Design and Analysis of Algorithms (CSC-314)

Department of B.Sc. CSIT

Godawari College

Unit-8: NP Completeness

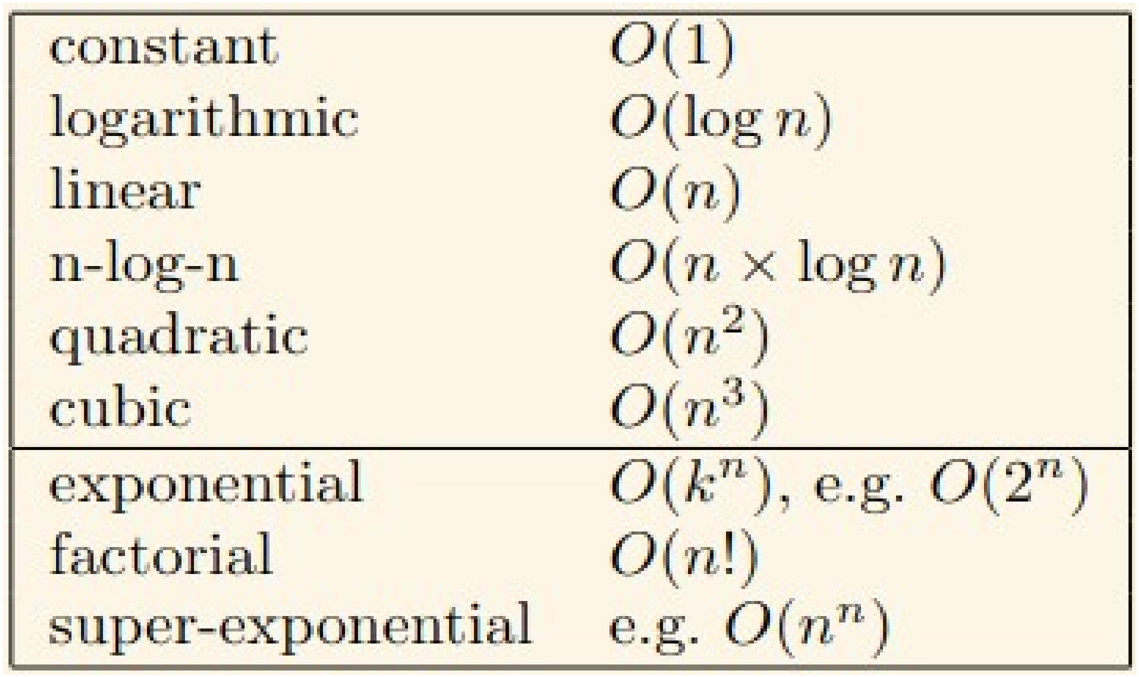
- **Complexity Theory:**
- Computational complexity theory focuses on classifying computational problems according to their resource usage, and relating these classes to each other.
- A computational problem is a task solved by a computer.
- A computation problem is solvable by mechanical application of mathematical steps, such as an algorithm.
- We can say, Computational complexity theory is a subfield of theoretical computer science one of whose primary goals is to classify and compare the practical difficulty of solving computational problems.
  - e.g. given two natural numbers n and m, are they relatively prime?

Unit-8: NP Completeness

- **Complexity Theory:**
- Closely related fields in theoretical computer science are analysis of algorithms and computability theory.
- A key distinction between analysis of algorithms and computational complexity theory is that
  - the former is devoted to analyzing the amount of resources needed by a particular algorithm to solve a problem,
  - whereas the latter asks a more general question about all possible algorithms that could be used to solve the same problem.
  - More precisely, computational complexity theory tries to classify problems that can or cannot be solved with appropriately restricted resources.

