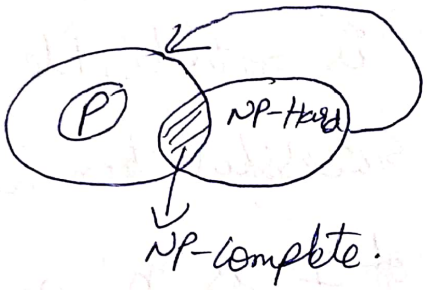
UNIT-5 DAA

10:- Cook's theorem:- (or) cook - Levin theorem:-

The cook - Levin theorem, also known as cook's theorem, states that the Boolean satisfiability (SAT) problem is NP-complete. That is, it is in NP, and any problem in NP can be reduced in polynomial time by a deterministic Turning machine to the Boolean satisfiability problem.



→ Stephen Arthur Cook and L.A. Levin in 1973 independently proved that the satisfiability problem (SAT) is NP-complete.

⇒ It "states that satisfiability is in P if and only if $P = NP$".

→ According to defination of satisfiability

we already seen that satisfiability is in NP;
Hence, P=NP. i.e., satisfiability is also in P.

To prove the steps are :-

1. To show how to obtain from any polynomial time non-deterministic decision algorithms "A" and input "I" a formula Q(A, I) such that Q is satisfiable if and only if A has a successful termination with input I.
→ if length of I is 'n' and time complexity of A is P(n) for some polynomial time, then length of queue is given as

$$O(P^3(n) \log n) = O(P^4(n)).$$

the Time needed to construct Q is

$$O(P^3(n) \log n).$$

Scanned with OKEN Scanner

2. A deterministic algorithm to determine outcome of A on any input I can be easily obtained.

→ Algorithm Z simply computes Q and then uses a deterministic algorithm for satisfiability problem to determine whether queue is satisfiable.

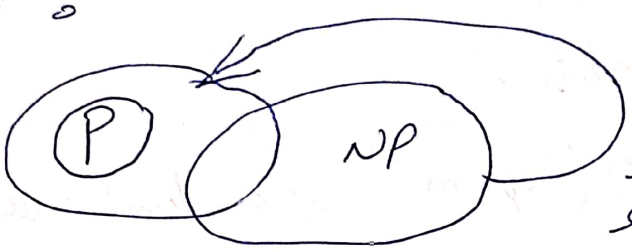
→ if $O(q(m))$ is a time needed to determine whether a formula of length m is satisfiable then complexity of Z is $O(p^3(n) \log n) + q(p^3(n) \log n)$

→ if satisfiability is in P then $q(m)$ is a polynomial function of M and complexity of Z becomes $O(r(n))$ for some polynomial R.

→ Hence, if satisfiability is in P, then for

every non-deterministic algorithm A in NP Can obtain a deterministic Z in P so the above construction shows that if Satisfiability is in P, then P=NP.

Short cut :-



any problem in NP can be reduced in P time.

It states, SAT is in P $\xrightarrow{\text{if and only if}}$ P=NP.

Hence P=NP i.e (SAT) is also in P.

Prove :-

to show How to obtain from any Polynomial time P → NP (non-deterministic decision algorithms "A" and input "I".

Scanned with OKEN Scanner

a formula Q(A, I),

such that Q is SAT $\xrightarrow{\text{if and only if}}$ A has successful termination with input i.

⇒ 1. length of I = n,

Time complexity of A = P(n)

then, length of queue is given as

$$O(P^3(n) \log n) = o(P^4(n)).$$

Time needed to construct Q is

$$O(P^3(n) \log n).$$

2. A deterministic algorithm to Determine Outcome of A on any input I can be easily obtained.

⇒ Alg Z computes Q and uses a deterministic algorithm for SAT problem to determine whether Queue is SAT.

