Design and Analysis of Algorithms

Session -33

BASIC CONCEPTS

The computing times of algorithms fall into two groups.

- **Group1**– Consists of problems whose solutions are bounded by the polynomial of small degree. Example – Binary search O(logn) , sorting O(nlogn), matrix multiplication $O(n^{2.81})$
- **Group2** – Contains problems whose best known algorithms are non polynomial.

  Example –Traveling salesperson problem $0(n^2 2^n)$, knapsack

  problem $0(2^{n/2})$ etc.

There are two classes of non polynomial time problems

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.

“All NP-Complete problems are NP-Hard but not all NP Hard problems are not NP-Complete”

Deterministic and Non-Deterministic Algorithms

Algorithms with the property that the result of every operation is uniquely defined are termed deterministic.

- Such algorithms agree with the way programs are executed on a computer.
- When the outcome is not uniquely defined but is limited to a specific set of possibilities, we call it non deterministic algorithm

To specify such algorithms in SPARKS, we introduce three statements

i) choice(S) : arbitrarily chooses one of the elements of the set S.

ii) failure : Signals an unsuccessful completion.

iii) Success : Signals a successful completion.

EXAMPLE OF A DETERMINISTIC ALGORITHM

```
Algorithm Lsearch (A, n, x)  {
   for i:= 1 to n do  {
          if(a[i]=x) then
                 return i;
   }
   return 0;
}
```

EXAMPLE OF A NON DETERMINISTIC ALGORITHM

```
Algorithm Search(A, n, x)  {
   j:= choice(1,n);
   if(a[j]=x) then   {
            return j;
            success();
   }
   return 0;
   failure();
}
```

DEFINITIONS

- **Decision problem**

  Any problem whose answer is yes or no

- **Decision algorithm**

  An algorithm for a decision problem is termed as a decision algorithm

- **Optimization problem**

  Any problem that involves the identification of an optimal (either minimum or maximum) values of a given cost function is known as an optimization problem.

- **Optimization algorithm**

  An optimization algorithm is used to solve an optimization problem.

- **P** is the set of all decision problems solvable by a deterministic algorithm in polynomial time.

Sample Problems in P :

Fractional Knapsack, MST , Sorting.

- **NP** is the set of all decision problems solvable by a nondeterministic algorithm in polynomial time.

Sample Problems in NP :

Fractional Knapsack, MST , Sorting and Hamiltonian Cycle (Traveling Salesman), Graph Coloring

SATISFIABILITY

- Let x1 ,x2 ,x3….,xn 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 the prepositional calculus is an expression that can be constructed using literals and the operators ʌ(AND) and v(OR).

- A formula is in Conjunctive Normal Form (CNF) iff it is represented as $\wedge C_i$, where the C_i are clauses each represented as $\vee l_{ij}$.
- It is in Disjunctive Normal Form (DNF) iff it is represented as $\vee C_i$ and each clause is represented as $\wedge l_{ij}$
- The satisfiability problem is to determine if a formula is true for some assignment of truth values to the variables
- CNF-Satisability is the satisfiability problem for CNF formulas

ALGORITHM FOR SATISFIABILITY

```
Algorithm EVAL(E, n)
{
for i := 1 to n do
    xi := choice(true, false);
if E(x1,...,xn)
        then success();
else    failure();
}
```

Reducibility

Let L_1 and L_2 be problems. L_1 reduces to L_2 ($L_1 \alpha L_2$) if and only if there is a deterministic polynomial time algorithm to solve L_1 that solves L_2 in polynomial time.

- If $L_1 \alpha L_2$ and $L_2 \alpha L_3$ then $L_1 \alpha L_3$.

COOK's Theorem

Satisfiability is in P if and only if P = NP.

