Basic concept of NP Hard & NP Complete

NP-Hard and NP-Complete Problems

For many of the problems we know and study, the best algorithms for their solution have computing times can be clustered into two groups-

1. **Solutions are bounded by the polynomial**
2. **Solutions are bounded by a nonpolynomial**

No one has been able to device an algorithm which is bounded by the polynomial of small degree for the problems belonging to the second group.

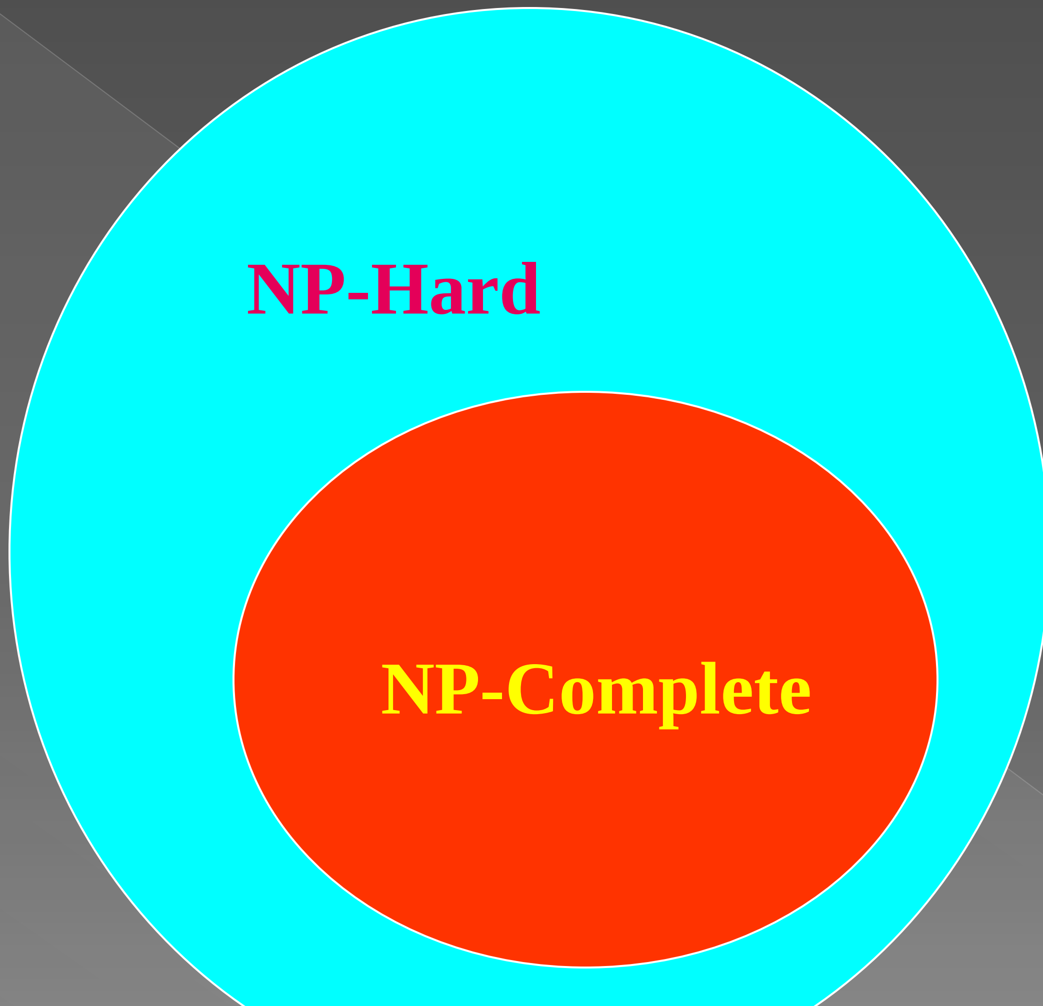
Perhaps with Quantum Computing ! Who knows? Open ?

The theory of the NP-Completeness does not provide any method of obtaining polynomial time algorithms for the problems of the second group. "***Many of the problems for which there is no polynomial time algorithm available are computationally related***".

Two classes

1. **NP-Complete-** have the property that it can be solved in polynomial time if all other NP-Complete problems can be solved in polynomial time.

2. **NP-Hard-** if it can be solved in polynomial time then all NP-Complete can be solved in polynomial time.


NP-Hard
NP-Complete

"***All NP-Complete problems are NP-Hard but not all NP-Hard problems are not NP-Complete.***"

- NP-Complete problems are subclass of NP-Hard

Non deterministic algorithms

When the result of every operation is uniquely defined then it is called **deterministic algorithm**. When the outcome is not uniquely defined but is limited to a specific set of possibilities, we call it **non deterministic algorithm**.

We use new statements to specify such algorithms.

1. choice(S) arbitrarily choose one of the elements of set S

2. failure signals an unsuccessful completion

3. success signals a successful completion

The assignment X:= choice(1:n) could result in X being assigned any value from the integer range[1..n]. There is no rule specifying how this value is chosen.

