Polynomial Time Reductions

Lecture 20

November 10, 2011

Part I

Introduction to Reductions

Reductions

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

Using Reductions

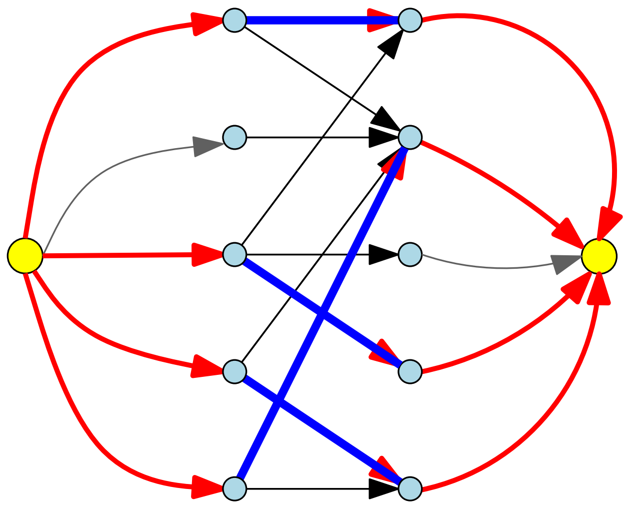
- We use reductions to find algorithms to solve problems.
- We also use reductions to show that we can't find algorithms for some problems. (We say that these problems are hard.)

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 $\boldsymbol{G} = (\boldsymbol{U} \cup \boldsymbol{V}, \boldsymbol{E})$ and number $\boldsymbol{k}$, does $\boldsymbol{G}$ have a matching of size $\geq \boldsymbol{k}$?



Solution

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

Types of Problems

Decision, Search, and Optimization

- Decision problems (example: given n, is n prime?)
- Search problems (example: given n, find a factor of n if it exists)
- Optimization problems (example: find the smallest prime factor of n.)

For **Max-Flow**, the Optimization version is: Find the Maximum flow between s and t. The Decision Version is: Given an integer k, is there a flow of value $\geq k$ between s and t?

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.

Problems vs Instances

- A problem Π 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, $sol(I)$ is a set of feasible solutions to I.
- For optimization problems each solution $s \in sol(I)$ has an associated value.

Examples

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.

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".

Encoding an instance into a string

- I; Instance of some problem.
- I can be fully and precisely described (say in a text file).
- Resulting text file is a binary string.
- $\implies$ Any input can be interpreted as a binary string S.
- ... Running time of algorithm: function of length of S (i.e., n).

Decision Problems and Languages

- A finite alphabet Σ. Σ^* is set of all finite strings on Σ.
- A language L is simply a subset of Σ^*; a set of strings.

For every language L there is an associated decision problem Π_L and conversely, for every decision problem Π there is an associated language L_Π.

- Given L, Π_L is the following problem: given $x \in \Sigma^*$, is $x \in L$? Each string in Σ^* is an instance of Π_L and L is the set of instances for which the answer is YES.
- Given Π the associated language $L_\Pi = \{I \mid I \text{ is an instance of } \Pi \text{ for which answer is YES}\}$.

Thus, decision problems and languages are used interchangeably.

Example

Reductions, revised.

For decision problems $\boldsymbol{X}, \boldsymbol{Y}$, a ***reduction from $\boldsymbol{X}$ to $\boldsymbol{Y}$*** is:

- An algorithm ...
- Input: $\boldsymbol{I_X}$, an instance of $\boldsymbol{X}$.
- Output: $\boldsymbol{I_Y}$ an instance of $\boldsymbol{Y}$.
- Such that:

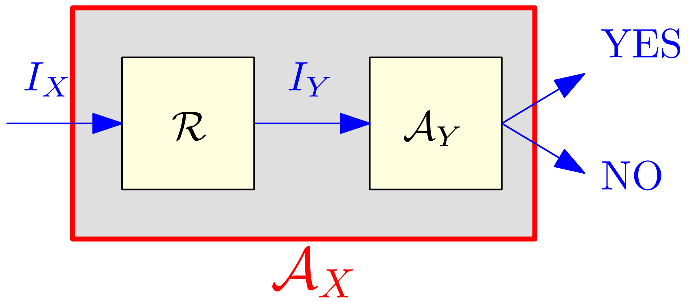
$$\boxed{\boldsymbol{I_Y} \text{ is YES instance of } \boldsymbol{Y}} \iff \boxed{\boldsymbol{I_X} \text{ is YES instance of } \boldsymbol{X}}$$

(Actually, this is only one type of reduction, but this is the one we'll use most often.)

Using reductions to solve problems

- $\mathcal{R}$: Reduction $\boldsymbol{X} \rightarrow \boldsymbol{Y}$
- $\mathcal{A}_Y$: algorithm for $\boldsymbol{Y}$:
- $\implies$ New algorithm for $\boldsymbol{X}$:

```
A_X(I_X):
    // I_X: instance of X.
    I_Y <= R(I_X)
    return A_Y(I_Y)
```



In particular, if $\mathcal{R}$ and $\mathcal{A}_Y$ are polynomial-time algorithms, $\mathcal{A}_X$ is also polynomial-time.