Unit-8: NP Completeness

- **Complexity Theory:**
- Let's start by reminding ourselves of some common functions, ordered by how fast they grow.



constant	$O(1)$
logarithmic	$O(\log n)$
linear	$O(n)$
n-log-n	$O(n \times \log n)$
quadratic	$O(n^2)$
cubic	$O(n^3)$
exponential	$O(k^n)$, e.g. $O(2^n)$
factorial	$O(n!)$
super-exponential	e.g. $O(n^n)$

- Computer Scientists divide these functions into two classes:
  - Polynomial functions
  - Exponential functions

Unit-8: NP Completeness

- **Polynomial functions / Polynomial :**
- Any function that is O(nk), i.e. bounded from above by n^k for some constant k.
- E.g. O(1), O(log n), O(n), O(n × log n), $O(n^2)$, $O(n^3)$
- we defined ‘polynomial’ to be any function of the form
  - $a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n^{11} + a_0$.
- But here the word ‘polynomial’ is used to merge together functions that are bounded from above by polynomials.
- So, log n and n × log n, which are not polynomials in our original sense, are polynomials by our alternative definition, because they are bounded from above by, e.g., n and n^2 respectively..

Unit-8: NP Completeness

- **Polynomial functions / Polynomial :**
- An algorithm is said to be solvable in polynomial time if the number of steps required to complete the algorithm for a given input is $O(n^k)$ for some non-negative integer k, where n is the complexity of the input.
- Polynomial-time algorithms are said to be "fast."
- Most familiar mathematical operations such as addition, subtraction, multiplication, and division, as well as computing square roots, powers, and logarithms, can be performed in polynomial time.
- Computing the digits of most interesting mathematical constants, including pi and e, can also be done in polynomial time.

Unit-8: NP Completeness

- **Exponential functions/ Super Polynomial :**
- The remaining functions
  - E.g. $O(2^n)$, $O(n!)$, $O(n^n)$
- This is a real abuse of terminology.
- A function of the form k^n is genuinely exponential.
- But now some functions which are worse than polynomial but not quite exponential, such as $O(n^{\log n})$, are also (incorrectly) called exponential.
- On the other hands, some functions which are worse than exponential, such as the super exponential, e.g. $O(n^n)$, will also (incorrectly) be called exponential.
- A better word than ‘exponential’ would be ‘super-polynomial’. But ‘exponential’ is what everyone uses, so it’s what we’ll use.

Unit-8: NP Completeness

- **Tractable Problem:**
- A problem that is solvable by a polynomial-time algorithm.
- The upper bound is polynomial.
- Examples: Searching of ordered and unordered list, sorting list, MST, multiplication of integer etc.

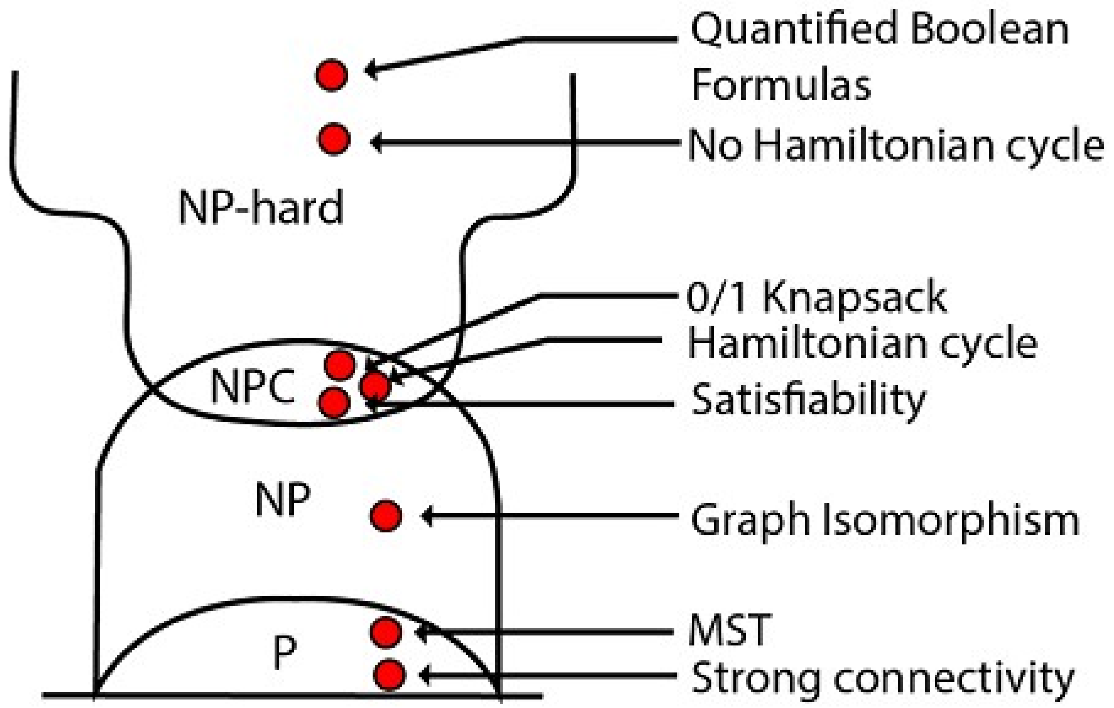
- **Intractable Problem:**
- A problem that cannot be solved by a polynomial-time algorithm.
- The lower bound is exponential.
- Examples: Tower of Hanoi, list of all permutations of n numbers etc.

