ASSIGNMENT - 3 & 4

Q1) Explain polynomial time reducibility with example.

A1) Polynomial Time Reducibility is a core concept in computational complexity theory.

Polynomial Time Reducibility, denoted as $A \leq_p B$, is a method for comparing the relative difficulty of two computational problems, A and B.

A problem A is polynomial time reducible to a problem B if there exists an algorithm, called a reduction, that solves problem A.

Key Implication →

The primary implication is that problem B is at least as hard as problem A.

- If we have an efficient algo. for B, we can solve A efficiently by first reducing A to B and then using the efficient algo. for B.
- Conversely, if A is a hard problem, then B must also be a hard problem.

Example → Finding a path $\leq_p$ Finding a cycle

The PATH problem is polynomial time reducible to the CYCLE problem

The Problems →

1) PATH Problem (Problem A)
2) CYCLE Problem (Problem B)

The Reduction $A \rightarrow B$

Path exists from s to t in $G \leftrightarrow G'$ contains a cycle.

Q2) Define and Differentiate:
NP Packet Graph
NP Scheduling Problem

A2) NP Packet Graph →

An NP Packet Graph problem is a computational task that involves graph theory and typically models the movement, routing or colouring of packets across a network.

These problems focus on finding an optimal configuration in a graph structure, and the NP-Complete difficulty often arises.

Examples →

- Hamiltonian Cycle
- Clique Problem
- Graph Coloring

NP Scheduling Problem →

An NP Scheduling Problem is a computational task that involves optimally assigning a set of jobs (tasks) to a set of resources (processors/machines) over a period of time.

The NP-hard difficulty arises from trying to optimize one or more metrics when factors like ~~pred~~ precedence constraints, job deadlines and machine availability are introduced.

Examples →

- Job-Shop Scheduling
- Multiprocessor Scheduling
- Resource Constrained Project Scheduling

Feature	NP Packet Graph	NP Scheduling Problem
Domain	Topology (Nodes, Edges, Connectivity).	Time and Resources (Jobs, Machines, Duration).
Goal	Optimal configuration on a static structure.	Optimal assignment over time on a dynamic processor.
Typical Reduction	From: 3-SAT, Vertex Cover, Hamiltonian cycle.	From: Partition, 3-Partition.

Q3) What is a non-deterministic algorithm? Give one example.

A3) A non-deterministic algorithm is a theoretical construct or an algo. in which the same input can lead to different behaviours or different outputs on different runs, unlike a deterministic algo. where the next step is always uniquely determined.

Two Primary Ways →

1) Theoretical Non-Deterministic (Computational Complexity) → This is the context used in defining the complexity class NP. In this view, the algo. can make a "lucky guess" or split into multiple paths simultaneously.

2) Practical Non-Deterministic (Real-World Computing) → This refers to algorithms whose behaviour varies due to factors outside the input, such as:

- Randomness
- Concurrency / Parallelism
- External Input

Example → Searching for a path in a graph

The most common way to illustrate a non-deterministic algo. is in the theoretical context of complexity, specially for problems that are hard to solve but easy to verify, like the Hamiltonian cycle Problem.

The Problem → Hamiltonian Cycle (Decision Problem).

Q4) What is a state space tree? State two ways to search for an answer node.

A4) A state space tree is a fundamental concept in algo. design, particularly for search and optimization problems.
It's a tree structure where:

- Each node represents a partial solution or a specific state of problem
- The root node represents the initial state.
- Branches (edges) represent the choices or moves that can be made from one state to the next.
- An answer node (or goal node) is a node that represents a valid and complete solution to the problem.

Two ways to search for an answer node →

1) Depth First Search (DFS) →

DFS explores the tree by going as deep as possible along a single path before backtracking and trying the next path. From the current node, it always expands the deepest unexpanded node first. It uses a stack.

Advantages → It requires minimal storage space because it only needs to store the current path from the root. It is excellent for finding solutions that are deep in the tree.

Disadvantages → It is not guaranteed to find the shortest path to an answer node and in an infinite or very deep tree.

