Polynomial Time Reductions

Lecture 20
April 15, 2014
Part I

Introduction to Reductions
A 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 $X$. 
Reductions

A reduction from Problem \textbf{X} to Problem \textbf{Y} means (informally) that if we have an algorithm for Problem \textbf{Y}, we can use it to find an algorithm for Problem \textbf{X}.

Using Reductions

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

A 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.)
A 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.)

Also, the right reductions might win you a million dollars!
Example 1: Bipartite Matching and Flows

How do we solve the Bipartite Matching Problem?

Given a bipartite graph \( G = (U \cup V, E) \) and number \( k \), does \( G \) have a matching of size \( \geq k \)?

Solution
Reduce it to Max-Flow. \( G \) has a matching of size \( \geq k \) iff there is a flow from \( s \) to \( t \) of value \( \geq k \).
Example 1: Bipartite Matching and Flows

How do we solve the Bipartite Matching Problem?

Given a bipartite graph $G = (U \cup V, E)$ and number $k$, does $G$ have a matching of size $\geq k$?

Solution

Reduce it to Max-Flow. $G$ has a matching of size $\geq k$ iff there is a flow from $s$ to $t$ of value $\geq k$. 
How do we solve the **Bipartite Matching** Problem?

Given a bipartite graph $G = (U \cup V, E)$ and number $k$, does $G$ have a matching of size $\geq k$?

Solution: Reduce it to Max-Flow. $G$ has a matching of size $\geq k$ iff there is a flow from $s$ to $t$ of value $\geq k$. 

Chandra (UIUC) CS473 Spring 2014 4 / 59
How do we solve the Bipartite Matching Problem?

Given a bipartite graph $G = (U \cup V, E)$ and number $k$, does $G$ have a matching of size $\geq k$?

Solution

Reduce it to Max-Flow. $G$ has a matching of size $\geq k$ iff there is a flow from $s$ to $t$ of value $\geq k$. 
Types of Problems

Decision, Search, and Optimization

- **Decision problem.** Example: given \( n \), is \( n \) prime?

- **Search problem.** Example: given \( n \), find a factor of \( n \) if it exists.

- **Optimization problem.** Example: find the smallest prime factor of \( n \).
Types of Problems

Decision, Search, and Optimization

1. **Decision problem**. Example: given $n$, is $n$ prime?

2. **Search problem**. Example: given $n$, find a factor of $n$ if it exists.
## Types of Problems

### Decision, Search, and Optimization

1. **Decision problem.** Example: given $n$, is $n$ prime?

2. **Search problem.** Example: given $n$, find a factor of $n$ if it exists.

3. **Optimization problem.** Example: find the smallest prime factor of $n$. 

---

Chandra (UIUC)  
CS473  
Spring 2014
Optimization and Decision problems

For max flow...

Problem (\textbf{Max-Flow} optimization version)

Given an instance $G$ of network flow, find the maximum flow between $s$ and $t$. 

Problem (\textbf{Max-Flow} decision version)

Given an instance $G$ of network flow and a parameter $K$, is there a flow in $G$, from $s$ to $t$, of value at least $K$?

While using reductions and comparing problems, we typically work with the decision versions. Decision problems have Yes/No answers. This makes them easy to work with.
### Problem (Max-Flow optimization version)

Given an instance $G$ of network flow, find the maximum flow between $s$ and $t$.

### Problem (Max-Flow decision version)

Given an instance $G$ of network flow and a parameter $K$, is there a flow in $G$, from $s$ to $t$, of value at least $K$?
Optimization and Decision problems

For max flow...

Problem (Max-Flow optimization version)

Given an instance $G$ of network flow, find the maximum flow between $s$ and $t$.

Problem (Max-Flow decision version)

Given an instance $G$ of network flow and a parameter $K$, is there a flow in $G$, from $s$ to $t$, of value at least $K$?

While using reductions and comparing problems, we typically work with the decision versions. Decision problems have Yes/No answers. This makes them easy to work with.
A problem $\Pi$ consists of an infinite collection of inputs $\{I_1, I_2, \ldots,\}$. Each input is referred to as an instance.

