P, NP, NP-Hard and NP-Complete

Dr. Shruti Mishra
SCOPE

Contents

- Types of Problems
- P, NP, NP-Hard and NP-Complete

Types of Problems

- Tractable
- Intractable
- Decision
- Optimization

Tractable : Problems that can be solvable in a reasonable (polynomial) time.

Intractable : Some problems are *intractable*, as they grow large, we are unable to solve them in reasonable time.

Tractability

- What constitutes reasonable time?

– Standard working definition: *polynomial time*

– On an input of size n the worst-case running time is $O(n^k)$ for some constant k

– $O(n^2)$, $O(n^3)$, $O(1)$, $O(n \lg n)$, $O(2^n)$, $O(n^n)$, $O(n!)$

– Polynomial time: $O(n^2)$, $O(n^3)$, $O(1)$, $O(n \lg n)$

– Not in polynomial time: $O(2^n)$, $O(n^n)$, $O(n!)$

- Are all problems solvable in polynomial time?

– No: Turing's "Halting Problem" is not solvable by any computer, no matter how much time is given.

Optimization /Decision Problems

<u>Optimization Problems</u>

– An optimization problem is one which asks, “What is the optimal solution to problem X?”

– Examples:

- 0-1 Knapsack
- Fractional Knapsack
- Minimum Spanning Tree

<u>Decision Problems</u>

– An decision problem is one with yes/no answer

– Examples:

- Does a graph G have a MST of weight ≤ W?

[contd..]

- An optimization problem tries to find an optimal solution.
- A decision problem tries to answer a yes/no question.
- Many problems will have decision and optimization versions
  - Eg: Traveling salesman problem
    - optimization: find hamiltonian cycle of minimum weight
    - decision: is there a hamiltonian cycle of weight $\leq k$

What is Language?

- Every decision problem can have only two answers, yes or no.
- Hence, a decision problem may belong to a language if it provides an answer ‘yes’ for a specific input.
- A language is the totality of inputs for which the answer is Yes.

P-Class

- The class P consists of those problems that are solvable in polynomial time, i.e. these problems can be solved in time $\boldsymbol{O(n^k)}$ in worst-case, where **k** is constant.
- These problems are called **tractable**, while others are called **intractable or super polynomial**.
- Formally, an algorithm is polynomial time algorithm, if there exists a polynomial $\boldsymbol{p(n)}$ such that the algorithm can solve any instance of size **n** in a time $\boldsymbol{O(p(n))}$.

P-Class [Contd..]

- Problem requiring $\boldsymbol{\Omega(n^{50})}$ time to solve are essentially intractable for large $\boldsymbol{n}$. Most known polynomial time algorithm run in time $\boldsymbol{O(n^k)}$ for fairly low value of $\boldsymbol{k}$.
- The advantages in considering the class of polynomial-time algorithms is that all reasonable **deterministic single processor model of computation** can be simulated on each other with at most a polynomial slow-d.

What is the complexity of primality testing?

```
public static boolean isPrime(int n){
  boolean answer = (n>1)? true: false;

  for(int i = 2; i*i <= n; ++i)
  {
    System.out.printf("%d\n", i);
    if(n%i == 0)
    {
      answer = false;
      break;
    }
  }
  return answer;
}
```

This loops until the square root of n
So this should be $O(\sqrt{n})$

But what is the input size?
How many bits does it take to represent the number n?
log(n) = k

What is $\sqrt{n}$

$$\sqrt{n} = \sqrt{2^{log(n)}} = (2^k)^{0.5}$$

Naïve primality testing is exponential!!

Why obsess about primes?

- Crypto uses it heavily
- Primality testing actually is in P
- Proven in 2002
  - Uses complicated number theory
  - AKS primality test

NP- Class

- **NP is not the same as non-polynomial complexity/running time. NP does not stand for not polynomial.**
- **NP = Non-Deterministic polynomial time**
- NP means verifiable in polynomial time
- Verifiable?
  - If we are somehow given a ‘certificate’ of a solution we can verify the legitimacy 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 aid of a little extra information.
- Hence, we aren’t 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.

