The Sage Notebook
Version 8.1

hogben Toggle | Home | Published | Log | Settings | Help | Report a Problem | Sign out

Product_Throttling

Save | Save & quit | Discard & quit

last edited Jan 17, 2021, 2:39:31 PM by hogben

File... | Action... | Data... | sage | Typeset | Load 3-D Live | Use java for 3-D

Print | Worksheet | Edit | Text | Revisions | Share | Publish

```
#####################################
# This Sage worksheet is written by Leslie Hogben
# based on code written by numerous other people, some of whom
are listed.
# It contains examples illustrating results in the paper
# Survey of product throttling
# by Sarah Anderson, Karen Collins, Daniela Ferrero, Leslie
Hogben, Carolyn Mayer, Ann Trenk, Shanise Walker
# Please report errors to hogben@aimath.org
```

```
#####################################
# To run the examples, you must got to the bottom of this sheet
# and run all the function cells before running the examples.
# Look for "FUNCTIONS-HERE"
#
# Exampkes are given for both kinds of product throttling for
# power domination, standard zero forcing, and PSD forcing.
# There is no material for Cops and Robbers.
#####################################
```

```
#####################################
# Section 3
# Illustrate th_pd^*(G) = gamma(G) for several families of
graphs
```

```
# cycle
g=graphs.CycleGraph(8)
print thpda(g), dom(g)
```

```
    (3, 3) 3
```

```
# gridgraph
p3=graphs.PathGraph(3)
p5=graphs.PathGraph(5)
g=p3.cartesian_product(p5)
print thpda(g), dom(g)
```

```
(4, 4) 4
```

```
####################################
# Section 5
# Illustrate th^x(G) = order n of G and th^*(G) = k(G,1) = least
k such that pt(G,k)=1
####################################
```

```
# th^x(G)
# Since it is known that th^x(G) = order n of G and no set with
n/2 < k <n has thZx(G,k) <= n, and thZx(G,n)=n,
# thZx(G) is written as a demo and prints lists of k, pt(G,k),
and thZx(G,k) for k=Z(G),...,n/2 for comarison
# and returns (thZx(G,ko),ko) such that ko is least k having
```

```
p=graphs.PathGraph(6)
thZx(p)
```

```
1 5 6
2 2 6
3 1 6
(6, 1)
```

```
g=graphs.CycleGraph(7)
thZx(g)
```

```
2 3 8
3 2 9
(7, 7)
```

```
g=graphs.CycleGraph(10)
thZx(g)
```

```
2 4 10
3 4 15
4 2 12
5 2 15
(10, 2)
```

```
g=graphs.CompleteBipartiteGraph(4,5)
thZx(g)
```

```
    (9, 9)
```

```
# th^a(G)
# Since it is known that th^*(G) = k(G,1) = least k such that
pt(G,k)=1,
# thZx(G) is written as a demo and p prints lists of k, pt(G,k),
and thZa(G,k) for k=Z(G),..,k(G,1)
```

```
# Ex 5.8 th^a(P_n)=n/2 for n even

p=graphs.PathGraph(10)
```

```
    1 9 9
    2 4 8
    3 3 9
    4 2 8
    5 1 5
    (5, 5)
```

```
# th^a(P_n) < th^a(C_n) and th^a(P_n) = th^a(C_n) are both
```

```
g=graphs.CycleGraph(10)
thZa(g)
```

```
    2 4 8
    3 4 12
    4 2 8
    5 2 10
    6 1 6
    (6, 6)
```

```
p=graphs.PathGraph(8)
thZa(p)
```

```
    1 7 7
    2 3 6
    3 2 6
    4 1 4
    (4, 4)
```

```
g=graphs.CycleGraph(8)
thZa(g)
```

```
2 3 6
3 3 9
4 1 4
(4, 4)
```

```
# complete graph
```

```
g=graphs.CompleteGraph(10)
thZa(g)
```

```
9 1 9
(9, 9)
```

```
# complete bipartite graph
```

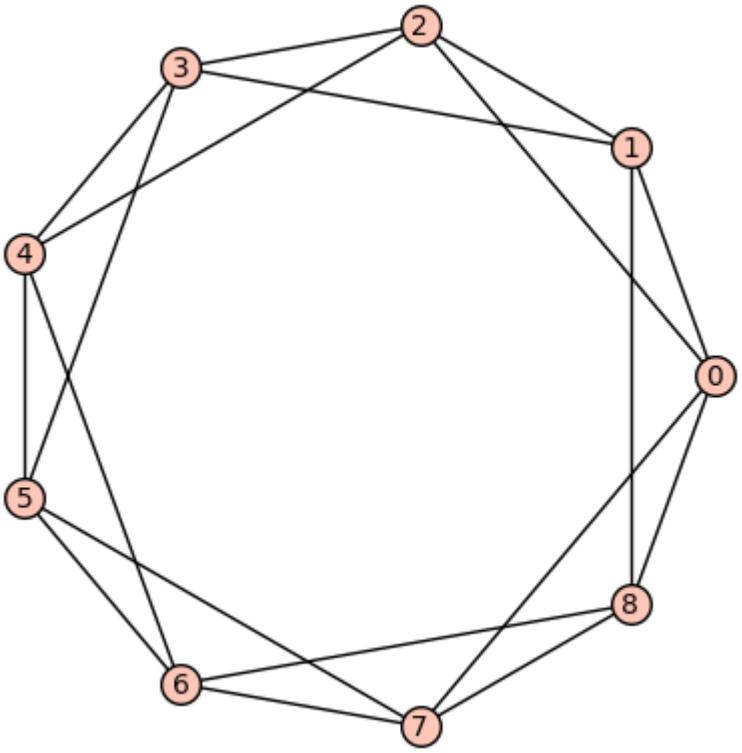
```
g=graphs.CompleteBipartiteGraph(5,8)
thZa(g)
```

```
11 1 11
(11, 11)
```

```
# circulant
```

```
g=graphs.CirculantGraph(9,[1,2])
show(g)
thZa(g)
```

```
4 3 12
5 2 10
6 2 12
7 1 7
(7, 7)
```




```
####################################
# Section 7
# Illustrate th_pd^*(G) and th_pd^x(G)
####################################
```