Unit-8: NP Completeness

- **Complexity class:**
- A complexity class is the set of all of the computational problems which can be solved using a certain amount of a certain computational resource.
  - **Class P problem**
  - **Class NP problem**
  - **Class NP Complete problem**
  - **Class NP Hard problem**

Unit-8: NP Completeness

- **Complexity class:**
-



Unit-8: NP Completeness

- **The complexity class P:**
- P is the complexity class containing decision problems which can be solved by a deterministic Turing machine using a polynomial amount of computation time, or polynomial time.
- P is often taken to be the class of computational problems which are "efficiently solvable" or "tractable“.
- Problems that are solvable in theory, but cannot be solved in practice, are called intractable.
- There exist problems in P which are intractable in practical terms; for example, some require at least $n^{1000000}$ operations.
- P is known to contain many natural problems, including the decision versions of linear programming, calculating the greatest common divisor, and finding a maximum matching.

Unit-8: NP Completeness

- **The complexity class NP:**
- In computational complexity theory, NP ("Nondeterministic Polynomial time") is the set of decision problems solvable in polynomial time on a nondeterministic Turing machine.
- It is the set of problems that can be "verified" by a deterministic Turing machine in polynomial time.
- All the problems in this class have the property that their solutions can be checked effectively.
- This class contains many problems that people would like to be able to solve effectively, including
  - the Hamiltonian path problem (special case of TSP)
  - the Vertex cover problem

Unit-8: NP Completeness

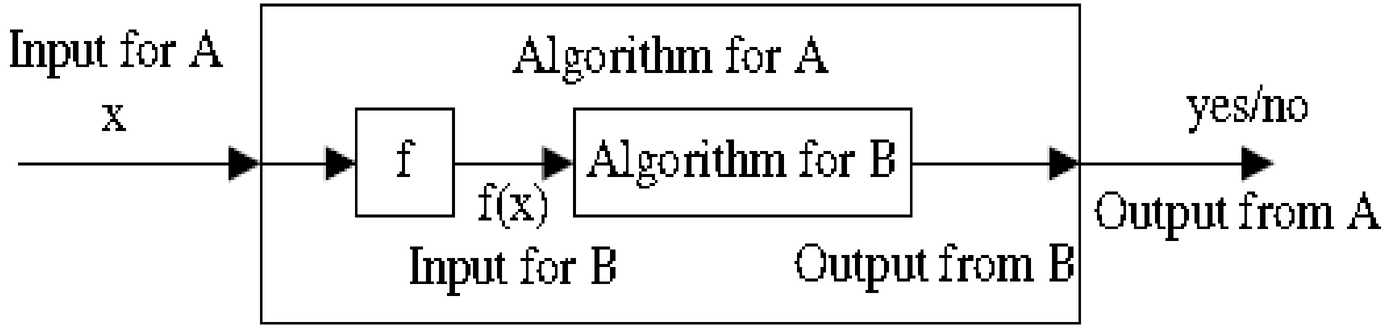
- **The complexity class NP Complete:**
- In complexity theory, the NP complete problems are the most difficult problems in NP ("non deterministic polynomial time") in the sense that they are the ones most likely not to be in P.
- If one could find a way to solve any NP complete problem quickly (in polynomial time), then they could use that algorithm to solve all NP problems quickly.
- At present, all known algorithms for NP complete problems require time that is super-polynomial in the input size.
- To solve an NP complete problem for any nontrivial problem size, generally one of the following approaches is used:
  - Approximation
  - Probabilistic
  - Npuzzle
  - Knapsack problem
  - Hamiltonian cycle problem
  - Traveling salesman problem
  - Subgraph isomorphism problem
  - Subset sum problem

