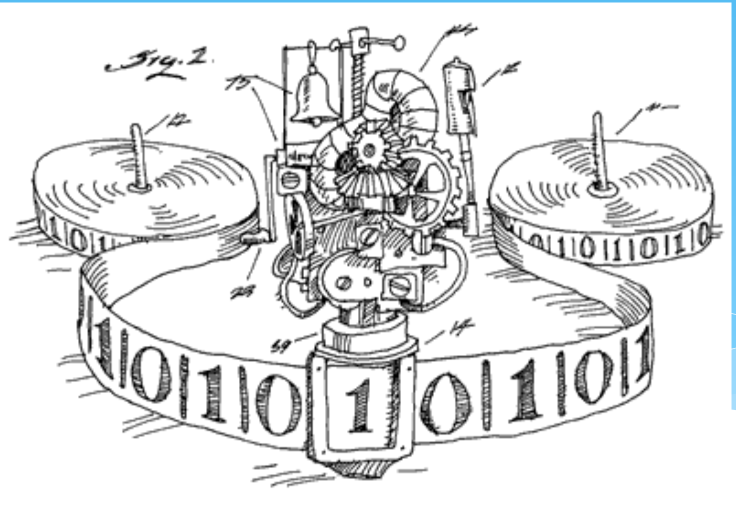
EECS 376: Foundations of Computer Science

Chris Peikert

15 March 2023



Recall

- **Definition:** A language B is $\mathbf{NP}$-***Complete*** if:
  1. $B \in \mathbf{NP}$
  2. B is $\mathbf{NP}$-Hard: $A \leq_p B$ for *every* language $A \in \mathbf{NP}$.
     Easier route: $A \leq_p B$ for *some NP-hard* language A.
- **Reminder:** Language A is ***polynomial-time mapping reducible*** to language B, written $A \leq_p B$, if there is a polynomial-time algorithm f such that:
  - $$x \in A \iff f(x) \in B.$$
  - Implies: If $B \in \mathbf{P}$, then $A \in \mathbf{P}$.
- **Known $\mathbf{NP}$-C languages: SAT, 3SAT, CLIQUE, …**

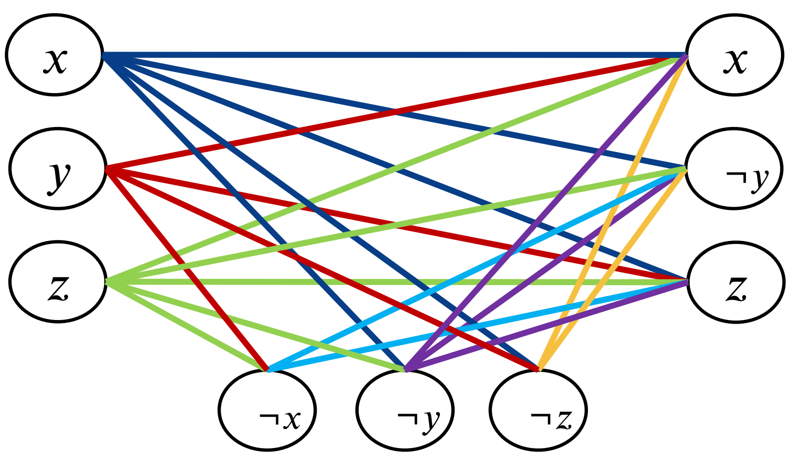
Recall: 3SAT $\leq_p$ CLIQUE

Goal: "*transform*" 3CNF formula ϕ into (G, k) such that:
- ϕ satisfiable $\Longleftrightarrow$ G has a k-clique

* Consider the following example formula:

$$\phi = (x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z) \wedge (x \vee \neg y \vee z)$$

* **Result of transform: has a 3-clique**



3SAT $\leq_p$ CLIQUE

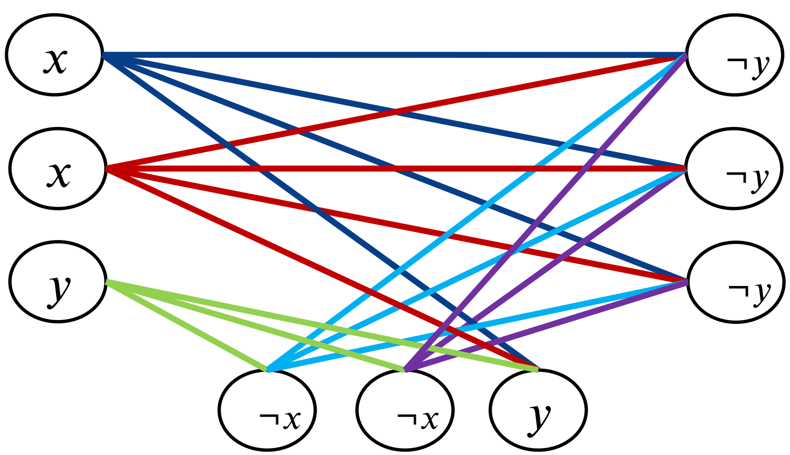
Goal: "*transform*" 3CNF formula ϕ into (G, k) such that:

- ϕ satisfiable $\Longleftrightarrow$ G has a k-clique

* Example that is not satisfiable:

$$\phi = (x \vee x \vee y) \wedge (\neg x \vee \neg x \vee y) \wedge (\neg y \vee \neg y \vee \neg y)$$

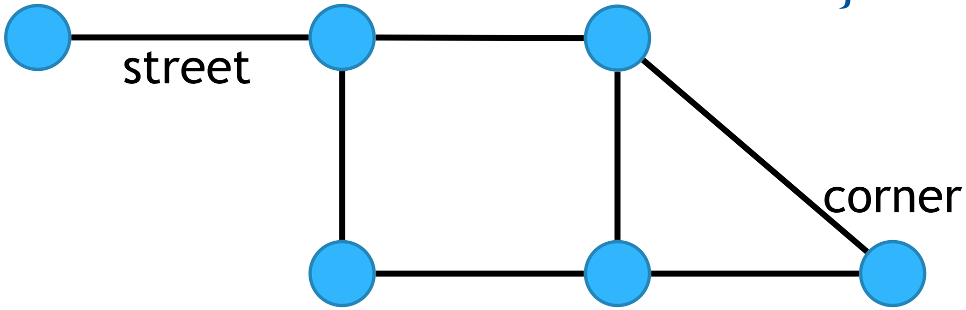
* **Result of transform: no 3-clique**



Vertex Cover

(“Starbucks Problem”)

