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

In [2]:
```
np.random.seed(200318155)
P = np.random.uniform(size=(4,4))
P = P / P.sum(axis=1, keepdims=1)
print("One step transition matrix is \n", P)
print("Sum of rows of one step transition matrix \n",P.sum(axis=1))

B = np.random.uniform(size=(4,3))
B = B / B.sum(axis=1, keepdims=1)
print("Event matrix", B)
print("Sum of rows of event matrix \n", B.sum(axis=1))

## Task1
Q = [1]
Obs = []
for i in range(1000):
    state = Q[-1] - 1
    r = np.random.uniform()
    if r <= B[state][0]:
        Obs.append(1)
    elif (B[state][0]) < r <= (B[state][0]+B[state][1]):
        Obs.append(2)
    else:
        Obs.append(3)

    r_state = np.random.uniform()
    if r_state <= P[state][0]:
        Q.append(1)
    elif P[state][0] < r_state <= (P[state][1]+P[state][0]):
        Q.append(2)
    elif (P[state][1]+P[state][0]) < r_state <= (P[state][1]+P[state][
        Q.append(3)
    else:
        Q.append(4)
print("\n Generated set of observations :{}".format(Obs))
```

```
One step transition matrix is
 [[0.04095739 0.01635738 0.53344914 0.40923609]
 [0.09889592 0.05779718 0.45731745 0.38598945]
 [0.45124562 0.27183985 0.24855018 0.02836435]
 [0.37391087 0.40131007 0.19378268 0.03099638]]
Sum of rows of one step transition matrix
 [1. 1. 1. 1.]
Event matrix [[0.32306158 0.24573388 0.43120454]
 [0.69434845 0.29448027 0.01117128]
 [0.35683131 0.29334779 0.3498209 ]
 [0.33332781 0.16557221 0.50109998]]
```

```
Sum of rows of event matrix
 [1. 1. 1. 1.]

 Generated set of observations :[3, 1, 1, 3, 2, 3, 2, 1, 2, 2, 3, 1,
3, 2, 3, 1, 2, 1, 3, 3, 3, 2, 3, 1, 2, 3, 3, 2, 3, 1, 2, 1, 3, 1, 1,
2, 3, 1, 1, 2, 2, 1, 3, 1, 2, 1, 1, 1, 1, 3, 3, 3, 1, 1, 3, 3, 3, 1,
2, 1, 3, 1, 1, 1, 3, 1, 2, 3, 3, 1, 2, 1, 2, 1, 3, 3, 1, 3, 3, 3, 2,
3, 3, 1, 2, 1, 3, 1, 3, 2, 2, 1, 1, 3, 3, 3, 1, 1, 1, 1, 2, 1, 1, 3,
3, 1, 3, 3, 2, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 3, 3, 2, 2, 3, 3, 3, 1,
1, 3, 3, 2, 1, 2, 2, 3, 3, 3, 1, 1, 3, 3, 1, 3, 3, 1, 1, 3, 3, 3, 1,
2, 3, 2, 2, 3, 1, 1, 1, 1, 1, 3, 1, 3, 1, 1, 2, 1, 3, 2, 3, 2, 2, 3,
3, 1, 2, 1, 1, 1, 1, 1, 2, 2, 3, 2, 1, 1, 3, 1, 1, 1, 2, 2, 2, 3, 3,
1, 1, 1, 1, 3, 1, 3, 3, 1, 3, 3, 2, 2, 3, 3, 3, 2, 3, 1, 3, 1, 3, 2,
1, 2, 1, 3, 1, 2, 3, 2, 1, 3, 2, 3, 1, 3, 2, 3, 1, 3, 1, 2, 3, 2, 1,
