

NP-Hard and NP-Complete:

Syllabus: NP-Hard and NP-Complete problems: Basic concepts, non-deterministic algorithms, NP-Hard and NP-Complete classes, Cook's theorem.

What is a Polynomial-Time Algorithm?

An algorithm is polynomial if its running time is of the form: $O(n^k)$ where k is a constant (1, 2, 3, 7...).

Such algorithms grow reasonably fast and are considered efficient.

Examples of Polynomial-Time Algorithms

Problem	Algorithm	Time
Searching	Hash table	$O(1)$
Sorting	Merge Sort	$O(n \log n)$
Shortest Path	Dijkstra	$O(V^2)$ or $O((V+E) \log V)$
Minimum Spanning Tree	Kruskal	$O(E \log V)$
Matrix Multiplication	Standard	$O(n^3)$
Checking primality	AKS Primality Test	$O(n^6)$

What is a Non-Polynomial Algorithm? (Exponential / Factorial)

An algorithm is non-polynomial if its running time is like:

$O(2^n)$, $O(3^n)$, . . . , $O(n!)$, $O(n \log n)$

These grow extremely fast, considered inefficient for large input sizes.

Problem	Time	Category
Traveling Salesman Problem (TSP)	$O(n!)$ or $O(2^n \cdot n^2)$	NP-hard
0/1 Knapsack	$O(2^n)$	NP-complete
Subset Sum	$O(2^n)$	NP-complete
Hamiltonian Cycle	$O(n!)$	NP-complete
Graph Coloring	$O(k^n)$	NP-complete
SAT (Brute Force)	$O(2^n)$	NP-complete

Deterministic and non-deterministic algorithms

Deterministic: The algorithm in which every operation is uniquely defined is called deterministic algorithm.

Non-Deterministic: The algorithm in which the operations are not uniquely defined but are limited to specific set of possibilities for every operation, such an algorithm is called non-deterministic algorithm.

The non-deterministic algorithms use the following functions:

1. Choice: Arbitrarily chooses one of the elements from given set.
2. Failure: Indicates an unsuccessful completion
3. Success: Indicates a successful completion

Search(**Arr**, **key**) in **O(1)** (nondeterministic)

begin

i = choice() //Step 1: Nondeterministic Guessing in *O(1)*, choice() picks '*j*' in *O(1)*

 if *Arr*[*i*] == *key* then //Step 2: Polynomial-Time Verification

 write(*i*)

 success()

 else

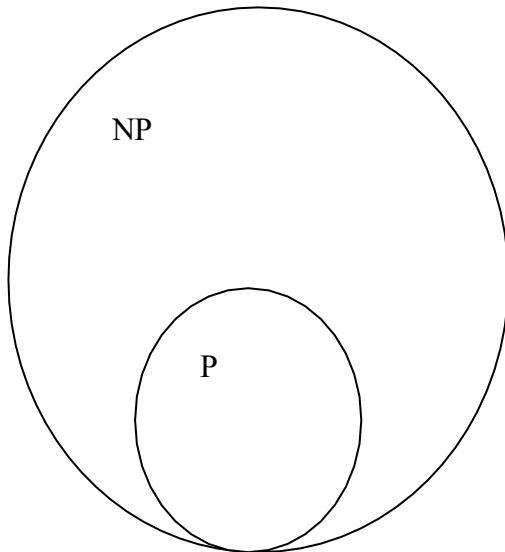
 write(0)

 failure()

 end if

end

A nondeterministic algorithm is considered as a superset for deterministic algorithm i.e., the deterministic algorithms are a special case of nondeterministic algorithms.



Satisfiability Problem:

The satisfiability is a boolean formula that can be constructed using the following literals and operations.

1. A literal is either a variable or its negation of the variable.
2. The literals are connected with operators \vee , \wedge , \Rightarrow , \Leftrightarrow
3. Parenthesis

The satisfiability problem is to determine whether a Boolean formula is true for some assignment of truth values to the variables. In general, formulas are expressed in Conjunctive Normal Form (CNF).