The size of an instance $I$ is the number of bits in its representation.

For an instance $I$, $\text{sol}(I)$ is a set of feasible solutions to $I$.

For optimization problems each solution $s \in \text{sol}(I)$ has an associated value.
Examples

Example

An instance of **Bipartite Matching** is a bipartite graph, and an integer $k$. 

The solution to this instance is "YES" if the graph has a matching of size $\geq k$, and "NO" otherwise.

Example

An instance of **Max-Flow** is a graph $G$ with edge-capacities, two vertices $s$, $t$, and an integer $k$. The solution to this instance is "YES" if there is a flow from $s$ to $t$ of value $\geq k$, else 'NO''.

What is an algorithm for a decision Problem $X$?

It takes as input an instance of $X$, and outputs either "YES" or "NO".
Examples

Example

An instance of Bipartite Matching is a bipartite graph, and an integer $k$. The solution to this instance is “YES” if the graph has a matching of size $\geq k$, and “NO” otherwise.
Examples

Example

An instance of **Bipartite Matching** is a bipartite graph, and an integer $k$. The solution to this instance is “YES” if the graph has a matching of size $\geq k$, and “NO” otherwise.

Example

An instance of **Max-Flow** is a graph $G$ with edge-capacities, two vertices $s, t$, and an integer $k$. The solution to this instance is “YES” if there is a flow from $s$ to $t$ of value $\geq k$, else ‘NO’.
Examples

Example
An instance of **Bipartite Matching** is a bipartite graph, and an integer $k$. The solution to this instance is “YES” if the graph has a matching of size $\geq k$, and “NO” otherwise.

Example
An instance of **Max-Flow** is a graph $G$ with edge-capacities, two vertices $s, t$, and an integer $k$. The solution to this instance is “YES” if there is a flow from $s$ to $t$ of value $\geq k$, else ‘NO’.

What is an algorithm for a decision Problem $X$?
It takes as input an instance of $X$, and outputs either “YES” or “NO”.

1. Instance of some problem.
2. It can be fully and precisely described (say in a text file).
3. Resulting text file is a binary string.
4. Any input can be interpreted as a binary string $S$.
5. ... Running time of algorithm: Function of length of $S$ (i.e., $n$).
1. A finite alphabet $\Sigma$. $\Sigma^*$ is set of all finite strings on $\Sigma$.
2. A language $L$ is simply a subset of $\Sigma^*$; a set of strings.
A finite alphabet $\Sigma$. $\Sigma^*$ is set of all finite strings on $\Sigma$.

A language $L$ is simply a subset of $\Sigma^*$; a set of strings.

For every language $L$ there is an associated decision problem $\Pi_L$ and conversely, for every decision problem $\Pi$ there is an associated language $L_\Pi$. 
A finite alphabet $\Sigma$. $\Sigma^*$ is set of all finite strings on $\Sigma$.

A language $L$ is simply a subset of $\Sigma^*$; a set of strings.

For every language $L$ there is an associated decision problem $\Pi_L$ and conversely, for every decision problem $\Pi$ there is an associated language $L_\Pi$.

1. Given $L$, $\Pi_L$ is the following decision problem: Given $x \in \Sigma^*$, is $x \in L$? Each string in $\Sigma^*$ is an instance of $\Pi_L$ and $L$ is the set of instances for which the answer is YES.

2. Given $\Pi$ the associated language is

$$L_\Pi = \left\{ I \mid I \text{ is an instance of } \Pi \text{ for which answer is YES} \right\}.$$

Thus, decision problems and languages are used interchangeably.
Example

1. The decision problem **Primality**, and the language

\[ L = \left\{ \#p \mid p \text{ is a prime number} \right\} \]

Here \( \#p \) is the string in base 10 representing \( p \).

2. **Bipartite** (is given graph bipartite?). The language is

\[ L = \left\{ S(G) \mid G \text{ is a bipartite graph} \right\} \]

Here \( S(G) \) is the string encoding the graph \( G \).
What is the language accepted?
What is the language accepted?

$L = \{ s \in \{a, b\}^* \mid s \text{ contains at most one pair of consecutive } a\text{'s}\}$
What is the language accepted?

