# Problem 1:

(a) & (b)

for example, we output the second row of the matrix xij:

```
In [1]:
from pyGM import *
import numpy as np
# Load the data points D, and the station locations (lat/lon)
D = np.genfromtxt('data/data.txt',delimiter=None)
loc = np.genfromtxt('data/locations.txt',delimiter=None)
m,n = D.shape # m = 2760 data points, n=30 dimensional # D[i,j] = 1 if
station j observed rainfall on day i

count = np.zeros(30)
for j in range(n):
    for i in range(m):
        if D[i][j] == 1:
            count[j] += 1

x = count/m

count = np.zeros((30, 30, 4))
for i in range(n):
    for j in range(n):
        for k in range(m):
            if D[k][i] == 0 and D[k][j] == 0:
                count[i][j][0] += 1
            if D[k][i] == 1 and D[k][j] == 0:
                count[i][j][1] += 1
            if D[k][i] == 0 and D[k][j] == 1:
                count[i][j][2] += 1
            if D[k][i] == 1 and D[k][j] == 1:
                count[i][j][3] += 1

xij = np.zeros((30, 30, 4))
for i in range(n):
    for j in range(n):
        for k in range(4):
            xij[i][j][k] = count[i][j][k]/m
```

```
In [2]: xij[1]

Out[2]:
array([[ 0.59130435,  0.11123188,  0.03514493,  0.26231884],
       [ 0.62644928,  0.        ,  0.        ,  0.37355072],
       [ 0.57681159,  0.07536232,  0.04963768,  0.29818841],
       [ 0.51014493,  0.06304348,  0.11630435,  0.31050725],
       [ 0.55036232,  0.05      ,  0.07608696,  0.32355072],
       [ 0.5134058 ,  0.05507246,  0.11304348,  0.31847826],
       [ 0.53442029,  0.06086957,  0.09202899,  0.31268116],
       [ 0.40905797,  0.0576087 ,  0.2173913 ,  0.31594203],
       [ 0.30362319,  0.04637681,  0.32282609,  0.32717391],
       [ 0.37246377,  0.05144928,  0.25398551,  0.32210145],
       [ 0.42826087,  0.06449275,  0.19818841,  0.30905797],
       [ 0.44456522,  0.04963768,  0.18188406,  0.32391304],
       [ 0.47246377,  0.06195652,  0.15398551,  0.3115942 ],
       [ 0.32862319,  0.04021739,  0.29782609,  0.33333333],
       [ 0.44021739,  0.06050725,  0.18623188,  0.31304348],
       [ 0.36413043,  0.07391304,  0.26231884,  0.29963768],
       [ 0.59202899,  0.1423913 ,  0.03442029,  0.23115942],
       [ 0.58152174,  0.11050725,  0.04492754,  0.26304348],
       [ 0.59130435,  0.1326087 ,  0.03514493,  0.24094203],
       [ 0.58949275,  0.16231884,  0.03695652,  0.21123188],
       [ 0.56268116,  0.09963768,  0.06376812,  0.27391304],
       [ 0.56956522,  0.11992754,  0.05688406,  0.25362319],
       [ 0.47572464,  0.08695652,  0.15072464,  0.2865942 ],
       [ 0.45108696,  0.075     ,  0.17536232,  0.29855072],
       [ 0.53985507,  0.12898551,  0.0865942 ,  0.24456522],
       [ 0.5807971 ,  0.14528986,  0.04565217,  0.22826087],
       [ 0.47898551,  0.11123188,  0.14746377,  0.26231884],
       [ 0.55507246,  0.09384058,  0.07137681,  0.27971014],
       [ 0.51521739,  0.09927536,  0.11123188,  0.27427536],
       [ 0.52246377,  0.07934783,  0.10398551,  0.2942029 ]])
```