Sample Problems in NP

- Fractional Knapsack
- MST
- Others?
  - ✓ Travelling Salesman
  - ✓ Graph Colouring
  - ✓ Satisfiability (SAT)
    - ✓ the problem of deciding whether a given Boolean formula is satisfiable

P versus NP

- Every decision problem that is solvable by a deterministic polynomial time algorithm is also solvable by a polynomial time non-deterministic algorithm.
- All problems in P can be solved with polynomial time algorithms, whereas all problems in *NP* - *P* are intractable.
- It is not known whether ***P = NP***. However, many problems are known in NP with the property that if they belong to P, then it can be proved that P = NP.
- If ***P ≠ NP***, there are problems in NP that are neither in P nor in NP-Complete.
- The problem belongs to class **P** if it's easy to find a solution for the problem. The problem belongs to **NP**, if it's easy to check a solution that may have been very tedious to find.

NP-hard

What does NP-hard mean?

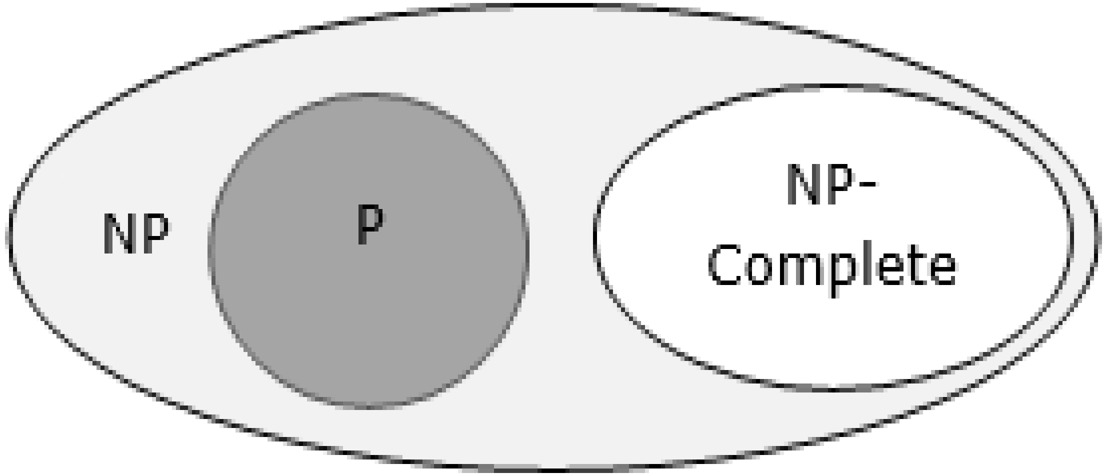
- A lot of times you can solve a problem by reducing it to a different problem. I can reduce Problem B to Problem A if, given a solution to Problem A, I can easily construct a solution to Problem B. (In this case, "easily" means "in polynomial time.“).
- A problem is **NP-hard** if all problems in NP are polynomial time reducible to it, ...
- Ex- Hamiltonian Cycle
- Every problem in NP is reducible to HC in polynomial time. Ex:- TSP is reducible to HC.

NP-Hard Problems

- The following problems are NP-Hard
  - The circuit-satisfiability problem
  - Set Cover
  - Vertex Cover
  - Travelling Salesman Problem

NP-complete

- A problem is **NP-complete** if the problem is both
  - NP-hard, and
  - NP



Definition of NP-Completeness

- A language **B** is ***NP-complete*** if it satisfies two conditions
  - **B** is in NP
  - Every **A** in NP is polynomial time reducible to **B**.
- If a language satisfies the second property, but not necessarily the first one, the language **B** is known as **NP-Hard**.
- Informally, a search problem **B** is **NP-Hard** if there exists some **NP-Complete** problem **A** that Turing reduces to **B**.
- The problem in NP-Hard cannot be solved in polynomial time, until **P = NP**.
- If a problem is proved to be NPC, there is no need to waste time on trying to find an efficient algorithm for it.
- Instead, we can focus on design approximation algorithm.

NP-Complete Problems

- Following are some NP-Complete problems, for which no polynomial time algorithm is known.
  - Determining whether a graph has a Hamiltonian cycle
  - Determining whether a Boolean formula is satisfiable, etc.

Solve?

TSP is NP-Complete

- The traveling salesman problem consists of a salesman and a set of cities.
- The salesman has to visit each one of the cities starting from a certain one and returning to the same city.
- The challenge of the problem is that the traveling salesman wants to minimize the total length of the trip