Recall that \( P \) is the set of all decision problems that can be solved by polynomial-time algorithms, and that a \emph{polynomial-time reduction} from the decision problem \( A \) to \( B \), written \( A \leq_P B \), exists when there is a polynomial-time algorithm \( F \) that transforms inputs \( I_A \) to \( A \) into inputs \( I_B \) to \( B \) that have the same answer. In today’s lecture, we continue our exploration of polynomial-time reductions and introduce the idea of polynomial-time verifiers.

1. More polynomial-time reductions

Consider the following problems on graphs:

- **Clique**: Given the graph \( G \) and a positive integer \( k \), determine if \( G \) has a clique of size at least \( k \).
- **IndepSet**: Given the graph \( G \) and a positive integer \( k \), determine if \( G \) has an independent set of size at least \( k \).
- **VertexCover**: Given the graph \( G \) and a positive integer \( k \), determine if \( G \) has a vertex cover (=a set \( S \) of vertices that “cover” all the edges \((u,v) \in E\) in the sense that at least one of \( u \) or \( v \) is in \( S \)) of size at most \( k \).
- **NonEmpty**: Given the graph \( G \), determine if it has at least one edge.

In the last lecture, we saw that \( \text{Clique} \leq_P \text{IndepSet} \) and \( \text{IndepSet} \leq_P \text{Clique} \).

We can prove a number of other polynomial-time reductions between these problems. Some other reductions can also be very simple.

**Lemma 19.1.** \( \text{NonEmpty} \leq_P \text{Clique} \).

**Proof.** Consider the polynomial-time algorithm \( F \) that transforms the input graph \( G \) into the pair \((G,2)\) that includes the same graph and the positive integer 2: the graph \( G \) is nonempty if and only if it includes an edge, which is a clique of size two. \( \square \)

Note that the direction of the reduction is very important: can you show that \( \text{Clique} \leq_P \text{NonEmpty} \)?

Probably not: whether this polynomial-time reduction holds or not is an open question that is in fact equivalent to the famous \( P \) vs. \( \text{NP} \) problem, as we will see in the next lectures.

We can also obtain non-trivial polynomial-time reductions without modifying the graph itself.

**Lemma 19.2.** \( \text{IndepSet} \leq_P \text{VertexCover} \).
Proof. Consider the polynomial-time algorithm $F$ that transforms the input $(G, k)$ to \textsc{IndepSet} into the input $(G, n - k)$ to \textsc{VertexCover}.

If $G$ contains an independent set $I \subseteq V$ of size $k$, then the set $S = V \setminus I$ is a vertex cover of $G$ of size $n - k$. (It is a vertex cover since no edge can have both endpoints in $I$ since it is an independent set.)

And if $G$ contains a vertex cover $S \subseteq V$ of size $n$, then the set $I = V \setminus S$ is a set of size $n - (n - k) = k$ that must be an independent set, since each edge in the graph $G$ has at least one endpoint in $S$. □

All of these examples provide reductions between different problems on graphs, but this does not have to be the case: we can have reductions between very different types of problems as well. Consider for example the \textsc{SetCover} problem.

\textbf{SetCover:} Given a collection $S$ of subsets of $\{1, 2, \ldots, m\}$ and a positive integer $k$, determine if there are $k$ sets $S_1, \ldots, S_k \in S$ such that $S_1 \cup \cdots \cup S_k = \{1, 2, \ldots, m\}$.

\textbf{Lemma 19.3.} \textsc{VertexCover} $\leq_P \textsc{SetCover}$.

\textit{Proof.} Let $F$ be the algorithm that transforms a graph $G = (V, E)$ into a collection of subsets over $\{1, 2, \ldots, m\}$ with $m = |E|$. The transformation is done by labeling each edge in $E$ with the numbers $1, \ldots, m$. Then for each vertex $v \in V$ we create the set

$$S_v = \{i \leq m : v \text{ is incident to edge } i \text{ in } G\}.$$ 

Let $S = \{S_v\}_{v \in V}$. The result of this transformation is $(S, k)$. The transformation can be completed in polynomial time. To verify that it is a reduction from \textsc{VertexCover} to \textsc{SetCover}, we need to verify that $G$ has a vertex cover of size $k$ if and only if $S$ has a set cover of size at most $k$.

