Code for problem:

```
import numpy as np
# Sayantani Karmakar, 20CS8024
def min_zero_row(zero_mat, mark_zero):

    '''
    The function can be splitted into two steps:
    #1 The function is used to find the row which containing the fewest 0.
    #2 Select the zero number on the row, and then marked the element
corresponding row and column as False
    '''

    #Find the row
    min_row = [99999, -1]

    for row_num in range(zero_mat.shape[0]):
        if np.sum(zero_mat[row_num] == True) > 0 and min_row[0] >
np.sum(zero_mat[row_num] == True):
            min_row = [np.sum(zero_mat[row_num] == True), row_num]

    # Marked the specific row and column as False
    zero_index = np.where(zero_mat[min_row[1]] == True)[0][0]
    mark_zero.append((min_row[1], zero_index))
    zero_mat[min_row[1], :] = False
    zero_mat[:, zero_index] = False

def mark_matrix(mat):

    '''
    Finding the returning possible solutions for LAP problem.
    '''

    #Transform the matrix to boolean matrix(0 = True, others = False)
    cur_mat = mat
    zero_bool_mat = (cur_mat == 0)
    zero_bool_mat_copy = zero_bool_mat.copy()

    #Recording possible answer positions by marked_zero
    marked_zero = []
    while (True in zero_bool_mat_copy):
        min_zero_row(zero_bool_mat_copy, marked_zero)

    #Recording the row and column positions seperately.
    marked_zero_row = []
    marked_zero_col = []
    for i in range(len(marked_zero)):
        marked_zero_row.append(marked_zero[i][0])
        marked_zero_col.append(marked_zero[i][1])

    #Step 2-2-1
    non_marked_row = list(set(range(cur_mat.shape[0])) -
```

```
set(marked_zero_row))

    marked_cols = []
    check_switch = True
    while check_switch:
        check_switch = False
        for i in range(len(non_marked_row)):
            row_array = zero_bool_mat[non_marked_row[i], :]
            for j in range(row_array.shape[0]):
                #Step 2-2-2
                if row_array[j] == True and j not in marked_cols:
                    #Step 2-2-3
                    marked_cols.append(j)
                    check_switch = True

        for row_num, col_num in marked_zero:
            #Step 2-2-4
            if row_num not in non_marked_row and col_num in marked_cols:
                #Step 2-2-5
                non_marked_row.append(row_num)
                check_switch = True
    #Step 2-2-6
    marked_rows = list(set(range(mat.shape[0])) - set(non_marked_row))

    return(marked_zero, marked_rows, marked_cols)
# Sayantani Karmakar, 20CS8024
def adjust_matrix(mat, cover_rows, cover_cols):
    cur_mat = mat
    non_zero_element = []

    #Step 4-1
    for row in range(len(cur_mat)):
        if row not in cover_rows:
            for i in range(len(cur_mat[row])):
                if i not in cover_cols:
                    non_zero_element.append(cur_mat[row][i])
    min_num = min(non_zero_element)

    #Step 4-2
    for row in range(len(cur_mat)):
        if row not in cover_rows:
            for i in range(len(cur_mat[row])):
                if i not in cover_cols:
                    cur_mat[row, i] = cur_mat[row, i] - min_num
    #Step 4-3
    for row in range(len(cover_rows)):
        for col in range(len(cover_cols)):
            cur_mat[cover_rows[row], cover_cols[col]] =
cur_mat[cover_rows[row], cover_cols[col]] + min_num
    return cur_mat

def hungarian_algorithm(mat):
    dim = mat.shape[0]
    cur_mat = mat
```

```
    #Step 1 - Every column and every row subtract its internal minimum
    for row_num in range(mat.shape[0]):
        cur_mat[row_num] = cur_mat[row_num] - np.min(cur_mat[row_num])

    for col_num in range(mat.shape[1]):
        cur_mat[:,col_num] = cur_mat[:,col_num] -
np.min(cur_mat[:,col_num])
    zero_count = 0
    while zero_count < dim:
        #Step 2 & 3
        ans_pos, marked_rows, marked_cols = mark_matrix(cur_mat)
        zero_count = len(marked_rows) + len(marked_cols)

        if zero_count < dim:
            cur_mat = adjust_matrix(cur_mat, marked_rows, marked_cols)

    return ans_pos
# Sayantani Karmakar, 20CS8024
def ans_calculation(mat, pos):
    total = 0
    fill_value = "X"
    ans_mat = np.full((mat.shape[0], mat.shape[1]),fill_value)
    for i in range(len(pos)):
        total += mat[pos[i][0], pos[i][1]]

        ans_mat[pos[i][0], pos[i][1]] = mat[pos[i][0], pos[i][1]]
    return total, ans_mat


def main():
    print("Sayantani Karmakar, 20CS8024")
    # Input for the number of rows and columns
    n = int(input("Enter the number of rows: "))
    m = int(input("Enter the number of columns: "))

    # Check if the input is valid (n and m should be positive integers)
    if n <= 0 or m <= 0:
        print("Error: Please enter positive integers for the number of rows
and columns.")
        return

    # Initialize an empty list to store the matrix
    matrix_rows = []

    # Input for the matrix values row-wise
    for i in range(n):
        row = input(f"Enter values for row {i+1} separated by spaces: ")
        row_values = list(map(int, row.split()))

        # Check if each row has m values
        if len(row_values) != m:
            print(f"Error: Row {i+1} does not have {m} values. Please enter
{m} values.")
