DAA UNIT 6 SOLUTION :-

1. **Prove : P ⊆ NP .**

Ans:-

1. Here are important key points on P subset NP.

1. P and NP are complexity classes in theoretical computer science.
2. P comprises problems solvable by deterministic Turing machines in polynomial time.
3. NP encompasses problems that can be verified in polynomial time by a nondeterministic Turing machine.
4. The statement "P ⊆ NP" implies that every problem in P is also in NP.
5. This means that any problem whose solution can be efficiently verified is also efficiently solvable.
6. The inclusion of P in NP signifies that verification is inherently easier than solving for many problems.
7. The relationship between P and NP is one of the most significant unsolved problems in computer science.
8. The P versus NP problem addresses whether a polynomial-time algorithm can solve every problem in NP.
9. A positive resolution to the P versus NP problem would have profound implications for computational complexity.
10. The P versus NP problem remains a central open question in theoretical computer science, with ongoing research focused on exploring its implications.

Page No.

Date / /

DAA :- UNIT 6 :-

Q.1) Prove : $P \subseteq NP$.

Soln:- 1. A decision problem is in P if there exists an algorithm that can solve it in polynomial time.

2. On other hand, a decision problem is in NP if a solution to it can be verified in polynomial time.

3. This implies that every problem in P can also be verified in polynomial time, as the solution is already known.
$\therefore$ P is included in NP.

4. To formally prove that the complexity class P is a subset of the complexity class NP, we need to demonstrate that every problem in P is also in NP.

Here is step by step explanation is follows :-

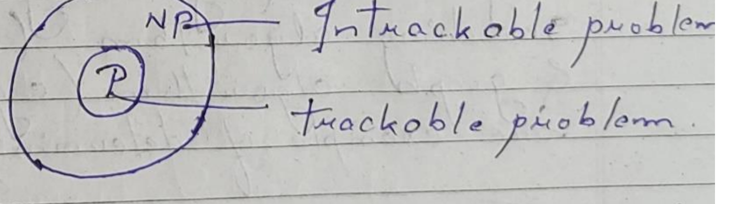
1. Let a decision problem L ~~can~~ be in P. This means that there exists a deterministic Turing machine that can decide L in polynomial time.

2. We need to show that L is also in NP. To do this, we need to provide a polynomial time verification algorithm that

can verify the solution for L. Since L is in P, it means that there exist a polynomial-time algorithm A that can decide L.

3. Given an instance of the decision problem L & a solution (certificate) for L, the verification algorithm A can use the certificate to verify the solution in polynomial time. This means that the verification algorithm can run in polynomial time in the size of the input, as the certificate is of polynomial size.

4. By showing that for every problem in P there exists a polynomial time verification algorithm, we prove that P is a subset of NP.



$\therefore P \subseteq NP$

2. Write non – deterministic algorithm to generate CLIQUE of size k from graph of n vertices.

(4) Write non-deterministic algorithm to generate CLIQUE of size k from graph of n vertices.

soln A clique is a maximal complete subgraph of a given graph G(V,E). The size of the clique is the no. of vertices in it. The clique decision problem is to find out whether graph G contains a clique of size at least x for $x \leq n$.

```
Algorithm NClique (G, n, x)
being
    // Intialize set S to empty set
    S = φ
    // compute set S
    for i = 1 to x
    begin
        y = choice (1, n)
        if (y ∈ S) then Failure
        // otherwise add y to set S
        S = S ∪ {y}
    end for
    // check whether S forms clique
    for each pair (i, j) of vertices in S such that
    i ∈ S, j ∈ S, i ≠ j do
        if edge <i, j> ∉ E
        then failure ();
    end for
    Succes ();
end
```

The time complexity of this algo is $O(n+x^2) = O(n^2)$

3.Explain the relationship between P, NP, NP - complete and NP - hard. OR

6. Write notes on: i) Deterministic Algorithm. ii) P class problem. iii) NP – Hard. iv) NP – complete.

Ans: -Here are important key points on this …

1.Deterministic Algorithm

1. Definition: A deterministic algorithm is a step-by-step procedure that always produces the same output for a given input.
2. Characteristics: Deterministic algorithms follow a fixed sequence of instructions and produce predictable outcomes.
3. Examples: Sorting algorithms like bubble sort and insertion sort are examples of deterministic algorithms.
4. Advantages: Deterministic algorithms provide reliable and consistent results, making them suitable for tasks requiring g precision.
5. Limitations: Some problems, like finding the largest clique in a graph, may not have efficient deterministic algorithms.

