Polynomial Time Reductions

Lecture 22
Tuesday, April 16, 2019
Part I

(Polynomial Time) Reductions
Reductions

Reduction from Problem $X$ to Problem $Y$ means (informally) that if we have an algorithm for Problem $Y$, we can use it to find an algorithm for Problem $X$.

Using Reductions

1. We use reductions to find algorithms to solve problems.
Reductions

Reduction from Problem $X$ to Problem $Y$ means (informally) that if we have an algorithm for Problem $Y$, we can use it to find an algorithm for Problem $X$.

Using Reductions

1. We use reductions to find algorithms to solve problems.
Reductions

Reduction from Problem \( X \) to Problem \( Y \) means (informally) that if we have an algorithm for Problem \( Y \), we can use it to find an algorithm for Problem \( X \).

Using Reductions

1. We use reductions to find algorithms to solve problems.
2. We also use reductions to show that we can’t find algorithms for some problems. (We say that these problems are hard.)
For languages $L_X, L_Y$, a *reduction from $L_X$ to $L_Y$* is:

1. An algorithm ...
2. Input: $w \in \Sigma^*$
3. Output: $w' \in \Sigma^*$
4. Such that:

\[
\begin{align*}
  w \in L_Y &\iff w' \in L_X \\
\end{align*}
\]

(Actually, this is only one type of reduction, but this is the one we’ll use most often.) There are other kinds of reductions.
For languages $L_X, L_Y$, a **reduction from $L_X$ to $L_Y$** is:

1. An algorithm ...
2. Input: $w \in \Sigma^*$
3. Output: $w' \in \Sigma^*$
4. Such that:
   
   $w \in L_Y \iff w' \in L_X$

(Actually, this is only one type of reduction, but this is the one we’ll use most often.) There are other kinds of reductions.
For decision problems $X$, $Y$, a \textit{reduction from $X$ to $Y$} is:

1. An algorithm ...
2. Input: $I_X$, an instance of $X$.
4. Such that:

\[
I_Y \text{ is YES instance of } Y \iff I_X \text{ is YES instance of } X
\]
Using reductions to solve problems

1. $\mathcal{R}$: Reduction $X \rightarrow Y$

2. $A_Y$: algorithm for $Y$:

3. $\implies$ New algorithm for $X$:

   \[
   A_X(I_X): \\
   \quad // I_X: instance of X. \\
   \quad I_Y \leftarrow \mathcal{R}(I_X) \\
   \quad \text{return } A_Y(I_Y)
   \]

If $\mathcal{R}$ and $A_Y$ polynomial-time $\implies A_X$ polynomial-time.
Using reductions to solve problems

1. $\mathcal{R}$: Reduction $X \rightarrow Y$

2. $A_Y$: algorithm for $Y$:

3. $\iff$ New algorithm for $X$:

   $A_X(I_X)$:
   
   // $I_X$: instance of $X$.
   $I_Y \leftarrow \mathcal{R}(I_X)$
   return $A_Y(I_Y)$

If $\mathcal{R}$ and $A_Y$ polynomial-time $\iff A_X$ polynomial-time.
Using reductions to solve problems

1. \( \mathcal{R} \): Reduction \( X \rightarrow Y \)
2. \( \mathcal{A}_Y \): algorithm for \( Y \):
3. \( \implies \) New algorithm for \( X \):

\[
\mathcal{A}_X(I_X):
// I_X: instance of X.
I_Y \leftarrow \mathcal{R}(I_X)
\text{return } \mathcal{A}_Y(I_Y)
\]

If \( \mathcal{R} \) and \( \mathcal{A}_Y \) polynomial-time \( \implies \mathcal{A}_X \) polynomial-time.
Comparing Problems

1. If there is reduction from $X$ to $Y$...
2. “Problem $X$ is no harder to solve than Problem $Y$”.
3. If Problem $X$ reduces to Problem $Y$ (we write $X \leq Y$), then $X$ cannot be harder to solve than $Y$.
4. $X \leq Y$:
   1. $X$ is no harder than $Y$, or
   2. $Y$ is at least as hard as $X$. 
Part II

Examples of Reductions
Independent Sets and Cliques

Given a graph $G$, a set of vertices $V'$ is:
Independent Sets and Cliques

Given a graph $G$, a set of vertices $V'$ is:

1. **independent set**: no two vertices of $V'$ connected by an edge.
Independent Sets and Cliques

Given a graph $G$, a set of vertices $V'$ is:

1. **independent set**: no two vertices of $V'$ connected by an edge.
2. **clique**: every pair of vertices in $V'$ is connected by an edge of $G$. 