Unit-8: NP Completeness

- **The complexity class NP Hard:**
- NP-Hard problems (say X) can be solved if and only if there is a NP-Complete problem c(say Y) that can be reducible into X in polynomial time.
- NP-Hard Problem need not be in NP class
- NP-hard therefore means "at least as hard as any NP-problem," although it might, in fact, be harder.
- Example: Halting problem, Vertex cover problem, etc.

Unit-8: NP Completeness

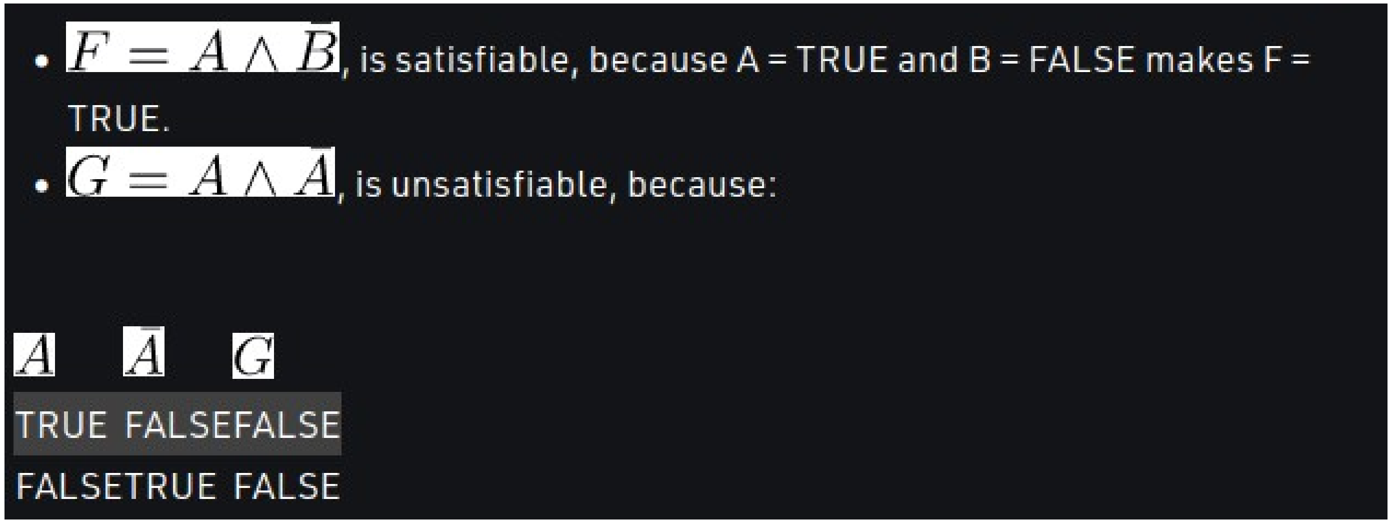
- **Polynomial time reduction**
- Given two problems A and B, a polynomial time reduction from A to B is a polynomial time function f that transforms the instances of A into instances of B.
- such that the output of algorithm for the problem A on input instance x must be same as the output of the algorithm for the problem B on input instance f(x) as shown in the figure below.
- If there is polynomial time computable function f such that it is possible to reduce A to B, then it is denoted as A ≤ p B.
- The function f described above is called reduction function and the algorithm for computing f is called reduction algorithm.



Unit-8: NP Completeness

Boolean Satisfiability Problem

- Boolean Satisfiability or simply SAT is the problem of determining if a Boolean formula is satisfiable or unsatisfiable.
  - **Satisfiable** :
    - If the Boolean variables can be assigned values such that the formula turns out to be TRUE, then we say that the formula is satisfiable.
  - **Unsatisfiable** :
    - If it is not possible to assign such values, then we say that the formula is unsatisfiable.



