[1]:
```
import numpy as np

data_mega_poll_results = np.array([
    [54, 17], [87, 84], [70, 17], [46, 17], [81, 17], [32, 46], [17, 70], [84,␣
↪81], [80, 63], [72, 2],
    [83, 30], [98, 43], [3, 56], [9, 53], [59, 83], [21, 63], [90, 9], [86, 36],␣
↪[65, 63], [91, 8],
    [75, 45], [78, 77], [29, 43], [6, 63], [47, 43], [51, 96], [85, 12], [89,␣
↪6], [92, 5], [42, 88],
    [77, 72], [2, 90], [53, 77], [43, 77], [30, 45], [1, 29], [8, 36], [37, 29],␣
↪[48, 43], [44, 66],
    [88, 53], [63, 37], [64, 2], [45, 59], [100, 69], [95, 85], [5, 76], [76,␣
↪86], [56, 64], [52, 56],
    [40, 88], [12, 5], [69, 44], [36, 92], [66, 56], [96, 45], [68, 82], [26,␣
↪58], [58, 26], [82, 68],
    [93, 38], [34, 61], [39, 16], [99, 38], [73, 60], [38, 93], [28, 19], [13,␣
↪15], [15, 39], [71, 50],
    [20, 39], [50, 62], [24, 4], [74, 11], [11, 71], [19, 14], [57, 18], [18,␣
↪27], [27, 20], [41, 38],
    [61, 20], [67, 60], [62, 74], [4, 99], [14, 38], [49, 73], [60, 11], [16,␣
↪4], [22, 55], [55, 22],
    [23, 31], [10, 35], [33, 23], [25, 23], [97, 23], [31, 33], [35, 25], [7,␣
↪10], [79, 23]
])
```

Question 1-2

Initially we create a matrix with zeros entries of 100 by 100 then supose citizen ID 35 vote to citizen ID 25 then the entry (row,col)=(25,35) of M matrix is 1 and (35,35) entry of M matrix is -1.

[2]:
```
M = np.zeros((100, 100))
for col, (col_idx, row_idx) in enumerate(data_mega_poll_results):
    col_idx -= 1  # Adjust for zero-based indexing
    row_idx -= 1  # Adjust for zero-based indexing
    M[col_idx, col_idx] = -1
    M[row_idx, col_idx] = 1
```

Question 3 (reduced row echelon form of M)

[3]:
```
from sympy import Matrix
from scipy.linalg import null_space
M_rref=Matrix(M).rref()
```

Question 4

Check the columns for scalor multiplication, remove detected columns which represent the reciprocal votes.

[4]:
```
B = M
from sympy import Matrix
from scipy.linalg import null_space
# Number of columns
num_cols = B.shape[1]

# Find columns where one is -1 times another
remove_indices = set()  # Store indices of columns to remove

for i in range(num_cols):
    for j in range(i + 1, num_cols):  # Avoid duplicate checks
        if np.all(B[:, i] == -B[:, j]):  # Check if column i is -1 times column j
            remove_indices.add(i)  # Remove both columns
            remove_indices.add(j)

# Create new matrix without the identified columns
M_R= np.delete(B, list(remove_indices), axis=1)
```

Question 5

when the column rank of M and M_R are the same, then the the columns of M_R constitute a basis of $Col(M)$. This happens when there are no reciprocal voting activity in the election. However, if there are any cases of reciprocal votes, then this result is not true.

Question 6

Because every citizen must be assigned to a community, it is necessary to take the transpose of M_R so that the columns of M_R^T can be mapped to corresponding citizens of the population. This will allow us to perform the matrix vector multiplication appropriately (dimensions will match) as explained in the next question.

Question 7

The communities form "closed" groups, i.e. there are no inter-community affiliations(votes). Now relate this idea to the cases of disjoint tandem accounts in our example on the banking and accounting systems discussed in the lectures. This should tell you that the appropriate vector space is the $null(M_R^T)$. The basis vectors of $null(M_R^T)$ prescribe the individual communities.

[5]:
```
null_basis_MR=Matrix(np.transpose(M_R)).nullspace()
```

[6]:
```
from sympy import Matrix
# List to store disjoint communities
disjoint_communities = []

# Identifying positions of 1s in each column
for mat in null_basis_MR:
    positions = []  # Create a new list for each column
    for i, val in enumerate(mat):
```

```
        if val != 0:
            positions.append(i + 1)  # Convert to 1-based index
    disjoint_communities.append(positions)  # Append to main list

# Printing disjoint communities
for idx, members in enumerate(disjoint_communities, start=1):
    print(f"Disjoint community {idx} = {members}")
```

```
Disjoint community 1 = [22]
Disjoint community 2 = [26]
Disjoint community 3 = [55]
Disjoint community 4 = [58]
Disjoint community 5 = [68]
Disjoint community 6 = [70]
Disjoint community 7 = [11, 49, 50, 60, 62, 67, 71, 73, 74]
Disjoint community 8 = [82]
Disjoint community 9 = [17, 32, 46, 54, 81, 84, 87]
Disjoint community 10 = [93]
Disjoint community 11 = [94]
Disjoint community 12 = [5, 8, 12, 36, 76, 85, 86, 91, 92, 95]
Disjoint community 13 = [30, 45, 51, 59, 75, 83, 96]
Disjoint community 14 = [7, 10, 23, 25, 31, 33, 35, 79, 97]
Disjoint community 15 = [4, 13, 14, 15, 16, 18, 19, 20, 24, 27, 28, 34, 38, 39,
41, 57, 61, 99]
Disjoint community 16 = [1, 2, 3, 6, 9, 21, 29, 37, 40, 42, 43, 44, 47, 48, 52,
53, 56, 63, 64, 65, 66, 69, 72, 77, 78, 80, 88, 89, 90, 98, 100]
```