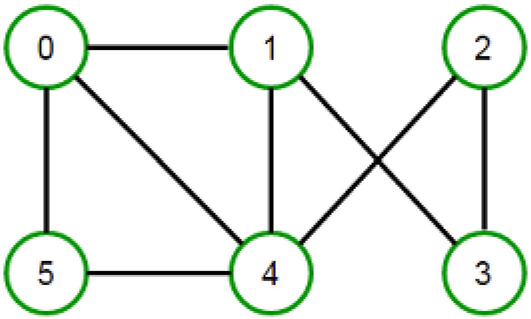
RA1911026010029
Thejaswin S

EXP-2 Graph Colouring Problem

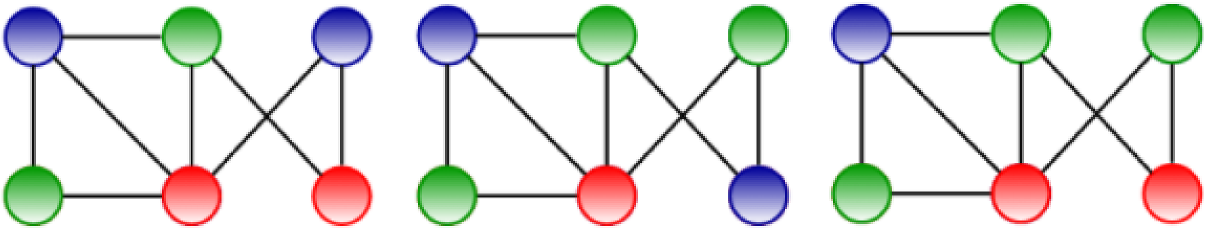
AIM

To implement graph coloring problem in python and verify its output.

Graph coloring (also called vertex coloring) is a way of coloring a graph’s vertices such that no two adjacent vertices share the same color.
For example, consider the following graph:



We can color it in many ways by using the minimum of 3 colors.



Algorithm

Greedy Algorithm

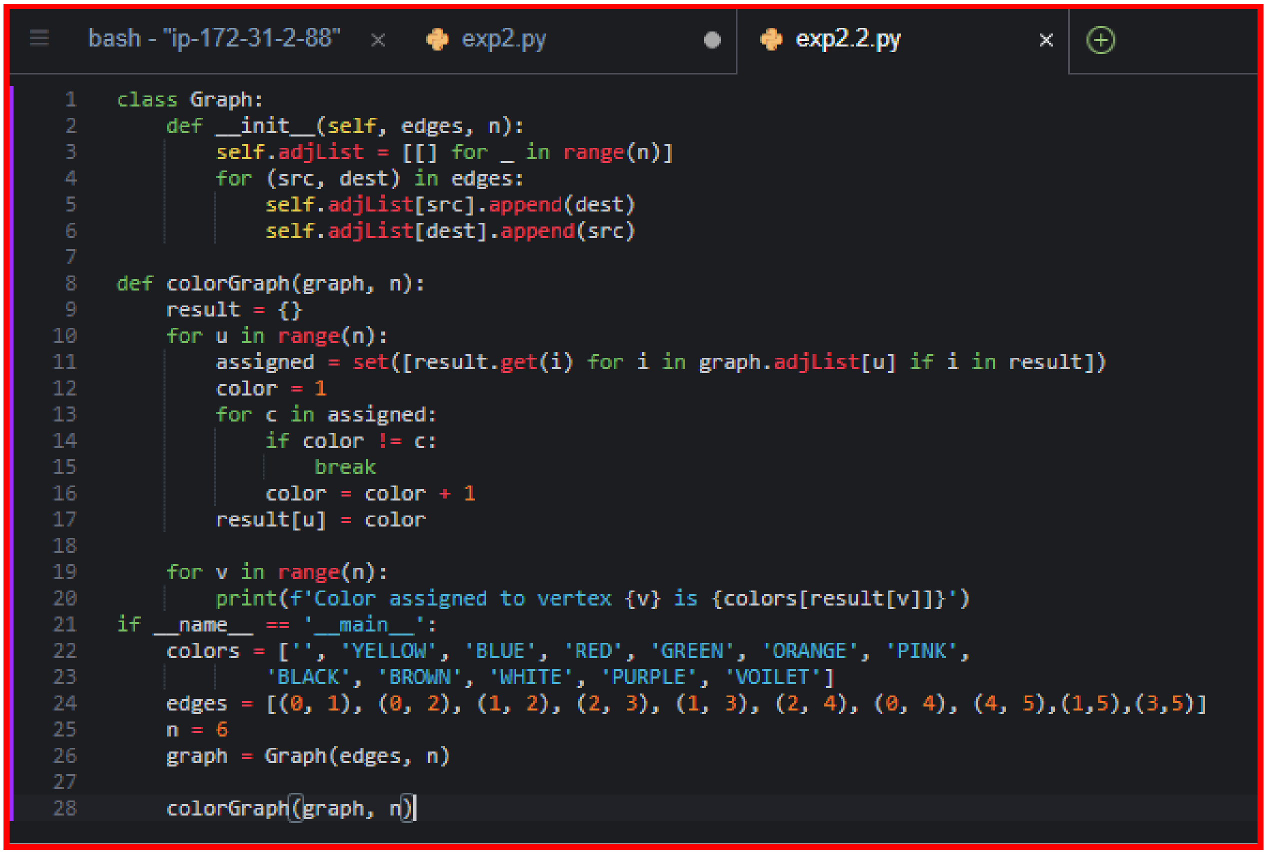
1. Color the first vertex with the first lowest color.
2. Repeat the following for the remaining V-1 vertices.
    - Consider the currently picked vertex and color it with the lowest numbered color that has not been used on any previously colored vertices adjacent to it. If all previously used colors appear on vertices adjacent to v, assign a new color to it.

