CSE 331:
Algorithms & Complexity
"More Concerns..."

Prof. Charlie Anne Carlson (She/Her)

Lecture 2

Monday August 29th, 2025

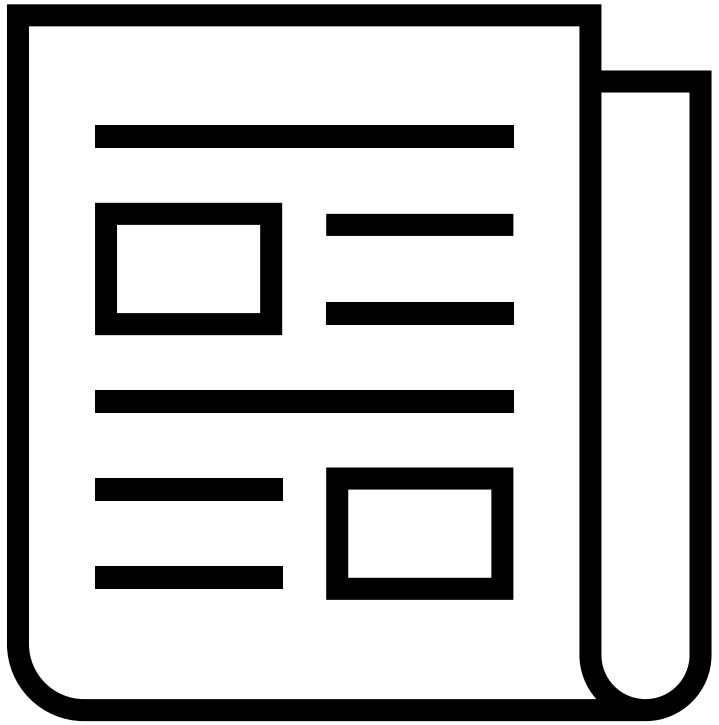
University at Buffalo®

Schedule

1. Course Updates
2. Notation
3. Halting Problem
4. Note on Induction
5. Learning Outcomes
6. When to Algorithm
7. Matchings

Course Updates

- Office Hours Posted Soon
- Complete Syllabus Quiz
- HW0 is Due on Tuesday
- Next Week Project Talk



Notation

Motivational Stuff

Sometime a few choices that I make in CSE 331 might seem archaic so in these pages I try to motivate why we do things a certain way.

- CSE 331 TA advice
- Do we need asymptotic analysis?
- Why do we need proofs?
- CSE 331 testimonials

Mathematical Background

CSE 331 will need a fair bit of math: most of which you must have seen earlier. However, if you have not used those material for a bit then you might be a bit rusty. The pages linked below are some notes that I wrote up that might help you refresh the material that you might have seen in CSE 116, 191 or 250.

- Logarithms.
- Proof by Induction.
- Proving an implication (with some common proof techniques).
- Proving $S = T$.
- Reduction.
- Using a Progress Measure.
- Pigeon-hole Principle.
- Some Useful Identities.
- Some Useful Notation.

Other Material

Finally sometimes (but hopefully not often!) we will use material that might not have been covered in previous courses and we did not have much time to cover in class: these pages will fill in those gaps.

- Analyzing the worst-case runtime of an algorithm
- Using the adversarial argument to prove lower bounds

Notation

Notation is incredibly useful to write proofs. This page collects some notation that is used in this course.