2.P-class Problem

1. Definition: A P-class problem is a decision problem that can be solved by a deterministic Turing machine in polynomial time
2. Characteristics: P-class problems are efficiently solvable, meaning the solution time grows at a manageable rate as the problem size increases.
3. Examples: Finding the shortest path between two nodes in a graph is an example of a P-class problem.
4. Significance: P-class problems represent a subset of tractable problems that can be solved efficiently using known algorithms.
5. Relationship to NP: P-class problems are a subset of NP-problems, implying that any problem efficiently solvable by a deterministic machine is also efficiently verifiable by a nondeterministic machine.

3. NP-Hard Problem

1. Definition: An NP-hard problem is an optimization or search problem that is at least as hard as any problem in NP.
2. Characteristics: NP-hard problems are not known to have efficient algorithms, and their solutions can become intractable for large problem instances.
3. Examples: Finding the optimal solution to the traveling salesman problem and determining the satisfiability of a Boolean formula are examples of NP-hard problems.
4. Significance: NP-hard problems represent a class of computationally challenging problems that are believed to not have efficient solutions.
5. Relationship to NP-complete: NP-hard problems are at least as hard as NP-complete problems, but not all NP-hard problems are NP-complete.

4. NP-complete Problem

1. Definition: An NP-complete problem is an optimization or search problem in NP such that any problem in NP can be reduced to it in polynomial time.
2. Characteristics: NP-complete problems are the hardest problems in NP, and their solutions are considered intractable for large problem instances.
3. Examples: The Boolean satisfiability problem and the clique problem are examples of NP-complete problems.
4. Significance: NP-complete problems represent a benchmark for computational hardness, and their efficient solution would imply that all problems in NP can be solved efficiently.
5. Relationship to NP-hard: NP-complete problems are a subset of NP-hard problems, meaning that any NP-complete problem is also NP-hard.

1) Write a Notes on:

(I) Deterministics Algorithm.

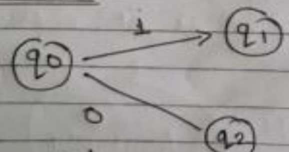
① A deterministic algorithm is a specific type of algorithm that, given a particular input, will always produce the same o/p, with the underlying computations performed in a prescribed order.

② Deterministics algorithms are characterised by their predictability and repeatability, as they follow a well-defined sequence of steps & are not influenced by randomization or external factors.

③ Many classical algorithms, such as sorting algorithms (eg. merge sort, quicksort), searching algorithms (eg. binary search), and graph algorithms (eg. Dijkstra's algorithm), are example of deterministic algorithms.

④ They are widely used in various fields due to their reliability and consistent behavior.

Example



Deterministic algorithm.

⑤ Deterministic algorithm usually have a well defined worst case time complexity

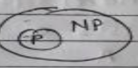
⑥ It can solve the problem in polynomial time

⑦ It usually provides the precise solutn to problem.

(8) It commonly used in applicatn where precision is critical, such as in cryptography, numerical analysis, and computer graphics.

(2) P Class problem

(1) P class problem can be solved in "polynomial time," which means that an algorithm exists for its solution such that the no. of steps in the algorithm is bounded by a polynomial function of n, where n corresponds to the length of the input for the problem.
(2) P class problem is easy to understand & tractable.
(3) P problems are a subset of NP problem



(4) All P problems are deterministic in nature.
(5) It takes polynomial time to solve a problem like n, n^2, $n\log n$ etc.
(6) Examples
(1) selection sort (2) linear search

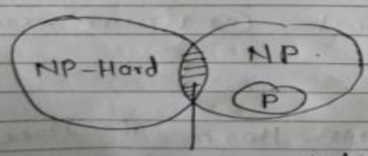
(3) NP-Hard

(1) A problem X is NP-Hard if there is an NP-complete problem Y, such that Y is reducible to X in polynomial time.
(2) NP Hard problem are as hard as NP-complete problem.
(3) NP-Hard problem need not be in NP-class.
(4) If every problem of NP can be polynomial time reduced to it called as NP-Hard.
(5) A lot of times takes the particular problem solve and reducing different problems.

example

(1) Hamiltonian cycle
(2) optimization problem
(3) shortest path
(4) Halting problem
(5) vertex cover problem

(6) Time is unknown in NP-Hard
~~(7) All NP-Hard~~
(7) A problem A is in NP-Hard if, for every problem L in NP, there exists a polynomial-time reduction from L to A.
(8) If a solutn for an NP-Hard problem is given then it takes a long time to check whether it is right or not.



NP complete.

