Complexity Classes

Definition of NP class Problem: - The set of all decision-based problems came into the division of NP Problems who can't be solved or produced an output within polynomial time but verified in the **polynomial time**. NP class contains P class as a subset. NP problems being hard to solve.

> ***Note: - The term "NP" does not mean "not polynomial." Originally, the term meant "non-deterministic polynomial. It means according to the one input number of output will be produced.***

Definition of P class Problem: - The set of decision-based problems come into the division of P Problems who can be solved or produced an output within polynomial time. P problems being easy to solve

Definition of Polynomial time: - If we produce an output according to the given input within a specific amount of time such as within a minute, hours. This is known as Polynomial time.

Definition of Non-Polynomial time: - If we produce an output according to the given input but there are no time constraints is known as Non-Polynomial time. But yes output will produce but time is not fixed yet.

Definition of Decision Based Problem: - A problem is called a decision problem if its output is a simple "yes" or "no" (or you may need this of this as true/false, 0/1, accept/reject.) We will phrase many optimization problems as decision problems. For example, Greedy method, D.P., given a graph G= (V, E) if there exists any Hamiltonian cycle.

Definition of NP-hard class: - Here you to satisfy the following points to come into the division of NP-hard

1. If we can solve this problem in polynomial time, then we can solve all NP problems in polynomial time
2. If you convert the issue into one form to another form within the polynomial time

Definition of NP-complete class: - A problem is in NP-complete, if

1. It is in NP
2. It is NP-hard

Pictorial representation of all NP classes which includes NP, NP-hard, and NP-complete

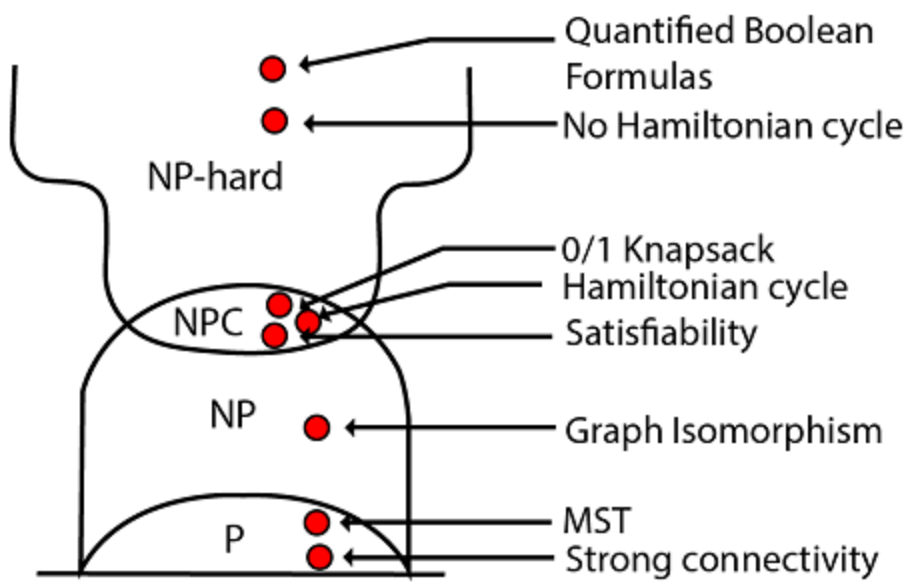


Fig: Complexity Classes

Polynomial Time Verification

Before talking about the class of NP-complete problems, it is essential to introduce the notion of a verification algorithm.

Many problems are hard to solve, but they have the property that it easy to authenticate the solution if one is provided.

Hamiltonian cycle problem:-

Consider the Hamiltonian cycle problem. Given an undirected graph G, does G have a cycle that visits each vertex exactly once? There is no known polynomial time algorithm for this dispute.