Discrete Math Notation

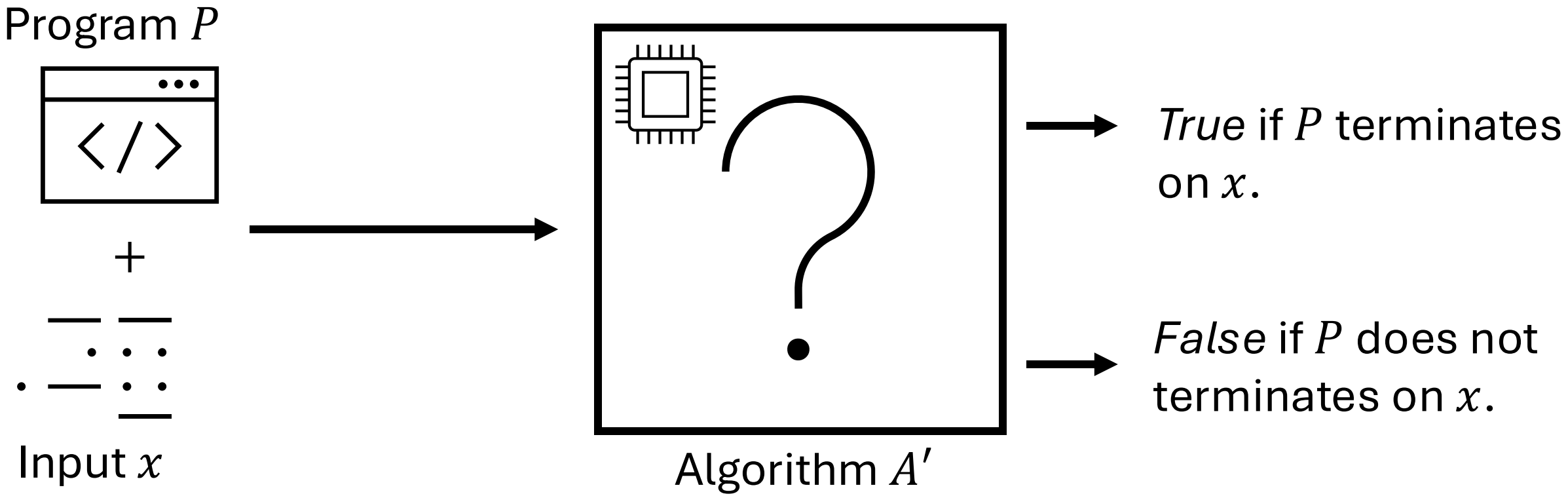
In this part of the note we collect notation that you should have seen in CSE 191.

Logical Connectives

Notation	Meaning
$P \wedge Q$	The logical AND of boolean variables P and Q
$P \vee Q$	The logical OR of boolean variables P and Q
$\neg P$	Negation of the boolean variable P
$P \implies Q$	Logically equivalent to $\neg P \vee Q$
$\forall x P(x)$	For every x (in the appropriate domain), the boolean predicate $P(x)$ is true
$\exists x P(x)$	There exists at least one x_0 (in the appropriate domain), the boolean predicate $P(x_0)$ is true

Halting Problem 2.0

Input: A Program P and Input x.
Output: True if P terminates on x. Otherwise, false.



Halting Problem 2.0 is “impossible”

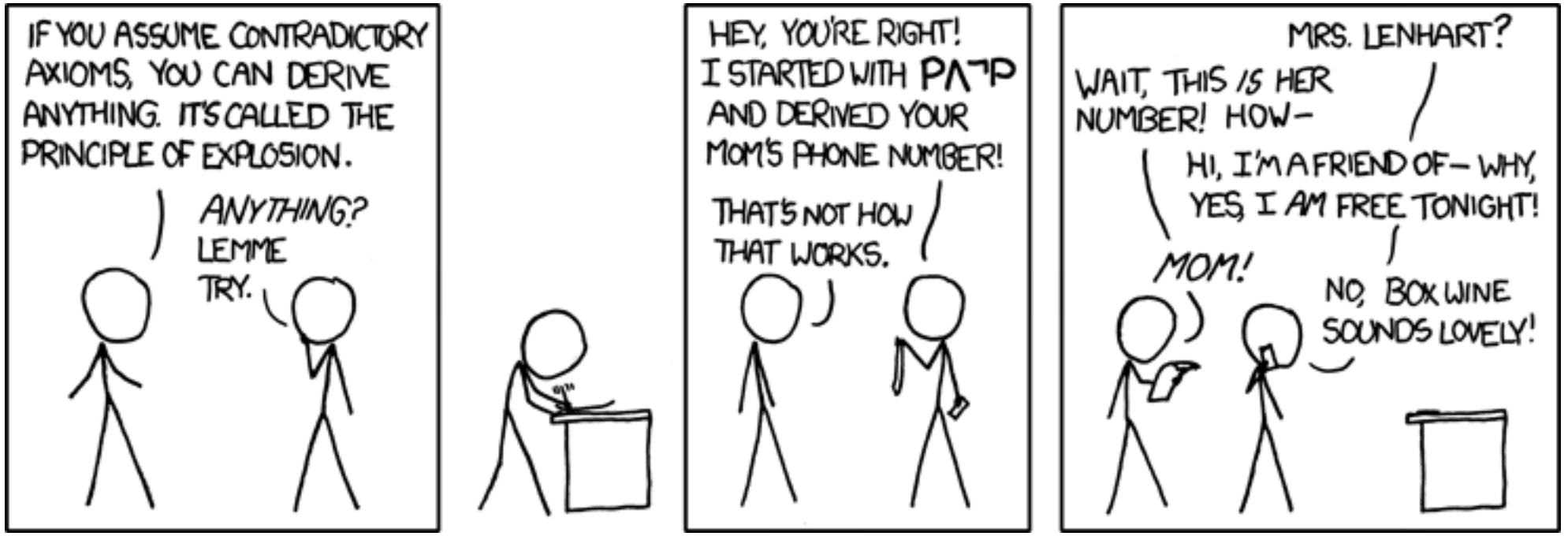
Theorem: There exist no algorithm/program that will always halt and will always correctly *decide* Halting Problem 2.0.