⇒ if O(q(n)) is time needed,

Scanned with OKEN Scanner

formula of length n is SAT then,

Complexity of $Z = O(P^3(n)\log n) + q(P^3(n)\log n)$

⇒ if SAT is in P then $q(n)$ is P function of n and

complexity of Z becomes $O(r(n))$ for some polynomial R.

⇒ Hence, if SAT $\xrightarrow{\text{is in}}$ P, then every Non-deterministic alg A in NP can obtain a deterministic Z in P.

it shows if SAT $\xrightarrow{\text{is in}}$ P, then P=N.

Scanned with OKEN Scanner

Q:- Explain P-class, NP, NP-Hard, NP complete, Non-Deterministic and Deterministic problems.

A:- **P-class:-** the computational problems that are solvable is called P-class.

P:- Problems can be solved in Polynomial time.

Polynomial Time: Eg:- work should be complete in 1 hour.
work completed in 1 hour.
Polynomial Time work ~~may~~ complete in given time.

There are 2 types of groups to solve the problems,

1. **NP-Complete** (Deterministic) :- Problems that can be solved in Polynomial (P) time. called (Deterministic)

→ A problem that is NP-complete has the property that it can be solved in polynomial time if and only if all other NP-complete problems can also be solved in Polynomial time.
Ex:- Quick sort, binary search etc.

Scanned with OKEN Scanner

2. NP-Hard :- (Non-deterministic)

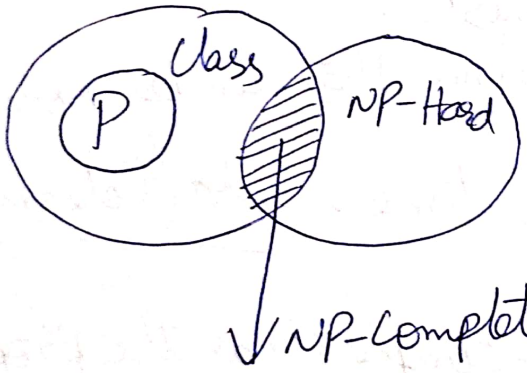
Problems that cannot be solved in Polynomial time.

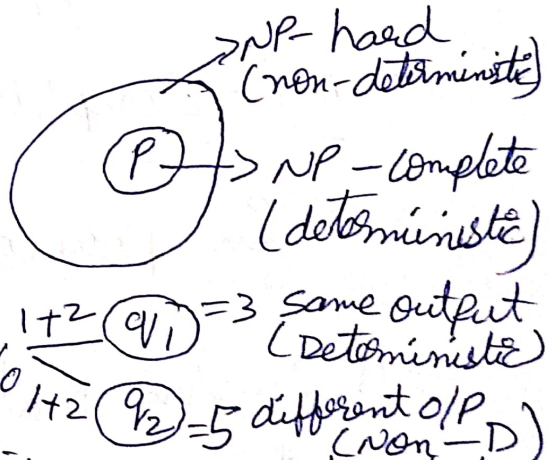
⇒ if an NP-Hard problems can be solved in Polynomial time then all NP-complete problems can be solved in Polynomial time.

⇒ All NP-complete problems are NP-Hard.

NP-complete ⊆ NP-Hard.

But some NP-Hard problems are not NP-complete.





Deterministic :- An algorithm with the property that result of every operation is Uniquely defined.

Non-Deterministic Algorithm :- An algorithm whose result of every operation is not uniquely defined.

In order to specify non-deterministic problems

we consider 3 functions (non-deterministic),

1. choice(s): Chooses one of element in given set S.
2. failure(): returns Un-successful completion.
3. Success(): returns successful completion.

Ex:-

```
j := choice (1, n);
if A[j] = x then
{
  write (j);
  Success();
}
write (0)
Failure();
```

Remaing Answer. ⟹ Problems go to pg (6)

3Q:- Explain about satisfiability problem with Example.

A:- Satisfiability :- SAT

⇒ A Boolean satisfiability function (SAT) is the problem of determining if a Boolean formula is: Satisfiable or Un-satisfiable.

