Communication Network

Computer assignment 2

Vinay Chittam

Fet110

1. Bellman Ford

**Output**

)

The distances of each node to destination node in each hops 0,1,2--9 are as follows

The coloumns represent nodes

The rows represent hops

A0 B0 -- J0

A1 B1 -- J1

A2 B2 -- J2

[[ 0. inf inf inf inf inf inf inf inf inf]

[ 0. 1. inf inf 1. inf inf inf inf inf]

[ 0. 1. 2. 6. 1. inf 2. inf inf inf]

[ 0. 1. 2. 6. 1. 5. 2. 3. 7. 6.]

[ 0. 1. 2. 4. 1. 5. 2. 3. 6. 6.]

[ 0. 1. 2. 4. 1. 5. 2. 3. 5. 6.]

[ 0. 1. 2. 4. 1. 5. 2. 3. 5. 6.]

[ 0. 1. 2. 4. 1. 5. 2. 3. 5. 6.]

[ 0. 1. 2. 4. 1. 5. 2. 3. 5. 6.]

[ 0. 1. 2. 4. 1. 5. 2. 3. 5. 6.]]

This matrix represents nodes i.e. coloums represent nodes and rows represent hops

[['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']]

This matrix represents corresponding next node in the shortest path from a partricular node to destination

[['a' '-' '-' '-' '-' '-' '-' '-' '-' '-']

['a' 'a' '-' '-' 'a' '-' '-' '-' '-' '-']

['a' 'a' 'b' 'e' 'a' '-' 'e' '-' '-' '-']

['a' 'a' 'b' 'e' 'a' 'c' 'e' 'g' 'd' 'c']

['a' 'a' 'b' 'h' 'a' 'c' 'e' 'g' 'f' 'c']

['a' 'a' 'b' 'h' 'a' 'c' 'e' 'g' 'd' 'c']

['a' 'a' 'b' 'h' 'a' 'c' 'e' 'g' 'd' 'c']

['a' 'a' 'b' 'h' 'a' 'c' 'e' 'g' 'd' 'c']

['a' 'a' 'b' 'h' 'a' 'c' 'e' 'g' 'd' 'c']

['a' 'a' 'b' 'h' 'a' 'c' 'e' 'g' 'd' 'c']]

**Code:**

#!/usr/bin/env python

import numpy as np

from operator import itemgetter

#cmat=np.matrix([[daa,dab,dac,dad,dae,daf,dag,dah,dai,daj,dak],[dba,dbb,dbc,dbd,dbe,dbf,dbg,dbh,dbi,dbj,dbk]])

#cmat=np.matrix

inf=float('inf')

lis0=[]

lis1=[1,inf,1,inf,inf,inf,inf,inf,inf,inf]

lis2=[inf,1,inf,inf,inf,3,1,inf,inf,4]

lis3=[inf,inf,inf,inf,5,inf,inf,1,1,2]

lis4=[1,inf,inf,5,inf,inf,1,inf,inf,inf]

lis5=[inf,inf,3,inf,inf,inf,inf,inf,1,inf]

lis6=[inf,inf,1,inf,1,inf,inf,1,inf,inf]

lis7=[inf,inf,inf,1,inf,inf,1,inf,inf,inf]

lis8=[inf,inf,inf,1,inf,1,inf,inf,inf,inf]

lis9=[inf,inf,4,2,inf,inf,inf,inf,inf,inf]

cmat=np.array([lis1,lis2,lis3,lis4,lis5,lis6,lis7,lis8,lis9])

b=['a','b','c','d','e','f','g','h','i','j']

nodes=np.array([b,b,b,b,b,b,b,b,b,b])

print nodes.shape

ll=[0,inf,inf,inf,inf,inf,inf,inf,inf,inf]

print cmat.shape

node=np.array([b,b,b,b,b,b,b,b,b,b])

#span\_tree=np.zeros((10,10))

dmat=np.array([ll,ll,ll,ll,ll,ll,ll,ll,ll,ll])

print dmat.shape

#i=0;j=0;ci=0;cj=0;di=0;dj=0

i=0;j=0;ci=0;cj=0;di=0;dj=0

#temp=[dmat[di,dj],dmat[di,dj+1],dmat[di,dj+2],dmat[di,dj+3],dmat[di,dj+4],dmat[di,dj+5],

#dmat[di,dj+6],dmat[di,dj+7],dmat[di,dj+8],dmat[di,dj+9]