Proof (by contradiction)

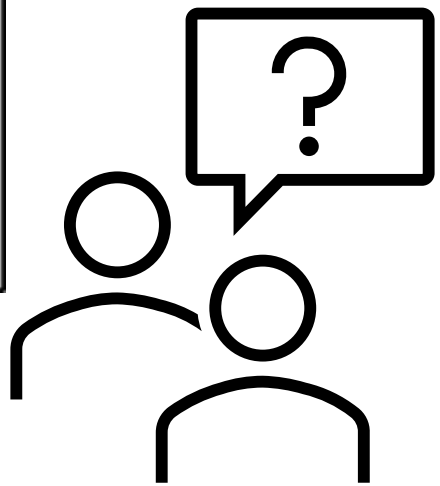
1. Assume the opposite: There is a problem that solves Halting Problem 2.0. Call this `magic(P,I)`.
    1. It must always return something.
    2. It must always return the right answer.
2. Use `magic` to construct a **contradiction**.
3. Conclude that original assumption was wrong.

You keep using that word…

Q1: What is a contradiction?



https://xkcd.com/704/



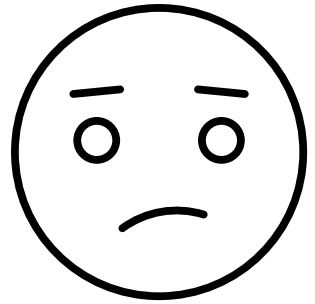
Proof (by contradiction)

```
def contradiction ( P ): # This function takes a program as an input

#Run magic on (P,P)
if magic(P,P): # Use an UTM to make this call
    while True:
        continue # Do nothing

return # Just terminate if magic(P,P) returns False
```

Q2: What happens when we run `contradiction(contradiction)`?



Proof (by contradiction)

1. Assume the opposite: There is a problem that solves Halting Problem 2.0. Call this `magic(P,I)`.
    1. It must always return something.
    2. It must always return the right answer.
2. Use `magic` to construct a **contradiction**.
3. **Conclude that original assumption was wrong**.

Prime Numbers

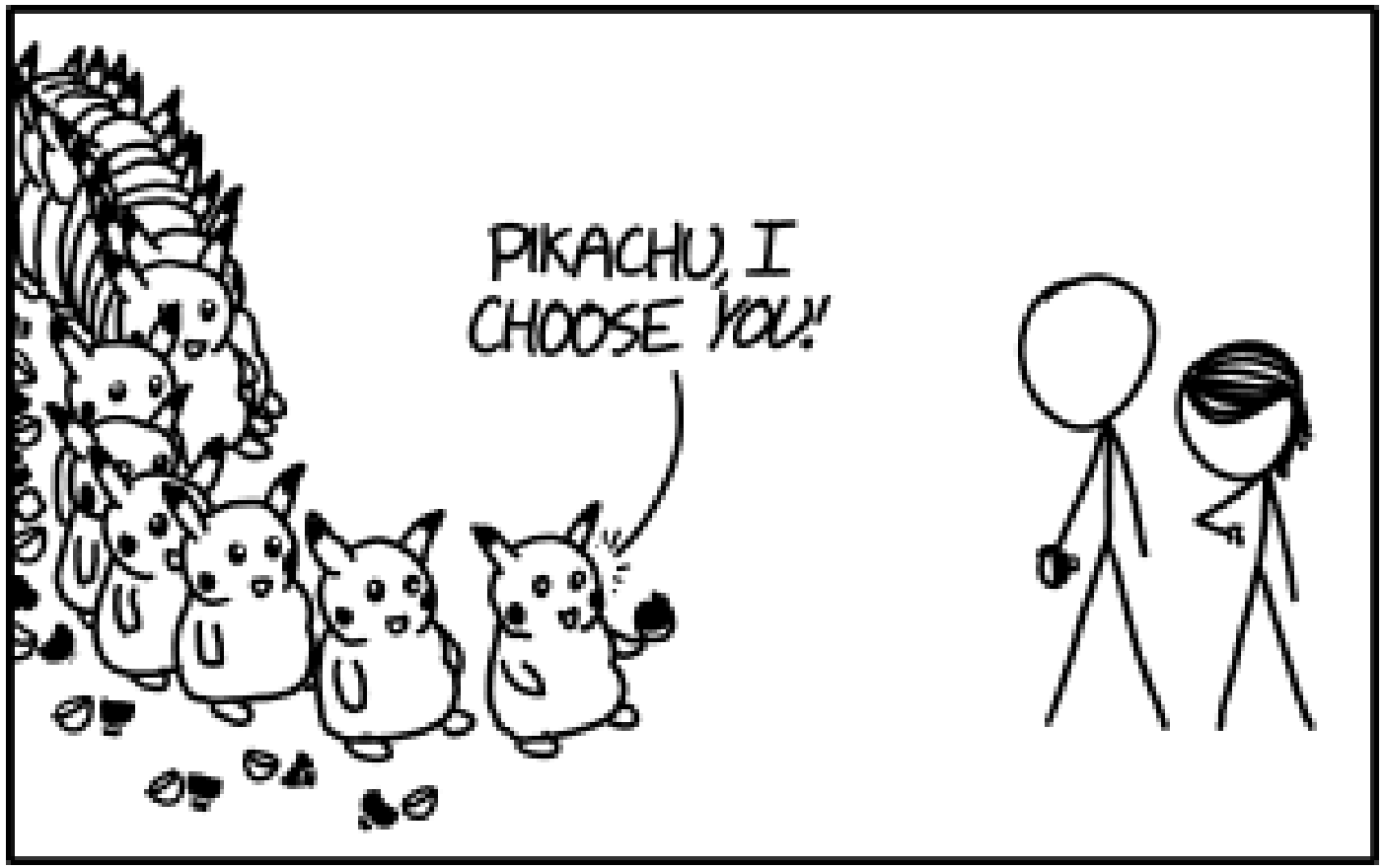
- Recall that natural number z is *prime* if it is not the product of two smaller numbers.
- **Q0:** How many prime numbers are there?

