P and NP class Problems:

There are two groups in which a problem can be classified

1) **P-Class problems:**

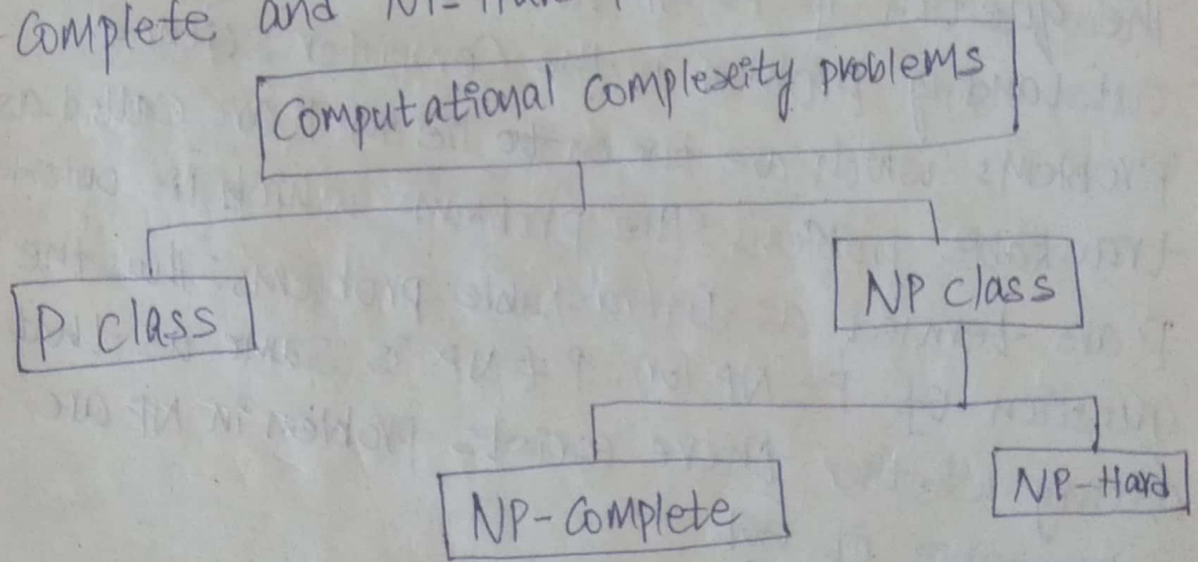
problems that can be solved in polynomial time. We provide the solution to these problems by using deterministic algorithm

Eg: Kruskal's algorithm
 Searching an element from the list
 Sorting of elements

2) **NP-class Problems:**

In this class the problems can be solved in non-deterministic polynomial time.

The NP-class problems can be further categorised into NP-Complete and NP-Hard problems



NP-Complete:

A problem D is called NP-Complete if it satisfies the following conditions

i) it belongs to NP-class

2) every problem in NP can be solved in polynomial-time

NP-Hard:

An NP-problem which is hard can be solved in polynomial time then such problems will be called as NP-hard problems

Scanned with CamScanner

* If an NP-Hard problem can be solved in polynomial time then all NP-Complete problems can also be solved in polynomial time.

* All NP-Complete problems are NP-Hard problems but all NP-Hard problems cannot be NP-Complete problems.

* The NP class problems are the decision problems which can be solved by Non-deterministic Polynomial algorithm.

↳ relation b/w P, NP and P-Complete & NP-Hard

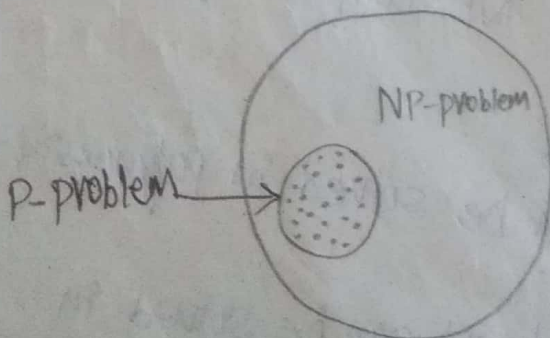
Properties of NP-Complete and NP-Hard Problems:

As we know, 'P' denotes the class of all deterministic polynomial language problems and 'NP' denotes the class of all Non-deterministic polynomial language problems. Hence $P \subseteq NP$

The question if P=NP holds the most famous outstanding problem in the computer science.

Problems which are known to lie in P are called as trackable problems. The problems which lie outside of P are termed as Intrackable problems. Thus the question of P=NP (or) $P \neq NP$ is same as that of asking whether there exists problem in NP are Intrackable or not.

