DAA -UNIT V

NP-Hard and NP-Complete

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

Important Questions for exam point of view:

1. (a) Write short notes on
   i) Classes of NP-hard
   ii) Classes of NP-complete
   (b) Prove that if NP ≠ CO − NP, then P ≠ NP

2. Consider the problem DNF-DISSAT which takes a Boolean formula S in
   (a) Disjunctive normal form (DNF) as input and asks if S is dissatisfiable that is variable of S so that if evaluates to 0. Show that DNF – DISSAT is NP- complete.

3. (a) Show that any language in NP can be decided by an algorithm running in time $2^{o(nk)}$ for some constant k.
   (b) How are P and NP problems related?

4. Given an integer m x n matrix A and an integer m-vector b, the 0-1 integer programming problem asks whether there is an integer n-vector x with elements in the set {0,1} such that Ax ≤ b. Prove that 0-1 integer programming is Np-complete.

Basic concepts:

NP→ Nondeterministic Polynomial time

The problems has best algorithms for their solutions have "Computing times", that cluster into two groups

Group 1	Group 2
➢ Problems with solution time bound by a polynomial of a small degree. ➢ It also called "Tractable Algorithms" ➢ Most Searching & Sorting algorithms are polynomial time algorithms ➢ **Ex:** Ordered Search **(O (log n)),** Polynomial evaluation **O(n)** Sorting **O(n.log n)**	➢ Problems with solution times not bound by polynomial (simply non polynomial) ➢ These are hard or intractable problems ➢ None of the problems in this group has been solved by any polynomial time algorithm ➢ **Ex:** Traveling Sales Person **$O(n^2 2^n)$** Knapsack **$O(2^{n/2})$**

No one has been able to develop a polynomial time algorithm for any problem in the 2nd group (i.e., group 2)

So, it is compulsory and finding algorithms whose computing times are greater than polynomial very quickly because such vast amounts of time to execute that even moderate size problems cannot be solved.

Theory of NP-Completeness:

Show that may of the problems with no polynomial time algorithms are computational time algorithms are computationally related.

There are two classes of non-polynomial time problems

1. NP-Hard
2. NP-Complete

NP Complete Problem: A problem that is NP-Complete can solved in polynomial time if and only if (iff) all other NP-Complete problems can also be solved in polynomial time.

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 know to be NP-Complete.

Nondeterministic Algorithms:

Algorithms with the property that the result of every operation is uniquely defined are termed as deterministic algorithms. Such algorithms agree with the way programs are executed on a computer.

Algorithms which contain operations whose outcomes are not uniquely defined but are limited to specified set of possibilities. Such algorithms are called nondeterministic algorithms.

The machine executing such operations is allowed to choose any one of these outcomes subject to a termination condition to be defined later.

To specify nondeterministic algorithms, there are 3 new functions.

Choice(S)→ arbitrarily chooses one of the elements of sets S

Failure ()→ Signals an Unsuccessful completion

Success ()→ Signals a successful completion.

Example for Non Deterministic algorithms:

Algorithm Search(x){
//Problem is to search an element x
//output J, such that A[J]=x; or J=0 if x is not in A
J:=Choice(1,n);
if(A[J]:=x) then {
Write(J);
Success();
}
else{
write(0);
failure();
}
	Whenever there is a set of choices that leads to a successful completion then one such set of choices is always made and the algorithm terminates.
	
	A Nondeterministic algorithm terminates unsuccessfully if and only if (iff) there exists no set of choices leading to a successful signal.

Nondeterministic Knapsack algorithm	

Algorithm DKP(p, w, n, m, r, x){
W:=0;
P:=0;
for i:=1 to n do{
x[i]:=choice(0, 1);
W:=W+x[i]*w[i];
P:=P+x[i]*p[i];
}
if((W>m) or (P<r)) then Failure();
else Success();
}
	p→ given Profits
	w→ given Weights
	n→ Number of elements (number of p or w)
	m→ Weight of bag limit
	P→Final Profit
	W→Final weight

The Classes NP-Hard & NP-Complete:

For measuring the complexity of an algorithm, we use the input length as the parameter. For example, An algorithm A is of polynomial complexity p() such that the computing time of A is O(p(n)) for every input of size n.

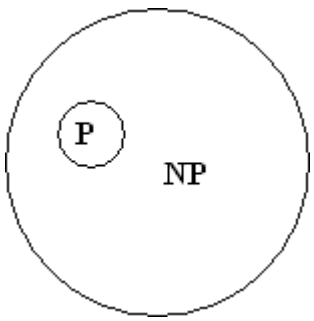
Decision problem/ Decision algorithm: Any problem for which the answer is either zero or one is decision problem. Any algorithm for a decision problem is termed a decision algorithm.

Optimization problem/ Optimization algorithm: Any problem that involves the identification of an optimal (either minimum or maximum) value of a given cost function is known as an optimization problem. An optimization algorithm is used to solve an optimization problem.

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

NP→ is the set of all decision problems solvable by nondeterministic algorithms in polynomial time.

