Recap

What we saw last time..

- (Deterministic) Turing machines
- Decision problems
- Polynomial time and the class P
What will we do today?

- The universal Turing machine
- Nondeterministic Turing machines
- More complexity classes: $\text{EXP}$, $\text{NP}$, $\text{coNP}$
- Polynomial-time reductions
- $\text{NP}$-hardness and $\text{NP}$-completeness
We can encode Turing machines into binary strings, such that:

1. each string $s \in \{0, 1\}^*$ represents some Turing machine $M$

2. each Turing machine $M$ is represented by infinitely many strings $s \in \{0, 1\}^*$

3. given a TM $M$, we can efficiently compute a string $s$ that represents $M$

**Idea:**

- Write out the tuple $(\Gamma, Q, \delta)$, together with starting and halting states, in an appropriate alphabet, and then encode into binary

- Allow padding (cf. comments in programming languages)
Proposition

There exists a TM $U$ such that for every $x, s \in \{0, 1\}^*$ it holds that $U(x, s) = M_s(x)$, where $M_s$ is the TM represented by the string $s$.

Moreover, if $M_s$ halts on $x$ in time $T$, then $U(x, s)$ halts in time $C \cdot T \log T$, where $C$ depends only on $s$ (and not on $x$).

- $U$ is an efficient universal Turing machine: it can simulate other TMs in an efficient way.
(In)tractability

- **Tractability**: there exists a polynomial-time algorithm that solves the problem

- **Intractability**: there exists no polynomial-time algorithm that solves the problem

  (or sometimes: all algorithms that solve the problem take exponential time, in the worst case)

- How do we find out which of these two is the case for—for example—the problem of 3-coloring?
"I can’t find an efficient algorithm, I guess I’m just too dumb."
"I can’t find an efficient algorithm, because no such algorithm is possible!"
Showing intractability: using NP-completeness

“I can’t find an efficient algorithm, but neither can all these famous people.”
Definition (DTIME)

Let $T : \mathbb{N} \to \mathbb{N}$ be a function. A language $L \subseteq \Sigma^*$ is in $\text{DTIME}(T(n))$ if there exists a Turing machine that decides $L$ and that runs in time $O(T(n))$.

Definition (the complexity classes $P$ and $\text{EXP}$)

$$P = \bigcup_{c \geq 1} \text{DTIME}(n^c)$$
$$\text{EXP} = \bigcup_{c \geq 1} \text{DTIME}(2^{n^c})$$
The complexity class NP

Definition (the complexity class NP)

A problem \( L \subseteq \Sigma^* \) is in the complexity class \( NP \) if there is a polynomial \( p : \mathbb{N} \rightarrow \mathbb{N} \) and a polynomial-time Turing machine \( \mathbb{M} \) (the verifier) such that for every \( x \in \Sigma^* \):

\[
x \in L \quad \text{if and only if} \quad \text{there exists some } u \in \{0, 1\}^{p(|x|)} \text{ such that } \mathbb{M}(x, u) = 1.
\]

The string \( u \in \{0, 1\}^{p(|x|)} \) is called a certificate for \( x \) if \( \mathbb{M}(x, u) = 1 \).
Let’s see why the (decision) problem of 3-coloring is in NP.

Let $G = (V, E)$ be a graph with $m$ nodes.

Consider as witness a binary string $u$ of length $2m$, where the coloring of each node $i$ is given by the $i$’th pair of bits—say, 01 for red, 10 for green, and 11 for blue.

Given $G$ and $u$, we can check in polynomial time if the coloring given by $u$ is proper.
A nondeterministic Turing machine (NTM) $M$ is a variant of a (deterministic) Turing machine, where some things are modified.

- Instead of a single transition function $\delta$, there are two transition functions $\delta_1, \delta_2$.
- At each step, one of $\delta_1, \delta_2$ is chosen nondeterministically to determine the next configuration.
- (As halting states, it has an accept state $q_{acc}$ and a reject state $q_{rej}$.)

