VERTEX COVER PROBLEM

Rudrani Chavarkar - 13
Vaishnavi Hindalekar - 40
Atharva Jadhav - 44
Arush Karekar - 57

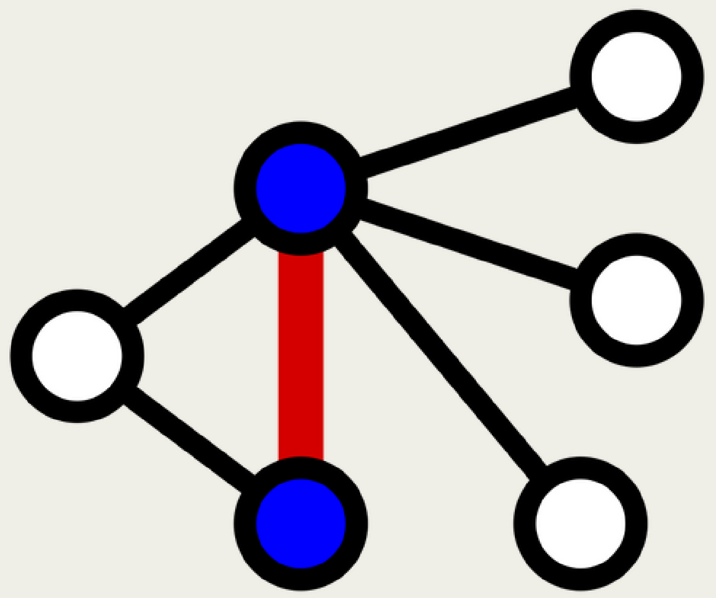
ADSA
T1

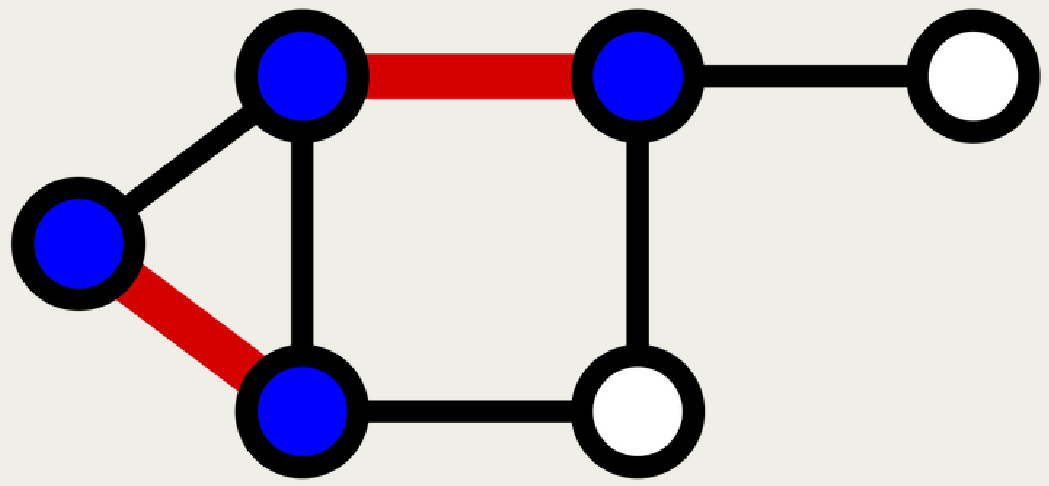
WHAT IS VERTEX COVER PROBLEM?

The vertex cover problem is a mathematical problem in graph theory. You are given an **undirected graph G = (V, E)** and you need to find a subset of vertices V' such that for each edge (u, v) in E, at least one of the vertices u or v belongs to V'. The goal is to minimize the size of V'.

In simpler terms, the vertex cover problem is to find the **smallest subset of vertices** that covers every edge in an undirected graph.

An edge is covered if one of its endpoint is chosen.





WHAT IS VERTEX COVER PROBLEM?

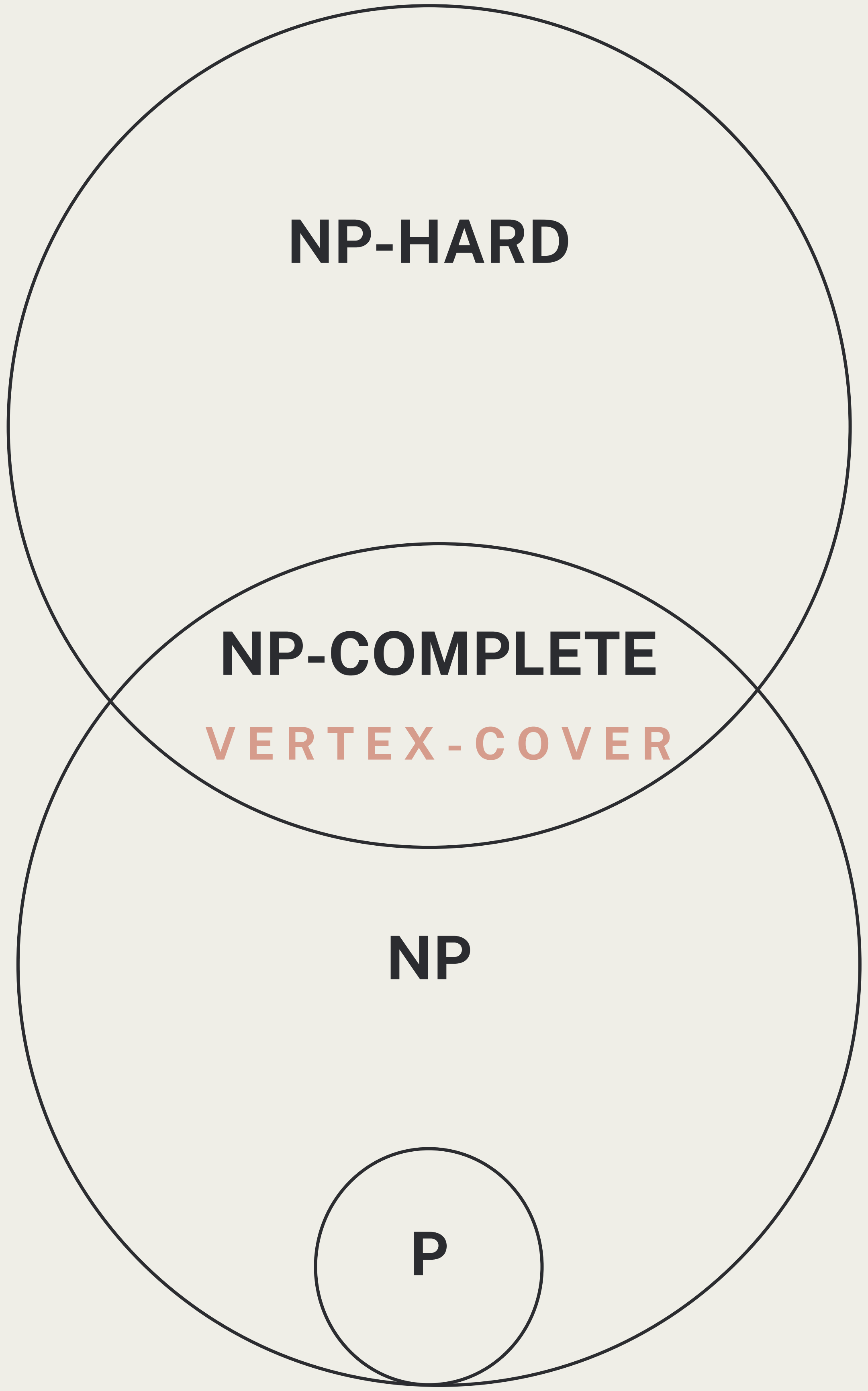
The vertex cover problem is an **NP-Complete problem**, which means that there is **no known polynomial-time solution** for finding the minimum vertex cover of a graph unless it can be **proven that P = NP.**

Algorithms to find the vertex cover of a graph:

- **Approximation algorithms** for Vertex Cover guarantee a solution that is at most twice as large as the optimal vertex cover, ensuring a reasonably close approximation to the best solution.
- **Greedy algorithms** for Vertex Cover choose vertices based on their local properties, such as high degree (vertex with the highest number of edges), without guaranteeing an optimal solution.