(i) NP- Complete

(1) NP complete problems can be solved by a non-deterministic Algorithm / Turing Machine in polynomial Time.

(2) To solve this problem, it must be both NP & NP-hard problem.

(3) Time is known as it is fixed in NP-Hard.

(4) NP- complete is exclusively a decision problem

(5) All NP- complete problems are NP-hard.

(6) example.

(1) Decision problems

(2) Regular graphs.

(7) If con could solve an NP - complete problem in polynomial time, then one could also solve any NP problem in polynomial time.

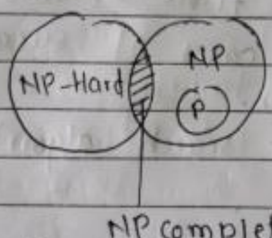


fig: Relatn betn P & NP.

4. Write a note on Polynomial reduction.

Ans:-

1. **Definition:** Polynomial reduction, also known as Turing reduction, is a technique used to compare the computational difficulty of problems.
2. **Concept:** It involves transforming instances of one problem into instances of another problem in polynomial time.
3. **Significance:** If a problem A can be reduced to a problem B in polynomial time, then B is at least as hard as A.
4. **NP-Completeness:** The concept of polynomial reduction is crucial in establishing NP-completeness.
5. **Example:** The Cook-Levin theorem proves NP-completeness by reducing Boolean satisfiability to the circuit value problem in polynomial time.
6. **Implications:** Polynomial reduction allows us to infer the hardness of problems based on the hardness of known problems.
7. **Transitivity:** Polynomial reduction is transitive, meaning if A can be reduced to B and B can be reduced to C in polynomial time, then A can also be reduced to C in polynomial time.
8. **Limitations:** Polynomial reduction cannot determine whether a problem is in P, the class of efficiently solvable problems.
9. **Applications:** Polynomial reduction is widely used in theoretical computer science to classify problems based on their computational complexity.
10. **Role in Algorithm Design:** It guides the development of algorithms by identifying problems that are likely to share computational difficulty.

(4) Polynomial Reductions.

⇒ The class NP - complete (NPC) problems consist of a set of decision problems that no one knows how to solve efficiently. But if there were a polynomial solution for even a single NP- complete problem then every problem in NPC will be solvable in polynomial time. For this we need the concept of reductions.

Suppose there are two problems, A & B ~~you~~ It is impossible to solve problem A in polynomial time. Prove that B cannot be explained in polynomial time
Show that $(A \notin P) \Rightarrow (B \notin P)$

Consider we have two decision problems P_1 and P_2

P_1 → Input (I_1)
P_1 → Algorithm (A1) - unknown

P_2 → Input (I_2)
P_2 → Algorithm (A2) - known

In order to solve the P_1, we ~~know~~ we, ~~we~~ you need an alogrithm A1 that you don't know.

& In order to solve the problem. P2, we need the algorithm A2 that we known. or given.

If P_1 can be solved with the help of A2 then we convert i/p I_1 into I_2 and find solution for P2
Then P_1 is reducible to P2.

I_1 → $\boxed{f(I) = I_2 \xrightarrow{I_2} \boxed{\text{Algorithm A2}}}$ → o/p

(I_1: i/p for problem 1)

(o/p: solution for P)

& here we are not bother about A1.

This is called reduction as P_1 is reduced to B

5. Write algorithm for Non-deterministic sorting.

(3) Write an algorithm for Non-deterministic sorting

Soln: Let P[1:n] is an input array to be sorted in ascending order. The sequence in which choice (1,n) returns indices is used to create output array Q[ ] from P[ ]. Finally if Q[ ] is sorted, then algorithm completes successfully, otherwise terminated unsuccessfully.

```
Algorithm Nsort (P, n)
 Begin
    // Initialize array Q to 0
  for i = 1 to n
    Q[i] = 0
  for i = 1 to n
    begin
    x = choice(1, n)
    if Q[x] = 0
    then
    Q[x] = P[i]
    else
     failure ();
     end if
    end for
  // check whether Q is sorted
  for i = 1 to n-1
  if (Q[i] > Q[i+1])
  Failure ();
  end if
  end for
  success ();
 end
```

The computing time of this algorithm is O(n+n+n) = O(n)

where as the deterministic algorithms have minimum complexity O(n log n).

8. Write the algorithm for non - deterministic searching.

(2) Write the algorithm for non - deterministic searching :

Soln Suppose unsorded input List L[1:n] is an array of size n , n ≥ 1.

We want to search an element x in array L[]. If element x is found its positn should be output , otherwise output is 0.

