Stable Matchings

CS/MCS 401: Computer Algorithms I
Lecture 2
Professor: Jonathan Liu

Sit next to someone! If you don't do it now, you'll have to move later.

Introduce yourself!
Icebreaker: If you were an ice cream flavor, which one would you be?

Lesson Goals

Today we'll take a closer look at one algorithm in particular.

We'll discuss how it is represented in pseudocode.

We'll also discuss what it means and looks like to make arguments about its correctness.

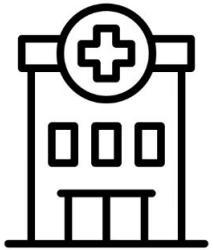
The Real-World Problem

National Resident Matching Program (NRMP)

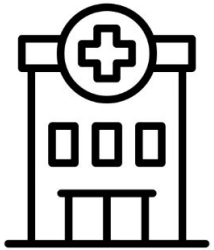
Newly-graduated doctors need to be matched to residency programs.

Doctor 0

Doctor 1



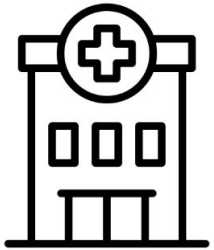
Hospital A



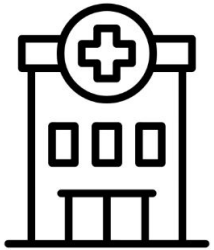
Hospital B

Doctor 2

Doctor 3

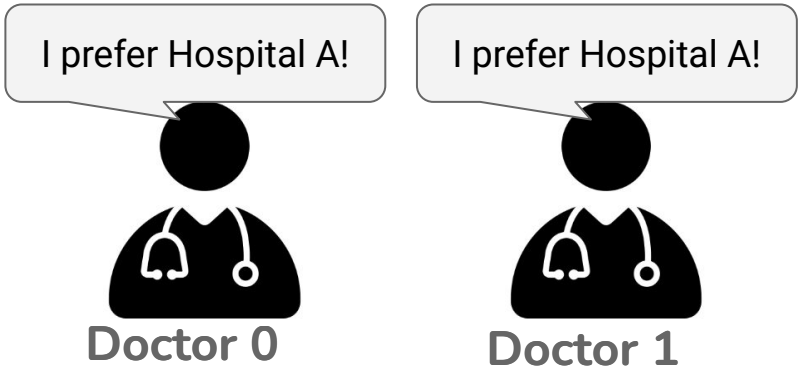


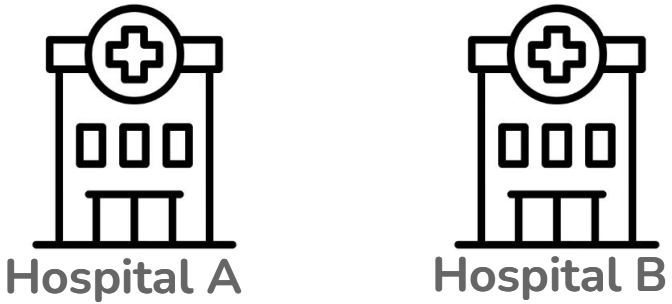
Hospital C



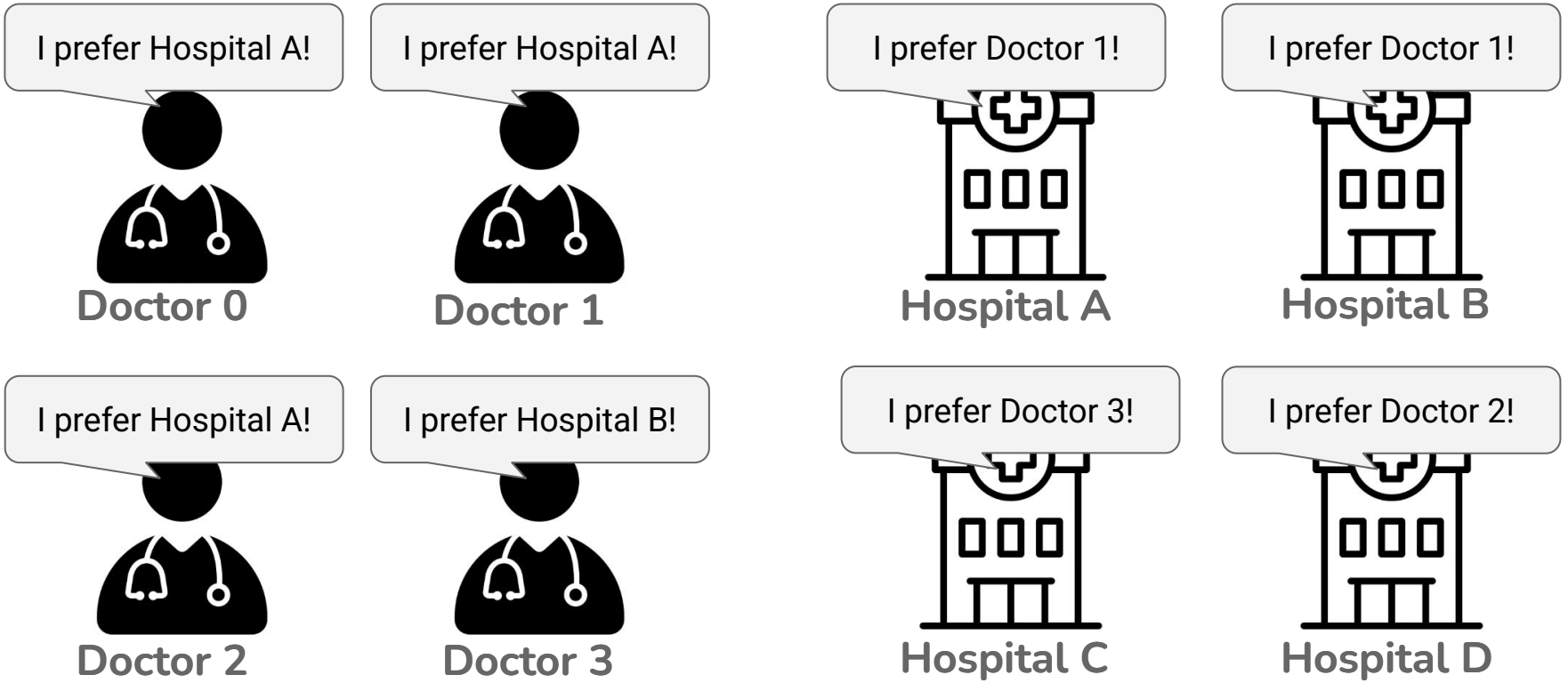
Hospital D

National Resident Matching Program (NRMP)





National Resident Matching Program (NRMP)



National Resident Matching Program (NRMP)

A > B > C > D

Doctor 0

A > C > B > D

Doctor 1

A > C > D > B

Doctor 2

B > A > C > D

Doctor 3

A **matching** is a set of (doctor, hospital) pairs where each entity is in exactly one pair.

The goal is, of course, to find a "good" matching.

1 > 3 > 0 > 2

Hospital A

1 > 2 > 0 > 3

Hospital B

3 > 1 > 2 > 0

Hospital C

2 > 3 > 0 > 1

Hospital D

National Resident Matching Program (NRMP)

A > B > C > D

Doctor 0

A > C > B > D

Doctor 1

A > C > D > B

Doctor 2

B > A > C > D

Doctor 3

Discuss with a partner/group:

What makes a matching "good"?

Name an objective measure of "goodness" for matchings.

Bonus: Can you find a matching that is "good" using your criterion?