![Graph diagrams showing independent sets and cliques]
Problem: **Independent Set**

**Instance:** A graph $G$ and an integer $k$.

**Question:** Does $G$ has an independent set of size $\geq k$?
The **Independent Set** and **Clique** Problems

**Problem: Independent Set**

**Instance:** A graph $G$ and an integer $k$.

**Question:** Does $G$ has an independent set of size $\geq k$?

**Problem: Clique**

**Instance:** A graph $G$ and an integer $k$.

**Question:** Does $G$ has a clique of size $\geq k$?
Recall

For decision problems \( X, Y \), a reduction from \( X \) to \( Y \) is:

1. An algorithm ...
2. that takes \( I_X \), an instance of \( X \) as input ...
3. and returns \( I_Y \), an instance of \( Y \) as output ...
4. such that the solution (YES/NO) to \( I_Y \) is the same as the solution to \( I_X \).
Reducing **Independent Set** to **Clique**

An instance of **Independent Set** is a graph $G$ and an integer $k$. 

$G$: 

![Graph representation of Independent Set](image-url)
Reducing **Independent Set** to **Clique**

An instance of **Independent Set** is a graph $G$ and an integer $k$.

Reduction given $<G, k>$ outputs $<\overline{G}, k>$ where $\overline{G}$ is the *complement* of $G$. $\overline{G}$ has an edge $(u, v)$ if and only if $(u, v)$ is not an edge of $G$.

\[G:\]

\[\overline{G}:\]
An instance of **Independent Set** is a graph $G$ and an integer $k$.

Reduction given $<G, k>$ outputs $<\overline{G}, k>$ where $\overline{G}$ is the *complement* of $G$. $\overline{G}$ has an edge $(u, v)$ if and only if $(u, v)$ is not an edge of $G$.

Independent set in $G$. 

![Graph](image-url)
Reducing **Independent Set** to **Clique**

An instance of **Independent Set** is a graph $G$ and an integer $k$.

Reduction given $< G, k >$ outputs $< \overline{G}, k >$ where $\overline{G}$ is the *complement* of $G$. $\overline{G}$ has an edge $(u, v)$ if and only if $(u, v)$ is not an edge of $G$.

![Independent set in G.](image1)

![Clique in G.](image2)
Correctness of reduction

Lemma

$G$ has an independent set of size $k$ if and only if $\overline{G}$ has a clique of size $k$.

Proof.

Need to prove two facts:

$G$ has independent set of size at least $k$ implies that $\overline{G}$ has a clique of size at least $k$.

$\overline{G}$ has a clique of size at least $k$ implies that $G$ has an independent set of size at least $k$.

Easy to see both from the fact that $S \subseteq V$ is an independent set in $G$ if and only if $S$ is a clique in $\overline{G}$.
1 **Independent Set** \( \leq \) **Clique**.
What does this mean?

2 If have an algorithm for **Clique**, then we have an algorithm for **Independent Set**.

3 **Clique** is at least as hard as **Independent Set**.

4 Also... **Clique** \( \leq \) **Independent Set**. Why? Thus **Clique** and **Independent Set** are polynomial-time equivalent.
Independent Set and Clique

1. **Independent Set \( \leq \) Clique.**
   What does this mean?

2. If have an algorithm for **Clique**, then we have an algorithm for **Independent Set**.

3. **Clique** is *at least as hard as Independent Set*.