Program

```
class Graph:
    def __init__(self, edges, n):
        self.adjList = [[] for _ in range(n)]
        for (src, dest) in edges:
            self.adjList[src].append(dest)
            self.adjList[dest].append(src)
```

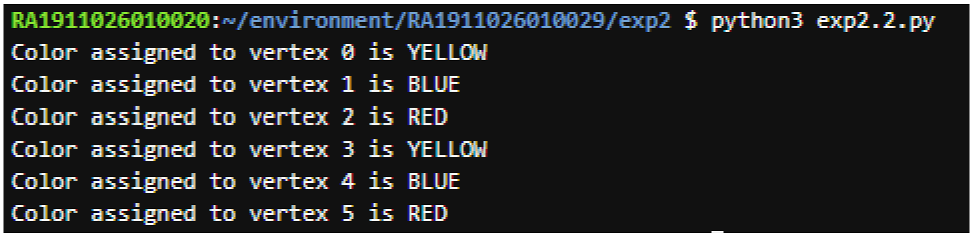
```
def colorGraph(graph, n):
    result = {}
    for u in range(n):
        assigned = set([result.get(i) for i in graph.adjList[u] if i in result])
        color = 1
        for c in assigned:
            if color != c:
                break
            color = color + 1
        result[u] = color

    for v in range(n):
        print(f'Color assigned to vertex {v} is {colors[result[v]]}')
if __name__ == '__main__':
    colors = ['', 'YELLOW', 'BLUE', 'RED', 'GREEN', 'ORANGE', 'PINK',
            'BLACK', 'BROWN', 'WHITE', 'PURPLE', 'VOILET']
    edges = [(0, 1), (0, 2), (1, 2), (2, 3), (1, 3), (2, 4), (0, 4), (4, 5),(1,5),(3,5)]
    n = 6
    graph = Graph(edges, n)

    colorGraph(graph, n)
```

Time complexity - O(V × E)



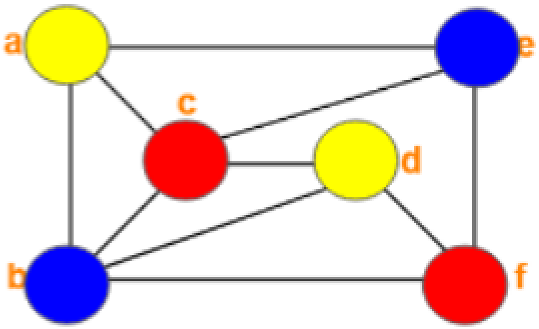
```
class Graph:
    def __init__(self, edges, n):
        self.adjList = [[] for _ in range(n)]
        for (src, dest) in edges:
            self.adjList[src].append(dest)
            self.adjList[dest].append(src)

def colorGraph(graph, n):
    result = {}
    for u in range(n):
        assigned = set([result.get(i) for i in graph.adjList[u] if i in result])
        color = 1
        for c in assigned:
            if color != c:
                break
            color = color + 1
        result[u] = color

    for v in range(n):
        print(f'Color assigned to vertex {v} is {colors[result[v]]}')
if __name__ == '__main__':
    colors = ['', 'YELLOW', 'BLUE', 'RED', 'GREEN', 'ORANGE', 'PINK',
              'BLACK', 'BROWN', 'WHITE', 'PURPLE', 'VOILET']
    edges = [(0, 1), (0, 2), (1, 2), (2, 3), (1, 3), (2, 4), (0, 4), (4, 5),(1,5),(3,5)]
    n = 6
    graph = Graph(edges, n)

    colorGraph(graph, n)
```