(c) First, we calculate the information matrix.

```
In [3]:
from math import log
I = np.zeros((30,30))
for i in range(n):
    I[i][i] = x[i]*log(x[i])+(1-x[i])*log(1-x[i])
    for j in range(i+1, n):
        I[j][i] = I[i][j] =
                xij[i][j][0]*log(xij[i][j][0]/((1-x[i])*(1-x[j])))+xij[
                i][j][1]*log(xij[i][j][1]/(x[i]*(1-x[j])))+xij[i][j][2]
                *log(xij[i][j][2]/((1-x[i])*x[j]))+xij[i][j][3]*log(xij
                [i][j][3]/(x[i]*x[j]))
```

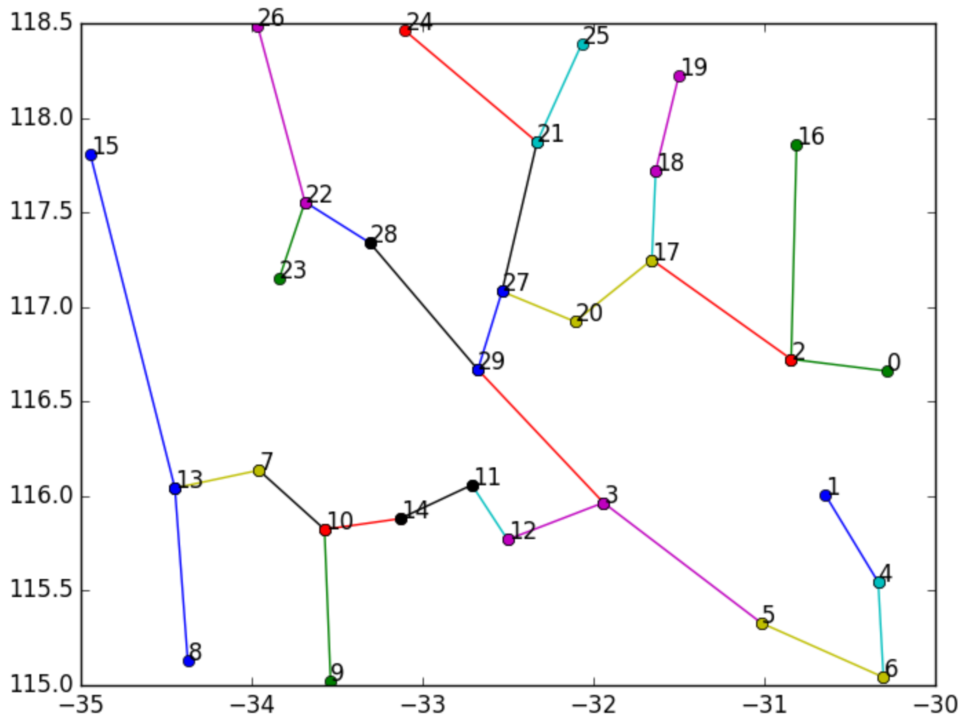
Then, a modified Prim's algorithm will be utilized to find out the maximum spanning tree.

```
In [4]:
#A = adjacency matrix, u = vertex u, v = vertex v
def weight(A, u, v):
    return A[u][v]

#A = adjacency matrix, u = vertex u
def adjacent(A, u):
    L = []
    for x in range(len(A)):
        if x != u:
            L.insert(0,x)
    return L

#Q = max queue
def extractMax(Q):
    q = Q[0]
    Q.remove(Q[0])
    return q

#Q = max queue, V = vertex list
def increaseKey(Q, K):
    for i in range(len(Q)):
        for j in range(len(Q)):
            if K[Q[i]] > K[Q[j]]:
                s = Q[i]
                Q[i] = Q[j]
                Q[j] = s

#V = vertex list, A = adjacency list, r = root
def prim(V, A, r):
    u = 0
    v = 0

    # initialize and set each value of the array P (pi) to none
    # pi holds the parent of u, so P(v)=u means u is the parent of v
    P = [None]*len(V)

    # initialize and set each value of the array K (key) to -999999
    K = [-999999]*len(V)

    # initialize the max queue and fill it with all vertices in V
    Q = [0]*len(V)
    for u in range(len(Q)):
        Q[u] = V[u]

    # set the key of the root to 0
    K[r] = 0
    increaseKey(Q, K)   # maintain the max queue
```

```
  # loop while the max queue is not empty
  while len(Q) > 0:
       u = extractMax(Q)   # pop the first vertex off the max queue

       # loop through the vertices adjacent to u
       Adj = adjacent(A, u)
       for v in Adj:
          w = weight(A, u, v)   # get the weight of the edge uv

          # proceed if v is in Q and the weight of uv is great than v's key
          if Q.count(v)>0 and w > K[v]:
             # set v's parent to u
             P[v] = u
             # v's key to the weight of uv
             K[v] = w
             increaseKey(Q, K)   # maintain the min queue
   return P

V = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
22, 23, 24, 25, 26, 27, 28, 29]

P = prim(V, I, 0)
print(P)

Out [4]:
[None, 4, 0, 29, 6, 3, 5, 10, 13, 10, 14, 12, 3, 7, 11, 13, 2, 2, 17, 18, 17, 27,
28, 22, 21, 21, 22, 20, 29, 27]
```