> ***Note: - It means you can't build a Hamiltonian cycle in a graph with a polynomial time even if there is no specific path is given for the Hamiltonian cycle with the particular vertex, yet you can't verify the Hamiltonian cycle within the polynomial time***

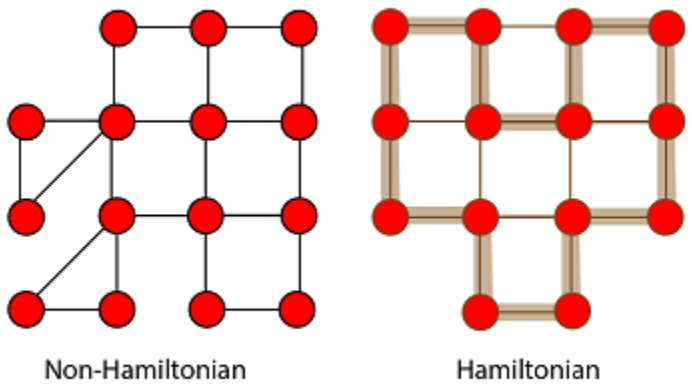


Fig: Hamiltonian Cycle

Let us understand that a graph did have a Hamiltonian cycle. It would be easy for someone to convince of this. They would similarly say: "the period is hv3, v7, v1....v13i.

We could then inspect the graph and check that this is indeed a legal cycle and that it visits all of the vertices of the graph exactly once. Thus, even though we know of no efficient way to solve the Hamiltonian cycle problem, there is a beneficial way to verify that a given cycle is indeed a Hamiltonian cycle.

> ***Note:-For the verification in the Polynomial-time of an undirected Hamiltonian cycle graph G. There must be exact/specific/definite path must be given of Hamiltonian cycle then you can verify in the polynomial time.***

Definition of Certificate: - A piece of information which contains in the given path of a vertex is known as certificate

Relation of P and NP classes

1. P contains in NP
2. P=NP

1. Observe that P contains in NP. In other words, if we can solve a problem in polynomial time, we can indeed verify the solution in polynomial time. More formally, we do not need to see a certificate (there is no need to specify the vertex/intermediate of the specific path) to solve the problem; we can explain it in polynomial time anyway.
2. However, it is not known whether P = NP. It seems you can verify and produce an output of the set of decision-based problems in NP classes in a polynomial time which is impossible because according to the definition of NP classes you can verify the solution within the polynomial time. So this relation can never be held.

Reductions:

The class NP-complete (NPC) problems consist of a set of decision problems (a subset of class NP) 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 and B. You know that it is impossible to solve problem A in polynomial time. You want to prove that B cannot be explained in polynomial time. We want to show that ($A \notin P$) => ($B \notin P$)

Consider an example to illustrate reduction: The following problem is well-known to be NPC:

3-color: Given a graph G, can each of its vertices be labeled with one of 3 different colors such that two adjacent vertices do not have the same label (color).

Coloring arises in various partitioning issues where there is a constraint that two objects cannot be assigned to the same set of partitions. The phrase "coloring" comes from the original application which was in map drawing. Two countries that contribute a common border should be colored with different colors.

It is well known that planar graphs can be colored (maps) with four colors. There exists a polynomial time algorithm for this. But deciding whether this can be done with 3 colors is hard, and there is no polynomial time algorithm for it.

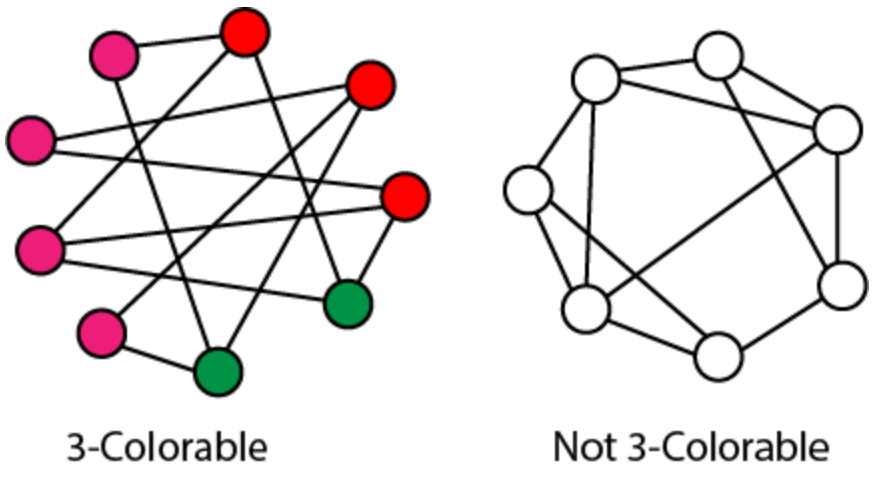


Fig: Example of 3-colorable and non-3-colorable graphs.