#this for loop is for each hop and in each hop i calculated distance of all nodes to destination and incremented hops

for h in [1,2,3,4,5,6,7,8,9]:

dmat[i+1,j]=0

#j=j+1

ind,dmat[i+1,j+1]= min(enumerate([cmat[ci,cj]+dmat[di,dj],cmat[ci,cj+1]+

dmat[di,dj+1],cmat[ci,cj+2]+dmat[di,dj+2],cmat[ci,cj+3]+dmat[di,dj+3],cmat[ci,cj+4]+

dmat[di,dj+4],cmat[ci,cj+5]+dmat[di,dj+5],cmat[ci,cj+6]+dmat[di,dj+6],cmat[ci,cj+7]+

dmat[di,dj+7],cmat[ci,cj+8]+dmat[di,dj+8],cmat[ci,cj+9]+dmat[di,dj+9]]),key=itemgetter(1))

nodes[i+1,j+1]=node[ci,ind]

ci=ci+1

ind,dmat[i+1,j+2]= min(enumerate([cmat[ci,cj]+dmat[di,dj],cmat[ci,cj+1]+

dmat[di,dj+1],cmat[ci,cj+2]+dmat[di,dj+2],cmat[ci,cj+3]+dmat[di,dj+3],cmat[ci,cj+4]+

dmat[di,dj+4],cmat[ci,cj+5]+dmat[di,dj+5],cmat[ci,cj+6]+dmat[di,dj+6],cmat[ci,cj+7]+

dmat[di,dj+7],cmat[ci,cj+8]+dmat[di,dj+8],cmat[ci,cj+9]+dmat[di,dj+9]]),key=itemgetter(1))

nodes[i+1,j+2]=node[ci,ind]

ci=ci+1

ind,dmat[i+1,j+3]= min(enumerate([cmat[ci,cj]+dmat[di,dj],cmat[ci,cj+1]+

dmat[di,dj+1],cmat[ci,cj+2]+dmat[di,dj+2],cmat[ci,cj+3]+dmat[di,dj+3],cmat[ci,cj+4]+

dmat[di,dj+4],cmat[ci,cj+5]+dmat[di,dj+5],cmat[ci,cj+6]+dmat[di,dj+6],cmat[ci,cj+7]+

dmat[di,dj+7],cmat[ci,cj+8]+dmat[di,dj+8],cmat[ci,cj+9]+dmat[di,dj+9]]),key=itemgetter(1))

nodes[i+1,j+3]=node[ci,ind]

ci=ci+1

ind,dmat[i+1,j+4]= min(enumerate([cmat[ci,cj]+dmat[di,dj],cmat[ci,cj+1]+

dmat[di,dj+1],cmat[ci,cj+2]+dmat[di,dj+2],cmat[ci,cj+3]+dmat[di,dj+3],cmat[ci,cj+4]+

dmat[di,dj+4],cmat[ci,cj+5]+dmat[di,dj+5],cmat[ci,cj+6]+dmat[di,dj+6],cmat[ci,cj+7]+

dmat[di,dj+7],cmat[ci,cj+8]+dmat[di,dj+8],cmat[ci,cj+9]+dmat[di,dj+9]]),key=itemgetter(1))

nodes[i+1,j+4]=node[ci,ind]

ci=ci+1

ind,dmat[i+1,j+5]= min(enumerate([cmat[ci,cj]+dmat[di,dj],cmat[ci,cj+1]+

dmat[di,dj+1],cmat[ci,cj+2]+dmat[di,dj+2],cmat[ci,cj+3]+dmat[di,dj+3],cmat[ci,cj+4]+

dmat[di,dj+4],cmat[ci,cj+5]+dmat[di,dj+5],cmat[ci,cj+6]+dmat[di,dj+6],cmat[ci,cj+7]+

dmat[di,dj+7],cmat[ci,cj+8]+dmat[di,dj+8],cmat[ci,cj+9]+dmat[di,dj+9]]),key=itemgetter(1))

nodes[i+1,j+5]=node[ci,ind]

ci=ci+1

ind,dmat[i+1,j+6]= min(enumerate([cmat[ci,cj]+dmat[di,dj],cmat[ci,cj+1]+

dmat[di,dj+1],cmat[ci,cj+2]+dmat[di,dj+2],cmat[ci,cj+3]+dmat[di,dj+3],cmat[ci,cj+4]+

