CS340: Theory of Computation Sem I 2021-22

Lecture Notes 19: NP-Completeness

Raghunath Tewari IIT Kanpur

1 Definition of NP-Completeness

We again shift our focus back to reductions. Specifically we consider a type of reductions known as polynomial time (or polytime) reductions.

Definition 1.1. A language L_1 *polynomial time reduces* to a language L_2 (denoted as $L_1 \leq_p L_2$) if there is a polynomial time computable function $f : \Sigma^* \longrightarrow \Sigma^*$ such that for all $x \in \Sigma^*$,

$$x \in L_1 \iff f(x) \in L_2.$$

Definition 1.2.

- A language L is said to be NP-*hard* if for all languages $L' \in \mathsf{NP}$ we have that $L' \leq_p L$.
- A language L is said to be NP-*complete* if L is NP-hard and $L \in \mathsf{NP}$.

Therefore, in some sense NP-complete problems are the hardest problems present in NP.

Note 1. A reduction should always be of "lesser power" than the complexity class under consideration. Otherwise one can always cheat by solving the problem first by using the power of reduction and then mapping it to a trivial Yes/No instance.

2 Properties of NP-Completeness

Theorem 1. *If L is* NP*-complete and $L \in \mathsf{P}$ then $\mathsf{P} = \mathsf{NP}$.*

Proof. Let L' be a language in NP. Since L is NP-complete therefore $L' \leq_p L$ via a polynomial time computable function, say f. Also let M be a polynomial time TM for L (since $L \in \mathsf{P}$).
We construct a polynomial time machine M' for L'. On input x, M' first constructs $f(x)$ then simulates M on $f(x)$ and outputs whatever M outputs.
Since f is polynomial time computable and M is a polynomial time TM therefore M' is also a polynomial time TM. Now,

$$x \in L' \iff f(x) \in L \iff M \text{ accepts } f(x) \iff M' \text{ accepts } x.$$

Therefore $\mathsf{NP} \subseteq \mathsf{P}$. □

Theorem 2. *If A is* NP*-hard and $A \leq_p B$ then B is* NP*-hard.*

We first prove the following Lemma.

Lemma 3. *If $L_1 \leq_p L_2$ and $L_2 \leq_p L_3$ then $L_1 \leq_p L_3$.*

Proof. Suppose $L_1 \leq_p L_2$ and $L_2 \leq_p L_3$ via polynomial time computable functions, say f and g respectively.
We construct a polynomial time computable function h that on input x outputs $h(x) = g(f(x))$. On input x, first compute $f(x)$ using the polytime machine for f. Then compute $g(f(x))$ using the polytime machine for g. Output $g(f(x))$.
Since f and g are polynomial time computable therefore h is also a polynomial time computable (sum of two polynomials is a polynomial). Now,

$$x \in L_1 \iff f(x) \in L_2 \iff g(f(x)) \in L_3 \iff h(x) \in L_3.$$

Therefore $L_1 \leq_p L_3$. □

Proof of Theorem 2. Let L be a language in NP. Since A is NP-hard therefore $L \leq_p A$. Also we have that $A \leq_p B$. Applying Lemma 3 we have $L \leq_p B$. Therefore B is also NP-hard. □

3 SAT is NP-complete

3.1 Terminology and Notations

- *Boolean variables* are variables that take the value 0 or 1.
- *Boolean operators*: AND ($\wedge$), OR ($\vee$) and NOT ($\neg$)
- A *Boolean formula* is an expression that is defined inductively over Boolean variables and Boolean operators.
- Examples
  (i) $\phi_1 : (x \wedge \neg y) \vee \neg x$
  (ii) $\phi_2 : (x \vee y) \wedge (\neg x \vee z) \wedge (\neg y \vee \neg z)$
- A *truth assignment* τ is an assignment of values 0 or 1 to every variable. Specifically, $\tau : \{x_1, x_2, \ldots, x_n\} \longrightarrow \{0, 1\}$.
- Evaluation of a Boolean formula ϕ on a truth assignment τ is the assignment of the values of τ to the variables in ϕ so that it evaluates to 0 or 1. It is denoted as $\phi(\tau)$.
- Example of a truth assignment, τ, for ϕ_2 above. $\tau(x) = 1$, $\tau(y) = 0$, $\tau(z) = 0$. Then $\phi_2(\tau) = 0$ since the second clause evaluates to 0.
- A Boolean formula ϕ is said to be *satisfiable* if there exists a truth assignment τ such that $\phi(\tau) = 1$.
- SAT $= \{\langle\phi\rangle \mid \phi$ is a satisfiable Boolean formula$\}$.
- A *literal* is a variable x or its negation $\neg x$.
- A *clause* is an OR of literals.
- A *CNF formula* is an AND of clauses.
- A *3CNF formula* is CNF formula where every clause has exactly 3 literals.
- 3SAT $= \{\langle\phi\rangle \mid \phi$ is a satisfiable 3CNF Boolean formula$\}$.

