1

In [1]:
```
import numpy as np
```

In [3]:
```
def GaleShapleyAlgorithm(P1,P2):
    """
    P1:column rank
    P2:row rank
    Group1 propose(P1 column)
    """
    Match = np.zeros(P1.shape)
    NumStages = 0
    m = P1.shape[0] #group2
    n = P1.shape[1] #group1
    num = n

    while(np.sum(Match) < num):
        p1_idx = np.argmin(P1,axis=0) #

        #find match
        for i,j in enumerate(p1_idx): #i is the index
            Match[j,i] = 1

        t = Match * P2 #temp
        t[t == 0] = n + 1 #clear unrelevant match with large number
        rej = (t > np.min(t,axis=1,keepdims=True)) & (t < n+1) #find the lower p
        t[rej] = n + 1 #reject the lower priority
        Match = (t <= n).astype(int) #update match

        rej_idx = np.where(rej == True)
        P1[rej_idx[0],rej_idx[1]]=m+1 #clear the rejected match

        num = num - np.sum((np.sum(P1,axis=0)==m*(m+1))) #if all the match of on

        NumStages = NumStages + 1
        # print(np.sum(Match))
        # print(np.min([m,n]))

    return Match, NumStages
```

(a)

For Q1

In [4]:
```
P1 = np.array([[3,3,2,3],[4,1,3,2],[2,4,4,1],[1,2,1,4]])
P2 = np.array([[1,2,3,4],[1,4,2,3],[2,1,3,4],[4,2,3,1]])
Match, NumStages = GaleShapleyAlgorithm(P1,P2)
print(Match)
print("stages:", NumStages)

Match, NumStages = GaleShapleyAlgorithm(P2.T,P1.T)
print(Match.T)
print("stages:",NumStages)
```

```
[[0 0 1 0]
 [0 0 0 1]
 [1 0 0 0]
 [0 1 0 0]]
stages: 5
[[1 0 0 0]
 [0 0 1 0]
 [0 1 0 0]
 [0 0 0 1]]
stages: 2
```

For Q2

In [5]:
```
P1 = np.array([[1,1,2,3,3],[2,3,1,1,2],[3,2,3,2,1]])
P2 = np.array([[2,1,3,4,5],[3,1,2,5,4],[3,1,4,2,5]])
Match, NumStages = GaleShapleyAlgorithm(P1,P2)
print(Match)
print("stages:", NumStages)

Match, NumStages = GaleShapleyAlgorithm(P2.T,P1.T)
print(Match.T)
print("stages:", NumStages)
```

```
[[0 1 0 0 0]
 [0 0 1 0 0]
 [0 0 0 1 0]]
stages: 4
[[0 1 0 0 0]
 [0 0 1 0 0]
 [0 0 0 1 0]]
stages: 2
```

2

In [6]:
```
import scipy.io as scio

def test(datafile):
    data = scio.loadmat(datafile)
    # print(data.keys())
    P1 = data['P1'] #row rank
    P2 = data['P2'] #colonm rank

    #from one side
    Match1, NumStages = GaleShapleyAlgorithm(P2,P1)
    print("students propose:\n", Match1, "stages:", NumStages)

    #from the other side
    Match2, NumStages = GaleShapleyAlgorithm(P1.T,P2.T)
    print("hospitals propose:\n", Match2.T, "stages:", NumStages)

    print("uniqueness:", np.sum(Match1 != Match2.T)) #0 means same and unique
```

Q1

In [7]:
```
Q1 = 'gale_shapley_programming_files\gale_shapley_programming_files\Q1.mat'
test(Q1)
```

```
students propose:
 [[1 0 0 0 0]
 [0 0 0 1 0]
 [0 0 0 0 1]
 [0 0 1 0 0]
 [0 1 0 0 0]] stages: 5
hospitals propose:
 [[1 0 0 0 0]
 [0 0 0 1 0]
 [0 0 0 0 1]
 [0 0 1 0 0]
 [0 1 0 0 0]] stages: 2
uniqueness: 0
```

It is the unique stable matching, taking 5 and 2 stages from students and hospitals side respectively.
It is optimal for both groups.

Q2

In [8]:
```
Q2 = 'gale_shapley_programming_files\gale_shapley_programming_files\Q2.mat'
test(Q2)
```

```
students propose:
 [[0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]] stages: 5
hospitals propose:
 [[0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]] stages: 1
uniqueness: 0
```

It is the unique stable matching, taking 5 and 1 stages from students and hospitals side respectively.
It is optimal for both groups.

Q3

In [9]:
```
Q3 = 'gale_shapley_programming_files\gale_shapley_programming_files\Q3.mat'
test(Q3)
```

```
students propose:
 [[1 0 0 0]
 [0 0 1 0]
 [0 0 0 1]
 [0 1 0 0]] stages: 3
hospitals propose:
 [[1 0 0 0]
 [0 0 1 0]
 [0 0 0 1]
 [0 1 0 0]] stages: 3
uniqueness: 0
```

It is the unique stable matching, taking 3 and 3 stages from students and hospitals side respectively.

It is optimal for both groups.

3

The group who propose will get a optimal match for them.
I think Gale-Shapley algorithm should be run with hospitals proposing, because the stages it takes are less than applicants proposing, especially when hospitals are much less than applicants.
However, if the matches are not unique, the result may harm applicants' preference.