The relationship b/w P & NP is depicted as follows



We don't know if P=NP however in 1971 a person Cook proved that a particular NP-problem called

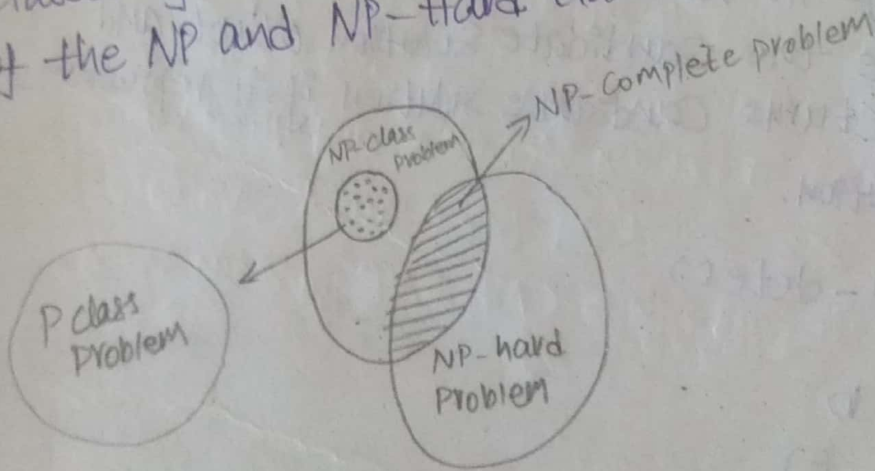
Scanned with CamScanner

SAT(satisfiability of sets of boolean clauses) has a property that, if it is solvable in polynomial time then we can prove P=NP

Let A and B are the two problems then problem A reduces to B if and only if there is a way to solve 'A' by deterministic polynomial time ~~algorithm~~ using and solves 'B' in polynomial time

'A' reduces to 'B' can be denoted as $A \propto B$. In other words we can say that if there exists any ~~deterministic~~ polynomial time algorithm which solves B then we can solve 'A' in polynomial time.

A NP-problem, such that if it is in 'P' then NP=P if the problem has the property that it is not nessarily necessarily NP then it is called as NP-Hard. The NP-Complete problem is the intersection classes of the NP and NP-Hard classes



Normally the decision problems are NP-complete and the optimisation problems are NP-Hard however if the problem 'A' is a decision problem and B is an optimisation problem then it is possible that $A \propto B$

For instance the knapsack decision problem can be changed to a knapsack optimisation problem. There are NP-Hard problems that are not NP-complete problems.

Scanned with CamScanner

example: Halting problem

Non-deterministic Algorithm:

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

* The algorithm in which every operation may not have unique result, rather there can be specified set of possibilities for every operation such an algorithm is called Non-deterministic algorithm.

Non deterministic means that no particular rule is followed to make the guess. The NDA is a two stage algorithm

Stage 1: Non deterministic stage (guessing stage)

- generate an arbitrary string that can be part of a candidate solution.

stage 2: Deterministic stage (verification stage)

- In this stage, it takes ~~the~~ i/p and verify that the result will be the candidate solution or not. and this solution returns candidate solution if it represents the actual solution.

```
Algorithm Non-dete()
{
for i=1 to n do
  A[i] = choose(i)
  if (A[i] = x) then
  {
  write (i);
  Success();
  }
  write (0);
  fail()
}
```

Scanned with CamScanner

In the above algorithm we are using 3 functions

i) choose() : Chooses one of the element arbitrary from the given i/p set

ii) success(): indicates successful completion. This function returns that the candidate solution represents the actual solution

iii) fail():

indicates the unsuccessful completion. Here the a candidate solution does not represent the actual solution

Example: If the NDA for sorting the elements in non-decreasing order.

```
Algorithm Non-deteSort()
{
for i<-1 to n do
 B[i]<-0
for i<-1 to n do
 {
  j<-choose(1,n)
  if (B[j]!=0) then
   fail();
   B[j]<- A[i];
 }
for i<-1 to n-1 do
if (B[i] > B[i+1]) then
  fail();
write (B[1:n]);
 success();
 }
```

The time required by a NDA with some input set is Minimum no. of steps needed to reach a successful completion. If there are multiple choices from input set it requires maximum no. of steps to a successful completion and the time complexity is $O(f(n))$ where n is the total size of input.

Scanned with CamScanner

CNF-SAT Problem:

This problem is based on boolean formula. The boolean formula has various boolean operations such as AND, OR and NOT. Here we have some notations such as $\rightarrow$ (implies), $\leftrightarrow$ (if and only if)

A boolean formula is in Conjuctive Normal Form (CNF), if it is formed as Collection of sub-expressions. This sub-expressions are called as Clauses

Example:

$(\overline{a}+b+d+\overline{g}).(c+\overline{e}).(\overline{b}+d+\overline{f}+h).(a+c+e+\overline{h})$

This formula evaluates to '1' if b, c, d are '1'

The CNF SAT is a problem which takes boolean formula in Conjuctive normal form and checks any assignment of boolean values for the formula

Theorem: CNF SAT is in NP-Complete

Proof: Let s be the formula for which we can Construct a simple Non-deterministic algorithm which can guess the values of variables in boolean formula and then evaluates each clause of 's' if all the clauses evaluate 's' to 1 then s is satisfied. Thus CNF SAT problem is in NP-Complete.

