Questions

1. Define algorithm

An algorithm is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output. An algorithm is thus a sequence of computational steps that transform the input into the output.

2. How can we make an estimate on the time an algorithm will take for execution?

To calculate the time taken by any algorithm, we normally evaluate the number of steps the algorithm takes to solve the problem, assuming each step takes unit time.
Example, if ever an algorithm requires the addition of two numbers, we will assume this can be done in unit time, irrespective of the value of the numbers.

3. What is Big-O Notation?

Big-O or Upper bound of the algorithm is the pessimistic view of the algorithm. It can be used to indicate worst case analysis of the algorithm and often expressed as a function. It can be viewed as a proof that the given problem can be solved using at most 'n' operations, even in the worst case.
It is denoted by O(n).

4. What is Polynomial time?

An algorithm is said to be polynomial time if the worst case time for solving the problem can be expressed as a polynomial (i.e n^r)
In other words, an algorithm is polynomial-time if there exists a constant r such that the running time on an input of size n is O(n^r).
Polynomial time is the shortest class of running times that is invariant across the vast majority of reasonable, mainstream models of computation.

5. Name the different complexity classes for classification of problems based on the difficulty level of solving the problem?

P, NP, NP-Complete, NP-Hard

6. What types of problems are considered to be Decision problems?

We consider decision problems, whose answer is YES or NO.
E.g., "Does the given network have a flow of value at least k?"
For such problems, we split all instances into two categories:

YES-instances (whose correct answer is YES) and
NO-instances (whose correct answer is NO). We put any ill-formed instances into the NO category.

7. What is class P?

The complexity class P is the set of decision problems that can be solved in polynomial-time. In other words, there exists an polynomial time algorithm to solve it.
E.g., Searching, Sorting, Matrix Multiplication, Root finding methods

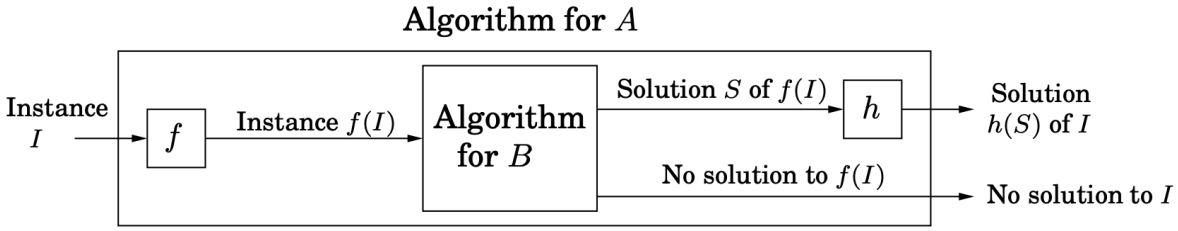
8. What is class NP?

NP stands for "nondeterministic polynomial time. A decision problem X is said to be in NP, if for every "yes" instance there exists a polynomial sized "certificate" which can be "verified" in polynomial time.

A “certificate” for the yes-instance is a proof that the instance is indeed a yes-instance. The size of the certificate is required to be a polynomial.
A “verifier” is nothing but a polynomial time algorithm which takes input the yes-instance and the certificate. If for each yes-instance there exists a certificate, and if the yes-instance can be verified in polynomial time in the size of the input, then the problem is in NP.

Thus NP can also be thought of as the class of problems which cannot be solved in polynomial time but “whose solutions can be verified in polynomial time using a certificate”

9. How can we reduce one problem to another problem?
Reduction algorithm reduces a problem to another problem.
A reduction from a problem A to a problem B is a polynomial-time algorithm f that transforms any instance I of A into an instance f(I) of B, together with another polynomial-time algorithm h that maps any solution S of f(I) back into a solution h(S) of I; see the following diagram. If f(I) has no solution, then neither does I. These two translation procedures f and h imply that any algorithm for B can be converted into an algorithm for A by bracketing it between f and h.
Example: lcm(m, n) = (m * n) / gcd(m, n),
Least common multiple (LCM) of (m,n) problem is reduced to Greatest common divisor (GCD)of (m, n) problem



For example, if we have library functions to solve certain problems and if we can reduce a new problem to one of the solved problems, we save a lot of time.