4

In [10]:
```
def GaleShapleyAlgorithm_student(P1,P2,quota):
    """
    P1:column rank, student
    P2:row rank, hospital
    Group1 propose(P1 column)
    """
    Match = np.zeros(P1.shape)
    NumStages = 0
    m = P1.shape[0] #group2
    n = P1.shape[1] #group1
    quota = quota.reshape(1,-1) #reshape quota
    f = Match.copy() #every proposal match
    s = Match.copy() #save match as output


    while(1):
        p1_idx = np.argmin(P1,axis=0) #

        #find match
        for i,j in enumerate(p1_idx): #i is the index
            Match[j,i] = 1

        if (f == Match).all(): #if the match is not changed,
            Match = s.copy() #output the last match
            break
        else:
            f = Match.copy()

        q = np.sum(Match,axis=1,keepdims=True) #q choose the top quota
        for i,j in enumerate(q):
            # print(j[0],quota.shape)
            if j[0] > quota[0][i]:
                t = Match[i,:] * P2[i,:] #temp
                t[t == 0] = n + 1 #clear unrelevant match with large number

                rej = (np.argsort(t))[quota[0][i]:int(-n+j)] #find the lower pri
                Match[i,rej] = 0 #reject the lower priority

                P1[i,rej]=m+1 #clear the rejected match

        s = Match.copy() #save the match

        NumStages = NumStages + 1

        # print(np.sum(Match))
```

```
        # print(np.min([m,n]))

    return Match, NumStages
```

In [11]:
```
def GaleShapleyAlgorithm_hospital(P1,P2,quota):
    """
    P1:column rank, hospital
    P2:row rank, student
    Group1 propose(P1 column)
    """
    Match = np.zeros(P1.shape)
    NumStages = 0
    m = P1.shape[0] #group2
    n = P1.shape[1] #group1
    quota = quota.reshape(1,-1) #reshape quota
    num = np.sum(quota) #total match number
    f = Match.copy() #every proposal match
    s = Match.copy() #save match as output

    while(1):
        for i in range(n):
            pps = (np.argsort(P1[:,i]))[:quota[0][i]]
            Match[pps,i] = 1

        if (f == Match).all(): #if the match is not changed
            Match = s.copy() #output the last match
            break
        else:
            f = Match.copy()

        q = np.sum(Match,axis=1,keepdims=True) #q choose the top quota
        for i,j in enumerate(q):
            if j > 1:
                t = Match[i,:] * P2[i,:] #temp
                t[t == 0] = n + 1 #clear unrelevant match with large number

                rej = (np.argsort(t))[1:] #find the lower priority

                Match[i,rej] = 0 #reject the lower priority
                P1[i,rej]=m+1 #clear the rejected match

        s = Match.copy() #save the match

        NumStages = NumStages + 1

        # print(np.sum(Match))
        # print(np.min([m,n]))

    return Match, NumStages
```

In [12]:
```
def test_quota(Q):
    data = scio.loadmat(Q)

    P1 = data['P1'] #row rank
    P2 = data['P2'] #colonm rank
    quota = data['Quota']

    #from one side
    Match1, NumStages = GaleShapleyAlgorithm_student(P2,P1,quota)
```

```
    print("students propose:\n", Match1, "stages:", NumStages, "num:", np.sum(Ma

    #from the other side
    Match2, NumStages = GaleShapleyAlgorithm_hospital(P1.T,P2.T,quota)
    print("hospitals propose:\n", Match2.T, "stages:", NumStages, "num:", np.sum

    print("uniqueness:", np.sum(Match1 != Match2.T)) #0 means same and unique
```

In [13]:
```
test_quota(Q1)
```

```
students propose:
 [[1. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 1. 0. 0. 1.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0.]] stages: 4 num: 5.0
hospitals propose:
 [[1. 1. 0. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0.]] stages: 3 num: 5.0
uniqueness: 2
```

There is no unique stable matching.
It takes 4 and 3 stages from students and hospitals side respectively.
The result is optimal for the group who propose.

In [14]:
```
test_quota(Q2)
```

```
students propose:
 [[1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 1. 1. 1. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 1. 0.]] stages: 14 num: 1
8.0
hospitals propose:
 [[1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 1. 0. 0. 0. 0. 1. 1. 1. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 1. 1. 0. 0. 0. 0. 1.]
 [0. 1. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]] stages: 5 num: 1
8.0
uniqueness: 8
```

There is no unique stable matching.
It takes 14 and 5 stages from students and hospitals side respectively.
The result is optimal for the group who propose.

In [15]:
```
test_quota(Q3)
```

```
students propose:
 [[1. 0. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]
 [0. 1. 0. 0.]] stages: 3 num: 4.0
hospitals propose:
 [[1. 0. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]
 [0. 1. 0. 0.]] stages: 4 num: 4.0
uniqueness: 0
```

It is the unique stable matching, taking 3 and 4 stages from students and hospitals side respectively.
It is optimal for both groups.

When students are much more than hospitals, we should run with hospitals proposing. It takes less stages, while the result will harm students' preference.

5
There are 2 different stable matchings in Q1 and Q2. The number of unoccupied residency position are the same.
It implies that stability means the same occupied residency position while does not mean optimal for everyone and may harm fairness.