A Boolean formula is in conjunctive normal form if it is represented by

$$(x_i \vee x_j \vee x_k^1) \wedge (x_i \vee x^1 \vee x_k) \quad j$$

A Boolean formula is in 3CNF if each clause has exactly 3 distinct literals. Example:

The non-deterministic algorithm that terminates successfully iff a given formula $E(x_1, x_2, x_3)$ is satisfiable.

Reducibility:

A problem L1 can be reduced to L2 if any instance of L1 can be easily rephrased as an instance of L2. If the solution to the problem L2 provides a solution to the problem L1, then these are said to be reducible problems.

Let L1 and L2 are the two problems. L1 is reduced to L2 if there is a way to solve L1 by a deterministic polynomial time algorithm that solves L2 in polynomial time and is denoted by $L1 \alpha L2$.

If we have a polynomial time algorithm for L2 then we can solve L1 in polynomial time. Two problems L1 and L2 are said to be polynomial equivalent if $L1 \alpha L2$ and $L2 \alpha L1$.

Example: Let P1 be the problem of selection and P2 be the problem of sorting. Let the input have n numbers. If the numbers are sorted in array $A[]$ the i^{th} smallest element of the input can be obtained as $A[i]$. Thus, P1 reduces to P2 in $O(1)$ time.

NP-Hard and NP-Complete Problem:

Let P denote the set of all decision problems solvable by deterministic algorithm in polynomial time. NP denotes set of decision problems solvable by nondeterministic algorithms in polynomial time. Since, deterministic algorithms are a special case of nondeterministic algorithms, $P \subseteq NP$.

The nondeterministic polynomial time problems can be classified into two classes. They are

1. NP Hard and
2. NP Complete

NP-Hard: A problem L is NP-Hard if satisfiability reduces to L i.e., any nondeterministic polynomial time problem is satisfiable and reducible then the problem is said to be NP-Hard.

NP-Complete: A problem L is NP-Complete if L is NP-Hard and L belongs to NP (nondeterministic polynomial).

→ A problem that is NP-Complete has the property that it can be solved in polynomial time if all other NP-Complete problems can also be solved in polynomial time. ($NP=P$)

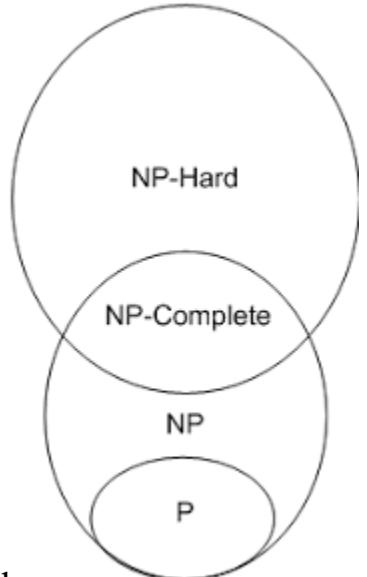
If an NP-hard problem can be solved in polynomial time, then all NP-complete problems can be solved in polynomial time. All NP-Complete problems are NP-hard, but some NP-hard problems are not known to be NP-Complete.

Normally the decision problems are NP-complete but the optimization problems are NP-Hard.

However, if problem L_1 is a decision problem and L_2 is an optimization problem, then it is possible that $L_1 \neq L_2$.

Example: Knapsack decision problem can be reduced to knapsack optimization problem.

There are some NP-hard problems that are not NP-Complete.



Relationship between P, NP, NP-hard, NP-Complete

Let P , NP , NP -hard, NP -Complete are the sets of all possible decision problems that are solvable in polynomial time by using deterministic algorithms, non-deterministic algorithms, NP-Hard and NP-complete respectively. Then the relationship between P , NP , NP -hard, NP -Complete can be expressed using Venn diagram as:

Design and Analysis of Algorithm

