Lesson 2: P, NP, NP-Hard and NP-Complete Problems

Table of Contents:

1. Introduction
2. P (Polynomial Time)
3. NP (Non-deterministic Polynomial Time)
4. Reduction
5. NP-Hard Problems
6. NP-Complete Problems
7. References

Introduction

- In the world of computer science, we encounter puzzling problems that haven't been solved yet. To make sense of these challenges, we group them into different classes, which helps us understand how hard they are to solve. This area of study is called complexity theory. It helps us figure out how much time and memory we need to tackle these problems.

- Two important things we look at are time (how long an algorithm takes to solve a problem) and space (how much computer memory it uses).

- For example, let's say you want to solve a puzzle. The time complexity is like counting how many steps you take to solve it, and the space complexity is like figuring out how much space you need on the table to work on the puzzle.

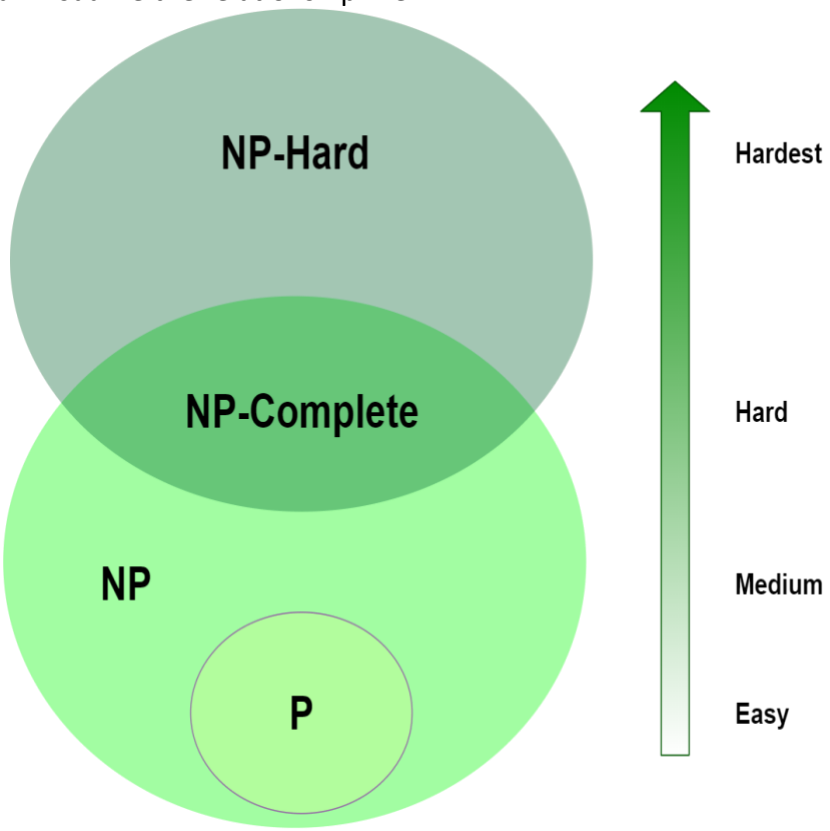
- Complexity classes help us put similar problems in the same group. There are different classes and four of them are particularly important:

    - P Class: This group has problems that can be solved relatively quickly using a computer.
    - NP Class: Here, problems can be checked quickly, but finding the solution might not be as fast.
    - NP-hard: These problems are as tough as the hardest problems in the NP class.
    - NP-complete: This is a special kind of tough problem that combines the difficulties of both NP problems and NP-hard problems.

- By using these complexity classes, we can better understand how hard problems are and what we can expect when trying to solve them with computers.

Lesson 2: P, NP, NP-Hard and NP-Complete Problems

- We can visualize the relationship like:



P (Polynomial Time):

In the context of computational complexity theory, "P" refers to the complexity class of decision problems that can be solved by a deterministic Turing machine in polynomial time. In simpler terms, P consists of problems for which there exists an algorithm that can find a solution in a reasonable amount of time as the problem size grows, specifically in time proportional to a polynomial function of the input size. These problems are considered efficiently solvable within polynomial time bounds and are a fundamental class in understanding algorithmic efficiency.

we can formally define its time complexity as:

$$T(n)= O(C*n^k) \text{ where } C>0 \text{ and } K>0$$

Examples of P:
Linear and Binary Search Algorithms, string operations and Hashtable Lookup

Lesson 2: P, NP, NP-Hard and NP-Complete Problems

NP (Non-deterministic Polynomial Time):

NP comprises decision problems solvable by a Non-deterministic Turing Machine within Polynomial-time. Within NP, there exists a subset known as P, where any problem solvable by a deterministic machine in polynomial time is also solvable by a non-deterministic machine in polynomial time i.e.

