Recall

Summary of Lecture 18

We will study which problems (seemingly) cannot be solved in polynomial time.

\[ P = \text{the class of decision problems that have polynomial time algorithms} \]

\[ X \leq_P Y, \text{ for problems } X, Y, \text{ “}X \text{ reduces to } Y \text{ in polynomial time”}, \text{ means: we can use a polynomial time algorithm for } Y \text{ to make a polynomial time algorithm for } X. \]
The class NP

A few decision problems in NP:

- Hamiltonian path/cycle
- Travelling Salesman Problem
- Independent Set

Common feature: if the answer is YES, then there is some succinct information (a certificate) to verify that the answer is YES.

**Example: Independent Set.** Given graph $G$, and number $k$, does $G$ have an independent set of size $\geq k$?

How can I convince you that Yes, there is an independent set of size $\geq 5$?

How can I convince you that No, there is no independent set of size $\geq 7$?
A *verification algorithm* takes input + certificate and checks it. Formally:

**Definition.** Algorithm A is a *verification algorithm* for the decision problem X if

- A takes two inputs \( x, y \) and outputs YES or NO
- for every input \( x \) for problem X, \( x \) is a YES for X iff there exists a \( y \) (a *certificate*) such that \( A(x,y) \) outputs YES

Furthermore, A is a *polynomial time verification algorithm* if

- A runs in polynomial time
- there is a polynomial bound on the size of the certificate \( y \)

We say X “can be verified in polynomial time” if there is a poly time verification algorithm for X.

**Definition.**

\[ \text{NP} = \text{the class of decision problems that can be verified in polynomial time} \]

\[ \text{NP} = \text{Non-deterministic Polynomial time} \quad \text{— because the certificate is like a non-deterministic guess} \]

CS 360 covers non-deterministic Turing machines
Examples

Subset Sum ∈ NP

Given numbers \( w_1, \ldots, w_n, W \) is there a subset \( S \subseteq \{1, \ldots, n\} \)
such that \( \sum_{i \in S} w_i = W \)

TSP (decision version) ∈ NP

Given a graph \( G \), weights on edges, number \( k \), does \( G \) have a TSP tour of length \( \leq k \)
Examples that don’t seem to be in NP

Unique Subset Sum

Given numbers \( w_1, \ldots, w_n, W \) is there a unique subset \( S \subseteq \{1, \ldots, n\} \) such that \( \sum_{i \in S} w_i = W \)

Steiner tree in the plane

Given points in the plane, can you connect them (using extra points) with a tree of Euclidean length \( \leq k \)
Definition. \( \text{coNP} = \) the class of decision problems where the NO instances can be verified in polynomial time

Example. **Primes**: Given a number \( n \), is it prime?

Primes \( \in \) coNP

In fact, Primes \( \in \) P. A poly time algorithm was found in 2002. [https://en.wikipedia.org/wiki/AKS_primality_test](https://en.wikipedia.org/wiki/AKS_primality_test)
OPEN QUESTIONS

1. $P =? NP$  
   worth $1$ million (Millenium Prize)  
   \[\text{Wikipedia: P versus NP problem}\]

2. $NP =? \text{coNP}$

3. $P =? NP \cap \text{coNP}$
OPEN QUESTIONS

1. $P = ? NP$  
   worth $1$ million (Millenium Prize)  
   \[ \text{https://en.wikipedia.org/wiki/P\_versus\_NP\_problem} \]

2. $NP = ? coNP$

3. $P = ? NP \cap coNP$

Properties

1. $P \subseteq NP, \ P \subseteq coNP$

2. Any problem in NP can be solved in time $O(2^n)$ by trying all certificates one by one
Summary of Lecture 19, Part 1

classes NP, coNP

What you should know from Lecture 19, Part 1:

- how to prove that a problem is in NP (certificate, verification)

Next:

- NP-complete problems
A decision problem $X$ is \textbf{NP-complete} if

- $X \in \text{NP}$
- for every $Y$ in $\text{NP}$, $Y \leq_P X$

i.e. $X$ is [one of] the hardest problem in $\text{NP}$.

Two important implications of $X$ being NP-complete

- if $X$ can be solved in polynomial time then so can every problem in $\text{NP}$ (if $X \in \text{P}$ then $\text{P} = \text{NP}$)
- if $X$ cannot be solved in polynomial time then no NP-complete problem can be solved in polynomial time
- if $X \in \text{co-NP}$ then $\text{NP} = \text{coNP}$ (this needs proof)
The first NP-completeness proof is difficult — must show that every problem $Y \in NP$ reduces to $X$

Subsequent NP-completeness proofs are easier because $\leq_P$ is transitive:

**Claim.** If $Y \leq_P X$ and $X \leq_P Z$ then $Y \leq_P Z$

So to prove $Z$ is NP-complete, we just need to prove $X \leq_P Z$ where $X$ is a known NP-complete problem.
Summary: to prove a decision problem Z is NP-complete

1. prove Z in NP
2. prove $X \leq_P Z$ for some known NP-complete problem X.

Our first NP-complete problem: Circuit Satisfiability
[definition and proof later]

second NP-complete problem: Satisfiability
[proof later, but definition now]

Satisfiability (SAT)
**Input:** a Boolean formula made of Boolean variables, and logical operands $\land$ “and”, $\lor$ “or”, $\neg$ “not”

e.g.