Exercise 1. Show that SAT is in NP.

3.2 Cook-Levin Theorem

We next show that SAT is NP-hard. This was proven independently by Stephen Cook and Leonid Levin in 1971 and 1973 respectively. It is one the fundamental results in computer science and the P versus NP problem originated from this result.

Theorem 4 (Cook-Levin Theorem)**.** SAT *is* NP*-hard.*

Proof Idea. Let $L \in \mathsf{NP}$ accepted by a NDTM N. Suppose the running time on N is n^k on a input of length n. Note that the machine N can visit at most n^k cells on its tape. Given an input x to N such that $|x| = n$, we will construct a Boolean formula ϕ such that $x \in L$ if and only if $\phi \in \mathsf{SAT}$.
Consider an $n^k \times n^k$ matrix, say M, whose i-th row stores the i-th configuration of N on x. Each element of M is a symbol from the set $C := Q \cup \Gamma$, that is, it contains a state symbol or a tape symbol.
Remark. For a NDTM N, every $n^k \times n^k$ matrix corresponds to *some* computation path of N on an input x.

- Construct a formula ϕ_{cell} which is satisfiable if and only if every cell of the matrix holds exactly one symbol. Define Boolean variables $x_{i,j,s}$, where $1 \leq i, j \leq n^k$ and $s \in C$, such that, $x_{i,j,s} = 1$ if and only if $M(i, j)$ contains s. Then

$$\phi_{cell} = \bigwedge_{1 \leq i,j \leq n^k} \left(\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{s,t \in C,\ s \neq t} (\neg x_{i,j,s} \vee \neg x_{i,j,t}) \right) \right).$$

  Essentially the above formula encodes the fact that for cell (i, j), $M(i, j)$ stores some symbol s and $M(i, j)$ does not store two different symbols s and t.

- Next construct a formula ϕ_{start} which is satisfiable if and only if the first row of M encodes the start configuration. That is,

$$\phi_{start} = x_{1,1,q_0} \wedge x_{1,2,x_1} \wedge x_{1,3,x_2} \wedge \ldots \wedge x_{1,n+1,x_n} \wedge x_{1,n+2,\sqcup} \ldots \wedge x_{1,n^k,\sqcup}.$$

- Next construct a formula ϕ_{accept} which is satisfiable if and only if some row of M encodes an accept configuration. That is,

$$\phi_{accept} = \bigvee_{1 \leq i,j \leq n^k} x_{i,j,q_{accept}}.$$

- Finally we define a formula ϕ_{move} which is satisfiable if and only if the $(i + 1)$-th row follows legally (according to the transition function of N) from the i-th row of M. To check this we consider every possible 2×3 window in M and verify whether its a legal window (that is the bottom row follows legally from the top row). We consider a window of dimension 2×3 because (i) a configuration depends only on the previous configuration (hence we consider two rows in every window), and (ii) between two successive configurations only left cell or the right cell of a given cell gets modified (hence we consider three columns in every window). Hence define

$$\phi_{move} = \bigwedge_{\substack{1 \le i < n^k \\ 1 < j < n^k}} \left(\bigvee_{\substack{a_1, \ldots, a_6 \\ \text{is a legal} \\ \text{window}}} (x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j+1,a_3} \wedge x_{i+1,j-1,a_4} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a_6}) \right).$$

- The final formula ϕ is

$$\phi = \phi_{cell} \wedge \phi_{start} \wedge \phi_{accept} \wedge \phi_{move}.$$

The key property that we use in the above construction is that computation of a TM happens *locally*. This allows us to restrict ourselves to a constant sized window at every step.