1) If you are providing the same input but receiving different outputs, it indicates a non-deterministic algorithm.
2) A non-deterministic algorithm takes multiple paths, and we cannot definitively determine the next step of execution.
3) In this scenario, we will only obtain approximate solutions, not precise ones.

we can formally define its time complexity as:

$$T(n)= O(C1*k^{(c2*n)}) \text{ where } C1>0 ,C2>0 \text{ and } K>0$$

where and are constants and is the input size. is a function of exponential-time when at least and

Let us take a real-time scenario of an NP:

Imagine you're taking a personality quiz online. The quiz wants to understand not just your answers but also your mood and circumstances at that moment. Here's

How the algorithm works:

Process:

The algorithm gets your answers.
It randomly picks things like your current mood, time of day, and recent experiences – things that could affect your responses.
It combines your answers with these random factors to create different possible situations reflecting your state of mind while taking the quiz.
For each situation, it predicts the personality traits you might show, based on rules and calculations.
This creates various personality profiles (Y1, Y2, Y3, ...), each showing a different you based on those situations.

Output:

The algorithm gives you possible personality profiles (Y1, Y2, Y3, ...). Each profile suggests how your traits could be influenced by your mood and context.

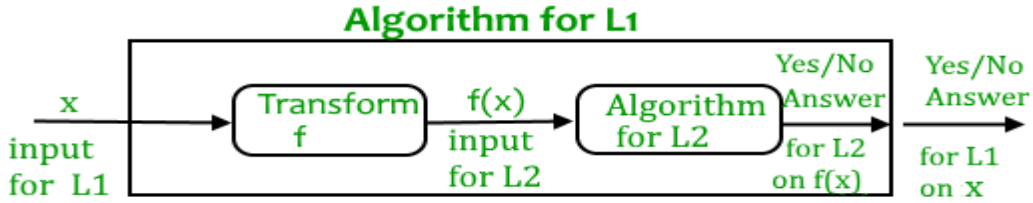
Lesson 2: P, NP, NP-Hard and NP-Complete Problems

Reduction:

Reduction is a powerful technique in computer science used to analyze the complexity of two decision problems, denoted as p1 and p2. If we have an unknown algorithm A1 and input L1 to solve p1, and we can already solve p2 using a known algorithm A2 and input L2, then we can say p1 is reducible to p2 if we can transform L1 into L2 in a way that if p2 is solved with I2, we can also solve p1 with I1. This implies that p1 is at least as easy to solve as p2, making reduction a key tool for comparing and understanding the relative difficulty of problems in computational theory.

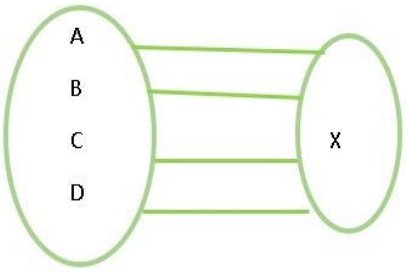
Let us Consider,

Algorithm for L1=A1, L2= A2



NP-Hard Problems:

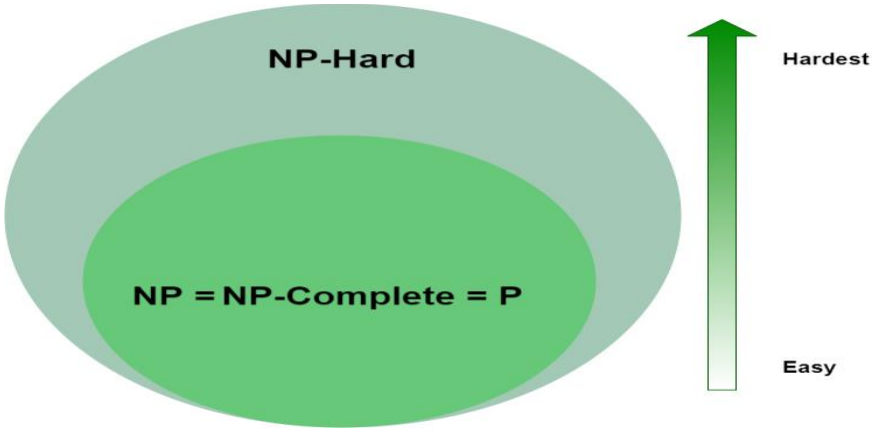
A problem is labelled NP-hard when it's feasible to polynomials convert any problem within the NP complexity class to that specific problem. For instance, if we have a set of NP problems - let's call them A, B, C, and D - and we can reduce all of these problems into another problem x, then we can classify A, B, C, and D as NP-hard problems.



NP-Complete Problems:

An NP-complete problem meets two conditions. Firstly, it falls within the NP complexity class, implying that its potential solutions can be swiftly verified. Secondly, it satisfies the NP-hard condition, implying that every problem in NP can be transformed into this particular NP-complete problem through polynomial reduction. Essentially, NP-complete problems embody the toughest challenges within NP, and they share the characteristic that solving one of them in polynomial time would enable solving all NP problems efficiently.

Lesson 2: P, NP, NP-Hard and NP-Complete Problems



References:

1. https://www.geeksforgeeks.org/introduction-to-np-completeness/

2. https://www.youtube.com/watch?v=valpD5Jx7aw&list=RDCMUCHNO_Y3DskuKiw9VTvo8AMw&start_radio=1&rv=valpD5Jx7aw&t=14

3. https://www.youtube.com/watch?v=e2cF8a5aAhE&t=472s

4. https://www.baeldung.com/cs/p-np-np-complete-np-hard