dmat[di,dj+4],cmat[ci,cj+5]+dmat[di,dj+5],cmat[ci,cj+6]+dmat[di,dj+6],cmat[ci,cj+7]+

dmat[di,dj+7],cmat[ci,cj+8]+dmat[di,dj+8],cmat[ci,cj+9]+dmat[di,dj+9]]),key=itemgetter(1))

nodes[i+1,j+6]=node[ci,ind]

ci=ci+1

ind,dmat[i+1,j+7]= min(enumerate([cmat[ci,cj]+dmat[di,dj],cmat[ci,cj+1]+

dmat[di,dj+1],cmat[ci,cj+2]+dmat[di,dj+2],cmat[ci,cj+3]+dmat[di,dj+3],cmat[ci,cj+4]+

dmat[di,dj+4],cmat[ci,cj+5]+dmat[di,dj+5],cmat[ci,cj+6]+dmat[di,dj+6],cmat[ci,cj+7]+

dmat[di,dj+7],cmat[ci,cj+8]+dmat[di,dj+8],cmat[ci,cj+9]+dmat[di,dj+9]]),key=itemgetter(1))

nodes[i+1,j+7]=node[ci,ind]

ci=ci+1

ind,dmat[i+1,j+8]= min(enumerate([cmat[ci,cj]+dmat[di,dj],cmat[ci,cj+1]+

dmat[di,dj+1],cmat[ci,cj+2]+dmat[di,dj+2],cmat[ci,cj+3]+dmat[di,dj+3],cmat[ci,cj+4]+

dmat[di,dj+4],cmat[ci,cj+5]+dmat[di,dj+5],cmat[ci,cj+6]+dmat[di,dj+6],cmat[ci,cj+7]+

dmat[di,dj+7],cmat[ci,cj+8]+dmat[di,dj+8],cmat[ci,cj+9]+dmat[di,dj+9]]),key=itemgetter(1))

nodes[i+1,j+8]=node[ci,ind]

ci=ci+1

ind,dmat[i+1,j+9]= min(enumerate([cmat[ci,cj]+dmat[di,dj],cmat[ci,cj+1]+

dmat[di,dj+1],cmat[ci,cj+2]+dmat[di,dj+2],cmat[ci,cj+3]+dmat[di,dj+3],cmat[ci,cj+4]+

dmat[di,dj+4],cmat[ci,cj+5]+dmat[di,dj+5],cmat[ci,cj+6]+dmat[di,dj+6],cmat[ci,cj+7]+

dmat[di,dj+7],cmat[ci,cj+8]+dmat[di,dj+8],cmat[ci,cj+9]+dmat[di,dj+9]]),key=itemgetter(1))

nodes[i+1,j+9]=node[ci,ind]

i=i+1

di=di+1

ci=0

for u in [1,2,3,4,5,6,7,8,9]:

nodes[0,u]='-'

nodes[1,2]='-'

nodes[1,3]='-'

nodes[1,5]='-'

nodes[2,5]='-'

nodes[1,6]='-'

nodes[1,7]='-'

nodes[2,7]='-'

nodes[1:3,8]='-'

nodes[1:3,9]='-'

print "The distances of each node to destination node in each hops 0,1,2--9 are as follows "

print "The coloumns represent nodes"

print "The rows represent hops"

print "A0 B0 -- J0\nA1 B1 -- J1\nA2 B2 -- J2"

print dmat

print "\nThis matrix represents nodes i.e. coloums represent nodes and rows represent hops"

print node

print "\nThis matrix represents corresponding next node in the shortest path from a partricular node to destination "

print nodes

#cmat1=cmat.reshape(9,10)

#print cmat1.shape

2. Dijkstra’s

**Output**:

The coloumns represent nodes

The rows represent hops

This matrix represents the distance matrix of all nodes

[[ 0. 1. inf inf 1. inf inf inf inf inf]

[ 0. 1. 2. inf 1. inf inf inf inf inf]

[ 0. 1. 2. 6. 1. inf 2. inf inf inf]

[ 0. 1. 2. 6. 1. 5. 2. inf inf 6.]

[ 0. 1. 2. 6. 1. 5. 2. 3. inf 6.]

[ 0. 1. 2. 4. 1. 5. 2. 3. inf 6.]

[ 0. 1. 2. 4. 1. 5. 2. 3. 5. 6.]

