What is computation?
What is an algorithm?
How can we mathematically define them?

Inspirational quotation #1:
"An algorithm is a finite answer to an infinite number of questions"
Stephen Kleene

Inspirational quotation #2:
An algorithm solves a problem if it gives the correct solution on every instance.

We’ll define last 3 terms now. We’ll save algorithm for later.

Example problem 1: PRIMALITY

<table>
<thead>
<tr>
<th>Instance</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>1</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>No</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>42</td>
<td>No</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>251</td>
<td>Yes</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>170141183460469231731687303715884105727</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Example problem 2: **PALINDROME**

<table>
<thead>
<tr>
<th>Instance</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Yes</td>
</tr>
<tr>
<td>10101</td>
<td>Yes</td>
</tr>
<tr>
<td>selfless</td>
<td>No</td>
</tr>
<tr>
<td>dammitimnad</td>
<td>Yes</td>
</tr>
<tr>
<td>zxckallkdfsidsdfk</td>
<td>Yes</td>
</tr>
<tr>
<td>parahaziramarizaharap</td>
<td>Yes</td>
</tr>
</tbody>
</table>

These are examples of **decision problems**: Problems where the solution is **Yes / No**.

(Also known as **True / False**, **1 / 0**, **accept / reject**.)

Example problem 3: **MULTIPLICATION**

<table>
<thead>
<tr>
<th>Instance</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>3, 7</td>
<td>21</td>
</tr>
<tr>
<td>610, 25</td>
<td>15250</td>
</tr>
<tr>
<td>50, 610</td>
<td>30500</td>
</tr>
<tr>
<td>15251, 252</td>
<td>3843252</td>
</tr>
<tr>
<td>12345679, 9</td>
<td>111111111</td>
</tr>
</tbody>
</table>

Example problem 4: **SORTING**

<table>
<thead>
<tr>
<th>Instance</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>[vanilla, mind, Anil, yogurt, doesn’t]</td>
<td>[Anil, doesn’t, mind, vanilla, yogurt]</td>
</tr>
</tbody>
</table>

Representing problems

A **problem** is a collection of (naturally related) instances and solutions.

As you know... Can all be conveniently encoded by **strings**. Even just by **binary** (0/1) strings.
String notation

Alphabet: A nonempty finite set $\Sigma$ of symbols. $\Sigma = \{0,1\}$ is a popular choice.

String: A finite sequence of 0 or more symbols. The length-0 string is denoted $\epsilon$. $\Sigma^n$ means all strings over $\Sigma$ of length $n$. $\Sigma^*$ means all strings over $\Sigma$.

Language: A collection of strings. In other words any subset $L \subseteq \Sigma^*$.

Representing problems

We can encode instances/solutions with strings.

Thus we can think of a problem as a function

$$f : \Sigma^* \rightarrow \Sigma^*$$

mapping instances to solutions.

A decision problem can be thought of as

$$f : \Sigma^* \rightarrow \{\text{No, Yes}\}$$

Representing problems

A decision problem can be thought of as

$$f : \Sigma^* \rightarrow \{\text{No, Yes}\}$$

or equivalently as a language

$$L \subseteq \Sigma^*$$

$L = \{x \in \Sigma^* : f(x) = \text{Yes}\}$

$$f(x) = \begin{cases} \text{Yes} & \text{if } x \in L \\ \text{No} & \text{if } x \not\in L \end{cases}$$

E.g.: PALINDROME = $\{x \in \Sigma^* : x = \text{Reverse}(x)\}$

What is computation? What is an algorithm?

This lecture:

A very simple computational model: Deterministic Finite Automata

It’s so wimpy, it can only implement an extremely restricted kind of algorithm.

Has some interesting applications.

A good warmup before we study general models of computations next lecture.

Deterministic Finite Automata (DFAs)

A DFA over alphabet $\Sigma = \{0,1\}$ is something that looks like this:

DFA accepts its input if the process ends in a double-circle.

Anatomy of a DFA

states

accepting states

the start state

transition rule: the labeled arrows
Computing with DFAs

Let $M$ be a DFA, using alphabet $\Sigma$.

We think of $M$ as a computing mechanism, which "accepts" some strings in $\Sigma^*$ and "rejects" the others.

Definition: $L(M) = \{x \in \Sigma^*: M \text{ accepts } x\}$

Called the "language decided/accepted by $M$".

If $P$ is a decision problem,
we say $M$ solves it if $L(M) = P$.

What language does this DFA decide?
All binary strings with an even number of 1's.

What language does this DFA decide?
All binary strings with even length.

$L(M) = \{a, b, cb, cc\} \subseteq \{a, b, c\}^*$

Formal definition of DFAs

Let's give a very formal definition of DFAs.

Having some notations can help us reason about them.

Also illustrates that we can completely formalize this notion of computation.
Formal definition of DFAs

A deterministic finite automaton is a 5-tuple:
\[ M = (Q, \Sigma, \delta, q_0, F) \]

where
- \( Q \) is a nonempty finite set of states,
- \( \Sigma \) is an alphabet,
- \( \delta : Q \times \Sigma \to Q \) is the transition function,
- \( q_0 \in Q \) is the start state,
- \( F \subseteq Q \) is the set of accepting states.