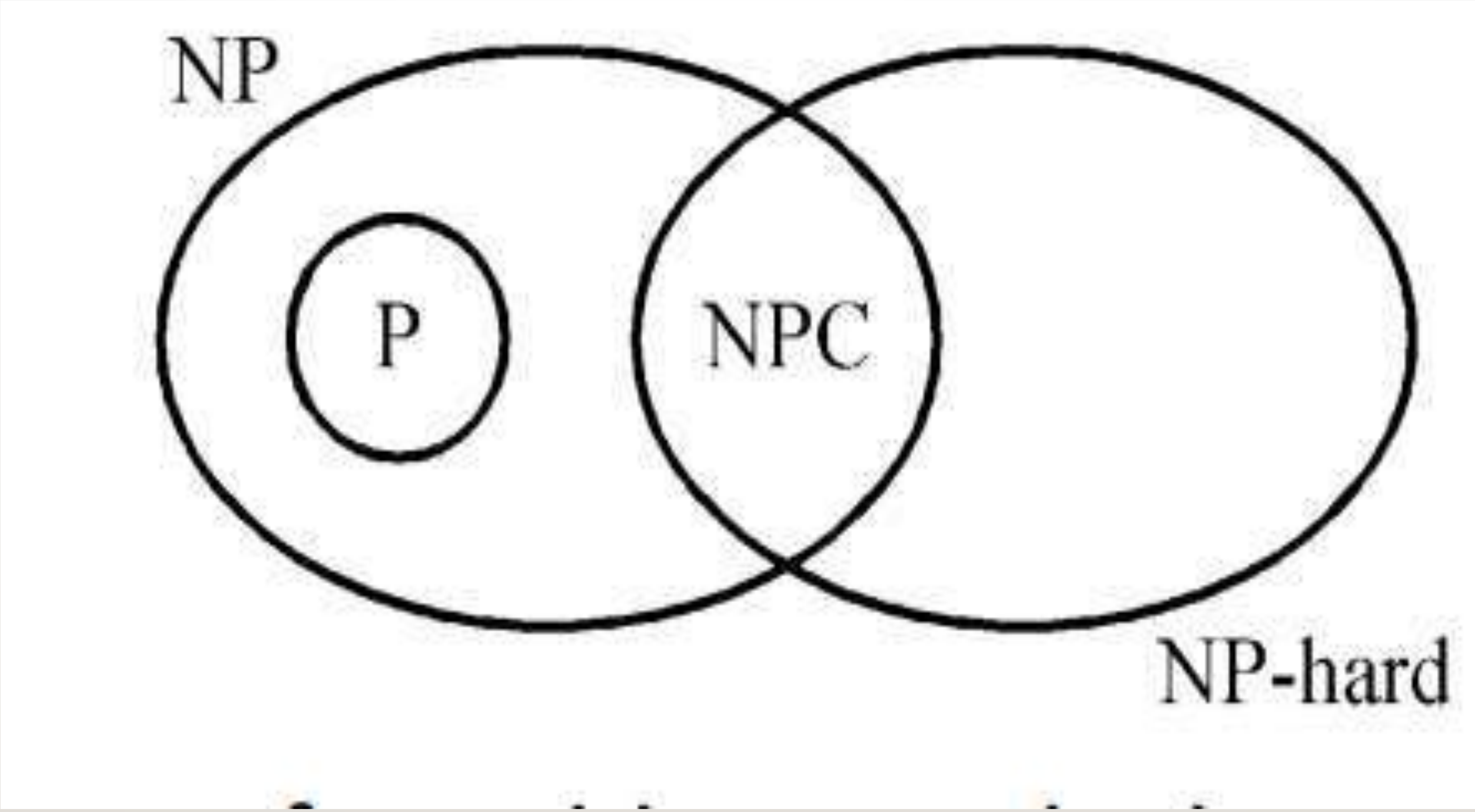
NP-Hard

A problem L is NP-hard if any only if satisfiability reduces to L.

NP-complete

A problem L is NP-complete if and only if L is NP-hard and L ϵ NP.

RELATIONSHIP BETWEEN P, NP, NP-HARD AND NP-COMPLETE



COOK'S THEOREM

- Theorem states that satisfiability is in P if and only if P = NP.
- To prove this, we show how to obtain from any polynomial time nondeterministic decision algorithm A and input I a formula Q(A, I) such that Q is satisfiable if 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 the length of Q is $O(p^3(n) \log n) = O(p^4(n))$.

- The time needed to construct Q is also $O(p^3(n)\log n)$.
- A deterministic algorithm Z to determine the outcome of A on any input I can be easily obtained.
- Algorithm Z simply 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 $O(p^3(n)\log n + q(p^3(n)\log n))$.

- If satisfiability is in P, then q(m) is a polynomial function of m and 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.
- So, the above construction shows that if 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, etc.

Example : Halting problem is NP-hard decision problem, but it is not NP-complete.

Halting problem is NP-hard

To show that Halting problem is NP-hard, we show that

satisfiability α halting problem.

For this let us construct an algorithm A

whose input is a prepositional formula X.

- Suppose X has n variables.
- Algorithm A tries out all 2^n possible truth assignments and verifies if X is satisfiable.

Halting problem is un-decidable.

- Hence there exists no algorithm to solve this problem.
- So, it is not in NP. Therefore, it is not NP-complete.

Questions:

1. With suitable example, explain nondeterministic algorithm.
2. Explain terminology used in Satisfiability Problem.
3. Explain Cook's theorem.
4. Differentiate NP-Hard with NP-Completeness.

THANK YOU