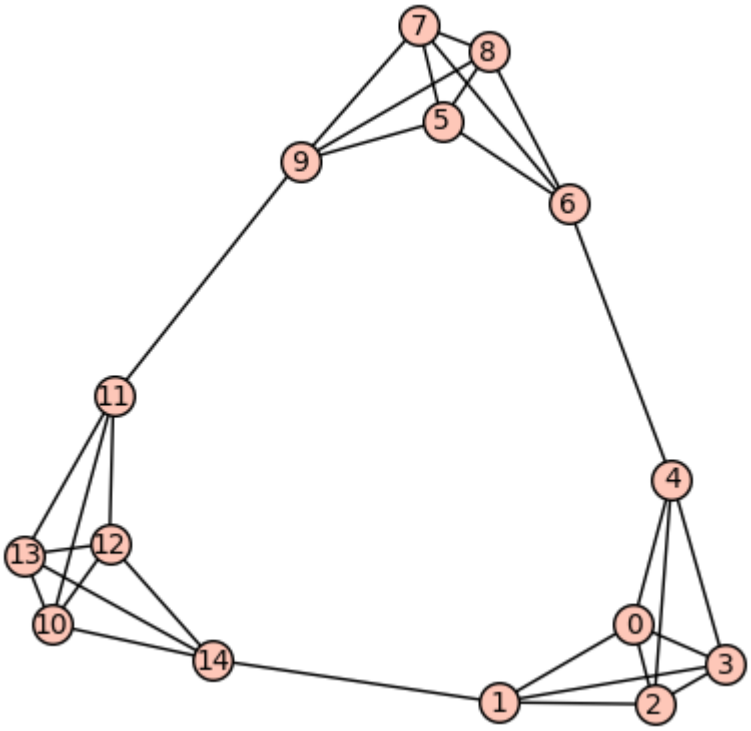
```
# 7.1 th_pd^*(G) for additional families
```

```
# hypercube
for d in [1..4]:
    g=graphs.CubeGraph(d)
    print d, thpda(g), dom(g)
```

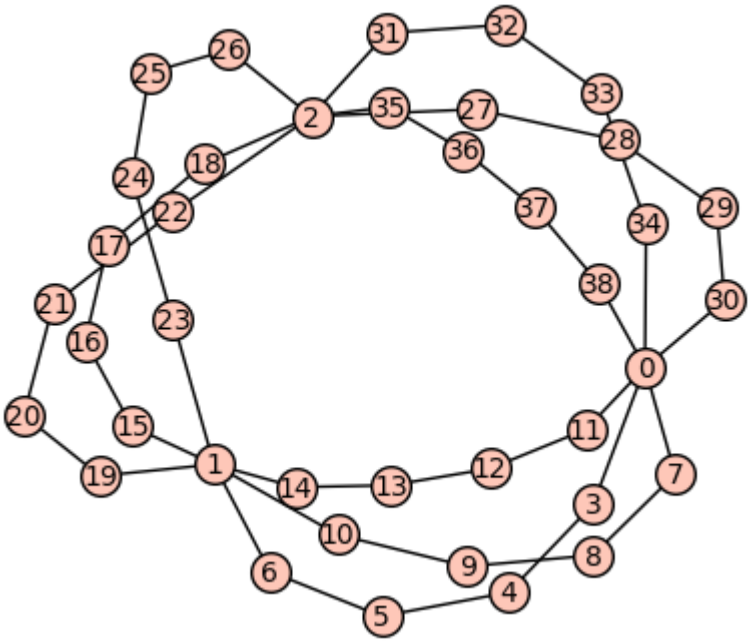
```
1 (1, 1) 1
2 (2, 2) 2
3 (2, 2) 2
4 (4, 4) 4
```

```
# necklace
k5a=graphs.CompleteGraph(5)
k5a.delete_edge([1,4])
k5b=graphs.CompleteGraph(5)
k5b.relabel([5,6,7,8,9])
k5b.delete_edge([6,9])
k5c=graphs.CompleteGraph(5)
k5c.relabel([10,11,12,13,14])
k5c.delete_edge([11,14])
h=k5a.union(k5b)
g=h.union(k5c)
g.add_edges([(4,6),(9,11),(14,1)])
show(g)
print thpda(g), dom(g)
```

```
(3, 3) 3
```



```
# G(3,3,4)
p6a=graphs.PathGraph(6)
p6a.relabel([0,3,4,5,6,1])
p6b=graphs.PathGraph(6)
p6b.relabel([0,7,8,9,10,1])
h1=p6a.union(p6b)
p6c=graphs.PathGraph(6)
p6c.relabel([0,11,12,13,14,1])
h2=h1.union(p6c)
p6d=graphs.PathGraph(6)
p6d.relabel([1,15,16,17,18,2])
h3=h2.union(p6d)
p6e=graphs.PathGraph(6)
p6e.relabel([1,19,20,21,22,2])
h4=h3.union(p6e)
p6f=graphs.PathGraph(6)
p6f.relabel([1,23,24,25,26,2])
h5=h4.union(p6f)
p6g=graphs.PathGraph(6)
p6g.relabel([2,27,28,29,30,0])
h6=h5.union(p6g)
p6h=graphs.PathGraph(6)
p6h.relabel([2,31,32,33,34,0])
h7=h6.union(p6h)
p6i=graphs.PathGraph(6)
p6i.relabel([2,35,36,37,38,0])
g=h7.union(p6i)
show(g)
```



```
dom(g)
```

```
12
```

```
pdg=PowerDom(g)
print pdg
pptg=ppt(g)
print pptg
pdg*pptg
```

```
2
5
10
```

```
thpda(g)
```

```
(6, 3)
```

```
# 7.2 th_pd^x(G)
```

```
# cycle
g=graphs.CycleGraph(10)
thpdx(g)
```

```
(6, 1)
```

```
# complete graph
g=graphs.CompleteGraph(15)
thpdx(g)
```

```
(2, 1)
```

```
# complete bipartite graphs
```

```
g=graphs.CompleteBipartiteGraph(1,12)
thpdx(g)
```

```
(2, 1)
```

```
g=graphs.CompleteBipartiteGraph(2,12)
thpdx(g)
```

