Dhruv Roy Talukdar

3rd Year

***Sub:* Soft Computing**

Assignment 1

(Sample output is mentioned in each code snippet as comments)

Q1) Crisp Composition by Max-Min Method

```python
def inputMatrix(which):
    row = int(input(f"Enter the number of rows for the {which}
matrix:\n"))
    matrix = []
    for i in range(0, row):
        mat_row = [int(a) for a in input(
            f"Enter {i+1}th row as space separated
list:\n").split(" ")]
        matrix.append(mat_row)
    return matrix


matrix1 = inputMatrix("first")
print("")
matrix2 = inputMatrix("second")


def getMaxMinComposition(matrix1, matrix2):
    row1 = len(matrix1)
    col2 = len(matrix2[0])
    row2 = len(matrix2)

    matrix3 = []
    for i in range(0, row1):
        li = []
        for j in range(0, col2):
            res = 0
            for k in range(0, row2):
                res = max(res, min(matrix1[i][k],
matrix2[k][j]))
            li.append(res)
        matrix3.append(li)
    return matrix3


print("\nThe resultant matrix:\n",
getMaxMinComposition(matrix1, matrix2))

# SAMPLE RUN
############
# Inputs
# ------

# Enter the number of rows for the first matrix:
# 3
# Enter 1th row as space separated list:
# 1 0 1 0
# Enter 2th row as space separated list:
# 0 0 0 1
# Enter 3th row as space separated list:
# 0 0 0 0

# Enter the number of rows for the second matrix:
# 4
# Enter 1th row as space separated list:
# 0 1
# Enter 2th row as space separated list:
# 0 0
# Enter 3th row as space separated list:
# 0 1
# Enter 4th row as space separated list:
# 0 0

# Outputs
# -------
# The resultant matrix:
#  [[0, 1], [0, 0], [0, 0]]
```

Q2) Crisp Composition Max-Dot Method

```
def inputMatrix(which):
    row = int(input(f"Enter the number of rows for the {which}
matrix:\n"))
    matrix = []
    for i in range(0, row):
        mat_row = [int(a) for a in input(
            f"Enter {i+1}th row as space separated
list:\n").split(" ")]
        matrix.append(mat_row)
    return matrix


matrix1 = inputMatrix("first")
print("")
matrix2 = inputMatrix("second")


def getMaxDotComposition(matrix1, matrix2):
    row1 = len(matrix1)
    col2 = len(matrix2[0])
    row2 = len(matrix2)

    matrix3 = []
    for i in range(0, row1):
        li = []
        for j in range(0, col2):
            res = 0
            for k in range(0, row2):
                res = max(res, matrix1[i][k] * matrix2[k][j])
            li.append(res)
        matrix3.append(li)
    return matrix3


print("\nThe resultant matrix:\n",
getMaxDotComposition(matrix1, matrix2))

# SAMPLE RUN
############

# Inputs
# ------
# Enter the number of rows for the first matrix:
# 3
# Enter 1th row as space separated list:
# 1 0 1 0
# Enter 2th row as space separated list:
# 0 0 0 1
# Enter 3th row as space separated list:
# 0 0 0 0

# Enter the number of rows for the second matrix:
# 4
# Enter 1th row as space separated list:
# 0 1
# Enter 2th row as space separated list:
# 0 0
# Enter 3th row as space separated list:
# 0 1
# Enter 4th row as space separated list:
# 0 0

# Outputs
# --------
# The resultant matrix:
#  [[0, 1], [0, 0], [0, 0]]
```

Q3) Fuzzy Composition Max-Min Method

```
def inputMatrix(which):
    row = int(input(f"Enter the number of rows for the {which}
matrix:\n"))
    matrix = []
    for i in range(0, row):
        mat_row = [float(a) for a in input(
            f"Enter {i+1}th row as space separated
list:\n").split(" ")]
        matrix.append(mat_row)
    return matrix
```

```
matrix1 = inputMatrix("first")
print("")
matrix2 = inputMatrix("second")


def getMaxMinComposition(matrix1, matrix2):
    row1 = len(matrix1)
    col2 = len(matrix2[0])
    row2 = len(matrix2)

    matrix3 = []
    for i in range(0, row1):
        li = []
        for j in range(0, col2):
            res = 0
            for k in range(0, row2):
                res = max(res, min(matrix1[i][k],
matrix2[k][j]))
            li.append(res)
        matrix3.append(li)
    return matrix3


print("\nThe resultant matrix:\n",
getMaxMinComposition(matrix1, matrix2))

# SAMPLE OUTPUT
###############

# Inputs
# ------
# Enter the number of rows for the first matrix:
# 2
# Enter 1th row as space separated list:
# 0.7 0.5 0.2
# Enter 2th row as space separated list:
# 0.1 0.4 0.6

# Enter the number of rows for the second matrix:
# 2
# Enter 1th row as space separated list:
# 0.2 0.9
# Enter 2th row as space separated list:
# 0.7 0.1

# Outputs
# --------
# The resultant matrix:
#  [[0.5, 0.7], [0.4, 0.1]]
```

