CSCE 411: Design and Analysis of Algorithms

Lecture 19: Computational Complexity: P and NP

Date: April 8 *Nate Veldt, updated by Samson Zhou*

1 The universe of computational problems

Refresher on computational problems It will be important for us to distinguish between three concepts in defining and solving computational problems:

1. Problem: a computational task that takes in an input and seeks a specific output.

2. Instance: one specific input for the problem

3. Candidate solution: the data encoding a possible solution to an instance.

1.1 Decision Problems and Optimization Problems

A computational problem Q is a *decision* problem if

An optimization problem is a problem that seeks the minimum or maximum value of a function subject to constraints.

Question 1. *Which of the following is a decision problem?*

A *Find the maximum flow for a graph $G = (V, E)$ with source/sink nodes s and t.*

B *Find if the graph $G = (V, E)$ is connected.*

C *Find the shortest path of a graph $G = (V, E)$*

D *Find if $G = (V, E)$ is a directed acyclic graph*

E *More than one of the above*

For any optimization problem, we can define a corresponding *decision* problem by taking an extra input k and asking:

1.2 The problem class P

A problem Q is said to be in P if there is a polynomial time algorithm that solves the problem.

In other words, if n is the size of the input data, the problem has a runtime such as:

These are often referred to as _______________ problems and are treated in theory as problems that can be solved ____________.

All of the problems we have considered so far are polynomial time, but this is only a small sample of problems.

1.3 The problem class NP

We'll define NP problems using a few different levels of technical difficulty.

Level 1: informal, gets us most of the way there. A problem is in NP if we can check in polynomial time whether a given candidate solution solves the instance or not.

Level 2: certificates and verifiers. NP is the set of decision problems with the following property:

If the answer to an instance is *yes*, there exists some data called a *certificate* with which an algorithm can *verify* the answer is YES in polynomial time.

The *certificate* is

The *verifier* is

Level 3: the technically precise definition. NP is the set of decision problems that can be solved by

This involves defining Turing machines, formal languages, and a whole lot more.

You do not need to process this definition (or all the textbook details) for this course. But it does tell us where the name "NP" comes from:

Examples A problem is in NP if for every YES instance there is a *certificate* that an algorithm can use to *verify* in polynomial time that it is truly a YES instance.

Example 1. Does $G = (V, E)$ have a clique of size at least k?

Certificate

Verifier

Example 2. Is $G = (V, E)$ a connected graph?

Certificate

Verifier

Proving something is in NP: *If you can check in polynomial time whether a candidate solution solves the problem or not, then the problem is in NP.*

1.4 The co-NP class

A problem is in co-NP if for every ______________________ there is a *certificate* (some data) that an algorithm can use to *verify* in polynomial time that it is truly a ______________________

To be clear, we will often use the terms ______-certificate and ______-verifier.

Example problem Is $G = (V, E)$ a directed acyclic graph?

Certificate

Verifier

Conclusion: Checking whether G is a DAG is ________________

Question 2. *The verifier we showed for the clique problem will return YES if the k nodes are a clique, and will return NO if the k nodes are not a clique.*

True or false: this means that this algorithm is also a NO-verifier, and so the clique problem is also in co-NP.

A *True*

B *False*

Question 3. *Checking whether G is a DAG in in co-NP. Is it also in NP?*

A *Yes*

B *No*

2 Reductions

A reduction is a mapping from one computational problem to another.

Definition Let A and B be decision problems. We say that A can be reduced to B if

- For every instance of A we can define a corresponding instance of B such that
- All YES instances in A map to YES instances in B
- All NO instances in A map to NO instances in B

This is a polynomial-time reduction if this conversion process takes polynomial time.

This seems obvious: what's an example of a reduction that is not polynomial-time?

2.1 Reduction and complexity

If A can be reduced to B in polynomial time, we denote this by

$$\rule{2cm}{0.4pt}$$

This implies that:

- A is easier[1] than B, or equivalently
- B is at least as hard as A is to solve

Lemma 2.1. *If* $B \in P$ *and* ______________, *then* ______________

[1] where easier might mean "no harder than"

3 NP-completeness

Definition: A problem Q is NP-complete if:

1. $Q \in \mathrm{NP}$
2. for every problem $B \in \mathrm{NP}$, $B \leq_p Q$.

In words:

This implies that NP-complete problems are the *hardest* problems in NP.

Why? Remember that $A \leq_p B$ means A is easier than B.

In fact, if you only take the second part of the definition, this defines the set of NP-hard problems.