$L = \{ s \in \{0,1\}^* | \text{there is a 1 in the last 5 positions of } s \}$
What is the language accepted?

\[ L = \{ s \in \{0, 1\}^* \mid \text{there is a 1 in the last 5 positions of } s \} \]
Are regular languages good?

Let $L$ be a regular language. Then the decision problem associated with $L$ can be solved in

(A) Constant time.
(B) Linear time.
(C) Quadratic time.
(D) Exponential time.
(E) Doubly exponential time (i.e., $2^{2n}$).
(F) Octly exponential time (i.e., $2^{2^{2^{2^{2^{2^{2^{2^{2n}}}}}}}}$).
Reductions, revised.

For decision problems $X, Y$, a **reduction from $X$ to $Y$** is:

1. An algorithm ... 
2. Input: $I_X$, an instance of $X$. 
4. Such that:
   
   $$I_Y \text{ is YES instance of } Y \iff I_X \text{ is YES instance of } X$$

There are other kinds of reductions.
For decision problems $X, Y$, a reduction from $X$ to $Y$ is:

1. An algorithm . . .
2. Input: $I_X$, an instance of $X$.
4. Such that:
   
   \[
   I_Y \text{ is YES instance of } Y \iff I_X \text{ is YES instance of } X
   \]

There are other kinds of reductions.
Using reductions to solve problems

1. \( R \): Reduction \( X \rightarrow Y \)
2. \( A_Y \): algorithm for \( Y \):

\[
A_X(I_X) = \begin{cases} 
A_Y(I_Y), & \text{if } R \text{ and } A_Y \text{ are polynomial-time} \end{cases}
\]