- $F = A \wedge B$, is satisfiable, because A = TRUE and B = FALSE makes F = TRUE.
- $G = A \wedge \bar{A}$, is unsatisfiable, because:

A	$\bar{A}$	G
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE

Unit-8: NP Completeness

Cook–Levin theorem or Cook’s theorem

- In computational complexity theory, the Cook–Levin theorem, also known as Cook’s theorem, states that the Boolean satisfiability problem is NP-complete.
- That is, it is in NP, and any problem in NP can be reduced in polynomial time by a deterministic Turing machine to the Boolean satisfiability problem.

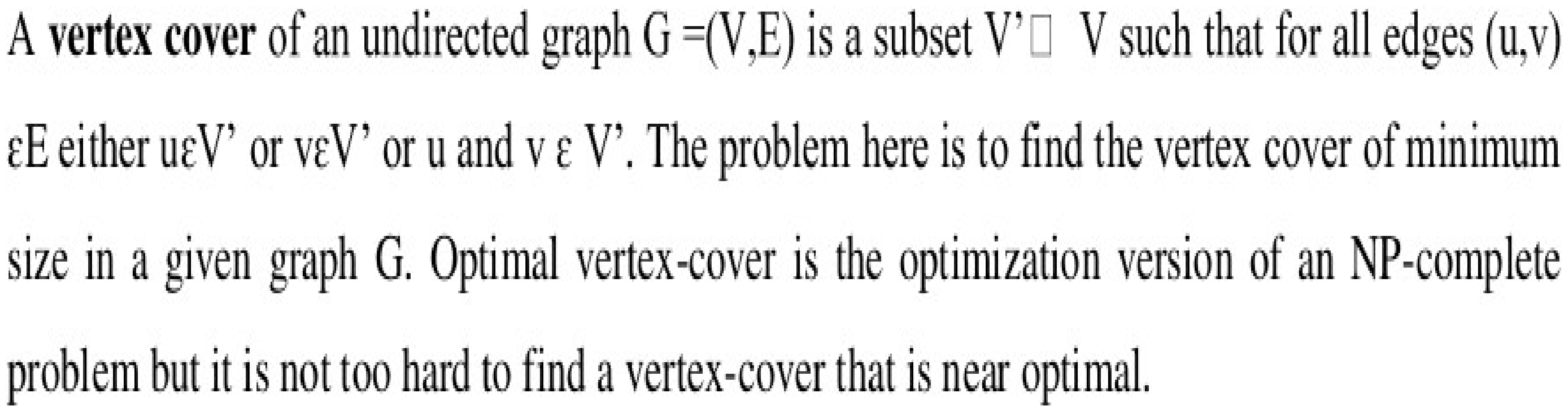
Unit-8: NP Completeness

Approximation Algorithms

- An approximate algorithm is a way of dealing with NP-completeness for optimization problem.
- This technique does not guarantee the best solution.
- The goal of an approximation algorithm is to come as close as possible to the optimum value in a reasonable amount of time which is at most polynomial time.
- If we are dealing with optimization problem (maximization or minimization) with feasible solution having positive cost then it is worthy to look at approximate algorithm for near optimal solution.

Unit-8: NP Completeness

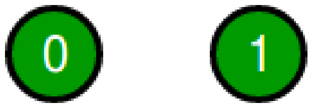
Vertex Cover Problem



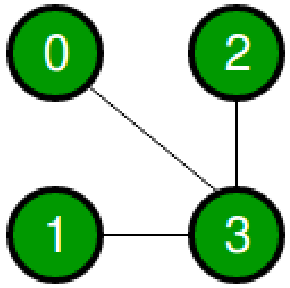
A **vertex cover** of an undirected graph G =(V,E) is a subset V’□ V such that for all edges (u,v) εE either uεV’ or vεV’ or u and v ε V’. The problem here is to find the vertex cover of minimum size in a given graph G. Optimal vertex-cover is the optimization version of an NP-complete problem but it is not too hard to find a vertex-cover that is near optimal.