Exercise 2. Verify that the size of ϕ constructed in the above proof is $O(n^{2k})$. Also verify that the construction takes polynomial time.

$\square$

Exercise 3. Show that $\mathsf{SAT} \le_p \mathsf{3SAT}$.

4 More NP-complete Problems

1. Clique is NP-complete.

   - Let $G = (V, E)$ be a graph. A subset $S \subseteq V$ forms a *clique* if for every pair of vertices $u, v \in S$, $(u, v) \in E$.
   - $\mathsf{Clique} = \{\langle G, k \rangle \mid G$ has a clique of size at least $k\}$.

   Theorem 5. Clique *is* NP*-complete.*

   (a) Showing that the problem is in NP.

   Exercise 4. Show that Clique is in NP.

   (b) Choosing a suitable NP-complete problem. We will show that

   $$\mathsf{3SAT} \le_p \mathsf{Clique}.$$

   (c) The reduction. Let ϕ be an instance of 3SAT where

   $$\phi := (l_{11} \vee l_{12} \vee l_{13}) \wedge (l_{21} \vee l_{22} \vee l_{23}) \wedge \ldots \wedge (l_{m1} \vee l_{m2} \vee l_{m3}).$$

   Construct a graph $G_\phi = (V, E)$ as follows:

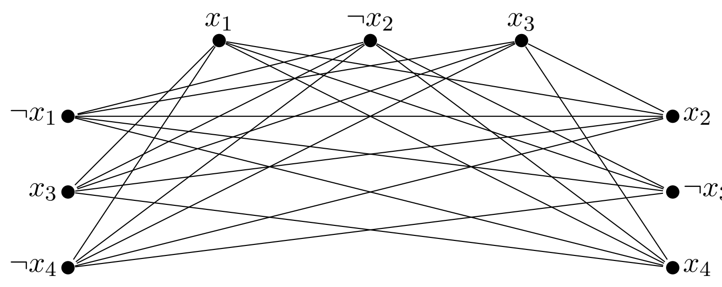
   $$\begin{aligned} V &= \{v_{ij} \mid 1 \le i \le m,\ 1 \le j \le 3\} \qquad \text{i.e. one vertex per literal} \\ E &= \{(v_{ij}, v_{i'j'}) \mid i \ne i' \text{ and } l_{ij} \ne \neg l_{i'j'}\} \end{aligned}$$

   In other words, we place an edge between two vertices if their corresponding literals belong to different clauses and one literal is not the negation of the other literal.

Set $k = m$.

Given below is an example of this construction.

$$\phi := (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_3 \vee \neg x_4) \wedge (x_2 \vee \neg x_3 \vee x_4)$$



Graph G_ϕ

(d) The construction of G_ϕ from ϕ can be achieved in quadratic time in the length of ϕ.

(e) Proof of correctness.

If ϕ is satisfiable the there exists a truth assignment τ such that each clause has at least one satisfying literal. Let $L = \{l_1, l_2, \ldots l_m\}$ be a set of m satisfying literals, one from each clause. And $U = \{v_1, v_2, \ldots v_m\}$ be the corresponding vertices in the graph G_ϕ. Since every literal $l_i \in L$ belongs to a different clause and every literal is a satisfying literal, therefore it must be that for every $i \neq j$, $l_i \neq \neg l_j$. Hence for every $i \neq j$, (v_i, v_j) is an edge in E. Therefore the subgraph induced by U is a complete subgraph.

For the other direction, let G_ϕ have a complete subgraph of size at least k. Since there are no edges in G_ϕ between the vertices of the same clause and since the number of clauses is $m(= k)$ therefore G_ϕ cannot have a complete subgraph of size strictly more than k. Let $L = \{l_1, l_2, \ldots l_m\}$ be the literals corresponding to the complete subgraph. We will construct a satisfying assignment as follows. For a literal $l \in L$, if $l = x_i$ then set $x_i = 1$ and if $l = \neg x_i$ then set $x_i = 0$. Other variables can be assigned any value 0 or 1. This assignment is consistent because both a literal l and $\neg l$ cannot be in the set L because L corresponds to a complete graph. Also since every literal in L is from a different clause, the above assignment is a satisfying assignment for ϕ.

