In [30]:

```
# problem b
import numpy as np
import queue
def computeMarginals(G, F):
    N = np.shape(G)[0]
    K = np.shape(G)[1]
    VF = np.ones((N, K, 2))
    FV = np.ones((K, N, 2))
    Vsent = np.zeros((N, K))
    Fsent = np.zeros((K, N))

#     leaves to root
    q = queue.Queue()
    for i in range(N):
        sum = 0
        for j in range(K):
            sum += G[i,j]
            if sum > 1:
                break
        if sum <= 1:
            q.put(i)
    root = q.get()
    while q.empty() == False:
        cur = q.get()
        k = 0
        f = -1
        temp0 = 1
        temp1 = 1
        while k < K:
            if G[cur, k] == 1:
                if Fsent[k, cur] == 1:
                    temp0 *= FV[k, cur, 0]
                    temp1 *= FV[k, cur, 1]
                else:
                    f = k
            k += 1
        VF[cur, f, 0] = temp0
        VF[cur, f, 1] = temp1
        Vsent[cur, f] = 1
        l = 0
        while l < N:
            if l != cur and G[l, f] == 1:
                break;
            l += 1
        if cur < l:
            FV[f, l, 0] = F[f, 1, 0]*VF[cur, f, 1] \
            + F[f, 0, 0]*VF[cur, f, 0]
            FV[f, l, 1] = F[f, 0, 1]*VF[cur, f, 0] \
            + F[f, 1, 1]*VF[cur, f, 1]
        else:
            FV[f, l, 0] = F[f, 0, 1]*VF[cur, f, 1] \
```

```
            + F[f, 0, 0]*VF[cur, f, 0]
            FV[f, l, 1] = F[f, 1, 0]*VF[cur, f, 0] \
            + F[f, 1, 1]*VF[cur, f, 1]
        Fsent[f, l] = 1

        #add l to queue if only one f->l is unmarked
        count = 0
        for j in range(K):
            if G[l, j] == 1 and Fsent[j, l] == 0:
                count += 1
        if count == 1:
            q.put(l)


    # root to leaves
    q = queue.Queue()
    q.put(root)
    while q.empty() == False:
        cur = q.get()
        for k in range(K):
            if G[cur, k] == 1 and Vsent[cur, k] == 0:
                temp0 = 1
                temp1 = 1
                for l in range(K):
                    if l != k and Fsent[l, cur] == 1:
                        temp0 *= FV[l, cur, 0]
                        temp1 *= FV[l, cur, 1]
                VF[cur, k, 0] = temp0
                VF[cur, k, 1] = temp1
                Vsent[cur, k] = 1
                j = 0
                while j < N:
                    if j != cur and G[j, k] == 1:
                        break
                    j += 1
                if cur < j:
                    FV[k, j, 0] = F[k, 1, 0]*VF[cur, k, 1] \
                    + F[k, 0, 0]*VF[cur, k, 0]
                    FV[k, j, 1] = F[k, 0, 1]*VF[cur, k, 0] \
                    + F[k, 1, 1]*VF[cur, k, 1]
                else:
                    FV[k, j, 0] = F[k, 0, 1]*VF[cur, k, 1] \
                    + F[k, 0, 0]*VF[cur, k, 0]
                    FV[k, j, 1] = F[k, 1, 0]*VF[cur, k, 0] \
                    + F[k, 1, 1]*VF[cur, k, 1]
                Fsent[k, j] = 1
                q.put(j)


    result = np.ones((N,2))
    for i in range(N):
        for k in range(K):
            result[i, 0] *= FV[k, i, 0]
```

```
            result[i, 1] *= FV[k, i, 1]
        sum = (result[i, 0] + result[i, 1])
        result[i, 0] = result[i, 0] / sum
        result[i, 1] = result[i, 1] / sum
    return result

# problem d
import numpy as np
def ComputeJoint(G, F, x):
    N = np.shape(G)[0]
    K = np.shape(G)[1]
    result = 1
    for i in range(K):
        n1 = -1
        n2 = -2
        for n in range(N):
            if G[n, i] == 1:
                if n1 < 0:
                    n1 = n
                else:
                    n2 = n
        result *= F[i, x[n1], x[n2]]
    return result

def Combination(G, F, i, val, index, x):
    if index == np.shape(G)[0]:
        return ComputeJoint(G, F, x)
    res = 0
    if index == i:
        x.append(val)
        res += Combination(G, F, i, val, index+1, x)
        x.pop()
    else:
        x.append(0)
        res += Combination(G, F, i, val, index+1, x)
        x.pop()
        x.append(1)
        res += Combination(G, F, i, val, index+1, x)
        x.pop()
    return res


def bruteForce(G, F):
    N = np.shape(G)[0]
    K = np.shape(G)[1]
    result = np.ones((N,2))
    for i in range(N):
        sum = 0
        result[i, 0] = Combination(G, F, i, 0, 0, [])
        result[i, 1] = Combination(G, F, i, 1, 0, [])
        sum = result[i,0] + result[i, 1]
        result[i, 0] = result[i, 0]/sum
        result[i,1] = result[i,1]/sum
```

```
    return result

G = np.array([[1,0,0,0,0],
              [1,0,1,0,1],
              [0,1,1,1,0],
              [0,1,0,0,0],
              [0,0,0,0,1],
              [0,0,0,1,0]])
K = np.shape(G)[1]
F = np.ones((K, 2, 2))
F[0,1,1] = 5
F[1,0,1] = 0.5
F[2,0,0] = 0
F[3,0,0] = 2
B = computeMarginals(G, F)
print("computeMarginals")
print(B)
B = bruteForce(G, F)
print("bruteForce")
print(B)
```

```
computeMarginals
[[ 0.21186441  0.78813559]
 [ 0.13559322  0.86440678]
 [ 0.45762712  0.54237288]
 [ 0.57627119  0.42372881]
 [ 0.5         0.5       ]
 [ 0.57627119  0.42372881]]
bruteForce
[[ 0.21186441  0.78813559]
 [ 0.13559322  0.86440678]
 [ 0.45762712  0.54237288]
 [ 0.57627119  0.42372881]
 [ 0.5         0.5       ]
 [ 0.57627119  0.42372881]]
```