*A polynomial-time solution is an algorithm that can solve a problem in a time frame where the time it takes grows at most as a polynomial function of the problem size.

CLASSIFICATION OF PROBLEMS



P

- consists of problems for which there **exists** an algorithm that can solve them in **polynomial time**
- considered **"easy"** in terms of computational complexity

NP

- consists of problems for which a **proposed solution** can be **verified** in **polynomial time**
- **not clear** whether **NP problems** can also be solved in polynomial time
- P vs. NP problem, famous open problem

NP-HARD

- can't be solved in polynomial time with an efficient algorithm
- at least as **hard** as any NP problem

NP-COMPLETE

- **subset** of NP-hard problems
- if you could find a **polynomial-time algorithm** to solve it, you could use that algorithm to solve all other problems in NP in polynomial time

WHY IS VERTEX COVER NP-COMPLETE ?

To prove vertex-cover is np-complete we have to prove that it is both :

- NP
- NP-HARD

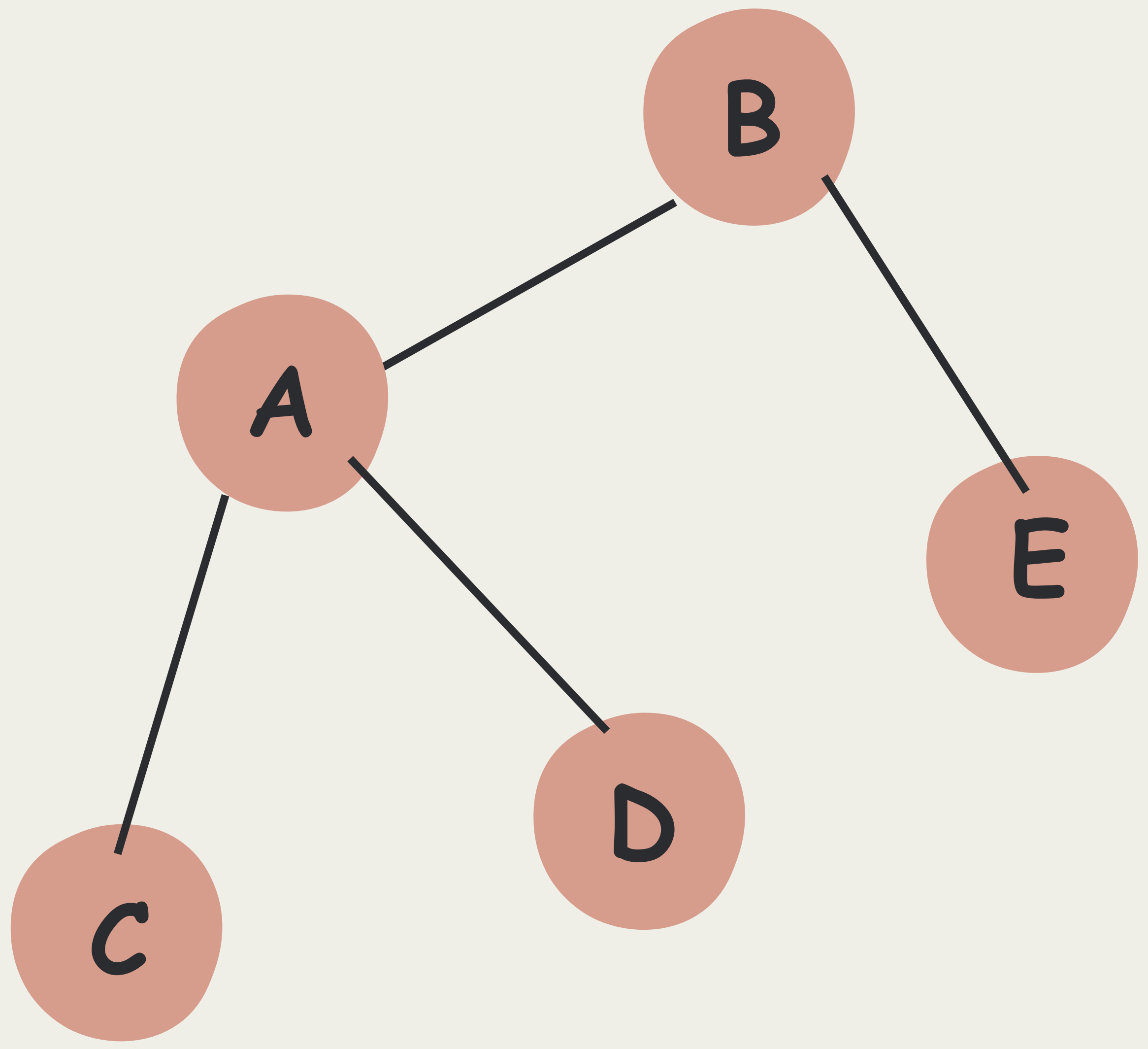
VERTEX-COVER IS NP

Vertices: {A, B, C, D, E}
Edges: {(A, B), (A, C), (B, E), (C, D)}

Let's say we have a proposed vertex cover, **V'= {A, C, E}.** We want to verify whether this is a valid vertex cover for the graph.

We'll check each edge to see if at least one of its endpoints is in the proposed vertex cover.

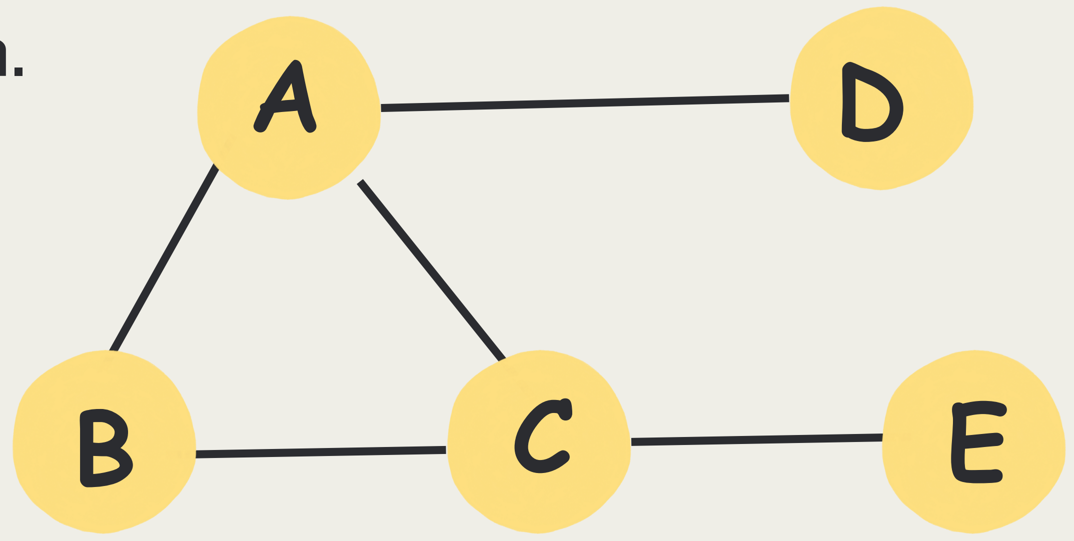
1. Edge (A, B): One endpoint (A) is in V'. This edge is covered.
2. Edge (A, C): Both endpoints (A and C) are in V'. This edge is covered.
3. Edge (B, E): One endpoint (E) is in V'. This edge is covered.
4. Edge (C, D): One endpoint (C) is in V'. This edge is covered.



In this example, we verified the validity of the proposed vertex cover in polynomial time (in this case, in constant time) by checking each edge individually to ensure that at least one of its endpoints is included in the proposed cover

VERTEX-COVER IS NP-HARD

Vertex Cover is NP-hard because it can be used to solve the Clique problem, which is a classic NP-complete problem. In the Clique problem, you're asked to find a complete subgraph (a set of vertices where every pair is connected by an edge) of a specified size in a given graph.



Reduction from Clique to Vertex Cover goes like this:

Given an instance of the Clique problem,

- Create a complement graph .
- Then, find the minimum Vertex Cover in this complement graph
- The size of this minimum cover is equal to the size of the maximum clique in the original graph.