Since deterministic algorithms are just a special case of nondeterministic, by this we concluded that **P ⊆ NP**



Commonly believed relationship between P & NP

The most famous unsolvable problems in Computer Science is Whether P=NP or P≠NP
In considering this problem, s.cook formulated the following question.

If there any single problem in NP, such that if we showed it to be in 'P' then that would imply that P=NP.

Cook answered this question with

Theorem: Satisfiability is in P if and only if (iff) P=NP

→Notation of Reducibility

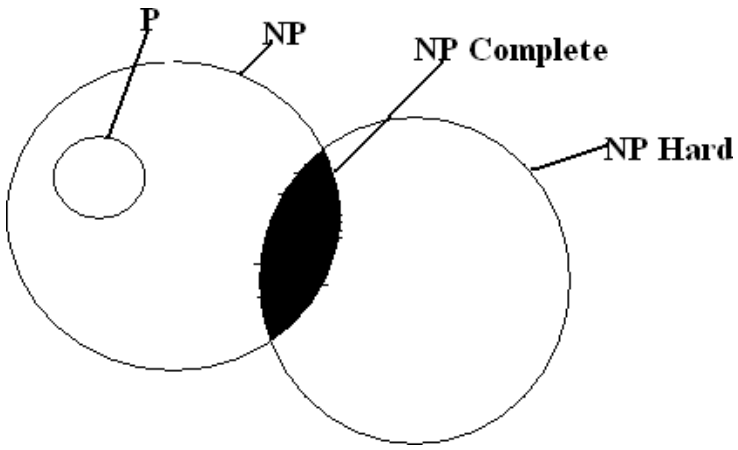
Let L_1 and L_2 be problems, Problem L_1 reduces to L_2 (written $\mathbf{L_1 \alpha L_2}$) iff there is a way to solve L_1 by a deterministic polynomial time algorithm using a deterministic algorithm that solves L_2 in polynomial time

This implies that, if we have a polynomial time algorithm for L_2, Then we can solve L_1 in polynomial time.

Here α→ is a transitive relation i.e., $\mathbf{L_1 \alpha L_2}$ and $\mathbf{L_2 \alpha L_3}$ then $\mathbf{L_1 \alpha L_3}$

A problem L is NP-Hard if and only if (iff) satisfiability reduces to L ie., **Statisfiability α L**

A problem L is NP-Complete if and only if (iff) L is NP-Hard and **L Є NP**



Commonly believed relationship among P, NP, NP-Complete and NP-Hard

Most natural problems in NP are either in P or NP-complete.
Examples of NP-complete problems:

- Packing problems: SET-PACKING, INDEPENDENT-SET.
- Covering problems: SET-COVER, VERTEX-COVER.
- Sequencing problems: HAMILTONIAN-CYCLE, TSP.
- Partitioning problems: 3-COLOR, CLIQUE.
- Constraint satisfaction problems: SAT, 3-SAT.
- Numerical problems: SUBSET-SUM, PARTITION, KNAPSACK.

Cook's Theorem: States that satisfiability is in P if and only if P=NP
If P=NP then satisfiability is in P
If satisfiability is in P, then P=NP
To do this

- A→ Any polynomial time nondeterministic decision algorithm.

  I→Input of that algorithm

  Then formula Q(A, I), Such that Q is satisfiable iff '**A'** has a successful termination with Input **I**.
- If the length of '**I**' is '**n'** and the time complexity of **A** is **p(n)** for some polynomial **p()** then length of Q is $\mathbf{O(p^3(n) \log n)=O(p^4(n))}$

  The time needed to construct Q is also $\mathbf{O(p^3(n) \log n).}$
- A deterministic algorithm '**Z'** to determine the outcome of '**A**' on any input '**I**'

  Algorithm **Z** computes '**Q**' and then uses a deterministic algorithm for the satisfiability problem to determine whether '**Q**' is satisfiable.
- If **O(q(m))** is the time needed to determine whether a formula of length '**m**' is satisfiable then the complexity of '**Z**' is $\mathbf{O(p^3(n) \log n + q(p^3(n) \log n)).}$
- If satisfiability is '**p**', then '**q(m)**' is a polynomial function of '**m**' and the complexity of '**Z**' becomes '**O(r(n))'** for some polynomial '**r()'.**
- Hence, if satisfiability is in **p**, then for every nondeterministic algorithm **A** in **NP**, we can obtain a deterministic **Z** in **p.**

By this we shows that satisfiability is in **p** then **P=NP**

Difference between NP-Hard and NP-Complete:

NP-hard	NP-Complete
NP-Hard problems (say X) can be solved if and only if there is a NP-Complete problem (say Y) that can be reducible into X in polynomial time.	NP-Complete problems can be solved by a non-deterministic Algorithm/Turing Machine in polynomial time.
To solve this problem, it do not have to be in NP .	To solve this problem, it must be both NP and NP-hard problems.
Do not have to be a Decision problem.	It is exclusively a Decision problem.
Example: Halting problem, Vertex cover problem, etc.	Example: Determine whether a graph has a Hamiltonian cycle, Determine whether a Boolean formula is satisfiable or not, Circuit-satisfiability problem