1 > 3 > 0 > 2

Hospital A

1 > 2 > 0 > 3

Hospital B

3 > 1 > 2 > 0

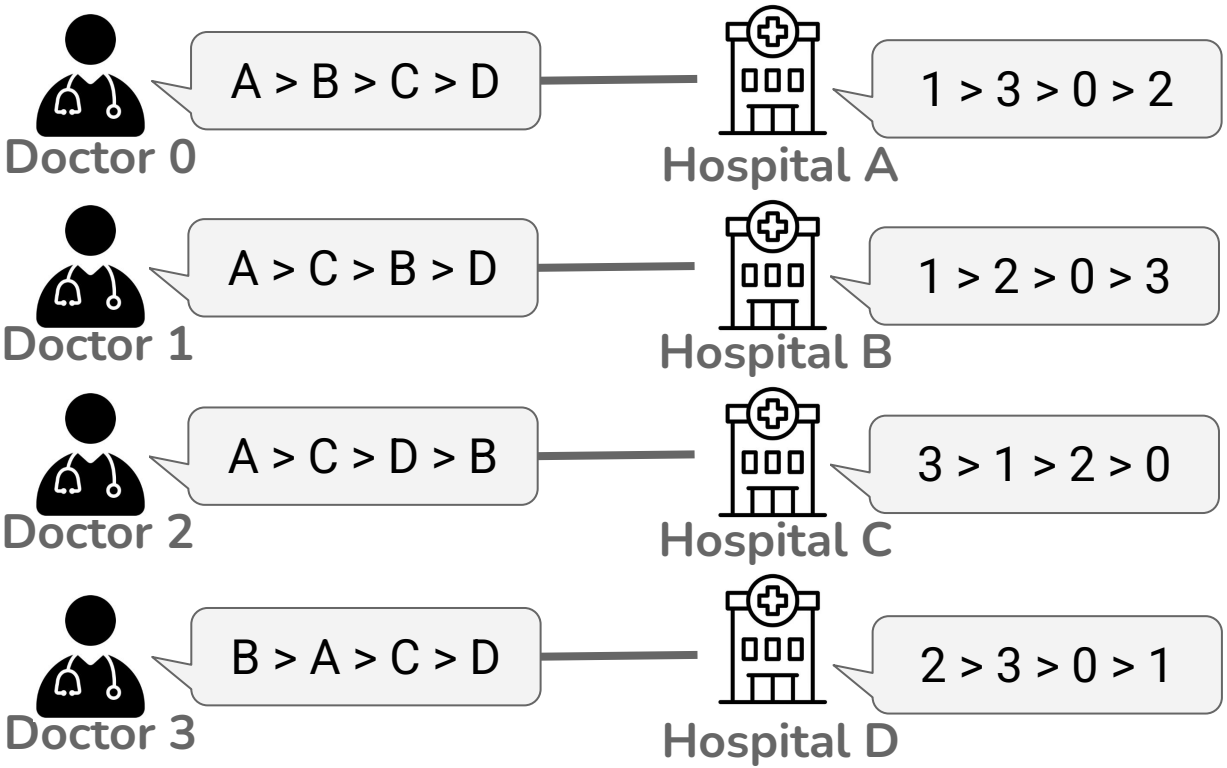
Hospital C

2 > 3 > 0 > 1

Hospital D

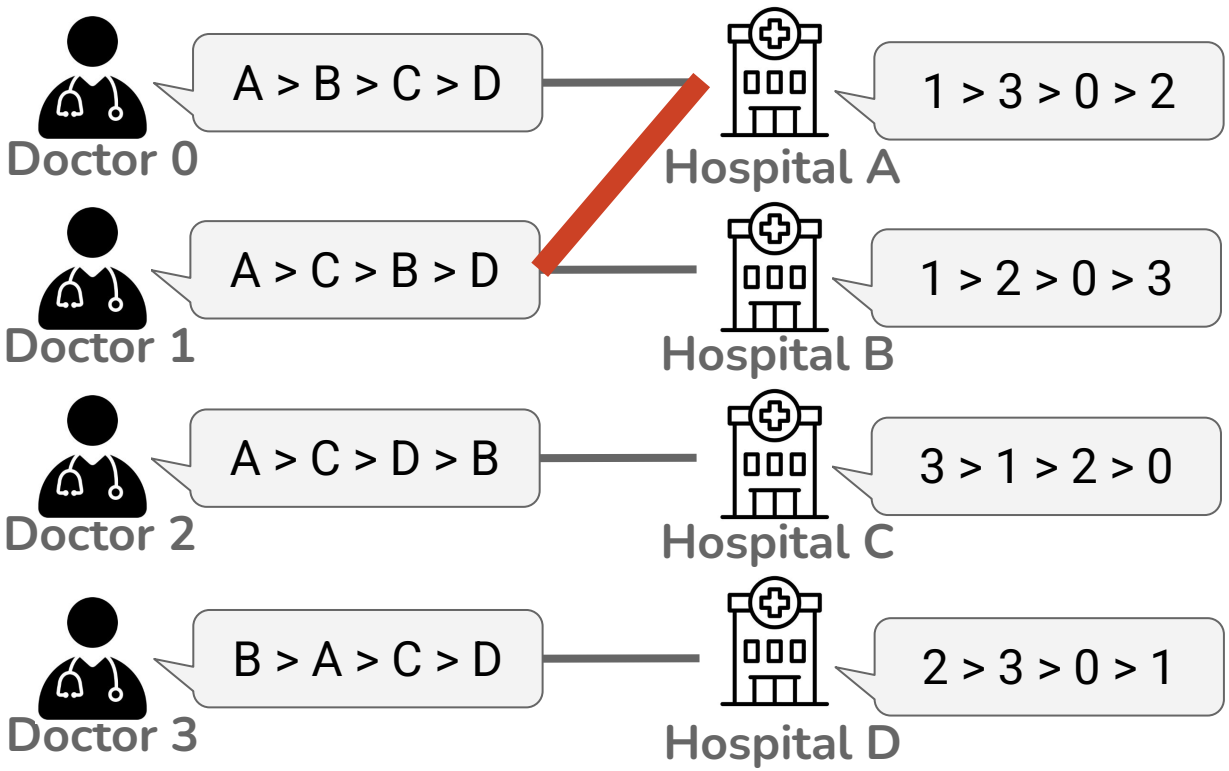
The Stable Matching Problem

Stable Matchings



A matching is **stable** if there is no pair that prefers each other over their current assignments.

Stable Matchings

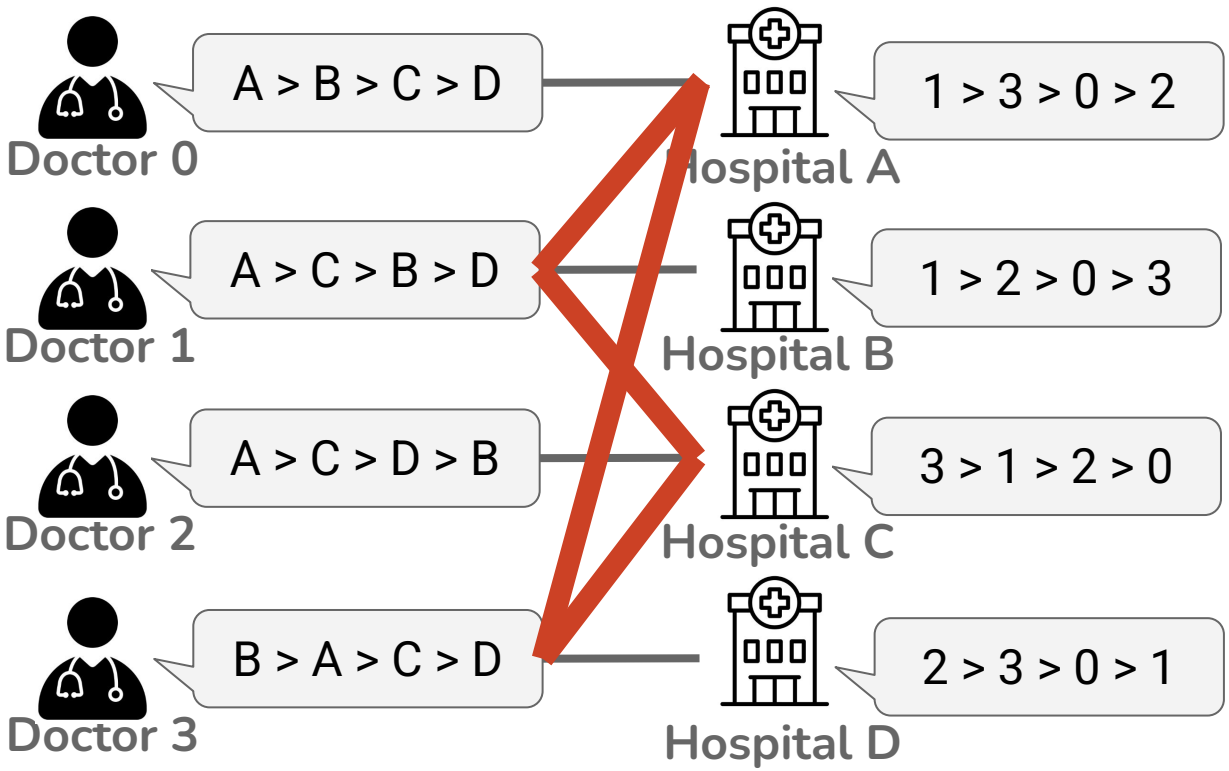


A matching is **stable** if there is no pair that prefers each other over their current assignments.