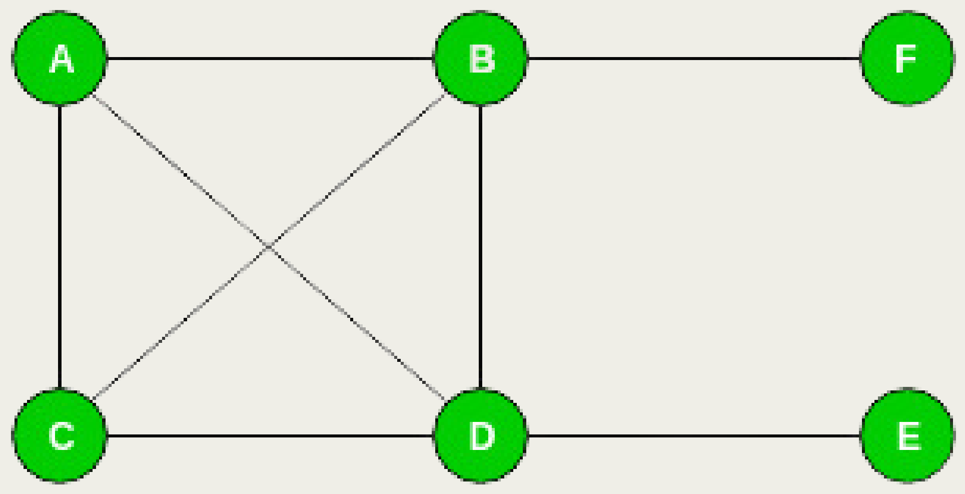
Since the Clique problem is NP-complete and Vertex Cover can be reduced to it, Vertex Cover is also NP-complete.

VERTEX-COVER IS NP-HARD

In graph G , the size of maximum clique(k) is 4 .

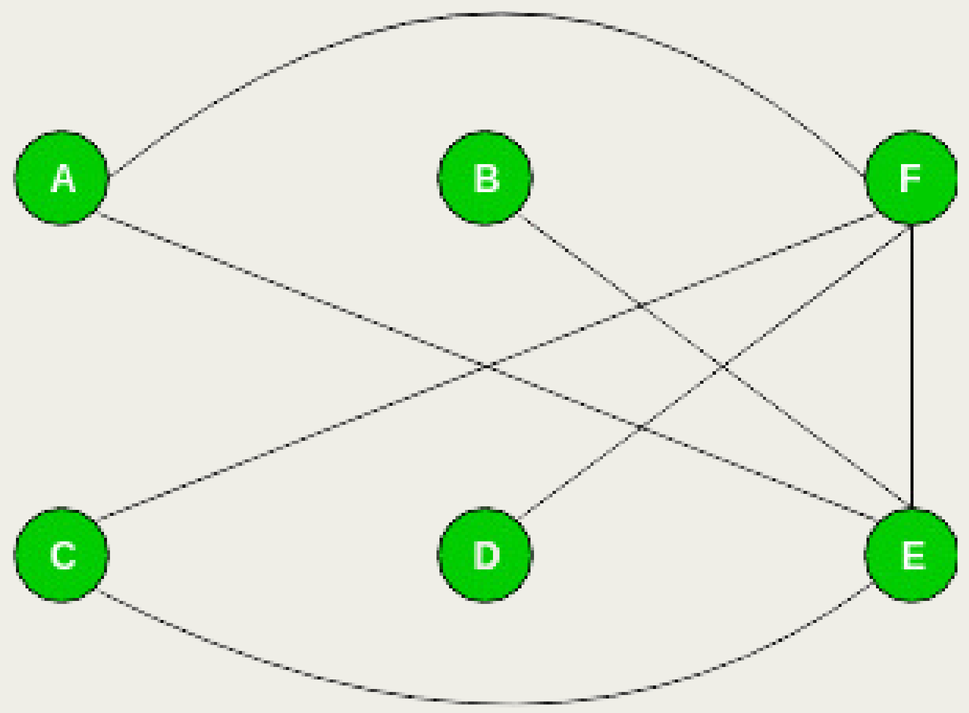
In graph G', the size of minimum vertex cover is 2.

Thus, I V I - k = 6 - 4 = 2 i.e. equal to MVC in G'.



G

k = 4 {A,B,C,D}



G'

v' = 2 {E,F}

Thus, we can say that there is a clique of size k in graph G if and only if there is a vertex cover of size |V| – k in G', and hence, any instance of the clique problem can be reduced to an instance of the vertex cover problem. Thus, vertex cover is NP Hard. Since vertex cover is in both NP and NP Hard classes, it is **NP Complete**.

APPROXIMATION METHOD: ALGORITHM

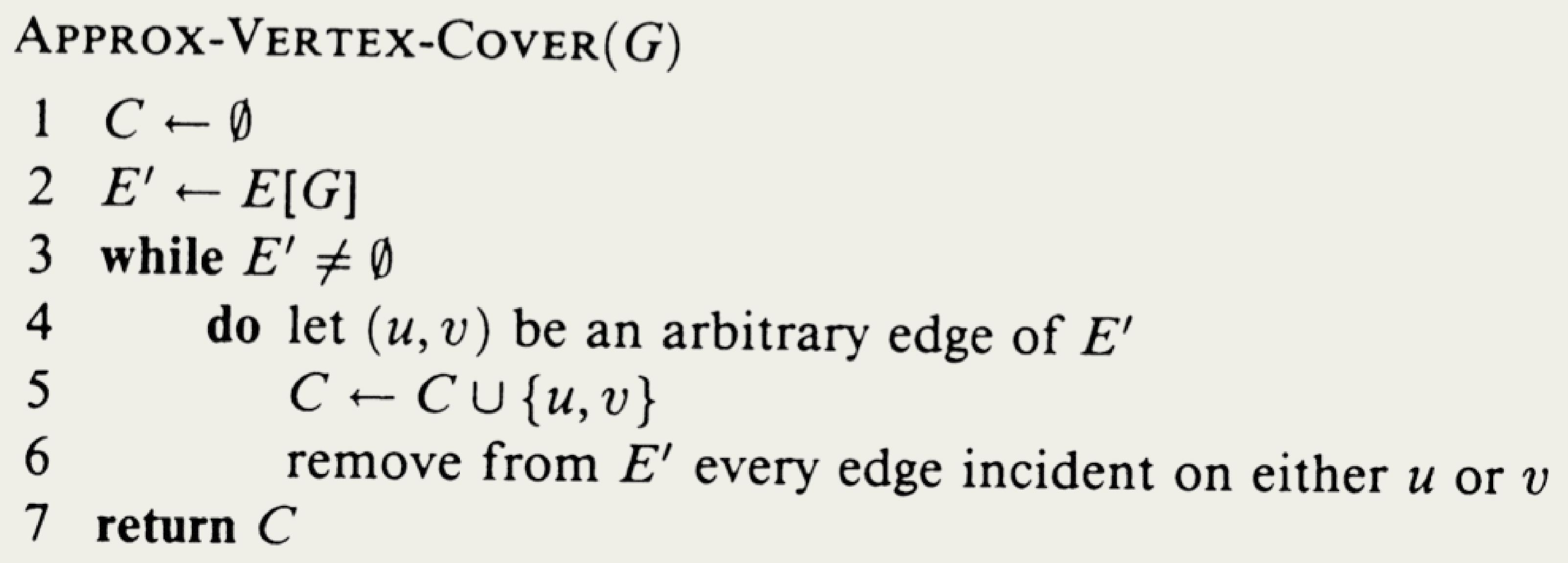
Overview :

An approximation algorithm is a way of dealing with NP-completeness for an optimization problem. This technique does not guarantee the minimum Vertex Cover. The goal of the approximation algorithm is to come as close as possible to the optimal solution in polynomial time. Such algorithms are called approximation algorithms or heuristic algorithms.

Idea:

We have to consider a random edge, eliminate all edges associated to it. Repeat till all edges are covered.

APPROXIMATION METHOD: ALGORITHM



```
APPROX-VERTEX-COVER($G$)
$C \leftarrow \emptyset$
$E' \leftarrow E[G]$
while $E' \neq \emptyset$
      do let $(u, v)$ be an arbitrary edge of $E'$
         $C \leftarrow C \cup \{u, v\}$
         remove from $E'$ every edge incident on either $u$ or $v$
return $C$
```

1) Initialize the result as {}

2) Consider a set of all edges in given graph. Let the set be E.

3) Do following while E is not empty

    a) Pick an arbitrary edge (u, v) from set E and add ‘u' and 'v' to result

    b) Remove all edges from E which are either incident on u or v.