Scanned with OKEN Scanner

if the output is true for a given input values then it is called satisfiability.

→ Let $x_i = x_1, x_2 \cdots$ denote boolean variables.

Let $\overline{x_i}$ denotes negations of x_i.

→ A literal is either a variable or its negation.

→ A formula in the propositional calculus is an expression that can be constructed using literals and operations "$\wedge$" or "$\vee$"

⇒ A formula in conjunctive normal form (CNF) if and only if it is represented as

CNF ⇒ $\bigwedge_{i=1}^{K} = C_i$ → $\bigvee L_{ij}$ → literals

where C_i are clauses each represent as

$\bigvee L_{ij}$ where L_{ij} are literals.

↑ literals

→

⇒ A formula is in Disjunctive normal form (DNF) if and only if it is represented as $\bigvee_{i=1}^{k} = c_i$

where c_i are clauses each represented as $\bigwedge_{L_{ij}}$.

⇒ The satisfiability problem is to determine whether the formula is true for some assignment of truth values to variables.

Ex:- $(x_1 \wedge x_2) \vee (x_3 \wedge \overline{x_4})$ [DNF]

$(x_3 \vee \overline{x_4}) \wedge (x_1 \vee \overline{x_2})$

$x_1 = T$
$x_2 = T$
$x_3 = T$
$x_4 = F$

for DNF,

$(T \wedge T) \vee (T \wedge T)$

$T \vee T = T$ ✓

For CNF,

$(T \vee T) \wedge (T \vee F)$

$T \wedge T = T$ ✓

if CNF satisfies, then it is called satisfiability problem for CNF formulas and vice-versa →

Scanned with OKEN Scanner

Non-deterministic satisfiability algorithm :-

```
Algorithm EVal(E, n)
{
    for i := 1 to n do
    xi = choice(false, true);
    if E(x1, x2 ··· xn) then success();
    else Failure();
}
```

Q:- Classes of NP-Hard and NP-complete :-

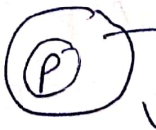
A:-

1. → P is a set of all decision problems solvable by the deterministic algorithms in polynomial time.

→ NP is a set of all decision problems solvable by the non-deterministic algorithms in polynomial time.

$P \subseteq NP$; $P = NP$ (or) $P \neq NP$

Sorting, searching, all pairs shortest path



Travelling salesman problem, graph colouring.

→

Scanned with OKEN Scanner

2. Let L_1, L_2 be problems. if problem L_1 reduces to L_2, i.e., $L_1 \propto L_2$ if and only if there is a way to solve L_1 by deterministic Polynomial time algorithm using a deterministic algorithm that shows L_2 in Polynomial time. i.e., if we have a polynomial time algorithm for L_2 then we can solve L_1 in Polynomial time.

$L_1 \xrightarrow{\propto} L_2$ $\quad L_1 \propto L_2$

$L_2 \xrightarrow{\propto} L_3$ $\quad L_2 \propto L_3$

$\quad\quad\quad\quad\quad\quad\quad L_1 \propto L_3$

$\therefore L_1 \xrightarrow{\propto} L_3$ (According to transitive property).

↳ → A problem L is NP-Hard if and only if satisfiability reduces to L.

A problem L is NP-complete if and only if L is NP-Hard and $L \in NP$. $L \rightarrow$ NP-Hard & $L \in NP$.

→

Scanned with OKEN Scanner

2. Two problems L_1 and L_2 are said to be Polynomially equivalent if and only if L_1 reduces to L_2 and L_2 reduces to L_1 i.e.,

a problem L_2 is NP-Hard since L_1 is some problem already known to be NP-Hard. Since it is using transitive relation it follows that Satisfiability $\propto L_1$

reduces

$L_1 \propto L_2$

then Satisfiability $\propto L_2$.

= END 5th UNIT =

Scanned with OKEN Scanner