Unit-8: NP Completeness

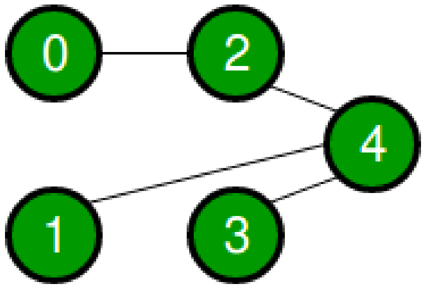
Vertex Cover Problem



Minimum vertex cover is empty{}



Minimum vertex cover is {3}



Minimum vertex cover is {4, 2} or {4, 0}

Unit-8: NP Completeness

Vertex Cover Problem: Pseudo code

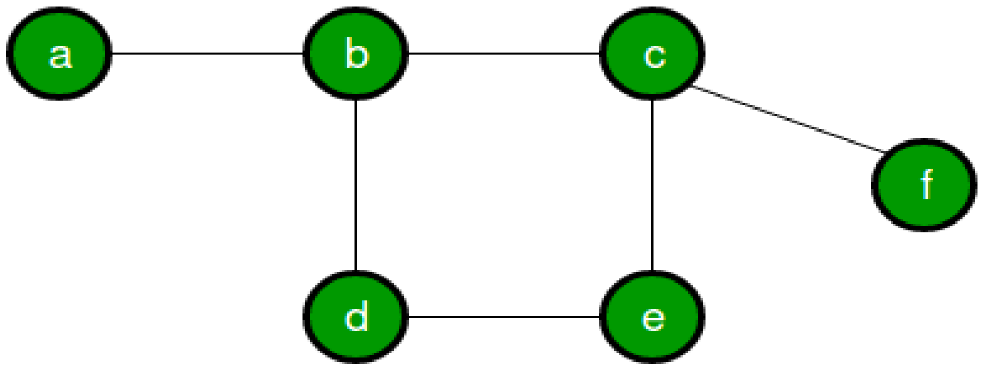
```
ApproxVertexCover (G)
{
    C ={ } ;
    E’ = E
    while E` is not empty
        do Let (u, v) be an arbitrary edge of E`
        C = C U {u, v}
        Remove from E` every edge incident on either u or v
    return C
}
```

- Since E is represented using adjacency list the above algorithms takes O(V+E), since each edges and vertex are processed only once.

Unit-8: NP Completeness

Vertex Cover Problem: Find the minimum vertex covered by the given graph using vertex cover problem:

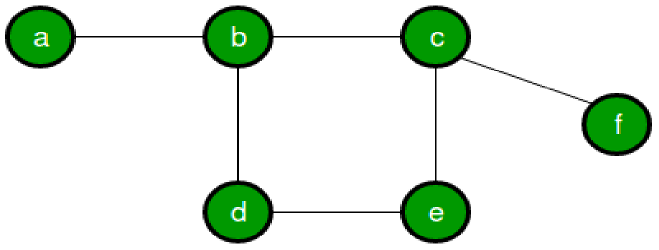
-



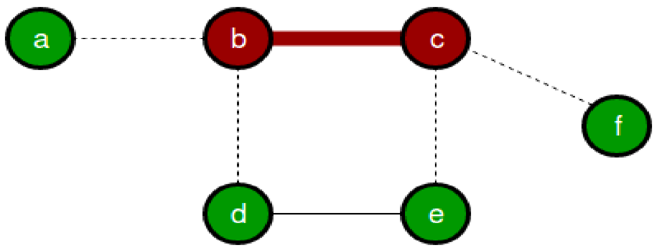
Unit-8: NP Completeness

Vertex Cover Problem: (vertex cover running example for graph below)

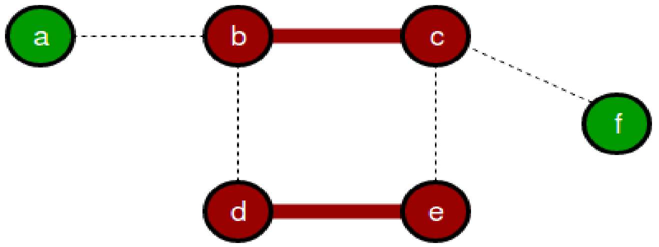
-



(a)



(b)