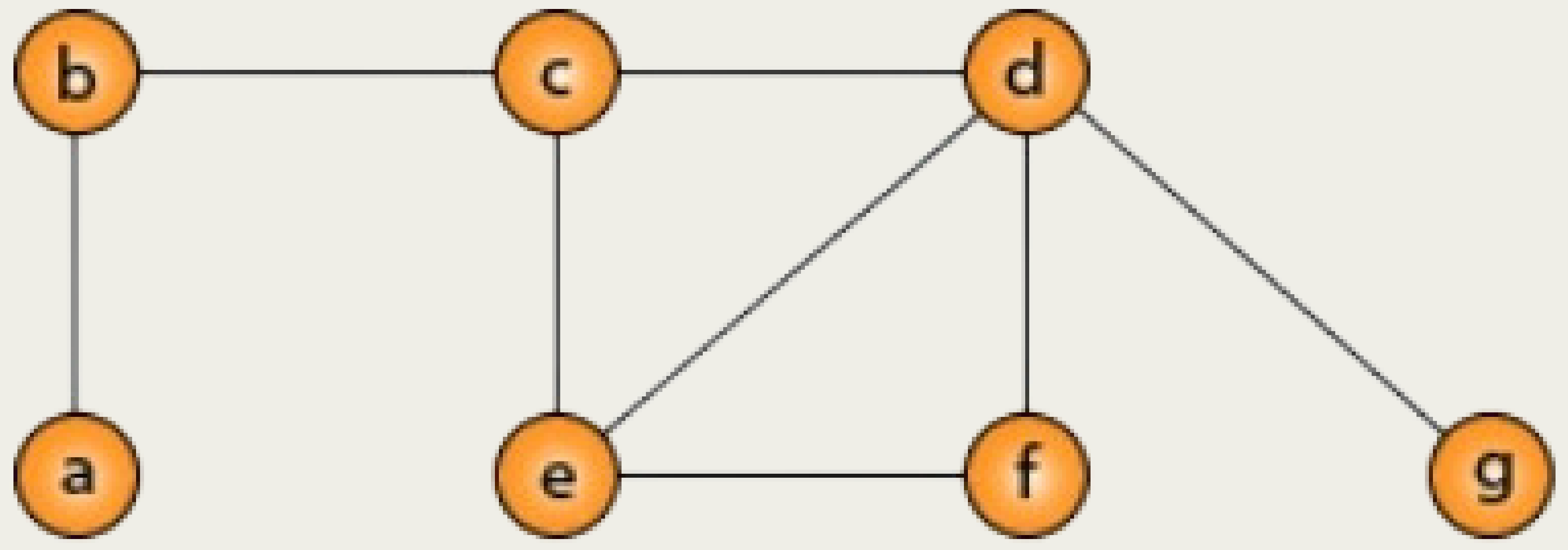
4) Return result

APPROXIMATION METHOD EXAMPLE

Let us consider this graph, for which we have to find the Vertex Cover.

E: Set of all uncovered edges

V: Vertex Cover solution set



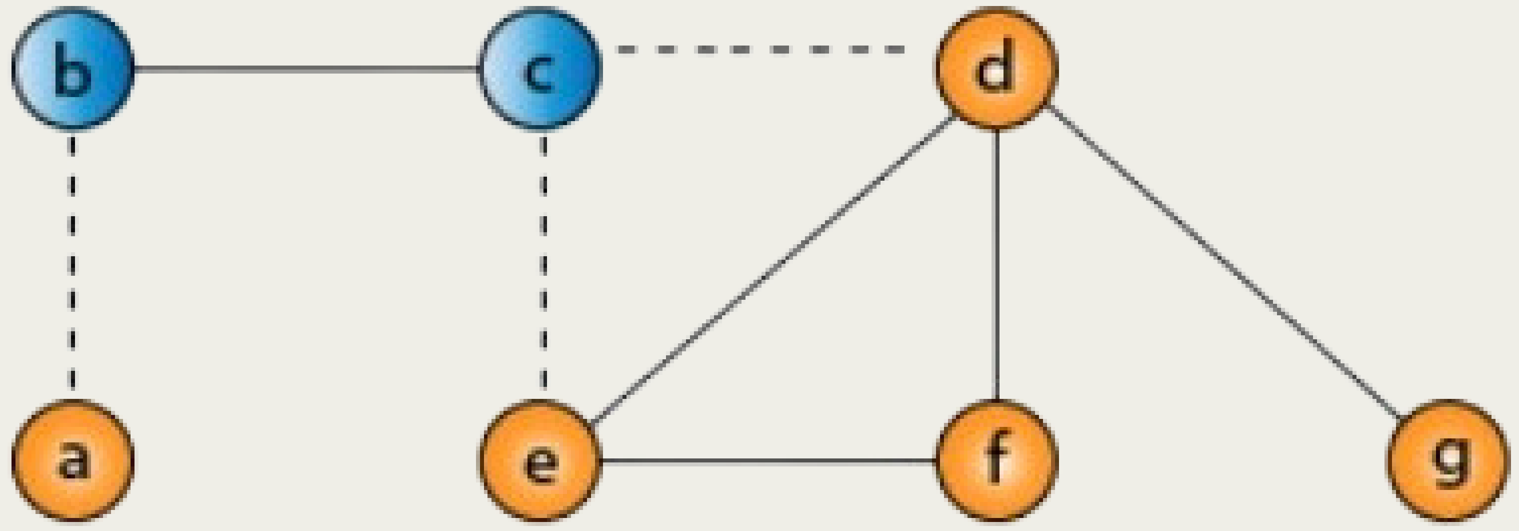
(1)

E = {(b,c), (b,a), (c,d), (c,e), (d,e), (d,f), (d,g), (e,f)}

V={}

APPROXIMATION METHOD EXAMPLE

Firstly, let’s consider edge ‘bc ‘ as an arbitrary edge. Upon adding ‘b’ & ‘c’ to the vertex cover and eliminating all edges connected to vertices ‘b’ & ‘c’ from E;



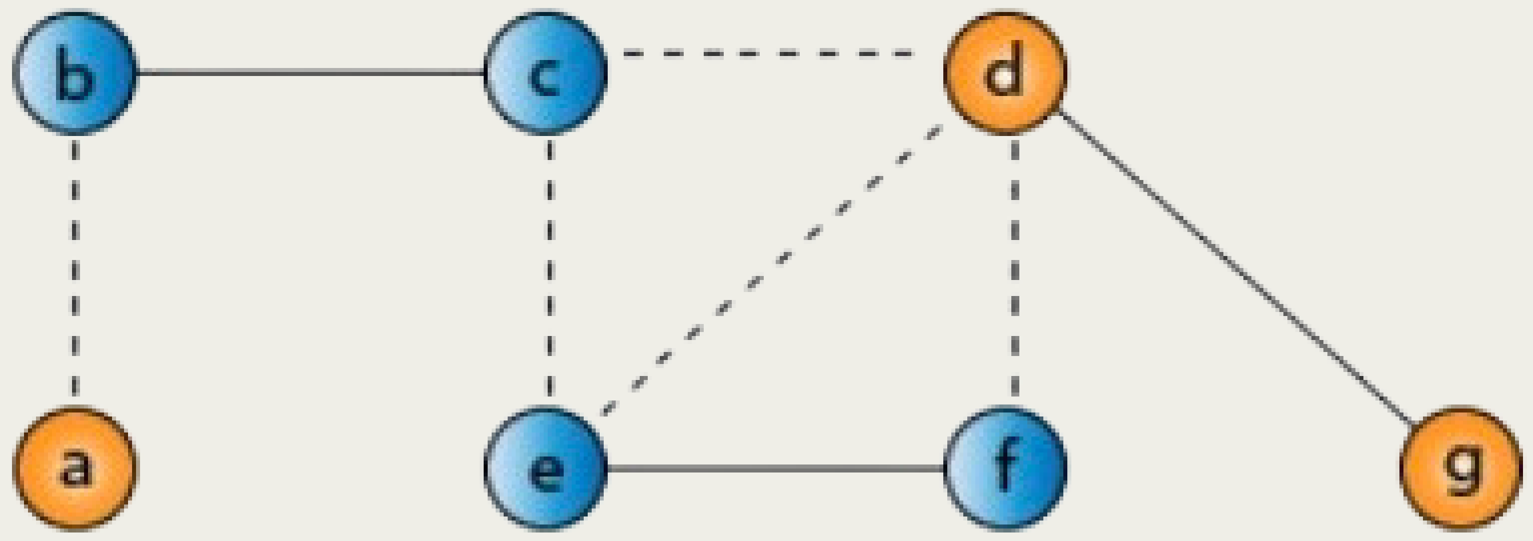
(2)