Last but not least, with the following code we can output the plot of the graph:

```
In [5]:
import matplotlib.pyplot as plt
p,q = loc.shape
fig,ax=plt.subplots(1,1)
for i in range(1,p):
   x = [loc[i][0],loc[P[i]][0]]
   y = [loc[i][1],loc[P[i]][1]]
   plt.plot(x,y,'o-')
for i in range(p):
   plt.text(loc[i][0],loc[i][1],str(i))
plt.show()
```



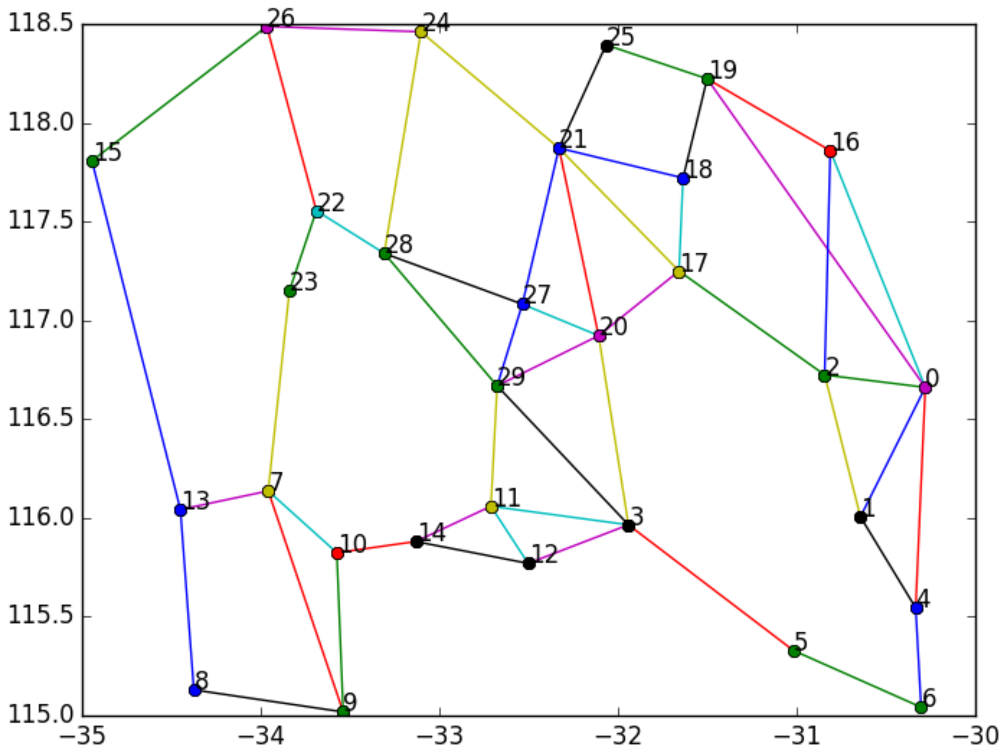

(d)

```
In [6]:
ll = I[0][0]
for i in range(1,n):
   I1 = I[i][i] + I[P[i]][i]
   ll += I1
print('log likelihood: {}'.format(ll))

Out [6]:
log likelihood: -11.098614327815277
```

## Problem 2:

(a)

The edges are plotted as follows:



(b) & (c)

First, the data will be transferred into pyGM objects.

