Chapter - 6

NP - COMPLETENESS AND APPROXIMATION ALGORITHMS

* PROBLEM CATEGORIES
  - Tractable Problems } decidable problems
  - Intractable Problems }
  - Undecidable Problems.

1) Tractable Problems
   - A problem is called tractable if there is an efficient algorithm that solves the problem in polynomial time.
   - On an input of size n, the worst case running time of such algorithms is $O(n^k)$, for some constant k.
     eg. $O(n)$, $O(n^2)$ ... $O(n \log n)$
   - Example of such problems are sorting, searching, matrix multiplication etc.

2) Intractable Problems
   - A problem is called intractable if algorithm solving the problem takes super polynomial time i.e exponential time.
   - With a moderate increase in input size, these problems grows very large and are unable to solve in reasonable time.
     eg. they have complexities like $O(2^n)$, $O(n^n)$, $O(n!)$
   - Example of such problems are Graph coloring, TSP, vertex cover, clique problems etc.

3) Undecidable Problems
   - These problems do not have algorithms of any known complexity i.e neither polynomial nor super polynomial.
   - Example :) > halting problems etc.

DECISION / OPTIMIZATION PROBLEMS

* TYPES OF DECIDABLE PROBLEMS
  - Decision problems
  - Counting problems
  - Optimization problems.

1) Decision Problems
   - In this class of problems, output is either 'yes' or 'No'.
   - Example: To check whether positive integer 'n' is prime or not?

2) Counting Problems
   - Output of this class of problem is a natural number.
   - Example: For a given positive integer, count the number of distinct factorization.

3) Optimization Problems
   - This class of problem optimizes some objective function based on problem instance.
   - In this, from all feasible solution, find the one which minimizes or maximizes our objective based on problem instance.
   - Example: For a given graph, find shortest simple path from a vertex s to vertex t out of all possibilities.

For every optimization problem, a decision or counting version can be created easily.

For example: Consider the knapsack problem.

Given a Profit vector $P = (P_1, P_2 \ldots P_n)$ and
Weight Vector $W = (w_1, w_2 \ldots w_n)$ and
Knapsack capacity C.

Decision problem: Does there exist a n-tuple $(x_1, x_2 \ldots x_n) \in \{0,1\}^n$ such that $\sum_{i=1}^{n} P_i x_i \geq K$ where K is a profit value and $\sum_{i=1}^{n} x_i w_i \leq C$?

Counting problem : How many n-tuples are there such that $\sum_{i=1}^{n} x_i p_i \geq K$ where K is profit value and $\sum_{i=1}^{n} x_i w_i \leq C$?

Optimization problem : Find an n-tuple $(x_1, x_2 \ldots x_n) \in \{0,1\}^n$ such that $\sum_{i=1}^{n} p_i x_i$ is maximized and $\sum_{i=1}^{n} x_i w_i \leq C$.

" IF ONE CAN ANSWER DECISION PROBLEM (IN POLYNOMIAL TIME) THEN ONE CAN FIND ALGORITHM FOR OPTIMIZATION VERSION (IN POLYNOMIAL TIME)."

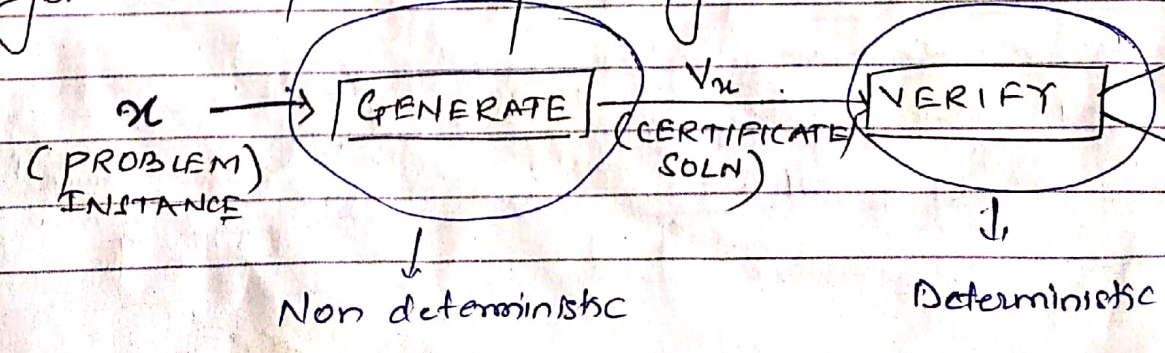
P PROBLEMS & NP PROBLEMS

TWO CLASSES of DECIDABLE PROBLEMS
- P - problems.
- NP - problems.

1] class P (Polynomial Time)
- Class of problems that have polynomial time deterministic algo.
  - i.e solvable in polynomial time.
- Deterministic algorithms always computes correct answer.
- The notion of polynomial time computability is robust i.e is an independent of models use for computation.
- Example: Sorting, searching, matrix multiplication, single-source shortest path, Minimum Spanning tree etc.

2] class NP (Non-Deterministic Polynomial Time)
- class of problems that are solvable in polynomial time with a non-deterministic algorithm.
- A non deterministic algorithm can give different outputs for same input on different execution.
- These algorithms are useful for finding approximate solutions, when exact solution is difficult or expensive.
- Alternative definition: class of problems that are verifiable in polynomial time i.e each 'yes' instance must have atleast one 'certificate' (solution) which can be verified or validated in polynomial time.
- Example:- Satisfiability, Graph coloring, Clique, vertex cover, Hamiltonian cycle etc.