2. IndSet is NP-complete.

- Let $G = (V, E)$ be a graph. A subset $S \subseteq V$ is an *independent set* if for every pair of vertices $u, v \in S$, $(u, v) \notin E$.
- $\mathsf{IndSet} = \{\langle G, k\rangle \mid G$ has an independent set of size at least $k\}$.

Theorem 6. IndSet *is* NP*-complete.*

(a) Showing that the problem is in NP.

Exercise 5. Show that IndSet is in NP.

(b) Choosing a suitable NP-complete problem. We will show that

$$\mathsf{Clique} \leq_p \mathsf{IndSet}.$$

(c) The reduction. Let $\langle G = (V, E), k\rangle$ be an instance of Clique. We will construct an instance of IndSet, $\langle G' = (V', E'), k'\rangle$ as follows:

$$\begin{aligned} V' &= V \\ E' &= \{(u, v) \mid (u, v) \notin E\} \end{aligned}$$

In other words, G' is the complement of the graph G.
Set $k' = k$.

(d) The construction of G' can be achieved in linear time from the graph G.

(e) Proof of correctness.
G has a clique of size m if and only if the same set of vertices forms an independent set of size m in G'.

3. VertexCover is NP-complete.

- Let $G = (V, E)$ be a graph. A subset $S \subseteq V$ is a *vertex cover* if every edge in G has at least one of its endpoints in S.
- VertexCover $= \{\langle G, k\rangle \mid G$ has a vertex cover of size at most $k\}$.

Exercise 6. Show that VertexCover is in NP-complete.

4. HamPath is NP-complete.

- Let $G = (V, E)$ be a directed graph. A *Hamiltonian path* in G is a path that visits every vertex exactly once.
- HamPath $= \{\langle G, s, t\rangle \mid G$ is a directed graph having a Hamiltonian path from s to $t\}$.

Theorem 7. HamPath *is* NP*-complete.*

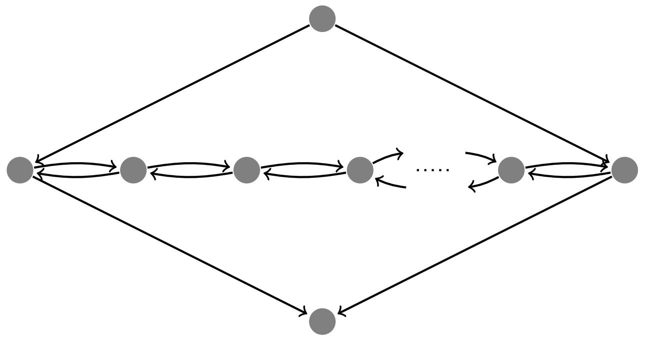
(a) Showing that the problem is in NP.

Exercise 7. Show that HamPath is in NP.

