**21**

**3COLBUT1**: Given an undirected graph G, is there a way to color G using at most 3 colors such that at most 1 edge violates the coloring constraint (i.e., there can be at most 1 edge with same coloured endpoints).

Prove that **3COLBUT1** is **NP** complete.

The **3COLBUT1** problem takes as input an undirected graph **G(V,E)** and asks if there is a way to color **G** using at most 3 colors such that at most 1 edge violates the coloring constraint (i.e., there can be at most 1 edge with same coloured endpoints).

**Verification Algorithm for YES-instance of the decision problem 3COLBUT1:**

Consider the following algorithm.

Use proof **P =** mapping of each vertices {R,G,B}, such that color of each vertex **∈** {R,G,B}

def **verify3ColorBut1Proof(Input G, Proof P)**:

1. Initialise counter=0;
2. for every edge(u,v) **∈** G
   1. If [ col(u) == col (v) ] : counter=counter+1
   2. If [ counter > 1 ] : return **FALSE**
3. return **TRUE**

**Complexity analysis:**

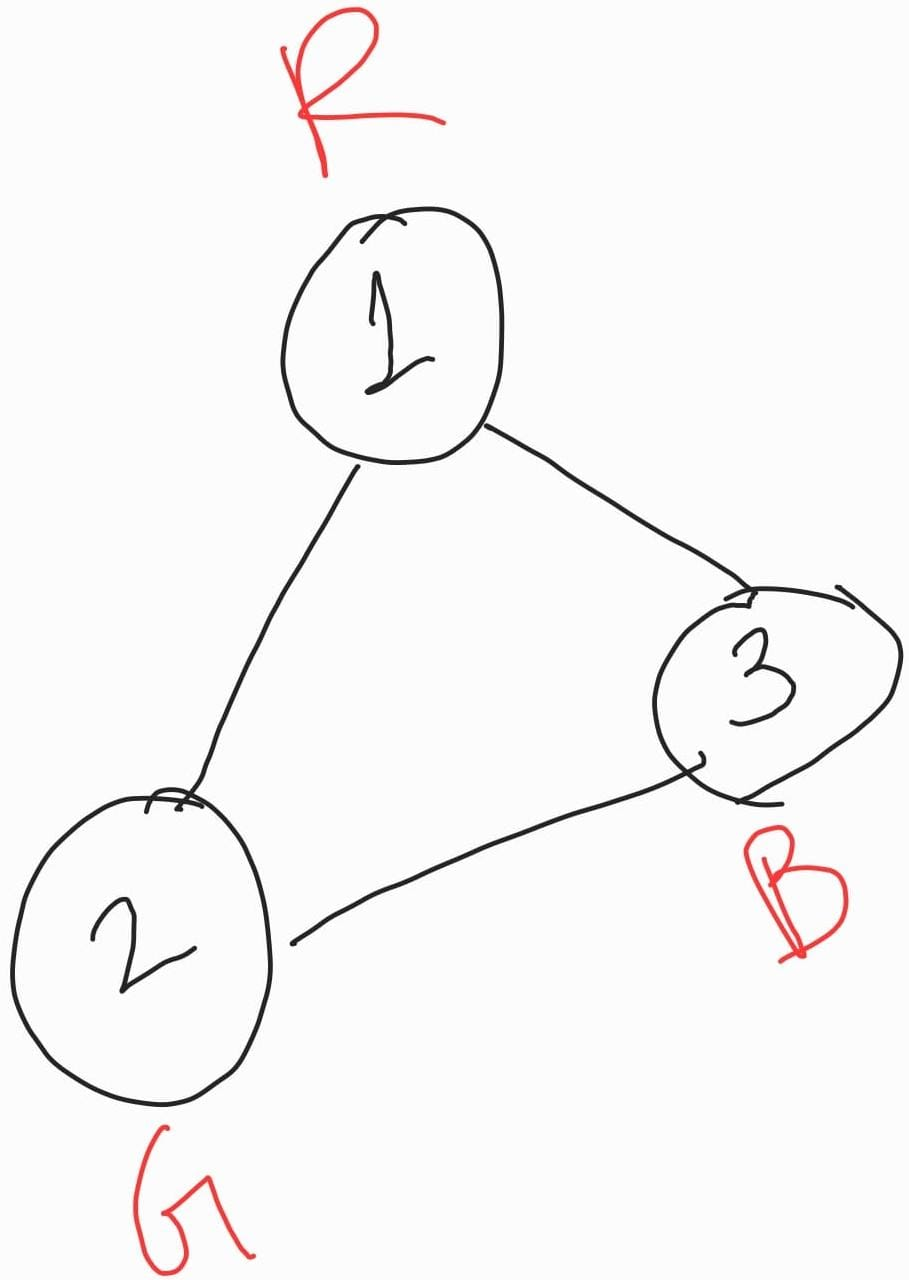
Line **(1)** in the reduction step takes constant time. The for loop in Line **(2)** will run as many times as the number of edges **E** in **G**. Thus its a traversal in **G.**