Q4) Fuzzy Composition Max-Dot Method

```
def inputMatrix(which):
    row = int(input(f"Enter the number of rows for the {which}
matrix:\n"))
    matrix = []
    for i in range(0, row):
        mat_row = [float(a) for a in input(
            f"Enter {i+1}th row as space separated
list:\n").split(" ")]
        matrix.append(mat_row)
    return matrix


matrix1 = inputMatrix("first")
print("")
matrix2 = inputMatrix("second")


def getMaxDotComposition(matrix1, matrix2):
    row1 = len(matrix1)
    col2 = len(matrix2[0])
    row2 = len(matrix2)

    matrix3 = []
    for i in range(0, row1):
        li = []
        for j in range(0, col2):
            res = 0
            for k in range(0, row2):
                res = max(res, matrix1[i][k] * matrix2[k][j])
            li.append(res)
        matrix3.append(li)
    return matrix3


print("\nThe resultant matrix:\n",
getMaxDotComposition(matrix1, matrix2))

# SAMPLE OUTPUT
###############

# Inputs
# ------
# Enter the number of rows for the first matrix:
# 3
# Enter 1th row as space separated list:
# 0.8 0.9 0.6
# Enter 2th row as space separated list:
# 0.2 0.1 0.4
# Enter 3th row as space separated list:
# 0.9 0.2 0.5

# Enter the number of rows for the second matrix:
# 3
# Enter 1th row as space separated list:
# 0.2 0.9
# Enter 2th row as space separated list:
# 0.7 0.5
# Enter 3th row as space separated list:
# 0.8 0.7

# Outputs
# -------
# The resultant matrix:
#  [[0.63, 0.7200000000000001], [0.32000000000000006,
0.27999999999999997], [0.4, 0.81]]
```

Q5) Crisp Equivalence

```
r = int(input("Enter number of rows: "))
```

```python
mat = []
print("Enter the matrix")
for i in range(0, r):
    mat.append([float(i) for i in input().split(" ")])


def checkReflexive(mat):
    r = len(mat)
    for i in range(0, r):
        if mat[i][i] != 1:
            print("The relation is not reflexive")
            return False
    return True


def checkSymmetric(mat):
    r = len(mat)
    for i in range(0, r):
        for j in range(0, r):
            if mat[i][j] == 1 and mat[j][i] != 1:
                print("The relation is not symmetric")
                return False
    return True


def checkTransitive(mat):
    r = len(mat)
    for p1 in range(0, r):
        for p2 in range(0, r):
            for p3 in range(0, r):
                if p1 != p2 and p2 != p3 and p1 != p3:
                    if mat[p1][p2] == 1 and mat[p2][p3] == 1
and mat[p1][p3] != 1:
                        print(
                            f"The relation is not transitive
at {p1, p2} and {p2,p3}")
                        return False
    return True


def checkEquivalence(mat):
    return checkReflexive(mat) and checkSymmetric(mat) and
checkTransitive(mat)
```

```
print("The relation is an equivalence relation." if
checkEquivalence(
    mat) else "The relation is not an equivalence relation.")

# Sample Input:
# ==============
# Enter number of rows: 5
# Enter the matrix
# 1 1 0 0 1
# 1 1 0 0 1
# 0 0 1 1 0
# 0 0 1 1 0
# 1 1 0 0 1

# Sample Output:
# ===============
# The relation is a equivalence relation.
```

Q6) Fuzzy Equivalence

```
r = int(input("Enter number of rows: "))

mat = []
print("Enter the matrix")
for i in range(0, r):
    mat.append([float(i) for i in input().split(" ")])


def checkReflexive(mat):
    r = len(mat)
    for i in range(0, r):
        if mat[i][i] != 1:
            print("The relation is not reflexive")
            return False
    return True


def checkSymmetric(mat):
    r = len(mat)
```

```python
    for i in range(0, r):
        for j in range(0, r):
            if mat[i][j] != mat[j][i]:
                print("The relation is not symmetric")
                return False
    return True


def checkTransitive(mat):
    r = len(mat)
    for p1 in range(0, r):
        for p2 in range(0, r):
            for p3 in range(0, r):
                if p1 != p2 and p2 != p3 and p1 != p3:
                    if mat[p1][p3] < min(mat[p1][p2],
mat[p2][p3]):
                        print(
                            f"The relation is not transitive
at {p1, p2} and {p2,p3}")
                        return False
    return True


def checkEquivalence(mat):
    return checkReflexive(mat) and checkSymmetric(mat) and
checkTransitive(mat)

print("The relation is an equivalence relation." if
checkEquivalence(
    mat) else "The relation is not an equivalence relation.")

# Sample Input:
# =============
# Enter number of rows: 3
# Enter the matrix
# 1 0.8 1
# 0.8 1 0.4
# 1 0.4 1
# Sample Output:
# ==============
# The relation is not transitive at (1, 0) and (0, 2)
# The relation is not an equivalence relation
```