates
If on day t a candidate g has a job b on a string,
any job, b', on candidate g's string for any day $t' > t$
is at least as good as b.

Example: Candidate "Alice" has job "Amalmagated Concrete" on string on day 5.

She has job "Amalmagated Asphalt" on string on day 7.

Does Alice prefer "Almalmagated Asphalt" or "Amalmagated Concrete"?

g - 'Alice', b - 'Am. Con.', b' - 'Am. Asph.', $t = 5$, $t' = 7$.

Improvement Lemma says she prefers 'Amalmagated Asphalt'.

Day 10: Can Alice have "Amalmagated Asphalt" on her string?

It gets better every day for candidates.

Improvement Lemma: It just gets better for candidates
If on day t a candidate g has a job b on a string,
any job, b', on candidate g's string for any day $t' > t$
is at least as good as b.

Example: Candidate “Alice” has job “Amalmagated Concrete” on string on day 5.

She has job “Amalmagated Asphalt” on string on day 7.

Does Alice prefer “Almalmagated Asphalt” or “Amalmagated Concrete”?

g - 'Alice', b - 'Am. Con.', b' - 'Am. Asph.', $t = 5$, $t' = 7$.

Improvement Lemma says she prefers 'Amalmagated Asphalt'.

Day 10: Can Alice have “Amalmagated Asphalt” on her string? Yes.

It gets better every day for candidates.

Improvement Lemma: It just gets better for candidates
If on day t a candidate g has a job b on a string,
any job, b', on candidate g's string for any day $t' > t$
is at least as good as b.

Example: Candidate “Alice” has job “Amalmagated Concrete” on string on day 5.

She has job “Amalmagated Asphalt” on string on day 7.

Does Alice prefer “Almalmagated Asphalt” or “Amalmagated Concrete”?

g - 'Alice', b - 'Am. Con.', b' - 'Am. Asph.', $t = 5$, $t' = 7$.

Improvement Lemma says she prefers 'Amalmagated Asphalt'.

Day 10: Can Alice have “Amalmagated Asphalt” on her string? Yes.

Alice prefers day 10 job as much as day 7 job.

It gets better every day for candidates.

Improvement Lemma: It just gets better for candidates
If on day t a candidate g has a job b on a string,
any job, b', on candidate g's string for any day $t' > t$
is at least as good as b.

Example: Candidate "Alice" has job "Amalmagated Concrete" on string on day 5.

She has job "Amalmagated Asphalt" on string on day 7.

Does Alice prefer "Almalmagated Asphalt" or "Amalmagated Concrete"?

g - 'Alice', b - 'Am. Con.', b' - 'Am. Asph.', $t = 5$, $t' = 7$.

Improvement Lemma says she prefers 'Amalmagated Asphalt'.

Day 10: Can Alice have "Amalmagated Asphalt" on her string? Yes.

Alice prefers day 10 job as much as day 7 job. Here, $b = b'$.

It gets better every day for candidates.

Improvement Lemma: It just gets better for candidates
If on day t a candidate g has a job b on a string,
any job, b', on candidate g's string for any day $t' > t$
is at least as good as b.

Example: Candidate "Alice" has job "Amalmagated Concrete" on string on day 5.

She has job "Amalmagated Asphalt" on string on day 7.

Does Alice prefer "Almalmagated Asphalt" or "Amalmagated Concrete"?

g - 'Alice', b - 'Am. Con.', b' - 'Am. Asph.', $t = 5$, $t' = 7$.

Improvement Lemma says she prefers 'Amalmagated Asphalt'.

Day 10: Can Alice have "Amalmagated Asphalt" on her string? Yes.

Alice prefers day 10 job as much as day 7 job. Here, $b = b'$.

Why is lemma true?

It gets better every day for candidates.

Improvement Lemma: It just gets better for candidates
If on day t a candidate g has a job b on a string,
any job, b', on candidate g's string for any day $t' > t$
is at least as good as b.

Example: Candidate “Alice” has job “Amalmagated Concrete” on string on day 5.

She has job “Amalmagated Asphalt” on string on day 7.

Does Alice prefer “Almalmagated Asphalt” or “Amalmagated Concrete”?

g - 'Alice', b - 'Am. Con.', b' - 'Am. Asph.', $t = 5$, $t' = 7$.

Improvement Lemma says she prefers 'Amalmagated Asphalt'.

Day 10: Can Alice have “Amalmagated Asphalt” on her string? Yes.

Alice prefers day 10 job as much as day 7 job. Here, $b = b'$.

Why is lemma true?

It gets better every day for candidates.

Improvement Lemma: It just gets better for candidates
If on day t a candidate g has a job b on a string,
any job, b', on candidate g's string for any day $t' > t$
is at least as good as b.

Example: Candidate "Alice" has job "Amalmagated Concrete" on string on day 5.

She has job "Amalmagated Asphalt" on string on day 7.

Does Alice prefer "Almalmagated Asphalt" or "Amalmagated Concrete"?

g - 'Alice', b - 'Am. Con.', b' - 'Am. Asph.', $t = 5$, $t' = 7$.

Improvement Lemma says she prefers 'Amalmagated Asphalt'.

Day 10: Can Alice have "Amalmagated Asphalt" on her string? Yes.

Alice prefers day 10 job as much as day 7 job. Here, $b = b'$.

Why is lemma true?

Proof Idea:

It gets better every day for candidates.

Improvement Lemma: It just gets better for candidates
If on day t a candidate g has a job b on a string,
any job, b', on candidate g's string for any day $t' > t$
is at least as good as b.

Example: Candidate "Alice" has job "Amalmagated Concrete" on string on day 5.

She has job "Amalmagated Asphalt" on string on day 7.

Does Alice prefer "Almalmagated Asphalt" or "Amalmagated Concrete"?

g - 'Alice', b - 'Am. Con.', b' - 'Am. Asph.', $t = 5$, $t' = 7$.

Improvement Lemma says she prefers 'Amalmagated Asphalt'.

Day 10: Can Alice have "Amalmagated Asphalt" on her string? Yes.

Alice prefers day 10 job as much as day 7 job. Here, $b = b'$.

Why is lemma true?

Proof Idea: She can always keep the previous job on the string.

It gets better every day for candidates.

Improvement Lemma: It just gets better for candidates
If on day t a candidate g has a job b on a string,
any job, b', on candidate g's string for any day $t' > t$
is at least as good as b.

Example: Candidate "Alice" has job "Amalmagated Concrete" on string on day 5.

She has job "Amalmagated Asphalt" on string on day 7.

Does Alice prefer "Almalmagated Asphalt" or "Amalmagated Concrete"?

g - 'Alice', b - 'Am. Con.', b' - 'Am. Asph.', $t = 5$, $t' = 7$.

Improvement Lemma says she prefers 'Amalmagated Asphalt'.

Day 10: Can Alice have "Amalmagated Asphalt" on her string? Yes.

Alice prefers day 10 job as much as day 7 job. Here, $b = b'$.

Why is lemma true?

Proof Idea: She can always keep the previous job on the string.

Improvement Lemma

Improvement Lemma: It just gets better for candidates.

Improvement Lemma

Improvement Lemma: It just gets better for candidates.

If on day t a candidate g has a job b on a string, any job, b', on g's string for any day $t' > t$ is at least as good as b.

Improvement Lemma

Improvement Lemma: It just gets better for candidates.

If on day t a candidate g has a job b on a string, any job, b', on g's string for any day $t' > t$ is at least as good as b.

Proof:

Improvement Lemma

Improvement Lemma: It just gets better for candidates.

If on day t a candidate g has a job b on a string, any job, b', on g's string for any day $t' > t$ is at least as good as b.

