Class P, Class NP and NP-Completeness

Almost all the algorithms we have studied thus far have been polynomial-time algorithms: on inputs of size n, their worst-case running time is $O(n^k)$ for some constant k. It is natural to wonder whether all problems can be solved in polynomial time. The answer is no. For example, there are problems, such as Turing's famous "Halting Problem," that cannot be solved by any computer, no matter how much time is provided.

P Class

The P in the P class stands for Polynomial Time. It is the collection of decision problems (problems with a “yes” or “no” answer) that can be solved by a deterministic machine in polynomial time. 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.

Deterministic Machines :

- A deterministic machine operates based on deterministic decisions. At any given point in its computation, it follows a single, well-defined set of rules to determine the next step. Given the same input and internal state, a deterministic machine will always take the same action.

- A deterministic machine follows a single path of computation. It doesn't have the capability to explore multiple possibilities simultaneously.

- Examples of deterministic machines include deterministic finite automata (DFA), deterministic Turing machines (DTM), and traditional computers like your desktop or laptop.

Features:

- The solution to P problems is easy to find.

- P is often a class of computational problems that are solvable and tractable. Tractable means that the problems can be solved in theory as well as in practice. But the problems that can be solved in theory but not in practice are known as intractable.

Examples of P Class Algorithms:

Quicksort: Quicksort is a widely used sorting algorithm. It has an average-case time complexity of O(n log n) and a worst-case time complexity of O(n^2)

Merge Sort: Merge Sort is another sorting algorithm with a time complexity of O(n log n) in both the average and worst cases.

Binary Search: Binary search is an algorithm used to find a specific element in a sorted array. Its time complexity is O(log n), making it very efficient for large datasets.

Linear Search: While not as efficient as binary search for sorted data, linear search has a time complexity of O(n) and is suitable for unsorted lists or small datasets.

Breadth-First Search (BFS): BFS is an algorithm used for traversing or searching tree or graph data structures. Its time complexity is O(V + E), where V is the number of vertices and E is the number of edges.

Depth-First Search (DFS): DFS is another graph traversal algorithm with a time complexity of O(V + E), similar to BFS.

NP Class

The NP in NP class stands for Non-deterministic Polynomial Time. It is the collection of decision problems that can be solved by a non-deterministic machine in polynomial time.

Non-deterministic Machines:

- A non-deterministic machine can make non-deterministic decisions during computation. It has the ability to choose between multiple options at certain decision points. This means it can follow different computation paths at the same time.

- A non-deterministic machine can explore multiple possible computation paths simultaneously. It can "guess" or "branch" into different choices and explore different scenarios concurrently.

- Examples of non-deterministic machines include non-deterministic finite automata (NFA), non-deterministic Turing machines (NTM), and certain computational models used in theoretical computer science.

Features:

The solutions of the NP class are hard to find since they are being solved by a non-deterministic machine but the solutions are easy to verify.

Problems of NP can be verified by a Turing machine in polynomial time

Non Deterministic Search Algorithms

- A decision problem gives answer either 0 or 1
- Search a element x in a given set A, write index else 0
  - → **J = choice (1,n); // Select form a set**
  - → **If A[j] = x then {write (j);success();} //write index**
  - → **Write(0); failure();**
- Computing time of this algorithm is O(1) in both cases , HOW ?
- Any Deterministic search algorithm requires O(n) time – Linear

Examples of NP Class Algorithms:

Traveling Salesman Problem (TSP): Given a list of cities and the distances between each pair of cities, find the shortest possible route that visits each city exactly once and returns to the starting city. Verification involves checking if the proposed route is indeed a valid tour that visits each city once and has a total distance within a given limit.

Hamiltonian Cycle Problem: Given a graph, determine whether there exists a Hamiltonian cycle (a cycle that visits each vertex exactly once). Verification involves checking if the proposed cycle is Hamiltonian.

