Tutorial - 03 - Answers

01. Professor has only shown that **the specific problem L can be solved in polynomial time**, but this does not mean that **all NP problems can be solved in polynomial time**.

To prove P=NP, it would require showing that **every problem in NP**, including all NP-complete problems, has a polynomial-time solution. The result for L does not extend to other problems in NP, so it does not imply P=NP.

02.

Belonging to NP:

- **NP-Hard**: May or may not belong to NP.
- **NP-Complete**: Always belongs to NP.

Solution Verification:

- **NP-Hard**: Solutions may not be verifiable in polynomial time.
- **NP-Complete**: Solutions are verifiable in polynomial time.

Problem Type:

- **NP-Hard**: Can be either decision, optimization, or other types of problems.
- **NP-Complete**: Must be a decision problem (yes/no answer).

Reduction:

- **NP-Hard**: Every NP problem can be reduced to an NP-hard problem in polynomial time, but the NP-hard problem itself may not reduce to other NP problems.
- **NP-Complete**: Every NP problem can be reduced to an NP-complete problem in polynomial time, and vice versa.

Subset Relationship:

- **NP-Complete** problems are a subset of **NP-Hard** problems. All NP-complete problems are NP-hard, but not all NP-hard problems are NP-complete.

Examples:

- **NP-Hard**: Halting Problem, TSP (optimization version).
- **NP-Complete**: 3-SAT, CLIQUE, Vertex Cover, TSP (decision version).

03.

By considering common assignments across clauses, one possible satisfying assignment is:

- w=True
- x=True
- y=True
- z=False

Verification:

- **Clause 1**: (w ∨ x ∨ ¬z): True ∨ True ∨ True=True
- **Clause 2**: (x ∨ ¬y ∨ z): True ∨ False ∨ False=True
- **Clause 3**: (w ∨ ¬x ∨ z): True ∨ False ∨ False=True
- **Clause 4**: (¬w ∨ ¬x ∨ y): False ∨ False ∨ True=True

So, this assignment satisfies the formula and there can be other satisfying assignment

---

04.

I. To say that the Boolean Satisfiability problem (SAT) is **polynomially reducible** to CLIQUE means that for any given SAT formula F, we can construct a graph G_F in polynomial time such that:

F is satisfiable **if and only if** the graph G_F contains a clique of size k, where k is the number of clauses in F.

In other words, solving the CLIQUE problem on G_F provides a solution to the SAT problem for F, showing that CLIQUE is at least as hard as SAT.

II.

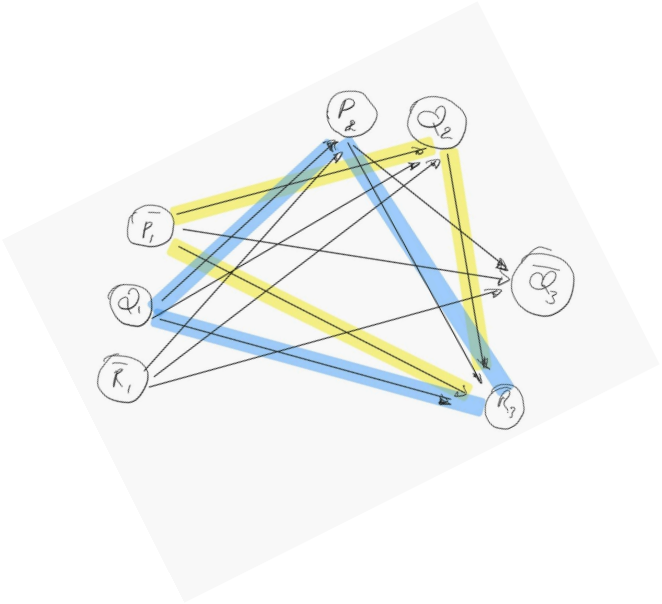
F = (¬P ∨ Q ∨ ¬R) ∧ (P ∨ Q) ∧ (¬Q ∨ R)

This formula consists of **three clauses**, so, construct a graph G_F where finding a clique of size k=3 corresponds to satisfying all three clauses.