DFA-construction practice:

Let \( w = w_1w_2w_3 \cdots w_n \), where each \( w_i \in \Sigma \).

We say that \( M \) accepts string \( w \) if:
- There exist states \( r_0, r_1, r_2, \ldots, r_n \in Q \) such that:
  - \( r_0 = q_0 \), the initial state;
  - \( \delta(r_{t-1}, w_t) = r_t \) for all \( t = 1, 2, 3, \ldots, n \);
  - \( r_n \in F \) (the accepting states).

Otherwise we say \( M \) rejects \( w \).

The sequence \( r_0, r_1, r_2, \ldots, r_n \) is called the computation trace.

Regular Languages

Definition:

A language \( L \subseteq \Sigma^* \) is regular if there is a DFA which decides it.

Questions:

1. Are all languages regular?
2. Are there other ways to tell if \( L \) is regular?
**Theorem:** $L = \{0^n 1^n : n \in \mathbb{N}\}$ is not regular

**Notation:**

If $s \in \Sigma$ is a symbol and $n \in \mathbb{N}$ then $s^n$ denotes the string $ssss\ldots$ (n times).

E.g., $s^3$ means $sss$, $s^5$ means $sssss$, $s^1$ means $s$, $s^0$ means $\epsilon$, etc.

Thus $L = \{\epsilon, 01, 0011, 000111, 00001111, \ldots\}$.

**Intuition:**

For a DFA to decide $L$, it seems like it needs to "remember" how many 0's it sees at the beginning of the string, so that it can "check" there are equally many 1's.

But a DFA has only finitely many states — shouldn't be able to handle arbitrary $n$.

But we need to be careful: the following language is regular:

$C = \{x \in \{0,1\}^* : 10$ and $01$ occur equally often in $x\}$

**Warmup:** No DFA with, say, 7 states can decide $L$.

Input: $00000000000011111111111111$


**Input:** $00$


**Input:** $000$

Theorem: \( L = \{0^n1^n : n \in \mathbb{N}\} \) is not regular

Warmup: No DFA with, say, 7 states can decide \( L \).

Input: \[ 0000000000001111111111111111 \]

Transitions? Accept states? Let’s keep it hazy.

Theorem: \( L = \{0^n1^n : n \in \mathbb{N}\} \) is not regular

Warmup: No DFA with, say, 7 states can decide \( L \).

Input: \[ 0000000000001111111111111111 \]

Transitions? Accept states? Let’s keep it hazy.

Theorem: \( L = \{0^n1^n : n \in \mathbb{N}\} \) is not regular

Warmup: No DFA with, say, 7 states can decide \( L \).

Input: \[ 0000000000001111111111111111 \]

Transitions? Accept states? Let’s keep it hazy.

Theorem: \( L = \{0^n1^n : n \in \mathbb{N}\} \) is not regular

Warmup: No DFA with, say, 7 states can decide \( L \).

Input: \[ 0000000000001111111111111111 \]

Transitions? Accept states? Let’s keep it hazy.

Theorem: \( L = \{0^n1^n : n \in \mathbb{N}\} \) is not regular

Warmup: No DFA with, say, 7 states can decide \( L \).

Input: \[ 0000000000001111111111111111 \]

DFA can’t tell the difference between starting 00 and starting 000000

Um, we were in this state before.

Theorem: \( L = \{0^n1^n : n \in \mathbb{N}\} \) is not regular

Warmup: No DFA with, say, 7 states can decide \( L \).

Input: \[ 0000000000001111111111111111 \]

DFA can’t tell the difference between starting 00 and starting 000000
**Theorem:** \( L = \{0^n1^n : n \in \mathbb{N}\} \) is not regular

**Warmup:** No DFA with, say, 7 states can decide \( L \).

Input: \( 0000000 \)

DFA can’t tell the difference between starting 00 and starting 0000000

**Theorem:** \( L = \{0^n1^n : n \in \mathbb{N}\} \) is not regular

**Proving a language \( L \) is not regular**

*Most* of the time, the proof looks like this:

1. Assume for contradiction there is a DFA \( M \) which decides language \( L \).
2. Argue (usually by Pigeonhole) there are two strings \( x \) and \( y \) which reach the same state in \( M \).
3. Show there is a string \( z \) such that \( xz \in L \) but \( yz \notin L \). Contradiction, since \( M \) acts the same (accept/reject) on both.

---

**Regular Languages**

**Definition:**

A language \( L \subseteq \Sigma^* \) is **regular** if there is a DFA which decides it.

**Questions:**

1. Are all languages regular?
2. Are there other ways to tell if \( L \) is regular?

---

**Union Theorem**

**Definition:**

Let \( L_1 \) and \( L_2 \) be any languages. Their union, \( L_1 \cup L_2 \), is \( \{x : x \in L_1 \text{ or } x \in L_2\} \).