4. Also... **Clique \( \leq \) Independent Set**. Why? Thus **Clique** and **Independent Set** are polynomial-time equivalent.
Independent Set and Clique

1. Independent Set ≤ Clique. What does this mean?
2. If have an algorithm for Clique, then we have an algorithm for Independent Set.
3. Clique is at least as hard as Independent Set.
4. Also... Clique ≤ Independent Set. Why? Thus Clique and Independent Set are polynomial-time equivalent.
Independent Set and Clique

1. Independent Set \( \leq \) Clique.
   What does this mean?

2. If have an algorithm for Clique, then we have an algorithm for Independent Set.

3. Clique is at least as hard as Independent Set.

4. Also… Clique \( \leq \) Independent Set. Why? Thus Clique and Independent Set are polynomial-time equivalent.
Assume you can solve the **Clique** problem in $T(n)$ time. Then you can solve the **Independent Set** problem in

- $O(T(n))$ time.
- $O(n \log n + T(n))$ time.
- $O(n^2 T(n^2))$ time.
- $O(n^4 T(n^4))$ time.
- $O(n^2 + T(n^2))$ time.

Does not matter - all these are polynomial if $T(n)$ is polynomial, which is good enough for our purposes.
A DFA $M$ is universal if it accepts every string. That is, $L(M) = \Sigma^*$, the set of all strings.

**Problem (DFA universality)**

**Input:** A DFA $M$.
**Goal:** Is $M$ universal?

How do we solve DFA Universality?
We check if $M$ has any reachable non-final state.
A DFA $M$ is universal if it accepts every string. That is, $L(M) = \Sigma^*$, the set of all strings.

**Problem (DFA universality)**

**Input:** A DFA $M$.

**Goal:** Is $M$ universal?

How do we solve DFA Universality?
We check if $M$ has any reachable non-final state.
A DFA $M$ is universal if it accepts every string. That is, $L(M) = \Sigma^*$, the set of all strings.

**Problem (DFA universality)**

**Input:** A DFA $M$.
**Goal:** Is $M$ universal?

How do we solve **DFA Universality**?

We check if $M$ has any reachable non-final state.
A DFA $M$ is universal if it accepts every string. That is, $L(M) = \Sigma^*$, the set of all strings.

**Problem (DFA universality)**

**Input:** A DFA $M$.

**Goal:** Is $M$ universal?

How do we solve **DFA Universality**?

We check if $M$ has any reachable non-final state.
An NFA $N$ is said to be universal if it accepts every string. That is, $L(N) = \Sigma^*$, the set of all strings.

**Problem (NFA universality)**

**Input:** A NFA $M$.

**Goal:** Is $M$ universal?

How do we solve NFA Universality?

Reduce it to DFA Universality?

Given an NFA $N$, convert it to an equivalent DFA $M$, and use the DFA Universality Algorithm.

The reduction takes exponential time!

NFA Universality is known to be PSPACE-Complete and we do not expect a polynomial-time algorithm.
An NFA $N$ is said to be universal if it accepts every string. That is, $L(N) = \Sigma^*$, the set of all strings.

**Problem (NFA universality)**

**Input:** A NFA $M$.

**Goal:** Is $M$ universal?

How do we solve NFA Universality?

Reduce it to DFA Universality?

Given an NFA $N$, convert it to an equivalent DFA $M$, and use the DFA Universality Algorithm.

The reduction takes exponential time!

NFA Universality is known to be PSPACE-Complete and we do not expect a polynomial-time algorithm.
NFA Universality

An NFA \( N \) is said to be universal if it accepts every string. That is, \( L(N) = \sum^* \), the set of all strings.

Problem (NFA universality)

Input: A NFA \( M \).
Goal: Is \( M \) universal?

How do we solve NFA Universality?
Reduce it to DFA Universality?
Given an NFA \( N \), convert it to an equivalent DFA \( M \), and use the DFA Universality Algorithm.

The reduction takes exponential time!

NFA Universality is known to be PSPACE-Complete and we do not expect a polynomial-time algorithm.
An NFA $N$ is said to be universal if it accepts every string. That is, $L(N) = \Sigma^*$, the set of all strings.

Problem (NFA universality)

Input: A NFA $M$.
Goal: Is $M$ universal?