This matching is **not stable**! The pair *(Doctor 1, Hospital A)* is a **blocking pair**, meaning they both prefer each other over this matching.

Discuss: Any other blocking pairs?

Stable Matchings

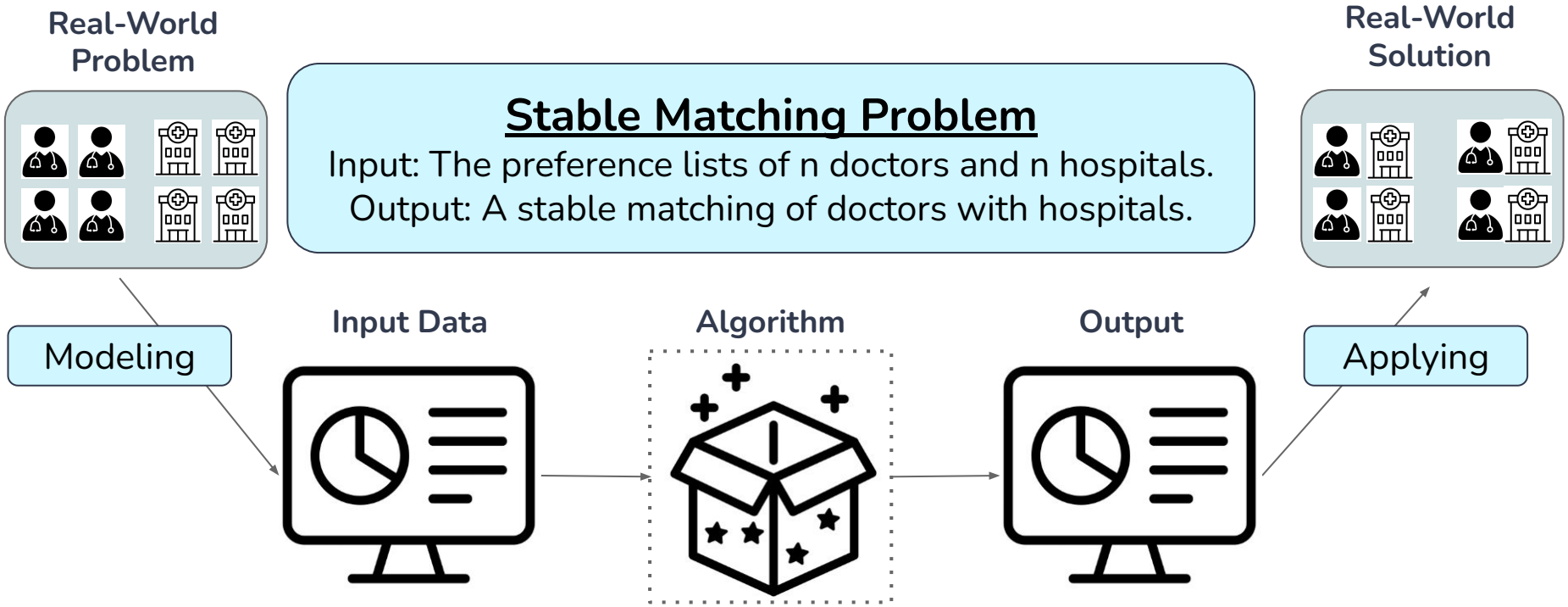


A matching is **stable** if there is no pair that prefers each other over their current assignments.

This matching is **not stable**! The pair *(Doctor 1, Hospital A)* is a **blocking pair**, meaning they both prefer each other over this matching.

Discuss: Any other blocking pairs?

The Big Picture: Stable Matching



Finding a Stable Matching

Gale-Shapley Algorithm

(David Gale and Lloyd Shapley, 1962)

```
matched_pairs = []

while there are unmatched hospitals:

    an unmatched hospital makes an
    offer to their favorite doctor
    (that they haven't requested
    before)

    if the doctor already has an offer
    they prefer: skip

    if the doctor has no prior offer:
    add this pair to matched_pairs

    if the doctor has a prior offer but
    prefers this one: replace their
    pairing in matched_pairs with this
    one

return matched_pairs
```

A Note on Pseudocode

This is an example of pseudocode.

What is like code:

- Control Flow (Loops, Returns, Ifs)
- Everything is in the order of execution

What isn't like code:

- Mostly natural language for each line

Overall Goal: Someone who doesn't understand the algorithm should be able to implement the code.

```
matched_pairs = []

while there are unmatched hospitals:

    an unmatched hospital makes an
    offer to their favorite doctor
    (that they haven't requested
    before)

    if the doctor already has an offer
    they prefer: skip

    if the doctor has no prior offer:
    add this pair to matched_pairs

    if the doctor has a prior offer
    but prefers this one: replace
    their pairing in matched_pairs
    with this one

return matched_pairs
```

A Run of Gale-Shapley


Doctor 0


Doctor 1


Doctor 2


Doctor 3


Hospital A


Hospital B

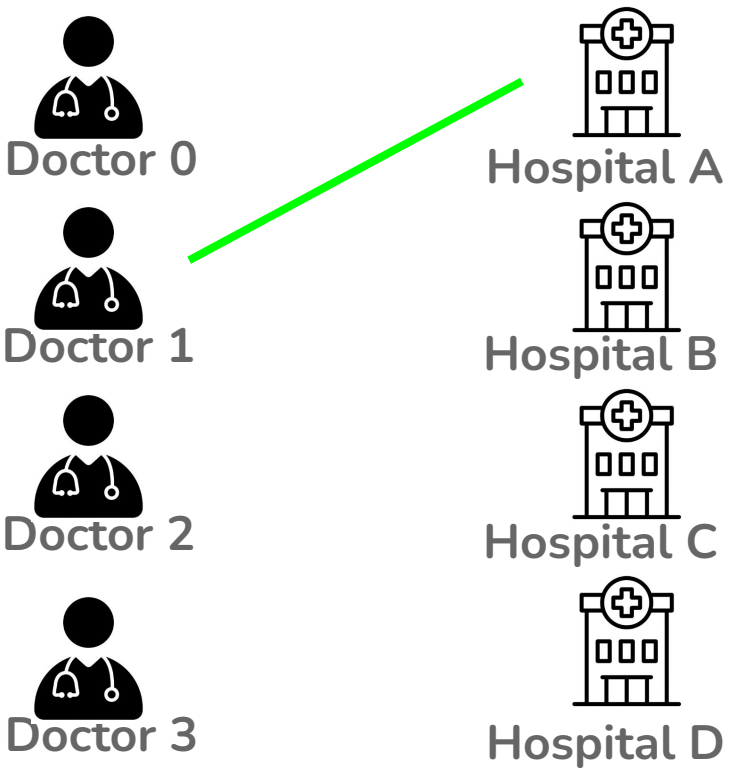

Hospital C


Hospital D

```
while there are unmatched hospitals:

    the unmatched hospital that is first alphabetically makes an offer to their favorite doctor (that they haven't requested before)

    if the doctor already has an offer they prefer: skip

    if the doctor has no prior offer: add this pair to matched_pairs

    if the doctor has a prior offer but prefers this one: replace their pairing in matched_pairs with this one

return matched_pairs
```

```
matched_pairs = []
```