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599…

Prime Numbers

1. Assume there are only n prime numbers, $p_1, \ldots, p_n$.
2. Let $P = 1 + \prod_{i=1}^{n} p_i$.
3. Is P prime?
    1. By assumption it can't be and so it must be composite.
4. Since P is composite, there must exist a $p_j > 1$ such that p_j divides P.
    1. This implies that p_j divides 1... -><-

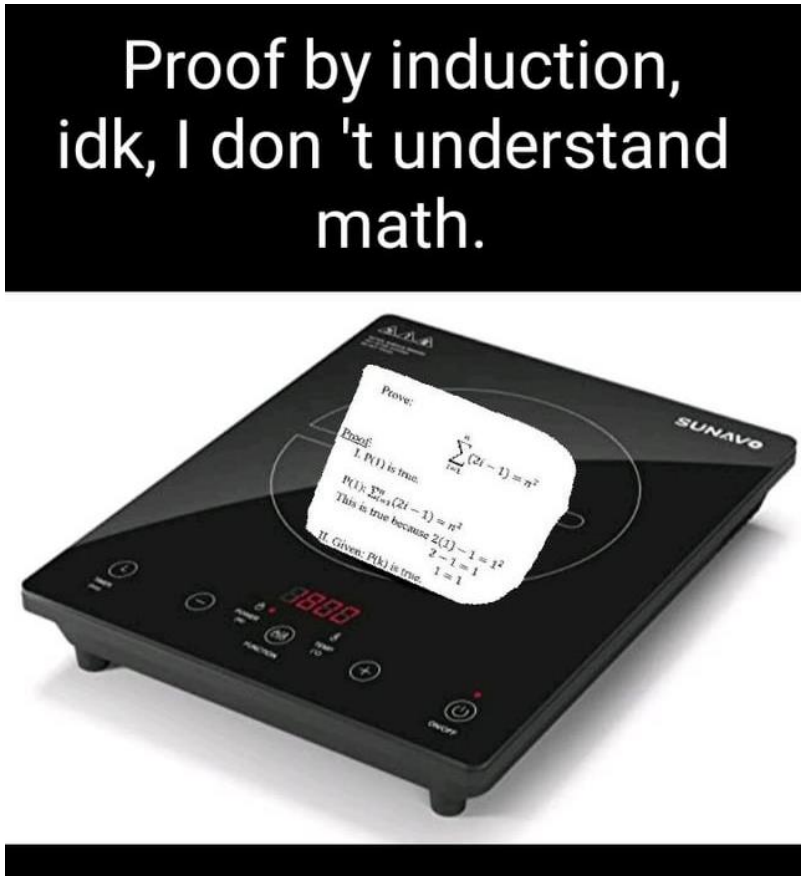
Proof by Induction



https://xkcd.com/1516/

Proof by Induction

- Statement of Intent
- Base Case
- Inductive Hypothesis
- Inductive Step
- Conclusion