How do we solve NFA Universality?
Reduce it to DFA Universality?
Given an NFA $N$, convert it to an equivalent DFA $M$, and use the DFA Universality Algorithm.
The reduction takes exponential time!
NFA Universality is known to be PSPACE-Complete and we do not expect a polynomial-time algorithm.
We say that an algorithm is **efficient** if it runs in polynomial-time.

To find efficient algorithms for problems, we are only interested in polynomial-time reductions. Reductions that take longer are not useful.

If we have a polynomial-time reduction from problem $X$ to problem $Y$ (we write $X \leq_P Y$), and a poly-time algorithm $A_Y$ for $Y$, we have a polynomial-time/efficient algorithm for $X$. 

![Diagram](https://via.placeholder.com/150)
We say that an algorithm is **efficient** if it runs in polynomial-time. To find efficient algorithms for problems, we are only interested in polynomial-time reductions. Reductions that take longer are not useful.

If we have a polynomial-time reduction from problem $X$ to problem $Y$ (we write $X \leq_p Y$), and a poly-time algorithm $A_Y$ for $Y$, we have a polynomial-time/efficient algorithm for $X$. 

![Diagram showing polynomial-time reductions](image)
Polynomial-time reductions

We say that an algorithm is **efficient** if it runs in polynomial-time.

To find efficient algorithms for problems, we are only interested in polynomial-time reductions. Reductions that take longer are not useful.

If we have a polynomial-time reduction from problem \( X \) to problem \( Y \) (we write \( X \leq_P Y \)), and a poly-time algorithm \( A_Y \) for \( Y \), we have a polynomial-time/efficient algorithm for \( X \).
A polynomial time reduction from a decision problem $X$ to a decision problem $Y$ is an algorithm $\mathcal{A}$ that has the following properties:

1. given an instance $I_X$ of $X$, $\mathcal{A}$ produces an instance $I_Y$ of $Y$
2. $\mathcal{A}$ runs in time polynomial in $|I_X|$.
3. Answer to $I_X$ YES iff answer to $I_Y$ is YES.

**Proposition**

If $X \leq_P Y$ then a polynomial time algorithm for $Y$ implies a polynomial time algorithm for $X$.

Such a reduction is called a *Karp reduction*. Most reductions we will need are Karp reductions. Karp reductions are the same as mapping reductions when specialized to polynomial time for the reduction step.
Reductions again...

Let $X$ and $Y$ be two decision problems, such that $X$ can be solved in polynomial time, and $X \leq_P Y$. Then

- $Y$ can be solved in polynomial time.
- $Y$ can NOT be solved in polynomial time.
- If $Y$ is hard then $X$ is also hard.
- None of the above.
- All of the above.
Polynomial-time reductions and hardness

For decision problems $X$ and $Y$, if $X \leq_P Y$, and $Y$ has an efficient algorithm, $X$ has an efficient algorithm.

If you believe that Independent Set does not have an efficient algorithm, why should you believe the same of Clique?

Because we showed Independent Set $\leq_P$ Clique. If Clique had an efficient algorithm, so would Independent Set!

If $X \leq_P Y$ and $X$ does not have an efficient algorithm, $Y$ cannot have an efficient algorithm!
Polynomial-time reductions and hardness

For decision problems $X$ and $Y$, if $X \leq_p Y$, and $Y$ has an efficient algorithm, $X$ has an efficient algorithm.

If you believe that Independent Set does not have an efficient algorithm, why should you believe the same of Clique?

Because we showed Independent Set $\leq_p$ Clique. If Clique had an efficient algorithm, so would Independent Set!

If $X \leq_p Y$ and $X$ does not have an efficient algorithm, $Y$ cannot have an efficient algorithm!
Polynomial-time reductions and hardness

For decision problems $X$ and $Y$, if $X \leq_P Y$, and $Y$ has an efficient algorithm, $X$ has an efficient algorithm.

If you believe that Independent Set does not have an efficient algorithm, why should you believe the same of Clique?

Because we showed Independent Set $\leq_P$ Clique. If Clique had an efficient algorithm, so would Independent Set!