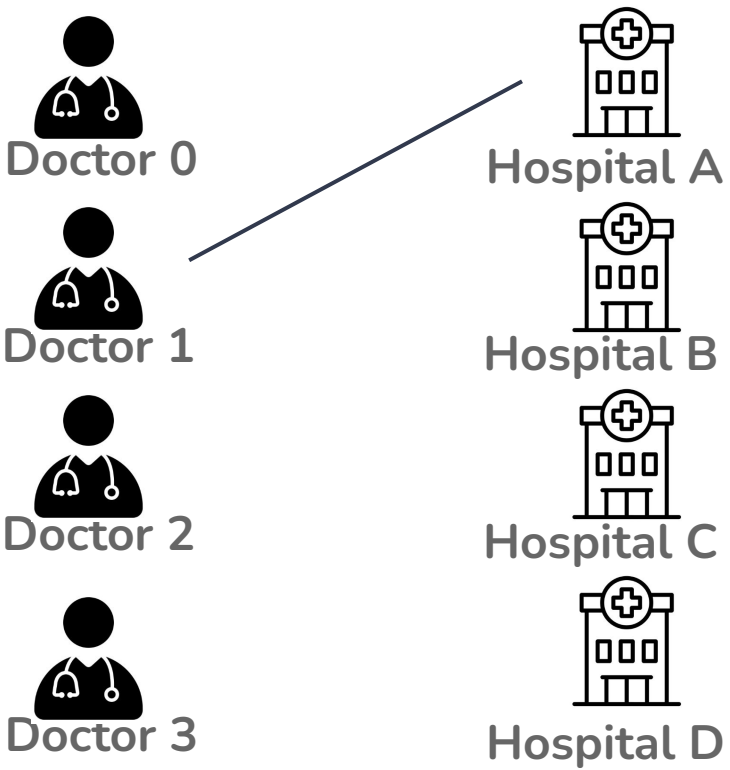
A Run of Gale-Shapley




```
while there are unmatched hospitals:

    the unmatched hospital that is first alphabetically makes an offer to their favorite doctor (that they haven't requested before)

    if the doctor already has an offer they prefer: skip

    if the doctor has no prior offer: add this pair to matched_pairs

    if the doctor has a prior offer but prefers this one: replace their pairing in matched_pairs with this one

return matched_pairs
```

```
matched_pairs = []
```

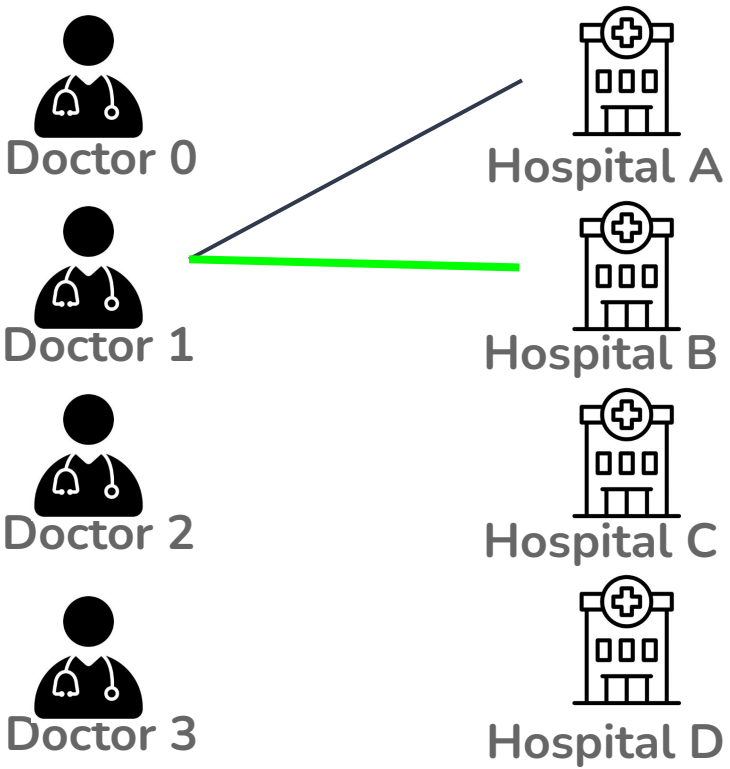
A Run of Gale-Shapley




```
while there are unmatched hospitals:

    the unmatched hospital that is first alphabetically makes an offer to their favorite doctor (that they haven't requested before)

    if the doctor already has an offer they prefer: skip

    if the doctor has no prior offer: add this pair to matched_pairs

    if the doctor has a prior offer but prefers this one: replace their pairing in matched_pairs with this one

return matched_pairs
```

```
matched_pairs = [(A,1)]
```

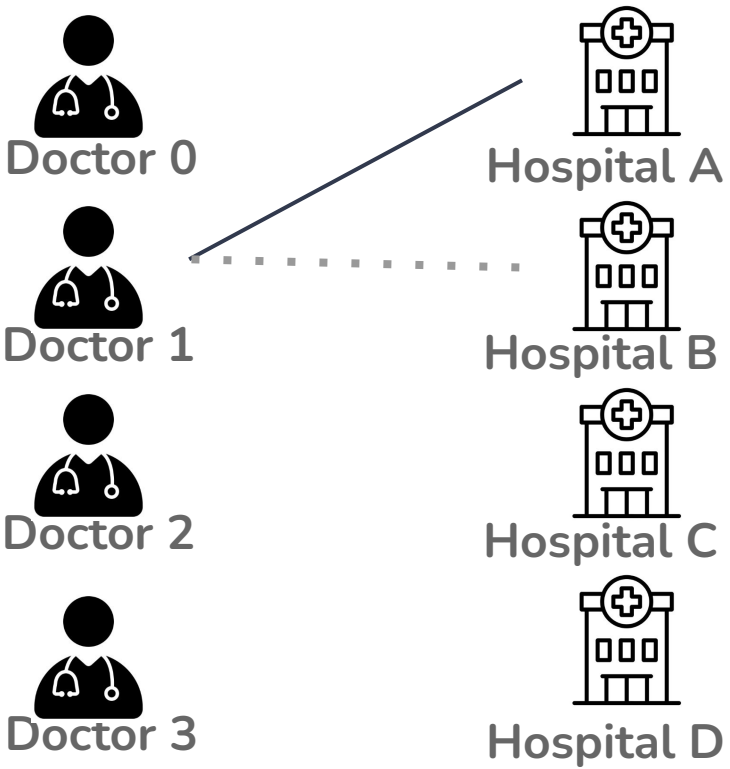
A Run of Gale-Shapley




```
while there are unmatched hospitals:

    the unmatched hospital that is first alphabetically makes an offer to their favorite doctor (that they haven't requested before)

    if the doctor already has an offer they prefer: skip

    if the doctor has no prior offer: add this pair to matched_pairs

    if the doctor has a prior offer but prefers this one: replace their pairing in matched_pairs with this one

return matched_pairs
```

```
matched_pairs = [(A,1)]
```

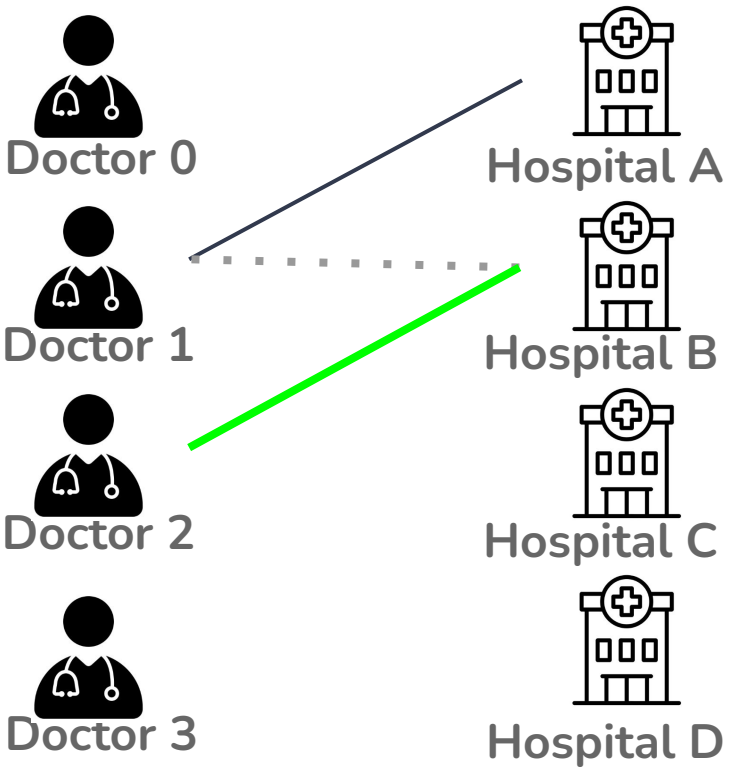
A Run of Gale-Shapley




```
while there are unmatched hospitals:

    the unmatched hospital that is first alphabetically makes an offer to their favorite doctor (that they haven't requested before)

    if the doctor already has an offer they prefer: skip

    if the doctor has no prior offer: add this pair to matched_pairs

    if the doctor has a prior offer but prefers this one: replace their pairing in matched_pairs with this one

return matched_pairs
```

```
matched_pairs = [(A,1)]
```