Proof:
$P(k)$- - "job on g's string is at least as good as b on day $t+k$"

Improvement Lemma

Improvement Lemma: It just gets better for candidates.

If on day t a candidate g has a job b on a string, any job, b', on g's string for any day $t' > t$ is at least as good as b.

Proof:
$P(k)$- - "job on g's string is at least as good as b on day $t+k$"

$P(0)$– true. Candidate has b on string.

Improvement Lemma

Improvement Lemma: It just gets better for candidates.

If on day t a candidate g has a job b on a string, any job, b', on g's string for any day $t' > t$ is at least as good as b.

Proof:
$P(k)$- - "job on g's string is at least as good as b on day $t+k$"

$P(0)$– true. Candidate has b on string.

Assume $P(k)$. Let b' be job **on string** on day $t+k$.

Improvement Lemma

Improvement Lemma: It just gets better for candidates.

If on day t a candidate g has a job b on a string, any job, b', on g's string for any day $t' > t$ is at least as good as b.

Proof:
$P(k)$- - "job on g's string is at least as good as b on day $t+k$"

$P(0)$– true. Candidate has b on string.

Assume $P(k)$. Let b' be job **on string** on day $t+k$.

On day $t+k+1$, job b' comes back.

Improvement Lemma

Improvement Lemma: It just gets better for candidates.

If on day t a candidate g has a job b on a string, any job, b', on g's string for any day $t' > t$ is at least as good as b.

Proof:
$P(k)$- - "job on g's string is at least as good as b on day $t+k$"

$P(0)$– true. Candidate has b on string.

Assume $P(k)$. Let b' be job **on string** on day $t+k$.

On day $t+k+1$, job b' comes back.
 Candidate g can choose b',

Improvement Lemma

Improvement Lemma: It just gets better for candidates.

If on day t a candidate g has a job b on a string, any job, b', on g's string for any day $t' > t$ is at least as good as b.

Proof:
$P(k)$- - "job on g's string is at least as good as b on day $t+k$"

$P(0)$– true. Candidate has b on string.

Assume $P(k)$. Let b' be job **on string** on day $t+k$.

On day $t+k+1$, job b' comes back.
Candidate g can choose b', or do better with another job, b''

Improvement Lemma

Improvement Lemma: It just gets better for candidates.

If on day t a candidate g has a job b on a string, any job, b', on g's string for any day $t' > t$ is at least as good as b.

Proof:
$P(k)$- - "job on g's string is at least as good as b on day $t+k$"

$P(0)$– true. Candidate has b on string.

Assume $P(k)$. Let b' be job **on string** on day $t+k$.

On day $t+k+1$, job b' comes back.
Candidate g can choose b', or do better with another job, b''

That is,

Improvement Lemma

Improvement Lemma: It just gets better for candidates.

If on day t a candidate g has a job b on a string, any job, b', on g's string for any day $t' > t$ is at least as good as b.

Proof:
$P(k)$- - "job on g's string is at least as good as b on day $t+k$"

$P(0)$– true. Candidate has b on string.

Assume $P(k)$. Let b' be job **on string** on day $t+k$.

On day $t+k+1$, job b' comes back.
Candidate g can choose b', or do better with another job, b''

That is, $b' \geq b$ by induction hypothesis.

Improvement Lemma

Improvement Lemma: It just gets better for candidates.

If on day t a candidate g has a job b on a string, any job, b', on g's string for any day $t' > t$ is at least as good as b.

Proof:
$P(k)$- - "job on g's string is at least as good as b on day $t+k$"

$P(0)$– true. Candidate has b on string.

Assume $P(k)$. Let b' be job **on string** on day $t+k$.

On day $t+k+1$, job b' comes back.
Candidate g can choose b', or do better with another job, b''

That is, $b' \geq b$ by induction hypothesis.
And b'' is better than b' by algorithm.

Improvement Lemma

Improvement Lemma: It just gets better for candidates.

If on day t a candidate g has a job b on a string, any job, b', on g's string for any day $t' > t$ is at least as good as b.

Proof:
$P(k)$- - "job on g's string is at least as good as b on day $t+k$"

$P(0)$– true. Candidate has b on string.

Assume $P(k)$. Let b' be job **on string** on day $t+k$.

On day $t+k+1$, job b' comes back.
Candidate g can choose b', or do better with another job, b''

That is, $b' \geq b$ by induction hypothesis.
And b'' is better than b' by algorithm.
$\implies$ Candidate does at least as well as with b.

Improvement Lemma

Improvement Lemma: It just gets better for candidates.

If on day t a candidate g has a job b on a string, any job, b', on g's string for any day $t' > t$ is at least as good as b.

Proof:
$P(k)$- - "job on g's string is at least as good as b on day $t+k$"

$P(0)$– true. Candidate has b on string.

Assume $P(k)$. Let b' be job **on string** on day $t+k$.

On day $t+k+1$, job b' comes back.
Candidate g can choose b', or do better with another job, b''

That is, $b' \geq b$ by induction hypothesis.
And b'' is better than b' by algorithm.
$\implies$ Candidate does at least as well as with b.

$P(k) \implies P(k+1)$.

Improvement Lemma

Improvement Lemma: It just gets better for candidates.

If on day t a candidate g has a job b on a string, any job, b', on g's string for any day $t' > t$ is at least as good as b.

Proof:
$P(k)$- - "job on g's string is at least as good as b on day $t+k$"

$P(0)$– true. Candidate has b on string.

Assume $P(k)$. Let b' be job **on string** on day $t+k$.

On day $t+k+1$, job b' comes back.
Candidate g can choose b', or do better with another job, b''

That is, $b' \geq b$ by induction hypothesis.
And b'' is better than b' by algorithm.
$\implies$ Candidate does at least as well as with b.

$P(k) \implies P(k+1)$.
And by principle of induction, lemma holds for every day after t.

Improvement Lemma

Improvement Lemma: It just gets better for candidates.

If on day t a candidate g has a job b on a string, any job, b', on g's string for any day $t' > t$ is at least as good as b.

Proof:
$P(k)$- - "job on g's string is at least as good as b on day $t+k$"

$P(0)$– true. Candidate has b on string.

Assume $P(k)$. Let b' be job **on string** on day $t+k$.

On day $t+k+1$, job b' comes back.
Candidate g can choose b', or do better with another job, b''

That is, $b' \geq b$ by induction hypothesis.
And b'' is better than b' by algorithm.
$\Longrightarrow$ Candidate does at least as well as with b.

$P(k) \implies P(k+1)$.
And by principle of induction, lemma holds for every day after t. $\square$

Poll

Question: It just gets better for candidates,

(A) By induction on days.

(B) When the economy is good.

(C) The candidate can always keep the job on the string.

Poll

Question: It just gets better for candidates,

(A) By induction on days.

(B) When the economy is good.

(C) The candidate can always keep the job on the string.

(A) and (C).

Poll

Question: It just gets better for candidates,

(A) By induction on days.

(B) When the economy is good.

(C) The candidate can always keep the job on the string.

(A) and (C).

Sure on (B), but that Econ not CS.

Poll

Question: It just gets better for candidates,

(A) By induction on days.

(B) When the economy is good.

(C) The candidate can always keep the job on the string.

(A) and (C).

Sure on (B), but that Econ not CS.

Then again, stable matching is from Econ. Professsor's: Gale and Shapley.

Matching when done.

Lemma: Every job is matched at end.

Matching when done.

Lemma: Every job is matched at end.

Proof:

Matching when done.

Lemma: Every job is matched at end.

Proof:
If not, a job b must have been rejected n times.

Matching when done.

Lemma: Every job is matched at end.

Proof:
If not, a job b must have been rejected n times.

Every candidate has been proposed to by b,

Matching when done.

Lemma: Every job is matched at end.

