Généré par ChatGPT (qui le génère lui-même sans révéler ses sources :-))

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

# Create an empty graph
G = nx.Graph()

# Add vertices (nodes) to the graph
G.add_nodes_from([1, 2, 3, 4, 5])

# Add edges to the graph
G.add_edges_from([(1, 2), (2, 3), (3, 4), (4, 5), (5, 1)])

# Draw the graph
nx.draw(G, with_labels=True, node_color='lightblue', edge_color='gray')

# Show the graph
plt.show()
```

Généré par ChatGPT (qui le génère lui-même sans révéler ses sources :-))

```
class Graph:
    def __init__(self, vertices):
        self.vertices = vertices
        self.adjMatrix = [[0] * vertices for _ in range(vertices)]

    def add_edge(self, u, v):
        self.adjMatrix[u][v] = 1
        self.adjMatrix[v][u] = 1

    def is_safe(self, v, color, c):
        for i in range(self.vertices):
            if self.adjMatrix[v][i] == 1 and color[i] == c:
                return False
        return True

    def graph_coloring_util(self, m, color, v):
        if v == self.vertices:
            return True

        for c in range(1, m+1):
            if self.is_safe(v, color, c):
                color[v] = c
                if self.graph_coloring_util(m, color, v+1):
                    return True
                color[v] = 0

    def graph_coloring(self, m):
        color = [0] * self.vertices
        if not self.graph_coloring_util(m, color, 0):
            return False

        print("Graph can be colored using {} colors:".format(m))
        for i in range(self.vertices):
            print("Vertex {}: Color {}".format(i, color[i]))

        return True
```

To use this code, you can create a `Graph` object, add edges, and call the `graph_coloring` method to find a valid graph coloration. Here's an example usage:

```python
g = Graph(5)

g.add_edge(0, 1)
g.add_edge(0, 2)
g.add_edge(1, 2)
g.add_edge(1, 3)
g.add_edge(2, 3)
g.add_edge(3, 4)

m = 3  # Number of colors
g.graph_coloring(m)
```

Généré par ChatGPT (qui le génère lui-même sans révéler ses sources :-))

```python
class Graph:
    def __init__(self, vertices):
        self.vertices = vertices
        self.adjMatrix = [[0] * vertices for _ in range(vertices)]

    def add_edge(self, u, v):
        self.adjMatrix[u][v] = 1
        self.adjMatrix[v][u] = 1

    def welsh_powell(self):
        # Initialize the degree of each vertex
        degree = [sum(row) for row in self.adjMatrix]
        # Create a list of (vertex, degree) tuples
        vertices_degrees = [(i, degree[i]) for i in range(self.vertices)]
        # Sort the vertices in decreasing order of degrees
        sorted_vertices = sorted(vertices_degrees, key=lambda x: x[1], reverse=True)

        color = [-1] * self.vertices
        current_color = 0

        # Assign the first color to the first vertex
        color[sorted_vertices[0][0]] = current_color

        # Color the remaining vertices
        for vertex, _ in sorted_vertices[1:]:
            if color[vertex] == -1:
                # Find an uncolored adjacent vertex
                uncolored_adjacent = [adj_vertex for adj_vertex in range(self.vertices) if
                                      self.adjMatrix[vertex][adj_vertex] == 1 and color[adj_vertex] == -1]
                # Assign a different color to the uncolored adjacent vertices
                for adj_vertex in uncolored_adjacent:
                    if not any(self.adjMatrix[adj_vertex][v] == 1 and color[v] == current_color for v in
                               range(self.vertices)):
                        color[vertex] = current_color
                        break
                # If no adjacent vertex has the current color, assign the next color
                if color[vertex] == -1:
                    current_color += 1
                    color[vertex] = current_color

        num_colors = max(color) + 1

        print("Graph colored using {} colors:".format(num_colors))
        for i in range(self.vertices):
            print("Vertex {}: Color {}".format(i, color[i]))

        return num_colors
```

Généré par ChatGPT (qui le génère lui-même sans révéler ses sources :-))

```c
#include <stdio.h>
#define MAX_VERTICES 100

struct Graph {
    int vertices;
    int adjMatrix[MAX_VERTICES][MAX_VERTICES];
};

void addEdge(struct Graph* graph, int u, int v) {
    graph->adjMatrix[u][v] = 1;
    graph->adjMatrix[v][u] = 1;
}

void welshPowell(struct Graph* graph) {
    int degree[MAX_VERTICES] = {0};
    int i, j, current_color;

    // Calculate the degree of each vertex
    for (i = 0; i < graph->vertices; i++) {
        for (j = 0; j < graph->vertices; j++) {
            if (graph->adjMatrix[i][j] == 1)
                degree[i]++;
        }
    }

    int sorted_vertices[MAX_VERTICES];

    // Initialize vertices in sorted order of degrees
    for (i = 0; i < graph->vertices; i++)
        sorted_vertices[i] = i;

    // Sort vertices in decreasing order of degree using bubble sort
    for (i = 0; i < graph->vertices - 1; i++) {
        for (j = 0; j < graph->vertices - i - 1; j++) {
            if (degree[sorted_vertices[j]] < degree[sorted_vertices[j + 1]]) {
                int temp = sorted_vertices[j];
                sorted_vertices[j] = sorted_vertices[j + 1];
                sorted_vertices[j + 1] = temp;
            }
        }
    }

    int color[MAX_VERTICES] = {-1};

    current_color = 0;
    color[sorted_vertices[0]] = current_color;

    // Assign colors to the remaining vertices
    for (i = 1; i < graph->vertices; i++) {
        int vertex = sorted_vertices[i];
        if (color[vertex] == -1) {
            int uncolored_adjacent = 1;
            for (j = 0; j < graph->vertices; j++) {
                if (graph->adjMatrix[vertex][j] == 1 && color[j] == current_color) {
                    uncolored_adjacent = 0;
                    break;
                }
            }
            // Assign a different color to the uncolored adjacent vertices
            if (uncolored_adjacent) {
                color[vertex] = current_color;
            } else {
                current_color++;
                color[vertex] = current_color;
            }
        }
    }

    int num_colors = current_color + 1;

    printf("Graph colored using %d colors:\n", num_colors);
    for (i = 0; i < graph->vertices; i++) {
        printf("Vertex %d: Color %d\n", i, color[i]);
    }
}

int main() {
    struct Graph graph;
    graph.vertices = 5;

    // Add edges
    addEdge(&graph, 0, 1);
    addEdge(&graph, 0, 2);
    addEdge(&graph, 1, 2);
    addEdge(&graph, 1, 3);
    addEdge(&graph, 2, 3);
    addEdge(&graph, 3, 4);

    welshPowell(&graph);

    return 0;
}
```