3 SAT Problem:

A 3 SAT Problem which takes a boolean formula 's' in CNF form with each clause having exactly three literals and check whether s is satisfied or not

Following formula is an instance of 3 SAT problem

$s=(\overline{a}+b+\overline{g}).(c+\overline{e}+f)(\overline{b}+d+\overline{f})(a+e+\overline{h})$

Theorem: 3 SAT is in NP-Complete

3SAT is NP-complete

Proof: Let 's' be the boolean-formula having 3-literals in each clause for which we can construct a simple Non-Deterministic Algorithm which can guess an assignment of boolean values to 's' is evaluated as 1 then 's' is satisfied and we can say that 3 SAT is NP Complete

Cook's Theorem:

Scientist steaphen Cook in 1971 stated that Boolean Satisfiability Problem is NP-complete.

Proof: Any instance of SAT Problem is a boolean expression in which boolean variables are combined using boolean operators. An expression is satisfiable if it's value results to be true on some assignments of boolean variables. The SAT Problem is in NP, this is because a Non-deterministic algorithm can guess an assignment of truth values of variables.

This algorithm can also determine the value of expression for corresponding assignment and can accept the entire expression is true.

The algorithm is composed of

i) Input Tape
ii) Read/Write head
iii) Each cell contains only one symbol at a time
iv) Computation is performed in number of states
v) The algorithm terminates when it reaches the accept state.

The conjunction clauses for boolean expression are given in the following table:

Scanned with CamScanner

Clauses	Meaning	Time Complexity
T_{ij0}	cell i of input tape contains symbol 'j'	$O(P(n^2))$
Q_{s0}	initial state	$O(1)$
H_{00}	initial position of tape head	$O(1)$
$T_{ijk} = T_{ij(k+1)} \vee H_{ik}$	tape remains unchanged unless ~~return~~ written	$O(P(n^2))$
$Q_{qk} \rightarrow \sim Q_{qk}$	one state at a time	$O(P(n))$
$H_{ik} \rightarrow \sim H_{ik}$	one read/write head position at a time	$O(P(n^2))$
$T_{ijk} \rightarrow \sim T_{ijk}$	one symbol per tape cell at a time	$O(P(n^2))$

Note that 'H' denotes head, 'Q' denotes states and 'T' denotes tape.

The disjunction clauses for the boolean expressions are given in the following table

Clauses	Meaning	Time Complexity
$(H_{ik} \wedge Q_{qk} \wedge T_{ijk}) \rightarrow H_{(i+j)(k+1)} \wedge Q_{q(k+1)} \wedge T_{ij(k+1)}$	possible transitions	$O(P(n^2))$
Disjunction of all clauses 'Q'	moving to accept state	$O(1)$

If B is satisfiable then there is an accepting state in the algorithm. Thus the proof shows that Boolean satisfiability problem solved in polynomial time. Hence

Scanned with CamScanner

all problem in NP could be solved in polynomial time and hence class NP could be equal to P.

Halting Problem:-

The Halting problem

It is not possible to determine for an arbitrary deterministic algorithm 'A' and input I. whether the algorithm A with input I ever terminates or enters into an infinite loop. This type of problems are undecidable problems. Hence there is no algorithm to solve this problem. This type of problems comes under NP-Hard. To prove that the halting problem is NP-Hard which is not in NP, we will make use of satisfiability.

Construct an algorithm 'A' with an input 'W'. The input W is a proportional formula with n-variables. Then algorithm 'A' tries all 2^n possible truth assignments and determines whether W is satisfiable or not. If W is satisfiable then 'A' halts successfully but if it is not, then A enters in an infinite loop.

If we had a polynomial time algorithm for the halting problem then we could solve the satisfiability problem in polynomial time with the help of algorithm 'A' and input W. This proves that Halting problem is NP-Hard that is not in NP

Scanned with CamScanner

Reductions for some known problems:

A clique in an undirected graph $G=(V,E)$ is a subset of V vertices, each pair of which is connected by an edge in E. In other words clique is a complete subgraph of G. The size of clique is the no. of vertices it contains. The clique problem is the optimisation problem of finding a clique of maximum size in a graph. As a decision problem V as simply whether a clique of a given size k exists in the graph. The formal definition is clique = {<G, k> : G is a graph with a clique of size k}

A Naive Algorithm for determining whether a graph $G=(V,E)$ with $|V|$ vertices as a clique of size k is to list all k subsets of V and check each one to see whether it forms a clique. The running time of this algorithm is $\Omega(k^2 C k^v)$, which is polynomial, if k is a constant. In general however k could be proportional to $|V|$ in which case the algorithm runs in super polynomial time.

The Max clique is an optimisation problem that has to determine the size of a largest clique in G. The corresponding decision problem is to determine whether G is a clique of size atleast 'k' for some given k.
Let Dclique (G, k) be a deterministic decision Algorithm for the click decision problem if the no. of vertices in G is n, the size of a max clique in G

can be found by making several applications of Dclique. Dclique is used once for each k, k = n, n-1, n-2 .... until the output from Dclique is '1'. If the time complexity of Dclique is f(n) then the size of a max clique can be found in time ≤ n.f(n). Also, if the size of a max clique can be determined in g(n) time, then the decision problem can also be solved in g(n) time. Hence the max clique problem can be solved in polynomial time if and only if the clique decision problem can be solved in polynomial time.

Scanned with CamScanner