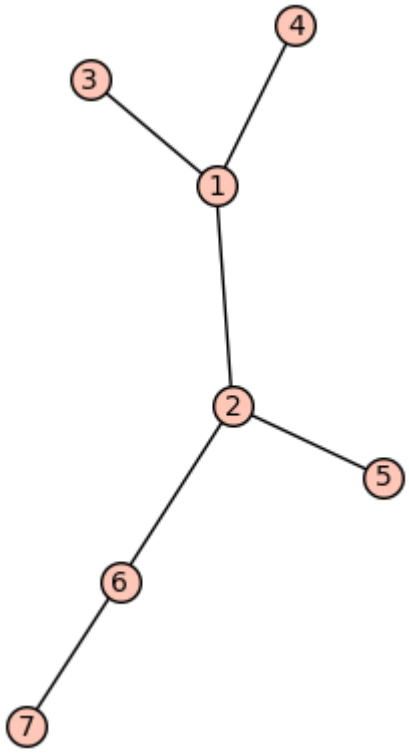
```
(3, 1)
```

```
g=graphs.CompleteBipartiteGraph(3,12)
thpdx(g)
```

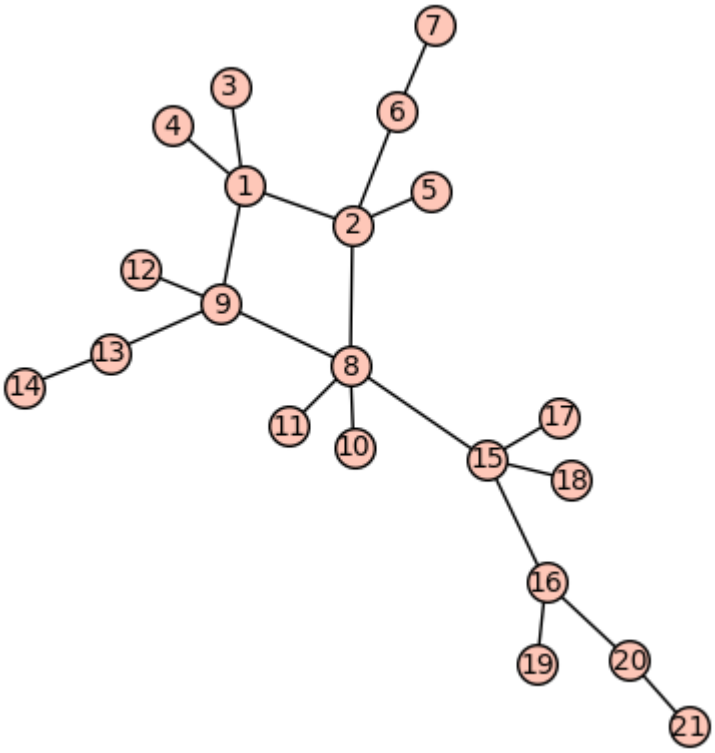
```
(4, 2)
```

```
g1=Graph({1:[2,3,4],2:[5,6],6:[7]})
show(g1)
thpdx(g1)
```

```
(6, 2)
```




```
g2=g1.copy()
g2.relabel([8..14])
g3=g1.copy()
g3.relabel([15..21])
h=g1.union(g2)
g=h.union(g3)
g.add_edges([(1,9),(2,8),(8,15)])
show(g)
```




```
thpdx(g)
```

```
(18, 6)
```

```
# grid graphs
```

```
p2=graphs.PathGraph(2)
p4=graphs.PathGraph(4)
g=p2.cartesian_product(p4)
thpdx(g)
```

```
(5, 1)
```

```
p2=graphs.PathGraph(2)
p6=graphs.PathGraph(6)
g=p2.cartesian_product(p6)
thpdx(g)
```

```
(6, 2)
```

```
####################################
# Section 8
# Illustrate th_+^x(G) and th_+^*(G)
####################################
```

```
# 8.1 th_+^x(G)
```