Proof:
If not, a job *b* must have been rejected *n* times.

Every candidate has been proposed to by *b*,
and Improvement lemma

Matching when done.

Lemma: Every job is matched at end.

Proof:
If not, a job b must have been rejected n times.

Every candidate has been proposed to by b, and Improvement lemma

$\implies$ each candidate has a job on a string.

Matching when done.

Lemma: Every job is matched at end.

Proof:
If not, a job b must have been rejected n times.

Every candidate has been proposed to by b,
and Improvement lemma

$\implies$ each candidate has a job on a string.

and each job is on at most one string.

Matching when done.

Lemma: Every job is matched at end.

Proof:
If not, a job b must have been rejected n times.

Every candidate has been proposed to by b,
and Improvement lemma

$\implies$ each candidate has a job on a string.

and each job is on at most one string.

n candidates and n jobs.

Matching when done.

Lemma: Every job is matched at end.

Proof:
If not, a job b must have been rejected n times.

Every candidate has been proposed to by b,
and Improvement lemma

$\Longrightarrow$ each candidate has a job on a string.

and each job is on at most one string.

n candidates and n jobs. Same number of each.

Matching when done.

Lemma: Every job is matched at end.

Proof:
If not, a job b must have been rejected n times.

Every candidate has been proposed to by b,
and Improvement lemma

$\implies$ each candidate has a job on a string.

and each job is on at most one string.

n candidates and n jobs. Same number of each.

Matching when done.

Lemma: Every job is matched at end.

Proof:
If not, a job b must have been rejected n times.

Every candidate has been proposed to by b,
and Improvement lemma

$\implies$ each candidate has a job on a string.

and each job is on at most one string.

n candidates and n jobs. Same number of each.

$\implies$ b must be on some candidate's string!

Matching when done.

Lemma: Every job is matched at end.

Proof:
If not, a job b must have been rejected n times.

Every candidate has been proposed to by b,
and Improvement lemma

$\implies$ each candidate has a job on a string.

and each job is on at most one string.

n candidates and n jobs. Same number of each.

$\implies$ b must be on some candidate's string!

Matching when done.

Lemma: Every job is matched at end.

Proof:
If not, a job *b* must have been rejected *n* times.

Every candidate has been proposed to by *b*,
and Improvement lemma

$\Longrightarrow$ each candidate has a job on a string.

and each job is on at most one string.

n candidates and *n* jobs. Same number of each.

$\Longrightarrow$ *b* must be on some candidate's string!

Contradiction.

Matching when done.

Lemma: Every job is matched at end.

Proof:
If not, a job b must have been rejected n times.

Every candidate has been proposed to by b,
and Improvement lemma

$\implies$ each candidate has a job on a string.

and each job is on at most one string.

n candidates and n jobs. Same number of each.

$\implies$ b must be on some candidate's string!

Contradiction. $\square$

Question: The argument for termination ...

(A) Implies: no unmatched job at end.

(B) Uses Improvement Lemma: every candidate matched.

(C) From Algorithm: unmatched job would ask everyone.

(D) Implies: every one gets their favorite job.

Matching is Stable.

Lemma: There is no rogue couple for the matching formed by traditional marriage algorithm.

Matching is Stable.

Lemma: There is no rogue couple for the matching formed by traditional marriage algorithm.

Proof:
Assume there is a rogue couple; (b, g^*)

Matching is Stable.

Lemma: There is no rogue couple for the matching formed by traditional marriage algorithm.

Proof:
Assume there is a rogue couple; (b, g^*)

$$b^* \text{ ——— } g^*$$

$$b \text{ ——— } g$$

Matching is Stable.

Lemma: There is no rogue couple for the matching formed by traditional marriage algorithm.

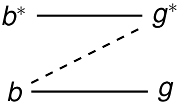
Proof:
Assume there is a rogue couple; (b, g^*)

$$\begin{tikzcd} b^* \arrow[r, dash] & g^* \\ b \arrow[r, dash] \arrow[ur, dash, dashed] & g \end{tikzcd}$$

Matching is Stable.

Lemma: There is no rogue couple for the matching formed by traditional marriage algorithm.

Proof:
Assume there is a rogue couple; (b, g^*)

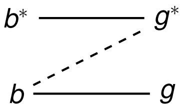


b prefers g^* to g.

Matching is Stable.

Lemma: There is no rogue couple for the matching formed by traditional marriage algorithm.

Proof:
Assume there is a rogue couple; (b, g^*)



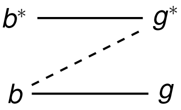
b prefers g^* to g.

g^* prefers b to b^*.

Matching is Stable.

Lemma: There is no rogue couple for the matching formed by traditional marriage algorithm.

Proof:
Assume there is a rogue couple; (b, g^*)



b prefers g^* to g.

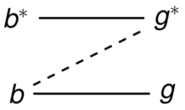
g^* prefers b to b^*.

Job b proposes to g^* before proposing to g.

Matching is Stable.

Lemma: There is no rogue couple for the matching formed by traditional marriage algorithm.

Proof:
Assume there is a rogue couple; (b, g^*)



b prefers g^* to g.

g^* prefers b to b^*.

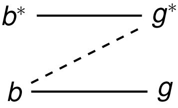
Job b proposes to g^* before proposing to g.

So g^* rejected b (since he moved on)

Matching is Stable.

Lemma: There is no rogue couple for the matching formed by traditional marriage algorithm.

Proof:
Assume there is a rogue couple; (b, g^*)



b prefers g^* to g.

g^* prefers b to b^*.

Job b proposes to g^* before proposing to g.

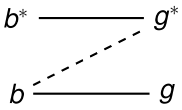
So g^* rejected b (since he moved on)

By improvement lemma, g^* prefers b^* to b.

Matching is Stable.

Lemma: There is no rogue couple for the matching formed by traditional marriage algorithm.

Proof:
Assume there is a rogue couple; (b, g^*)



b prefers g^* to g.

g^* prefers b to b^*.

Job b proposes to g^* before proposing to g.

So g^* rejected b (since he moved on)

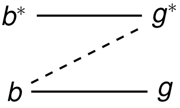
By improvement lemma, g^* prefers b^* to b.

Contradiction!

Matching is Stable.

Lemma: There is no rogue couple for the matching formed by traditional marriage algorithm.

Proof:
Assume there is a rogue couple; (b, g^*)



b prefers g^* to g.

g^* prefers b to b^*.

Job b proposes to g^* before proposing to g.

So g^* rejected b (since he moved on)

By improvement lemma, g^* prefers b^* to b.

Contradiction! $\square$

Question: The SMA produces a stable pairing is a proof by?

(A) Contradiction.

(B) Uses the improvement lemma.

(C) Induction.

(D) Direct.

Question: The SMA produces a stable pairing is a proof by?

(A) Contradiction.

(B) Uses the improvement lemma.

(C) Induction.

(D) Direct.

(A), (B), (C). (Maybe (D) internally. Semantics.)

Good for jobs? candidates?

Is the Job-Proposes better for jobs?

Good for jobs? candidates?

Is the Job-Proposes better for jobs? for candidates?

Good for jobs? candidates?

Is the Job-Proposes better for jobs? for candidates?

Definition: A **matching is x-optimal** if $x's$ partner
is its best partner in any stable pairing.

Good for jobs? candidates?

Is the Job-Proposes better for jobs? for candidates?

Definition: A **matching is** x**-optimal** if $x's$ partner
is its best partner in any stable pairing.

Definition: A **matching is** x**-pessimal** if $x's$ partner
is its worst partner in any stable pairing.

Good for jobs? candidates?

Is the Job-Proposes better for jobs? for candidates?

Definition: A **matching is** x**-optimal** if $x's$ partner
is its best partner in any stable pairing.