The nondeterministic algorithms terminates unsuccessfully iff there is no set of choices which leads to the successful signal.

The computing time for failure and success is taken to be $O(1)$. A machine capable of executing a nondeterministic algorithms are known as nondeterministic machines **(does not exist in practice).**

Ex. Searching an element **x** in a given set of elements A(1:n). We are required to determine an index **j** such that A(j) = x or j = 0 if x is not present.

j := choice(1:n)

if A(j) = x then print(j); success endif

print('0'); failure

```
procedure NSORT(A,n);
//sort n positive integers//
var integer A(n), B(n), n, i, j;
begin
	B := 0; //B is initialized to zero//
	for i := 1 to n do
	begin
		j := choice(1:n);
		if B(j) <> 0 then failure;
		B(j) := A(j);
	end;
```

```
    for i := 1 to n-1 do  //verify order//
        if B(i) > B(i+1) then failure;
    print(B);
    success;
end.
```

A deterministic interpretation of the nondeterministic algorithm can be done by making unbounded parallelism in the computation.

Each time a choice is to be made, the algorithm makes several copies of itself, one copy is made for each of the possible choices.

"***Nondeterministic machines does not make any copies of an algorithm every time a choice is to be made. Instead it has the ability to correctly choose an element from the given set***".

Nondeterministic decision algorithm- Generates 0 or 1 as their output.

Many optimization problems can be recast in to decision problems with the property that the decision algorithm can be solved in polynomial time iff the corresponding optimization problem.

Definition: The ***time required by a nondeterministic algorithm*** performing on any given input is the minimum number of steps required to reach to a successful completion if there exists a sequence of choices leading to a such completion.

In case the successful completion is not possible then the time required is $O(1)$. A nondeterministic algorithm is of complexity $O(f(n))$ if for all input size n, $n \geq n_0$, that results in a successful completion the time required is at most $c.f(n)$ for some constant c and n_0.

procedure DKP(P, W, n, M, R, X);

var integer P(n), W(n), R, X(n), n, M, i;

begin

for i := 1 to n do

X(i) := choice(1, 0);

if $\sum_{1 \le i \le n} (W(i)X(i)) > M$ or $\sum_{1 \le i \le n} (P(i)X(i)) < R$ then failure

else success;

end.

Time complexity is O(n).

Satisfiability problem: Let $x_1, x_2, \ldots, x_n$ denotes boolean variables. Let $\bar{x}_i$ denotes the negation of x_i. A literal is either a variable or its negation. A formula in propositional calculus is an expression that can be constructed using literals and ***and*** or ***or***.

Formula is in ***conjugate normal form*** (CNF) iff it is represented as $\wedge_{i=1}^{k} c_i$, where the c_i are clauses each represented as V l_{ij}.

It is in ***disjunctive normal form*** (DNF) iff it is represented as $\vee_{i=1}^{k} c_i$ and each clause is represented as $\wedge l_{ij}$.

thus $(x_1 \wedge x_2) \vee (x_3 \wedge \bar{x}_4)$ is in DNF while $(x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_2)$ is in CNF. The satisfiability problem is to determine if a formula is true for some assignment of truth values to the variables.

procedure EVAL(E, n);

//determines if the propositional formula E is satisfiable//

var boolean: x[1..n];

begin

    for i := 1 to n do   //choose a truth value assignment//

        x_i := choice(true, false);

    if E(x_1,...,x_n) is true then success   //satisfiable//

    else failure

end.

NP-Hard and NP-Complete

An algorithm A is of ***polynomial complexity*** is there exist a polynomial $p(\ )$ such that the computing time of A is $O(p(n))$.

Definition: P is a set of all decision problems solvable by a deterministic algorithm in polynomial time. **NP** is the set of all decision problems solvable by a nondeterministic algorithm in polynomial time.

$$\Rightarrow P \subseteq NP$$

The most famous unsolved problem in Computer Science is whether $P=NP$ or $P \neq NP$ $\Rightarrow P \stackrel{?}{=} NP$

Cook's theorem: Satisfiability is in *P* if *P* = *NP*

Definition. Let L_1 and L_2 be problems. L_1 ***reduces*** to L_2 $(L_1 \alpha L_2)$ iff there is a way to solve L_1 by deterministic polynomial time algorithm that solve L_2 in polynomial time.

$\Rightarrow$ if we have a polynomial time algorithm for L_2 then we can solve L_1 in polynomial time.

Definition. A problem L is ***NP-Hard*** if and only if satisfiability reduces to L. ($satisfiability \, \alpha \, L$).

Definition. A problem L is ***NP-Complete*** if and only if L is NP-Hard and $L \in NP$.

Halting problem: An example of NP-Hard decision problem which is not NP-Complete.

Assignment

Q.1)What are the classes of NP problem?
Q.2)Explain nondeterministic algorithm with an example.
Q.3)Which problem is NP prpblem?