2) Breadth First Search (BFS) →

BFS explores the tree layer by layer, expanding all nodes at a given depth before moving to the next level.
It expands all nodes at depth d before expanding any node at depth d+1. It uses a Queue (First-In, First-Out or FIFO).

Advantages → It is guaranteed to find the shortest path (in terms of number of steps/edges) to the goal node.

Disadvantages → It requires significant memory storage because it must hold all generated nodes for the current level in the queue before processing the next level.

Q5) Explain Boolean satisfiability (SAT) problem with one example.

A5) The Boolean Satisfiability (SAT) is a fundamental decision problem in computer science and mathematical logic.
It asks the question: Does there exist an assignment of truth values (TRUE or FALSE) to the variables of a given Boolean formula such that the entire formula evaluates to TRUE?

- A boolean formula is an expression involving Boolean variables, logical operators and parenthesis.
- If such an assignment exists, the formula is said to be satisfiable; otherwise, it is unsatisfiable.

Example → The SAT problem is often studied with formulas expressed in Conjuctive Normal Form (CNF), which is a conjuction (AND) of one or more clauses.

~~F = (A V ¬B~~

$$F = (A \vee \neg B \vee C) \wedge (\neg A \vee B) \wedge (\neg C)$$

Clause	Formula
Clause 1	$(A \vee \neg B \vee C)$
Clause 2	$(\neg A \vee B)$
Clause 3	$(\neg C)$

Q6) Define Cook's Theorem in one line and state one application.

A6) Cook's Theorem states that the Boolean Satisfiability Problem (SAT) is NP-complete. This monumental theorem was the first to prove a problem's membership in this class, establishing SAT as the "master problem" for all problems in NP.

Application → Establishing the foundation for all NP-completeness proofs because the theorem provides a way to reduce any NP problem to SAT in polynomial time, it allows computer scientists to prove that a new problem is also NP-complete by simply showing a polynomial-time reduction from SAT.

A practical application that relies on this reduction power is Automated Verification (Formal Verification) in hardware design.

Q7) Differentiate:
P, NP, NP-Complete, NP-Hard.
Write two examples for each class (short titles only).

A7)

Class	Definition	Role	Examples
P	Problems that can be solved by a deterministic algorithm.	The class of "easy" problems.	Primality Testing, Linear Programming.
NP	Problems where a "yes" answer can be verified by a deterministic algorithm.	The class of problems solvable in polynomial time by a non-deterministic machine.	Boolean satisfiability (SAT), Travelling Salesperson
NP-Complete (NPC)	Problems in NP such that every other problem in NP can be reduced to it in polynomial time.	The "hardest" problems in NP. If one is solved in polynomial time, all problems in NP are in P (P=NP).	Boolean Satisfiability (SAT), 3-SAT.

NP-Hard (NPH)	Problems such that every problem in NP can be reduced to it in polynomial time.	At least as hard as the hardest problems in NP. Includes all NP-complete problems, but can also include problems harder than NP (like optimization problems).	Travelling Salesperson (Optimization), Halting Problem.

Q8) Explain the Hamiltonian Cycle problem in context of NP.

A8) The Hamiltonian Cycle Problem (HCP) is a decision problem that asks: Given a graph G, does it contain a cycle that visits every vertex exactly once?

1) HCP is in NP →

The HCP belongs to the class NP because if a "yes" answer is correct, you can verify it in polynomial time.

- The Certificate (Proof)
- The Verifier (Checking)
  1) Check if the sequence contains all n vertices.
  2) Check if the sequence contains no repeated vertices.

2) HCP is NP-complete (NPC) →

While HCP is easy to verify, finding Hamiltonian Cycle is incredibly difficult. HCP is NP-complete because it satisfies two conditions:

1) HCP ∈ NP
2) HCP ∈ NP-Hard

The proof that HCP is NP-Complete often involves a polynomial-time reduction from a known NP-complete problem, such as the 3-satisfiability Problem (3-SAT) or the Travelling Salesperson Problem.