01204211 Discrete Mathematics
Lecture 1: Introduction

Jittat Fakcharoenphol

June 27, 2021

What is this course about?

This is a math class.

What is this course about?

This is a math class.
But, what is mathematics?

What is this course about?

This is a math class.
But, what is mathematics?
Ah... that's a philosopical question.

What is this course about?

This is a math class.
But, what is mathematics?
Ah... that's a philosopical question.
IMHO, mathematics is a mean to communicate *precise* ideas.

It's like learning a new language

- Do you remember the time when you start learning English?

It's like learning a new language

- Do you remember the time when you start learning English?
- There are a few things you have to learn and get used to.

It's like learning a new language

- Do you remember the time when you start learning English?
- There are a few things you have to learn and get used to.
- They might not make so much sense in the beginning, but over time, you will get comfortable with how the language is used.

It's like learning a new language

- Do you remember the time when you start learning English?
- There are a few things you have to learn and get used to.
- They might not make so much sense in the beginning, but over time, you will get comfortable with how the language is used.
- As your knowledge of the language gets better, everything becomes more natural. Learning a new language sometimes expands your view of the world.

It’s like learning a new language

- Do you remember the time when you start learning English?
- There are a few things you have to learn and get used to.
- They might not make so much sense in the beginning, but over time, you will get comfortable with how the language is used.
- As your knowledge of the language gets better, everything becomes more natural. Learning a new language sometimes expands your view of the world.
- I hope it is also true with this course.

The goals of this course

There are two goals:

- To learn how to make mathematical arguments.

The goals of this course

There are two goals:

- To learn how to make mathematical arguments.
- To learn various fundamental mathematical concepts that are very useful in computer science.

Why should we learn how to prove? (1)

Why should we learn how to prove? (1)

Look at this program.

```
if a > b:
    return a
else:
    return b
```

The author claims that this program takes two variables a and b and returns the larger one.

Why should we learn how to prove? (1)

Look at this program.

```
if a > b:
    return a
else:
    return b
```

The author claims that this program takes two variables a and b and returns the larger one.

Do you believe the author of the code?

Why should we learn how to prove? (1)

Look at this program.

```
if a > b:
    return a
else:
    return b
```

The author claims that this program takes two variables a and b and returns the larger one.

Do you believe the author of the code? Why?

Finding the maximum value. (1)

Now look at this program.

```
if a > b:
    if a > c:
        return a
    else:
        return c
else:
    if c > b:
        return c
    else:
        return b
```

The author claims that this program takes three variables a, b and c and returns the largest one.

Finding the maximum value. (1)

Now look at this program.

```
if a > b:
    if a > c:
        return a
    else:
        return c
else:
    if c > b:
        return c
    else:
        return b
```

The author claims that this program takes three variables a, b and c and returns the largest one.
Do you believe the author of the code?

Finding the maximum value. (1)

Now look at this program.

```
if a > b:
    if a > c:
        return a
    else:
        return c
else:
    if c > b:
        return c
    else:
        return b
```

The author claims that this program takes three variables a, b and c and returns the largest one.
Do you believe the author of the code? Why?

Finding the maximum value. (2)

Finally, look at this program.

```
// Input: array A with n elements: A[0],...,A[n-1]
m = 0
for i = 0, 1, ..., n-1:
  if A[i] > m:
    m = A[i]
return m
```

The author claims that this program takes an array A with n elements and returns the maximum element.

Finding the maximum value. (2)

Finally, look at this program.

```
// Input: array A with n elements: A[0],...,A[n-1]
m = 0
for i = 0, 1, ..., n-1:
  if A[i] > m:
    m = A[i]
return m
```

The author claims that this program takes an array A with n elements and returns the maximum element.
Do you believe the author of the code?

Finding the maximum value. (2)

Finally, look at this program.

```
// Input: array A with n elements: A[0],...,A[n-1]
m = 0
for i = 0, 1, ..., n-1:
  if A[i] > m:
    m = A[i]
return m
```

The author claims that this program takes an array A with n elements and returns the maximum element.
Do you believe the author of the code? Why?

Finding the maximum value. (2)

Finally, look at this program.

```
// Input: array A with n elements: A[0],...,A[n-1]
m = 0
for i = 0, 1, ..., n-1:
  if A[i] > m:
    m = A[i]
return m
```

The author claims that this program takes an array A with n elements and returns the maximum element.

