In [45]:

```
import networkx as nx
import matplotlib.pyplot as plt
```

```
class Queen:

    def __init__(self, N):
        self.N= N
        #chessboard
        #NxN matrix with all elements 0
        self.board = [[0]*N for _ in range(N)]

    def disp_board(self):
        for row in self.board:
            print()
            for col in row:
                if col==1:
                    print(u"\U0001F451", end=' ')
                else:
                    print(u"\u274C", end=' ')
        print(end= '\n')

    def is_attack(self, i, j):
        #checking if there is a queen in row or column
        for k in range(0,self.N):
            """
            In slicing, if 'k' is used in first '[]' (slicing index)
            then it traverses row and if it is used in second slicing
            index dimension, then it traverses columns
            """
            if self.board[i][k]==1 or self.board[k][j]==1:
                return True
        #checking diagonals
        for k in range(0,self.N):
            for l in range(0,self.N):
                if (k+l==i+j) or (k-l==i-j):
                    # k+l checks left to right diagonal and
                    # k-l checks right to left diagonal
                    if self.board[k][l]==1:
                        return True
        return False

    def N_queen(self, n):
        #if n is 0, solution found
        if n==0: # n is the number of queens yet to be placed
            return True

        print('----','\n','Current State:')
        self.disp_board()
        for i in range(0,self.N):
            for j in range(0,self.N):
                '''checking if we can place a queen here or not
                queen will not be placed if the place is being attacked
                or already occupied'''
                if (not(self.is_attack(i,j))) and (self.board[i][j]!=1):
                    self.board[i][j] = 1
                    #recursion
                    #wether we can put the next queen with this arrangment or not
                    if self.N_queen(n-1)==True:
                        return True
                    self.board[i][j] = 0

        return False
```

In [48]:

```
class Graph():

    def __init__(self, fname):

        ip= open(fname, 'r') # Input file
        raw= ip.read().splitlines() # split into lines
        ip.close # close file
        # Attributes
        self.V = len(raw)  # No. of vertices
        self.colour = [0] * self.V # List for assigning colours
        self.graph = [i.split() for i in raw] # splitting adjacency matrix
        self.colors = [] # for user required colors
        self.chromes= {}

    def disp_graph_colors(self): # method to display assigned colors
        chromatic= max(self.colour) # Chromatic number of graph
        print(f'Chromatic number of given graph is {chromatic}')
        print(f'Enter {chromatic} colors')
        # input of user required colors
        self.colors= input("Enter the colors separated by <space>").split()
        while True:
            if len(self.colors)==chromatic:
                break
            if len(self.colors)>chromatic:
                print("Entered colors are more then required")
                dl=input('Please delete a colour: ')
                self.colors.remove(dl)
            if len(self.colors)<chromatic:
                print(f'Entered {chromatic- len(self.colors)} Less colour:')
                al=input('Enter remainig colors: ').split()
                self.colors= self.colors+ al

        assigned= {}

        for num in self.colour:
            if num in assigned:
                continue
            assigned[num]=self.colors[0]
            self.colors.pop(0)

        for c in range(self.V):
            self.chromes[c+1]=assigned[self.colour[c]]
        self.g_color()


    """
    A utility function to check if the current
    color assignment is safe for vertex v
    """
    def isSafe(self, v, c):
        for i in range(self.V): # to check edges of selected vertex 'v'
            # check  if selected vertex has any adjacent vertex of selected color
            if self.graph[v][i] == '1' and self.colour[i] == c:
                return False
        return True

    # A recursive utility function to solve map coloring  problem
    def graphColourUtil(self, v):
        if v == self.V: # end recursion when all vertices are colored
            return True

        for c in range(1,self.V+1): # selecting a color
            # check if selected vertex can be colored with selected color
            if self.isSafe(v, c) == True:
                self.colour[v] = c # color vertex with selected color