Definition: A **matching is** x**-pessimal** if $x's$ partner
is its worst partner in any stable pairing.

Definition: A **matching is job optimal** if it is x-optimal for **all** jobs x.

Good for jobs? candidates?

Is the Job-Proposes better for jobs? for candidates?

Definition: A **matching is** x**-optimal** if $x's$ partner
is its best partner in any stable pairing.

Definition: A **matching is** x**-pessimal** if $x's$ partner
is its worst partner in any stable pairing.

Definition: A **matching is job optimal** if it is x-optimal for **all** jobs x.

..and so on for job pessimal, candidate optimal, candidate pessimal.

Good for jobs? candidates?

Is the Job-Proposes better for jobs? for candidates?

Definition: A **matching is** x**-optimal** if $x's$ partner
is its best partner in any stable pairing.

Definition: A **matching is** x**-pessimal** if $x's$ partner
is its worst partner in any stable pairing.

Definition: A **matching is job optimal** if it is x-optimal for **all** jobs x.

..and so on for job pessimal, candidate optimal, candidate pessimal.

Claim: The optimal partner for a job must be first in its preference list.

Good for jobs? candidates?

Is the Job-Proposes better for jobs? for candidates?

Definition: A **matching is** x**-optimal** if $x's$ partner
is its best partner in any stable pairing.

Definition: A **matching is** x**-pessimal** if $x's$ partner
is its worst partner in any stable pairing.

Definition: A **matching is job optimal** if it is x-optimal for **all** jobs x.

..and so on for job pessimal, candidate optimal, candidate pessimal.

Claim: The optimal partner for a job must be first in its preference list.

True?

Good for jobs? candidates?

Is the Job-Proposes better for jobs? for candidates?

Definition: A **matching is x-optimal** if $x's$ partner
is its best partner in any stable pairing.

Definition: A **matching is x-pessimal** if $x's$ partner
is its worst partner in any stable pairing.

Definition: A **matching is job optimal** if it is x-optimal for **all** jobs x.

..and so on for job pessimal, candidate optimal, candidate pessimal.

Claim: The optimal partner for a job must be first in its preference list.

True? False?

Good for jobs? candidates?

Is the Job-Proposes better for jobs? for candidates?

Definition: A **matching is x-optimal** if $x's$ partner
is its best partner in any stable pairing.

Definition: A **matching is x-pessimal** if $x's$ partner
is its worst partner in any stable pairing.

Definition: A **matching is job optimal** if it is x-optimal for **all** jobs x.

..and so on for job pessimal, candidate optimal, candidate pessimal.

Claim: The optimal partner for a job must be first in its preference list.

True? False? False!

Good for jobs? candidates?

Is the Job-Proposes better for jobs? for candidates?

Definition: A **matching is *x*-optimal** if $x's$ partner
is its best partner in any stable pairing.

Definition: A **matching is *x*-pessimal** if $x's$ partner
is its worst partner in any stable pairing.

Definition: A **matching is job optimal** if it is x-optimal for **all** jobs x.

..and so on for job pessimal, candidate optimal, candidate pessimal.

Claim: The optimal partner for a job must be first in its preference list.

True? False? False!

Subtlety here: Best partner in any stable matching.

Good for jobs? candidates?

Is the Job-Proposes better for jobs? for candidates?

Definition: A **matching is** x**-optimal** if $x's$ partner
is its best partner in any stable pairing.

Definition: A **matching is** x**-pessimal** if $x's$ partner
is its worst partner in any stable pairing.

Definition: A **matching is job optimal** if it is x-optimal for **all** jobs x.

..and so on for job pessimal, candidate optimal, candidate pessimal.

Claim: The optimal partner for a job must be first in its preference list.

True? False? False!

Subtlety here: Best partner in any stable matching.
As well as you can be in a globally stable solution!

Good for jobs? candidates?

Is the Job-Proposes better for jobs? for candidates?

Definition: A **matching is x-optimal** if $x's$ partner
is its best partner in any stable pairing.

Definition: A **matching is x-pessimal** if $x's$ partner
is its worst partner in any stable pairing.

Definition: A **matching is job optimal** if it is x-optimal for **all** jobs x.

..and so on for job pessimal, candidate optimal, candidate pessimal.

Claim: The optimal partner for a job must be first in its preference list.

True? False? False!

Subtlety here: Best partner in any stable matching.
As well as you can be in a globally stable solution!

Question: Is there a job or candidate optimal matching?

Good for jobs? candidates?

Is the Job-Proposes better for jobs? for candidates?

Definition: A **matching is x-optimal** if $x's$ partner
is its best partner in any stable pairing.

Definition: A **matching is x-pessimal** if $x's$ partner
is its worst partner in any stable pairing.

Definition: A **matching is job optimal** if it is x-optimal for **all** jobs x.

..and so on for job pessimal, candidate optimal, candidate pessimal.

Claim: The optimal partner for a job must be first in its preference list.

True? False? False!

Subtlety here: Best partner in any stable matching.
As well as you can be in a globally stable solution!

Question: Is there a job or candidate optimal matching?
Is it possible:

Good for jobs? candidates?

Is the Job-Proposes better for jobs? for candidates?

Definition: A **matching is x-optimal** if $x's$ partner
is its best partner in any stable pairing.

Definition: A **matching is x-pessimal** if $x's$ partner
is its worst partner in any stable pairing.

Definition: A **matching is job optimal** if it is x-optimal for **all** jobs x.

..and so on for job pessimal, candidate optimal, candidate pessimal.

Claim: The optimal partner for a job must be first in its preference list.

True? False? False!

Subtlety here: Best partner in any stable matching.
As well as you can be in a globally stable solution!

Question: Is there a job or candidate optimal matching?
Is it possible:
b-optimal pairing different from the b'-optimal matching!

Good for jobs? candidates?

Is the Job-Proposes better for jobs? for candidates?

Definition: A **matching is** x**-optimal** if $x's$ partner
is its best partner in any stable pairing.

Definition: A **matching is** x**-pessimal** if $x's$ partner
is its worst partner in any stable pairing.

Definition: A **matching is job optimal** if it is x-optimal for **all** jobs x.

..and so on for job pessimal, candidate optimal, candidate pessimal.

Claim: The optimal partner for a job must be first in its preference list.

True? False? False!

Subtlety here: Best partner in any stable matching.
As well as you can be in a globally stable solution!

Question: Is there a job or candidate optimal matching?
Is it possible:
b-optimal pairing different from the b'-optimal matching!
Yes?

Good for jobs? candidates?

Is the Job-Proposes better for jobs? for candidates?

Definition: A **matching is** x**-optimal** if $x's$ partner
is its best partner in any stable pairing.

Definition: A **matching is** x**-pessimal** if $x's$ partner
is its worst partner in any stable pairing.

Definition: A **matching is job optimal** if it is x-optimal for **all** jobs x.

..and so on for job pessimal, candidate optimal, candidate pessimal.

Claim: The optimal partner for a job must be first in its preference list.

True? False? False!

Subtlety here: Best partner in any stable matching.
As well as you can be in a globally stable solution!

Question: Is there a job or candidate optimal matching?
Is it possible:
b-optimal pairing different from the b'-optimal matching!
Yes? No?

Good for jobs? candidates?

Is the Job-Proposes better for jobs? for candidates?

Definition: A **matching is x-optimal** if $x's$ partner
is its best partner in any stable pairing.

Definition: A **matching is x-pessimal** if $x's$ partner
is its worst partner in any stable pairing.

Definition: A **matching is job optimal** if it is x-optimal for **all** jobs x.

..and so on for job pessimal, candidate optimal, candidate pessimal.

Claim: The optimal partner for a job must be first in its preference list.

True? False? False!