```

```
            return

        matrix_rows.append(row_values)

    # Convert the list of rows into a NumPy matrix
    cost_matrix = np.array(matrix_rows)

    # If the number of rows is not equal to the number of columns, add
dummy rows or columns with zero values
    if n < m:
        dummy_rows = m - n
        dummy_data = np.zeros((dummy_rows, m))
        cost_matrix = np.vstack((cost_matrix, dummy_data))
    elif n > m:
        dummy_cols = n - m
        dummy_data = np.zeros((n, dummy_cols))
        cost_matrix = np.hstack((cost_matrix, dummy_data))


    ans_pos = hungarian_algorithm(cost_matrix.copy())
    ans, ans_mat = ans_calculation(cost_matrix, ans_pos)

    # Show the result
    print(f"Linear Assignment problem result: {ans:.0f}\n{ans_mat}")


if __name__ == '__main__':
    main()
```

Output for problem 1:

```
Sayantani Karmakar, 20CS8024
Enter the number of rows: 8
Enter the number of columns: 6
Enter values for row 1 separated by spaces: 5 8 7 6 6 7
Enter values for row 2 separated by spaces: 8 7 8 8 7 6
Enter values for row 3 separated by spaces: 6 9 8 9 9 8
Enter values for row 4 separated by spaces: 7 5 6 6 6 7
Enter values for row 5 separated by spaces: 6 7 5 5 5 6
Enter values for row 6 separated by spaces: 9 7 6 6 6 6
Enter values for row 7 separated by spaces: 5 6 7 8 9 7
Enter values for row 8 separated by spaces: 8 8 9 8 7 9
Linear Assignment problem result: 33
[['X' 'X' 'X' '6' 'X' 'X' 'X' 'X']
 ['X' 'X' 'X' 'X' 'X' '6' 'X' 'X']
 ['X' 'X' 'X' 'X' 'X' 'X' '0' 'X']
 ['X' '5' 'X' 'X' 'X' 'X' 'X' 'X']
 ['X' 'X' '5' 'X' 'X' 'X' 'X' 'X']
 ['X' 'X' 'X' 'X' '6' 'X' 'X' 'X']
 ['5' 'X' 'X' 'X' 'X' 'X' 'X' 'X']
 ['X' 'X' 'X' 'X' 'X' 'X' 'X' '0']]
```

Code for problem 2:

```
import numpy as np
# Sayantani Karmakar, 20CS8024
def min_zero_row(zero_mat, mark_zero):

    '''
    The function can be splitted into two steps:
    #1 The function is used to find the row which containing the fewest 0.
    #2 Select the zero number on the row, and then marked the element
corresponding row and column as False
    '''

    #Find the row
    min_row = [99999, -1]

    for row_num in range(zero_mat.shape[0]):
        if np.sum(zero_mat[row_num] == True) > 0 and min_row[0] >
np.sum(zero_mat[row_num] == True):
            min_row = [np.sum(zero_mat[row_num] == True), row_num]

    # Marked the specific row and column as False
    zero_index = np.where(zero_mat[min_row[1]] == True)[0][0]
    mark_zero.append((min_row[1], zero_index))
    zero_mat[min_row[1], :] = False
    zero_mat[:, zero_index] = False

def mark_matrix(mat):

    '''
    Finding the returning possible solutions for LAP problem.
    '''

    #Transform the matrix to boolean matrix(0 = True, others = False)
    cur_mat = mat
    zero_bool_mat = (cur_mat == 0)
    zero_bool_mat_copy = zero_bool_mat.copy()

    #Recording possible answer positions by marked_zero
    marked_zero = []
    while (True in zero_bool_mat_copy):
        min_zero_row(zero_bool_mat_copy, marked_zero)

    #Recording the row and column positions seperately.