- Given a city, is it possible to put stores on k street corners so that *every* street is “covered” by some store?
- **Formally:** A ***vertex cover*** of a graph $G = (V, E)$ is a set $C \subseteq V$ s.t. for all $(u, v) \in E$: $u \in C$ or $v \in C$ (or both). (all *edges* are “***covered***” by C)
  - $\text{VERTEXCOVER} = \{(G, k) : G \text{ is an undirected graph that has a vertex cover of size } k\}$



VERTEXCOVER is NP-C

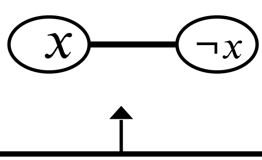
* **Claim:** VERTEXCOVER is $\mathbf{NP}$-Complete
* **Proof:** General Strategy:
  1. VERTEXCOVER $\in \mathbf{NP}$ (Exercise)
  2. $A \leq_p$ VERTEXCOVER for *some* $\mathbf{NP}$-C language A
* We will show that $3\mathrm{SAT} \leq_p$ VERTEXCOVER.
* **Detailed Goal:** Given an algorithm

  $$f : \{3CNF\ formula\} \rightarrow \{(graph,\ k)\}$$

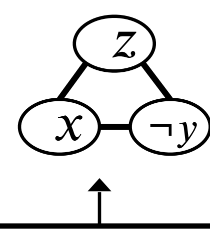
  3. f is efficient
  4. ϕ is satisfiable $\iff G$ has a vertex cover of size k

3SAT $\leq_p$ VERTEXCOVER

- **Proof idea:**
  - Given a 3CNF formula ϕ with n variables, m clauses:
  - Make "***gadget***" subgraphs that represent variables and clauses.
  - Connect the gadgets together in the right way.



Variable gadget for x



Clause gadget for $(x \vee \neg y \vee z)$

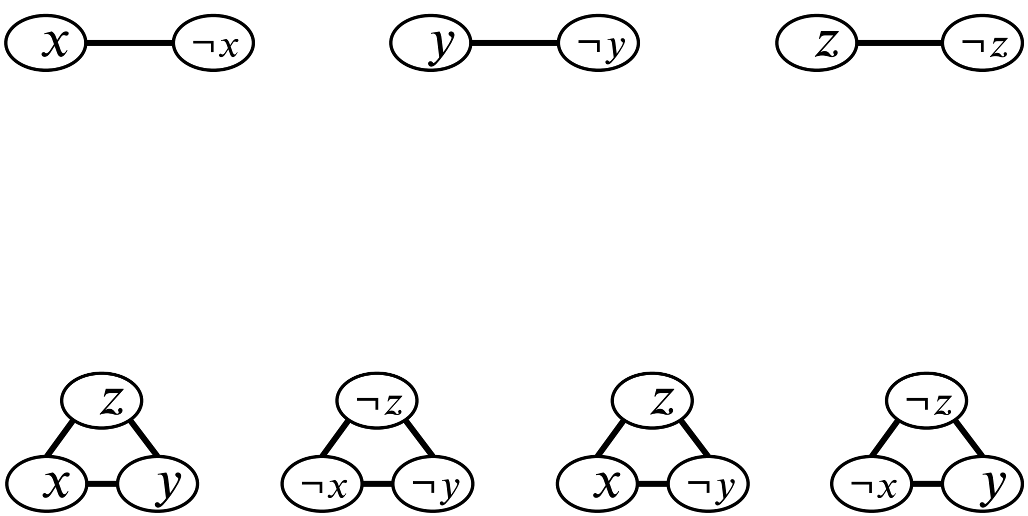
3SAT $\leq_p$ VERTEXCOVER

- Include the edge (u, v) if:
  - u is in a ***variable gadget*** and v is in a ***clause gadget***, ***AND***
  - u and v have the ***same variable label*** (e.g., x, $\neg z$, etc.)
- **Example:**

$$(x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z) \wedge (x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$