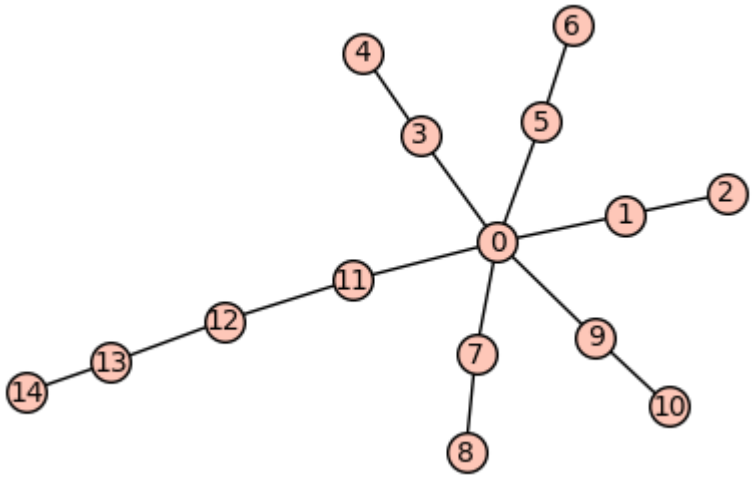
```
# th_+^x(T) = 1 + rad T using k = 1 if T is a tree
```

```
g=graphs.PathGraph(7)
print thpx(g), g.radius()
```

```
(4, 1) 3
```

```
g=Graph({1:[0,2],3:[0,4],5:[0,6],7:[0,8],9:[0,10],0:[11],12:
[11,13],14:[13]})
show(g)
```

```
(4, 1) 3
```




```
# cycles do interesting things
```

```
g=graphs.CycleGraph(6)
thpx(g)
```

```
(4, 2)
```

```
g=graphs.CycleGraph(11)
thpx(g)
```

```
(8, 2)
```

```
g=graphs.CycleGraph(15)
thpx(g)
```

```
(9, 3)
```

```
# Remark 8.4
# analysis of order 4 graphs (computations below)
# need to list: 4K_1, K_2 U 2K_1, K_3 U K_1, K_4
# thpx=4 but covered but covered by other case: P_3 U K_1, 2K_2,
paw, C_4, diamond
# thpx<4:K_1,3, P_3
```

```
want=[]
for g in graphs.nauty_geng("%s %s:%s "%(4,0,6)):
    want.append(g.canonical_label().graph6_string());
print len(want)
def thpx4(i):
    g=Graph(want[i])
    print thpx(g)
    show(g)
```

```
11
```

```
thpx4(0)
```

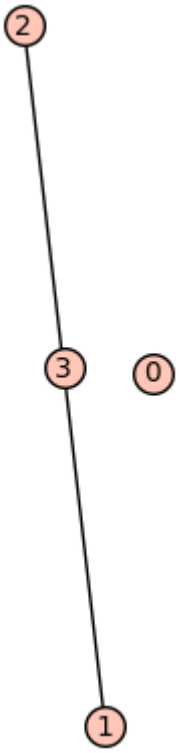
```
(4, 4)
```


```
thpx4(1)
```

```
(4, 4)
```