(a)

```
In [7]:
import numpy as np
import matplotlib.pyplot as plt
loc = np.genfromtxt('data/locations.txt',delimiter=None)
E = np.genfromtxt('data/edges.txt', delimiter=None)

X = [Var(i,2) for i in range(30)]
P = [Factor(X[i],0.0) for i in range(n)]
for i in range(n):
  P[i][0] = 1-x[i]
  P[i][1] = x[i]

W = [[0] * n for i in range(n)]
for i in range(n):
  W[i][i] = x[i]
  for j in range(i+1, n):
    W[i][j] = Factor([X[i], X[j]], 0.0)
    W[i][j][0,0] = xij[i][j][0]
    W[i][j][1,0] = xij[i][j][1]
    W[i][j][0,1] = xij[i][j][2]
    W[i][j][1,1] = xij[i][j][3]
    W[j][i] = W[i][j]
```

The empirical marginal probabilities are already included in xij, which is calculated in problem1 part b. The marginal probability of edge (i, j ) can be found in xij[i][j].

The IPF process is operate with the following code.

```
In [8]:
factors = [Factor([X[int(e[0])], X[int(e[1])]], 1.0) for e in E]
pri = [1.0 for Xi in X]
for i in range(15):
    ll = 0
    for j, e in enumerate(E):
        model_ve = GraphModel(factors)
        k,l = int(e[0]), int(e[1])
        pri[k],pri[l] = 2.0, 2.0
        order = eliminationOrder(model_ve, orderMethod = 'minfill', priority =
pri)[0]
        sumElim = lambda F,Xlist: F.sum(Xlist)   # helper function for eliminate
        model_ve.eliminate(order[:-2], sumElim)  # eliminate all but last two
        p = model_ve.joint()
        p /= p.sum()
        factors[j] *= W[k][l]/p  # update the factors
        pri[k],pri[l] = 1.0, 1.0

    # calculating the Z value
    model_ve = GraphModel(factors)
    order = eliminationOrder(model_ve, orderMethod = 'minfill', priority = pri)[0]
    sumElim = lambda F,Xlist: F.sum(Xlist)   # helper function for eliminate
    model_ve.eliminate(order, sumElim)  # eliminate all variables to get Z
    Z = model_ve.joint()

    # calculating the log likelihood
    for k in range(m):
        for j,e in enumerate(E):
            u, v = int(e[0]), int(e[1])
            a, b = int(D[k][u]), int(D[k][v])
            ll += log(factors[j][a,b])
    ll /= m
    ll -= log(Z.table)
    print('step: {} log likelihood: {} logZ: {}'.format(i+1, ll, log(Z.table)))

Out [8]:
step: 1 log likelihood: -10.527552408560377 logZ: 20.79441541679836
step: 2 log likelihood: -9.766228337941332 logZ: 20.79441541679836
step: 3 log likelihood: -9.709305478153006 logZ: 20.79441541679836
step: 4 log likelihood: -9.695579756168586 logZ: 20.79441541679836
step: 5 log likelihood: -9.692246358683846 logZ: 20.79441541679836
step: 6 log likelihood: -9.691553503619216 logZ: 20.79441541679836
step: 7 log likelihood: -9.691414842252 logZ: 20.794415416798362
step: 8 log likelihood: -9.691386009889872 logZ: 20.794415416798362
step: 9 log likelihood: -9.69137972473651 logZ: 20.794415416798362
step: 10 log likelihood: -9.691378298007859 logZ: 20.794415416798362
step: 11 log likelihood: -9.691377938590907 logZ: 20.794415416798362
step: 12 log likelihood: -9.691377836134777 logZ: 20.794415416798362
step: 13 log likelihood: -9.691377806937822 logZ: 20.794415416798362
step: 14 log likelihood: -9.691377799601709 logZ: 20.794415416798362
step: 15 log likelihood: -9.691377798040286 logZ: 20.794415416798362
```