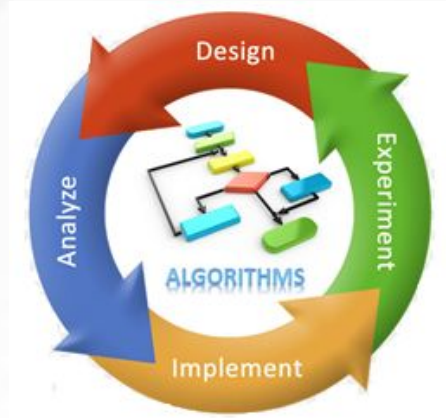
Analysis and Design of Algorithms
(ADA)
GTU # 3150703

Dedicated Faculty, Committed Education
Darshan
Institute of Engineering & Technology

Unit-9:
Introduction to
NP-Completeness



Dr. Gopi Sanghani
Computer Engineering Department
Darshan Institute of Engineering & Technology,
Rajkot
gopi.sanghani@darshan.ac.in
9825621471

Outline

- The class P and NP
- Polynomial reduction
- NP- Completeness Problem
- NP-Hard Problems
  - ✔ Travelling Salesman problem
  - ✔ Hamiltonian problem
- Approximation algorithms
- Randomized algorithms

The class P and NP

Time Complexity of an Algorithm

- Time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the input.
- Asymptotic notations are mathematical notations used to represent the time complexity of algorithms for Asymptotic analysis.
- Following are the commonly used asymptotic notations to calculate the running time complexity of an algorithm.
  1. O Notation
  2. Ω Notation
  3. θ Notation
- This is also known as an algorithm's growth rate.
- Asymptotic Notations are used,
  1. To characterize the complexity of an algorithm.
  2. To compare the performance of two or more algorithms solving the same problem.

The Class P

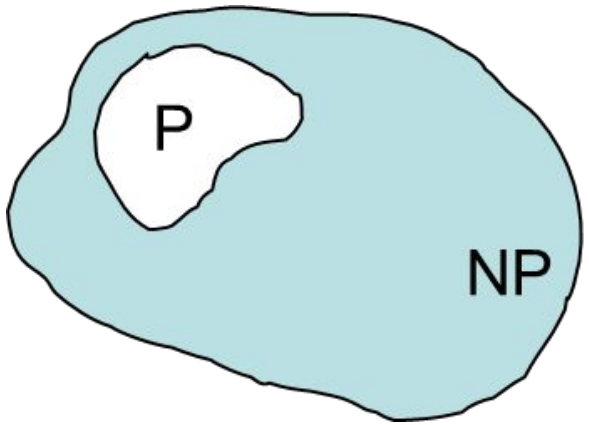
- The class P consists of those problems that are solvable in polynomial time by deterministic algorithms.
- More specifically, they are problems that can be solved in time $O(n^k)$ for some constant k, where n is the size of the input to the problem.
- For example, $O(n^3)$, $O(n^4)$, $O(logn)$, Fractional Knapsack, MST, Sorting algorithms etc…
- P is a complexity class that represents the set of all decision problems that can be solved in polynomial time.
- That is, given an instance of the problem, the answer yes or no can be decided in polynomial time.

The NP class

- NP is Non-Deterministic polynomial time.
- The class NP consists of those problems that are verifiable in polynomial time.
- NP is the class of decision problems for which it is easy to check the correctness of a claimed answer, with the help of a little extra information.
- Hence, we are not asking for a way to find a solution, but only to verify that an alleged solution really is correct.
- Every problem in this class can be solved in exponential time using exhaustive search.

P and NP Class Problems

- P = set of problems that **can be solved** in polynomial time
- NP = set of problems for which a solution **can be verified** in polynomial time
- $P \subseteq NP$