**Question:** Is there an assignment of True/False to the variables to make the formula True?

**Exercise.** Prove that Satisfiability is in NP.
SAT is NP-complete, even the special case of “CNF” — Conjunctive Normal Form

**Definition** of CNF

A formula is a conjunction of clauses; a clause is a disjunction of literals; a literal is a variable or its negation.

\[(x_1 \lor \neg x_2 \lor x_3) \land (\neg x_1 \lor x_4) \land (x_3 \lor x_4 \lor \neg x_5)\]

In fact, SAT is still NP-complete when all clauses have 3 literals — this is called 3-SAT

**3-SAT**

**Input:** A Boolean formula that is an conjunction of clauses, each clause a disjunction of 3 literals, each literal a variable or negation of a variable.

**Question:** Is there an assignment of True/False to the variables to make the formula True?

**Theorem.** 3-SAT is NP-complete [proof later]

but 2-SAT is in P

There is a linear time algorithm for 2-SAT that uses strong connectivity of a directed graph.

Summary of Lecture 19, Part 2

definition of NP-complete, the first NP-complete problems: SAT, 3-SAT

What you should know from Lecture 19, Part 2:

- what are the two steps to proving a problem is NP-complete

Next:

- examples of NP-completeness proofs
Independent Set
Input: Graph $G = (V,E)$, number $k$.
Question: Does $G$ have an independent set of size $\geq k$?

Theorem. Independent Set is NP-complete.
Proof.

1. Independent Set is in NP — we already saw the idea of this in Part 1.
2. $\leq_P$ Independent Set
Independent Set
Input: Graph G = (V,E), number k.
Question: Does G have an independent set of size \( \geq k \)?

Theorem. Independent Set is NP-complete.
Proof.
1. Independent Set is in NP — we already saw the idea of this in Part 1.
2. 3-SAT \( \leq_p \) Independent Set

Suppose we have a polynomial time algorithm for Independent Set. Give a polynomial time algorithm for 3-SAT.

Input: A 3-SAT formula F with clauses \( C_1 \ldots C_m \) on variables \( x_1 \ldots x_n \)
Output: Is F satisfiable?

Idea: - construct a graph G and choose a number k such that
  \( G \) has an independent set of size \( \geq k \) iff F is satisfiable ★
- run the Independent Set algorithm on G, k
- return its answer

This is a many-one (“one-shot”) reduction. To prove correctness, just prove ★
To prove poly time, just prove that G can be constructed in poly time (in size of F).
Proof. continued

Input: A 3-SAT formula $F$ with clauses $C_1 \ldots C_m$ on variables $x_1 \ldots x_n$

Output: Is $F$ satisfiable?

Idea: - construct a graph $G$ and choose a number $k$ such that
  $G$ has an independent set of size $\geq k$ iff $F$ is satisfiable
- run the Independent Set algorithm on $G$, $k$
- return its answer
Input: A 3-SAT formula \( F \) with clauses \( C_1 \ldots C_m \) on variables \( x_1 \ldots x_n \)

Output: Is \( F \) satisfiable?

Idea: - construct a graph \( G \) and choose a number \( k \) such that
   - \( G \) has an independent set of size \( \geq k \) iff \( F \) is satisfiable
   - run the Independent Set algorithm on \( G, k \)
   - return its answer

Construction:
- For each clause \( C_i \) with literals \( l_1, l_2, l_3 \), make 3 vertices joined by 3 edges
- if two literals are opposite, join them with an edge.
- \( k := m \)

Runtime: Prove that \( G \) can be constructed in poly time (in the size of \( F \)).
\( G \) has \( 3m \) vertices and can be constructed in time polynomial in \( m \) and \( n \)

Correctness: prove \( G \) has an independent set of size \( \geq k \) iff \( F \) is satisfiable
   - if \( F \) is satisfiable then each clause has (at least) one True literal. Choose the corresponding \( m \) vertices of \( G \). They are independent.
   - if \( G \) has an independent set of size \( \geq m \) there must be one in each triangle. Set the corresponding literals True. This is valid, and satisfies \( F \).

This completes the proof that Independent Set is NP-complete.
Definition. Problem $X$ reduces to problem $Y$, written $X \leq Y$, if an algorithm for $Y$ can be used to make an algorithm for $X$.

Definition. A many one reduction $X \leq Y$ uses the algorithm for $Y$ once and outputs its answer.

Mnemonic: many-one = “one-shot”

The form of a polynomial time many-one reduction $X \leq_p Y$:

Assume we have an algorithm $A$ for $Y$

Algorithm for $X$:
- take input $x$ and construct an input $y$ for problem $Y$
- run $A$ on $y$
- return the answer

For correctness we just need to prove:
the answer for $x$ is YES iff the answer for $y$ is YES

For poly time we just need to prove:
the construction of $y$ takes polynomial time.
How to prove that a decision problem Z is NP-complete

1. prove Z in NP
2. prove $X \leq_P Z$ for some known NP-complete problem X.
   Use a many-one reduction.
Summary of Lecture 19

definition of NP-complete, first NP-completeness proofs

What you should know from Lecture 19 (and Lecture 20)

- how to prove a problem is NP-complete using a polynomial time many-one reduction

Next:

- more examples of NP-completeness proofs