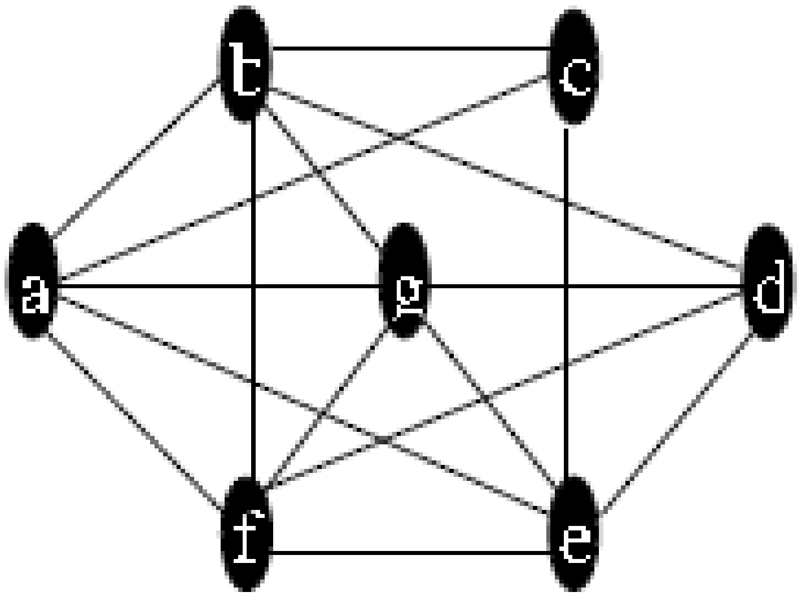
(c)

Minimum Vertex Cover is {b, c, d} or {b, c, e}

Unit-8: NP Completeness

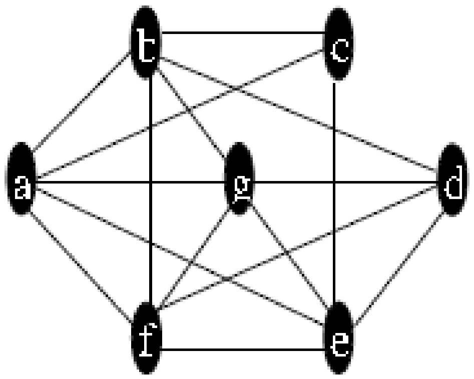
Vertex Cover Problem: Find the minimum vertex covered by the given graph using vertex cover problem.

-

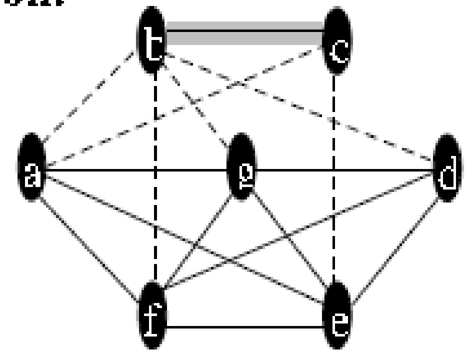


Unit-8: NP Completeness

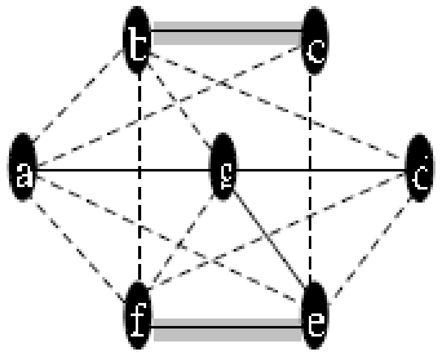
Vertex Cover Problem:Find the minimum vertex covered by the given graph using vertex cover problem.



Solution:



