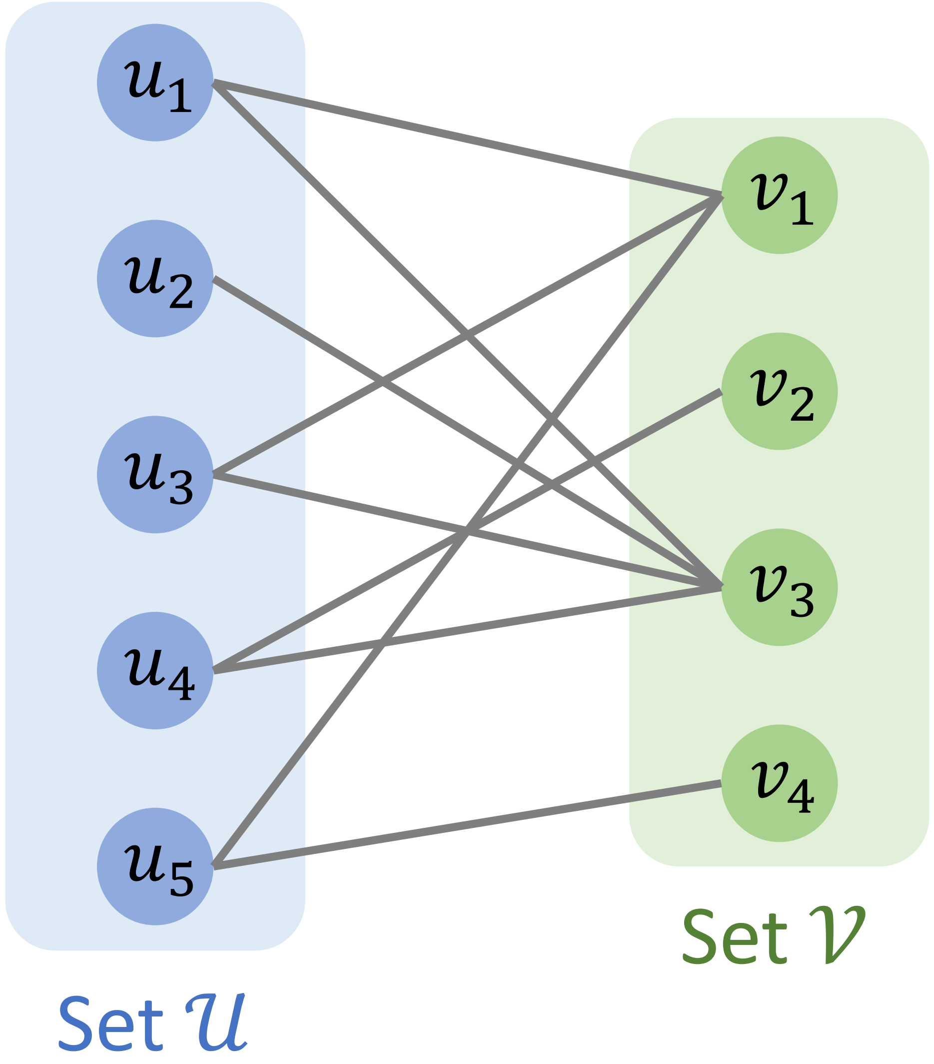
Bipartite Graph

Shusen Wang

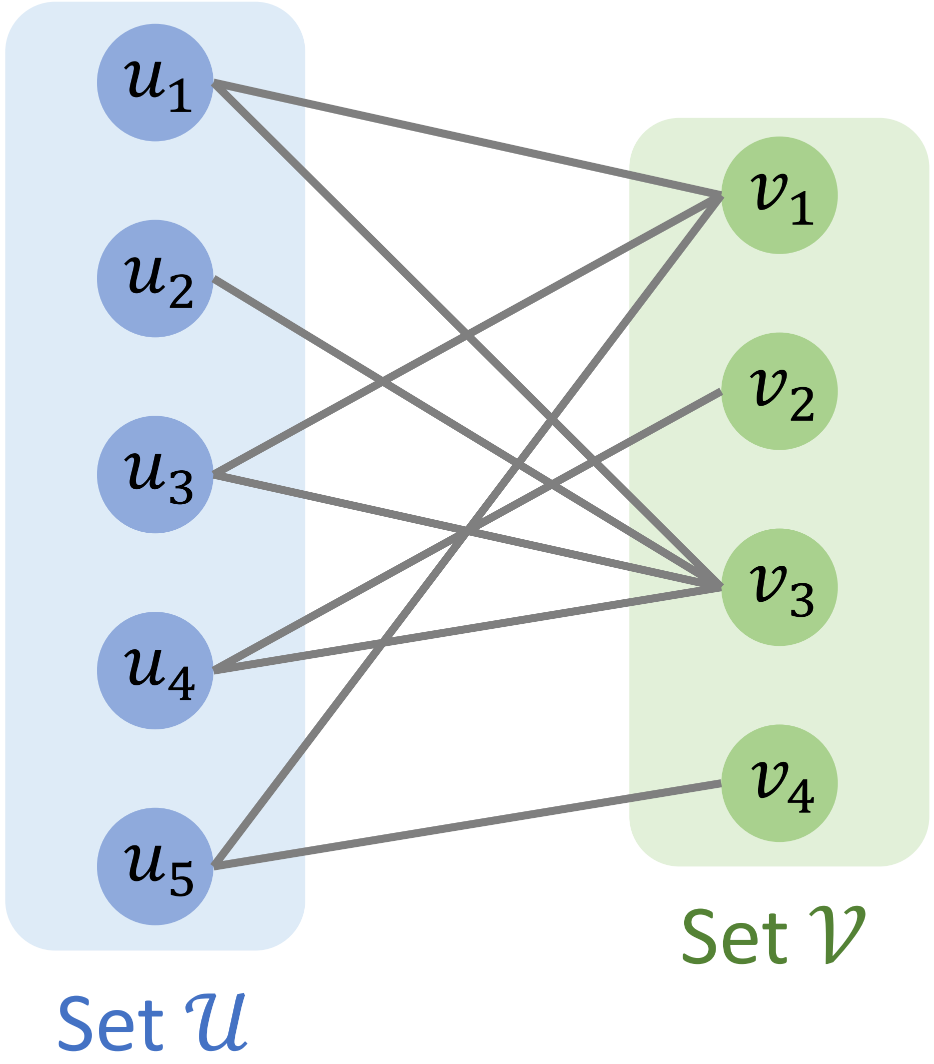
Definition

Bipartite Graph

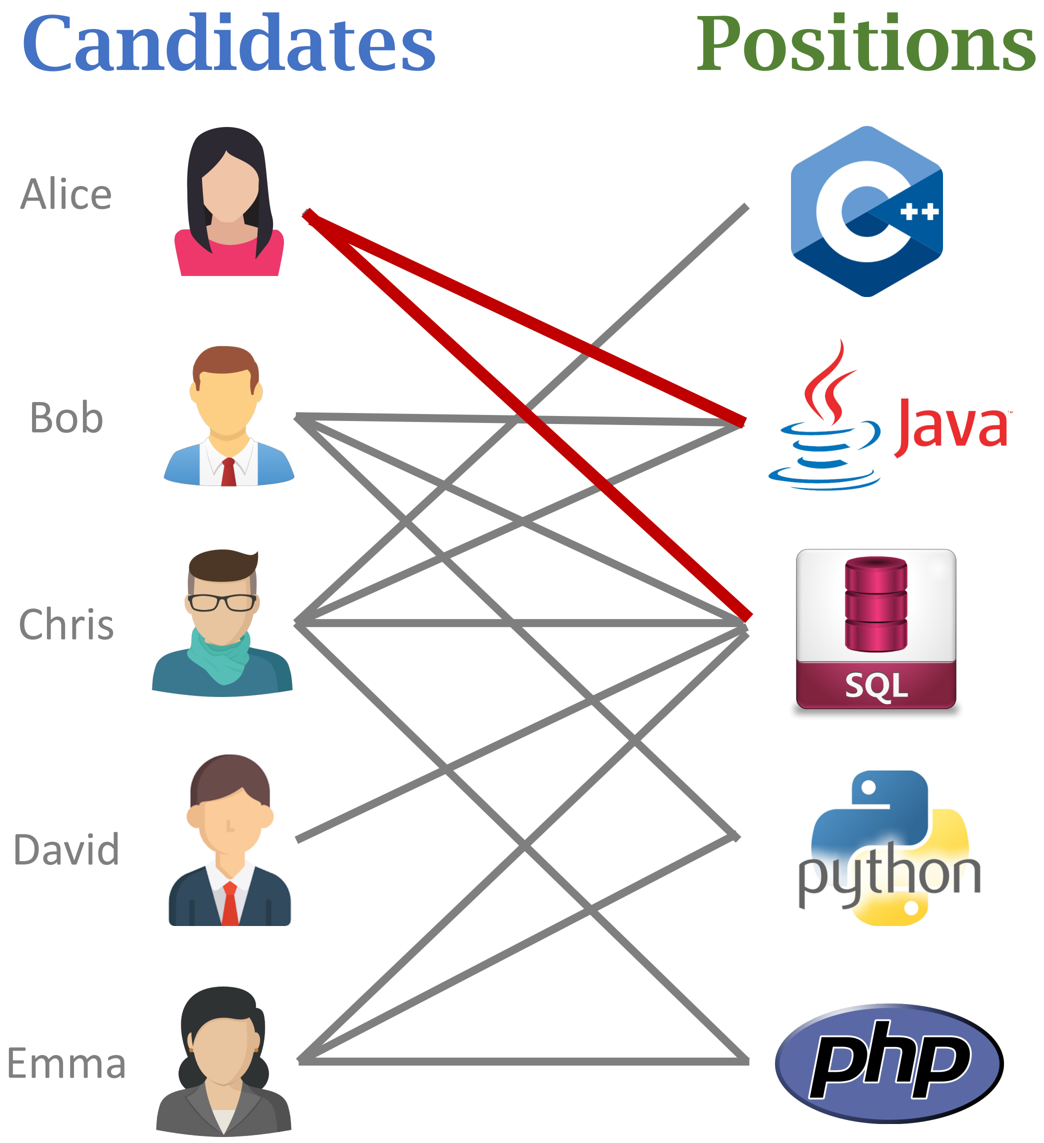


- Bipartite graph: $\mathcal{G} = (\mathcal{U}, \mathcal{V}, \mathcal{E})$.

Bipartite Graph

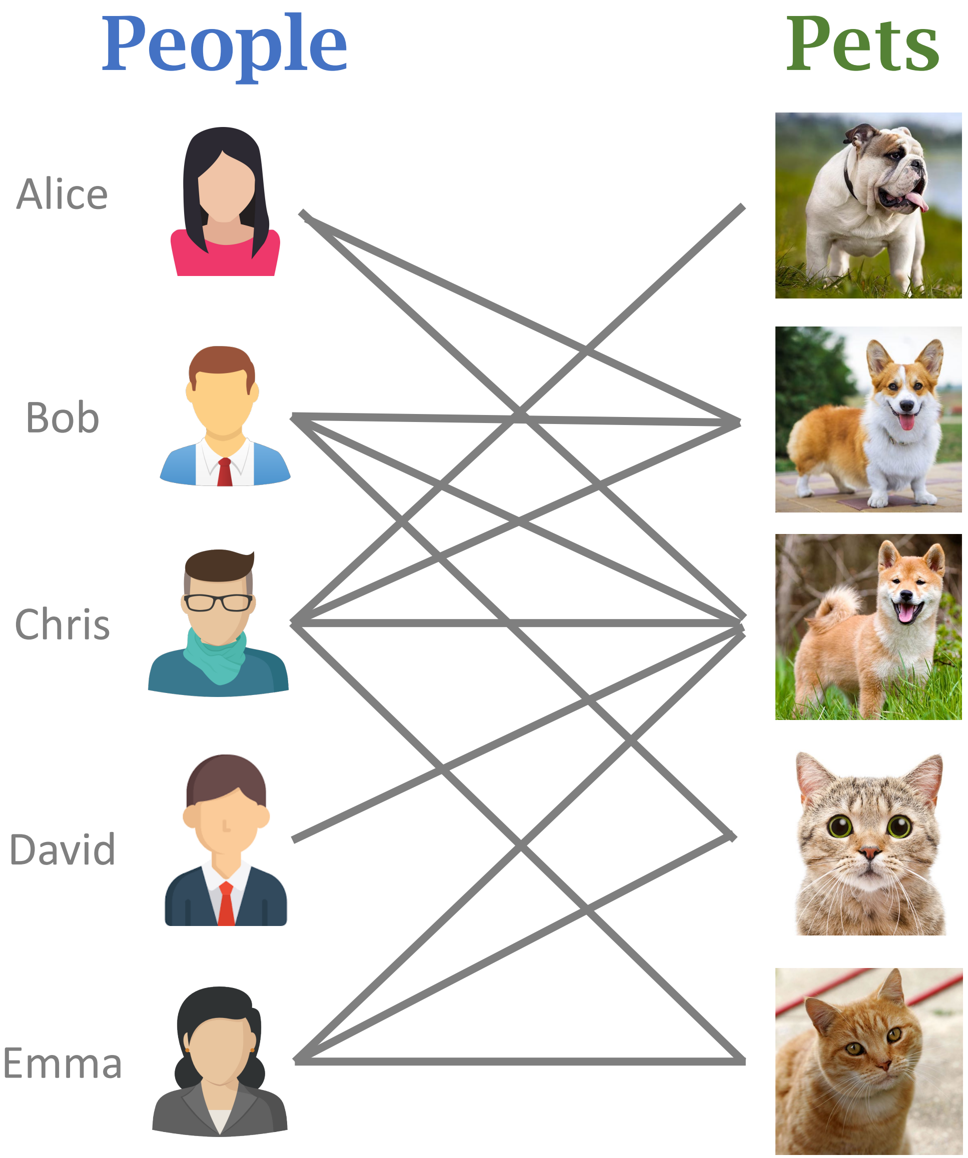


- Bipartite graph: $\mathcal{G} = (\mathcal{U}, \mathcal{V}, \mathcal{E})$.
- All the edges are between $\mathcal{U}$ and $\mathcal{V}$.
- No edge between two vertices in $\mathcal{U}$.
- No edge between two vertices in $\mathcal{V}$.



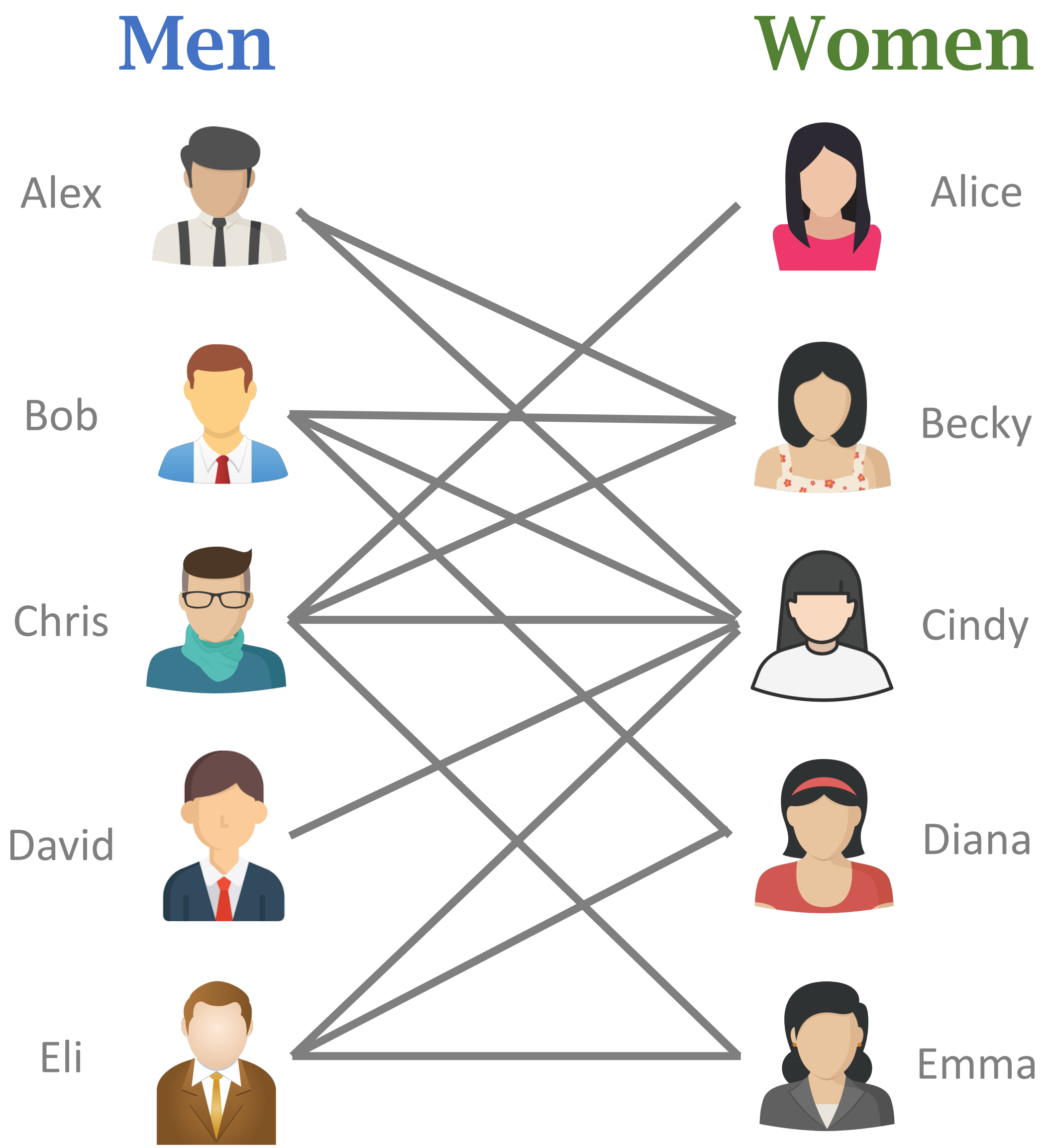
Matching candidates and positions.

- Bipartite graph: $\mathcal{G} = (\mathcal{U}, \mathcal{V}, \mathcal{E})$.
- Set $\mathcal{U}$ contains candidates.
- Set $\mathcal{V}$ contains jobs.
- Edges in $\mathcal{E}$ are candidates' skills.



Pet adoption

- Bipartite graph: $\mathcal{G} = (\mathcal{U}, \mathcal{V}, \mathcal{E})$.
- Set $\mathcal{U}$ contains people.
- Set $\mathcal{V}$ contains pets.
- Edges in $\mathcal{E}$ are people's preference.

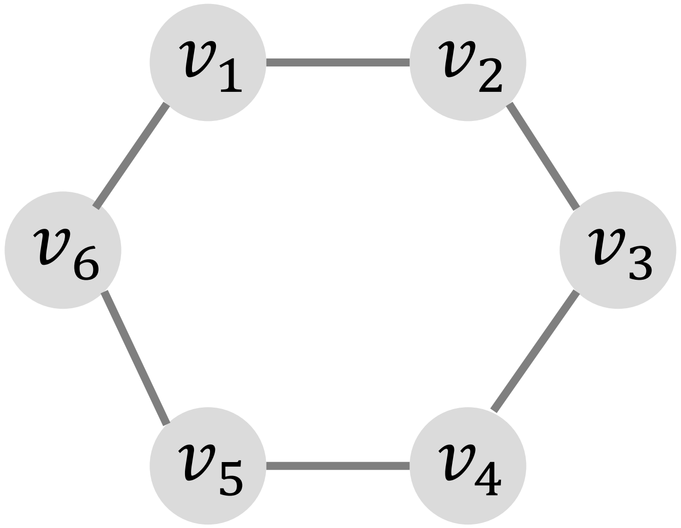


Dating

- Bipartite graph: $\mathcal{G} = (\mathcal{U}, \mathcal{V}, \mathcal{E})$.
- Set $\mathcal{U}$ contains males.
- Set $\mathcal{V}$ contains females.
- Edges in $\mathcal{E}$ are people's preference.

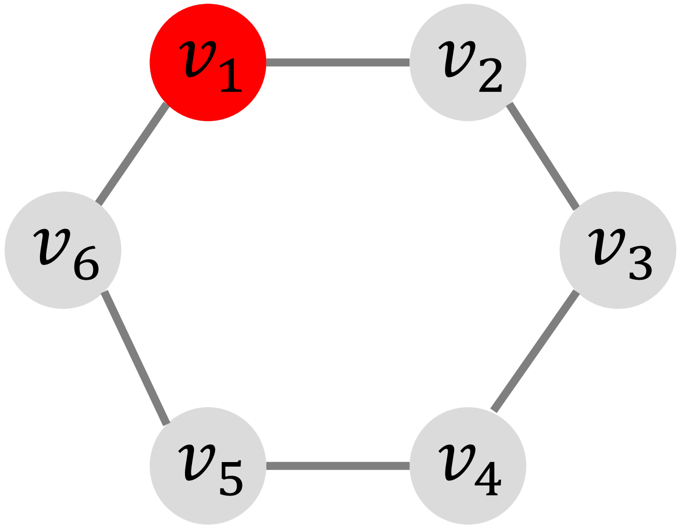
Testing Bipartiteness

Is the graph bipartite?



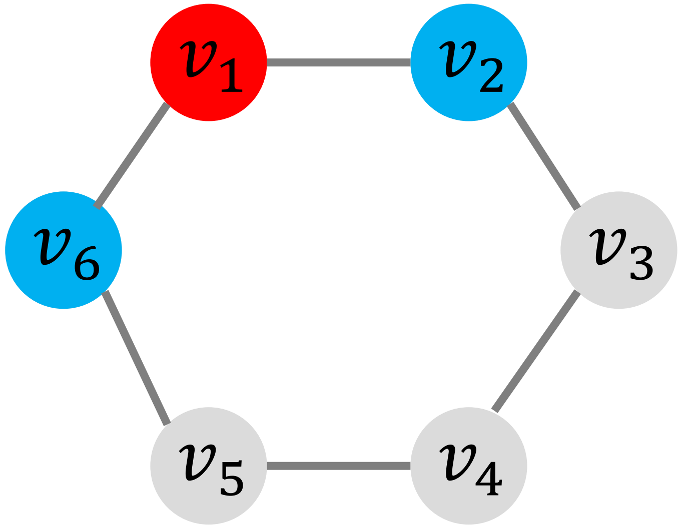
1. Select an arbitrary vertex and assign red color to it.
2. Repeat until all vertices are colored:
   - Color red vertices' neighbors as blue.
   - Color blue vertices' neighbors as red.
   - During the process, if a vertex has the same color as its neighbor, then output FALSE.
3. If no violation is found, return TRUE in the end.