NP-Completeness

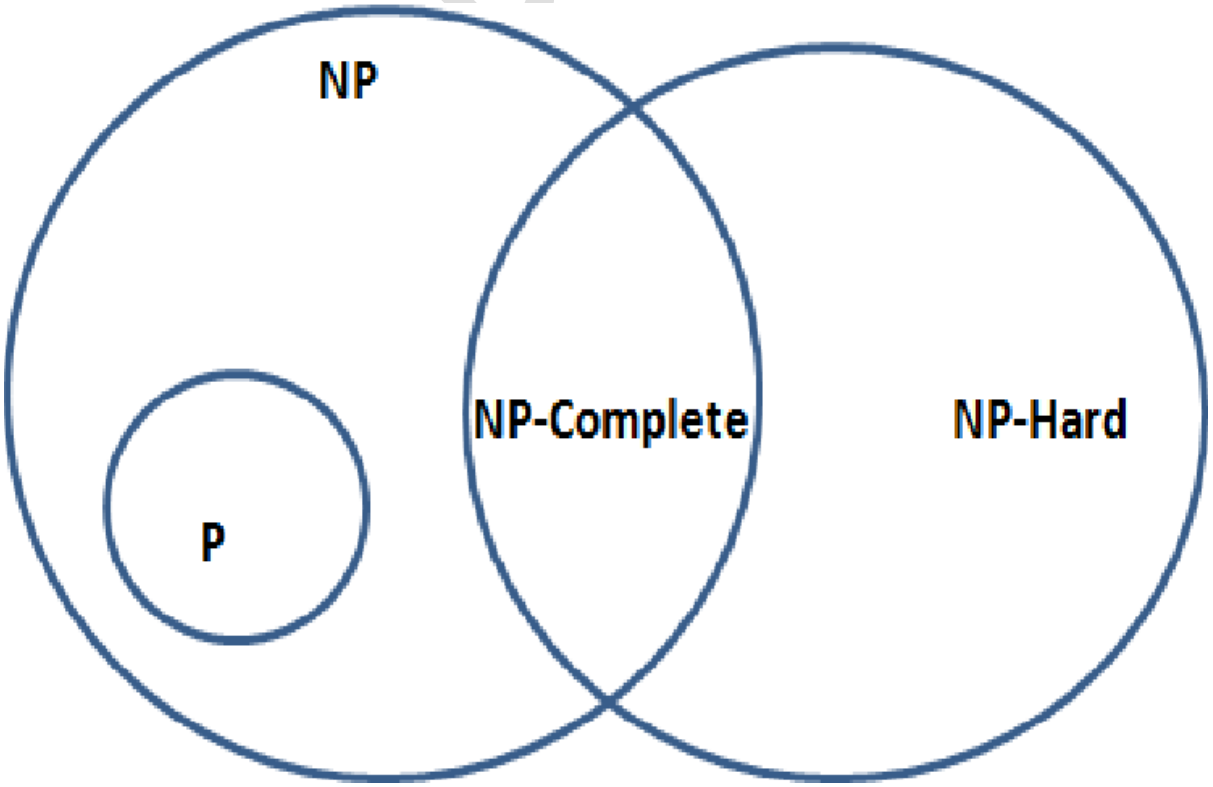
NP-completeness is a fundamental concept in computational complexity theory that helps classify and analyze problems based on their computational difficulty. Problems classified as NP-complete are some of the hardest problems in the NP (nondeterministic polynomial time) complexity class. Understanding NP-completeness is crucial in determining the difficulty of solving a wide range of computational problems. Here's a breakdown of NP-completeness:

Definition of NP-Complete:

NP-Complete (Nondeterministic Polynomial Time Complete): A problem is NP-complete if it satisfies two conditions:

a. It is in NP, meaning solutions can be verified in polynomial time.

b. It is at least as hard as the hardest problems in NP, in a certain sense.



Cook's Theorem:

Stephen Cook's theorem in 1971 was the first to introduce the concept of NP-completeness. He showed that the Boolean satisfiability problem (3-SAT) is NP-complete. This means that if you can efficiently solve 3-SAT, you can efficiently solve any problem in NP, making 3-SAT one of the hardest problems in NP.

Reduction:

The concept of NP-completeness relies heavily on the idea of reduction. If you can show that problem A can be reduced to problem B in polynomial time (i.e., solving problem A helps you solve problem B), and problem B is NP-complete, then problem A is also NP-complete.

Implications of NP-Completeness:

If you can prove that a new problem C is NP-complete by reducing an existing NP-complete problem to it, then problem C is at least as hard as the hardest problems in NP.

The existence of NP-complete problems implies that P (polynomial time) is not equal to NP, which is one of the most significant open questions in computer science and mathematics. If P = NP, it would mean that all problems in NP can be solved efficiently (in polynomial time), but this has not been proven or disproven.

Usefulness: NP-completeness is a valuable concept in computer science and problem-solving because it helps identify problems that are unlikely to have efficient (polynomial time) solutions. If you encounter a problem that is NP-complete, you can expect that finding an efficient algorithm for it is challenging, and you may need to explore heuristic or approximation approaches.