10. Explain the class NP-Complete.
A problem is NP-complete if the problem is both

- Problem is in NP (i.e. its solution can be verified in polynomial time)
- A lot of times you can solve a problem by reducing it to a different problem. One can reduce Problem B to Problem A if, given a solution to Problem A, one can easily construct a solution to Problem B. (In this case, "easily" means "in polynomial time.“).

In other words, the problem can be reduced to another problem and the solution can be verified

Interesting facts of NP-complete problems

- First, although no efficient algorithm for an NP-complete problem has ever been found, nobody has ever proven that an efficient algorithm for one cannot exist. In other words, no one knows whether or not efficient algorithms exist for NP-complete problems.
- Second, the set of NP-complete problems has the remarkable property that if an efficient algorithm exists for any one of them, then efficient algorithms exist for all of

them. This relationship among the NP-complete problems makes the lack of efficient solutions all the more tantalizing.

- Third, several NP-complete problems are similar, but not identical, to problems for which we do know of efficient algorithms. Computer scientists are intrigued by how a small change to the problem statement can cause a big change to the efficiency of the best known algorithm.

11. Give examples of NP-Complete problems

Example of NP complete

- Hamiltonian cycle
- SAT
- Travelling Salesman Problem (TSP)
- Vertex Cover Problem
- Set Cover Problem

12. What is SAT? Why is it important?

SAT means Circuit Satisfiability.

We say that a circuit is satisfiable if the combination of gates the circuit causes the output of the circuit to be 1.

SAT is of significance because SAT is the first problem to be proved to be NP-Complete. The proofs of all other NP-Complete problems are derived from SAT.

13. What is hamiltonian cycle

A hamiltonian cycle of an undirected graph G =(V, E) is a simple cycle that contains each and every vertex in the graph.

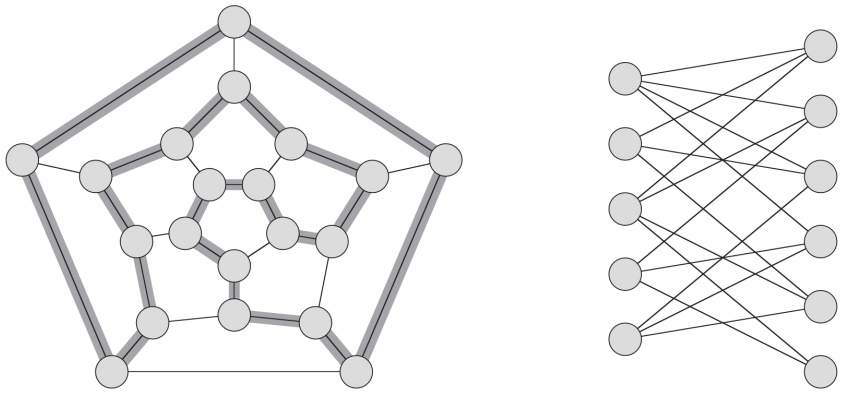


Figure 34.2 **(a)** A graph representing the vertices, edges, and faces of a dodecahedron, with a hamiltonian cycle shown by shaded edges. **(b)** A bipartite graph with an odd number of vertices. Any such graph is nonhamiltonian.

14. What is the Hamiltonian problem? How can we solve it?

We can define the hamiltonian-cycle problem, "Does a graph G have a hamiltonian cycle?"

Given a problem instance {G}, one possible decision algorithm lists all permutations of the vertices of G and then checks each permutation to see if it is a hamiltonian path.

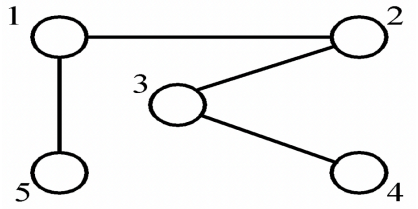
What is the running time of this algorithm? If we use the "reasonable" encoding of a graph as its adjacency matrix, the number m of vertices in the graph is $\Omega\left(\sqrt{n}\right)$, where n is the

length of the encoding of G. There are m! possible permutations of the vertices, and therefore the running time is $\Omega(2^{\sqrt{n}})$, which is not $O(n^k)$ for any constant k.
Thus, this naive algorithm does not run in polynomial time.
The problem statement "Does a graph G have a hamiltonian cycle?" cannot be solved in polynomial time but we can change the problem statement to "Verify that a particular cycle in a graph is hamiltonian ?"

