class NodArbore:

def \_\_init\_\_(self, info, parinte=None):

self.info = info

self.parinte = parinte

def drumRadacina(self) :

l=[]

nod=self

while nod is not None:

l.insert(0,nod)

nod=nod.parinte

return l

def vizitat(self) :

nod=self.parinte

while nod is not None:

if nod.info==self.info:

return True

nod=nod.parinte

return False

def \_\_str\_\_(self):

return str(self.info)

def \_\_repr\_\_(self):

return "({}, ({}))".format(self.info, "->".join([str(x) for x in self.drumRadacina()]))

class Graf:

def \_\_init\_\_(self, matr, start, scopuri):

self.matr=matr

self.start=start

self.scopuri=scopuri

def scop(self, infoNod):

return infoNod in self.scopuri

def succesori(self, nod):

l=[]

for i in range(len(self.matr)):

if self.matr[nod.info][i]==1:

nodNou=NodArbore(i,nod)

if not nodNou.vizitat():

l.append(nodNou)

return l

def breadth\_first(gr, nsol):

c=[NodArbore(gr.start)]

while c:

nodCurent=c.pop(0)

if gr.scop(nodCurent.info):

print(repr(nodCurent))

nsol-=1

if nsol==0:

return

lSuccesori=gr.succesori(nodCurent)

c+=lSuccesori

m = [

[0, 1, 0, 1, 1, 0, 0, 0, 0, 0],

[1, 0, 1, 0, 0, 1, 0, 0, 0, 0],

[0, 1, 0, 0, 0, 1, 0, 1, 0, 0],

[1, 0, 0, 0, 0, 0, 1, 0, 0, 0],

[1, 0, 0, 0, 0, 0, 0, 1, 0, 0],

[0, 1, 1, 0, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 1, 0, 0, 0, 0, 0, 0],

[0, 0, 1, 0, 1, 0, 0, 0, 1, 1],

[0, 0, 0, 0, 0, 0, 0, 1, 0, 0],

[0, 0, 0, 0, 0, 0, 0, 1, 0, 0]

]

start = 0

scopuri = [5, 9]

gr=Graf(m, start, scopuri)

breadth\_first(gr,3)