Problem #01: Let A be the set {1, 2, 3, 4}. Write a program to find the ordered pairs are in the relation $\mathbf{R1} = \{(\mathbf{a}, \mathbf{b}) \mid \mathbf{a\ divides\ b}\}$ $\mathbf{R2} = \{(\mathbf{a}, \mathbf{b}) \mid \mathbf{a} \leq \mathbf{b}\}$

```
A = {1, 2, 3, 4}          # Define the set A


R1 = set()                        # Initialize the relations R1 and R2 as empty sets

R2 = set()


for a in A:                           # Iterate through all possible pairs of elements from set A

 for b in A:

    if a != 0 and b % a == 0:                 # Check if a divides b (R1 condition)

   R1.add((a, b))


    if a <= b:                                   # Check if a is less than or equal to b (R2 condition)

      R2.add((a, b))


print("Ordered pairs in relation R1:")             # Print the ordered pairs in relations R1 and R2

for pair in R1:

  print(pair)


print("\nOrdered pairs in relation R2:")

for pair in R2:

  print(pair)
```

Problem2:Suppose that $\mathbf{A} = \{\mathbf{1}, \mathbf{2}, \mathbf{3}\}$ and $\mathbf{B} = \{\mathbf{1}, \mathbf{2}\}$. Let R be the relation from A to B containing (a, b) if $\mathbf{a} \in \mathbf{A}$, $\mathbf{b} \in \mathbf{B}$ and $\mathbf{a} > \mathbf{b}$. Write a program to find the relation R and also represent this relation in matrix form.

```
# Define sets A and B

A = {1, 2, 3}

B = {1, 2}
```

```
relation_R = []                    # Initialize an empty list to store the relation R

for a in A:                # Find the relation R
   for b in B:
      if a > b:
         relation_R.append((a, b))

print("Relation R:")                  # Print the relation R as a set of ordered pairs
for pair in relation_R:
   print(pair)
                                                  # Create a matrix representation of the relation R
matrix_R = [[1 if (a, b) in relation_R else 0 for b in B] for a in A]

print("\nMatrix Representation of R:")        # Print the matrix representation of the relation R
for row in matrix_R:
   print(row)
```

Problem3: Suppose that the relations R1 and R2 on a set A are represented by the matrices MR1 = [1 1 1 1 0 1 0 1 0] and MR2 = [1 0 1 0 1 1 1 1 0]. Write a program to find the MR1∪R2 and MR1⊕R2 .

```
import numpy as np

MR1 = np.array([              # Define the matrices for MR1 and MR2
   [1, 1, 1],
   [1, 0, 1],
   [0, 1, 0]
])
```

```python
MR2 = np.array([
    [1, 0, 1],
    [0, 1, 1],
    [1, 1, 0]
])
                                            # Calculate the union (MR1 ∪ MR2) by element-wise logical OR
union_result = np.logical_or(MR1, MR2).astype(int)
                                   # Calculate the symmetric difference (MR1 ⊕ MR2) by element-wise XOR
symmetric_diff_result = np.logical_xor(MR1, MR2).astype(int)

print("MR1 ∪ MR2 (Union):")        # Print the results
print(union_result)

print("\nMR1 ⊕ MR2 (Symmetric Difference):")
print(symmetric_diff_result)
```

Problem4: Write a program to find shortest path by Warshall's algorithm.

```python
INF=1000000000
def floyd_warshall(graph):          # Function to find the shortest path using Warshall's algorithm
    num_vertices = len(graph)

    dist = [row[:] for row in graph]          # Create a copy of the graph to store the shortest distances

                                   # Consider each vertex as an intermediate vertex and update the distances
    for k in range(num_vertices):
```

```
        for i in range(num_vertices):
            for j in range(num_vertices):
                dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j])

    return dist

                                                  # Example graph represented as an adjacency matrix
graph = [                               # Replace the values with float('inf') for unconnected vertices
[ 0, 5, INF, 10],
 [ INF, 0, 3, INF],
 [ INF, INF, 0, 1],
[INF, INF, INF, 0]
 ]
shortest_distances = floyd_warshall(graph)

                                                  # Print the shortest distances between all pairs of vertices
for i in range(len(shortest_distances)):
    for j in range(len(shortest_distances[i])):
        print(f"Shortest distance from vertex {i} to vertex {j}: {shortest_distances[i][j]}")
```

Problem5: Write a program for the solution of graph coloring problem by Welch-Powell's algorithm.

```
def color_nodes(graph):
    color_map = {}
                                                  # Consider nodes in descending degree
    for node in sorted(graph, key=lambda x: len(graph[x]), reverse=True):
        neighbor_colors = set(color_map.get(neigh) for neigh in graph[node])
        color_map[node] = next(
```

```
            color for color in range(len(graph)) if color not in neighbor_colors
        )
    return color_map


                                                                # Adjacent list
graph = {'a': list('bcd'), 'b': list('ac'), 'c': list('abdef'), 'd': list('ace'), 'e': list('cdf'), 'f': list('ce')}
print(color_nodes(graph))
```