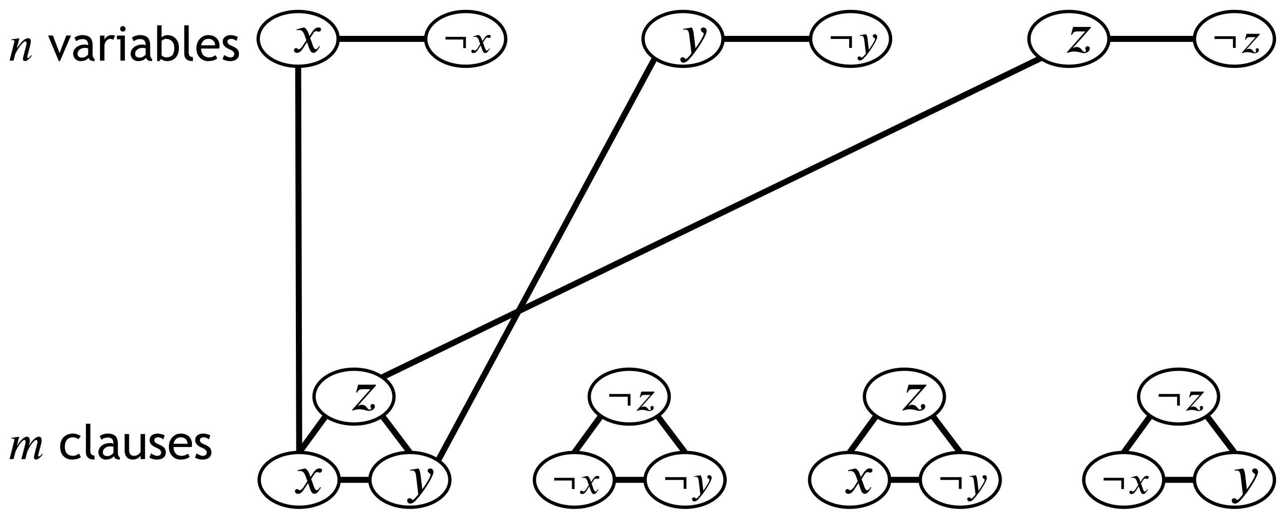
n variables

m clauses



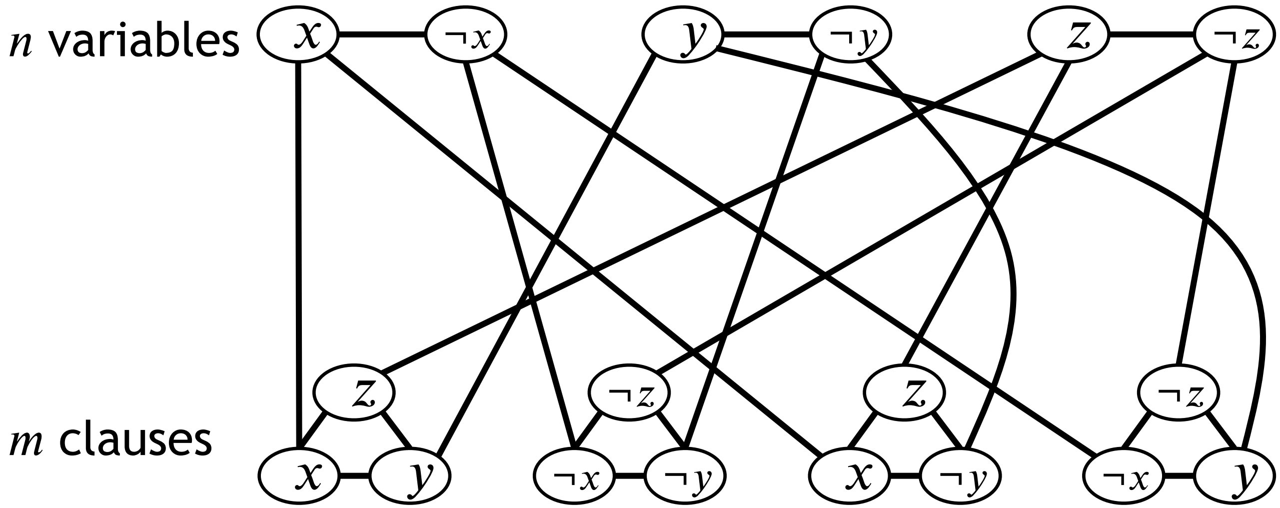
3SAT $\leq_p$ VERTEXCOVER

- Include the edge (u, v) if:
  - u is in a ***variable gadget*** and v is in a ***clause gadget*** *AND*
  - u and v have the ***same variable label*** (e.g., x, $\neg z$, etc.)
- **Example:**
  $(x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z) \wedge (x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$



$3SAT \leq_p$ VERTEXCOVER

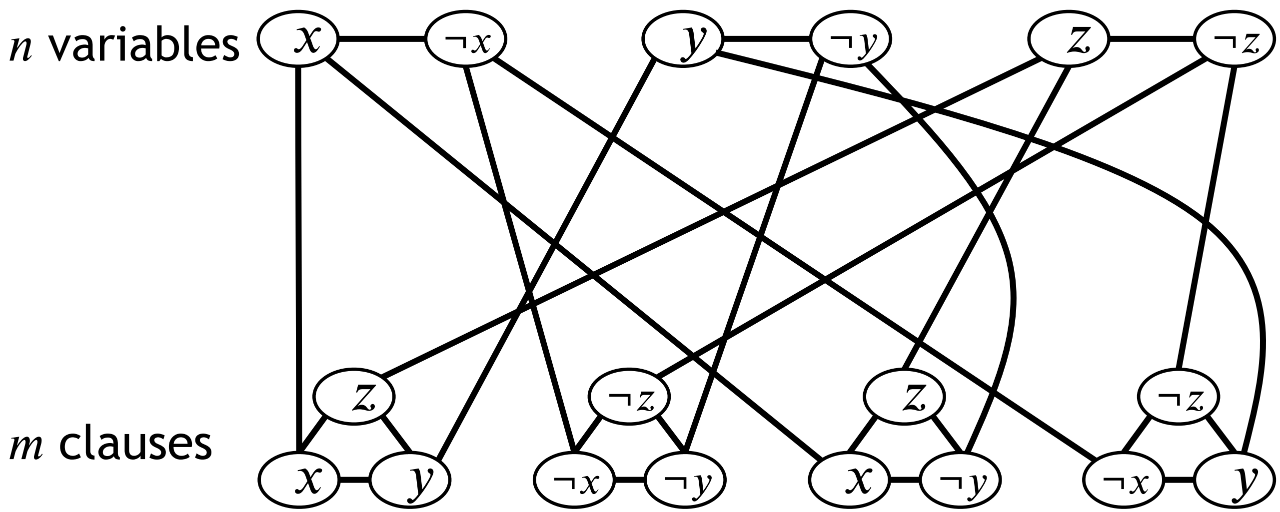
- Include the edge (u, v) if:
  - u is in a ***variable gadget*** and v is in a ***clause gadget*** *AND*
  - u and v have the ***same variable label*** (e.g., x, $\neg z$, etc.)
- **Example:**
  $(x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z) \wedge (x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$