Subtlety here: Best partner in any stable matching.
As well as you can be in a globally stable solution!

Question: Is there a job or candidate optimal matching?
Is it possible:
b-optimal pairing different from the b'-optimal matching!
Yes? No?

Understanding Optimality: by example.

A:	1,2	1:	A,B
B:	1,2	2:	B,A

Understanding Optimality: by example.

A:	1,2	1:	A,B
B:	1,2	2:	B,A

Consider pairing: $(A,1),(B,2)$.

Understanding Optimality: by example.

A:	1,2	1:	A,B
B:	1,2	2:	B,A

Consider pairing: $(A,1),(B,2)$.

Stable?

Understanding Optimality: by example.

A:	1,2	1:	A,B
B:	1,2	2:	B,A

Consider pairing: $(A,1),(B,2)$.

Stable? Yes.

Understanding Optimality: by example.

A:	1,2	1:	A,B
B:	1,2	2:	B,A

Consider pairing: $(A,1),(B,2)$.

Stable? Yes.

Optimal for B?

Understanding Optimality: by example.

A:	1,2	1:	A,B
B:	1,2	2:	B,A

Consider pairing: $(A,1),(B,2)$.

Stable? Yes.

Optimal for B?

Notice: only one stable pairing.

Understanding Optimality: by example.

A:	1,2	1:	A,B
B:	1,2	2:	B,A

Consider pairing: $(A,1),(B,2)$.

Stable? Yes.

Optimal for B?
Notice: only one stable pairing.
So this is the best B can do in a stable pairing.

Understanding Optimality: by example.

A:	1,2	1:	A,B
B:	1,2	2:	B,A

Consider pairing: $(A,1),(B,2)$.

Stable? Yes.

Optimal for *B*?
Notice: only one stable pairing.
So this is the best *B* can do in a stable pairing.
So optimal for *B*.

Understanding Optimality: by example.

A:	1,2	1:	A,B
B:	1,2	2:	B,A

Consider pairing: $(A,1),(B,2)$.

Stable? Yes.

Optimal for *B*?
Notice: only one stable pairing.
So this is the best *B* can do in a stable pairing.
So optimal for *B*.

Also optimal for *A*, 1 and 2.

Understanding Optimality: by example.

A:	1,2	1:	A,B
B:	1,2	2:	B,A

Consider pairing: $(A,1),(B,2)$.

Stable? Yes.

Optimal for B?
Notice: only one stable pairing.
So this is the best B can do in a stable pairing.
So optimal for B.

Also optimal for A, 1 and 2. Also pessimal for A,B,1 and 2.

Understanding Optimality: by example.

A:	1,2	1:	A,B
B:	1,2	2:	B,A

Consider pairing: $(A,1),(B,2)$.

Stable? Yes.

Optimal for *B*?
Notice: only one stable pairing.
So this is the best *B* can do in a stable pairing.
So optimal for *B*.

Also optimal for *A*, 1 and 2. Also pessimal for *A*,*B*,1 and 2.

A:	1,2	1:	B,A
B:	2,1	2:	A,B

Understanding Optimality: by example.

A:	1,2	1:	A,B
B:	1,2	2:	B,A

Consider pairing: $(A,1),(B,2)$.

Stable? Yes.

Optimal for B?
Notice: only one stable pairing.
So this is the best B can do in a stable pairing.
So optimal for B.

Also optimal for A, 1 and 2. Also pessimal for A,B,1 and 2.

A:	1,2	1:	B,A
B:	2,1	2:	A,B

Pairing S: $(A,1),(B,2)$.

Understanding Optimality: by example.

A:	1,2	1:	A,B
B:	1,2	2:	B,A

Consider pairing: $(A,1),(B,2)$.

Stable? Yes.

Optimal for B?
Notice: only one stable pairing.
So this is the best B can do in a stable pairing.
So optimal for B.

Also optimal for A, 1 and 2. Also pessimal for A,B,1 and 2.

A:	1,2	1:	B,A
B:	2,1	2:	A,B

Pairing S: $(A,1),(B,2)$. Stable?

Understanding Optimality: by example.

A:	1,2	1:	A,B
B:	1,2	2:	B,A

Consider pairing: $(A,1),(B,2)$.

Stable? Yes.

Optimal for *B*?
Notice: only one stable pairing.
So this is the best *B* can do in a stable pairing.
So optimal for *B*.

Also optimal for *A*, 1 and 2. Also pessimal for *A*,*B*,1 and 2.

A:	1,2	1:	B,A
B:	2,1	2:	A,B

Pairing *S*: $(A,1),(B,2)$.   Stable? Yes.

Understanding Optimality: by example.

A:	1,2	1:	A,B
B:	1,2	2:	B,A

Consider pairing: $(A,1),(B,2)$.

Stable? Yes.

Optimal for B?
Notice: only one stable pairing.
So this is the best B can do in a stable pairing.
So optimal for B.

Also optimal for A, 1 and 2. Also pessimal for A,B,1 and 2.

A:	1,2	1:	B,A
B:	2,1	2:	A,B

Pairing S: $(A,1),(B,2)$. Stable? Yes.

Understanding Optimality: by example.

A:	1,2	1:	A,B
B:	1,2	2:	B,A

Consider pairing: $(A,1),(B,2)$.

Stable? Yes.

Optimal for B?
Notice: only one stable pairing.
So this is the best B can do in a stable pairing.
So optimal for B.

Also optimal for A, 1 and 2. Also pessimal for A,B,1 and 2.

A:	1,2	1:	B,A
B:	2,1	2:	A,B

Pairing S: $(A,1),(B,2)$.   Stable? Yes.

Pairing T: $(A,2),(B,1)$.

Understanding Optimality: by example.

A:	1,2	1:	A,B
B:	1,2	2:	B,A

Consider pairing: $(A,1),(B,2)$.

Stable? Yes.

Optimal for B?
Notice: only one stable pairing.
So this is the best B can do in a stable pairing.
So optimal for B.

Also optimal for A, 1 and 2. Also pessimal for A,B,1 and 2.

A:	1,2	1:	B,A
B:	2,1	2:	A,B

Pairing S: $(A,1),(B,2)$. Stable? Yes.

Pairing T: $(A,2),(B,1)$. Also Stable.

Understanding Optimality: by example.

A:	1,2	1:	A,B
B:	1,2	2:	B,A

Consider pairing: $(A,1),(B,2)$.

Stable? Yes.

Optimal for B?
Notice: only one stable pairing.
So this is the best B can do in a stable pairing.
So optimal for B.

Also optimal for A, 1 and 2. Also pessimal for A,B,1 and 2.

A:	1,2	1:	B,A
B:	2,1	2:	A,B

Pairing S: $(A,1),(B,2)$. Stable? Yes.

Pairing T: $(A,2),(B,1)$. Also Stable.

Which is optimal for A?

Understanding Optimality: by example.

A:	1,2	1:	A,B
B:	1,2	2:	B,A

Consider pairing: $(A,1),(B,2)$.

Stable? Yes.

Optimal for B?
Notice: only one stable pairing.
So this is the best B can do in a stable pairing.
So optimal for B.

Also optimal for A, 1 and 2. Also pessimal for A,B,1 and 2.

A:	1,2	1:	B,A
B:	2,1	2:	A,B

Pairing S: $(A,1),(B,2)$. Stable? Yes.

Pairing T: $(A,2),(B,1)$. Also Stable.

Which is optimal for A? S

Understanding Optimality: by example.

A:	1,2	1:	A,B
B:	1,2	2:	B,A

Consider pairing: $(A,1),(B,2)$.

Stable? Yes.

Optimal for *B*?
Notice: only one stable pairing.
So this is the best *B* can do in a stable pairing.
So optimal for *B*.