Example 1



1. Select any vertex and assign red color to it.

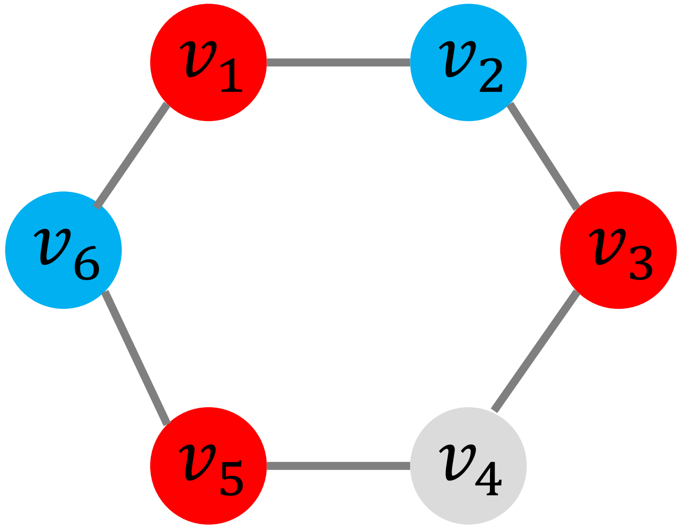
Example 1



1. Select any vertex and assign red color to it.

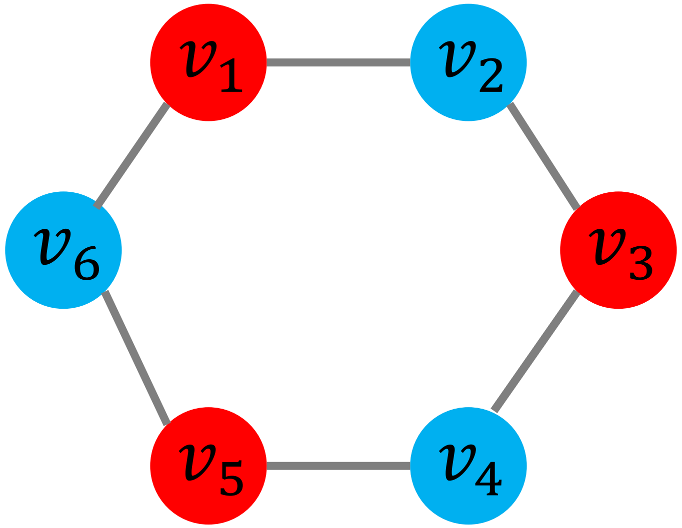
2. Color red vertices' neighbors as blue.

Example 1



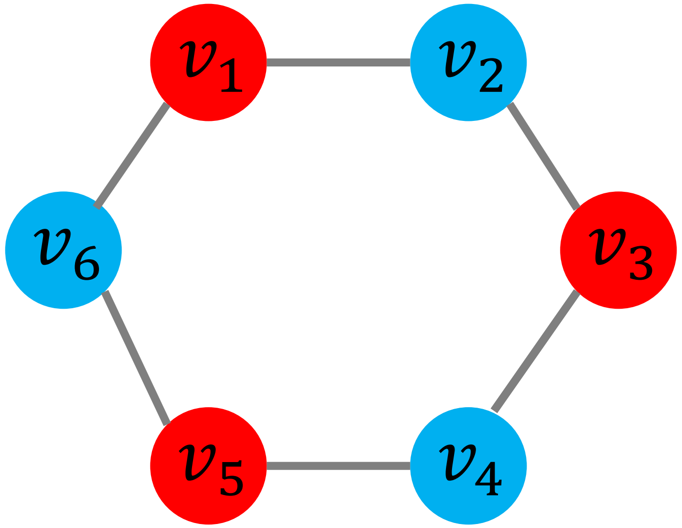
1. Select any vertex and assign red color to it.
2. Color red vertices' neighbors as blue.
3. Color blue vertices' neighbors as red.

Example 1



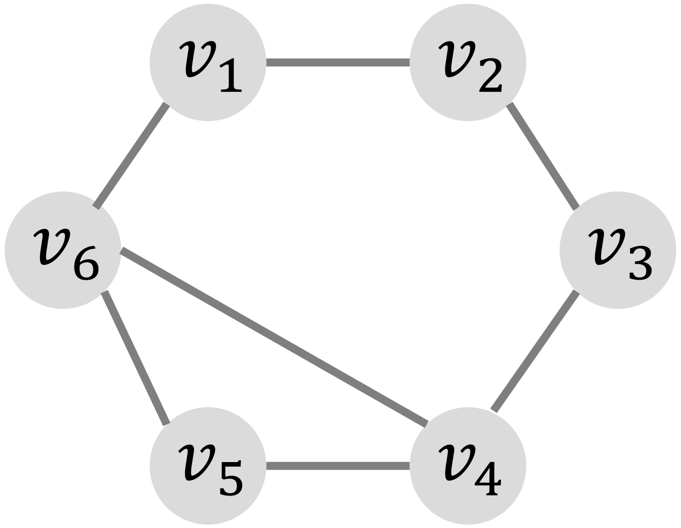
1. Select any vertex and assign red color to it.
2. Color red vertices' neighbors as blue.
3. Color blue vertices' neighbors as red.
4. Color red vertices' neighbors as blue.

Example 1

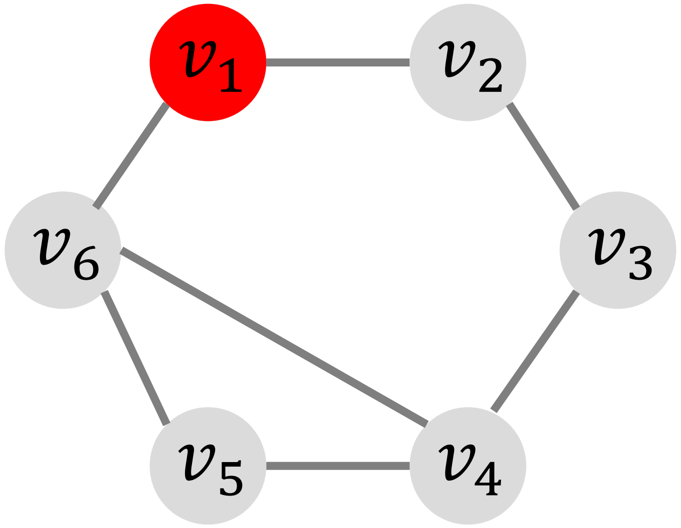


- No violation has been found!
- It is bipartite graph.

Example 2

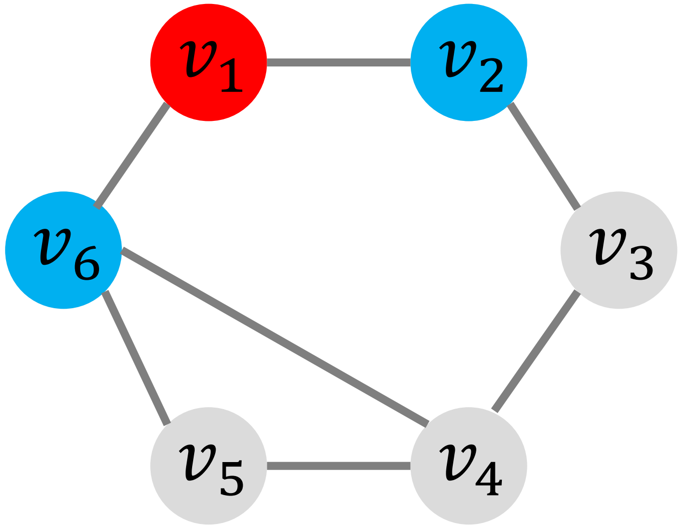


Example 2



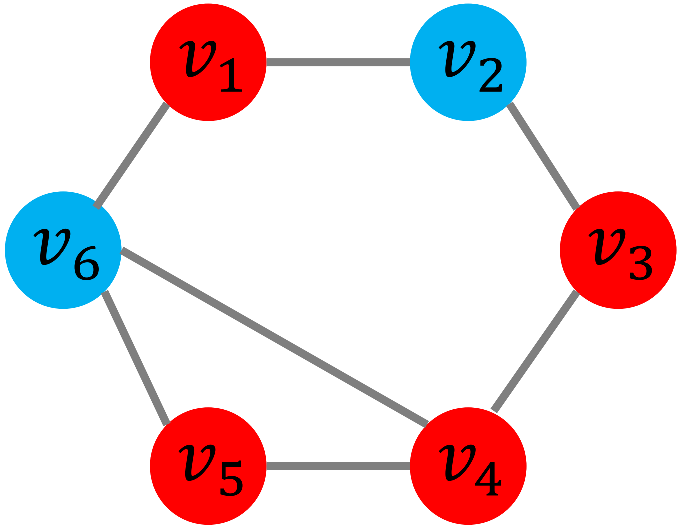
1. Select any vertex and assign red color to it.

Example 2



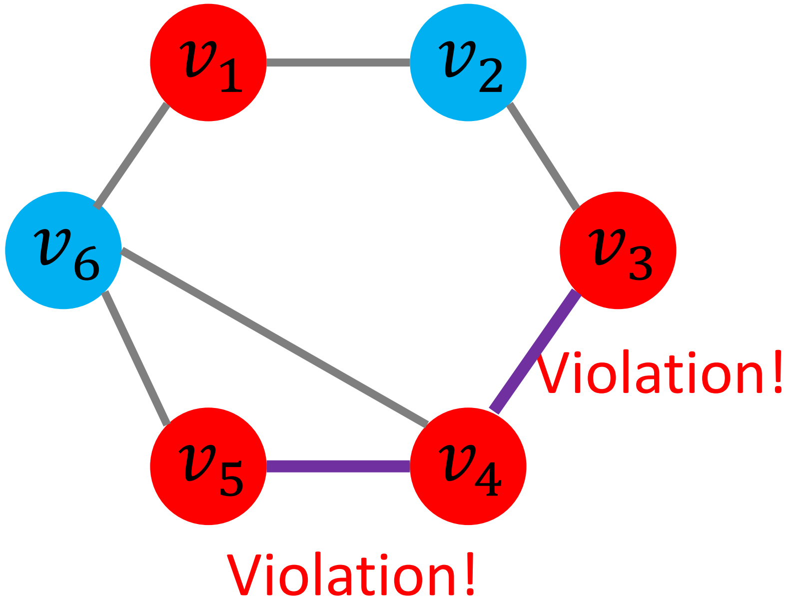
1. Select any vertex and assign red color to it.
2. Color red vertices' neighbors as blue.

Example 2



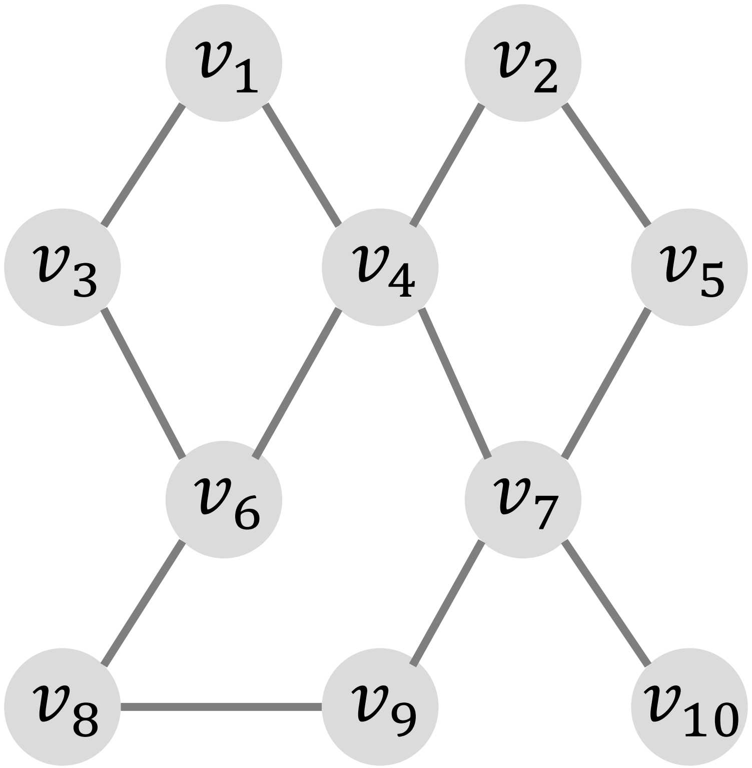
1. Select any vertex and assign red color to it.
2. Color red vertices' neighbors as blue.
3. Color blue vertices' neighbors as red.

Example 2

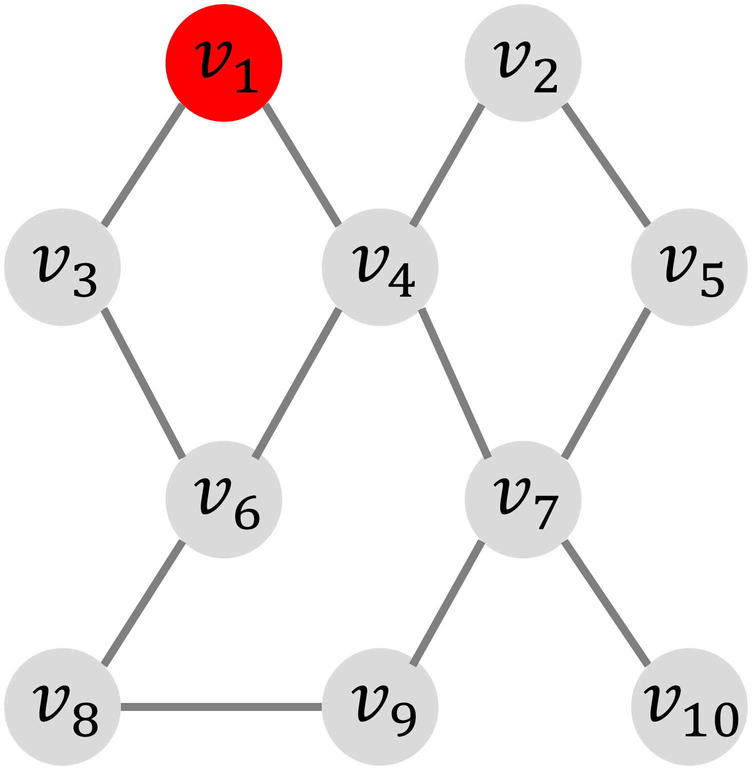


- Violation found!
- It is not bipartite graph.

Example 3

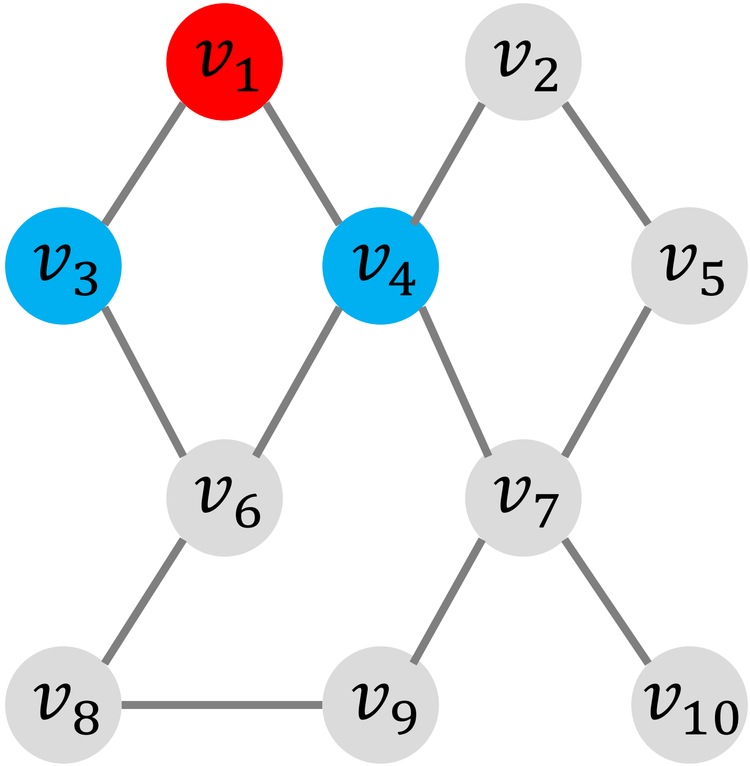


Example 3



1. Select any vertex and assign red color to it.

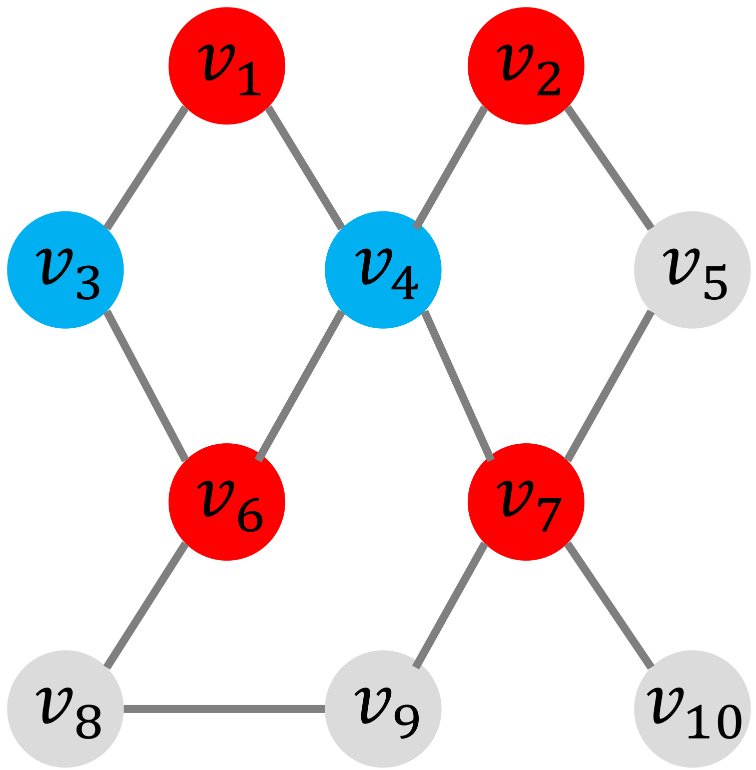
Example 3



1. Select any vertex and assign red color to it.

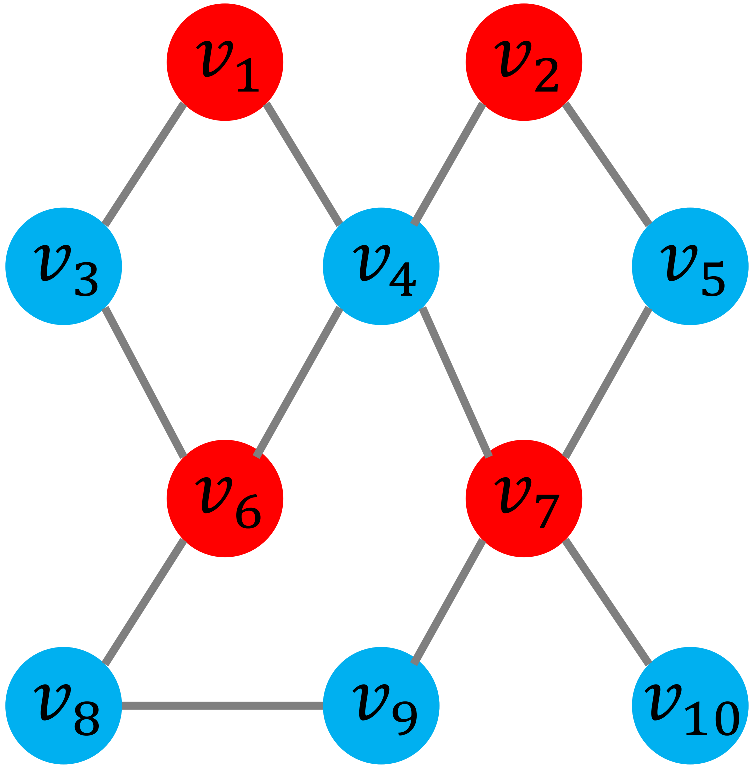
2. Color red vertices’ neighbors as blue.

Example 3



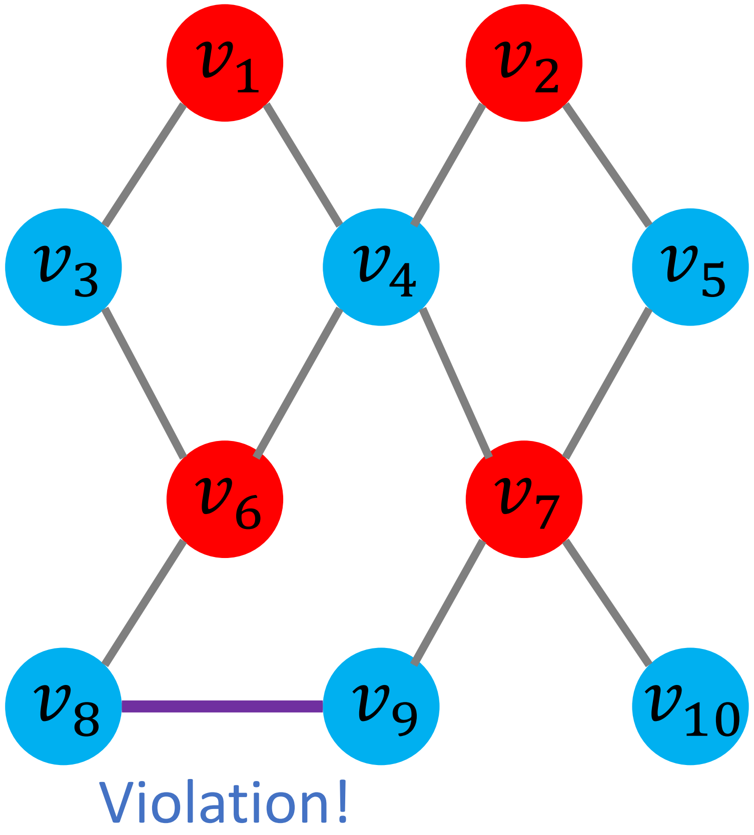
1. Select any vertex and assign red color to it.
2. Color red vertices’ neighbors as blue.
3. Color blue vertices’ neighbors as red.