Comparing Problems

- Reductions allow us to formalize the notion of "Problem X is no harder to solve than Problem Y".
- If Problem X reduces to Problem Y (we write $X \leq Y$), then X cannot be harder to solve than Y.
- **Bipartite Matching** $\leq$ **Max-Flow**.
  Therefore, **Bipartite Matching** cannot be harder than **Max-Flow**.
- Equivalently,
  Max-Flow is at least as hard as **Bipartite Matching**.
- More generally, if $X \leq Y$, we can say that X is no harder than Y, or Y is at least as hard as X.

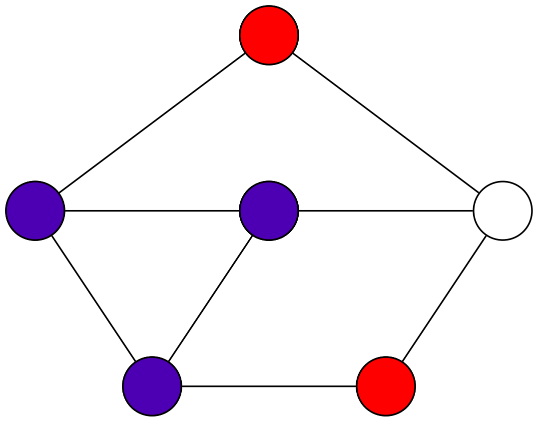
Part II

Examples of Reductions

Independent Sets and Cliques

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

- An ***independent set***: if no two vertices of V' are connected by an edge of G.
- ***clique***: *every* pair of vertices in V' is connected by an edge of G.



The Independent Set and Clique Problems

Independent Set Problem

- **Input:** A graph G and an integer k.
- **Goal;** Decide whether G has an independent set of size $\geq k$.

Clique Problem

- **Input:** A graph G and an integer k.
- **Goal:** Decide whether G has a clique of size $\geq k$.

Recall

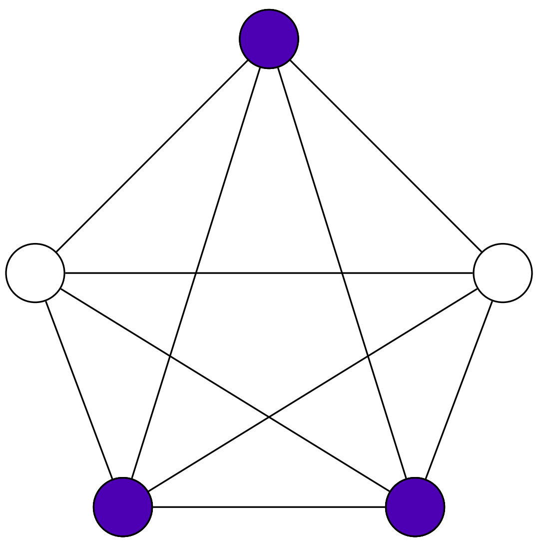
For decision problems X, Y, a reduction from X to Y is:

- An algorithm ...
- that takes I_X, an instance of X as input ...
- and returns I_Y, an instance of Y as output ...
- 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.

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**.
  What does this mean?
- If have an algorithm for **Clique**, then we have an algorithm for **Independent Set**.
- **Clique** is *at least as hard as* **Independent Set**.
- Also... **Independent Set** is *at least as hard as* **Clique**.

DFAs and NFAs

DFAs (Remember 373?) are automata that accept regular languages. NFAs are the same, except that they are non-deterministic, while DFAs 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.

DFA Universality

A DFA M is universal if it accepts every string.
That is, $L(M) = \Sigma^*$, the set of all strings.

The **DFA Universality** Problem:

- **Input**: A DFA M
- **Goal**: Decide whether M is 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.

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.

The **NFA Universality** Problem:

Input An NFA N

Goal Decide whether N is 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.

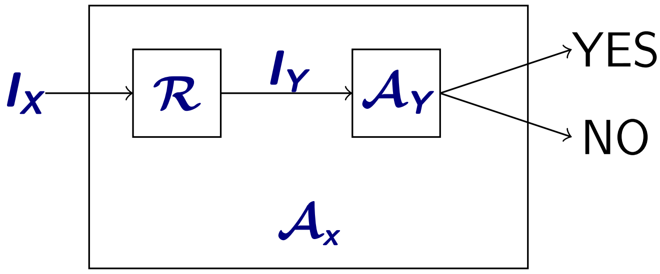
The reduction takes exponential time!

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 $\mathcal{A}_Y$ 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* $\mathcal{A}$ that has the following properties:

- given an instance I_X of X, $\mathcal{A}$ produces an instance I_Y of Y
- $\mathcal{A}$ runs in time polynomial in $|I_X|$.
- 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.