Bongcloudforthewin. "**Proof by induction.**" **Reddit, August 16, 2020.** https://www.reddit.com/r/mathmemes/comments/iasdsb/proof_by_induction/

Learning Outcomes

(ABET) Learning Outcomes

This course is required of all computer science students and after the completion of the course, students should demonstrate mastery of the concepts/skills/knowledge expressed in the following learning outcomes for computer science:

- `(4)` recognize professional responsibilities and make informed judgments in computing practice based on legal and ethical principles.
- `(5)` Function effectively as a member or leader of a team engaged in activities appropriate to the program's discipline.
- `(6)` Apply computer science theory and software development fundamentals to produce computing-based solutions. [CS]

Course Learning Outcome	Program Outcomes / Competencies	Instructional Method(s)	Assessment Method(s)
Be able to design algorithms to solve given problems	ABET (6)	Lectures	Homeworks, Exams
Be able to prove correctness of designed algorithms	ABET (6)	Lectures	Homeworks, Exams
Be able to identify ethical and societal implications of algorithms	ABET (4)	Lectures	Project
Be able to effectively work in a team	ABET (5)	Lectures	Project

The Student Outcomes from the Computing Accreditation Commission (CAC) of ABET have been adopted.

Program Outcome Support (Computer Science ABET Outcomes):

Program Outcome	1	2	3	4	5	6
Support Level	No coverage	No coverage	No coverage	Demonstrate mastery of skill/concept	Demonstrate mastery of skill/concept	Demonstrate mastery of skill/concept

Learning Outcomes

(ABET) Learning Outcomes

This course is required of all computer science students and after the completion of the course, students should demonstrate mastery of the concepts/skills/knowledge expressed in the following learning outcomes for computer science:

- `(4)` recognize professional responsibilities and make informed judgments in computing practice based on legal and ethical principles.
- `(5)` Function effectively as a member or leader of a team engaged in activities appropriate to the program's discipline.
- `(6)` Apply computer science theory and software development fundamentals to produce computing-based solutions. [CS]

Course Learning Outcome	Program Outcomes / Competencies	Instructional Method(s)	Assessment Method(s)
Be able to design algorithms to solve given problems	ABET (6)	Lectures	Homeworks, Exams
Be able to prove correctness of designed algorithms	ABET (6)	Lectures	Homeworks, Exams
Be able to identify ethical and societal implications of algorithms	ABET (4)	Lectures	Project
Be able to effectively work in a team	ABET (5)	Lectures	Project

The Student Outcomes from the Computing Accreditation Commission (CAC) of ABET have been adopted.

Program Outcome Support (Computer Science ABET Outcomes):

Program Outcome	1	2	3	4	5	6
Support Level	No coverage	No coverage	No coverage	Demonstrate mastery of skill/concept	Demonstrate mastery of skill/concept	Demonstrate mastery of skill/concept

When to Algorithm?

- Identify a **problem/need/desire**
- Convert to formal **“math”** definition**:**
  - Identify the **inputs**
  - Identify the **outputs/goals**
  - Make **assumptions** and identify **limitations**.
- Design an algorithm to solve the problem
- Analyze the algorithm.

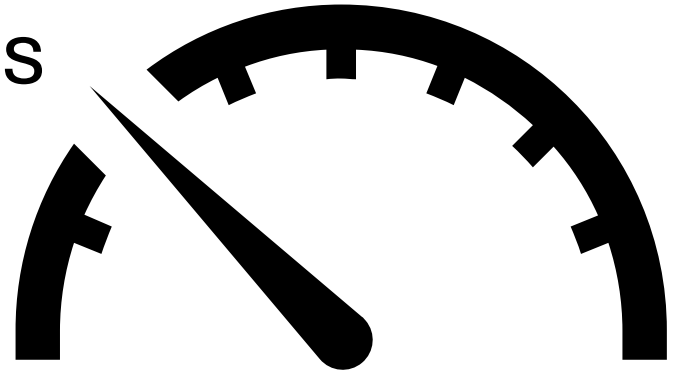
When to Algorithm?

- Identify a **problem/need/desire**
- Convert to formal **“math”** definition**:**
  - Identify the **inputs**
  - Identify the **outputs/goals**
  - Make **assumptions** and identify **limitations**.
- Design an algorithm to solve the problem
- Analyze the algorithm.
- Consider wider impact of solution?!

Too Fast/Too Slow

An instructor with a big class wants live feedback from their class on if they are going **too fast** or **too slow**. To this end, the instructor decides to create a phone app that lets students vote during class on if the instructor should speed up or slow down.