1, 2, 3, 1, 3, 3, 1, 2, 2, 2, 3, 1, 2, 3, 1, 1, 3, 2, 2, 3, 1, 3, 3,
3, 1, 3, 1, 1, 3, 1, 1, 1, 1, 3, 3, 2, 2, 1, 1, 3, 1, 1, 3, 3, 1, 1,
1, 3, 2, 1, 1, 3, 1, 2, 1, 3, 1, 2, 1, 1, 1, 3, 1, 2, 1, 1, 3, 1, 2,
3, 1, 2, 1, 3, 1, 3, 2, 1, 2, 3, 1, 1, 3, 3, 3, 3, 1, 1, 1, 3, 1, 1,
1, 1, 1, 1, 1, 1, 3, 1, 3, 3, 1, 2, 1, 2, 1, 3, 1, 2, 3, 3, 3, 2, 3,
1, 1, 3, 1, 3, 2, 1, 3, 3, 2, 2, 1, 2, 2, 2, 3, 3, 1, 1, 2, 2, 1, 2,
1, 2, 3, 3, 2, 2, 1, 3, 1, 2, 1, 2, 3, 2, 2, 1, 3, 3, 3, 2, 3, 1, 3,
1, 3, 3, 1, 2, 2, 1, 3, 1, 1, 3, 1, 3, 2, 2, 2, 1, 3, 3, 1, 2, 3, 3,
1, 3, 1, 3, 3, 1, 1, 3, 1, 2, 3, 1, 2, 3, 1, 2, 1, 2, 3, 2, 2, 1, 3,
3, 1, 2, 1, 3, 2, 1, 3, 1, 3, 1, 1, 3, 3, 1, 3, 3, 3, 1, 2, 1, 1, 1,
3, 1, 3, 3, 1, 3, 2, 1, 2, 1, 2, 2, 3, 3, 2, 2, 3, 2, 1, 1, 1, 1, 1,
1, 2, 2, 3, 2, 2, 3, 1, 3, 2, 1, 1, 1, 3, 3, 2, 3, 1, 1, 2, 2, 1, 2,
3, 1, 2, 3, 3, 3, 3, 3, 3, 2, 1, 1, 3, 3, 1, 2, 1, 2, 1, 3, 3, 3, 1,
1, 3, 2, 3, 1, 3, 2, 2, 1, 3, 3, 2, 1, 3, 2, 2, 2, 3, 3, 2, 1, 1, 3,
2, 2, 1, 3, 1, 1, 1, 2, 1, 1, 3, 1, 1, 1, 3, 1, 2, 2, 3, 3, 2, 3, 1,
1, 2, 1, 2, 3, 1, 3, 3, 1, 1, 3, 2, 3, 3, 2, 3, 1, 1, 1, 3, 2, 3, 1,
2, 3, 3, 1, 3, 3, 2, 1, 1, 2, 2, 3, 2, 1, 1, 1, 2, 3, 3, 3, 1, 3, 3,
3, 3, 3, 2, 3, 1, 3, 1, 2, 3, 3, 3, 1, 1, 3, 2, 1, 3, 1, 2, 2, 3, 3,
2, 3, 3, 1, 1, 1, 1, 1, 1, 2, 1, 3, 3, 2, 2, 1, 3, 3, 1, 1, 2, 3, 2,
3, 1, 1, 3, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 3, 3, 1, 3, 2, 3, 1, 1, 3,
1, 1, 3, 2, 3, 1, 1, 2, 3, 2, 2, 1, 1, 3, 2, 3, 2, 1, 1, 2, 1, 3, 3,

3, 3, 3, 1, 1, 2, 3, 2, 2, 3, 3, 2, 3, 2, 1, 1, 1, 3, 2, 1, 2, 1, 1,
2, 3, 1, 1, 3, 3, 3, 1, 1, 1, 1, 3, 3, 1, 3, 3, 3, 1, 3, 2, 3, 1, 1,
2, 3, 3, 1, 3, 3, 1, 3, 1, 3, 3, 3, 3, 1, 3, 3, 1, 1, 1, 2, 2, 2, 3,
3, 3, 1, 1, 2, 1, 2, 3, 3, 3, 2, 2, 1, 3, 1, 2, 1, 1, 2, 1, 3, 3, 2,
3, 3, 3, 2, 2, 3, 3, 3, 3, 3, 2, 3, 1, 1, 1, 1, 1, 3, 3, 1, 1, 3, 3,
1, 1, 2, 1, 3, 3, 3, 3, 1, 1, 2, 3, 1, 3, 3, 3, 2, 2, 3, 2, 3, 3, 1,
3, 3, 1, 3, 3, 1, 1, 3, 2, 3, 3, 1, 3, 3, 3, 3, 1, 1, 3, 2, 1, 3, 1,
2, 3, 1, 3, 3, 1, 2, 1, 1, 3, 1, 2, 1, 3, 1, 3, 3, 1, 3, 1, 1, 3, 3,
1, 1, 3, 3, 3, 3, 1, 3, 2, 1, 1, 3, 3, 1, 3, 2, 3, 1, 2, 1, 3, 2, 1,
1, 2, 1, 3, 3, 1, 1, 3, 2, 3, 1, 3, 1, 2, 3, 3, 2, 1, 1, 2, 2, 1, 3,
1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 1, 2, 2, 1, 1, 1, 3, 3, 3, 3, 3,
3, 3, 1, 2, 1, 1, 1, 1, 3, 3, 1, 1, 1, 1, 3, 3, 1, 1, 3, 2, 1, 2]
```