*Hence, the verification algo. would take* ***O(V+E)*** *time i.e. polynomial time****.***

**Correctness:**

* ***If G*** *is a* ***YES*** *instance, then there must be a* ***P*** *for which* ***verify3ColorfBut1Proof*** *returns* ***TRUE*** *:*

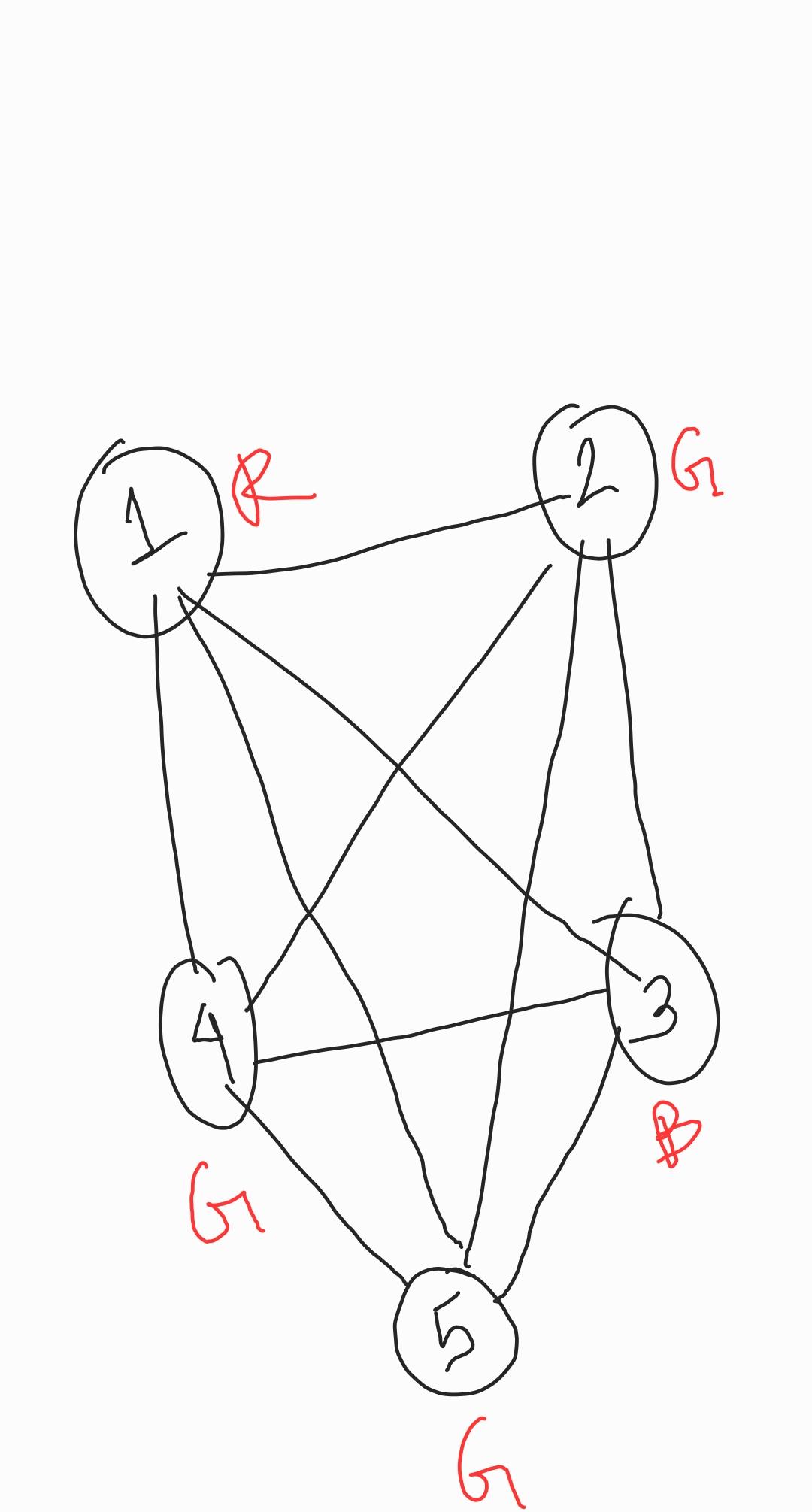
Let the input be a graph **G:**



**verify3ColorBut1Proof** will return TRUE as there is a coloring possible in **G** with **P**={ (1,R), (2,G), (3,B) }, such that no two neighbors have the same color.

* ***If G*** *is a* ***NO*** *instance, then there for every* ***P****,* ***verify3ColorfBut1Proof*** *returns* ***FALSE*** *:*

Let the input be a graph **G:**



Clearly, there are **2** edges **(2,4)** & **(1,5)** which violate the coloring constraint. Hence it cannot be colored appropriately with 3 colors such that at most 1 edge violates the coloring constraint. Hence its a no instance of **3COLBUT1.**

Now, let's assume that **verify3ColorBut1Proof** returns **TRUE** on G. Hence the value of the **‘counter’** variable should be <= 1. That would mean there is at most one edge in G which violates the coloring constraint, which would further make G a **TRUE** instance of **3COLBUT1.** Hence, it is contradicting our initial belief that **G** was a **TRUE** instance of **3COLBUT1**.

*(Observe that there are already more than 1 edge (as stated above) whose vertices are coloured with same color)*

Hence**, verify3ColorBut1Proof** cannot return **TRUE** for a **FALSE** input instance for any **P.**

Hence**, 3COLBUT1** belongs to **NP** class as the verification for its **YES** instance can be done in polynomial time.

**Proof of NP Completeness**

**3-COL** is a known NP-Hard problem: Given a graph G(V, E), return 1 if and only if there is a proper coloring of G using at most 3 colors.