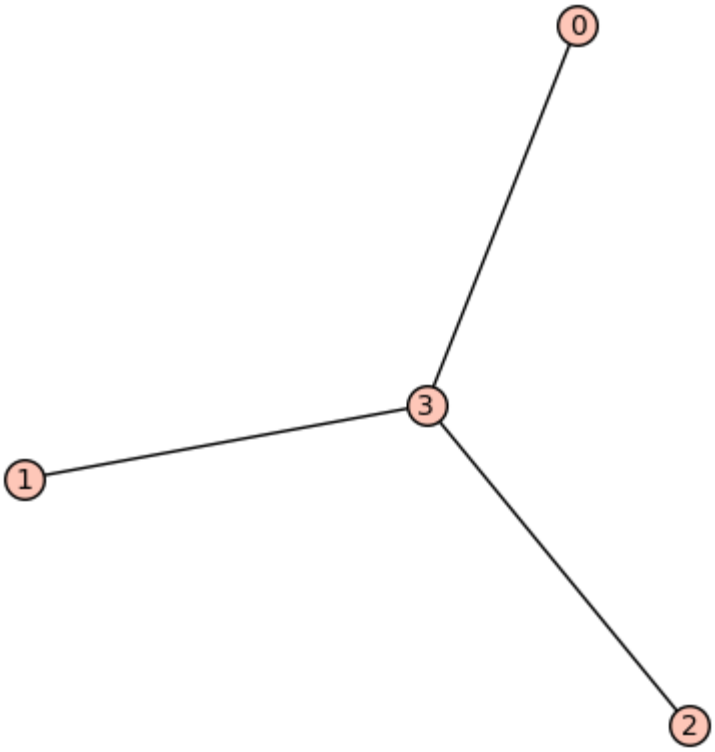

```
thpx4(2)
```

```
(4, 2)
```



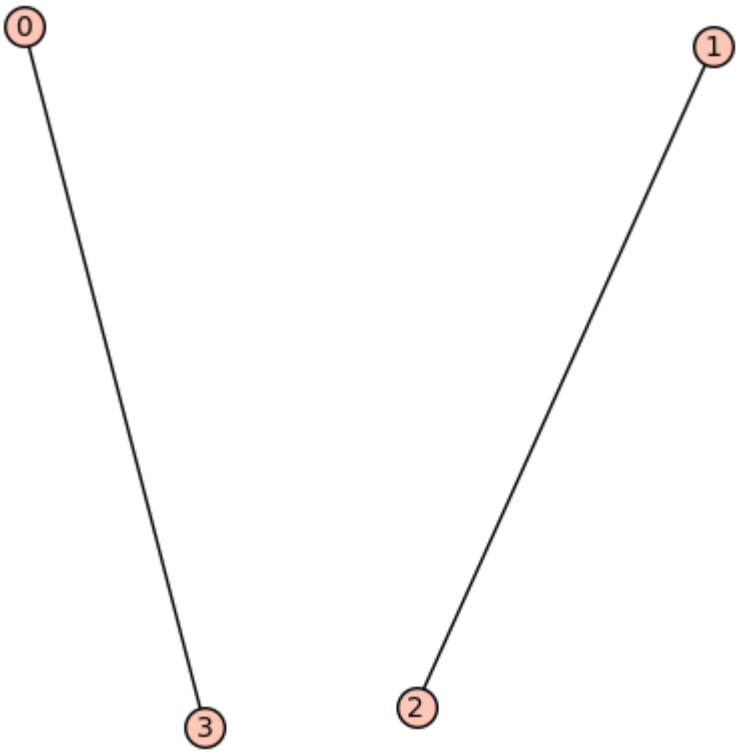

```
thpx4(3)
```

```
(2, 1)
```



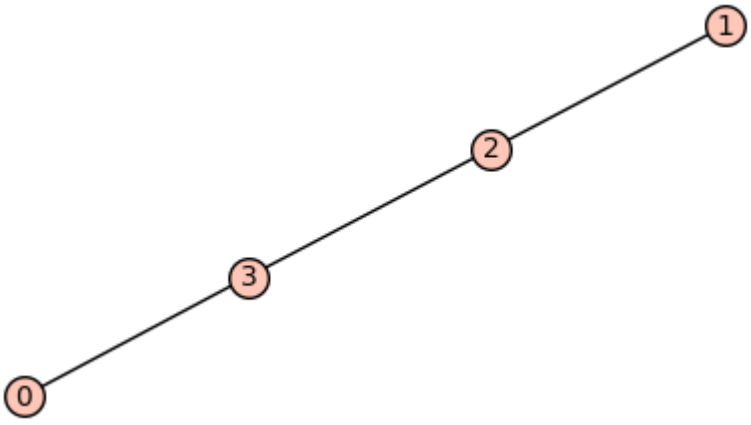

```
thpx4(4)
```

```
(4, 2)
```



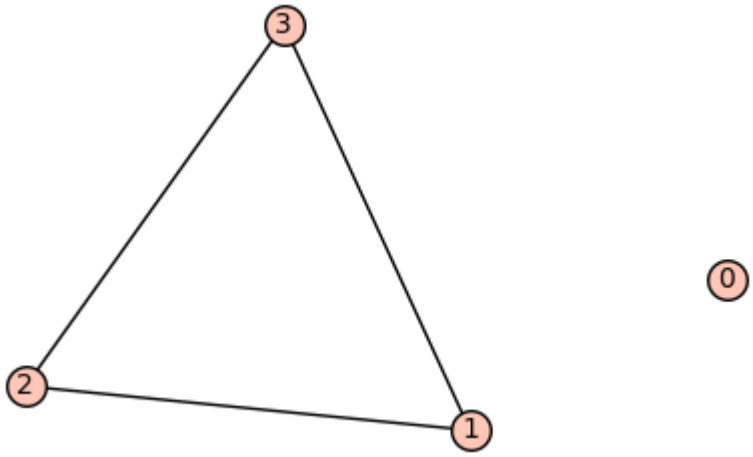

```
thpx4(5)
```

```
(3, 1)
```



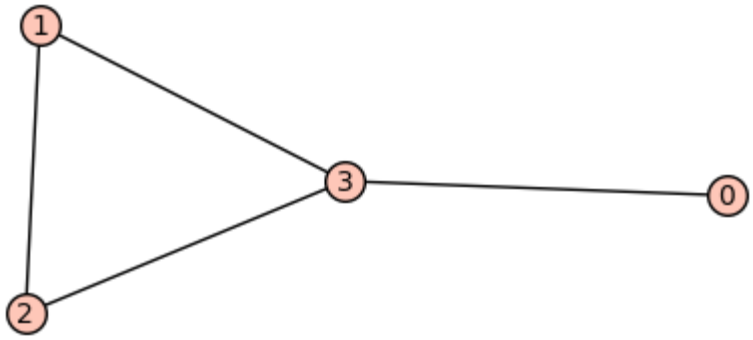

```
thpx4(6)
```

```
(4, 4)
```



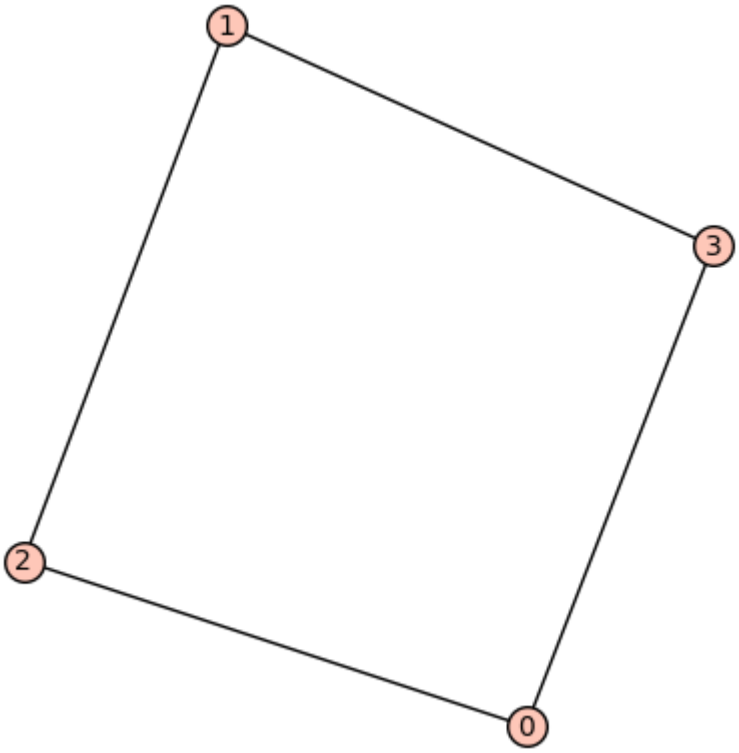

```
thpx4(7)
```

