ASSIGNMENT → V

1. Write the algorithm and also give an example to explain working of -

i) P NP. Co-NP. NP Hard. NP Complete

a. P (Polynomial Time) Algorithm

STEP 1- Identify if the problem can be solved in polynomial Time.

STEP 2- Construct an algorithm that can solve the problem in $O(n^k)$ Time. where n is the input size and k is a constant.

STEP 3- Implement the algorithm and test it against various inputs to ensure it runs in polynomial Time.

STEP 4- Optimize if necessary to further reduce Time complexity.

Example - Sorting (Merge Sort)

1. Divide the input array into two halves.
2. Recursively sort each half.
3. Merge the sorted halves to produce the sorted output.

Time Complexity : $O(n \log n)$

b. NP (Non-deterministic Polynomial Time) -

STEP 1- Identify a proposed solution or guess.

STEP 2- Check the validity of the solution in polynomial Time.

If a valid solution exists, verify it with a polynomial-Time verification process.

Date : / /

STEP 3 - If no solution is found, repeat with a different guess or approach.

WORKING EXAMPLE - Sudoko Solver

1. Guess a number for an empty cell.
2. Check if the number violates Sudoko rules.
3. If valid, proceed to fill in the next cell.
4. Repeat the process until the puzzle is solved or no valid solution exists.

Time - Complexity - Verifying a solution is $O(n^2)$. But finding the solution might take exponential Time due to trial and error.

3. Co-NP

STEP 1 - Given a decision problem A, convert it to its complement problem $\neg A$.

STEP 2 - Check if $\neg A$ can be verified in polynomial Time.

STEP 3 - If the complement can be verified in polynomial Time, conclude that the original problem is in Co-NP.

Working Example - UNSAT (Un-Satisfiability)

1. Convert the Boolean formula to a CNF formula.
2. Check if the formula is unsatisfiable.

- If no assignment satisfies the formula, then the answer is "yes".
- If an assignment satisfies it, the answer is "no".

Date : __/__/____

4. NP-Hard
STEP 1- Check if the problem is reducible to another NP-Hard or NP-complete problem.
STEP 2- Attempt to find an approximate or heuristic solution since exact solutions may take exponential Time.
STEP 3- For specific instances, use known reductions from NP problems (Eg- TSP, Knapsack)

WORKING EXAMPLE - Travelling Salesman Problem
1 List all possible permutations of cities
2 Compute the total distance for each permutation
3 Return the permutation with the smallest total distance.

Time Complexity - O(n!) which is factorial Time.

5. NP-Complete
STEP 1- Verify if the problem is in NP by checking if a given solution can be verified in polynomial Time.
STEP 2- Reduce an already-known NP-complete problem to this problem to show that it is NP-complete.
STEP 3- If no polynomial Time solution exists, use approximation algorithms or heuristics for practical solutions.

Working Example - 3-SAT
1. Convert the Boolean formula into conjuctive normal form (CNF) with exactly three literals per clause.

Date : / /

2. Guess a truth assignment for the variables
3. Check if the assignment satisfies the formula
4. Return "yes" if a satisfying assignment is found, otherwise, return "no".

2. Time-Complexity-Exponential in the worst case

2. Cook's Theorem

ANSWER. COOK'S Theorem Algorithm

STEP 1-Define the Input problem in NP

- Let the input be a problem L in NP
- Define a non-deterministic Turing machine M that decides L in polynomial Time.

STEP 2- Simulate the Turing Machine

- Represent the computation of M using a Boolean formula
- Encode the input, states and transitions of M as Boolean variables

STEP 3-Representation as Boolean variables

- Variables: Represent the tape contents, head position, and state at each step using Boolean variables - $x_{i,j,k}$, $h_{i,k}$, and $s_{i,q}$

STEP 4- Construct Boolean Constraints

- Initial Configuration
- Transition Constraints
- Accepting Configuration

Step 5- Combine into a Boolean formula
- Combine all constraints into a single Boolean formula ϕ in CNF.

Step 6- Solve the SAT Problem-
- Use a SAT solver to check if ϕ is satisfiable.

Step 7- Conclusion
- If the formula ϕ is satisfiable, the original problem L is solvable.
- Thus, the problem L is reducible to SAT and SAT is NP- complete.