- Non-deterministic algorithm can be thought as generate & test paradigm

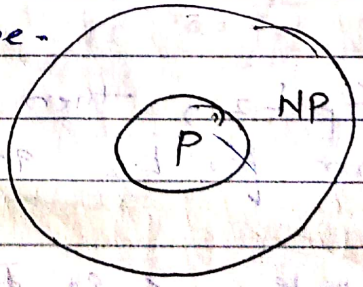


- Verification algorithm has 2 arguments as i/p.
  (i) Problem i/p (ii) Certificate (Solution)
  and output is 'YES' or 'NO'.
- So, Verifier verifies that the given solution exist for the problem or not in polynomial time.

RELATING P AND NP

- P - problems are polynomial time solvable &
  NP - problems are polynomial time verifiable.
- P is definitely subset of NP because every problem with polynomial time solution is also verifiable in polynomial time.

i.e $P \subseteq NP$ i.e



Polynomial Time Reduction

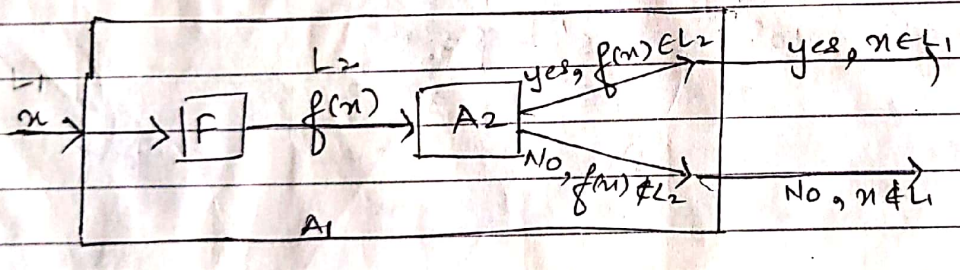
- A polynomial time reduction from a decision problem L_1 to a decision problem L_2 is a procedure that transforms any instance of L_1 into an instance of L_2.

- This transformation or reduction procedure has following characteristics:
  (a) Conversion a reduction procedure should take polynomial time.
  (b) Answer for an instance of L_1 is yes if and only if answer for the corresponding of L_2 is yes.

- Reducibility is polynomial time is represented as
$$L_1 \leq_p L_2$$

- If $L_1 \leq_p L_2$, then one can turn a polynomial time algorithm for L_2 into an algorithm for L_1.

- The relation $\leq$ is transitive, i.e., if $L_1 \leq_p L_2$ and $L_2 \leq_p L_3$ then $L_1 \leq_p L_3$.



- In this diagram, F is the reduction algorithm that transforms instance of L_1 i.e x to instance of L_2 i.e $f(x)$ in polynomial time.
- A_2 is a polynomial time known algorithm for problem L_2.

- If $f(x) \in L_2$ then $x \in L_1$.
- Since, reduction i.e F and Algo for L_2 ie A_2 takes polynomial time, therefore algorithm for L_1 ie A_1 also takes polynomial time.

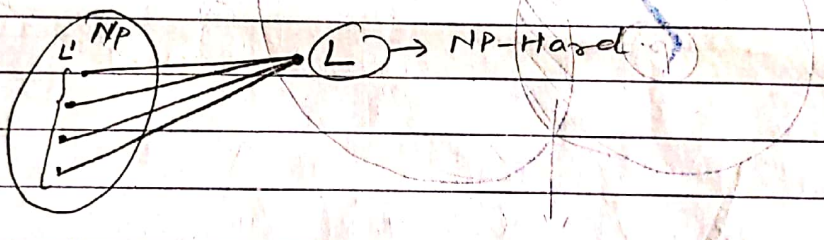
NP-Hard & NP-Complete Problems

1] NP-Hard problems:

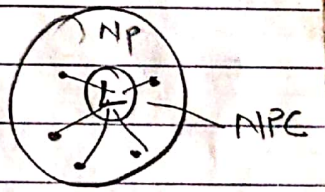
- A decision problem L is NP-hard if every problem in NP is polynomial time reducible to problem L.
- In symbols,

  L is NP-Hard if, for every $L' \in NP$,
  $L' \xrightarrow{\text{Polynomial time}} L$ i.e $L' \leq_p L$

- It doesn't require L to be in NP.
- Informally, it means that L is 'as hard as' all the problems in NP.



2] NP-Complete problems (NPC):



- A decision problem L is NPC if
  (a) it is NP-Hard
  (b) $L \in NP$ i.e L is known NP-problems.
- In symbols,

  L is NPC if, $L \in NP$ and for every $L' \in NP$,
  $L' \leq_p L$

- Informally, it means that L is one of the hardest problems in NP.

- If anyone ever shows that an NP-Complete problem is tractable then
  (a) Every NPC is also tractable.
  (b) Indeed, every problem in NP is tractable.

i.e If $L \in NPC$, then existence of polynomial time algorithm for L implies that there is a polynomial time algorithm for every $L' \in NP$.

- Diagrammatic representation of P, NP, NPC, NP-Hard.