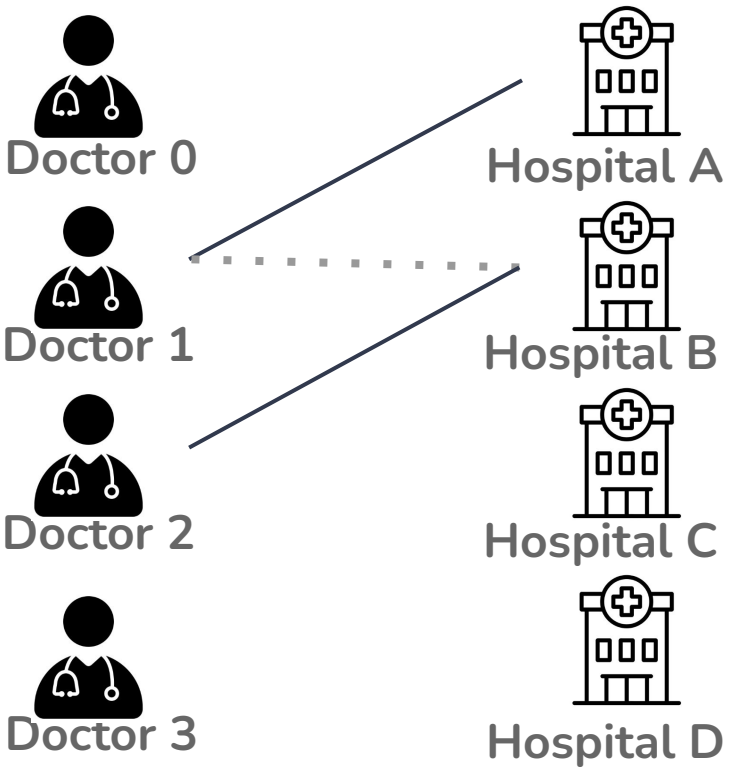
A Run of Gale-Shapley




```
while there are unmatched hospitals:

    the unmatched hospital that is first alphabetically makes an offer to their favorite doctor (that they haven't requested before)

    if the doctor already has an offer they prefer: skip

    if the doctor has no prior offer: add this pair to matched_pairs

    if the doctor has a prior offer but prefers this one: replace their pairing in matched_pairs with this one

return matched_pairs
```

```
matched_pairs = [(A,1)]
```

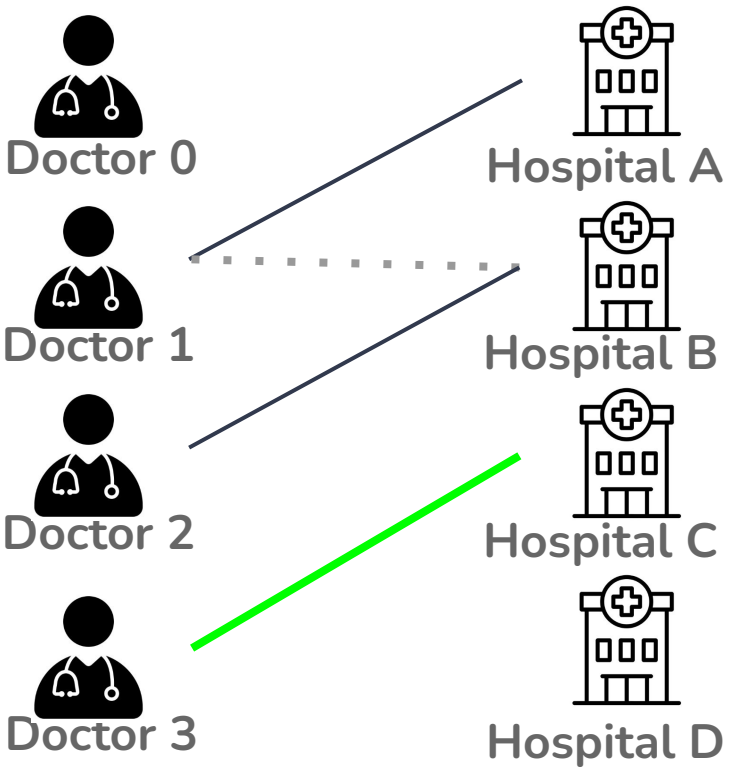
A Run of Gale-Shapley




```
while there are unmatched hospitals:

    the unmatched hospital that is first alphabetically makes an offer to their favorite doctor (that they haven't requested before)

    if the doctor already has an offer they prefer: skip

    if the doctor has no prior offer: add this pair to matched_pairs

    if the doctor has a prior offer but prefers this one: replace their pairing in matched_pairs with this one

return matched_pairs
```

```
matched_pairs = [(A,1), (B, 2)]
```

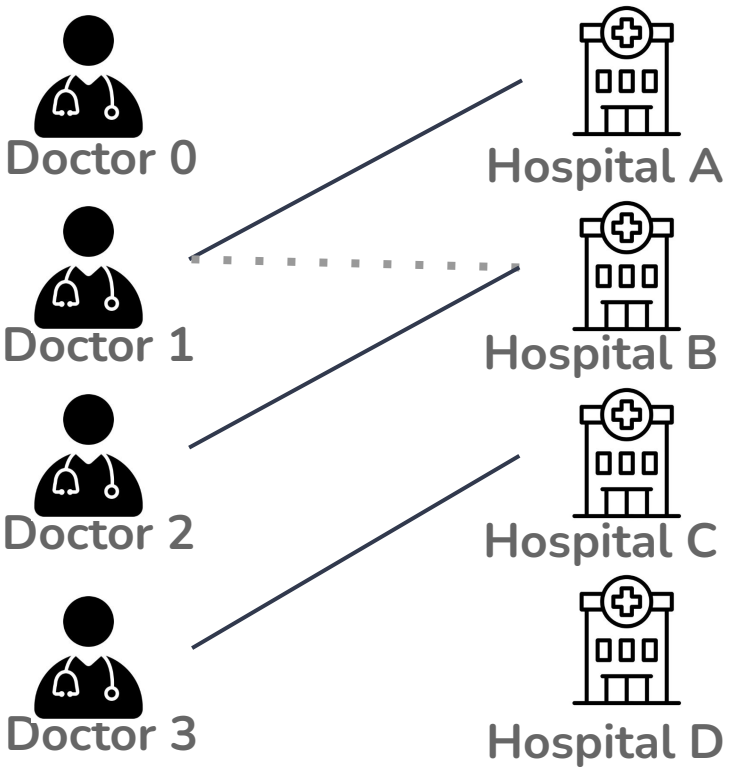
A Run of Gale-Shapley




```
while there are unmatched hospitals:

    the unmatched hospital that is first alphabetically makes an offer to their favorite doctor (that they haven't requested before)

    if the doctor already has an offer they prefer: skip

    if the doctor has no prior offer: add this pair to matched_pairs

    if the doctor has a prior offer but prefers this one: replace their pairing in matched_pairs with this one

return matched_pairs
```

```
matched_pairs = [(A,1), (B, 2)]
```

