THIS TIME

- Complexity class $\mathbf{NP}$
- Oracles, certificates, polytime verification algorithms
- Two problems in $\mathbf{NP}$
  - Subset sum
  - Hamiltonian Cycle
- Relationship between $\mathbf{P}$ and $\mathbf{NP}$
- Polynomial transformations

---

**EXAMPLE: SUBSET-SUM PROBLEM**

- Suppose we are given some integers, $-7, -3, -2, 5, 8$
- Does some subset of these sum to zero?
  - In this case, yes: $(-3) + (-2) + 5 = 0$

Finding such a subset can be extremely difficult!

Of course, I might lie and give you a subset that does not sum to zero...

I could even give you numbers that are not in the input...

Can you determine whether I am lying in polynomial time?

---

**SUBSET-SUM VIA NON-DETERMINISTIC ORACLE**

- Suppose there is a non-deterministic oracle, which returns a subset that sums to 0 if one exists and otherwise can return anything (even garbage)
- We call the oracle’s output a certificate
- Given a certificate, can you verify in polytime whether it describes a solution to the problem?

Given such an oracle, this algorithm would solve subset-sum

Else, return false

Test whether $C$ is a subset of $I$

Otherwise, either $C$ is not a subset of the input (return false), or $C$ sums to a non-zero value (return false)

If there exists a subset that sums to 0, then $C$ is one such subset, and we return true

Here "non-deterministic" just means the oracle is magically guaranteed to return a yes-certificate if one exists

---

**SUBSET-SUM VIA NON-DETERMINISTIC ORACLE**

- Suppose there is a non-deterministic oracle, which returns a subset that sums to 0 if one exists and otherwise can return anything (even garbage)
- We call the oracle’s output a certificate
- Given a certificate, can you verify in polytime whether it describes a solution to the problem?

Given a certificate from the oracle, would you verify it solves the problem in polytime?

Test whether $C$ is a subset of $I$

Return whether $C$ sums to 0

For loop with $|I| + |C|$ time...
DUMB SUBSET-SUM ALGORITHM: PRETEND YOU’RE AN ORACLE AND MAKE CERTS.

1. Subsetsum \(X, \text{sum}\) \(\rightarrow\) for every possible subset \(S\) of \(X\)
2. \(S\) \(\rightarrow\) if \(X\) does not divide \(\text{sum}\) then return false.
3. Generate every subset certificate \(S\).
4. Verify certificate \(S\).

(Valid + sums to zero)

Generating these certificates is expensive, exponential time!

But verifying one certificate is fast; runtime is \(\mathcal{P}(S)\).

If any certificate \(S\) sums to zero, it is a yes-certificate (a proof that the answer to the decision problem is “true”), and we return true.

A certificates that does not sum to zero doesn’t really prove anything (would need to know that all certificates sum to non-zero).

If there was such a thing as a no-certificate, what would it look like? How long would it take to verify it?

GENERALIZING BEYOND SUBSET-SUM

• You can solve any decision problem in non-deterministic poly-time, given:
  1. a poly-time non-deterministic oracle, and
  2. a poly-time verify algorithm

• Such that:
  • If \(I\) is a yes-instance, then the oracle returns a yes-certificate \(C\) (i.e., a “proof” the answer is “yes”) and verify\((I, C)\) returns true.
  • If \(I\) is a no-instance, then verify\((I, C)\) returns false for all \(C\) (i.e., it must be impossible to fool verify into returning true).

The algorithm:

DEFINING NP

Intuition: For a yes-instance, there must exist some certificate that verify would accept. If one exists, the oracle would find it, solving the problem. For a no-instance, verify must always reject.

A decision problem \(I\) is solved by a poly-time verify algorithm if:
• for every yes-instance \(I\), there exists a certificate \(C\) such that verify\((I, C)\) returns true, and
• for every no-instance \(I\), verify\((I, C)\) returns false for every \(C\).

The complexity class \(\mathcal{NP}\) denotes the set of all decision problems that can be solved by poly-time verify algorithms.

• No oracle needed! Note it is not necessary for an oracle to actually exist for a problem to be in \(\mathcal{NP}\). We can simply assume certificates come from an oracle, and show a poly-time verify algorithm exists.
MECHANICS OF SHOWING A PROBLEM IS IN NP

1. Define a yes-instance.
2. Design a polytime verify(I,C) algorithm.
3. Correctness proof:
   - Case 1: Let I be any yes-instance.
     - Find C such that verify(I,C) = true.
   - Case 2: Let I be any no-instance.
     - C be any certificate.
     - Prove verify(I,C) = false.