Output

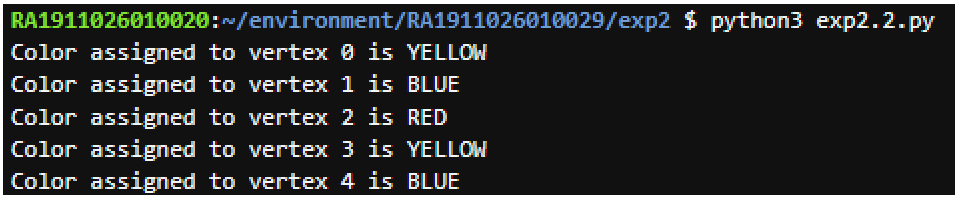


```
RA1911026010020:~/environment/RA1911026010029/exp2 $ python3 exp2.2.py
Color assigned to vertex 0 is YELLOW
Color assigned to vertex 1 is BLUE
Color assigned to vertex 2 is RED
Color assigned to vertex 3 is YELLOW
Color assigned to vertex 4 is BLUE
Color assigned to vertex 5 is RED
```

Graph 1

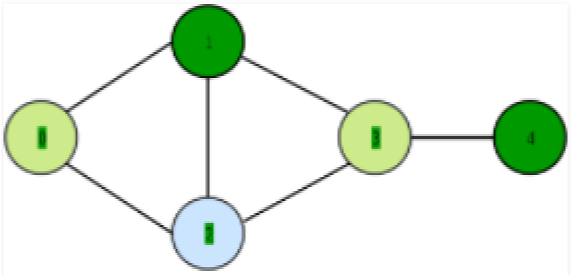


Output-2



```
RA1911026010020:~/environment/RA1911026010029/exp2 $ python3 exp2.2.py
Color assigned to vertex 0 is YELLOW
Color assigned to vertex 1 is BLUE
Color assigned to vertex 2 is RED
Color assigned to vertex 3 is YELLOW
Color assigned to vertex 4 is BLUE
```

