Chapter 20

Polynomial Time Reductions

CS 473: Fundamental Algorithms, Fall 2011
November 10, 2011

20.1 Introduction to Reductions

20.2 Overview

20.2.0.1 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

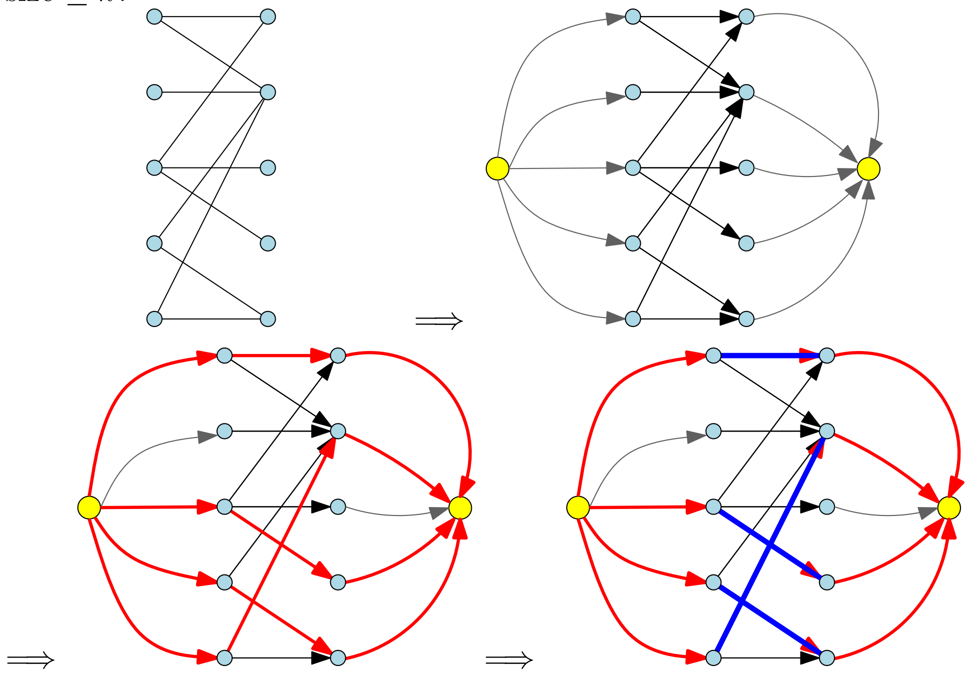
(A) We use reductions to find algorithms to solve problems.
(B) We also use reductions to show that we *can't* find algorithms for some problems. (We say that these problems are *hard*.)

Also, the right reductions might win you a million dollars!

20.2.0.2 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 $\geq k$.

20.3 Definitions

20.3.0.3 Types of Problems

Decision, Search, and Optimization

(A) Decision problems (example: given n, *is* n prime?)
(B) Search problems (example: given n, *find* a factor of n if it exists)
(C) 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.

20.3.0.4 Problems vs Instances

(A) A *problem* Π consists of an *infinite* collection of inputs $\{I_1, I_2, \ldots, \}$. Each input is referred to as an *instance*.
(B) The *size* of an instance I is the number of bits in its representation.
(C) For an instance I, $sol(I)$ is a set of *feasible solutions* to I.
(D) For optimization problems each solution $s \in sol(I)$ has an associated *value*.

20.3.0.5 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".

20.3.0.6 Encoding an instance into a string

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

20.3.0.7 Decision Problems and Languages

(A) A finite *alphabet* Σ. Σ^* is set of all finite strings on Σ.
(B) 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.

(A) 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.
(B) 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.

20.3.0.8 Example
20.3.0.9 Reductions, revised.

For decision problems X, Y, a ***reduction from X to Y*** is:
(A) An algorithm ...
(B) Input: I_X, an instance of X.
(C) Output: I_Y an instance of Y.