3SAT $\leq_p$ VERTEXCOVER

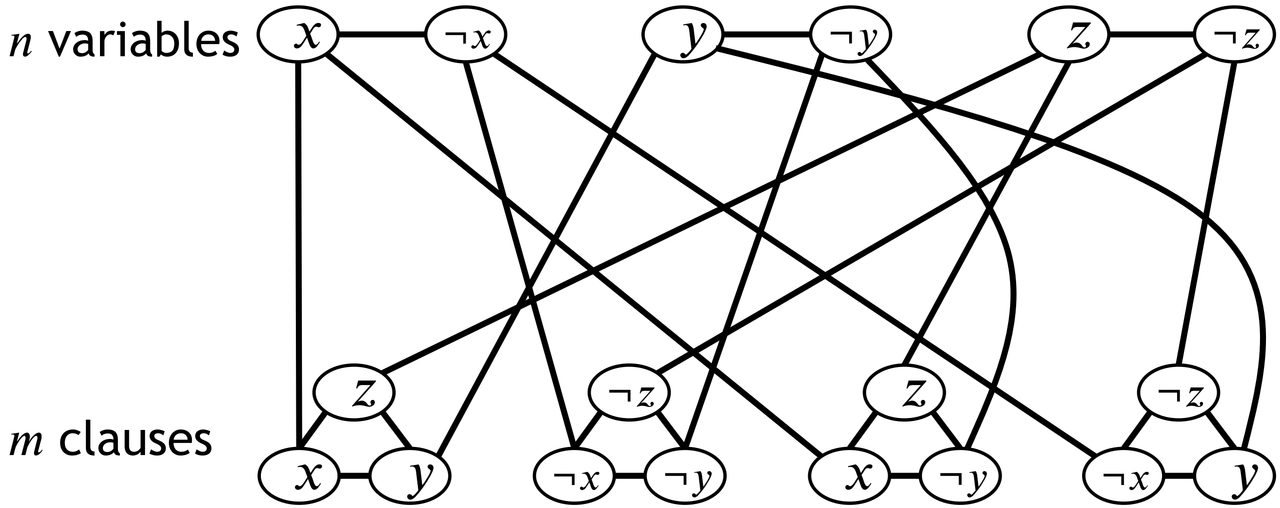
$f(\phi) = (G, n+2m)$

* **Claim:** Let ϕ be a 3CNF with n variables and m clauses; then
  1. The graph G is constructible in time $O(mn)$.
  2. ϕ is satisfiable iff G has a V.C. of size $k = n + 2m$.



$3\text{SAT} \le_p \text{VERTEXCOVER}$

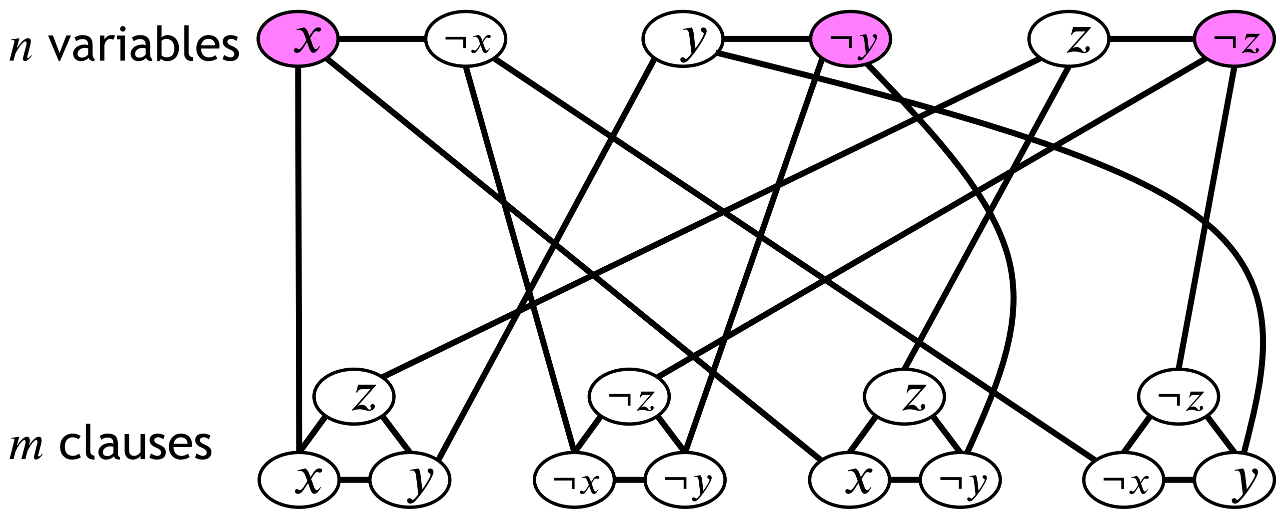
* **Observation:** Any vertex cover has size $\geq n + 2m$.
  * Needs ≥ 1 node per variable gadget and ≥ 2 nodes per clause
* **Example:**
  $(x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z) \wedge (x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$



3SAT $\leq_p$ VERTEXCOVER

* **If ϕ satisfiable**: Let α be a satisfying assignment (e.g., (1,0,0)).
  * For each variable gadget: take x if $\alpha_x = 1$ and $\neg x$ if $\alpha_x = 0$.
  * Each clause has ≥ 1 literal covered.
* **Example:**

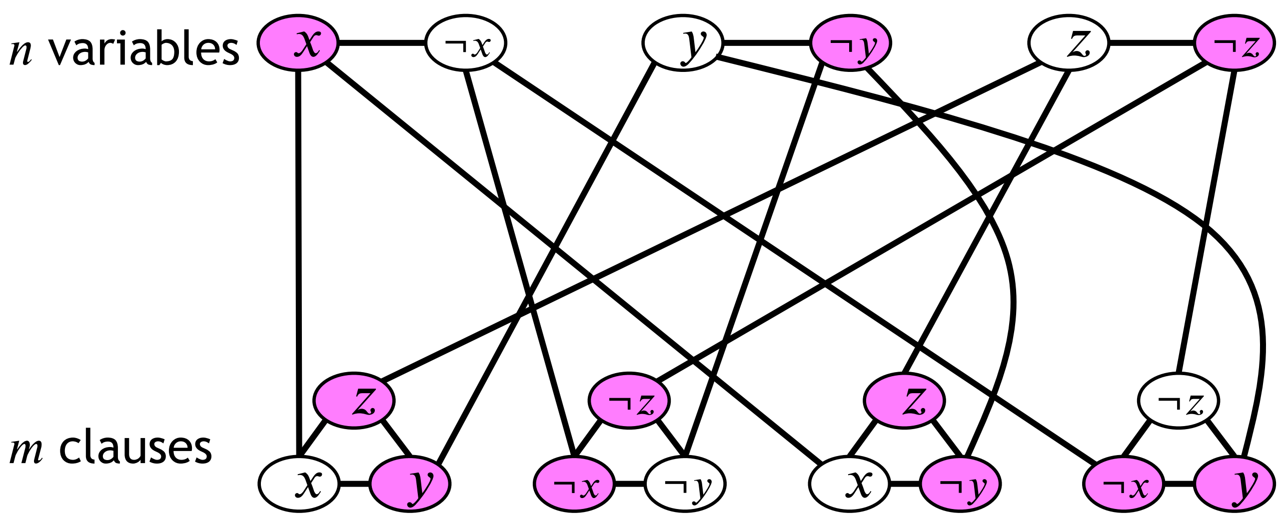
$$(x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z) \wedge (x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$$



