# NP-Completeness

# Background Knowledge

- To understand NP-Completeness, need to know these concepts

1. Decision and Optimization Problems
2. Turing Machine and class P
3. Nondeterminism and class NP
4. Polynomial Time Reduction (Problem Transformation)

# Decision and Optimization Problems

- What is the Shortest Path from A to B?
  - This is an Optimization Problem.
- Is there a Path from A to B consisting of at most K edges?
  - This is the related Decision Problem.

We consider only Decision Problems!

# Turing Machine and Class P

- **P :** The class of problems that are decidable in polynomial time on a Turing machine.
- Sorting, Shortest Path are in P!

# Nondeterminism and Class NP

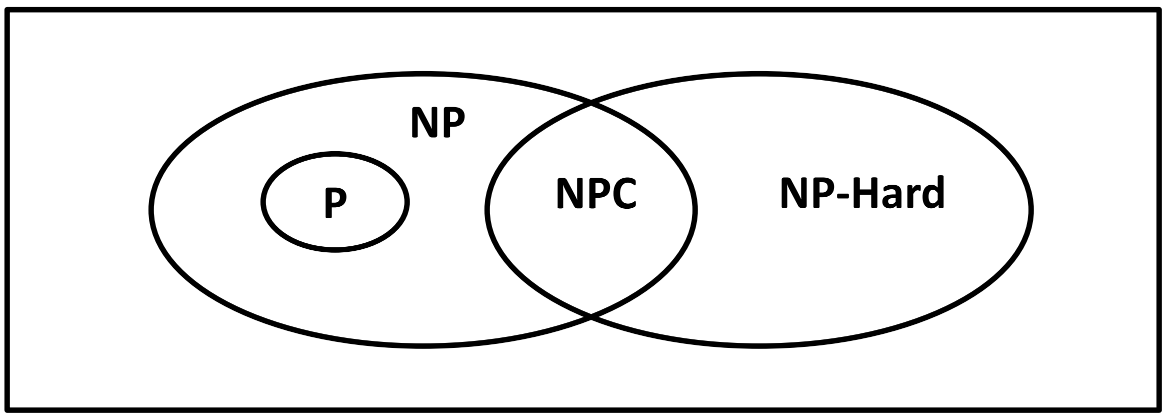
- **NP :** The class of problems that are decidable in polynomial time on a nondeterministic Turing machine
- Solutions of problems in NP can be checked (verified) in polynomial time.
- Determining whether a directed graph has a Hamiltonian cycle does not have a polynomial time algorithm (yet!)
- However if someone was to give you a sequence of vertices, determining whether or not that sequence forms a Hamiltonian cycle can be done in polynomial time
- Therefore Hamiltonian cycles are in NP

# Class P and NP

- P = the class of problems where membership can be **decided** quickly.
- NP = the class of problems where membership can be **verified** quickly.