- If $G$ has a vertex cover $C$ of size $k$, then the family of sets $S_v$ for each $v \in C$ satisfy $\bigcup_{v \in C} S_v = \{1, 2, \ldots, m\}$ since each edge is adjacent to at least one of the vertices in $C$.

- If $S$ has a set cover $S_{v_1}, \ldots, S_{v_k}$ of size $k$, then the set $C = \{v_1, \ldots, v_k\}$ is a vertex cover for $G$ since the edge $i$ is adjacent to the vertex $v_j$ corresponding to the set $S_{v_j}$ that contained $i$. □

2. \textbf{FACTS ABOUT POLYNOMIAL-TIME REDUCTIONS}

Having now seen quite a few polynomial-time reductions, we can take a step back and try to identify some of the important basic properties of these reductions. The first one is that polynomial-time reductions are \textit{transitive} operations on decision problems.

\textbf{Theorem 19.4.} If $A$, $B$, and $C$ are decision problems that satisfy $A \leq_P B$ and $B \leq_P C$, then $A \leq_P C$.

\textit{Proof.} Let $F_1$ be the polynomial-time algorithm that transforms inputs for $A$ into inputs for $B$, and let $F_2$ be the polynomial-time algorithm that transforms inputs for $B$ into inputs for $C$. Let $F$ be the algorithm that runs $F_1$ (on the input to $A$) and then $F_2$ (on the result of $F_1$). This algorithm also runs in polynomial time, and transforms inputs for $A$ into inputs for $C$ that satisfy the conditions of polynomial-time reductions. □

This lets us obtain new reductions by combining the ones we have already established.
**Lemma 19.5.** Clique $\leq_p$ SetCover.

*Proof.* This result follows immediately from the transitivity of polynomial-time reductions and the fact that we have already showed each of the following polynomial-time reductions:

$$\text{Clique} \leq_p \text{IndepSet} \leq_p \text{VertexCover} \leq_p \text{SetCover}.$$

□

Another important property of polynomial-time reductions that we mentioned at the beginning of the lecture but is worth repeating is that it is *not* symmetric.

**Fact 1.** There are some decision problems $A$ and $B$ such that $A \leq_p B$ but $B \not\leq_p A$.

This is a point worth emphasizing one last time: the direction that you do a polynomial-time reduction really matters! And the way we will be doing these reductions will usually follow the same pattern: we will reduce a known hard problem to a new problem, so that we show that the new problem is hard as well.

**Theorem 19.6.** If $\text{Hard}$ is a decision problem that satisfies $\text{Hard} \not\in \text{P}$ and $B$ is a decision problem that satisfies $\text{Hard} \leq_p B$, then $B \not\in \text{P}$.

*Proof.* Assume for contradiction that $B \in \text{P}$. Then there is a polynomial-time algorithm $F$ that transforms inputs for $\text{Hard}$ into inputs for $B$ (which preserves the $\text{Yes}$ and $\text{No}$ answers) and there is a polynomial-time algorithm $A_B$ that solves $B$, so we obtain a polynomial-time algorithm that solves $\text{Hard}$ by running $F$ and then $A_B$ and outputting the result; this contradicts the fact that $\text{Hard} \not\in \text{P}$. □

**Remark.** But note that if $\text{Hard} \not\in \text{P}$ and we show $A \leq_p \text{Hard}$, this says nothing about whether $A$ is easy or hard!

### 3. Polynomial-time verifiers and NP

All the problems that we have examined last week have two interesting properties in common: first, we believe that they are hard to solve (or, at least, we haven’t found any efficient algorithms to solve them yet!), and they are all easy to *verify*. We can make this idea precise by introducing a new type of algorithm that doesn’t try to solve decision problems by itself, but instead expects to be given some help in the form of a *certificate* that some input $x$ to the problem should have the answer $\text{Yes}$ for the problem.

**Definition 19.7.** A *verifier* for the decision problem $X$ is an algorithm $A$ that takes in as input an input $x$ to problem $X$ and an additional input (that we will call a *potential certificate*) $y$ and satisfies two conditions

1. For every $\text{Yes}$ input $x$ to $X$, there is a certificate $y$ that causes $A(x, y)$ to output $\text{Yes}$; and
2. For every $\text{No}$ input $x$ to $X$, for any possible input $y$ the algorithm $A(x, y)$ outputs $\text{No}$.