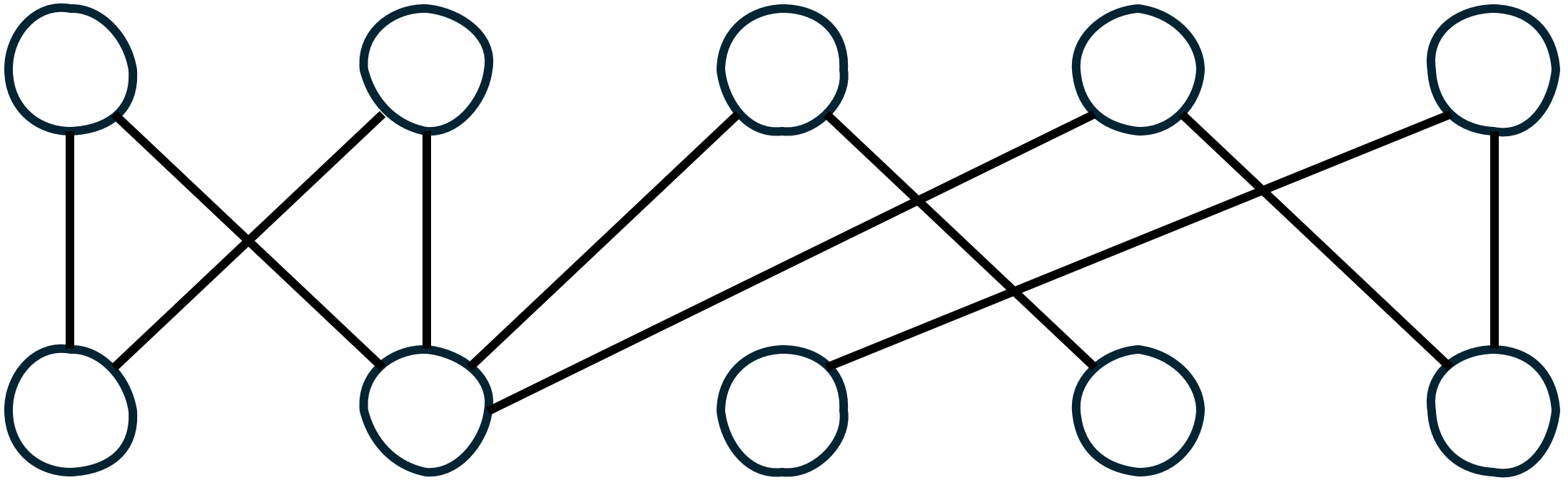
Q3: How would you go about creating this app/algorithm?



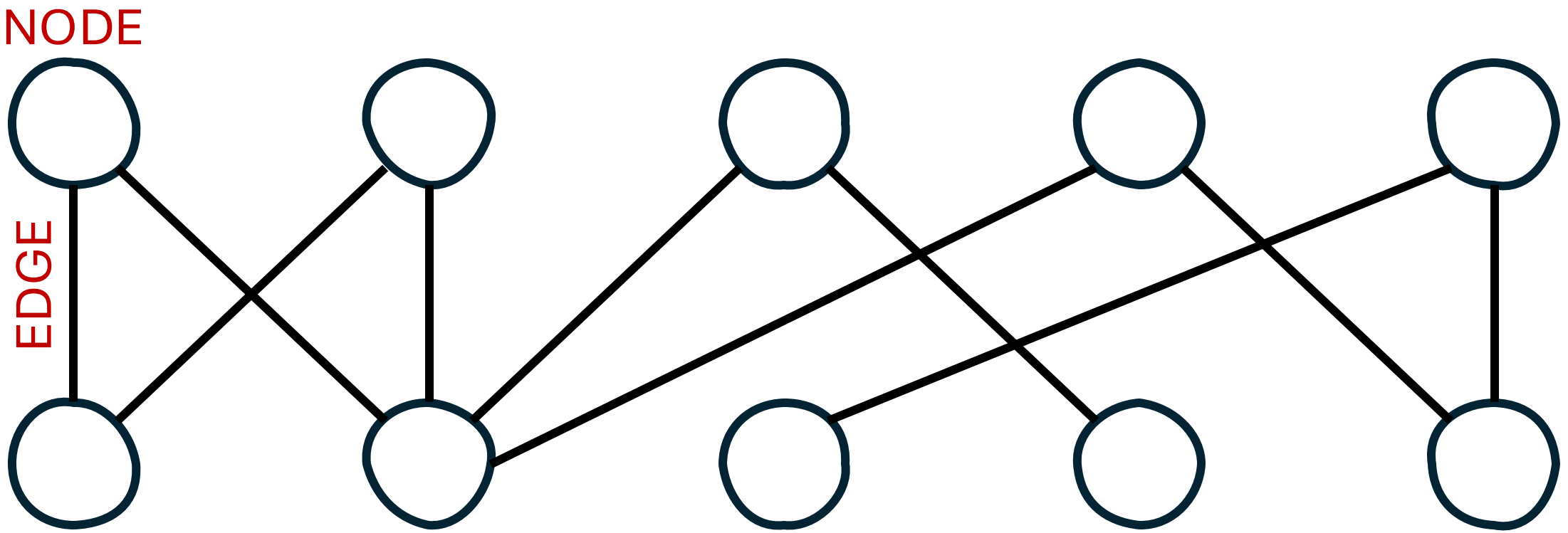
When to Algorithm?