If \( R \) and \( A_Y \) are polynomial-time, then \( A_X \) is also polynomial-time.
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)
   \text{return } A_Y(I_Y)
   ```
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 $X$
   
   $I_Y \leftarrow \mathcal{R}(I_X)$
   
   return $\mathcal{A}_Y(I_Y)$

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

1. “Problem \( X \) is no harder to solve than Problem \( Y \)”.

2. If Problem \( X \) reduces to Problem \( Y \) (we write \( X \leq Y \)), then \( X \) cannot be harder to solve than \( Y \).

3. **Bipartite Matching \( \leq \) Max-Flow.**
   Bipartite Matching cannot be harder than Max-Flow.

4. Equivalently,
   Max-Flow is at least as hard as Bipartite Matching.

5. \( X \leq Y \):
   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 $G$. 
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 Example]

Chandra (UIUC) CS473 Spring 2014 19 / 59
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 Diagram]
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 Example]

Chandra (UIUC)
Problem: Independent Set

Instance: A graph $G$ and an integer $k$.
Question: Does $G$ has an independent set of size $\geq k$?
**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 $k$. 
Reducing **Independent Set** to **Clique**

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

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

Convert $G$ to $\overline{G}$, in which $(u, v)$ is an edge iff $(u, v)$ is not an edge of $G$. ($\overline{G}$ is the *complement* of $G$.)
We use $\overline{G}$ and $k$ as the instance of **Clique**.
Reducing **Independent Set** to **Clique**

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

Convert $G$ to $\overline{G}$, in which $(u, v)$ is an edge iff $(u, v)$ is **not** an edge of $G$. ($\overline{G}$ is the *complement of* $G$.) We use $\overline{G}$ and $k$ as the instance of **Clique**.
Reducing **Independent Set** to **Clique**

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

Convert $G$ to $\overline{G}$, in which $(u, v)$ is an edge iff $(u, v)$ is not an edge of $G$. ($\overline{G}$ is the *complement* of $G$.)

We use $\overline{G}$ and $k$ as the instance of **Clique**.
Reducing **Independent Set** to **Clique**

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

Convert $G$ to $\overline{G}$, in which $(u, v)$ is an edge iff $(u, v)$ is not an edge of $G$. ($\overline{G}$ is the *complement* of $G$.)

We use $\overline{G}$ and $k$ as the instance of **Clique**.
Independent Set and Clique

Independent Set $\leq$ Clique.
**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**.
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.
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... Independent Set is at least as hard as Clique.
Independent Set and Clique

Assume you can solve the **Clique** problem in $T(n)$ time. Then you can solve the **Independent Set** problem in

(A) $O(T(n))$ time.
(B) $O(n \log n + T(n))$ time.
(C) $O(n^2 T(n^2))$ time.
(D) $O(n^4 T(n^4))$ time.
(E) $O(n^2 + T(n^2))$ time.
(F) Does not matter - all these are polynomial if $T(n)$ is polynomial, which is good enough for our purposes.
DFA\(_s\) (Remember 373?) are automata that accept regular languages. NFA\(_s\) are the same, except that they are non-deterministic, while DFA\(_s\) are deterministic.
**DFA**s (Remember 373?) are automata that accept regular languages. **NFA**s are the same, except that they are non-deterministic, while **DFA**s are deterministic.

Every **NFA** can be converted to a **DFA** that accepts the same language using the **subset construction**.

(How long does this take?)
**DFA**s (Remember 373?) are automata that accept regular languages. **NFA**s are the same, except that they are non-deterministic, while **DFA**s are deterministic.

Every **NFA** can be converted to a **DFA** that accepts the same language using the **subset construction**.

(How long does this take?)
The smallest **DFA** equivalent to an **NFA** with \( n \) states may have \( \approx 2^n \) states.
A DFA $M$ is universal if it accepts every string. That is, $L(M) = \Sigma^*$, the set of all strings.
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?
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?