```
(4, 2)
```



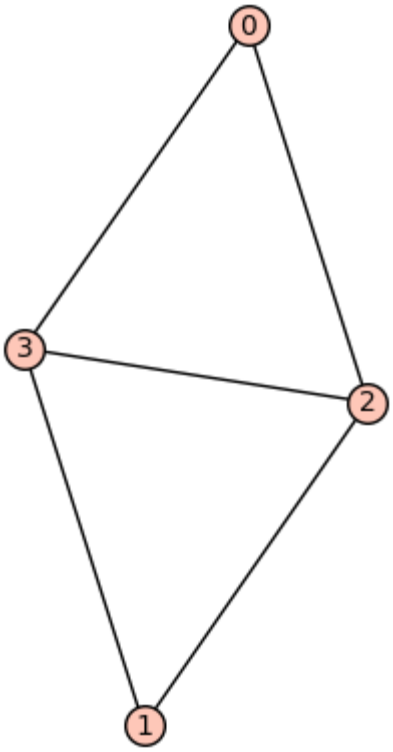

```
thpx4(8)
```

```
(4, 2)
```



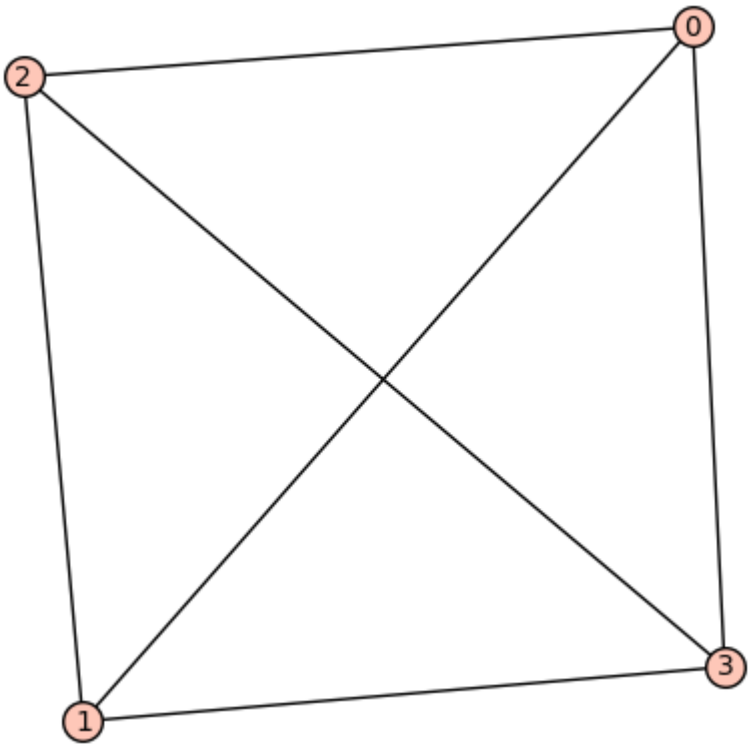

```
thpx4(9)
```

```
(4, 2)
```




```
thpx4(10)
```

```
(4, 4)
```




```
# 8.2 th_+^a(G)
```

```
# for paths and cycles, th_+^a(G) = th_c^a(G) = th_pd^a(G) =
gamma(G) = ceiling(n/3) with propagation time 1
```

```
g=graphs.CycleGraph(6)
thpa(g)
```

```
(2, 2)
```

```
g=graphs.CycleGraph(11)
thpa(g)
```