If $X \leq_P Y$ and $X$ does not have an efficient algorithm, $Y$ cannot have an efficient algorithm!
For decision problems $X$ and $Y$, if $X \leq_P Y$, and $Y$ has an efficient algorithm, $X$ has an efficient algorithm.

If you believe that Independent Set does not have an efficient algorithm, why should you believe the same of Clique?

Because we showed Independent Set $\leq_P$ Clique. If Clique had an efficient algorithm, so would Independent Set!

If $X \leq_P Y$ and $X$ does not have an efficient algorithm, $Y$ cannot have an efficient algorithm!
Proposition

Let $\mathcal{R}$ be a polynomial-time reduction from $\mathbf{X}$ to $\mathbf{Y}$. Then for any instance $I_X$ of $\mathbf{X}$, the size of the instance $I_Y$ of $\mathbf{Y}$ produced from $I_X$ by $\mathcal{R}$ is polynomial in the size of $I_X$.

Proof.

$\mathcal{R}$ is a polynomial-time algorithm and hence on input $I_X$ of size $|I_X|$ it runs in time $p(|I_X|)$ for some polynomial $p()$.

$I_Y$ is the output of $\mathcal{R}$ on input $I_X$.

$\mathcal{R}$ can write at most $p(|I_X|)$ bits and hence $|I_Y| \leq p(|I_X|)$.

Note: Converse is not true. A reduction need not be polynomial-time even if output of reduction is of size polynomial in its input.
Proposition

Let $\mathcal{R}$ be a polynomial-time reduction from $X$ to $Y$. Then for any instance $I_X$ of $X$, the size of the instance $I_Y$ of $Y$ produced from $I_X$ by $\mathcal{R}$ is polynomial in the size of $I_X$.

Proof.

$\mathcal{R}$ is a polynomial-time algorithm and hence on input $I_X$ of size $|I_X|$ it runs in time $p(|I_X|)$ for some polynomial $p()$. $I_Y$ is the output of $\mathcal{R}$ on input $I_X$. $\mathcal{R}$ can write at most $p(|I_X|)$ bits and hence $|I_Y| \leq p(|I_X|)$.

Note: Converse is not true. A reduction need not be polynomial-time even if output of reduction is of size polynomial in its input.
Proposition

Let $\mathcal{R}$ be a polynomial-time reduction from $X$ to $Y$. Then for any instance $I_X$ of $X$, the size of the instance $I_Y$ of $Y$ produced from $I_X$ by $\mathcal{R}$ is polynomial in the size of $I_X$.

Proof.

$\mathcal{R}$ is a polynomial-time algorithm and hence on input $I_X$ of size $|I_X|$ it runs in time $p(|I_X|)$ for some polynomial $p()$.

$I_Y$ is the output of $\mathcal{R}$ on input $I_X$.

$\mathcal{R}$ can write at most $p(|I_X|)$ bits and hence $|I_Y| \leq p(|I_X|)$.

Note: Converse is not true. A reduction need not be polynomial-time even if output of reduction is of size polynomial in its input.
Polynomial-time Reduction

A polynomial time reduction from a decision problem $X$ to a decision problem $Y$ is an algorithm $\mathcal{A}$ that has the following properties:

1. Given an instance $I_X$ of $X$, $\mathcal{A}$ produces an instance $I_Y$ of $Y$.
2. $\mathcal{A}$ runs in time polynomial in $|I_X|$. This implies that $|I_Y|$ (size of $I_Y$) is polynomial in $|I_X|$.
3. Answer to $I_X$ YES iff answer to $I_Y$ is YES.

Proposition

If $X \leq_P Y$ then a polynomial time algorithm for $Y$ implies a polynomial time algorithm for $X$. 
Proposition

\[ X \leq_P Y \text{ and } Y \leq_P Z \text{ implies that } X \leq_P Z. \]

Note: \( X \leq_P Y \) does not imply that \( Y \leq_P X \) and hence it is very important to know the FROM and TO in a reduction.

To prove \( X \leq_P Y \) you need to show a reduction FROM \( X \) TO \( Y \). That is, show that an algorithm for \( Y \) implies an algorithm for \( X \).