E = {(d,e), (d,f), (d,g), (e,f)}

V={b, c}

APPROXIMATION METHOD EXAMPLE

Now, consider ‘ef’ as the next arbitrary edge. Upon adding ‘e’ & ‘f’ to the vertex cover and eliminating all edges connected to vertices ‘e’ & ‘f’ from E;



(3)

E = {(d,g)}

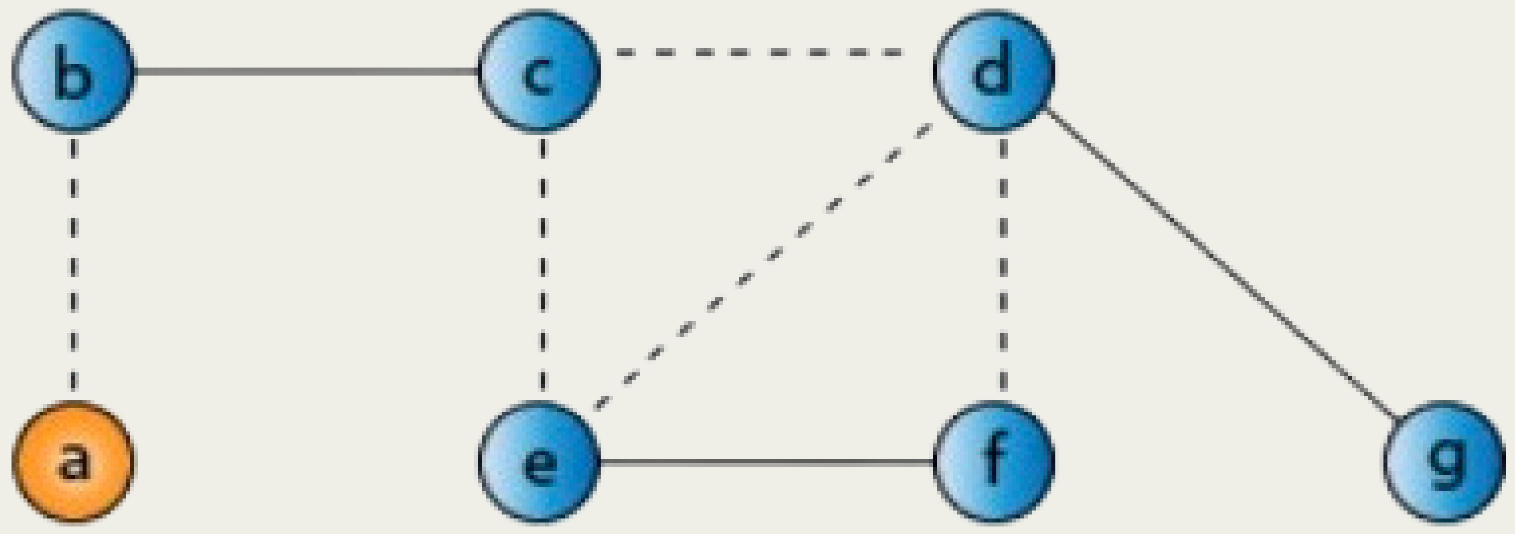
V={b, c, e, f}

APPROXIMATION METHOD: EXAMPLE

Finally, select ‘dg’ as the next edge. Add ‘d’ and ‘g’ to the vertex cover. Upon doing so, the set of edges ‘E’ is now exhausted. So we can terminate this process to obtain the final vertex cover as follows:

E = {}

V={b, c, e, f, d, g}



(4)

Although this is a valid Vertex Cover, it is far from being a minimum vertex cover. In view of this, we can implement a greedy algorithm on the same problem to obtain a smaller Vertex Cover.

USING GREEDY APPROACH

A QUICK OVERVIEW:

A greedy approach is a problem-solving strategy that makes locally optimal choices at each step with the hope of finding a global optimum. In other words, it seeks to maximize or minimize a particular objective by making the best choice available at each decision point without considering the long-term consequences.

IDEA:

We have to keep finding the vertex which covers the maximum number of edges.

USING GREEDY APPROACH

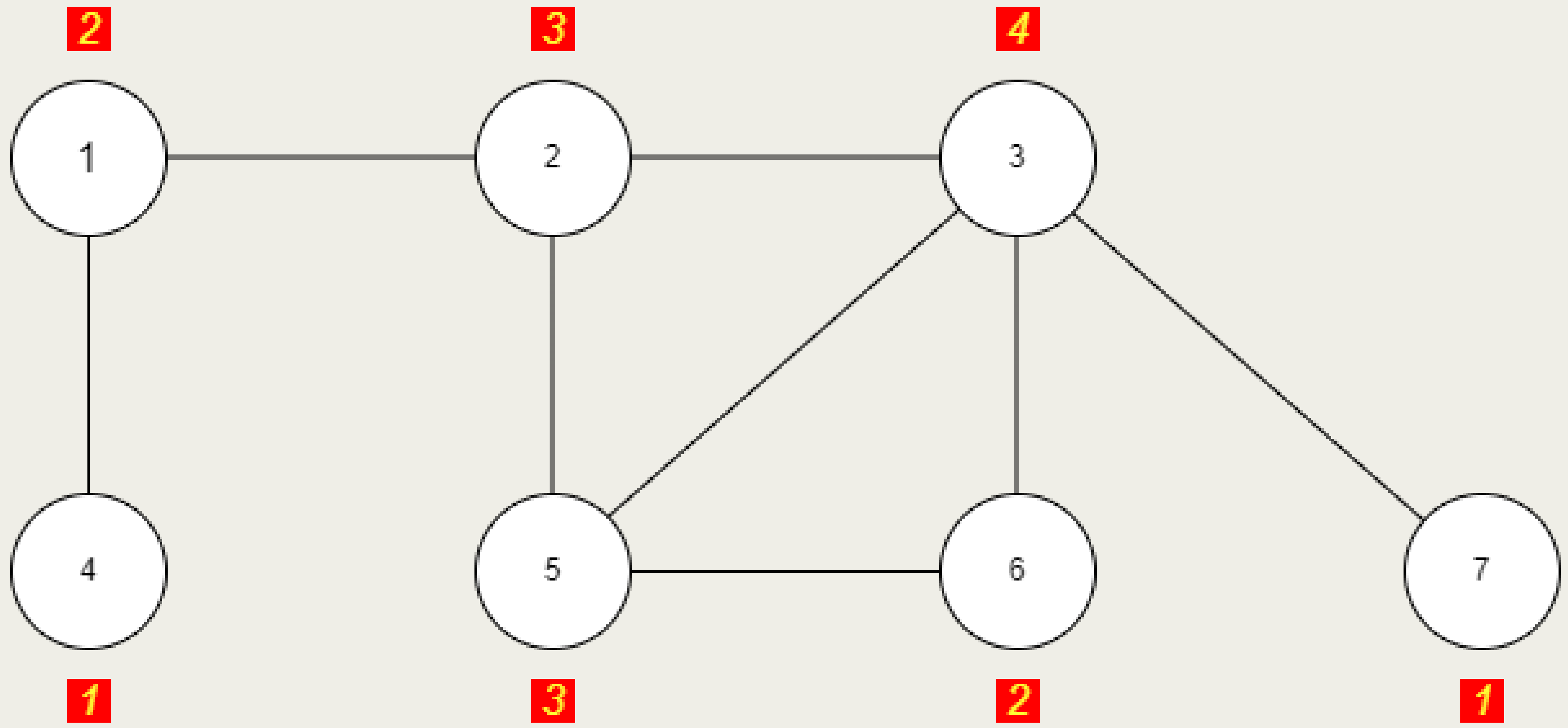
ALGORITHM:

```
# Initialize an empty set to represent the vertex cover
vertex_cover = empty_set()

# Repeat until all edges are covered
while there are uncovered edges in the graph {

   v = find_vertex_with_highest_degree() # Find a vertex 'v' with the most edges

   add 'v' to vertex_cover  # Add 'v' to the vertex cover

   remove_edges_incident_on_vertex(v)   # Remove all edges incident on 'v' from the graph
}

print vertex_cover  # Print the vertex cover
```