In other words, when $x$ is a $\text{Yes}$ input there exists a certificate $y$ that helps convince $A$ that it is indeed a $\text{Yes}$ input, but when $x$ is a $\text{No}$ input there does not exist any claimed certificate that incorrectly causes $A$ to output $\text{Yes}$. The idea of an *efficient* verifier can be defined formally by again associating efficiency with polynomial-time complexity.

---

1. We present this result as a fact because we do not include a proof. If you’re interested in proving it yourself, a hint is that you can take $A$ to be any decision problem in $\text{P}$ and $B$ to be any decision problem that is not in $\text{P}$.
2. i.e., any input $x$ where an algorithm that solves problem $X$ should output $\text{Yes}$ on input $x$. 
Definition 19.8. An algorithm $A$ is a polynomial-time verifier for the decision problem $X$ if it is a verifier for $X$ with time complexity that is polynomial in the size of the input $x$ to $X$.

Remark. Note that having a time complexity that is polynomial in the size of $x$ only means that for every problem $X$ that has a polynomial-time verifier $A$ and every Yes input $x$, there must be a certificate $y$ of size polynomial in the size of $x$ that causes $A(x, y)$ to accept (since the verifier does not have enough time to even read longer certificates!). You will sometimes see this condition explicitly stated in the definition.

NP (for nondeterministic polynomial-time) is the name that was given to the class of all decision problems that can be efficiently verified.

Definition 19.9. NP is the set of all decision problems that have polynomial-time verifiers.

Let’s see some examples.

Lemma 19.10. CLIQUE $\in$ NP.

Proof. We need to show that there is a polynomial-time verifier for CLIQUE. Consider the following algorithm that takes in as input the graph $G = (V, E)$ and the positive integer $k$, and as a potential certificate a set $S \subseteq V$ of size $|S| = k$.

Algorithm 1: CLIQUEVerifier($G = (V, E), k, S$)

\[
\text{for each } u \in S \text{ do}
\]
\[
\quad \text{for each } v \in S \setminus \{u\} \text{ do}
\]
\[
\quad \quad \text{if } (u, v) \notin E \text{ return No;}
\]
\[
\text{return Yes;}
\]

When $G$ has a clique of size $k$ and this clique is provided as the certificate $S$ to the algorithm, then $(u, v) \in E$ for every $u \neq v \in S$ so the algorithm will return Yes when provided with this valid certificate.

When $G$ does not contain a clique of size $k$, then for any set $S$ of size $k$, there must exist two vertices $u, v \in S$ such that $(u, v) \notin E$ (otherwise $S$ would be a clique of size $k$) and so the algorithm returns No for all potential certificates. \hfill \Box

Lemma 19.11. SUBSETSUM $\in$ NP.

Proof. Consider the following algorithm that takes in as input the set of $n$ integers $a_1, \ldots, a_n$ and the target value $t$, and as a potential certificate a subset $S \subseteq \{1, 2, \ldots, n\}$.

Algorithm 2: SUBSETSUMVerifier($a_1, \ldots, a_n, t, S$)

\[
\text{if } \sum_{i \in S} a_i = t \text{ then}
\]
\[
\quad \text{return Yes;}
\]
\[
\text{else}
\]
\[
\quad \text{return No;}
\]

When there is a subset $S \subseteq \{1, 2, \ldots, n\}$ of the integers that satisfy $\sum_{i \in S} a_i = t$, providing this set $S$ as the certificate causes the verifier to return Yes.

When there is no subset $S \subseteq \{1, 2, \ldots, n\}$ of the integers that satisfy $\sum_{i \in S} a_i = t$, then no matter what set $S$ is provided as a potential certificate, the algorithm returns No. \hfill \Box
4. P vs. NP

We now have enough definitions to see that the famous $P$ vs. $NP$ problem can thus be restated as follows:

*Can every problem that has a polynomial-time verifier also be solved with a polynomial-time algorithm?*

Answer this question—and provide a valid proof that justifies your answer!—and you will have solved one of the most significant open problems in all of computer science and mathematics today. But this is bad news for us: it means that until the $P$ vs. $NP$ problem has been resolved, we will not be able to show that $\text{Clique} \notin P$, $\text{SubsetSum} \notin P$, etc.—since all these problems are in $NP$ and it is possible that $P = NP$, then it’s possible that all these problems are in $P$ as well. Starting this week, we will show how we can do the next best thing, and see that if $P \neq NP$, then $\text{Clique}$ and many other “hard” problems are not in $P$. 