# NP-Complete and NP-Hard

**NP-Hardness:** Some problems are at least as hard to solve as any problem in NP. We call them NP-Hard.

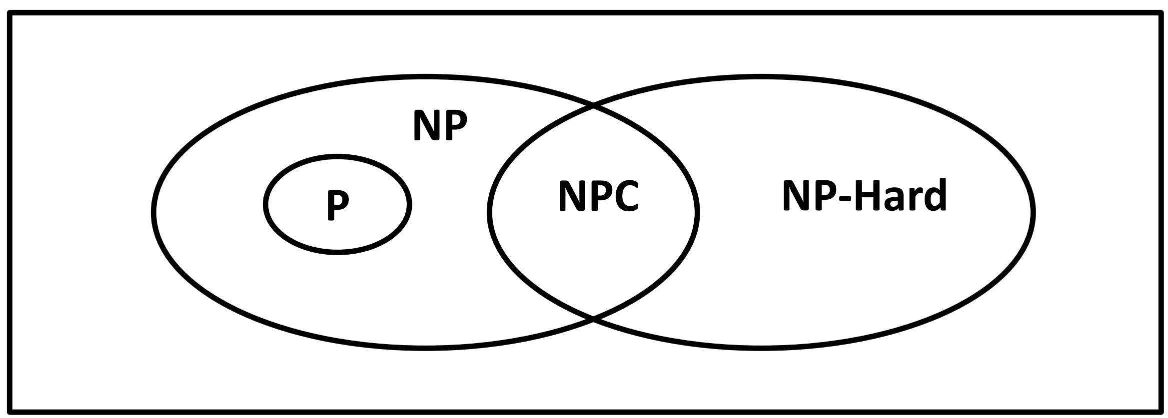



- If an NP-hard problem can be solved in polynomial time, then all NP-complete problems can be solved in polynomial time.
- All NP-complete problems are NP-hard, but all NP- hard problems are not NP-complete.

# NP-Complete and NP-Hard

**NP-Completeness:** A problem X is NP-complete if it satisfies two conditions:

1. X is in NP, and
2. Every problem A in NP is polynomial time reducible to X.

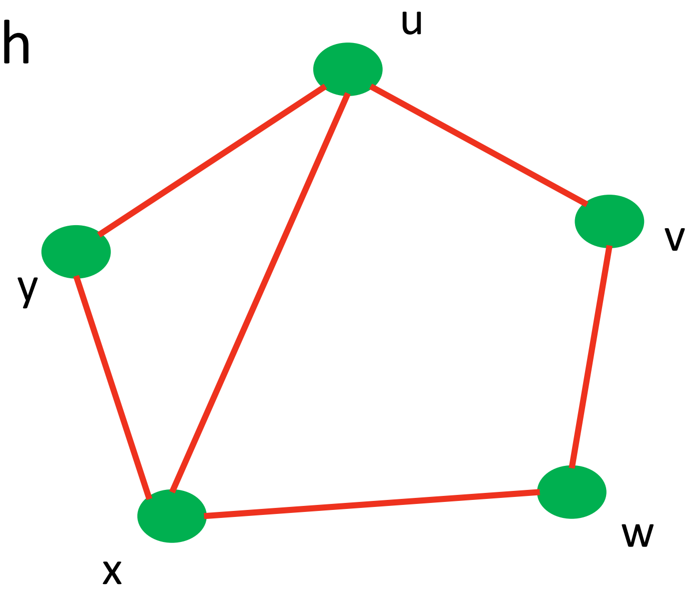


# Polynomial Time Reduction

- A polynomial-time reduction proves that the first problem is no more difficult than the second one.
- Because whenever an efficient algorithm exists for the second problem, one exists for the first problem as well.
- For example: if problem A is polynomial time reducible to problem B, then it means when an algorithm exists for problem B, problem A can also have another.
- It is denoted as $A \leq_p B$

# Independent Set

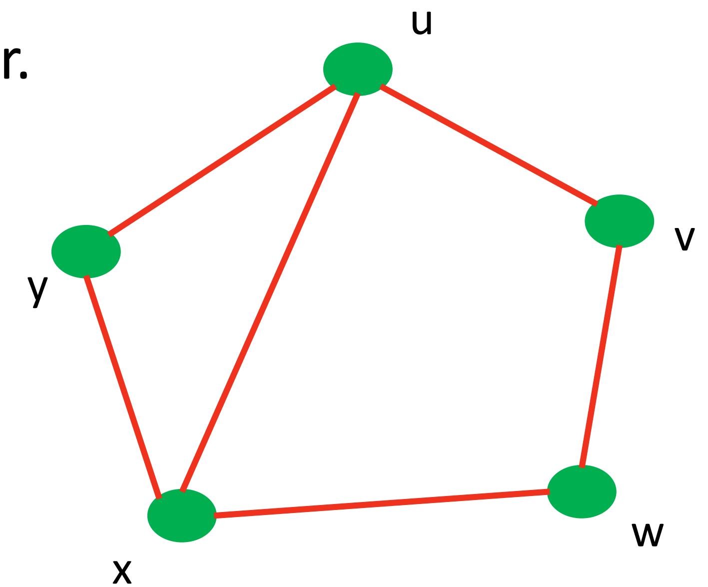
- It is a set of vertices in a graph, no two of which are adjacent.
- That is, it is a set S of vertices such that for every two vertices in S, there is no edge connecting the two.
- Equivalently, each edge in the graph has **at most** one endpoint in S.
- For example: in the following graph {u, w} is an independent set.
- The set of each vertex is an independent set.