3SAT $\leq_p$ VERTEXCOVER

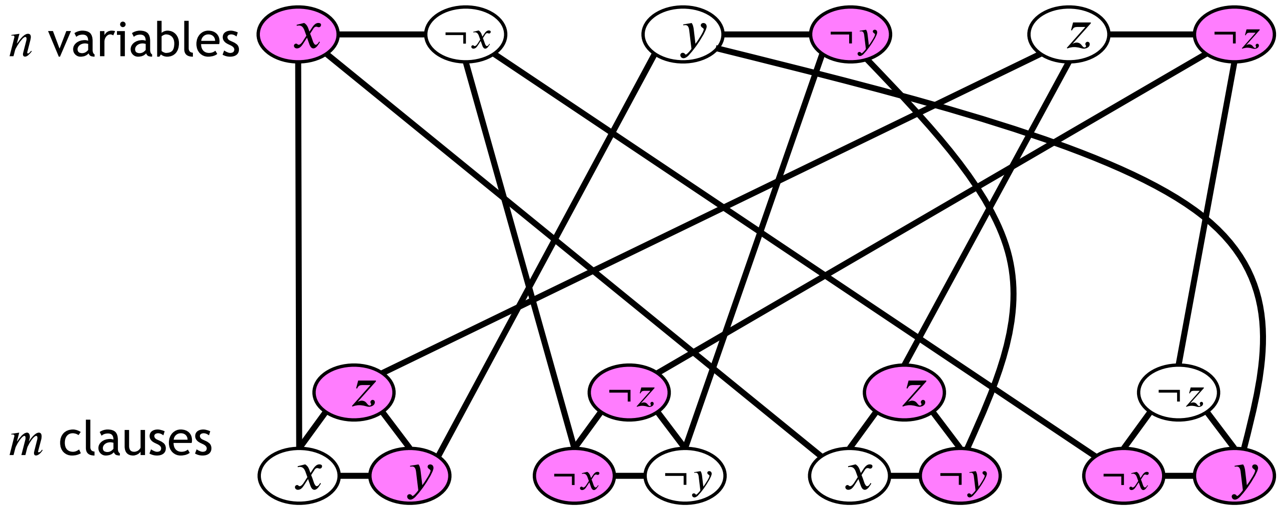
- **If ϕ satisfiable**: Let α be a satisfying assignment (e.g., (1,0,0)).
  - For each variable gadget: take x if $\alpha_x = 1$ and $\neg x$ if $\alpha_x = 0$.
  - Each clause has ≥ 1 literal covered, so take ≤ 2 more.
- **Example:**

$(x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z) \wedge (x \vee \neg y \vee z) \wedge (\neg x \vee y \vee \neg z)$



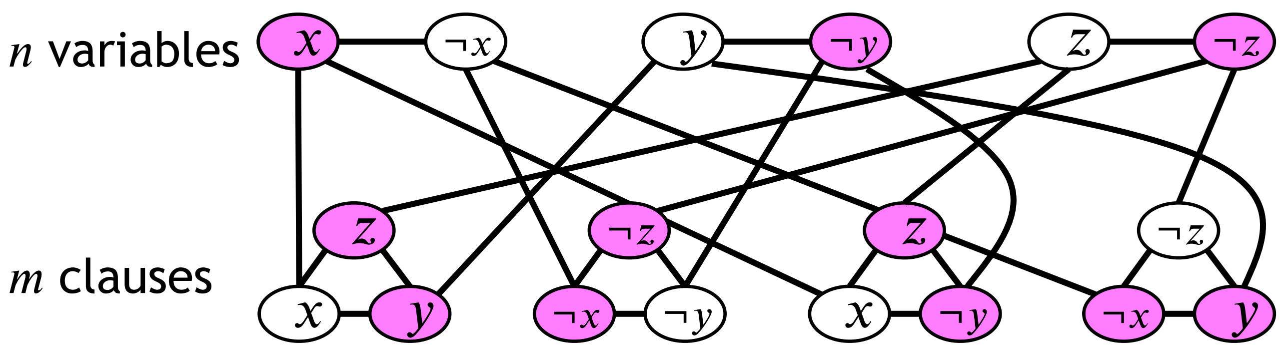
$3SAT \leq_p$ VERTEXCOVER

* **Conclusion/Claim 1:**
  $\phi \in 3SAT \Longrightarrow (G, n + 2m) \in \text{VERTEXCOVER}$

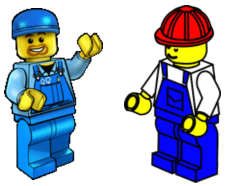


$3SAT \leq_p$ VERTEXCOVER

- **Claim 2:** $\phi \in 3SAT \Longleftarrow (G, n+2m) \in \text{VERTEXCOVER}$
  - A vertex cover of size $n + 2m$ *must* include the following:
    - Exactly 1 vertex from each variable gadget, to cover its edge.
    - Exactly 2 vertices from each clause gadget, to cover its 3 edges.
  - The 2 vertices from a clause gadget cover only 2 of the "crossing" edges.
  - So, the 3rd crossing edge is covered by the variable gadget's vertex.
  - Setting the literals from the selected vertices of the *variable* gadgets to "true" satisfies every clause, so the whole formula is satisfied.