- Identify a **problem/need/desire**
- Convert to formal **“math”** definition**:**
  - Identify the **inputs**
  - Identify the **outputs/goals**
  - Make **assumptions** and identify **limitations**.
- Design an algorithm to solve the problem
- Analyze the algorithm.
- Consider wider impact of solution?!

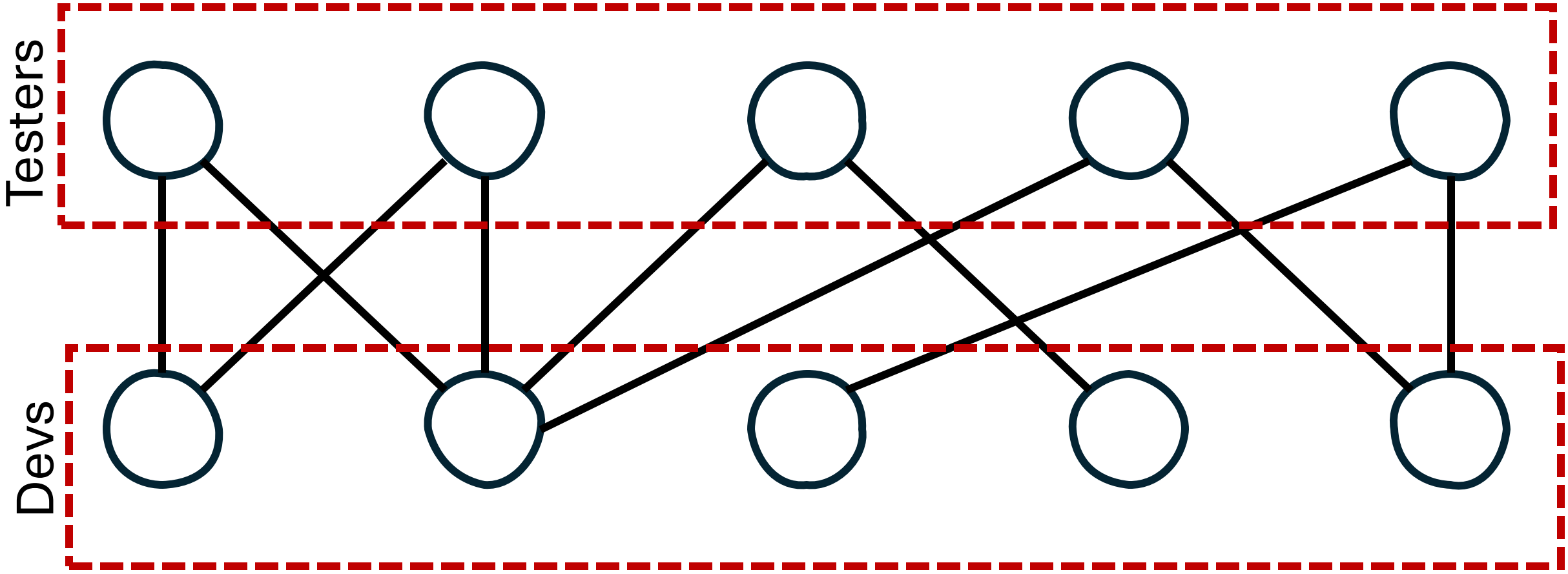