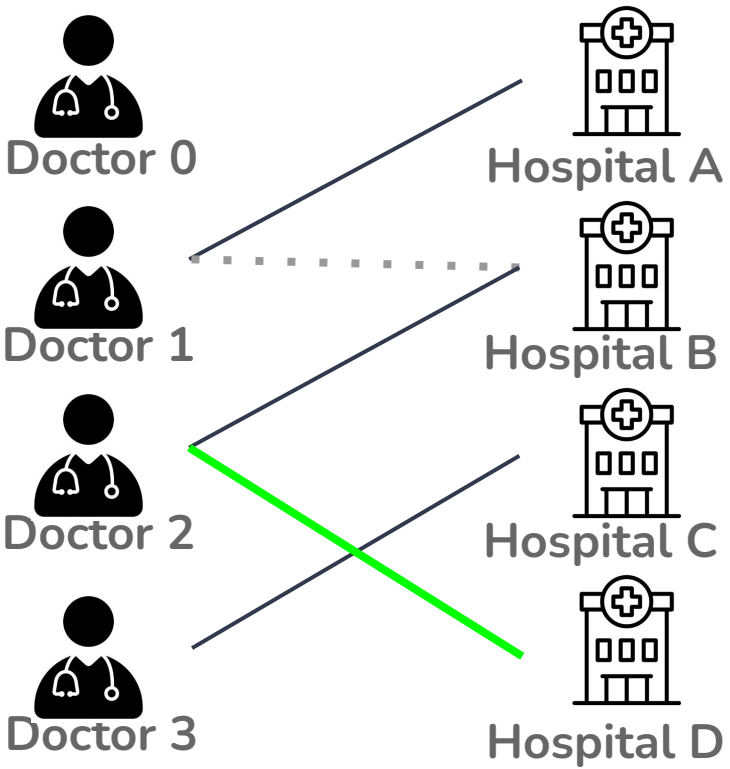
A Run of Gale-Shapley




```
while there are unmatched hospitals:

    the unmatched hospital that is first alphabetically makes an offer to their favorite doctor (that they haven't requested before)

    if the doctor already has an offer they prefer: skip

    if the doctor has no prior offer: add this pair to matched_pairs

    if the doctor has a prior offer but prefers this one: replace their pairing in matched_pairs with this one

return matched_pairs
```

```
matched_pairs = [(A,1), (B, 2), (C, 3)]
```

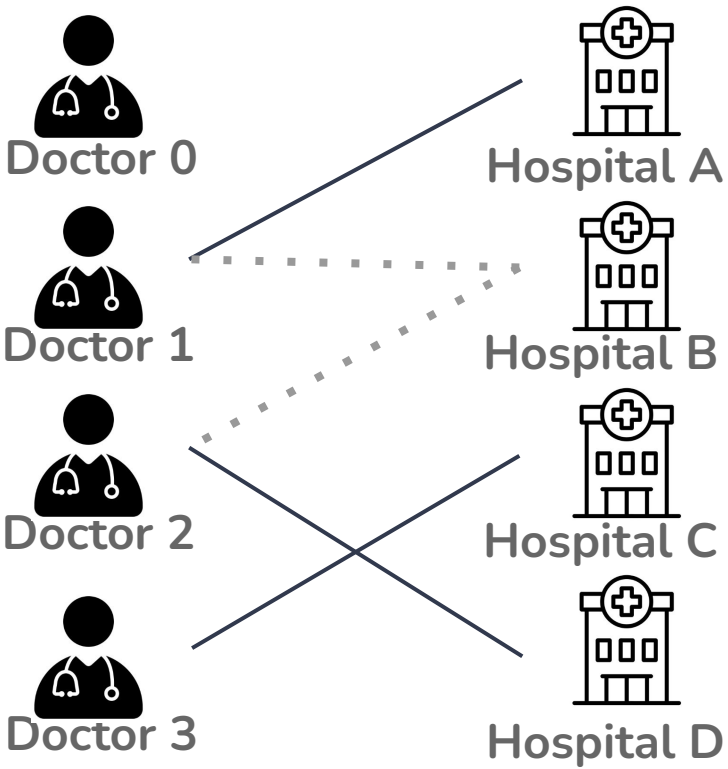
A Run of Gale-Shapley




```
while there are unmatched hospitals:

    the unmatched hospital that is first alphabetically makes an offer to their favorite doctor (that they haven't requested before)

    if the doctor already has an offer they prefer: skip

    if the doctor has no prior offer: add this pair to matched_pairs

    if the doctor has a prior offer but prefers this one: replace their pairing in matched_pairs with this one

return matched_pairs
```

```
matched_pairs = [(A,1), (B, 2), (C, 3)]
```

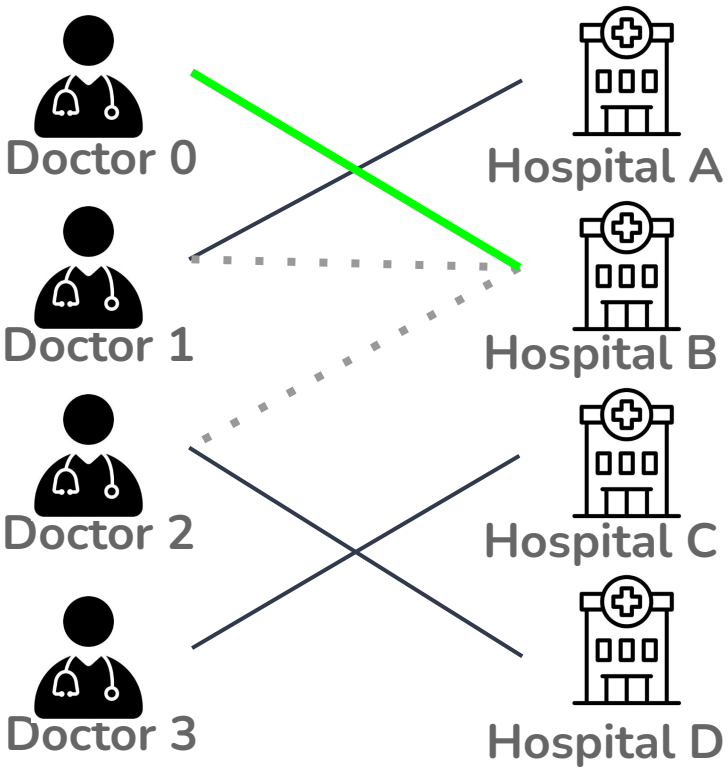
A Run of Gale-Shapley




```
while there are unmatched hospitals:

    the unmatched hospital that is first alphabetically makes an offer to their favorite doctor (that they haven't requested before)

    if the doctor already has an offer they prefer: skip

    if the doctor has no prior offer: add this pair to matched_pairs

    if the doctor has a prior offer but prefers this one: replace their pairing in matched_pairs with this one

return matched_pairs
```

```
matched_pairs = [(A,1), (D, 2), (C, 3)]
```

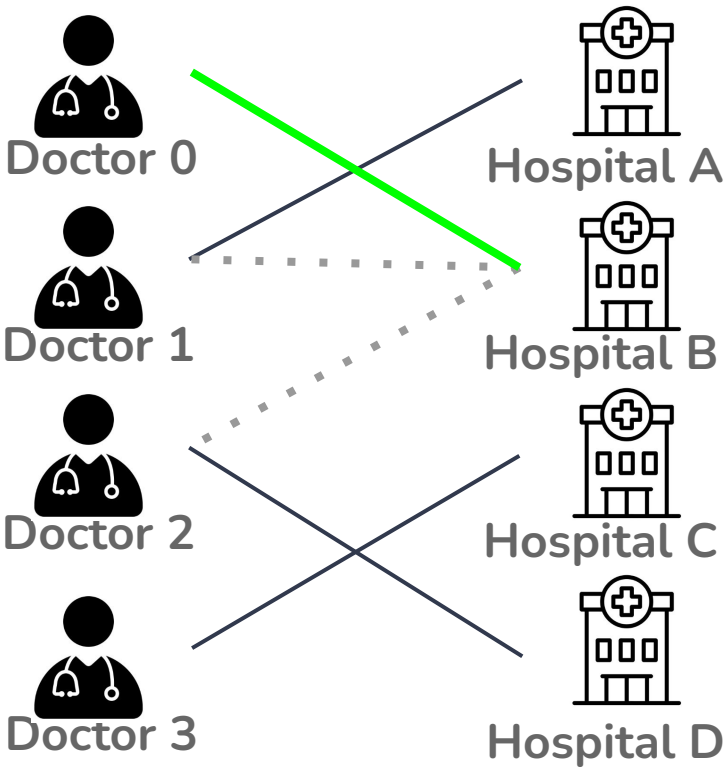
A Run of Gale-Shapley




```
while there are unmatched hospitals:

    the unmatched hospital that is first alphabetically makes an offer to their favorite doctor (that they haven't requested before)

    if the doctor already has an offer they prefer: skip

    if the doctor has no prior offer: add this pair to matched_pairs

    if the doctor has a prior offer but prefers this one: replace their pairing in matched_pairs with this one

return matched_pairs
```

```
matched_pairs = [(A,1), (D, 2), (C, 3)]
```