Example 3



1. Select any vertex and assign red color to it.
2. Color red vertices' neighbors as blue.
3. Color blue vertices' neighbors as red.
4. Color red vertices' neighbors as blue.

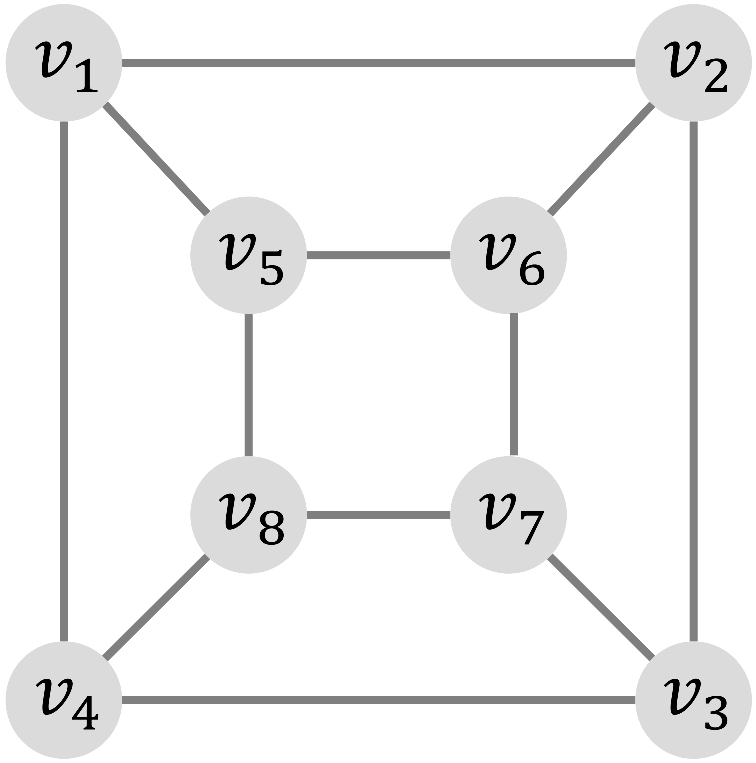
Example 3



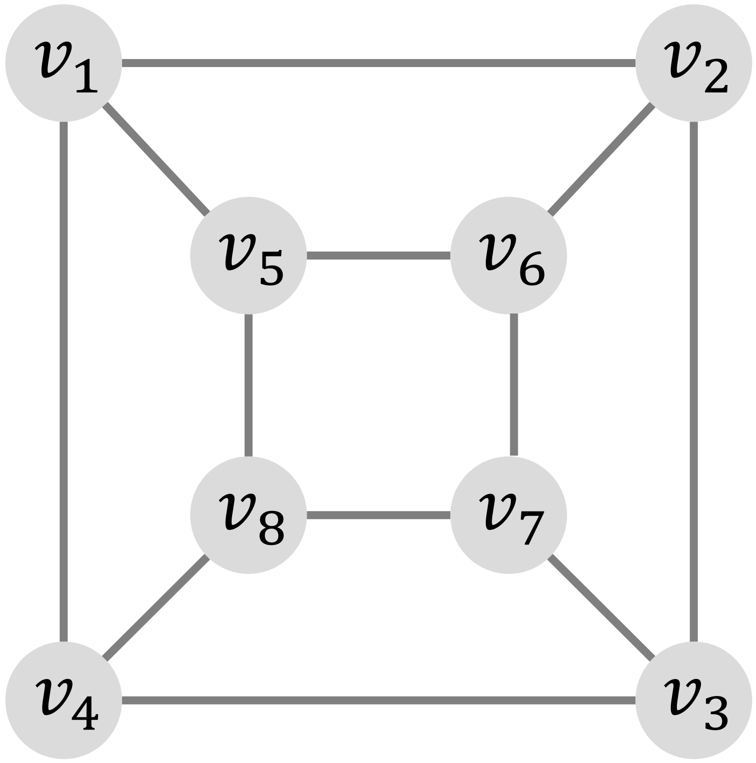
- Violation found!
- It is not bipartite graph.

Algorithm Details

Is the graph bipartite?

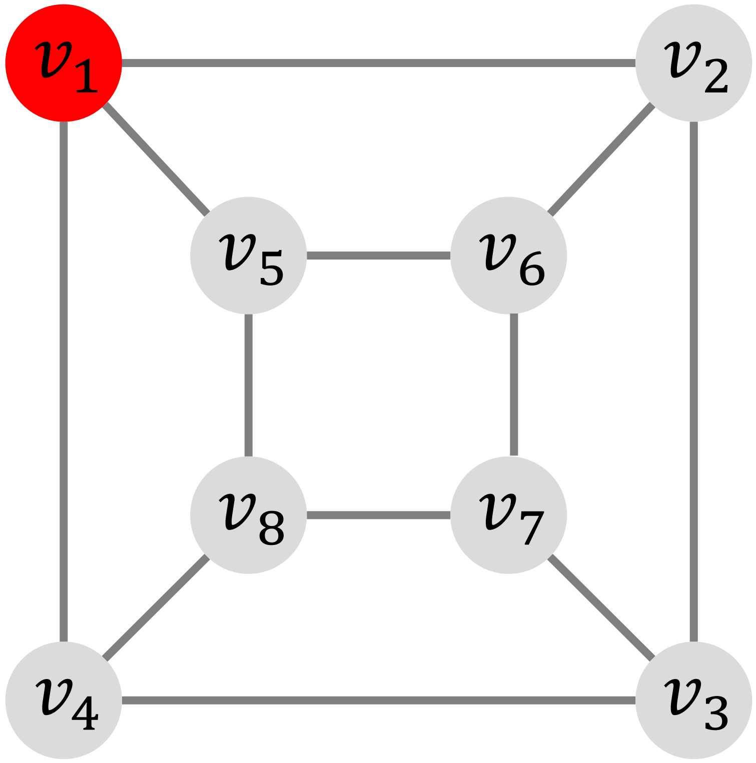


Initial State



Queue:

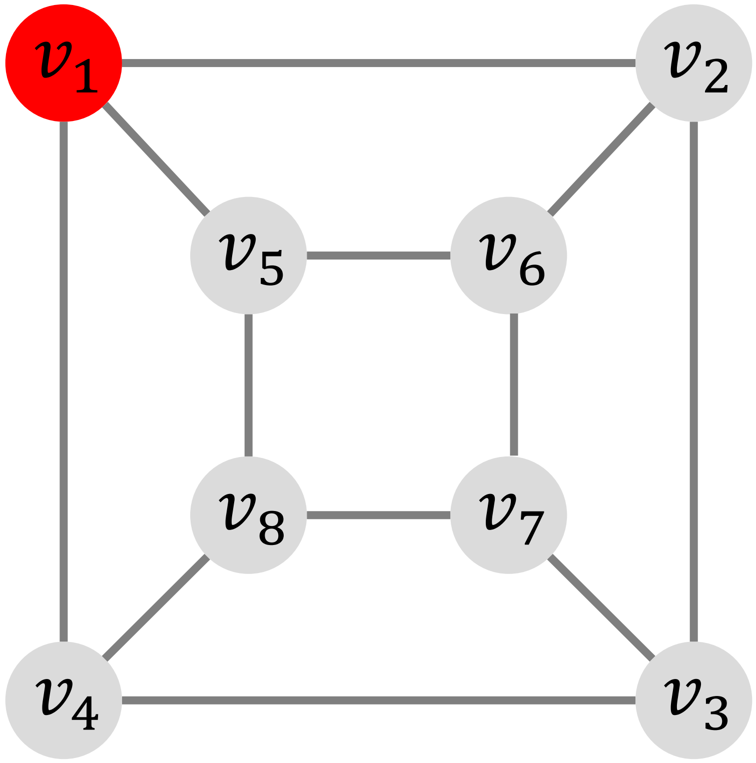
Initial State

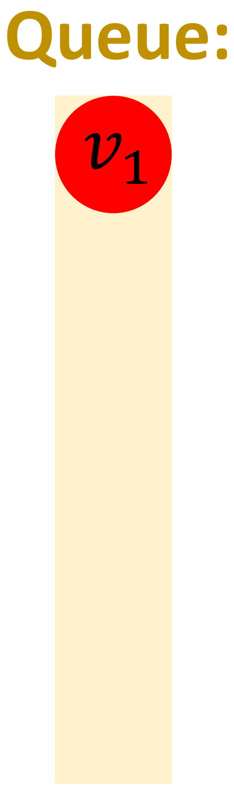


Queue:

- Assign red color to v_1.

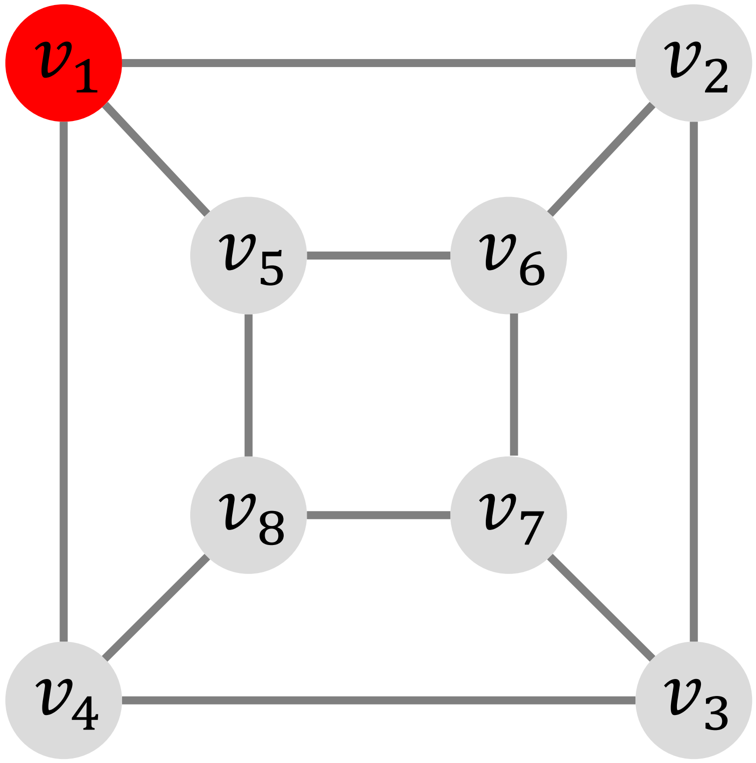
Initial State



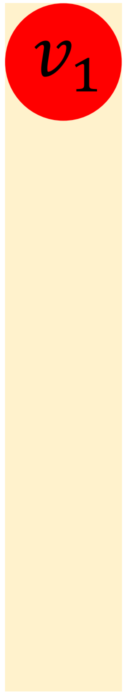


- Assign red color to v_1.
- enqueue(v_1).

Iteration 1

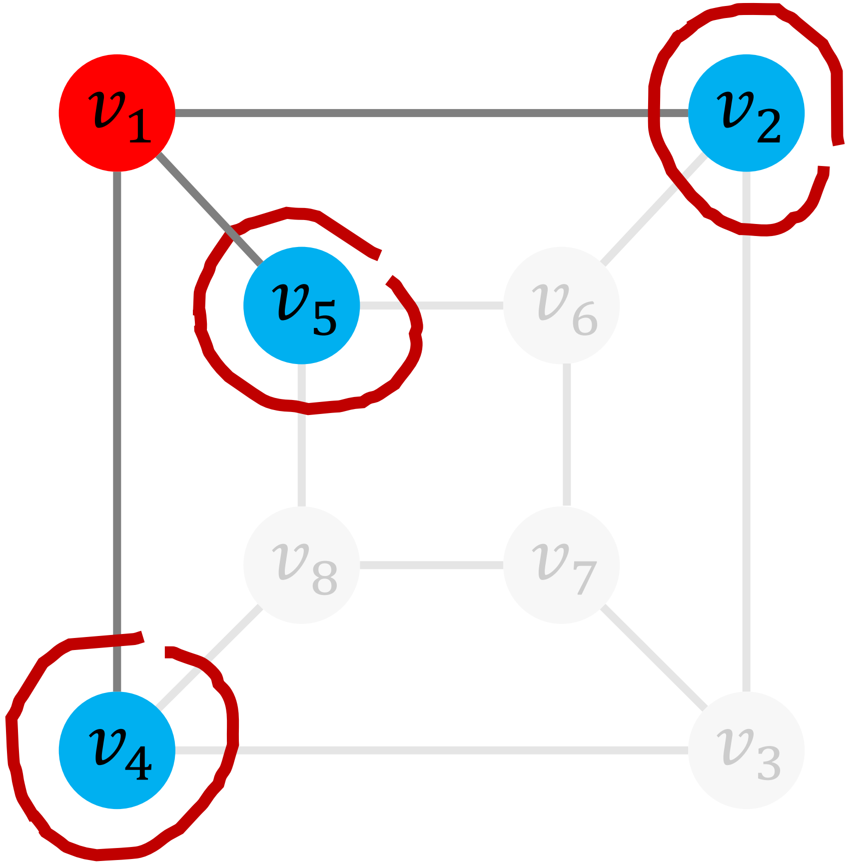


Queue:



- $v_1 \leftarrow$ dequeue( ).

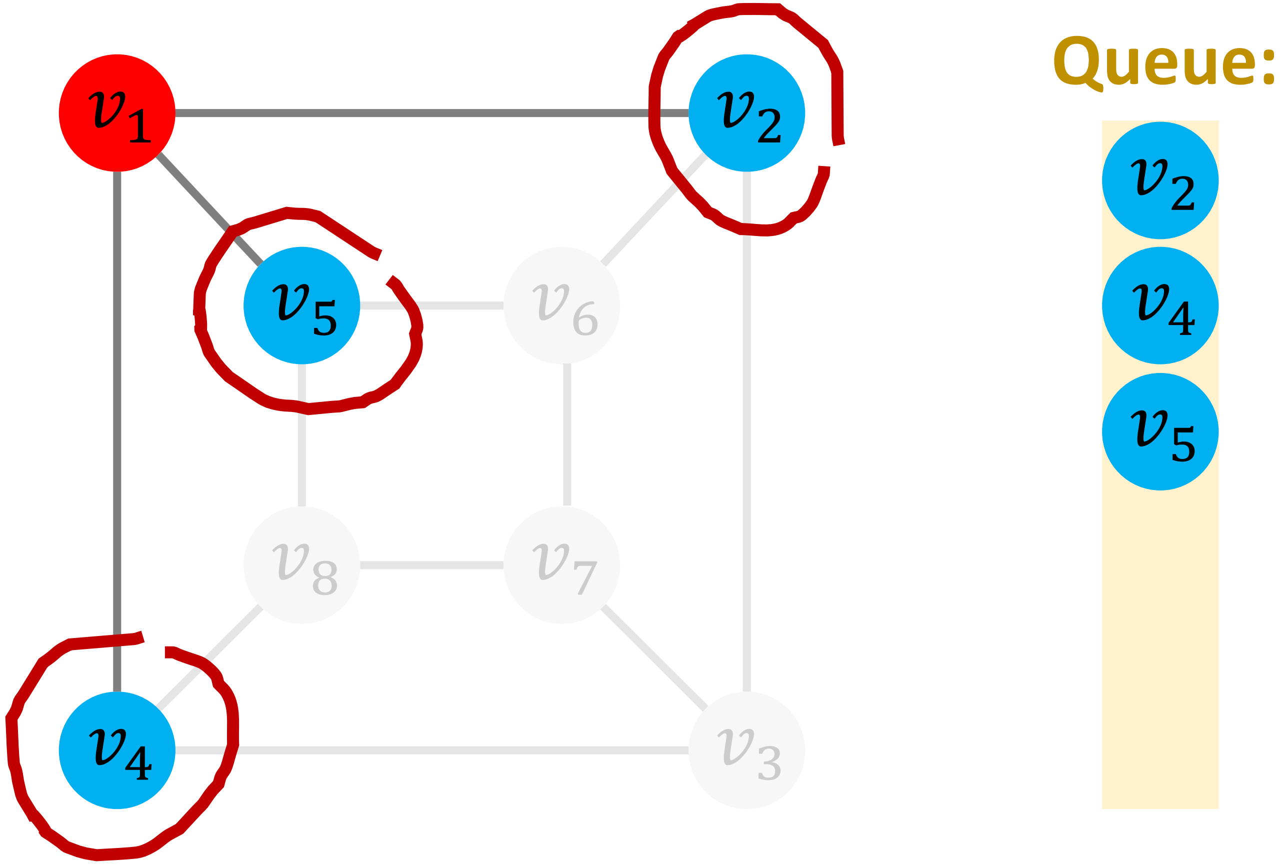
Iteration 1



Queue:

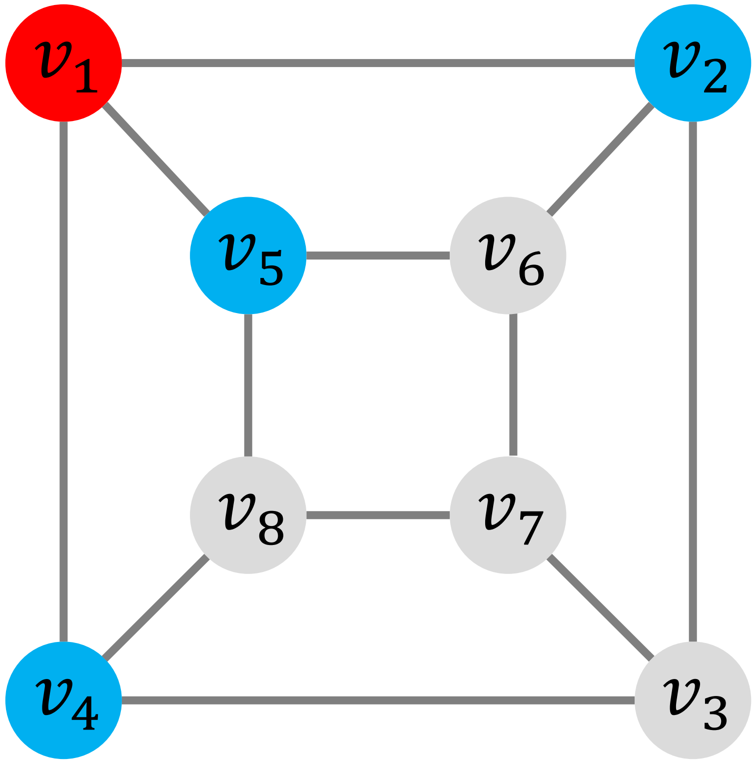
- $v_1 \leftarrow$ dequeue( ).
- Assign blue color to its unvisited neighbors, v_2, v_4, and v_5.

Iteration 1

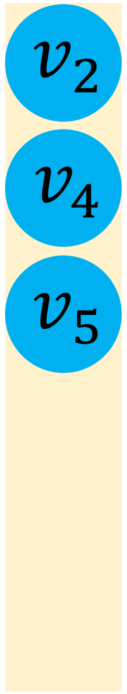


- $v_1 \leftarrow$ dequeue( ).
- Assign blue color to its unvisited neighbors, v_2, v_4, and v_5.
- Put the unvisited neighbors, v_2, v_4, and v_5, in the queue.

Iteration 2

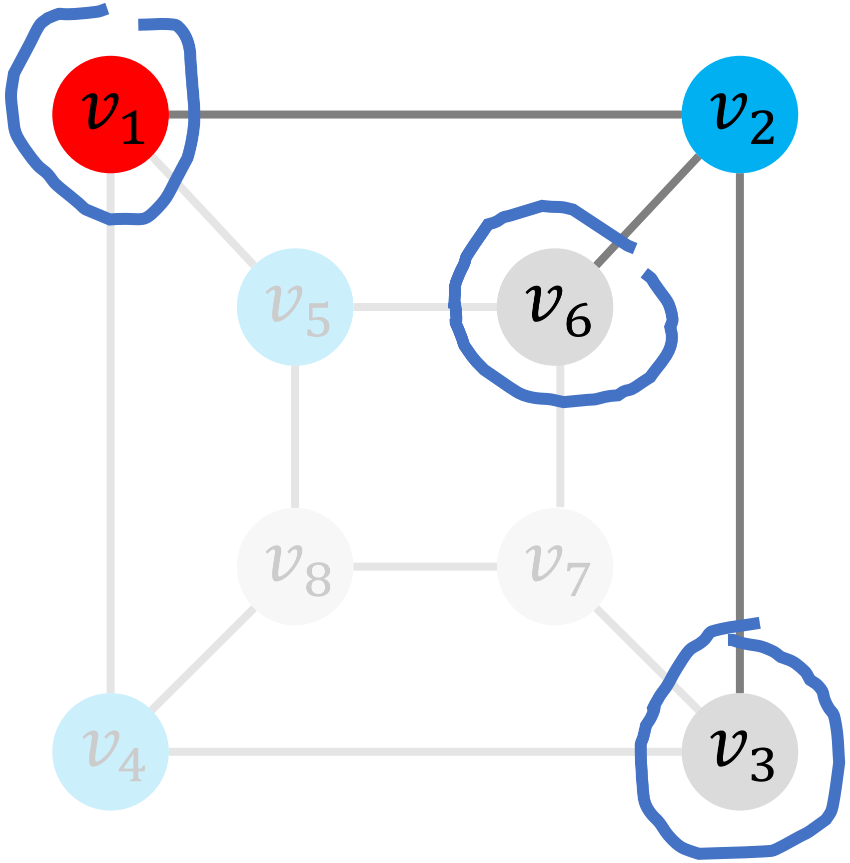


Queue:

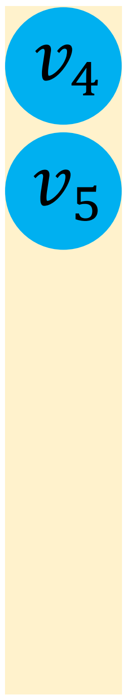


- $v_2 \leftarrow$ dequeue( ).