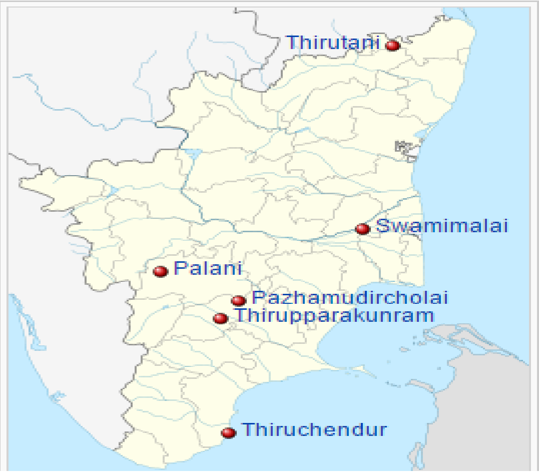
Graph-2

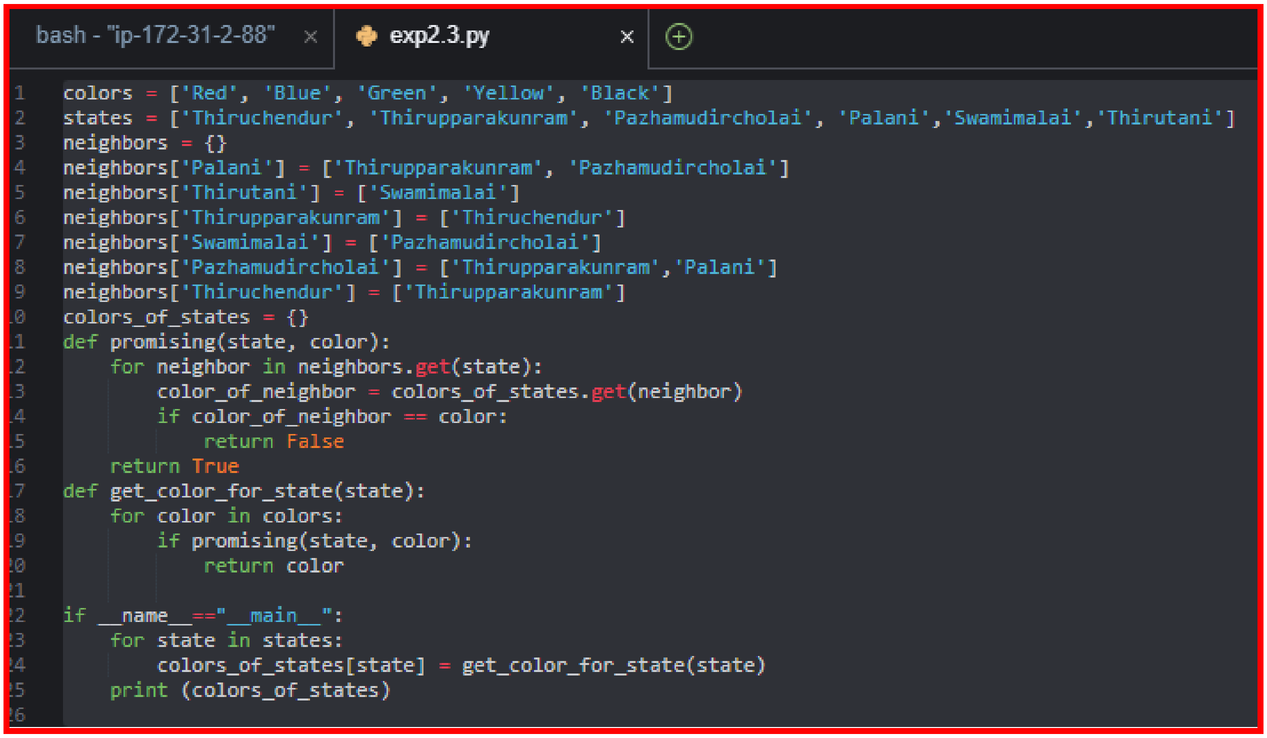


Graph coloring for the real-life problem:

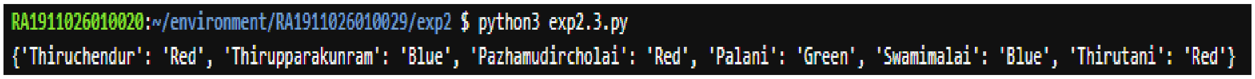
To color the different temples cities with different colors.

```
colors = ['Red', 'Blue', 'Green', 'Yellow', 'Black']
states = ['Thiruchendur', 'Thirupparakunram', 'Pazhamudircholai',
'Palani','Swamimalai','Thirutani']
neighbors = {}
neighbors['Palani'] = ['Thirupparakunram', 'Pazhamudircholai']
neighbors['Thirutani'] = ['Swamimalai']
neighbors['Thirupparakunram'] = ['Thiruchendur']
neighbors['Swamimalai'] = ['Pazhamudircholai']
neighbors['Pazhamudircholai'] = ['Thirupparakunram','Palani']
neighbors['Thiruchendur'] = ['Thirupparakunram']
colors_of_states = {}
def promising(state, color):
    for neighbor in neighbors.get(state):
        color_of_neighbor = colors_of_states.get(neighbor)
        if color_of_neighbor == color:
            return False
    return True
def get_color_for_state(state):
    for color in colors:
        if promising(state, color):
            return color

if __name__=="__main__":
    for state in states:
        colors_of_states[state] = get_color_for_state(state)
    print (colors_of_states)
```





```python
colors = ['Red', 'Blue', 'Green', 'Yellow', 'Black']
states = ['Thiruchendur', 'Thirupparakunram', 'Pazhamudircholai', 'Palani','Swamimalai','Thirutani']
neighbors = {}
neighbors['Palani'] = ['Thirupparakunram', 'Pazhamudircholai']
neighbors['Thirutani'] = ['Swamimalai']
neighbors['Thirupparakunram'] = ['Thiruchendur']
neighbors['Swamimalai'] = ['Pazhamudircholai']
neighbors['Pazhamudircholai'] = ['Thirupparakunram','Palani']
neighbors['Thiruchendur'] = ['Thirupparakunram']
colors_of_states = {}
def promising(state, color):
    for neighbor in neighbors.get(state):
        color_of_neighbor = colors_of_states.get(neighbor)
        if color_of_neighbor == color:
            return False
    return True
def get_color_for_state(state):
    for color in colors:
        if promising(state, color):
            return color

if __name__=="__main__":
    for state in states:
        colors_of_states[state] = get_color_for_state(state)
    print (colors_of_states)
```



```
RA1911026010020:~/environment/RA1911026010029/exp2 $ python3 exp2.3.py
{'Thiruchendur': 'Red', 'Thirupparakunram': 'Blue', 'Pazhamudircholai': 'Red', 'Palani': 'Green', 'Swamimalai': 'Blue', 'Thirutani': 'Red'}
```

Result

Thus the program for graph coloring program has been successfully implemented and real-life problem example is shown and output is verified.