1. **Cyclic paths of hell.** A *Hamiltonian cycle* in a graph is a cycle that visits every vertex exactly once. A *Hamiltonian cycle* in a graph is a path that visits every vertex exactly once, but it need not be a cycle (the last vertex in the path may not be adjacent to the first vertex in the path.)

   Consider the following three problems:

   - *Directed Hamiltonian Cycle* problem: checks whether a Hamiltonian cycle exists in a *directed* graph,
   - *Undirected Hamiltonian Cycle* problem: checks whether a Hamiltonian cycle exists in an *undirected* graph.
   - *Undirected Hamiltonian Path* problem: checks whether a Hamiltonian path exists in an *undirected* graph.

a. Give a polynomial time reduction from the *directed* Hamiltonian cycle problem to the *undirected* Hamiltonian cycle problem.

Solution: For any arbitrary directed graph $G_d := \left\{V_d, E_d\right\}$, construct the following undirected graph $G_u := \left\{V_u, E_u\right\}$:

- $V_u := \left\{ v_{in}, v_{mid}, v_{out} \middle| v \in V_d \right\}$. For each of the vertices in the directed graph, we split them into a triplet of in, mid, and out.
- $E_u := \left\{ \left(u_{out}, v_{in}\right) \middle| (u,v) \in E_d \right\} \cup \left\{ \left(v_{in}, v_{mid}\right), \left(v_{mid}, v_{out}\right) \middle| v \in V_d \right\}$. For each of the triplets that comes from the same vertex, we connect them in the order of in-mid-out. The directed edges in the V_d become the undirected ones that connect out and in between corresponding triplets.

Notice that $\left|V_u\right| = 3\left|V_d\right|$ and $\left|E_u\right| = \left|E_d\right| + 2\left|V_d\right|$, so this reduction is linear.

$\Rightarrow$: Suppose that in G_d there exists a Hamiltonian cycle $C_d := \left(c_1, c_2, \ldots, c_{\left|V_d\right|}\right)$, where $c_i \in V_d$. Then in G_u there should also exist

$$C_u := \left(c_{1in}, c_{1mid}, c_{1out}, c_{2in}, c_{2mid}, c_{2out}, \ldots, c_{\left|V_d\right|_{in}}, c_{\left|V_d\right|_{mid}}, c_{\left|V_d\right|_{out}}\right),$$

which is a Hamiltonian cycle in G_u.

$\Leftarrow$: Suppose that in G_u there exists a Hamiltonian cycle C_u'. By definition, within each of the triplets there should only be a path of order in-mid-out, and between two triplets there should only be an edge of out-in. Thus, C_u' should always be of the following form

$$C_u' := \left(c_{1in}', c_{1mid}', c_{1out}', c_{2in}', c_{2mid}', c_{2out}', \ldots, c_{\left|V_d\right|_{in}}', c_{\left|V_d\right|_{mid}}', c_{\left|V_d\right|_{out}}'\right),$$

which corresponds to a Hamiltonian cycle $C'_d := \left(c'_1, c'_2, \ldots, c'_{|V_d|}\right)$ in G_d.

$\blacksquare$

b. Give a polynomial time reduction from the *undirected* Hamiltonian Cycle to *directed* Hamiltonian cycle.

Solution: This reduction is simpler than the previous one. Given an instance G of undirected Hamiltonian cycle, Let G' be the directed graph with the same vertices as G and containing edges $u \to v$ and $v \to u$ for every edge $uv \in G$.

($\Rightarrow$) If C is a cycle in G, then C is also a cycle in the directed graph G'. For every $u \to v \in G'$, the edge uv is in G.

($\Leftarrow$) If C is a cycle in G', then C is also a cycle in the original graph G. For every $uv \in G$, the edge $u \to v \in G'$ by the construction.

■

c. Give a polynomial-time reduction from an undirected Hamiltonian *path* to an undirected Hamiltonian *cycle*.

Solution: Let the input to this problem be an undirected graph G. The goal is to produce G' such that G has a Hamiltonian path if G' has a Hamiltonian cycle.

This can be done by adding a vertex v with edges to every vertex in the original graph G, this will be G'.

$\Rightarrow$: If there exists a Hamiltonian path P in G, starting with vertex s and ending with vertex t. Then $[v, s, P, t, v]$ is a Hamiltonian cycle in G'.

$\Leftarrow$: In the other case, if C is the Hamiltonian cycle in G', then removing v from C will return a Hamiltonian path in G.

■

2. **Pls No Yapping. Just Circle Em'.**

   1. **True/False:** If L is an NP-complete language and $L \in P$, then $P = NP$.

   2. **True/False:** There exists a polynomial-time reduction from every problem in NP to every problem in P.

   3. **True/False:** If a problem is both NP-hard and co-NP-hard, then it must be in NP.

   4. **True/False:** If there is a polynomial-time reduction from problem A to problem B and B is in NP, then A must also be in NP.

   5. **True/False:** If a problem is solvable in polynomial space, then it is also solvable in polynomial time.

Solution:

1. True. If L is an NP-complete language and $L \in P$, then $P = NP$. This is because NP-complete problems are the hardest problems in NP, and if any NP-complete problem is in P, then all problems in NP are in P.

2. False. There does not necessarily exist a polynomial-time reduction from every problem in NP to every problem in P. A reduction must map a problem in NP to another problem in NP or a problem in P to another problem in P.

3. False. If a problem is both NP-hard and co-NP-hard, it does not necessarily imply it is in NP. Being NP-hard means every problem in NP can be reduced to it in polynomial time, and co-NP-hard means every problem in co-NP can be reduced to it in polynomial time.

4. True. If there is a polynomial-time reduction from problem A to problem B and B is in NP, then A must also be in NP. This is because the reduction allows us to transform A into B, and since B is in NP, the transformed instance can be verified in polynomial time, implying that A is also in NP.

5. False. If a problem is solvable in polynomial space, it does not necessarily mean it is solvable in polynomial time. Polynomial space includes problems that might take exponential time but use polynomial memory, such as PSPACE-complete problems. ■

3. **Cooking Levin so hard he won't even be able to SAT** Special4SAT is a version of 4SAT in which each literal appears at most once in Φ. We write a solver for Special4SAT as follows:

For each literal q in Φ:

- Flip a fair coin
- If heads, assign $q = \text{True}$
- If tails, assign $q = \text{False}$

In terms of m, what is the expected number of clauses satisfied? Write the most efficient solver for Special4SAT.

Solution: Expected Number of Clauses Satisfied

Each clause in Φ is a disjunction (OR) of exactly four literals. Since each literal is independently assigned `True` or `False` with equal probability, we calculate the probability of satisfying a clause.

Probability of Satisfying a Single Clause

For a clause with four literals:

- Probability that a single literal is `False`: $\frac{1}{2}$
- Probability that all four literals are `False`: $\left(\frac{1}{2}\right)^4 = \frac{1}{16}$
- Probability that at least one literal is `True` (clause is satisfied): $1 - \frac{1}{16} = \frac{15}{16}$

Expected Number of Satisfied Clauses

Let m be the number of clauses in Φ. The expected number of satisfied clauses is:

$$\mathbb{E}[\text{Number of satisfied clauses}] = m \times \frac{15}{16}$$

The most efficient solver is just to **return True**! Why? since every literal appears once, just set every variable to True and the formula is always satisfiable.

■