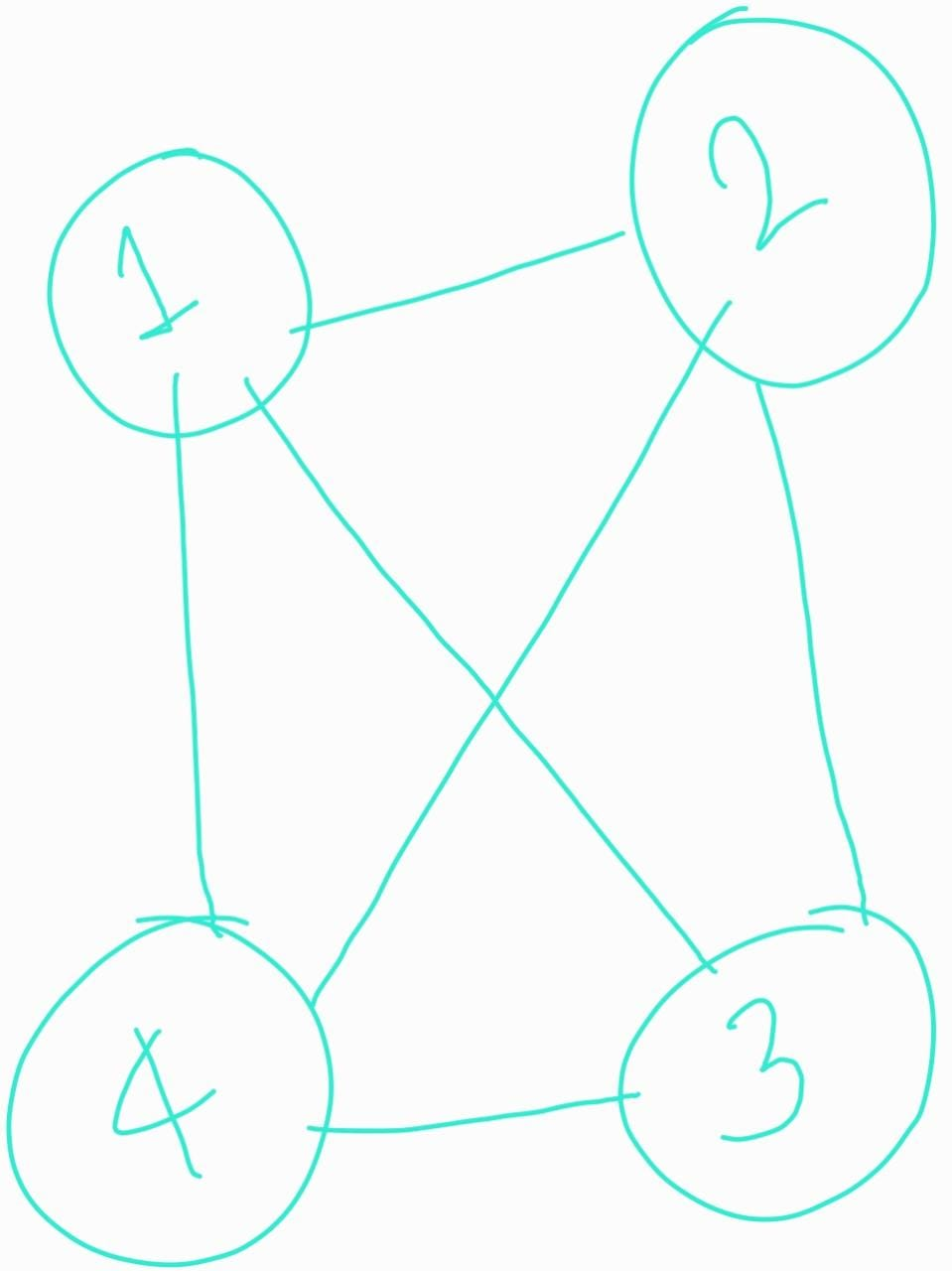
To prove: **3COLBUT1** is NP complete.

Aim: **3COL** <= **3COLBUT1**

Consider the following algorithm.

def **reduce(G)**:

1. Create a graph H which is a copy of G.
2. Add a 4 vertex clique to H as a separate component.



1. Return the transformed graph as **H.**

**Complexity analysis:**

Line **(1)** in the reduction step takes **O(V+E)** time to copy G i.e. linear time in G.

Line **(2)** takes constant time.

Hence, the reduction takes ***O(V+E)*** time i.e. its a poly-time reduction.

**Statement of correctness lemma for the reduction:**

*Given an undirected graph* ***G, 3COL*** *returns* ***TRUE*** *for the original graph* ***G*** *(i.e. there exists a proper 3 coloring in* ***G****), iff* ***3COLBUT1*** *returns* ***TRUE*** *for the reduced graph* ***H****(i.e. there a way to color H using at most 3 colors such that at most 1 edge violates the coloring constraint****. [H=reduce(G)]***

