Show that an instance (input) to HC can be converted to an instance of TSP:

\[
\text{instance (HC) } \rightarrow \text{ Instance converter } \rightarrow \text{ instance (TSP) } \rightarrow \text{ ALG(TSP) } \rightarrow \text{ sol(TSP) } \rightarrow \text{ solution converted (Yes/No) } \rightarrow \text{ sol(HC) (Yes/No)}
\]

* All we have to do is find these →

Instance Converter first.

Consider the differences b/w an instance of HC and an instance of TSP:

- **Instance of HC**: general (not necessarily complete) unweighted graph
- **TSP**:
  - (a) complete graph
  - (b) weights on edges
  - (c) bound k

Given an instance \( (G = (V, E)) \) for HC need to:

convert it to an instance \( (G' = (V', E'), k) \) for TSP s.t. a solution for TSP can be easily converted to a solution for HC.

A tour of cost \( \leq k \) in \( G' \) indicates a HC in \( G \) (in polytime).
(1) $V' = V$

(2) $E'$ has edges to make $G'$ complete

How to set weights on edges in $E'$ so that a TSP tour of cost $\leq k$ is a HC in $G$?

(3) For each edge $(u, v)$ in $E'$
   - if $(u, v) \in E$ set $c(u, v) = 1$
   - else $c(u, v) = \infty$

* Which of these edges will indicate a HC in $G'$? Those w/cost = "How many?" $|V|$

* Cycle that visits every internal node exactly once has exactly $|V| - 1$ edges

(Recall: acyclic path has length at most $|V| - 1$)

Set $k = |V|.$

* Updates above convert the instance.

Solution converter: $\text{sol}(HC) = \text{sol}(TSP) (= \text{Yes or No})$

ex: $G'$ has TSP tour $b-a-c-d-e-b$ with cost $\leq k = |V| = 5.$

$G$ is not HC: $b-a-c-d-e-b$
Claim: HC \equiv_p TSP

⇒ There is a TSP tour of cost ≤ k in G' iff there is a HC in G:
   
   HC \implies \text{tour}
   
   \text{tour} \implies HC

\[ \text{(1)} \]

A cycle that visits every internal node exactly once has exactly 1V1 edges.

⇐ If there is a TSP tour of cost ≤ k in G', then there is a HC in G.

Since the cost of the tour in G' ≤ 1V1, each edge of the tour must have cost = 1, so must correspond to an edge from G. These 1V1 edges will make a HC in G

⇒ If there is a HC in G, then there is a TSP tour of cost ≤ k - 1V1 in G'.

The 1V1 edges that make the HC in G each have cost 1 in G', so these edges will make a TSP tour of cost = 1V1 in G'.

Note: There are other correct reductions:

- Z instead of \( \infty \)
- Set cost of G edges to 0 instead of 1, and

\( \alpha + \nu = 0 \) (Choice: 1) \( \xi + r = (1 + \nu)^{-1} \)
Some questions:

1) Why not show TSP reducible to HC?

But if we could solve HC we could solve TSP.

Does not imply: Since we can't solve HC, can't solve TSP.

Logic:

- solve HC \rightarrow solve TSP.

Does not imply: 7 solve HC \rightarrow 7 solve TSP.

Showing TSP reducible to HC:

- instance(TSP) \rightarrow \[ \square \] \rightarrow instance (HC) \rightarrow [ALG(HC) \rightarrow sol(HC) \rightarrow sol(TSP)]

We know this doesn't exist but possible to solve TSP another way.

Still may be possible:

- instance (TSP) \rightarrow [\square] \rightarrow instance (Some Other Problem) \rightarrow [ALG]...

2) Why have to prove it? and not just sol(TSP) = Yes \rightarrow sol(HC) = Yes?

Want to show:

\[ sol(TSP) = Yes \rightarrow sol(HC) = Yes \quad \text{AND} \quad sol(TSP) = No \rightarrow sol(HC) = No \equiv 7 sol(TSP) \rightarrow 7 sol(HC) \equiv sol(HC) \rightarrow sol(TSP). \]
Another NPC Proof

Problem known to be NPC: 3-SAT

X: set of n boolean variables x₁, x₂, ... xₙ
⇒ each can be either 0 (false) or 1 (true)

literal: "2 versions of a variable: xᵢ or \( \overline{xᵢ} \)

clause - disjunction (OR's) of distinct literals

ex: \( x₁ u x₄ u \overline{x₅} \)

Satisfying assignment - assignment of values (0 or 1) for every literal s.t. the expression evaluates to true.