Classification of NP Problems

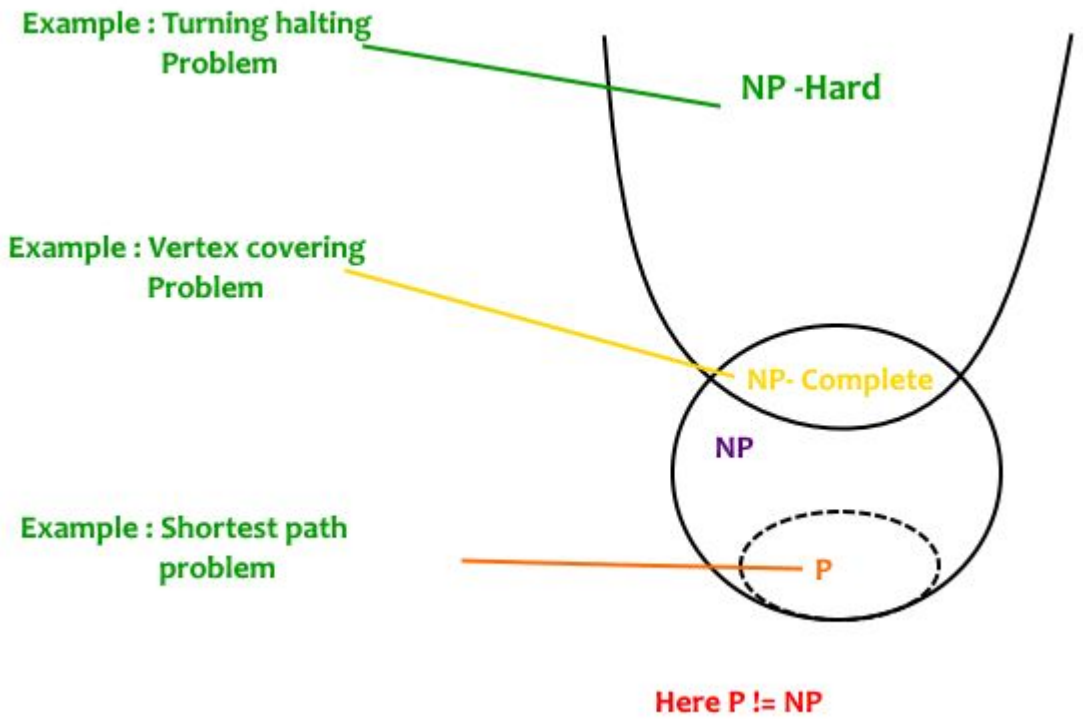
NP Complete

- NP-complete problems are a set of problems to each of which any other NP-problem can be reduced in polynomial time, and whose solution may still be verified in polynomial time.
- No polynomial-time algorithm has been discovered for an NP-Complete problem.
- *NP-Complete is a complexity class which represents the set of all problems X in NP for which it is possible to reduce any other NP problem Y to X in polynomial time.*

NP Hard

- NP-hard problems are those at least as hard as NP problems, i.e., all NP problems can be reduced (in polynomial time) to them.
- NP-hard problems need not be in NP, i.e., **they need not have solutions verifiable in polynomial time.**
- *The precise definition here is that a problem X is NP-hard, if there is an NP-complete problem Y, such that Y is reducible to X in polynomial time.*

P, NP Complete and NP Hard



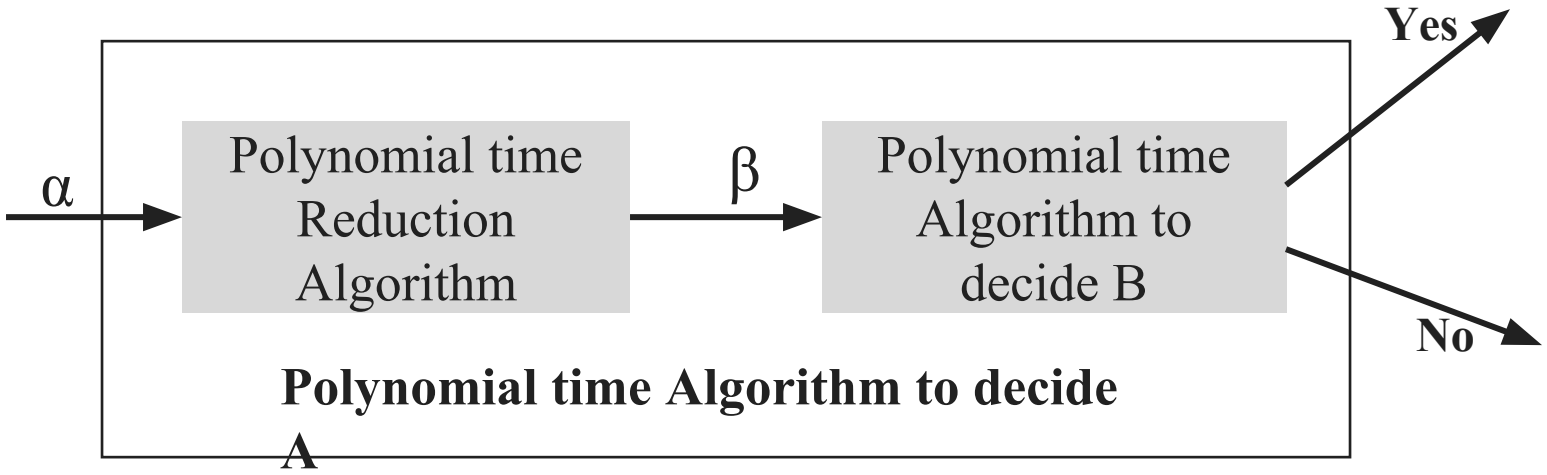
Polynomial Reduction

Introduction

- Let us consider a decision problem A, which we would like to solve in polynomial time.
- Now suppose, we already know how to solve a different decision problem B in polynomial time.
- Finally we have a procedure that transforms any instance α of A into some instance β of B with the following characteristics.
  1. The transformation takes polynomial time.
  2. The answers are same. That is the answer for α is "yes" if and only if the answer of β is also "yes".
- We call such a procedure a polynomial-time reduction algorithm and it provides us a way to solve problem A in polynomial time:

Polynomial Reduction

1. Given an instance α of problem A, use a polynomial-time reduction algorithm to transform it to an instance β of problem B.
2. Run the polynomial-time decision algorithm for B on the instance β.
3. Use the answer for β as the answer for α.