Also optimal for *A*, 1 and 2. Also pessimal for *A*,*B*,1 and 2.

A:	1,2	1:	B,A
B:	2,1	2:	A,B

Pairing *S*: $(A,1),(B,2)$. Stable? Yes.

Pairing *T*: $(A,2),(B,1)$. Also Stable.

Which is optimal for *A*? *S* Which is optimal for *B*?

Understanding Optimality: by example.

A:	1,2	1:	A,B
B:	1,2	2:	B,A

Consider pairing: $(A,1),(B,2)$.

Stable? Yes.

Optimal for B?
Notice: only one stable pairing.
So this is the best B can do in a stable pairing.
So optimal for B.

Also optimal for A, 1 and 2. Also pessimal for A,B,1 and 2.

A:	1,2	1:	B,A
B:	2,1	2:	A,B

Pairing S: $(A,1),(B,2)$. Stable? Yes.

Pairing T: $(A,2),(B,1)$. Also Stable.

Which is optimal for A? S Which is optimal for B? S

Understanding Optimality: by example.

A:	1,2	1:	A,B
B:	1,2	2:	B,A

Consider pairing: $(A,1),(B,2)$.

Stable? Yes.

Optimal for B?
Notice: only one stable pairing.
So this is the best B can do in a stable pairing.
So optimal for B.

Also optimal for A, 1 and 2. Also pessimal for A,B,1 and 2.

A:	1,2	1:	B,A
B:	2,1	2:	A,B

Pairing S: $(A,1),(B,2)$. Stable? Yes.

Pairing T: $(A,2),(B,1)$. Also Stable.

Which is optimal for A? S Which is optimal for B? S
Which is optimal for 1?

Understanding Optimality: by example.

A:	1,2	1:	A,B
B:	1,2	2:	B,A

Consider pairing: $(A,1),(B,2)$.

Stable? Yes.

Optimal for B?
Notice: only one stable pairing.
So this is the best B can do in a stable pairing.
So optimal for B.

Also optimal for A, 1 and 2. Also pessimal for A,B,1 and 2.

A:	1,2	1:	B,A
B:	2,1	2:	A,B

Pairing S: $(A,1),(B,2)$. Stable? Yes.

Pairing T: $(A,2),(B,1)$. Also Stable.

Which is optimal for A? S Which is optimal for B? S
Which is optimal for 1? T

Understanding Optimality: by example.

A:	1,2	1:	A,B
B:	1,2	2:	B,A

Consider pairing: $(A,1),(B,2)$.

Stable? Yes.

Optimal for B?
Notice: only one stable pairing.
So this is the best B can do in a stable pairing.
So optimal for B.

Also optimal for A, 1 and 2. Also pessimal for A,B,1 and 2.

A:	1,2	1:	B,A
B:	2,1	2:	A,B

Pairing S: $(A,1),(B,2)$. Stable? Yes.

Pairing T: $(A,2),(B,1)$. Also Stable.

Which is optimal for A? S Which is optimal for B? S
Which is optimal for 1? T Which is optimal for 2?

Understanding Optimality: by example.

A:	1,2	1:	A,B
B:	1,2	2:	B,A

Consider pairing: $(A,1),(B,2)$.

Stable? Yes.

Optimal for B?
Notice: only one stable pairing.
So this is the best B can do in a stable pairing.
So optimal for B.

Also optimal for A, 1 and 2. Also pessimal for A,B,1 and 2.

A:	1,2	1:	B,A
B:	2,1	2:	A,B

Pairing S: $(A,1),(B,2)$. Stable? Yes.

Pairing T: $(A,2),(B,1)$. Also Stable.

Which is optimal for A? S Which is optimal for B? S
Which is optimal for 1? T Which is optimal for 2? T

Understanding Optimality: by example.

A:	1,2	1:	A,B
B:	1,2	2:	B,A

Consider pairing: $(A,1),(B,2)$.

Stable? Yes.

Optimal for B?
Notice: only one stable pairing.
So this is the best B can do in a stable pairing.
So optimal for B.

Also optimal for A, 1 and 2. Also pessimal for A,B,1 and 2.

A:	1,2	1:	B,A
B:	2,1	2:	A,B

Pairing S: $(A,1),(B,2)$. Stable? Yes.

Pairing T: $(A,2),(B,1)$. Also Stable.

Which is optimal for A? S Which is optimal for B? S
Which is optimal for 1? T Which is optimal for 2? T

Pessimality?

Job Propose and Candidate Reject is optimal!

For jobs?

Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

Theorem: Job Propose and Reject produces a job-optimal pairing.

Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

Theorem: Job Propose and Reject produces a job-optimal pairing.

Proof:

Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

Theorem: Job Propose and Reject produces a job-optimal pairing.

Proof:
Assume not:

Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

Theorem: Job Propose and Reject produces a job-optimal pairing.

Proof:
Assume not: there is a job b does not get optimal candidate, g.

Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

Theorem: Job Propose and Reject produces a job-optimal pairing.

Proof:
Assume not: there is a job b does not get optimal candidate, g.

There is a stable pairing S where b and g are paired.

Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

Theorem: Job Propose and Reject produces a job-optimal pairing.

Proof:
Assume not: there is a job b does not get optimal candidate, g.

There is a stable pairing S where b and g are paired.

Let t be first day job b gets rejected

Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

Theorem: Job Propose and Reject produces a job-optimal pairing.

Proof:
Assume not: there is a job b does not get optimal candidate, g.

There is a stable pairing S where b and g are paired.

Let t be first day job b gets rejected
by its optimal candidate g who it is paired with

Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

Theorem: Job Propose and Reject produces a job-optimal pairing.

Proof:
Assume not: there is a job b does not get optimal candidate, g.

There is a stable pairing S where b and g are paired.

Let t be first day job b gets rejected
by its optimal candidate g who it is paired with
in stable pairing S.

Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

Theorem: Job Propose and Reject produces a job-optimal pairing.

Proof:
Assume not: there is a job b does not get optimal candidate, g.

There is a stable pairing S where b and g are paired.

Let t be first day job b gets rejected
by its optimal candidate g who it is paired with
in stable pairing S.

b^* - knocks b off of g's string on day t

Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

Theorem: Job Propose and Reject produces a job-optimal pairing.

Proof:
Assume not: there is a job b does not get optimal candidate, g.

There is a stable pairing S where b and g are paired.

Let t be first day job b gets rejected
by its optimal candidate g who it is paired with
in stable pairing S.

b^* - knocks b off of g's string on day $t \implies g$ prefers b^* to b

Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

Theorem: Job Propose and Reject produces a job-optimal pairing.

Proof:
Assume not: there is a job b does not get optimal candidate, g.

There is a stable pairing S where b and g are paired.

Let t be first day job b gets rejected
by its optimal candidate g who it is paired with
in stable pairing S.

b^* - knocks b off of g's string on day $t \implies g$ prefers b^* to b

By choice of t, b^* likes g at least as much as optimal candidate.

Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

Theorem: Job Propose and Reject produces a job-optimal pairing.

Proof:
Assume not: there is a job b does not get optimal candidate, g.

There is a stable pairing S where b and g are paired.

Let t be first day job b gets rejected
by its optimal candidate g who it is paired with
in stable pairing S.

b^* - knocks b off of g's string on day $t \implies g$ prefers b^* to b

By choice of t, b^* likes g at least as much as optimal candidate.

$\implies b^*$ prefers g to its partner g^* in S.

Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

Theorem: Job Propose and Reject produces a job-optimal pairing.

Proof:
Assume not: there is a job b does not get optimal candidate, g.

There is a stable pairing S where b and g are paired.

Let t be first day job b gets rejected
by its optimal candidate g who it is paired with
in stable pairing S.

b^* - knocks b off of g's string on day t $\implies$ g prefers b^* to b

By choice of t, b^* likes g at least as much as optimal candidate.