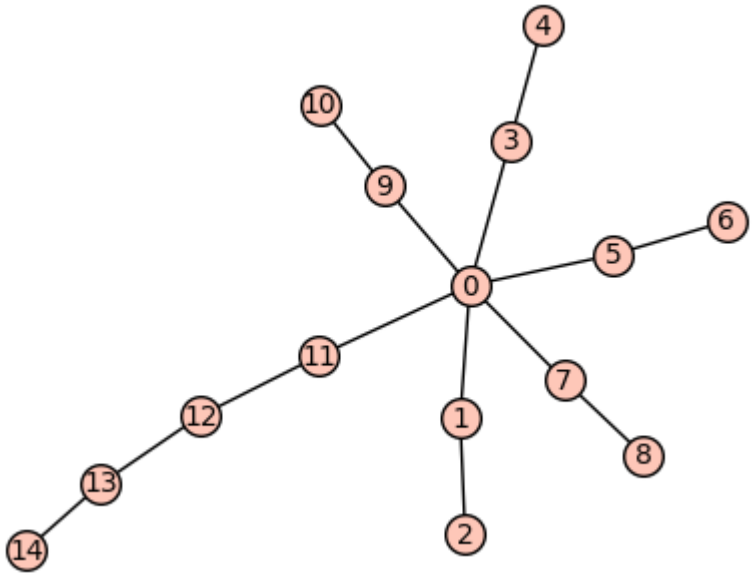
```
(4, 4)
```

```
g=graphs.CycleGraph(15)
thpa(g)
```

```
(5, 5)
```

```
# for trees T, th_+^a(T) = th_+^c(T) but th_+^pd(T) may differ
```

```
g=Graph({1:[0,2],3:[0,4],5:[0,6],7:[0,8],9:[0,10],0:[11],12:
[11,13],14:[13]})
```




```
thpa(g)
```

```
(3, 1)
```

```
thpda(g)
```

```
(4, 1)
```

```
####################################
# FUNCTIONS-HERE
# These are the main function cells: evaluate each one
# before trying to run the examples above.
####################################
```

```
####################################
# Standard zero forcing, propagation, and throttling
####################################
```

```
# Load main zero forcing functions
# zero forcing, PSD forcing, and minimum rank code
# developed by S. Butler, L. DeLoss, J. Grout, H.T. Hall, J.
LaGrange, J.C.-H. Lin,T. McKay, J. Smith, G. Tims
# updated and maintained by Jephian C.-H. Lin
URL='https://raw.githubusercontent.com/jephianlin/mr_JG/py2/'
files=
['Zq_c.pyx','Zq.py','zero_forcing_64.pyx','zero_forcing_wavefron
t.pyx','minrank.py', 'inertia.py']
for f in files:
    print("Loading %s..."%f);
```

```
Loading Zq_c.pyx...
Compiling
/home/sageuser/7/.sage/temp/sage.math.iastate.edu/27269/tmp_RN
.
Loading Zq.py...
Loading zero_forcing_64.pyx...
Compiling
/home/sageuser/7/.sage/temp/sage.math.iastate.edu/27269/tmp_wq
.
Loading zero_forcing_wavefront.pyx...
Compiling
/home/sageuser/7/.sage/temp/sage.math.iastate.edu/27269/tmp_rf
.
Loading minrank.py...
Loading inertia.py...
```

```
# Additional function for zero forcing number using brute_force
def Z(g):
    z=len(zero_forcing_set_bruteforce(g))
    return z
```

```
# Function for standard propagation time of set S, pt(G,S)
# adapted by Leslie Hogben from Steve Butler's skew propagation
time code
# input: a graph G and a set S of vertices
# output: pt(G,S) = prop time of S in G (if -1 is returned then
S is not a zero forcing set)
def ptz(G,S):
    V=set(G.vertices())
    count = 0
    done = False
    active = set(S)
    filled = set(S)
    for v in V:
        N=set(G.neighbors(v))
        if v in active and N.issubset(filled):
            active.remove(v)
        if (v in filled) and (v not in active) and
(len(N.intersection(filled)) == G.degree(v)-1):
            active.add(v)
    while not done:
        done = True
        new_active = copy(active)
        new_filled = copy(filled)
        for v in active:
            N=set(G.neighbors(v))
            if len(N.intersection(filled)) == G.degree(v)-1:
                if done:
                    done = False
                    count += 1

N.symmetric_difference_update(N.intersection(filled))
                u=N.pop()
                new_active.remove(v)
                new_active.add(u)
                new_filled.add(u)
        active = copy(new_active)
        filled = copy(new_filled)
        # print filled
        for v in V:
            N=set(G.neighbors(v))
            if v in active and N.issubset(filled):
                active.remove(v)
            if (v in filled) and (v not in active) and
(len(N.intersection(filled)) == G.degree(v)-1):
                active.add(v)
    if len(filled)==len(V):
```

```
# More functions for standard propagation time
# pt(G,k) = min prop time over zero forcing sets of G of size k
# this is the function used to compute propation time of G and
also throttling
# input: a graph G and a positive integer k >= Z(G)
# output: pt(G,k) = min prop time over zero forcing sets G of
size k (if order+1 is returned then k<Z(G))
def ptk(G,k):
    V=set(G.vertices())
    sets = subsets(V,k)
    savept=len(V)+1
    for S in sets:
        pt=ptz(G,S)
        if 0 <= pt < savept:
            savept=pt
    return savept

# propagation time pt(G)
# input: a graph G
# output: propagation time pt(G)=pt(G,Z(G))
```

```
# Function for product-x Z throttling number
#
# This function computes th^x(G) by minimizing th^x(G,k)
# It is known theoretically that this is always the order of the
graph.
# It contains a print statement to illustrate this
#
# input: a graph g
# output: product throttling number th^x(g) and least k such
that th^x(g) = th^x(g,k)
def thZx(g):
    nn=g.order()
    saveko=Z(g)
    ptko=ptk(g,saveko)
    savept=ptko
    for ko in [saveko..nn/2]:
        ptko=ptk(g,ko)
        print ko, ptko, ko*(1+ptko)
        if ko*(1+ptko)<saveko*(1+savept):
            saveko=ko
            savept=ptko
    if saveko*(1+savept)>nn:
        saveko=nn
        savept=0
```

```
# Functions for product-* Z throttling number
#
# This function computes th^*(G) by minimizing th^*(G,k)
# It is known theoretically that this is the least ko such that
pt(G,k0)=1.
# It contains a print statement to illustrate this
#
# input: a graph G
# output: product throttling number th^*(G) and least k such
that th^*(G) = th^*(G,k)
def thZa(g):
    ko=Z(g)
    nn=g.order()
    ptko=ptk(g,ko)
    savept=ptko
    saveko=ko
    while ptko >1:
        print ko, ptko, ko*ptko
        ko=ko+1
        ptko=ptk(g,ko)
        if ko*ptko<saveko*savept:
            saveko=ko
            savept=ptko
    print ko, ptko, ko*ptko
```

```
####################################
# Power domination, propagation, and throttling
####################################
```

```
# Power dominating set functions
# by Brian Wissman

# input: a graph G and a subset of its vertices V
# output: true/false depending if V is a power dominating set of
G
def isPowerDominatingSet(G,V):
    N=[]
    for i in V:
        N+=G.neighbors(i)