Polynomial Time Reduction:

We say that Decision Problem L_1 is Polynomial time Reducible to decision Problem L_2 ($L_1 \leq_p L_2$) if there is a polynomial time computation function f such that of all x, $x \epsilon L_1$ if and only if $x \epsilon L_2$.

NP-Completeness

A decision problem L is NP-Hard if

L' ≤p L for all L' ∈ NP.

Definition: L is NP-complete if

1. L ∈ NP and
2. L' ≤ p L for some known NP-complete problem L.' Given this formal definition, the complexity classes are:

P: is the set of decision problems that are solvable in polynomial time.

NP: is the set of decision problems that can be verified in polynomial time.

NP-Hard: L is NP-hard if for all L' ∈ NP, L' ≤p L. Thus if we can solve L in polynomial time, we can solve all NP problems in polynomial time.

NP-Complete L is NP-complete if

1. L ∈ NP and
2. L is NP-hard

If any NP-complete problem is solvable in polynomial time, then every NP-Complete problem is also solvable in polynomial time. Conversely, if we can prove that any NP-Complete problem cannot be solved in polynomial time, every NP-Complete problem cannot be solvable in polynomial time.

Reductions

Concept: - If the solution of NPC problem does not exist then the conversion from one NPC problem to another NPC problem within the polynomial time. For this, you need the concept of reduction. If a solution of the one NPC problem exists within the polynomial time, then the rest of the problem can also give the solution in polynomial time (but it's hard to believe). For this, you need the concept of reduction.

Example: - Suppose there are two problems, **A** and **B**. You know that it is impossible to solve problem **A** in polynomial time. You want to prove that B cannot be solved in polynomial time. So you can convert the problem **A** into problem **B** in polynomial time.

Example of NP-Complete problem

NP problem: - Suppose a DECISION-BASED problem is provided in which a set of inputs/high inputs you can get high output.

Criteria to come either in NP-hard or NP-complete.

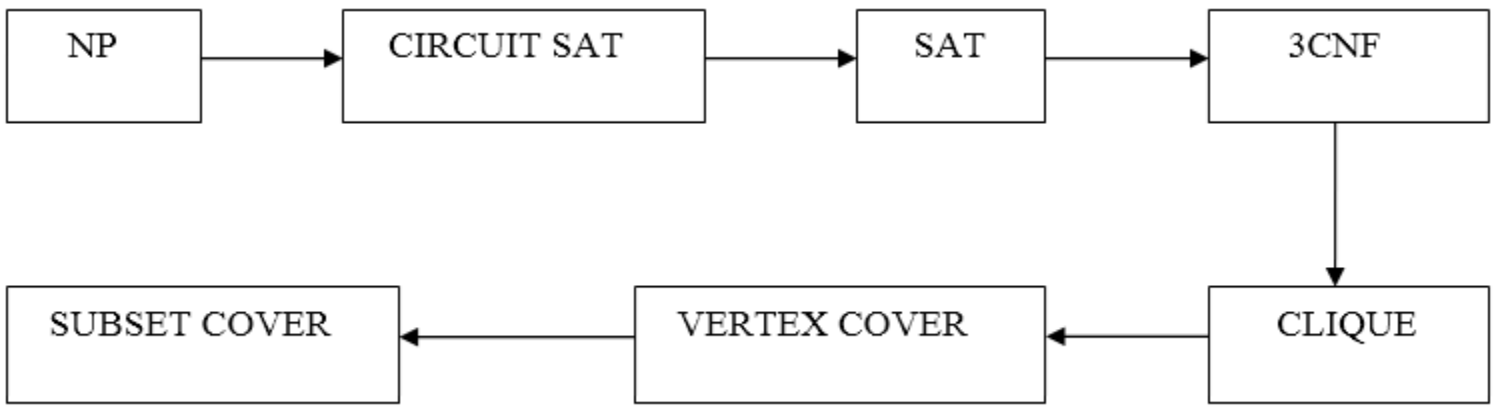
1. The point to be noted here, the output is already given, and you can verify the output/solution within the polynomial time but can't produce an output/solution in polynomial time.