**Correctness of the reduction:**

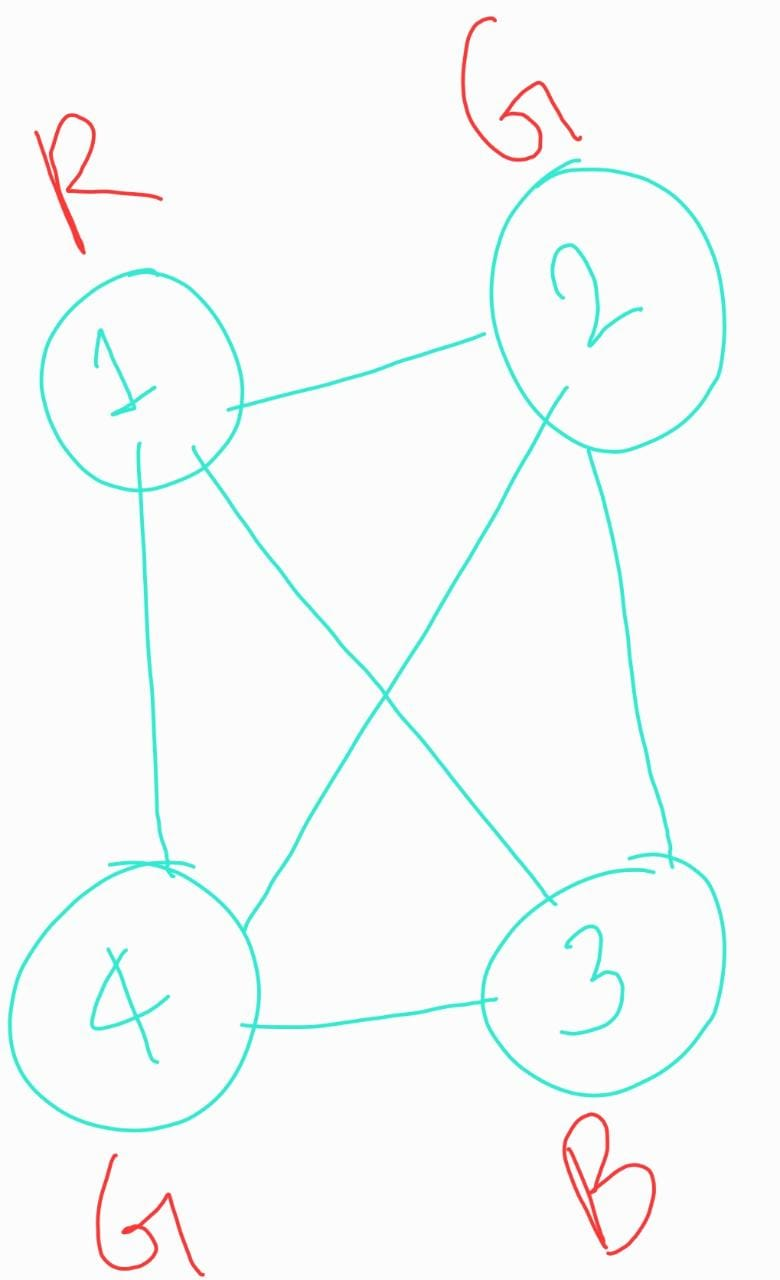
In other words, say the reduce algorithm is converting an instance x of ***3COL*** into a special

instance y of **3*COLBUT1***, then we need to prove:

* (=>) If x is a “TRUE” instance of ***3COL***, then y is a “TRUE” instance of **3*COLBUT1***.
* (<=) If y is a “TRUE” instance of **3*COLBUT1***, then x is a “TRUE” instance of ***3COL.***

**Forward proof(=>):**

Say graph **G** has a proper 3 coloring.The reduction step will add one 4-vertex clique. When H is attempted to color using at most 3 colors, the clique component would be colored as:



Clearly, there is an edge **(2,4)** whose vertices would be colored using the same color.

When **H** is given as input to **3COLBUT1**, we can clearly notice that if the edge **(2,4)** is ignored, then **H** can also be colored properly using 3 colors.

There will not be any other edge in H violating the coloring constraint, as there was no such edge in **G.** So, **(2,4)** is the only violating edge in **H.**

Thus, **3COLBUT1** will also return **TRUE**.