For each clause in F, represent each literal as a vertex. Connect vertices across different clauses if and only if the corresponding literals are **non-conflicting** (i.e., they can both be true at the same time).

- **Clause 1:** (¬P ∨ Q ∨ ¬R)→ Vertices: $\neg P_1, Q_1, \neg R_1$
- **Clause 2:** (P ∨ Q) → Vertices: P_2, Q_2
- **Clause 3:** (¬Q ∨ R) → Vertices: $\neg Q_3, R_3$

Add edges between vertices of **different clauses** where the literals do not conflict.



05.

Knapsack Problem Explanation:

The **Knapsack problem** involves selecting a subset of items, each with a given weight and value, to maximize the total value without exceeding a weight capacity W.

In this case:

- Values: [10,20,30,40]
- Weights: [30,10,40,20]
- Capacity W=40

This **0/1 Knapsack Problem** is NP-complete because:

- There is no known polynomial-time algorithm to solve it.
- However, it can be solved using **Dynamic Programming** in pseudo-polynomial time.

Use the **Dynamic Programming** approach to find the maximum value.

Steps:

- ➢ Define a table dp[i][w], where:
  - i = number of items considered.
  - W = current weight capacity.
- ➢ Recurrence relation:
  dp[i][w]=max(dp[i−1][w], dp[i−1][w−$weight_i$]+$Value_i$)
  - If the item is not included: dp[i−1][w].
  - If the item is included: dp[i−1][w−$weight_i$]+$Value_i$.

Dynamic Programming Table Construction

Capacity W	Item 1 ($v = 10, w = 30$)	Item 2 ($v = 20, w = 10$)	Item 3 ($v = 30, w = 40$)	Item 4 ($v = 40, w = 20$)
$0 - 9$	0	0	0	0
$10 - 19$	0	20	20	20
$20 - 29$	0	20	20	40
$30 - 39$	10	30	30	50
40	10	30	30	60

Maximum Value:

The maximum value for the given capacity W=40 is **60**, achieved by taking:

- Item 2 (v=20, w=10,) and
- Item 4 (v=40, w=20,).

Illustration:

- Start with an empty knapsack.
- Add Item 2 (weight = 10, value = 20). Remaining capacity = 40−10=30.
- Add Item 4 (weight = 20, value = 40). Remaining capacity = 30−20=10.
- The total value = 20+40=60.

06.

Part A: Does there exist a vertex cover of G at most k nodes?

1. **Definition of Vertex Cover**:
   A vertex cover is a set of vertices such that every edge in the graph has at least one endpoint in this set. Removing the vertex cover from the graph ensures that no edges remain.
2. **Given Vertex Cover**:
   In the graph, the set {A,C,D,F} is mentioned as a vertex cover. This vertex cover contains 4 nodes.
3. **Checking for Smaller Vertex Covers**:
   - Let k=3. We need to check if a vertex cover of size at most 3 exists.
   - Consider any subset of 3 vertices (e.g., {A,C,F}):
     - If these are removed, edges like (B,C) and (E,F) are still covered.
     - However, edge (D,E) will not be covered by this set.
4. Therefore, **a vertex cover of size 3 does not exist** for this graph. The minimum vertex cover is at least 4 nodes.

Part B: Showing Vertex Cover Problem is NP-Complete Using 3SAT Reduction

To prove that the Vertex Cover problem is NP-complete, we use a reduction from the 3SAT problem.

1. **3SAT Problem Structure**:
   - The 3SAT problem has variables and clauses.
   - Each clause contains three literals, and the goal is to find a truth assignment that satisfies all clauses.
2. **Graph Construction** (Mapping 3SAT to Vertex Cover):
   - **For Each Variable**: Create **two connected nodes** to represent the variable (x) and its negation (¬x).
   - **For Each Clause**:Create **three connected nodes** to represent the literals of the clause.

By constructing such a graph, solving the Vertex Cover problem on this graph (finding a vertex cover of size k) will directly correspond to finding a satisfying assignment for the 3SAT problem. Since 3SAT is NP-complete and reducible in polynomial time, this proves that the Vertex Cover problem is also NP-complete.