(D) Such that:

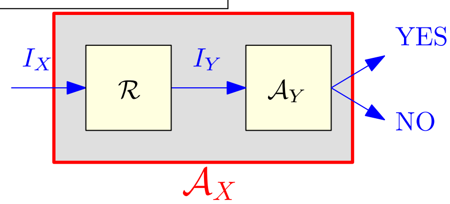
$$\boxed{I_Y \text{ is YES instance of } Y} \iff \boxed{I_X \text{ is YES instance of } X}$$

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

20.3.0.10 Using reductions to solve problems

(A) $\mathcal{R}$: Reduction $X \to Y$
(B) $\mathcal{A}_Y$: algorithm for Y:
(C) $\implies$ New algorithm for 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.

20.3.0.11 Comparing Problems

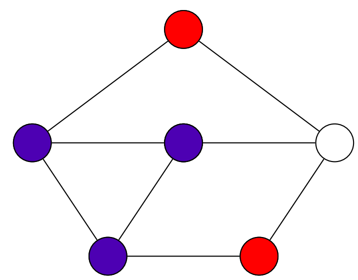
(A) Reductions allow us to formalize the notion of "Problem X is no harder to solve than Problem Y".
(B) If Problem X *reduces to* Problem Y (we write $X \leq Y$), then X cannot be harder to solve than Y.
(C) **Bipartite Matching** $\leq$ **Max-Flow**.
Therefore, **Bipartite Matching** cannot be harder than **Max-Flow**.
(D) Equivalently,
Max-Flow is *at least as hard as* **Bipartite Matching**.
(E) 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.

20.4 Examples of Reductions

20.5 Independent Set and Clique

20.5.0.12 Independent Sets and Cliques

Given a graph G, a set of vertices V' is:
(A) An ***independent set***: if no two vertices of V' are connected by an edge of G.
(B) ***clique***: *every* pair of vertices in V' is connected by an edge of G.



20.5.0.13 The **Independent Set** and **Clique** Problems

Independent Set Problem

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

Clique Problem

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

20.5.0.14 Recall

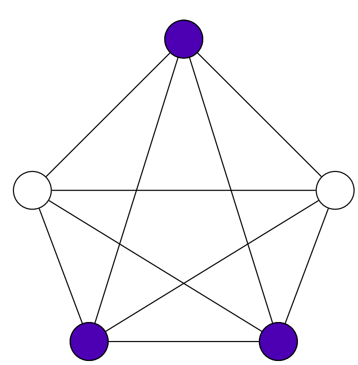
For decision problems X, Y, a reduction from X to Y is:
(A) An algorithm ...
(B) that takes I_X, an instance of X as input ...
(C) and returns I_Y, an instance of Y as output ...
(D) such that the solution (YES/NO) to I_Y is the same as the solution to I_X.

20.5.0.15 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**.



20.5.0.16 Independent Set and Clique

(A) **Independent Set** $\leq$ **Clique**.
    What does this mean?
(B) If have an algorithm for **Clique**, then we have an algorithm for **Independent Set**.
(C) **Clique** is *at least as hard as* **Independent Set**.
(D) Also... **Independent Set** is *at least as hard as* **Clique**.

20.6 NFAs/DFAs and Universality

20.6.0.17 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.

20.6.0.18 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:
(A) **Input**: A DFA M
(B) **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.

20.6.0.19 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.

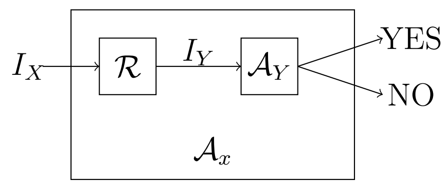
The reduction takes *exponential time*!

20.6.0.20 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.



20.6.0.21 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:
(A) given an instance I_X of X, $\mathcal{A}$ produces an instance I_Y of Y
(B) $\mathcal{A}$ runs in time polynomial in $|I_X|$.
(C) Answer to I_X YES *iff* answer to I_Y is YES.

Proposition 20.6.1 *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.

20.6.0.22 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!

20.6.0.23 Polynomial-time reductions and instance sizes

Proposition 20.6.2 *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.

