```
#ZFN on GP Graphs

#Nick Montana

import itertools

#First, I need to create a funtion that creates a 2D array that is the adjacency
#matrix of a Generalized Petersen Graph.

#First, let's define circulation
def circ (myarray):
    circarray = []
    circarray.append (myarray)

    #I need a previous array.
    prevarray = myarray

    for i in range (1, len(myarray)):
        temparray = []
        temparray.append (prevarray [len (prevarray) - 1])
        for i in range (1, len(myarray)):
            temparray.append (prevarray [i - 1])
        prevarray = temparray
        circarray.append (temparray)
    return circarray

#Let's create an Identity Matrix:
def idenMat (n):
    idenarray = []

    for i in range (0, n):
        temparray = []
        for s in range (0, n):
            if (s == i):
                temparray.append (1)
            else:
                temparray.append (0)
        idenarray.append (temparray)
    return idenarray


#Let's just try to do the very first vertex and then see if we could generalize:
print ("Enter the parameters for a Generalized Petersen Graph defined by")
print ("GP (n, k): ")
n = int (input ("n: "))
k = int (input ("k: "))

#Let's create what to take the circ of:
```

```
#Outer
def outer ():
    myarray = []
    for i in range (0, n):
        if i == 1:
            myarray.append (1)
        elif i == n - 1:
            myarray.append (1)
        else:
            myarray.append (0)
    return myarray

#Inner
def inner ():
    myarray = []
    for i in range (0, n):
        if i == k:
            myarray.append (1)
        elif i == n - 1 - (k - 1):
            myarray.append (1)
        else:
            myarray.append (0)
    return myarray

#This function creates the adjacency matrix:
def adjMat ():
    gparray = []
    Outer = circ (outer ())
    Inner = circ (inner ())
    for x in range (0, n):
        temparray = []
        for i in range (0, n):
            temparray.append (Outer [x] [i])

        for i in range (0, n):
            temparray.append (idenMat (n) [x] [i])
        gparray.append (temparray)

    for x in range (0, n):
        temparray = []
        for i in range (0, n):
            temparray.append (idenMat (n) [x] [i])

        for i in range (0, n):
            temparray.append (Inner [x] [i])

        gparray.append (temparray)
    return gparray
```

```
G = adjMat ()


def vertexSet ():
    vertexSet = set ()
    for x in range (0, len (G)):
        vertexSet.add (x)
    return vertexSet

#Let's form a neigbor set funciton.
#This function returns the neighbors of v as a set.

def N (v, G):
    neighbors = set ()
    for i in range (0, len (G)):
        if (G [v] [i] == 1):
            neighbors.add (i)
    return neighbors

print ("The Adjacency Matrix of GP(",n,",",k,") is: ")
for s in range (0, 2*n):
    print (G [s])

#Now let's do the ZFN:

def isForce (Set):
    isForce = False
    force = True
    mySet = set ()
    size = len (Set)
    while Set != vertexSet () and force:
        for i in Set:
            counter = 0
            tempSet = set ()
            for x in N (i, G):
                if x in Set:
                    counter = counter + 1
                else:
                    tempSet.add (x)
            if counter == len (N (i,G)) - 1:
                mySet = set.union (mySet, tempSet)
        Set = set.union (Set, mySet)
        if len(Set) == size:
            force = False
        size = len (Set)
    if (Set == vertexSet ()):
        isForce = True
    return isForce

def halfVSet ():
    x = int ((n / 2) + 1)
    mySet = set ()
    for i in range (0, x):
        mySet.add (i)
    for j in range (n, n + x):
        mySet.add (j)

    return mySet


#Finds min ZFS
def findZFS ():
    for i in range (0, len (G)):
        for s in set (itertools.combinations (halfVSet (), i)):
            print ("Testing ..................", set (s))
            if (isForce (set (s))):
                return s

ZFS = findZFS ()
ZFN = len (findZFS ())
print ("The smallest ZFS set is ", ZFS)
print ("The ZFN is ", ZFN)




#WOW! Using halfVSet () works SO much better!!!
#print ("The vertex set is ... ", vertexSet ())
#print (halfVSet ())






#Okay, so I'm noticing that this is working very well for smaller graphs:
#GP(4, 3) = 4
#GP(5, 3) = 5
#GP(6, 3) = 4
#
#
#Basically if the answer is 7, it will have a hard time because there are so
#many different combinations for 6. I wonder if there are ways to make it
#faster, like are the really bad collections of vertices that we know we don't
#want the computer to test?
```

```
#I've noticed that itertools.combinations has been repeating things.

#I also need to consider cutting large graphs in half. I'll cross that bridge
#when I get there.
```