Iteration 2

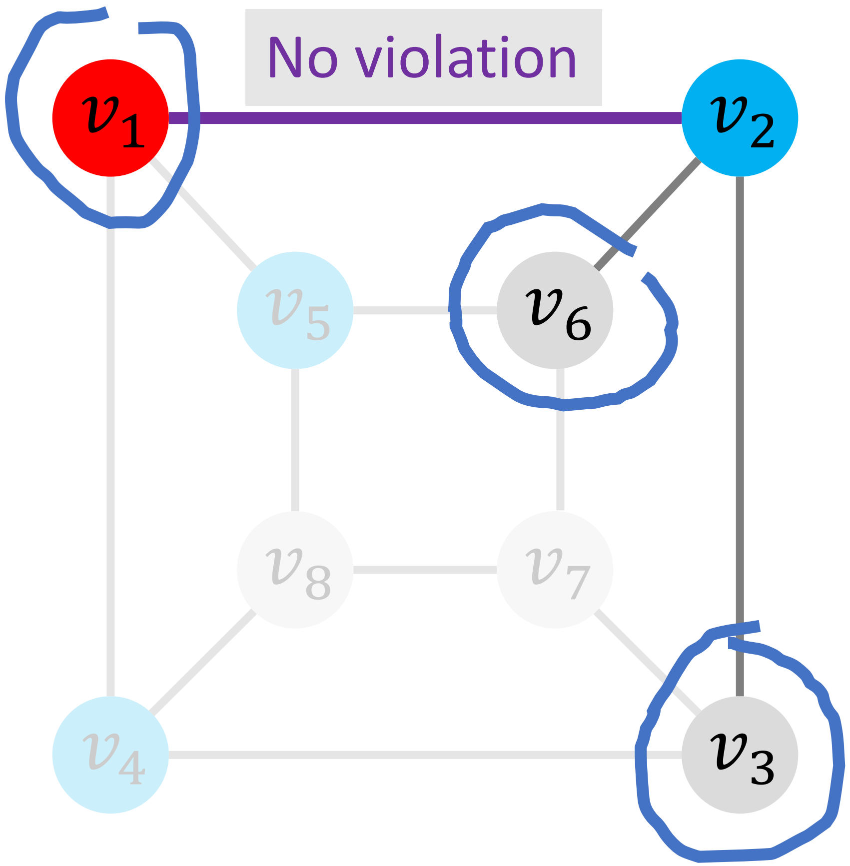


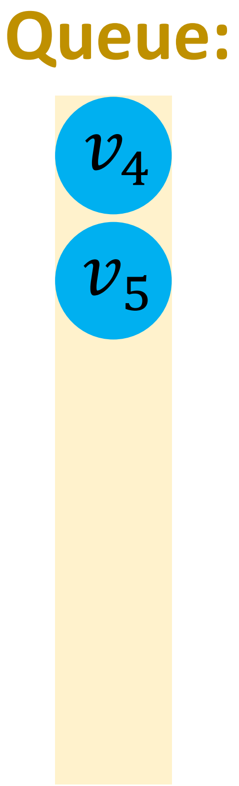
Queue:



- $v_2 \leftarrow \text{dequeue}(\,)$.

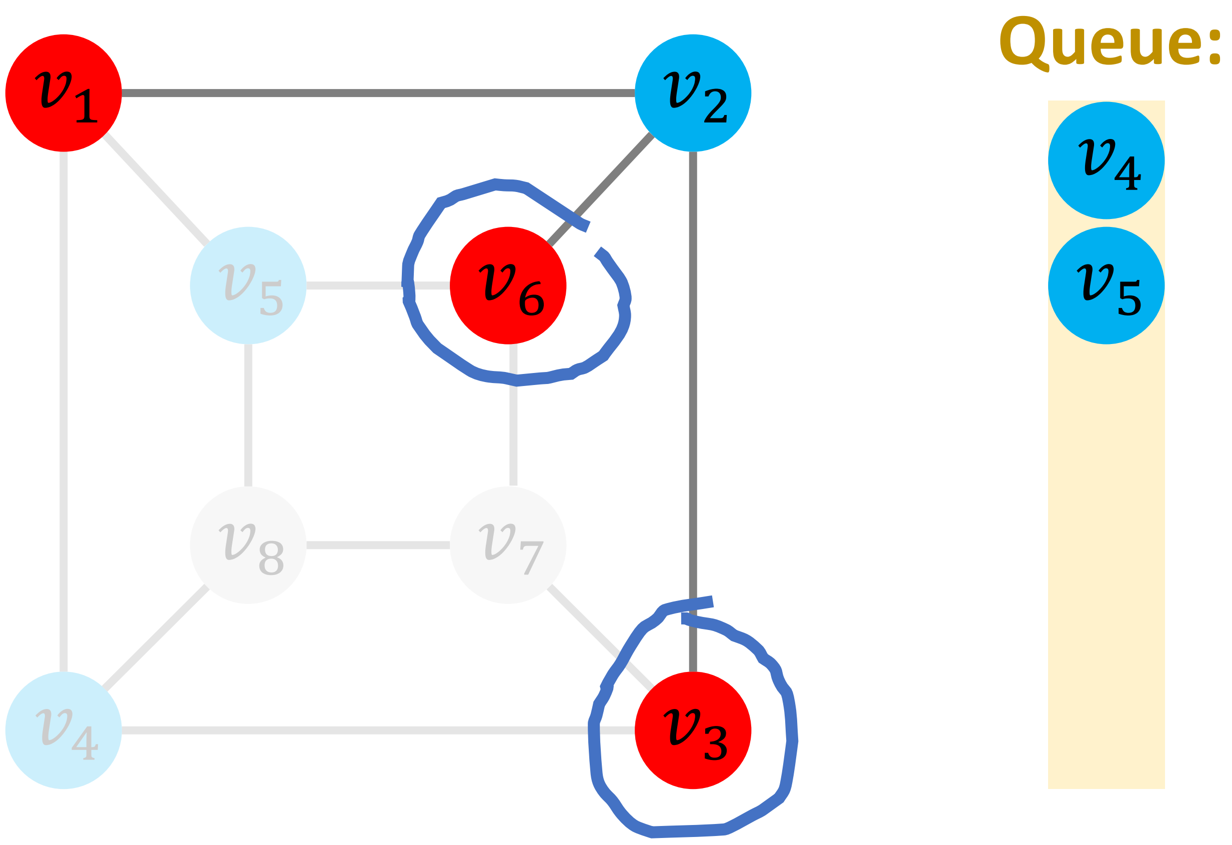
Iteration 2





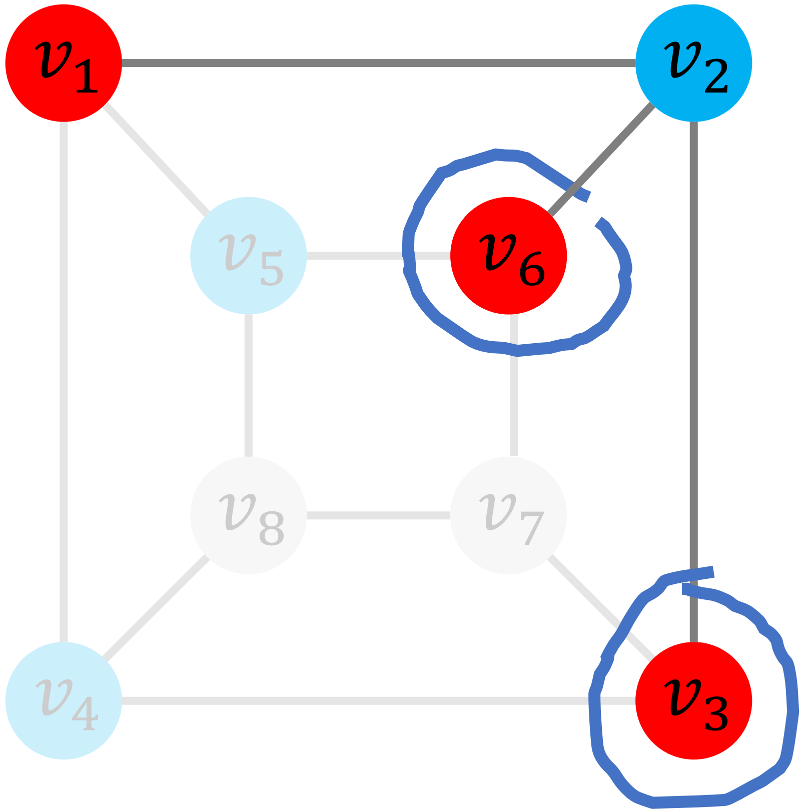
- $v_2 \leftarrow \text{dequeue}(\,)$.
- Check the visited neighbors to see if there is any violation.

Iteration 2

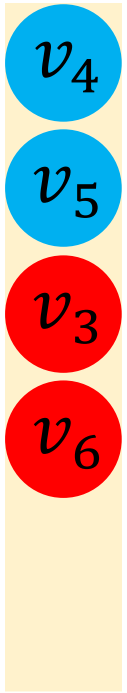


- $v_2 \leftarrow$ dequeue( ).
- Check the visited neighbors to see if there is any violation.
- Assign red color to its neighbors, v_3 and v_6.

Iteration 2

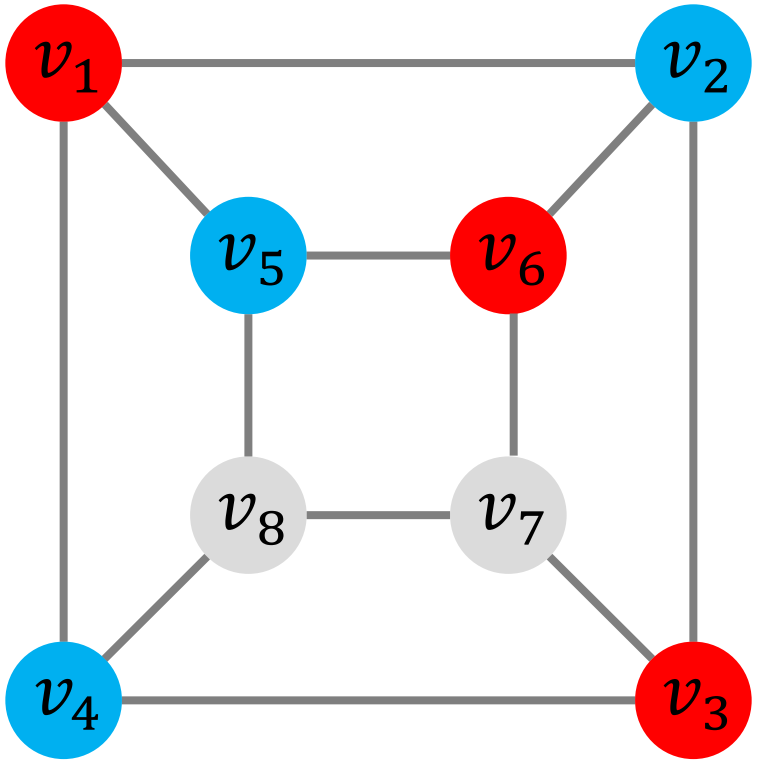


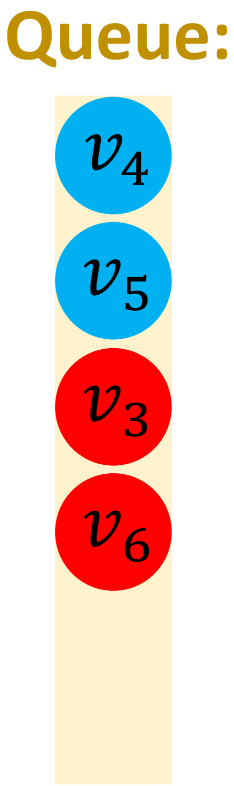
Queue:



- $v_2 \leftarrow$ dequeue( ).
- Check the visited neighbors to see if there is any violation.
- Assign red color to its neighbors, v_3 and v_6.
- Put the unvisited neighbors, v_3 and v_6, in the queue.

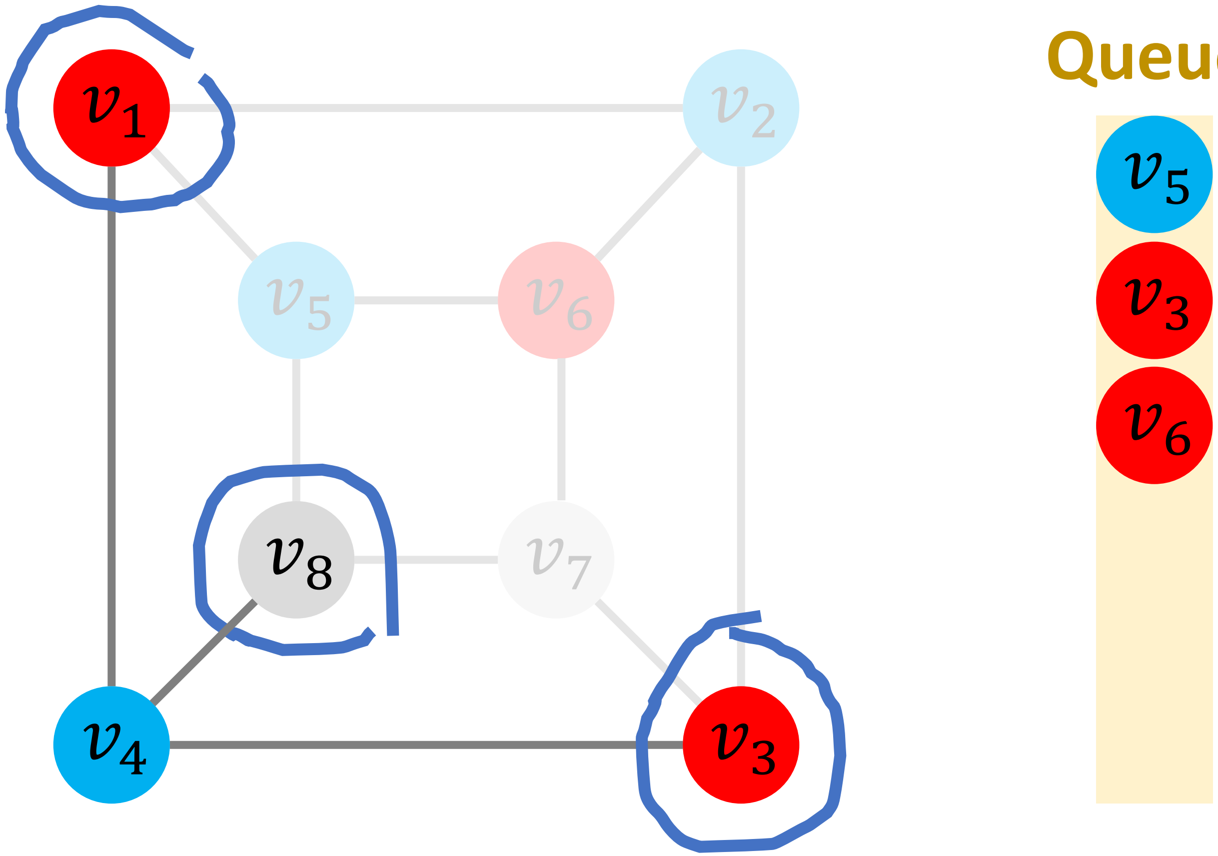
Iteration 3





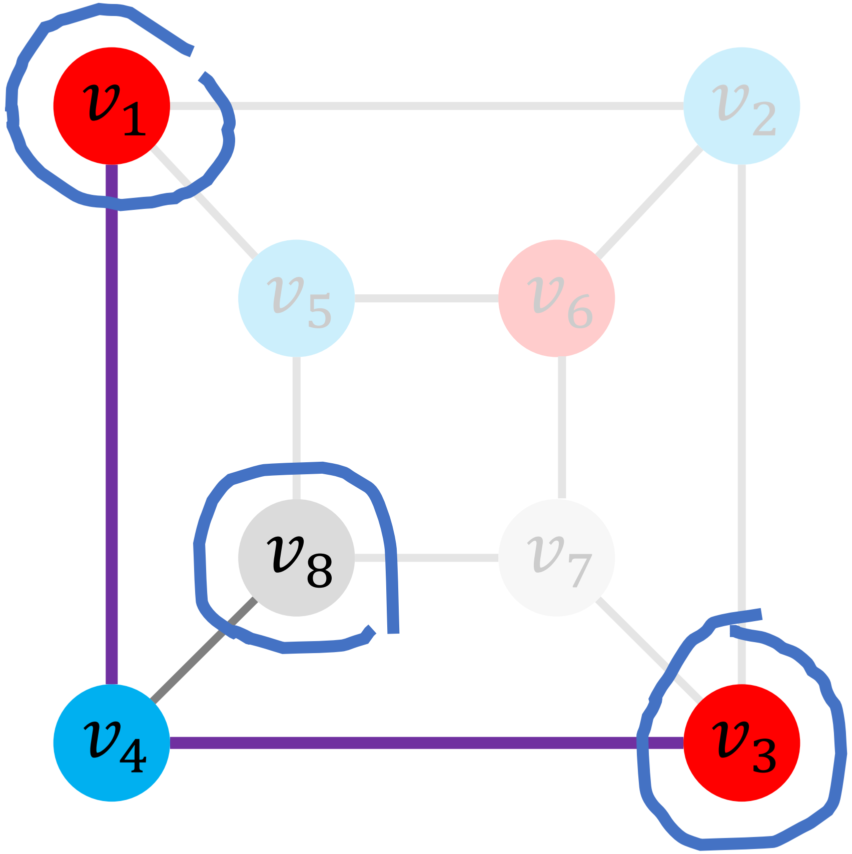
- $v_4 \leftarrow$ dequeue( ).

Iteration 3



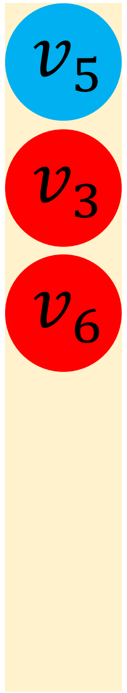
- $v_4 \leftarrow$ dequeue( ).

Iteration 3



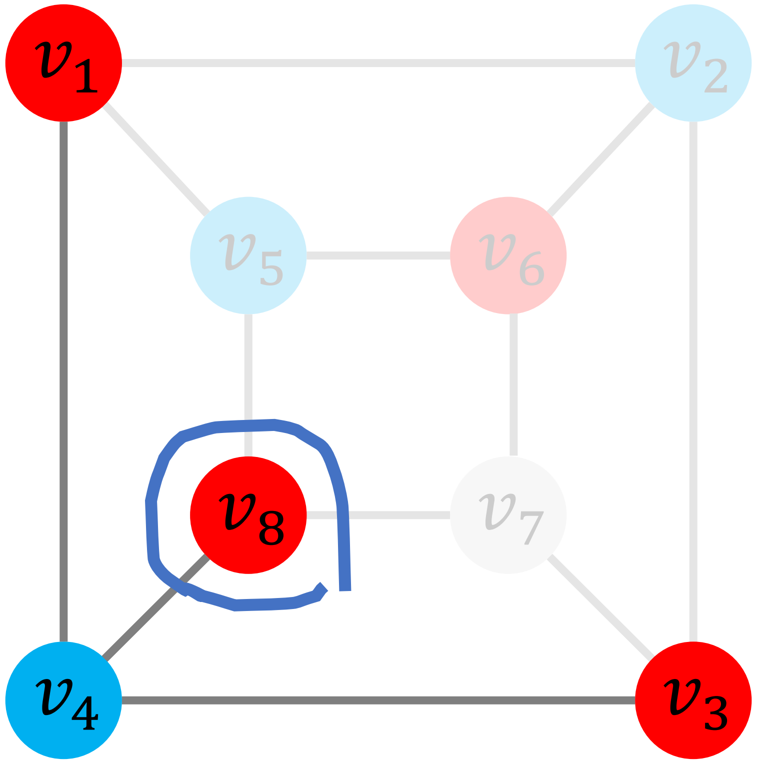
No violation

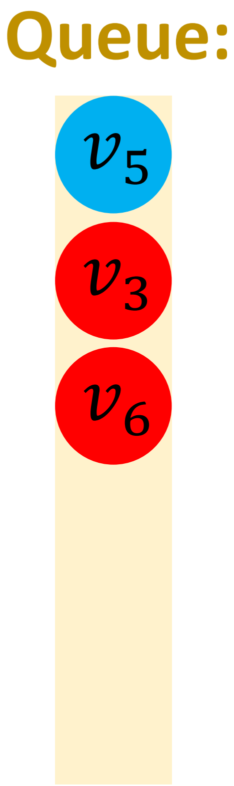
Queue:



- $v_4 \leftarrow \text{dequeue}(\,)$.
- Check the visited neighbors to see if there is any violation.

