```
# matrices from the document
R1 = [[1,0,1],[0,1,0],[1,0,0]]
R2 = [[1,1,0],[0,0,1],[1,0,1]]
R3 = [[1, 1, 1], [0, 0, 1], [0, 0, 1]]
R4 = [[1, 1, 0, 1], [1, 1, 0, 1], [0, 0, 1, 0], [1, 1, 0, 1]]
R5 = [[0, 1, 0], [0, 0, 1], [1, 0, 0]]
R6 = [[1, 1, 1], [0, 0, 1], [1, 0, 0]]

tests = [R1, R2, R3, R4, R5, R6]

CHAMPIONS = [[1,1,0,1,1],
             [0,1,0,0,0],
             [1,1,1,1,1],
             [1,1,0,1,1],
             [0,1,0,0,1]]
```

```
# Check if a matrix is reflexive
def reflexive(R):
    for i in range(len(R)):
        if R[i][i] != 1:
            return False
    return True

# Check if a matrix is symmetric
def symmetrical(R):
    for i in range(len(R)):
        for j in range(len(R)):
            if R[i][j] != R[j][i]:
                return False
    return True

# Check if a matrix is anti-symmetric
def anti_symmetrical(R):
    for i in range(len(R)):
        for j in range(len(R)):
            if R[i][j] > 0 and R[j][i] > 0 and i != j:
                return False
    return True

# Check if a matrix is asymmetrical
def asymmetrical(R):
    for i in range(len(R)):
        for j in range(len(R)):
            if R[i][j] > 0 and R[j][i] > 0:
                return False
    return True

# Check if a matrix is irreflexive
def irreflexive(R):
    for i in range(len(R)):
```

```
            if R[i][i] > 0:
                return False
        return True

    # Check if a matrix is transitive
    def transitive(matrix):
        is_transitive = True
        for i in range(len(matrix)):
            for j in range(len(matrix)):
                if matrix[i][j] > 0:
                    for k in range(len(matrix)):
                        if matrix[j][k] > 0:
                            is_transitive = is_transitive and
    matrix[i][k] > 0

        return is_transitive

    # Check if a matrix is intransitive
    def intransitive(matrix):
        is_transitive = True
        for i in range(len(matrix)):
            for j in range(len(matrix)):
                if matrix[i][j] > 0:
                    for k in range(len(matrix)):
                        if matrix[j][k] > 0:
                            is_transitive = is_transitive and
    matrix[i][k] != 1

        return is_transitive

    # Check if a matrix is non-transitive
    def non_transitive(matrix):
        n = len(matrix)

        for i in range(n):
            for j in range(n):
                if matrix[i][j] > 0:
                    for k in range(n):
                        if matrix[j][k] > 0 and matrix[i][k] ==
    0:
                            return True
        return False
```

```
    # Testing all the matrices from the slides

    for i in range(len(tests)):
        print("-------------------------------------------------
    ")
        print("Matrix: R", i+1)
        a = reflexive(tests[i])
        print("reflexive: ", a)
        a = irreflexive(tests[i])
```

```
    print("irreflexive: ", a)
    a = symmetrical(tests[i])
    print("symmetrical: ", a)
    a = anti_symmetrical(tests[i])
    print("anti_symmetrical: ", a)
    a = asymmetrical(tests[i])
    print("asymmetrical: ", a)
    a = transitive(tests[i])
    print("transitive: ", a)
    a = intransitive(tests[i])
    print("intransitive: ", a)
    a = non_transitive(tests[i])
    print("non_transitive: ", a)
```

```
--------------------------------------------------
Matrix: R 1
reflexive:  False
irreflexive:  False
symmetrical:  True
anti_symmetrical:  False
asymmetrical:  False
transitive:  False
intransitive:  False
non_transitive:  True
--------------------------------------------------
Matrix: R 2
reflexive:  False
irreflexive:  False
symmetrical:  False
anti_symmetrical:  True
asymmetrical:  False
transitive:  False
intransitive:  False
non_transitive:  True
--------------------------------------------------
Matrix: R 3
reflexive:  False
irreflexive:  False
symmetrical:  False
anti_symmetrical:  True
asymmetrical:  False
transitive:  True
intransitive:  False
non_transitive:  False
--------------------------------------------------
Matrix: R 4
reflexive:  True
irreflexive:  False
symmetrical:  True
anti_symmetrical:  False
asymmetrical:  False
transitive:  True
```

```
intransitive:  False
non_transitive:  False
--------------------------------------------------
Matrix: R 5
reflexive:  False
irreflexive:  True
symmetrical:  False
anti_symmetrical:  True
asymmetrical:  True
transitive:  False
intransitive:  True
non_transitive:  True
--------------------------------------------------
Matrix: R 6
reflexive:  False
irreflexive:  False
symmetrical:  False
anti_symmetrical:  False
asymmetrical:  False
transitive:  False
intransitive:  False
non_transitive:  True
```

```python
# Equivalence classes

def equivalence_classes(R):
    n = len(R)
    classes = []
    scores = []

    # Count number of (x,y) existing on each row
    for x in range(n):
        score = 0
        for y in range(n):
            if R[x][y] > 0:
                score += 1
        scores.append(score)

    lastScore = 0

    for i in range(n):
        max_score = 0
        max_index = 0
        for j in range(n):
            if scores[j] > max_score:
                max_score = scores[j]
                max_index = j

        if(max_score == lastScore):
            classes[len(classes)-1].append(max_index)
        else:
            # Add the class to the list of classes
```

```
            classes.append([max_index])

        lastScore = max_score
        scores[max_index] = 0

    return classes, scores
```

```
print(equivalence_classes(CHAMPIONS))
```

```
([[2], [0, 3], [4], [1]], [0, 0, 0, 0, 0])
```

```
def transitive_closure(adjacency_matrix):
    """
    Calculate the transitive closure of a directed graph
represented as an adjacency matrix.