    NVert=uniq(N+list(V))

    A=zerosgame(G,NVert)
    if len(A)==G.order():
        return True
    else:
        return False

# input: a graph G
# output: a power dominating set of G that achieves the power
domination number
def minPowerDominatingSet(G):
    V = G.vertices()
    for i in range(1,len(V)+1):
        S = subsets(V,i)
        for s in S:
            if isPowerDominatingSet(G,s):
                return s

# input: a graph G
# output: the power domination number of G, gamma_P(G)
def PowerDom(G):
    V = G.vertices()
    for i in range(1,len(V)+1):
        S = subsets(V,i)
```

```
# Functions for power propagation time

# function to compute closed neighborhood N[S]
# input: a graph G and a list S of vertices
# output: set N[S]
def nhbd(G,S):
    NS=set(S)
    for x in S:
        nx=G.neighbors(x)
        NS=NS.union(set(nx))
    return NS

# function to compute ppt(G,S) = power propagation time of a set
S
# input: a graph G and a subset S of vertices
# output: ppt(G,S)
def pptS(G,S):
    ord=G.order()
    V = G.vertices()
    NS=nhbd(G,S)
    ptNS=ptz(G,NS)
    if ptNS>-1:
        ptNS=ptNS+1
    return ptNS

# function to compute ppt(G,k) = minimum power propagation time
over sets of size k
# input: a graph G and a positive integer k
# output: ppt(G,k)
def pptk(G,k):
    ord=G.order()
    V = G.vertices()
    S = subsets(V,k)
    ptm = -1
    for s in S:
        ptms=pptS(G,s)
        if (ptm < 0):
            ptm=ptms
        if (ptms >= 0) and (ptms < ptm):
            ptm=ptms
    return ptm

# function to compute ppt(G) = ppt(G,gamma_P(G))
```

```
# compute domination number of graph g
def dom(g):
    gds=g.dominating_set()
    ds=len(gds)
    return ds
```

```
# Function for product-x power throttling number
#
# This function computes th_pd^x(G) by minimizing th_pd^x(G,k)
over k in {1,2,...,2/3gamma(G),gamma(G)}
#
# input: a graph g
# output: product throttling number th_pd^x(g) and least k such
that th_pd^x(g) = th_pd^x(g,k)
def thpdx(g):
    nn=g.order()
    saveko=PowerDom(g)
    savept=pptk(g,saveko)
    gam=dom(g)
    for ko in [saveko+1..2*gam/3]:
       ptko=pptk(g,ko)
       if ko*(1+ptko)<saveko*(1+savept):
           saveko=ko
           savept=ptko
    if 2*gam<saveko*(1+savept):
        saveko=gam
        savept=1
    if nn<saveko*(1+savept):
        saveko=nn
        savept=0
    return saveko*(1+savept), saveko
```

```
# Function for product-* power throttling number
#
# This function computes th_+^*(G) by minimizing th_+^*(G,k)
#
# input: a graph G
# output: product throttling number th_pd^*(G) and k such that
th_pd^*(G) = th_pd^*(G,k)
#         if th_pd^*(G)=gamma(G) then k=gamma(G) is returned;
othersie least k is returned
def thpda(g):
    nn=g.order()
    gam=dom(g)
    saveko=gam
    savept=1
    kmin=PowerDom(g)
    for ko in [kmin..gam/2]:
        ptko=pptk(g,ko)
        if ko*ptko<saveko*savept:
           saveko=ko
           savept=ptko
        if saveko*savept <= gam/2:
            return saveko*savept, saveko
    return saveko*savept, saveko
```

```
####################################
# PSD zero forcing, propagation, and throttling
####################################
```

```
# Load PSD propagation time funtions, including pt_plus(G,S) =
PSD propagation time of a set S
# code by Nathan Warnberg
# updated and maintained by Jephian C.-H. Lin
load('https://raw.githubusercontent.com/jephianlin/zero_forcing
/master/psd_prop_time_interval.py')
```

```
    xrange test passed
    Loading Zq_c.pyx...
    Compiling
    /home/sageuser/7/.sage/temp/sage.math.iastate.edu/27269/tmp_ol
    .
    Loading Zq.py...
    Loading zero_forcing_64.pyx...
    Compiling
    /home/sageuser/7/.sage/temp/sage.math.iastate.edu/27269/tmp_fl
    .
```

```
    Loading zero_forcing_wavefront.pyx...
    Compiling
    /home/sageuser/7/.sage/temp/sage.math.iastate.edu/27269/tmp_HA
    .
    Loading minrank.py...
    Loading inertia.py...
```

```
# Functions for PSD propagation time

# function to compute pt_+(G,k) = minimum PSD propagation time
over sets of size k
# input: a graph G and a positive integer k
# output: pt_+(G,k)
def ptpk(G,k):
    ord=G.order()
    V = G.vertices()
    S = subsets(V,k)
    ptp = -1
    for s in S:
        ptps=pt_plus(G,s)
        if (ptp < 0):
            ptp=ptps
        if (ptps >= 0) and (ptps < ptp):
            ptp=ptps
    return ptp

# function to compute pt_+(G) = pt_+(G,Z_+(G))
# input: a graph G
# output: pt_+(G)
def ptp(G):
```

```
# Function for product-x PSD throttling number
#
# This function computes th_+^x(G) by minimizing th_+^x(G,k)
#
# input: a graph g
# output: product throttling number th^x(g) and least k such
that th^x(g) = th^x(g,k)
def thpx(g):
    nn=g.order()
    saveko=Zplus(g)
    savept=ptpk(g,saveko)
    for ko in [saveko+1..nn/2]:
       ptko=ptpk(g,ko)
       if ko*(1+ptko)<saveko*(1+savept):
           saveko=ko
           savept=ptko
    if nn<saveko*(1+savept):
        saveko=nn
        savept=0
    return saveko*(1+savept), saveko
```

```
# Function for product-* PSD throttling number
#
# This function computes th_+^*(G) by minimizing th_+^*(G,k)
#
# input: a graph G
# output: product throttling number th_+^*(G) and least k such
that th_+^*(G) = th_+^*(G,k)
def thpa(g):
    ko=Zplus(g)
    nn=g.order()
    ptko=ptpk(g,ko)
    savept=ptko
    saveko=ko
    while ptko >1:
        ko=ko+1
        ptko=ptpk(g,ko)
        if ko*ptko<saveko*savept:
            saveko=ko
            savept=ptko
    return saveko*savept, saveko
```