EXAMPLE: SHOWING "HAMILTONIAN CYCLE" IS IN NP

How to verify a certificate:

If C is a yes-instance of the problem, then must show there exists a possible certificate X for which this procedure returns true.

What would such a certificate look like?

Yes-instance implies there is a Hamiltonian cycle, suppose X is a sequence of n consecutive nodes on that cycle. Then we return true!

If G is a no-instance of the problem, then every possible certificate should cause verify to return false.

Easier to prove the contrapositive.

If we return true, then the graph contains a cycle with n distinct nodes... So G is a yes-instance.

So, Hamiltonian Cycle is in NP

HOW ARE P AND NP RELATED?

P ⊆ NP

- Consider a problem I ∈ P
  - We show there exists a poly-time verify(I,C) such that:
    - For every yes-instance i of I, verify(I,C) = true for some C
    - For every no-instance i of I, verify(I,C) = false for all C
  - By definition, there is a poly-time algorithm A to solve I
    - Implement verify(I,C) by simply running A(I) [ignoring C]
    - Regardless of what C is, verify(I,C) satisfies the above
- How about NP ⊆ P?
  - Million dollar question. We think not!

POLYNOMIAL TRANSFORMATIONS

A subclass of poly-time reductions commonly used for NP-completeness and impossibility results
POLYNOMIAL TRANSFORMATIONS

For a decision problem \( \Pi_1 \), let \( \mathcal{I}(\Pi) \) denote the set of all instances of \( \Pi \). Let \( \mathcal{I}_{\text{yes}}(\Pi_1) \) and \( \mathcal{I}_{\text{no}}(\Pi_1) \) denote the set of all yes-instances and no-instances (respectively) of \( \Pi_1 \).

Suppose that \( \Pi_1 \) and \( \Pi_2 \) are decision problems. We say that there is a polynomial transformation from \( \Pi_1 \) to \( \Pi_2 \) (denoted \( \Pi_1 \leq_{\text{poly}} \Pi_2 \)) if there exists a function \( f: \mathcal{I}(\Pi_1) \to \mathcal{I}(\Pi_2) \) such that the following properties are satisfied:

1. \( f(\cdot) \) is computable in polynomial time (as a function of size \( |\cdot| \)), where \( \ell \in \mathcal{I}(\Pi_1) \)
2. \( f(\ell) \in \mathcal{I}_{\text{yes}}(\Pi_2) \) if \( \ell \in \mathcal{I}_{\text{yes}}(\Pi_1) \)
3. \( f(\ell) \in \mathcal{I}_{\text{no}}(\Pi_2) \) if \( \ell \in \mathcal{I}_{\text{no}}(\Pi_1) \)

This is the contrapositive. Was previously (2 slides ago): If \( \ell \in \mathcal{I}_{\text{no}}(\Pi_1) \) then \( f(\ell) \in \mathcal{I}_{\text{no}}(\Pi_2) \)

This property justifies correctness for the following generic poly-time Karp reduction:

\[
\begin{align*}
\text{P1toP2KarpReduction}(\ell) &= f(\ell) \\
\text{return OracleForP2}(f(\ell))
\end{align*}
\]

SUMMARIZING THE MORE CONVENIENT DEFINITION

- Let \( \Pi_1 \) and \( \Pi_2 \) be decision problems
- \( \Pi_1 \leq_{\text{poly}} \Pi_2 \) if there exists \( f: \mathcal{I}(\Pi_1) \to \mathcal{I}(\Pi_2) \) such that:
  1. \( f(\cdot) \) is computable in poly-time, for all \( \ell \in \mathcal{I}(\Pi_1) \)
  2. \( f(\ell) \in \mathcal{I}_{\text{yes}}(\Pi_2) \) if \( \ell \in \mathcal{I}_{\text{yes}}(\Pi_1) \)
  3. \( f(\ell) \in \mathcal{I}_{\text{no}}(\Pi_2) \) if \( \ell \in \mathcal{I}_{\text{no}}(\Pi_1) \)

This is the same as saying

\[
\begin{align*}
\text{if } \ell \in \mathcal{I}_{\text{no}}(\Pi_1) & \Rightarrow f(\ell) \in \mathcal{I}_{\text{no}}(\Pi_2) \text{ then } f(\ell) \in \mathcal{I}_{\text{no}}(\Pi_2) \\
\text{if } f(\ell) \in \mathcal{I}_{\text{yes}}(\Pi_2) & \Rightarrow \ell \in \mathcal{I}_{\text{yes}}(\Pi_1) \text{ then } \ell \in \mathcal{I}_{\text{yes}}(\Pi_1)
\end{align*}
\]

Also known as Karp reductions and many-one reductions.