2. Here we need the concept of reduction because when you can't produce an output of the problem according to the given input then in case you have to use an emphasis on the concept of reduction in which you can convert one problem into another problem.

Note1:- If you satisfy both points then your problem comes into the category of NP-complete class
Note2:- If you satisfy the only 2nd points then your problem comes into the category of NP-hard class

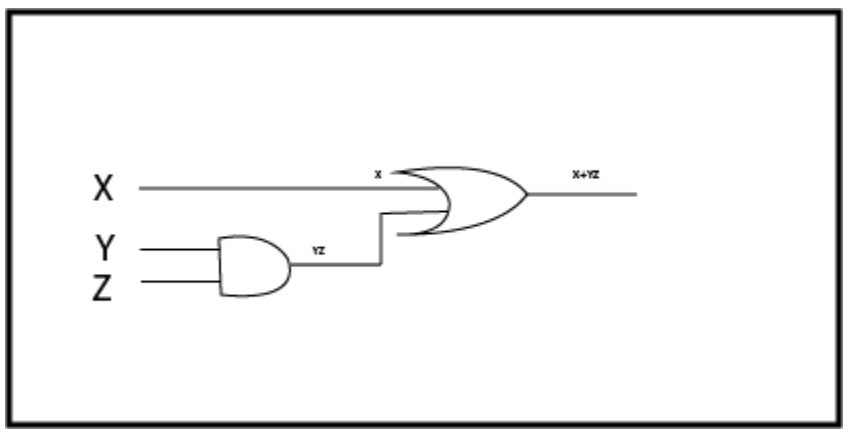
So according to the given decision-based NP problem, you can decide in the form of yes or no. If, yes then you have to do verify and convert into another problem via reduction concept. If you are being performed, both then decision-based NP problems are in NP compete.

Here we will emphasize NPC.



CIRCUIT SAT

According to given decision-based NP problem, you can design the CIRCUIT and verify a given mentioned output also within the P time. The CIRCUIT is provided below:-



Note:- You can design a circuit and verified the mentioned output within Polynomial time but remember you can never predict the number of gates which produces the high output against the set of inputs/high inputs within a polynomial time. So you verified the production and conversion had been done within polynomial time. So you are in NPC.

SAT (Satisfiability):-

A Boolean function is said to be SAT if the output for the given value of the input is true/high/1

F=X+YZ (Created a Boolean function by CIRCUIT SAT)

These points you have to be performed for NPC

1. CONCEPTS OF SAT
2. CIRCUIT SAT≤ρ SAT
3. SAT≤ρ CIRCUIT SAT
4. SAT ϵ NPC

1. **CONCEPT:** - A Boolean function is said to be SAT if the output for the given value of the input is true/high/1.
2. **CIRCUIT SAT≤ρ SAT:** - In this conversion, you have to convert CIRCUIT SAT into SAT within the polynomial time as we did it
3. **SAT≤ρ CIRCUIT SAT:** - For the sake of verification of an output you have to convert SAT into CIRCUIT SAT within the polynomial time, and through the CIRCUIT SAT you can get the verification of an output successfully
4. **SAT ϵ NPC:** - As you know very well, you can get the SAT through CIRCUIT SAT that comes from NP.

Proof of NPC: - Reduction has been successfully made within the polynomial time from CIRCUIT SAT TO SAT. Output has also been verified within the polynomial time as you did in the above conversation.