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 instance sizes

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:

- given an instance I_X of X, $\mathcal{A}$ produces an instance I_Y of Y
- $\mathcal{A}$ runs in time polynomial in $|I_X|$. This implies that $|I_Y|$ (size of I_Y) is polynomial in $|I_X|$
- 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 $X \leq_P Z$.

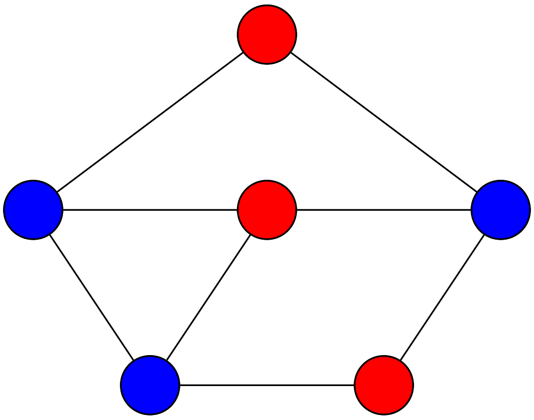
Note: $X \leq_P Y$ does not imply that $Y \leq_P X$ and hence it is very important to know the FROM and TO in a reduction.

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:

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



The **Vertex Cover** Problem

The **Vertex Cover** Problem:

Input A graph G and integer k

Goal Decide whether there is a vertex cover of size $\leq k$

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

- Consider any edge $(u, v) \in E$
- Since S is an independent set, either $u \notin S$ or $v \notin S$
- Thus, either $u \in V \setminus S$ or $v \in V \setminus S$
- $V \setminus S$ is a vertex cover

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

- Consider $u, v \in S$
- (u, v) is not edge, as otherwise $V \setminus S$ does not cover (u, v)
- S is thus an independent set □

Independent Set $\leq_P$ Vertex Cover

- G: graph with n vertices, and an integer k be an instance of the **Independent Set** problem.
- G has an independent set of size $\geq k$ iff G has a vertex cover of size $\leq n - k$
- (G, k) is an instance of **Independent Set** , and $(G, n - k)$ is an instance of **Vertex Cover** with the same answer.
- Therefore, **Independent Set** $\leq_P$ **Vertex Cover**. Also **Vertex Cover** $\leq_P$ **Independent Set**.

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

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 is 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

$$\begin{array}{ll} S_1 = \{3, 7\} & S_2 = \{3, 4, 5\} \\ S_3 = \{1\} & S_4 = \{2, 4\} \\ S_5 = \{5\} & S_6 = \{1, 2, 6, 7\} \end{array}$$

$\{S_2, S_6\}$ is a set cover

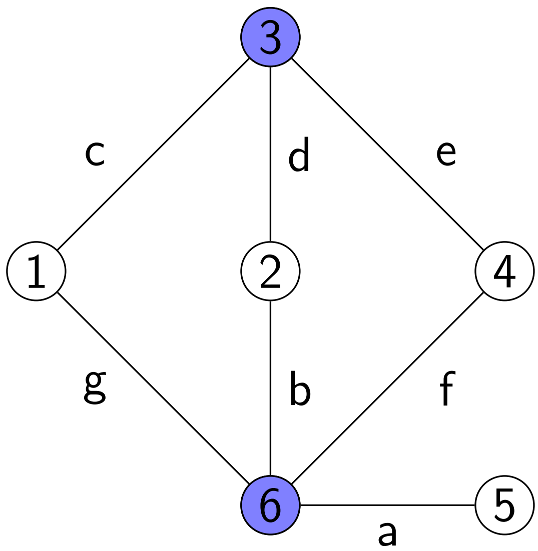
Vertex Cover $\le_P$ Set Cover

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

- Number k for the **Set Cover** instance is the same as the number k given for the **Vertex Cover** instance.
- $U = E$
- 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 $\le_P$ Set Cover: Example



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

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

$\{S_3, S_6\}$ 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 $\mathcal{A}$ that

- transforms an instance I_X of X into an instance I_Y of Y
- 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 (equivalenly answer to I_X is NO if answer to I_Y is NO)
- runs in *polynomial* time

Example of incorrect reduction proof

Try proving **Matching** $\leq_P$ **Bipartite Matching** via following reduction:

- Given graph $G = (V, E)$ obtain a bipartite graph $G' = (V', E')$ as follows.
  - 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)
  - $E' = \{(u_1, v_2) \mid u \neq v \text{ and } (u, v) \in E\}$
- Given G and integer k the reduction outputs G' and k.

Example

"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!

Summary

We looked at polynomial-time reductions.

Using polynomial-time reductions

- If $X \leq_P Y$, and we have an efficient algorithm for Y, we have an efficient algorithm for X.
- If $X \leq_P Y$, and there is no efficient algorithm for X, there is no efficient algorithm for Y.

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