# Vertex Cover

- It is a set of vertices such that each edge of the graph is incident to **at least** one vertex of the set.
- Such a set is said to **cover** the edges of the graph.
- For example: in the following graph

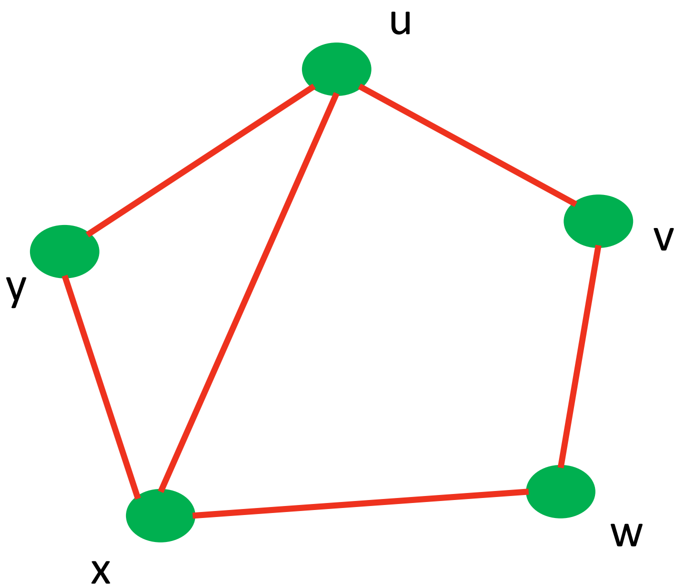
  {u, w, x} is the vertex cover.
- The set of all vertices is a vertex cover.



# Independent Set $\leq_p$ Vertex Cover

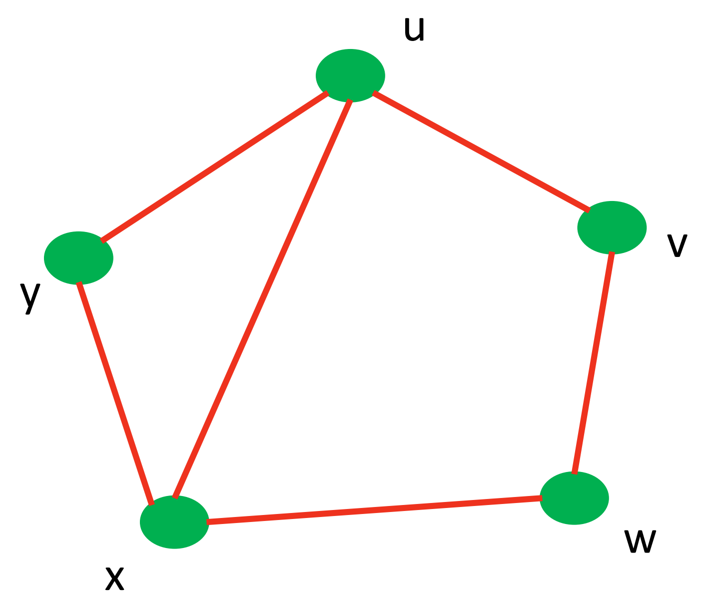
- **Decision for independent set:** Does graph G have an independent set of size of **at least** k?
- **Decision for vertex cover:** Does graph G have a vertex cover of size of **at most** V-k?

**S** is an independent set
if there are no edges within **S**.
≡
**S** is an independent set
if every edge is not in **S**.



# Independent Set $\leq_p$ Vertex Cover

**S** is an independent set
if there are no edges within **S**.
≡
**S** is an independent set
if every edge is ~~not in **S**~~ incident in **V-S**.




- **Observation:** **S** is an independent set
  iff **V-S** is a vertex cover.
- **Corollary:** G has an independent set ≥ k
  **iff** G has a vertex cover ≤ V-k.