Set Cover
(Contractor Problem)

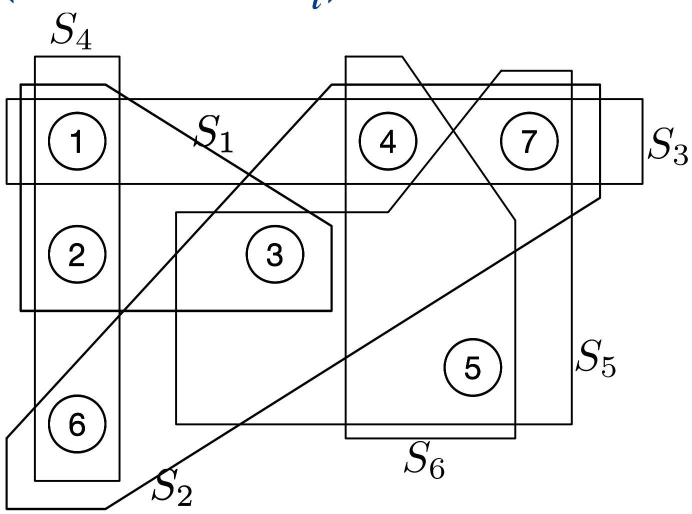


- **Problem Setup:**
  - n workers, each worker i has a set of skills S_i
  - To complete a task, we need to hire a team of workers that jointly have all the required skills.
  - What is the smallest team of workers we can hire?

- **Formally:**
  - A set of elements (all the required skills) U
  - n sets: $S_i \subseteq U$ (the set of skills worker i has)
  - Find the smallest number of S_i that "cover" U. That is, their union is U.

Set Cover

* **Example:** $U = \{1,2,3,4,5,6,7\}$
* $S_1 = \{1,2,3\}$; $S_2 = \{3,4,6,7\}$; $S_3 = \{1,4,7\}$
* $S_4 = \{1,2,6\}$; $S_5 = \{3,5,7\}$; $S_6 = \{4,5\}$
* **Question:** What is the size (number of S_i) of a smallest set cover?
* **Answer:**
  * A) 1
  * B) 2
  * C) 3
  * D) 4



Set Cover

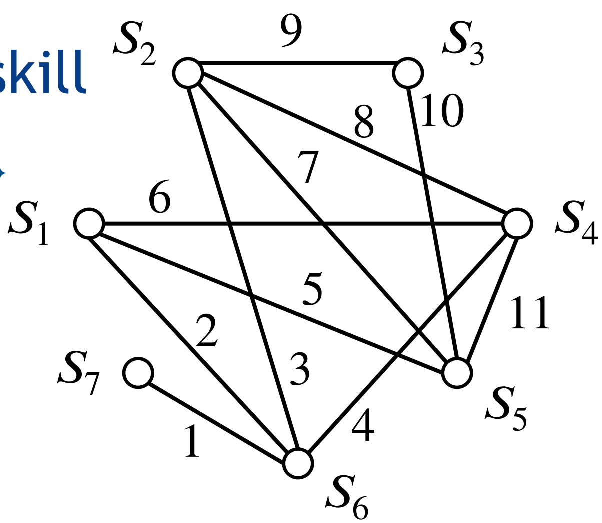
* **Definition:**

$$\text{SETCOVER} = \left\{ \begin{array}{l} \big(U, S_1, S_2, \ldots, S_n, k\big) \ : \ S_i \subseteq U \text{ and there} \\ \text{ exists a collection of size } k \text{ that covers } U \end{array} \right\}$$

* **Theorem:** SETCOVER is $\mathbf{NP}$-Complete
* **Proof:**
  1. Desired certificate $= k$ indices of S_i that cover U.
  2. $\text{VERTEXCOVER} \leq_p \text{SETCOVER}$
     * Give an efficient transform f:
       G has a VC of size $k \iff f(G,k) = (U, S_1, \ldots, S_n, k')$
       has a set cover of size k'.
* **Intuition:** vertex $\rightarrow$ worker ; edge $\rightarrow$ skill

Vertex Cover $\leq_p$ Set Cover

- **Definition:**
  $$\text{SETCOVER} = \left\{ \begin{array}{l} (U, S_1, S_2, \ldots, S_n, k) : S_i \subseteq U \text{ and there} \\ \text{exists a collection of size } k \text{ that covers } U \end{array} \right\}$$
- **Claim:** $\text{VERTEXCOVER} \leq_p \text{SETCOVER}$
- **Intuition:** vertex → worker ; edge → skill
- $U = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$
- $S_1 = \{2, 5, 6\}, S_2 = \{3, 7, 8, 9\}, \ldots$
- $k' = k$



Vertex Cover $\leq_p$ Set Cover

transformVertexCover(graph $G = (V, E)$, int k):
1. $U \leftarrow E$
2. for each vertex $v \in V$: $S_v = \emptyset$
3. for each edge $e = (u, v) \in E$:
4. Add e to both S_u and S_v
5. **return** $(U, S_v$ for $v \in V, k)$

* **Runtime:** **convertCover** is efficient.
* **Correctness:**
  $(G, k) \in \text{VERTEXCOVER} \iff (U, S_1, \ldots S_n, k) \in \text{SETCOVER}$
* Let $C \subseteq V$ be a vertex cover of size k. Then the vertices in C cover all the edges in E. Therefore, $\bigcup_{i \in C} S_i = \bigcup_{i \in C} \{e \in E : e \text{ is adjacent to } v_i\} = E = U.$
* Let W be the collection of k S_i that covers U. Then $\bigcup_{i \in W} S_i = U = E$, so the vertices i of the selected S_i form a vertex cover of size k for G.

Set Cover Post-Mortem

- We saw how to efficiently map an instance of VERTEXCOVER to an instance of SETCOVER, preserving the yes/no answer.
- Not every SETCOVER instance corresponds directly to a VERTEXCOVER instance!
  - In a VERTEXCOVER instance, every skill (edge) is shared by exactly two workers (vertices).
- VERTEXCOVER is a "specialization" of SETCOVER.
- **Observation:** An efficient algorithm for a problem also solves any specialized sub-problem efficiently.
- **Alternatively:** If a specialized sub-problem is hard, then so is the original problem.