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. Alternatively, minimize $M$ to obtain $M'$ and see if $M'$ has a single state which is an accepting 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?
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?
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.
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! Problem is known to be PSPACE-Complete and we do not expect a polynomial-time algorithm.
Polynomial-time reductions

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


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 $X$.
Polynomial-time Reduction

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

1. given an instance $I_X$ of $X$, $A$ produces an instance $I_Y$ of $Y$
2. $A$ 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.
Let $X$ and $Y$ be two decision problems, such that $X$ can be solved in polynomial time, and $X \leq_p Y$. Then

(A) $Y$ can be solved in polynomial time.
(B) $Y$ can NOT be solved in polynomial time.
(C) If $Y$ is hard then $X$ is also hard.
(D) None of the above.
(E) 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!
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?
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 \textbf{Independent Set} does not have an efficient algorithm, why should you believe the same of \textbf{Clique}?

Because we showed \textbf{Independent Set} $\leq_p \textbf{Clique}$. If \textbf{Clique} had an efficient algorithm, so would \textbf{Independent Set}!
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 $X$ 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 $R$ 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 $R$ is polynomial in the size of $I_X$.

Proof.

$R$ 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 $R$ on input $I_X$. $R$ can write at most $p(|I_X|)$ bits and hence $|I_Y| \leq p(|I_X|)$. \hfill \square
Proposition

Let $R$ 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 $R$ is polynomial in the size of $I_X$.

Proof.

$R$ 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 $R$ on input $I_X$. $R$ 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.
A polynomial time reduction from a decision problem $X$ to a decision problem $Y$ is an algorithm $A$ that has the following properties:

2. $A$ 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 $X$.

Such a reduction is called a Karp reduction. Most reductions we will need are Karp reductions.
Transitivity of Reductions

Proposition

\( X \leq_P Y \) and \( Y \leq_P Z \) 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 \( Y \). In other words show that an algorithm for \( Y \) implies an algorithm for \( X \).
Vertex Cover

Given a graph $G = (V, E)$, a set of vertices $S$ is:
Vertex Cover

Given a graph $G = (V, E)$, a set of vertices $S$ is:

1. **A vertex cover** if every $e \in E$ has at least one endpoint in $S$. 
Given a graph $G = (V, E)$, a set of vertices $S$ is:

1. A **vertex cover** if every $e \in E$ has at least one endpoint in $S$. 

![Diagram of a vertex cover in a graph](image)
Given a graph $G = (V, E)$, a set of vertices $S$ is:

1. A **vertex cover** if every $e \in E$ has at least one endpoint in $S$. 
Vertex Cover

Given a graph $G = (V, E)$, a set of vertices $S$ is:

1. A **vertex cover** if every $e \in E$ has at least one endpoint in $S$.
The **Vertex Cover** Problem

**Problem (Vertex Cover)**

**Input:** A graph $G$ and integer $k$.

**Goal:** Is there a vertex cover of size $\leq k$ in $G$?
**The **Vertex Cover** Problem**

Problem (**Vertex Cover**)  

**Input:** A graph $G$ and integer $k$.  
**Goal:** Is there a vertex cover of size $\leq k$ in $G$?

Can we relate **Independent Set** and **Vertex Cover**?
Proposition

Let $G = (V, E)$ be a graph. $S$ is an independent set if and only if $V \setminus S$ is a vertex cover.

Proof.

$(\Rightarrow)$ Let $S$ be an independent set

1. Consider any edge $uv \in E$.
2. Since $S$ is an independent set, either $u \not\in S$ or $v \not\in S$.
3. Thus, either $u \in V \setminus S$ or $v \in V \setminus S$.
4. $V \setminus S$ is a vertex cover.

$(\Leftarrow)$ Let $V \setminus S$ be some vertex cover:

1. Consider $u, v \in S$
2. $uv$ is not an edge of $G$, as otherwise $V \setminus S$ does not cover $uv$.
3. $\Rightarrow$ $S$ is thus an independent set.