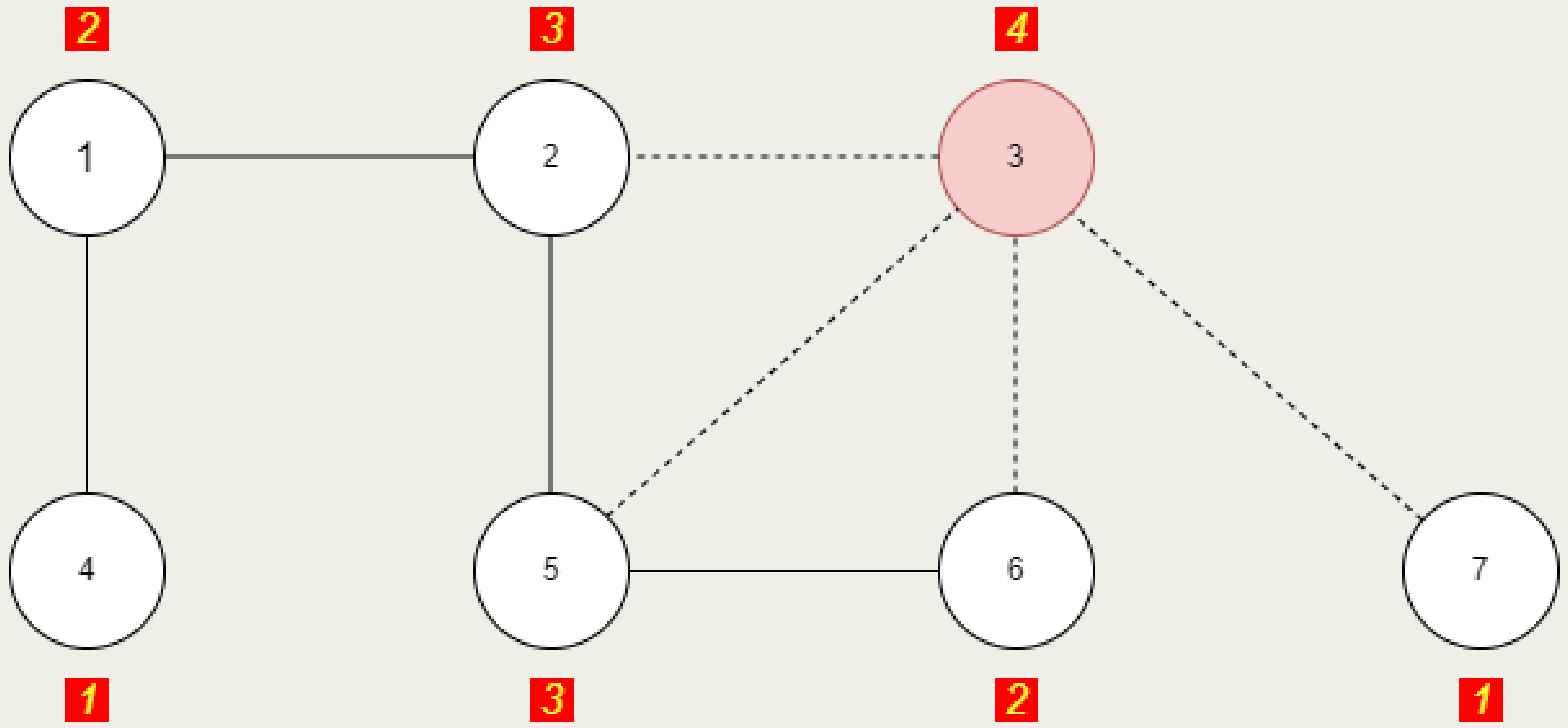
USING GREEDY APPROACH- EXAMPLE

vertex_cover = [ ]

No. of Edges = 8



USING GREEDY APPROACH- EXAMPLE

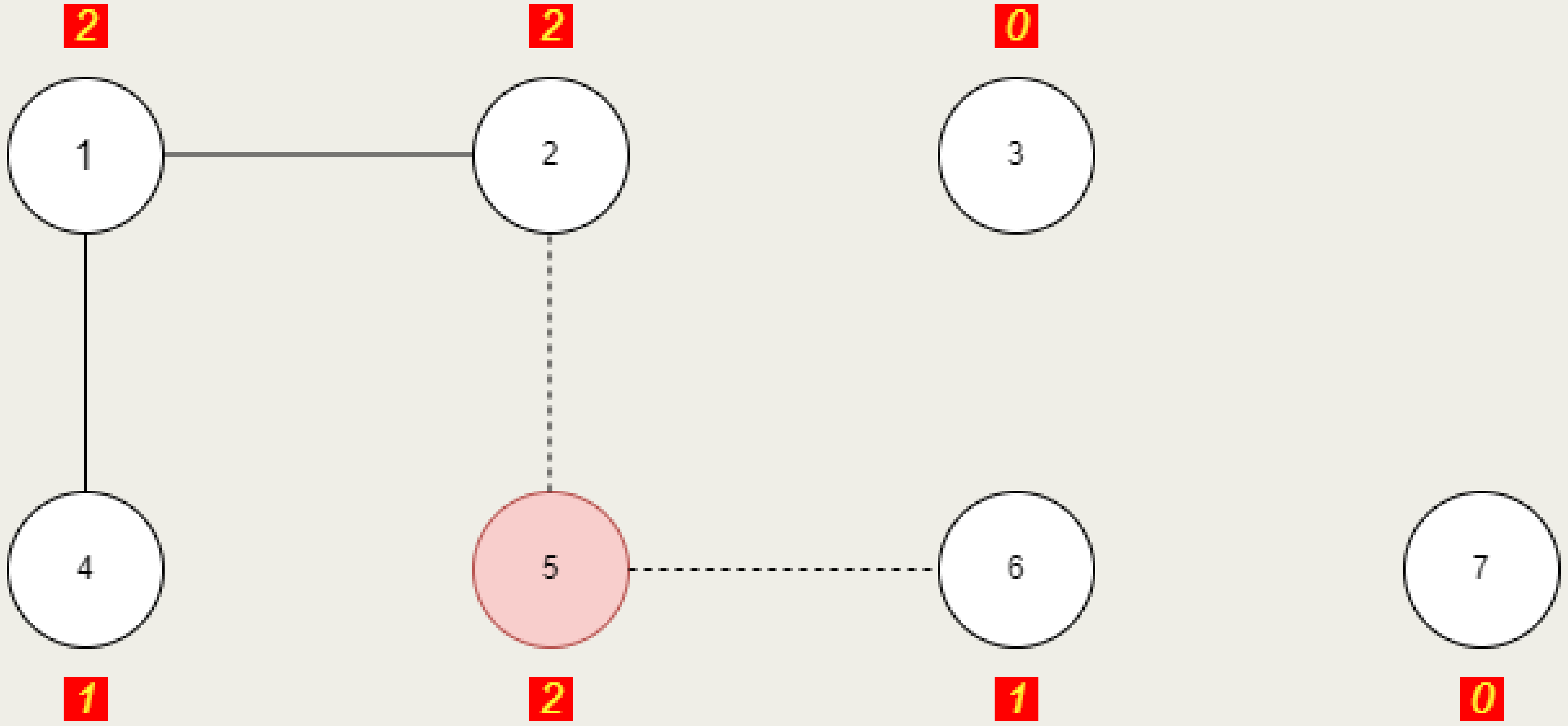


Select the Vertex with the highest degree and remove all the edges corresponding to it