Do you believe the author of the code? Why?

Can we try to test the code with all possible inputs?

input → Code → output

expected output ✓ ✗

Finding the maximum value. (3)

Let's try again.

```
// Input: array A with n elements: A[0],...,A[n-1]
m = A[0]
for i = 1, 2, ..., n-1:
  if A[i] > m:
    m = A[i]
return m
```

Do you believe the author of the code?

Finding the maximum value. (3)

Let's try again.

```
// Input: array A with n elements: A[0],...,A[n-1]
m = A[0]
for i = 1, 2, ..., n-1:
  if A[i] > m:
    m = A[i]
return m
```

Do you believe the author of the code? Why?

Finding the maximum value. (3)

Let's try again.

```
// Input: array A with n elements: A[0],...,A[n-1]
m = A[0]
for i = 1, 2, ..., n-1:
  if A[i] > m:
    m = A[i]
return m
```

Do you believe the author of the code? Why?

Can we try to test the code with all possible inputs?

Another example: testing primes (1)

1 ไม่ใช่ จน.เฉพาะ

A *prime* is a natural number greater than 1 that has no positive divisors other 1 and itself. E.g., 2,3,5,7,11 are primes.

Another example: testing primes (1)

A *prime* is a natural number greater than 1 that has no positive divisors other 1 and itself. E.g., 2,3,5,7,11 are primes.

```
Algorithm CheckPrime(n):       // Input: an integer n
    if n <= 1:
        return False
    i = 2
    while i <= n-1:
        if n is divisible by i:
            return False
        i = i + 1
    return True
```

The code above checks if n is a prime number. How fast can it run?

Another example: testing primes (1)

A *prime* is a natural number greater than 1 that has no positive divisors other 1 and itself. E.g., 2,3,5,7,11 are primes.

```
Algorithm CheckPrime(n):      // Input: an integer n
    if n <= 1:
        return False
    i = 2
    while i <= n-1:
        if n is divisible by i:
            return False
        i = i + 1
    return True
```

The code above checks if n is a prime number. How fast can it run?

Note that if n is a prime number, the for-loop repeats for $n - 2$ times. Thus, the running time is approximately proportional to n.

Another example: testing primes (1)

A *prime* is a natural number greater than 1 that has no positive divisors other 1 and itself. E.g., 2,3,5,7,11 are primes.

```
Algorithm CheckPrime(n):     // Input: an integer n
    if n <= 1:
        return False
    i = 2
    while i <= n-1:
        if n is divisible by i:
            return False
        i = i + 1
    return True
```

- The code above checks if n is a prime number. How fast can it run?
- Note that if n is a prime number, the for-loop repeats for $n-2$ times. Thus, the running time is approximately proportional to n.
- Can we do better?

Another example: testing primes (2)

Consider the following code.

```
Algorithm CheckPrime2(n):  // Input: an integer n
    if n <= 1:
        return False
    let s = square root of n
    i = 2
    while i <= s:
        if n is divisible by i:
            return False
        i = i + 1
    return True
```

$\sqrt{n}$

$\sqrt{n}$ sov.

How fast can it run?

Another example: testing primes (2)

10 3
100 10
1000 [illegible]
1000000 1000

Consider the following code.

```
Algorithm CheckPrime2(n):  // Input: an integer n
    if n <= 1:
        return False
    let s = square root of n
    i = 2
    while i <= s:
        if n is divisible by i:
            return False
        i = i + 1
    return True
```

How fast can it run? Note that $s = \sqrt{n}$; therefore, it takes time approximately proportional to $\sqrt{n}$ to run.

Another example: testing primes (2)

CheckPrime(n):
~~return False~~

Consider the following code.

```
Algorithm CheckPrime2(n):  // Input: an integer n
    if n <= 1:
        return False
    let s = square root of n
    i = 2
    while i <= s:
        if n is divisible by i:
            return False
        i = i + 1
    return True
```

1, 2, 2, 3, ..., n-1

2, 3, ..., √n

ถ้า n เป็น prime | ถ้า n ไม่เป็น prime

false

✓

How fast can it run? Note that $s = \sqrt{n}$; therefore, it takes time approximately proportional to $\sqrt{n}$ to run.

Ok, it should be faster. **But is it correct?** ????

Informal arguments (1)

- Let's try to argue the Algorithm `CheckPrime2` works correctly.