[ 0. 1. 2. 4. 1. 5. 2. 3. 5. 6.]

[ 0. 1. 2. 4. 1. 5. 2. 3. 5. 6.]

[ 0. 1. 2. 4. 1. 5. 2. 3. 5. 6.]]

The set of nodes that are added every hop

['a', 'b', 'e', 'c', 'g', 'h', 'd', 'f', 'i', 'j']

This represents all nodes

[['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']]

This represents next node in the shortest path to destination

[['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'b' 'b' 'e' 'b' 'b' 'b' 'b' 'b']

['a' 'b' 'c' 'e' 'e' 'e' 'e' 'e' 'e' 'e']

['a' 'b' 'c' 'd' 'e' 'c' 'g' 'c' 'c' 'c']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'g' 'g' 'j']

['a' 'b' 'c' 'h' 'e' 'f' 'g' 'h' 'h' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'd' 'd']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']]

The above matrix is calculated using the formula :

Dj=min(Dj,dji+Di)

**Code:**

#!/usr/bin/env python

import numpy as np

from operator import itemgetter

inf=float('inf')

lis0=[]

lis1=[1,inf,1,inf,inf,inf,inf,inf,inf,inf]

lis2=[inf,1,inf,inf,inf,3,1,inf,inf,4]

lis3=[inf,inf,inf,inf,5,inf,inf,1,1,2]

lis4=[1,inf,inf,5,inf,inf,1,inf,inf,inf]

lis5=[inf,inf,3,inf,inf,inf,inf,inf,1,inf]

lis6=[inf,inf,1,inf,1,inf,inf,1,inf,inf]

lis7=[inf,inf,inf,1,inf,inf,1,inf,inf,inf]

lis8=[inf,inf,inf,1,inf,1,inf,inf,inf,inf]

lis9=[inf,inf,4,2,inf,inf,inf,inf,inf,inf]

cmat=np.array([lis1,lis2,lis3,lis4,lis5,lis6,lis7,lis8,lis9])

p=[0]

pcomp=[1,2,3,4,5,6,7,8,9]

b=['a','b','c','d','e','f','g','h','i','j']

nodes=np.array([b,b,b,b,b,b,b,b,b,b])

#print nodes.shape

ll=[0,inf,inf,inf,inf,inf,inf,inf,inf,inf]

print cmat.shape

node=np.array([b,b,b,b,b,b,b,b,b,b])

#span\_tree=np.zeros((10,10))

dmat=np.array([ll,ll,ll,ll,ll,ll,ll,ll,ll,ll])

dmat[0,1]=1;dmat[0,4]=1

pd=[dmat[0,0]]

print dmat.shape

#i=0;j=0

#min([dmat[i,j+1],cmat[j]] )

i=0;j=0;ci=0;cj=0;di=0;dj=0

tp=[dmat[di,dj]]

temp=[dmat[di,dj+1],dmat[di,dj+2],dmat[di,dj+3],dmat[di,dj+4],dmat[di,dj+5],

dmat[di,dj+6],dmat[di,dj+7],dmat[di,dj+8],dmat[di,dj+9]]

for h in [1,2,3,4,5,6,7,8,9]:

#counter=0

#while counter<=8:

#ind,nn=min(enumerate([dmat[di,dj],dmat[di,dj+1],dmat[di,dj+2],dmat[di,dj+3],dmat[di,dj+4],dmat[di,dj+5],

#dmat[di,dj+6],dmat[di,dj+7],dmat[di,dj+8],dmat[di,dj+9]]),key=itemgetter(1))

temp=[dmat[di,dj+1],dmat[di,dj+2],dmat[di,dj+3],dmat[di,dj+4],dmat[di,dj+5],

dmat[di,dj+6],dmat[di,dj+7],dmat[di,dj+8],dmat[di,dj+9]]

if h==2:

del temp[ind]

if h==3:

ind,nn=min(enumerate(temp),key=itemgetter(1))

del temp[ind]

ind,nn=min(enumerate(temp),key=itemgetter(1))

del temp[ind]

if h==4:

ind,nn=min(enumerate(temp),key=itemgetter(1))

del temp[ind]

ind,nn=min(enumerate(temp),key=itemgetter(1))

del temp[ind]

ind,nn=min(enumerate(temp),key=itemgetter(1))

del temp[ind]

if h==5:

for o in [1,2,3,4]:

ind,nn=min(enumerate(temp),key=itemgetter(1))

