P vs NP

- A TM M is of **time complexity** $T(n)$ if whenever M is given an input w, $|w| = n$, then M *halts* after making at most $T(n)$ many moves.

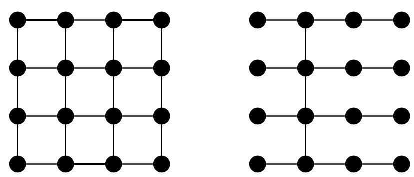
- A language L is in the class **P** if there is a polynomial $T(n)$ (e.g., $T(n) = n^5 + 3n^2 + 13$) such that $L = L(M)$ for some deterministic TM M of time complexity $T(n)$.

- A language L is in the class **NP** if there is a polynomial $T(n)$ such that $L = L(M)$ for some non-deterministic TM M of time complexity $T(n)$.

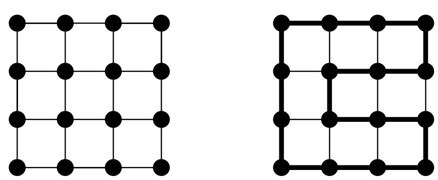
- Observation: $\mathbf{P} \subseteq \mathbf{NP}$

- Question: $\mathbf{NP} \subseteq \mathbf{P}$?

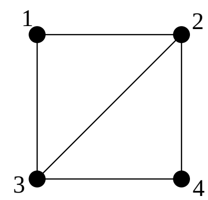
- Ex. of a language in **P**:
  $\{\langle G, k\rangle | G$ has a spanning tree of weight $\leq k\}$.
  $(k = 15)$



  Ex. of a language in **NP** believed not to be in **P**:
  $\{\langle G, k\rangle | G$ has a complete cycle of weight $\leq k\}$.
  $(k = 16)$



- A graph G can be encoded as an adjacency matrix. For example, the graph:



would have the adjacency matrix given by:

$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

- If P is a *decision* problem, the related language L_P consists of the encodings (under some fixed convention) of all the "yes" instances of P.

- Polynomial time algorithm $\equiv$ polynomial time TM: **Feasibility Thesis**.

- A problem P_1 is **reducible in polynomial time** to a problem P_2 if there exists a polynomial time function f such that:

$$\langle I \rangle \in L_{P_1} \iff \langle f(I) \rangle \in L_{P_2}$$

- L is **NP-complete** if:

  1. $L \in$ **NP**

  2. Every language $L' \in$ **NP** is polynomial time reducible to L.

- Ex. of an **NP-complete** language is the **Traveling Salesman Problem**.

- If L is **NP-complete** we view this as *evidence* of L not being in **P***.

*Computers and Intractability: A Guide to the Theory of NP-Completeness by Michael Garey and David Johnson.

Thm: If P_1 is **NP-complete**, P_2 is in **NP**, and there is a polynomial time reduction of P_1 to P_2, then P_2 is also **NP**-complete.

PROOF: Every language L in **NP** is reducible to L_{P_1}, by completeness, and P_1 is reducible to P_2. Enough to show transitivity of reductions.

Thm: If some **NP-complete** problem P is in **P**, then $\mathbf{P} = \mathbf{NP}$.

PROOF: Follows from the fact that all languages in **NP** are polynomial time reducible to P.

Exercises: 10.1.1–3,5–7.

Satisfiability

- **Boolean Expressions** are built from: Boolean variables $x, y, z, x_3, \ldots$, Boolean values $0, 1$, and Boolean connectives: $\vee, \wedge, \neg$, and parenthesis.

- Ex. $\neg x \vee (y \wedge z)$

- If ϕ is a Boolean expression, then a **truth assignment** T is an assignment of truth values to the variables of ϕ.

- Ex. $T(x) = 0, T(y) = 1, T(z) = 1$, then $T(\neg x \vee (y \wedge z)) = \neg 0 \vee (1 \wedge 1) = 1 \vee 1 = 1$.

- T **satisfies** ϕ if $T(\phi) = 1$, and ϕ is **satisfiable** if $\exists T$ s.t. $T(\phi) = 1$.

- The **satisfiability problem** is: given a Boolean expression, is it satisfiable?

- SAT $= \{\langle\phi\rangle | \phi$ is satisfiable$\}$
(i.e., SAT is the language corresponding to the satisfiability problem).

Cook's Theorem: SAT is **NP-complete**.

PROOF: SAT is in **NP**.
Let L be any language in **NP**.
We show there exists a polynomial time function f s.t.:

$$w \in L \iff f(w) = \phi \in \text{SAT}$$

$\exists$ non-det TM M s.t. $L = L(M)$ and M always halts within n^k many steps on inputs w, $|w| = n$, for fixed k.
Given w, f outputs a Boolean formula ϕ which encodes a computation of M on w and is satisfiable $\iff$ M accepts w.