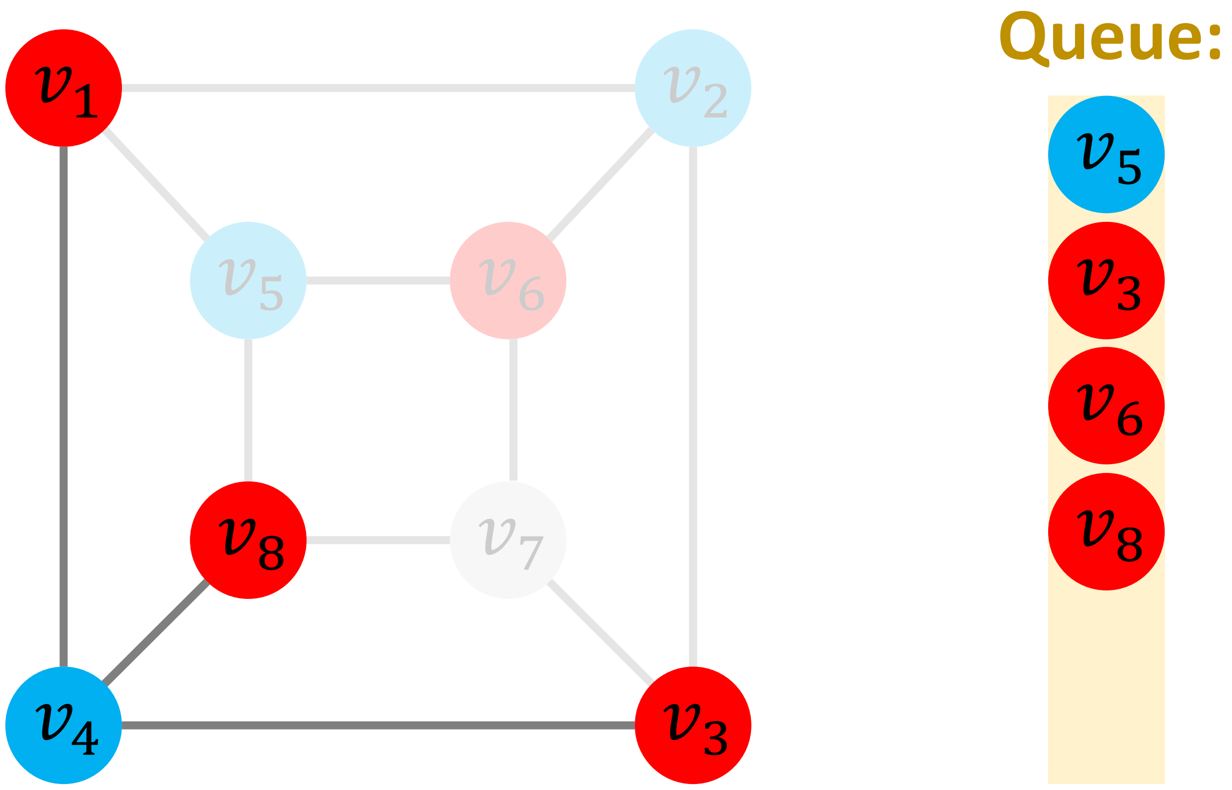
Iteration 3





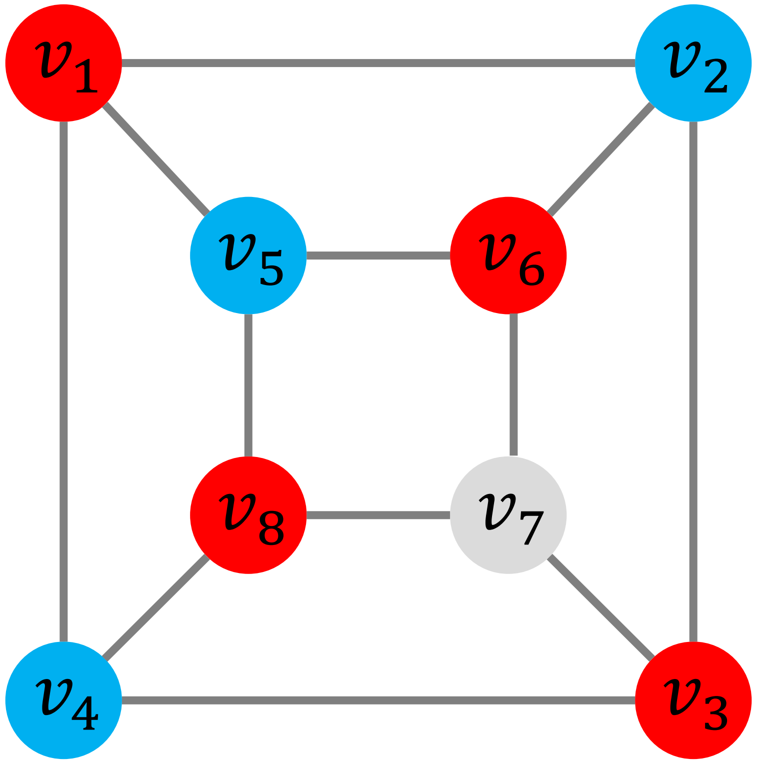
- $v_4 \leftarrow \text{dequeue}(\,)$.
- Check the visited neighbors to see if there is any violation.
- Assign red color to its neighbor, v_8.

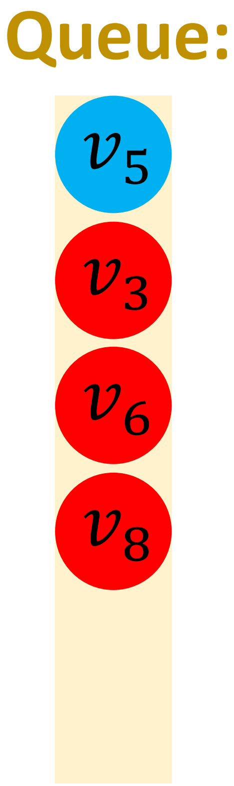
Iteration 3



- $v_4 \leftarrow$ dequeue( ).
- Check the visited neighbors to see if there is any violation.
- Assign red color to its neighbor, v_8.
- Put the unvisited neighbor, v_8, in the queue.

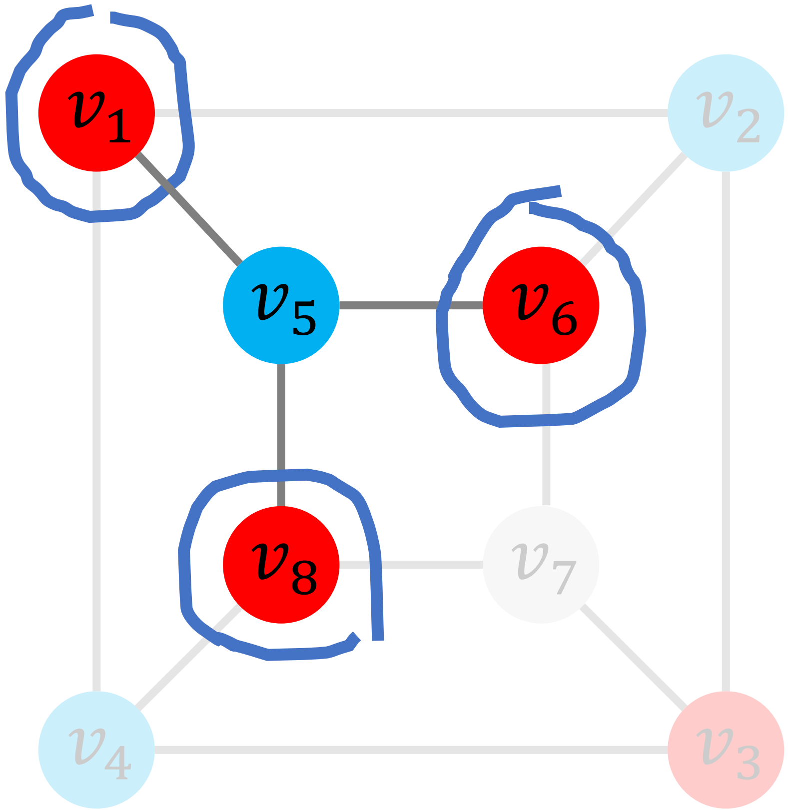
Iteration 4





- $v_5 \leftarrow$ dequeue( ).

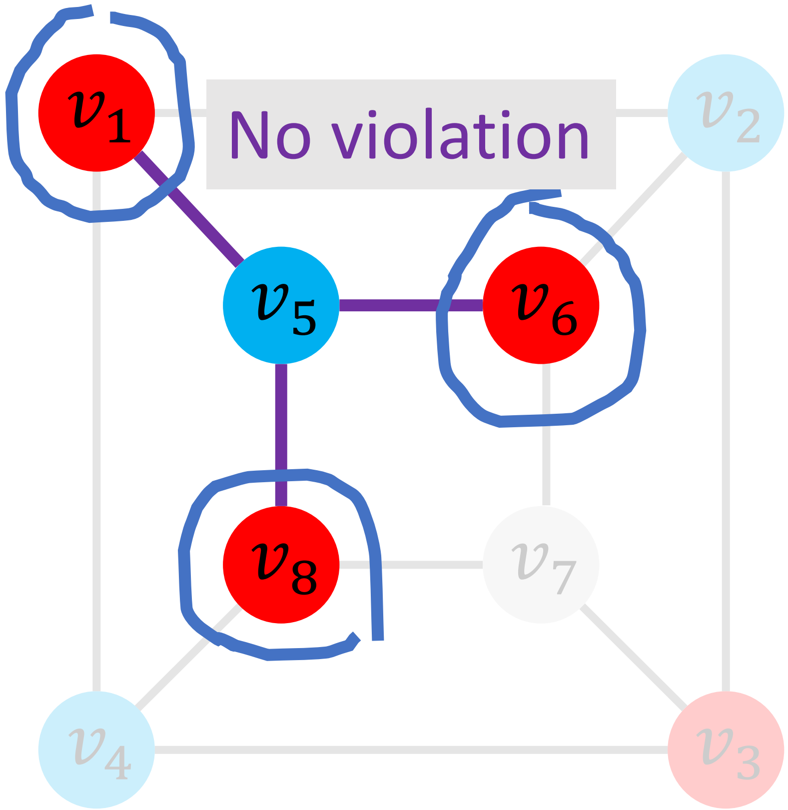
Iteration 4

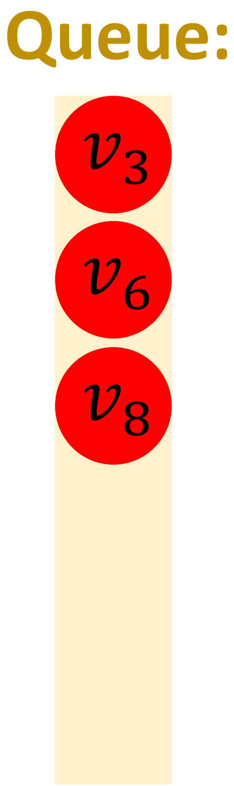


Queue:

- $v_5 \leftarrow$ dequeue( ).

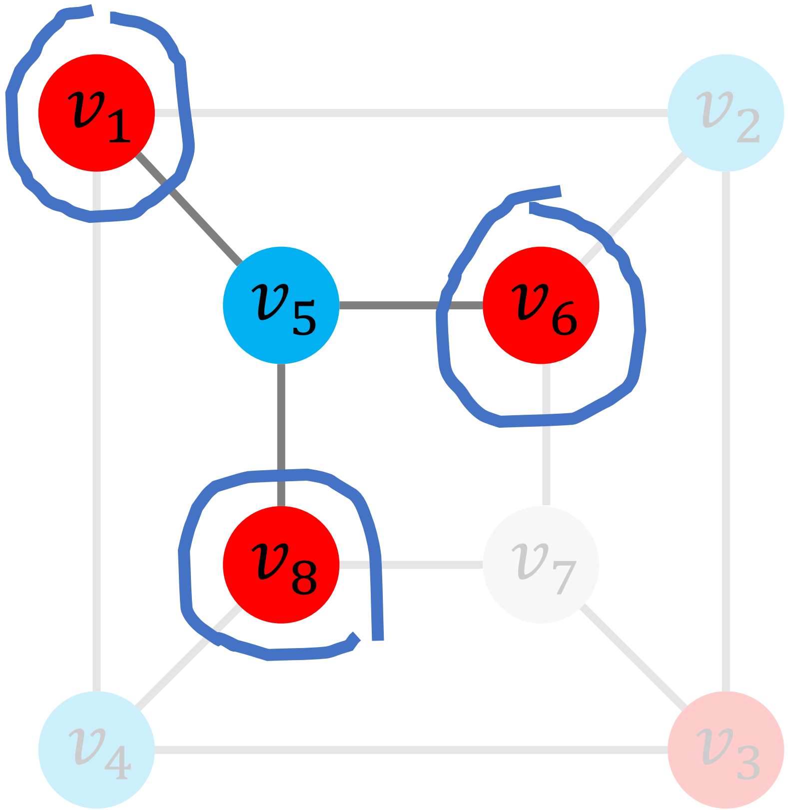
Iteration 4





- $v_5 \leftarrow$ dequeue( ).
- Check the visited neighbors to see if there is any violation.

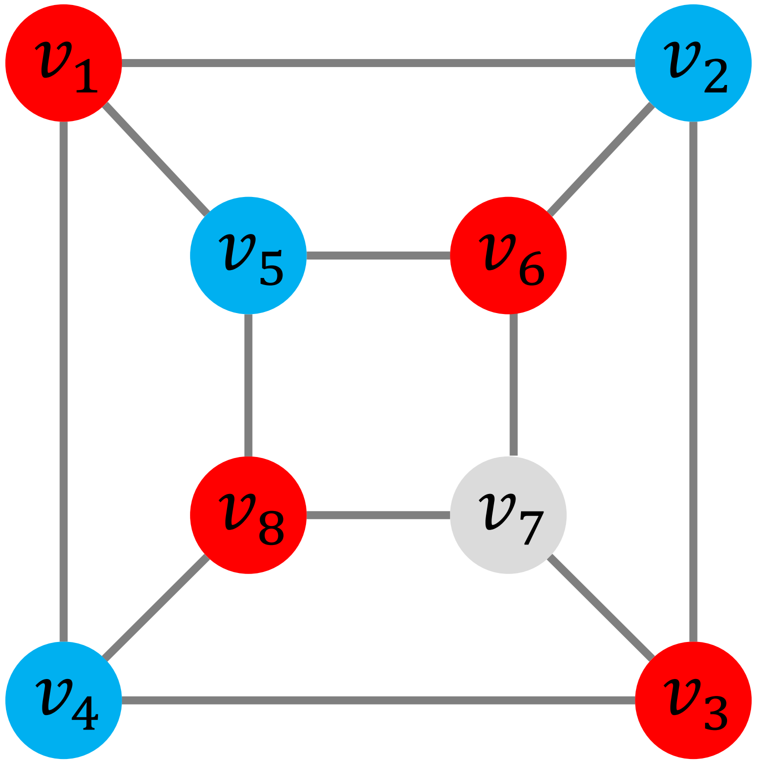
Iteration 4



Queue:

- $v_5 \leftarrow$ dequeue( ).
- Check the visited neighbors to see if there is any violation.
- Do not put visited neighbors in the queue.

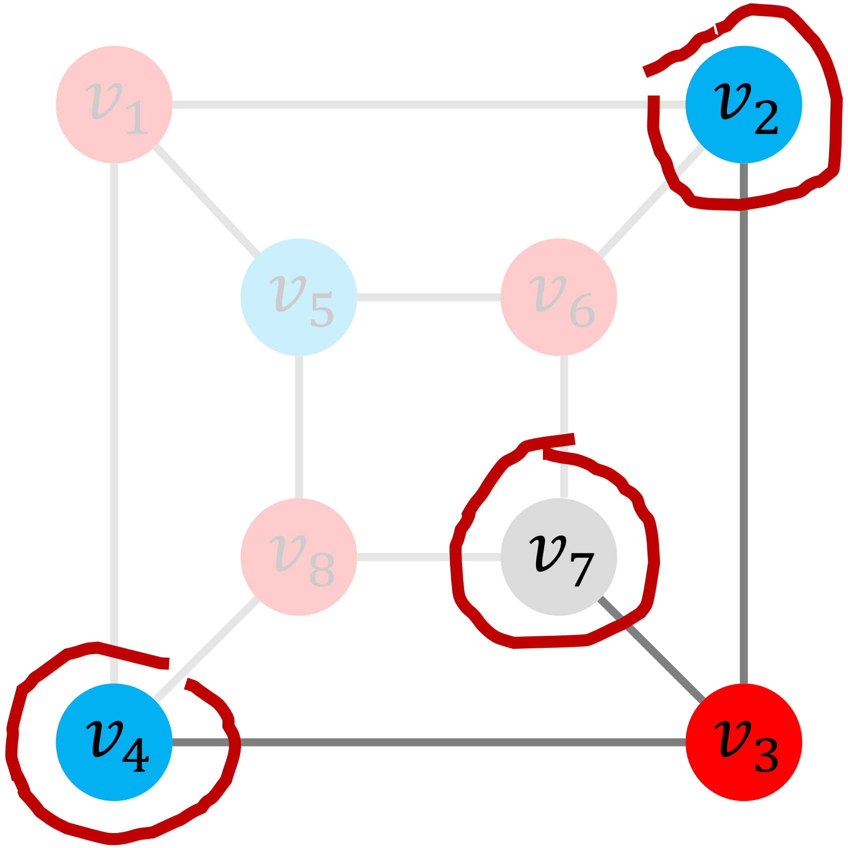
Iteration 5



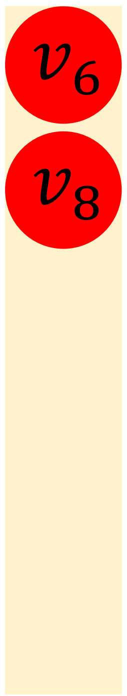
Queue:

- $v_3 \leftarrow$ dequeue( ).

Iteration 5

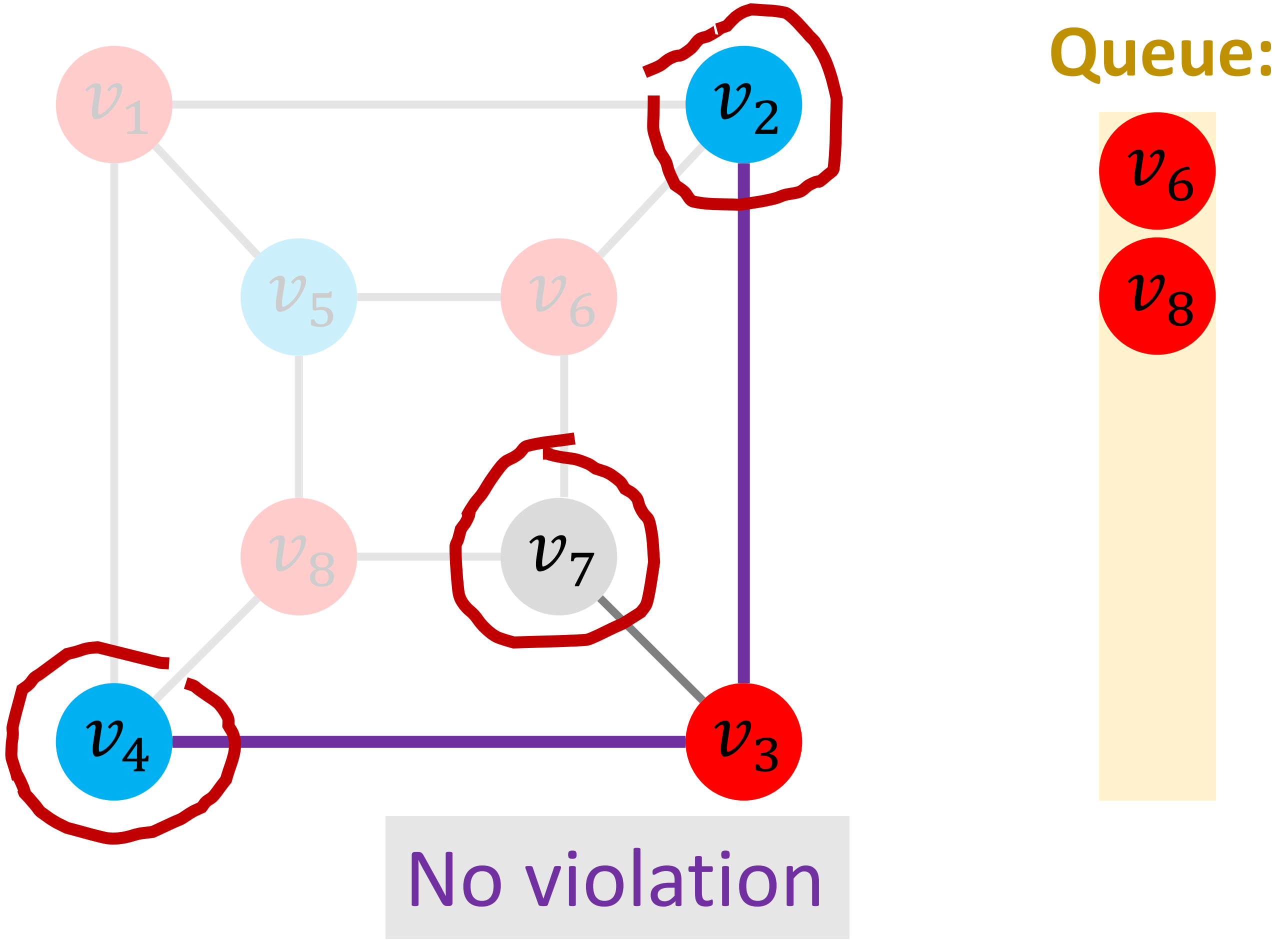


Queue:



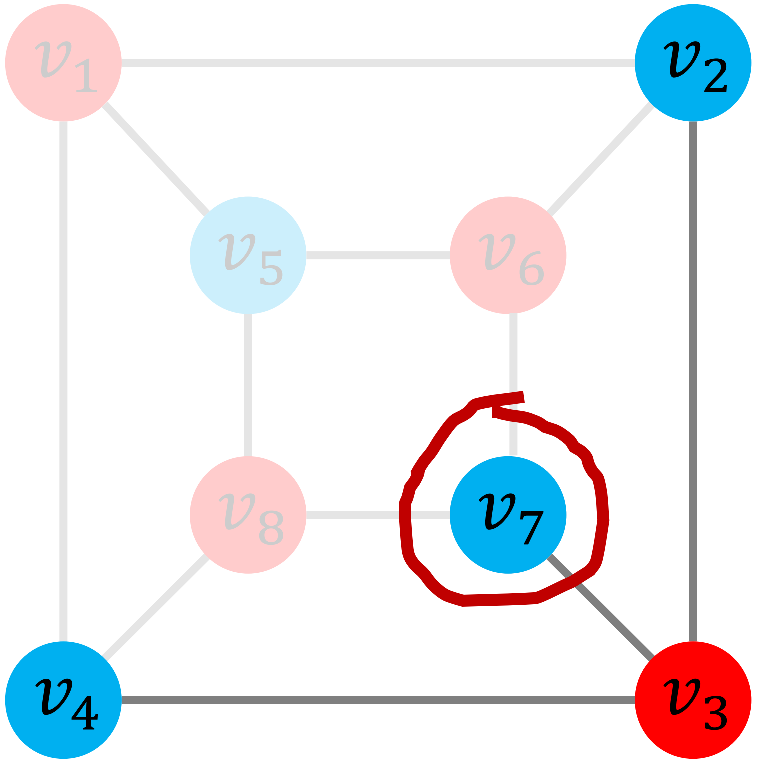
- $v_3 \leftarrow$ dequeue( ).

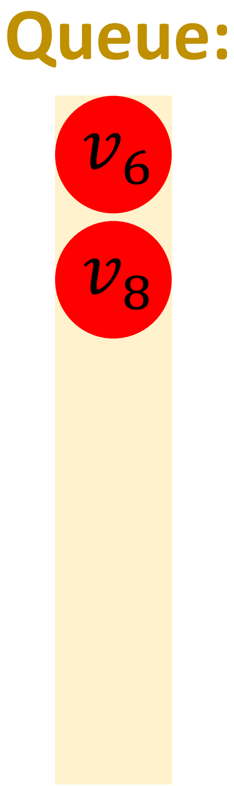
Iteration 5



- $v_3 \leftarrow$ dequeue( ).
- Check the visited neighbors to see if there is any violation.