Informal arguments (1)

- Let's try to argue the Algorithm `CheckPrime2` works correctly.
- Note that if n is a prime number, the algorithm answers correctly. (Why?)

Informal arguments (1)

- Let's try to argue the Algorithm `CheckPrime2` works correctly.
- Note that if n is a prime number, the algorithm answers correctly. (Why?)
- Therefore, let's consider the case when n is not prime (i.e., n is a composite).

Informal arguments (1)

- Let's try to argue the Algorithm `CheckPrime2` works correctly.
- Note that if n is a prime number, the algorithm answers correctly. (Why?)
- Therefore, let's consider the case when n is not prime (i.e., n is a composite).
- If that's the case, n has some positive divisor which is not 1 or n. Let's call this number a.

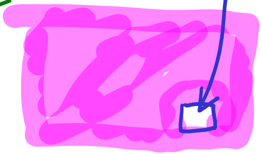
Informal arguments (1)

- Let's try to argue the Algorithm `CheckPrime2` works correctly.
- Note that if n is a prime number, the algorithm answers correctly. (Why?)
- Therefore, let's consider the case when n is not prime (i.e., n is a composite).
- If that's the case, n has some positive divisor which is not 1 or n. Let's call this number a.
- Now, if $\boxed{2 \leq a \leq \sqrt{n},}$ at some point during the execution of the algorithm, $i = a$ and i should divides n; thus the algorithm correctly returns `False`. ✓

Informal arguments (1)

- Let's try to argue the Algorithm `CheckPrime2` works correctly.
- Note that if n is a prime number, the algorithm answers correctly. (Why?)
- Therefore, let's consider the case when n is not prime (i.e., n is a composite).
- If that's the case, n has some positive divisor which is not 1 or n. Let's call this number a.
- Now, if $2 \leq a \leq \sqrt{n}$, at some point during the execution of the algorithm, $i = a$ and i should divides n; thus the algorithm correctly returns `False`. ✓
- Are we done?

hole



Informal arguments (2)

ex: $10 = 5 \times \boxed{2}$

$2, 3, \cancel{4}$

ABSTRACT

$\updownarrow$

CONCRETE

- Recall that we are left with the case that (1) n is not prime and (2) its positive divisor a is larger than $\sqrt{n}$.

Informal arguments (2)

- Recall that we are left with the case that (1) n is not prime and (2) its positive divisor a is larger than $\sqrt{n}$.
- Let $b = n/a$. Since n and a are positive integers and a divides n, b is also a positive integer.

Informal arguments (2)

- Recall that we are left with the case that (1) n is not prime and (2) its positive divisor a is larger than $\sqrt{n}$.
- Let $b = n/a$. Since n and a are positive integers and a divides n, b is also a positive integer.
- Note that if we can argue that $\boxed{2 \le b \le \sqrt{n},}$ we are done. (why?)

ถ้า n เป็น composite

$\Rightarrow$ จะมี a ที่หาร n

$\rightarrow$ ถ้า $a \le \sqrt{n}$ ✓

$\rightarrow$ มี $b = n/a$ ที่ $b \le \sqrt{n}$

และ: b สามารถ หาร ลงตัว

Informal arguments (2)

- Recall that we are left with the case that (1) n is not prime and (2) its positive divisor a is larger than $\sqrt{n}$.
- Let $b = n/a$. Since n and a are positive integers and a divides n, b is also a positive integer.
- Note that if we can argue that $2 \leq b \leq \sqrt{n}$, we are done. (why?)
- How can we do that?

The goals

- Let's take a break and look back at what we are trying to do.

The goals

- Let's take a break and look back at what we are trying to do.

[illegible]
$2, 3, \ldots \sqrt{n}$

> **Original goal:** To show that Algorithm `CheckPrime2` is correct.

The goals

- Let's take a break and look back at what we are trying to do.

Original goal: To show that Algorithm `CheckPrime2` is correct.

Current (sub) goal: Consider a positive composite n and its positive divisor a, where $a > \sqrt{n}$. Let $b = n/a$. We want to show that $2 \leq b \leq \sqrt{n}$.

The goals

- Let's take a break and look back at what we are trying to do.

> **Original goal:** To show that Algorithm `CheckPrime2` is correct.

> **Current (sub) goal:** Consider a positive composite n and its positive divisor a, where $a > \sqrt{n}$. Let $b = n/a$. We want to show that $2 \leq b \leq \sqrt{n}$.