**Union Theorem:**

If \( L_1 \) and \( L_2 \) are both regular languages over \( \Sigma \) then so is \( L_1 \cup L_2 \).
Union Theorem

E.g.:
\(L_1\) = strings with even # of 1's
\(L_2\) = strings with length divisible by 3

\[M_1\]
\[M_2\]

Input: 101001

Union Theorem

Input: 101001

Union Theorem

Input: 101001

Union Theorem

Input: 101001

Union Theorem

Input: 101001

Union Theorem

Input: 101001
Union Theorem

Input: 101001

Accept.

Make a DFA keeping track of both at once.
Suppose $L_1$ is decided by $M_1 = (Q, \Sigma, \delta, q_0, F)$.
Suppose $L_2$ is decided by $M_2 = (Q', \Sigma, \delta', q_0', F')$.
Define the DFA $M = (Q \times Q', \Sigma, \beta, (q_0, q_0'), G)$,
where $G = \{(q, q') : q \in F \text{ or } q' \in F'\}$
and $\beta((q, q'), c) = (\delta(q, c), \delta'(q', c))$.

Then... (it's not hard to see that)... $L(M) = L_1 \cup L_2$. 
More “closure” theorems

Theorem: $L_1 \cup L_2$ is regular if $L_1$, $L_2$ are.

“Concatenation”: $L_1 \cdot L_2 = \{xy : x \in L_1, y \in L_2\}$

Theorem: $L_1 \cdot L_2$ is regular if $L_1$, $L_2$ are.

“Star”: $L^* = \{x_1x_2\cdots x_k : k \geq 0, \text{ each } x_i \in L\}$

Theorem: $L^*$ is regular if $L$ is.

The Regular Operations

Theorem: $L_1 \cup L_2$ is regular if $L_1$, $L_2$ are.

Theorem: $L_1 \cdot L_2$ is regular if $L_1$, $L_2$ are.

Theorem: $L^*$ is regular if $L$ is.

The latter two theorems are somewhat more tricky to prove.

You will prove them on the homework!

A Deductive System for regular languages

Objects: Languages over alphabet $\Sigma$

Initial objects: $\emptyset$, $\{a\}$ for each $a \in \Sigma$

Deduction rules:
- From $L_1$, $L_2$, can deduce $L_1 \cup L_2$
- From $L_1$, $L_2$, can deduce $L_1 \cdot L_2$
- From $L$, can deduce $L^*$

From the previous slide, we know that any deducible language is regular.

A Deductive System for regular languages

Objects: Languages over alphabet $\Sigma$

Initial objects: $\emptyset$, $\{a\}$ for each $a \in \Sigma$

Deduction rules:
- From $L_1$, $L_2$, can deduce $L_1 \cup L_2$
- From $L_1$, $L_2$, can deduce $L_1 \cdot L_2$
- From $L$, can deduce $L^*$

Fact: Every regular language is deducible.
I.e., if $\exists$ a DFA deciding $L$, then you can deduce $L$.
Proving this fact is also a little tricky.

Regular Expressions

A regular expression over $\Sigma$ (say, $\{a,b\}$)
is something that looks like this:

$$a(aUb)^*a \cup b(aUb)^*b \cup a \cup b$$

It is a syntactic representation of the deduction of a regular language in the Deductive System.

Also stands for the deduced language; e.g., the regular expression above stands for $\{x \in \{a,b\}^* : x \text{ starts \& ends with same char}\}$.

Regular Expressions

Commonly used in string searching (e.g., grep).

You’ll also see some shorthands in practice:

- $|$ instead of $\cup$
- $R^*$ for $RR^*$
- $\varepsilon$ for $\emptyset^*$
- $\Sigma$ or $\cdot$ for the union of all single characters
- $R^n$ for $RR\cdots R$ (n times)
- $R?\ v e r i f y$ (R|\varepsilon) and more...
String Searching

The simplest string searching problem:

Instance: Text $T$, length $n$. Substring $w$, length $k$.
Solution: Yes/No: Does $w$ occur in $T$?

Naive algorithm:

$$T = [a_1, a_2, a_3, a_4, a_5, \ldots, a_n]$$

Running time: about $nk$ steps

Finite automata — to the max

Finite automata were first studied in the 1940’s in the context of neurophysiology.

Finite automata — to the max

1959: DFAs codified & this lecture’s results proved by Michael Rabin & Dana Scott

Knuth–Morris–Pratt ‘77: # steps $\sim k$

Pittsburgh native, CMU bachelor’s, CMU professor.

CMU prof. emeritus
Finite automata — to the max

Rabin & Scott also invented DFAs with certain “magical superpowers” which you’ll investigate on the homework.

Actually, they showed that adding these superpowers does not increase the set of languages accepted by DFAs.

For this they won the Turing Award.

Finite automata — to the max

A further generalization of DFAs: “nondeterministic pushdown automata”.

These decide the “context-free languages”.

A further further generalization: “linear bounded automata”.
These decide the “context-sensitive languages”.

Finite automata — to the max

A further further further generalization: “Turing Machines”.

These decide the “decidable languages”.

We discuss them in the next lecture!

Definitions:
Problems, instances, strings, languages.
DFAs.
Regular operations.
Regular expressions.

Theorem/proof:
$0^n1^n$ is not regular.
Union Theorem.

Practice:
Building/analyzing DFAs.