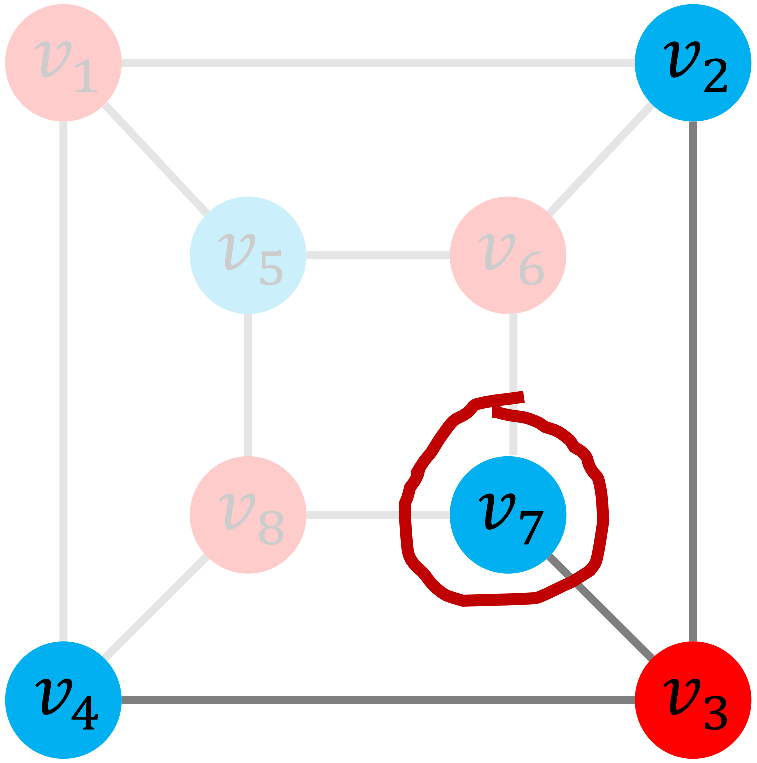
Iteration 5



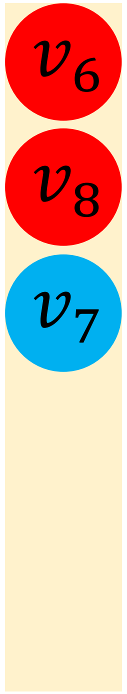


- $v_3 \leftarrow \text{dequeue}(\,)$.
- Check the visited neighbors to see if there is any violation.
- Assign blue color to its neighbor v_7.

Iteration 5

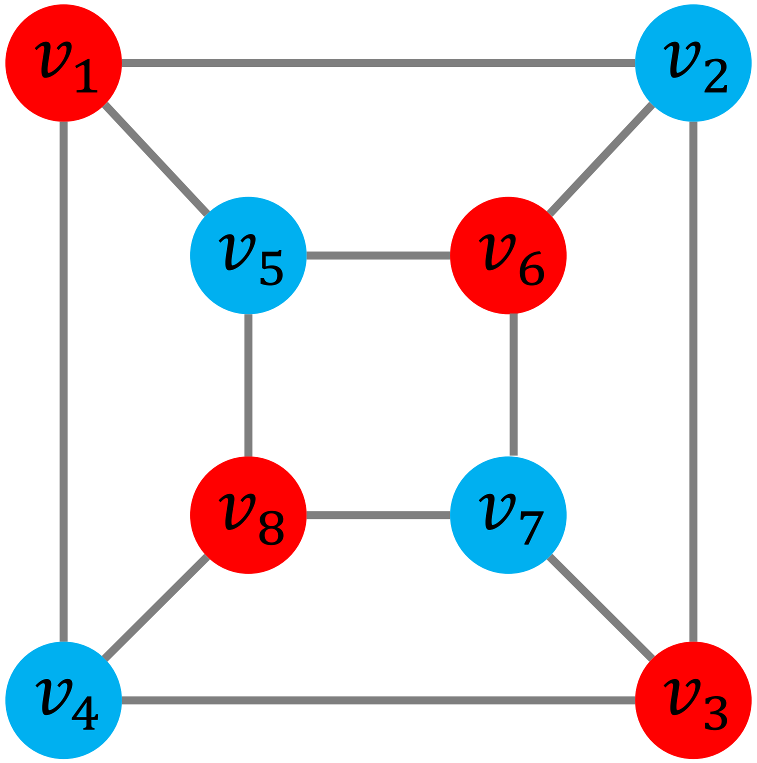


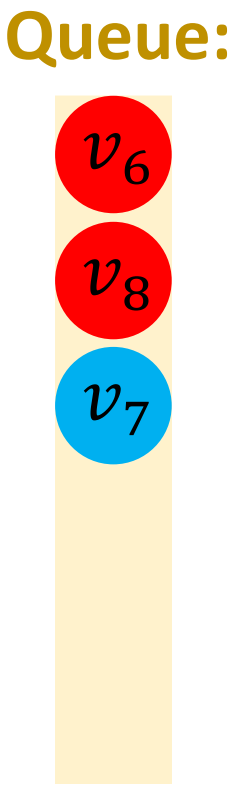
Queue:



- $v_3 \leftarrow \text{dequeue}(\,)$.
- Check the visited neighbors to see if there is any violation.
- Assign blue color to its neighbor v_7.
- Put the unvisited neighbor v_7 in the queue.

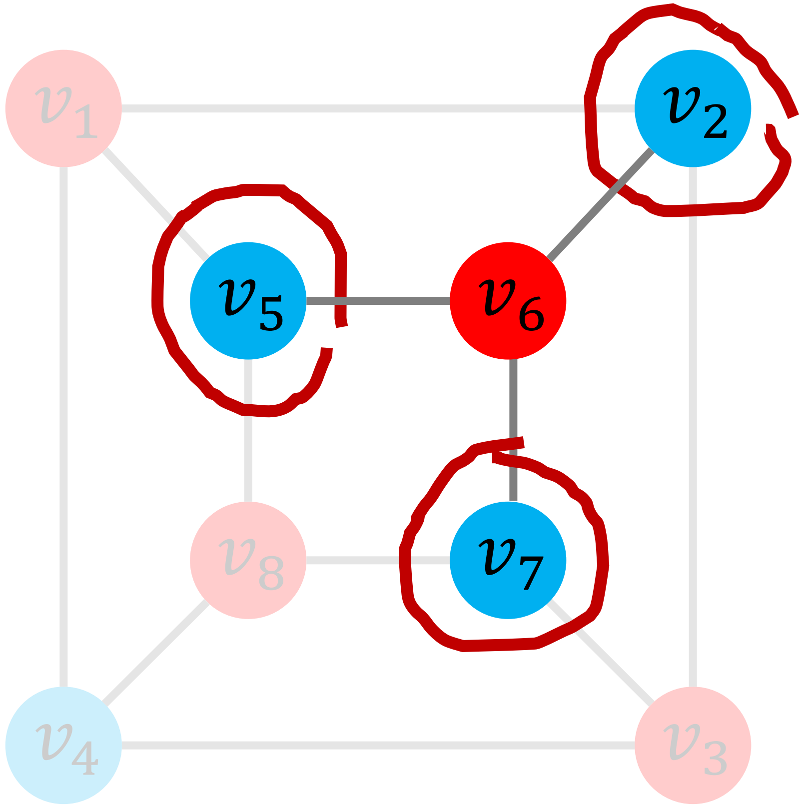
Iteration 6

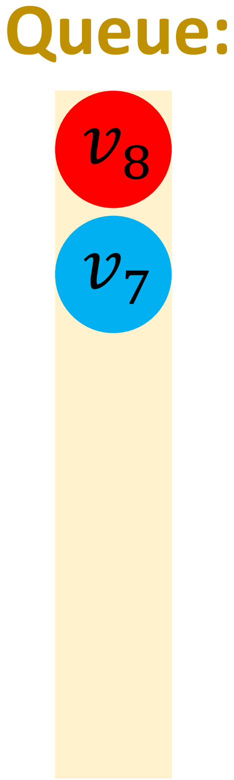




- $v_6 \leftarrow$ dequeue( ).

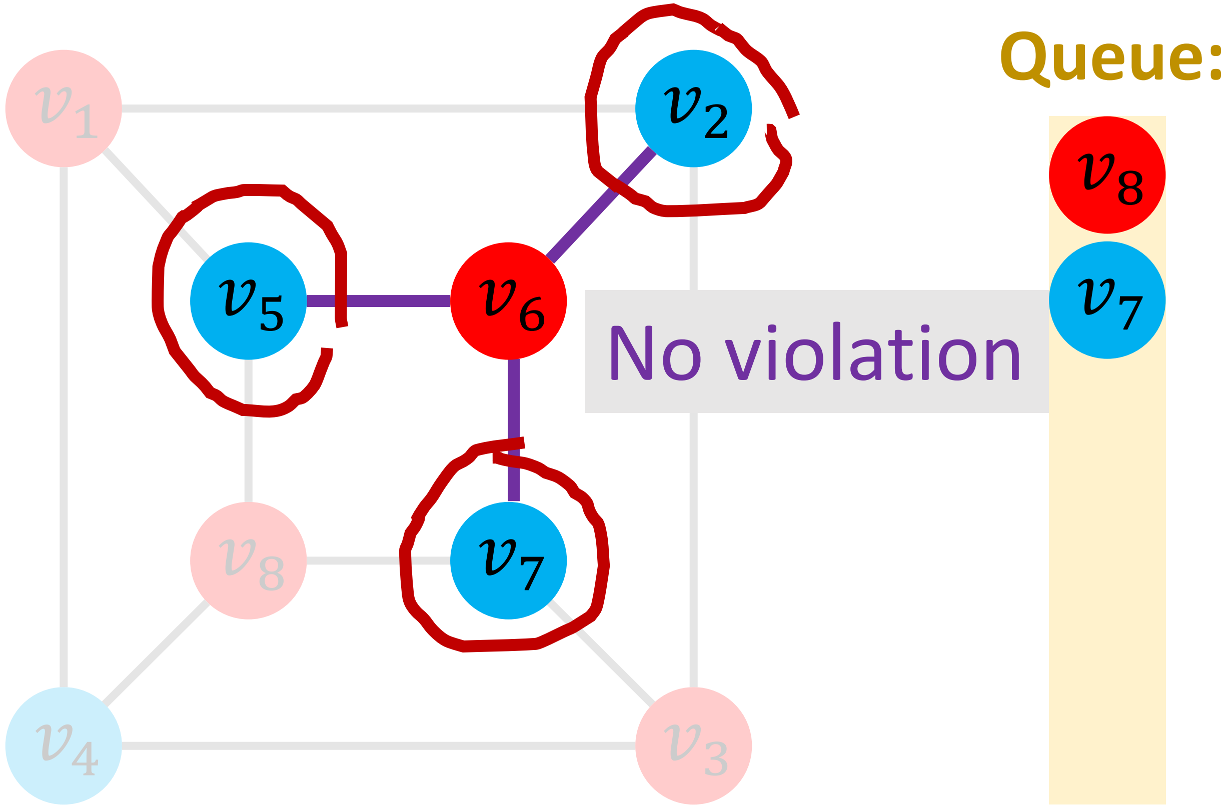
Iteration 6





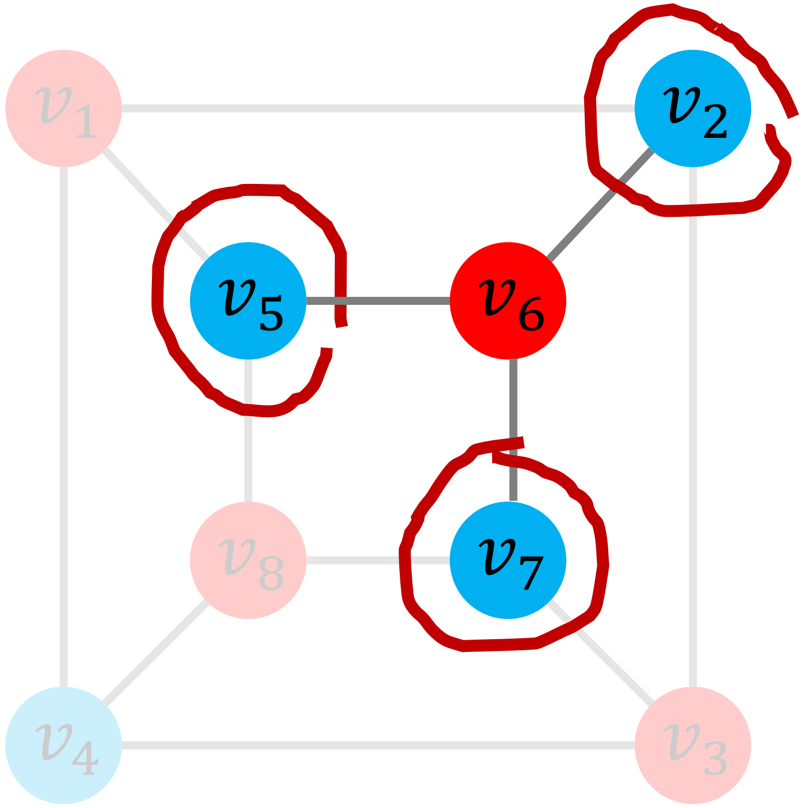
- $v_6 \leftarrow$ dequeue( ).

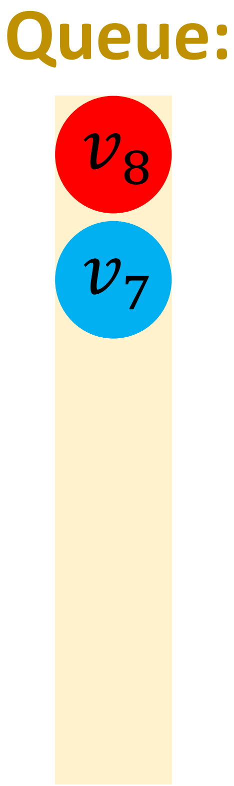
Iteration 6



- $v_6 \leftarrow$ dequeue( ).
- Check the visited neighbors to see if there is any violation.

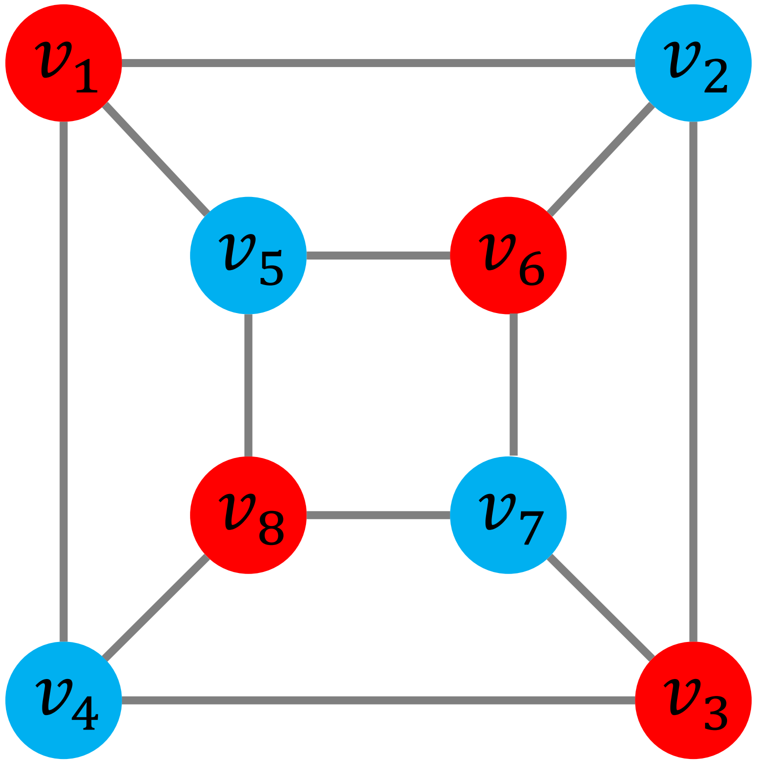
Iteration 6



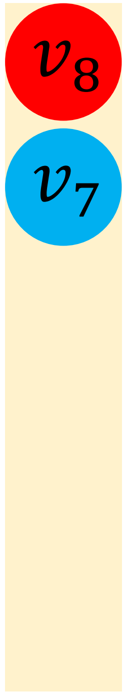


- $v_6 \leftarrow$ dequeue( ).
- Check the visited neighbors to see if there is any violation.
- Do not put visited neighbors in the queue.

Iteration 7

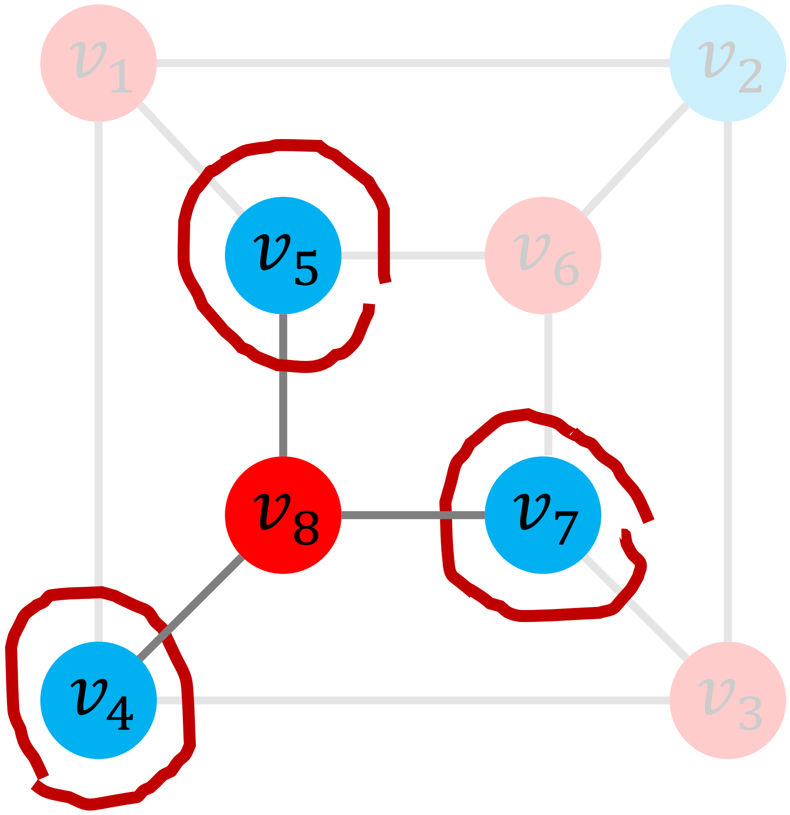


Queue:

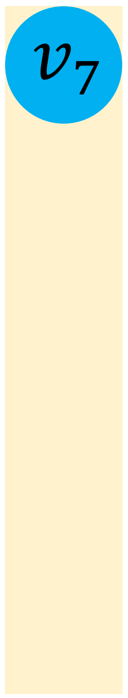


- $v_8 \leftarrow$ dequeue( ).

Iteration 7

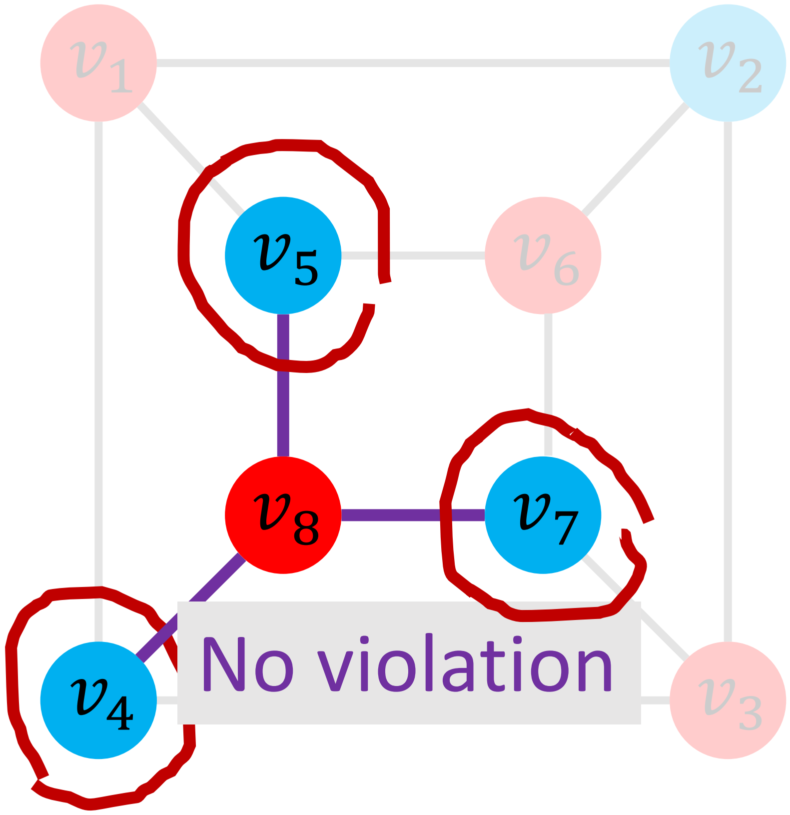


Queue:

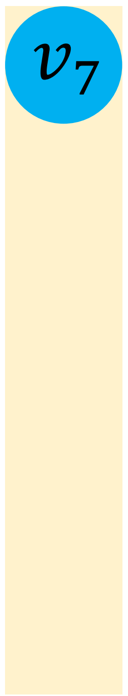


- $v_8 \leftarrow$ dequeue( ).