Working Example- Reduction of a simple problem to SAT

Problem: 3-Colorability of a Graph

Reduction Steps

1. Define variables
- Let $x_{v,c}$ be a Boolean variables that is true if vertex v is assigned color c. where $c \in 1,2,3$

2. Construct Constraints -
Vertex Coloring Constraint: Each vertex must have exactly one color

3. Combine into CNF Formula- Combine all vertex and edge constraints into a single CNF formula ϕ.

4. Solve the SAT problem- Use a SAT solver to determine if ϕ is satisfiable.

Date : / /

3. Vertex Cover Algorithm

ANSWER Vertex Cover Algorithm - (Approximation Algorithm)

STEP 1- Initialize the vertex cover $C = \emptyset$

STEP 2- While there are uncovered edges:

- Pick any uncovered edge (u, v).
- Add both u and v to C.
- Mark all edge incident to u or v as covered.

STEP 3- Return C.

Time Complexity : O(m).

Working Example -

Graph $G = (V, E)$

$V = 1, 2, 3, 4$

$E = (1,2), (2,3), (3,4), (4,1)$

Goal : Find a vertex cover of size $k = 2$.

Solution

1. Start with $C = \emptyset$.
2. Pick edge (1,2): add 1, 2 to C.
   Mark edges (1,2), (4,1) and (2,3) as covered.
3. No edges remain uncovered: return
   C = 1, 2

Date: __/__/____

Set Cover Algorithm

SET COVER ALGORITHM -
By Greedy Approximation Algorithm

STEP 1- Initialize an empty cover $C = \emptyset$.

STEP 2- While there are uncovered elements in U:
- Pick the subset S_i from S that covers the largest number of uncovered elements.
- Add S_i to C.
- Mark the elements of S_i as covered

STEP 3- Return C.

Working Example-
Problem: Universal set $U = 1, 2, 3, 4, 5, 6, 7$
Subsets $S = S_1, S_2, S_3, S_4, S_5$
$S_1 = 1, 2, 3$
$S_2 = 2, 4, 5$
$S_3 = 3, 6$
$S_4 = 4, 5, 6, 7$
$S_5 = 7$

Solution:
1. Initialize $C = \emptyset$, uncovered elements $U = 1, 2, 3, 4, 5, 6, 7$.
2. Find the subset that covers the most uncovered elements:
$S_1 = 3$ elements $(\{1, 2, 3\})$ $S_2 = 3$ elements $(\{2, 4, 5\})$
$S_3 = 2$ elements $(\{3, 6\})$ $S_4 = 4$ elements $(\{4, 5, 6, 7\})$
$S_5 = 1$ element $(\{7\})$
Choose S_4.

3. Add S_4 to C: $C = S_4$
Covered elements = {4, 5, 6, 7}
Uncovered elements = {1, 2, 3}

4. Repeat:
Remaining subsets
S_1 = 3 elements ({1, 2, 3})
S_2 = 2 elements ({2, 4, 5})
S_3 = 1 element ({3, 6})
Choose S_1.

5. Add S_1 to C: $C = S_4, S_1$
Covered elements = {1, 2, 3, 4, 5, 6, 7}
Uncovered elements = ∅.

Result: The selected subsets are $C = S_4, S_1$.

Time - Complexity - O(n.m)

5. Satisfiability Problem
ANSWER Satisfiability Problem Algorithm -
By DPLL Algorithm

STEP 1 - Simplify clauses -
- If any clause is satisfied, then skip it.
- If a clause is empty (unsatisfiable), backtrack.

STEP 2 - Unit Propagation -
If a clause contains a single literal, assign the value that satisfies the literal.

Date : / /

STEP 3- Pure Literal Elimination-
If a literal appears with only one polarity, assign it a value that satisfies all clauses containing it.

STEP 4- Branching -
Choose a variable and assign it a truth value. then recursively solve the simplified problem.

STEP 5 - Backtracking -
If a branch leads to a contradiction, backtrack and try the opposite value.

Working Example -
Problem - $(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (\neg x_1 \vee x_3)$
Solution -
Simplify clauses
Initial formula
$(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (\neg x_1 \vee x_3)$

Choose a variable
Assign x_1 = True.

Simplify with $x-1 = \text{\{True\}}$
Substituting x_1 = True
$(x_1 \vee \neg x_2)$ becomes True
$(\neg x_1 \vee x_3)$ becomes x_3
Remaining formula : $(x_2 \vee x_3) \wedge x_3$.

Date : __/__/____

4. Unit propagation - x_3 is a unit clause Assign x_3 = True
Simplifying $(x_2 \vee x_3)$: It becomes True

5. Result:- All clauses are satisfied
Solution: x_1 = True, x_3 = True, and any value for x_2.