USING GREEDY APPROACH- EXAMPLE

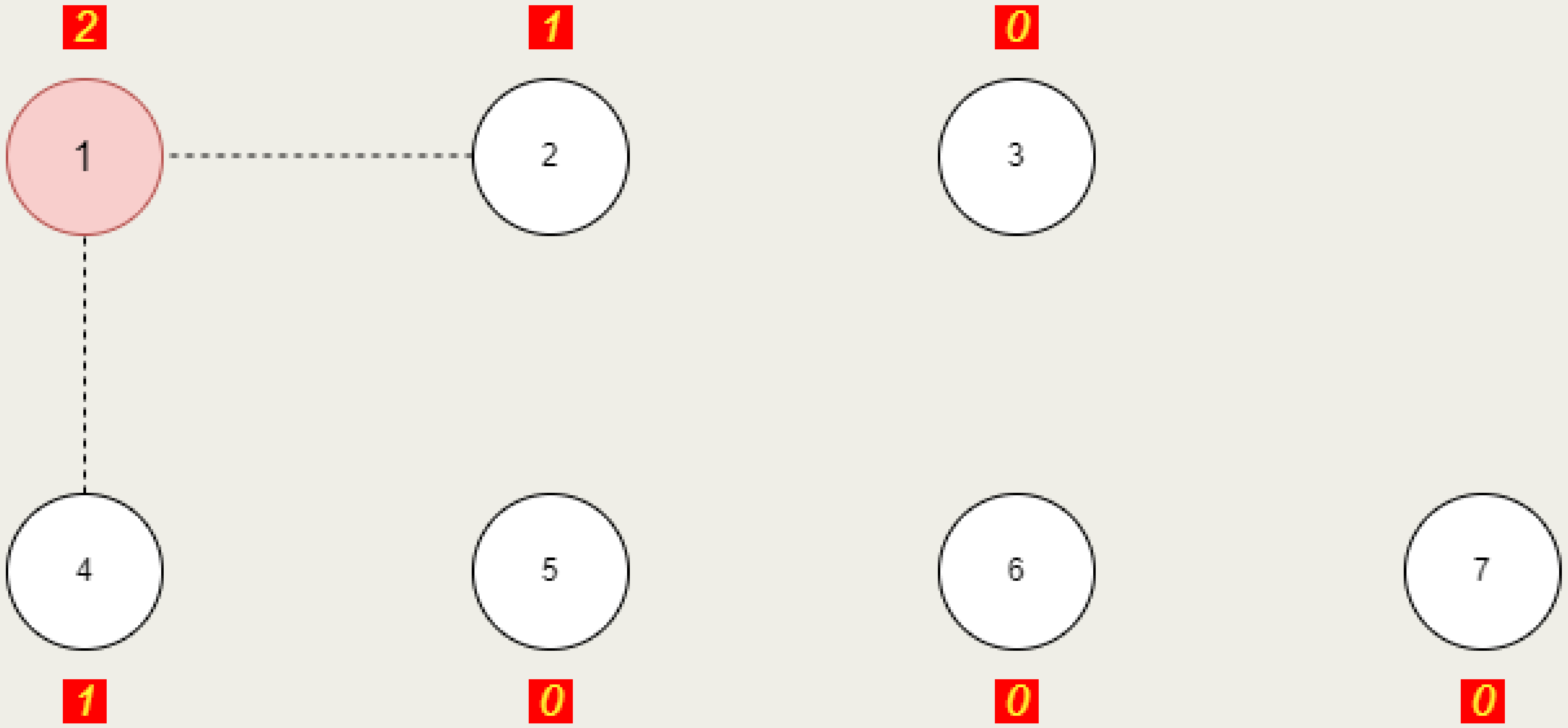
vertex_cover = [3]

No. of Edges = 4



Select the Vertex with the highest degree and remove all the edges corresponding to it

USING GREEDY APPROACH- EXAMPLE

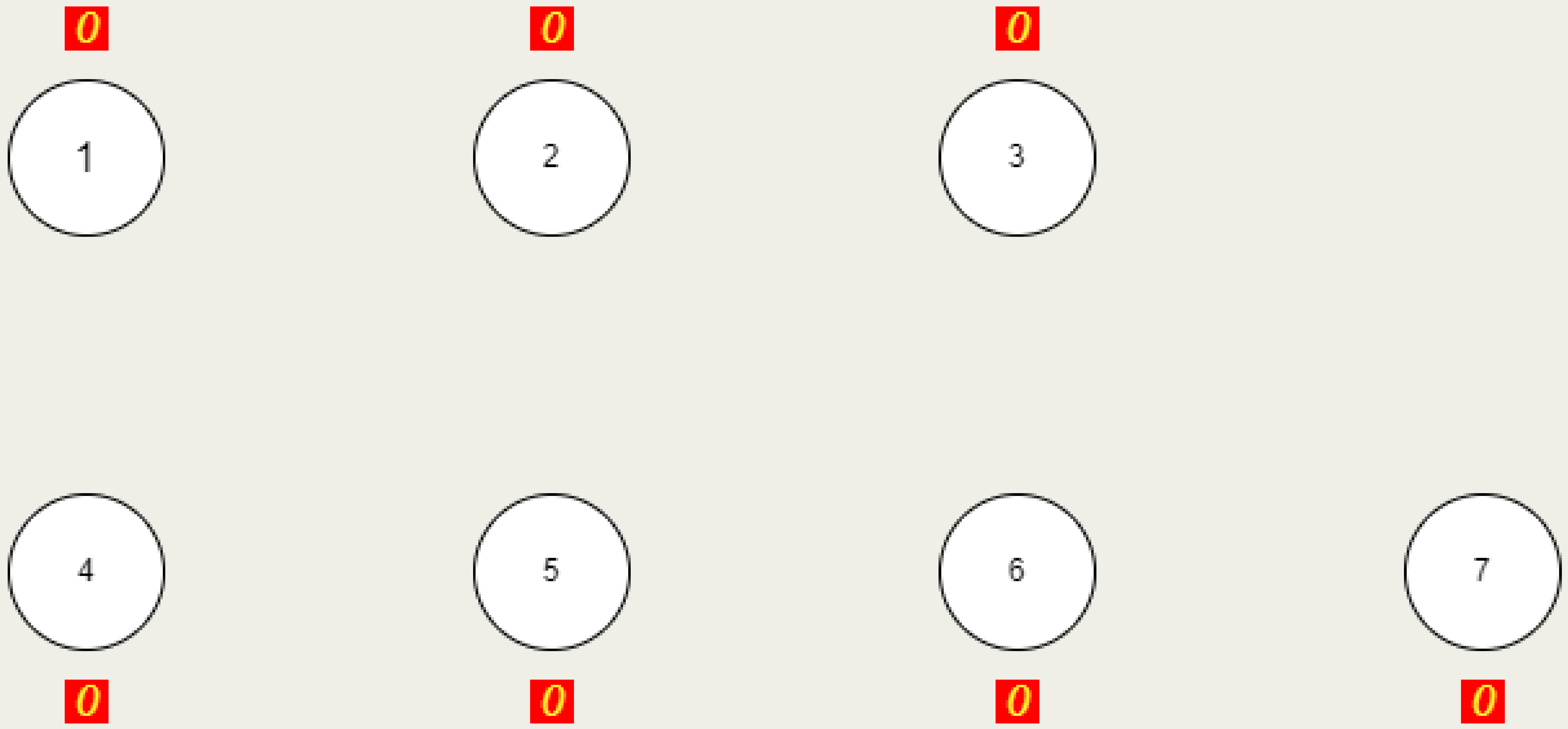


vertex_cover = [3, 5]

No. of Edges = 2

Select the Vertex with the highest degree and remove all the edges corresponding to it

USING GREEDY APPROACH - EXAMPLE



vertex_cover = [3, 5, 1]