Edge chosen is (b,c) C = {b,c}

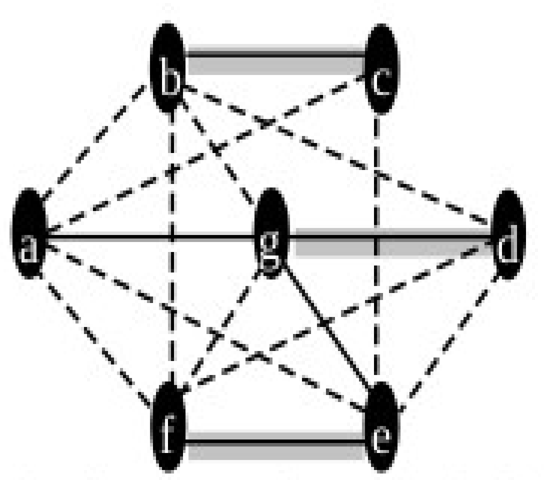


Edge chosen is (f,e) C = {b,c,e,f}

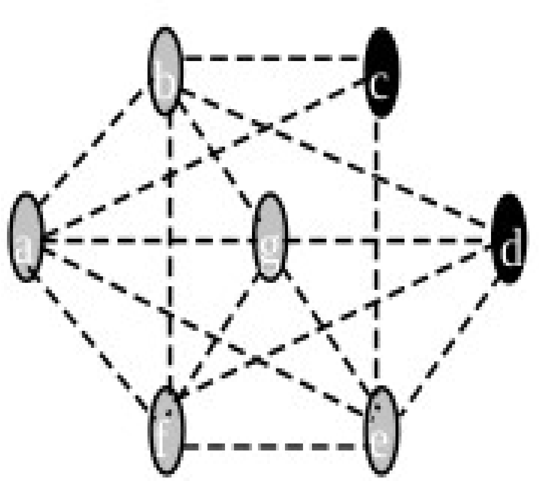
Unit-8: NP Completeness

Vertex Cover Problem:Find the minimum vertex covered by the given graph using vertex cover problem.

-



Edge chosen is (g,d) C = {b,c,d,e,f,g}

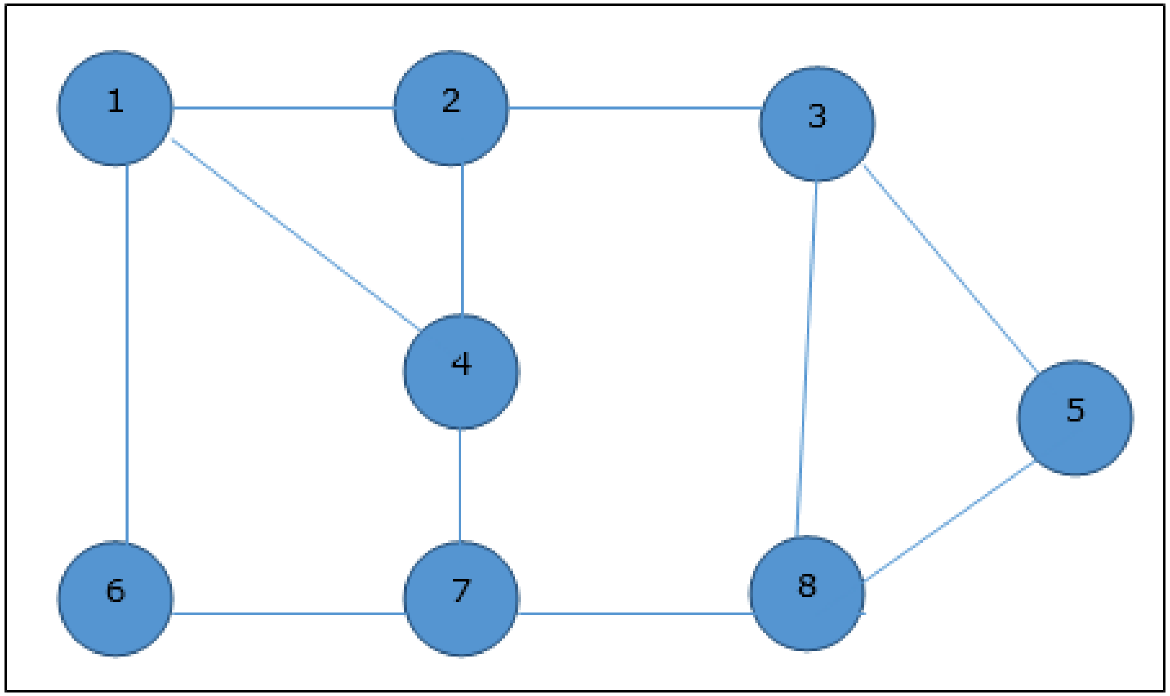


Optimal vertex cover as lightly shaded vertices

Unit-8: NP Completeness

Find the minimum vertex covered by the given graph using vertex cover problem.

-



Unit-8: NP Completeness

Thank You!