del temp[ind]

if h==6:

for o in [1,2,3,4,5]:

ind,nn=min(enumerate(temp),key=itemgetter(1))

del temp[ind]

if h==7:

for o in [1,2,3,4,5,6]:

ind,nn=min(enumerate(temp),key=itemgetter(1))

del temp[ind]

if h==8:

for o in [1,2,3,4,5,6,7]:

ind,nn=min(enumerate(temp),key=itemgetter(1))

del temp[ind]

if h==9:

for o in [1,2,3,4,5,6,7,8]:

ind,nn=min(enumerate(temp),key=itemgetter(1))

del temp[ind]

ind,nn=min(enumerate(temp),key=itemgetter(1))

pd.append(nn)

p.append(pcomp[ind])

del temp[ind]

del pcomp[ind]

if p[-1]==1:

cc=1

k='b'

elif p[-1]==2:

cc=2

k='c'

elif p[-1]==3:

cc=3

k='d'

elif p[-1]==4:

cc=4

k='e'

elif p[-1]==5:

cc=5

k='f'

elif p[-1]==6:

cc=6

k='g'

elif p[-1]==7:

cc=7

k='h'

elif p[-1]==8:

cc=8

k='i'

elif p[-1]==9:

cc=9

k='j'

#temp[ind]=inf

#j=j+1

dmat[i+1,j+1]= min([dmat[i,j+1],cmat[j+1-1,cc]+nn])

if dmat[i,j+1]>=cmat[j+1-1,cc]+nn:

node[i+1,j+1]=nodes[i,cc]

else:

node[i+1,j+1]=nodes[i,j+1]

dmat[i+1,j+2]= min([dmat[i,j+2],cmat[j+2-1,cc]+nn])

if dmat[i,j+2]>=cmat[j+1-1,cc]+nn:

node[i+1,j+2]=nodes[i,cc]

else:

node[i+1,j+2]=nodes[i,j+2]

dmat[i+1,j+3]= min([dmat[i,j+3],cmat[j+3-1,cc]+nn])

if dmat[i,j+3]>=cmat[j+3-1,cc]+nn:

node[i+1,j+3]=nodes[i,cc]

else:

node[i+1,j+3]=nodes[i,j+3]

dmat[i+1,j+4]= min([dmat[i,j+4],cmat[j+4-1,cc]+nn])

if dmat[i,j+4]>=cmat[j+4-1,cc]+nn:

node[i+1,j+4]=nodes[i,cc]

else:

node[i+1,j+4]=nodes[i,j+4]

dmat[i+1,j+5]= min([dmat[i,j+5],cmat[j+5-1,cc]+nn])

if dmat[i,j+5]>=cmat[j+5-1,cc]+nn:

node[i+1,j+5]=nodes[i,cc]

else:

node[i+1,j+5]=nodes[i,j+5]

dmat[i+1,j+6]= min([dmat[i,j+6],cmat[j+6-1,cc]+nn])

if dmat[i,j+6]>=cmat[j+6-1,cc]+nn:

node[i+1,j+6]=nodes[i,cc]

else:

node[i+1,j+6]=nodes[i,j+6]

dmat[i+1,j+7]= min([dmat[i,j+7],cmat[j+7-1,cc]+nn])

if dmat[i,j+7]>=cmat[j+7-1,cc]+nn:

node[i+1,j+7]=nodes[i,cc]

else:

node[i+1,j+7]=nodes[i,j+7]

dmat[i+1,j+8]= min([dmat[i,j+8],cmat[j+8-1,cc]+nn])

if dmat[i,j+8]>=cmat[j+8-1,cc]+nn:

node[i+1,j+8]=nodes[i,cc]

else:

node[i+1,j+8]=nodes[i,j+8]

dmat[i+1,j+9]= min([dmat[i,j+9],cmat[j+9-1,cc]+nn])

if dmat[i,j+9]>=cmat[j+9-1,cc]+nn:

node[i+1,j+9]=nodes[i,cc]

else:

node[i+1,j+9]=nodes[i,j+9]

#if min(temp)==inf:

#print "over"

#break

i=i+1

di=di+1

print pd

print "\nThe coloumns represent nodes"

print "\nThe rows represent hops"

print "\nThis matrix represents the distance matrix of all nodes"

print dmat

print "The set of nodes that are added every hop"

print ['a','b','e','c','g','h','d','f','i','j']

print "\nThis represents all nodes"

print nodes

print "\nThis represents next node in the shortest path to destination"

print node

3.