(b) Choosing a suitable NP-complete problem. We will show that

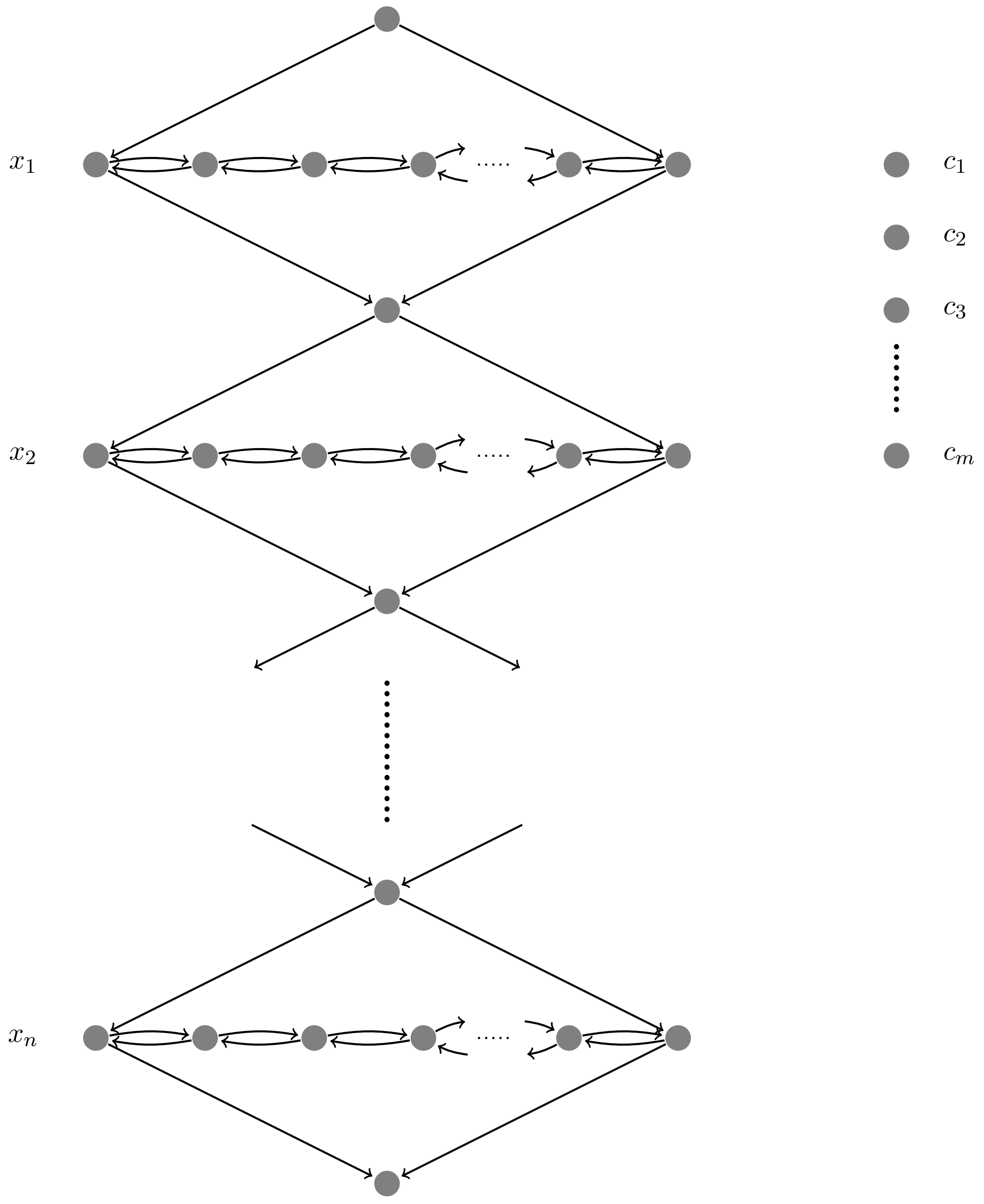
$$\textsf{3SAT} \leq_p \textsf{HamPath}.$$

(c) The reduction. Let ϕ be an instance of 3SAT having n variables $x_1, \ldots, x_n$ and m clauses $c_1, \ldots, c_m$. Construct the following gadget corresponding to each variable.

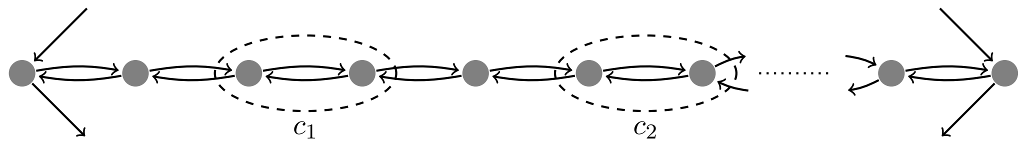

Gadget for variable x_i

The number of vertices that appear on the horizontal row is $3m + 1$ excluding the two end vertices. For every clause we add a single vertex.

The gadgets corresponding to the variables are connected as shown in the figure below together with the vertices for the m clauses.

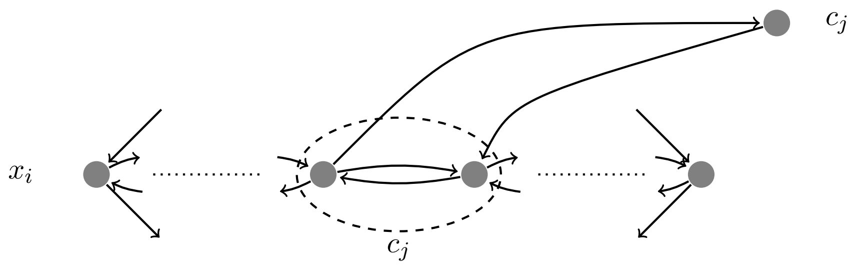


The horizontal row of a variable gadget has $3m + 1$ vertices with two adjacent vertices for each clause and a single *buffer vertex* in the middle. The following figure gives a better explanation.