So concluded that SAT ϵ NPC.

3CNF SAT

Concept: - In 3CNF SAT, you have at least 3 clauses, and in clauses, you will have almost 3 literals or constants

Such as (X+Y+Z) (X+Y+Z) (X+Y+Z)
You can define as (XvYvZ) ^ (XvYvZ) ^ (XvYvZ)
V=OR
^ =AND operator

These all the following points need to be considered in 3CNF SAT.

To prove: -

1. Concept of 3CNF SAT
2. SAT≤ρ 3CNF SAT
3. 3CNF≤ρ SAT
4. 3CNF ϵ NPC

1. **CONCEPT:** - In 3CNF SAT, you have at least 3 clauses, and in clauses, you will have almost 3 literals or constants.
2. **SAT ≤ρ 3CNF SAT:**- In which firstly you need to convert a Boolean function created in SAT into 3CNF either in POS or SOP form within the polynomial time

   F=X+YZ
   = (X+Y) (X+Z)
   = (X+Y+ZZ') (X+YY'+Z)
   = (X+Y+Z) (X+Y+Z') (X+Y+Z) (X+Y'+Z)
   = (X+Y+Z) (X+Y+Z') (X+Y'+Z)
3. **3CNF ≤p SAT:** - From the Boolean Function having three literals we can reduce the whole function into a shorter one.

   F= (X+Y+Z) (X+Y+Z') (X+Y'+Z)
   = (X+Y+Z) (X+Y+Z') (X+Y+Z) (X+Y'+Z)
   = (X+Y+ZZ') (X+YY'+Z)
   = (X+Y) (X+Z)
   = X+YZ
4. **3CNF ϵ NPC:** - As you know very well, you can get the 3CNF through SAT and SAT through CIRCUIT SAT that comes from NP.

Proof of NPC:-

1. It shows that you can easily convert a Boolean function of SAT into 3CNF SAT and satisfied the concept of 3CNF SAT also within polynomial time through Reduction concept.
2. If you want to verify the output in 3CNF SAT then perform the Reduction and convert into SAT and CIRCUIT also to check the output

If you can achieve these two points that means 3CNF SAT also in NPC

Clique

To Prove: - Clique is an NPC or not?

For this you have to satisfy the following below-mentioned points: -

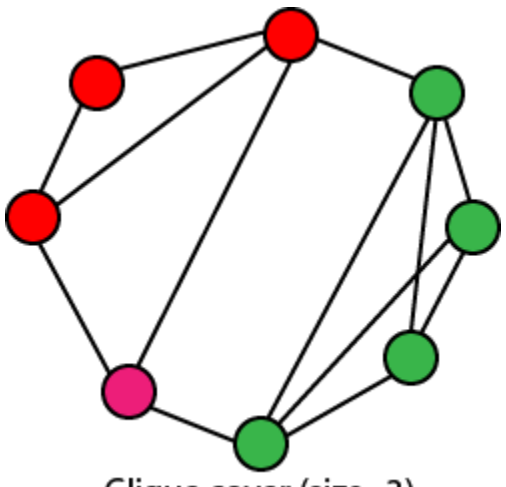
1. Clique
2. 3CNF ≤ρ Clique
3. Clique ≤ρ 3CNF≤SAT
4. Clique ϵ NP

1) Clique

Definition: - In Clique, every vertex is directly connected to another vertex, and the number of vertices in the Clique represents the Size of Clique.

CLIQUE COVER: - Given a graph G and an integer k, can we find k subsets of verticesV_1, V_2...V_K, such that UiVi = V, and that each Vi is a clique of G.

The following figure shows a graph that has a clique cover of size 3.



Clique cover (size=3)

2)3CNF ≤ρ Clique

Proof:-For the successful conversion from 3CNF to Clique, you have to follow the two steps:-

Draw the clause in the form of vertices, and each vertex represents the literals of the clauses.