Example Polynomial Transformation

- We denote Clique by \( C_l \) and Vertex-Cover by \( V_C \)
- \( C_l \leq_{\text{poly}} V_C \) if there exists \( f: \mathcal{I}(C_l) \to \mathcal{I}(V_C) \) such that:
  1. \( f(\cdot) \) is computable in poly-time, for all \( \ell \in \mathcal{I}(C_l) \)
  2. \( f(\ell) \in \mathcal{I}_{\text{yes}}(V_C) \) if \( \ell \in \mathcal{I}_{\text{yes}}(C_l) \)
  3. \( f(\ell) \in \mathcal{I}_{\text{no}}(V_C) \) if \( \ell \in \mathcal{I}_{\text{no}}(C_l) \)

First let's show this

- Given an adjacency matrix for \( G \), get \( f(\cdot) \) by flipping 0's and 1's

\[
\begin{align*}
\text{f}(\cdot) &= \text{flip}\{0,1\} \\
\text{return OracleForV2}(f(\cdot))
\end{align*}
\]
Suppose $I = (\mathcal{G}, k)$ is an instance of Clique where $\mathcal{G} = (V, \mathcal{E})$, $V = \{v_1, \ldots, v_n\}$ and $1 \leq k \leq n$.

Therefore, we denote Clique by $\mathcal{C}_L$ and Vertex $\mathcal{V}_C$ to obtain a contradiction.

- Construct instance $f(I) = (G, n - k)$ of Vertex-Cover, where $G = (V, E)$ and $v_i, v_j \in E \Rightarrow v_i, v_j \notin B$.

**COMPLEXITY OF THE TRANSFORMATION**
- Want to solve Clique in poly time.
- Therefore, we need a polynomial transformation.
- Idea: reduce to Vertex-Cover $(G, n-k)$.

**PROVING THIS IS A POLYNOMIAL TRANSFORMATION**
- We denote Clique by $CL$ and Vertex-Cover by VC.
- $CL \leq VC$ there exists $f : CL \rightarrow VC$ such that:
  - $f(I)$ is computable in poly-time, for all $I \in CL$.
  - If $I \in \text{yes}(CL)$ then $f(I) \in \text{yes}(VC)$.
  - If $f(I) \in \text{yes}(VC)$ then $I \in \text{yes}(CL)$.

PROVING: $I \in \text{yes}(CL) \Rightarrow f(I) \in \text{yes}(VC)$
- Suppose $I = (\mathcal{G}, k)$ is a yes-instance of Clique.
- Then there is a set $W$ of $k$ vertices in a clique (with all-to-all edges).
- Define $W = V \setminus W$. Clearly $|W| = n - k$.
- We claim $W$ is a vertex cover of $\mathcal{G}$.
- Consider any edge $(u, v) \in E$.
  - If either $u$ or $v$ is in $W$, then we are done.
  - So assume $u, v \notin W$ to obtain a contradiction.
- Then $u, v \in W$, and $W$ is a clique in $G$, so $(u, v) \in G$.
- But $(u, v) \in \mathcal{G}$ implies $(u, v) \notin G$. Contradiction.

PROVING: $f(I) \in \text{yes}(VC) \Rightarrow I \in \text{yes}(CL)$
- Suppose $f(I) = (G, n - k)$ is a yes-instance of VC.
- Then there is a set of $n - k$ vertices $W$ that is a vertex cover of $\mathcal{G}$.
- Define $W = V \setminus W$. Clearly $|W| = k$.
- We claim $W$ is a clique in $G$.
- Since $W$ is a vertex cover of $\mathcal{G}$, every edge in $\mathcal{G}$ has at least one endpoint in $W$.
- Therefore, no edge in $\mathcal{G}$ has two endpoints in $W$.
- So, in $G$, there are edges between all pairs of nodes in $W$. So, $W$ is a clique in $G$.

So, we have demonstrated a polynomial transformation from Clique to Vertex-Cover.