Independent Set $\leq_P$ Vertex Cover

1. $G$: graph with $n$ vertices, and an integer $k$ be an instance of the Independent Set problem.
Independent Set $\leq_P$ Vertex Cover

1. **G**: graph with $n$ vertices, and an integer $k$ be an instance of the Independent Set problem.

2. $G$ has an independent set of size $\geq k$ iff $G$ has a vertex cover of size $\leq n - k$
**Independent Set \( \leq_P \) Vertex Cover**

1. **G**: graph with \( n \) vertices, and an integer \( k \) be an instance of the **Independent Set** problem.

2. **G** has an independent set of size \( \geq k \) iff **G** has a vertex cover of size \( \leq n - k \)

3. \((G, k)\) is an instance of **Independent Set**, and \((G, n - k)\) is an instance of **Vertex Cover** with the same answer.
Independent Set $\leq_P$ Vertex Cover

1. $G$: graph with $n$ vertices, and an integer $k$ be an instance of the Independent Set problem.

2. $G$ has an independent set of size $\geq k$ iff $G$ has a vertex cover of size $\leq n - k$.

3. $(G, k)$ is an instance of Independent Set, and $(G, n - k)$ is an instance of Vertex Cover with the same answer.

4. Therefore, Independent Set $\leq_P$ Vertex Cover. Also Vertex Cover $\leq_P$ Independent Set.
Problem: Edge Cover

**Instance:** A graph \( G \) and integer \( k \).

**Question:** Is there a subset of \( k \) edges such that all the vertices of \( G \) are adjacent to one of these edges.

We have that:

(A) **Edge Cover** is polynomially equivalent to **Independent Set**.

(B) **Edge Cover** is polynomially equivalent to **Vertex Cover**.

(C) **Edge Cover** is polynomially equivalent to **Clique**.

(D) **Edge Cover** is polynomially equivalent to **3 COLORING**.

(E) None of the above.
Can you reduce between these problems

Problem: **2SAT**

**Instance:** $F$: a 2CNF formula.

**Question:** Is there a satisfying assignment to $F$?

Problem: **Max Flow**

**Instance:** $G$, $s$, $t$, $k$: Instance of network flow.

**Question:** Is there a valid flow in $G$ from $s$ to $t$ of value larger than $k$?

(A) $2SAT \leq_p Max \ Flow$.

(B) $Max \ Flow \leq_p 2SAT$.

(C) $2SAT \leq_p Max \ Flow$ and $Max \ Flow \leq_p 2SAT$.

(D) There is NO polynomial time reduction from $2SAT$ to $Max \ Flow$, or vice versa.

(E) All your reduction belong to us.
Suppose you work for the United Nations. Let $U$ be the set of all languages spoken by people across the world. The United Nations also has a set of translators, all of whom speak English, and some other languages from $U$. 
Suppose you work for the United Nations. Let $U$ be the set of all languages spoken by people across the world. The United Nations also has a set of translators, all of whom speak English, and some other languages from $U$.

Due to budget cuts, you can only afford to keep $k$ translators on your payroll. Can you do this, while still ensuring that there is someone who speaks every language in $U$?
A problem of Languages

Suppose you work for the United Nations. Let $U$ be the set of all languages spoken by people across the world. The United Nations also has a set of translators, all of whom speak English, and some other languages from $U$.

Due to budget cuts, you can only afford to keep $k$ translators on your payroll. Can you do this, while still ensuring that there is someone who speaks every language in $U$?

More General problem: Find/Hire a small group of people who can accomplish a large number of tasks.
The **Set Cover** Problem

**Problem (Set Cover)**

**Input:** Given a set $U$ of $n$ elements, a collection $S_1, S_2, \ldots, S_m$ of subsets of $U$, and an integer $k$.

**Goal:** Is there a collection of at most $k$ of these sets $S_i$ whose union is equal to $U$?
The **Set Cover** Problem

**Problem (Set Cover)**

**Input:** Given a set $U$ of $n$ elements, a collection $S_1, S_2, \ldots, S_m$ of subsets of $U$, and an integer $k$.

**Goal:** Is there a collection of at most $k$ of these sets $S_i$ whose union is equal to $U$?

**Example**

Let $U = \{1, 2, 3, 4, 5, 6, 7\}$, $k = 2$ with

$S_1 = \{3, 7\}$  
$S_2 = \{3, 4, 5\}$  
$S_3 = \{1\}$  
$S_4 = \{2, 4\}$  
$S_5 = \{5\}$  
$S_6 = \{1, 2, 6, 7\}$  

$S_2, S_6$ is a set cover.
The **Set Cover** Problem