1. They do not complement each other
2. They don't belong to the same clause
   In the conversion, the size of the Clique and size of 3CNF must be the same, and you successfully converted 3CNF into Clique within the polynomial time

Clique ≤ρ 3CNF

Proof: - As you know that a function of K clause, there must exist a Clique of size k. It means that P variables which are from the different clauses can assign the same value (say it is 1). By using these values of all the variables of the CLIQUES, you can make the value of each clause in the function is equal to 1

Example: - You have a Boolean function in 3CNF:-

(X+Y+Z) (X+Y+Z') (X+Y'+Z)

After Reduction/Conversion from 3CNF to CLIQUE, you will get P variables such as: - x +y=1, x +z=1 and x=1

Put the value of P variables in equation (i)

(1+1+0)(1+0+0)(1+0+1)

(1)(1)(1)=1 output verified

4) Clique ϵ NP:-

Proof: - As you know very well, you can get the Clique through 3CNF and to convert the decision-based NP problem into 3CNF you have to first convert into SAT and SAT comes from NP.

So, concluded that CLIQUE belongs to NP.

Proof of NPC:-

1. Reduction achieved within the polynomial time from 3CNF to Clique
2. And verified the output after Reduction from Clique To 3CNF above
   So, concluded that, if both Reduction and verification can be done within the polynomial time that means **Clique also in NPC**.

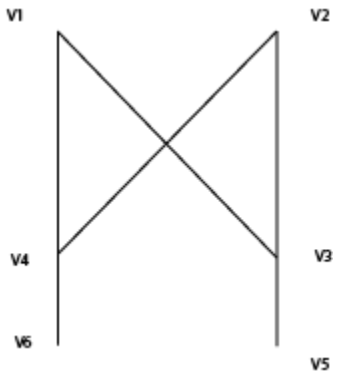
Vertex Cover

1. Vertex Cover Definition
2. Vertex Cover ≤ρ Clique
3. Clique ≤ρ Vertex Cover
4. Vertex Cover ϵ NP

1) Vertex Cover:

Definition: - It represents a set of vertex or node in a graph G (V, E), which gives the connectivity of a complete graph

According to the graph G of vertex cover which you have created, **the size of Vertex Cover =2**



2) Vertex Cover ≤ρ Clique

In a graph G of Vertex Cover, you have N vertices which contain a Vertex Cover K. There must exist of Clique Size of size N-K in its complement.

According to the graph G, you have Number of vertices=6
Size of Clique=N-K=4

You can also create the Clique by complimenting the graph G of Vertex Cover means in simpler form connect the vertices in Vertex Cover graph G through edges where edges don?t exist and remove all the existed edges

You will get the graph G with Clique Size=4

3) Clique ≤ρ Vertex Cover

Here through the Reduction process, you can get the Vertex Cover form Clique by just complimenting the Clique graph G within the polynomial time.

4) Vertex Cover ϵ NP

As you know very well, you can get the Vertex Cover through Clique and to convert the decision-based NP problem into Clique firstly you have to convert into 3CNF and 3CNF into SAT and SAT into CIRCUIT SAT that comes from NP.

Proof of NPC:-

1. Reduction from Clique to Vertex Cover has been made within the polynomial time. In the simpler form, you can convert into Vertex Cover from Clique within the polynomial time
2. And verification has also been done when you convert Vertex Cover to Clique and Clique to 3CNF and satisfy/verified the output within a polynomial time also, so it concluded that Reduction and Verification had been done in the polynomial time that means **Vertex Cover also comes in NPC**

Subset Cover

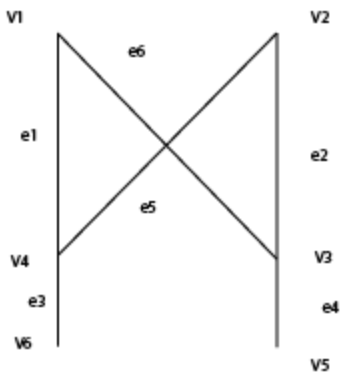
To Prove:-

1. Subset Cover
2. Vertex Cover ≤ρ Subset Cover
3. Subset Cover≤ρ Vertex Cover
4. Subset Cover ϵ NP

1) Subset Cover

Definition: - Number of a subset of edges after making the union for a get all the edges of the complete graph G, and that is called Subset Cover.