$\implies$ b^* prefers g to its partner g^* in S.

Rogue couple for S.

Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

Theorem: Job Propose and Reject produces a job-optimal pairing.

Proof:
Assume not: there is a job b does not get optimal candidate, g.

There is a stable pairing S where b and g are paired.

Let t be first day job b gets rejected
by its optimal candidate g who it is paired with
in stable pairing S.

b^* - knocks b off of g's string on day $t \implies g$ prefers b^* to b

By choice of t, b^* likes g at least as much as optimal candidate.

$\implies b^*$ prefers g to its partner g^* in S.

Rogue couple for S.
So S is not a stable pairing.

Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

Theorem: Job Propose and Reject produces a job-optimal pairing.

Proof:
Assume not: there is a job b does not get optimal candidate, g.

There is a stable pairing S where b and g are paired.

Let t be first day job b gets rejected
by its optimal candidate g who it is paired with
in stable pairing S.

b^* - knocks b off of g's string on day $t \implies g$ prefers b^* to b

By choice of t, b^* likes g at least as much as optimal candidate.

$\implies b^*$ prefers g to its partner g^* in S.

Rogue couple for S.
So S is not a stable pairing. Contradiction.

Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

Theorem: Job Propose and Reject produces a job-optimal pairing.

Proof:
Assume not: there is a job b does not get optimal candidate, g.

There is a stable pairing S where b and g are paired.

Let t be first day job b gets rejected
by its optimal candidate g who it is paired with
in stable pairing S.

b^* - knocks b off of g's string on day $t \implies g$ prefers b^* to b

By choice of t, b^* likes g at least as much as optimal candidate.

$\implies b^*$ prefers g to its partner g^* in S.

Rogue couple for S.
So S is not a stable pairing. Contradiction. $\square$

Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

Theorem: Job Propose and Reject produces a job-optimal pairing.

Proof:
Assume not: there is a job b does not get optimal candidate, g.

There is a stable pairing S where b and g are paired.

Let t be first day job b gets rejected
by its optimal candidate g who it is paired with
in stable pairing S.

b^* - knocks b off of g's string on day t $\implies$ g prefers b^* to b

By choice of t, b^* likes g at least as much as optimal candidate.

$\implies$ b^* prefers g to its partner g^* in S.

Rogue couple for S.
So S is not a stable pairing. Contradiction. $\square$

Notes:

Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

Theorem: Job Propose and Reject produces a job-optimal pairing.

Proof:
Assume not: there is a job b does not get optimal candidate, g.

There is a stable pairing S where b and g are paired.

Let t be first day job b gets rejected
by its optimal candidate g who it is paired with
in stable pairing S.

b^* - knocks b off of g's string on day $t \implies g$ prefers b^* to b

By choice of t, b^* likes g at least as much as optimal candidate.

$\implies b^*$ prefers g to its partner g^* in S.

Rogue couple for S.
So S is not a stable pairing. Contradiction. $\square$

Notes: S - stable.

Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

Theorem: Job Propose and Reject produces a job-optimal pairing.

Proof:
Assume not: there is a job b does not get optimal candidate, g.

There is a stable pairing S where b and g are paired.

Let t be first day job b gets rejected
by its optimal candidate g who it is paired with
in stable pairing S.

b^* - knocks b off of g's string on day $t \implies g$ prefers b^* to b

By choice of t, b^* likes g at least as much as optimal candidate.

$\implies b^*$ prefers g to its partner g^* in S.

Rogue couple for S.
So S is not a stable pairing. Contradiction. $\square$

Notes: S - stable. $(b^*, g^*) \in S$.

Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

Theorem: Job Propose and Reject produces a job-optimal pairing.

Proof:
Assume not: there is a job b does not get optimal candidate, g.

There is a stable pairing S where b and g are paired.

Let t be first day job b gets rejected
by its optimal candidate g who it is paired with
in stable pairing S.

b^* - knocks b off of g's string on day $t \implies g$ prefers b^* to b

By choice of t, b^* likes g at least as much as optimal candidate.

$\implies b^*$ prefers g to its partner g^* in S.

Rogue couple for S.
So S is not a stable pairing. Contradiction. $\square$

Notes: S - stable. $(b^*, g^*) \in S$. But (b^*, g)

Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

Theorem: Job Propose and Reject produces a job-optimal pairing.

Proof:
Assume not: there is a job b does not get optimal candidate, g.

There is a stable pairing S where b and g are paired.

Let t be first day job b gets rejected
by its optimal candidate g who it is paired with
in stable pairing S.

b^* - knocks b off of g's string on day $t \implies g$ prefers b^* to b

By choice of t, b^* likes g at least as much as optimal candidate.

$\implies b^*$ prefers g to its partner g^* in S.

Rogue couple for S.
So S is not a stable pairing. Contradiction. $\square$

Notes: S - stable. $(b^*, g^*) \in S$. But (b^*, g) is rogue couple!

Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

Theorem: Job Propose and Reject produces a job-optimal pairing.

Proof:
Assume not: there is a job b does not get optimal candidate, g.

There is a stable pairing S where b and g are paired.

Let t be first day job b gets rejected
by its optimal candidate g who it is paired with
in stable pairing S.

b^* - knocks b off of g's string on day $t \implies g$ prefers b^* to b

By choice of t, b^* likes g at least as much as optimal candidate.

$\implies b^*$ prefers g to its partner g^* in S.

Rogue couple for S.
So S is not a stable pairing. Contradiction. $\square$

Notes: S - stable. $(b^*, g^*) \in S$. But (b^*, g) is rogue couple!

Used Well-Ordering principle...

Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

Theorem: Job Propose and Reject produces a job-optimal pairing.

Proof:
Assume not: there is a job b does not get optimal candidate, g.

There is a stable pairing S where b and g are paired.

Let t be first day job b gets rejected
by its optimal candidate g who it is paired with
in stable pairing S.

b^* - knocks b off of g's string on day t $\implies$ g prefers b^* to b

By choice of t, b^* likes g at least as much as optimal candidate.

$\implies$ b^* prefers g to its partner g^* in S.

Rogue couple for S.
So S is not a stable pairing. Contradiction. $\square$

Notes: S - stable. $(b^*, g^*) \in S$. But (b^*, g) is rogue couple!

Used Well-Ordering principle...Induction.

How about for candidates?

How about for candidates?

Theorem: Job Propose and Reject produces candidate-pessimal pairing.

How about for candidates?

Theorem: Job Propose and Reject produces candidate-pessimal pairing.

T – pairing produced by JPR.

How about for candidates?

Theorem: Job Propose and Reject produces candidate-pessimal pairing.

T – pairing produced by JPR.

S – worse stable pairing for candidate g.

How about for candidates?

Theorem: Job Propose and Reject produces candidate-pessimal pairing.

T – pairing produced by JPR.

S – worse stable pairing for candidate g.

In T, (g, b) is pair.

How about for candidates?

Theorem: Job Propose and Reject produces candidate-pessimal pairing.

T – pairing produced by JPR.

S – worse stable pairing for candidate g.

In T, (g, b) is pair.

In S, (g, b^*) is pair.

How about for candidates?

Theorem: Job Propose and Reject produces candidate-pessimal pairing.

T – pairing produced by JPR.

S – worse stable pairing for candidate g.

In T, (g, b) is pair.

In S, (g, b^*) is pair.

g prefers b to b^*.

How about for candidates?

Theorem: Job Propose and Reject produces candidate-pessimal pairing.

T – pairing produced by JPR.

S – worse stable pairing for candidate g.

In T, (g, b) is pair.

In S, (g, b^*) is pair.

g prefers b to b^*.

T is job optimal, so b prefers g to its partner in S.

How about for candidates?