Graphs



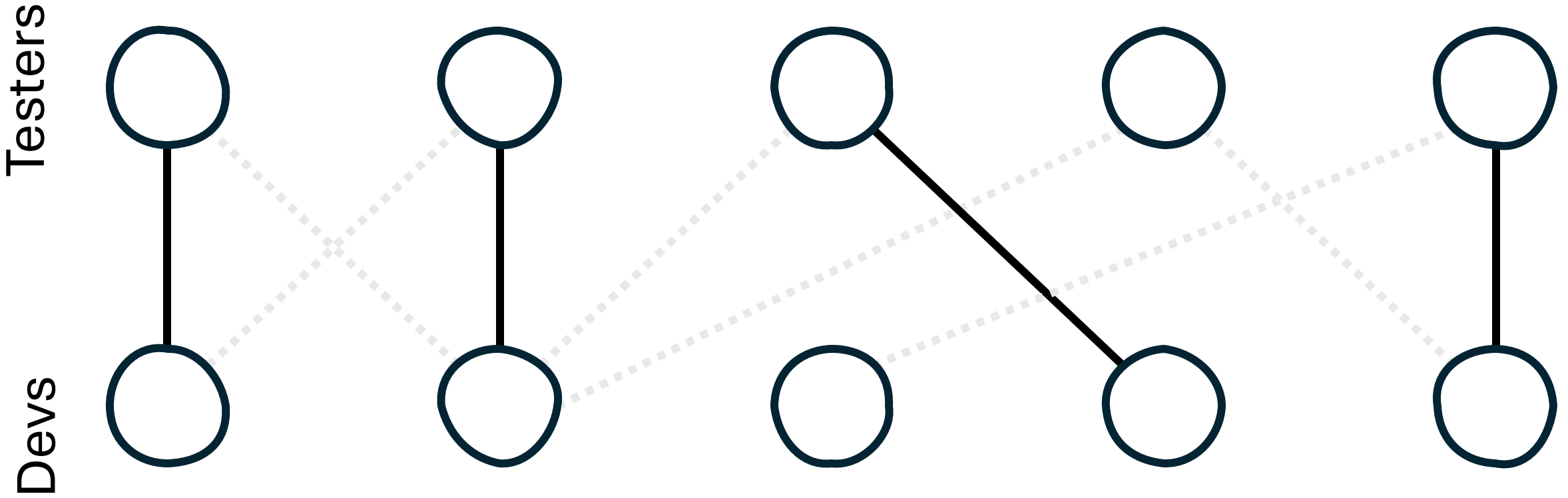
Graphs



Devs + Testers = Success

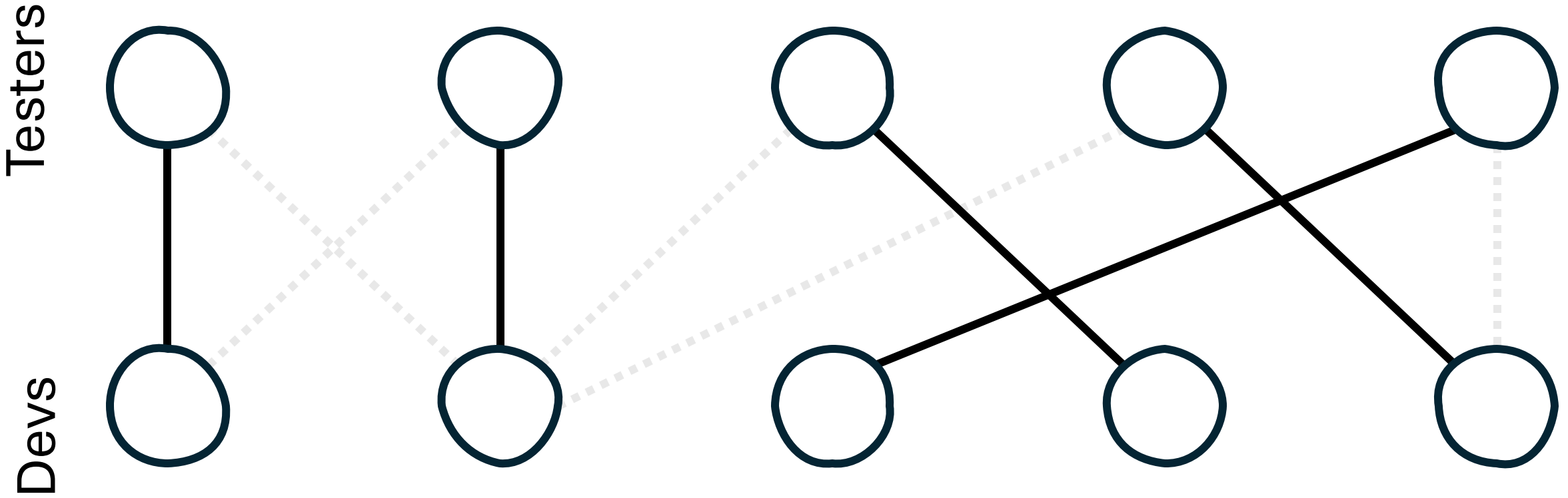


Matching



Definition: A ***matching*** is a collection of edges such that no two share a node.

Perfect Matching



Definition: A ***perfect matching*** is a matching such that every node is incident to an edge.