**Backward proof(<=):**

If y is a “TRUE” instance of ***3COLBUT1***, then x is a “TRUE” instance of ***3COL.***

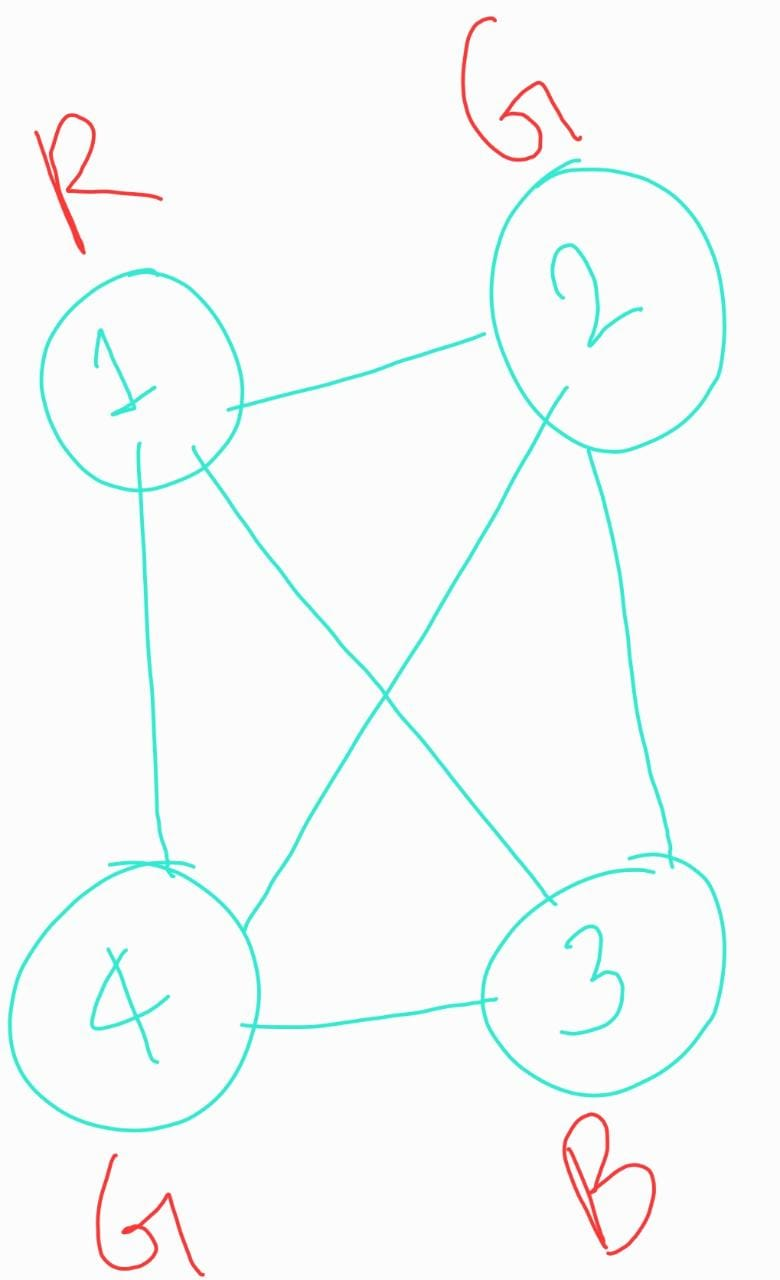
***OR,*** *we can prove its contrapositive statement:*

If x is a “FALSE” instance of ***3COL***, then y will also be a “FALSE” instance of ***3COLBUT1.***

Say graph **G** does not have any proper 3 coloring i.e. **i.e. G** is a **FALSE** instance of ***3COL.***

Then there must be ***at least one edge*** in **G** which has 2 vertices of the same color.

The reduction step will add one 4-vertex clique. When **H** is attempted to color using at most 3 colors, the clique component would be colored as:



Thus in **H**, there would be at least **2 edges** with vertices with the same color: at least one would be inherited from **G &** another one would be from the newly added clique component.

When **H** is given as input to **3COLBUT1**, even if it excludes one edge while trying to color using 3 colors, there would still be at least one edge left to ignore. Thus, there will not be any proper 3 coloring in **H.**

Hence, **3COLBUT1** will also return **FALSE**.

Hence, the above reduction algorithm is a polynomial-time reduction from **3COL** to **3COLBUT1,** & since **3COL** is already **NP Hard**, so **3COLBUT1** will be NP Hard.

And, since **3COLBUT1** also belongs to the **NP** class, hence it is **NP Complete**.