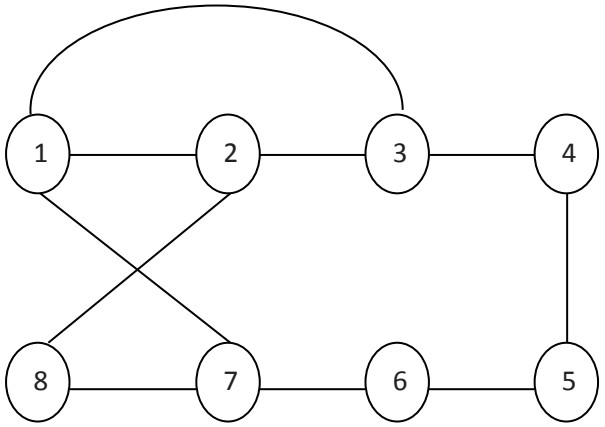


- In other words, by "reducing" **solving problem A to solving problem B,** we use the "easiness" of B to prove the "easiness" of A.

NP Hard Problems

Hamiltonian Cycles

- Hamiltonian Path in an undirected graph is a path that visits each vertex exactly once.
- A Hamiltonian cycle (or Hamiltonian circuit) is a Hamiltonian Path such that there is an edge (in the graph) from the last vertex to the first vertex of the Hamiltonian Path.



The graph has Hamiltonian cycles:
1, 3, 4, 5, 6, 7, 8, 2, 1 and 1, 2, 8, 7, 6, 5, 4, 3, 1.

- Given a list of vertices and to check whether it forms a Hamiltonian cycle or not:
- Counts the vertices to make sure they are all there, then checks that each is connected to the next by an edge, and that the last is connected to the first.

Hamiltonian Cycles

- It takes time proportional to n, because there are n vertices to count and n edges to check. n is a polynomial, so the check runs in polynomial time.
- To find a Hamiltonian cycle from the given graph: There are $n!$ different sequences of vertices that might be Hamiltonian paths in a given n-vertex graph, so a brute force search algorithm that tests all possible sequences can not be solved in polynomial time.
- In the traveling salesman Problem, a salesman must visits n cities.
- We can say that salesman wishes to make a tour or Hamiltonian cycle, visiting each city exactly once and finishing at the city he starts from.

Traveling Salesman Problem

- In TSP, we find a tour and check that the tour contains each vertex once. Then the total cost of the edges of the tour is calculated.
- How would you verify that the solution you're given really is the shortest loop? In other words, how do you know there's not another loop that's shorter than the one given to you?
- The only known way to verify that a provided solution is the shortest possible solution is to actually solve TSP.
- Since it takes exponential time to solve NP, the solution cannot be checked in polynomial time. Thus this problem is NP-hard, but not in NP.

Approximation Algorithm

- An optimization problem is the problem of finding the best solution from the set of all feasible solutions.
- The objective is a quantitative measure of system's performance that may be either minimized or maximized depending on the nature of a problem considered.
- A large number of optimization problems which are required to be solved in practice are NP-complete or NP-hard.
- For such problems, it is not possible to design algorithms that can find exactly optimal solution to all instances of the problem in polynomial time.
- But it may still be possible to find near-optimal solutions in polynomial time. In practice, near-optimality is often good enough.

Approximation Algorithm

- An algorithm that returns near-optimal solutions is called an approximation algorithm.
- Approximation Algorithm Definition: Given an optimization problem P, an algorithm A is said to be an approximation algorithm for P, if for any given instance I, it returns an approximate solution, that is a feasible solution.
- The goal of an approximation algorithm is to come as close as possible to the optimum value in a reasonable amount of time which is at the most polynomial time.

Randomized Algorithms

- Probability and randomness are often used as a tool for algorithm design and analysis, as in many cases, we know very little about the input distribution.
- Even if we do know something about the distribution, we may not be able to model this knowledge computationally.
- More generally, we call an algorithm randomized if its behavior is determined not only by its input but also by values produced by a random-number generator.
- A randomized algorithm is an algorithm that employs a degree of randomness as part of its logic, for example, in Randomized Quick Sort, we use a random number to pick the next pivot.
- The algorithm typically uses uniformly random bits as an auxiliary input to guide its behavior, in the hope of achieving good performance in the "average case" over all possible choices of random bits.
- It is typically used to reduce either the running time, or time complexity; or the memory used, or space complexity, in a standard algorithm.

Thank You!