    Args:
    adjacency_matrix (list of lists): The adjacency matrix of
the graph.

    Returns:
    list of lists: The transitive closure matrix.
    """
    n = len(adjacency_matrix)

    # Initialize the transitive closure matrix as a copy of
the adjacency matrix
    closure_matrix = [row[:] for row in adjacency_matrix]

    # Perform the Floyd-Warshall algorithm to compute
transitive closure
    for k in range(n):
        for i in range(n):
            for j in range(n):
                closure_matrix[i][j] = closure_matrix[i][j]
or (closure_matrix[i][k] and closure_matrix[k][j])

    return closure_matrix

# Example usage:
adjacency_matrix = [
    [0, 1, 0, 0],
    [0, 0, 0, 1],
    [1, 0, 0, 0],
    [0, 0, 1, 0]
]
```

```
ex_cours = [
    [0, 0, 1, 0],
    [0, 0, 0, 0],
    [0, 1, 1, 1],
    [1, 0, 0, 0]
]

transitive_closure_matrix = transitive_closure(ex_cours)

# Print the transitive closure matrix
for row in transitive_closure_matrix:
    print(row)
```

```
[1, 1, 1, 1]
[0, 0, 0, 0]
[1, 1, 1, 1]
[1, 1, 1, 1]
```

```
# Make transitive closure

transitive_closure_matrix = [
    [1, 0, 0, 0],
    [0, 1, 0, 1],
    [1, 0, 0, 0],
    [0, 0, 1, 1]
]

def transitive_closure(matrix):
    modification = True
    while modification:
        modification = False
        for i in range(len(matrix)):
            for j in range(len(matrix)):
                for k in range(len(matrix)):
                    if matrix[i][j] > 0 and matrix[j][k] > 0
and matrix[i][k] == 0:
                        matrix[i][k] = matrix[i][j] +
matrix[j][k]
                        modification = True
                    else:
                        modification = False

    return matrix

full_closure = transitive_closure(ex_cours)

for row in full_closure:
    print(row)
```

```
[3, 2, 1, 2]
[0, 0, 0, 0]
[2, 1, 1, 1]
[1, 3, 2, 3]
```

```
# Remove transitive closure

transitive_closure_matrix = [
[1, 0, 0, 0],
[0, 1, 2, 1],
[1, 0, 0, 0],
[2, 0, 1, 1]
]

m = [
    [1,1,1,1,1,1,1],
    [0,1,0,0,1,1,1],
    [0,0,1,0,0,0,1],
    [0,0,0,1,1,1,1],
    [0,0,0,0,1,1,1],
    [0,0,0,0,0,1,1],
    [0,0,0,0,0,0,1]
]

def remove_transitive_closure(matrix):
    n = len(matrix)
    modification = True

    while modification:
        # reset modification
        modification = False
        for i in range(n):
            for j in range(n):
                # checking if there is a path from i to j
                if matrix[i][j] > 0:
                    for k in range(j,n):
                        # checking if there is a path from i
k to  to k
                        if matrix[i][k] > 0 and matrix[j][k]
> 0:
                            matrix[i][k] = 0
                            modification = True
                            print("Removing ", i, k)
                            # restart while loop
                            break
                    if modification:
                        break

    return matrix

no_closure =
```

```
remove_transitive_closure(transitive_closure_matrix)

for row in no_closure:
    print(row)

no_closure = remove_transitive_closure(m)

for row in no_closure:
    print(row)
```

```
Removing  0 0
Removing  1 1
Removing  1 3
Removing  3 3
[0, 0, 0, 0]
[0, 0, 2, 0]
[1, 0, 0, 0]
[2, 0, 1, 0]
Removing  0 0
Removing  1 1
Removing  2 2
Removing  3 3
Removing  4 4
Removing  5 5
Removing  6 6
Removing  0 4
Removing  1 5
Removing  3 5
Removing  4 6
Removing  0 6
[0, 1, 1, 1, 0, 1, 0]
[0, 0, 0, 0, 1, 0, 1]
[0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 1, 0, 1]
[0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 0]
```