Iteration 7

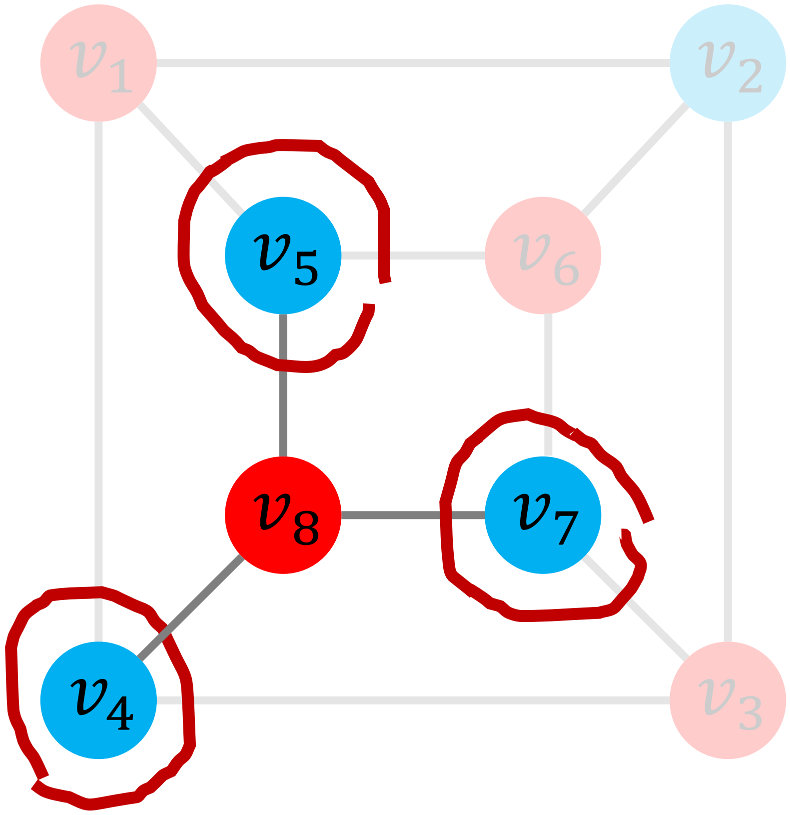


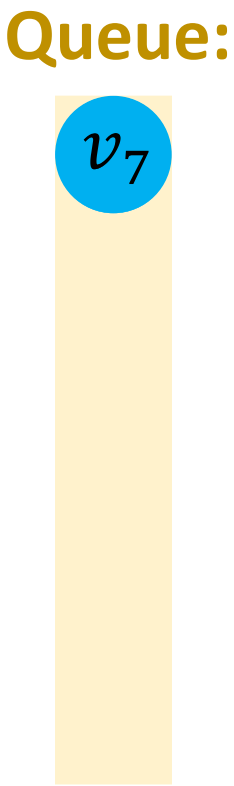
Queue:



- $v_8 \leftarrow \text{dequeue}(\,)$.
- Check the visited neighbors to see if there is any violation.

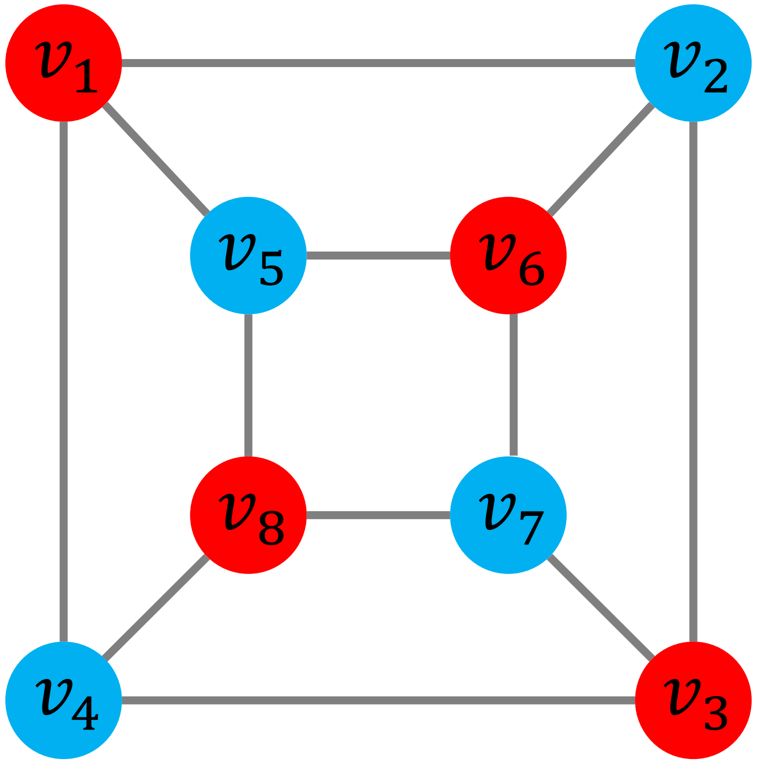
Iteration 7



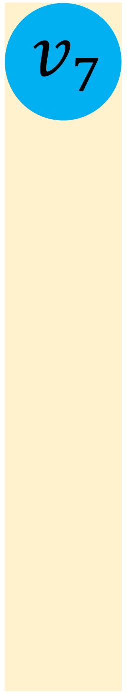


- $v_8 \leftarrow \text{dequeue}(\,)$.
- Check the visited neighbors to see if there is any violation.
- Do not put visited neighbors in the queue.

Iteration 8

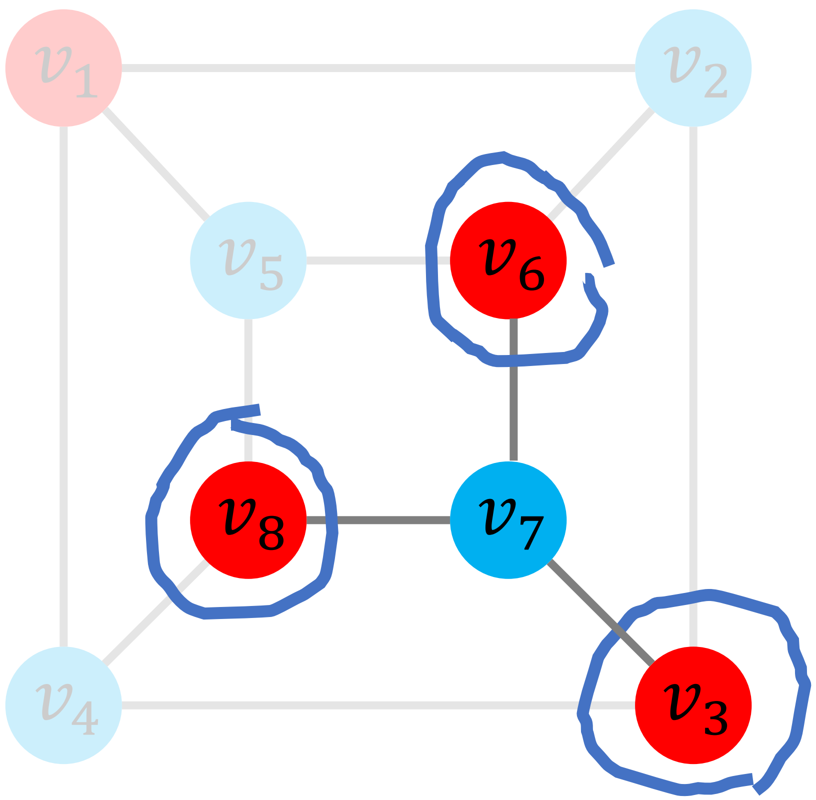


Queue:



- $v_7 \leftarrow$ dequeue( ).

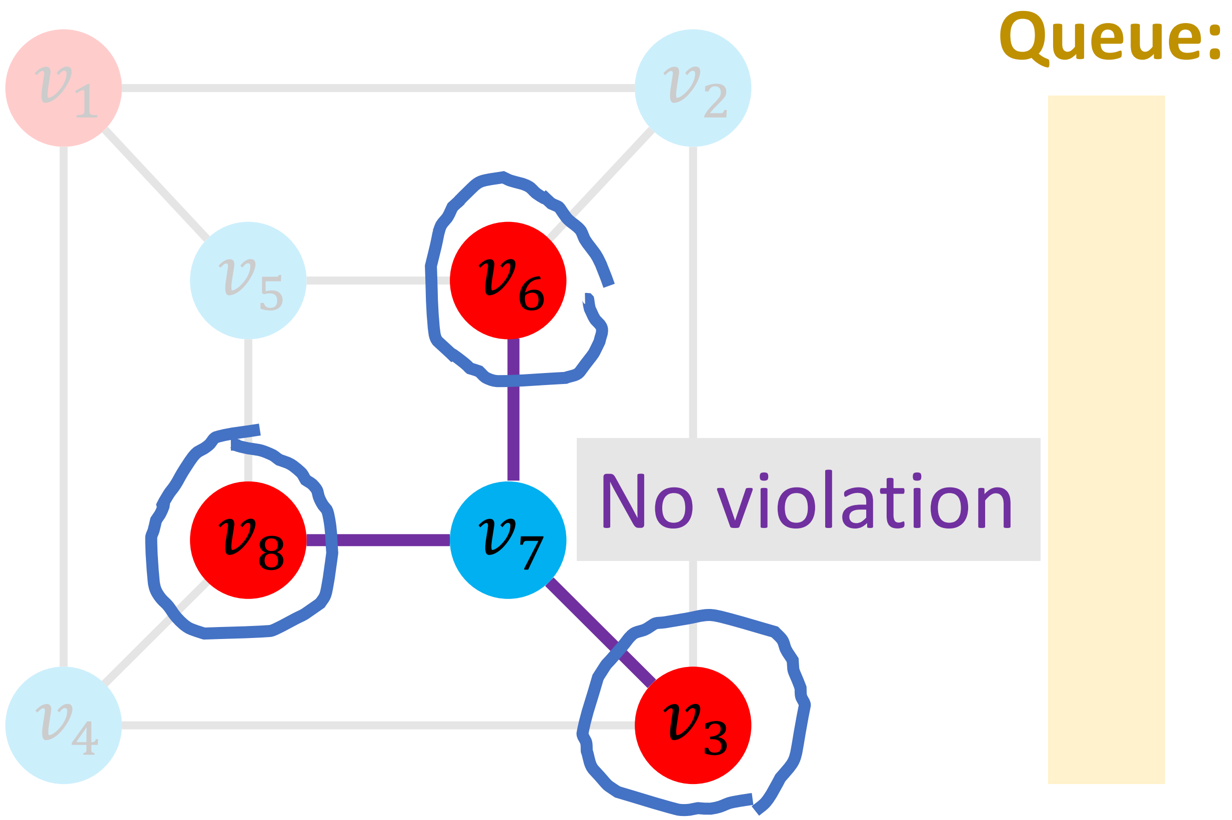
Iteration 8



Queue:

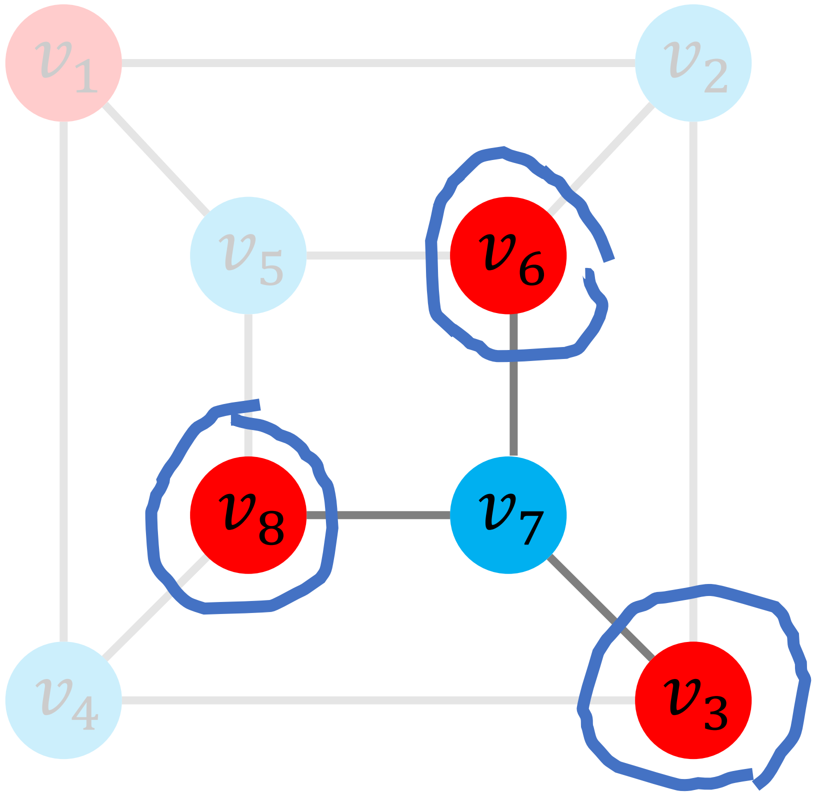
- $v_7 \leftarrow$ dequeue( ).

Iteration 8



- $v_7 \leftarrow \text{dequeue}(\,)$.
- Check the visited neighbors to see if there is any violation.

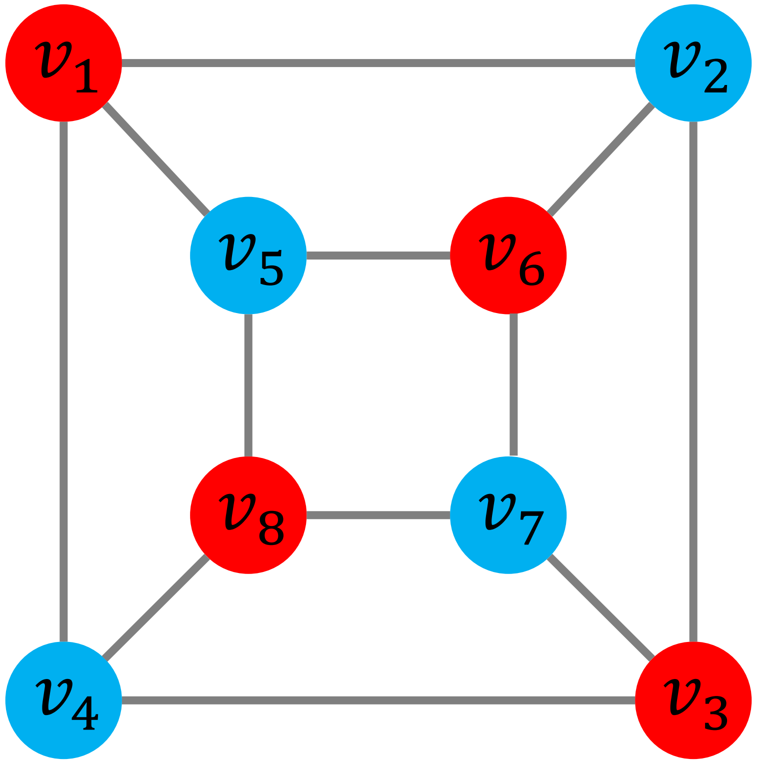
Iteration 8



Queue:

- $v_7 \leftarrow \text{dequeue}(\,)$.
- Check the visited neighbors to see if there is any violation.
- Do not put visited neighbors in the queue.

End of Procedure

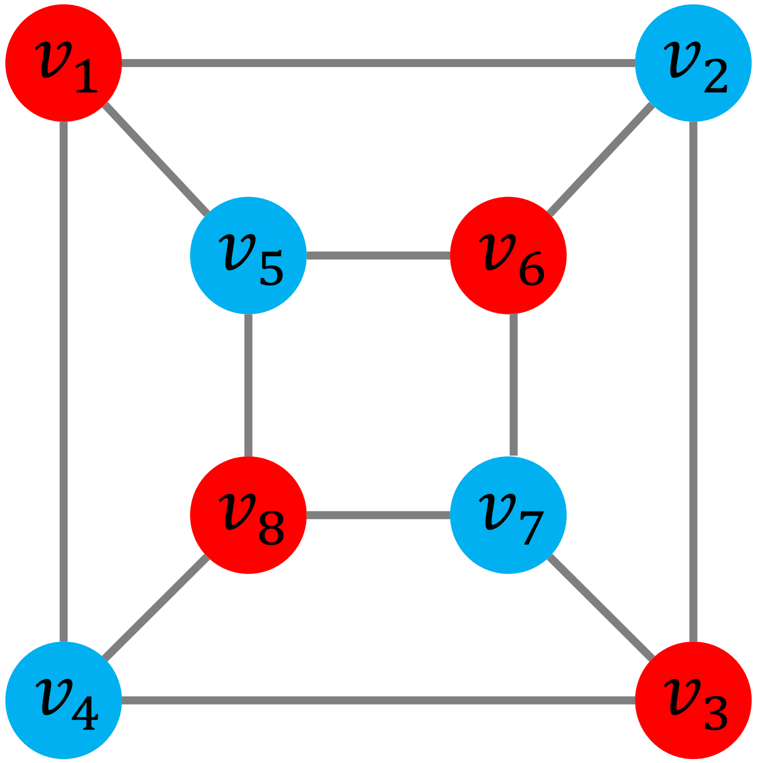


Queue:

- All the vertices have been colored.
- The queue is empty.

End the program

End of Procedure



Queue:

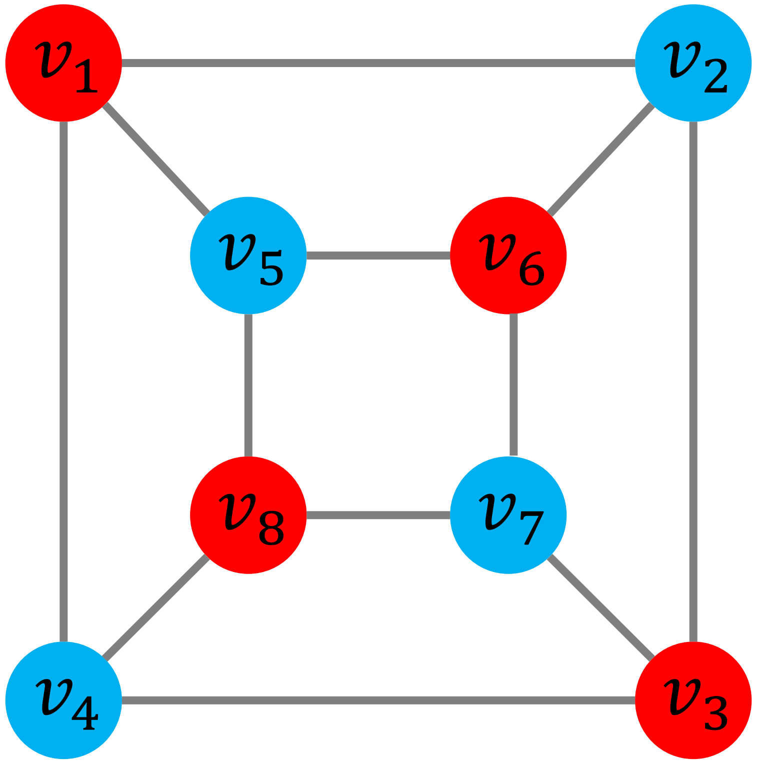
- All the vertices have been colored.
- The queue is empty.

End the program

- No violation is found.

The graph is bipartite

End of Procedure



Queue:

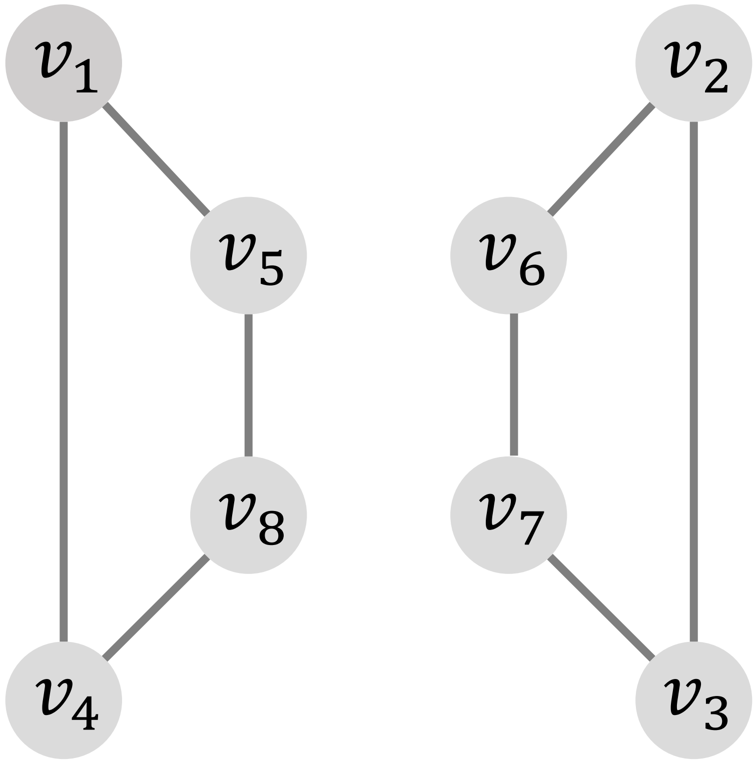
- All the vertices have been colored.

What if some vertices have not been colored?

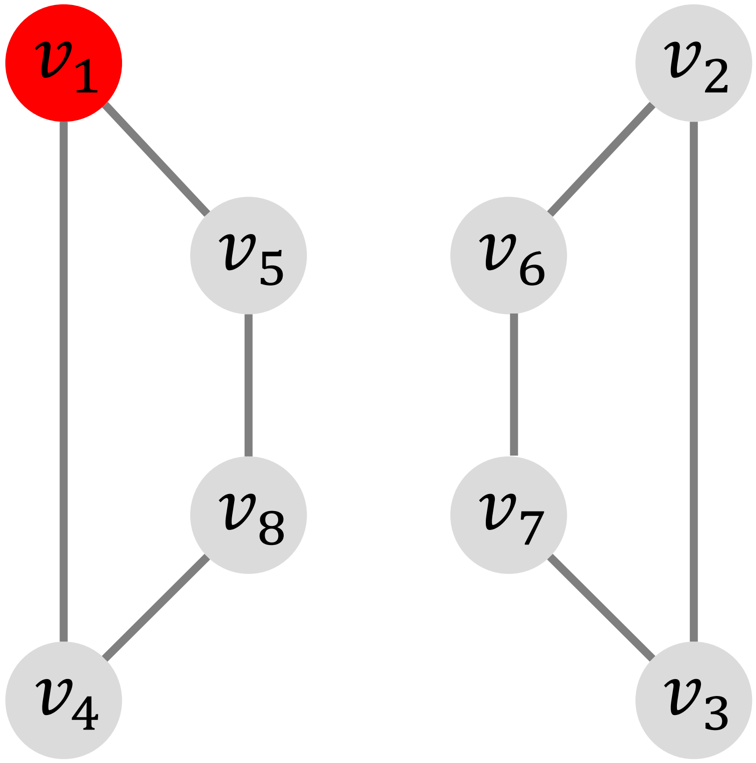
- No violation is found.