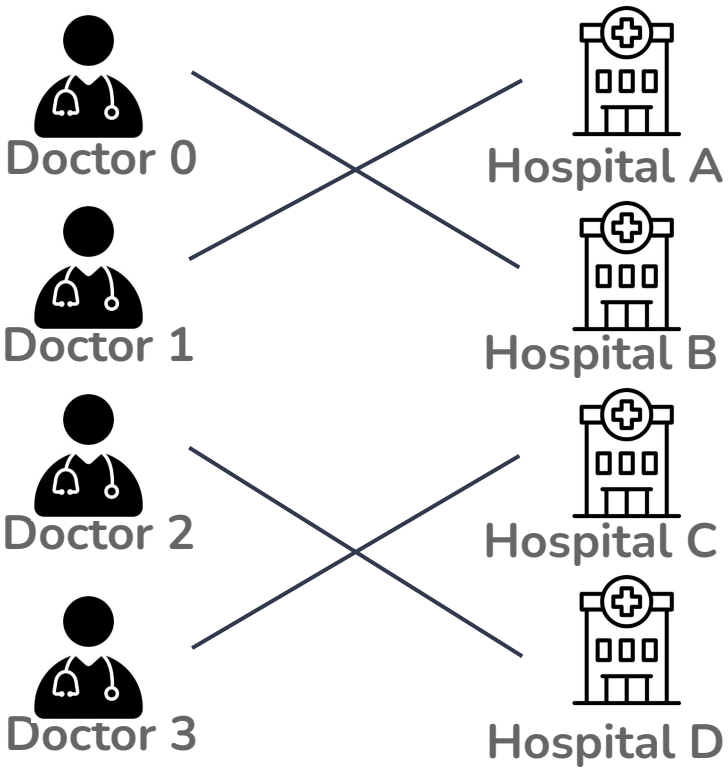
A Run of Gale-Shapley




```
while there are unmatched hospitals:

    the unmatched hospital that is first alphabetically makes an offer to their favorite doctor (that they haven't requested before)

    if the doctor already has an offer they prefer: skip

    if the doctor has no prior offer: add this pair to matched_pairs

    if the doctor has a prior offer but prefers this one: replace their pairing in matched_pairs with this one

return matched_pairs
```

```
matched_pairs=[(A,1),(D, 2),(C, 3),(B, 0)]
```

A Run of Gale-Shapley




```
while there are unmatched hospitals:

    the unmatched hospital that is first alphabetically makes an offer to their favorite doctor (that they haven't requested before)

    if the doctor already has an offer they prefer: skip

    if the doctor has no prior offer: add this pair to matched_pairs

    if the doctor has a prior offer but prefers this one: replace their pairing in matched_pairs with this one

return matched_pairs
```

```
matched_pairs=[(A,1),(D, 2),(C, 3),(B, 0)]
```

A code implementation

https://colab.research.google.com/drive/10k6jSZziqO3pdetLPrXw3yEsOcB3z-ru?usp=sharing

Proving an Algorithm’s Correctness

How do we prove that an algorithm is correct?

Overall, we need to prove the following:

For any input, this algorithm outputs the desired output.

For this problem, that reads as:

For any set of input preferences from doctors and hospitals, this algorithm outputs a stable matching.

But how do we prove that?

How do we prove that an algorithm is correct?

Overall, we need to prove the following:

For any input, this algorithm outputs the desired output.

For this problem, that reads as:

For any set of input preferences from doctors and hospitals, this algorithm outputs a stable matching.

But how do we prove that?

Tip: Break the claim down as much as possible.

How do we prove that an algorithm is correct?

Break down the claim!

Claim: For any set of input preferences from doctors and hospitals, **this algorithm outputs a stable matching.**

How do we prove that an algorithm is correct?

Break down the claim!

Claim: For any set of input preferences from doctors and hospitals, **this algorithm outputs a stable matching.**

- Claim 1: This algorithm outputs a matching.
- Claim 2: The matching output by this algorithm is stable.

Proof of Claim 1: This algorithm outputs a matching.

```
while there are unmatched hospitals
and possible offers:
    an unmatched hospital makes an
    offer (no repeats)

    if the doctor has no prior offer:
    add this pair to matched_pairs

    if the doctor has a prior offer
    but prefers this one: replace
    their pairing in matched_pairs

return matched_pairs
```

Need to show: at the end of this algorithm, every hospital gets matched.

Proof of Claim 1: This algorithm outputs a matching.

```
while there are unmatched hospitals
and possible offers:
    an unmatched hospital makes an
    offer (no repeats)

    if the doctor has no prior offer:
    add this pair to matched_pairs

    if the doctor has a prior offer
    but prefers this one: replace
    their pairing in matched_pairs

return matched_pairs
```

Need to show: at the end of this algorithm, every hospital gets matched.

Proof by Contradiction:

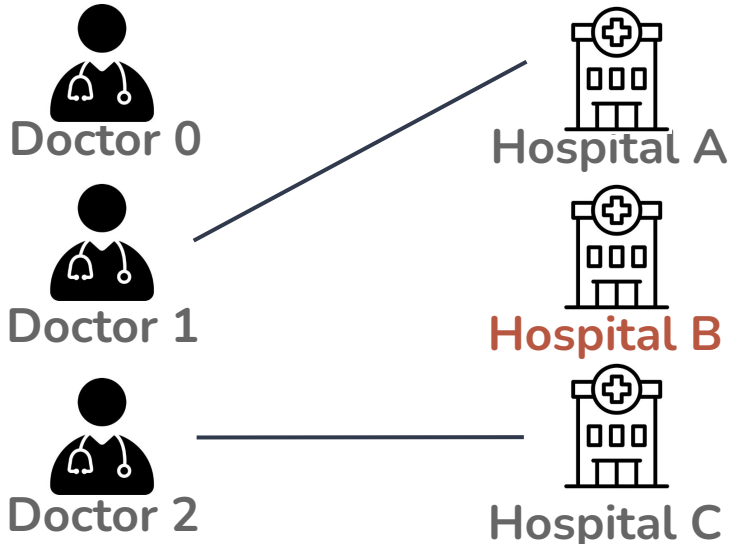
Proof of Claim 1: This algorithm outputs a matching.

```
while there are unmatched hospitals
and possible offers:
    an unmatched hospital makes an
    offer (no repeats)

    if the doctor has no prior offer:
    add this pair to matched_pairs

    if the doctor has a prior offer
    but prefers this one: replace
    their pairing in matched_pairs

return matched_pairs
```

Need to show: at the end of this algorithm, every hospital gets matched.

Proof by Contradiction:

Assume, at the end of the algorithm, some hospital didn't get matched.



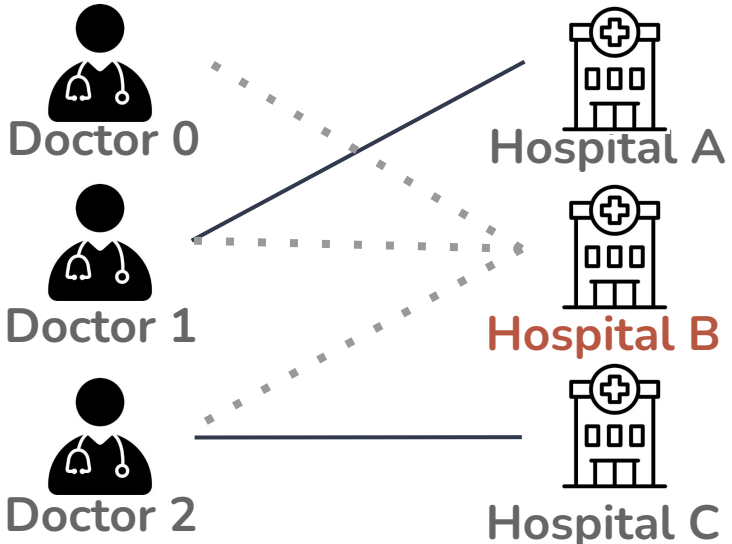
Proof of Claim 1: This algorithm outputs a matching.

```
while there are unmatched hospitals
and possible offers:
    an unmatched hospital makes an
    offer (no repeats)

    if the doctor has no prior offer:
    add this pair to matched_pairs

    if the doctor has a prior offer
    but prefers this one: replace
    their pairing in matched_pairs

return matched_pairs
```

Need to show: at the end of this algorithm, every hospital gets matched.