20.6.0.24 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:

(A) given an instance I_X of X, $\mathcal{A}$ produces an instance I_Y of Y

(B) $\mathcal{A}$ runs in time polynomial in $|I_X|$. This implies that $|I_Y|$ (size of I_Y) is polynomial in $|I_X|$

(C) Answer to I_X YES *iff* answer to I_Y is YES.

Proposition 20.6.3 *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

20.6.0.25 Transitivity of Reductions

Proposition 20.6.4 *$X \leq_P Y$ 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

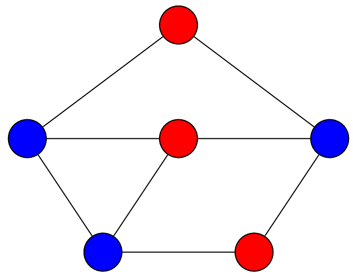
In other words show that an algorithm for Y implies an algorithm for X.

20.7 Independent Set and Vertex Cover

20.7.0.26 Vertex Cover

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

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



20.7.0.27 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**?

20.7.1 Relationship between...

20.7.1.1 Vertex Cover and Independent Set

Proposition 20.7.1 *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
- (A) Consider any edge $(u, v) \in E$
- (B) Since S is an independent set, either $u \notin S$ or $v \notin S$
- (C) Thus, either $u \in V \setminus S$ or $v \in V \setminus S$
- (D) $V \setminus S$ is a vertex cover

($\Leftarrow$) Let $V \setminus S$ be some vertex cover
- (A) Consider $u, v \in S$
- (B) (u, v) is not edge, as otherwise $V \setminus S$ does not cover (u, v)
- (C) S is thus an independent set

■

20.7.1.2 Independent Set $\leq_P$ Vertex Cover

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

20.8 Vertex Cover and Set Cover

20.8.0.3 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.

20.8.0.4 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 20.8.1 *¡2-¿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*

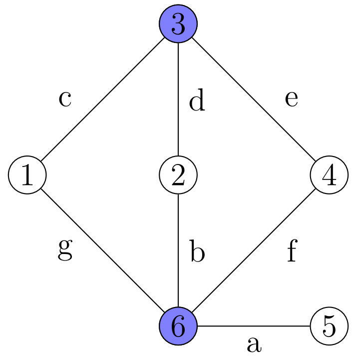
20.8.0.5 Vertex Cover $\leq_P$ Set Cover

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

(A) Number k for the **Set Cover** instance is the same as the number k given for the **Vertex Cover** instance.
(B) $U = E$
(C) 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.)

20.8.0.6 **Vertex Cover** $\leq_P$ **Set Cover**: Example



$\{3,6\}$ is a vertex cover

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

$$\begin{array}{ll} S_1 = \{c,g\} & S_2 = \{b,d\} \\ S_3 = \{c,d,e\} & S_4 = \{e,f\} \\ S_5 = \{a\} & S_6 = \{a,b,f,g\} \end{array}$$

$\{S_3, S_6\}$ is a set cover

20.8.0.7 Proving Reductions

To prove that $X \leq_P Y$ you need to give an algorithm $\mathcal{A}$ that

(A) transforms an instance I_X of X into an instance I_Y of Y

(B) satisfies the property that answer to I_X is YES iff I_Y is YES

- (A) typical easy direction to prove: answer to I_Y is YES if answer to I_X is YES
- (B) *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)

(C) runs in *polynomial* time

20.8.0.8 Example of incorrect reduction proof

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

(A) Given graph $G = (V,E)$ obtain a bipartite graph $G' = (V', E')$ as follows.

- (A) 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)
- (B) $E' = \{(u_1, v_2) \mid u \neq v \text{ and } (u,v) \in E\}$

(B) Given G and integer k the reduction outputs G' and k.

20.8.0.9 Example
20.8.0.10 "Proof"

Claim 20.8.2 *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 20.8.3 *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!

20.8.0.11 Summary

We looked at *polynomial-time reductions.*

Using polynomial-time reductions

(A) If $X \leq_P Y$, and we have an efficient algorithm for Y, we have an efficient algorithm for X.

(B) 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**.