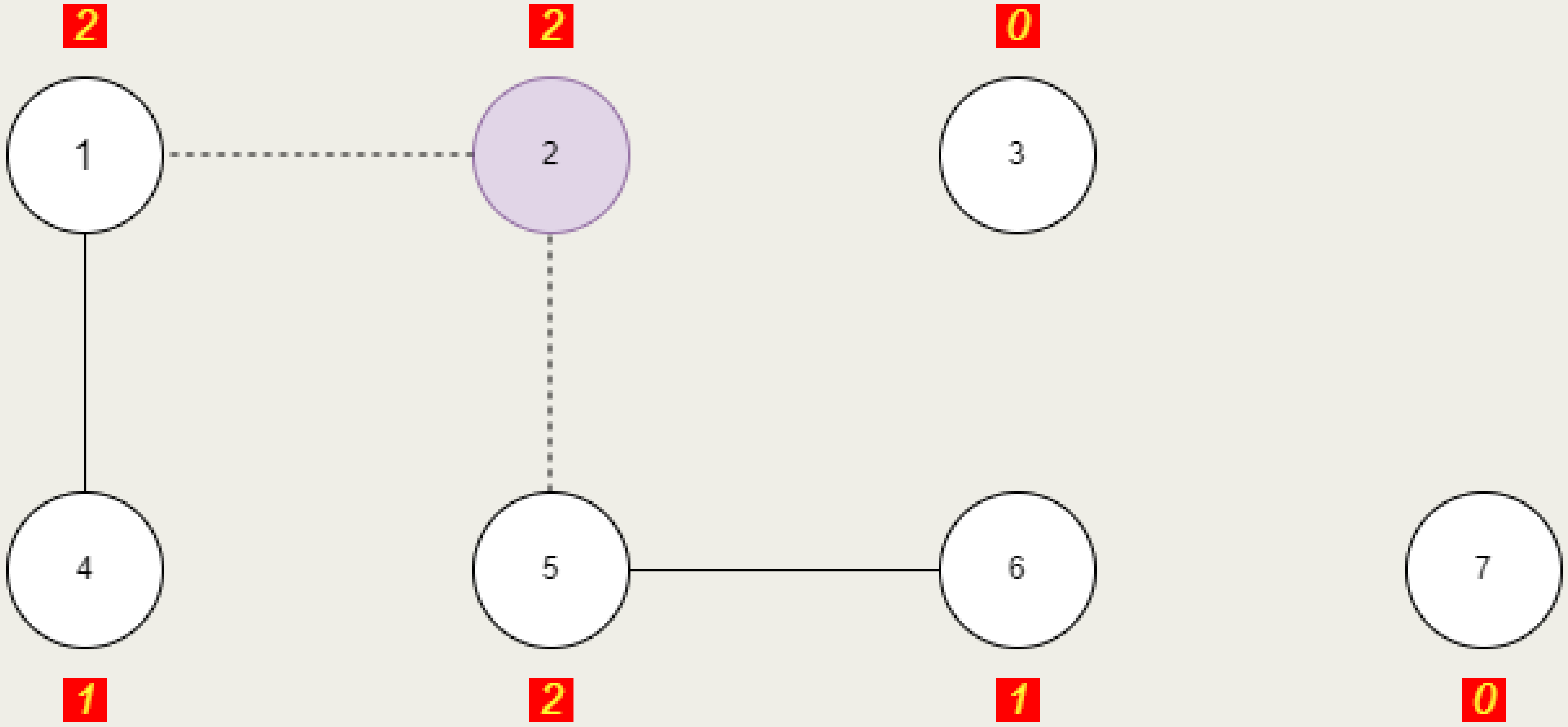
No. of Edges = 0

Do this until there are no edges in the graph

USING GREEDY APPROACH - EXAMPLE

vertex_cover = [3, 2]

No. of Edges = 2

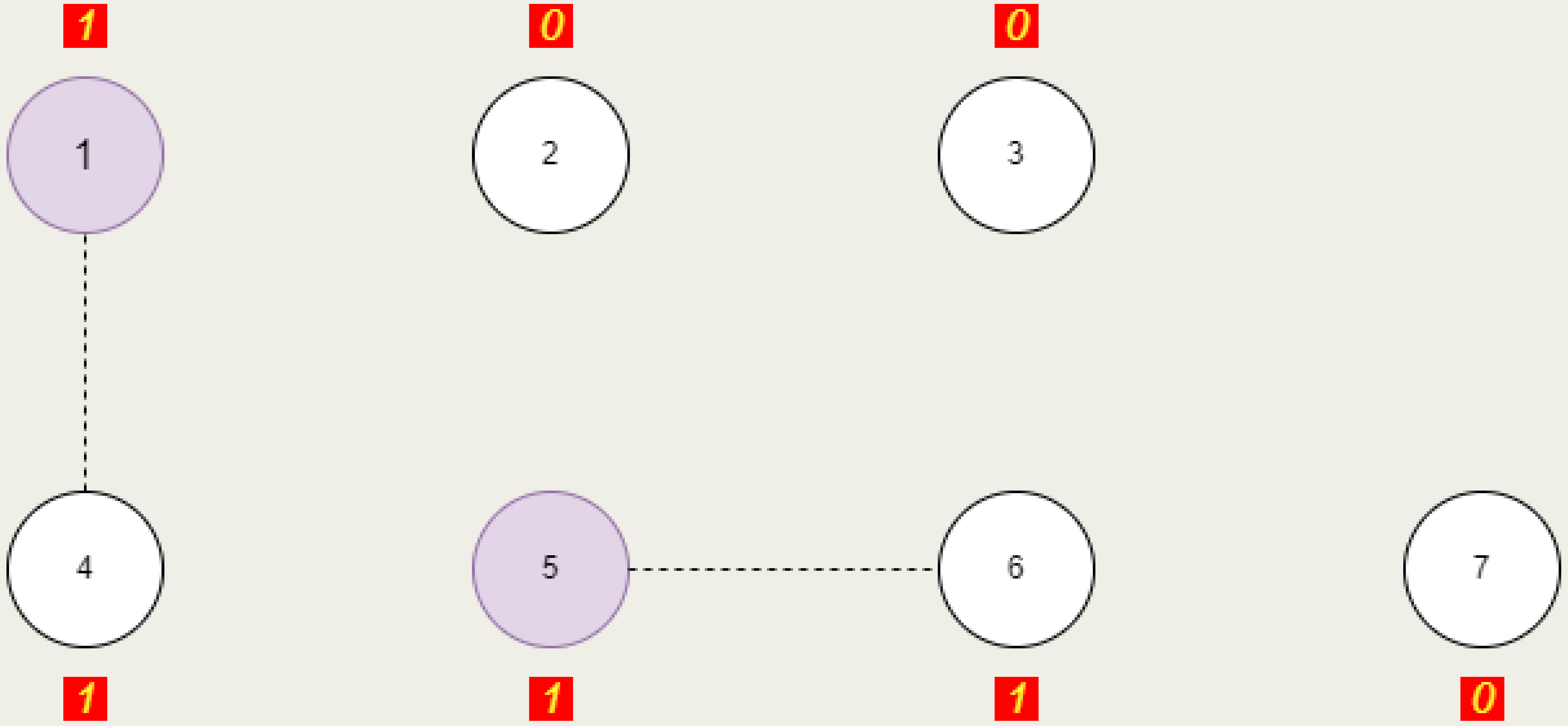


Select the Vertex with the highest degree and remove all the edges corresponding to it

USING GREEDY APPROACH - EXAMPLE

vertex_cover = [3, 2, 1, 5]

No. of Edges = 0



Do this until there are no edges in the graph

USING GREEDY APPROACH - LIMITATIONS

- **Sensitivity to vertex ordering:** The order in which vertices are considered by the greedy algorithm can affect the size of the vertex cover it finds. Choosing a different order may lead to different results.

- **Not guaranteed to be optimal:** The greedy algorithm does not guarantee an optimal solution. It may find a vertex cover that is larger than the minimum possible size (i.e., not a minimum vertex cover). It is not known how far the size of the solution produced by the greedy algorithm can be from the optimal size.

- **Trial and Error:** In order to find the optimal solution, we need to keep finding the vertex cover until we get the minumum vertex cover.

COMPLEXITY AND APPROXIMATION

Time Complexity : O(E)

- The running time of an algorithm to solve the Vertex Cover Problem is directly related to the number of edges (E) in the graph.
- This means that as the number of edges in the graph grows, the time it takes to find a solution grows in a similar manner.

The ratio bound can be proved to be 2.

- The ratio bound in the Vertex Cover Problem is a guarantee that any approximation algorithm's solution will be, at most, twice as large as the optimal solution.
- So, if the best possible solution (optimal) is, for example, 10 vertices, the algorithm's solution will be no more than 20 vertices.

REAL LIFE EXAMPLE/APPLICATION

Imagine you have a set of street intersections, and you want to place security cameras to monitor all the streets. Each camera can cover the intersection it's placed at and the adjacent streets. The Vertex Cover Problem is like figuring out the minimum number of intersections where you need to put these cameras so that every street is monitored.

It is significant in optimization and has wide applications in network design, resource allocation, and problem-solving, aiding in efficient resource utilization and decision-making.

Thank you!