**Problem (Set Cover)**

**Input:** Given a set $U$ of $n$ elements, a collection $S_1, S_2, \ldots, S_m$ of subsets of $U$, and an integer $k$.

**Goal:** Is there a collection of at most $k$ of these sets $S_i$ whose union is equal to $U$?

**Example**

Let $U = \{1, 2, 3, 4, 5, 6, 7\}$, $k = 2$ with

- $S_1 = \{3, 7\}$
- $S_2 = \{3, 4, 5\}$
- $S_3 = \{1\}$
- $S_4 = \{2, 4\}$
- $S_5 = \{5\}$
- $S_6 = \{1, 2, 6, 7\}$

$\{S_2, S_6\}$ is a set cover
Given graph $G = (V, E)$ and integer $k$ as instance of Vertex Cover, construct an instance of Set Cover as follows:
Vertex Cover $\leq_P$ Set Cover

Given graph $G = (V, E)$ and integer $k$ as instance of Vertex Cover, construct an instance of Set Cover as follows:

1. Number $k$ for the Set Cover instance is the same as the number $k$ given for the Vertex Cover instance.
Vertex Cover $\leq_P$ Set Cover

Given graph $G = (V, E)$ and integer $k$ as instance of Vertex Cover, construct an instance of Set Cover as follows:

1. Number $k$ for the Set Cover instance is the same as the number $k$ given for the Vertex Cover instance.

2. $U = E$.
Vertex Cover $\leq_p$ Set Cover

Given graph $G = (V, E)$ and integer $k$ as instance of Vertex Cover, construct an instance of Set Cover as follows:

1. Number $k$ for the Set Cover instance is the same as the number $k$ given for the Vertex Cover instance.
2. $U = E$.
3. We will have one set corresponding to each vertex; $S_v = \{e \mid e \text{ is incident on } v\}$. 

Chandra (UIUC) CS473 43 / 59
**Vertex Cover \( \leq_p \) Set Cover**

Given graph \( G = (V, E) \) and integer \( k \) as instance of Vertex Cover, construct an instance of Set Cover as follows:

1. Number \( k \) for the Set Cover instance is the same as the number \( k \) given for the Vertex Cover instance.
2. \( U = E \).
3. We will have one set corresponding to each vertex; \( S_v = \{e \mid e \text{ is incident on } v\} \).

Observe that \( G \) has vertex cover of size \( k \) if and only if \( U, \{S_v\}_{v \in V} \) has a set cover of size \( k \). (Exercise: Prove this.)
Vertex Cover $\leq_p$ Set Cover: Example

Let $U = \{a, b, c, d, e, f, g\}$, $k = 2$ with

- $S_1 = \{c, g\}$
- $S_2 = \{b, d\}$
- $S_3 = \{c, d, e\}$
- $S_4 = \{e, f\}$
- $S_5 = \{a\}$
- $S_6 = \{a, b, f, g\}$

Then $\{S_3, S_6\}$ is a set cover.
Vertex Cover $\leq_P$ Set Cover: Example

Let $U = \{a, b, c, d, e, f, g\}$, $k = 2$ with

$S_1 = \{c, g\}$  $S_2 = \{b, d\}$
$S_3 = \{c, d, e\}$  $S_4 = \{e, f\}$
$S_5 = \{a\}$  $S_6 = \{a, b, f, g\}$
Vertex Cover \( \leq_P \) Set Cover: Example

Let \( U = \{a, b, c, d, e, f, g\} \), \( k = 2 \) with

\[
\begin{align*}
S_1 &= \{c, g\} & S_2 &= \{b, d\} \\
S_3 &= \{c, d, e\} & S_4 &= \{e, f\} \\
S_5 &= \{a\} & S_6 &= \{a, b, f, g\}
\end{align*}
\]

\( \{S_3, S_6\} \) is a set cover

\( \{3, 6\} \) is a vertex cover
Proving Reductions

To prove that $X \leq_p Y$ you need to give an algorithm $A$ that:

1. Transforms an instance $I_X$ of $X$ into an instance $I_Y$ of $Y$.
2. Satisfies the property that answer to $I_X$ is YES iff $I_Y$ is YES.
   - typical easy direction to prove: answer to $I_Y$ is YES if answer to $I_X$ is YES
   - typical difficult direction to prove: answer to $I_X$ is YES if answer to $I_Y$ is YES (equivalently answer to $I_X$ is NO if answer to $I_Y$ is NO).
3. Runs in polynomial time.
Consider the statement: $\text{Set Cover} \leq_p \text{Vertex Cover}$. This statement is

(A) correct.
(B) correct (although the reduction seen is in the other direction - so not clear why this is correct).
(C) incorrect.
(D) incorrect (the reduction seen is in the other direction!)
Example of incorrect reduction proof

Try proving $\textbf{Matching} \leq_P \textbf{Bipartite Matching}$ via following reduction:

1. Given graph $G = (V, E)$ obtain a bipartite graph $G' = (V', E')$ as follows.
   1. Let $V_1 = \{u_1 \mid u \in V\}$ and $V_2 = \{u_2 \mid u \in V\}$. We set $V' = V_1 \cup V_2$ (that is, we make two copies of $V$).
   2. $E' = \{u_1v_2 \mid u \neq v \text{ and } uv \in E\}$

2. Given $G$ and integer $k$ the reduction outputs $G'$ and $k$. 
Claim

Reduction is a poly-time algorithm. If $G$ has a matching of size $k$ then $G'$ has a matching of size $k$. 

Incorrect! Why?

Vertex $u \in V$ has two copies $u_1$ and $u_2$ in $G'$. A matching in $G'$ may use both copies!
Claim

Reduction is a poly-time algorithm. If $G$ has a matching of size $k$ then $G'$ has a matching of size $k$.