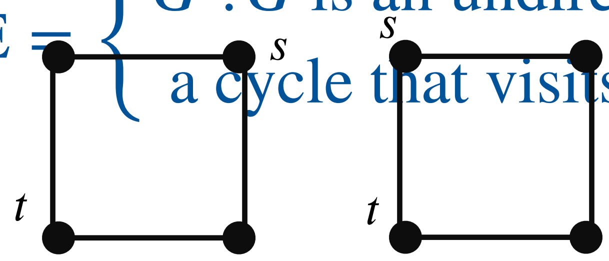
Even More **NP**-C languages!

- **Definition:** A ***Hamiltonian path*** from s to t in a graph G is a path that starts at s, ends at t, and visits each vertex in G **exactly** once.
- **Definition:**

$$\text{HAMPATH} = \left\{ \begin{array}{l} (G, s, t) : G \text{ is an undirected graph} \\ \text{with a Hamiltonian path from } s \text{ to } t \end{array} \right\}$$

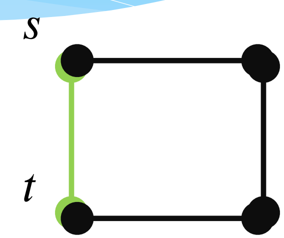
- **Fact:** HAMPATH is **NP**-Complete (grungy proof omitted).
- **Definition:**

$$\text{HAMCYCLE} = \left\{ \begin{array}{l} G : G \text{ is an undirected graph with} \\ \text{a cycle that visits all vertices once} \end{array} \right\}$$

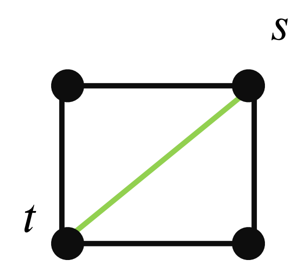


HAMCYCLE is **NP**-Complete

* **Claim:** $\text{HAMPATH} \leq_p \text{HAMCYCLE}.$
* We need to show an algorithm $f(G, s, t) = G'$ such that:
  1. f is polynomial-time computable
  2. $(G, s, t) \in$ HAMPATH $\iff$ $G' \in$ HAMCYCLE



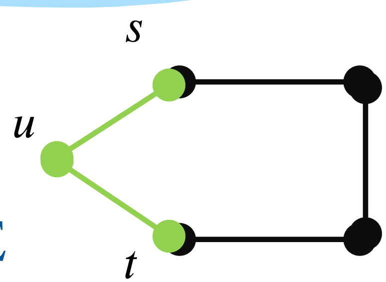
* **Attempt 1:** Given (G, s, t), define $G' = G \cup (s, t)$.
* **Claim:** $(G, s, t) \in \text{HAMPATH} \Longrightarrow G' \in \text{HAMCYCLE}$
* **Proof:** Obvious (right?).
* **Question:** What about $\Longleftarrow$ direction?



* **Example:** $G' \in \text{HAMCYCLE}$, however $(G, s, t) \notin \text{HAMPATH}.$
  * **Why:** We can't get an s->t Ham path from a Ham cycle, because the cycle might not go through the "new" edge!

HAMCYCLE is NP-Complete

transformGraph(graph $G = (V, E)$, vertex s, vertex t):
1. $V' \leftarrow V \cup \{u\}$ // u is a new vertex not in G
2. $E' \leftarrow E \cup \{(t, u), (u, s)\}$
3. **return** $G' = (V', E')$

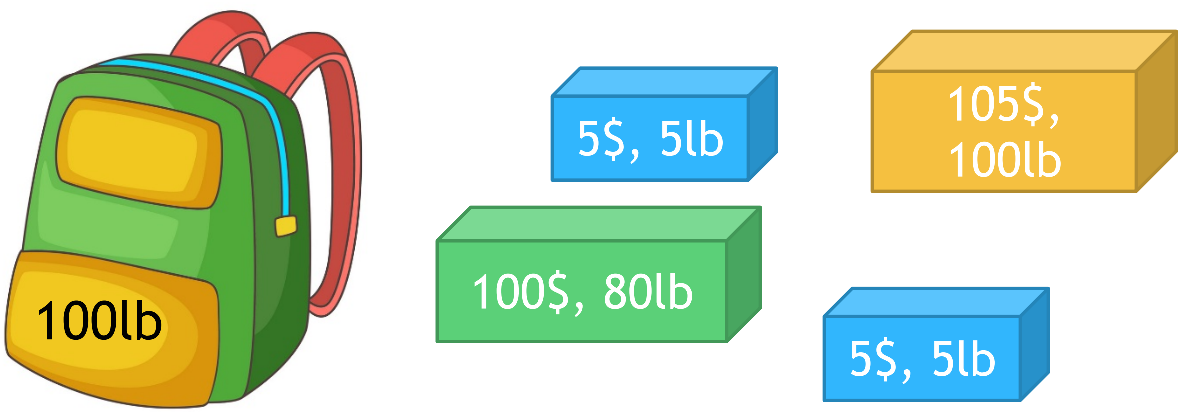


- **Claim:** $\text{HAMPATH} \leq_p \text{HAMCYCLE}$.
- Given (G, s, t), define $G' = G \cup (t, u) \cup (u, s)$.
- **Claim:** $(G, s, t) \in \text{HAMPATH} \iff G' \in \text{HAMCYCLE}$
- **Proof:**
- $(G, s, t) \in \text{HAMPATH} \implies \exists p = (s, \ \ldots, \ t)$, a Hamiltonian path in G.
  - Then $(s, \ldots, t, u, s)$ is a Hamiltonian cycle in $G' \implies G' \in \text{HAMCYCLE}$.
- $G' \in \text{HAMCYCLE} \implies \exists p = (s, \ \ldots, \ u, \ldots, s)$, a Hamiltonian cycle in G'.
  - By reversing p if needed, $p = (s, \ldots, t, u, s)$.
  - Remove final two edges from p to get an s->t Ham path in G:
  - Final conclusion: $(G, s, t) \in \text{HAMPATH}$.

Knapsack Problem

(Theft Problem)

- **Problem:** Given a set of items, each with a value and a weight, is there a subset of items with total value at least V and total weight at most W?
- **Exercise:** Knapsack problem is $\mathbf{NP}$-Complete.