15. What is Traveling-salesman problem? Prove that it is NP-Complete?
In the *traveling-salesman problem(TSP)*, which is closely related to the hamiltonian-cycle problem, a salesman must visit n cities. Modeling the problem as a complete graph with n vertices, we can say that the salesman wishes to make a *tour*, or hamiltonian cycle, visiting each city exactly once and finishing at the city he starts from. The salesman incurs a nonnegative integer cost c(i,j) to travel from city i to city j , and the salesman wishes to make the tour whose total cost is minimum, where the total cost is the sum of the individual costs along the edges of the tour.

Proof that Traveling-salesman problem is NP-Complete
We first show that TSP belongs to NP. Given an instance of the problem, we use as a certificate the sequence of n vertices in the tour. The verification algorithm checks that this sequence contains each vertex exactly once, sums up the edge costs, and checks whether the sum is at most k. This process can certainly be done in polynomial time.
The problem can be reduced to Hamiltonian cycle. Hence, we can say that Traveling-salesman problem is NP-complete, as Hamiltonian cycle is NP-Complete

16. What is Vertex Cover problem? Prove that it is NP-Complete?
Vertex cover is an important problem. It is formally stated as follows: Given a graph G=(V, E), S is the node cover if S ⊆ V and for every edge (u, v) ∈ E, either u ∈ S or v ∈ S or both.
The ***size*** of a vertex cover is the number of vertices in it.
Example : Consider the graph shown in Figure.
The vertex cover finds minimal vertices that cover the entire graph. Some of the possible solutions node covers of this problem are {1, 3} and {5, 2, 4}.



Proof that Vertex cover problem is NP-Complete
Vertex cover is a NP complete problem as the solution involves guessing a subset of vertices, count them, and show that each edge is covered. This is simple if the number of vertices is smaller. But if the number of vertices becomes large, the number of possibilities becomes larger. Therefore, the algorithm becomes exponential algorithm.

The Vertex Cover problem can be reduced to Hamiltonian problem and we can also verify the solution if we have a solution of the problem.

17. What is Set Cover problem? Prove that it is NP-Complete?
The set cover decision problem is to determine if F has a cover T containing no more than c sets.

Example : Consider the following sets :

$F = \{(a_1, a_3), (a_2, a_4), (a_2, a_3), (a_4), (a_1, a_3, a_4)\}$
$\quad\quad s_1 \quad\quad s_2 \quad\quad s_3 \quad\quad s_4 \quad\quad s_5$

Find set cover of the above problem?

Like vertex cover, set cover finds the minimum number of sets that covers all the elements. Some of the possible solutions are

$T = \{s_1, s_3, s_4\}$ and $T = \{s_1, s_2\}$.

Proof that Vertex cover problem is NP-Complete:
Set cover is a NP complete problem as the solution involves guessing a subset of vertices, count them, and show that universal set is well covered. This is simple if the number of elements is smaller. But if the number of elements becomes large, the number of possible set covers becomes larger. Therefore, the algorithm for set cover becomes exponential algorithm.
The Set Cover problem can be reduced to Hamiltonian problem and we can also verify the solution if we have a solution of the problem.
Therefore, set cover problem is NP complete.

18. What does NP-hard mean?
A lot of times you can solve a problem by reducing it to a different problem.
One can reduce Problem B to Problem A if, given a solution to Problem A, one can easily construct a solution to Problem B. (In this case, "easily" means "in polynomial time.“).
A problem is NP-hard if all problems in NP are polynomial time reducible to it.
In other word, the problem can be reduced to a different problem but we have no know way to verify if the solution is correct

Example of NP Hard

- Clustering
- Proof of P = NP

19. What is the Difference between NP hard and NP complete problem

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: Clustering, etc.	Example: Determine whether a graph has a Hamiltonian cycle, , Circuit-satisfiability problem, etc.

Questions

1. How are P and NP problems related?
2.Compare NP-hard and NP-completeness?
3.Explain the class of P and NP with example?
4. Differentiate between NP- complete and NP-hard problems?
5. Explain the satisifiability problem
6. Explain Reduction
7. Mention any two decision problems which are NP-Complete.
8. Prove that Vertex Cover problem is NP-Complete