                # recursion by selecting next vertex
                if self.graphColourUtil(v + 1) == True:
                    return
                return

    def g_color(self):
        # Create a graph
        G = nx.Graph()

        # Add nodes
        for node in range(len(self.graph)):
            G.add_node(node+1)

        for i in range(self.V):
            for j in range(self.V):
                if self.graph[i][j]=='1':
                    G.add_edge(i+1, j+1)

        # Draw the graph
        nx.draw(G, with_labels=True, node_color=[self.chromes[x] for x in G.nodes()],

        # Display the graph
        plt.show()

    def graphColouring(self):
        # calling recursive method for coloring vertices
        self.graphColourUtil(0)
        self.disp_graph_colors() # Print the solution

        return True
```

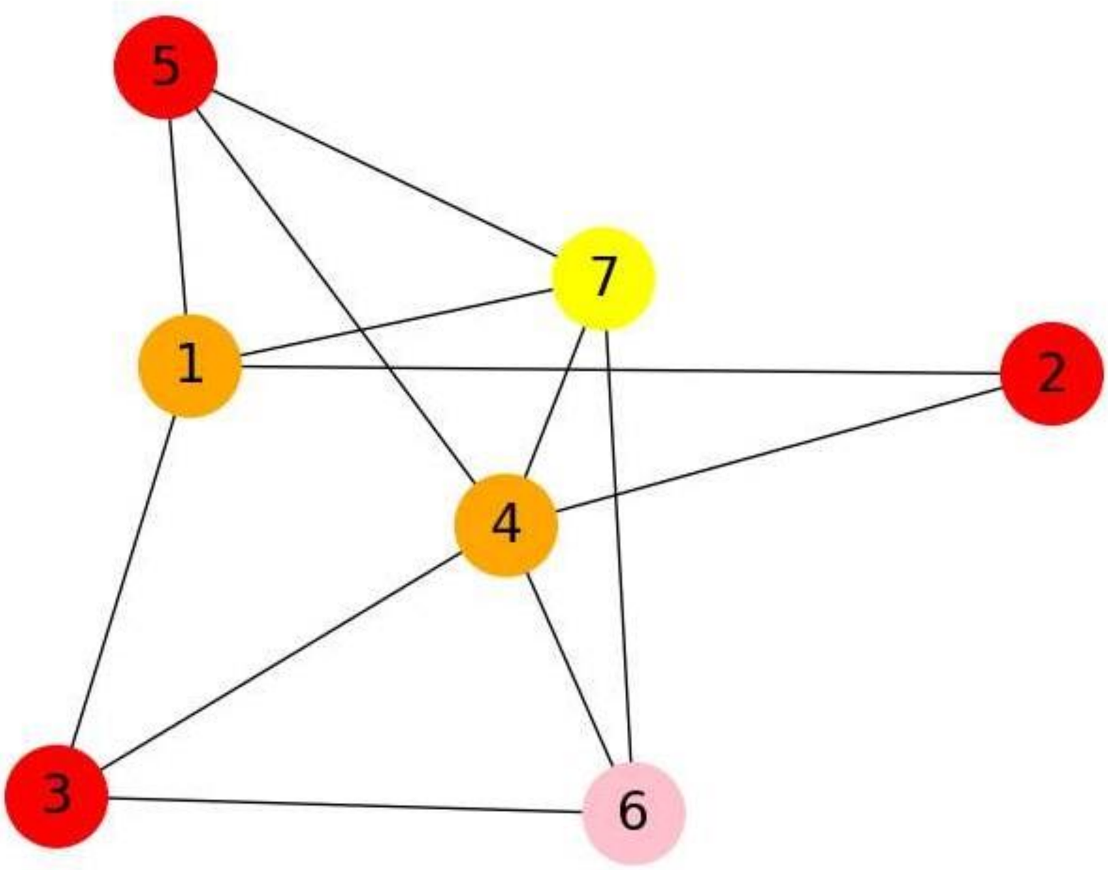
In [50]:

```
# Menu
while True:
    print()
    print("Menu:")
    print()
    print("1. N Queens")
    print("2. Graph Coloring")
    print('3. Exit')
    print()
    choice= int(input('Enter your choice: ')) # select choice
    if choice==1:
        # input for number of queens
        N = int(input("Enter the number of queens: "))
        Q= Queen(N) # constructor for object initialization
        Q.N_queen(N) # calling main method for NQueens
        print('Final State:')
        Q.disp_board()
    elif choice==2:
        # Input for filename of graph
        fname= input("Enter the name of file for input graph: ")
        g = Graph(fname) # constructor for object initialization
        g.graphColouring() # calling main method  for graph coloring

    elif choice==3: # Exit loop
        print('Thank You! ')
        break

    else:
        print('Invalid Input, please select one of the given options')
```

```
Menu:

1. N Queens
2. Graph Coloring
3. Exit

Enter your choice: 1
Enter the number of queens: 7
----
 Current State:

❌ ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
----
 Current State:

👑 ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
----
 Current State:

👑 ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ 👑 ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
----
 Current State:

👑 ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ 👑 ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ 👑 ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
----
 Current State:

👑 ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ 👑 ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ 👑 ❌ ❌
❌ 👑 ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
----
 Current State:
```

```
👑 ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ 👑 ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ 👑 ❌ ❌
❌ 👑 ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ 👑 ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
----
 Current State:

👑 ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ 👑 ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ 👑 ❌ ❌
❌ 👑 ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ 👑 ❌ ❌ ❌
----
 Current State:

👑 ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ 👑 ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ 👑 ❌ ❌
❌ 👑 ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ 👑 ❌
----
 Current State:

👑 ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ 👑 ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ 👑 ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ 👑
❌ ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
----
 Current State:

👑 ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ 👑 ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ 👑 ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ 👑
❌ 👑 ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
----
 Current State:

👑 ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ 👑 ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ 👑 ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ 👑
❌ 👑 ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ 👑 ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ ❌
Final State:
```

```
👑 ❌ ❌ ❌ ❌ ❌ ❌
❌ ❌ 👑 ❌ ❌ ❌ ❌
❌ ❌ ❌ ❌ 👑 ❌ ❌
❌ ❌ ❌ ❌ ❌ ❌ 👑
❌ 👑 ❌ ❌ ❌ ❌ ❌
❌ ❌ ❌ 👑 ❌ ❌ ❌
❌ ❌ ❌ ❌ ❌ 👑 ❌

Menu:

1. N Queens
2. Graph Coloring
3. Exit

Enter your choice: 2
Enter the name of file for input graph: g.txt
Chromatic number of given graph is 4
Enter 4 colors
Enter the colors separated by <space>orange red pink
Entered 1 Less colour:
Enter remainig colors: yellow green
Entered colors are more then required
Please delete a colour: green
```



```
Menu:

1. N Queens
2. Graph Coloring
3. Exit

Enter your choice: 3
Thank You!
```