The graph is bipartite

Another Example

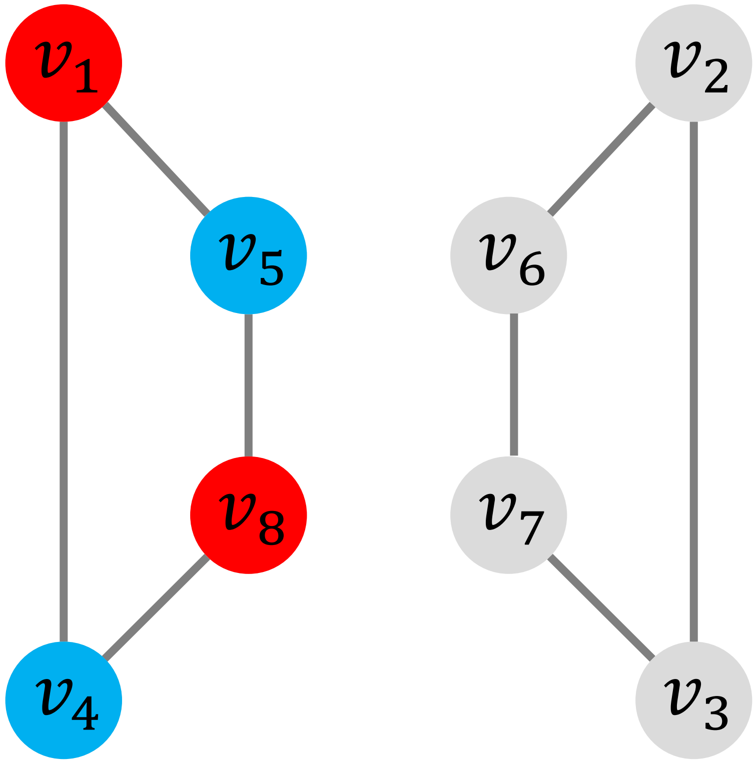


Another Example



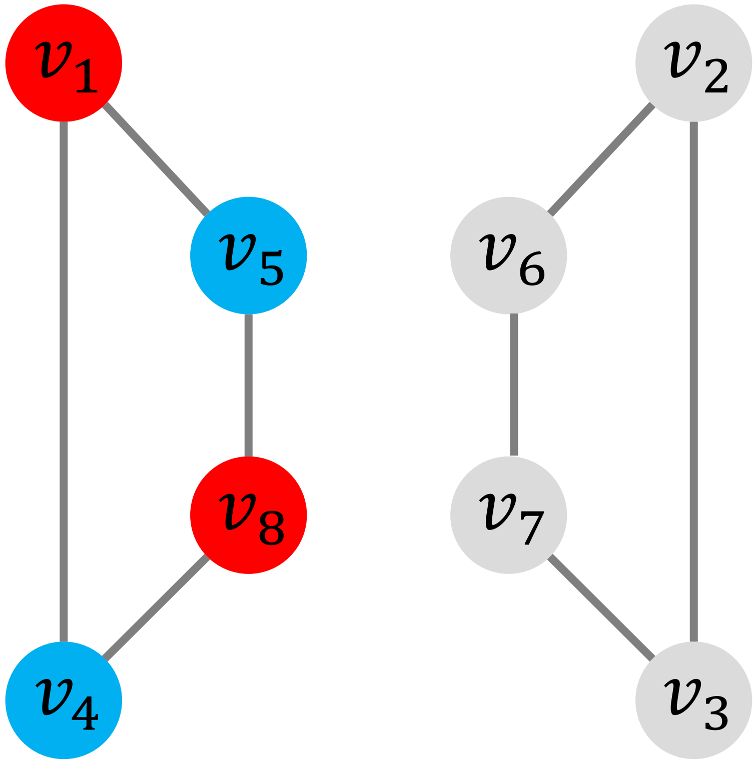
- Assign red color to v_1.

Another Example



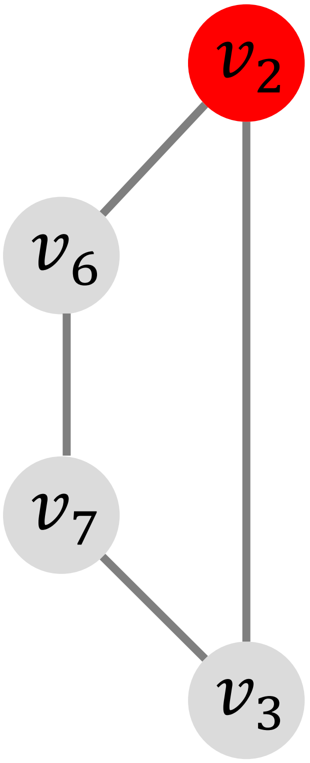
- Assign red color to v_1.
- Run the algorithm.

Another Example



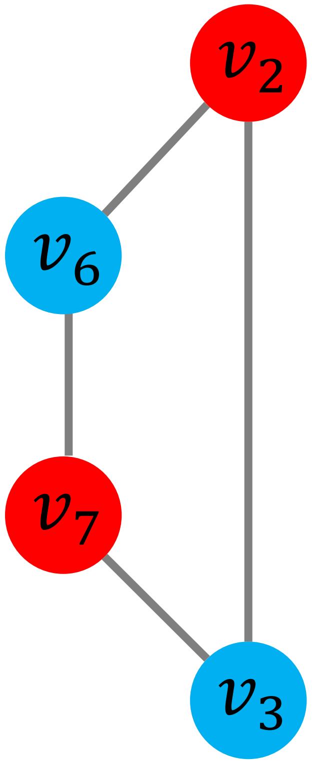
- Remove all the colored vertices.
- Run the algorithm on the remaining subgraph.

Another Example



- Assign red color to v_2.

Another Example



- Assign red color to v_2.
- Run the algorithm.

Testing Bipartiteness

1. Select a vertex, assign red color to it, and put it in the queue.

Testing Bipartiteness

1. Select a vertex, assign red color to it, and put it in the queue.
2. While the queue is not empty:
   a. $v \leftarrow \text{dequeue}(\,);$
   b. $c \leftarrow$ the opposite color of v;

Testing Bipartiteness

1. Select a vertex, assign red color to it, and put it in the queue.
2. While the queue is not empty:
   a. $v \leftarrow \text{dequeue}(\,);$
   b. $c \leftarrow$ the opposite color of v;
   c. For each $u \in \text{Neighbor}(v)$:
      i. If u has been colored, check whether there is a violation;
      ii. Otherwise, assign color c to u, and put u in the queue;

Testing Bipartiteness

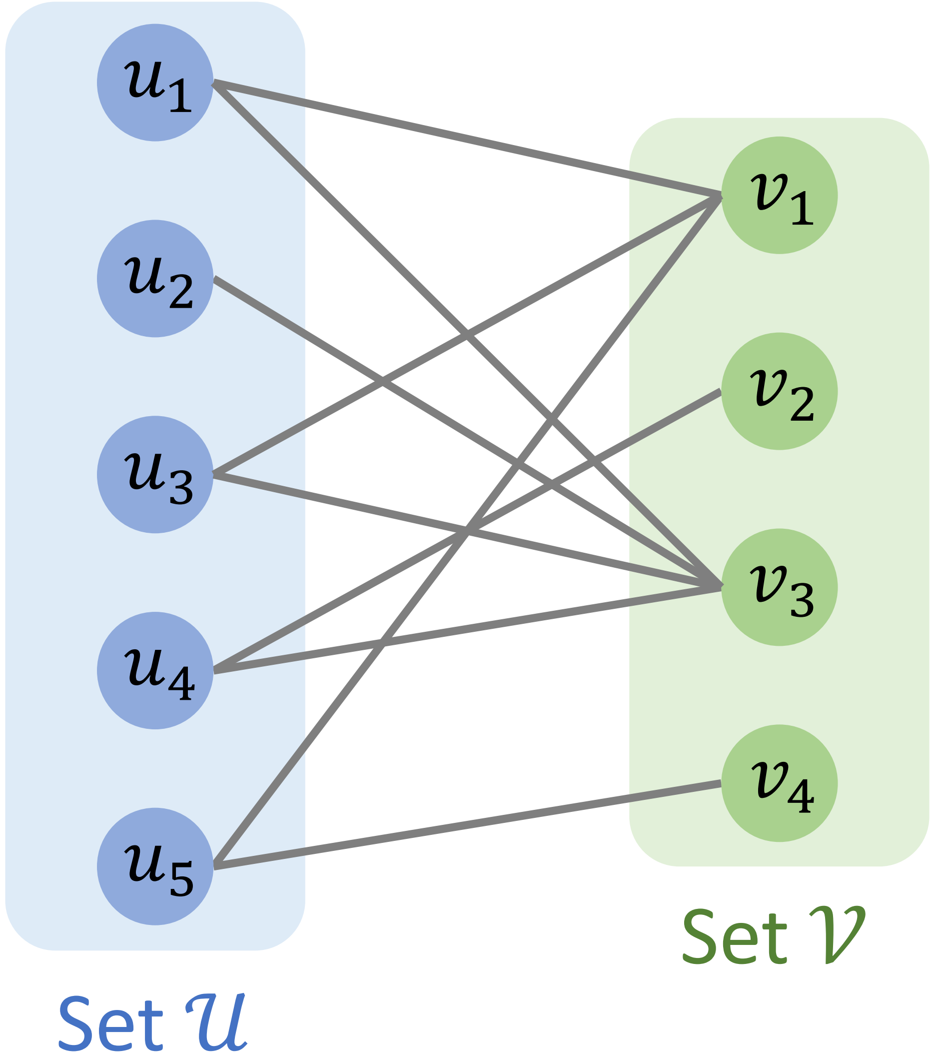
1. Select a vertex, assign red color to it, and put it in the queue.
2. While the queue is not empty:
   a. $v \leftarrow \text{dequeue}(\,);$
   b. $c \leftarrow$ the opposite color of v;
   c. For each $u \in \text{Neighbor}(v)$:
      i. If u has been colored, check whether there is a violation;
      ii. Otherwise, assign color c to u, and put u in the queue;
   d. If color violation is found, return FALSE (not bipartite).

Testing Bipartiteness

3. When the queue is empty, check if all the vertices have been colored.
    - If yes, end the program and return TRUE.
    - If no, delete the colored vertices, and apply the algorithm to the remaining subgraph.

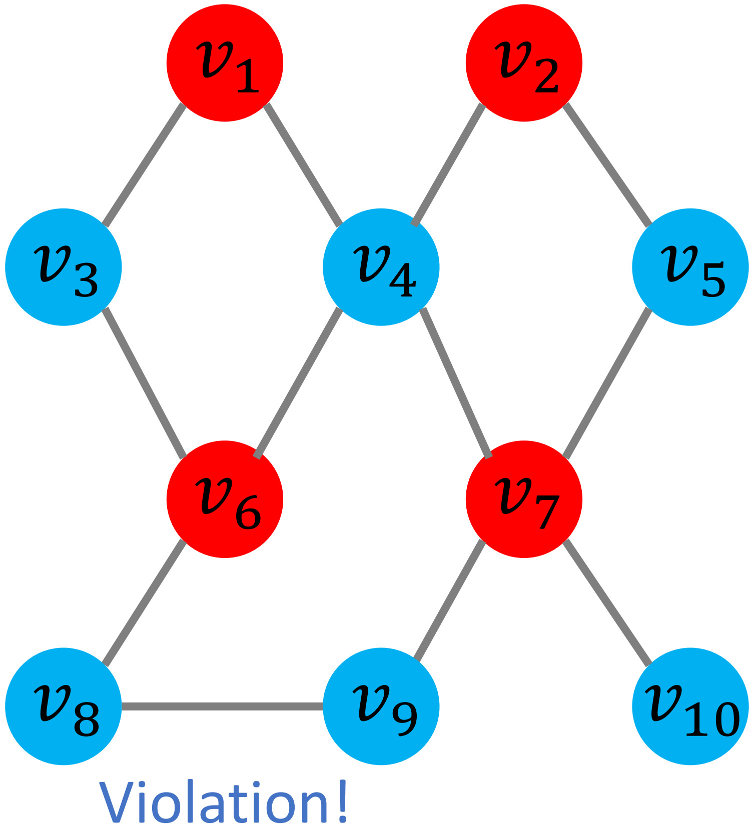
Summary

Bipartite Graph



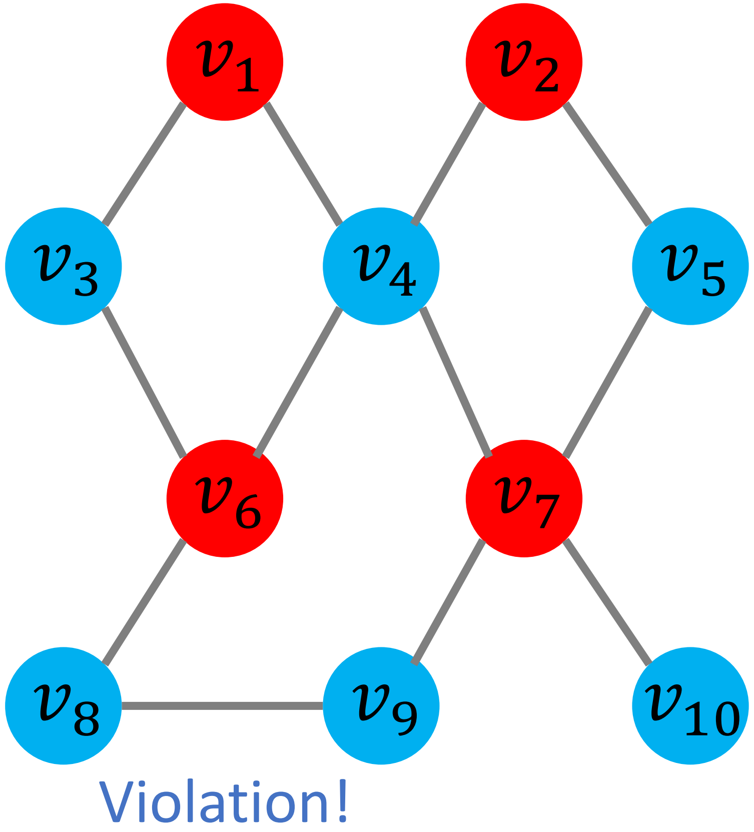
- The vertices can be partitioned into two subsets, $\mathcal{U}$ and $\mathcal{V}$.
- No edge between two vertices in $\mathcal{U}$.
- No edge between two vertices in $\mathcal{V}$.
- Application: matching.
  - Matching candidates and positions.
  - Pet adoption.
  - Dating.

Testing Bipartiteness



- Basic idea:
  - Coloring the vertices using red and blue.
  - Check whether there is any violation.

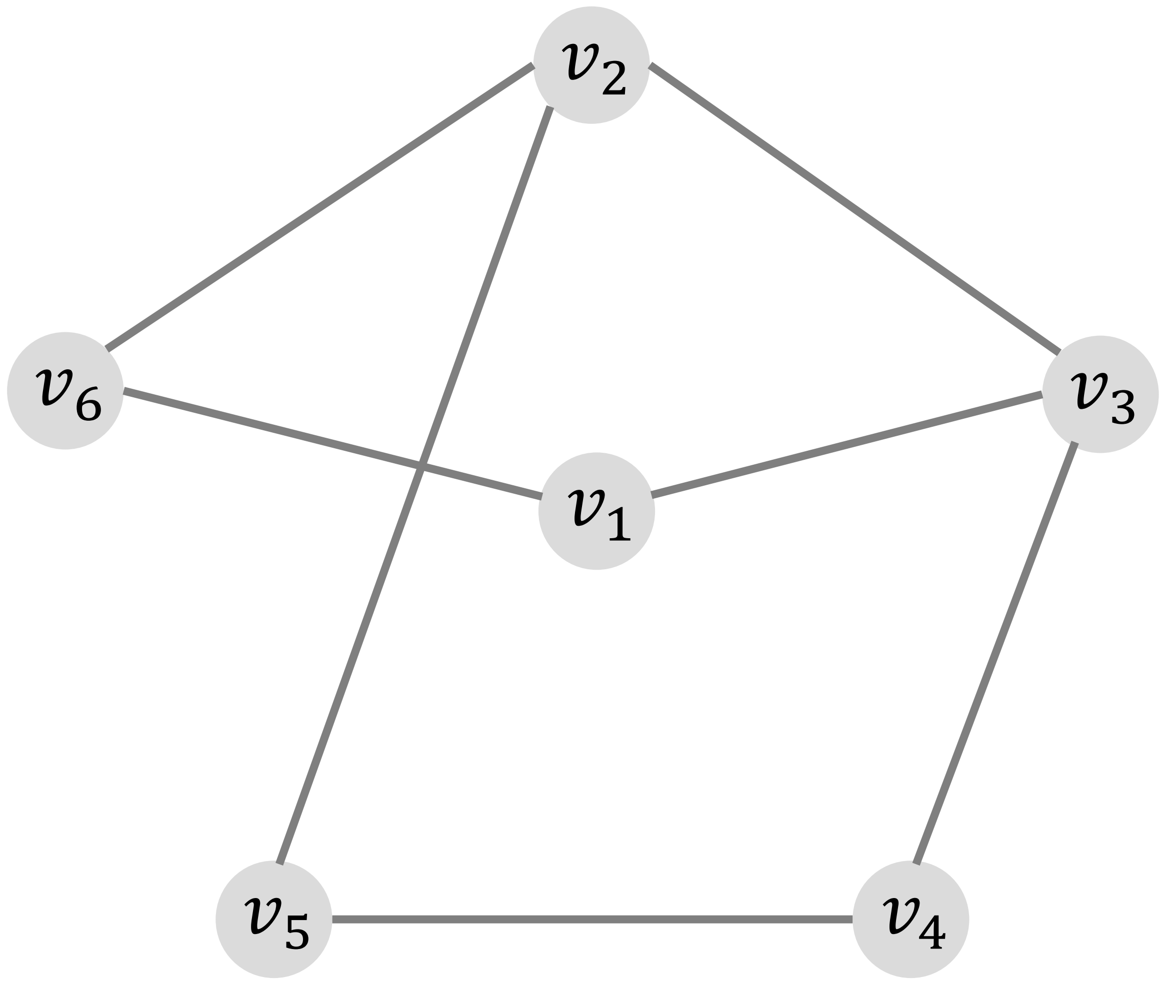
Testing Bipartiteness



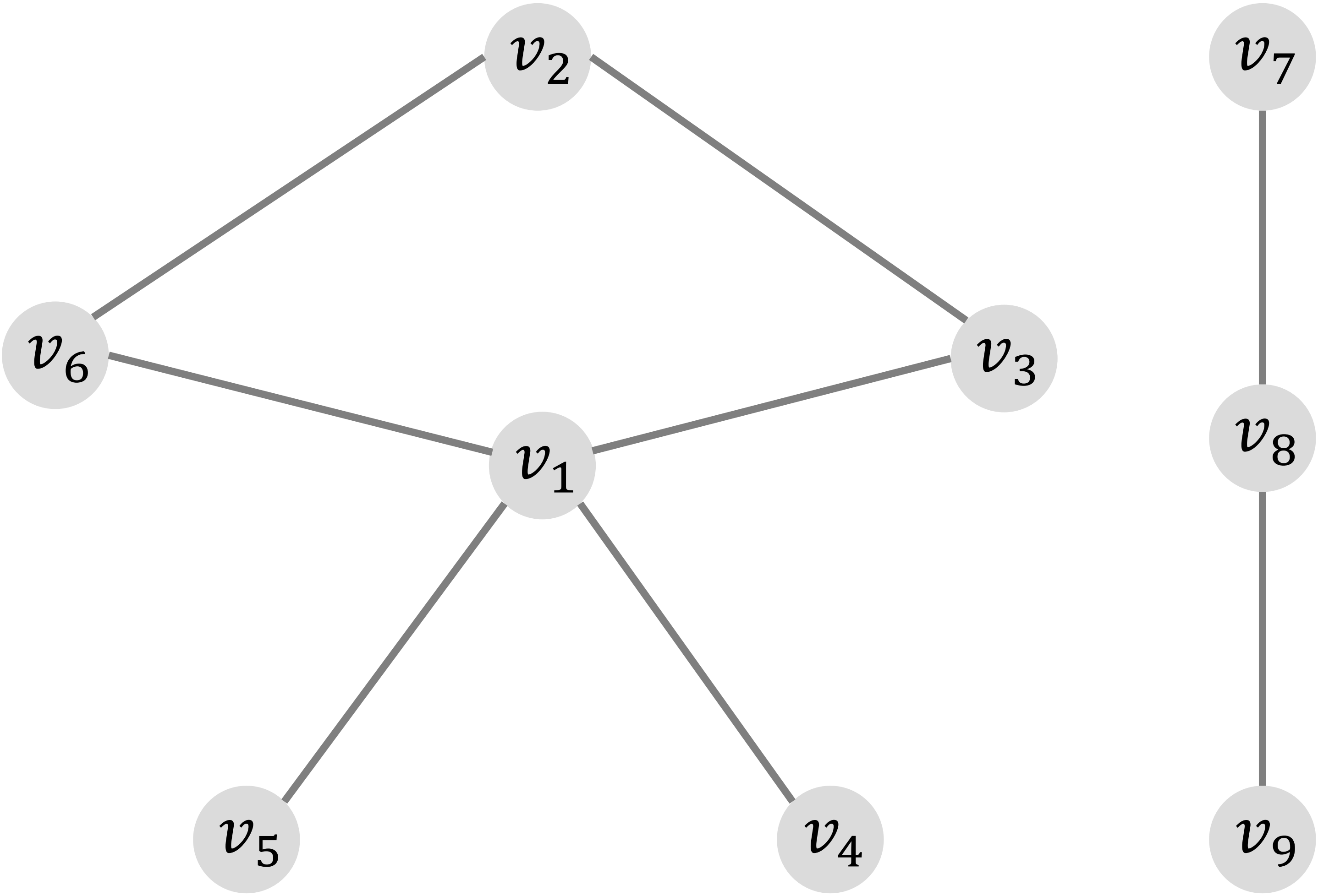
- Basic idea:
  - Coloring the vertices using red and blue.
  - Check whether there is any violation.
- Algorithm: breadth-first search (BFS).
- Time complexity: $O(|\mathcal{E}| + |\mathcal{V}|)$.

Questions

Q1: Is it a bipartite graph?



Q2: Is it a bipartite graph?



Thank You!