Proof by Contradiction:

Assume, at the end of the algorithm, some hospital didn't get matched.



Proof of Claim 1: This algorithm outputs a matching.

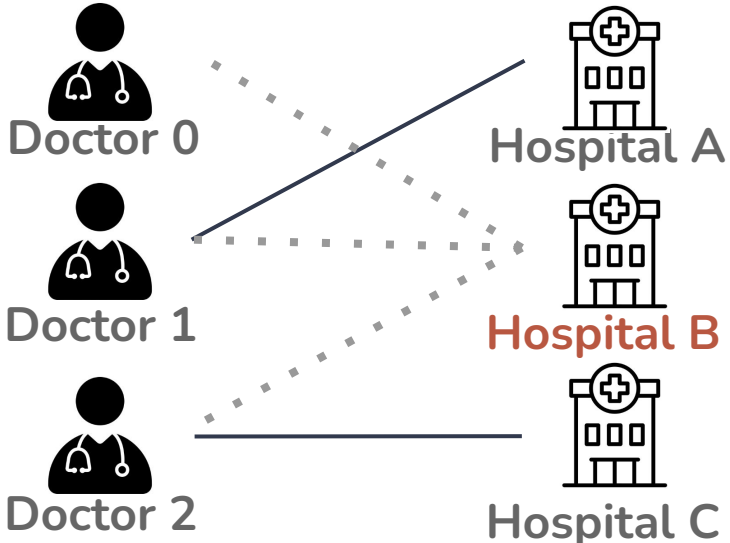
```
while there are unmatched hospitals
and possible offers:
    an unmatched hospital makes an
    offer (no repeats)

    if the doctor has no prior offer:
    add this pair to matched_pairs

    if the doctor has a prior offer
    but prefers this one: replace
    their pairing in matched_pairs

return matched_pairs
```

Need to show: at the end of this algorithm, every hospital gets matched.

Proof by Contradiction:

Assume, at the end of the algorithm, some hospital didn't get matched.

According to the while loop, that hospital must have made an offer to every doctor.



Proof of Claim 1: This algorithm outputs a matching.

```
while there are unmatched hospitals
and possible offers:
    an unmatched hospital makes an
    offer (no repeats)

    if the doctor has no prior offer:
    add this pair to matched_pairs

    if the doctor has a prior offer
    but prefers this one: replace
    their pairing in matched_pairs

return matched_pairs
```

Need to show: at the end of this algorithm, every hospital gets matched.

Proof by Contradiction:

Assume, at the end of the algorithm, some hospital didn't get matched.

That hospital must have made an offer to every doctor.

But if some hospital is unmatched, then some doctor is unmatched as well, so they couldn't have rejected the hospital. Contradiction.

Proof of Claim 1: This algorithm outputs a matching.

```
while there are unmatched hospitals
and possible offers:
    an unmatched hospital makes an
    offer (no repeats)

    if the doctor has no prior offer:
    add this pair to matched_pairs

    if the doctor has a prior offer
    but prefers this one: replace
    their pairing in matched_pairs

return matched_pairs
```

Need to show: at the end of this algorithm, every hospital gets matched.

Proof by Contradiction:

Assume, at the end of the algorithm, some hospital didn't get matched.

That hospital must have made an offer to every doctor.

But if some hospital is unmatched, then some doctor is unmatched as well, so they couldn't have rejected the hospital. Contradiction.

Need to show: at the end of this algorithm, every doctor gets matched.

Proof of Claim 1: This algorithm outputs a matching.

```
while there are unmatched hospitals
and possible offers:
    an unmatched hospital makes an
    offer (no repeats)

    if the doctor has no prior offer:
    add this pair to matched_pairs

    if the doctor has a prior offer
    but prefers this one: replace
    their pairing in matched_pairs

return matched_pairs
```

Need to show: at the end of this algorithm, every hospital gets matched.

Proof by Contradiction:

Assume, at the end of the algorithm, some hospital didn't get matched.

That hospital must have made an offer to every doctor.

But if some hospital is unmatched, then some doctor is unmatched as well, so they couldn't have rejected the hospital. Contradiction.

Need to show: at the end of this algorithm, every doctor gets matched.

Every hospital gets matched, so there are n pairs, so every doctor gets matched too.

Proof of Claim 2: The matching is stable.

```
while there are unmatched hospitals
and possible offers:
    an unmatched hospital makes an
    offer (no repeats)

    if the doctor has no prior offer:
    add this pair to matched_pairs

    if the doctor has a prior offer
    but prefers this one: replace
    their pairing in matched_pairs

return matched_pairs
```

Need to show: there are no blocking/unstable pairs.

Proof of Claim 2: The matching is stable.

```
while there are unmatched hospitals
and possible offers:
    an unmatched hospital makes an
    offer (no repeats)

    if the doctor has no prior offer:
    add this pair to matched_pairs

    if the doctor has a prior offer
    but prefers this one: replace
    their pairing in matched_pairs

return matched_pairs
```

Need to show: there are no blocking/unstable pairs.

Direct Proof:

Take any pair (*h*, *d*) that isn't in matched_pairs. Either *h* made an offer to *d*, or it didn't.

Say *h* made an offer to *d*. Then, either *d* rejected *h* immediately, or *d* replaced *h* down the line. This is only possible if *d* got a better offer, so *d* prefers its match over *h*.

Say *h* didn't make an offer to *d*. Then *h* is matched with someone it made an offer to before *d*, so it prefers its match to *d*.

Thus, this pair can not be unstable.

Congrats! You've won a Nobel Prize.

COLLEGE ADMISSIONS AND THE STABILITY OF MARRIAGE

D. GALE* AND L. S. SHAPLEY, Brown University and the RAND Corporation

1. Introduction. The problem with which we shall be concerned relates to the following typical situation: A college is considering a set of n applicants of which it can admit a quota of only q. Having evaluated their qualifications, the admissions office must decide which ones to admit. The procedure of offering admission only to the q best-qualified applicants will not generally be satisfactory, for it cannot be assumed that all who are offered admission will accept. Accordingly, in order for a college to receive q acceptances, it will generally have to offer to admit more than q applicants. The problem of determining how many and which ones to admit requires some rather involved guesswork. It may not be known (a) whether a given applicant has also applied elsewhere; if this is known it may not be known (b) how he ranks the colleges to which he has applied; even if this is known it will not be known (c) which of the other colleges will offer to admit him. A result of all this uncertainty is that colleges can expect only that the entering class will come reasonably close in numbers to the desired quota, and be reasonably close to the attainable optimum in quality.

KUNGL. VETENSKAPS-AKADEMIEN
THE ROYAL SWEDISH ACADEMY OF SCIENCES

The Prize in Economic Sciences 2012

This year's Prize to **Lloyd Shapley** and **Alvin Roth** extends from abstract theory developed in the 1960s, over empirical work in the 1980s, to ongoing efforts to find practical solutions to real-world problems. Examples include the assignment of new doctors to hospitals, students to schools, and human organs for transplant to recipients. Lloyd Shapley made the early theoretical contributions, which were unexpectedly adopted two decades later when Alvin Roth investigated the market for U.S. doctors. His findings generated further analytical developments, as well as practical design of market institutions.

Stable matching: Theory, evidence, and practical design

Traditional economic analysis studies markets where prices adjust so that supply equals demand. Both theory and practice show that markets function well in many cases. But in some situations, the standard market mechanism encounters problems, and there are cases where prices cannot be used at all to allocate resources. For example, many schools and universities are prevented from charging tuition fees and, in the case of human organs for transplants, monetary payments are ruled out on ethical grounds. Yet, in these – and many other – cases, an allocation has to be made. How do such processes actually work, and when is the outcome efficient?

Matching theory

The Gale-Shapley algorithm

Analysis of allocation mechanisms relies on a rather abstract idea. If rational people – who know their best interests and behave accordingly – simply engage in unrestricted

Summary

Today we saw our first algorithm: the Gale-Shapley algorithm for Stable Matchings.

We saw the real-world problem it stemmed from, got our first experience with pseudocode and its norms, and explored how to write proofs to verify the correctness of algorithms.

Next time, we'll analyze the efficiency of the algorithm, and discuss the algorithm's real-world implications.

These slides were strongly influenced by the official slides for Algorithm Design by Kleinberg and Tardos, and by slides from Professor Omar Ibrahim.