Theorem: Job Propose and Reject produces candidate-pessimal pairing.

T – pairing produced by JPR.

S – worse stable pairing for candidate g.

In T, (g, b) is pair.

In S, (g, b^*) is pair.

g prefers b to b^*.

T is job optimal, so b prefers g to its partner in S.

(g, b) is Rogue couple for S

How about for candidates?

Theorem: Job Propose and Reject produces candidate-pessimal pairing.

T – pairing produced by JPR.

S – worse stable pairing for candidate g.

In T, (g, b) is pair.

In S, (g, b^*) is pair.

g prefers b to b^*.

T is job optimal, so b prefers g to its partner in S.

(g, b) is Rogue couple for S

S is not stable.

How about for candidates?

Theorem: Job Propose and Reject produces candidate-pessimal pairing.

T – pairing produced by JPR.

S – worse stable pairing for candidate g.

In T, (g, b) is pair.

In S, (g, b^*) is pair.

g prefers b to b^*.

T is job optimal, so b prefers g to its partner in S.

(g, b) is Rogue couple for S

S is not stable.

Contradiction.

How about for candidates?

Theorem: Job Propose and Reject produces candidate-pessimal pairing.

T – pairing produced by JPR.

S – worse stable pairing for candidate g.

In T, (g, b) is pair.

In S, (g, b^*) is pair.

g prefers b to b^*.

T is job optimal, so b prefers g to its partner in S.

(g, b) is Rogue couple for S

S is not stable.

Contradiction. $\square$

How about for candidates?

Theorem: Job Propose and Reject produces candidate-pessimal pairing.

T – pairing produced by JPR.

S – worse stable pairing for candidate g.

In T, (g, b) is pair.

In S, (g, b^*) is pair.

g prefers b to b^*.

T is job optimal, so b prefers g to its partner in S.

(g, b) is Rogue couple for S

S is not stable.

Contradiction. $\square$

Notes:

How about for candidates?

Theorem: Job Propose and Reject produces candidate-pessimal pairing.

T – pairing produced by JPR.

S – worse stable pairing for candidate g.

In T, (g, b) is pair.

In S, (g, b^*) is pair.

g prefers b to b^*.

T is job optimal, so b prefers g to its partner in S.

(g, b) is Rogue couple for S

S is not stable.

Contradiction. $\square$

Notes: Not really induction.

How about for candidates?

Theorem: Job Propose and Reject produces candidate-pessimal pairing.

T – pairing produced by JPR.

S – worse stable pairing for candidate g.

In T, (g, b) is pair.

In S, (g, b^*) is pair.

g prefers b to b^*.

T is job optimal, so b prefers g to its partner in S.

(g, b) is Rogue couple for S

S is not stable.

Contradiction. $\square$

Notes: Not really induction.
Structural statement: Job optimality

How about for candidates?

Theorem: Job Propose and Reject produces candidate-pessimal pairing.

T – pairing produced by JPR.

S – worse stable pairing for candidate g.

In T, (g, b) is pair.

In S, (g, b^*) is pair.

g prefers b to b^*.

T is job optimal, so b prefers g to its partner in S.

(g, b) is Rogue couple for S

S is not stable.

Contradiction. $\square$

Notes: Not really induction.
Structural statement: Job optimality $\implies$ Candidate pessimality.

Quick Questions.

How does one make it better for candidates?

Quick Questions.

How does one make it better for candidates?

Propose and Reject - stable matching algorithm. One side proposes.

Quick Questions.

How does one make it better for candidates?

Propose and Reject - stable matching algorithm. One side proposes.

Jobs Propose $\implies$ job optimal.

Quick Questions.

How does one make it better for candidates?

Propose and Reject - stable matching algorithm. One side proposes.

Jobs Propose $\implies$ job optimal.

Candidates propose.

Quick Questions.

How does one make it better for candidates?

Propose and Reject - stable matching algorithm. One side proposes.

Jobs Propose $\implies$ job optimal.

Candidates propose. $\implies$ optimal for candidates.

Residency Matching..

Residency Matching..

The method was used to match residents to hospitals.

Residency Matching..

The method was used to match residents to hospitals.

Hospital optimal....

Residency Matching..

The method was used to match residents to hospitals.

Hospital optimal....

..until 1990's...

Residency Matching..

The method was used to match residents to hospitals.

Hospital optimal....

..until 1990's...Resident optimal.

Residency Matching..

The method was used to match residents to hospitals.

Hospital optimal....

..until 1990's...Resident optimal.

Another variation: couples.

Residency Matching..

The method was used to match residents to hospitals.

Hospital optimal....

..until 1990's...Resident optimal.

Another variation: couples.

Takeaways.

Analysis of cool algorithm with interesting goal: stability.

Takeaways.

Analysis of cool algorithm with interesting goal: stability.

“Economic”: different utilities.

Takeaways.

Analysis of cool algorithm with interesting goal: stability.

“Economic”: different utilities.

Definition of optimality: best utility in stable world.

Takeaways.

Analysis of cool algorithm with interesting goal: stability.

“Economic”: different utilities.

Definition of optimality: best utility in stable world.

Action gives better results for individuals but gives instability.

Takeaways.

Analysis of cool algorithm with interesting goal: stability.

"Economic": different utilities.

Definition of optimality: best utility in stable world.

Action gives better results for individuals but gives instability.

Induction over steps of algorithm.

Takeaways.

Analysis of cool algorithm with interesting goal: stability.

“Economic”: different utilities.

Definition of optimality: best utility in stable world.

Action gives better results for individuals but gives instability.

Induction over steps of algorithm.

Proofs carefully use definition:

Takeaways.

Analysis of cool algorithm with interesting goal: stability.

"Economic": different utilities.

Definition of optimality: best utility in stable world.

Action gives better results for individuals but gives instability.

Induction over steps of algorithm.

Proofs carefully use definition:
- Stability:
  - Improvement Lemma: every day the job gets to choose.
- Optimality proof:
  - Job Optimality:
    - contradiction of the existence of a better *stable* pairing.
  - Candidate Pessimality:
    - contradiction plus cuz job optimality implies better pairing.

Takeaways.

Analysis of cool algorithm with interesting goal: stability.

“Economic”: different utilities.

Definition of optimality: best utility in stable world.

Action gives better results for individuals but gives instability.

Induction over steps of algorithm.

Proofs carefully use definition:

- Stability:
  - Improvement Lemma: every day the job gets to choose.
- Optimality proof:
  - Job Optimality:
    - contradiction of the existence of a better *stable* pairing.
  - Candidate Pessimality:
    - contradiction plus cuz job optimality implies better pairing.

Life Lesson: ask,

Takeaways.

Analysis of cool algorithm with interesting goal: stability.

"Economic": different utilities.

Definition of optimality: best utility in stable world.

Action gives better results for individuals but gives instability.

Induction over steps of algorithm.

Proofs carefully use definition:
- Stability:
  - Improvement Lemma: every day the job gets to choose.
- Optimality proof:
  - Job Optimality:
    - contradiction of the existence of a better *stable* pairing.
  - Candidate Pessimality:
    - contradiction plus cuz job optimality implies better pairing.

Life Lesson: ask, you will do better

Takeaways.

Analysis of cool algorithm with interesting goal: stability.

"Economic": different utilities.

Definition of optimality: best utility in stable world.

Action gives better results for individuals but gives instability.

Induction over steps of algorithm.

Proofs carefully use definition:
- Stability:
  - Improvement Lemma: every day the job gets to choose.
- Optimality proof:
  - Job Optimality:
    - contradiction of the existence of a better *stable* pairing.
  - Candidate Pessimality:
    - contradiction plus cuz job optimality implies better pairing.

Life Lesson: ask, you will do better even if rejection is hard.