Comments for task2: The one step transition matrix and the event matrix are first generate by using random initialization using numpy random library with the random seed as my student id which is '200318155'. Then the P matrix and the even matrix both are normalized to have the sum of rows equal to 1. I have explicitly printed the P, B matrix and the sum of their rows to see whetther they satisfy the criteria or not. The observations and the next states are generated as per suggested in the project that is drawining a pseudo-random number and the comparing this number with the probabilities of event matrix and the once step transition matrix respectively.

In [3]:
```
## Task 2
init_state = [1,0,0,0]
O = [1,2,3,3,1,2,3,3,1,2,3,3]
alpha = np.zeros((4,len(O)))
alpha[0][0] = B[0][O[0]-1]*init_state[0]

for t in range(1,len(O)):
    for i in range(4):
        alpha[i][t] = alpha[0][t-1]*P[0][i]*B[i][O[t]-1] + alpha[1][t-

prob = alpha[0][-1] + alpha[1][-1] + alpha[2][-1] + alpha[3][-1]
print("The probability p(O|lambda) that the sequence of observations c
```

```
The probability p(O|lambda) that the sequence of observations comes f
rom HMM is 1.0823819970616577e-06
```

Comments for task2: As suggested in the project I have implemented the Forward equation to calculate the probabilty of the given sequence comes from the generated HMM. The probability calculate is 1.0823819970616577e-06 which is very low. Thus we can say that it is not possible that the given sequence has come from the generate HMM from task 1. The end probablity is calculated using the by adding the probablity Tth sequence for each and every state from the alpha matrix. which is calculated using the forward equations.

In [4]:
```
## Task 3
O = [1,2,3,3,1,2,3,3,1,2,3]
alpha = np.zeros((4,len(O)))
alpha[0][0] = B[0][O[0]-1]*init_state[0]

for t in range(1,len(O)):
    for i in range(4):
        alpha[i][t] = alpha[0][t-1]*P[0][i]*B[i][O[t]-1] + alpha[1][t-

beta = np.zeros((4,len(O)))
beta[:,-1] = 1
for t in range(len(O)-2,-1,-1):
    for i in range(4):
        beta[i][t] = P[i][0]*B[0][O[t+1]-1]*beta[0][t+1] + P[i][1]*B[1

gamma = np.zeros((4,len(O)))
for t in range(len(O)):
    for j in range(4):
        gamma[j][t] = (alpha[j][t]*beta[j][t])/(alpha[0][t]*beta[0][t]

Q_out = np.argmax(gamma, axis = 0) + np.ones((1,len(O)))
print("The most probable sequence to generate the give sequence O is",
```

```
The most probable sequence to generate the give sequence O is [1. 3.
1. 3. 2. 3. 1. 3. 3. 3. 1.]
```

Comments for Task3: For task 3 that is to produce the sequence is calculated by using Forward-Backward equations. I have calculated the alpha, beta, gamma matrices as per the formulations stated in the book. The gamma matrix give us the probability of every state. Thus the out that I have chosen is the argmax of the values of all the possible state thus which results in the sequence '1. 3. 1. 3. 2. 3. 1. 3. 3. 3. 1.'.

In [5]:
```
## Task4
np.random.seed(200318155)
from hmmlearn import hmm
obs = np.array(Obs) - np.ones((len(Obs)))
obs = np.array([obs], dtype=int).T

model = hmm.MultinomialHMM(n_components=4)
model.fit(obs)
print("Starting conditions", model.startprob_)
print("One step transition matrix is \n",model.transmat_)
print("Event Matrix",model.emissionprob_)
```

```
Starting conditions [9.99506614e-01 2.45029159e-10 4.93385427e-04 5.4
7083927e-13]
One step transition matrix is
 [[0.26201885 0.25842124 0.24551173 0.23404819]
 [0.23669043 0.24096905 0.25399353 0.26834699]
 [0.26583132 0.244944   0.25025828 0.23896641]
 [0.2623899  0.24207632 0.25189088 0.24364289]]
Event Matrix [[0.06376703 0.02544722 0.91078574]
 [0.74213899 0.15933275 0.09852826]
 [0.31018388 0.26278316 0.42703296]
 [0.49246156 0.45332808 0.05421036]]
```

Comments for taks4: For this task I have used hmmlearn libraryr which is used for generating HMM models. Using the multinomialHMM module I have generated a HMM model with four states. But the multinomial model takes the observations in the range of 0 to n-1 and in atleast 2-dimensional shape thus I have converted converted the observation in the range of 0,1,2,3. Then using these observations I have fit the model which is used to estimate the parameters for the HMM which are starting condition, one step transition matrix, event matrix. Thus using attributes model.startprob_, model.transmat_, model.emissionprob_ we get pi, P, B matrices for our HMM. I have printed the parameters from which we can clearly see that the starting conditions are almost close to (1,0,0,0) which are the initial conditions setup for the project. Also the event matrix and one step transition matrix are similar to the generated parameters in task1. The given module does not provide the p-values for individual parameters therefore I have not calculate p-values for the same as suggested by professor on the moodle.