- Before we continue, I'd like to add a bit of formalism to our thinking process.

The main goal

- **Original goal:** To show that Algorithm `CheckPrime2` is correct.

The main goal

- **Original goal:** To show that Algorithm `CheckPrime2` is correct.
- Let's focus on the statement we want to argue for:

> **"Algorithm `CheckPrime2` is correct."**

The main goal

- **Original goal:** To show that Algorithm `CheckPrime2` is correct.
- Let's focus on the statement we want to argue for:

> **"Algorithm `CheckPrime2` is correct."**

- Note that this statement can either be "true" or "false." If we can demonstrate, using logical/mathematical arguments that this statement is true, we can say that we **prove** the statement.

พิสูจน์

The (sub) goal

- **Current (sub) goal:** Consider a positive composite n and its positive divisor a, where $a > \sqrt{n}$. Let $b = n/a$. We want to show that $2 \leq b \leq \sqrt{n}$.

The (sub) goal

- **Current (sub) goal:** Consider a positive composite n and its positive divisor a, where $a > \sqrt{n}$. Let $b = n/a$. We want to show that $2 \leq b \leq \sqrt{n}$.
- Let's focus only on the statement we want to argue for:

The (sub) goal

- **Current (sub) goal:** Consider a positive composite n and its positive divisor a, where $a > \sqrt{n}$. Let $b = n/a$. We want to show that $2 \leq b \leq \sqrt{n}$.
- Let's focus only on the statement we want to argue for:

$$2 \leq b \leq \sqrt{n}.$$

The (sub) goal

- **Current (sub) goal:** Consider a positive composite n and its positive divisor a, where $a > \sqrt{n}$. Let $b = n/a$. We want to show that $2 \leq b \leq \sqrt{n}$.
- Let's focus only on the statement we want to argue for:

$$2 \leq b \leq \sqrt{n}.$$

- If we only look at this statement, it is unclear if the statement is true or false because there are variables b and n in the statement. It can be true in some case and it can be false in some case depending on the values of n and b.

The (sub) goal

- **Current (sub) goal:** Consider a positive composite n and its positive divisor a, where $a > \sqrt{n}$. Let $b = n/a$. We want to show that $2 \leq b \leq \sqrt{n}$.
- Let's focus only on the statement we want to argue for:

$$2 \leq b \leq \sqrt{n}.$$

- If we only look at this statement, it is unclear if the statement is true or false because there are variables b and n in the statement. It can be true in some case and it can be false in some case depending on the values of n and b.
- Are we doom?

The (sub) goal

- **Current (sub) goal:** Consider a positive composite n and its positive divisor a, where $a > \sqrt{n}$. Let $b = n/a$. We want to show that $2 \leq b \leq \sqrt{n}$.
- Let's focus only on the statement we want to argue for:

$$\text{``}\; 2 \leq b \leq \sqrt{n}. \;\text{''}$$

- If we only look at this statement, it is unclear if the statement is true or false because there are variables b and n in the statement. It can be true in some case and it can be false in some case depending on the values of n and b.
- Are we doom? Not really. The statement above is not precisely the statement we want to prove.

The (sub) goal (second try)

- **Current (sub) goal:** Consider a positive composite n and its positive divisor a, where $a > \sqrt{n}$. Let $b = n/a$. We want to show that $2 \leq b \leq \sqrt{n}$.
- We can be more specific about what values of n and b that we want to consider.

The (sub) goal (second try)

- **Current (sub) goal:** Consider a positive composite n and its positive divisor a, where $a > \sqrt{n}$. Let $b = n/a$. We want to show that $2 \le b \le \sqrt{n}$.
- We can be more specific about what values of n and b that we want to consider.

Revised statement

For all positive composite integer n, and for every divisor a of n such that $\sqrt{n} < a < n$,

bound

"$\forall n$"

$$2 \le b \le \sqrt{n},$$

free

where $\boxed{b = n/a.}$

- Note that this revised statement is now "quantified," that is, every variable in the statement has specific scope. Now the statement is either true or false.

Propositions[1] ประพจน์

- A *proposition* is a statement which is either **true** or **false**.

[1] This section follows the expositions in Berkeley's CS70 lecture notes.

Propositions[1]

- A *proposition* is a statement which is either **true** or **false**.
- It is our basic unit of mathematical "facts".
- Examples:
  - Algorithm `CheckPrime2` is correct.
  - $10^2 = 90$.
  - $\sqrt{2}$ is irrational.