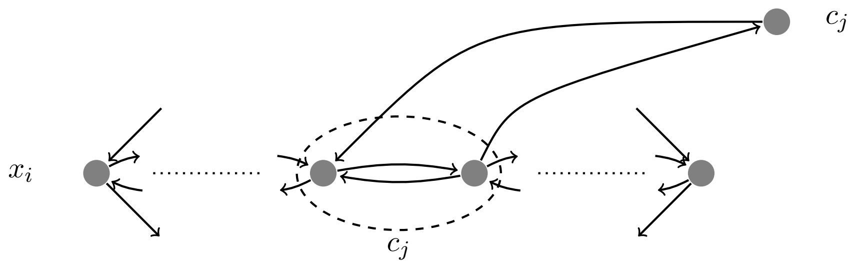
A variable gadget with the vertices for each clause

If a variable x_i appears in a clause c_j in its positive form we connect the clause gadget to its corresponding pair of variables in the variable gadget by the following two edges (shown in the figure below).



When clause c_j contains x_i

Now if a variable x_i appears in a clause c_j in its negated form we connect the clause gadget to its corresponding pair of variables in the variable gadget by the following two edges, say *clause edges* (shown in the figure below).



When clause c_j contains $\neg x_i$

Set s as the top vertex in the gadget corresponding to x_1 and t as the bottom vertex in the gadget corresponding to x_n. This ends the construction of the graph corresponding to the formula ϕ. Let us name the graph G_ϕ.

(d) The construction of G_ϕ from ϕ can be achieved in time $O(mn)$ which is also the size of G_ϕ.

(e) Proof of correctness.

If ϕ is satisfiable, then there is a truth assignment τ such that $\phi(\tau) = 1$. We describe a path from s to t that traverses every vertex in G_ϕ.

- We traverse a variable gadget starting from its top vertex to its bottom vertex in the following manner.
  - If $\tau(x_i) = 1$ we traverse the horizontal row of the gadget corresponding to x_i from left to right.
  - If $\tau(x_i) = 0$ we traverse the horizontal row of the gadget corresponding to x_i from right to left.

  This path covers all the vertices corresponding to the n variable gadgets.
- Since τ is a satisfying assignment, every clause in ϕ has at least one true literal with respect to τ. Let l_j be the true literal corresponding to a clause c_j. When traversing the variable gadget corresponding to the vertex of l_j, we do a detour and cover the vertex corresponding to c_j using the two clause edges. Note that if $l_j = x_i$ for some variable x_i then we would be traversing the horizontal row of x_i from left to right and if $l_j = \neg x_i$ for some variable x_i then we would be traversing the horizontal row of x_i from right to left. Hence by taking the detour we do not disturb the property that every vertex in traversed exactly once.

This gives a Hamiltonian path from s to t in G_ϕ.

On the other hand, suppose there is a Hamiltonian path from s to t in the graph G_ϕ.

- We first argue that this path cannot jump from one variable gadget to another variable gadget using a clause vertex. Suppose the path jumps out of a vertex u in a variable gadget to a clause vertex and then jumps in to a vertex in a different variable gadget. Then the buffer vertex after u can never be covered by the path as there is no way to reach that vertex. Hence the Hamiltonian path traverses every variable gadget completely from left to right or from right to left.
- We next define a truth assignment for ϕ. If the variable gadget corresponding to the variable x_i is traversed from left to right then set $x_i = 1$ and if it is traversed from right to left then set $x_i = 0$. It is easy to verify that this is indeed a satisfying assignment.

5. $\mathsf{UHamPath}$ is NP-complete.

   - Let $G = (V, E)$ be an undirected graph. A *Hamiltonian path* in G is a path that visits every vertex exactly once.
   - $\mathsf{UHamPath} = \{\langle G, s, t\rangle \mid G$ is undirected and has a Hamiltonian path from s to $t\}$.

Exercise 8. Show that $\mathsf{UHamPath}$ is in NP-complete.

(Hint: Reduce $\mathsf{HamPath}$ to $\mathsf{UHamPath}$. Don't see the proof from your textbook before giving it a try.)