According to the graph G, which you have created the size of Subset Cover=2



1. v1{e1,e6} v2{e5,e2} v3{e2,e4,e6} v4{e1,e3,e5} v5{e4} v6{e3}
2. v3Uv4= {e1, e2, e3, e4, e5, e6} complete set of edges after the union of vertices.

2) Vertex Cover ≤ρ Subset Cover

In a graph G of vertices N, if there exists a Vertex Cover of size k, then there must also exist a Subset Cover of size k even. If you can achieve after the Reduction from Vertex Cover to Subset Cover within a polynomial time, which means you did right.

3) Subset Cover ≤ρ Vertex Cover

Just for verification of the output perform the Reduction and create Clique and via an equation, N-K verifies the Clique also and through Clique you can quickly generate 3CNF and after solving the Boolean function of 3CNF in the polynomial time. You will get output. It means the output has been verified.

4) Subset Cover ϵ NP:-

Proof: - As you know very well, you can get the Subset-Cover through Vertex Cover and Vertex Cover through Clique and to convert the decision-based NP problem into Clique firstly you have to convert into3CNF and 3CNF into SAT and SAT into CIRCUIT SAT that comes from NP.

Proof of NPC:-

The Reduction has been successfully made within the polynomial time form Vertex Cover to Subset Cover

Output has also been verified within the polynomial time as you did in the above conversation so, concluded that **SUBSET COVER also comes in NPC**.

Independent Set:

An independent set of a graph G = (V, E) is a subset V'⊆V of vertices such that every edge in E is incident on at most one vertex in V.' The independent-set problem is to find a largest-size independent set in G. It is not hard to find small independent sets, e.g., a small independent set is an individual node, but it is hard to find large independent sets.

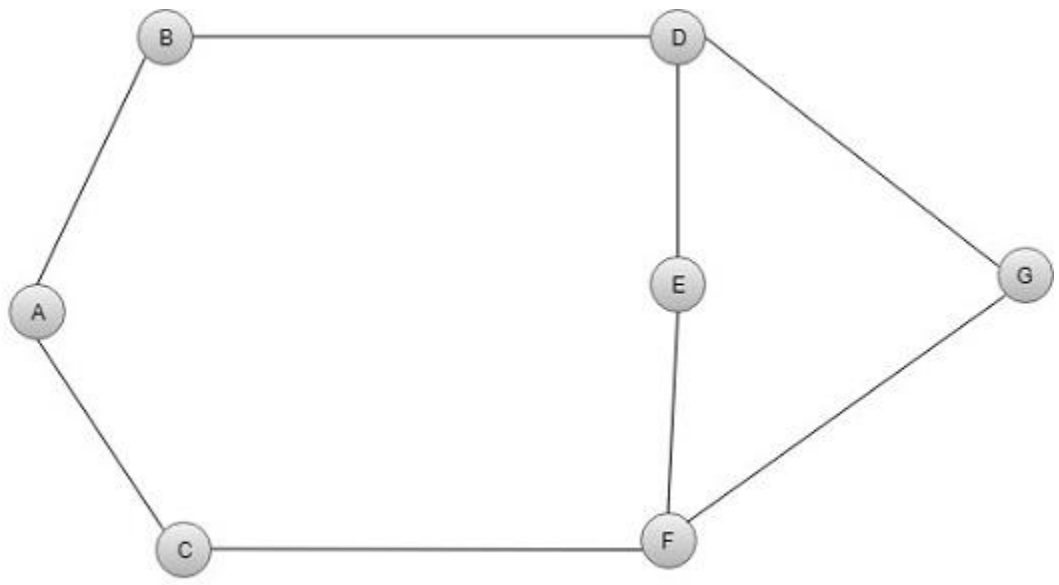


Fig:A graph with large independent sets of size 4 and smallest vertex cover of size 3.