[1]This section follows the expositions in Berkeley's CS70 lecture notes.

Propositions[1]

- A *proposition* is a statement which is either **true** or **false**.
- It is our basic unit of mathematical "facts".
- Examples:
  - Algorithm `CheckPrime2` is correct.
  - $10^2 = 90$.
  - $\sqrt{2}$ is irrational.
- Examples of statements which are not propositions (why?):
  - $x > 10$. ← x คืออะไร ?
  - $1 + 2 + \cdots + 10$. ← ไม่เป็น true/false
  - This algorithm is fast. ← ขึ้นกับค่าที่เร็ว
  - Run, run quickly.

↑ คำสั่ง

[1] This section follows the expositions in Berkeley's CS70 lecture notes.

Combining propositions

- We usually use a variable to refer to a proposition. For example, we may use P to stand for "it rains" or Q to stand for "the road is wet."

Combining propositions

- We usually use a variable to refer to a proposition. For example, we may use P to stand for "it rains" or Q to stand for "the road is wet."
- The truth value of a variable is the truth value of the proposition it stands for.

Combining propositions

- We usually use a variable to refer to a proposition. For example, we may use P to stand for "it rains" or Q to stand for "the road is wet."
- The truth value of a variable is the truth value of the proposition it stands for.
- Many propositions can be combined to get a complex statement using logical operators.

Combining propositions

- We usually use a variable to refer to a proposition. For example, we may use P to stand for "it rains" or Q to stand for "the road is wet."
- The truth value of a variable is the truth value of the proposition it stands for.
- Many propositions can be combined to get a complex statement using logical operators.
- For example, we can join P and Q with "and" (denoted by "$\wedge$") and get

$$P \wedge Q,$$

which stands for "it rains and the road is wet".

Combining propositions

- We usually use a variable to refer to a proposition. For example, we may use P to stand for "it rains" or Q to stand for "the road is wet."
- The truth value of a variable is the truth value of the proposition it stands for.
- Many propositions can be combined to get a complex statement using logical operators.
- For example, we can join P and Q with "and" (denoted by "$\wedge$") and get
$$P \wedge Q,$$
which stands for "it rains and the road is wet".
- An expression $P \wedge Q$ is an example of *propositional forms.* The logical value of a propositional form "usually" depends on the truth value of its variables.

Connectives: "and", "or", "not"

Given propositions P and Q, we can use connectives to form more complex propositions:

- **Conjunction:** $P \wedge Q$ ("P and Q"), (True when both P and Q are true)
- **Disjunction:** $P \vee Q$ ("P or Q"), (True when at least one of P and Q is true)
- **Negation:** $\boxed{\neg P}$ ("not P") (True only when P is false)

$\sim P$

$\boxed{\neg P}$

เข้มงวด

Connectives: "and", "or", "not"

Given propositions P and Q, we can use connectives to form more complex propositions:

- **Conjunction:** $P \wedge Q$ ("P and Q"),
  (True when both P and Q are true)
- **Disjunction:** $P \vee Q$ ("P or Q"),
  (True when at least one of P and Q is true)
- **Negation:** $\neg P$ ("not P")
  (True only when P is false)

If P stands for "today is Tuesday" and Q stands for "dogs are animals", then

- $P \wedge Q$ stands for "today is Tuesday and dogs are animals",
- $P \vee Q$ stands for "today is Tuesday or dogs are animals", and
- $\neg P$ stands for "today is not Tuesday".

Truth tables

To represents values of propositional forms, we usually use truth tables.

And/Or/Not

P	Q	$P \wedge Q$	$P \vee Q$	$\neg P$
T	T	T	T	F
T	F	F	T	
F	T	F	T	T
F	F	F	F	

5 ตัวแปร → 32 แถว

Quick check 1

For each of these statements, define propositional variables representing each proposition inside the statement and write the proposition form of the statement.

- All prime numbers are larger than 0 and all natural numbers is at least one.
- You are smart or you won't be taking this class.

P

Q

$P \wedge Q$

S

C "you take this class"

$S \vee (\neg C)$

Next lecture...

if
only-if

- We will discuss other ways to join two propositions, i.e., implications ($\Rightarrow$) and equivalences ($\Leftrightarrow$). iff
- We will look at two forms of quantifiers: universal quantifiers $\forall$ and existential quantifiers. $\exists$