CSC 373 - Algorithm Design, Analysis, and Complexity **Summer 2014**

Lalla Mouatadid

Introduction to Complexity Theory

What is **Complexity Theory**? Complexity theory is a formal mathematical theory, in which we study *computational problems* and the *algorithms* to solve them. We mean by a computational problem any problem that takes as input a binary string x encoding the object (think graph, set of intervals, sequence of numbers etc..), and produces a binary string y encoding the solution. We use $\{0,1\}^*$ to denote the set of all binary strings of any finite length, thus $x \in \{0,1\}^*$. For the purpose of this course, we mean by algorithm any program written in C. An algorithm takes as input a string $x \in \{0,1\}^*$ and outputs another string $y \in \{0,1\}^*$.

In complexity theory, we are interested in **decision problems**; there are problems of the form: Given an input $x \in \{0,1\}^*$, output 1 if x satisfies some property or 0 otherwise. More formally:

A **decision problem** is a boolean function:

$$\mathcal{L} : \{0,1\}^* \to \{0,1\}$$

For all $x \in \{0,1\}^*$, if $\mathcal{L}(x) = 1$, we say that x is an **accepting instance** of $\mathcal{L}$; otherwise x is a **rejecting instance**.

You may notice that all the problems we have studied so far are **not** decision problems! We've always returned a subset of the input (non-overlapping intervals), or a new set of outputs (a path with certain properties) etc ... Luckily, it is easy to transform our usual problems into decision problems without losing any of the problem's content.

Consider, for instance, the interval scheduling problem: Given a set $\mathcal{I} = \{I_1, I_2, ..., I_n\}$ of non-empty intervals, we want a subset $S \subseteq \mathcal{I}$ of non-overlapping intervals. To transform this problem into a decision problem, we introduce a threshold variable k, and formulate the problem as follows:

Input: A set $\mathcal{I} = \{I_1, I_2, ..., I_n\}$ where $I_i = (s_i, f_i)$ for all $1 \le i \le n$ and a positive interger k.
Output: Return 1 if there exists a subset $S \subseteq \mathcal{I}$ of non-overlapping intervals and $|S| \ge k$; otherwise return 0.

I claim that if we can solve the optimization problem then we can solve the decision problem too and vice versa. Clearly from optimization to decision is easy. How about the converse? Let's consider the example above. Suppose we have an algorithm A that solves the decision problem, we can perform a binary search over the value of k. Eventually, we'll find two values for k: k_1 and k_2 such that $A(k_1) = 1$ and $A(k_2) = 0$ and $k_2 = k_1 + 1$. This means that k_1 is the size of the largest subset $S \subseteq \mathcal{I}$ of non-overlapping intervals; thus solving the optimization problem.

Complexity Classes:

We call a **complexity class** a collection of decision problems. These classes are defined by some restricted classes of algorithms, and in this course we are interested in the classes **P** (polynomial time) and **NP** (non-deterministic, polynomial time).

A decision problem $\mathcal{L}$ is in the complexity class **P** if and only there exists a **polynomial time** algorithm A

that computes $\mathcal{L}$. In other words, a decision problem $\mathcal{L}$ is in **P** if we can solve it *efficiently.*

On the other hand, a decision problem $\mathcal{L}$ is in **NP** if we can **verify** the solution/output *easily.* To verify a solution easily, we need to come up with a polynomial time algorithm V called a **verifier**.

An algorithm V is a **verifier** if for any input $x \in \{0,1\}^*$, the verifier also receives an additional string $y \in \{0,1\}^*$ called a **certificate** for V. A certificate y is not always the solution; V just uses y as a helper for its computation on x.

A decision problem $\mathcal{L}$ is in the complexity class **NP** iff there exists a polynomial time verifier V such that for any n-bit input string $x \in \{0,1\}^*$, there is a certificate y such that:

$$V(x,y) = 1 \iff \mathcal{L}(x) = 1$$

and y has length $\mathcal{O}(n^c)$ for some constant c.

In other words, we can verify the solution in polynomial time if we are given the right certificate. What's a good certificate? That depends on the problem.

Let's go back to the interval scheduling decision problem, a good certificate could just be the set S. We can check in polynomial time if:

- $|S| \geq k$, and if
- all intervals in S are pairwise non-overlapping.

We know that $V(\mathcal{I}, S) = 1$ if and only if we satisfy these two constraints. Therefore there exists a poly-time verifier for the interval scheduling problem, which means this problem is in NP. But wait[1]! We've already solved the interval scheduling problem using the EFT algorithm in $\mathcal{O}(n \log n)$ time. This means that $P \subseteq NP$. So does $NP \subseteq P$? No one knows, but most people (theoreticians? smart people in general..) think that $P \neq NP$. How do we go about proving such a thing? To show that NP is not in P, we need to find a problem $\mathcal{L}$ such that $L \in NP$ and $L \notin P$. We know that P is the class of "easily solvable" problems, so if $L \in NP \backslash P$, then L must be some sort of "hard problem". The question then is how can we measure the hardness of a problem? Well! Welcome to the Theory of NP-completeness! This is precisely what we'll look at in the next 2 weeks.

Before getting our hands dirty with NP-completeness, we first need a few more definitions.

- Let A and B de decision problems. We say that A is **Karp reducible** (or many-to-one reducible) to B if there exists a polynomial time algorithm R for which:

  $$\forall x \in \{0,1\}^*, A(x) = 1 \iff B(R(x)) = 1$$

  We write $A \leq_p B$ if A is Karp reducible to B. Notice that this means that B is at least hard as A.

Karp reducibility has the following (somewhat obvious) properties :

- If H is a poly time algorithm for B, and $A \leq_p B$ then there is a poly time algorithm for A.
- (**transitivity**): If $A \leq_p B$ and $B \leq_p C$ then $A \leq_c C$.

[1]There is more! ... (Sorry!)

Now we define what it means for a problem to be **NP-hard** and **NP-complete**.

Let $\mathcal{L}$ be a decision problem.

- $\mathcal{L}$ is **NP-hard** iff $A \leq_p \mathcal{L}$ **for every** $A \in NP$. This means that $\mathcal{L}$ is at least as hard as every other problem in NP.
- $\mathcal{L}$ is **NP-complete** if $\mathcal{L}$ is NP-hard and is in NP.

What the above definitions tell us is that if there exists a problem A in NP but not in P, then any other NP-complete problem is not going to be in P either.

Now that we have all the definitions layed out, we will take the theorem below for granted, and will use it, combined with polynomial time reducibility, to show that other problems are NP-complete.

Theorem 1. *Circuit-SAT is NP-complete.*