Shrikrupa

The algorithm is as follows :

```
Algorith NSearch (L, n, x)
  begin
     p = Choice (1, n)  // Returns index betn 1 to n.
  // if x is found at position p, print p,
     else print 0.
  if (L[p] = x)
    then
        print p;
        Success ();
    end if
    print 0;
    Failure ();
  end.
```

The computing time of this algorithm is O(1) in both the cases of successful search & unsuccessful search.

Any deterministic search algorithm requires O(n) time.

9. What are decision and optimization problems?

① What are decision & optimization problem?

Ans: Decision problems and optimization problems are two fundamental types of computational problems.

① Decision problem.

① A decision problem is a computational problem where the answer is a simple "yes" or "no" in response to a specific question.

② Examples of decision problems includes the Boolean satisfiability problem (SAT), the halting problem and the graph connectivity problem.

③ In the context of complexity theory, decision problems are often categorized into various complexity classes, such as P, NP, NP-complete, & NP-hard.

② Optimization problems.

① An optimization problem is a computational problem where the goals is to find the best solution from all feasible solutions.

② Optimization problems often involve maximizing or minimizing an objective function, subject to a set of constraints.

③ example of optimization problem includes the travelling salesman problem (TSP), the knapsack problem, and linear programming problems

④ Optimization problems can be classified based on the nature of the objective function & the constraints such as linear optimization, nonlinear optimization and combinatorial optimization

" While decision problems focus on determining whether a solution exists, optimization problems are concerned with finding the best solution among a set of possible solutions.

Many real-world problems can be formulated as either decision problems or optimization problems, depending on the specific context & requirement of the problem.

10. Illustrate COOK Theorem.

Ans:-

1. The Cook–Levin theorem is a fundamental result in computational complexity theory, proving that the Boolean satisfiability problem (SAT) is NP-complete.

2. SAT is a problem of determining whether a given Boolean formula can be satisfied by assigning truth values to its variables.

3. NP is the class of problems for which a solution can be verified in polynomial time, but no polynomial-time algorithm for finding the solution is known.

4. NP-complete problems are the hardest problems in NP, as any problem in NP can be reduced to an NP-complete problem in polynomial time.

5. The Cook–Levin theorem was independently proved by Stephen Cook and Leonid Levin in 1971 and 1973, respectively.

6. The theorem has had a profound impact on computational complexity theory, leading to many important results and the development of the field.

7. The theorem implies that if there exists a polynomial-time algorithm for solving SAT, then there also exists a polynomial-time algorithm for solving every problem in NP.

8. The theorem has been used to show that many other problems are also NP-complete, strengthening the understanding of computational complexity.

9. The Cook–Levin theorem is a powerful tool for proving that problems are NP-complete, demonstrating its significance in the field.

10. The theorem's simplicity and elegance have had a profound impact on our understanding of computational complexity, highlighting its importance.

7. Explain following NP-problems with respect to graph.

i) CLIQUE ii) Independent set problem iii) Graph partitioned into triangle

Ans:

1. Clique Problem

- Definition: The clique problem involves finding the largest clique in a graph, where a clique is a subset of vertices in which every pair of vertices is connected by an edge.
- Characteristics: The clique problem is a decision problem, asking whether a graph contains a clique of a given size.
- Applications: The clique problem has applications in social network analysis, community detection, and optimization problems.
- Computational Complexity: The clique problem is NP-complete, indicating its computational difficulty and the lack of efficient algorithms for large graphs.
- Approximation Algorithms: Heuristic and approximation algorithms are often used to find approximate solutions to the clique problem due to its computational intractability.

2. Independent Set Problem

- Definition: The independent set problem involves finding the largest independent set in a graph, where an independent set is a subset of vertices in which no two vertices are connected by an edge.
- Characteristics: The independent set problem is an optimization problem, aiming to find the largest independent set in a given graph.
- Applications: The independent set problem has applications in conflict resolution, task scheduling, and resource allocation.
- Computational Complexity: The independent set problem is NP-complete, similar to the clique problem, indicating its computational difficulty.
- Approximation Algorithms: Heuristic and approximation algorithms are also commonly used for the independent set problem due to its computational challenges.

3.Graph Partition into Triangles :-

- Goal: Divide a graph into the minimum number of subgraphs, each consisting of non-overlapping triangles.
- Difficulty: NP-complete, meaning efficient algorithms are unlikely to exist for large graphs.
- Approaches: Heuristic and approximation algorithms are commonly used for practical solutions.
- Applications: Data analysis, network optimization, pattern recognition.
- Significance: Uncovers cohesive subgroups, optimizes network structures, reveals underlying patterns.