Proof.

Exercise.
Proof

Claim

Reduction is a poly-time algorithm. If \( G \) has a matching of size \( k \) then \( G' \) has a matching of size \( k \).

Proof.

Exercise.

Claim

If \( G' \) has a matching of size \( k \) then \( G \) has a matching of size \( k \).
**Proof**

**Claim**

Reduction is a poly-time algorithm. If $G$ has a matching of size $k$ then $G'$ has a matching of size $k$.

**Proof.**

Exercise.

**Claim**

If $G'$ has a matching of size $k$ then $G$ has a matching of size $k$.

Incorrect! Why?
“Proof”

Claim

Reduction is a poly-time algorithm. If \( G \) has a matching of size \( k \) then \( G' \) has a matching of size \( k \).

Proof.

Exercise.

Claim

If \( G' \) has a matching of size \( k \) then \( G \) has a matching of size \( k \).

Incorrect! Why? Vertex \( u \in V \) has two copies \( u_1 \) and \( u_2 \) in \( G' \). A matching in \( G' \) may use both copies!
**Problem: Subset Sum**

**Instance:** \( S \) - set of positive integers, \( t \): - an integer number (target).

**Question:** Is there a subset \( X \subseteq S \) such that \( \sum_{x \in X} x = t \)?

**Problem: Partition**

**Instance:** A set \( S \) of \( n \) numbers.

**Question:** Is there a subset \( T \subseteq S \) s.t. \( \sum_{t \in T} t = \sum_{s \in S \setminus T} s \)?

Assume that we can solve **Subset Sum** in polynomial time, then we can solve **Partition** in polynomial time. This statement is

(A) True.

(B) Mostly true.

(C) False.

(D) Mostly false.
### II: Partition and subset sum?

#### Problem: Partition

| Instance: | A set $S$ of $n$ numbers. |
| Question: | Is there a subset $T \subseteq S$ s.t. $\sum_{t \in T} t = \sum_{s \in S \setminus T} s$? |

#### Problem: Subset Sum

| Instance: | $S$ - set of positive integers, $t$: - an integer number (target). |
| Question: | Is there a subset $X \subseteq S$ such that $\sum_{x \in X} x = t$? |

Assume that we can solve **Partition** in polynomial time, then we can solve **Subset Sum** in polynomial time. This statement is

- **(A)** True.
- **(B)** Mostly true.
- **(C)** False.
- **(D)** Mostly false.
III: Partition and Halting?

**Problem: Halting**

**Instance:** \( P \): Program, \( I \): Input.
**Question:** Does \( P \) stop on the input \( I \)?

**Problem: Partition**

**Instance:** A set \( S \) of \( n \) numbers.
**Question:** Is there a subset \( T \subseteq S \) s.t. \( \sum_{t \in T} t = \sum_{s \in S \setminus T} s \)?

Assume that we can solve **Halting** in polynomial time, then we can solve **Partition** in polynomial time. This statement is

(A) True.
(B) Mostly true.
(C) False.
(D) Mostly false.
IV: Halting and Partition?

Problem: Partition

Instance: A set $S$ of $n$ numbers.

Question: Is there a subset $T \subseteq S$ s.t. $\sum_{t \in T} t = \sum_{s \in S \setminus T} s$?

Problem: Halting


Question: Does $P$ stop on the input $I$?

Assume that we can solve Partition in polynomial time, then we can solve Halting in polynomial time. This statement is

(A) True.
(B) Mostly true.
(C) False.
(D) Mostly false.
What we know...

1. **Partition** \( \approx_p \) **Subset sum.**

2. **Halting** is way way way way way way way way **harder.**
We looked at polynomial-time reductions.
Summary

We looked at polynomial-time reductions.

Using polynomial-time reductions

1. If $X \leq_P Y$, and there is no efficient algorithm for $X$, there is no efficient algorithm for $Y$. 
We looked at polynomial-time reductions.

Using polynomial-time reductions

1. If $X \leq_p Y$, and we have an efficient algorithm for $Y$, we have an efficient algorithm for $X$.

2. If $X \leq_p Y$, and there is no efficient algorithm for $X$, there is no efficient algorithm for $Y$. 
We looked at polynomial-time reductions.

Using polynomial-time reductions

1. If $X \leq_P Y$, and we have an efficient algorithm for $Y$, we have an efficient algorithm for $X$.

2. If $X \leq_P Y$, and there is no efficient algorithm for $X$, there is no efficient algorithm for $Y$.
Summary

We looked at polynomial-time reductions.

Using polynomial-time reductions

1. If $X \leq_P Y$, and we have an efficient algorithm for $Y$, we have an efficient algorithm for $X$.

We looked at some examples of reductions between Independent Set, Clique, Vertex Cover, and Set Cover.