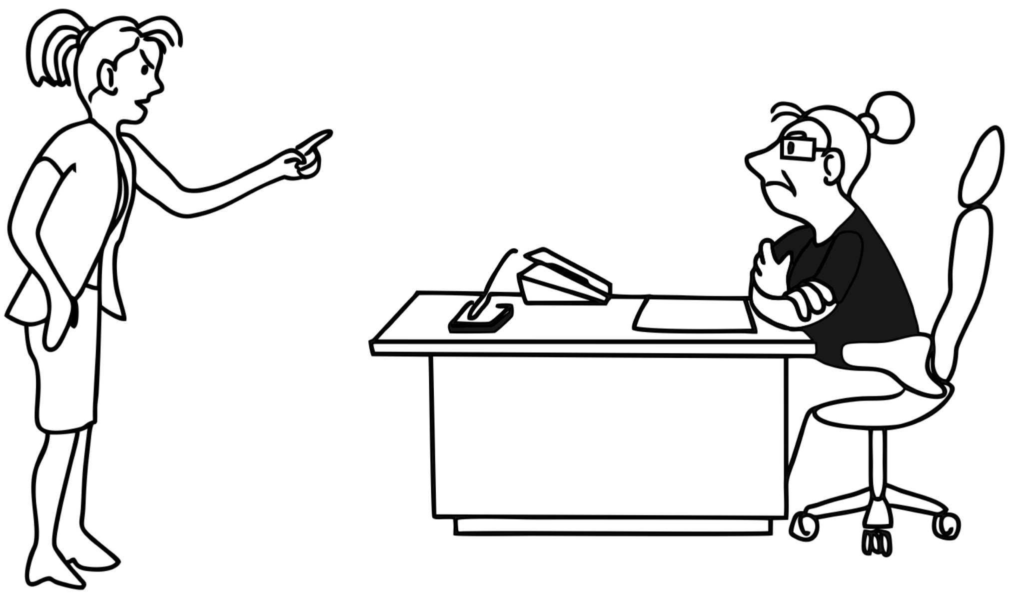
NP-Completeness is Everywhere

- **Constraint Satisfaction:** SAT, 3SAT
- **Covering Problems:** Vertex Cover, Set Cover
- **Coloring Problem:** 3-colorability of a graph
- **Scheduling Problems:** optimal class schedules
- **Model Checking:** context-bounded reachability
- **Social Networks:** Clique, Maximal-Cut
- **Routing:** Longest Path, HAMPATH, TSP
- **Games:** Sudoku, Battleship, Super Mario, Pokémon
- … One ALGORITHM would rule them all!

NP-Completeness

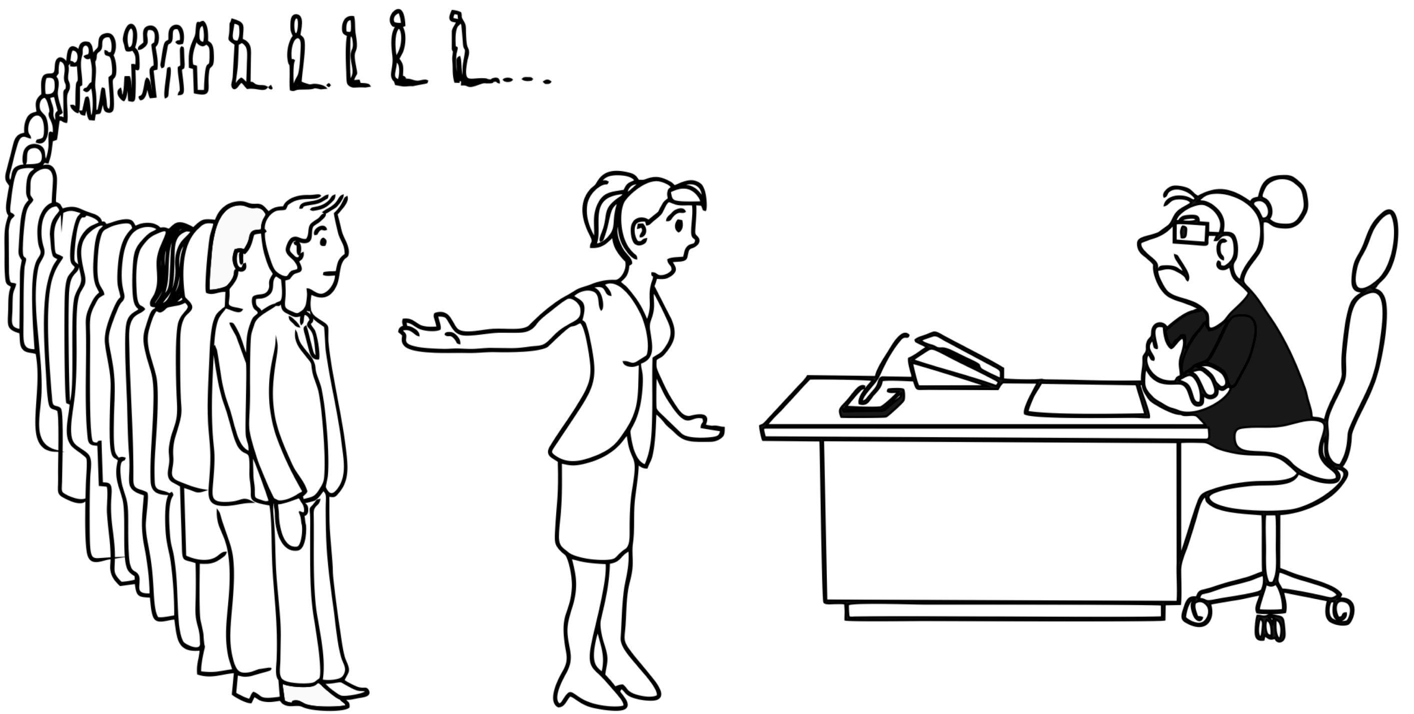
"I can't find an efficient algorithm, I guess I'm just too dumb."

NP-Completeness



“I can’t find an efficient algorithm, because no such algorithm is possible!”

NP-Completeness



"I can't find an efficient algorithm, but neither can all these famous people."