Outputs:

**Output Bellman Ford**

)

The distances of each node to destination node in each hops 0,1,2--9 are as follows

The coloumns represent nodes

The rows represent hops

A0 B0 -- J0

A1 B1 -- J1

A2 B2 -- J2

[[ 0. inf inf inf inf inf inf inf inf inf]

[ 0. 1. inf inf 1. inf inf inf inf inf]

[ 0. 1. 2. 6. 1. inf 2. inf inf inf]

[ 0. 1. 2. 6. 1. 5. 2. 3. 7. 6.]

[ 0. 1. 2. 4. 1. 5. 2. 3. 6. 6.]

[ 0. 1. 2. 4. 1. 5. 2. 3. 5. 6.]

[ 0. 1. 2. 4. 1. 5. 2. 3. 5. 6.]

[ 0. 1. 2. 4. 1. 5. 2. 3. 5. 6.]

[ 0. 1. 2. 4. 1. 5. 2. 3. 5. 6.]

[ 0. 1. 2. 4. 1. 5. 2. 3. 5. 6.]]

This matrix represents nodes i.e. coloums represent nodes and rows represent hops

[['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']]

This matrix represents corresponding next node in the shortest path from a partricular node to destination

[['a' '-' '-' '-' '-' '-' '-' '-' '-' '-']

['a' 'a' '-' '-' 'a' '-' '-' '-' '-' '-']

['a' 'a' 'b' 'e' 'a' '-' 'e' '-' '-' '-']

['a' 'a' 'b' 'e' 'a' 'c' 'e' 'g' 'd' 'c']

['a' 'a' 'b' 'h' 'a' 'c' 'e' 'g' 'f' 'c']

['a' 'a' 'b' 'h' 'a' 'c' 'e' 'g' 'd' 'c']

['a' 'a' 'b' 'h' 'a' 'c' 'e' 'g' 'd' 'c']

['a' 'a' 'b' 'h' 'a' 'c' 'e' 'g' 'd' 'c']

['a' 'a' 'b' 'h' 'a' 'c' 'e' 'g' 'd' 'c']

['a' 'a' 'b' 'h' 'a' 'c' 'e' 'g' 'd' 'c']]

**Output Dijstraws**:

The coloumns represent nodes

The rows represent hops

This matrix represents the distance matrix of all nodes

[[ 0. 1. inf inf 1. inf inf inf inf inf]

[ 0. 1. 2. inf 1. inf inf inf inf inf]

[ 0. 1. 2. 6. 1. inf 2. inf inf inf]

[ 0. 1. 2. 6. 1. 5. 2. inf inf 6.]

[ 0. 1. 2. 6. 1. 5. 2. 3. inf 6.]

[ 0. 1. 2. 4. 1. 5. 2. 3. inf 6.]

[ 0. 1. 2. 4. 1. 5. 2. 3. 5. 6.]

[ 0. 1. 2. 4. 1. 5. 2. 3. 5. 6.]

[ 0. 1. 2. 4. 1. 5. 2. 3. 5. 6.]

[ 0. 1. 2. 4. 1. 5. 2. 3. 5. 6.]]

The set of nodes that are added every hop

['a', 'b', 'e', 'c', 'g', 'h', 'd', 'f', 'i', 'j']

This represents all nodes

[['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']]

This represents next node in the shortest path to destination

[['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'b' 'b' 'e' 'b' 'b' 'b' 'b' 'b']

['a' 'b' 'c' 'e' 'e' 'e' 'e' 'e' 'e' 'e']

['a' 'b' 'c' 'd' 'e' 'c' 'g' 'c' 'c' 'c']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'g' 'g' 'j']

['a' 'b' 'c' 'h' 'e' 'f' 'g' 'h' 'h' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'd' 'd']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']

['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j']]

Spanning Tree:

Dijkstras:

a->’-’

b->a

c->b

d->h

h->g

g->e

e->a

f->c

i->d

j->c

Bellman Ford:

a->’-‘

b->’a’

c->’b’

d->h

h->g

g->e

e->a

f->c

i->d

j->c

The spanning tree for both the algorithms are same. I calculated it using the next hop that each node takes in order to reach its destination and the distance matrix I calculated. It matches perfectly with the network.

Note: I took node k as I from the question figure.