Polynomial Time Reductions

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 \( \geq k \).

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 \textit{smallest} prime factor of \( n \).
Optimization and Decision problems

For max flow...

1. Max-flow as optimization problem:
   
   **Problem (Max-Flow optimization version)**

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

2. Max-flow as decision problem:
   
   **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.

Examples

1. Instance **Bipartite Matching**: a bipartite graph, and integer $k$.
   - Solution is “YES” if graph has matching size $\geq k$, else “NO”.

2. Instance **Max-Flow**: graph $G$ with edge-capacities, two vertices $s$, $t$, and an integer $k$.
   - Solution to instance is “YES” if there is a flow from $s$ to $t$ of value $\geq k$, else “NO”.

   An algorithm for a decision Problem $X$?

   **Decision algorithm**: Input an instance of $X$, and outputs either “YES” or “NO”.

Problems vs Instances

- 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.

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 $\Sigma$. $\Sigma^*$ is set of all finite strings on $\Sigma$.
- A language $L$ is simply a subset of $\Sigma^*$; a set of strings.
- Language $\equiv$ decision problem.
  - For any language $L$ there is a decision problem $\Pi_L$.
  - For any decision problem $\Pi$ there is an associated language $L_\Pi$.
- Given $L$, $\Pi_L$ is the 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.
- Given $\Pi$ the associated language is $L_\Pi = \{I \mid I$ is an instance of $\Pi$ for which answer is YES $\}$.
- Thus, decision problems and languages are used interchangeably.

Example

- The decision problem Primality, and the language
  $$L = \{\#p \mid p \text{ is a prime number}\}.$$  
  Here $\#p$ is the string in base 10 representing $p$.
- Bipartite (is given graph is bipartite. The language is
  $$L = \{S(G) \mid G \text{ is a bipartite graph}\}.$$  
  Here $S(G)$ is the string encoding the graph $G$.

Reductions, revised.

For decision problems $X$, $Y$, a reduction from $X$ to $Y$ is:
- An algorithm . . .
- Input: $I_X$, an instance of $X$.
- Output: $I_Y$ an instance of $Y$.
- Such that:
  $I_Y$ is YES instance of $Y$ $\iff$ $I_X$ is YES instance of $X$
- (Actually, this is only one type of reduction, but this is the one we'll use most often.)

Using reductions to solve problems

- $R$: Reduction $X \rightarrow Y$
- $A_Y$: algorithm for $Y$:
  - $\implies$ New algorithm for $X$:
    $$A_X(I_X):
    \begin{array}{l}
    \text{// $I_X$: instance of $X$.} \\
    I_Y \leftarrow R(I_X) \\
    \text{return } A_Y(I_Y)
    \end{array}$$

In particular, if $R$ and $A_Y$ are polynomial-time algorithms, $A_X$ 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 \( X \).

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:

- 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 if \( (u, v) \) is not an edge of \( G \). (\( \overline{G} \) is the complement of \( G \).)
- \( \langle G, k \rangle \) is an 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.

DFA s and NFAs

- DFAs (Remember 373?) are deterministic automata that accept regular languages.
- NFAs are the same, except that non-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.
- DFA universality problem:
  - 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.
NFA Universality

- An NFA $N$ is universal if it accepts every string. That is, $L(N) = \Sigma^*$, the set of all strings.
- NFA universality problem:

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!

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:
- given an instance $I_X$ of $X$, $A$ produces an instance $I_Y$ of $Y$
- $A$ runs in time polynomial in $|I_X|$.
- Answer to $I_X$ YES $\iff$ answer to $I_Y$ is YES.

Polynomial transitivity:

Proposition

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

Such a reduction is 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.

---

Transitivity of Reductions

- Reductions are transitive:

**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 $S$. 

---

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.
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?

Relationship between...

**Vertex Cover and Independent Set**

**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 \( u \notin S \) or \( v \notin S \).
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. \( \implies S \) is thus an independent set.

Inequalities 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.

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.
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

---

**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:

- 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; $S_v = \{e \mid e$ 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.)

---

**Proving Reductions**

To prove that $X \leq_P Y$ you need to give an algorithm $A$ 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 answer to $I_Y$ 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).
- Runs in *polynomial* time.
Example of incorrect reduction proof

Try proving $\text{Matching} \leq_p \text{Bipartite Matching}$ via following reduction:

1. 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. $E' = \{u_1v_2 \mid u \neq v \text{ and } uv \in E\}$

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

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

Summary

1. We looked at polynomial-time reductions.
2. 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 $Y$.
3. We looked at some examples of reductions between $\text{Independent Set}$, $\text{Clique}$, $\text{Vertex Cover}$, and $\text{Set Cover}$.