"easy for disjunction - just assign one literal to 1"

"harder when we have to satisfy multiple disjunctions."

hard: conjunction - AND's of clauses (OR's)

ex: \( (x₁ u \overline{x₂}) \land (\overline{x₂ u x₃ u x₁}) \land (\overline{x₁ u x₂}) \)

"can't just assign one variable in each clause to 1."
For $x_1 = T$, $x_2 = T$ bloc then 3rd clause: $F \lor \overline{F} = F$.

**3-SAT:** Given a conjunction $B$, of $k$ clauses $c_1, c_2, \ldots, c_k$ where each clause contains 3 literals over a set $X$ of $n$ boolean variables $X = \{x_1, x_2, \ldots, x_n\}$, determine if there exists a satisfying assignment. (Yes/No?)

Fundamentally hard: $n$ independent decisions, each has 2 options (0/1) $\Rightarrow 2^n$.

We will reduce 3-SAT to new problem: Independent Set.

**Independent Set:** in a graph $G = (V, E)$, a subset of vertices $S$, s.t. no two vertices are adjacent.

**ex**

```
\begin{align*}
\text{Ind. Sets:} \\
&= \{1, 3\}, \{1, 4, 5\}, \{1, 6, 5, 3\} \\
\text{largest: } &\{1, 4, 5, 6, 3\}
\end{align*}
```

Application: open restaurants such that no 2 locations are adjacent.
Ind-Set Problem: given a graph $G = (V, E)$ and an integer $c$, is there an ind-set of size $\geq c$?

Thm: J-S is NPC.

1) $IS \in NP$.
   certificate is set $S$ of vertices. Check:
   * $|S| \geq c$
   * For every pair $i, j \in S$, $(i, j)$ and $(j, i) \notin E$.

   can verify in poly-time!

2) Show $3$-SAT $\leq_p$ IS.
   If we can solve IS, we can solve $3$-SAT.
   Since we know we can't solve $3$-SAT, we can deduce that we can't solve IS.

   Show that a solver for IS can solve $3$-SAT:

   3-SAT instance $\rightarrow$ Instance converter $\rightarrow$ IS instance $\rightarrow$ Alg(IS) $\rightarrow$ Sol(IS) $\rightarrow$ Solution converter $\rightarrow$ Sol(3-SAT)

Instance converter:

Instance of 3-SAT:

* Conjunction $B = C_1 \land C_2 \land \cdots \land C_k$ over
  variables $X_1, X_2, \ldots, X_n$
Instance of IS?
  * Graph \( G = (V, E) \).
  * Constant \( c \).

Convert instance \( <B> \) of 3-SAT to instance \( <G=(V,E), c> \) of IS s.t. an ind-set of size \( \geq c \) in \( G \) indicates there is a sat. assignment in \( E \).

\[ E = (x_1 \lor x_3 \lor \overline{x}_6) \lor (\overline{x}_5 \lor \overline{x}_4 \lor x_2) \lor (\overline{x}_1 \lor \overline{x}_2 \lor x_6) \lor \ldots \]

*Note:* In general, \( B \) can have \( \geq 3 \) clauses!

**Hint:** How could we find a sat. assignment for \( E \)?

*For each \( \lor \) clause, choose a literal, set it to true if it has not already been set to false.*

\[ E: \]
- \( C_1: \text{set } x_1 = 1 \)
- \( C_2: \text{set } \overline{x}_5 = 1 \)
- \( C_3: \text{Can't set } \overline{x}_1 \text{ to } 1, \text{ so set } \overline{x}_2 = 1 \).

3-SAT: For a variable \( x_i \), either \( x_i \) or its negation \( \overline{x}_i = 1 \), not both.

*How is this similar to ind-set?*

IS: For each edge \((i,j)\) either \( i \) or \( j \in S \), not both!
For every variable $x_i$, create an edge $\overline{x_i} - x_i$.

Instance Converter

1. Given $B = C_1 \land C_2 \land \ldots \land C_k$ for 3-SAT, construct $G = (V,E)$ for JS as follows:
   - for every variable $x_i$ in $B$, $G$ has nodes $x_i, \overline{x_i}$
   - and an edge $x_i - \overline{x_i}$

2. More...

This ensures both $x_i, \overline{x_i}$ not set to true

How to ensure that at least one literal from each clause is set to true?

$\Rightarrow$ Connect the 3 literals of each clause so at least one can be set to true if it is used in the ind. set.