- We write $M(x) = 1$ if there is some sequence of nondeterministic choices such that $M$ reaches the state $q_{acc}$ on input $x$.
- The machine $M$ runs in time $T(n)$ if for every input $x$ and every sequence of nondeterministic choices, $M$ halts within $T(|x|)$ steps.
Definition (NTIME)

Let \( T : \mathbb{N} \rightarrow \mathbb{N} \) be a function. A problem \( L \subseteq \Sigma^* \) is in \( \text{NTIME}(T(n)) \) if there exists a nondeterministic Turing machine that decides \( L \) and that runs in time \( O(T(n)) \).

Proposition (characterization of NP)

\[
\text{NP} = \bigcup_{c \geq 1} \text{NTIME}(n^c)
\]
The complexity class \(\text{coNP}\)

**Definition (the complexity class \(\text{coNP}\))**

A problem \(L \subseteq \Sigma^*\) is in \(\text{coNP}\) if \(\overline{L} \in \text{NP}\), where \(\overline{L} = \{ x \in \Sigma^* \mid x \not\in L \}\).

**Proposition (verifier characterization of \(\text{coNP}\))**

A problem \(L \subseteq \Sigma^*\) is in \(\text{coNP}\) if there is a polynomial \(p : \mathbb{N} \rightarrow \mathbb{N}\) and a polynomial-time Turing machine \(M\) (the *verifier*) such that for every \(x \in \Sigma^*\):

\[
x \in L \iff \text{for all } u \in \{0, 1\}^{p(|x|)} \text{ it holds that } M(x, u) = 1.
\]
Proposition

\( \text{NP} \subseteq \text{EXP}. \)

Proof (idea).

- Iterate over all possible witnesses \( u \in \{0, 1\}^{p(|x|)} \), and check if \( M(x, u) = 1 \).
- If for any \( u \) this is the case, return 1—otherwise, return 0.
- There are \( 2^{p(|x|)} \) such strings \( u \), and so this takes time \( 2^{p(|x|)} \cdot q(|x|) \), for some polynomial \( q \).
An overview of complexity classes

(That we’ve seen so far..)
Definition (polynomial-time reductions)

A problem $L_1 \subseteq \Sigma^*$ is polynomial-time reducible to a problem $L_2 \subseteq \Sigma^*$ if there is a polynomial-time computable function $f : \Sigma^* \rightarrow \Sigma^*$ (the reduction) such that for every $x \in \Sigma^*$ it holds that:

$$x \in L_1 \text{ if and only if } f(x) \in L_2.$$

We write $L_1 \leq_p L_2$ to indicate that $L_1$ is polynomial-time reducible to $L_2$. 

\[ x \rightarrow f(x) \]
\[ x \in L_1? \iff f(x) \in L_2? \]
**Definition (NP-hardness)**

A problem $L \subseteq \Sigma^*$ is **NP-hard** if every problem in NP is polynomial-time reducible to $L$.

**Definition (NP-completeness)**

A problem $L \subseteq \Sigma^*$ is **NP-complete** if $L \in$ NP and $L$ is NP-hard.
Some properties

**Proposition**

Polynomial-time reductions are transitive. That is, if $L_1 \leq_p L_2$ and $L_2 \leq_p L_3$, then $L_1 \leq_p L_3$.

**Proposition**

Take two problems $L_1, L_2 \subseteq \Sigma^*$. If $L_1$ is polynomial-time reducible to $L_2$ and $L_2 \in P$, then $L_1 \in P$. 
Proposition

Take an NP-complete problem $L \subseteq \Sigma^*$. If $L \in P$, then $P = NP$. In other words, assuming that $P \neq NP$, $L \notin P$.

Proof.

Since deterministic TMs can be seen also as nondeterministic TMs, we get $P \subseteq NP$.

We show that if $L \in P$, then $NP \subseteq P$.

(1) Take an arbitrary problem $M \in NP$.

(2) Since $L$ is NP-complete, $M \leq^p L$.

(3) Since $L \in P$, then also $M \in P$.

Since $M$ was arbitrary, we know that $NP \subseteq P$. \qed
Showing intractability: using NP-completeness

“I can’t find an efficient algorithm, but neither can all these famous people.”
Recap

- The universal Turing machine
- Nondeterministic Turing machines
- More complexity classes: EXP, NP, coNP
- Polynomial-time reductions
- NP-hardness and NP-completeness
Next time

- Proving that NP-complete problems exist :-)