    marked_zero_row = []
    marked_zero_col = []
    for i in range(len(marked_zero)):
        marked_zero_row.append(marked_zero[i][0])
        marked_zero_col.append(marked_zero[i][1])

    #Step 2-2-1
    non_marked_row = list(set(range(cur_mat.shape[0])) -
```

```
set(marked_zero_row))

    marked_cols = []
    check_switch = True
    while check_switch:
        check_switch = False
        for i in range(len(non_marked_row)):
            row_array = zero_bool_mat[non_marked_row[i], :]
            for j in range(row_array.shape[0]):
                #Step 2-2-2
                if row_array[j] == True and j not in marked_cols:
                    #Step 2-2-3
                    marked_cols.append(j)
                    check_switch = True

        for row_num, col_num in marked_zero:
            #Step 2-2-4
            if row_num not in non_marked_row and col_num in marked_cols:
                #Step 2-2-5
                non_marked_row.append(row_num)
                check_switch = True
    #Step 2-2-6
    marked_rows = list(set(range(mat.shape[0])) - set(non_marked_row))

    return(marked_zero, marked_rows, marked_cols)
# Sayantani Karmakar, 20CS8024
def adjust_matrix(mat, cover_rows, cover_cols):
    cur_mat = mat
    non_zero_element = []

    #Step 4-1
    for row in range(len(cur_mat)):
        if row not in cover_rows:
            for i in range(len(cur_mat[row])):
                if i not in cover_cols:
                    non_zero_element.append(cur_mat[row][i])
    min_num = min(non_zero_element)

    #Step 4-2
    for row in range(len(cur_mat)):
        if row not in cover_rows:
            for i in range(len(cur_mat[row])):
                if i not in cover_cols:
                    cur_mat[row, i] = cur_mat[row, i] - min_num
    #Step 4-3
    for row in range(len(cover_rows)):
        for col in range(len(cover_cols)):
            cur_mat[cover_rows[row], cover_cols[col]] =
cur_mat[cover_rows[row], cover_cols[col]] + min_num
    return cur_mat

def hungarian_algorithm(mat):
    dim = mat.shape[0]
    cur_mat = mat
```

```
    #Step 1 - Every column and every row subtract its internal minimum
    for row_num in range(mat.shape[0]):
        cur_mat[row_num] = cur_mat[row_num] - np.min(cur_mat[row_num])

    for col_num in range(mat.shape[1]):
        cur_mat[:,col_num] = cur_mat[:,col_num] -
np.min(cur_mat[:,col_num])
    zero_count = 0
    while zero_count < dim:
        #Step 2 & 3
        ans_pos, marked_rows, marked_cols = mark_matrix(cur_mat)
        zero_count = len(marked_rows) + len(marked_cols)

        if zero_count < dim:
            cur_mat = adjust_matrix(cur_mat, marked_rows, marked_cols)

    return ans_pos
# Sayantani Karmakar, 20CS8024
def ans_calculation(mat, pos):
    total = 0
    fill_value = "X"
    ans_mat = np.full((mat.shape[0], mat.shape[1]),fill_value)
    for i in range(len(pos)):
        total += mat[pos[i][0], pos[i][1]]

        ans_mat[pos[i][0], pos[i][1]] = mat[pos[i][0], pos[i][1]]
    return total, ans_mat


def main():
    print("Sayantani Karmakar, 20CS8024")
    # Input for the number of rows and columns
    n = int(input("Enter the number of rows: "))
    m = int(input("Enter the number of columns: "))

    # Check if the input is valid (n and m should be positive integers)
    if n <= 0 or m <= 0:
        print("Error: Please enter positive integers for the number of rows
and columns.")
        return

    # Initialize an empty list to store the matrix
    matrix_rows = []

    # Input for the matrix values row-wise
    for i in range(n):
        row = input(f"Enter values for row {i+1} separated by spaces: ")
        row_values = list(map(int, row.split()))

        # Check if each row has m values
        if len(row_values) != m:
            print(f"Error: Row {i+1} does not have {m} values. Please enter
{m} values.")
```

```
            return

        matrix_rows.append(row_values)

    # Convert the list of rows into a NumPy matrix
    cost_matrix = np.array(matrix_rows)

    # If the number of rows is not equal to the number of columns, add
dummy rows or columns with zero values
    if n < m:
        dummy_rows = m - n
        dummy_data = np.zeros((dummy_rows, m))
        cost_matrix = np.vstack((cost_matrix, dummy_data))
    elif n > m:
        dummy_cols = n - m
        dummy_data = np.zeros((n, dummy_cols))
        cost_matrix = np.hstack((cost_matrix, dummy_data))


    ans_pos = hungarian_algorithm(cost_matrix.copy())
    ans, ans_mat = ans_calculation(cost_matrix, ans_pos)

    # Show the result
    print(f"Linear Assignment problem result: {ans:.0f}\n{ans_mat}")


if __name__ == '__main__':
    main()
```

Output for problem 2:

```
Sayantani Karmakar, 20CS8024
Enter the number of rows: 5
Enter the number of columns: 5
Enter values for row 1 separated by spaces: 12 7 9 7 9
Enter values for row 2 separated by spaces: 8 9 6 6 6
Enter values for row 3 separated by spaces: 7 17 12 14 9
Enter values for row 4 separated by spaces: 15 14 6 6 10
Enter values for row 5 separated by spaces: 4 10 7 9 12
Linear Assignment problem result: 32
[['X' '7' 'X' 'X' 'X']
 ['X' 'X' '6' 'X' 'X']
 ['X' 'X' 'X' 'X' '9']
 ['X' 'X' 'X' '6' 'X']
 ['4' 'X' 'X' 'X' 'X']]
```